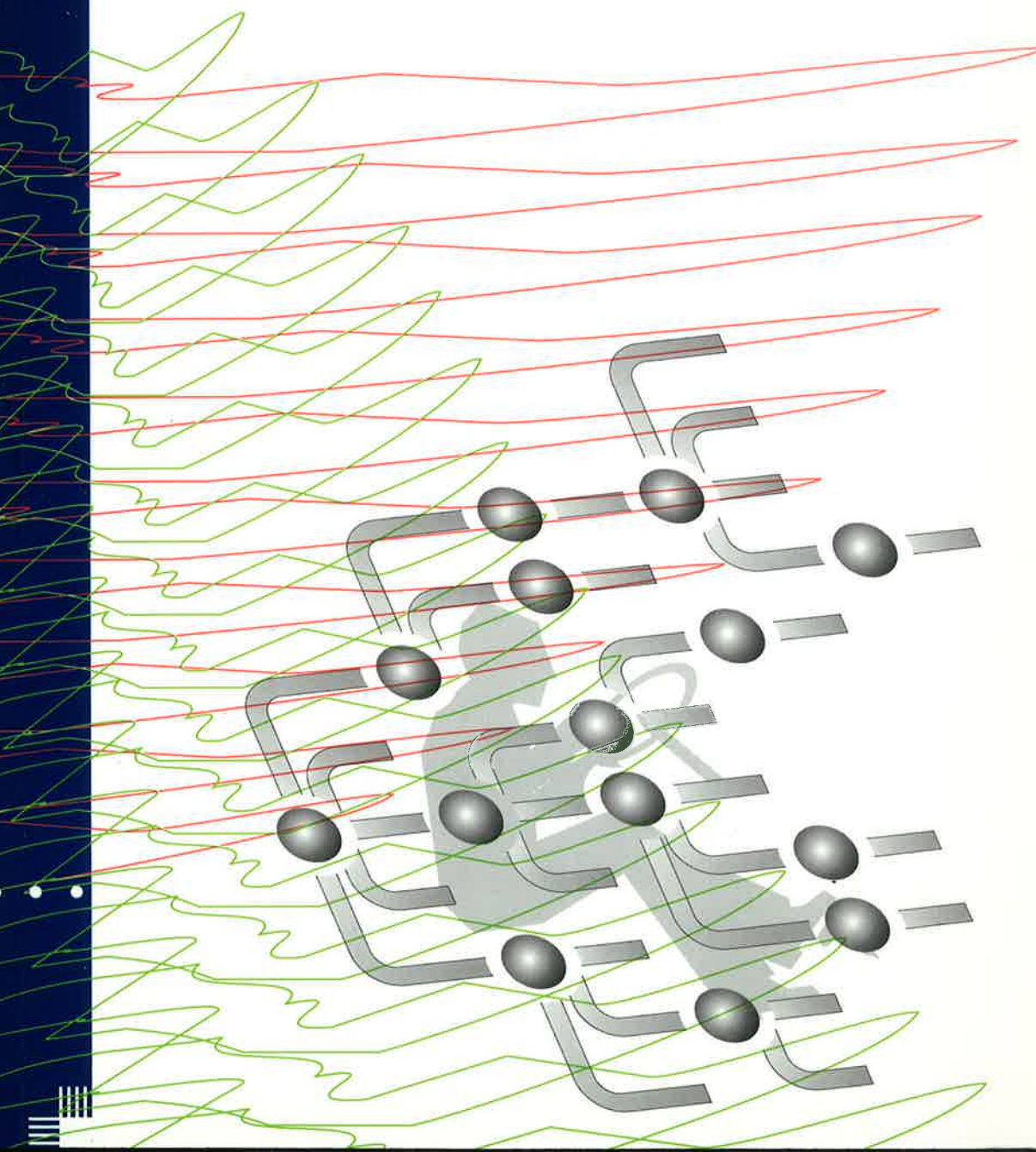


Jans Aasman

Modelling Driver Behaviour In Soar



Modelling Driver Behaviour in Soar

CIP-GEGEVENS KONINKLIJKE BIBLIOTHEEK, DEN HAAG

Aasman, Jannes

Modelling Driver Behaviour in Soar / Jannes Aasman – Leidschendam:
KPN Research. – Ill.

Proefschrift Rijksuniversiteit Groningen. – Met lit. opg., – Met Nederlandse
samenvatting.

ISBN 90-72125-50-9

Trefw.: cognitive psychology, driving, modelling

© 1995 by Royal PTT Nederland NV, KPN Research

Subject to the exceptions provided for by law, no part of this publication may be reproduced and/or published in print, by photocopying on microfilm or in any other way without the written consent of the copyright owner. The same applies to whole or partial adaptations. The copyright owner retains the sole right to collect from third parties fees payable in respect of copying and/or to take legal or other action for this purpose.

Stellingen

behorende bij het proefschrift

**Modelling Driver Behaviour
in Soar**

Jans Aasman
4 mei 1995

1. De standaard zoekregels die in dit proefschrift beschreven worden, sluiten beter aan bij (a) beperkingen van het menselijk werkgeheugen, (b) het gemak waarmee mensen met onderbrekingen omgaan, (c) het gemak waarmee zij fouten herstellen, en (d) de progressief-verdiepende zoekmethode die zij hanteren. Deze regels zullen daarom op termijn Soar's huidige standaard zoekregels dienen te vervangen.
2. DRIVER heeft bij het naderen en afhandelen van wegkruisingen net voldoende tijd om alle noodzakelijke interne en externe rijtaken uit te voeren. Voor een extra taak schiet er in het geheel geen tijd meer over. Dit toont aan dat de in dit proefschrift gebruikte duur van 30 milliseconden voor een elaboratiecyclus een absolute bovengrens is.
3. In de cognitief-psychologische georiënteerde Soar literatuur wordt Soar steeds minder beschreven als een architectuur voor het implementeren van allerlei zoekmethoden en steeds meer als een gedetailleerd model van menselijke informatieverwerking. Dit lijkt geheel in de lijn der bedoelingen van wijlen Allen Newell.
4. Soar zal pas succesvol met de wereld in wisselwerking kunnen treden als het voorzien wordt van een vervalproces voor het werkgeheugen en van een methode om foutief geleerde kennis makkelijk af te leren.
5. Elke wetenschappelijke poging tot classificatie van emoties verbleekt bij de inhoudsopgave van Komrij's *Humeuren en Temperamenten*.
6. Het uitstellen van de invoering van een intelligente snelheid-begrenzer die de snelheid van de auto aanpast aan de eisen van de omgeving is misdadig en barbaars.

7. De aarzeling van de minister van Verkeer en Waterstaat om fietsers van rechts voorrang te geven is aanvaardbaar, indien deze aarzeling ingegeven wordt door haar verlangen om daarnaast ook fietsers van links voorrang te geven.
8. Een simpele index voor de beschavingsgraad van een samenleving is:
$$B \approx 1/(D_v/I_v + D_h/I_h),$$
waarin B de beschavingsgraad is, D_v het aantal doden in het verkeer per jaar, I_v de *jaarlijkse toename* in de investeringen om deze terug te dringen, D_h het aantal mensen op een wachtlijst van hartpatiënten dat jaarlijks sterft en I_h de *jaarlijkse toename* in investeringen om deze sterfte tegen te gaan.
9. Ouders moeten professionele bijstand kunnen krijgen bij het omgaan met het pedagogische dilemma of ze hun kinderen wel of niet vertellen dat auto's stoppen voor een zebraapad.
10. Als de door de Nederlandse regering gewenste energieheffing niet bedoeld is om de schatkist te spekken maar om werkelijk de verspilling van de wereldenergievoorraad tegen te gaan, dan is de enige zinvolle besteding van de opbrengst het stimuleren van intensieve lobbycampagnes voor energiebesparende maatregelen in landen waar de meeste energie verspild wordt.
11. Als wij niet oppassen gaat ook de multimedia revolutie aan de vrouw voorbij.
12. Over 10 jaar zal de telecommunicatie-industrie 50 procent van de ontwikkelbudgetten voor nieuwe diensten bestemmen voor het bedieningsgemak van de gebruiks-interfaces.
13. Treinreizigers die wel een geldig kaartje hebben maar de trein missen door een perroncontrole, behoren onverwijld en gratis per taxi naar hun bestemming gebracht te worden.

Rijksuniversiteit Groningen

Modelling Driver Behaviour in Soar

Proefschrift

ter verkrijging van het doctoraat in de
Psychologische, Pedagogische en Sociologische Wetenschappen
aan de Rijksuniversiteit Groningen
op gezag van de Rector Magnificus Dr. F. van der Woude
in het openbaar te verdedigen op
donderdag 4 mei 1995 des namiddags te 2.45 uur precies

door

Jannes Aasman

geboren op 29 juli 1958 te Emmen

Leiden
1995

Promotores:

Prof. Dr. J.A. Michon
Prof. Dr. Ir. N.J.I. Mars

Voorwoord

Graag wil ik op deze plaats het Verkeerskundig Studiecentrum der Rijksuniversiteit Groningen bedanken voor de mij geboden gelegenheid het onderzoek voor dit proefschrift uit te voeren. Daarnaast wil ik KPN Research bedanken voor de mogelijkheid om het proefschrift af te ronden.

Veel collega's hebben direct of indirect invloed gehad op de uiteindelijke inhoud en vorm van dit proefschrift.

Het Verkeerskundig Studiecentrum (VSC) bood mij voor elke vraag betreffende menselijk gedrag in het verkeer minstens één expert-collega. Daarnaast zorgden de collega's van het VSC voor een bijzonder prettige en vriendschappelijke werksfeer.

De VSC-collega's Jaap de Velde Harsenhorst en Peter Lourens leenden mij de empirische data die het mogelijk maakten om een, in eerste instantie nogal theoretisch, model uit te werken naar een concreet en realistisch model. Ik kan ze hier niet genoeg voor bedanken. Marcel Wierda leverde mij de kennis om een psychologisch relevant model van visuele oriëntatie in DRIVER te ontwikkelen. Mijn inzicht in de donkere diepten van de Soar architectuur zijn vooral gevormd door discussies met Ep Piersma en Aladin Akyürek. Met Aladin heb ik daarnaast gewerkt aan nieuwe standaard zoekregels voor Soar, zonder welke het niet mogelijk is om in Soar real-time gedrag te genereren.

Mijn periode als *visiting scientist* bij Carnegie Mellon University leverde mij de mogelijkheid om samen te werken met één van de meest inspirerende wetenschappers van onze tijd: Allen Newell. Als ik van één ding spijt heb met betrekking tot dit proefschrift dan is wel dat ik te laat ben geweest om het nog aan Allen te kunnen overhandigen.

In dezelfde periode ontmoette ik vriend en collega Doug die net als ik met het modelleren van verkeerswerelden bezig bleek. Ik bewaar in mijn Email folder "Douglas Reece" bijna 700 Kb aan discussie over het modelleren van rijgedrag.

Professor Koos Mars heeft in het laatste stadium van de totstandkoming van dit proefschrift enkele essentiële knopen doorgehakt. Zo dwong hij mij door

streng doorvragen het doel van dit proefschrift eindelijk eens goed te formuleren.

Over de invloed van Professor John Michon op de inhoud van dit proefschrift en over zijn unieke manier van begeleiden, zou ik vele voorwoorden vol kunnen schrijven. Laat ik er na acht jaren samenwerking mee volstaan dat ik hem zeer dankbaar ben voor mijn academische opvoeding. Naast al het inhoudelijke, heb ik van hem geleerd: het denken in grotere verbanden en hoe structuur te scheppen in chaos; het begin van schrijverschap; het inzicht in het belang van toegepast wetenschappelijk onderzoek; het creatief en assertief omgaan met bureaucratie; de noodzaak om medewerkers te vertrouwen en verantwoordelijkheden te geven; en vooral: het belang van strategisch denken.

Na deze lijst van mensen die invloed hebben gehad op de vorm en inhoud van dit proefschrift wil ik nog enkele mensen speciaal bedanken. Ik bedank mijn nieuwe collega's bij KPN Research, in het bijzonder Hennie Dijkhuis, voor het geduld dat zij met mij leken te hebben. Daarnaast bedank ik mijn zus Susan voor haar hulp bij de laatste loodjes, Klaas van Slooten voor de vormgevingsadviezen, Jan Hein Donker voor het drukklaar maken van dit boek, en mijn vriendin voor de afleiding en de zonen.

Jans Aasman
Leiden, maart 1995

aan mijn ouders

Table of contents

1 Introduction	1
1.1 Introduction	1
1.2 Driver behaviour	3
1.2.1 Models in traffic science	5
1.2.2 The psychological perspective of this study	8
1.3 Soar	10
1.3.1 The basic Soar mechanisms	10
1.3.2 Other cognitive modelling tools	16
1.4 Using Soar	17
1.4.1 The Soar theory of immediate behaviour or the “minimal scheme for immediate responses”	17
1.4.2 Unresolved research issues in Soar	19
1.4.3 Evaluating Soar’s suitability	21
1.5 Layout of the study	21
1.5.1 Part I	21
1.5.2 Part II	22
1.6 Notes	23
2 Multitasking in driving	25
2.1 Introduction	25
2.1.1 Using Soar in modelling multitasking driver behaviour	27
2.2 The Model	29
2.2.1 Perception and internal representation of the world	29
2.2.2 Multitasking: Managing multiple tasks in the process manager space	31
2.2.3 Process objects represent goals; Process operators implement tasks	31
2.2.4 Interrupt objects signal possible change of process	32
2.2.5 Change-process operators shift processes	35
2.3 Driving Tasks	37
2.3.1 The control-speed and control-course spaces	37
2.3.2 The negotiate-intersection space	38
2.3.3 The find-destination space	38
2.4 Discussion and Evaluation of the Model	42
2.4.1 Mapping processing types onto the driver model	42

2.4.2 <i>Evaluation and suggestions for future research</i>	43
2.5 Notes	47
3 Flattening Goal Hierarchies	49
3.1 Introduction	49
3.2 The Default Mechanism for Operator Tie Impasses	50
3.3 Issues	52
3.3.1 <i>Memory</i>	52
3.3.2 <i>Search</i>	53
3.3.3 <i>Learning</i>	54
3.4 Alternative Approaches to Deal With Ties	54
3.4.1 <i>Alternative 1 eliminates the selection space</i>	54
3.4.2 <i>Alternative 2 avoids ties</i>	56
3.4.3 <i>Alternative 3 eliminates state copying</i>	57
3.5 The Interrupt Issue and Its Relation to Default Behaviour	61
3.6 Concluding Remarks	62
3.7 Notes	63
4 Basic driver operations in negotiating general-rule intersections	65
4.1 Introduction	65
4.1.1 <i>Selected driver operations</i>	66
4.1.2 <i>Situational factors determining driver behaviour</i>	67
4.1.3 <i>The relevance of the De Velde Harsenhorst and Lourens data</i>	68
4.1.4 <i>Using the data</i>	69
4.2 The original De Velde Harsenhorst and Lourens analysis	69
4.2.1 <i>Results of the analysis of instructor comments</i>	70
4.2.2 <i>Results of the analysis of the 20-minute sections</i>	71
4.3 Reanalysis of De Velde Harsenhorst and Lourens's data	72
4.3.1 <i>Subjects</i>	72
4.3.2 <i>Material and apparatus</i>	73
4.3.3 <i>Locations and manoeuvres</i>	73
4.3.4 <i>Pilot procedure</i>	73
4.3.5 <i>Data analysis</i>	75
4.4 Results	78
4.4.1 <i>The speed profile</i>	78
4.4.2 <i>The brake profile</i>	79
4.4.3 <i>Manipulation of other car-control devices</i>	82
4.4.4 <i>Visual orientation</i>	83
4.4.5 <i>Cues used</i>	85
4.5 Concluding remark	88
4.6 Notes	88
5 Introduction to Part II	91
5.1 Introduction	91
5.1.1 <i>An overview of DRIVER</i>	91

5.2 Implementation notes	93
5.3 Layout of the remainder of this study	94
5.4 Notes	96
6 DRIVER's small world	97
6.1 Introduction	97
6.1.1 Constraints on DRIVER that shape WORLD	98
6.2 Implementation of WORLD	99
6.2.1 Representation of objects	99
6.2.2 The semi-intelligent agents and the overall control loop in WORLD	101
6.2.3 The speed control rules	101
6.3 Discussion	104
6.3.1 The "naturalness" of semi-intelligent agents' behaviour	104
6.3.2 The semi-intelligent agents are not instantiations of a cognitive model of driver behaviour	105
6.3.3 The integration of DRIVER in WORLD	105
6.4 Notes	106
7 Basic motor control.....	107
7.1 Introduction	107
7.2 The constraints shaping DRIVER's motor control.....	109
7.3 Implementation of motor control in DRIVER	110
7.3.1 DRIVER's body representations	112
7.3.2 Move operator and the (high-level) motor command language.....	112
7.3.3 Lower-level motor module (LLMM)	115
7.4 Discussion	117
7.4.1 Timing.....	118
7.4.2 Experimenting with the control mechanisms.....	118
7.5 Notes	118
8 Motor planning and vehicle control.....	121
8.1 Introduction	121
8.1.1 Types of knowledge in gear-changing.....	121
8.1.2 Aspects of learning in gear changing.....	122
8.1.3 Preprogrammed and learned knowledge in DRIVER	123
8.2 Implementation of car control in DRIVER	124
8.2.1 The change-gear operator	124
8.2.2 Building device plans.....	124
8.2.3 Building a motor-plan	126
8.2.4 Learning the device plan and motor plan.....	128
8.2.5 Execution	131
8.2.6 Learning timing knowledge by correction	133
8.3 Discussion	135
8.3.1 Designing the plans	135
8.3.2 The execution of plans.....	135

8.3.3 <i>Learning issues</i>	136
8.4 Notes	141
9 Basic perception	143
9.1 Introduction	143
9.2 The difficulty of the visual orientation task in driving	143
9.3 General requirements for a model of visual orientation in driving	145
9.3.1 <i>A theory of object recognition</i>	145
9.3.2 <i>A theory of visual orientation</i>	145
9.4 Constraints shaping DRIVER'S perception	146
9.4.1 <i>Soar constraints</i>	146
9.4.2 <i>Field constraints</i>	148
9.4.3 <i>Eye and head movement constraints</i>	149
9.5 Implementation of low-level perception in DRIVER	149
9.5.1 <i>Lower-Level Perception Module</i>	151
9.5.2 <i>DRIVER's orientation operators</i>	153
9.5.3 <i>Move-attend operator</i>	153
9.5.4 <i>Move-eye operator</i>	154
9.5.5 <i>Move-head operator</i>	156
9.6 Discussion	157
9.7 Notes	159
10 Visual orientation.....	161
10.1 Introduction	161
10.2 Novice and experienced drivers in the approach to an intersection	161
10.3 The visual orientation rules	162
10.3.1 <i>DRIVER's default orientation rules</i>	162
10.3.2 <i>DRIVER's orientation rules for intersections</i>	164
10.3.3 <i>An example of DRIVER's behaviour</i>	166
10.4 Fitting other visual orientation phenomena	167
10.5 Discussion	171
10.6 Notes.....	171
11 Speed control	173
11.1 Introduction	173
11.2 An overview of DRIVER's speed control	173
11.3 Design decisions and discussion	177
11.3.1 <i>Perceptual information in speed control</i>	177
11.3.2 <i>Perceptual information used by DRIVER</i>	179
11.3.3 <i>Using operators for monitoring and updating speed representation</i>	180
11.3.4 <i>The mental model of the external world</i>	181
11.3.5 <i>Traffic rules</i>	184
11.4 Notes.....	186
12 Steering and Lane Keeping	189

12.1 Introduction	189
12.2 An overview of steering and lane keeping in DRIVER	189
12.3 Discussion	191
12.3.1 Cues used	191
12.3.2 Handling curves	194
12.3.3 Differences between the mechanisms for speed control and steering	194
12.4 Notes	195
13 Navigation	197
13.1 Introduction	197
13.2 Navigation in human drivers	198
13.3 Implementation of navigation in DRIVER	199
13.3.1 "Internal" navigation	200
13.3.2 "External" navigation:	202
13.3.3 Plan repair or error recovery in Navigation	202
13.4 Task interruption and resumption in navigation	203
13.5 Discussion and future research	204
13.6 Notes	204
Appendix 1	206
14 Integration and multitasking	211
14.1 Introduction	211
14.2 A trace of DRIVER's behaviour	212
14.3 DRIVER's performance compared to human drivers	220
14.3.1 Fitting observable behaviour	220
14.3.2 Additional fits	222
14.4 DRIVER's multitasking mechanism	226
14.4.1 Mechanisms and knowledge involved in multitasking	226
14.5 Discussion	228
14.5.1 The spontaneity and the specificity of DRIVER's driving behaviour	228
14.5.2 Learning to multitask	229
14.5.3 DRIVER compared to the earlier model	230
14.6 Notes	231
15 Discussion	233
15.1 Introduction	233
15.2 Mapping tasks and behaviours onto Soar	235
15.2.1 Basic driver tasks	235
15.2.2 Attention and visual orientation	235
15.2.3 Motor control: planning and execution	236
15.2.4 Multiple types of goals	236
15.2.5 Parallel processing and multitasking	238
15.2.6 Soar enables both controlled and automatic behaviour in DRIVER	239
15.2.7 External memory and situated action	240
15.2.8 Bottom-up and top-down control	241

15.2.9 <i>Declarative vs procedural knowledge, knowledge compilation</i>	242
15.3 Fundamental problems with Soar and DRIVER	243
15.3.1 <i>Soar's default search control rules</i>	243
15.3.2 <i>Problems with Soar's learning mechanisms</i>	243
15.3.3 <i>Problems with Soar's working memory</i>	249
15.3.4 <i>Perception and motor control</i>	250
15.3.5 <i>Time and timing problems</i>	251
15.3.6 <i>Summary of our problems</i>	251
15.4 Suggestions for future research	252
15.4.1 <i>Extend the driving task domain</i>	252
15.4.2 <i>Representations and mechanisms for working with quantities</i>	252
15.4.3 <i>Memory for frequency of occurrence</i>	253
15.4.4 <i>Emotions</i>	253
15.4.5 <i>Individual differences</i>	255
15.4.6 <i>Incorporate natural language</i>	256
15.4.7 <i>Soar and connectionism</i>	256
15.5 Final conclusions	256
15.6 Notes.....	258
List of Abbreviations.....	261
References	263
Appendix 1: Learning and error correction for external operators.....	277
A.1 Specimen runs	278
A.1.1 <i>First run</i>	278
A.1.2 <i>Second run</i>	279
A.1.3 <i>Third run</i>	280
A.2 Productions for learning from external interaction and error correction	280
Appendix 2: Basic driver operations in more detail	291
B.1 Introduction.....	291
B.1.1 <i>Building the events database</i>	291
B.1.2 <i>Results</i>	293
Samenvatting.....	307

1 Introduction

1.1 Introduction

This study evaluates the practical and theoretical suitability of the cognitive modelling tool Soar for modelling complex dynamic task behaviour. This tool is based on psychological insights in human problem solving (Laird, Newell, & Rosenbloom, 1987; Newell, 1990; Michon & Akyürek, 1993). In fact, one of the claims of the makers of Soar is that Soar may be considered a general theory of human problem solving or, even stronger, a unified theory of cognition. Much of the literature that describes the use of Soar as a cognitive modelling tool focuses on fairly static tasks that do not require interactive real-time problem solving. There is far less literature that describes the cognitive modelling of more dynamic tasks. In this study we will concentrate on complex, interactive real-time problem solving behaviour and examine the extent to which Soar is also suitable for modelling this type of behaviour.

The task that we consider to be eminently suitable as an example of a complex, interactive real-time task is that of driving a car in critical traffic situations. In this study we describe how we developed a cognitive model of driver behaviour. The model is in fact a computer simulation of a car driver able to survive in a simulated traffic world. It handles the elementary driver tasks of navigation, speed control and lane keeping as well as the more basic tasks like visual orientation and motor control.

We aimed for a psychologically valid model of driving behaviour. To give an idea of just what we mean by this, we give here the assumptions and hypotheses that underlie the development of our cognitive driver model: (1) we view the driving task as a collection of (interdependent) sub-tasks that will often be simultaneously active and will also often simultaneously call upon the same resources; this means that in this study we will not concentrate on isolated sub-tasks but instead attempt, wherever possible, to incorporate multiple sub-tasks, focusing primarily on the integration of these tasks and on the multi-tasking mechanisms that make the driving task possible; (2) we are convinced that driver behaviour can be described as problem solving behaviour; each sub-task of the driving task will be dealt with in a problem solving context; (3)

each of the sub-tasks uses not only the cognitive system but also the perceptual and motor system; it is clear that a genuinely psychologically valid driver model should therefore take into account the resources and limitations of these systems; (4) as in every modern cognitive system, the model must be so detailed that, in our case, it is able to survive in a dynamic and interactive, albeit simulated traffic world.

Although we aimed for a psychological valid model of driver behaviour, we did find that it would be too ambitious to aim for a complete coverage of the driving task. We therefore decided to focus on one manoeuvre that exercises the main driver sub-tasks. Manoeuvres that involve a number of sub-tasks and in which many decisions must be taken in a short period include overtaking, merging into a stream, negotiating a complex curve and handling an intersection. The specimen task that we will focus on in this study is the approach to and negotiation of an unordered intersection. Unordered intersections are fairly common in Dutch suburban areas. The right of way at these intersections is not regulated by traffic signs but by a set of rules, the main one being that cars approaching from the right have to be given right of way.

There are several reasons why we focus only on negotiating intersections. In the first place because it involves a range of sub-tasks that, with respect to the necessary skills, cover nearly the whole driver task. Section 1.2 provides an overview of the sub-tasks involved in crossing intersections. We are convinced that most of the insights acquired from this model can be transferred to other tasks. In the second place, negotiating an intersection is one of the hardest tasks to learn, as illustrated in Chapter 4 of this study. For example, almost a third of all remarks made by instructors during driving lessons are about handling intersections. A third and very practical reason is that we possess a unique set of data that describes the behaviour of subjects approaching an intersection and that can be used to validate our driver model. The data come from experiments conducted by De Velde Harsenhorst and Lourens (1987), who used an instrumented car to record eye movements, handling of car devices, speed control and course control.

A final justification to focus on negotiating intersections was that this task was also intensively studied in GIDS, a large European research project that ran parallel to this study at the Traffic Research Centre (Michon, 1993). The aim of this project was to design a generic intelligent driver support system that will help drivers to manage a number of (future) electronic in-car support devices such as electronic route guidance, anti-collision, and car-following systems. It was found in this study that by focusing on only a few main tasks, most of the driver sub-tasks are exercised.

The use of the driving task *as an example* of a complex dynamic task does not mean that we are marginalizing the importance of creating a psychologically valid model of the driving task. Initially this study even started with the as-

signment from the Traffic Research Centre of the University of Groningen to consider a general model of driving behaviour. The ideal approach to this study would be a cognitive-psychological one, in which the driving task is regarded as a complex dynamic task. It was decided at a fairly early stage to opt for the Soar architecture for implementing this model. In the course of this implementation the advantages and disadvantages of using this system became clear and as a result this study evolved into an evaluation study of the suitability of Soar for modelling complex dynamic behaviour. In the following sections of this introductory chapter we will keep to this historical sequence. Section 1.2 will discuss our interest in driver behaviour and the psychological perspective we take. Section 1.3 discusses why Soar was chosen to model driver behaviour. Many reasons are given in favour of Soar in this section, but we will also discuss potential problems. Section 1.4 looks into these problems, thereby explaining the first objective of the study. The final section describes the structure of the thesis.

1.2 Driver behaviour

Our motivation to focus on driver behaviour is that the driving task provides an excellent domain to study decision-making and human problem solving in interactive, dynamic situations¹. Problem solving as described in the cognitive psychology literature is usually concerned with problems that are of a somewhat less dynamic nature than the driving task. The problem solving involved in car driving is exemplary for all those tasks that require real-time decision-making under time pressure in a dynamic environment, interaction with other intelligent agents, and an extensive use of perceptual and motor systems.

In order to clarify the proposition that car driving involves regular problem solving, the following discusses the main driver sub-tasks. We shall illustrate how each of these tasks requires some form of problem solving. In addition, we will illustrate how nearly every sub-task requires actions from the perceptual and motor systems. The approach and negotiation of intersections will be our example task.

Visual orientation. A major and continuous task facing drivers in all manoeuvres is to look in the 'right' direction at the 'right' time. Empirical research shows that novice drivers find this task the most difficult (De Velde Harsenhorst & Lourens, 1987; Mourant and Rockwell, 1972; see also Chapter four). In addition, it seems that perceptual errors are the main contributory factor in causing accidents (Smiley, 1989). The main reason for this difficulty is that people can see only a small part of the visual field sharply. The major part of the visual field provides useful but fairly vague information.² Depending on the complexity of the environment and task, a certain number of eye movements must be made to obtain a comprehensive picture. However, the time it takes to make an eye movement is relatively long – about a fifth of a second. It is

therefore nearly impossible to sample the entire environment in the few seconds before reaching an intersection if one considers that (a) the relevant field covers the left, right and front arm of the intersection and (b) other traffic participants are usually moving too. It requires genuine problem solving to look in the 'right' direction at the 'right' time³.

Speed control. Another continuous problem facing drivers in critical situations is to select the right speed and acceleration. For example, when approaching an intersection it is obviously important to choose the right crossing speed in order to avoid other road users. To solve this problem the driver is required to interpret observed traffic objects (and their behaviour) within the context of the traffic rules that are in force in a particular situation and then decide how this will affect his own speed and acceleration. It is clear that this decision task is closely related to both the visual orientation task and the motor control task.

Lane keeping and curve handling. A third continuous problem is to keep the car within the lane and avoid obstacles. When approaching an intersection there is the additional problem of starting a left-hand or right-hand turn at the right place and finding the appropriate steering angle in the curve. If we examine data obtained from observing novice drivers, we see that all these problems require considerable learning curves and that this task too is closely related to the visual orientation and motor control tasks.

Motor control in handling car devices. The fourth problem is handling the car in order to steer or change speed. For novice drivers the latter problem especially requires full-blown problem solving in the form of complex motor planning and motor execution. For example, a change from third to first gear requires about 15 discrete motor actions in order to manipulate the gear-stick and the accelerator, clutch and brake pedals (see Chapter 4). To add to this complexity it must be remembered that most of the time more than one limb is moving at the same time.

Navigation. A fifth problem, especially in unfamiliar environments, is deciding which direction to turn at the approaching intersection. This requires problem solving in the traditional sense (relatively long-term planning), though the driving context adds a few interesting aspects, e.g. matching internal network representation with the external network, visually sampling the world in order to determine where one is and replanning when roads on the planned route are blocked.

Coordination and multitasking. The sixth problem is a universal one – to do the 'right' thing at the 'right' moment. People feel this is a problem particularly

when they have to carry out many different tasks at the same time in critical situations. Approaching an intersection is a good example of a task where many different sub-tasks need to be carried out in a very short time. Each sub-task occasionally requires a discrete visual or motor action and all sub-tasks seem to place a constant burden on central cognition. It is clear that drivers are multitasking and that a process is required that integrates and coordinates all the tasks. This process involves choosing and planning the order of tasks and actions and is thus in itself a problem solving task.

The difficulty of coordinating multiple sub-tasks is shown in the behaviour of novice drivers. Apart from the fact that each of the sub-tasks has to be learned, it is also necessary to learn to multitask each one. One of the symptoms of this difficulty is the slow speed at which novice drivers approach an intersection in order to compensate for the large number of decisions which need to be taken (De Velde Harsenhorst & Lourens, 1987, 1988)⁴.

The idea to approach the driver task from a problem solving perspective is relatively new. Michon (1976, 1985, 1989) was one of the first to explicitly argue that the driving task is an ideal and challenging application area for psychologists interested in complex dynamic problem solving. He claimed that it would be a worthwhile endeavour to consider car-driving as a problem solving and rule-based task that requires both short-term and long-term planning. Other authors who have also proceeded in this direction are Reason (1987), Rasmussen (1985, 1987), Hale, Stoop, and Hommels (1990) and Aasman (1988, 1991). The fact that this perspective is a relatively new one is illustrated by an overview of prevalent models in traffic science presented in the following section.

1.2.1 Models in traffic science

Modelling driver behaviour has been a popular activity in traffic science. Michon (1985, 1989) provides a critical overview of the theories and models that have been created in the last few decades. He categorises the models using a simple two-way classification system (see Table 1).

The first distinction made is between input-output (or stimulus-response) and internal state (or psychological) models. Input-output models describe driver behaviour without reference to the internal or psychological state of the driver. They describe the relation between the external conditions (traffic environment + state of car) and the actual or desired behaviour. Internal state models describe how psychological variables and constituents of the driver influence driving behaviour, but they usually ignore very specific driving tasks or driving conditions.

The second distinction made is between taxonomic and functional models. Taxonomic models are 'static' models that describe relations between input-output factors or internal psychological variables, but lack a process descrip-

tion. Functional models on the other hand are more 'dynamic' models that describe behavioural processes. A lengthy treatment of all driver behaviour models falls outside the scope of this introduction, but we will provide an example for each cell.

Table 1. Overview of theories and models in traffic science (adapted from Michon, 1985)

	taxonomic	functional
input-output (behavioural)	task analysis	mechanistic models adaptive control models · servo-control · information flow control
internal state (psychological)	trait models	motivational models cognitive process models

Task analysis

The best example of a task analysis is that by McKnight and Adams (1970a, 1970b). It is an exhaustive inventory of automobile driving and essentially describes the facts about the driving task (driver task requirements), behaviour requirements (performance objectives) and ability requirements. The analysis divides the driving task into some 45 sub-tasks, composed of more than 1700 elementary tasks altogether. It is admirable in its completeness and certainly useful as a basis for any cognitive model of car-driving behaviour, as it describes the normative and observable behaviour that any model should display. However, despite this completeness, the analysis is not directly applicable in a cognitive model because (a) the dynamic and temporal aspects are missing; nowhere do the authors specify exactly when or in what order actions should be carried out and (b) there is no reference to psychological variables or processes or any human information processing constraints.

Trait model

A typical example of a trait model was created by Fleishman (1967) who developed a factorial model for perceptual, cognitive and motor skills. Such skills are the result of combining elementary traits (e.g. reaction speed and spatial orientation). A trait model can be used indirectly to build a cognitive model because it can predict car-driving skills. However, its usefulness is limited because it pays no attention to specific traffic situations (such as in the task analysis), the dynamics of the task or the driving task as a whole.

Mechanistic models

There are several types of models that can be placed in the functional input/output cell. Those most relevant to this study are the information flow

control models in the form of computer simulations of driver behaviour. The first to build such simulations were Kidd and Laughery (1964) and Wolf and Barrett (1978a). Their simulations performed basic driver tasks such as crossing an intersection or merging. However, one of their drawbacks was that the behaviour of the other traffic participants was relatively fixed, in other words they were not endowed with intelligence. This is a drawback if we realise that driving is an interactive task in which drivers influence each other's behaviour.

These drawbacks were corrected in later simulations or 'small worlds' (Aasman, 1986; Reece, 1988, 1992; Wierda & Aasman, 1988; Van Winsum & Van Wolffelaar, 1993). Small worlds consist of a number of roads and traffic participants (currently only car drivers and cyclists), with each participant being provided with a set of traffic and behaviour rules. In general, all the participants must follow the same rules, though this is not absolutely necessary. By varying parameters in rules or by adding or removing rules, individual differences in behaviour can be generated. The behaviour in these small worlds is far more realistic than in the earlier simulations because all participants interact now intelligently and in real time. An interesting aspect of all these simulations is that control of one of the cars is taken over by another type of intelligence – in Wierda and Aasman (1988) by a child who learns to guide a bicycle safely through the simulated world by manipulating a keyboard; in Van Winsum and Van Wolffelaar (1993) by a human being in an advanced driving simulator; in Reece (1992) by a robot that includes a detailed model of intelligent visual orientation; and in Aasman (1992) by the cognitive model described later in this study.

We do not consider simulations of traffic participants (or agents) within the small world to be cognitive models. The most important reason is that the agents are not restricted by any human constraints. There are no working memory limits, each agent in principle has 360 degrees of sight and is able to implement a desired change in speed at the next clock tick in the simulation. In Chapter 6 we will explain in more detail why the agents are not cognitive models. Despite the fact that we do not see computer simulations as cognitive models, they are extremely interesting because they provide the cognitive model (or other types of intelligence) with a natural environment.

Cognitive process models

There is a whole host of theories that look at cognitive functions (beliefs, emotions, intentions) in the context of driving. Most of the theories focus on the driver's risk handling and threat avoidance, for example Taylor's (1964) risk-speed compensation model, Wilde's (1982) risk homeostasis theory, risk threshold theories by Klebelsberg (1971) and Näätänen and Summala (1974, 1976), Fuller's (1984) threat avoidance model and Van der Molen and Bötti-

chers's risk model (1988). Other theories focus on the relation between beliefs, attitudes and intentions and actual behaviour (Ajzen & Fishbein, 1977) or on ecological factors in driving (Van der Molen, 1983).

Special mention must be made here of yet another type of theory that attempts to explain driver behaviour – especially errors in driving – in a cognitive framework. Rasmussen (1985, 1987) and Reason (1987) in particular have conveniently introduced the distinction between knowledge-based, rule-based and skill-based performance to describe operator activities in complex tasks and apply it to the driving task².

Michon (1985, 1989) has three general points of criticism with respect to these cognitive models. In the first place, some models overly aggregate driver behaviour instead of predicting individual behaviour. These models cannot be applied to individual drivers or specific driving situations. Wilde's model is an example of this. A second remark is that most of the cognitive process models confuse intentional and procedural levels. The distinction between levels of explanation was made by Dennett (1978, 1987). Intentional explanations of behaviour are explanations in terms of beliefs, emotions and intentions. However, these concepts are themselves products of lower-level cognitive functions and procedures. Cognitive psychologists are especially interested in these lower-level functions that specify the procedures that generate behaviour, including the behaviour that we recognise as beliefs, emotions and intentions. None of the above models really delves into these lower-level functions. In the third place, Michon argues that most cognitive process models do not use insights derived from cognitive science (with the exception of a theory like Reason's). Thus we do not find theories that treat driving as problem solving or enable driver behaviour to be described in terms of rule-based behaviour.

1.2.2 The psychological perspective of this study

Following this description of the problem-solving nature of the driving task and the existing models of driver behaviour we now summarise the psychological perspective that we take in the design of our model of driving behaviour.

(1) The driving task consists of a large number of sub-tasks that are active simultaneously.

It will be clear that the driver task can be regarded as a collection of sub-tasks. In this study an attempt is made to incorporate and integrate a number of important sub-tasks in a cognitive model. We also see that in many situations multiple sub-tasks are simultaneously active. The main emphasis in this study is therefore directed towards integrating these tasks and towards the multitasking mechanisms that make it possible for several tasks to be simultaneously active.

Many of the theories and models discussed above neglect this important point and treat separate sub-tasks in isolation. It is entirely legitimate to do so if it helps to predict aspects of the driving task. However, in this study we wish to focus on the interplay between all sub-tasks.

(2) Describe all driver sub-tasks in terms of problem solving

A second starting-point is that we shall regard all these sub-tasks as problem solving tasks. Even the multitasking of all these sub-tasks will be treated as a problem solving task. The precise meaning of the statement that "a task can be described in problem solving terms" is discussed extensively in Chapter 2. We saw in our overview of the models in traffic science that the problem solving viewpoint is uncommon in traffic science. What we do see, however, is that a number of models describe driving behaviour from a rule-based viewpoint. Examples of this are the taxonomic models of McKnight and some of the mechanistic models and theories developed by Rasmussen (1985, 1987) and Reason (1987). Almost all cognitive models of problem solving (in the symbolic tradition) are rule-based. Soar, the architecture in which we have implemented the driving task, is also a rule-based system. In this sense, then, our model of driving behaviour is not unique.

(3) The driver model takes into account the main resources and the main limitations of the human cognitive, perceptual and motor systems

In some instances it is possible to accurately predict aspects of driving behaviour by modelling sub-tasks of the driver task without taking into account the limitations of the cognitive system (working memory size, memory speed and access), limitations of the visual field (restricted fields and relatively slow eye movements) or limitations of the motor systems. For example, there are models that provide accurate predictions of how people control their speed when approaching a crossing (Van der Horst 1990) or of how people use visual cues in straight road driving (Riemersma, 1987) without involving human limitations in visual orientation and motor behaviour.

In this study, however, our aim is to look at driver behaviour as a whole and we therefore incorporate a number of important limitations of the cognitive, perceptual and motor systems. Our main motivation comes from our earlier list of driver sub-tasks where we showed that the many sub-tasks (a) are highly dependent on perception and motor systems and (b) often even call on these systems simultaneously.

(4) The model must be able to survive in a dynamic, interactive, simulated traffic world

One of the characteristics of modern cognitive models is that they have been created in the form of a computer model. Most of theories discussed in our

overview cannot be tested in a dynamic interactive traffic world because they are not detailed enough to be implemented (with the exception of the information flow models). Again, this is not a criticism of the models. They may well be extremely useful for aggregating behaviour, for example. However, in this study we want our model to be detailed enough to be tested in a modern and “natural” small world simulation.

1.3 Soar

Our model of car driver behaviour has been implemented using the cognitive modelling tool Soar. One question to be answered is why Soar was selected. As this is a complex question, we will split it up into three simpler questions. First, why make a computational model in the first place? Second, what are Soar’s properties that make it so suitable for modelling driver behaviour? Lastly, could we have used another architecture or cognitive modelling tool instead of Soar?

The question why a cognitive scientist formulates a model of behaviour in terms of a computational model is asked less and less often. Until recently one could not discuss a computational model without dedicating a few pages or chapters to this question. However, if we examine a recent cognitive science handbook like *Foundations of Cognitive Science* (Posner, 1989), we see that only the introductory chapters devote some attention to this question. The remaining chapters simply discuss computational models without additional justification. This is not surprising, given the current conception of cognitive science as “the study of intelligence and intelligent systems, with particular reference to intelligent behaviour as computation” (Simon & Kaplan, 1989). Computer implementations of theories and computer simulation are thus important tools in cognitive science. They are used to express theories and to test theories in complex situations that could never be simulated by hand or formulated by explicit, i.e. analytic mathematical means. In addition they also play an important role in the exploratory phase of designing new theories, as they force theorists to specify their ideas to the implementation level⁶.

1.3.1 The basic Soar mechanisms

The next question was ‘What are Soar’s intrinsic properties that make it suitable for modelling (driver) behaviour?’ To answer this question we must return to the psychological perspective discussed in the previous section. We argued that we want to approach the driving task from the *problem solving* perspective. The main reason for choosing Soar is that the Soar system is based on concepts and insights derived from the work of Newell and Simon in the field of *human problem solving* (Laird, Newell, & Rosenbloom, 1987; Newell & Simon, 1972). Because of this problem solving orientation, cognitive psychologists use Soar for a large number of topics. Examples are modelling natural language processing, attention, concept formation, syllogistic

reasoning, musical cognition, error recovery, aspects of memory, planning, learning, etc. (see Lewis et al. (1990) for an overview of Soar's handling of cognitive phenomena and Steier et al. (1987) for Soar's handling of several types of learning).

The claim that Soar is based on concepts and insights derived from the psychology of human problem solving requires some explanation for readers not familiar with Soar. The main mechanisms of Soar architecture and its relation to psychological concepts are therefore explained below.

Production systems

Soar is a production system. Production systems (also called rule-based systems) have been used in cognitive science⁷ for many years, but were first proposed as a serious tool for describing human behaviour by Newell and Simon in their book *Human problem solving* (Newell & Simon, 1972). Newell and Simon described human performance in chess, cryptarithmic and logic by translating the introspections of subjects and observations of the behaviour of subjects into simple, very low-level if-then rules. These rules regenerated the behaviour of subjects but could only be simulated by hand because technology was not yet far enough advanced at that time to enable other methods to be used. It was only recently that the rules for the cryptarithmic task, described in Newell and Simon (1972), were successfully implemented in Soar (Newell, 1990).

Production systems enable cognitive scientists to generate behavioural predictions both on a qualitative and a quantitative level. Qualitative predictions relate to the incidence and order of behavioural actions. Quantitative predictions relate mainly to the temporal aspects of behaviour, for example to time of onset and duration of actions. Most production systems generate temporal predictions by counting the number of matching rounds required to finish a sub-task (see following sections for more detail).

How production systems work

In general a production system consist of a working memory, a set of rules, a matching mechanism, and a conflict resolution mechanism. *Working memory* is the storage for *temporary* information while *rules* in a production system embody *permanent knowledge*. An individual rule consists of a condition part and an action part (also referred to as 'if' and 'then' part or 'left-hand-side' and 'right-hand-side'). A *matching mechanism* matches all rules against the information in working memory. All rules whose condition sides are satisfied are placed in a so-called conflict set. One of the rules in the conflict-set is then chosen by applying a set of *conflict resolution rules* to the rules in the conflict set. These conflict resolution rules are in fact task-independent meta-rules that prioritise the rules in the conflict set on criteria such as the specificity of a rule, or the strength, or the success rate of a rule in the past. Once a rule is

chosen its action part is executed, thereby usually modifying the contents of working memory by adding or deleting information.⁸

Soar as a production system

Soar's most important deviation from this general scheme is that it uses a *preference based decision mechanism* instead of a conflict resolution mechanism. In contrast to other production systems, Soar allows every rule that matches to add new information to working memory. Chaos and overflow in working memory are prevented by Soar's *preference mechanism*. Soar-rules not only add ordinary data elements (working memory elements) to working memory but also *preferences*. These are distinct representations that express the desirability or 'preference' of a working memory element. Soar's decision mechanism uses a simple preference semantics to choose the 'most preferred' objects; all other objects are removed from working memory. The advantage of this scheme is that in the same time interval more knowledge can be applied to process information in working memory. The importance of this mechanism is explained in the following paragraphs.

One additional feature of Soar is that working memory is not only updated by productions. Information from the external world can also enter Soar's working memory via sensors and special input/output mechanisms. In Soar this information is added asynchronously and in principle independently of other active processes in the working memory. This feature provides the hook to model perceptual processes and bodily feedback mechanisms.

Soar as the embodiment of the Problem Space Hypothesis

The use of production systems is a basic element in Newell and Simon's work. Another corner stone of their work is the *Problem Space Hypothesis* (see Newell & Simon, 1972). This hypothesis states that all intelligent behaviour can be described as problem solving, where problem solving is defined as heuristic search through problem spaces. In Soar, the essential elements of this hypothesis are implemented as a set of architectural principles on top of Soar's production system mechanism. The following will explain this hypothesis and the relation, or rather the marriage, between the hypothesis and the production system Soar in more detail.

Problem space

Conceptually, a problem space incorporates all the task-specific knowledge that a system has about a particular task. In Soar, a problem space is a collection of states, operators, and rules. Both states and operators are data structures. A *state* is a datastructure that represents the current situation in the problem solving process. *Operators* are datastructures that carry (implicit) instructions for how a given state should be altered by adding, deleting or altering attributes of the state. The role of rules in Soar mainly revolves around the use of operators. First, rules *generate* (multiple) operators for the

current state. Secondly, rules *select* and de-select operators (from the set of operators that are generated for the current state) by adding preferences to Soar's working memory. Thirdly, they change states by *applying* operators and fourthly, they determine when an operator is *terminated*.

In the terminology of the Problem Space Hypothesis and in Soar, *problem solving* consists of finding a chain of operators that transform an *undesired initial state* into a *desired end state* or, in other words, finding the succession of steps that leads to a goal.

Impasses

Problem solving generates sub-problems. This is what makes problems problems. Only trivial actions and problems that have already been solved on a previous occasion immediately reveal the path from initial state to goal state. Usually, however, the problem-solver is confronted with impasses on the way to the goal state. An impasse is a genuine problem. It is a situation in which the next action to be taken is not immediately evident and therefore requires problem solving.

Soar, as a genuine problem solver, also encounters impasses. Soar distinguishes four types of impasse, the most important of which are (a) Operator No-Change Impasse – there is an operator for the current situation but no knowledge about how to apply it, and (b) Tie Impasse – there are several equally attractive operators that can be applied to the present situation. Consequently, the problem-solver is faced with a choice impasse.

Universal sub-goaling

An impasse implies that not enough knowledge is available to progress towards the goal state. The solution offered by Newell and his associates is the principle of Universal Sub-goaling (Laird, Rosenbloom & Newell, 1986). For each and every impasse Soar creates a new problem that is formally represented as a sub-goal. The system must first solve this new, presumably easier problem, before it can continue solving the original one. The new problem must be solved in a new problem space. If in turn another impasse occurs in the subsidiary problem space, Soar will again create an appropriate sub-goal. It will continue to do so until a solution is found for the prevailing impasse at a deep enough level. Alternatively, Soar may find that it does not have enough data to reach a solution, in which case it will stop and leave the problem unsolved. A system that operates in this fashion may be considered a recursive problem-solver⁹. A special feature of the technique described here is that the problem space in which a solution is attempted may be different from the original problem space. In fact the problem space may be different at each subsequent level.

Heuristic search

Problem solving is not only searching, but also involves *heuristic* search through problem spaces. This implies that in a problem space there must be additional knowledge that allows the problem-solver to order, accept or reject operators, given a particular problem state. A system without heuristic knowledge is forever bound to solve problems by applying all available operators in a trial-and-error fashion.¹⁰ Soar achieves heuristic search by employing the earlier described preference mechanism.

Chunking

Chunking is the 'natural' learning mechanism that derives from the theory of human problem solving underlying Soar. Chunking produces a new rule by summarizing the solution to an impasse. The condition side of a new rule consists of the state that existed when the impasse was encountered, while the right-hand side (the action part) of the new rule comprises the knowledge that Soar has extracted from the available knowledge that was required to solve the sub-goal. It has been shown that this elementary mechanism is rich enough to account for numerous findings from the psychology of learning and from the area of machine learning (Steier, Laird, Rosenbloom, Flynn, Golding, Polk, Shivers, Unruh & Yost, 1987; Rosenbloom & Aasman, 1992).

In the above description we noted that that Soar is the marriage between a production system and the problem space hypothesis. Table 2 presents an overview of the relation between the two.

Soar as a Unified Theory of Cognition

The question of whether Soar is psychologically relevant is in a sense trivial. It has been answered in Newell and Simon (1972), which treats the problem space hypothesis as the foundation of the theory of human problem solving and emphasises at the same time the significance of production systems as a 'calculus' for its formalization.

Although Soar is an intrinsic problem solving architecture, the question of whether and to what extent it is also a psychological theory of human behaviour remains a pertinent one. Newell held clear views on this matter. He claimed that Soar is not only a problem solving architecture and a powerful implementation language for AI-type problems, but also a candidate unified theory of cognition.

"Psychology has arrived at the possibility of unified theories of cognition - theories that gain their power by having a single system of mechanisms that operate together to produce the full range of human cognition..." (Newell, 1990, p. 1)

'Full range' in this context means that the architecture should be able to do tasks ranging from highly routine to extremely difficult open-ended problems. It should use the same uniform representations and mechanisms for all perceptual, motor and cognitive tasks and should use all the problem solving methods and representations to solve these tasks. Finally, learning should be an integral part of all aspects of the tasks (Laird, Newell & Rosenbloom, 1987).

Table 2. Summary of correspondence between the Problem Space Hypothesis and Soar.

Problem space hypothesis	The production system SOAR
Behaviour is searching for <i>goals</i> in <i>problem spaces</i>	Productions generate <i>problem spaces</i> for goals
Problem spaces consist of <i>states</i> and <i>operators</i>	Productions generate <i>states</i> and <i>operators</i> within problem spaces
<i>Problem solving</i> consists of finding the trajectory from an <i>initial state</i> to a <i>desired state</i>	Productions (a) change states by applying operators, (b) <i>evaluate states and operators</i> and (c) <i>recognise desired states</i>
Searching in problem spaces is <i>heuristic</i>	Productions generate <i>preferences</i> for objects (problem spaces, states and operators) that guide the search in problem spaces (Soar chooses objects on the basis of simple preference semantics).
An <i>impasse</i> occurring during the problem solving process generates a sub-goal.	Impasses arise if no decisions can be made on the basis of the prevailing preferences; Soar creates a sub-goal for every type of impasse (<i>Universal sub-goaling</i>); Soar has a set of default productions to deal with impasses.
	<i>Chunking</i> - When an impasse is solved Soar creates a new production (chunk) that summarises the conditions leading to the impasse and the actions that solved the impasse.

Why and to what extent Soar is a unified theory of cognition (UTC) is argued in great detail in Newell's monograph "Unified Theories of Cognition" (1990). For the reactions of the cognitive science community to this attempt at unification, see Behavioural and Brain Sciences (1992, pp. 425-492). These reactions are generally fairly positive, although there are many critical notes. Newell was the first to admit that Soar is only a *first step* towards a UTC and therefore included Soar in a list of *candidate UTCs*, of which Anderson ACT* (Anderson, 1983, 1993) is also an important member.

Newell supported his claim by pointing to (1) the features described above, which make Soar an ideal problem-solver and (2) the broad scope of cognitive phenomena ranging from reaction-time tasks to complex arithmetic puzzles that Soar can handle without changing its elementary mechanisms for every new task. Incorporating new tasks and behaviours into the Soar architecture

without changing existing Soar mechanisms or adding new mechanisms is fundamental in Newell's philosophy. He argues that psychologists should stop designing a new underlying framework or architecture for every new task they tackle. If science continues on this path we will never see the integration that other sciences are striving for¹¹.

The Soar research tradition

At present (summer 1994), Soar a public domain program, actively being used by about 150 cognitive scientists world-wide. The program is maintained by a team of programmers, who also provide support for users. Most users support Newell's philosophy that one should try to get as far as possible without changing the existing architecture. Only if it is clear that Soar is inadequate in some respect may a more or less democratic decision be taken to implement the necessary changes^{12 13}.

As stated earlier, the research topics for cognitive psychologists in the Soar community include natural-language processing, attention, concept formation, syllogistic reasoning, musical cognition, error recovery, memory phenomena, planning and learning. This group's ideal is for all these tasks and phenomena to be incorporated seamlessly into Soar in the future. Our present objective – to model driver behaviour and in a more general sense to model dynamic task performance – fits in well with this research tradition. When implementing the driver task, we tried to stay as close as possible to Soar's basic mechanisms, though we could not avoid introducing some other mechanisms.

1.3.2 Other cognitive modelling tools

The question '*Why chose Soar?*' implies we should also explain why we did not chose another problem solving architecture, cognitive modelling tool, cognitive architecture or cognitive framework.¹⁴ There are other problem solving architectures capable of learning – BBS (Hayes-Roth, 1984), Prodigy (Carbonell et al., 1989), Theo (Mitchell et al., 1989) and Max (Kuokka, 1988, 1990). The main reasons we did not opt for these models are that (a) they were not available when the study started and, more importantly, (b) psychological validity is not one of their main priorities.

A cognitive architecture that does make a psychological claim is Anderson's ACT* (Anderson, 1983). ACT* is a goal-based production system (like Soar, though not explicitly based on the problem space hypothesis) and is strongly directed at real-time performance and learning. However, one pragmatic reason for not using ACT* was that at the time we started modelling¹⁵ only partially implemented versions existed at Carnegie Mellon University and only a few local graduate students knew how to use these. Soar, however, is a

public domain program with a real maintenance team that receives and handles bug reports and releases new versions of Soar in a centralised way.

1.4 Using Soar

Section 1.3 focused on the positive aspects of Soar. However, there are a number of problems that we encountered in implementing driver behaviour. Some of these problems were foreseen, but others were not revealed until we started implementing the driver model in Soar. For a better understanding of these problems we will first discuss Soar at the level of *immediate behaviour*, i.e. behaviour at the reaction-time level.

1.4.1 The Soar theory of immediate behaviour or the "minimal scheme for immediate responses"

Newell argued that the total set of constraints that shape a cognitive architecture should apply to behavioural phenomena in several time domains. That is, the architectural constraints that make it possible to model tasks in the range from five to ten seconds should also be valid for elementary perceptual, motor and cognitive tasks that range from one to two seconds. Thus, a theory that explains chess behaviour on the level of moves but whose constraints do not enable it to predict behaviour in lower-level stimulus-response tasks is incomplete. The reverse is also true. A behavioural model that predicts behaviour in a simple reaction-time task but cannot be used to model chess behaviour is incomplete, to say the least.

The mechanisms built into Soar apply both to long-term and short-term tasks. The same mechanisms that make it possible to predict behaviour on Sternberg tasks are also the behavioural building blocks in a complicated search task like cryptarithmic or a syllogistic reasoning task (see Newell, 1990, for examples).

This is an important aspect in modelling driver behaviour, as driving consists of tasks in different time domains. The distinction between short-term operational tasks, medium-long-term tactical tasks and long-term strategic tasks is often used in traffic science (see Michon, 1971, 1985; Moraal, 1980; Van der Molen & Bötticher, 1988). Steering as a reaction to lateral deviation, speed control by manipulating the brakes, accelerator and clutch, reacting to a car on a collision course and deciding where to look next in a critical situation are all operational tasks in the one-second domain. Major driving manoeuvres such as overtaking or crossing an intersection are examples of tactical tasks in the 5 to 10-second domain. Navigation is an example of a long-term strategic task.

Soar's behaviour in short-term tasks is described by the *Soar theory of minimal behaviour or the minimal scheme for immediate responses* (Newell, 1990, Chapter 5). This scheme specifies the basic actions and perceptual, cognitive and

motor actions in the *perceive – decide – act* loop. Column 2 of Table 3 lists these functions. The theory also specifies the mapping of these actions and functions onto Soar. The third column of Table 3 specifies whether an action is carried out by productions only or whether an operator is required and the fourth column states the duration of the action and whether the action must always exist.

The main importance of the “minimal scheme for immediate responses” (MSIR) is that claims are made about which functions can be achieved by production sequences and what must be achieved by operators. In Soar productions fire in parallel but there is always only one current operator. This is therefore implicitly a claim about which functions can be carried out in parallel and which functions risk becoming bottlenecks because they have to be carried out in sequence. The justification for the fact that some functions are performed by operators and others by productions falls outside the scope of this chapter. (see Chapter 5 of Newell’s UTC (Newell, 1990)).

Table 3: The relation between psychological functions and Soar mechanisms in the basic perceive-decide-act cycle.

	Psychological function	Soar	Duration
Perception <i>perceive</i>	PERCEIVE sense the environment		uncontrolled, decreases with intensity
	ENCODE perceptual parsing of input	productions	increase with complexity, not required for detection.
Central cognition <i>decide</i>	ATTEND focus on input, select input	operator	delay + search: decreases with preparation
	COMPREHEND analyse for significance	operator	increases with complexity, decreases with preparation (must exist)
	TASK set task to be done	operator	decreases with preparation (to null)
	INTEND determine and release action (high-level motor command)	operator	decreases with preparation (must exist)
Motor system <i>act</i>	DECODE convert high-level motor commands	productions	increase with complexity, decrease with preparation (to zero)
	MOVE make commanded movement		uncontrolled, decreases with force, imprecision and preparation

There is one other aspect of the theory of immediate behaviour worth mentioning here. Earlier we said that Soar allows for asynchronous input and output. What was not stated explicitly at that point is that both input and output occur only in the top-level problem space. This space, from which all sub-goals arise, is also called the *Base Level Space*.

According to the theory of immediate behaviour, input and output must take place here because otherwise their continuity would be endangered. Suppose that new perceptual information (or motor feedback) arrives in a sub-space. Since sub-spaces are terminated and removed from the working memory every time a sub-goal is reached, the perceptual information would also be removed. The same applies also to the operators that were mentioned in the "minimal scheme of immediate responses".

1.4.2 Unresolved research issues in Soar

We want to stress that in this study we will comply with the requirements of the theory of immediate behaviour. This means that in our model (a) input and output take place in the base level space (BLS) and (b) all attend and intend operators function in the BLS. The problems generated by this decision are described in the following sections.

Multitasking

Human drivers seem capable of handling multiple tasks simultaneously and we argued earlier that multitasking certainly *is* a requirement in the driving task. One implication for our driver model, and thus for Soar, is that multiple sub-tasks and sub-goals must be represented in the working memory at the same time. In addition, we argued that the representation for these sub-tasks must reside in the base level space. The reasoning is analogous to the reasoning that input and output should occur in the base level space. Suppose multiple tasks were represented in multiple sub-goals and one of the highest sub-goals in the goal-hierarchy was solved. This would imply that all sub-tasks would be lost because all lower sub-goals would be removed from the working memory. Chapter 2 deals with this problem and suggests solutions.

Interruptibility, deep goal stacks vs handling at top level

Related to the foregoing is the issue of interruptibility of goal stacks. When Soar is confronted with a task that requires search, deep goal stacks usually arise. Setting up such a goal stack requires a number of elaboration and decision cycles and therefore takes a considerable amount of time. Problems arise when during deep search actions have to take place in the top-space. For example, suppose that Soar is engaged in a task that generates deep goal stacks and then finds itself in a situation that requires frequent interactions with the external world (for example: in order to survive, a car driver must frequently perform steering actions, motor actions to control speed and make frequent eye movements). According to the *Soar theory of immediate behaviour*, all eye movements and motor actions involve *intend* and *attend* operators that *must* occur in the base level space. This means the main task is often interrupted and there is hardly any time to build up a deep goal stack.

Soar Input and Output

Driver behaviour is partly determined by the constraints of the human perceptual and motor systems. However, Soar in its current form has little to say about the particular constraints involved. The formulation of a theory concerning the transition from analog input to symbols, object recognition, the distinction between foveal and peripheral vision and the lower-level attention mechanism is still in its early stages, though some important work has already been done, particularly on the subject of lower-level attention (Wiesmeyer, 1992). The development of a theory on lower-level motor control and feedback mechanisms has not really started yet. Newell warns us that as far as motor control is concerned Soar only provides the interface from cognition to motor control.

Despite Soar's current shortcomings in these areas, we decided to include a number of important constraints in the perception and motor control units of our model of driver behaviour. The perceptual constraints we implemented in our model are: (1) a distinction between functional and peripheral visual fields; (2) mechanisms to attend and select objects from the functional field; and (3) a simple model to control eye and head movements so that we can at least simulate the time it takes to shift the field of vision. The motor constraints included in our model primarily define an execution mechanism for motor commands that can be given from Soar's working memory. This mechanism is required to provide an external representation of the position of the extremities and to simulate the time it takes to move an extremity over a certain distance. These constraints are primarily implemented in Lisp and are described in Chapters 7 and 9.

Learning from external interaction

Chunking, Soar's learning mechanism, is especially appropriate for learning in a closed world. A closed world is jargon for a system that solves goals using all internally available knowledge and in doing so does not, or may not, use (new) information from the external world. Soar in closed-world mode learns whilst performing tasks how to select the right operator or how to implement new operators; and it can even learn new problem spaces. This learning takes place by sub-goaling into other problem spaces, which implies that the knowledge in the newly learned rules is already available somewhere in these other problem spaces. It might be said that Soar only learns to perform tasks more efficiently if it uses only information that is already present in one of the other problem spaces and is already part of its own closed world.

The implicit criticism that Soar cannot learn anything new is not justified. Many Soar systems already interact with the external world and are consequently able to acquire new knowledge. There are systems that can learn the acquisition of new tasks from external specifications, take external guidance or

control a robot arm with a vision system (Golding, Rosenbloom & Laird, 1987; Laird, 1990; Laird & Rosenbloom, 1990; Laird, Hucka, Yager & Tuck, 1990). However, learning from interaction with the external world has remained extremely limited to date and the principles of learning from interaction are not properly understood yet. The reasons why learning from external interaction is a problem and possible solutions are discussed in later chapters.

1.4.3 Evaluating Soar's suitability

Having dealt with the reasons for choosing Soar and the problems we shall encounter in implementing a cognitive model of driving behaviour, we return now to the objective of this dissertation, namely an evaluation of Soar's practical and theoretical suitability for implementing complex dynamic tasks.

In the chapters in which we develop the model of driving behaviour we shall concentrate primarily on implementing the various aspects of driving behaviour. The practical and theoretical problems will frequently be touched upon. The final evaluation is not made until the final chapter of this study. In that chapter we will give an overview of both those aspects of the driving task which fit naturally into the Soar architecture and the practical and theoretical problems involved in implementing a driver model.

1.5 Layout of the study

This study consists of two parts. Part I lays the groundwork for Part II, which discusses the development of the actual driver model. Two of the three remaining chapters in Part I were published previously but have been included because they are vital to an understanding of the choices made in building the actual cognitive model.

1.5.1 Part I

Chapter 2, published previously as Aasman and Michon (1992), describes our first attempt to build a cognitive model of car driver behaviour. This model focused only on the problem of multiple goals and multitasking within driving and Soar. It did not address the issues of visual orientation or motor control. The model demonstrates that task-switching, task interruption, automatic and controlled processing, as well as bottom-up and top-down information processing, all come fairly naturally to Soar. There is, however, a serious problem that we come up against in this chapter and this relates to multitasking. We see that Soar's current default rules for dealing with tie impasses are far too susceptible to interruptions.

Chapter 3, published previously as Aasman and Akyürek (1992), is dedicated to solving the biggest problem identified in the previous chapter, namely inefficient task interruption, task-switching and task resumption. This chapter

presents variations to Soar's current default rules which make it possible to switch far more efficiently between tasks and to resume tasks more efficiently after they have been interrupted. The chapter is probably too (Soar) technical for most readers, though the introduction and conclusion will give an insight into some fundamental psychological and Soar-related problems with working memory size and depth of sub-goaling.

In order to build and test the model, detailed empirical data are required. In our case we need data that describe visual orientation, speed control, course control and motor control when approaching an intersection. We were very fortunate to have access to the raw data of a detailed experiment by De Velde Harsenhorst and Lourens (1987), which investigated speed control, visual orientation and motor control when approaching and handling intersections. Chapter 4 presents a summary of these data.

1.5.2 Part II

Chapter 5 is the introduction to the second part of this study, which presents DRIVER – a far more elaborate cognitive model of car driver behaviour. DRIVER was to a large extent built in response to inadequacies of and criticisms about the first model. The model described in Chapter 2 (a) had a very underdeveloped theory of perception, (b) had no real facilities for motor control, (c) used Soar's standard default rules that prevented efficient task-switching and (d) was clearly unable to perform real-time tasks. DRIVER addresses all these inadequacies by providing (1) low-level perception mechanisms and higher-level visual orientation strategies that include attention, eye movements and head movements; (2) low-level motor control mechanisms, including the use of these mechanisms in a complex task like shifting gears; (3) efficient task-switching and (4) real-time behaviour that fits empirical data.

In Chapter 5 we provide an overview of DRIVER. In addition we describe the methodology followed in this chapter. Chapter 6 discusses the dynamic simulated traffic world that DRIVER occupies. Chapter 7 describes how lower-level motor control is simulated in DRIVER. Lower-level motor control is primarily required (a) to effectuate motor commands in the external world and (b) to simulate the time it takes to move an extremity. Chapter 8 describes how DRIVER plans and executes complicated motor plans for gear-changing and speed control. Motor plans as used in Chapter 8 consist of lists of motor commands that can be carried out by the lower-level motor control mechanisms described in Chapter 7.

Chapter 9 describes how lower-level perception mechanisms are simulated. It describes how the functional and peripheral fields are implemented, how attention selects objects from the functional field and how eye movements and head movements are simulated (partly using the lower-level motor control

described in Chapter 7). Chapter 10 describes how lower-level perception mechanisms are used in visual orientation strategies in actual driving. Chapters 11 and 12 describe speed control and lane keeping. Chapter 13 describes navigation, which is a classical search task. We will attempt to show in this chapter how one variation of default rules developed in Chapter 3 can ensure smooth task interruption and task resumption. Chapter 14 integrates all the previous topics into a discussion about multitasking in driver. In this chapter we can also see to what extent the model models actual driver behaviour, both quantitatively and qualitatively. Chapter 15 returns to the original objectives of the study and discusses the extent to which they have been achieved.

1.6 Notes

¹ An additional reason is of course that car driving is an important issue for society. In the first place because the car continues to be a vital means of transport and therefore a significant economic factor. In the second place is driving a car simply a very dangerous activity. Many people die in car accidents, or are injured, or are disabled for the rest of their lives. Imperfections in car design or the road infrastructure are often the cause of accidents, but more often the cause can be attributed to imperfections in human driver performance. It is therefore not surprising that governments, the car industry and many scientists consider it important to study human driving performance.

² The small part which is sharply perceived is sometimes referred to as the functional visual field (Sanders, 1963). In the experimental psychology literature the functional field, defined as the field within which attention can be given to objects and features without eye movements, is 2 to 3 degrees. Experiments in traffic psychology (Miura, 1986) indicate that drivers can effectively use information from a larger area up to 20 degrees. However, even with a larger area, there is still the problem that it takes a considerable amount of time to extract the right information from the functional field.

³ A further difficulty is that the novice driver has to learn which objects are important.

⁴ Contrast this with expert drivers, who are so good at multitasking that they can even keep up a conversation in most situations. However, even they will stop talking if the situation becomes really complicated.

⁵ The distinction between these three levels is explained in more detail in Chapter 2. Note that this theory is currently only *paper based*. It is not implemented as a computational model.

⁶ Newell (1990) formulates a further argument, which is in fact generally accepted, based on the idea that theories exist to answer questions. A problem with most non-computational theories is that a trained scientist is often needed to interpret the questions in the context of the theory. An extreme example is Freud's theory. Theories expressed as mathematical formulas, information flow models or computational models do not require interpretation. A given input automatically gives rise to an unambiguous answer.

⁷ Some examples of production systems that are used in cognitive science are CAPS for reading and language comprehension (Just & Carpenter, 1987), HAM for modelling semantic memory (Anderson & Bower, 1973) and ACT* for modelling skill acquisition (Anderson, 1983).

⁸ A trivial example to explain the control loop in a production system:

Suppose that working memory contains only the number "1". The production system has two normal rules and one meta rule.

Rules		
rule 1	if	number X
	then	add X + 1 and delete number X
rule 2	if	number X = 10
	then	stop
Meta rules		
metarule1	choose most specific rule.	

In step one, only rule 1 will be true and therefore executed automatically. "1" is deleted and "2" is added to working memory. In step 9 both Rule 1 and Rule 2 will be 'true'. The metarule ensures that rule 2 is chosen from the conflict set and the system will stop.

⁹ The notion of *universal sub-goaling* is relatively new, although the notions of sub-goaling, goal hierarchies and recursive problem solving have existed since the beginning of cognitive science (Newell, Simon & Shaw, 1962; Newell & Simon, 1972).

¹⁰ Heuristic knowledge is the hallmark of the expert. In the ideal expert - as expressed by a computational or competence theory of a particular task - this knowledge is essentially complete, though not necessarily deterministic. Consequently, the ideal expert never encounters a choice within the confines of the prevailing task domain, but can always instantly choose the correct operator and solve any impasses encountered in the most efficient way possible

¹¹ In this philosophy, the qualifying test between candidate unified theories of cognition is the number of different tasks that a theory (or its implementation in the form of an architecture) can handle. The reasoning is that every new task brings with it a new set of constraints. Adding a new task to the architecture in such a way that it integrates, but does not interfere with the existing tasks, means that the architecture is also able to incorporate this new set of constraints.

¹² However, in general the original makers have more say than the rest. Most Soar researchers tend to go along with the changes, somewhat reluctantly adapting existing models to the latest version of Soar.

¹³ A total of 150 users world-wide is not very many. The main reason Soar does not have more users is its complexity. The system is difficult to explain to new users and is user-unfriendly.

¹⁴ The reasons why well-established AI languages like Lisp or Prolog were not used may require some explanation. First, The objective of modelling was to describe car driving from a cognitive psychological and problem-solving perspective. A tool like Soar has, relatively speaking, more cognitive psychological and problem-solving constraints than a general-purpose language like Lisp or Prolog. It would be pointless for us to spend years designing a plausible cognitive system for problem-solving.

¹⁵ Note, however, that Anderson (1993) offers a public domain version of his production system with a number of demo programs.

2 Multitasking in driving

Summary: This chapter, published earlier as Aasman and Michon (1992), describes our first attempt to model driver behaviour.¹ The paper specifically addressed the phenomenon of multitasking in Soar. Driving an automobile is a highly complex task requiring concurrent execution of a number of subtasks. The first problem that we encountered in the design of a cognitive model of driver behaviour in Soar was how to bring Soar to multitask. Soar is in principle a problem solving system dedicated to solving only one goal at a time. For several reasons the attempt was not entirely successful but the insights in the problems that multitasking generates for a symbolic problem solver like Soar justifies the inclusion of this attempt in this study. It introduces the issues that we will address in later chapters.

2.1 Introduction

Driving is a well practised and comparatively well studied example of complex dynamic performance in which an operator is required to cope with several tasks at once. The set of driver subtasks appears to be structured as a functional hierarchy in which conventionally three performance levels are distinguished (Michon, 1976, 1985): long term, strategic tasks, involving route planning and navigation; tactical or manoeuvring tasks which comprise the various real time interactions with objects in the traffic environment including other road users; and finally, short term operational tasks which include the basic handling and control tasks that are required to operate the vehicle in a proper fashion. A second way of looking at the driving task takes the driver's cognitive functioning as its vantage point. Rasmussen (1985, 1987) and Reason (1987) in particular, have conveniently introduced the distinction between knowledge-based, rule-based, and skill-based performance to describe operator activities in complex tasks. Since these two perspectives are more or less orthogonal, one may classify various driver subtasks as in Table 1 (see Hale, Stoop, & Hommels, 1990, p. 1383).

Table 1. Matrix of Tasks and Types of Knowledge

	Planning	Manoeuvring	Control
Knowledge	navigation in unfamiliar town	controlling skid on icy roads	learner on first lesson
Rule	choice between familiar routes	passing other cars	driving an unfamiliar car
Skill	commuter travel	negotiating familiar junctions	road following around corners

Task elements from different levels frequently need to be performed simultaneously. Drivers must be able to handle both speed and heading, while, at the same time they are trying to survey the trajectories of other cars and the intentions of their operators, and to plan their own route towards their destination. An experienced driver is capable of integrating these tasks seemingly without effort, whereas learner drivers may show high levels of stress in their effort to coordinate the various driver subtasks. Descriptions of the driver's capability of simultaneously performing multiple tasks often refer to the distinction between automatic and controlled, or to the distinction between conceptually driven (or top-down) processing and data-driven processing (Norman & Bobrow, 1975). The distinction between *automatic* and *controlled* behaviour was first systematically described by Shiffrin and Schneider (1977). Automatic behaviour is characterised by its parallel nature. Such behaviour cannot be inhibited, it is fast, independent of workload, and people are unaware of any processing going on. For an expert the basic control tasks of operating the vehicle seem to fall under this type of behaviour (see, for instance, the lower right cell in Table 1). In contrast, controlled behaviour is characterised by its serial nature. People can inhibit this type of behaviour, it is slow, workload dependent and there is awareness of the processing. Navigational tasks and the tasks that comprise the real time interactions with objects in the traffic environment seem to fall for a considerable part under this regime (covering, in fact, the upper left cells in Table 1). Effortless simultaneous execution of multiple 'controlled behaviour' tasks in driving requires a long learning period to allow some driving subtasks to become really automatic (vehicle control) and to achieve coordination between other driving subtasks that cannot be made automatic. De Velde Harsenhorst and Lourens (1987, 1988) have given a detailed account of this learning process in their in-depth error analysis of a learner driver.

The second distinction is that between *top-down* controlled and *data-driven* behaviour. In driving, there is ample evidence for top-down control—sometimes also referred to as conceptually driven or goal-oriented. Drivers do apply fixed plans and use simple, overlearned strategies in their driving. Aasman and Lourens (1991) describe such relatively fixed strategies (in terms of speed control, car control, and visual orientation) as a function of the type of intersection and the intended manoeuvre at the intersection (See also

chapter 4, this volume). The data-driven nature of at least part of the driving task is evident too. Thus, for instance, an ongoing manoeuvre may be interrupted by critical events in the outside world (e.g., children dashing out in front of the car). In the second place, driver goals appear to be determined by local circumstances, also causing driving to be partly data-driven.

2.1.1 Using Soar in modelling multitasking driver behaviour

The aim of this chapter is to review a number of aspects of a cognitive model of multitasking in driving. The model to be discussed in this chapter takes into account the distinctions between automatic and controlled, as well as between top-down and data-driven task performance. Soar (Laird, Newell, & Rosenbloom, 1987) has been chosen as the medium for implementing the model as a computational system. One important reason for this choice is that the Soar architecture directly supports both the distinction between automatic and controlled processing, and that between goal-directed and data-driven behaviour (Newell, 1990). This choice determines our definition of task and multiple task performance to a significant extent.

Pursuing multiple goals is obviously of the essence of any multitasking theory. Less clear, on closer inspection, appears to be the nature of the goals that an autonomous agent in a dynamic environment might pursue. Covrigaru (1990) has identified several types of goals that characterise systems that can survive in a dynamic world and that, therefore, qualify as autonomous. Five distinguishing characteristics of goals appear to be important from our point of view. The first is that between *achievable* and *homeostatic* goals. Achievable goals have a well defined set of initial and final states in a state space, and activity will stop once the goal has indeed been achieved. Such goals are common in AI systems. Planning a route before one actually goes behind the wheel is a task involving this type of goal. Homeostatic goals, in contrast, are pursued and 'achieved' continuously. Activity does not terminate when the system is in one of its final states because the world will keep changing and persistent control activity is required to remain in the desired state or to reach it again. Lane keeping and speed control are examples of tasks aiming for homeostatic goals.

The second distinction is that between *exogenous* and *endogenous* goals. Exogenous goals are set by the external environment. As soon as a driver perceives he is approaching an intersection he will set up a goal to deal with that intersection; that at least, is what happens in our driver model. Endogenous goals, in contrast, are generated within the system. Searching for the presence of a parking sign may be considered an endogenous goal. A third distinction is that between top-level goals and subgoals. The existence of top goals and subgoals is obvious in a goal-directed architecture like Soar. The main activity in Soar's subgoals is directed at solving impasses occurring at the top level. In

general this amounts either to directly selecting a particular operator from the available set at the top level, or solving the problem and learning how to apply operators at the top level.

The fourth distinction is that between *long term* and *short term* goals. The importance to our present concern is that the lifetime of goals should be an obvious determinant of the performance of an autonomous system. The final distinction is directly relevant to our multitasking theory. It distinguishes between *multiple top-level* and *single top-level* goals. As stated at the outset, the need for multiple top goals is obvious in a task such as driving. The driver model discussed in this chapter is capable of controlling course, speed, and navigation at the same time²

The goals pursued by the driver model presented in this chapter can be characterised in terms of these five polarities. Thus, for instance, some of the goals in the driving task may be considered homeostatic whilst others are achievable. Most of the goals pursued in driving are endogenous, but the environment can set goals too.

Multiple top-level goals have become a research issue in the Soar community quite recently (Covrigaru, 1990; Hucka, 1991). The present chapter deals with the specific option of handling multiple goals at the top level. In its bare essence, multitasking or the handling of multiple top goals in this chapter, amounts to the following, quite simple and straightforward position. In Soar only one task can be active at any one time, that is, a single operator implementing a task or a step in a task. Given this constraint, multitasking can be defined as some form of *switching between subtasks*.

Our motivation to focus on multiple goals in the top-goals has been explained in section 1.4 of the previous chapter: in this chapter and in this study we try to adhere to Newell's "minimal scheme for immediate responses". Thus attention—in Soar the *attend* operator—and the setting of new tasks—the *tasking* operator—reside in the *base level problem space*, the highest goal context in the goal stack. The attend operator functions as an interrupt, halting all ongoing activity and focusing the system on other tasks (Newell, 1990, p. 262). The essential consequence of the claim that interrupt operators are applied in the base level space is that the interrupt operator has the power to break the goal hierarchy and to destroy all ongoing activity. The driver model, to be discussed in detail in the next section, is capable of handling a set of active top goals or tasks within an environment that represents a simple four-way intersection. The tasks in the model are speed control, course control, and navigation. A goal for a particular task is achieved by applying a sequence of internal and external operators. An internal operator makes a change to an internal

representation of the world, as does for instance, the look-ahead operator in navigation. In contrast an external operator, for instance a change of speed, will attempt a change in the real world. Only one operator may be selected and applied at any one time. Multitasking is achieved by a uniform mechanism that allows for task switching and task resumption. Task switching in this case refers to the switching of control between different subtasks and basically amounts to interleaving operators from different tasks.

2.2 The Model

In this section we will describe the multitasking driver model, the perception of the driver, and the simulated environment in which we test the model.

Starting with the latter, the model is tested in a small, simulated traffic environment. This allows us to provide our model with real time input that simulates some aspects of the dynamic traffic environment. The implementation of a small world and its usage as a testbed for intelligent architectures has been described by several authors (e.g., Aasman, 1988; Reece & Shafer, 1988; Wierda & Aasman, 1988). Basically a small world simulation comprises a number of intelligent agents (drivers, cyclists, and other road users) moving around in a network of streets and intersections. The term intelligent in this case refers to the fact that each agent has the ability to perceive the environment, including the other agents, and a set of decision rules to interpret these perceptions in order to produce an optimal speed and course decision. Two aspects characterise the present implementation. First, it is intelligent in a distinctly non-psychological sense: perception is perfect, in this case with a 360 degrees visual field, control of speed is immediate rather than by way of motor commands, and the agents have no limits on their working memory. Second, the implementation is interactive, agents basing their behaviour, in real time, on their own goals, as well as on the goals and behaviour of other agents. This property distinguishes the present implementation from the traffic flow models conventionally used in traffic engineering. The resulting interactive behaviour of agents in this small world turns out to be amazingly 'natural,' that is, close to expectation if we assume the agents to behave rationally. Altogether, this need not come as a surprise if one considers that the important speed-control and risk parameter estimates that have been used to calibrate the model were derived from empirical observations. Note also that the driver model to be tested in this simulated environment is the only entity in the small world endowed thus far with a Soar intelligence; all other agents use the perfect non-psychological perception and decision mechanisms mentioned above.

2.2.1 Perception and internal representation of the world

The simulated driver, as one of the agents in the simulated traffic world, is able to perceive both the road environment and the other traffic agents.

Perception in humans is presumably a massively parallel, continuous, and asynchronous process, and this is in a very primitive way reflected by the perception unit in the model. In terms of Soar a continuous stream of inputs enters the system and is added to the top state, the highest state in the goal stack. These inputs will be referred to as perceptual objects. The perception unit is currently realised as a Lisp program, simulating a transducer process that can translate properties of objects and relations between objects in the small world into Soar data structures. These Soar structures constitute the internal representation of the world (see Figure 2-1).

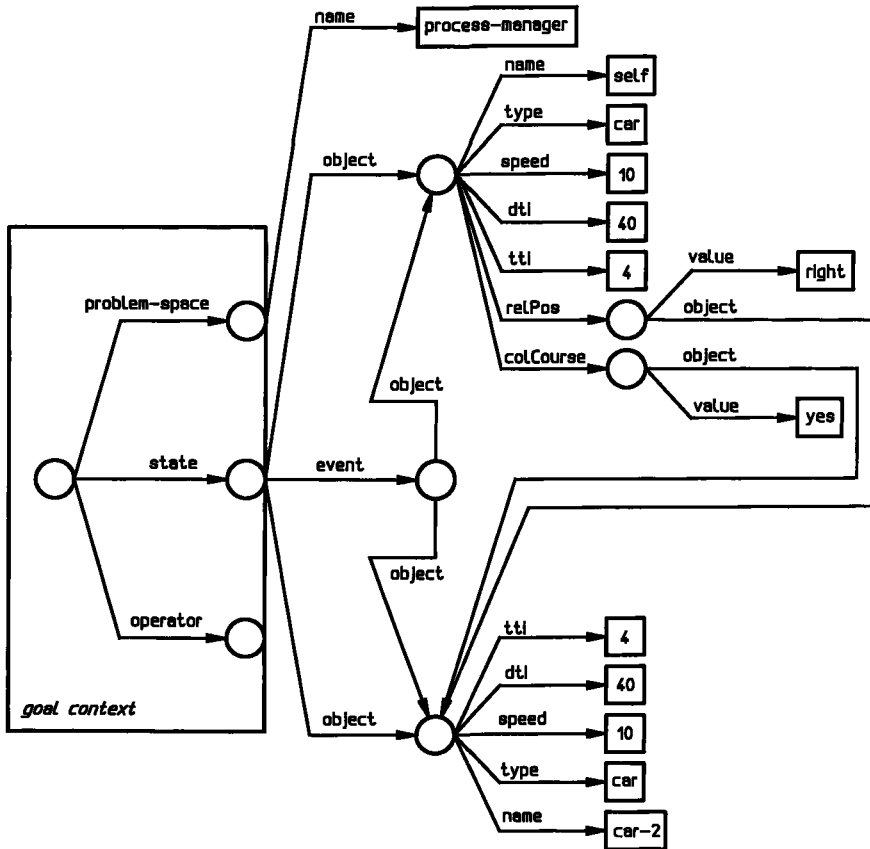


Figure 2-1. A graphical representation of the contents of Soar's working memory, showing two cars approaching an intersection. The upper object represents the modelled driver, here referred to as the 'self.' The lower object is the representation of the other car. Note how relations between objects require special relation objects: car-2 is coming from the right and is on a collision course from the point of view of the 'self.' Event objects represent time; dti is distance to intersection, and ttl is time to intersection, that is, dti/speed. All objects seen in the most recent decision cycle are attached to the event generated in this decision cycle

Perceptual objects are elaborated to the degree that Soar knows their type—*automobile, bicycle, pedestrian, or intersection*. Every object is assigned an

identity (a unique name in Soar's internal model). A moving object is represented as a sequence of objects with the same identity. The principal properties of every moving object are its position, direction, speed, and the required time to reach the intersection. The main spatial and temporal relations between objects are the relative position of objects (where an object can be of any type) and collision course information. Time is represented as a list of so called event objects. During the decision cycle such an event object is generated and linked to both the state and the previous event object. All perceptual objects that enter in the same decision phase are linked to appropriate perceptual object. This mechanism enables the model to distinguish between newly observed objects and less recently perceived objects.³

It may be argued that there is too much intelligence in the perceptual unit. In the first place, relating the various dynamic objects may require the help of higher processes in cognition. At least some of these relations will be inferred at a high level, the detection of a collision course being a likely candidate. In the second place, we totally ignore low-level manifestations of selective attention. Thirdly, we are making no difference between foveal and peripheral vision. Still other, equally critical problems include the representation of quantities, such as the distance to intersection, speed, and the spatio-temporal relations between objects. In the current version of the model real numbers are used to represent these variables.

2.2.2 Multitasking: Managing multiple tasks in the process manager space

The essence of the multitasking theory set forth in this chapter is that the management of multiple, concurrent goals is represented as a problem that, like any other problem can be solved by Soar in an appropriate problem space. The problem space in this case will be referred to as the *process manager space* (PMS). The purpose of the PMS is to select the right subgoal to be performed at the right moment since only one subtask can be active at any one moment. The subtasks—also called goals in Soar, or processes in other AI contexts—that this space has to manage and that we use as an example in our model are shown in Table 2. The table describes the four subtasks to be managed by the PMS in terms of task, goal type, and action proposed.

2.2.3 Process objects represent goals; Process operators implement tasks

Goals or subtasks are represented as *process objects* on the top state. A goal is conventionally represented as a name and a desired on the goal object. However, in our multitasking theory it is the task of the PMS to manipulate goals as normal objects. This justifies our choice of representing goals on the top state—quite apart from the fact that in Soar it would be impossible to manipulate multiple goals through a single desired on the goal object. A process object has three basic properties: the name of the process that it stands for, the priority of this process, and the type. The use of these properties will be

elucidated in the following sections. Figure 2-2 shows the representation of the control-speed, control-course, and find-destination processes. These examples reveal that, apart from the basic properties, objects may carry additional information such as, for instance, a range in the case of control-speed, or a current value in the case of the find-destination process.

Table 2. Goals Subsumed Under the Process Manager Space

Control speed

task: keep the speed within a given range
type: homeostatic, long term, high priority, operational, interrupt-driven.
action: change-speed (external operator)

Control course

task: keep the car within lanes (with a certain tolerance)
type: homeostatic, long term, high priority, operational, interrupt-driven.
action: change-course (external operator)

Negotiate intersection

task: deal with intersection and traffic on intersection
type: achievable/homeostatic, short term, high priority, tactical, both interrupt-driving and top-down.
action: choose correct acceleration and set speed parameters
(to be dealt with by control speed)

Find destination

task: find a route from origin to destination
type: achievable, middle long term, lower priority, strategic runs in the slack time of other goals.
action: choices at nodes in network.

Processes are implemented in lower level problem spaces. The name of such a space is by default the name of the process object. The model arrives in such a space through a so-called process operator. This type of operator has two basic properties, the name of and a pointer to the process it intends to implement. The application of a process operator will lead to a no-change impasse and a default multitasking rule will select a problem space with the name of the process. The resolution of impasses in a lower-level space can, in principle, lead to application chunks that avoid going into lower levels in the future. We will discuss these chunks later.

2.2.4 Interrupt objects signal possible change of process

Switching between processes is initiated by interrupts. In this case an interrupt object is generated for an process object on the top state. It is, however, possible that an interrupt is generated for a process that is not yet available at the top state. Interrupt objects have three basic properties: a name for the process for which they are generated, priority information, and the current event object. Remember that these event objects were generated at each wave

of perceptual objects entering cognition. Figure 2-3 shows a few examples of interrupt objects.

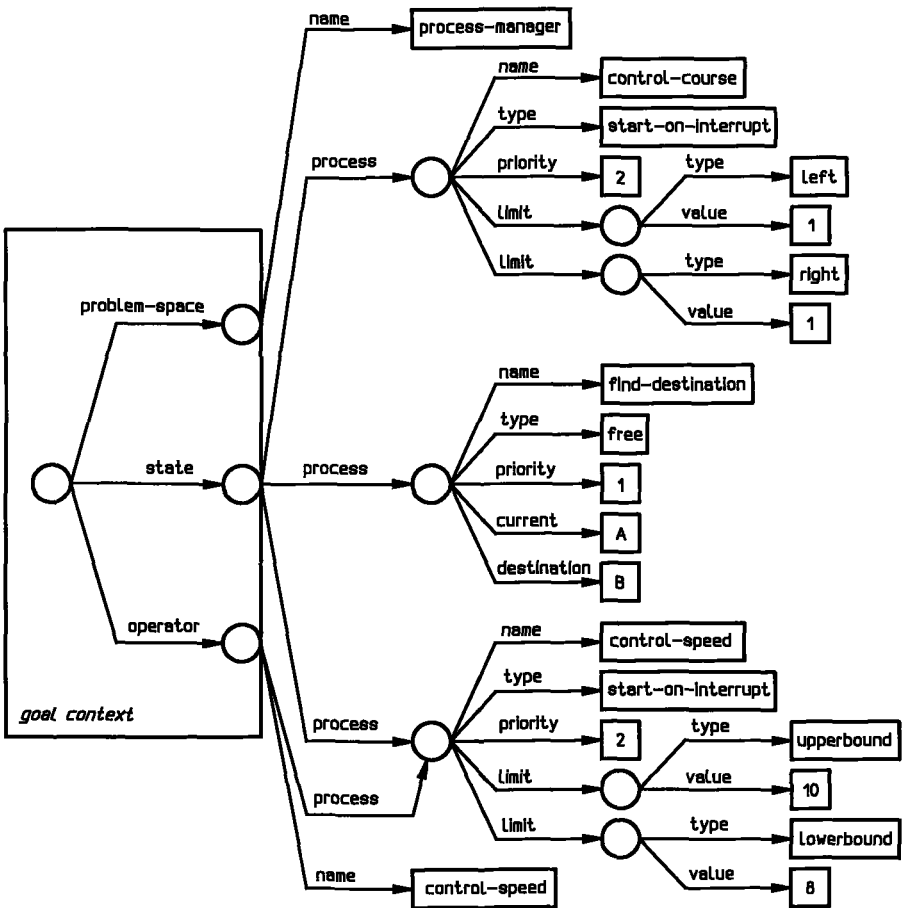


Figure 2-2. The process objects that exist in the top state. Control-speed is currently selected and implemented in a subgoal (not shown) by a process operator that has the name control speed and a pointer to the control-speed object.

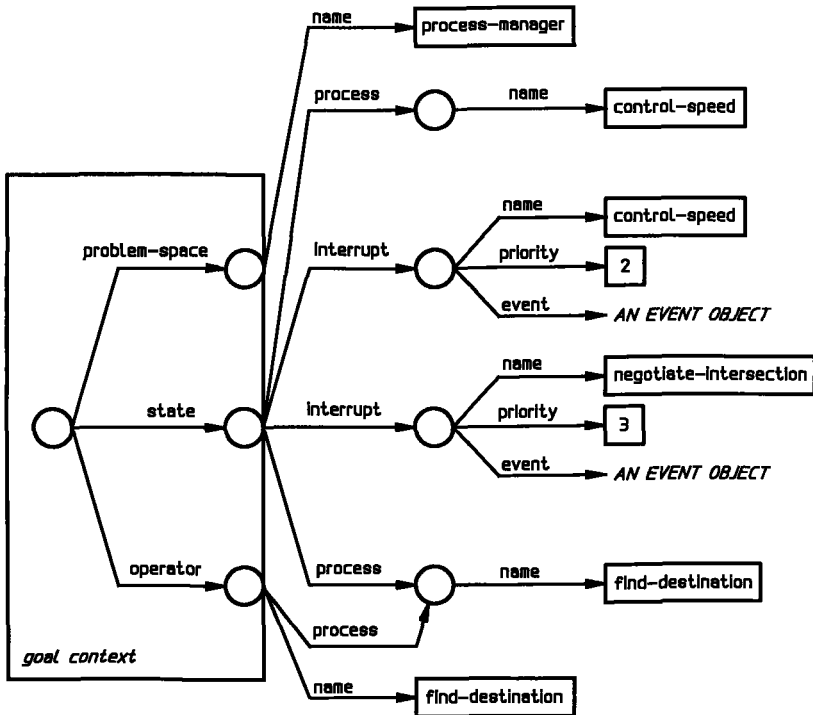


Figure 2-3. The process find-destination is currently selected and implemented by a process operator. Two interrupts are generated for the processes control-speed and negotiate intersection. Control-speed is generated for an already available process whilst the process negotiate-intersection does not exist yet.

The find-destination interrupt has as its type 'free.' This means that default multitasking rules will propose it if—and only if—the system is not actively involved in controlling speed, course, or negotiating an intersection. Finding the destination is thus a low priority process, compared to the more urgent driving tasks. The interrupt negotiate-intersection is added to the top state whenever a driving related rule notices that there is an intersection less than 100 meters away. Note that this is an interrupt for an process that is not yet available on the top state: the process object for negotiating intersections will only become active (i.e., become available on the top state) when the process is actually relevant. Interrupts for the negotiate-intersection process may also be generated when this process is already active. This happens for example when rules related to driving notice that a car is coming from the right and is on collision course. The control-speed interrupt is generated by a rule that constantly monitors both the bounds on the control-speed object and the current speed. The rules that generate interrupts for processes which start-on interrupt can be functionally described as 'monitors.' Monitor rules respond

to perceptual information and process information on the top state independently of other goal-directed activity. The following structure shows a monitor production in Soar format (with comments).

```
(sp drive* check&mark-on-upper-speed-limit
  (goal <g> ^problem-space <p> ^state <s> .^supergoal)
  If goal is top goal and
  (state <s> ^process <pr> ^object <o>)
  (process <pr> ^name control-speed ^limit <limit>)
  there is a process with name control-speed
  (limit <limit> ^upperbound ^value <limitvalue>)
  and an upperbound limit on speed
  (object <o> ^name self ^next ^speed { > <limitvalue> })
  and self has higher value,
  -->
  (state <s> ^interrupt <intr>)
  (interrupt <intr> ^priority 2 ^name control-speed))
then add interrupt for control-speed to state.
```

2.2.5 Change-process operators shift processes

For each interrupt that is added to the top state a change-process operator is generated. Figure 2-4 shows the basic properties of such an operator. The interrupt on the operator is essentially the one for which the operator was generated, whilst the allow feature carries the name of the process that is to be installed. This name is the same as that on the interrupt that generated the operator (see preceding section). The identifier, finally, is the event object that we already mentioned in the section about perception. This allows the process manager to know the most recent interrupts in the system.

Choosing Between Change-Process Operators

Whenever an interrupt occurs and the corresponding change-process operator is generated, this new operator will compete with the current process operator for the operator slot in the PMS. It is even possible that there are multiple interrupts in the PMS so that there are multiple change-process operators competing for the operator slot. If the competition is not resolved within the current decision cycle, a tie impasse will arise and the current subtask is interrupted. In the following cases this tie impasse will not arise: (1) a change-process operator for a process that is already active will be rejected; (2) as in most modern operating systems, interrupts in our multitasking model may be disabled in some time-critical task. For instance, in the current version of the model interrupts are disabled when there is already a tie impasse for conflicting change-process operators. When interrupts are disabled, no new change-process operators for an interrupt will be generated; (3) there are chunks that deal with a particular type of conflict and that 'know' how to choose between operators; (4) a change-process operator has a lower priority than the current

task or a competing change-process operator, and can thus be rejected. In all other cases a tie will arise that is handled in the interrupt handler space, to be discussed in the next paragraph.

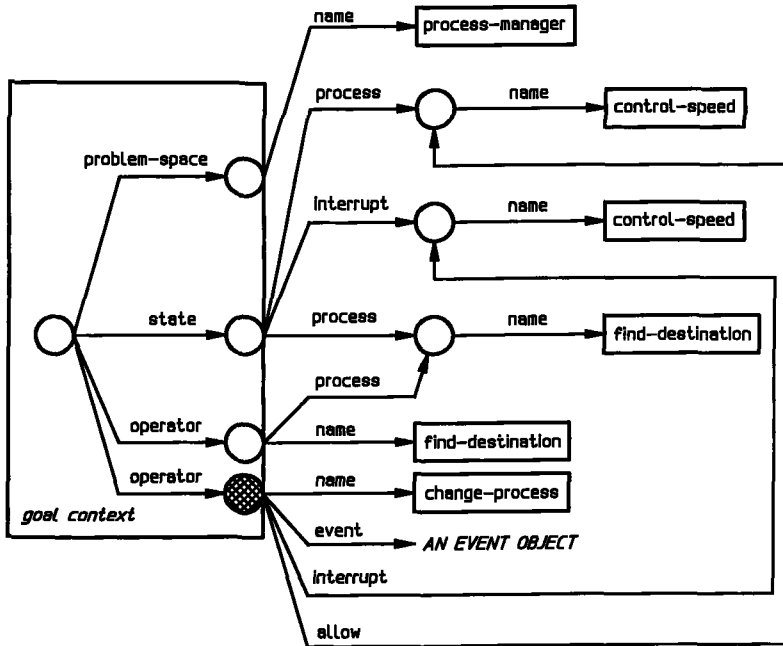


Figure 2-4. The process find-destination is currently selected and implemented by a find destination process operator. An interrupt was generated for control-speed. A change process operator is proposed for this interrupt.

Choosing Between Change-Process Operators in the Interrupt Handler Space

The interrupt handler space replaces the so-called selection space, Soar's normal space for dealing with ties. The main reason for this replacement is that the original selection space is intended for choosing between tied operators by employing look-ahead search and not for dealing with evaluations of external operators in time-critical situations. We must be able to account, however, for the fact that there is simply no time in critical situations to evaluate all the change-process operators in the selection space by applying all the operators to copies of the top state, quite apart from the fact that some change-process operators can only be evaluated after they have been applied in the real world. Soar's normal default rules for deciding between two unevaluated operators requires at least 15 decision cycles. Using Newell's estimate of 100 ms per decision cycle, this would require at least 1.5 seconds. Approximate as these calculations may be, we see that in situations generating many interrupts the system would quickly grind to a halt if it were to use the normal Soar default rules for deciding between competing change-process

operators (see Aasman & Akyürek, 1991, for an alternative to Soar's default rules). In the present version of the model not much knowledge is available to the interrupt handler space. The only rule that is implemented states that during driving any driving-related process is more important than any process that is not related to the driving task.

Applying the Change-Process Operator

If the change-process operator is preferred over the current process operator the change-process operator is eventually applied and this will result in a new top state in the PMS with the new process allowed. For this new allowed process a process operator is generated that has both the name of the new allowed process plus a pointer to that process (see Figure 2-2). If no application chunk is available the operator will lead to a no-change impasse and start the process in a subspace in which the processes that implement process objects and process operators apply (see the discussion in the preceding section). The multitasking mechanisms discussed so far do not alter the ways tasks are normally coped with in Soar. One way of looking at what is achieved by these mechanisms is that they act like a shell in which Soar proceeds in an entirely normal fashion with its problem solving activity. The exclusive function of this shell is to determine the next process to be initiated.

Summary of Multitasking Mechanism

The following five points summarise the multitasking mechanisms discussed so far: (1) Process objects (declarative structures) on the top state represent multiple goals. (2) Only one process is active at any one time. A process operator initialises a process in a subgoal via a no-change impasse. (3) Events in the external world as well as internal events may lead to interrupts on the top state. (4) Interrupts lead to change-process operators: (a) In some cases knowledge is available to reject the new change-process operator and the current process can proceed; (b) In some cases knowledge is available to directly choose the new change-process operator and a new task can be installed; (c) If no knowledge is available to reject the new change-process operators then an operator tie arises between current process operator and new change-process operators. This tie is resolved in the interrupt handler space. (5) Application of change-process leads to a new 'allowed' process on the state and a process operator to implement the task.

2.3 Driving Tasks

In this section we will discuss in greater detail the specific driving tasks that are managed by the model's top level process manager.

2.3.1 The control-speed and control-course spaces

Some of the functionality of the control-speed space has already been discussed. In the preceding section it was explained how interrupts for this space

are created by a monitor rule whenever the speed of the car exceeds one of the limits defined in the control-speed process object on the top state. This makes speed-control a purely reactive process. In a more advanced model, however, the driver should be able to actively search for speed information, for example, by looking at the speedometer. Presently speed is controlled by simply decelerating when the speed is too high and by accelerating when speed is too low. The acceleration and deceleration commands are given directly to the Lisp motor function from within the control-speed space. In a later version of the model the change in speed will be effected by means of motor commands that translate into vehicle control actions which, in turn, will be transformed, into speed commands.

The control-course space resembles the control-speed space in many respects: it too features a passive strategy, for controlling course, and course commands are directly given in Lisp. That is, whenever the car swerves out of bounds the system will react by installing the control-course space and correct the course.

2.3.2 The negotiate-intersection space

The negotiate-intersection space is proposed when an interrupt signals an approaching intersection. Whereas control-speed and control-course are continually active homeostatic goals, the process negotiate-intersection is set by the environment and remains active for the short period during which the intersection that triggered the process is a relevant feature of the environment. Two basic actions can be proposed in this space. The first is simply to reduce speed whenever the approach to an intersection is noticed. In the present model this is done without regard for the type of intersection, the intended manoeuvre at the intersection, or the visibility at the intersection. In a later version these important factors will play a role. Speed is reduced by changing the limits on the process object for control-speed. The second action in the negotiate intersection space is to reduce speed whenever a car from the right is found to be on a collision course. Whenever a monitor rule notices that a car is approaching from the right on a collision course, an interrupt for the negotiate intersection space will be generated. A high priority is assigned to this space and accordingly will be selected with very high probability. Within the space, speed limits on the control-speed process object will be reduced to avoid a collision. Once these limits are set, the negotiate-intersection space can be left.

2.3.3 The find-destination space

The third task under concern is the navigation or find-destination task. Navigation is a subtask that is handled by drivers at a different performance level than are speed and course control. Coping with a navigation task while driving involves a heavy use of the goal stack and a good deal of multi tasking, that is, task interruption and task resumption. The navigation task, performed

by our model, referred to as find-destination task, consists of finding a path through a directed graph, representing a road network map. A detailed description of this task can be found in Van Berkum (1988). The nodes in this graph are states in the find-destination task. Move operators in the task are the direction choices at each node. Available search control knowledge classifies operators as 'good,' 'medium,' or 'bad' by comparing the directions of the destination and the current move operator. Problem solving here is a simple hill climbing search where states in the goal stack indicate the path followed so far. The main chunks learned in the problem solving process are evaluation chunks that avoid dead branches in future search.

The interesting aspect of this task is not in the first place that the model will eventually reach its destination or learn the network; in this respect it is actually little else than a simplified version of the real task. The real interest lies in the process of interrupting and resuming such a task. Whenever the search for the destination is interrupted by one of the other tasks, for example, control-speed, the whole goal stack within the find-destination space will be lost in principle. This derives from our decision to have the interrupts and change-process operators at the top level. The consequence is that when later there is slack time again to do the find-destination task, the whole goal stack must be rebuilt before the model can proceed from the node where search was initially interrupted.

This leads to the basic question in task interruption and resumption: What is it that needs to be remembered about the interrupted task in order to be able to resume it efficiently at a later instant? One answer is to store the entire search tree on the process object generated up to the time of the interruption. In this case a residue of all the intermediate states is stored onto the process operator so that in the resumption of the task this information may be used to rebuild the goal stack as fast and efficiently as possible. For at least two reasons this procedure seems unattractive. First, it is psychologically implausible that subjects will store extremely large data structures in working memory about one (inactive task) while executing another task, even if the two tasks do not interfere. A second, quite mundane reason is that the administration of states and operators applied is almost impossible in the Soar version we initially used. The current version (i.e., Soar5) can indeed destructively modify data structures, making this administrative problem somewhat more tractable.

In our model we have used a much less extreme and more plausible way of storing state information to allow task resumption. This procedure retains the basic network map on the process object, thus creating an analogy with the concept of mental map (Pailhous, 1970). Searching this map for the destination, evaluations of operators are stored in chunks, whilst the nodes that are 'visited' are indeed marked as such, both on the network map and in the

chunks that are being built. If search is interrupted and the goal stack is lost, only the basic network remains on the process object. After resumption both the visit-marks, visit-chunks, and the operator-evaluation chunks will guide the search very quickly and efficiently to the point where the search was interrupted. The visit-marks and visit-chunks force Soar to explore trodden paths, while the evaluation-chunks (both rejections and better preferences) keep the search from going into wrong directions.

Visit-chunks may seem redundant when there are visit-marks on the network map. However, it is conceivable that somehow the find-destination process is lost from the top state and thereby also losing the map's visit-marks. In these cases the visit-chunks will take over by quickly restoring all the visit-marks on the network map. Although this procedure is very efficient, it is still lacking in psychological plausibility, since human subjects—unlike certain species of animals—will resume a search task from some landmark in their mental map rather than reconstruct their entire trip from their point of departure, however efficiently and effectively they may be able to do this.

Multitasking Driver Model At Work

The following is a brief "artist's impression" of the model at work in a prototypical situation that it can (learn to) cope with. Trace 1 shows a Soar trace of this scenario. Initially, the model only uses the processes of controlling speed and course since these have a higher priority than navigation. If, however, no attention to speed and course is required then the system will be able to engage in solving the navigation task, that is, in finding its preset destination. While at the navigation task, the system will notice that an intersection is less than 100 meters ahead. An interrupt operator for negotiate intersection is proposed which will replace the navigation operator. In case the system does not have the knowledge that negotiate-intersection has a higher priority than the find-destination task it will drop into the interrupt handler space where it finds the general rule that "driving-related tasks are more important than non driving-related tasks." This will generate a chunk that will give preference to the negotiate-intersection process operator and break the tie. The next time around the navigation task will automatically be replaced by the negotiate-intersection process. In the negotiate-intersection space the limits on the control-speed process are lowered. This will immediately generate an interrupt for the control-speed process. Once in the control-speed problem space speed will be reduced simply by lowering acceleration and the control-speed problem space is terminated. After giving the reduce-speed command the system will engage in navigation again. A short while later the system observes a car, a perceptual object designating a car is added to the top state. Another interrupt is generated, this time in the intersection problem space where knowledge resides about how to deal with cars at an intersection. Again speed is reduced upon the observation that the car comes from the right. Finally the system can finish its navigation task.

Trace 1. A trace segment of the model at work with learning disabled. Our comments are in italics.

0 G: G1
1 P: P3 PROCESS-MANAGER-SPACE
2 S: S4
The initial state s4 contains the process objects find-destination, control-speed and control-course.
3 O: O13 FIND-DESTINATION
A process operator for find-destination activates find-destination process.
4 -- > G: G14 (FIND-DESTINATION OPERATOR NO-CHANGE)
Soar goes into a subgoal
5 P: P16 FIND-DESTINATION
6 S: S17 X->A
As a result, a search is started to reach the destination during which an interrupt arrives
7 O: O41 CHANGE-PROCESS
for negotiate-intersection.
8 S: S42
The new state will now allow process negotiate-intersection
9 O: O43 NEGOTIATE-INTERSECTION
and an operator activates it.
10 -- > G: G44 (NEGOTIATE-INTERSECTION OPERATOR NO-CHANGE)
11 P: P46 NEGOTIATE-INTERSECTION
12 S: S47
Interrupt handler space changes limits on control-speed process
13 -- > G: G68 (UNDECIDED OPERATOR TIE)
which immediately generates interrupt for control-speed.
14 P: P69 INTERRUPT-HANDLER-SPACE
This causes a tie to occur between negotiate-intersection and control-speed,
15 S: S70
which is won by control-speed.
16 O: O67 CHANGE-PROCESS
So a change-process operator for control-speed is installed
17 S: S84
18 O: O85 CONTROL-SPEED
19 -- > G: G86 (CONTROL-SPEED OPERATOR NO-CHANGE)
20 P: P88 CONTROL-SPEED
A set of operator applications bring speed down to the new desired value after which
21 S: S89
22 O: O103
23 S: S104
24 O: O117
25 S: S118
26 O: O131
27 S: S132
28 O: O145
29 S: S146
30 O: O159
31 S: S160
32 O: O173 CHANGE-PROCESS
there is again time for find-destination ...
33 S: S174
34 O: O175 FIND-DESTINATION

2.4 Discussion and Evaluation of the Model

2.4.1 Mapping processing types onto the driver model

Our aim in this study was to present a cognitive model of multitasking in driving that can take into account the distinctions between automatic and controlled on the one hand, and between data-driven and top-down processing on the other. It was claimed in the introduction that Soar does support these various types of processing but thus far we did not really justify this statement. Having discussed our model of multitasking in driving we are now in a position to discuss the relation between types of processing in this driver model.

The mapping of automatic and controlled behaviour on Soar and our driver model is fairly straightforward. Soar is a parallel production system where productions are matched and fire in parallel. The seriality in the Soar is forced by the goal stack—the goal hierarchy—because at any one time only one operator can occupy the operator slot in a goal context. Thus executing a sequence of operators and solving impasses in subgoals translates directly into controlled processing; selecting and applying operators in sequence is a slow process, requiring a minimum of 100 ms per operator, which can be interrupted at will. The parallel activity in the elaboration phases, involving the generation of operators and their preferences and the application of operators to states is automatic because it is fast and cannot be interrupted⁴. The serial activity in the multitasking activity of the driver model is clear. At the top level we observe a continuous stream of process and change-process operators. Switching between processes is for the greater part a serial, controlled process.

Prominent automatic and parallel behaviour in the driver model is found in the perceptual unit of the model, in the productions that generate interrupts and in the automatic resolution of multiple interrupts. The productions that handle perception and the automatic updating of the world are clearly autonomous, that is, independent of the activity in the goal stack. This activity will therefore proceed irrespective of whatever else is happening on the goal stack. A second, more interesting form of automatic behaviour lies in the generation of interrupts. The model contains several types of 'monitor' productions (Kuokka, 1990) that allow the system to remain reactive independent of its current focus of activity. We have already discussed a production that will generate an interrupt whenever the speed exceeds one of the speed limits. There are similar types of monitor functions for course control and for noticing the presence of traffic objects, especially those on a collision course. These productions too are independent of activity in the goal stack: they will only react to process objects on the top state and to perceptual objects on the top state. The third form of automatic behaviour, finally, lies in the automatic resolution of change-process operator ties. In some cases the system has

knowledge to automatically select or reject new change-process operators. Mapping the distinction between top-down and data-driven behaviour onto Soar and onto the driver model is also fairly straightforward. Soar is a goal oriented system, making it, predominantly, a top-down control system; nearly all actions in the system are performed in order to serve goals. The driver model is facing the problem of integrating multiple tasks. Solving this problem is certainly a goal-directed process: given a set of tasks, that is, a set of goals in this environment, find the optimal ordering of operators that comprise these tasks. If the driver model has the knowledge that in this particular situation it is best to switch operators in a certain order we may speak of top-down controlled switching. Soar, like other production systems, can also be data or interrupt driven: activity within elaboration cycles cannot be stopped by external or internal interrupts. However, the decision procedure allows for the interruption of a string of operators and the temporary insertion of other operators. The essence of data-driven or interrupt-driven processing is that a string of operators working on a main task can at any time be interrupted by another, secondary and maybe temporary task. If the model perceives an unexpected situation and is required to set a new goal in order to deal with this new situation we may speak of data-driven or, better perhaps, interrupt driven switching. In order to enable interrupt-driven switching between tasks the model has several important features. First, it has the capability of perceiving more than is required for the current task; thus, for example, while approaching the intersection the system must be able to perceive that the car phone begins to buzz or that a child is dashing out into the street. In the second place there is a way to interrupt the current task in order to install operators that will address the critical event.

2.4.2 Evaluation and suggestions for future research

The remainder of this chapter will be devoted to a discussion of some specific features of our model and the direction in which the next steps towards a more detailed model will take us. First some issues concerning multitasking will be dealt with. Next some implications for driver task descriptions will be discussed, and finally we will make a few remarks about the Soar version used to implement the model.

Process objects. The process objects in the current model have priorities and types that are entirely built in by the designer. By and large these prevent ties between process and change-process operators. However, there are problems associated with the use of these priorities and types. In the first place, the priorities and types are absolute in the sense that they are not context dependent. Whether, for instance, navigation, that is, thinking about the route, is a free process or not should eventually depend on the current task being executed. A second problem facing the present approach to process objects is that the priorities and types in fact should be learnable. Programming them in advance, however, prevents the program from learning the priorities. A third

issue, the difference between types and priorities has come up in hind-sight but the issue is not a serious one. One can, of course, replace a type by a very low priority, making the type obsolete. The fourth issue is that priorities are a kind of indirect preference mechanism; one might therefore conceivably use Soar's inherent preference mechanism to order priorities. The reason for using the indirect mechanism is that it is possible to reason about priorities but not about preferences.

Interrupt objects. It is a meaningful question to ask why interrupts are generated as data structures on the top state rather than directly as change process operators. The principal reason for employing an indirect mechanism in this case is that it is more convenient to work with data structures on the top state than having operators dangling somewhere in memory. It would, however, have been possible to directly generate operators instead of interrupt objects.

Rules learned in multitasking. Multitasking was described earlier as the ability to integrate several tasks or to switch between them in real time. Our first concern was to realise a model of multitasking, the second was to study the role of learning within the mechanism of multitasking. In order to study learning we had to slightly modify the model discussed so far. In the modified version the command for changing the acceleration is not directly given within the control-speed problem space but as a declarative structure on the top state where default multitasking rules interpret this command. Two types of chunks are learned in this version. The first type implements a task and is, in Soar terms, an operator-application chunk. The second type is search control knowledge that affects the choice between change-process operators. An example of each type follows.

If	a process operator for control-speed is selected and the upperbound limit is 10 and the current speed of self is 11
Then	modify acceleration < -0.5 >
If	there is an acceptable preference for an operator that will invoke find-destination process and there is an acceptable preference for an operator that will invoke negotiate-intersection process
Then	create a better preference for the second operator, over the first

The main conclusion regarding these chunks is that they are not behaving very well during the learning process. They are either too specific (the first example) or overly general (the second example). Suppose that the driver model is in the interrupt handler space and is required to choose between two change-process operators; one for a control-speed interrupt and the other for a negotiate-intersection interrupt because a car is perceived to be on a collision course. If the default multitasking rules test for entire interrupts in the interrupt handler space or in the control-speed problem space, the resulting

chunks will contain overly specific knowledge as in the first example. The reason is that traffic objects are represented here using real numbers. These values will therefore appear in the chunks that are learned, making them too specific. When, for example, will a driver at 15.3 m from an intersection and advancing with a speed of 16.3 m/s encounter a car approaching from the right with a speed of 15.3 m/s?

On the other hand, we obtain overly general chunks, as in the second example above, if we test only on the names of the interrupts pertaining to the change-process operators. Such chunks will be context-independent and consequently fire far too often and, as a result, slow the model down. In a future version of the model the problems of specific and overly general chunks should be remedied by a form of reasoning about interrupts, and by addressing the problem of quantitative code in Soar (Newell, 1988), that is, by replacing numbers for speed and distance by symbols such as close, medium, and far, perhaps in a fuzzy fashion.

Multitasking as integrating problem spaces. A further general observation is that the model provides, in Soar terms, a problem space switching mechanism. One way to avoid the overhead of switching between problem spaces or tasks is by integrating them into one larger task (e.g., driving as such) and execute this task at the top level. Integration is achieved by the two types of chunks discussed above. This is essentially what should happen in the modified version of the model in the learning mode. The first type of chunk will avoid going into the control-speed space and directly implement a desired amount of acceleration. This chunk features, in a sense, a generic 'drive' operator at the top level. The second type will avoid going into tie impasses and automatically select the right task at the top level. A process manager space that has such chunks would behave as the drive task space. The issue of task integration and incremental problem space expansion is a current research topic in the Soar community. Covrigaru (1990), for instance, deals with the issue of merging interrelated tasks in Soar by proposing an explicit mechanism to intentionally create a new problem space from other problem spaces. Laird (1989) proposes to discard the problem space as the first class object in the goal stack and make it a 'replaceable' feature on the goal instead. This would make it possible to add multiple problem spaces (as features) to the top goal or even replace problem spaces at the top level and thus simplify task switching considerably.

Task resumption. As mentioned earlier, task interruption and task resumption have recently become research topics in Soar (Hucka, 1989, 1991; Laird, in press; Laird & Rosenbloom, 1990). Hucka (1989) proposes to deal with interrupts at the deepest level space in the goal stack, thus making it possible to keep the goal stack intact while processing them. On the other hand, the principle of top level interrupt operators implies that the goal stack is lost after

an interrupt. As we have seen, the consequence is that rebuilding the whole stack may take a considerable amount of time, proportional to the depth of the goal stack. However, the model presented in this chapter demonstrates that there is no need for a special task resumption mechanism to support rebuilding. Task resumption is made possible by keeping a minimal amount of task information about the interrupted task on the top state, whilst the chunks learned before the interrupt do guide the search back to the point where processing was interrupted. If the model in its present form must cope with too many interrupts, the result is that there never will be time to solve the navigation task. Aasman and Akyürek (1992) propose several variations on Soar's default rules (for look-ahead search) to decrease the time required to rebuild the goal stack by making the goal stack flatter. The most drastic of these proposals is the destructive look-ahead. This variation has become feasible because of the destructive state modification that is available in Soar5.

The driver tasks. The model in its present form succeeds in handling tasks at the strategic, tactical, and operational levels of driving performance. However, all of the driver tasks in the current version are rather knowledge-lean. Control-speed, control-course, and negotiate-intersection have only the barest minimum of the knowledge required to get safely across the intersection. Two issues came up in the discussion of the driver problem spaces. First, it was noted that these spaces are too reactive; there is, for example, no 'active search' for speed or course deviations. We rely on the 'monitor' rules to signal that speed is too low or high. This is what makes the system almost entirely data driven. In future versions we shall employ more active, and primarily visual strategies (including eye and head movement) in order to get information from the environment. The second issue was that commands to correct speed and course were given directly in the subspaces and directly to a transducer routine implemented in Lisp. In the forthcoming version of the model we will account for the manual control of the vehicle, following Newell's base level theory by giving the motor commands at the top level.

Visual strategies and motor behaviour. The current model lacks both perceptual and action strategies. A future perceptual unit will have to have a foveal as well as peripheral vision, and perceptual objects will be more elaborate, or less, depending on the part of the visual field in which they originate. Decision making about where to look next, involving eye movements and head movements as well as operators within the functional field (Wiesmeyer & Laird, 1990), will constitute an important extension to the enhanced model. A second addition is the development of motor control. A forthcoming Soar model of manipulating the car by controlling the extremities will be integrated into the multitasking structure, and speed will be controlled by active manipulation of the car rather than direct value settings set up through Lisp commands.

Limitations of Soar4. One conclusion from the attempt to use Soar in modelling multitasking behaviour is that Soar4, the Soar version that was used to implement the model discussed in the present chapter, is not an entirely adequate vehicle for this endeavour. The most constraining factor is that, in Soar4, data structures in working memory cannot be modified destructively: data can, in principle, only be added. Yet, the deletion of data can be achieved in Soar4 in two ways. The first, conventional way, is to replace a state by a new state and to copy only some of the contents of the old state, thereby indirectly achieving deletion. The second way of deleting, used heavily in the present model, is to simulate deletion by invalidating data structures. Invalidation is achieved by marking data structures as 'old' or 'non-existent.' However, this procedure slows down Soar's matcher considerably with the increasing number of objects made invalid. The main reason for having deletion at the top level is to allow both (parallel) perception and (parallel) interrupts at the top level without interrupting the activity in the goal stack. This, however, requires some very complicated bookkeeping of interrupts and perceptual objects at the top level. With each new perceptual object coming in, old objects have to be marked as 'old' and a link between the new object and the old one must be established. When processing in the goal stack (while doing the navigation task) proceeds for some protracted period of time the list of objects at the top level becomes intolerably long, bringing Soar almost to a halt. Only after an interrupt most of the perceptual objects and the old interrupts at the top level can be removed.

The revised model of multitasking behaviour described in part II of this study will adopt Soar5 because it supports both destructive state modification and a truth maintenance system (see Laird et al, 1990), thus removing, or at least reducing, these essentially logistic problems.

2.5 Notes

¹ The paper was slightly edited to avoid redundancy.

² One could conceivably use a single goal representation by defining DRIVE as the top goal that includes all other goals related to the driving task. However, since people are apparently capable of engaging in other tasks while driving, the single goal would be self-defeating.

³ Representing moving objects as a sequence of objects with the same identity is one of the drawbacks in Soar4 that was eliminated when Soar5 was introduced. The reason is that, in Soar5, it is possible to destructively modify data structures. As a result, the properties of an object and its relations to other objects may now change without affecting its identity.

⁴ One difficulty that we have with the mapping of automatic and controlled processing onto Soar is that a string of operators might be so well learned that no subgoals are required to solve the ordering between operators. This state of affairs has been identified by some authors as "veiled processing," different from automatic processing in the proper sense.

3 Flattening Goal Hierarchies

Summary: this chapter, published earlier as Aasman and Akyürek (1992), addresses some of the problems raised in the previous chapter. We showed that there is at least a practical inconvenience with and we might even argue fundamental inconsistency in Soar's default rules and Newell's tasking and interrupt operators. The present chapter examines the current default rules for operator tie impasses in Soar in relation to known constraints on human memory and real-time requirements imposed on human behaviour in dynamic environments. The analysis shows that an alternative approach is required for resolving such impasses, which does not ignore these constraints. We will explore alternative sets of rules that appear promising in meeting them. One of these variations is used in the driver model discussed in part two of this study.

3.1 Introduction

The growing body of Soar literature shows that both artificial intelligence (AI) researchers and cognitive psychologists do benefit from using the Soar system in their research (e.g., Steier et al., 1987; Lewis et al., 1990). Soar is considered an AI architecture for general intelligence (e.g., Laird, Newell, & Rosenbloom, 1987; Laird, Rosenbloom, & Newell, 1986; Rosenbloom, Laird, Newell, & McCarl, 1991) as well as an architecture for cognition (Newell, Rosenbloom, & Laird, 1989), and is said to instantiate a unified theory of cognition (Newell, 1990). The notion of architecture is taken here to mean "the fixed system of mechanisms that underlies and produces cognitive behaviour" (Newell, Rosenbloom, & Laird, 1989, p. 94). The claim that Soar is an AI architecture for general intelligence is not at stake here. The real concern of this chapter is Soar as a unified theory of cognition (UTC). Some properties of Soar as a system for general intelligence appear to weaken the Soar theory as a candidate UTC, that is, they make the system too powerful as a theory of human cognition. One of the strongest points of Soar is its combination of the basically 'chaotic' nature of a parallel production system with the tight administrative powers of a *goal stack* and the simplicity of an efficient

chunking mechanism. Another is the set of default rules that supports the problem-solving behaviour of Soar as a weak method problem solver. But, why would psychologists worry about such a powerful combination? The problem is that Soar seems to deviate from human cognition in significant ways: (a) the size of working memory is virtually unbounded; (b) learning that occurs in a single problem solving episode is too fast; (c) backtracking to previous problem states and noticing duplicate problem states during search are virtually effortless; and (d) as a real-time system Soar does not seem to adequately match the interrupt-driven character of human behaviour. It is the contention of the present chapter that these issues arise in large part from the behaviour of the current default rules that involve operator tie impasses. The following section briefly describes the key concepts involved and their relationship to the above issues. The *interrupt* issue will be dealt with in a section of its own.

3.2 The Default Mechanism for Operator Tie Impasses

A goal (or context) stack is a temporary data structure in working memory that links together the *goal contexts*, each of which is composed of a goal (G), together with slots for the current problem space (P), state (S), and operator (O). A goal context is frequently referred to, in an abbreviated form, as context or simply goal. Any context slot is allowed only a single value at a time. The desired state¹ that Soar is set to achieve is usually specified through a slot of G, called *desired*. The first element of the stack is called the top level goal or just top goal for the obvious reasons. All other goals below it are subgoals, created in response to impasses in problem solving. Hence, a goal stack represents a *goal hierarchy*. An *operator tie* impasse occurs when two or more operators are proposed while none of these has a better preference than the others. An operator tie impasse can be resolved when additional preferences are created that prefer one option over others, eliminate alternatives, or make all of the operators indifferent (Laird, Congdon, Altmann, & Swedlow, 1990). In Soar, a subset of default rules implements a mechanism in order to decide operator tie impasses. This mechanism works as follows. Given a task, task rules will select a problem space within which to attempt the task. In this problem space, an initial state is selected and problem solving proceeds by repetitively proposing, selecting, and applying task operators to the state. In response to an operator tie impasse Soar itself sets up a subgoal. The goal context of a tie impasse is also called a *tie-context*. Default rules that are responsive to the tie impasse propose the *selection problem space* in the associated subgoal. If no competing problem spaces exist, the selection problem space gets selected. An empty initial state is then created, where evaluations are posted as soon as they become available. The *evaluation* operators of this space evaluate the tied task operators, a process that allows preferences to be created for the latter. To compute an evaluation, an *evaluation operator* is proposed for each of the tied task operators. The evaluation operators are mu-

tually indifferent, so Soar can select one at random. Since this is an abstract operator in an abstract space, an *operator no-change* impasse arises. As a result a further subgoal is created, known as the *evaluation subgoal*.

There exist default rules that are sensitive to the evaluation subgoal and they propose the problem space of the tie-context, that is, the one above the selection problem space. If this problem space gets selected, these rules also propose a new state and upon its selection, they copy down relevant information from the state of the tie-context to the new state. The principle behind this is that Soar will try to assess the ramifications of the task operators without actually modifying the original task state. Technically, the system is said to engage in a *look-ahead search*. Thus, the operator that is linked to the evaluation operator is selected in the evaluation subgoal and applied to the *copy* of the task state, leading to one of the following results.

Case 1: Another operator is selected. The modified state does not match the desired state, and there is only one operator available or the preferences favour a single operator. This operator then gets applied to the current state without further subgoaling.

Case 2: Another operator tie impasse occurs. The modified state does not match the desired state, and there is a set of instantiated task operators to select from. This will cause additional levels of subgoaling.

Case 3: An evaluation is computed. The modified state does not match the desired state, but a symbolic or numeric evaluation exists for the modified state. Task specific rules put this evaluation into a slot of the state of the selection problem space as the result of the evaluation operator. Default rules that are sensitive to this slot will then terminate the current evaluation operator and proceed to the selection of another evaluation operator. When eventually enough evaluations have been computed so that the operators in the tie set can be compared, default rules convert them into preferences that break the tie in the original task space. Of course, if success is detected—that is, the desired state has been attained in the evaluation subgoal—a best preference will immediately resolve the tie. This eliminates the need to apply the rest of evaluation operators. Figure 1 summarizes the whole process described so far in a schematic way. When an evaluation is computed, two chunks are learned: one pertaining to the selection problem space, the other to the problem space of the tie-context. In the process, it is possible and, in fact, fairly common that the whole solution path, the operator sequence that achieves the desired state, is learned at once.

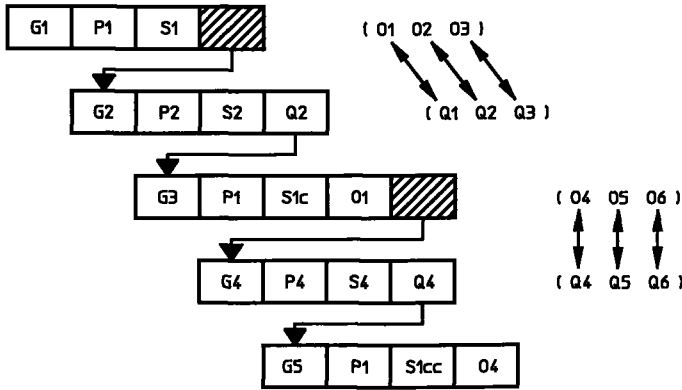


Figure 3-1. Soar's default mechanism for operator tie impasses. Shaded areas indicate tie-contexts where an operator tie impasse occurs. G1 is the top-level goal: P1 is the task problem space. S1 the task state, and {O1 O2 O3} the set of tied task operators. G2 is the subgoal generated because of the tie impasse: P2 is the selection space and S2 will contain the evaluations for O1 to O3 when they become available. {Q1 Q2 Q3} is the set of evaluation operators for {O1 O2 O3}. G3 (i.e., the evaluation subgoal) is the result of the operator no-change impasse associated with O2: It shares P1 with G1 and its state, S1c, is a copy of the task state S1. It can be seen that O1 is tried on S1c. Again an operator tie impasse occurs, followed by another instantiation of the selection space, and so on.

3.3 Issues

3.3.1 Memory

Current default rules behave such that the goal stack can grow to arbitrary depths. As a result, they impose an unrealistic load on the working memory. Newell signalled this issue and its psychological implications with respect to Soar4, and discussed the *single-state principle* as a potential answer (Newell, 1990). Soar5, the current version of the Soar system, does indeed employ this principle according to which Soar should keep only a single state in working memory when doing search in a problem space. Whilst copying states within goals is reduced to some extent in Soar5, between goals it is still very much present. This is due to the fact that default rules do not obey the single-state principle when they implement look-ahead search to determine the relevance of tying operators. As described in the preceding section, their effects are simulated on *copies* of the superstates. Since the goal stack is unbounded, the number of intermediate state copies that need to be held in working memory is in principle also unbounded.

It is fairly trivial to show that the look-ahead search these rules implement cannot be the scheme that humans use. With just the rules of chess at its disposal, the default rules would allow Soar, unlike humans, to traverse a search tree of, say, 20 levels deep in a game without using, for example, an *external*

chessboard for guidance. Improbable for humans but quite possible for Soar is that, while 20 levels down in the search tree, it can decide on rejecting, say, a move at level 8, go back to that level, and consider a different move.

Human working memory may normally contain up to seven or eight chunks of items at a time. Making this constraint a built-in feature of Soar by *fixing* the size of its working memory will not help, because memory-hungry processing mechanisms like the current look-ahead would immediately break down. It is rather more likely that the processing mechanisms themselves are adapted to this feature of working memory, that is, the size of their output is inherently constrained (cf. Bobrow & Norman, 1975; Elgot-Drapkin, Miller, & Perlis, 1987). This point will be treated here as a requirement.

3.3.2 Search

One issue related to search is that in Soar the goal stack allows for virtually effortless backtracking, that is, branching back to a previous problem state to consider alternatives without this incurring any cognitive or computational cost. This is presumably a desirable property for an AI system, but it does not quite mimic human problem solving where one may occasionally hear: "OK, this is all wrong, but how did I get here in the first place?" The default rules make it possible to maintain a large search tree in the goal stack with relevant intermediate states available at each branch. Evaluation operators that are not in the operator slot will stay around in memory and remain 'visible' to productions. This enables Soar—to return for a moment to the chess example—to see at level 20 that an operator at level 8 is 'wrong,' then backtrack to that level and continue the search from there. However, a closer examination of the problem solving protocols and their analysis provided in Newell and Simon (1972) makes it convincingly clear that backtracking is never this simple. Most of the time it requires an operator or a set of productions. If humans jump back to a previous state, they will nearly always jump back to the initial state, or to a stable intermediate state—often using external memory to find such a state. In any case it is much less automatic than is currently in Soar. One partial way out of this problem is progressive deepening. Newell and Simon (1972) have claimed that this is the strategy humans appear to use most in their problem solving. Although progressive deepening has been attempted in Soar, it has always been implemented, like most other weak methods, as a method increment to the selection problem space and evaluation subgoalings (see Laird, 1984). Such an implementation imparts of course processing characteristics similar to the current look-ahead method. An ancillary issue is the relative ease of detecting duplicate states. Because copies of problem states are available throughout the goal stack, it is very simple for Soar to detect recurring problem states. Humans too are capable of noticing such duplicate states: "Hey, I was here a few steps before, what did I do wrong?" But the question is, if they cannot apparently hold extended records of their problem solving in their working memories, how do they succeed in noticing such states?

A requirement that derives from the discussion above is the following: the process that investigates the effects of a candidate set of task operators must be organized in a such way that it does not capitalize on deep goal stacks, while known weak methods of problem solving (e.g., see Laird & Newell, 1983; Laird, 1984; Nilsson, 1971; Rich, 1983) can be based on it.

3.3.3 Learning

Currently, in look-ahead search, learning comes at almost no cost. Operators are applied to copies of higher states, and the resulting evaluations passed up to the operators in the original space. Because an evaluation is dependent on the changed copy of a superstate and a superoperator, the resulting chunk will contain appropriate state and operator conditions so that a correct evaluation chunk is learned. As such, this is a very useful mechanism to have. However, in combination with a deep goal stack it may lead to curious phenomena such as learning an entire solution path as soon as success is detected while looking ahead. If one accepts that ten seconds per chunk is a serious estimate for the speed of learning (Newell & Simon, 1972, p. 793), then it is a bit odd to observe that Soar should be capable of learning, say, 40 chunks in just a few simulated seconds.

There is also sufficient evidence that learning can increase the computational cost of processes that underlie problem solving (Minton, 1988; Tambe & Newell, 1988; Tambe & Rosenbloom, 1989). The chunks that are learned may have, for example, low utility and/or high matching cost. A contributing factor no doubt is the number of chunks learned, that is, how efficiently the acquired knowledge is stored. A requirement that follows from these observations is that default rules should not cause learning more chunks than are needed for efficient problem solving.

3.4 Alternative Approaches to Deal With Ties

As was pointed out before, an operator tie impasse is currently resolved in the selection problem space. As can be seen from Figure 1, the knowledge needed for distinguishing a single choice among the operators of a tie-context is often secured in the evaluation subgoals below a selection problem space. Evidently, the selection problem space adds to the depth of the goal stack that obtains during the look-ahead search. Since the Soar architecture creates a chunk for each result obtained in a subgoal, many chunks will be included into its production memory, ensuing from both the selection and evaluation subgoals. As will be shown below, it appears that the selection problem space is not really needed to generate the knowledge required for deciding among alternatives. In fact, several sets of rules can be conceived that do not call for it.

3.4.1 Alternative 1 eliminates the selection space

In the first scheme to be considered, following an operator tie impasse the problem space of the tie-context is selected, and a new state is created in the

associated subgoal for every tying task operator (see Figure 2). These states are given indifferent preferences and one is chosen at random. From this point on, the problem solving that takes place is analogous to that of the current default rules when they attempt to evaluate task operators. Thus, the superoperator is installed in the subgoal, while the relevant aspects of the task state are copied to the state specific to that operator. This operator is then applied to the copied state.

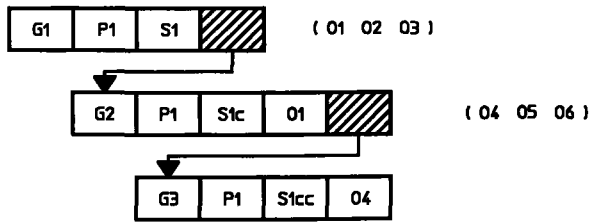


Figure 3-2. An alternative default mechanism for operator tie impasses that eliminates the selection problem space. The shaded areas indicate tie-contexts. In this scheme, an operator tie impasse may also occur following an operator application in the same subgoal, set up to resolve an earlier tie impasse. G1 is the top-level goal; P1 is the task problem space. S1 the task state, and {01 02 03} the set of tied task operators. G2 is the subgoal generated as a result of the tie impasse: it has the same problem space as the supergoal, that is, the task problem space P1. Its state is a copy of the task state, and it is created to evaluate a specific task operator. Thus S1c is a copy of task state from the supergoal on which 01 is tried out. As soon as 01 has been applied another tie impasse arises, consisting of {04 05 06}. This causes Soar to set up another subgoal, G3, whose state S1cc is a copy of the superstate, created to try out 04 in turn.

If no evaluation is available, problem solving continues, possibly with a new set of instantiated task operators. If a numeric evaluation is available, this evaluation is stored on the operator, another operator from the tie set is selected, and again the superstate is copied down to the state created for it. When the tie set is exhausted, available evaluations are converted into preferences, breaking the tie, upon which the winning operator is selected in the supergoal and applied. If success is detected before the tie set is exhausted, the subgoal terminates instantly, and the operator responsible for success is directly selected and applied in the supergoal. If a failure is detected, the operator gets a reject preference, and problem solving continues with another operator from the tie set. Note that all problem solving in this scheme is carried out in the original task problem space.

Compared with Soar's current default mechanism, the scheme outlined above (a) requires fewer default rules, (b) produces a smaller goal stack, (c) reduces the number of chunks learned by one half—due to the absence of the selection problem space, and (d) induces faster performance, both during learning and after learning. Moreover, its rules are much simpler to understand. How-

ever, the processing that this scheme instantiates still does not constrain the size of the goal stack, and the state copying that it needs for evaluating candidate task operators is identical to that of the current Soar scheme.

3.4.2 Alternative 2 avoids ties

In the second scheme, to which we turn now, ties do not arise. The idea is that if a task operator is proposed that does not have an evaluation attribute, the operator may be conjoined with a *what-if operator* (see Figure 3). Instantiations of the latter are made better than the task operators, and are also made indifferent to one another so that one of them will be selected immediately at random. And, as soon as Soar enters an operator no-change subgoal, a default rule will bring down all of the superoperators to the new context *without testing the what-if operator in the supergoal's operator slot*. The reason for doing this is to ascertain that the what-if operators will not show up in the chunks that are built. The supergoal's problem space and a new state which is intended to hold a copy of the superstate are then proposed. If the latter are selected in the subgoal, the superstate is copied down to the new state. Next, the task operator which was conjoined with the what-if operator that occupies the supergoal's operator slot is selected and applied to the state in the subgoal.

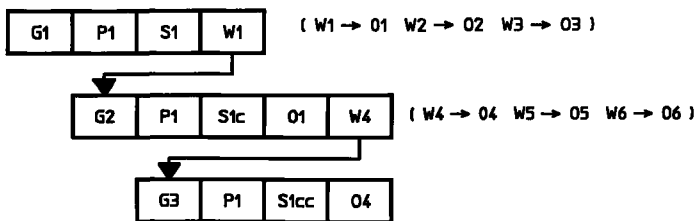


Figure 3-3. An alternative default mechanism for look-ahead search in which operator tie impasses will not occur, for each task operator instantiation is assigned a *what-if operator* with an indifferent preference. G1 is the top-level goal: P1 is the task problem space. S1 the task state, and {W1 W2 W3} is the set of what-if operators whose every element is paired with an element of {O1 O2 O3}, the set of instantiated task operators. G2 is the subgoal generated because of the operator no-change impasse, associated with W1: the subgoal has the same problem space as the supergoal, that is, the task problem space P1. Its state is a copy of the task state, and it is created to evaluate a specific task operator. Thus S1c is a copy of the task state from the supergoal on which O1 is tried out. As soon as O1 has been applied, other task operators {O4 O5 O6} are instantiated, which leads to the creation of corresponding what-if operators {W4 W5 W6}. This causes Soar to set up another subgoal, G3, whose state S1cc is a copy of the superstate S1c, created to try out O4 in turn.

If no evaluation is available, problem solving will continue, possibly with a fresh set of instantiated task operators. If a numeric evaluation is available, however, that evaluation is passed up to the supergoal. The associated what-if operator retracts, and as a result, the current subgoal will be terminated. Next, another what-if operator gets selected, and again Soar enters a no-

change subgoal. This process is repeated until all task operators have their evaluations computed. When the set of what-if operators is exhausted, the available evaluations are converted into preferences so that a choice decision can be made in the supergoal. The winning operator is subsequently selected in the supergoal and applied. If success is detected, the current subgoal is terminated and the operator responsible for success is given a require preference upon which this operator is selected immediately in the supergoal, and applied to the state of that goal. All remaining what-if operators are then rejected, eliminating the need to evaluate the task operators with which they are paired. If a failure is detected, the task operator gets a 'failure' preference, the associated subgoal terminates due to the retraction of the what-if operator in the supergoal, and problem solving proceeds with another what-if operator from the extant set.

Like in the first alternative, all problem solving is again carried out in the original task problem space. Also, it should be noted that in the present scheme—as well as in the next alternative—a particular preference language is used on top of the one provided in Soar. The reason for introducing such a language is to guarantee that every production that creates a preference will participate in backtracing during chunking process.

The present alternative, however, forces Soar to enter into a no-change subgoal even if there exists only one instantiated task operator. Note that this can be avoided by having rules that determine the number of the instantiated task operators, $n(O)$, and create what-if operators only when $n(O) > 1$. It compares nevertheless to Soar's current default mechanism as favourably as the first alternative above. The next alternative addresses the issue of state copying by introducing destructive look-ahead.

3.4.3 Alternative 3 eliminates state copying

It is also possible to shadow operator tie impasses, as in Alternative 2, and use the task state for operator evaluations instead of copies of the superstate. This scheme runs as follows. All task operators are by default indifferent, that is, they are proposed with indifferent preferences. Soar will select one at random and apply it, thereby directly modifying the task state. As such, this scheme differs from the previous ones and from Soar's current scheme in that it implements a destructive look-ahead search: whenever an operator is selected, the actual task state is changed, instead of a copy of that state. A record is kept, however, of the operators that have been applied (see below).

If a failure is detected, a *reversal operator* is applied to the result state, say S' , to restore the state S that prevailed prior to the failed operator. It should be clear that returning to the previous state leads to new operators being proposed, identical to those proposed earlier. One of them is of course a new instantiation of the operator that led to failure. Two approaches can be taken from here to prevent the latter operator from being reselected and applied again. The first is to let a default rule compare each operator to the most recent operator that failed and give it a reject preference if their descriptions

match. The second is to *explicitly* learn that the failed operator is wrong. The second approach seems more appropriate, for it is in accordance with the psychological observation that failure is often a trigger for deliberate behaviour. Thus, having reinstated the original state after an operator which is evaluated to failure, default rules install a *learn-reject operator* with the failed operator as its argument. Using common data chunking procedures, a chunk is learned to the effect that if task description = x and operator = y then y will be rejected. After rejecting this operator, other task operators get a chance. The processing just outlined can also be used for other symbolic evaluations such as success as well as numerical evaluations. This has not been tried out thus far, but no difficulties are expected.

Whereas it is important that destructive look-ahead search does not require a goal stack due to the absence of selection problem space and state copying, it does have a number of drawbacks. In the first place, the capability for backtracking is lost and, consequently, effort must be invested in storing the applied task operators. Effort is also required for creating and applying reversal operators—partly in order to regain a *non-automatic* capability to enable backtracking. In the second place, *explicit* learning is needed to control problem solving. And, finally, duplicate states become difficult to detect.

In non-destructive look-ahead search, the goal stack ‘remembers,’ as it were, choice points in the state space, that is, points where to backtrack. In destructive look-ahead search, on the other hand, it is necessary to take explicit action in order to return to previous states. This requires recalling at least the last operator that was applied so that Soar can go back at least one step and learn an evaluation for that operator. It is obvious that search efficiency will improve as the number of recalled operators increases. Also, Soar must be provided knowledge about how to create reversal operators and when to install them. The latter requirement is not too severe considering current Soar programming practice, because most Soar programs already contain productions that reject an inverse operator. Thus, for instance, after a $Move(x, L_1, L_2)$ has taken effect, $Move(x, L_2, L_1)$ is rejected as the next action. With the present scheme it must be enforced that they are generated. These inverse operators should not be rejected but just made worse and *marked* as reversal to the effect that they can be recognized when needed.

Reversal operators are, as argued above, also associated with the problem of finding a place to return to in the state space when it is or seems appropriate to do so. Note that automatic backtracking is one of the issues that the present scheme is meant to address. A phenomenological description of this situation is that occasionally we seem not to remember how we got into a certain state, that is, we do not know which operator or operator sequence will bring us back to some previous state. One reason may be that reversing a partial operator path leads to a state where no reversal operators can be recognized or created because there is no path information available. There are basically three options a search-based problem solver can adopt after reaching a state that does not contain reversal information. The first option is to continue with

the current state and hope that eventually a solution will be found. The second is to jump back to the initial state. This is probably the simplest strategy a problem solver can use, also because the initial state often can be delivered by perception. The third option is to jump back to a stable intermediate state, which is only possible if the problem solver invests sufficient effort in 'remembering' important, or promising intermediate states it encounters. As was pointed out earlier, with Soar's current default rules which arrange for look-ahead search, learning comes at almost no cost. When an operator is applied to a copy of a higher state and leads to an evaluation, this evaluation is translated into an appropriate preference for the operator. Since such an evaluation depends on the changed copy and the operator, and the copy in turn depends on the superstate, the resulting chunk will contain the appropriate state and operator conditions so that a correct evaluation chunk is learned. When, however, the goal stack is not used directly to select from a set of candidate operators, learning must be *deliberate*. It is indicated that we design a mechanism which guides building a chunk that references the appropriate state and operator in its condition side while its action side can add the relevant evaluation to the operator. To this end, we first explain how, in the destructive look-ahead scheme under discussion, explicit rejections are learned. Proposing and selecting a *learn-reject* operator has been explained previously. The recipe for implementing this operator consists of the following seven steps: (1) Operator *learn-reject* is installed, with as argument the operator to be rejected; (2) In every no-change subgoal that occurs the problem space "LEARN-REJECT" is proposed; (3) This particular problem space is preferred over all others whenever a *learn-reject* operator is detected in the operator slot of the supergoal: This problem space is now independent of the superoperator because the chunking mechanism does not backtrace through desirability preferences; (4) All superoperators are proposed; (5) The operator linked to the *learn-reject* operator as its argument is made best and installed in the operator slot of the subgoal; (6) Two productions *recognize* the top state and the operator in the slot, each adding a unique symbol to the state; And, (7) a default production looks for these two unique symbols and rejects the superoperator that should be rejected. This results in a chunk that is exactly the same as a chunk which gets built during Soar's current look-ahead search. It should be noted that this recipe can be used for explicit learning of all evaluations.

Having the reject action built into the right hand side of the chunk is the easiest part: it is just a consequence of the fact that a reject evaluation is the only 'result' that can obtain in the learn-reject problem space, once it is selected. The hard part is getting conditions on the left hand side right. For simple tasks it is easy to test for all aspects of the task representation and get the task representation as a whole into the condition part of the rule. However, this approach sometimes turns up overly specific chunks. Also, the recognition production is committed to test for all aspects of the task, whereas only a few of them may be needed. In the farmer puzzle, for instance, it will do no harm

to test for the whole state, though the resulting chunks will be quite large, and probably expensive (Tambe & Newell, 1988). A further point is that a recognition production has to be written for every task, and this could be difficult if a task's representation tends to change during problem solving. One solution is to have a general *recognize operator* that checks the problem space symbol for the name of the relevant task class so that only the task-relevant information is recognized. Unfortunately, we will still get the whole task representation into the chunk and, consequently, the recognition process will be very slow. Another solution is to remember all the task objects that were modified, added, or deleted while applying an operator. When we backtrack by applying a reversal operator and want to have an evaluation chunk built for the companion task operator, we would need to recognize only the objects that have changed.² This solution is actually a syntactic one; it might be far better to rely on a semantic recognizer which can use the available task knowledge to investigate the relation between the operator to apply and the task objects.

A last point to consider is the detection of duplicate states. State-space search must also deal with the occurrence of cycles or duplicate states. Although it is undecidable whether a state higher up in the goal hierarchy and the current state that is a duplicate of the former can be said to serve the same function, it is nevertheless a good heuristic to avoid duplicate states. Using Soar's current default rules, for most tasks a production can be written that creates a reject preference for the current state when it is the same as a state higher in the context stack. Likewise, the present scheme needs a way to tell duplicate states apart and to act accordingly. It is obvious that dealing with such states has now become much more complex: how can a (current) state be compared with a previous state when the latter is nowhere around? If a state does not have a recognizer chunk, a *recognize-state operator* will build one for that state. A recognizer chunk will test for relevant aspects of the task state for which it was built and add a unique recognition symbol to it. The recognition symbol will disappear from a state if the state changes. Whenever an attempt is made to install the *recognize-state operator* for recognizing a given state and that state has already a unique symbol added to it, this should warn for the fact that the current state might be a duplicate of some previous state. This, of course, presupposes that one knows in advance that the state representation will never change.

A crucial advantage of the alternative just reviewed is that it potentially meets all requirements imposed by the various issues related to memory, search, and learning. As will become clear in the next section, it also provides a better ground for dealing with interrupts.

It should be noted that some attractive variations to Alternatives 1 and 2 can be constructed, for example by allowing only a single copy in the look-ahead search that they implement and then applying the ideas developed under Alternative 3. This suggestion implies that the goal stack can expand at most

one level or, in other words, it can have just a top state and one substate, where the substate is a copy of the top state.

3.5 The Interrupt Issue and Its Relation to Default Behaviour

Previous sections have dealt with memory, search, and learning issues engendered by a subset of Soar's default rules that organize problem solving in terms of the selection problem space, and alternative sets of rules that did not use this problem space at all. In this section we discuss the processing of interrupts which, in our opinion, must be taken as a basic, real-time operating characteristic of human behaviour. For this reason, it constitutes an additional constraint on the "natural modes of operation" of problem solvers, human or artificial. Default behaviour is of course one such mode.

If you are interrupted by the phone that starts ringing while in the midst of figuring out an algebra problem, you normally stop what you were doing, attend to the phone, and later return to the algebra problem. Or, while you are driving, you engage, still driving, in a conversation to settle a financial matter with a friend in the back seat. Since such examples can be multiplied indefinitely, it should be obvious that human behaviour is strongly "interrupt-driven" (e.g., Reitman, 1965). Does Soar behave in the same way? Newell (1990) argues that it does. Soar is in principle capable of multitasking, of doing several tasks simultaneously (see Aasman & Michon, 1992). But, the fact is that the processing of interrupts doesn't mesh well with the basic "cognitive mode of operation" that current default rules implement.

Building the goal stack during tie handling is a process that takes a certain amount of time. Independent of the tie processing related to a task *A*, an interrupt cue may penetrate into the system (e.g., a ringing phone), or operators may be created at the top level for a different task (e.g., a conversation). These "interrupt" operators have in principle the capability to destroy the entire goal stack. Technically speaking, such operators (generated at the top level) may displace the original tied operators. As a result, the goal stack will be lost and needs to be rebuilt afterwards. The real-time constraint that this process imposes on the default rules is obvious: since the depth of a goal stack determines the time to rebuild the stack, it is more efficient to have flatter stacks. As was shown, this objective can be achieved by eliminating the selection problem space and/or state copying. It must be noted that there are alternatives to the view that the goal stack is lost. One is to include, like in older versions of Soar, a 'suspension' mechanism that restores the complete goal stack after an interrupt. Another is to insert interrupt operators in the most recent goal context (Hucka, 1989). The former requires extra space, and so aggravates the memory issue. The latter appears to be at variance with the principle that interrupts should be processed in the base-level problem space (Newell, 1990).

Table 1. Rebuild times involved with different rule sets (Learning On)

Rules involved	1st Operator	2nd Operator
Current default rules	20	34
Alternative 1	11	18
Alternative 2	13	26
Alternative 3	3	9

Adopting a set of default rules obviously determines to a great extent the temporal unfolding of behaviour. Table 1 shows stack rebuild times that different sets of default rules require in terms of number of production cycles. Its first column indicates how long it takes to select and apply the first of the tied operators in look-ahead.

It is obvious that Soar's standard default rules take more cycles than any of the other approaches discussed in this chapter. We first have to go through a selection problem space before we can apply an operator in look-ahead. Note that we have gained nine cycles by eliminating the selection problem space (Alternative 1). If we use the what-if approach (Alternative 2), the gain is slightly less but still impressive. Alternative 3, which implements destructive look-ahead, clearly wins. Measuring the rebuild time (in number of production cycles) by looking only at the first operator does not show the efficiency of the approach. We therefore included in the last column the time to reject the first operator, and then to select and try another operator. Here we observe that the elimination of the selection problem space is a real time saver; the overhead per operator is only seven cycles. It should be noted that the table includes numbers for Alternative 3 when no explicit learning is done. When learning is deliberate (see p. 201), Alternative 3 needs 20 cycles to reject the first operator, and then to select and try the second operator. This overhead is rather large, but for small n this alternative set of rules is still more efficient than Soar's current set of default rules.

3.6 Concluding Remarks

In this chapter we have presented the results of a "breadth-first" effort to find suitable alternative sets of rules, similar in function to the current default rules, but satisfying an important set of requirements. The alternatives that have been considered have one property in common. They do not use the selection problem space which is essential to Soar's current default rules. If the selection problem space is eliminated, the goal stack flattens considerably. Alternative 3 is probably the most attractive one in that it adheres most strongly to the single state principle and obeys known constraints most closely. Also, it probably makes interaction with the external world as tractable as a recent scheme, based on the means-ends heuristic, that operates directly on the original task state (Akyürek, 1992). Some real problems seem to be raised by Alternative 3, and these will have to be addressed, but the approach successfully avoids serious issues related to memory, search, learn-

ing, and interaction with dynamic environments. Psychologists therefore should not be weary to use the destructive look-ahead search that it implements. With some effort, its current implementation can be changed so that it will allow the full functionality of the current default rules.

Acknowledgements

This work was supported by the Dienst Verkeerskunde van Rijkswaterstaat (Netherlands Ministry of Transport) under project number TO 8562, by IBM Nederland under a Study Contract with John A. Michon, and also in part by a grant (Dossier No. R 57-273) from the Netherlands Organization for Scientific Research (NWO) to the second author.

3.7 Notes

¹ Also known as "goal state" in artificial intelligence".

² This option has not been implemented.

4 Basic driver operations in negotiating general-rule intersections

Summary: This chapter presents an analysis of basic driver operations in the approach to and negotiation of general-rule intersections. The analysis is performed on data taken from experiments by De Velde Harsenhorst and Lourens (1987, 1988). The basic operations discussed are speed control, car control and visual orientation. The analysis is used to validate the driver model developed in the second part of this study.

4.1 Introduction

In this chapter we describe a number of observations with respect to human driver behaviour in the approach and negotiation of unordered intersections. The observations derive from two field experiments by De Velde Harsenhorst and Lourens (1987, 1988). In their experiments an instrumented car was used to record a broad range of driver behaviour. The first experiment recorded 25 driving lessons of one novice driver. To enable her progress to be monitored, the novice completed a fixed 20-minute route through a residential quarter at the end of each lesson. In the second field experiment a group of 24 more advanced drivers also completed the same 20-minute route. The experiments generated a rich set of data as the instrumented car recorded speed, steering-wheel angle and brake, accelerator and clutch pressure. A set of video cameras recorded the eye and head movements and the traffic environment (covering 180 degrees in the forward direction). In addition, all the instructor's comments in the first field experiment were recorded.

Since the data from these field experiments are essential to our modelling efforts in part II of this study, we were very pleased that we were given permission to analyse these raw data in greater detail for our own purposes. The relevance of the De Velde Harsenhorst and Lourens data is twofold. In the first place it provided a description of the behaviour of drivers in the approach to and negotiation of intersections. Secondly it also provides some insight into the underlying rules that generate this intersection behaviour.

Before describing the reanalysis in more detail, the following section discusses why we chose the De Velde Harsenhorst and Lourens data for this study and how these data will be used later on in this study. The question why these data are so important starts with two other questions. First, *given the objectives of our cognitive model*, what are the driver operations that we need empirical information about? And, second, what do we wish to know about the main factors that determine the timing and ordering of driver operations in the negotiation of intersections?

4.1.1 Selected driver operations

The main objective that determines the selection of the driver operations for which an empirical underpinning is required is our constraint to take into account the human *perceptual* and *motor* systems. This constraint alone will not generate a list of behaviours, so let us combine it with a simple analysis of the driving task.

The crucial fact in the driving task is that a human driver operates in the traffic world by manipulating principally two parameters, namely speed and course¹. This implies that in any case we require empirical information about the overall speed and course profiles in the approach to and negotiation of an intersection. The second step in this analysis is that a human driver manipulates both speed and course by operating the basic car devices (brakes, accelerator, clutch, steering wheel, gear-stick). Since our cognitive model models motor actions we need empirical data about the timing and use of these car devices. The third step in this analysis is the fact that a driver bases his actions on his perceptual information. This, primarily visual information², can only be obtained optimally by scanning the external world intelligently. Since our model also models eye and head movements, we need empirical information about visual orientation.

The following provides a summary of research issues and driver operations that are of interest. In Section 4.4, where the results of our analysis are described, the relevance of these issues to the behaviour of our cognitive model is discussed in more detail.

Speed control. The main issue here is the overall shape of the speed profile, including for example, the maximum speed before the intersection, the speed when entering the intersection, the shape of the speed profile in terms of maximum acceleration and constancy of acceleration.

Motor actions and car-device control. Speed control and course control are implemented by manipulating car devices. If the use of the brake is taken as an example one can ask questions such as: What does the overall profile look like? Is it a simple function with discrete events or is it a complex, non-linear

function? When does a driver start to brake and when does he start to release the brake? Does a driver time his deceleration by varying the brake pressure or by timing the release of the brake? Similar questions can be asked regarding the use of the accelerator and clutch.

Visual orientation. The main question here is: how do factors such as time-to-intersection, distance-to-intersection and the prevalent traffic situation determine the direction in which a driver looks? At any one time a driver can perceive only a small part of his environment in focus, i.e. just a few degrees of his field of vision. However, the interesting and for his driving behaviour possibly relevant events may take place anywhere in the entire 360-degree area³. The driver is thus required to intelligently sample his environment to obtain the information about the most relevant data.

Course control and steering. The relevant issues here are for example the frequency with which course corrections are carried out and, for turning at an intersection, when and where the steering manoeuvre starts, when the maximum angle is reached, etc.

Integration. Drivers carry out many actions (some of them in parallel) in the short time before the intersection. The question is whether there is any consistent relation between these actions. For example, when and where does a driver start to release the brake in the approach to the intersection: before or after looking to the right?

Timing cues. The final issue that is relevant for the implementation of our cognitive model is what cues drivers use in timing the basic driver operations. For example, is the moment at which a driver starts to brake determined by the *distance-to-intersection* or, as Van der Horst (1990) suggests, by *time-to-intersection*, a measure that corrects the distance to intersection for speed?

Individual variation. Another interesting issue is the consistency of behaviour between drivers. The more consistent the behaviour of drivers is, the more useful it is to compare behaviour of our cognitive model with that of 'average' drivers.

4.1.2 Situational factors determining driver behaviour

It is clear that driver behaviour and driver operations are determined by situational factors. In the work of De Velde Harsenhorst and Lourens the following four (common-sense) determinants seem to be the most prominent in the approach to and negotiation of intersections.

Type of intersection. The first important factor is the type of intersection. This factor determines both speed control and visual orientation. Car drivers reduce their speed more if they are formally required to yield at the intersection or if

visibility at the intersection is restricted or if they approach an intersection with a higher traffic volume (Van der Horst, 1990). Drivers will approach an intersection more boldly if they have right of way or if they are approaching a smaller road. An example of the influence of "type of intersection" on visual orientation is that if a driver is approaching an intersection where he will have right of way, it is less important to look to the left or right than if he is approaching a yield intersection⁴.

Manoeuvre. A second determinant is the manoeuvre performed on the intersection. One might expect a difference between left turns, crossings and right turns with respect to speed profile and visual orientation. For example, if we compare a left turn to a crossing manoeuvre we might expect that a driver will choose a higher speed while crossing. In the first place because for the left turn the driver has to worry not only about traffic from the right but also about traffic from the opposite direction and in the second place because too high a speed will result in an uncomfortable lateral acceleration while turning. We also expect that a driver turning left will look more often at oncoming traffic than when turning to the right.

Presence of other traffic. A third determinant is obviously the presence of other traffic. Depending on the type of intersection and the intended manoeuvre, formal rules (the traffic code) and informal rules (i.e. experience-based rules learned during and after driving lessons) determine how other traffic is negotiated and thus how the basic driver operations are carried out. For example, if a car is approaching an unordered intersection and another car is coming from the right, the driver will have to look to the right more often to find out whether he has to stop or reduce speed so that the other car can pass. There are, thus, consequences for visual orientation, speed control and car handling.

Level of expertise. There is a considerable amount of literature concerning the differences between expert drivers and novice drivers, much of which is devoted to visual orientation. For example, Mourant and Rockwell (1972) have described how expert drivers have different and probably more efficient patterns of eye movements in various traffic situations (For more recent research see also Wierda, Schagen, & Brookhuis, 1990 and Wierda & Aasman, 1991). We therefore might expect to find differences between novices and more advanced drivers in the approach to an intersection.

4.1.3 The relevance of the De Velde Harsenhorst and Lourens data

The correspondence between our research questions and the information in data of De Velde Harsenhorst and Lourens is fairly direct. The recordings from the instrumented car provide information about speed control, the timing and ordering of basic car-device actions, lateral deviation and steering,

eye and head movements and even some information about the cues that a driver might use in timing his actions.

In addition, the data set contains several types of intersections and manoeuvres. We selected three intersections where drivers reduced speed considerably and two intersections for which the speed was reduced to a significantly lesser extent. On the first three intersections the drivers made either a right turn, left turn or a crossing manoeuvre. This selection thus enables a comparison to be made between different types of intersections and manoeuvres.⁵

Unfortunately the De Velde Harsenhorst and Lourens studies contain very few systematic data about the effects of the presence of other traffic participants. Consequently we will not say much about this factor in this chapter. Another factor that will not be covered in this chapter is the level of experience: the group of advanced drivers consisted of three groups with differing levels of experience (in terms of kilometres driven). The differences between the groups proved to be so small, however, that we decided to regard the group as homogeneous.

4.1.4 Using the data

The main goal of our re-analysis was to obtain a description of behaviour at intersections. In addition we hoped to extract the underlying rules that generate this behaviour. In our analysis we followed a GOMS like approach by performing a sequential analysis on the action protocols of drivers approaching intersections (John, 1988). The output of this analysis is (1) a detailed and largely parametrised description of driver behaviour and (2) conjectures about the rules that human drivers might use⁶.

Section 4.2 discusses the original analysis of the material by De Velde Harsenhorst and Lourens (1987, 1988). Though most of this chapter is devoted to our analysis of their data, we will also refer to their original analysis in later chapters⁷. Section 4.3 discusses the parameters that were extracted from their data. Section 4.4 deals with the results of our reanalysis and their relevance for the cognitive model developed in part II. To enhance readability a too detailed treatment of this analysis is avoided; however, in Appendix 2 of this study the interested reader will find a more technical discussion.

4.2 The original De Velde Harsenhorst and Lourens analysis

The De Velde Harsenhorst and Lourens (1987, 1998) study consisted of two parts. In the first part 25 lessons of a novice driver were recorded. Also recorded were her driving test and four driving sessions a year after her test. The driving lessons were no different from ordinary driving lessons apart from the fact that in the last 20 minutes of each lesson a fixed route through a residential area was followed. Both the comments of the instructor and the comments of the driver, who was encouraged to think aloud, were recorded.

The second part of their study used three groups of eight young male drivers. The groups differed in the length of time they had held a driving licence. They too followed the same fixed 20-minute route, after a short training period.

Two analyses were performed on the material recorded. The first analysis focused on the comments the instructor made during the driving lessons. The purpose of this analysis was to describe the development of a novice driver in terms of the feedback she receives from the instructor. The second analysis focused on visual orientation, speed control and course control during the 20-minute routes, both of the novice driver and the three groups of eight young drivers.

4.2.1 Results of the analysis of instructor comments

In the first analysis *all* comments made by the instructor were classified. Table 1 contains the summary of this analysis. Each remark was assigned a value in each of three classifications. The three classifications are shown in the first column of Table 1. The first classification is the type of manoeuvre the remark was about. The second classification is the basic task involved, for example visual orientation or motor control. The third classification is whether the remark was about the manoeuvre as a whole (tactical) or about only a part of a manoeuvre (operational). An additional classification, shown in the first row of the table, refers to the type of remark. Four types were distinguished. The first type is the supportive remark ("yes, try it"), the second type is the correction ("no, that's wrong"), the third type is the verification ("why did you do that?") and finally instructions ("do this to achieve that"). The justification for this classification is given in Lourens and Van der Molen (1986) and falls outside the scope of this study.

A few general conclusions relevant to this study are discussed below.

(1) For this particular instructor 66 percent of all comments were corrections. The relevance for our cognitive driver model is that much of the learning process seems to be trial-and-error learning. Learning from instruction seems to make up only a very small proportion of the total learning time.

(2) The number of comments in the first lessons is nearly two per minute. In the last few lessons this is reduced to one per minute. However, this says more about the instructor than about the driver. A study by Groeger et al. (1990) shows that instructors seem to have a personal rate of instruction frequency. The instructor in the De Velde Harsenhorst and Lourens study falls in the range of the Groeger et al. study. The relevance of this finding for a model of a novice driver is that we know at least the speed with which remarks need to be processed and learned.

(3) Crossing an intersection is a difficult task to learn. The main tasks involved in crossing an intersection (tasks 10, 11 and 12 in the table) account for almost 40 percent of all comments. This figure becomes even higher if we

consider that tasks 2 and 3 (stopping and moving off) are also involved in handling intersections. We regard this as one justification for our decision to focus in this study only the approach to and negotiation of an intersection. (4) If we look at task 1 of the basic tasks in Table 1 we see that visual orientation (looking in the right direction at the right time) is the hardest basic task of all. It generates more support, correction, verification and instruction-related comments than any other task. This result justifies the efforts devoted to low-level perception and visual orientation in the later chapters of this study.

Table 1. Instructor comments during driving lessons. From De Velde Harsenhorst and Lourens (1988). For explanation, see text.

	support	correction	verification	instruction
MANOEUVRE				
1. Preparation	18	11	9	28
2. Moving off	41	194	4	8
3. Driving forward	350	773	82	74
4. Stopping	72	360	10	30
5. Reversing	7	72	3	18
6. Overtaking	13	39	6	3
7. Merging in	36	84	10	18
8. Merging out	9	51	2	0
9. Parking	3	22	0	7
10. Crossing intersection	49	127	10	10
11. Turning right	145	429	21	22
12. Turning left	178	512	36	27
13. Roundabout	21	74	3	10
14. Turning on road	17	69	1	11
BASIC TASKS				
1. Visual orientation	320	717	93	66
2. Speed control	246	600	33	52
3. Course control	187	630	24	56
4. Traffic rules	51	89	27	12
5. Motor control	86	549	18	30
6. Special tasks	69	232	12	50
TASK LEVEL				
1. Tactical	712	2190	124	189
2. Operational	247	627	73	77
TOTAL	959	2817	197	266

Of course, De Velde Harsenhorst and Lourens's analysis contains many more interesting tables and relevant results for our modelling work (for example, the distribution of comments over the driving lessons) but for these we refer to the original reports.

4.2.2 Results of the analysis of the 20-minute sections

The second analysis performed by De Velde Harsenhorst and Lourens on these data is also interesting for our purposes. From the fixed route that all subjects followed they selected 6 intersections. This selection contained roughly two types of intersections. Three intersections for which drivers

reduced their speed by more than 50 percent, and three intersections for which drivers reduced their speed less significantly (by 15 to 30 percent; see Figure 4-2). At the low-speed intersections the drivers made either a right turn, left turn or crossing manoeuvre. At the other intersections drivers only crossed.

The video recordings were used to extract, for each intersection, eye-movement data and speed data. The main conclusions are only summarised here, as they will be more extensively dealt with in the following sections. De Velde Harsenhorst and Lourens's conclusions are the following:

- (1) there is considerable regularity in the visual orientation behaviour of the young drivers, depending on manoeuvre and type of intersection.
- (2) the randomness in the visual orientation behaviour of a novice driver decreases during the lessons and converges towards the behaviour of the young drivers.
- (3) there are clear differences between the intersections in the approach speed and entrance speed at the intersection, but within the groups there are hardly any differences. Thus there is a fairly stable speed-control pattern, depending on the type of intersection.

Relevant aspect for this study is that there are regularities in visual orientation behaviour and in speed control which are dependent on the type of intersection and manoeuvre but independent of the level of expertise. In the following we will thus not look at the differences between the three groups of 8 drivers, but treat them as a single homogeneous group of 24 subjects.

4.3 Reanalysis of De Velde Harsenhorst and Lourens's data

In their analysis De Velde Harsenhorst and Lourens used only a small part of the available data. For example, they did not use the data for the steering-wheel angle, the clutch, brake and accelerator pedals and the gear-stick position. In addition, they evaluated the eye movements and the speed data only in terms of the observed time-to-intersection. For our purposes we also wanted to express the events in other measures. This section discusses what extra information was extracted from their raw data. In addition it provides a more formal background for these results.

4.3.1 Subjects

Our reanalysis will consider only the behaviour of the young drivers. The main reasons for this is that we wanted DRIVER to model reasonable driver behaviour. The data of the novice driver are, certainly in the first half of her driving lessons, far too random and irregular to obtain valid regularities (which in itself is also an interesting result).

In the original De Velde Harsenhorst and Lourens experiments (1987, 1988), three groups of eight young male subjects were recruited as volunteers. The subjects were chosen on the basis of their age and the length of time they had held a driving licence. As we mentioned before, the differences found between these groups were so small that we will consider the subjects as a homogeneous group.

4.3.2 Material and apparatus

The instrumented car was equipped with three video cameras, one directed towards the frontal scene, one directed to the right and one directed at the driver's face. Speed, steering-wheel angle, brake pressure and the pressure on the accelerator and clutch pedals were recorded at a sampling rate of 5 Hz. Sampled data were recorded on the reserved audio tracks of the video tape, thus allowing a synchronous link between video data and all other recordings.

4.3.3 Locations and manoeuvres

De Velde Harsenhorst and Lourens (1987, 1998) originally selected six residential locations in order to determine eye and head movements of drivers and their approach speeds for different types of intersections and manoeuvres. (Note, however, that all intersections are general-rule intersections where traffic from the right always has right of way.) Five of their locations are used in our analyses⁸. Figure 4-1 shows a plan of L1, the intersection where the drivers made a left turn. Figure 4-2 displays the approach speeds for the five intersections we selected. The left-hand bar shows the maximum approach speed before the subjects start to decelerate, the right-hand bar shows the entrance speed at the intersection. It is clear from the figure that there are no significant differences between the maximum approach speeds. There are however large differences between the entrance speeds on the intersection. The differences between the first three intersections and L4 are highly significant, as is the difference between L4 and L6 (L1 vs L4: $F=22.64$ (df 1,46) $p<0.001$, L4 vs L6: $F=55.34$ (df 1,46) $p<0.001$)⁹.

Following Van der Horst's (1990) terminology we will call locations L1, L2 and L3 the *major* intersections, location L6 the *minor* intersection and location L4 the *intermediate* intersection¹⁰.

4.3.4 Pilot procedure

The experimental procedure consisted of two phases. In the first phase a subject familiarised himself with the instrumented car by driving for about twenty minutes. In the second phase the subject followed a fixed route that contained all the locations described above. Directions were given by the investigator in the back of the car.

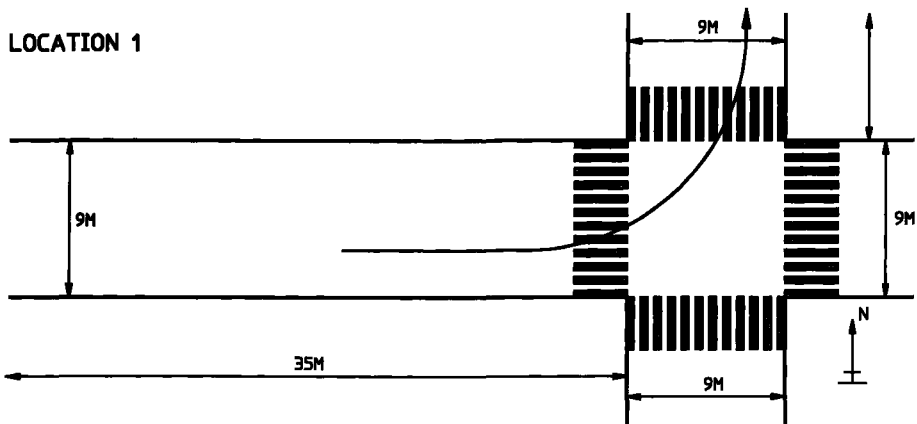


Figure 4-1. This figure shows a plan of one of the intersections, in this case location L1.

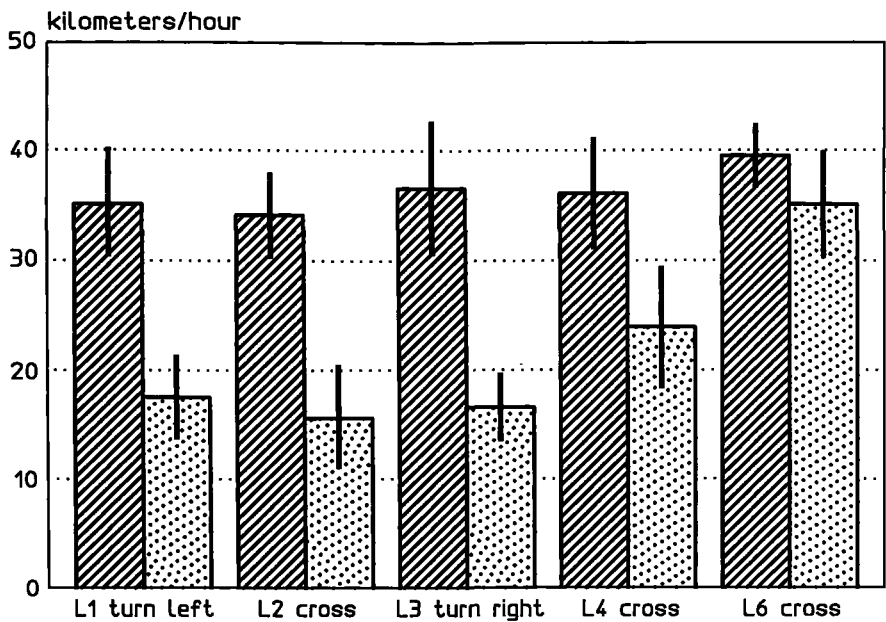


Figure 4-2. The average approach speeds for the selected intersections. The left-hand bars show the maximum approach speed before subjects start to decelerate, the right-hand bars shows the entrance speed at the intersection. The vertical sticks in each bar show the standard deviation. Each bar is based on 24 subjects.

4.3.5 Data analysis

The data analysis was conducted in several passes and resulted in an *events database*. Table 2 lists the events that were extracted from the data. Interested readers who want to know how these events were obtained are referred to Appendix 2. The table starts with two speed-related measures, the speed at the intersection and the maximum speed before the intersection. The following groups show the car-device manipulations. The events with a “max” refer to the maxima that could be detected in the usage profiles of the pedals. The last group in Table 2 shows the visual orientation events. Because these events were obtained from video we distinguish only between the events listed in the table. Because of their importance the first glance to the right and left are included as special events.

To give an idea of the course of the overall approach procedure to the crossing and to provide additional information regarding the interrelationships in Table 2, we show in Figure 4-3 the averaged profile for speed and pedals in a left-turn manoeuvre for our 24 subjects.

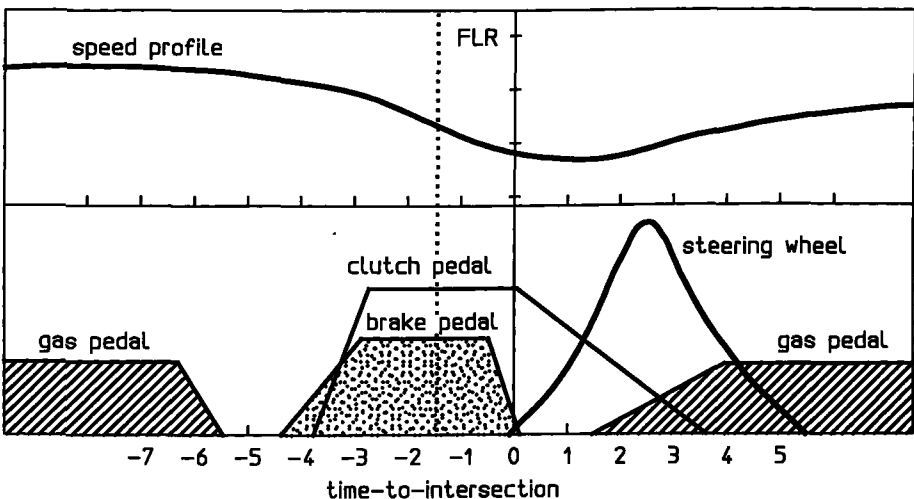


Figure 4-3. The upper part of the picture shows the mean speed profile at location 1, the left turn. The times on the x-axis represent real time to intersection; thus 0 is the entrance of the intersection. The lower part displays the profiles for the accelerator, brake and clutch pedals and the use of the steering wheel. The dotted line around -1.5 seconds to the intersection represents the first glance to the right (FLR). The profiles are averaged over the 24 subjects.

All these events are coded in time, distance and speed-based measures. In addition, for some variables the pedal pressure at that event is listed. The term pressure is slightly misleading here as the numbers (for pressures) in this study do not really indicate the pressure but the distance that the brake, accelerator

or clutch was pressed as a percentage of the total possible distance. The measures are listed in Table 3.

Table 2: Events database

Vmax	Maximum speed before intersection (in the 15-second period)
Vint	Speed at intersection
GasMax1	Release of accelerator before intersection
Gas0	Accelerator completely released before intersection
Gas0*	Accelerator in after intersection
GasMax2	First maximum accelerator pressure after intersection
Brake0	Start of braking manoeuvre before intersection
BrakeMax1	Brake reaches first maximum
BrakeMax2	Start of release
Brake0*	Brake completely released
Clutch0	Clutch pressed for gear-change manoeuvre
ClutchMax1	Clutch reaches first maximum
ClutchMax2	Start of release
Clutch0*	Clutch completely released
Gear	Gear-stick is used
Steer0	Beginning of steering for curve
SteerMax	Maximum steering-wheel angle in curve
Steer0*	End of steering manoeuvre
LL	Looking left
LR	Looking right
LF	Looking forward
LRM	Looking rear mirror
LLM	Looking left mirror
LRS	Looking right shoulder
FL-right	Firstglance to the right
FL-left	First glance to the left.

Table 3 : Time, distance and pressure variables for defined events.

T	real time to intersection at time of event
DTI	distance to intersection at time of event
V	speed at time of event
TTI	time to intersection, computed from DTI/Vat time of event
Acc0	average deceleration from event to intersection
press	the pressure on a pedal.

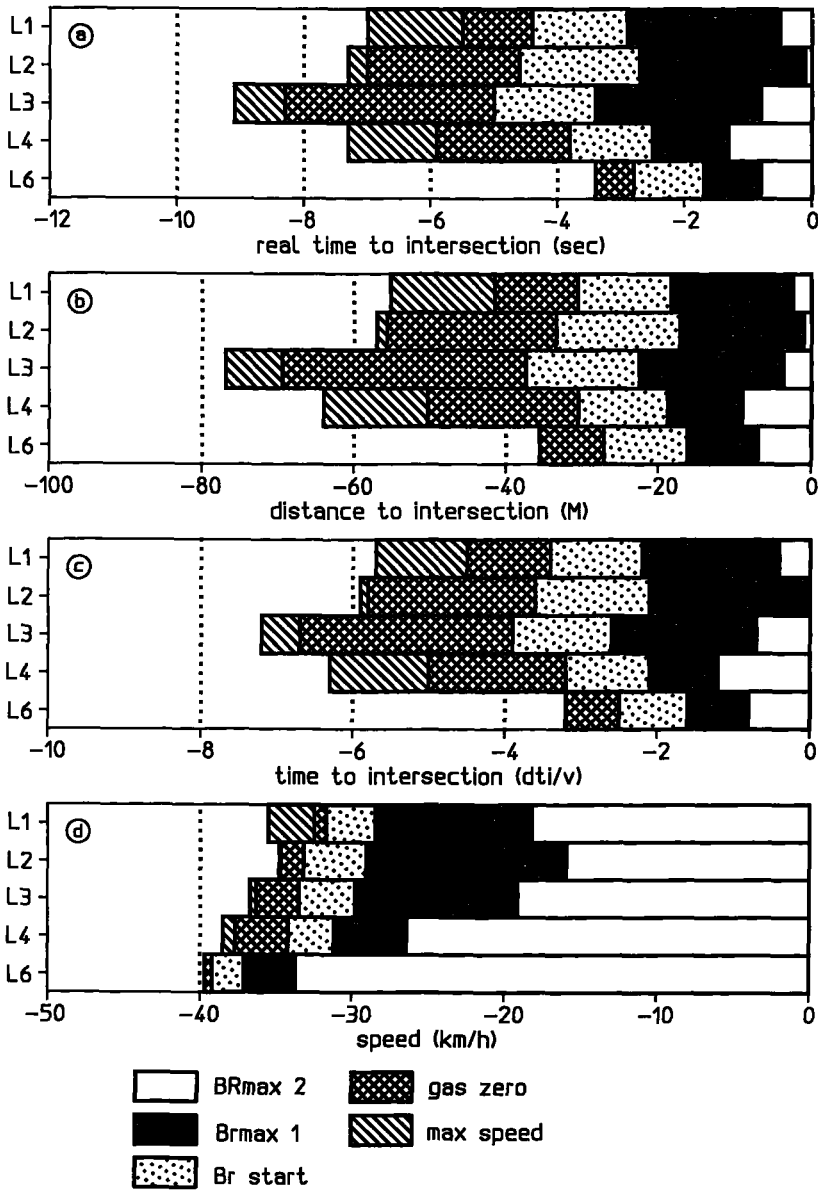


Figure 4-4 shows the main speed-related events in the approach to the intersections L1 to L4 and L6. Figure 4a shows the events in real time to intersection. Let us for example look at location L1: the left-most part of the bar shows the period from maximum speed to the moment when the accelerator is finally released; the next part shows the time that both the accelerator and the brake are completely idle; the third part shows the period from zero to maximum brake pressure; the black part shows the interval that the brake is at its maximum; and the right-most, white part, then shows the release of the brake. Figures (b) to (d) show the same events but then in terms of distance-to-intersection, time-to-intersection and speed respectively.

4.4 Results

In this section we discuss 15 observations that are important for the implementation of the cognitive model to be developed in part II of this study. A number of these regularities are qualitative in the sense that they describe the order and the conditions under which certain actions occur. There are also several regularities in which the parameters can be quantified. An example is the precise moment at which the brake is pressed, depending on the type of crossing. For a number of regularities we will even speculate about the internal rules which human drivers seem use to generate this behaviour.

In this section the full empirical support for these observation is not discussed as we wanted to avoid a too detailed a technical analysis. A detailed technical validation can be found in Appendix 2 of this study.

The observations are arranged in five sub-sections. Section 4.4.1 discusses the main regularities in respect of the speed profile. Section 4.4.2 discusses braking behaviour as one of the most important determinants of this speed profile. Section 4.4.3 discusses all the car-device events, including the speed control events, as a whole. Section 4.4.4 then discusses the most important visual orientation events and, finally, Section 4.4.5 discusses the cues a driver may use in the timing of his behaviour.

4.4.1 The speed profile

All subjects display a slow-down in their speed profile as they approach the intersections. The most relevant aspects of this profile are (1) the maximum approach speed before the intersection, (2) the onset of the deceleration in terms of distance and time to intersection, (3) whether the deceleration is constant or not, (4) the maximum deceleration and (5) the place where the end of the deceleration is reached. The main conclusions that we derived from these profiles are the following:

[1] The maximum speed before an intersection is independent of type of intersection or type of manoeuvre.

Figure 4-4d shows that the maximum speed in the approach lies between 35 and 40 kilometres per hour¹¹. The fact that these differences are small is entirely to be expected, considering that the subjects are driving in a relatively unfamiliar residential area where they cannot know what type the following intersection will be. This maximum speed thus reflects the normal speed for *this* residential area.

[2] At major and intermediate intersections deceleration begins between 7 and 9 seconds before the intersection. At the minor intersection this takes place only shortly before the crossing.

Figure 4-4a shows that at major intersections deceleration starts at a distance of as much as 9 seconds or 75 metres (see 4b) before the intersection and at the minor intersection at a distance of as little as only 3 seconds or 35 metres. The relevance for our modelling efforts is that human drivers seem to begin the entire approach process sometimes as much as 75 metres before the crossing. By starting so early drivers allow themselves a long time, sometimes as much as 9 seconds, to complete the crossing.

[3] The entrance speeds at major intersections are not determined by the type of manoeuvre.

At the three major intersections subjects chose the same entrance speed, about 17 km/h, irrespective of the manoeuvre to be performed¹² (see Figures 4-2 and 4-4). In theory it could be expected that there would be a difference between a crossing manoeuvre and the two turn manoeuvres, since with a crossing manoeuvre it is only necessary to consider the *risk factor* 'traffic from left'. With a turn-left manoeuvre the driver also has to take into account the *comfort factor*. Taking the bend too quickly is both unsafe and uncomfortable. Given the results, we cannot deduce that either the risk factor (conflicting traffic) or the comfort factor (while turning) is the more important one. It therefore looks highly probable that drivers have a standard speed control strategy for approaching an intersection, independent of the manoeuvre to be performed. It seems that when recognizing a major intersection human drivers aim for a fixed desired speed at the entrance of the intersection, thereby reducing the number of decisions to be made.

[4] We find nearly constant decelerations for minor (-1.0 m/s^2) and intermediate and major (-0.7 m/s^2) intersections. There are no differences between manoeuvres.

In Section 4.2 we shall see that the drivers maintain a constant brake pressure after the initial brake maximum has been reached. One of the consequences of this is that we also find reasonably constant decelerations in the speed profiles. There are two reasons why the profile is not 100 percent constant. First, there is a brief period during which the accelerator has been released but the brake has not yet been pressed and, secondly, it takes a while before the brake pressure reaches its maximum.

4.4.2 The brake profile

The speed profile in the approach to an intersection is largely determined by the use of the brake. This section discusses a number of observations with respect to the use of the brake. We will deal with the relevance of these observations at the end of this section.

[5] The brake profile consists of three clearly distinguishable actions. Phase 1 is pressing the brake at a fixed speed until a maximum is reached. Phase 2 is a period during which the brake is kept at a fixed maximum and phase 3 is the release of the brake at a fixed speed.

Figure 4-3 provides a sample profile for a left turn at location 1. In the analysed material we found a first and a second maximum for all 24 subjects. We also found that for all the subjects that both pressing and releasing the brake are linear actions.

If we compare the profiles for the brake, the clutch and the accelerator pedal, we find the tightest profiles for the brake (see the Appendix 2 for details of how these maxima were determined).

[6] The brake pressure is kept constant between the first and the second brake maximum.

Observation [6] is actually part of observation [5], but since it is so important we would like to emphasise again that for all subjects the brake pressure between the first and second maximum was kept constant.

[7] At the moment of the first brake maximum (brakemax1) we find no correlation between brake pressure and speed, distance-to-intersection or time-to-intersection. The entrance speed at the intersection is (therefore) achieved by timing braking actions and not by varying brake pressure.

Given a certain period of a fixed brake pressure there are two possibilities for controlling the deceleration before a crossing, namely (1) selecting the first brake maximum and (2) varying the timing of the first and second brake maximum. The first option can be eliminated, however, since we did not find any significant correlation between brake pressure and speed, distance-to-intersection or time-to-intersection (see also Appendix 2). We therefore deduce from this that the entrance speed at the intersection is achieved by timing the braking actions and not by varying the first brake maximum or the brake pressure between maxima.

[8] Surprisingly, the distance to the intersection at the moment of the first brake maximum is independent of the type of manoeuvre or the type of intersection.

Figure 4-4 shows considerable variation in the moment at which subjects start to release the accelerator pedal for the different locations. However, this variation is reduced for the moment at which subjects start to brake and has almost disappeared when subjects reach their first brake maximum (see also Figure 4-5). For all intersections this maximum lies between 16 and 22 metres before the intersection and we find no significant differences between the locations for *distance-to-intersection* (DTI). In Section 4.5, however, we will see that the *time-to-intersection* (TTI) at the moment of braking does show large differences between the major, intermediate and minor intersections.

The relevance of observations [5] to [8] for our cognitive model is as follows: initially we see that human drivers select a very simple braking strategy with only linear components (fixed speed for pressing and releasing and a constant brake pressure between maxima). In itself it is not so surprising that a fixed

brake pressure should be selected, since this ensures a reasonably constant deceleration factor. If a non-constant brake pressure were selected, there would also be a non-constant deceleration factor and higher-order tracking problems would therefore arise (for visual acceleration problems see Regen, Kaufman & Lincoln, 1986; for higher-order tracking problems see Wickens, 1986).

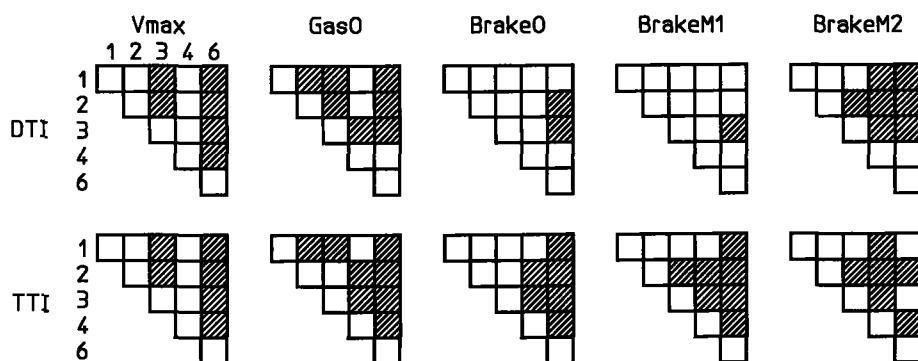


Figure 4-5. This figure contrasts the five selected locations for DTI and TTI. The numbers refer to the location numbers. A hatched area indicates that the difference between the two locations indicated is significant ($P < 0.05$). Note that the first brake maximum seems to occur at the same DTI for all intersections.

The second thing we see is that speed control is also regulated very simply: irrespective of the type of manoeuvre or type of crossing, braking starts at about the same point. Differences in entrance speed at the intersection arise by varying the brake maximum or the moment at which the brake is released again.

It will be clear that it must be possible to translate the observations obtained here into a number of rules that can generate this behaviour. An overly simple translation could be the following.

```

If DTI=20 and minor_intersection then
    press brake with constant speed till brake pressure = 70
If DTI=20 and major_intersection then
    press brake with constant speed till brake pressure = 50
If DTI=5 and minor_intersection then
    release brake with constant speed till brake pressure = 0
If DTI=10 and major_intersection then
    release brake with constant speed till brake pressure = 0
  
```

It should be noted, incidentally, that these rules only describe the surface structure of speed behaviour. In the later chapters we will see that the underlying rules in our cognitive model are considerably more complex. We will return to this fundamental issue in later chapters.

4.4.3 Manipulation of other car-control devices

We were also interested in the use of the clutch and gas pedal because of their relation to speed-control. In addition, we are interested in how all the car-device actions are integrated.

[9] In the approach to the major intersection, all drivers release the accelerator and use their brakes. In addition, more than 2/3 of the subjects change to a lower gear.

[10] As with the brake profile, the clutch profile consists of three distinguishable actions. A phase of pressing down with a constant speed, a phase of constant pressure and then a phase of releasing.

For most people the phase of releasing is found not to occur with a constant speed. In fact this phase should be broken down further into two sub-phases. The first sub-phase is letting the clutch come up until the moment when the clutch plates make contact; the second sub-phase is then the period when the clutch pedal comes up entirely in coordination with the accelerator pedal being pressed.

It seems again that, as in the brake profile, behaviour can be generated by a number of simple rules.

[11] The ordering of actions (but not the timing) is fixed for manoeuvres and types of intersection.

Figure 4-6 displays most of the events discussed thus far. The x-axis lists the abbreviated names of the events. The y-axis shows the real time to intersection. The interesting observation is that, with the exception of the steering movements in the right-turn manoeuvre in L3, all the lines show an unvaryingly increasing function. This means, therefore, that the order of actions for all of these locations is the same, while the timing may differ. We show in appendix 2 that this pattern also applies at the individual level.

In itself, of course, this is not at all surprising, since for most drivers braking before a crossing will be an automatic action. There appears to be no reason whatsoever why braking and changing down should be done differently for a left-turn manoeuvre than for a right-turn manoeuvre. What is surprising, however, is that the first look to the right always falls between the first and second brake maximum (see Section 4.4).

The divergence in L3 for the steering behaviour can be explained by the fact that steering behaviour is in fact separate from braking behaviour and that the timing for starting the steering movement will of course be different for a left-turn and for a right-turn manoeuvre.

The relevance for our cognitive model is that drivers follow a *fixed sequence of car-device actions*. It seems that human drivers learn a *fixed motor program* (in the sense of, for example, Jordan & Rosenbaum, 1989) in which the overall structure has been determined, but in which the timing of the individual

actions is determined by local conditions (For a similar conclusion concerning the serial order in speech behaviour, see Wickelgren (1969)).

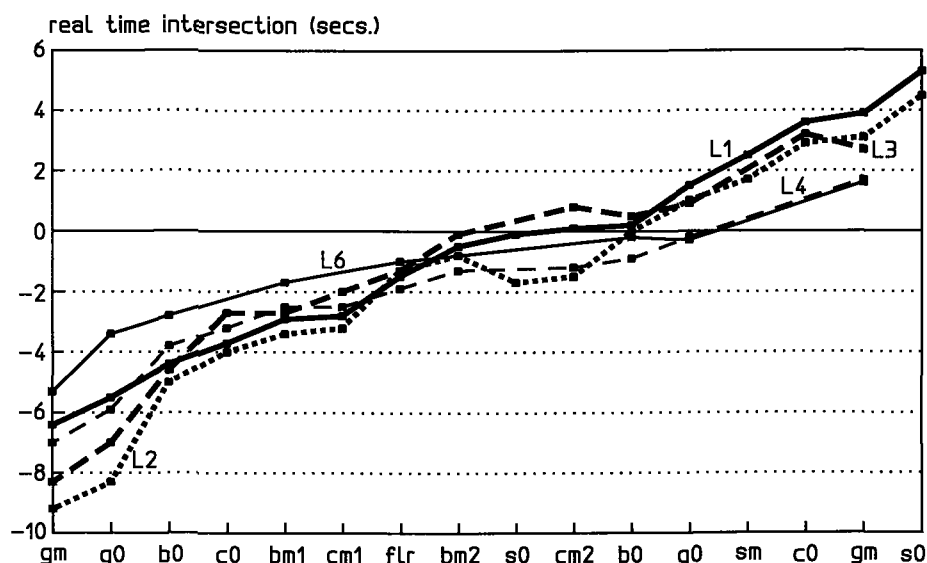


Figure 4-6. The y-axis shows the real time to intersection. The x-axis shows observed events. Negative numbers refer to the period before the intersection. The accelerator events before the intersection are accelerator maximum (gm) and release of accelerator (g0). The brake is represented by the start (b0), first maximum (bm1), second maximum (bm2), and the final release (b0). The clutch is represented by the start (c0), first maximum (cm1), second maximum (cm2), and the final release (c0); and the steering wheel is represented by start of turning (s0), maximum angle (sm) and return of wheel to original position (s0). flr stands for first look to the right.

4.4.4 Visual orientation

When does a driver look in which direction and how is this determined by type of intersection and manoeuvre? This is the question discussed in this section. Unfortunately, given the way the looking directions were obtained (from video), we can distinguish only very roughly between glances to the right, left, and front and into at mirrors and glances over the shoulder.

[12] Depending on the type of intersection and type of manoeuvre, we find clearly distinguishable visual orientation strategies.

To get an idea of the visual orientation pattern and the (relative) consistency between subjects, see Figure 4-7, in which a left-turn manoeuvre is shown. At first sight the figure may look rather messy. Upon closer examination, however, a number of regularities can be found. In the first place we see that almost all subjects first look right and then left. In the second place almost all subjects show a glance to the right between 1 and 2 seconds before the crossing. In the third place we see that all subjects briefly glance straight ahead

before turning left. Table 4 summarises the patterns for the different intersections and manoeuvres. A detailed validation can be found in Appendix 2.

Table 4. Summary of visual orientation strategies

L1 Left turn at major intersection.	First short look to the right, then longer look to the left.
L2 Cross at major intersection	Half of the subjects first look left, then right.
L3 Right turn at major intersection	Short look to the left, then longer look to the right.
L4 Cross at minor intersection	All subjects look to the right. 7 out of 23 first look to the left
L6 Cross at T-junction	Only look to the right.

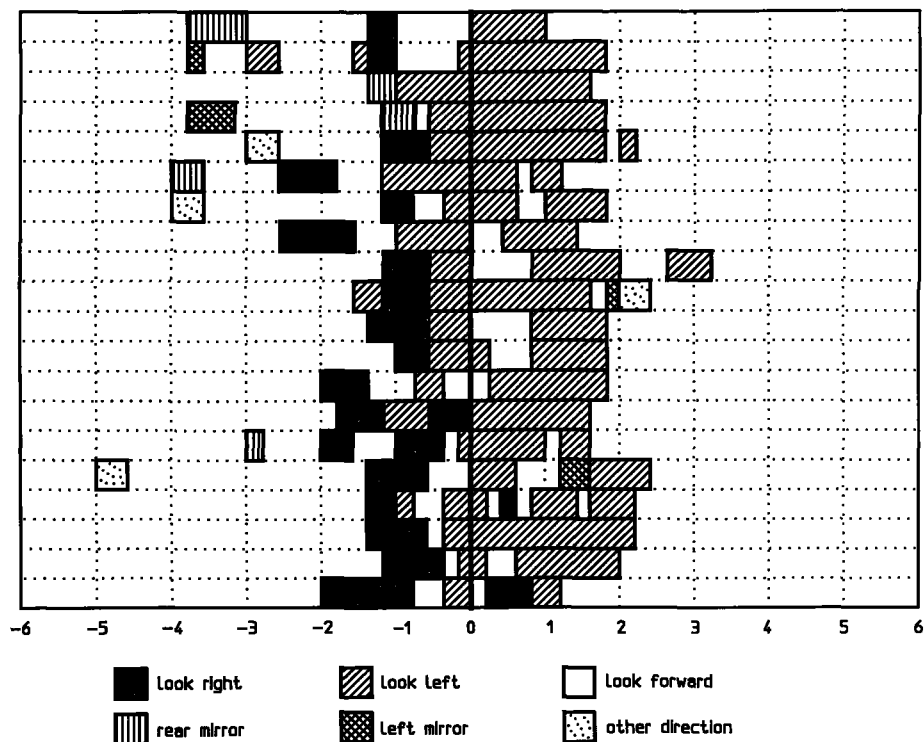


Figure 4-7. The figure shows the main looking directions for a left-turn manoeuvre at intersection L1. The x-axis shows time in real time to intersection. Negative times represent the approach.

Of course, there is more to a strategy than simply the ordering of actions. Another important regularity is the following:

[13] First look to the right very constant for intersections and manoeuvres. First look to the left clearly less important.

The first look to the right (Fl-right) proves to be very constant for intersections and manoeuvres (T between 1.0 and 1.9 seconds, DTI between 7.4 and 13 metres and TTI between 0.9 and 1.7 seconds before the intersection). If we look at Figure 4-4 we see that of all the observed actions the difference between manoeuvres is here visibly the smallest. In contrast to the first look to the right, the first look to the left is very dependent on manoeuvre or type of intersection.

[14] First look to the right almost invariably falls between first and second brake maximum.

Another regularity that can be derived from Figure 4-6 is that subjects always look right between the first and the second brake maximum. One might argue that Figure 4-6 shows only averages. However, the data were very consistent. For example, in a left-turn manoeuvre only one subject looked to the right after releasing the brake.

[15] First look to the right is probably connected with time-to-stop at major intersections

Another interesting fact is that at the first look to the right, the deceleration required to come to a halt before the intersection is -3 m/s^2 for major intersections (including L4). The speed at the Fl-right for L6 is so high that even a deceleration of -6 m/s^2 , a maximum for most cars, would result in a stopping distance that is greater than the distance to intersection at Fl-right. What we see here, then, is that when approaching a major intersection drivers make sure that they can come to a stop if a car from the right is seen. [When approaching a minor intersection drivers have their foot on the brake and probably hope that any car coming from the right will stop.]

The relevance of these regularities for our cognitive model is this: first, the order in which drivers look to the right, forward or left is determined by the manoeuvre or type of intersection; however, the first look to the right serves as a kind of pivot. Second, drivers always look to the right at between 8 and 13 metres, making sure (at least for major intersections) that it is possible to stop if a car is seen from the right.

4.4.5 Cues used

What are the cues a car driver uses to time his actions in the approach to an intersection? Is it the distance to intersection or is it a combination of distance, speed and possibly acceleration?

Van der Horst (1990) has tackled this issue by looking at the onset of braking for different types of intersections and railway crossings. He found that if the moment of braking is expressed in *time-to-intersection* the most consistent differences are seen between the various types of crossing. His results match ours as regards the point of braking (TTI = 3 to 4 seconds for major intersections). Van der Horst's time-to-intersection is the same measure that we use in this chapter, namely the distance-to-intersection divided by speed. Adding

an acceleration factor to a time-to-intersection measure is not found to yield anything extra regarding the prediction of the moment of braking. One of the things that Van der Horst's study also showed was that it appears that people can deduce TTI directly from the visual flow field.

We shall see below what we can say on the basis of our material regarding the cues used for timing the actions.

[16] It is not possible to conclude that time-to-intersection is a more important cue than distance-to-intersection.

We saw in observation [8] above that, surprisingly, the distance to the intersection at the moment of the first brake maximum is independent of the type of manoeuvre or the type of intersection. Figures 4-4b and 4-4c showed no large differences for the onset of braking or first brake maximum, either for DTI or TTI. However, if we use a simple T-test to compare all locations with one another, we see in Figure 4-5 that for the onset of braking (Brake0) and for the first brake maximum (BrakeMax1) there are scarcely any differences for DTI, whereas there are a number of differences for TTI. What does this mean? Initially one would think that DTI is the cue that determines when people start to brake. On the other hand one might also say that by correcting distance for speed (i.e. the definition of TTI) one would find consistent differences between locations, which would then indicate that TTI could be a cue.

A question to be asked is therefore: can we see in the individual data that people are able to correct distance for speed?

[17] Subjects are able to correct distance for speed.

Figure 4-8 is rather unusual in that correlations are compared in a bar chart. The figure shows the correlation between speed and DTI for the moment the accelerator is released, the start of braking and the first brake maximum. The figure shows a very high correlation at the major intersections (L123) and L4 for the brake events. Thus, if a driver approaches the intersection faster, he will release the accelerator and brake earlier. Figure 4-9 shows an overall scattergram for locations 1,2,3 and 4 at the start of braking.

The fact that L6 shows such a low correlation between DTI and speed, while the DTI at the moment of braking resembles that of the other locations, might again be a factor in favour of DTI.

The question what the cues are that our cognitive model should use in timing behaviour remains unanswered for the time being. Given the arguments in [6] and [14] DTI looks attractively simple. However, there are two arguments in favour of TTI. First, we *did* find that subjects correct distance for speed and, second, Van der Horst (1990) makes a plausible case for subjects being able to perceive TTI information directly.

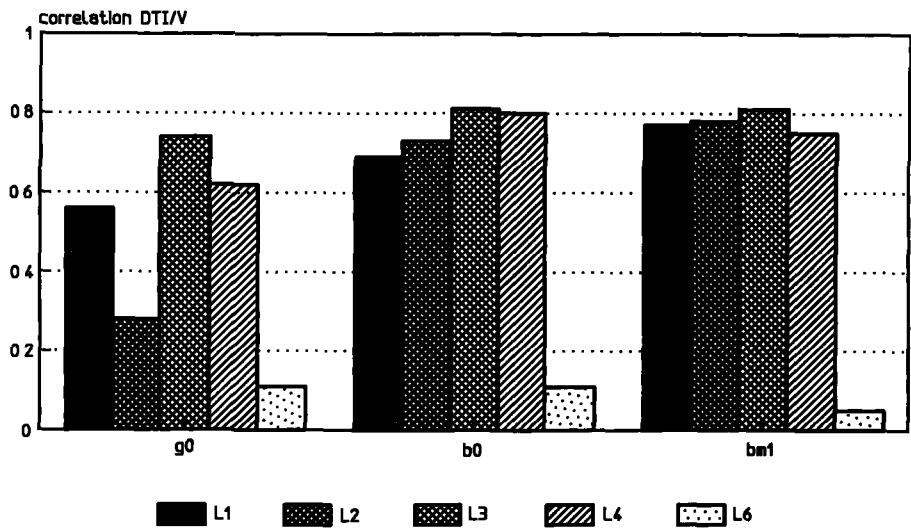


Figure 4-8. The correlation between distance-to-intersection and speed at the moment the accelerator is released (a0), at the start of braking (b0) and at the moment of the first maximum (bm1).

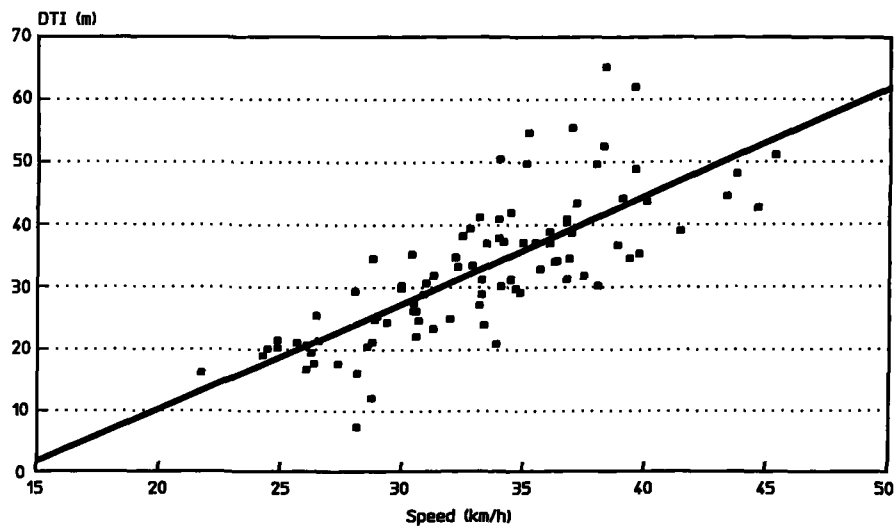


Figure 4-9. A scattergram that shows the correlation between distance-to-intersection and speed at the onset of braking for locations 1,2,3,4.

4.5 Concluding remark

The relevance of various regularities was argued earlier. The question remains whether these regularities fully cover our questions relating to the cognitive model discussed in the remainder of this study. The answer to this is a simple “no”. What we are short of is, for example, accurate eye-movement recordings, but we have also got only indirect data about current motor behaviour: we would for example like to have more data about foot and hand movement, data on moving the hands from steering wheel to gear-stick and vice versa. Nevertheless the material does give us a description that is good enough to enable our cognitive model to be tested. Chapter 14 will show how these data are used.

4.6 Notes

¹An example of another, for this analysis less important manipulator, is signalling.

²Hills (1980) estimates that 90 percent of all perceptual input that is used in driving is visual.

³Although certain directions are of course more likely to contain interesting objects than others.

⁴We will not deal here with why drivers reduce speed when approaching intersections (although we know that the risk factor and the comfort factor play a part when turning left or right). We will also not go into how a driver recognizes an intersection with a high traffic volume.

⁵One question is whether there are other interesting data sets that we might have used. The answer is that the De Velde Harsenhorst and Lourens data set is, to our knowledge, unique in that it provides an integrated recording of the behaviour of drivers approaching and negotiating intersections. Of course there are numerous studies that look at visual orientation in the approach to intersections (Miura, 1986; Wierda, Van Schagen and Brookhuis, 1990) or speed control at intersections (Van der Horst, 1990), but none have looked in such detail at a combination of so many driver operations

⁶This analysis resembles also a protocol analysis of the material available to us. Normally a protocol analysis is carried out on (1) the verbal statements of a subject when performing a complex task and (2) the task behaviour displayed, including reaction times and errors. The output from a protocol analysis may be a computational model in the form of a set of simple rules (production system) that can reproduce the behaviour. A problem, however, in applying classical protocol analysis methods (see for example Ericsson and Simon, 1993) is that our action protocols of drivers approaching an intersection are non-verbal. Moreover the task is a very dynamic one with many dependencies on perceptual and motor systems. Nevertheless we tried to ascertain the underlying rules of intersection behaviour along the lines of a protocol analysis.

⁷Another reason for briefly reproducing their work here is that it has never appeared in English.

⁸Location 5 in their analysis is not a simple intersection in that the crossing road is divided into two lanes separated by a soft central reservation. Because of this extra complexity we decided to omit this location from the analyses.

⁹Although we will not go into the reasons for the differences between entrance speeds, we are inclined to guess that the width of the road is an important factor. Another factor might be the reduced visibility at some intersections, were it not for the fact that visibility was invariably terrible at all intersections.

¹⁰Although there are to our knowledge no formal definitions of the concepts *major* and *minor* intersections, these concepts are generally used to denote intersections for which drivers do or do

not reduce speed. We use the concepts because they enhance the readability of the results considerably.

¹¹ The approach in location 6 is significantly faster but it is only a small difference.

¹² It is of course true that we selected the crossings with speed in mind. Nevertheless it is still interesting to see that there are no differences for the various manoeuvres.

5 Introduction to Part II

5.1 Introduction

The model presented in Chapter 2 focused on the problem of multiple goals and multitasking within driving and Soar. It was demonstrated that task-switching and task interruption, automatic and controlled processing, and bottom-up and top-down processing all come fairly naturally to Soar. However, it was also clear that this first attempt ran short of achieving our original goals. To remedy the deficiencies noted for that model, DRIVER, the cognitive model to be developed in this part, adds (a) a small set of constraints for the motor system so that we can model more realistically the use of the arms, legs, eyes and head in vehicle control and visual orientation, (b) a set of constraints for the perceptual system that enables us to model both covert attention and eye and head movements, and (c) finally, a more efficient set of default rules replacing Soar's native rules so that task-switching and interrupts have a far less devastating effect on the continuity of cognitive tasks such as navigation. In addition, with DRIVER we have tried to adhere to the empirical data discussed in Chapter 4.

5.1.1 An overview of DRIVER

The chapters in Part II of this study describe different driving tasks and aspects in relative isolation. In order to help the reader to maintain a good overview of DRIVER as a whole, the following is a summary of the model.

DRIVER competently manoeuvres through a residential area in search of its destination. This area consists of a network of two-lane roads and intersections. At the side of the road we find traffic signs, houses, parked cars, and trees. DRIVER is not alone on the road but is required to negotiate other car drivers and bicyclists. These other traffic participants act intelligently; they display 'natural' speed and course control both in response to each other and in response to DRIVER. DRIVER'S main driving tasks are the same as those in the first model: speed control, lane keeping and navigation.

DRIVER's first task is *speed control*, that is, the task to choose the optimal speed in all conceivable traffic situations. This requires that DRIVER continuously monitor its speed and the relevant features in the traffic environment. Relevant features include for example other traffic participants, traffic signs, and the topography of the approaching intersection. Changes in speed are achieved by manipulation of the car engine, that is by manipulating brakes, clutch pedal, accelerator pedal and gear-stick. DRIVER's second task is *steering and lane keeping*. Whenever DRIVER perceives that it is too far off course it can influence the car's heading angle by manipulating the steering wheel. DRIVER's third task is *navigation* or the task to find a pre-set destination in the residential area. As DRIVER knows this residential area, planning the trip is relatively easy. However, to its surprise DRIVER often finds that some intersections on its route are blocked, forcing it to replan its route while driving.

The above tasks are specific to driving. The following discusses a second set of tasks, namely the planning and execution of visual and motor actions. These tasks are not specific to driving, though they are the basic building blocks upon which the main driving tasks rest.

Motor planning and execution: DRIVER changes the speed of its car by manipulating the car engine, or rather by manipulating the gear-stick, clutch, accelerator and brake pedal. In addition, DRIVER controls the car's heading angle by turning the steering wheel. Manipulation of the in-car devices is achieved by planning and actual execution of motor plans. DRIVER as a beginner spends a considerable amount of time in figuring out (i.e. planning) which body commands to give and in what order so that gear changes are made without protest from the clutch plates. DRIVER at the medium stage of experience knows which commands to give at what time, but still has to learn how to fine-tune the brake, the accelerator or the combination of clutch and accelerator pedal to achieve the desired changes in speed.

Visual orientation: DRIVER's perceptual apparatus is, like that of humans, seriously hampered by several constraints. The first constraint is the restricted width of the *functional visual field* (FVF). Objects that fall within this field can be attended to without eye movements and have a near 100 % chance of being detected. Objects positioned outside the functional visual field are said to be in the *peripheral visual field* (PVF) and have a far lower chance of being detected.

The second constraint is that eye and head movements are time-consuming processes. Due to the restricted functional field DRIVER sees only a small part of the relevant traffic environment and is thus forced to move its head and eyes continuously to bring objects out of the PVF and into the FVF. These constraints, the limited size of the functional field, the low quality of information in the periphery and the temporal constraints, force DRIVER to plan and

execute its visual orientation plans carefully. Note that visual orientation also includes the internal attention mechanism within the functional field.

5.2 Implementation notes

One main goal of this study is to evaluate the suitability of Soar for implementing complex dynamic tasks. This evaluation will take place in the final chapter. All the other chapters in part II are dedicated to the implementation of the various subtasks and subparts of DRIVER. Another goal of this study was to build a cognitive driver model with a reasonable degree of psychological validity. We tried to achieve this by taking into account four different types of constraints. Those constraints were discussed in the previous chapters but we summarise them here.

1 Soar constraints. In a sense this is a trivial constraint. By using the Soar architecture we automatically build a model in the Newell and Simon tradition. In the following chapters we will see many times how the Soar architecture determines the shape of the driver model. We will also have to conclude that some aspects cannot be implemented because of the Soar architecture.

2 UTC constraints. We stressed that in this study our most important constraint is to comply with the requirements of the theory of immediate behaviour in our modelling efforts. This means that in our model (a) input and output take place in the base level space (BLS) and (b) all attend and intend operators function in the BLS. Note how most of the problems that we will encounter in the following chapters are a result of this constraint.

3 Basic constraints for perception and motor control. We argued how important it is to include the constraints for perception and motor control. Newell's UTC does provide some constraints but is in general underconstrained with respect to both perception and motor control. We were thus forced to include additional constraints. However, it is clear that one can not even start to include all the facts and regularities of *The Handbook of Perception and Human Performance* (Boff, Kaufman & Thomas, 1986) within DRIVER. In the following chapters we will try to choose (and justify) only those constraints that we consider to be the most important for a functional model of driver behaviour.

4 Use of empirical data. Our final guideline in building the model is to keep a fit between the behaviour of DRIVER and the observed behaviour of the young drivers described in chapter 4.

A final note: we will see that DRIVER is currently more a modelling tool or a shell than a complete cognitive model of driver behaviour. In a sense DRIVER is little more than Soar specialised to model driver behaviour¹. The reader should keep the tool-oriented nature in mind when reading the following chapters. Each of these chapters describes an aspect of driving that in itself would justify a dissertation. One can therefore scarcely expect every aspect to be covered down to the last detail in DRIVER. This is not an excuse for provid-

ing only sloppy implementations of single aspects, but points to the real value of DRIVER, which lies in the integration of multiple driver tasks and the uncovering of fundamental issues in modelling complex dynamic tasks. DRIVER is thus a model under construction². Most of the tasks discussed in the following chapters are an integral part of DRIVER, but some of the tasks do not yet fit in seamlessly. The reader will be notified when this is the case.

5.3 Layout of the remainder of this study

The model covers many themes and some of these cannot be treated independently of other themes. However, a lucid presentation of the model requires segmentation. It was impossible to find an arrangement of sections that avoided forward referencing entirely. Figure 5-1 displays the road map that indicates how chapters fit together.

Chapter 6 - DRIVER's small world. This chapter presents WORLD, the simulated traffic world inhabited by DRIVER. This chapter discusses (a) the representations and control structure used in the implementation of WORLD and (b) the rules that generate the behaviour of the semi-intelligent agents with which DRIVER interacts.

Chapter 7 - Basic motor control. A few simple constraints of the human motor system were added to Soar so that we could model the movement times of the extremities and of the eye and head movements. Motor control in DRIVER is divided into a high-level motor command language and lower-level motor module. This chapter describes the latter. Its basic function is to execute motor commands (issued from Soar's working memory) asynchronously from internal Soar processes.

Chapter 8 - Motor planning and vehicle control. While Chapter 7 discusses motor control mechanisms without regard to driver behaviour, this chapter discusses the uses of these mechanisms in driving. Using gear-changing as the example we see how DRIVER as a beginner is heavily involved in learning motor plans that will get the car into the right gear. After DRIVER has learned these plans, most of its learning time is then dedicated to fine-tuning its movements.

Chapter 9 - Basic perception. Adaptive visual orientation – looking in the right direction at the right time – is probably the most difficult task in driving. This chapter discusses the constraints of human perception that we had to include in Soar in order to model visual orientation. The chapter describes the implementation of the lower level perceptual module for object recognition, attention, eye and head movement control, and the relation to the lower level motor module.

Chapter 10 - Visual orientation. While the previous chapter described DRIVER's perception without regard to traffic, this chapter describes a mapping of human visual orientation onto DRIVER. We will see a) how the environment largely constrains the actual behaviour and b) how different strategies are relatively easily induced by only a few Soar productions.

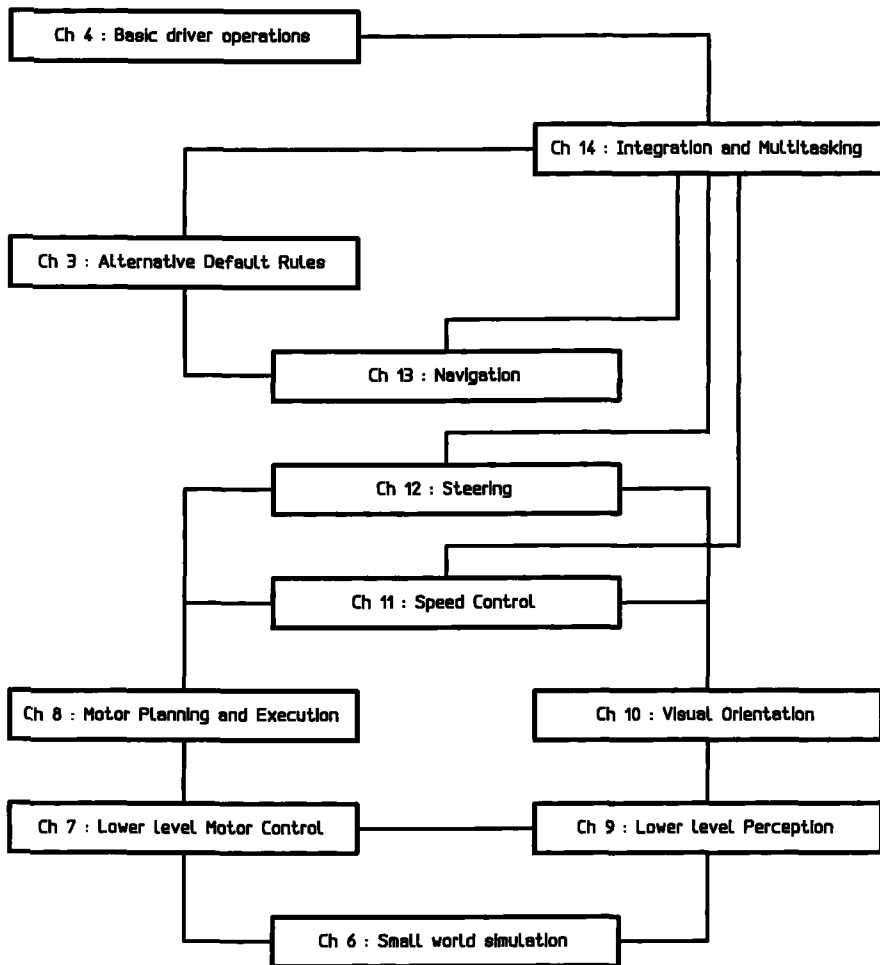


Figure 5-1. Structure of part II

Chapter 11 - Speed control. This chapter describes how perceptual and motor control mechanisms are integrated in the speed control task (see Figure 5-1).

Chapter 12 - Steering and lane keeping. As with almost all the other subjects handled in these chapters, complete coverage of steering and lane keeping would require a dissertation in itself. Steering and lane keeping in DRIVER has been kept fairly simple, though it is realistic enough to require a reasonable amount of effort (in terms of motor control and eye movement behaviour) to keep the car on the road. The interesting part of this chapter is how both open-loop and closed-loop steering are modelled in Soar.

Chapter 13 - Navigation in DRIVER. Navigation is one of DRIVER's main internal tasks that requires a considerable amount of searching. DRIVER uses an

alternative set of default search rules (described in Chapter 3) that allows for deliberate reversal and deliberate rejection and thereby provides both a form of progressive deepening and a simple form of error recovery. The interesting part of this chapter is how DRIVER follows a learned plan, finds that an intersection has been blocked and is forced to replan on the fly.

Chapter 14 - Integration and multitasking. In Figure 5-1 we see how the topics discussed in part I and part II finally come together in chapter 14 which discusses the integration of all the driving subtasks. Soar traces are provided of DRIVER at work, using the machinery discussed so far. We will relate the empirical data in Chapter 4 to these traces. The traces are also input for a discussion on the multitasking mechanisms in DRIVER.

Chapter 15 - Discussion. This chapter finally evaluates the main goals of this study. How suitable is Soar to model complex dynamic behaviour and how good is DRIVER as a model of driver behaviour? The chapter concludes with a list of future research items.

Appendix 1. Learning and error recovery. There are basically two varieties of learning in Soar and DRIVER. Variety one, learning in a closed world (that is, learning without input from the external world,) is the normal mode for Soar and is easily achieved by Soar's native chunking mechanism. Learning in interaction with the external world is feasible but difficult with Soar's chunking mechanism. Another complicated problem is error recovery based on external input. This appendix discusses possible solutions to these problems in the context of Soar and DRIVER.

5.4 Notes

¹ The essential difference between Soar and DRIVER is that we have given driver hands, feet, eyes and a head, we have implemented some important constraints from the literature on perception and motor control (in traffic), we have built a simulated traffic world around Soar and we have built in some basic driving knowledge so that she could start driving. It is up to the modeller to provide additional constraints, modify existing constraints or even add other ways of learning, etc.

² Interested readers can obtain the source code for the implementation of driver that is described in this study.

6 DRIVER's small world

Summary: this chapter presents WORLD, the simulated traffic world that DRIVER lives in. WORLD is filled with semi-intelligent agents that drive or ride a cycle on the roads and intersections of WORLD. This chapter discusses (a) the representations and control structure used in the implementation of WORLD and (b) the rules that generate the behaviour of the semi-intelligent agents.

6.1 Introduction

The general idea of simulated traffic environments and their usage as a test-bed for intelligent architectures was described in Chapters 1 and 2. Relatively detailed implementations of these environments can be found in Aasman (1986, 1988), Reece and Shafer (1988), Wierda and Aasman (1988), Reece (1992) and Van Winsum and Van Wolffelaar (1993). This chapter describes in somewhat more detail the *implementation* of WORLD, the simulated traffic environment DRIVER lives in. We have included a relatively detailed description in this study for several reasons. First, we want to give an impression of this world and the behaviour of the semi-intelligent agents that DRIVER will encounter. Second, the representations of the objects and roads are used by both the semi-intelligent agents and by DRIVER's perceptual mechanisms. Third, we want to show how the relatively complex behaviour of the semi-intelligent agents is generated by a relatively small set of rules.

It should be noted that, in contrast to DRIVER, the descriptions of the rules and the behaviour of the semi-intelligent agents are expressly not intended as a cognitive model of driving behaviour. At the end of this chapter we will discuss why. This chapter describes primarily the technical implementation of WORLD and the semi-intelligent agents that drive in it. Before embarking on this, however, we shall deal with the matter of how our choices were determined.

6.1.1 Constraints on DRIVER that shape WORLD

It seems clear that the required degree of complexity of an autonomous agent such as DRIVER is determined by the complexity of its environment. Conversely it is also true that the required complexity of the world is given by the desired functionality of the autonomous agent. In retrospect, the constraints on DRIVER were the determining factor in the design of WORLD. The list of constraints on DRIVER that largely shaped the design of WORLD basically stems from our objective of modelling visual orientation strategies and device-handling in the context of the main driver tasks: speed control, course control and navigation.

Constraints arising from visual orientation strategies. Given the goal to model visual orientation strategies in drivers, WORLD provides things to look at besides cars and bicycles. Most of the time human drivers look at irrelevant objects (Wierda, Van Schagen and Brookhuis, 1990). Only in critical situations will they search for relevant items to guide their behaviour. The implementation of WORLD thus includes the bulk of the (relevant and irrelevant) objects that human drivers look at.

Constraints arising from speed control: Speed control of human drivers is known to be determined by several external factors. The first factor is the type of road. Speed is determined for example by the road width, the road surface, the number of lanes and traffic regulations. WORLD currently contains only two-lane roads, though the width of the lanes may differ. Traffic signs indicate maximum speed. A second factor is the type of intersection. Speed is determined by the widths of the intersecting roads, visibility at the intersection and traffic regulations as indicated by traffic signs. The third factor that influences speed is the presence and behaviour of other traffic participants. Thus our world contains agents that display 'natural' or 'intelligent' behaviour: observing all the traffic laws while interacting with DRIVER.

Implications of lane keeping. Needless to say, steering requires a road on which to correct lateral deviations and bends to drive round. Both are provided in WORLD.

The above list lacks constraints for tasks such as overtaking and lane merging. In chapter 1 we argued that we would focus mainly on driver behaviour in the approach and negotiation of intersections. This intention is reflected in WORLD. The semi-intelligent agents in WORLD are fully capable of handling intersections and, to some extent, car following, but there are no provisions for tasks like overtaking and lane merging. In this respect our implementation of WORLD is far simpler than the simulations by Reece (1992) and Van Winsum and Wolfelaar (1993).

6.2 Implementation of WORLD

In describing WORLD we will focus on three topics. Section 6.2.1 describes the basic objects and their representations. Section 6.2.2 describes the control structure of WORLD and section 6.2.3 describes the speed control rules that generate the behaviour of the semi-intelligent agents.

6.2.1 Representation of objects

We divide the objects in WORLD into roughly three categories: active or *moving* objects, passive or *static* objects, and *road objects*. The moving objects in WORLD include cars and bicycles. The static objects include houses, trees, traffic signs and parked cars (though the small world implementation in principle allows all types of static objects to be included). We regard the road objects as a special case of static objects. In the following we discuss their basic properties and relations.

Road objects

The basic road objects describe the network of roads and lanes. We distinguish between several types of areas: lanes, roads, intersections, collision areas, and off-road areas. Areas are meaningful spatial abstractions as they reduce the computational load in the control structure that runs WORLD, and the perception module of DRIVER.

Static objects : houses, trees, signs

In the implementation of WORLD a provision is made for the inclusion of user-defined, arbitrarily complex static objects in the database of objects. The minimal attributes that should be provided for any object are its name and type, its position and, if the object occludes other objects, its spatial extent. In addition, the user may add a list of arbitrary attribute-value pairs, possibly nested, so that properties such as colour or shape can be added. In the current version of WORLD we have added houses, trees and traffic and an occasional traffic sign.

Moving objects

The moving objects in WORLD are the semi-intelligent agents that drive cars and bicycles. Table 1 lists the properties and relations relevant to the speed control rules of the semi-intelligent agents and the perceptual functions of DRIVER.

The arguments *o*, *o1* and *o2* represent objects. The road and intersection require further explanation. Time-to-intersection (TTI) is computed as distance-to-intersection/speed¹ Manoeuvre(*o*) gives the manoeuvre that object *o* will perform at the next intersection. We assume that the WORLD agents can see what manoeuvre other agents intend to perform.² The relations between vehicles also seem fairly obvious. Distance-to-object (DTO) applies only to

objects in the same lane or opposite lanes. The time-gap-on-intersection is computed as $TTI(o_2) - TTI(o_1)$. A positive gap means that o_1 will cross before o_2 , a negative gap that o_2 will cross first. $Angle(o_1, o_2)$ is the angle between o_1 and o_2 , taking o_1 's heading angle as the basis. $Visible(o_1, o_2)$ means that o_1 can see o_2 . $Time\text{-}to\text{-}stop(o, a)$ and $distance\text{-}to\text{-}stop(o, a)$ give the time and distance respectively required to come to a full stop, given the deceleration a . $Accelerate\text{-}to\text{-}gap(o_1, o_2, g)$ represents the acceleration required to make the time gap at the intersection between o_1 and o_2 to g seconds. Note that the properties and relations described in this table are in fact the low-level perceptual-access functions of the semi-intelligent agents. We found that this set is sufficient to generate realistic behaviour.

Table 1. Vehicle properties.

Basic Properties		Values
Type(o)		car/bicycle
Position(o)		(x y) coordinates
Speed(o)		m/s
Acceleration(o)		m/s ²
Heading-angle(o)		α degrees
Identity(o)		unique symbol
Road and intersection-related properties		
Distance-to-intersection(o)		m
Time-to-intersection(o)		s
Going-to-intersection(o)		yes/no
On-intersection(o)		yes/no
Area-type(o)		road, lane, intersection, collision, off-road
Area-position(o)		far-left, near-left, far-right, near-right, right, left, behind, this, before, current, opposite
Manoeuvre(o)		turn-left, turn-right, cross, overtake, stop
Relations between objects		
Distance-to-object(o ₁ , o ₂)		m
Gap-on-intersection(o ₁ , o ₂)		s
Relative-position(o ₁ , o ₂)		from-right, from-left, before-me, etc...
Angle(o ₁ , o ₂)		α degrees
Visible(o ₁ , o ₂)		yes, no [objects may be occluded]
Auxiliary functions		
Time-to-stop(o, a)		s
Distance-to-stop(o, a)		m
Accelerate-to-gap(o ₁ , o ₂ , gap)		m/s ²

6.2.2 The semi-intelligent agents and the overall control loop in WORLD

A run in WORLD starts with setting the initial configuration of the semi-intelligent agents. This configuration consists of the initial positions, speeds, accelerations, directions and the route for each of the semi-intelligent agents. The initial position, speed and acceleration of DRIVER is also included in the configuration. This enables us to systematically study critical traffic situations for DRIVER. After setting this configuration, the basic control loop of WORLD takes over.

Table 2. Control loop of WORLD.

1. Update world-clock.
 2. For each vehicle O1
 - Update basic properties. New position and speed are computed using the old position, speed, acceleration and heading angle of vehicle.
 - Compute all the other properties and relations. Use new positions and speed.
 - 3 For each vehicle O1:
 - Apply speed control rules that are not related to other vehicles,
 - For each other vehicle O2
 - apply vehicle-related speed control rules $R(O1, O2)$ and add proposed acceleration to O1's proposal list
 - 4 For each vehicle O1:
 - Choose lowest acceleration from proposal list and set O1 to this acceleration.
-

Table 2 provides a simple description of the basic control loop that runs WORLD. The first step in this loop is the updating of the world-clock. The essence of this control loop is that the agents redetermine their acceleration every clock tick. As in most multiple-agent simulations, whether the agents be molecules or people queuing at a booking office, the world revolves around simulated clock ticks. Owing to the integration of DRIVER into WORLD a frequency of approximately 30 Hz was chosen (section 6.3.3 discusses why). The second step is that the current speed and position are computed using the acceleration chosen in the previous clocktick. The third step is that each vehicle then again determines its speed by applying a number of speed-control rules. There are basically two types. Rules that are about other moving objects and rules that only relate to such factors as the type of road, type of intersection and intended manoeuvre at intersection. The only output that rules produce are *proposals for a new acceleration*. These proposals are placed in the proposal list. Finally, when an agent has applied all its rules it makes a speed control decision. We found that *choosing the lowest acceleration or deceleration* resulted in the smoothest, most 'natural looking' speed profiles, and probably the safest behaviour, especially when crossing intersections.

6.2.3 The speed control rules

By experimenting we found that the aforementioned form of rule processing produced safe and realistic traffic behaviour. In fact, these rules are basically a formalization of the (official Dutch) traffic rules at ordered and unordered intersections. A problem, however, is that the official rules are not specific

enough. A simple example is the rule that on unordered crossings traffic from the right has priority. This rule says nothing, for example, about the speed and distance from the crossing, either for the driver's vehicle or for the crossing vehicle, at which the rule must be applied. We therefore attached a number of parameters to these rules to which we then assigned values on the basis of the empirical literature (primarily data from Chapter 4) and by a great deal of experimenting with the implementation of the small world.

Rules 4, 5 and 6 in Table 3 present an example of how we have formalised the traffic rules for an unordered intersection. For those who do not know the Dutch variant of these rules we give here a short summary: car drivers crossing an intersection yield only to cars from the right. Left-turning cars yield to cars from the right and to oncoming traffic that intends to cross. Right-turning cars yield to none but cyclists intending to cross or turn left. Cyclists basically observe the same rules. In addition, cyclists yield to both cars from the left and right and to cyclists from the right.

Table 3 shows nearly all the rules for speed control in the semi-intelligent agents. They are sufficient to generate fairly 'realistic' behaviour. What are not included in the table below are the car-bicycle, bicycle-car, and car-following rules.

The road and intersection-related rules. Rule 1a is an obvious rule: if no other rules prevail, drive close to the speed limit. Rule 1a is the only rule that ever proposes a positive acceleration. In a sense, then, this is the rule that keeps all agents in WORLD going³. Rule 2 prevents skidding on bends and ensures that turning left or right is not too uncomfortable. Rule 3 is required when houses and other visual obstacles are included in the simulation. This rule takes care of the worst-case scenario in which an invisible car is approaching the intersection at the same TTI. The acceleration function in the then part of the rule will assume a car from the right at the same distance and same speed (and thus the same TTI) and use that in the computation of the time gap⁴.

The car-car rules. Rule 4 takes care of cars from the right. The time gap factors 2 and -2 ensure reasonable behaviour at the intersection. We found this time gap by experimentation but the numbers seem to correspond to what human drivers do, see Van Wolffelaar et al (1991). The TTI tests in rules 2, 3 and 4 ensure that cars do not decelerate too early. The specific TTI value is obtained from the data described in Chapter 4. Rule 5 is an emergency rule that applies to cars from the right that do not stop or that look as though they are not going to stop. It might be asked why we would need such a rule if all cars and bicycles have the same rules? The answer is that *if* all the cars have the same rules then indeed this rule is not necessary. However, one of the interesting uses of WORLD is to experiment with agents that have different rules. Suppose that for one of the agents the time-gap factor in rule 4 is changed to .5. For such an 'inconsiderate' driver we certainly would need a rule of this kind.

The decision rule. It is clear that multiple rules might apply in complex situations. One can conceive of several types of conflict resolution schemes for choosing between proposed actions, but we found that the above conflict resolution rule worked best in situations where all cars and bicycles adhere to the traffic rules.⁵ This decision rule clearly makes this rule system risk-minimizing in most situations, though one can think of perverse starting conditions where accelerating would be safer than decelerating.

Table 3. Speed control rules employed by agents in WORLD

Road and intersection-related rules		
1a	If	speed(self) < maxspeed(road)
	Then	accelerate to maxspeed(road)
1b	If	speed(self) > maxspeed(road)
	Then	decelerate to maxspeed(road)
2	If	manoeuvre(self) == turn-right or turn-left TTI(self) < 4
	Then	decelerate to speed at intersection < 3
3	If	manoeuvre(self) == cross or turn-left TTI(self) < 5 visibility(self, right) < DTI(self)
	Then	decelerate such that (time-gap-oi(self, hypothesized-car) < -2)
Car-car rules		
4	If	manoeuvre(self) == cross or turn-left type(him) == car going-to-intersection(him) == yes relative-position(self, him) == right -2 < time-gap-oi(self, him) < 2 TTI(self) < 5
	Then	decelerate such that (time-gap-oi(self, him) < -2)
5	If	manoeuvre(self) == cross or turn-left type(him) == car or bicycle going-to-intersection(him) == yes relative-position(self, him) == left -1 < time-gap-oi(self, him) < 1 distance-to-stop(self, -4) == DTI(self)
	Then	decelerate till (time-gap-oi(self, him) < 0)
6	If	manoeuvre(self) == turn-left manoeuvre(him) == cross going-to-intersection(him) == yes relative-position(self, him) == opposite-direction -2 < time-gap-oi(self, him) < 2
	Then	decelerate such that speed at intersection < 0
Decision rule		
7	If	multiple accelerations proposed
	Then	take lowest acceleration

6.3 Discussion

We end this chapter with a few concluding notes. The first note concerns the validity of the behaviour of the semi-intelligent agents. The second note is about why the rules in Table 3 do not provide a psychological model of driver behaviour. The third note concerns the integration of DRIVER in WORLD and the final note elaborates on how we might use worlds like WORLD.

6.3.1 The "naturalness" of semi-intelligent agents' behaviour

In nearly all situations the rules ensure that cars and bicycles negotiate the road and intersections smoothly and without serious accidents (Wierda and Aasman, 1988). The only way to spoil the goings-on in WORLD is to crowd it with too many agents. In this case after a while all agents will have come to a halt because too many agents are waiting for too many other agents and an enormous log-jam occurs. It could be said that this also happens in normal traffic. However, in normal traffic someone will always take the initiative and break the log-jam. In WORLD, where all the drivers have the same rules, this will not happen. One solution is to make some drivers more aggressive than others, for example by changing the time-gap-on-intersection parameter. We will deal with the issue of individualism later on in this discussion.

In addition to saying that traffic proceeds so smoothly and without accidents we also claim that behaviour is fairly 'naturalistic'.

In the first place the naturalness is, so to speak, wired in. Most of the parameters in Table 3 were taken directly from the experiments described in Chapter 4. Examples are the speed at the intersection when turning, maximum accelerations and decelerations, distances at which a rule takes effect, etc. Other parameters were found by curve fitting⁶.

In the second place, many observers of WORLD find it fairly natural; however, we never systematically experimented with finding a 'naturalness' score for WORLD. Interested readers may request an MS-DOS version of WORLD and judge for themselves. The version is described in Wierda and Aasman (1988) and is available on a floppy disk.

A third reason for trusting the realism of WORLD comes from experiments with subjects driving in the 3D GIDS simulator at the Traffic Research Centre of the University of Groningen. This simulator, described in Michon (1993) and Van Winsum and Van Wolffelaar (1993), uses approximately the same rules and techniques for generating interactive driver behaviour. In this simulator subjects drive through a 3D projection of a far more elaborate world. The subjects actually have to manipulate the accelerator, brake, gear-stick, clutch, indicators and steering wheel of a prepared car in order to drive through this world. In general the subjects find the behaviour of other agents in the simulator very convincing.

6.3.2 The semi-intelligent agents are not instantiations of a cognitive model of driver behaviour

The claim that the rules of the semi-intelligent agents generate 'realistic' driver behaviour seems to imply that this body of rules provides at least a limited cognitive model of driver behaviour? We argue that this is not the case.

In the first place, the perceptual capabilities of the agents are too perfect. An agent in WORLD has a field of vision of 360 degrees, enabling it to see all other agents simultaneously. It does not suffer from human constraints such as a restricted functional field, or time-consuming eye and head movements. This enables agents to evaluate all other agents at each successive time-step.

Second, an agent in WORLD is not restricted by working memory capacity and working memory speed. Though an agent in WORLD applies the rule-set in a serial order to all other agents, there are no limits to the speed at which rules can be applied or to the number of agents to be considered. In a sense agents display perfect rationality.

Finally, an agent in WORLD does not suffer from physical constraints. There are no relatively slow arms and legs to be moved in order to manipulate the car. Speed control decisions made by the agents take effect at the next time-step, whereas in human drivers a speed control decision launches a flurry of activity, especially when gear-changing is involved, and it takes at least half a second before the decision even begins to show through in the speed of the car⁷.

6.3.3 The integration of DRIVER in WORLD

WORLD is designed to serve as a test-bed for DRIVER. This is therefore the place to discuss how DRIVER is integrated in this world. A problem, however, is that we have not yet dealt with DRIVER and in explaining DRIVER's integration in WORLD it is impossible to avoid some Soar technicalities.

Let us start with WORLD. The control loop of WORLD is indifferent with respect to the car that is controlled by DRIVER and the semi-intelligent agents. As far as WORLD is concerned the only difference lies in the determination of the new acceleration. The semi-intelligent agents use the previously discussed rule-set to generate a new acceleration, whereas DRIVER's acceleration is determined by the current speed and the position of the pedals and the gear-stick in her simulated car.

The real problem of integration is *synchronizing* DRIVER with WORLD. The semi-intelligent agents in WORLD show real-time behaviour. This is of course necessary if you want to use these simulated traffic environments in a driving simulator (see Van Winsum & Van Wolfelaar, 1993) or in an educational setting (see Wierda & Aasman, 1988). DRIVER, however, runs almost a hundred times slower than the semi-intelligent agents. The reason for this is fairly obvious. DRIVER handles not only speed control but also perceptual, motor

and navigation tasks. In addition, DRIVER was implemented in the not particularly efficient Soar 5.2 version⁸.

The solution to the problem was to let DRIVER control the timing of the WORLD clock. In DRIVER the control loop for WORLD (see Table 2) is called up every elaboration cycle. Soar's elaboration cycle is thus the clock tick of WORLD. The reason we selected the elaboration cycle rather than the decision cycle will become clear in the following chapters. In essence it was because the temporal grain for some of the actions that we model in DRIVER, for example eye and motor actions, is finer than the estimated 100 milliseconds for a decision cycle.

6.4 Notes

¹ Note the absence of acceleration in this computation. In Chapter 4 we describe how human drivers seem to use a TTI measure without acceleration

² Driver will have to look at the blinking indicators.

³ Note that the output of the function `accelerate to maxspeed(road)` is a function of the difference between the desired speed and the current speed, though with maximum and minimum decelerations.

⁴ A simple alternative would be to reduce speed to say 15km/h whenever visibility to the right is extremely bad, as this is the speed we found in the chapter that described behaviour at intersections with poor visibility. However, in WORLD visibility is a variable that can be set and thus not always as bad as the intersections described in chapter 4, so we chose the first alternative.

⁵ A note about this form of rule processing. The mechanism described above resembles a production system where multiple rules fire in parallel and where a conflict resolution scheme is used to choose the most appropriate rule in this situation. However, we did not use a production-system mechanism such as OPS5, but programmed the rules directly into Lisp.

⁶ Which sounds somewhat better than "endlessly trying". Note that here we are stepping somewhat carelessly over the issue of the naturalness of world. Remember however that the research aim is to provide driver with a realistic world and not the model behind world.

⁷ Although the rules do not provide a full cognitive model, they do give a good overall description of driver behaviour. We found therefore several other uses for these rules. One of these uses arises from the fact that these traffic worlds are ideal traffic research instruments in themselves. They provide an ideal tool for experimenting with the addition and deletion of new traffic rules or for parametrizing the behaviour of individuals. These simulations offer the possibility for studying non-adaptive behaviour and hence for ascertaining the robustness of the normal rule-sets. We can experiment with different definitions of collision courses by varying the accepted time gaps at intersections and simulating 'nervous', 'inconsiderate' and 'aggressive' drivers. Or we can vary minimum and maximum accelerations, thereby surprising other drivers. This might prove to be one of the strongest points of WORLD as a tool.

⁸ Until Soar 5.2 Soar was implemented in Lisp. The current version of Soar, Soar 6.2, has been rebuilt from scratch and is in general is an order of magnitude faster.

7 Basic motor control

Summary: DRIVER exercises motor control over two arms, two legs, a head and two eyes. This chapter discusses the high-level command language and the low-level motor modules that control the movements of these extremities. The high-level command language involves Soar operators that issue motor commands. The lower-level motor module (LLMM) executes these motor commands asynchronously from other Soar processes.

7.1 Introduction

In this chapter we describe the motor control mechanisms that enable DRIVER to exercise control over its arms, legs, eyes and head. We describe the implementation of these mechanisms with minimal regard to the driving task. In the following chapter we will examine how this implementation is used in the planning and execution of motor plans and in actual driving situations. Before dealing with the implementation, however, we would first like to discuss why we find it so important to include motor control in DRIVER. Next, in section 2, we will draw up a list of constraints that shaped the motor control mechanisms in DRIVER. The list is derived both from the motor control literature and from Soar itself.

A first reason for including elaborate motor control mechanisms in DRIVER is that motor control is an essential and mentally taxing part of the driver task. In terms of the Model Human Processor (Card, Moran & Newell, 1986), motor control consumes cycles of the central cognitive processor. Or, in terms of Soar, a number of operators in the base level space are dedicated to motor control, thereby taking up processing time that could be used for other tasks. In Chapter 4 it was shown that expert drivers perform at least five discrete motor operations per second in a critical situation such as the negotiation of an intersection¹. Studies that discuss the mental strain of steering and controlling the engine can be found in, for example, MacDonald (1977) and Wierda et al. (1987).

A further reason for including motor control is that learning to control the car is an important part of driver education. In their case study of a novice driver, De Velde Harsenhorst and Lourens (1987,1988) show that 16 percent of all

the comments, corrections, verifications, and suggestions made by the driving instructor pertain directly to problems involving control of the car engine. However, this is not the only task that bears a strong motor control component. Motor control is also required in visual orientation (27 percent)², lane keeping and negotiation of curves (21 percent), and speed control (also 21 percent). Remarks classified by De Velde Harsenhorst and Lourens (1987, 1988) as speed control or lane keeping remarks include statements such as "next time press the brake earlier", "next time brake a little more", "you started to steer too early", or remarks that instruct the driver how to hold the steering wheel. All these remarks bear a strong motor control component. De Velde Harsenhorst and Lourens also found, not very surprisingly, that the proportion of motor control remarks was highest in the first part of the novice's driving lessons. Studies by Lourens and Van der Molen (1987) and Groeger et al. (1990) confirm these findings.

However, the most important reason for the inclusion of motor control in DRIVER is that a unified theory is incomplete without it. The previous two reasons are actually an illustration of this. Newell (1990) notes that many cognitive theories treat motor control as completely external to the main part of the theory. However, as soon as a cognitive theory purports to explain interactive behaviour in a complex, dynamic environment, then motor behaviour, i.e. the planning and control of extremities, becomes an important issue. Unfortunately, according to Newell the integration of motor control in Soar is currently unsatisfactory:

".. the starting point for the extension of a unified theory of cognition into motor behaviour is the interface between central cognition and the motor system. Cognitive intention produces a command to release motor action. But what is the language of commands that cognition uses to send messages to the motor system? Knowing that would be enough to get started. The command language structures the way the motor systems appear to cognition and thus determines cognition's tasks..... Unfortunately the nature of the command language used for the motor system is exactly where obscurity is deepest....." (Newell, 1990, p 259)

Despite this pessimism we opted for inclusion of some form of motor control in DRIVER. We wanted at least to model the cognitive activity involved in motor control as it is described in the Minimal Scheme for Immediate Responses (Newell, 1990, Ch. 5). In addition we wanted a simple timing model for the movements of the hands, feet, eyes and head. Newell (1990, pp 261) supports such an approach when he writes that

"...for many cognitive activities (though far from all), the motor system operates almost as an exterior appendage. This is true of speeded response tasks which are actions such as a button push, a keystroke, or an uttered single word. They are overlearned skills that are tightly coordinated and under good control, whatever the internal dynamics of their realization. Some crude timing model is needed for the

motor system to fire up and execute, in order to compute the total time from stimulus to response, but that is entirely phenomenological..."

7.2 The constraints shaping DRIVER's motor control

In the design of DRIVER's motor control mechanism some important constraints provided by the literature and Soar itself were taken into consideration. Table 1 lists the constraints that will be discussed in the following paragraphs.

Table 1. Motor control constraints from Soar and the literature

Constraints from the Literature

Motor Programs in the form of tangled hierarchies
Parameters: target location, body part, speed, force, no scaling
Learning by chunking
Timing
Feedback

Constraints from Soar:

Intend operators occur in the base level space (BLS)
Decoded motor commands via Soar input/output mechanism
Chunking

The discussion of the main motor control theories was targeted at finding constraints that apply to the high-level command language that specifies the interface between cognition and motor control.

Motor programs. The first constraint is the use of motor programs or plans as hierarchically ordered lists of single motor commands (Jordan & Rosenbaum, 1989). Because the nodes in different branches within these hierarchies may have interconnections, the term tangled hierarchies is sometimes used. In the following chapter, in which motor planning is discussed, examples will be given of these tangled hierarchies³.

Motor learning. If one accepts the notion of motor programs then the next logical question is how these are acquired. One candidate learning mechanism in the field of motor learning is chunking. Chunking is one of the theories that explain why the power law of practice is applicable to motor learning and skill acquisition (Newell & Rosenbloom, 1981). Keele (1987) discusses the promising possibilities of using production systems and chunking in motor control and motor learning.^{4,5}

Motor command parameters. Within the group of researchers that accept the notion of motor programs there is also a discussion about the parameters of the single motor commands that comprise the motor programs. Schmidt (1980) argues that target location, body part to move, the speed of the

movement and the scale of the movement are definitely parameters. It is still unclear whether force is a parameter

Timing. Some researchers posit that timekeepers are essential in motor control (Shaffer, 1985). A timekeeper in its most stringent form is an internal clock providing regular clock ticks; in a somewhat less stringent form a timekeeper just provides the notion of time. Other authors argue that it is possible to do without timekeepers altogether. Nooteboom (1985) and Rumelhart and Norman (1982) model behavioural phenomena in speech and typing without having to refer to timekeepers. We found that in DRIVER's current simple form of motor control we could do without an explicit time keeper.

Role of feedback. There is also controversy with respect to the importance of feedback in motor control. A theory in favour of feedback is the reflex chaining hypothesis: a complex movement consists of a sequence of single movements activating one other. The perception of the result of the current sub-move will activate the next move (see Keele, 1987 for a discussion.)⁶ For DRIVER we found that the inclusion of feedback was indispensable.

Table 1 also lists the main Soar constraints incorporated in DRIVER's motor control. Note that these constraints do not come from the Soar architecture itself but from Soar as a UTC and specifically Newell's minimal scheme of immediate response (Newell, 1990, p. 262; see also Section 4 in Chapter 1 of this study).

The first constraint is that motor commands are given by an operator in the base level space. In Newell's minimal scheme of immediate responses this is called the *intend* operator (though in this chapter we will call it the *move* operator). This operator adds motor-command structures in symbolic format to working memory. Once a motor command is in working memory, Soar is committed to that command and control leaves central cognition. Control is still possible over the initiated move but this requires a new intend operator. The question of why we need an intend operator instead of productions that issue motor commands directly and why an intend operator should occur only in the BLS is discussed in detail in Newell's Unified Theory of Cognition. In summary, the answer to the first question is that we need an operator instead of only productions because the choice of operators is the only point in Soar where real decisions can be taken. The answer to the second question is that allowing intend operators only at the base level prohibits unintentional movements from within sub-goals in, for example, look-ahead.

7.3 Implementation of motor control in DRIVER

Figure 7-1 presents an overview of the motor control cycle. Move operators put motor-command structures on output links on Soar's top state. The Soar I/O takes these command structures from working memory and transfers

them to the Lower-Level Motor Module (LLMM). The LLMM takes care of the execution of the command and sends feedback to the Soar I/O, which channels the information through to the predefined input links on the top state in Soar's working memory.

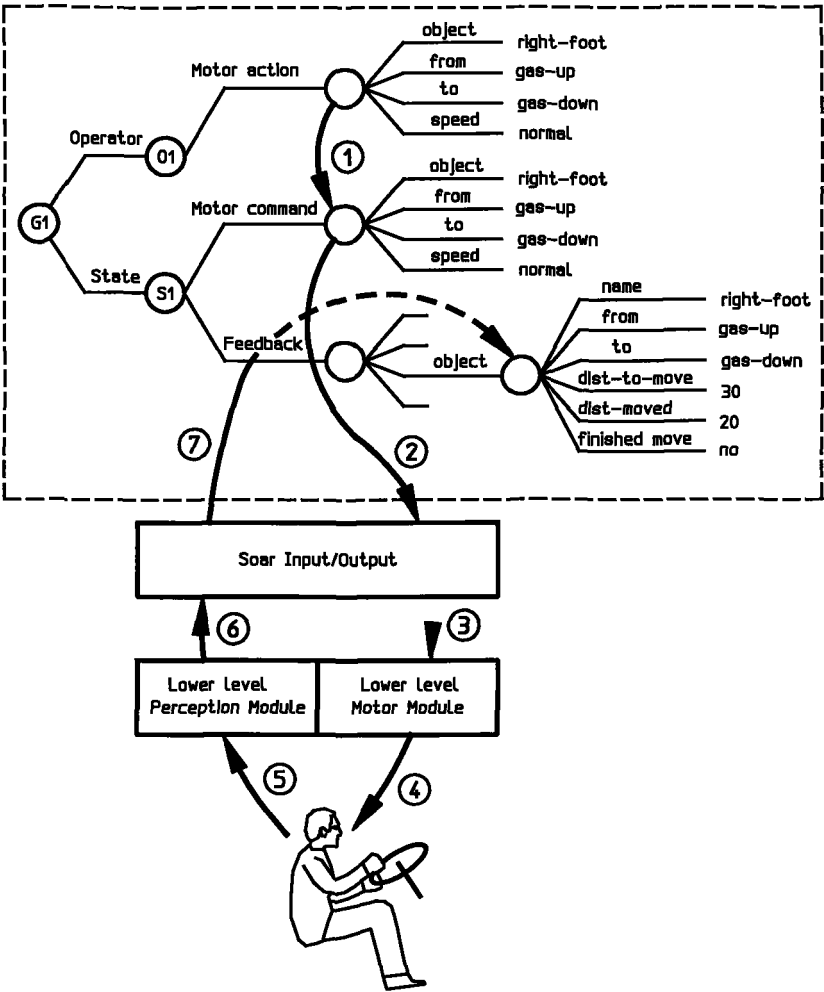


Figure 7-1. The basic motor control cycle. For an explanation we start at the top of the figure. O1 is a move operator in the current operator slot of goal G1. O1 carries a motor action MA that specifies that the right foot should be moved from accelerator up to accelerator down. After the application of the operator (1) a motor command is attached to the top state. The Soar I/O takes the motor command from the state (2) and feeds it to the LLMM (3). This module installs the command and executes it in the real world (4). The lower level perception module (LLPM) gathers all relevant motor feedback (5) and transfers it to the Soar I/O (6), which transfers it to the body representation on the top state (7). The picture of the driver behind the wheel symbolises (a) that the LLMM actually moves the body and (b) that the feedback concerning the state of the actual body is transferred back into the LLPM.

The following sections describe the implementation of DRIVER's motor control in more detail. Section 7.3.1 describes the representation of DRIVER's body in Soar's working memory. Section 7.3.2 describes the high-level command language that consists primarily of Soar operators that issue motor commands from Soar's working memory. Section 7.3.3 describes the lower-level motor module (LLMM) that executes these motor commands asynchronously relative to other Soar processes.

Table 2. DRIVER's basic body representation for hand and feet.

feature	value
from	in symbolic format (ground, accelerator-up, accelerator-down, brake-up, brake-down, clutch-up, clutch-down, steer, stick-1 to stick-5)
to	idem
current-position	idem
moving?	yes/no
direction	up/down
speed-s	speed in symbolic format: slow/normal/fast
speed	distance to move per clock tick
distance-to-move	distance between 'from' and 'to'
distance-moved	distance moved so far
current-command	pointer to current command
finished-move?	yes/no

7.3.1 DRIVER's body representations

Our occasional use of the term DRIVER's body is misleading as this body consists only of two hands, two feet, eyes and a head. DRIVER has two representations for its 'body'. The first representation is in Common Lisp, external to Soar. It represents the 'real, external' configuration of the hands, feet, eyes and head and is almost purely in numerical format. It is used by the Lower Level Motor Module to simulate the movements between the various pedals and the steering wheel.

The second representation resides in Soar's working memory in the Base Level Space. This representation is used by the high-level command language described in the following section. Table 2 displays the main attributes and their values for the representation. Note that the representation is a mixture of symbolic and numerical information. The positions are in symbolic format whereas speed, distance-to-move and distance-moved are in numerical format.

In a following section we will describe how the *motor feedback cycle* keeps the representation up to date with DRIVER's 'real' body'.

7.3.2 Move operator and the (high-level) motor command language

What Newell calls *the command language that sends messages to the motor system* (Newell, 1990, pp. 261) is in DRIVER the combination of (1) the representa-

tions of motor actions in working memory, (2) the *move operator* that operates on these representations, and (3) the rules that generate, select, and apply this operators.

The only way that DRIVER can move its extremities or eyes is by applying a *move operator* in Soar's top state. The basic role of the operator is, as we saw in Figure 7-1, to put actual motor commands on the output links on Soar's top state. The following paragraph discusses the representation of the move operator, after which we will cover the main phases in the life of the move operator, namely its generation, selection, application, and termination.

Operator representation

Move operators have links to so-called *motor actions* (see Figure 7-1). These motor actions are in effect the specifications of the movements to be executed. The most frequently used features of motor actions are shown in Table 3. Note that the parameters are not directly attached to the move operator; the indirection of using a motor-action object allows for multiple motor-action links on the operator. This makes possible the simultaneous initialization of moves. It is thus possible for one move operator to start the movement of two extremities. We allowed for this possibility because humans are also capable of starting several extremities at the same time. However, in DRIVER we usually have only one motor action per move operator⁷.

Table 3. Features of motor-action structures.

Features	Values
object	rightfoot, leftfoot, righthand, lefthand, eyes, head
from	relevant positions in car
to	idem
speed	number
movement-type	apply-force or free
till-position-is	percentage
device-action	pointer to corresponding device action
next-motor-action	pointer to next action

Only the *to* and the *object* parameters are mandatory, all other features are optional. Note that the *from* and *to* features that specify the origin and destination of the move are represented in symbolic terms and not in some type of coordinate system. Representing the location in terms of a coordinate system is possible (see for example Laird, 1990) but the current symbolic format suffices to demonstrate the usefulness of this type of representation in a cognitive model. Remember that the object of this chapter is a command language at the level of cognition. It is the task of the lower-level motor module (LLMM) to find the exact physical locations that go with the symbols. The *movement-type* feature indicates whether force has to be applied during a move. The *till-position-is* feature is meaningful when moves apply to pedals or the steering wheel. The *device-action* is a pointer to the device action for which

the move is to be executed. For example, if there is a device action to move the brake down, then the corresponding right-foot move operator will have a link to it.

Generation

Motor-command operators and motor-action objects are generated in the course of problem solving. In the following chapter we show how in gear changing they are usually generated from a pre-computed plan, whereas in steering they are usually generated directly by production, i.e. without any intervening plan. The following provides a Soar example of such a generator production.

```
(sp body-action*gen-move-command*brake*rightfoot
  (goal <g> ^problem-space <p> ^state <s>)
  (problem-space <p> ^name get-body-actions)
  (state <s> ^vehicle-action <va> ^<va> do .^<va> done)
  (object <va> ^from << brake-up brake-down >> <rpos>
    ^to << brake-up brake-down >> <vto>)
  (state <s> ^rightfoot <rpos>)
  -->
  (goal <g> ^operator <o> + -)
  (operator <o> ^name move ^motor-action <ba>)
  (object <ba> ^object rightfoot ^vehicle-action <va> ^from <rpos> ^to <vto>
    ^movement-type apply-force ^speed normal))
```

In English: if we want to move the brake and this has not yet been carried out and the right foot is on the brake somewhere, then propose a move operator with a motor-action object that specifies the right foot to be moved, using force but with normal speed.

Selection

Remember that selection ultimately refers to the ordering of operators. The ordering is based on preferences that are also generated during problem solving (to be discussed in next chapter). The rule below is a simple example of a preference or selection rule.

```
(sp motor-ops*prefer-earlier-ones*vehicle
  (goal <g> ^problem-space <p> ^state <s> ^operator <o1> + <o2> + ^object nil)
  (<s> ^body-plan <bp>)
  (object <bp> ^motor-action <ba1> <ba2>)
  (object <ba1> ^vehicle-action <va1>)
  (object <ba2> ^vehicle-action <va2>)
  (object <va1> ^general-before <va2>)
  (operator <o1> ^name move ^motor-action <ba1>)
  (operator <o2> ^name move ^motor-action <ba2>)
  -->
  (goal <g> ^operator <o1> > <o2>))
```

The rule states that if there are two operators <o1> and <o2> for motor actions <ba1> and <ba2>, and these motor actions refer to two vehicle actions <va1> and <va2>, and <va1> has to be done before <va2>, then <o1> is given a higher preference than <o2>.

Application

The result of the application of a motor-command operator is a new motor-command structure on the top state. This motor-command structure is defined as a regular Soar output link for the Soar I/O. Note that in the example below only one extremity is attached to the motor-command structure. However, if multiple motor actions were specified on the motor-command operator, then multiple extremities would also be attached to the motor-command structure. This is what enables multiple commands to be initiated at the same time.

```
(state <s> ^motor-commands <mc>)  
(o <mc> ^extremity <rf>)  
(o <rf> ^name right-foot ^from brake-up ^to brake-down  
  ^speed normal ^movement-type apply-force)
```

During the first elaboration cycle after the motor-command structure has been attached to the state, regular Soar I/O functions will transfer the structure to the lower-level motor module (LLMM).

Termination

As soon as the new motor-command structure is added to the state the operator is terminated. It is crucial to note that the operator is *not* waiting for the termination of the move. This in the spirit of Newell's intend operator that was described above. Thus as soon as the intend operator puts the encoded command in working memory, control leaves central cognition and is handed over to the lower-level motor control mechanisms.

7.3.3 Lower-level motor module (LLMM)

This study mainly focuses on the high-level command language. In-depth modelling of lower-level motor control has certainly not been attempted. The general idea behind the LLMM is to compute or simulate the time it takes to move an extremity over a certain distance. In order to avoid confusion in the following it should be noted that the LLMM is not part of Soar, but a simulation in Common Lisp.

Three functions of the LLMM will be discussed in the following subsections. The first function is to translate commands from the Soar I/O into LLMM structures (the LLMM body representation.) The second function is to update the position of the extremities and issue new commands, if available.

The third function is to return feedback to Soar about the state of the lower-level motor system.

Channelling and translating commands

The first function of the LLMM is to unravel the motor-command structures that were taken from Soar's top state and then channelled through the Soar I/O. The structures are translated into LLMM commands and added to a motor-command list of the extremity that the command related to. The representation of the body in the LLMM (in Lisp) largely shadows Table 2. Besides the *current command*, the LLMM provides a *command list* as DRIVER is allowed to issue move commands without waiting for previous commands to be finished.

The LLMM knows the interior of the car. If *from* and *to* parameters are given, the *distance-to-move* can be computed. *Distance-to-move* and the *speed* parameter are used to compute the *distance-moved* while updating extremity positions (see next section). Speed is given by the Soar move operator or obtained from a default table. For the latter table we tried to use the well-known Fitts' law (Fitts, 1954) to compute the movement time between positions, but found this to be rather unsatisfactory, especially for computing the movement time of the feet. We therefore used a modified version of Fitts' law for moving between car pedals (Drury, 1975) that takes the size of shoes into account.

Main control structure

The second function of the LLMM is to update the position of the extremities. Every elaboration cycle the following procedure is carried out for each extremity, including eyes and the head:

For each extremity on Soar's top state do the following:

- Search for new commands in Soar output link. Put new commands at the back of appropriate motor-command list.
 - If the current command is finished then the current command is set to the next command on the command list. If no speed is given it is set to the default speed, distance-to-move is computed and distance-moved is set to 0;
 - If the current command is still active (i.e. finished-move = no) then the new positions of the extremities and the distance-moved are computed using the speed parameter. If during this computation the target is reached the movement is stopped by the motor module and finished-move is set to yes.
 - Return feedback to the motor-feedback input link on the state via Soar I/O
-

This control structure assumes that elaboration cycles occur at fixed time intervals. Remember that we made the same assumption in controlling time in WORLD.

Feedback

The LLMM's third function is to provide the perception of the body, in particular the feedback concerning the current command. The most important information that the LLMM channels through the Soar I/O to working memory was listed in Table 2: the *from* and *to* parameters of the *current-*

command, the *moving?* status, the *distance-to-move*, and the *distance-moved*. The feedback information concerning the current command serves as the memory of the move operator which is removed from the operator slot as soon as the motor command is put on the state. By providing the feedback DRIVER still 'knows' what it was/is doing. The following shows a fragment of the motor feedback on the state:

```
(state <s> ^motor-feedback <fb>)  
(object <fb> ^extremity <lf> <rf> <lh> <rh>)  
(object <rf> ^name right-foot  
    ^from accelerator-up ^to accelerator-down  
    ^distance-to-move 30 ^distance-moved 20  
    ^moving yes ^finished-move no)  
(object <lf> ^name left-foot
```

The feedback provided by the Soar I/O is destructively modified (overwritten) every elaboration cycle. This implies that if DRIVER wants to use the information in the motor-feedback structures for later use it must make a copy of the feedback structures.

7.4 Discussion

This chapter showed how constraints from Soar and the motor control literature were taken into account for the high-level motor command language and lower-level motor modules. It is an important caveat that the LLMM provides only a crude timing model for generating the time it takes for an extremity to move over a certain distance. It bypasses all the complexities that reside in human low-level motor control (muscle tone, visual-motor interactions). The reason for including the LLMM is that a mechanism was required that (1) executes motor commands separately from other Soar processes and (2) takes approximately the same amount of time as the execution.

Given these requirements the whole setup of the motor control mechanism appears to be a way to obtain the functionality of the intend operator from Newell's *Minimal scheme of immediate responses*. Once the intend (in DRIVER: move) operator adds a motor command to WM, control is handed over to the LLMM and the intend operator is retracted from WM. This retraction is the reason feedback is so important. Feedback basically provides a kind of *flowing* body representation. DRIVER continuously knows (1) what it is doing and (2) when a command is finished. The obvious importance of this scheme is that Soar does not have to wait for a motor command to be finished in the external world. The move operator only initiates a movement. Control over execution is left to the LLMM, but by providing feedback Soar still has some form of control, as it can see how execution is going. If things go awry DRIVER can still intercept by issuing new commands.

7.4.1 Timing

One issue that also came up in the implementation of WORLD is that the LLMM uses the elaboration cycle as a basic clock tick. This presupposes a constancy of the elaboration cycle but unfortunately Newell's estimates range from 10 to 100 milliseconds. However, this is currently the only solution for synchronizing Soar with the external WORLD and the LLMM. One solution that is still some way from being realised is to have faster computers so that Soar can work in real time.

7.4.2 Experimenting with the control mechanisms

A final note is that the implementation of motor control still leaves considerable room for experimenting. Various things may be varied by the researchers. First, parameters may be changed in the LLMM, for example the default speed of an extremity. Second, new devices may be added from the car or old devices removed: for example it is relatively simple to add a radio or to change the currently manual transmission car to an automatic.

A third interesting use of the mechanism is to play with the way multiple commands are issued. We have seen that a move operator can in principle have one action object for each extremity. This allows the multiple initialization of movements by different extremities, a capability humans certainly have. What is also possible is to add action lists to the move operator (one list per extremity). This makes it possible to initiate complex moves with one extremity and even to initiate complex moves that involve multiple extremities. Typical examples of the latter are tasks such as typing, gear-changing by an expert or a musical activity like singing a Schubert song while accompanying oneself on the piano. Even when motor-action lists on the move operator concern only one extremity, the motor actions for different extremities may be connected. In the following chapter we will see how motor actions within and between extremities are ordered by next, before and wait links.

The next chapter

This chapter has considered the motor control mechanisms without regard to the driving task. The following chapter shows how the high-level command language is used in the relatively complex task of engine control and gear-changing. It also discusses the role of chunking in motor control. Chapters 9 and 10 discuss in more detail the role of the same motor control mechanisms in eye and head movements.

7.5 Notes

¹ That is, if we include eye movements and head movements.

² The percentages refer here to the percentage of remarks made by the instructor regarding the particular task.

¹ Current research is directed at proving the existence of motor programs (see Keele, 1987; Jordan and Rosenbaum, 1989; or Krammer, 1990, for general discussions). A number of authors have proposed that serial ordering of motor actions is achieved with hierarchically organized plans or programs, a view largely supported by error patterns in spontaneous behaviour. (See Jordan and Rosenbaum, 1989, for an overview).

² More than a page is dedicated to the use of chunking and production systems in motor control in Keele's excellent chapter in the celebrated *Handbook of Perception and Performance*. Surprisingly enough the general example in this treatment is not motor control but chess. This nicely demonstrates the state of the art in the understanding of chunking in motor control.

³ Another, currently dominant psychological theory of motor learning is schema theory (Schmidt, 1975, 1982). Schmidt defines a schema as the relationship, built up over past experience, between the response specification of a movement, the actual outcome, the sensory consequences and the initial conditions. However, as Schmidt himself states: "Like most programming theories, the schema theory does not specify from where the motor programs come. This is an important problem, but the level of knowledge at this time does not allow much to be said about this process. So, the theory has had to assume that programs are developed in some way and that they can be carried out by executing them with the proper parameters..." (Schmidt, 1982, p. 593).

⁴ Schmidt (1980) provides some counterarguments: (1) movements are often faster than the time required to process feedback; (2) complex movements are possible, even if feedback is disabled; and (3) the time elapsed before a movement is initiated is longer with more complex moves.

⁷ An additional method of issuing multiple commands to the LLMM with a single move operator is by allowing for lists of motor actions. The *next-motor-action* feature is the link to the next motor-action object. The only constraint that applies to these lists is that each member of the list refers to the same extremity. This feature is currently not used in DRIVER and I am also unsure about the psychological validity of this method. However, for certain compound ballistic movements it seems implausible that cognition controls the movement all the way.

8 Motor planning and vehicle control

Summary. This chapter describes the use of the basic mechanisms for motor control. DRIVER drives a manual-transmission car. By manipulating the gear-stick and the clutch, brake and accelerator pedals, DRIVER controls the speed of its simulated vehicle. This chapter focuses especially on gear-changing. It is shown how DRIVER as a beginner is heavily involved in problem-solving, i.e. sub-goaling, in order to design a movement plan that will get the car into the correct gear. Once DRIVER has learned how to change gear most of the further learning time will be dedicated to the exact timing of its movements.

8.1 Introduction

The topic of this chapter is how DRIVER builds and learns a motor program and how it executes it. We will deal with this subject on the basis of one of the most complex operations in a car, namely gear-changing. This introduction first discusses the types of knowledge that are involved in learning to change gear. The following section then presents a step-by-step description of the implementation of DRIVER's motor planning and execution.

8.1.1 Types of knowledge in gear-changing

A simple task analysis reveals that in order to perform a smooth gear-change a novice driver must acquire several types of knowledge. For convenience' sake we distinguish four types.

- 1) *functions*. The novice's first task is to learn the function of the various pedals and the gear-stick and which extremity goes with which device.
- 2) *constraints*. The second task is to learn the inherent constraints of car mechanisms. Novice drivers must learn that the accelerator pedal must be up when the clutch is pressed, or that the gear-stick may only be moved when the clutch is down.
- 3) *ordering*. The third task is to learn the correct ordering of device actions and corresponding body movements while incorporating these constraints. To

illustrate the complexity of this ordering for those who only drive an automatic: novice drivers learn that to change gear they must first release the accelerator pedal and press the clutch pedal, then bring the right arm from wherever it is to the gear-stick, move it through neutral to the desired new gear and then bring the right foot from wherever it is (it might be on the brake) to press the accelerator again while releasing the clutch pedal.

4) *fine tuning the execution*. The fourth task involves learning how to adapt or fine-tune the execution of the device actions and body actions to various situations. Fine-tuning in engine control and speed control involves both the *when* of commands and the *how-far* in pedal and steering control. De Velde Harsenhorst and Lourens (1987,1998) show how a novice driver during his or its learning process goes from rather chaotic timing to predictable expert timing behaviour. Remember how the experienced drivers described in Chapter 4 displayed remarkable inter-subject timing in their entire approach to an intersection.

8.1.2 Aspects of learning in gear changing

Learning the functions of the pedals and learning which extremity goes with which device is relatively simple. Learning the functions of the pedals is a matter of instructions and learning which extremity goes with which device is heavily constrained by the locations of the pedals and the orientation of the body. About the only complexity is that the right foot deals with both the accelerator and the brake.

More interesting is the acquisition of the car constraints and the ordering of motor actions. For example, a constraint that the accelerator must be released before pressing the clutch can be acquired in several ways. One way is for the instructor simply to provide step-by-step instructions. The instructor tells the novice driver that the first step is to release the accelerator and the second step is to press the clutch pedal, etc.

Another way is for the instructor to explain how the car engine works and how the pedals relate to the engine, in the hope that the novice will derive constraints and an action plan from these instructions. Both these types of instructions for working a device are described in the literature. The first method of instruction generates so-called *operational* or *how-to-do-the-task* knowledge. The second type of instruction generates so-called *figurative* or *how-the-device-works* knowledge (Halasz and Moran, 1983; Kieras and Bovair, 1984). The literature seems to support the view that teaching users a conceptual model of the device facilitates the development of expert performance, particularly with complex and novel tasks (Churchill and Young, 1990). The instructor in the De Velde Harsenhorst and Lourens (1987,1998) study employed both types of instruction; however, one criticism of this particular instructor was that his explanations of the workings of the car engine were far too abstract to be of any use to the novice driver. We also saw in the De Velde Harsenhorst and Lourens study that explicit instructions play a minor role in

driver education, especially in car control. Of the 683 car-control-related remarks, 549 were classified as corrections, whereas only 30 were classified as genuine instructions (see also table 1 in Chapter 4).

In the De Velde Harsenhorst and Lourens study we thus see primarily a third way of learning to handle a device, namely a combination of *learning by doing* and *learning by instruction*.

8.1.3 Preprogrammed and learned knowledge in DRIVER

We found that for this study it would be far too ambitious, that is, far too complex, to model the acquisition of all four types of knowledge. Therefore, DRIVER does not have to acquire all the types of knowledge discussed above. Some types of knowledge are preprogrammed by the programmer.

Preprogrammed knowledge

In the present version of DRIVER, the first two types of knowledge that were discussed above are preprogrammed in. Thus, DRIVER knows the names and the functions of the accelerator, brake and clutch pedals, the gear-stick and the steering wheel and it also knows which extremities belong to which device. For example, it knows that the right foot handles both the accelerator and the brake pedal.

Moreover, DRIVER knows the important constraints of the car mechanisms, especially with respect to changing gear. In fact there are only two important constraints. The first is that the accelerator may not be down when the clutch is down and the second constraint is that it is only possible to change gear when the clutch is down.

Knowledge learned by DRIVER

DRIVER does learn the ordering of action and, to some extent, the finetuning of plan execution. In Section 8.2 we describe in detail how DRIVER learns to build so-called *device and motor plans*. A *device-plan* is defined as a list of *device-actions* for the purpose of changing the state of the device. Thus for a gear change operation the plan begins with the device actions: move-accelerator-up, move-clutch-down, stick-to-neutral, stick-to-X, etc. A *motor plan* is defined as a list of *motor actions* that will carry out the device plan. A motor plan is directly derived from a device plan but it is far more elaborate than the device plan itself because it must also handle the movements of the extremities between the devices.

Fine-tuning. DRIVER does not come with predefined knowledge about when to execute a command or to what extent a pedal has to be pressed in. It will have to learn this by a process of trial and error and with the help of a (simulated) instructor. The interesting thing about acquiring this knowledge is that whereas in learning motor plans chunking works satisfactorily, it is much harder to get DRIVER (Soar) to learn this type of knowledge. Section 2 of this chapter and the chapter on speed control will discuss this in more detail.

8.2 Implementation of car control in DRIVER

The following sections describe the problem-solving and learning in gear-changing in chronological order. The first section describes how the process starts with a change-gear command. The next section describes how a *device plan* is built for this particular change-gear command. The third section shows how a *motor plan* is derived from the device plan. The following section discusses the chunks that are learned while building these two plans. The fifth describes how a motor plan is executed in the external world.

8.2.1 The change-gear operator

In our example DRIVER approaches a traffic situation that requires a reduction in speed. DRIVER's first action is to choose a *desired speed* and a *desired gear* for that speed. Because, in our example, speed has to be reduced relatively quickly, the change-gear operator is annotated with the *^use-brake* property. Just how DRIVER determines the desired speed and desired gear is described in Chapter 11, where speed control in DRIVER is discussed.

8.2.2 Building device plans

As DRIVER, in our example, does not yet possess the knowledge to implement the *change-gear* operator, an impasse arises. In order to deal with the impasse that arises for the change-gear operator, the *change-gear problem space* is proposed. The ultimate goal of this problem space is to design a *motor-plan*, i.e. a list of *motor-actions* that can be executed in the external world.

The first operator proposed in this space is the so-called *device-plan* operator, as DRIVER's first sub-goal is to build a *device-plan* consisting of multiple *device-actions* which together comprise the gear-changing operation. The set of device-actions relevant to gear-changing includes the set of operations for manipulating accelerator, brake, clutch and gear-stick. Note that these operations do not include any reference to an extremity that will carry out the device action. Table 1 lists some of the rules and constraints that propose the basic *device-actions* and build the device-plan.

These six rules suffice to build the skeleton of the device plan. Rule 1 ensures that the stick is brought from position X to position Y through the neutral position. Rules 2 and 3 embody the constraint that when the gear-stick is manipulated the clutch must be down. Rules 4 and 5 embody the constraint that when the clutch is down, the accelerator must be up. A few remarks are due here. First, note that these actions bear no reference to the extremities that will carry out the device actions. Second, note that these rules provide only a very general scheme of actions; no timing information or information about how far pedals have to be pressed is given. A third observation is that the ordering of the device actions after the application of these rules is locally restricted to one or two actions. In order to facilitate the generation of a

motor-plan from this device plan the general ordering for all these actions is established with a few simple additional rules. Figure 8-1 shows a Soar representation of the device-plan. The figure includes the brake-down and brake-up actions. Remember that in the example they were specified on the change-gear operator.

Table 1. Rules proposing basic device actions.

[1]	If Then	operator Change-Gear issues a change from gear X to gear Y add to the device-plan: a device-action A1 to move the stick from X to Neutral and add to A1 a next pointer to A2 and a device-action A2 to move the stick from Neutral to Y
[2]	If Then	device-action A1 moves the stick from X to Neutral and the clutch is up add to the device-plan: a device-action A2 to move the clutch down and add to A2 a next pointer to A1
[3]	If Then	device-action A1 moves the stick from Neutral to X add to the device-plan: a device-action A2 to move the clutch up and add to A1 a next pointer to A2
[4]	If Then	device-action A1 moves the clutch down add to the device-plan: a device-action A2 to release the accelerator and add to A2 a next pointer to A1
[5]	If Then	device-action A1 moves the clutch up add to the device-plan: a device-action A2 to press the accelerator and add to A1 a next pointer to A2
[6]	If Then	device-action A1 moves the accelerator down and the brake is down add to device-plan: a device-action A2 to release the brake and add to A1 a next pointer to A2

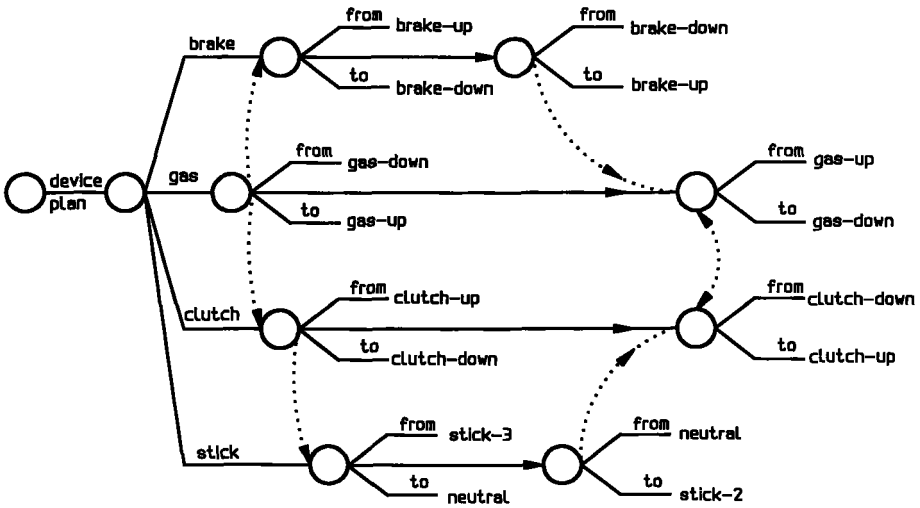


Figure 8-1. The Soar representation of a device plan. The nodes represent the device actions. All device actions are connected to the device-plan node but these connections are not drawn in order to avoid a cluttered picture. The solid lines represent *next* relations between actions. The curved lines represent *next* relations between different devices. The arrow heads show the direction of the next relation.

8.2.3 Building a motor-plan

After building the device plan DRIVER proposes and selects the *get-motor-actions* operator. This operator cannot be applied in the change-gear problem space, thus giving rise to a no-change impasse. DRIVER tackles this impasse in the *get-motor-actions* problem space that knows all about motor actions and body constraints. A new state is created in this problem space but pointers to motor feedback on the top state and the device-plan ensure the availability of the necessary context. These pointers are added by the *init-body-and-car* operator. After installing this context DRIVER proceeds to generate motor actions. The table 2 shows roughly the general format of the rules describing how the motor actions are generated.

Note that the motor actions are not added directly to the state (as is the case with the device actions) but to move operators. This indirect mechanism is used to facilitate the ordering of motor actions. Another aid in ordering actions is the use of pointers to device actions. Note that the above rules generate *too many* 'goto' operators; if an extremity is already at the appropriate device then this operator is superfluous and the ordering of the motor actions can thus be rejected. The difference between goto and apply-force motor actions was explained in the previous chapter.

Table 2. Rules that generate motor actions

[7]	If	device-action DA is required at device D extremity E is required to do DA
	Then	propose a move operator with a 'goto' motor-action MA1 that moves E to D and add a pointer to DA
[8]	If	device-action DA is required extremity E is required to do DA
	Then	propose a move-operator with an 'apply-force' motor-action MA to execute DA and add a pointer to DA
[9]	If	a move operator with a goto motor-action MA1 for device D a move operator with an apply-force motor-action MA2 for D
	Then	add to MA1 a next pointer to MA2

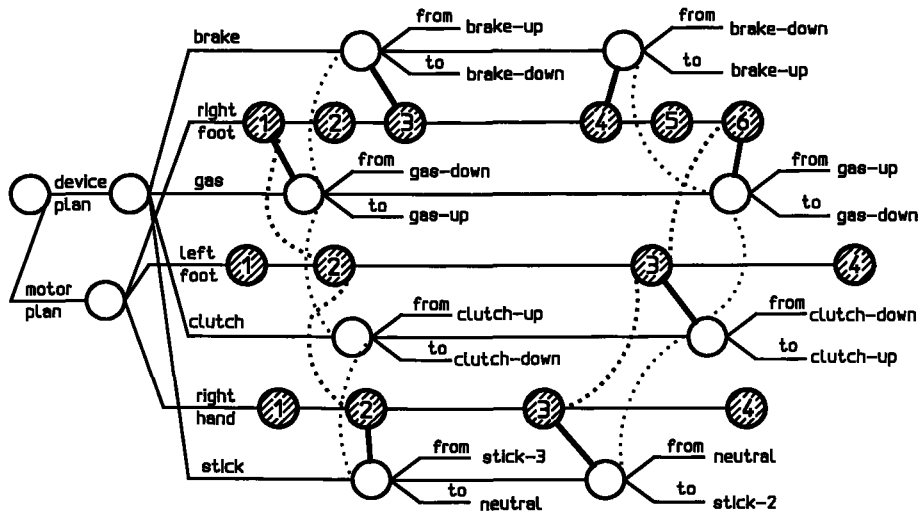


Figure 8-2. Soar representation of motor plan and device plan. The motor plan has been superimposed on the device plan from Figure 8-1. The dark nodes are the motor actions. As in the device plan all motor actions are attached to the motor plan but this is not shown in order to avoid a cluttered picture. The dark nodes that are connected to white nodes are motor actions that perform the related device actions. Dark nodes not connected are free moves between devices. Thus for the right foot we have (1) move right foot up, (2) move it from the accelerator pedal to the brake, (3) press brake, (4) release brake, (5) move foot back to accelerator and (6) press accelerator. For the left foot we have (1) move foot off floor, (2) press clutch and (3) and (4) are the reverse. For the right arm we have (1) move hand from steering wheel to stick, (2) move stick to neutral, (3) move stick to stick-2 and (4) move stick back to steering wheel.

The application of the above rules generates the appropriate motor actions but the ordering between the actions is unspecified¹. The ordering method used in DRIVER is to simulate the motor plan by applying the move operators internally. First the preferences between operators are generated on the basis of (1) existing next relations between motor actions and (2) the order con-

straints between device actions that the motor actions point to. The result of the (internal) application of a move operator is that a motor action on the move operator is attached to the appropriate extremity link on state. In addition, all the motor actions that are already on the state and that pertain to the same extremity as the motor action on the current move operator receive a *next* link to that motor action. In addition to *next* links, the ordering rules also specify *wait-for* and *together-with* relations between motor actions that reside on different extremities.

Note that there are thus two uses for move operators. The previous chapter showed that the application of a move operator at the top level resulted in the execution of a motor command in the external world. The application in the motor actions space is referred to as 'internal application' because the result of the application is not the execution of a move but the building of a plan. Figure 8-2 shows the complete motor plan and its relation to the device plan. The most important ordering relations between these actions are also depicted.

8.2.4 Learning the device plan and motor plan

After the device plan and motor plan have been created the motor actions and device actions are transferred to the top state and thereby chunked. The process of transferring these actions is standard Soar practice. First the motor actions are transferred to the higher change-gear space. The copies are then copied onto a newly created symbol that represents the motor-plan list. In the same space the device actions are copied onto a newly created symbol representing the device plan. Finally, both plans are copied up to the top state by a single production that tests for the existence of both plans. This ensures that both plans will end up in one chunk that represents an integrated body and device plan. Table 3 shows the chunk learned at the top level. The *action or then* part of the rule is rather lengthy but whether that is good or bad is discussed in the concluding chapter. The reader will see the overall structure from Figure 8-2 in the action part of the chunk. Note that the condition side of the rule contains both references to the change-gear operator and the position of the extremities and indirectly the state of the car. This chunk will fire when the right hand is on the wheel, the left foot on the floor and the right foot on the accelerator pedal. The reason that the extremities, and indirectly the state of the car, show up in the left-hand side of the rule is that in building the device plan it was necessary to test for the current state of the device. Trace 1 finally displays the entire problem-solving episode.

Table 3. The chunk that generates the motor-plan and device-plan

```

[sp p137
(goal <g1> ^object nil ^problem-space <p1> ^state <s1> ^operator <o1>)
(operator <o1> ^name change-gear ^use-brake t ^from stick3 ^to stick2)
(state <s1> ^motor-feedback <m1>)
(o <m1> ^rightfoot <r2> ^leftfoot <l1> ^righthand <r1>)
(o <r2> ^finished-move t ^to accelerator-down)
(o { <> <r2> <l1> } ^finished-move t)
(o <l1> ^to floor)
(o { <> <l1> <> <r2> <r1> } ^finished-move t)
(o <r1> ^to wheel)
->
(state <s1> ^vehicle-plan <v9> ^body-plan <b14>)
(o <b13> ^object rightfoot ^vehicle-action <v8> ^from accelerator-up
    ^to brake-up ^movement-type free ^after <b12> ^next <b11>)
(o <b10> ^object rightfoot ^vehicle-action <v7> ^from brake-down
    ^to brake-up ^movement-type apply-force ^after <b11> ^next <b9>)
(o <b8> ^object rightfoot ^vehicle-action <v6> ^from accelerator-up
    ^to accelerator-down ^movement-type apply-force ^after <b9>)
(o <b7> ^object leftfoot ^vehicle-action <v5> ^from clutch-up
    ^to clutch-down ^movement-type apply-force ^after <b6> ^next <b5>)
(o <b4> ^object righthand ^vehicle-action <v4> ^from wheel
    ^to stick3 ^movement-type free ^next <b3>)
(o <b2> ^object righthand ^vehicle-action <v3> ^from neutral
    ^to stick2 ^movement-type apply-force ^after <b3> ^next <b1>)
(o <b14> ^body-action <b12> <b8> <b5> <b1> <b9> <b2> <b3> <b10> <b4>
    <b11> <b7> <b13> <b6>)
(o <v7> ^object brake-pedal ^from brake-down ^to brake-up
    ^after <v8> ^next <v6> ^general-before <v6>)
(o <v3> ^object stick ^from neutral ^to stick2 ^after <v4> ^next <v2> <v6>
    ^general-before <v6> <v2>)
(o <v4> ^object stick ^from stick3 ^to neutral ^next <v3>
    ^general-before <v3> <v6> <v2> ^after <v5>)
(o <v8> ^object brake-pedal ^from brake-up ^to brake-down ^next <v7>
    ^general-before <v7> <v6> ^after <v1>)
(o <v9> ^vehicle-action <v1> <v6> <v2> <v5> <v4> <v3> <v8> <v7> <v3>)
(o <v5> ^after <v1> ^general-before <v2> <v6> <v3> <v4> <v2> ^next <v4> <v4> &
    ^to clutch-down ^from clutch-up ^object clutch-pedal)
(o <v2> ^after <v3> ^while <v6> ^to clutch-up ^from clutch-down
    ^object clutch-pedal)
(o <v6> ^after <v7> <v3> ^while <v2> ^to accelerator-down ^from accelerator-up ^object accelerator-pedal)
(o <v1> ^general-before <v7> <v6> <v2> <v4> <v3> <v8> <v5> <v7> ^next <v8> <v5>
    ^to accelerator-up ^from accelerator-down ^object accelerator-pedal)
(o <b1> ^after <b2> ^movement-type free ^to wheel
    ^from stick2 ^vehicle-action <v3> ^object righthand)
(o <b3> ^next <b2> ^after <b4> ^movement-type apply-force ^to neutral ^from stick3
    ^vehicle-action <v4> ^object righthand)
(o <b5> ^after <b7> ^movement-type apply-force ^to clutch-up
    ^from clutch-down ^vehicle-action <v2> ^object leftfoot)
(o <b6> ^next <b7> ^movement-type free ^to clutch-up
    ^from floor ^vehicle-action <v5> ^object leftfoot)
(o <b9> ^next <b8> ^after <b10> ^movement-type free
    ^to accelerator-up ^from brake-up ^vehicle-action <v6> ^object rightfoot)
(o <b11> ^next <b10> ^after <b13> ^movement-type apply-force ^to brake-down
    ^from brake-up ^vehicle-action <v8> ^object rightfoot)
(o <b12> ^next <b13> <b13> & ^movement-type apply-force ^to accelerator-up
    ^from accelerator-down ^vehicle-action <v1> ^object rightfoot))

```

Trace 1. A trace of the entire problem-solving episode and the start of the execution. The text between parenthesis are part of the Soar trace and indicate the names of states or operators. Our explanatory comments are in italics.

0 G: G1
1 P: P2 (BASE LEVEL SPACE)
2 S: S3 (BASE LEVEL STATE)
*We start DRIVER in a configuration where it is close to the intersection.
The Base Level Space P2 and S3 are installed .*
3 O: O29 (EYE-MOVEMENT-COMMAND)
DRIVER sees a car approaching from the right
4 O: O39 (HEAD-MOVEMENT-COMMAND)
Then head follows eyes
5 O: O33 (CHANGE-GEAR)
and DRIVER proposes operator O33 to reduce speed from 3rd to 2nd gear
6 --> G: G42 (OPERATOR NO -CHANGE)
however, driver does not know how to execute this operator and thus goes in a subgoal (note that depth of indentation shows the depth of the goalstack)
7 P: P43 (CHANGE-GEAR)
in this subgoal the problem space Change Gears is proposed
8 S: S44
9 O: O47 (DEVICE-PLAN)
DRIVER builds the device-plan in one decision cycle (see section 8.2.2)
10 O: O52 (GET-MOTOR-ACTIONS)
and then proceeds with the motor-plan (see section 8.2.3)
11 --> G: G57 (OPERATOR NO -CHANGE)
note that Soar now goes again a level deeper to attach motor actions to the motor-plan.
12 P: P58 (GET-MOTOR-ACTIONS)
13 S: S59
14 O: O61 (INIT-BODY-AND-CAR) ;
*The following operators are generated immediately after o61 is applied and
the gear-change operation is internally executed. Note that ACC stands for ACCELERATOR.*
15 O: O63 (MOTOR-COMMANDS) move RIGHTFOOT from ACC -DOWN to ACC-UP
16 O: O65 (MOTOR-COMMANDS) move LEFTFOOT from FLOOR to CLUTC H-UP
17 O: O67 (MOTOR-COMMANDS) move RIGHTFOOT from ACC -UP to BRAKE-UP
18 O: O69 (MOTOR-COMMANDS) move LEFTFOOT from CLUTC H-UP to CLUTCH-DOWN
19 O: O71 (MOTOR-COMMANDS) move RIGHTFOOT from BRAK E-UP to BRAKE-DOWN
20 O: O73 (MOTOR-COMMANDS) move RIGHTHAND from WHEEL to STICK3
21 O: O75 (MOTOR-COMMANDS) move RIGHTFOOT from BRAK E-DOWN to BRAKE-UP
22 O: O77 (MOTOR-COMMANDS) move RIGHTHAND from STICK3 to NEU TRAL
23 O: O79 (MOTOR-COMMANDS) move RIGHTFOOT from BRAK E-UP to ACC-UP
24 O: O81 (MOTOR-COMMANDS) move RIGHTHAND from NEU TRAL to STICK2
25 O: O83 (MOTOR-COMMANDS) move RIGHTFOOT from ACC -UP to ACC-DOWN
26 O: O85 (MOTOR-COMMANDS) move RIGHTHAND from STICK2 to WHEEL
27 O: O87 (MOTOR-COMMANDS) move LEFTFOOT from CLUTC H-DOWN to CLUTCH-UP
28 O: O62 (COPY-UP-MOTOR-ACTIONS)
Building:P89 to P101
29 O: O102 (MOTOR-ACTIONS-BACK-TO-TOP-STATE)
Build:P105 to p115
*The desired chunk is learned (See Section 8.2.4). Immediately after the plan lands on the top state DRIVER proceeds with the
execution of the plan (see section 8.2.5).*
30 O: O132 (M134 (MOTOR-COMMANDS) external move RIGHTHAND and LEFTHAND on WHEEL
*Before DRIVER starts with the execution of the learned plan it first make a small steering correction because during problem-solving
DRIVER drifted off course.*
After this correction it proceeds with the plan by lifting its right foot off the accelerator pedal

31 O: 0119 (M123 MOTOR-COMMANDS) external move RIGHTFOOT from ACC -DOWN to ACC -UP
 Moving RIGHTFOOT from m ACC-DOWN to ACC-UP, dtm = 60, dm = 15
 Moving RIGHTFOOT from ACC -DOWN to ACC-UP, dtm = 60, dm = 30

32 O: 0120 (M125 MOTOR-COMMANDS) external move LEFTFOOT from FLOOR to CLUTCH -UP ;
Note how the command to move the left foot up is given while the right foot is still moving.
 Moving RIGHTFOOT from ACC -DOWN to ACC-UP, dtm = 60, dm = 45
 Moving RIGHTFOOT from ACC -DOWN to ACC-UP, dtm = 60, dm = 60
 Moving LEFTFOOT from FLOOR to CLUTCH -UP, dtm = 100, dm = 30
 Moving LEFTFOOT from FLOOR to CLUTCH -UP, dtm = 100, dm = 60
 Moving LEFTFOOT from FLOOR to CLUTCH -UP, dtm = 100, dm = 90

33 O: 0137 (M138 MOTOR-COMMANDS) external move RIGHTFOOT from ACC -UP to BRAKE-UP
 Moving LEFTFOOT from FLOOR to CLUTCH -UP, dtm = 100, dm = 100
 Moving RIGHTFOOT from ACC -UP to BRAKE-UP, dtm = 100, dm = 30

8.2.5 Execution

The plan is finished and DRIVER can start to execute it. DRIVER's first action is to generate a move operator for the first motor action of each extremity. However, in order to avoid the untimely execution of motor actions, its second action is to hold the execution of these operators. Table 4 shows a sample of rules involved in the control of the hold mechanism.

Table 4. The hold and timing mechanism.

[10]	If	move operator O with motor-action MA and MA is an apply-force movement
	Then	add Hold = True to O
[11]	If	move operator O1 with motor-action MA1 and Hold = True move operator O2 with motor-action MA2 MA1 points to device-action VA1 MA2 points to device-action VA2 VA1 has a general-before pointer to VA2
	Then	add Hold = True to O2
[12]	If	move operator O with Hold = True
	Then	prohibit move operator O
[13]	If	move-operator O with motor-action MA and Hold = True motor-action MA issues move right foot up (release accelerator) approaching an intersection intended manoeuvre is right turn time to intersection < 7
	Then	remove Hold = True on O

Rule [10] ensures that all motor-actions that manipulate a device are held. Goto actions (*movement-type* is *free*) are in general always allowed as long as they do not conflict with the device-plan; rule [11] makes sure of this. Rule [12] is the general rule that prohibits operators that have a hold. Rule [13] is a

typical example of timing knowledge that removes a hold so that the operator can be applied and the command issued.

The first question to ask here is why is it necessary to have such a complex *hold mechanism*? The obvious reason for a hold mechanism is that it prevents the untimely execution of plans. Remember that the motor-plan and device-plan are finished before the execution. It is important that the first actions of the plan are executed right on time but it is just as important to time the subsequent actions. For example, the motor-plan does not specify that DRIVER should keep its right foot on the brake for about three seconds. Without the hold mechanism the brake would be released directly after it was pressed in. A second, but equally important reason for the indirect hold mechanism, however, is that it provides a handle for learning by correction (see next section).

A trace of the execution

Trace 2 shows a Soar trace of DRIVER executing the motor-plan. When the change-gear operator is issued the appropriate chunk fires and adds the motor-plan and device-plan to the top-state. The motor-command operators are identical to the move operators (as they have been called in this and the previous chapter.) The comments after the motor-command operators indicate what motor-action is being issued. The comments with the format "Moving RIGHTFOOT from ACC-DOWN to ACC-UP, dtm = 60, dm = 30" are debugging traces from the LLMM. *DTM* stands for the relative distance to move on the accelerator pedal, whereas *DM* is the distance moved so far. The *wait* operator in the trace is the default operator that DRIVER applies when there is nothing else to do. There are many things for DRIVER to do when approaching an intersection but in order to demonstrate the time and number of operators taken up by the execution of the motor-plan we have repressed the navigation task and the eye and head movements. The trace illustrates several issues that have come up in this and the previous chapter.

- *Extremities move asynchronously from internal Soar processes:* Some movements are finished within a decision cycle, for example the release of the accelerator pedal in D6 (Decision cycle number six) or the steering command in D7. The brake-down command issued in D10 is an example of a move that takes two decision cycles to finish. The clutch-down clutch-up command in D83 is an extreme example of a command that is executed over several decision cycles.
- *Multiple extremities moving in parallel:* There are many examples in the trace that demonstrate this. D85 provides a nice example of DRIVER steering while at the same time releasing the clutch pedal and pressing the accelerator pedal.

- *Multiple motor-actions per operator:* The trace provides several examples of DRIVER issuing a right-hand and left-hand steering action with the same operator.
- *Other motor-actions possible during plan execution:* Eye-move commands, head-move and steering commands may be issued while executing the plan. The execution of the gear-change plan does not prevent other extremities from acting. The trace provides some examples where steering interrupts the plan. In a trace elaborated in more detail in the chapter on multitasking (Chapter 15) we see how eye and head movements may also interrupt the execution of the gear-change plan.

During the entire gear-change operation only 15 percent of the operators are dedicated to execution of the motor-plan: the trace shows that only 14 motor commands related to gear-changing were issued. Of the 91 operators shown in the trace, 76 are wait operators. The longest wait period is 35 wait operators. This indicates that even during a complex task such as changing gear, DRIVER can spend nearly 85 percent of its precious decision cycles (and thus operators) on other tasks. The rather obvious implications of this for multitasking are discussed in the chapters on navigation and multitasking (Chapters 13 and 14).

8.2.6 Learning timing knowledge by correction

The two important things to be learned when fine-tuning plan execution are (1) the *when* of removing a *hold* and (2) the *degree* to which a device must be pressed in or released. In the current state DRIVER is not well equipped to fine-tune behaviour because the problem of learning from external interaction is still an unresolved research issue in the Soar community. Appendix 1 and the final chapter of this study deal with this issue more extensively and offer general solutions to this problem. Despite this external learning problem this section elaborates a little further on the problem of fine-tuning.

Let us first consider the *when* of removing a hold. For example, DRIVER has been instructed to release the brake closer to the intersection. In DRIVER's terms this implies that the hold should be removed later. In order to produce this new behaviour DRIVER must learn two things. First it must prevent the current production that removes the hold (hereafter called the *remover*) from firing. As productions themselves cannot be changed the solution to this problem is to add yet another indirection to the hold mechanism: *Remover* productions no longer remove the holds but add a remove object to the move operator. The remove objects are made unique by annotating them with a unique *^id* attribute. Table 5 shows how rule [13a] (compare this with rule [13] in Table 4) now adds such a structure instead of removing the hold. One other attribute that this remove object may have is a so-called *^invalid* attribute. The function of this attribute is to prevent the removal of a hold. An example of a rule that makes a removal invalid is rule [14]. Rule [12a] shows

how the removal of a hold is prevented. Holds are now only removed when operator *O* has no remove objects that have an *invalid* attribute. Rule [13b] shows an example of the new rule that DRIVER must learn in order to release the brake at a later time and the bottom of the table shows the structures involved in this alternative hold mechanism.

Table 5. The numbers in the left margin refer to the numbers in Table 4. The italics indicate the changes.

[12a]	If	move operator <i>O</i> with Hold-True <i>operator O has no remove object with invalid attribute</i>
	Then	prohibit move operator <i>O</i>
[13a]	If	move-operator <i>O</i> with motor-action <i>MA</i> and Hold-True motor-action <i>MA</i> issues move right foot up (release accelerator) approaching an intersection intended manoeuvre is right turn time to intersection < 7
	Then	<i>add a remove object with id=new-symbol to MA</i>
[13b]	If	move-operator <i>O</i> with motor-action <i>MA</i> and Hold-True motor-action <i>MA</i> issues move right foot up (release accelerator) approaching an intersection intended manoeuvre is right turn <i>time to intersection < 5</i>
	Then	add a remove object with id=X1000 to <i>MA</i>
[14]	If	move-operator <i>O</i> with motor-action <i>MA</i> and a remove object with id=X999
	Then	annotate remove object with an invalid attribute
(operator <o> ^name move ^body-action <ba> ^hold-true ^remove <r1> <r2>) (remove <r1> ^id X999 ^invalid T) (remove <r2> ^id X1000)		

One advantage of the scheme is that this way the holds may be changed many times, even back to the original rule [13] (from Table 4) where *time to intersection* < 7. This advantage is one of the justifications of the use of such an indirect hold mechanism. If Soar preferences were being used directly such a mechanism would be entirely impossible.

The second thing for DRIVER to learn is the new conditions under which this hold should be removed. See for example rule [13b] in Table 5. However, this requires learning from external interaction and is therefore a serious problem in Soar. Remember that motor actions are carried out in real time at top level and thus nothing can be learned by chunking. The only way out here is to use data-chunking so that instructions (from an instructor) or evaluations of actions (possibly acquired by problem-solving) are transformed into new remove rules. The discussion on how to do data-chunking and error-correction by data-chunking will be taken up in the final chapter of this study.

8.3 Discussion

This chapter has demonstrated that we have succeeded in getting DRIVER to learn the right plans and execute them successfully in the “external” world. We have focused almost completely on the mechanisms and postponed a discussion of the implications of various design choices. We shall discuss a few of these choices below.

8.3.1 Designing the plans

One of the interesting aspects of DRIVER’s motor planning is that the constraints provided by the car mechanisms and the body are sufficient to build adequate device and motor plans. From moving the gear-stick it follows that the clutch must be moved, from moving the clutch it follows that the accelerator must be released, etc. There is thus no need for a detailed conceptual model of the car engine, nor do we need step-by-step instructions. In other words, the type of knowledge that we have provided DRIVER with falls neither into the category of *operational* (how-to-do-the-task) knowledge, nor that of *figurative*, (how-the-device-works) knowledge (Halaz and Moran, 1983; Kieras and Bovair, 1984).

Another interesting aspect is the double use of the move operators. Their first use is in the internal simulation of the body plan. By applying the move operators internally, a body plan is built in the get-body-actions state. As we saw, this body plan consists of a complex hierarchy of motor-actions. The second use of the move operator is at execution time in the top state. The move operators are now generated from the motor-actions on the motor plan (in their first use they put motor-actions *on* the motor plan) and they will initiate an external motor action.

8.3.2 The execution of plans

Chapter 7 gave reasons for various design decisions without giving examples of how these decisions work out in real (Soar) life. The present chapter shows how these decisions work out in DRIVER. The traces of the execution of the motor-plan demonstrated (1) how extremities move asynchronously from other Soar processes, (2) how multiple extremities may move at the same time, (3) how multiple motor-actions per operator may be issued (that is, one intend for multiple actions) and (4) how interrupts from other actions, including motor-actions, must be possible during plan execution.

If we look to the large and complex tangled hierarchy of the device and motor plans one might ask whether the motor-plan doesn’t force DRIVER into a too fixed and rigid behaviour. The answer to that is a simple *no*. In the first place flexibility is provided by using the timing and hold mechanism. The same motor plans can be used in various situations (with respect to speed) just by

varying the release of hold structures on the operators or by varying the degree of how far a pedal is pressed in.

In the second place, other operators can be included in the motor plan. We have already noted how non-motor actions can easily be involved in the execution of the motor plan. The execution trace in this chapter shows numerous wait operators. Each wait operator can in principle be replaced by any other type of operator. In addition, whenever a non-motor action is given a higher priority than a motor action the non-motor action will be carried out and the execution of the motor plan can proceed. However, in the latter case, it is possible that if too many other operators have a higher preference than the motor operators, the timing of the motor-plan will be endangered. In the current version of DRIVER this hardly ever happens because the motor plan is given a high preference.

Including other motor-actions in the actions in the motor-plan is somewhat harder. However, the motor-plan is a list structure that can be manipulated at will. For example, if during the execution of a motor-plan another, more important, motor plan is proposed, the older motor-plan may be replaced with the new one.

However, manipulation of the list (which comprises the motor plan) is also possible. Motor-actions may be inserted, removed or replaced. This chapter demonstrated that motor-actions for turning the steering wheel are regularly inserted in the current motor plan. The chapter on speed control shows how pressing the accelerator a little further is an extra motor action that may be inserted in the current motor-plan.

8.3.3 Learning issues

The first interesting observation with respect to learning is that the rules that are learned include references to the current configuration of the car and body. Problem-solving in gear-changing is not a purely internal activity, but requires reference to the external world. Table 3 provided a detailed account of these references of which the general format is the following:

If	my body is in configuration BC and my car is in configuration CC and I need to change gear from XX to YY
Then	propose device plan D and motor plan M.

One question that might be asked is whether DRIVER must learn a new motor plan for every body configuration and for every combination of gear states? The answer to this question is *yes*, though during the problem-solving episode some chunks are learned that transfer to other plans.

The size of the motor-plan chunk is another issue that deserves some attention. The chunk that is currently being built seems overwhelmingly large (in terms of number of working memory elements), though it is also true that we still do not know the chunk size in humans. In the final discussion of this study we will return to the issue of chunk size and working memory size. One advantage of using smaller chunks that divide the motor-plan into smaller motor-plans is that the modularity of learning might be enhanced. For example, it seems more efficient to learn motor-plans for gear-stick actions (and related accelerator and clutch actions) from stick-X to stick-neutral and motor-plans for stick-neutral to stick-Y. Although we didn't follow this approach in DRIVER, it is a suggestion for further research.

In general the whole approach to motor handling is characterised by the number of indirections used. In the get-body-action space we have motor-actions on move operators instead of motor actions directly on the state, in the top level state we have motor-actions that generate operators instead of operators being generated directly and finally we have an indirect hold mechanism in timing. A general property of using indirection is that it slows a system down. However, in DRIVER fast moving is still possible because (a) motor handling is divided into planning and execution and (b) during execution operators are generated when other motor-actions are still performing, in other words preparation is made possible.

Trace 2. A trace of the execution.

0 G: G1
1 P: P2
2 S: S3
3 O: 029 (EYE-MOVEMENT-COMMAND)
DRIVER spots a car coming from the left
4 O: 039 (HEAD-MOVEMENT-COMMAND)
and turns its head to see it even better
5 O: 033 (CHANGE-GEAR)
DRIVER decides to change gear
Firing : p137
and immediately the chunk that adds the motor-plan fires.
6 O: 071 (M74 MOTOR-COMMANDS) moves RIGHTFOOT from ACC -DOWN to ACC -UP
The plan starts by moving the accelerator up
 Moving RIGHTFOOT from ACC -DOWN to ACC -UP, dtm = 60, dm = 15
The distance to move (dtm) is 60 % of the total accelerator range, and 15 % has been completed.
 Moving RIGHTFOOT from ACC -DOWN to ACC -UP, dtm = 60, dm = 30
and 30 % has been completed
 Moving RIGHTFOOT from ACC -DOWN to ACC -UP, dtm = 60, dm = 45
etc.
 Moving RIGHTFOOT from ACC -DOWN to ACC -UP, dtm = 60, dm = 60
7 O: 088 (M90 MOTOR-COMMANDS) 088 moves RIGHTHAND and LEFTHAND from WHEEL to WHEEL
Here the plan is temporarily interrupted by a steering-wheel correction
 Moving LEFTHAND from WHEEL to WHEEL, dtm = 6, dm = 6
 Moving RIGHTHAND from WHEEL to WHEEL, dtm = 6, dm = 6

8 O: 072 (M76 MOTOR-COMMANDS) move LEFTFOOT from FLOOR to CLUTCH -UP

The clutch must go down so the left foot is moved to the clutch.

Moving LEFTFOOT from FLOOR to CLUTCH -UP, dtm = 100, dm = 30

Moving LEFTFOOT from FLOOR to CLUTCH -UP, dtm = 100, dm = 60

Moving LEFTFOOT from FLOOR to CLUTCH -UP, dtm = 100, dm = 90

9 O: 081 (M82 MOTOR-COMMANDS) moves RIGHTFOOT from ACC -UP to BRAKE-UP

Moving LEFTFOOT from FLOOR to CLUTCH -UP, dtm = 100, dm = 100

Note that a) the command to move the rightfoot up is given before the previous move has finished and b) the left foot is still moving as the right foot travels from accelerator to brake

Moving RIGHTFOOT from ACC -UP to BRAKE-UP, dtm = 100, dm = 30

Moving RIGHTFOOT from ACC -UP to BRAKE-UP, dtm = 100, dm = 60

Moving RIGHTFOOT from ACC -UP to BRAKE-UP, dtm = 100, dm = 90

Moving RIGHTFOOT from ACC -UP to BRAKE-UP, dtm = 100, dm = 100

10 O: 096 (M97 MOTOR-COMMANDS) moves RIGHTFOOT from BRAKE -UP to BRAKE-DOWN

Moving RIGHTFOOT from BRAKE -UP to BRAKE-DOWN, dtm = 70, dm = 15

Moving RIGHTFOOT from BRAKE -UP to BRAKE-DOWN, dtm = 70, dm = 30

Moving RIGHTFOOT from BRAKE -UP to BRAKE-DOWN, dtm = 70, dm = 45

11 O: 093 (M94 MOTOR-COMMANDS) moves LEFTFOOT from CLUTCH-UP to CLUTCH-DOWN

Here is even a better example of a motor command that is issued and started right in the middle of another move.

Moving RIGHTFOOT from BRAKE -UP to BRAKE-DOWN, dtm = 70, dm = 60

Moving RIGHTFOOT from BRAKE -UP to BRAKE-DOWN, dtm = 70, dm = 70

Moving LEFTFOOT from CLUTCH -UP to CLUTCH-DOWN, dtm = 60, dm = 15

Moving LEFTFOOT from CLUTCH -UP to CLUTCH-DOWN, dtm = 60, dm = 30

Moving LEFTFOOT from CLUTCH -UP to CLUTCH-DOWN, dtm = 60, dm = 45

Moving LEFTFOOT from CLUTCH -UP to CLUTCH-DOWN, dtm = 60, dm = 60

12 O: 073 (M78 MOTOR-COMMANDS) moves RIGHTHAND from WHEEL to STICK3

Note that it takes four operators to move from wheel to stick3 to neutral to stick2 to wheel.

Moving RIGHTHAND from WHEEL to STICK3, dtm = 100, dm = 30

Moving RIGHTHAND from WHEEL to STICK3, dtm = 100, dm = 60

Moving RIGHTHAND from WHEEL to STICK3, dtm = 100, dm = 90

Moving RIGHTHAND from WHEEL to STICK3, dtm = 100, dm = 100

13 O: 0105 (M106 MOTOR-COMMANDS) moves RIGHTHAND from STICK3 to NEUTRAL

Moving RIGHTHAND from STICK3 to NEUTRAL, dtm = 100, dm = 30

Moving RIGHTHAND from STICK3 to NEUTRAL, dtm = 100, dm = 60

Moving RIGHTHAND from STICK3 to NEUTRAL, dtm = 100, dm = 90

Moving RIGHTHAND from STICK3 to NEUTRAL, dtm = 100, dm = 100

14 O: 0108 (M109 MOTOR-COMMANDS) moves RIGHTHAND from NEUTRAL to STICK2

Moving RIGHTHAND from NEUTRAL to STICK2, dtm = 100, dm = 30

Moving RIGHTHAND from NEUTRAL to STICK2, dtm = 100, dm = 60

15 O: 0111 (M112 MOTOR-COMMANDS) moves RIGHTHAND from STICK2 to WHEEL

Moving RIGHTHAND from NEUTRAL to STICK2, dtm = 100, dm = 90

Moving RIGHTHAND from NEUTRAL to STICK2, dtm = 100, dm = 100

Moving RIGHTHAND from STICK2 to WHEEL, dtm = 100, dm = 30

16 O: 034 (WAIT)

DRIVER has nothing more to do, even though the right hand is still moving.

Moving RIGHTHAND from STICK2 to WHEEL, dtm = 100, dm = 60

Moving RIGHTHAND from STICK2 to WHEEL, dtm = 100, dm = 90

17 O: 034 (WAIT)

Moving RIGHTHAND from STICK2 to WHEEL, dtm = 100, dm = 100

In the next period we see only wait operators. Normally we never see this operator in DRIVER but in order to demonstrate the execution of the gear plan we disabled eye moves and navigation.

18 O: 034 (WAIT)

19 O: 034 (WAIT)

20 O: 034 (WAIT)

21 O: 034 (WAIT)
22 O: 034 (WAIT)
23 O: 034 (WAIT)
24 O: 034 (WAIT)
25 O: 034 (WAIT)
26 O: 034 (WAIT)
27 O: 034 (WAIT)
28 O: 034 (WAIT)
29 O: 034 (WAIT)
30 O: 034 (WAIT)
31 O: 034 (WAIT)
32 O: 034 (WAIT)
33 O: 034 (WAIT)
34 O: 034 (WAIT)
35 O: 099 (M100 MOTOR-COMMANDS) moves RIGHTFOOT from BRAKE -DOWN to BRAKE -UP
Speed has been reduced enough and DRIVER can release the brake again
 Moving RIGHTFOOT from BRAKE -DOWN to BRAKE -UP, dtm = 70, dm = 15
 Moving RIGHTFOOT from BRAKE -DOWN to BRAKE -UP, dtm = 70, dm = 30
36 O: 0114 (M115 MOTOR-COMMANDS) moves RIGHTFOOT from BRAKE -UP to ACC-UP
 Moving RIGHTFOOT from BRAKE -DOWN to BRAKE -UP, dtm = 70, dm = 45
 Moving RIGHTFOOT from BRAKE -DOWN to BRAKE -UP, dtm = 70, dm = 60
 Moving RIGHTFOOT from BRAKE -DOWN to BRAKE -UP, dtm = 70, dm = 70
 Moving RIGHTFOOT from BRAKE -UP to ACC-UP, dtm = 100, dm = 30
37 O: 034 (WAIT)
 Moving RIGHTFOOT from BRAKE -UP to ACC-UP, dtm = 100, dm = 60
 Moving RIGHTFOOT from BRAKE -UP to ACC-UP, dtm = 100, dm = 90
38 O: 034 (WAIT)
 Moving RIGHTFOOT from BRAKE -UP to ACC-UP, dtm = 100, dm = 100
39 O: 034 (WAIT)
.....
.....
To save space we edited out 23 Wait operators
.....
.....
72 O: 034 (WAIT)
73 O: 0125 (M127 MOTOR-COMMANDS) moves RIGHTHAND and LEFTHAND from WHEEL to WHEEL
DRIVER is approaching the bend, so turning begins
 Moving LEFTHAND from WHEEL to WHEEL, dtm = 9, dm = 6
 Moving RIGHTHAND from WHEEL to WHEEL, dtm = 9, dm = 6
 Moving LEFTHAND from WHEEL to WHEEL, dtm = 9, dm = 9
 Moving RIGHTHAND from WHEEL to WHEEL, dtm = 9, dm = 9
74 O: 034 (WAIT)
75 O: 034 (WAIT)
76 O: 034 (WAIT)
77 O: 034 (WAIT)
78 O: 034 (WAIT)
79 O: 034 (WAIT)
80 O: 034 (WAIT)
81 O: 034 (WAIT)
82 O: 034 (WAIT)
83 O: 0102 (M103 MOTOR-COMMANDS) moves LEFTFOOT from CLUTCH -DOWN to CLUTCH -UP
 Moving LEFTFOOT from CLUTCH -DOWN to CLUTCH -UP, dtm = 60, dm = 3/4
 Moving LEFTFOOT from CLUTCH -DOWN to CLUTCH -UP, dtm = 60, dm = 3/2
 Moving LEFTFOOT from CLUTCH -DOWN to CLUTCH -UP, dtm = 60, dm = 9/4

- Moving LEFTFOOT from CLUTCH -DOWN to CLUTCH -UP, dtm = 60, dm = 3
Moving LEFTFOOT from CLUTCH -DOWN to CLUTCH -UP, dtm = 60, dm = 15/4
Moving LEFTFOOT from CLUTCH -DOWN to CLUTCH -UP, dtm = 60, dm = 9/2
Moving LEFTFOOT from CLUTCH -DOWN to CLUTCH -UP, dtm = 60, dm = 21/4
Moving LEFTFOOT from CLUTCH -DOWN to CLUTCH -UP, dtm = 60, dm = 6
Moving LEFTFOOT from CLUTCH -DOWN to CLUTCH -UP, dtm = 60, dm = 27/4
- 84 O: 0135 (M137 MOTOR-COMMANDS) moves RIGHTHAND from WHEEL to WHEEL
Moving LEFTFOOT from CLUTCH -DOWN to CLUTCH -UP, dtm = 60, dm = 15/2
Moving LEFTFOOT from CLUTCH -DOWN to CLUTCH -UP, dtm = 60, dm = 33/4
Moving LEFTFOOT from CLUTCH -DOWN to CLUTCH -UP, dtm = 60, dm = 9
Moving LEFTHAND from WHEEL to WHEEL, dtm = 33, dm = 6
Moving RIGHTHAND from WHEEL to WHEEL, dtm = 33, dm = 6
Moving LEFTFOOT from CLUTCH -DOWN to CLUTCH -UP, dtm = 60, dm = 39/4
Moving LEFTHAND from WHEEL to WHEEL, dtm = 33, dm = 12
Moving RIGHTHAND from WHEEL to WHEEL, dtm = 33, dm = 12
- 85 O: 0117 (M118 MOTOR-COMMANDS) moves RIGHTFOOT from GAS -UP to GAS-DOWN
And finally the last motor command
Moving LEFTFOOT from CLUTCH -DOWN to CLUTCH -UP, dtm = 60, dm = 21/2
Moving LEFTHAND from WHEEL to WHEEL, dtm = 33, dm = 18
Moving RIGHTHAND from WHEEL to WHEEL, dtm = 33, dm = 18
Moving LEFTFOOT from CLUTCH -DOWN to CLUTCH -UP, dtm = 60, dm = 45/4
Moving LEFTHAND from WHEEL to WHEEL, dtm = 33, dm = 24
Moving RIGHTHAND from WHEEL to WHEEL, dtm = 33, dm = 24
Moving LEFTFOOT from CLUTCH -DOWN to CLUTCH -UP, dtm = 60, dm = 12
Moving RIGHTFOOT from GAS -UP to GAS-DOWN, dtm = 60, dm = 3
Moving LEFTHAND from WHEEL to WHEEL, dtm = 33, dm = 30
Moving RIGHTHAND from WHEEL to WHEEL, dtm = 33, dm = 30
Moving LEFTFOOT from CLUTCH -DOWN to CLUTCH -UP, dtm = 60, dm = 51/4
Moving RIGHTFOOT from GAS -UP to GAS-DOWN, dtm = 60, dm = 6
Moving LEFTHAND from WHEEL to WHEEL, dtm = 33, dm = 33
Moving RIGHTHAND from WHEEL to WHEEL, dtm = 33, dm = 33
Moving LEFTFOOT from CLUTCH -DOWN to CLUTCH -UP, dtm = 60, dm = 27/2
Moving RIGHTFOOT from GAS -UP to GAS-DOWN, dtm = 60, dm = 9
Moving LEFTFOOT from CLUTCH -DOWN to CLUTCH -UP, dtm = 60, dm = 57/4
Moving RIGHTFOOT from GAS -UP to GAS-DOWN, dtm = 60, dm = 12
- 86 O: 034 (WAIT)
Moving LEFTFOOT from CLUTCH -DOWN to CLUTCH -UP, dtm = 60, dm = 15
Moving RIGHTFOOT from GAS -UP to GAS-DOWN, dtm = 60, dm = 15
Moving LEFTFOOT from CLUTCH -DOWN to CLUTCH -UP, dtm = 60, dm = 63/4
Moving RIGHTFOOT from GAS -UP to GAS-DOWN, dtm = 60, dm = 18
- 87 O: 034 (WAIT)
Moving LEFTFOOT from CLUTCH -DOWN to CLUTCH -UP, dtm = 60, dm = 33/2
Moving RIGHTFOOT from GAS -UP to GAS-DOWN, dtm = 60, dm = 21
Moving LEFTFOOT from CLUTCH -DOWN to CLUTCH -UP, dtm = 60, dm = 69/4
Moving RIGHTFOOT from GAS -UP to GAS-DOWN, dtm = 60, dm = 24
- 88 O: 034 (WAIT)
Moving LEFTFOOT from CLUTCH -DOWN to CLUTCH -UP, dtm = 60, dm = 18
Moving RIGHTFOOT from GAS -UP to GAS-DOWN, dtm = 60, dm = 27
Moving LEFTFOOT from CLUTCH -DOWN to CLUTCH -UP, dtm = 60, dm = 75/4
Moving RIGHTFOOT from GAS -UP to GAS-DOWN, dtm = 60, dm = 30
- 89 O: 034 (WAIT)
Moving LEFTFOOT from CLUTCH -DOWN to CLUTCH -UP, dtm = 60, dm = 39/2
Moving RIGHTFOOT from GAS -UP to GAS-DOWN, dtm = 60, dm = 33
Moving LEFTFOOT from CLUTCH -DOWN to CLUTCH -UP, dtm = 60, dm = 81/4

Moving RIGHTFOOT from GAS -UP to GAS-DOWN, dtm = 60, dm = 36
Moving LEFTFOOT from CLUTCH -DOWN to CLUTCH-UP, dtm = 60, dm = 21
Moving RIGHTFOOT from GAS -UP to GAS-DOWN, dtm = 60, dm = 39
Moving LEFTFOOT from CLUTCH -DOWN to CLUTCH-UP, dtm = 60, dm = 87/4
Moving RIGHTFOOT from GAS -UP to GAS-DOWN, dtm = 60, dm = 42
Moving LEFTFOOT from CLUTCH -DOWN to CLUTCH-UP, dtm = 60, dm = 45/2
Moving RIGHTFOOT from GAS -UP to GAS-DOWN, dtm = 60, dm = 45
Moving LEFTFOOT from CLUTCH -DOWN to CLUTCH-UP, dtm = 60, dm = 93/4
Moving RIGHTFOOT from GAS -UP to GAS-DOWN, dtm = 60, dm = 48
Moving LEFTFOOT from CLUTCH -DOWN to CLUTCH-UP, dtm = 60, dm = 24
Moving RIGHTFOOT from GAS -UP to GAS-DOWN, dtm = 60, dm = 51
Moving LEFTFOOT from CLUTCH -DOWN to CLUTCH-UP, dtm = 60, dm = 99/4
Moving RIGHTFOOT from GAS -UP to GAS-DOWN, dtm = 60, dm = 54

90 0: 034 (WAIT)

Moving LEFTFOOT from CLUTCH -DOWN to CLUTCH-UP, dtm = 60, dm = 51/2
Moving RIGHTFOOT from GAS -UP to GAS-DOWN, dtm = 60, dm = 57
Moving LEFTFOOT from CLUTCH -DOWN to CLUTCH-UP, dtm = 60, dm = 105/4
Moving RIGHTFOOT from GAS -UP to GAS-DOWN, dtm = 60, dm = 60
Moving LEFTFOOT from CLUTCH -DOWN to CLUTCH-UP, dtm = 60, dm = 27
Moving LEFTFOOT from CLUTCH -DOWN to CLUTCH-UP, dtm = 60, dm = 111/4

91 0: 034 (WAIT)

Moving LEFTFOOT from CLUTCH -DOWN to CLUTCH-UP, dtm = 60, dm = 57/2
Moving LEFTFOOT from CLUTCH -DOWN to CLUTCH-UP, dtm = 60, dm = 117/4
Moving LEFTFOOT from CLUTCH -DOWN to CLUTCH-UP, dtm = 60, dm = 30
Moving LEFTFOOT from CLUTCH -DOWN to CLUTCH-UP, dtm = 60, dm = 123/4
Moving LEFTFOOT from CLUTCH -DOWN to CLUTCH-UP, dtm = 60, dm = 63/2
Moving LEFTFOOT from CLUTCH -DOWN to CLUTCH-UP, dtm = 60, dm = 129/4
Moving LEFTFOOT from CLUTCH -DOWN to CLUTCH-UP, dtm = 60, dm = 33
Moving LEFTFOOT from CLUTCH -DOWN to CLUTCH-UP, dtm = 60, dm = 135/4
Moving LEFTFOOT from CLUTCH -DOWN to CLUTCH-UP, dtm = 60, dm = 69/2

8.4 Notes

¹ I experimented with two methods to provide such an ordering. The first method was to put the motor- action objects directly onto the state and not onto the operators (as illustrated in the table). We then used production match only to generate the ordering: a small set of rules, propagating in a cascade of elaboration cycles, determined the ordering relations between the device-actions. The problem in this approach was that these rules provided a very unstable and complex coding. A far more reliable method was the use of operators and the preference mechanism to establish the ordering for motor actions.

9 Basic perception

Summary: This chapter discusses DRIVER's lower-level perception mechanisms. First some general constraints on human perception and problems with how Soar handles perception are discussed. Then we will turn to the actual implementation of DRIVER's basic perceptual mechanisms: object recognition, attention and basic eye and head movement control.

9.1 Introduction

Adaptive visual orientation – looking in the right direction at the right time – is probably the most difficult task in driving. This and the following chapter describe how DRIVER copes with the visual orientation task. In the present chapter we discuss DRIVER's basic perceptual mechanisms: object identification and basic eye and head movement control. In the following chapter we then describe how these mechanisms are used in the approach and negotiation of an intersection.

This chapter is arranged as follows. First we examine why the visual orientation task is in fact so difficult and we discuss some general requirements for a theory of visual orientation. Next we discuss the list of constraints that determine the design decisions in DRIVER's low-level perception. As in the previous chapters we make a distinction between a list of Soar constraints and a list of constraints from the (traffic) literature. The fourth section covers the implementation of the low-level perception module. The final section summarizes this chapter and discusses some of the implications of our design decisions.

9.2 The difficulty of the visual orientation task in driving

The claim that visual orientation is difficult has several bases. First, it is commonly estimated that over 90 % of the information input to the driver is visual (Hills, 1980). This does not prove that visual orientation is a difficult sub-task, but we may be sure that nearly all input handled by drivers is visual.

A second basis is obtained from research into perceptual errors and accidents in driving. Several large-scale investigations have shown that perceptual errors are the main contributing factor in the causation of accidents. One of the largest of these investigations was carried out by the TRRL 'On-the-Spot' team (Sabey and Staughton, 1975; Staughton and Storie, 1977; Storie 1977). These investigators visited the scenes of 2036 accidents. According to the authors, errors by all types of road users were contributing to 95 percent of the accidents, and 44 percent of 'car drivers at fault' were judged to have made perceptual errors. *Distraction* and *looked-but-failed-to-see errors* were the two most frequent perceptual errors in the team's judgement. Other relevant errors are *misjudgement of speed and distance* and *incorrect interpretation* (for an overview of this research see Hills, 1980, and Smiley, 1989). These interesting phenomena, especially the attention-related distraction and looked-but-failed-to-see errors, are discussed in more detail in Chapter 10.

A third basis arises from research on novice drivers. De Velde Harsenhorst and Lourens (1987,1988) find in their case study of a novice driver that 1186 out of the 4239 remarks made by the instructor during 36 driving lessons pertain to visual orientation (see Table 1 in Chapter 4). Nearly 60 percent of these 1186 remarks were classified as corrections and 26 percent as instructions. This 26 percent for instructions is higher than for all other tasks. In motor control, for example, these figures were 80 and 12 percent. This seems to indicate that a considerable amount of high-level cognitive control is required to control visual orientation. Another way to get a feel for the enormous amount of visual orientation remarks given is to examine the frequency of instructions. In 36 driving hours 4239 remarks were made, which amounts to about one remark every 30 seconds or one visual remark every two minutes. These general findings are backed up by another case study involving several instructors and several novices performed by Groeger et al. (1990).

The origin of the difficulty of visual orientation in driving is clearly articulated by Hills (1980). He notes that in evolutionary terms driving is only a very recent human activity and in a number of aspects humans are ill-equipped for it. One of the most significant of our limitations is the small size of the centre of vision and the rapid fall in acuity in the periphery. Eye-marker studies show that drivers average only three eye fixations per second, fixation rates greater than five per second being rarely achieved (Mourant & Rockwell, 1970, 1972; Wierda, Van Schagen, & Brookhuis, 1990). Therefore, the proportion of the visual scene the driver sees in detail in 1 or 2 seconds is very limited and this is frequently the order of time in which some quite crucial decisions have to be made. Part of the art of driving may therefore involve developing the skill of looking in the right place at the right time. It may also involve the ability to predict accurately where the critical points in the scene will be in the next few seconds ahead.¹

We may conclude that the large number of perception, attention and visual orientation errors in driving accidents and the difficulty in learning seems enough to justify our efforts to include some restrictions of the human perceptual system in DRIVER.

9.3 General requirements for a model of visual orientation in driving

Before we present our list of constraints that we used to model the perception mechanisms in DRIVER we first will discuss two general requirements for a model of visual orientation in driving.

9.3.1 A theory of object recognition

The first requirement is that Soar must be able to recognize objects. It will be clear that if Soar cannot recognize objects nothing useful can be said about attention mechanisms or eye movements. Unfortunately Soar doesn't provide an elaborate computational theory of object recognition. Newell (1990) admitted that Soar is underconstrained with respect to visual perception; Soar provides only the later stages (or higher levels) of visual processing. That is, it provides the cognitive aspects of visual information processing and it provides an interface to lower-level perception. Nevertheless, Newell felt that Soar offers a number of important constraints which will have a great influence on future visual-processing models in Soar. The force of these constraints can be seen, for example, in a study by Wiesmeyer into the nature of covert visual attention (Wiesmeyer, 1992). Covert visual attention is that portion of attention that does not involve movements of the head or eyes. His theory, called NOVA, is implemented in Soar and is able to account for behaviour in seven of the most significant classes of visual attention experiments from the psychology literature. These include precuing, crowding, decay, illusory conjunctions, search, counting, and detection experiments. These constraints are described in more detail in section four.

In DRIVER object recognition is mostly simulated. If an object is located in the attentional area, a symbolic description of it is automatically formed. In Wierda and Aasman (1991) we describe a driver's specific object perception both in Marr's stages of visual processing and in Jackendoff's theory of the computational mind (Jackendoff, 1987). We argue, in line with Newell's argument, that Soar only handles the cognitive aspect of cognition and the interface to the lower levels and that a system like DRIVER operates only on Marr's 3D level or on Jackendoff's conceptual level.

9.3.2 A theory of visual orientation

The second requirement for our model is that we must have a theory of visual orientation especially geared to the driving task. Our first step was to search the literature for a theory of attention, eye and head movement control in driving that might be incorporated in Soar. The last decade has seen consid-

erable interest in eye movements in driving, especially now that the hardware for registering eye movements is becoming more available. An indication of this interest is the series *Vision in Vehicles* (Gale et al., 1986, 1987, 1988). However, the massive amount of experimental findings have not yet been translated into a computational model of attention and eye movement control in driving. Wierda and Aasman (1991) note how many experiments on eye movements end in a statistical description of successive saccades (voluntary eye movements); see for example Zwahlen (in press). Oculomotor information seeking in visual search, instrument monitoring and computer-menu scanning is often modelled as a random-walk process or as a stratified random process with replacement (see Ellis & Smith, 1985, for an overview). Although these models do describe the data reasonably well we may not conclude from these findings that the underlying mental processes are random generators. The apparent randomness of eye movements in many experiments may just indicate that we do not yet know what the internal cognitive processes are; what their timing is and to what extent internal processes determine eye movements in parallel or sequentially.

Despite the fact that there is no ready-made theory available which can be applied fairly directly to Soar, we can still say something about a number of important constraints that determine both low-level perception and visual orientation in Soar. The constraints that determine visual orientation when approaching intersections are listed in the following chapter. The constraints for low-level perception, which are the basis of visual orientation, are dealt with in the following sections.

9.4 Constraints shaping DRIVER's perception

We now will discuss the two sources of constraints that shape DRIVER's perception. Table 1 serves as our guide in the discussion of these constraints.

9.4.1 Soar constraints

Some important general assumptions and constraints for perception are provided by Newell in his explanation of the perception-related features of Soar's base level space (Newell, 1990, Ch. 5). Note, however, that these constraints refer only to the cognitive aspects of visual cognition and the interface from central cognition to lower-level perception mechanisms, where 'lower' refers to the earlier stages of visual information processing².

The first constraint is that there is an *impenetrable perceptual module*. Newell (1990) posits a lower-level perceptual module that is impenetrable to cognition but that can be matched by so-called *encoding productions*. The original format of information in this module is analog. *Encoding productions* translate/match analog perceptual information into a *symbolic format* and enter it into the *base level problem space*. Object recognition is partly achieved by these

Table 1. Constraints on DRIVER's perception

Soar constraints

- Early processing is impenetrable to Soar
- Encoding productions put objects in WM
- Objects enter WM only in the Base Level Space
- Objects enter WM in the form of object-attribute-value triples
- Objects enter WM asynchronously and 'overwrite' old information (destructive modification)
- Attend operators fix perceived objects.
- Top-down generation of attend operators: controlled search
- Data-driven generation of attend operators: interrupt function.
- Stimulus selection in the visual field by attend operators takes at least two elaboration cycles and a decision cycle per operator.

Field constraints

- Stimulus selection without eye movements is possible in functional field
- Width of functional field is 20 degrees.
- High-quality object information from functional field: Presence, Movement, Direction, Size, Colour, Shape, Object Type
- Width of peripheral field is 210 degrees horizontally, 90 degrees vertically
- Low-quality object information from peripheral field: Presence, Movement, Direction, Size.
- Information in periphery is primarily used to guide eye and head movements.

Eye and head movement constraints

- During an eye movement no new visual information enters working memory
 - Eye and head movement time depends on speed and distance to move
 - Voluntary and involuntary eye movements: data-driven and top-down control.
 - Eye strain forces head to move
 - Head mainly follows eye, but independent control is possible
-

encoding productions. The symbolic format is Soar's *object-attribute-value* triple as used in most production systems. Soar has a uniform coding for all information in working memory. This includes the representation of perceptual information, the internal representation of the external world, and the motor command language.

Encoding productions provide asynchronous input as in principle they work *independently of goals* in the goal stack. The result is that objects enter working memory relatively independently of other processes in working memory, thereby enabling ongoing internal activity in working memory to be interrupted by new information from the external world.

When new information enters working memory it will overwrite the old information. In an earlier Soar version it was only possible to add to WM. In the current versions information *destructively modifies* old information. The only way to protect perceived objects against 'overwriting' is to 'fix' them by operator application. Fixing is achieved by applying an *attend operator* to find a perceived object and then attach it to the state. The attend operator, which is always issued by central cognition, may be given a search specification such that it finds an element satisfying this specification from anywhere in working memory. The result of the attend operator is that it switches cognition to the

place of the attended element, thus functioning as a channel selector. Though all elements in the visual field can in principle be recognized as objects, attention has to focus on an object in order to get detailed information about its physical and functional properties. Search in the visual field may be either *top-down* or *bottom-up*-controlled. Top-down control over search is achieved by setting search criteria and favouring operators that satisfy these criteria. Bottom-up control is the default mode for DRIVER. Whenever no other goals apply, attend operators for moving objects will have a high preference. For example, in DRIVER cars are pop-out objects. This means that moving cars usually interrupt other visual processing.

This attention mechanism is also called *internal gating* in the psychological literature (see Wierda and Maring, in press; Posner, 1980; and Erikson and Yei-Yu Yeh, 1986). As this gating mechanism is implemented by attend operators, stimulus selection in Soar takes at least two elaboration cycles – one cycle for generation and one for selection – and a decision cycle.³

In addition to the Soar constraints mentioned above, there are actually only two other constraints that we have incorporated in Soar and DRIVER. These constraints are what Hills (1980) calls the two most important human limitations in visual orientation in traffic. The first limitation is the small size of the centre of vision and the rapid loss in acuity in the periphery. The second limitation is the relative slowness of eye movements in complex, dynamic real-time tasks. Both these constraints are discussed in the following sections.⁴

9.4.2 Field constraints

The second main entry in Table 1 lists the main *field constraints*. We provided DRIVER with two fields, a functional visual field (FVF) and a Peripheral Visual Field, (PVF). The term *functional visual field* stems from Sanders (1963) and refers to an attentional area within which objects can be visited and recognized without eye movements. Note that this area is not directly linked to underlying physical perceptual systems such as the fovea. It is a 'functional' field that differs in width depending on properties of the stimuli, the background, the task, or mental load. Miura (1986) investigated the size of the functional field in various traffic settings, including tasks such as crossing an intersection. He showed that the size is a function of the complexity of the traffic situation. The 20 degree width of the functional field used in DRIVER stems from his research. The PVF in DRIVER is the area outside the FVF and its size is given by the fact that people perceive movements up to 210 degrees horizontally.

The use in DRIVER of only two fields is overly simple if we consider Treisman's theory of feature maps (Treisman and Gelade, 1980). The width of the maps varies according to the stimulus dimension. The width for movement is roughly 210 degrees horizontally, whereas the width for colour is about 20

degrees. Literature on the conspicuity of objects (Theeuwes, 1989) indicates that, depending on the task, attention may be directed on the bases of location, size, movement, and colour. Pop-out effects in other laboratory tasks (Wiesmeyer, 1992) indicate that shapes are also in some situations a distinguishing feature. The foregoing maps onto DRIVER in such a way that objects in the functional field are easily recognized, given that attention is directed at those objects. The object properties that can be perceived include *exact position, movement, direction, size, colour, and shape*. Objects in the periphery have low information quality. The only properties perceived are *movement, direction, size*. Information in periphery is only used to guide eye and head movements. No attend operators can be applied to these objects.

9.4.3 Eye and head movement constraints

The third set of constraints in Table 1 mostly speaks for itself. Eye and head movements take a considerable amount of time. Actual saccadic⁵ eye-movement times (travel + fixation time) vary from 70 to 700 milliseconds. Russo (1978) lists 230 milliseconds as a typical time. Eye and head movements are thus relatively slow compared to Soar's internal processes. A typical saccade will take from 10 to 20 elaboration cycles. An additional constraint in humans and thus in DRIVER is that during an eye movement no new information enters cognition.

In DRIVER, head movements generally follow eye movements. In the De Velde Harsenhorst and Lourens (1987, 1988) experiments we found that in traffic the head mainly follows the eyes⁶, but independent control is possible. One reason why the head follows the eyes is that although the eye may be rotated to a considerable degree, extreme rotation results in an uncomfortable muscle strain. This strain functions therefore as a parameter in DRIVER's control of head movements.

9.5 Implementation of low-level perception in DRIVER

We begin the presentation of DRIVER's low-level perception mechanisms with an overview. Figure 9-1 shows the main information processing cycle in DRIVER with respect to perception and eye and head movements. The LLPM determines which objects from WORLD will be transferred to the Soar input and thus to WM, given the current eye angle and field widths. Eye-move and head-move operators put motor-command structures onto the output links on Soar's top state. The Soar I/O takes these command structures and transfers them to the Lower-Level Motor Module (LLMM). The LLMM takes care of the execution of the command and sends feedback to the Soar I/O, which channels the information through to the predefined input links on the top state.

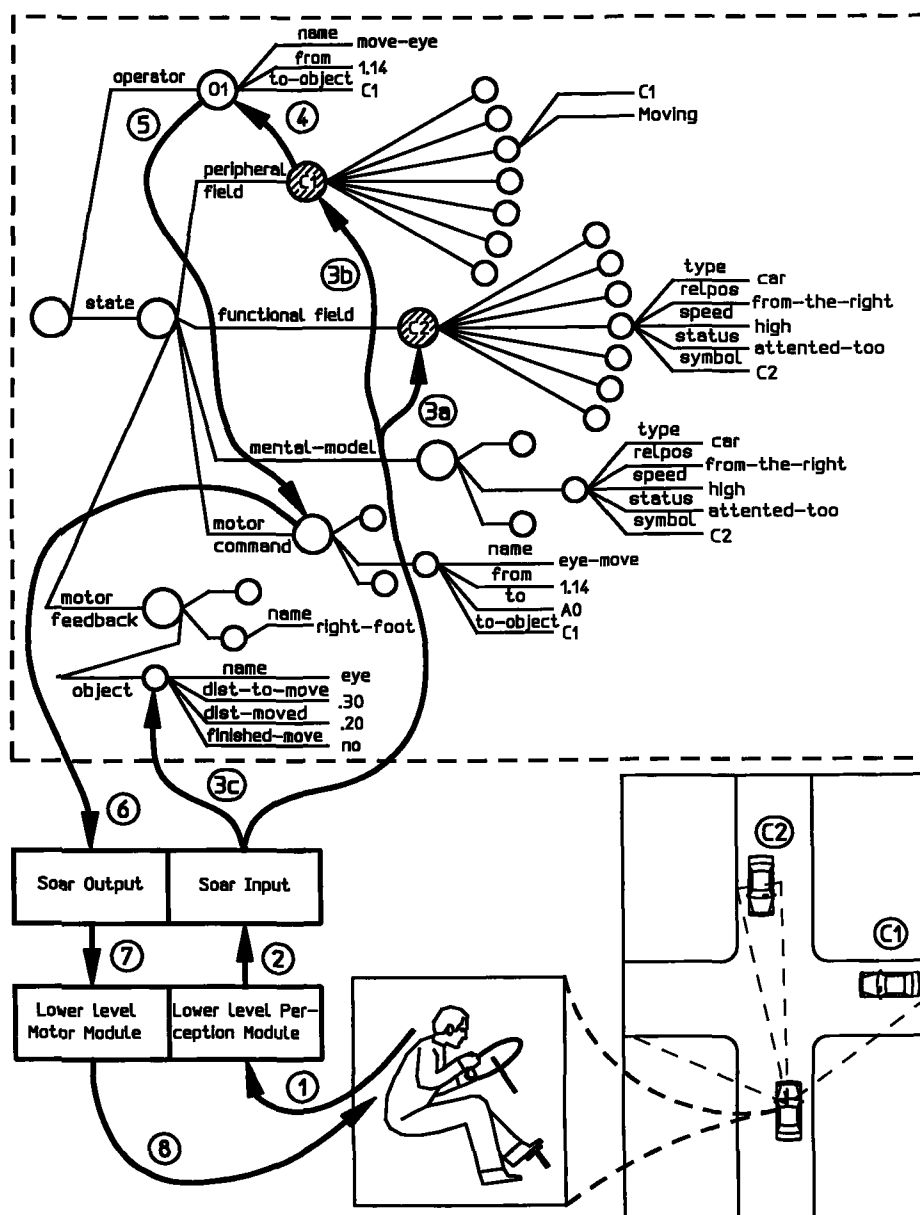


Figure 9-1. Perception cycle. (1) The LLPM computes that, given the current field of vision and the state of WORLD, C2 is in the functional field and C1 is moving in the periphery. (2) The appropriate information is filtered through to Soar Input and (3a, 3b) added to state S1. (4) DRIVER generates an eye-move operator for the moving object in its periphery and an eye-move command is added to S1. (5) Soar Output takes this structure from S1 and (6) feeds it to the LLMM. (7) Finally the LLMM will start an eye move towards C1. All the time feedback about the current position of the eye is fed to feedback structures on the state (3c)

The implementation of DRIVER's basic perception mechanisms is discussed in four steps. We start with a description the LLPM, the lower-level perception module. The following three sections then describe the attend operator, the eye-move operator and the head-move operator.

9.5.1 Lower-Level Perception Module

As discussed above, the main role of the LLPM is to filter the information available in WORLD and transfer this information to WM via the Soar input module. Each elaboration cycle the LLPM uses the current eye angle to determine for each object in WORLD whether it falls into the FVF, PVF or outside the PVF. Note that objects also may be occluded by houses or other visual obstacles. Depending on the visual field of a perceived object, the appropriate information is transferred to the Soar input so that the internal representation in Soar WM may be updated. Updating takes several forms: (1) if the object was not perceived earlier then the appropriate information is added to working memory; (2) if the object is already in working memory and the perceptual field is still the same then the old information about the object is overwritten; (3) if the object is already in working memory but the field of vision has changed, then, (a) if the change was from PVF to FVF then new information is added or (b) if the change was from FVF to PVF then, due to the restricted perceptibility, the old object is deleted from WM and a new one created.

Table 2 shows the main information per visual field. It is clear from this table that the LLPM 'does' all the simulation of object identification and recognition. Note that Figure 9-1 provided information about how objects are represented in working memory.

Table 2. Basic information for objects in functional field and periphery.

Object information	PVF	FVF
Identity (unique symbol)	•	•
Type of field (PVF,FVF)	•	•
Object in mirror (yes/no)	•	•
Moving (yes/no)	•	•
Going to intersection (yes/no)	•	•
Object left/right/in fixation point	•	•
Angle to fixation point	•	•
Type of object		•
Speed		•
Distance to object (DTO)		•
Distance to intersection (DTI)		•
Time to intersection (TTI)		•
Angle to own heading angle		•
Shape, colour, and size (optional)		•
Relative position		•

Table 3. Generation and preference productions for attend operators.

<pre> [1] (sp base-level*propose*move-attention (goal <g> ^state <s>) (state <s> ^perception <per>) (object <per> ^object <ob>) (object <ob> ^field functional) --> (goal <g> ^operator <o> +) (operator <o> ^name move-attention ^object <o>)) </pre>	Generate a move-attend operator for every object in functional field
<pre> [2] (sp base-level*prefer*move-attend*indifferent-moving (goal <g> ^state <s> ^operator <o1> + ^operator <o2> +) (<o1> ^name move-attention ^object <ob1>) (<ob1> ^moving yes) (<o2> ^name move-attention ^object <ob2>) (<ob2> ^moving yes) --> (goal <g> ^operator <ob1> - <ob2>)) </pre>	Attend operators for moving objects have same preference.
<pre> [3] (sp base-level*prefer*move-attention*moving-better (goal <g> ^state <s> ^operator <o1> + ^operator <o2> +) (<o1> ^name move-attention ^object <ob1>) (<ob1> ^moving yes) (<o2> ^name move-attention ^object <ob2>) (<ob2> ^moving no) --> (goal <g> ^operator <ob1> > <ob2>)) </pre>	Attend operators for moving object get higher preference than objects for static objects
<pre> [4] (sp base-level*prefer*move-attention*prefer-cars (goal <g> ^state <s> ^operator <o1> + ^operator <o2> +) (<o1> ^name move-attention ^object <ob1>) (<ob1> ^type car) (<o2> ^name move-attention ^object <ob2>) (<ob2> ^type bicycle) --> (goal <g> ^operator <ob1> > <ob2>)) </pre>	Cars have higher preference than bicycles
<pre> [5] (sp base-level*prefer*move-attention*cross-intersection (goal <g> ^state <s> ^operator <o1>) (<s> ^model <m>) (<m> ^object <ob>) (<ob> ^name SELF ^going-to-intersection yes ^manoeuvre cross) (<o1> ^name move-attention ^object <ob1>) (<ob1> ^type car ^direction right) --> (goal <g> ^operator <ob1> >)) </pre>	Cars from the right have the highest preference when ap- proaching an intersection.

9.5.2 DRIVER's orientation operators

The goal of this and the following chapter is to describe DRIVER's visual orientation strategies, where strategies are defined as the ordering and timing of its *visual orientation operators*. DRIVER employs three visual orientation operators: *move-attend*, *move-eye* and *move-head*. We will discuss each of the visual orientation operators in terms of generation, selection and application. In some cases we refer to strategies in driving but the real discussion of visual strategies takes place in the next chapter. The following subsections provide only the basic underlying mechanisms.

9.5.3 Move-attend operator

The LLPM automatically puts all objects that fall within the FVF on Soar's top state. Though all elements in the FVF can in principle be recognized, attention has to be focused on an object in order to get detailed information about its physical and functional properties. The introduction to this chapter referred to this mechanism as 'internal gating'.

Generation. In order to get some feeling for the knowledge involved in generating and selecting operators we present a few (fragments of) productions in Table 3. Production 1 in this table shows how DRIVER generates a move-attend operator for each object in the functional field.

Selection. When multiple move-attends are generated a selection has to be made by means of Soar's preference mechanism. All move-attend operators are in principle mutually indifferent so as to avoid tie impasses. It seems highly unlikely that Soar would get into an impasse that lasts two seconds in order to decide between attend operators. Rules [2] to [5] in Table 3 provide examples of rules that express preferences between operators. Production [2] expresses that move-attend operators generated for moving objects are indifferent to each other and rule [3] expresses that moving objects should receive more attention than static objects. In versions of DRIVER where we make shape a pop-out feature, productions [4] and [5] are possible⁷. Production [5] is a very specialised production which states that in the approach to an intersection cars from the right are most important.

Note that the rules in the table do not refer to factors such as speed and distance to intersection. It is impossible to have productions like:

```
If      there is a move-attend operator O for perceptual object OB and
        and OB is an object whose
        type is car
        coming from the right
        speed > 12
        dti < 40
        on collision course
        -->
        operator O is best
```


because this implies that the object had already been attended to (as too many features are recognized). This would defy the whole purpose of the move-attend operator, which was to visit an object to get to know more about it.

Application. The application of the operators has two results. The first is that the visited object is marked as noticed, the second result is that in some cases the perceptual object is 'fixed' by copying it to the state and in other cases is used to update the internal model of the world.

9.5.4 Move-eye operator

The LLPM dumps all objects within the 210-degree field width in the PVF. Note that the PVF contains unidentified and unrecognized objects. In order to identify and recognize an object we need to direct the eyes to that object. Eye moves are also possible within the FVF. Soar allows several alternatives for the generation, selection and application of move-eye operators.

Generation. There are two types of generation for move-eye operators: data-driven and top-down-controlled generation. Examples of data-driven generation are move-eye operators that are automatically created for each object in the FVF. Data-driven generation for objects in the PVF occurs only for objects moving in a lateral direction. Top-down-controlled generation happens when move operators are generated from plans (checklists) or goals. Table 4 gives two examples of rules of goal-based generation of operators.

Table 4. Examples of rules for search control

<pre>[6] (sp base-level*propose*eye-move*conflicting-car-from-the-right (goal <g> ^state <s>) (<s> ^model <m>) (<m> ^object <ob1> <ob2>) (<ob1> ^going-to-intersection yes ^dti close ^name self) (<ob2> ^angle-to-me <angle> ^going-to-intersection yes ^dti close ^type car) ..> (goal <g> ^operator <o>) (<o> ^name move-eye ^to direction ^value <angle> ^speed default))</pre>	If I, <o1>, am going to the intersection and there is a car <o2> going to the intersection and we are both close to the intersection then look in the direction where I saw <o2> last time
<pre>[7] (spe base-level*propose*eye-move*checklist*look-right (goal <g> ^state <s>) (<s> ^model <m> ^checklist <ch>) (<m> ^object <ob1>) (<ob1> ^going-to-intersection yes ^dti close) ..> (goal <g> ^operator <o1> <o2> <o3>) (<o1> ^name move-eye ^to direction ^value left) (<o2> ^name move-eye ^to direction ^value forward) (<o3> ^name move-eye ^to direction ^value right))</pre>	If I'm approaching the intersection then make sure that I generate a left, forward and right operator.

The move-eye operator has two parameters. The first is the *to* parameter. The eye may be directed to objects, i.e. to an object in the FVF or PVF, or to one of the two mirrors in the car. The *to* parameter may also be a direction, in which case there is an extra parameter 'value' that specifies the angle to move. The second, optional parameter is the *speed* of the eye movement. The following provides some examples of how eye moves are generated in DRIVER.

Example 1: During the driving process DRIVER builds up an internal representation of the external traffic environment. We refer to this internal model as the *mental model* of the world. The mental model might represent the situation in which in the approach to an intersection a car from the right was seen at a 45-degree angle. Rule [6] in Table 4 shows how this knowledge will guide DRIVER to look again in search of a car at 45 degrees as a sort of security check. This is thus an example of how DRIVER's goals and its mental model guide the way the environment is scanned.

Example 2: in some situations DRIVER uses a checklist of directions-to-look-in that must be completed before a certain event. The advantage of the term checklist is that it implies an unordered set of actions. An example of a checklist may be found in crossing an intersection. Before crossing the intersection one must have looked to the right and to the left and in front. The following production will generate three operators to look in all directions before crossing the intersection. It might be argued that by providing DRIVER with such a rule the desired behaviour is already provided by the cognitive modeller. There are two responses to this criticism. The first is that this rule can be learned from instruction, just as in human drivers. The second response is that the timing and ordering of these operators are far more important than the generation of the operators. Note that no timing or ordering information is given in this production.

Selection. The selection of the operators is, as always, a matter of problem solving. However, the issue of selecting the right move-eye operator is the central issue of the following chapter on visual orientation strategies, we will not go into details here.

Application. The move-eye operator works almost exactly like the move operator that was discussed in the previous chapters. The application of the move-eye operator results in a motor-command structure on an output link on the top state. The structure is transferred by the Soar output to the lower-level motor module (LLMM), which takes care of the execution and the eye-movement feedback. The main control structure from the LLMM and the structure of the returned feedback are described in the previous chapters and will not be repeated here. The main advantage of this scheme is that Soar does not have to wait for an eye movement to be finished but can proceed

with its normal execution. The move-eye operator only initiates a move. Productions may then check the unfolding of the process.

9.5.5 Move-head operator

Generation. The main reason for DRIVER moving its head is when eye strain becomes excessive. Eye strain is the variable that expresses the difference in angle between the orientation of the head and the eyes. As in the move-eye operator, the move-head operator has three parameters. The first is the *to* parameter. The head can be moved to an object, to the mirror or just in a direction. If the *to* parameter has the value *direction* a second value is added which specifies the visual angle to move. The third parameter specifies the speed at which the head should be turned. The following production [8] states that if the eye strain is high and the eye angle is <ea>, then move the head also to position <ea>. Note that both <ea> and the eye strain are derived from eye feedback.

Table 5. Examples of rules for head movements.

<pre>[8] (sp base-level*propose*move-head*to-eye-position (goal <g> ^state <s>) (<s> ^eye-feedback <ef>) (<ef> ^eye-strain high ^eye-angle <ea>) --> (goal <g> ^operator <o>) (<o> ^name move-head ^to direction ^value <ea> ^speed fast))</pre>	If eye strain is high then move the head in the direction of the eyes.
<pre>[9] (sp base-level*propose*move-head*to-eye-position (<o> ^name move-head ^to direction ^value <ea> ^speed fast) (goal <g> ^state <s> ^operator <o> +) (<s> ^eye-feedback <ef> ^model <m>) (<ef> ^eye-strain > 3 ^eye-angle <-30) (<m> ^object <ob1>) (<ob1> ^name SELF ^going-to-intersection yes ^manoeuvre turn-right ^distance-to-intersection close) --> (goal <g> ^operator <o> -))</pre>	Inhibit the operator discussed above when turning right and very close to the intersection.

Selection. The previous production proposed that the head should follow the eye in angular orientation when the eye strain is high. However, in some situations it may be uneconomical to turn the head. An expert might have production [9] inhibit turning the head to the left when turning right at an intersection.

Application. The application of the move-head operator is identical to that of the move-eye operator.

9.6 Discussion

This chapter described the design decisions that were made with respect to the granularity and depth of the simulation of low-level perception. All our design decisions were based on our goal to model *when* DRIVER will look in *what* direction after objects are recognized and have been entered in WM. Just as in motor control, the contributions of DRIVER's perception module lie at the cognitive level and the interface to the lower levels. A full discussion of the usefulness of DRIVER's low-level-perception mechanisms must be postponed until the use of these mechanisms is discussed in the following chapter. However, some issues can already be discussed here.

The first issue concerns the setting of (1) the width of the visual fields, (2) the quality of information from these fields, and (3) eye-movement constraints (speed, eye-head strain). The choices that we made with respect to these parameters are cursorily justified by references to the literature, but much more research is needed to establish the precise parameters. For each of these parameters it is reasonably clear how changes might propagate to the rest of the visual orientation task. Shrinking the size of the functional field undoubtedly forces DRIVER to move its eyes more often. Changing the information quality in fields has severe repercussions for the way the attend or eye-move operators are selected, for example. Changing, say, the speed of head moves (or the maximum rotation) to simulate older drivers will also induce different strategies in visual orientation.

Just as important as these parameters are the methods used to generate operators and to select operators. In a sense we come up here against a version of the generate and test problem (Newell & Simon, 1972). The generate and test problem in a classical AI search task is that a choice must be made between putting intelligence in the generator of options (so that not too many options are generated) and putting intelligence in the 'tester' of these options (so that the available options are pruned as efficiently as possible.) The solution to this problem in Soar and DRIVER, for example in the case of eye-movement operators, is that one can choose to (1) generate an operator for every object in the FVF and PVF and let the preferences sort out who wins or (2) generate operators based on expectations. The latter choice limits the number of operators generated, so that little preference knowledge is required. From a psychological viewpoint it is hard to decide which method is better as it is possible to implement the same type of visual orientation behaviour with either one. With the first method working memory will get swamped with operators and thus violate working-memory size constraints, whereas the second method is much more space-efficient. However, the evidence provided by data-driven and involuntary eye movements convinces us that the odds are still in favour of the first method. In the treatment of the 'visual' operators we

stipulated the standard methods for generation and selection, but we hope that it is clear by now that the user of DRIVER may vary these methods. From a practical (Soar programming) viewpoint, on some occasions the first method provides more flexibility. It is far easier to generate an eye-move or attend operator for each object and then reject most of them. The disadvantage (still from a Soar programming viewpoint) is that working memory becomes overloaded with operators, thereby slowing down the matcher. So we restricted automatic generation only to the FVF and moving objects in the PVF.

The most serious problem with DRIVER's low-level perception is undoubtedly object recognition. How objects are recognized and how they arrive in WM is to a large extent a matter of 'magic', despite the low-level constraints provided. We know that identification and recognition involves both data-driven automatic processing and top-down-controlled processing, but in DRIVER identification and recognition are purely data-driven processes. 3D reasoning and spatial/temporal reasoning in DRIVER are performed almost completely in terms of the road and intersection. For example, object O is going-to-intersection, coming-from-the-right, time-to-intersection = 5. However, all these properties of object O are 'derived features', requiring reasoning over and with 3D models of roads, intersections and the objects driving on the road. 3D models are also involved in the derivation of size and speed of objects.

One other object-recognition issue not yet covered is the time it takes to recognize an item. In DRIVER this time is set to the execution time of one attend operator. In some Soar research a comprehend operator is applied after an attend operator for the purpose of analysing the item. We refrained from using comprehend operators, as in DRIVER the attend operator also takes care of the object recognition. There is evidence that recognition of complex objects for *prepared* subjects is very fast. Potter and Fauconer (1975) report that when a picture of, say, a hammer is presented for 44 ms, subjects are able to link the image to its name in 50 % of the trials. At presentation times of around 250 ms performance is already perfect. Card, Moran and Newell (1986) estimate the range for the cycle time of the perceptual processor in the model human processor at between 50 and 200 ms, with the average at around 100 ms. Wierda and Aasman (1991) conclude that even when objects are visible for a very short time the driver can react, provided that he has recognized the situation as potentially dangerous. In DRIVER the time required to apply an attend operator range is between 100 and 170 ms⁸, which is well within the range needed for correct object recognition.

Another serious problem is DRIVER's disregard for feature maps. DRIVER allows only a distinction between FVF and PVF; within a field all objects are the same in terms of information content. This distinction is undoubtedly too crude. Feature maps differ in width for different types of features. Future

research will reveal whether we can manage with this simplification or whether we will have to provide Soar (i.e. DRIVER) with more elaborate feature maps.

Despite the list of critical issues listed above, we hope to show in the next chapter that DRIVER does provide an adequate apparatus for experimenting with attentional mechanisms.

9.7 Notes

¹ Visual orientation in driving is not only hard for humans. Reece (1992) describes the difficulty of robot perception in static domains and in complex, dynamic domains. First, Reece discusses several reports that estimate the complexity of vision computations in *static domains*. Template matching is polynomially complex in the size of the template model and image. Matching processed image features to model features is in general exponentially complex, but in some cases polynomial in the number of scene and model features. Interpreting all features in a scene together requires searching the space of all possible interpretations, which is exponentially large:

$$\text{Number of interpretations} = (\text{Number of region classes})^{\text{Number of Regions.}}$$

However, robot perception becomes even harder in *complex, dynamic domains* like driving:

- The world state is not known in advance and cannot be predicted. Therefore the robot must observe the environment continuously.
- Objects play different roles. For example, they are not all just obstacles. The driving environment has many objects and types of objects - cars, roads, markings, signs, signals, etc.
- The appearance of objects changes due to many factors, including location in the field of view, range, size, orientation, shape, colour, marking, occlusion, illumination, reflections, dirt, haze, etc. The appearance can vary as much as the product of all these factors.
- The environment is cluttered and distracting. It contains many background features that can be confused with the features of important objects.
- Domain dynamics place time constraints on perceptual computation.

All of the above explains why there are currently no robots that are capable of even simple tasks like autonomous road following and car and sign recognition at full driving speed. Reece concludes that for robots the computational costs are many orders of magnitudes too high for the time and resources available. We can say that perception is *effectively intractable*. Reece's thesis involves the intelligent sampling of the world, integrating reasoning with perception. In a sense this is the same approach followed in this chapter.

² The distinction between 'higher' and 'lower' refers to the psychological distinction between early and late visual processing (for a recent overview see van de Heijden and Joustra, 1992) and Marr's (1972) distinction between processing stages in visual information processing.

³ Note that the restriction to use attention operators in order to extract information from the visual store is self-imposed in Soar. In Soar it is possible to recognize and attend to objects in parallel.

⁴ Readers familiar with the perception literature will probably now be wondering why we have not included other low-level constraints such as stereopsis or convergence. The basic justifications for dispensing with these constraints stem from Hill (1985). Hill notes a confusion in the research field that is trying to establish the importance of basic driver vision and perception in accidents. The best research evidence available suggests that reduced visual field, defective colour vision,

and poor stereoscopic acuity are not associated with an increase in the driver's accident rate (Councel and Allen, 1974; Hills and Burg, 1977). An illustrative example of a study from which these results are drawn is a large-scale study by Burg (1967, 1968). This study involved visual measurements on over 17,500 California drivers. The vision tests used included static visual acuity, dynamic visual acuity, a low-light threshold recognition task, glare recovery, field of vision, and phoria. These measures were compared with the three-year accident histories of the drivers, which included over 5200 accidents. Contrary, perhaps, to intuitive expectations, Burg found only very weak (<0.06) correlation coefficients between the various vision scores and accident rates. These correlations were only slightly higher for subjects over 50 years old. Even with the above evidence one might assume that convergence and stereopsis play an important role in driving. However, whilst the binocular cues of ocular convergence and stereopsis may aid the driver in vehicle following, there is reason to believe that generally it is the various monocular cues of distance that predominate in driving, particularly as one-eyed drivers do not have undue problems. A recent study suggests, in fact, that one-eyed private pilots can land planes rather better than two-eyed pilots (Lewis et al., 1973).

⁵ DRIVER is only concerned with saccadic eye movements. Other types of eye movements such as smooth pursuit movements or vestibul-ocular movements are not included.

⁶ Personal observation from TRC colleague Ep Piersma, who analysed the DVH&L eye-movement data as well.

⁷ Production 4 will probably generate irritation among cyclists in that it states that cars are more important than bicycles. Note however that these are only examples and that there may be individual differences between drivers.

⁸ See chapter on multitasking at the end of this study.

10 Visual orientation

Summary: The previous chapter described DRIVER's perception without regard to traffic. This chapter describes visual orientation in the approach to unordered intersections, using the mechanisms developed in the previous chapter. We will see a) how the environment largely constrains actual behaviour and b) how top-down strategies are relatively easily induced by only a few Soar productions.

10.1 Introduction

The present chapter describes the visual orientation rules that build on the mechanisms discussed in the previous chapter. We shall focus especially on the visual orientation behaviour in the approach and negotiation of intersections.

We begin this chapter with a recapitulation of the visual orientation behaviour of the experienced drivers in the De Velde Harsenhorst and Lourens experiments (See also Chapter 4). Section three then describes the two types of rules that together determine DRIVER's visual orientation. The first type is the manoeuvre-independent and *data-driven* orientation rule. Because these rules are applicable in nearly all situations we refer to them as the *default orientation rules*. DRIVER also has a set of manoeuvre-dependent orientation rules. The type of behaviour generated by these rules may be characterised as *goal-driven* or *top-down-controlled*. We will describe how these two types of rules interact to generate the appropriate behaviour. The fourth section describes a list of other interesting visual orientation phenomena that fit the framework of DRIVER. The final section concludes with a discussion of some perceptual and visual orientation issues in DRIVER.

10.2 Novice and experienced drivers in the approach to an intersection

Chapter 4 described the behaviour of experienced drivers in their approach and negotiation of unordered intersections. Figure 10-1 shows both the visual orientation of the experienced drivers and that of the novice driver in the same left-turn manoeuvre (location 1). The upper figure shows 20 of a novice

driver's 36 driving lessons. Only the conflict-free approaches, i.e. approaches in which there was no distraction from traffic coming from the right, are depicted. The lower figure shows 20 conflict-free approaches by 20 experienced drivers. Many interesting facts can be derived from this figure, but we focus here on the areas that represent a right, forward and left orientation¹. First note the relative fixedness and stability in behaviour for the 20 experienced drivers compared to the intra-subject variance in the novice behaviour. Secondly, note that almost all the experienced drivers look to the right at almost the same time. Thirdly, note that the experienced drivers shift their visual field far less often than the novice driver, both before and after the intersection.

Chapter 4 described in more detail the visual orientation of experienced drivers. The following list provides a sample of the regularities in behaviour in an approach to an unordered intersection.

Table 1. A sample of visual orientation behaviour in approach to an intersection with restricted visibility in both directions.

-
- look in the mirror before the first look to the right
 - in all manoeuvres the first look to the right must occur before releasing the brake
 - in a right turn look left before looking right
 - in a left turn look right before looking left
 - when crossing the order of left/ right is not important
 - there is always one glance forward between left-right and right-left shifts
-

10.3 The visual orientation rules

The following sections describe the visual orientation rules that determine DRIVER's orientation behaviour. In the first subsection we describe the manoeuvre-independent orientation rules that generate *data-driven* behaviour. The second subsection describes the manoeuvre-dependent orientation rules that generate *goal-driven* or *top-down-controlled* behaviour. In the third subsection we then describe how these two types of rules interact and work together to generate behaviour of experienced drivers.

10.3.1 DRIVER's default orientation rules

DRIVER's *default orientation rules*² are manoeuvre-independent rules that play a role in almost all situations. The following lists the default orientation scheme that applies in most non-critical situations. Note that some of these rules were discussed in the previous chapter, though there may be slight deviations because only examples were discussed in that chapter.

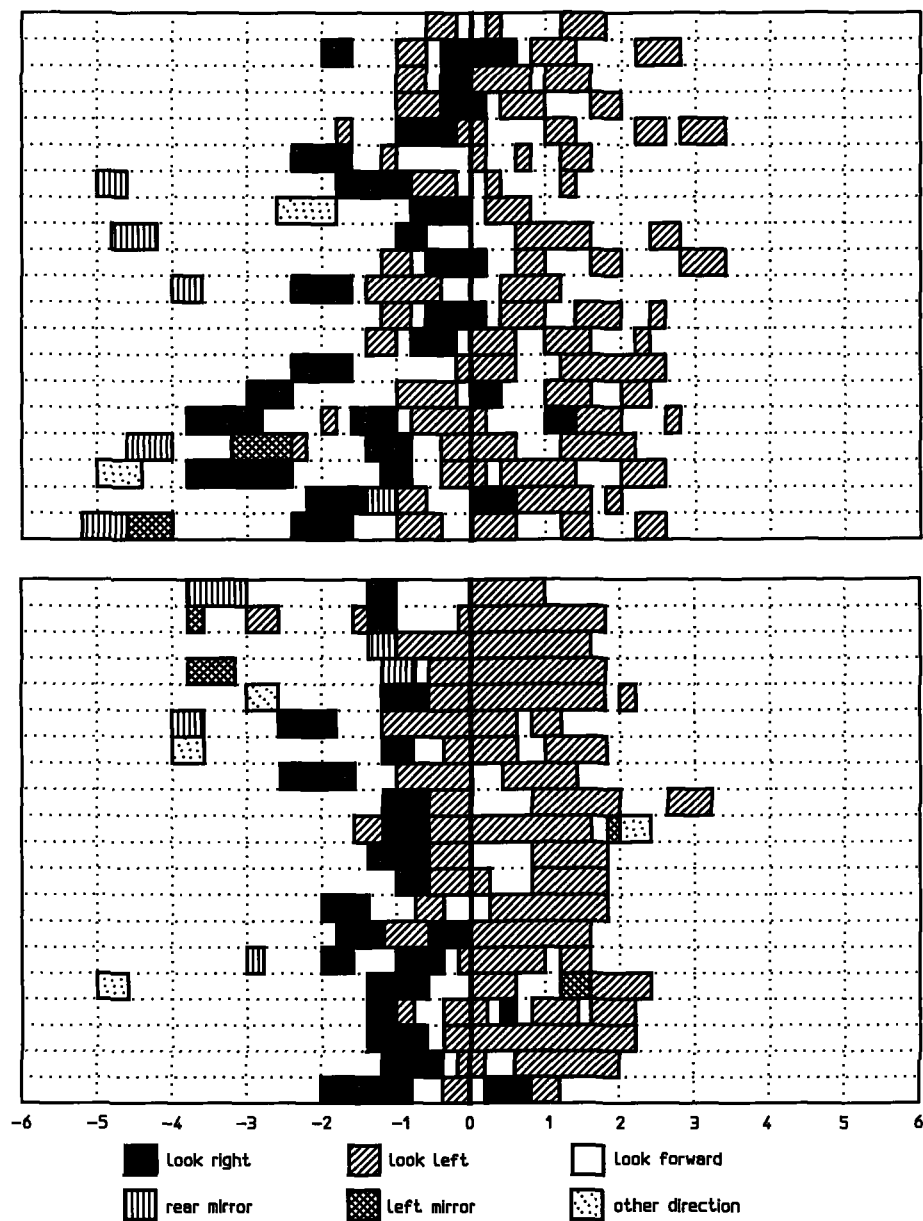


Figure 10-1. Visual orientation for a novice driver and 20 experienced drivers at a left turn. The upper part of the figure shows 20 of the 36 driving lessons of a novice driver. Only the conflict-free approaches to intersections are depicted. The lower half of the figure shows 20 conflict-free approaches by 20 different experienced drivers. The X-axis shows the time to intersection. Time runs from left to right, so negative times represent the approach to the intersection.

Table 2. Default orientation rules.

[1]	attention operators are preferred to eye-move operators and eye-move operators are preferred to head move operators.
[2]	moving objects in the functional field are preferred to moving objects in the peripheral field
[3]	within fields moving objects are preferred to stationary objects.
[4]	on straight lanes highest preference every 4 seconds for forward-directed eye-move operators.
[5]	on bends highest preference every 1 second for forward-directed eye-move operators.
[6]	highest preference for speedometer every 10 seconds.
[7]	moving objects in front have higher preference than moving objects behind
[8]	eye-move and head-move operators for moving objects on road have higher preference than operators for moving objects on the pavement

Rule [1] ensures that DRIVER really sees what it is looking at; most objects in the functional field, at least the moving ones, are analysed before eyes are moved to other objects. This rule, in combination with the other rules, also avoids random head movements. Rule [2] ensures that attention is first directed at objects in the functional field. If Rule [3] is left out, DRIVER's behaviour tends to become extremely dangerous, both to itself and others. Rule [4] is important for lane keeping, but it also avoids DRIVER being locked in the mirror³. Bartmann et al. (1991) found that depending on the speed, drivers fixate their lane (and look forward) from 15 to 70 percent of their time in an urban environment. Rule [5] requires no explanation. The ten seconds in rule [6] applies to normal straight-road driving. In other situations this parameter may vary. The empirical evidence in this case comes from Jessorun et al. (1990) who found in an experiment using eye movement recordings that drivers look at their speedometer surprisingly often. Rule [7] states that objects in front of DRIVER are in general more important than objects behind you. Finally, rule [8] makes sure that DRIVER attends to the objects on the road and does not get distracted by interesting sights on pavements.

Without the above rules the visual orientation behaviour of DRIVER would be mainly data-driven because operators are automatically generated for all objects in the functional field and for all moving objects in the peripheral field⁴. Note that the above rules do not generate operators themselves but only express preferences for operators that have already been generated. They ensure that the 'randomness' of visual behaviour is kept within certain bounds and that some basic safety measures are taken. However, even with the restrictiveness of the above rules the overall behaviour is still best described as predominantly data-driven.

10.3.2 DRIVER's orientation rules for intersections

It is clear that the above scheme is not enough for more specific situations such as handling an intersection. For example, in the approach to an intersection, active search for cars from the right or from the left is required. What DRIVER thus needs is top-down control on top of the basic scheme. The

following is a sample of DRIVER's top-down rules in the approach to an intersection.

Table 3. Intersection-specific visual orientation rules

[9]	generate operator to look in rear-mirror at TTI = 5 and give it highest preference
[10]	generate operator to look right at TTI = 3 and give it highest preference
[11]	generate operator to look right before releasing the brake and give it highest preference
[12]	if manoeuvre is 'right turn' then generate left-look, right-look, and forward-look operator and prefer left over forward over right.
[13]	if manoeuvre is 'left turn' then generate left-look, right-look, and forward-look operator and prefer right over forward over left
[14]	if manoeuvre is 'cross' then generate left-look, right-look, and forward-look operator and prefer right over forward over left ⁵
[15]	if intersection is seen then search for traffic sign near the intersection on the right pavement.

Rules [10] and [11] have the same outcome. The one that fires first is applied. Note that rules [12], [13] and [14] not only generate preferences but *also operators*. This in contrast to the default orientation rules which only express preferences for operators that are automatically generated for all objects in the FVF and for all moving objects in the PVF. It is essential that intersection-dependent rules generate operators in critical situations. For example, at a right turn DRIVER must look left before turning. In most situations working memory will contain an eye-move operator for which a preference can be expressed but not guaranteed. If DRIVER looks too far to the right or if nothing moves on DRIVER's left side then there will be no look-left operators.

The operators generated by these rules have in general a high preference, higher than the default orientation rules. The one exception is attention. Because attention has been given a lower preference than the eye-move operators generated in the above rules there is a risk that eye moves may follow one another consecutively without the functional field in between being analysed, i.e. without anything being seen. Rule [1] of the default rules ensures that everything DRIVER looks at is analysed by attention operators.

The preferences that are expressed in the above rules are not Soar's standard preferences. In Soar all preferences, with the exception of an acceptable preference, are invisible to productions once they are added to working memory. However, in order to facilitate error correction and flexibility in overriding current preferences, *explicit* and *unique* preferences are required. The same preferences have already been discussed in Chapters 7 and 8 on the default rules and the hold mechanisms and will be discussed again in more detail in the chapter on multitasking. The following is an example of such a preference.

```
(<o> ^name move-eye ^to direction ^value <angle>
  ^speed default ^evaluation <ev>)
(evaluation <ev> ^value 3 ^id x190)
```

10.3.3 An example of DRIVER's behaviour

Figure 10-2 shows a specimen left turn performed by DRIVER. The cones show the main orientation of the functional field. What we have not shown is that within these cones small eye movements and attention shifts take place. In broad terms, then, if we compare Figure 10-2 with the lower part of Figure 10-1 we find a reasonable fit between DRIVER and the experienced drivers with respect to the ordering and timing of actions.

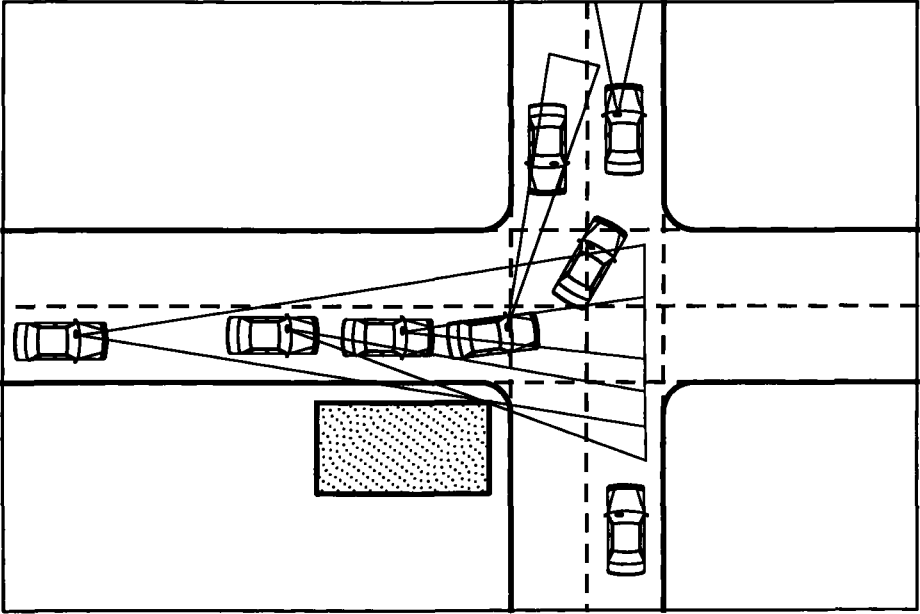


Figure 10-2. DRIVER's main orientation in the approach to an intersection. The cars display DRIVER at different points in time. The cones indicate the 20 degree functional field.

At this point we should however mention that finding a fit at the level of eye movements is complicated by the fact that the study by Velde Harsenhorst and Lourens provided the main orientations of the visual fields but not small eye movements. The behaviour of DRIVER is much more fine-grained, since it also includes attention and eye movements within the same general direction. The overall trace provided in Chapter 14 (about multitasking) will be more detailed with respect to visual orientation in the approach to an intersection. In this trace we see how attention operators always follow eye-move operators, that head-move operators seldom occur and that attention, head-move and eye-move operators have to compete with all other operators for their share of processing.

10.4 Fitting other visual orientation phenomena

We have now seen how the combination of the lower-level perception mechanisms, lower-level motor modules, a set of bottom-up rules and a number of intersection-specific rules together generate visual orientation behaviour in the approach to an intersection. Below we shall show that the lower-level mechanisms and the orientation rules are not so specific that they only specify intersection behaviour, but that they also describe other, more global visual orientation phenomena and regularities.

Table 4 presents a list of these phenomena that are, or might be, addressed by DRIVER.

Table 4. Other visual orientation regularities that fit DRIVER

-
- Experienced drivers look at the relevant items in the traffic environment.
 - Experienced drivers look earlier at the relevant items.
 - Experienced drivers may rely on peripheral vision to a greater extent.
 - Experienced drivers are able to internally update position of moving objects.
 - There is evidence of local and global scan paths in driving.
 - Looked-but-not-seen and seen-but-not-looked phenomena.
 - Experienced drivers develop strategies which have not been learned in driving lessons, and generate a-normative behaviour.
 - Experienced drivers employ fixed strategies in negotiating intersections.
 - Experienced drivers shift their main field of vision less often in negotiating intersections.
-

Experienced drivers look at the relevant items in the traffic environment.

Mourant and Rockwell (1972) found that experienced drivers look longer and more often at the relevant items in the traffic environment. Their findings and conclusions have been referred to for a long time in traffic psychology. However, there seems to be something of a paradox here. One might expect that especially experienced drivers would only need a short amount of time to take in a situation and thus have room for idle activity. This is exactly what has been found for example by Wierda, Schagen, and Brookhuis (1990) in an experiment that compared children and adults. Adults do indeed seem to search for the relevant items in a more systematic fashion, but in simple situations only a small fraction of all eye movements is dedicated to active search, the rest is idle visual activity. Only in more complex and critical situations do adults indeed look more often and longer at the relevant items. Especially in time-critical situations subjects tend to develop more top-down control over gaze direction.

The pattern that it is only in time-critical situations that relevant items are looked at is clearly discernible in DRIVER. Most of the time it indeed looks at irrelevant items such as houses and trees. The question of what the relevant objects for DRIVER are is implicitly answered by looking at both the default orientation rules and the intersection rules. However, as long as DRIVER is not close to the intersection and there are not many moving objects, DRIVER will not be constrained in the direction and objects it wants to look at.

Experienced drivers look earlier at the relevant items.

Mourant and Rockwell (1972) also found that experienced drivers look earlier at the relevant items. Recent research (Wierda, Van Schagen en Brookhuis, 1990) confirms these conclusions. Experienced drivers thus anticipate by actively searching for relevant information at an early stage. Active search in DRIVER for cars or signs is induced for example by rule [15]. However, most of the intersection rules induce active search for cars at the earliest possible moment at unordered intersections with restricted visibility in both directions.

Experienced drivers may rely on peripheral vision to a greater extent

There is evidence to indicate that experienced drivers may rely more on peripheral vision because they have more finely-tuned expectations. Mourant and Rockwell (1972) describe how experienced drivers seem to rely more on peripheral vision in the guidance of eye movements. A possible strategy that experienced drivers might employ in the approach to an intersection is that of looking forward as long as nothing is moving in the corners of their eyes (in Soar terms: if there are no move-eye operators for moving objects then look forward). Table 5 shows how this rather passive strategy can be induced using only a few productions.

Table 5. A passive strategy that relies on peripheral vision.

[16] if approaching intersection look forward
[17] if there is operator for moving object in periphery then look at that object
[18] operators for moving objects in right periphery are preferred to operators for moving objects in left periphery.

Note that the intersection rules [10] to [15] from Table 3 generate a more active strategy.

Experienced drivers are able to internally update position of moving objects.

An interesting phenomenon in experienced drivers is that they are able to predict the future locations of moving objects internally while they have their eyes closed or are looking in a different direction (Cremer, Snel and Brouwer, 1990). In their study, subjects would first look at a vehicle approaching the intersection, then they would look in another direction and finally directly at the point where the vehicle could be expected given its earlier speed and position.

DRIVER is also able to update the position of moving objects internally, but on closer inspection the explanation of the phenomenon in DRIVER is very simple. When for example an object from the left is seen, DRIVER will store the angle at which it was seen. When it later feels that it needs to check the location of the left vehicle again it will first look at that angle. Of course there will always be a little error but in most situations it will be right.

There is evidence of local and global scan paths in driving.

In the natural pattern of eye movements periods of small-amplitude moves in restricted areas alternate with large-amplitude eye moves to other areas. Groner, Walder and Groner (1984) called bottom-up-controlled eye movements 'local scan paths'. The conceptually based, top-down-controlled movements were labelled 'global scan paths'. Groner and Mentz (1985) suggest that the control over fixations shifts between global and local scans. Global scans indicate a search plan, but during the execution of such a plan stimuli in the field of vision may attract attention and subsequently the fixation. Examples of these scan paths are found in radiologists examining X-ray photos (Nodine and Kundal, 1987) and in chess (Newell and Simon, 1972), but they are also found in driving (Wierda and Maring, in press). DRIVER (both car drivers and cyclists) will spend some time in one small area and then jump to an entirely different area.

DRIVER clearly displays local and global scan paths. In non-critical situations the default rules prevail, which means that the eye moves will in general be directed at objects in the functional field, thus leading to local scans. In more critical situations, such as in the approach to the intersection, the intersection rules cause much more active search in remote areas, thus leading to global scans.

Looked-but-not-seen and seen-but-not-looked phenomena.

Both phenomena are critical research issues in the field of eye movement research (Groner, 1988). Experiments using eye movement recordings show that people sometimes look at a stimulus and do not see it – as indicated by verbal reports and other behaviour. It is also found that subjects never looked at a stimulus but from verbal reports and other behaviour it is clear that they must have seen the stimulus.

Both types of phenomena occur in driving and both can be explained with DRIVER's perceptual apparatus. Smiley (1989) describes how in a large proportion of accidents occurring during the negotiation of intersections looked-but-not-seen errors played a role.

Looked-but-not-seen errors.

The main reason for this type of error in DRIVER is that fixating the eye on an object is not the same as attending to it. The latter can only be achieved by an attention operator. What happens in DRIVER is that an object that is in the functional field is not visited by an attention operator because a task operator (e.g. navigation) has a higher preference. Once that task operator (or sequence of task operators) is finished the object may no longer be in the functional field so that its attention operator (which has already been generated) has been removed by Soar's TMS mechanism.

Another reason is that drawing out an object is one thing, but doing something with it is quite another. DRIVER might have paid attention to an object (applied a move-attention operator) but is not able to fit this object into the current mental model of the traffic situation. If DRIVER does not know the relevance of an item in the situation it is likely to forget it immediately.

Seen-but-not-looked phenomena.

Soar can test for the presence of operators and do something with that knowledge. DRIVER is thus able to do something with the knowledge that there is an operator for a moving object in the right periphery without moving its eye to that object. This explains in principle the seen-but-not-looked phenomenon, though it is not yet incorporated in DRIVER.

Experienced drivers develop strategies which have not been learned in driving lessons and generate a-normative behaviour.

Driving instructors enforce looking over the right shoulder before turning right to avoid running over cyclists who are going straight on. Once drivers have got their driving licence they tend to ignore this rule and begin to rely on their internal model (De Velde Harsenhorst and Lourens, 1987). It seems they have developed the following rule, which can easily be implemented in DRIVER:

If	I approach an intersection and
	I want to turn right and
	I detected no cyclists close to the intersection
Then	inhibit the move-head operator to look over the right shoulder

Chapter 4 describes how experienced drivers employ fixed strategies in negotiating intersections. The rule sets described in this chapter were especially set up to induce this behaviour and are not discussed any further.

Experienced drivers shift their main field of vision less often in negotiating intersections.

In Chapter 4 we saw that experienced drivers employ fixed strategies in negotiating intersections. In Section 2 of this chapter we demonstrated how these strategies could be induced. Something that has not yet been discussed in the previous section, however, and which is related to this is that experienced drivers shift their main field of vision less often in negotiating intersections (De Velde Harsenhorst and Lourens, 1987; see also Figure 10-1).

The default scheme described in this chapter induces orientation behaviour that shifts the main field of vision very often. The intersection-specific top-down rules were explicitly added to reduce these shifts in more critical situa-

tions. It would be interesting to see whether this phenomenon also occurs for other manoeuvres.

10.5 Discussion

We now conclude the two chapters on lower-level perception and visual orientation with a few notes.

First, finding a fit between DRIVER and the experienced drivers is important, but equally interesting and important is the cooperation between the lower-level perception mechanisms, the lower-level motor modules, the default rules and the top-down intersection rules. Remember that we started out with a few simple constraints from Soar and the perception literature that proved to be enough to build lower-level perception mechanisms on which sensible visual orientation strategies could be built.

The second note is an addition to the previous note. We showed that the visual orientation mechanisms in DRIVER cover a list of various other visual orientation phenomena (See Table 4). Note that DRIVER was specially designed to model actual intersection behaviour. The fact that DRIVER also covers or generates other phenomena is a side effect that gives us the assurance that the general setup of DRIVER is at least a step in the right direction.

Third, DRIVER currently does not cover learning in visual orientation. We saw that in a sense DRIVER was built as if it were an expert system where DRIVER is the expert-system shell and the investigator is the knowledge engineer who provides the knowledge to generate the right behaviour. One of the things that we would like DRIVER to learn in a future version is to discriminate between items that are relevant and items that are not relevant in various driving situations. Thus DRIVER should learn to suppress the operators that look at the wrong items and learn the cues on which to select the right operators. This also implies learning to actively search for the right items and the controlled generation of eye-move and head-move operators.

The reason why learning in visual orientation has not been covered satisfactorily is again our problem with *learning from external interaction*. In the final chapter of this study possible solutions to this problem are discussed.

10.6 Notes

¹ Remember that the representation is a very coarse one. Only the main direction of the visual field is displayed. So a black area does not mean that this is only one eye movement or head movement. It can be one head movement and several small saccades, all directed to the right.

² These default rules should not be confused with the default rules that come with Soar. Soar's own default rules provide basic search control in problem-solving.

¹ The latter happened regularly and understandably before this rule was installed. There were sometimes too many objects behind her to give objects in front of her a chance of ever being noticed.

⁴ Though one could argue that rule [4] to [6] are data driven in the sense that they react to events in the outside world.

¹ Subjects in the De Velde Harsenhorst sometimes preferred another order. This is thus the simulation of a single subject.

11 Speed control

Summary: The present chapter describes how DRIVER's visual orientation and motor control mechanisms are combined in the speed control task.

11.1 Introduction

This chapter describes speed control in DRIVER. This task requires a combination of three major processes. First, the visual orientation process to extract those features from the environment that are relevant for choosing the right speed. Second, the cognitive processes that build up an internal model of the traffic situation such that the right speed decision can be taken by DRIVER's speed control rules. And third, the motor processes that implement a speed decision by manipulating the in-car devices.

The preceding chapters discussed lists of constraints from various sources before the implementation of a particular component was implemented. The present chapter begins with the implementation of speed control in DRIVER. The main reason for this is that much of this implementation is a direct consequence of the way in which visual orientation and motor control are arranged in DRIVER. In addition, not a great deal is really known about the *integration* of perceptual, cognitive and motor processes in speed control. Information is of course available about how situational factors determine speed (See chapter 4, see also Van der Horst, 1990) and there is considerable knowledge about how people perceive and process velocity (Boff, Kaufman and Thomas, 1986)¹, but no integrated treatment of perceptual, cognitive and motor processes in driver speed control exists.

11.2 An overview of DRIVER's speed control

In this section we present the straightforward implementation of speed control in DRIVER. We show how Soar's basic mechanisms plus the apparatus described earlier for motor control and perception are combined into a simple

perceive – decide – act cycle that effectively achieves speed control. DRIVER's global mechanism for speed control is broadly outlined in Figure 11-1.

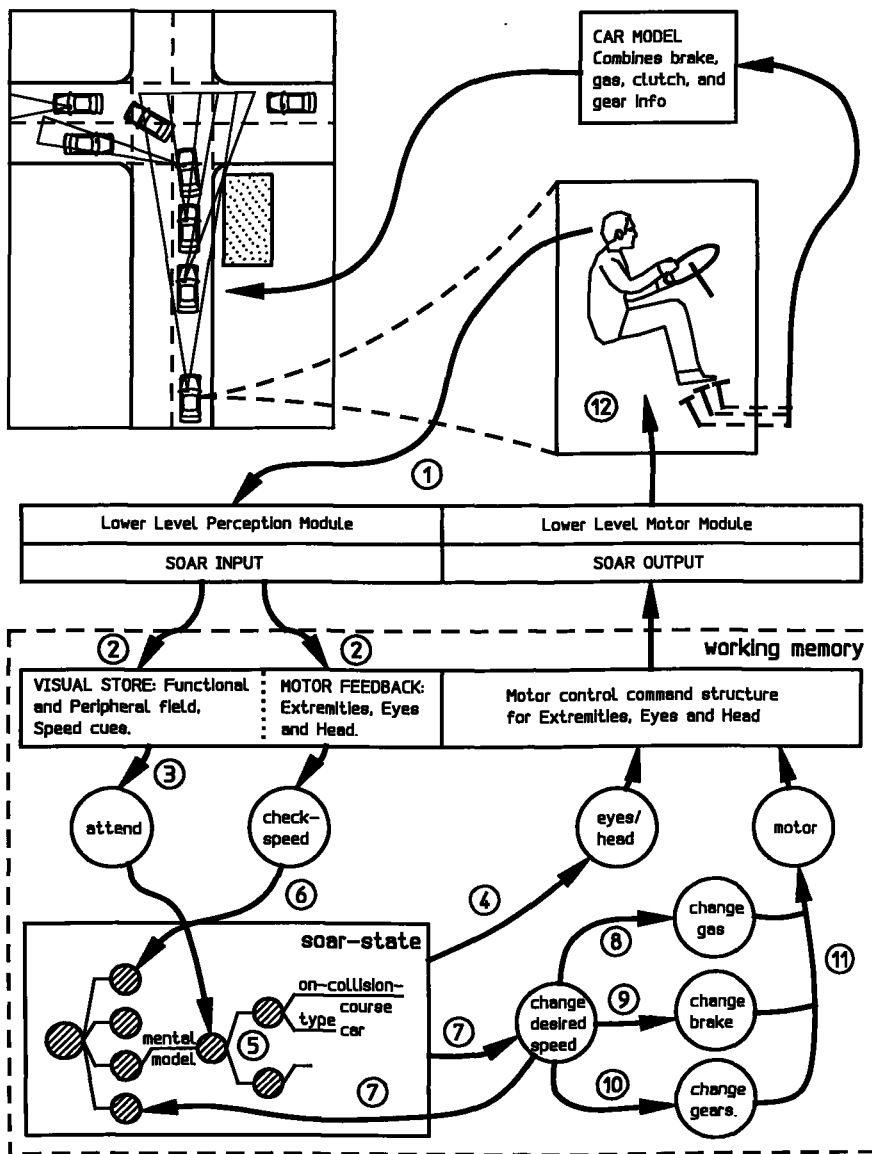


Figure 11-1. The basic perceive-decide-act cycle in speed control. The white circles denote operators. The explanation of this figure is given in the following text.

1-3 The relevant objects in the environment are perceived by the lower-level perception modules (LLPM). Chapter 9 described how objects enter working

memory (WM) via Soar-Input. We saw that these objects do not automatically constitute an internal model of the world. Objects must be *attended to* before they can be used to update the internal model of the world.

3,4 Attend, eye and head-movement operators determine what the LLPM is directed at (i.e. what part of the world is seen). Chapters 9 and 10 described how these operators are generated and selected in the approach to an intersection.

5 Attended objects are, if relevant, used to build up an internal representation of the world in a so-called *internal* or *mental model* on the Soar state. This internal representation is kept at a minimum in DRIVER and exists mainly for the objects seen and simple annotations like 'approaching-an-intersection' and 'car-from-the-right-on-collision-course'.

- To enable DRIVER to reason about speed, knowledge of its own speed and the speed of other objects is required. DRIVER employs three types of knowledge in its speed reasoning. The first type is its estimation of its own speed and the speed of others in numeric format, the second type is its knowledge of what gear its car is in and the third type of knowledge it uses is the perceived sound level of the engine. All these types of knowledge are represented in DRIVER's working memory at the same time.

6 The representation of its current speed is not automatically updated by the LLPM, but is periodically checked by a so-called *check-speed operator*. Speed perception is thus mainly established by polling. The frequency with which this operator is generated and selected is partly determined by an internal clock and partly by other factors, such as too high or too low noise levels or the distance to the intersection.

- The previous version of DRIVER (Chapter 2) declaratively represented desired speed and tolerances for speed deviations. In DRIVER, only the desired speed is declaratively represented in WM. Tolerances are implicitly programmed in productions.

7 The desired speed is changed by a *change-desired-speed* operator when the situation, internal goals and traffic rules so require.

8-10 When a signal is received that the current speed is different from the desired speed, three possible operators are proposed: a change-accelerator, a change-brake or a change-gear operator. The operator that will be chosen depends on the size of the difference between the current speed and the desired speed and the time within which the desired speed must be achieved.

11 The result of the application of these operators is a body plan that generates the appropriate motor command operators. (In the figure we should also have drawn this body plan on the Soar state but we have omitted this here so as not to make the picture too complex).

12 Manipulation of the car finally results in a change of speed.
After this outline, an illustrative Soar trace seems indicated. The following trace brings out some of the speed control issues.

Trace 1. Trace of the speed control cycle.

80 O: 013 CHECK-SPEED

It was time to check speed again but speed is OK.

...

112 O: 0119 (CHECK-SPEED)

DRIVER finds out that it is driving slightly too slow given the current desired speed.

113 O: 0127 (M86 MOTOR-COMMANDS)

Speed is increased by pressing down the accelerator a little.

Moving RIGHTFOOT from ACC-DOWN to ACC-DOWN, dtm = 10, dm = 6

Moving RIGHTFOOT from ACC-DOWN to ACC-DOWN, dtm = 10, dm = 10

115 O: 067 ((AUTO1A) ATTEND)

116 O: 0116 ((AUTO2) ATTEND)

117 O: 037 ((AUTO4) EYE-MOVEMENT-COMMAND)

Moving eye from 1.57 to 1.92 dtm = 0.35 dm = 0.35

118 O: 0137 (CHECK-SPEED)

The speed is OK now, the perceived cars are not relevant to the current speed.

...

...

...

143 O: 0291 ((AUTO1A) EYE-MOVEMENT-COMMAND)

144 O: 0333 ((SIGN1) ATTEND)

145 O: 0336 ((INTERSECTION1) ATTEND)

146 O: 0339 ((TREE1) ATTEND)

147 O: 0342 ((TREE2) ATTEND)

148 O: 0348 ((AUTO4) ATTEND)

DRIVER notices a car on collision course (AUTO4)

and proposes an operator to reduce speed

155 O: 077 (CHANGE-DESIRED-SPEED)

156 O: 074 (CHANGE-GEAR)

A change of gears is required. The chunk that installs the right motor program fires (see chapter on motor control).

157 O: 0102 (M106 MOTOR-COMMANDS) RIGHTFOOT from ACC-DOWN to ACC-UP

Moving RIGHTFOOT from ACC-DOWN to ACC-UP, dtm = 60, dm = 15

Moving RIGHTFOOT from ACC-DOWN to ACC-UP, dtm = 60, dm = 30

158 O: 0103 (M108 MOTOR-COMMANDS) LEFTFOOT from FLOOR to CLUTCH-UP

Moving RIGHTFOOT from ACC-DOWN to ACC-UP, dtm = 60, dm = 45

Moving RIGHTFOOT from ACC-DOWN to ACC-UP, dtm = 60, dm = 60

Moving LEFTFOOT from FLOOR to CLUTCH-UP, dtm = 100, dm = 30

Moving LEFTFOOT from FLOOR to CLUTCH-UP, dtm = 100, dm = 60

Moving LEFTFOOT from FLOOR to CLUTCH-UP, dtm = 100, dm = 90

159 O: 0115 (M117 MOTOR-COMMANDS) TURN WHEEL

11.3 Design decisions and discussion

After this overview we now can discuss the important design decisions in this implementation, referring to constraints from Soar, the traffic science literature and constraints from DRIVER's perception and motor control mechanisms.

Much of the implementation of speed control is a direct consequence of the way in which visual orientation and motor control are arranged in DRIVER. In the following sections we will try to justify the design decisions that are *not* a consequence of these arrangements. The first section discusses the perceptual (and other) information that human drivers use to estimate speed. The next one discusses the information that DRIVER uses. In the third section, we argue why DRIVER employs operators for monitoring and updating speed instead of using only the Soar I/O and productions and in the fourth section we deliberate on why DRIVER needs a mental model in its reasoning about the desired speed. The final section then discusses how these rules operate on this mental model.

11.3.1 Perceptual information in speed control

DRIVER's reasoning about its desired speed requires a representation of its own speed and that of other objects. The first issue is therefore: how do people perceive and estimate speed and velocity², what is the information they use and what information should therefore be included in DRIVER?

To start with, human drivers are notoriously bad at estimating their own speed and that of other objects. It is a well-known fact in traffic science that drivers *overestimate* low speeds and *underestimate* high speeds. In addition, drivers suffer from *speed adaptation*. Adaptation produces systematic underestimates of observed speeds below adaptation level (Denton, 1976). Thus, if drivers drive for a long time on a motorway and then enter a suburban area they will systematically underestimate their speed³. However, humans are able to scale speed. Scialfa, Lawrence, Leibowitz, Garvey and Tyrrell (1991) provide an overview of a number of experiments which show that Stevens's law applies to speed perception; subjects produce psychophysical functions for speed perception similar to those obtained with other intensive continua (see Stevens, 1975). Scialfa et al. report exponents that range from .7 to 1.8.

The fact that drivers are not perfect speed estimators does not answer the question of what the cues are that people use in speed control. In the following sections we will discuss three types of information human drivers seem to use.

The estimation of one's own speed

First, there is evidence that people are able to estimate speed directly from the optic flow pattern (for a general overview see Regan et al., 1986; for a discussion in the traffic literature see Lee, 1976; Van der Horst, 1990). Secondly, there are also reasons for assuming that *proprioceptive information* (feeling the pressure in your back while accelerating) and *auditive information* (the motor noise and wind turbulence) are factors that may be used in estimating speed (Wierda and Aasman, 1991). Thirdly, sometimes drivers do not even need to estimate how fast they are driving, but can *derive* it from the situation they are in. For example, knowing that the 'gear is in three' implies (for certain cars) that speed is between 40 and 60 km/h. An experienced driver will combine this knowledge with the sound of the engine to derive an even more accurate estimate (I'm in third gear and the engine sounds good so I must be doing around 60 km per hour). Another example of derived knowledge: if a driver is on a motorway that is not too busy and speed is roughly the same as the other cars and the driver is not overtaking (or being overtaken) most of the time and the driver is in the Netherlands then speed will be around 110 to 130 kilometres per hour (see also Wierda and Aasman, 1991, for a discussion on this derived knowledge).

The fourth way of estimating one's own speed is, not surprisingly, looking at the speedometer. Drivers look at their speedometer more often than one might expect. Wierda and Aasman (1991) reanalysed data from Jessurun, Steyvers, De Waard, Dekker and Brookhuis (1990) and found that in a straight-road driving experiment drivers looked at their speedometer an average of five times a minute to check their speed.

Estimation of speed of other objects moving in a longitudinal direction

The estimation of speed of other objects moving in a longitudinal direction, that is objects directly approaching or receding from the observer, is important in car following and overtaking. We discussed earlier that Stevens's law applies to speed perception, including the perception of the speed of other objects moving in a longitudinal direction. Besides the fairly direct observation of the speed of others, there are also other cues that are used in practice. Janssen et al. (1976) describes several dependencies between the detection of movement of a leading vehicle and (1) the initial distance between vehicles, (2) the initial angle between the tail-lights of the leading car and (3) the length of time the subject looks at the stimulus⁴.

Estimation of speed of objects moving in a lateral direction

The estimation of the speed of objects moving in a lateral direction is important in handling intersections. The driver must know what the speed is of objects that are approaching on the side roads. Again, people are not perfect at estimating the speed of lateral-moving objects, especially when the object is accelerating positively or negatively (Regan, Kaufman & Lincoln, 1986). The following describes three types of information that drivers seem to use.

The first type of information is the *projection onto the retina*. Runeson (1975) showed that when the projection of an object moves faster than 5 to 6 degrees per second across the retina, velocity is hard to estimate because of blur. When objects move less than 5 to 6 degrees, subjects are very good at estimates for uniform motion. But even non-uniform motion may be estimated as long it is not moving too fast across the retina. Fortunately, in conflict situations at intersections, the speed of an object will in general move less than 5 degrees per second⁷.

The second type of information is *angular information per se*. Drivers are able to use angular information per se in speed estimation (Wierda and Aasman, 1991; Janssen, 1985). If a driver is approaching an intersection that has perpendicular intersecting roads and a vehicle is driving along one of these intersecting roads at a constant visual angle of 45 degrees (making it about the same distance from the intersection) then 'his speed' will be the same as 'my speed'. An increasing angle indicates that the other vehicle is slower; a decreasing angle indicates that the other vehicle is faster. Speed estimation on the basis of angles other than 45 degrees is very complicated. Angular information is especially useful for determining collision courses. Under all conditions a constant angle with an object on an intersecting road always indicates a collision course.

The third type of information drivers seem to use is *time-to collision* information. Van der Horst (1990) demonstrates that people seem to be able to perceive TTC (time to collision computed without an acceleration component) in conflict situations.

11.3.2 Perceptual information used by DRIVER

Following this summary of the types of information used by human drivers, we shall now specify how much of this information is used in DRIVER in its speed control

Estimation of own speed

For the estimation of one's own speed human drivers do have a wealth of (perceptual and cognitive) information at their disposal: from optic flow information to situational cues. So what should DRIVER be given? We finally decided on three perceptual cues and one situational cue. The perceptual information is (1) the direct perception of speed as an integer value, (2) time-to-intersection and (3) the sound level of the engine. The situational or cognitive information is (4) DRIVER's knowledge of which gear it is in. It is a simple matter to demonstrate that we can deal with speed control in DRIVER using only (1), the direct perception of speed as an integer. After all, the semi-intelligent agents in WORLD also only use this cue for speed control. We considered it more interesting, however, to also include the knowledge of the current gear and the sound level of the engine. The most important reason is that it shows that you can use different forms of knowledge alongside one another (something which also appears to be the case for humans).

Estimation of longitudinal speed of other objects

Here we opted for the simplest solution, namely the real speed of the other objects. In fact we take here Stevens's law with an exponent of 1. We thus take no account of Jansen's much more complicated model.

Estimation of lateral speed

In DRIVER I experimented with two forms: *angular information* and *time to collision* (TTC, simply computed as $TTI(him) - TTI(me)$). In the current version of DRIVER I simplify by letting DRIVER perceive the TTI of the other car, indirectly allowing DRIVER the possibility of perceiving TTC.

11.3.3 Using operators for monitoring and updating speed representation

In the description of DRIVER's speed control mechanism we saw that at the lower level there is continuous perception of speed from multiple channels, particularly the absolute speed and engine sound level. What is striking, however, is that these channels have to be attended to by an attention operator, in this case called 'check-speed operator'. Only by applying an attend operator can there be a current speed representation in working memory that can be compared to the desired speed. This is analogous to attention for other perceptual objects. Multiple objects are present in functional and peripheral fields but these objects must first be attended to before they can enter working memory. The consequence of choosing attention (check-speed) operators is that this operator has to compete with attention operators for other perceptual events or motor control actions.

The question we have to answer here, then, is whether we really need to use operators for extracting the speed information. We could have made speed control a completely automatic process by piping speed control information right into working memory. This would certainly have given more operator cycles to other processes and tasks.

There are several reasons why we decided not to put speed information right into WM. First, if we look at De Velde Harsenhorst and Lourens's (1987) data we see that speed control errors, including errors on a straight road, make up 18 % of total errors. Much of the instructor's task seems to focus on getting the subject to attend to his speed. A second reason is derived from Jesserun et al.'s (1990) data, discussed earlier. Drivers in these experiments attended to the speedometer up to 5 times per minute. An example of a situation in which many people regularly look at their speedometer is when driving on a suburban road that 'feels' wide enough to drive at 100 km/h, but the speed regulations force a driver to drive at 50 km/h and it is known that there are regular speed checks by the police. In such a case drivers seem to look at their speedometer all the time (or instead check the environment for signs of the police).

Given our decision to provide DRIVER with an attend (check-speed) operator to check and update the current speed, we must explain how and when to generate and select this operator. The first thing we found is that some kind of clock information is required. If DRIVER is not engaged in a speed-changing operation (such as gear-changing or braking) then checking speed too often takes too many operator cycles away from other processes. DRIVER must therefore have a way of knowing how long it has been since a check-speed operator was applied. Soar currently doesn't have a real sense of time, so a clock mechanism has to be encoded by productions. In DRIVER a simple handle keeps track of the time that has passed since the check-speed operator was applied. During every decision cycle the value on this handle is decremented by one. When the value is zero a check-speed operator is generated. The initial value on the handle is determined by the situation. In a situation where speed is actively changed by an action of the driver it is of course important to check more often than in a situation where the driver intends to drive at a fixed speed. For example, if DRIVER sees that it is approaching an intersection, a production will change this initial value to a lower value.

The above thus provides semi-automatic generation and selection of check-speed operators. However, there are situations where it would be dangerous to wait for the next clock tick. For example, if the road suddenly becomes narrower or the noise of the engine is too low or too high or it is clear from the situation that the speed is too high. In such cases DRIVER will also generate a check-speed operator. The productions that notice these emergency situations are so-called monitor productions (Kuokka, 1990). Such productions were described earlier in Chapter 2.

The above schema thus generates closed-loop behaviour which can be interrupted in two ways. In the first place the seriousness of the situation determines how often a check has to be made. In the second place interruptions can be performed in crisis situations.

11.3.4 The mental model of the external world

DRIVER cannot base its desired speed on the direct perception of objects in the peripheral or functional field. Objects have to be attended to and extracted from those fields before they can be used. These extracted objects then comprise the mental model of the situation. The speed control rules discussed in later sections use the mental model as the basis for speed control decisions. The term *mental model* has gained prominence thanks to the efforts of Johnson-Laird (1983). The idea of a mental model is a reaction to those theories that claim that people use propositional representations in problem solving. A mental model – in Johnson-Laird's conception – lends itself to a representation in semantic networks and thus in Soar (see for example the work of Polk & Newell, 1988). A Soar representation of a mental model consists of objects, relations between objects and so-called annotations. A defining characteristic

of Polk's Soar work is that objects in a mental model cannot be inspected all at once but must be visited by so-called attention operators. Using the mental model representation and the visiting/attention mechanism he is then highly successful in modelling human syllogistic reasoning.

The mental model in DRIVER is more than just the objects extracted from the perceptual fields; it also includes the interpretations of and relations between these objects. Figure 11-2 provides an example of DRIVER's mental model.

The use of mental models in a dynamic task such as driving gives rise to two problems that are not encountered in static problems such as playing chess or solving syllogisms.

The problem of short-term memory decay

The first problem that we encounter in using a mental model to represent the external world is that Soar currently does not have a satisfactory theory of short-term memory decay. In human short-term memory objects disappear automatically if they do not receive any attention, but Soar does not actually have such a mechanism. The problem we are faced with is the overload of DRIVER's working memory if it is continuously seeing new objects. Before we describe how we addressed this problem in DRIVER, we will first describe how Soar removes objects from working memory.

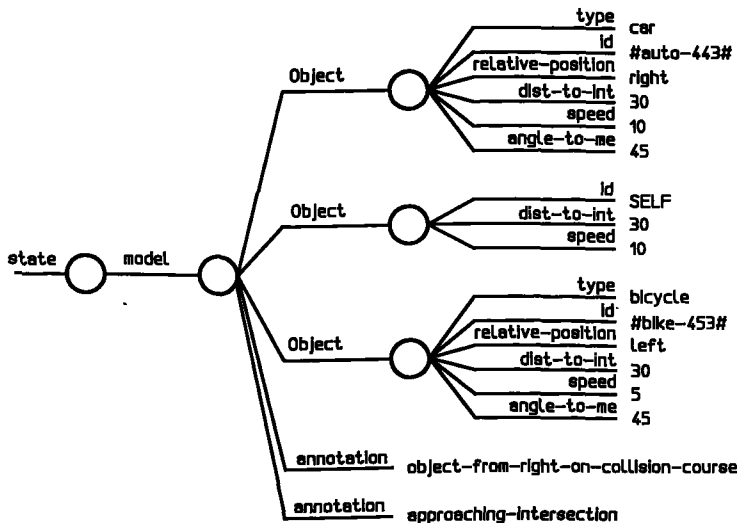


Figure 11-2. The Soar representation of the mental model of the external world in DRIVER. Note how this mental model consists of two types of information. Elaborated objects and simple annotations that describe the situation. Most of the speed decisions are based on these annotations.

The two ways of removing objects from Soar's working memory (WM) are connected to the two ways in which objects in Soar's WM are supported. If the objects in WM came to be there because of the application of an operator, they have so-called O-support and can also only be removed again by an operator application. All objects that have not got into WM as a result of an operation application have so-called I-support (Instantiation support). This means that objects that have entered via Soar-input working memory also have I-support. Objects with I-support are subject to the regime of a justification-based truth-maintenance system that is incorporated in Soar. This means that this system is also responsible for the removal of I-supported objects from Soar's WM⁶ (for a detailed discussion of the mechanisms for I-support and O-support and the consequences of this approach, see Laird et al., 1990). The choice we have for DRIVER, then, is a) to deliberately remove objects from WM, and thus from DRIVER's mental model, or b) to rely on the truth-maintenance system in Soar.

Let us start with the option of relying solely on Soar's truth-maintenance system. Soar in principle allows objects to be extracted from the functional field and stored in WM without an operator application (in the form of an attend operator). The WM representation will then be dependent on the existence of the object in the functional field. However, the problem of what will happen when an eye movement erases the objects in the functional field and thereby the representation in WM (and thus the representation in the mental model) is immediately evident.

So for DRIVER we were forced to use the first option: deleting objects from working memory on purpose. A simple solution is a time trace for every attended object in WM. If the object is not attended to again within a certain time then that object is deleted. DRIVER can attend in two ways to an object in the mental model. The first way is to look again at the object in the external world and attend to the object in the functional field; the second way is by attending to the object in the mental model⁷. This solution, however, has its own drawbacks. We will return to this problem in Chapter 15.

The frame problem

A second problem that we encounter is what has been known as the *frame problem* since 1969 (McCarthy & Hayes, 1981; See Akyürek, forthcoming, for the frame problem in the context of Soar and planning). An explanation of this problem in the context of planning actions in the external world is as follows. Imagine you have a system that attempts to make a plan in which the perception of the external world functions as the basic input. On the basis of this perceptual input inferences are made and on the basis of this inference yet more new inferences are made. In the end an entire structure of interrelated abstractions develops. Now imagine what happens if something suddenly changes in the input. People appear to have little difficulty in adapting the

entire structure of abstractions in line with the new situation. The *frame problem* derives from the fact that artificial systems, and primarily logic-based systems, have very great difficulty in keeping the entire inference structure consistent. It is often necessary to add a great deal of knowledge to a system in order to keep the system consistent for every conceivable change by means of so-called *frame axioms*.

Soar partly avoids the frame problem with its truth-maintenance system; we described in the previous paragraphs how inferences are automatically retained or removed, depending on the continued existence of previous inferences or input. (For more detail about truth-maintenance systems see Doyle, 1979; De Kleer, 1993).

However, a variety of the frame problem does occur in DRIVER. An inference or annotation like *a-car-from-the-right-on-collision-course* is an abstraction based upon the fixed objects in the mental model. Suppose that the annotation is added to the mental model by an operator application. The consequence would then be that the removal of the object on which it is based (for example because it is no longer seen) will not cause the removal of the annotation. The annotation should therefore be removed by an intentional action, if it ought to be removed at all.

The solution chosen in DRIVER is not to fix annotations by operator application but to rely on Soar's truth-maintenance. However, this too has its drawbacks. The main drawback is that we cannot always avoid annotations being fixed. Annotations may be the result of a problem-solving activity, for example an operator no-change impasse, that has been solved in the past. If this is the case then there will always be a fixed annotation. Note that we thus *do* prolong the objects in the mental model by means of an operator application and that in contrast we do *not* prolong the inferences (annotations), but instead rely on truth maintenance.

11.3.5 Traffic rules

In the overall description of speed control in DRIVER we did not say much about the traffic rules used to achieve a correct desired speed. The traffic rules used by DRIVER differ in some important aspects from the rules applied by the agents in WORLD: (1) in WORLD the semi-intelligent agents apply all rules to all other objects every cycle: (2) the output of a rule is a proposal for a negative or positive acceleration; (3) every cycle the lowest acceleration is chosen; and (4) the next clock tick this acceleration is immediately installed.

DRIVER must differ from agents in WORLD, as we argued earlier. First, DRIVER is a relatively slow system that uses its body to manipulate the car and to manipulate the speed. If a speed decision is chosen it takes more than a second before the right motor actions are taken to ensure that a new acceleration is implemented and even then it will take a few seconds before a new

desired speed is reached. Second, DRIVER cannot apply all its traffic rules to all other objects, simply because it cannot see them all at the same time.

Application of traffic rules in DRIVER

There seem to be two major possible ways of applying traffic rules in DRIVER. An automatic/direct method, without using operators, and a controlled/indirect method, with operators. In the first method the traffic rules directly alter the *desired speed symbol* which resides on the top state. Apart from the fact that in Soar all changes to a state are preferably made by means of an operator, the problems that arise when multiple rules fire in the same situation can easily be imagined. We saw in the chapter on motor control that it is not particularly convenient to initialise motor actions directly via productions. This is because there might be very unexpected and dangerous movements, since there is no moment when Soar has the opportunity to consider its choice. The same argument also applies to the direct regulation of speed control.

The method we have chosen in DRIVER, then, employs so-called *change-desired-speed* operators. Such an operator is a *proposal* for changing speed. Using operators, and thus preferences, enables DRIVER to deal with conflicting traffic rules in sub-goals. Speed control is then again reduced to a problem-solving activity.

The rules

All traffic rules are the same in that they generate the same type of change-desired-speed operator. However, for the sake of convenience we distinguish two rule categories. The first category does not involve other moving objects, but is more concerned with the infrastructure of the road and intersection. Examples of these rules are shown in the top half of Table 1. Note that the situations referred to in the rules change very slowly. Operators that are generated by these rules will normally remain in memory as long as they are not applied. The second type of rule involves other moving objects and the rules are basically the same as the rules in the WORLD agents. The bottom half of Table 1 displays some examples of rules that apply in the approach to an intersection.

The change-desired-speed operators will remain only for a short time in working memory as the TMS nature of Soar will remove them from working memory as soon as the objects for which they were generated are removed from the internal model. The following summarizes the Soar implementation of these operators in terms of the generation, selection and application of these operators.

Table 1. Examples of speed control rules in DRIVER.

infrastructure rules

- drive as fast as possible when driving on normal suburban roads and no other conditions apply .
- slow down when approaching intersection and visibility-to-the-right is bad.
- slow down when approaching intersection and turning right or left.

car-car rules

- slow down for cars from the right on collision course
 - slow down for cars from the left when on collision course and too close
 - slow down when turning left and car from opposite direction is going straight on through intersection
-

Generation. A change-desired-speed operator is generated (proposed) whenever a new situation is encountered or a relevant object attended to. The most important parameter on the operator is the proposed desired speed, which is used in the selection of the operators.

Selection. An operator is only selected for application when the proposed speed on that operator is lower than the current proposed speed (residing on the state) and lower than all other change-desired-speed operators residing in memory. However, note that the operator is not rejected or removed from working memory. It will remain in working memory as long as the situation for which it was generated does not change. In this way the operator might still be applied when the desired state on the state is reached.

One characteristic of this selection scheme is that the entire burden rests upon the matcher and the preference mechanism. Conflicts between multiple applicable rules are dealt with at the operator level and not at the data-structure level. So in principle we can use Soar's problem-solving mechanisms to resolve conflicts between operators.

Application. The application of the change-desired-speed operator is obviously the replacement of the current desired speed on the state.

11.4 Notes

¹ In Boff, Kaufman and Thomas's Handbook of Perception and Human Performance, seven chapters are devoted to perceptual aspects of motion perception.

² In the literature a distinction is made between speed and velocity. Velocity is usually defined as a vector quantity indicating the speed and direction of an object or point in a particular direction at a given moment in time (i.e. instantaneous velocity). Speed is a scalar quantity indicating the magnitude of the velocity of a target regardless of its direction.

³ Research by De Velde Harsenhorst and Lourens (1987) and Groeger et al. (1990) reveals that novice drivers have problems with speed control. Part of this problem may be caused by incorrect speed estimates.

For example, if you ask a novice driver how fast he was driving, he will often give the wrong answer. However, this does not prove that the novice is not making a correct estimate. In the first place it may be that the novice does not yet know how to translate his internal estimate into kilo-

metres per hour. In the second place it may be that the novice does not yet know how to relate his internal estimate to the present situation.

¹ One example of such a dependency is the following: suppose the leading vehicle is moving away from the observer. If the subject looks at the tail-lights for 2 seconds he is able to detect a relative movement of 5 km/h when the leading car is 40 metres ahead. At an initial distance of 80 metres he only detects movement when the leading vehicle is travelling 9 km/h faster than the observer. At 160 metres he detects only a speed difference of 30 km/h. The least noticeable difference increases rapidly to 80 km/h at a distance of 320 metres and to 100 km/h at 640 metres.

⁵ If I am stationary 40 metres before a crossing, a car coming from the right 40 metres from the crossing at a speed of 50 km/h (13 m/s) will move across my retina at approximately 11 degrees per second. If I am also driving at 50 km/h, this figure will be 0 degrees. In the majority of intersection situations we will both be moving, so normally the speed will be below six degrees per second. However, estimation of the lateral speed is then made more difficult by the fact that you are also moving at speed yourself.

⁶ An example of I-support and truth-maintenance: imagine objects A and B in the *working* memory and productions *if A & B then C* and *if C then D*. If A disappears out of the memory, then according to the principles of a truth-maintenance system C too, and then D, will be removed from the memory, because the conditions of the rule that generated C and D are no longer satisfied.

⁷ One consequence is that by varying the time that an object is allowed in WM without attention we can also vary the number of objects in WM.

12 Steering and Lane Keeping

Summary: The present chapter describes how DRIVER's visual orientation and motor control mechanisms are combined in the steering and lane-keeping task.

12.1 Introduction

This chapter describes lane keeping and steering in DRIVER. We define *steering* as the perceptual-motor task of changing the direction of the car by means of steering movements; *lane keeping* is defined as the task of keeping the car within the boundaries of a road lane. This chapter is considerably shorter than the previous one; the reason is that the lane-keeping mechanism is actually a simplified version of the speed-control mechanism. Furthermore, the motor-control mechanism for steering movements is simpler to describe than the motor-programming and execution mechanisms involved in gear-changing. So, although we do not deal with these mechanisms as extensively as we did for speed control, we will nevertheless demonstrate that DRIVER has the right mechanisms to address lane keeping and steering.

The structure of this chapter is the same as that of the previous chapter. We will use a global figure to describe the basic *perceive – decide – act* cycle. We will then use this description as a basis for making a number of further remarks.

12.2 An overview of steering and lane keeping in DRIVER

DRIVER's global mechanism for speed control is broadly outlined in Figure 12-1. The numbers in the first column of the following description refer to the numbers in the figure.

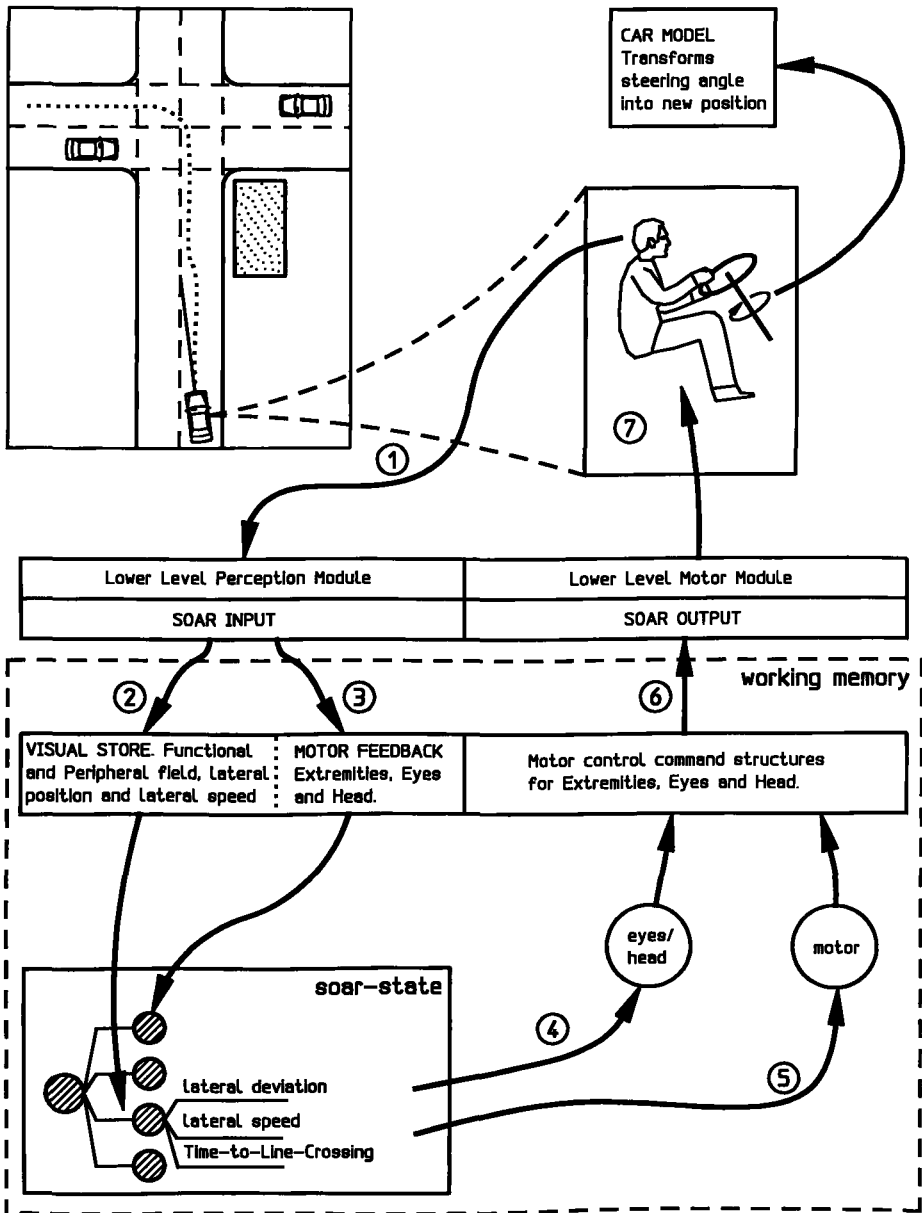


Figure 12-1. The basic perceive-decide-act cycle in steering and lane keeping. Section 2 is entirely devoted to the explanation of this figure.

1 DRIVER perceives several cues that may be used in lane keeping. The cues that it currently perceives are heading angle, lateral deviation from its ideal

course, lateral speed and Time-to-Line-Crossing (TLC). The latter cue is an indication of how long it will take before the boundary of a lane is crossed.

2,3 These cues reach the Soar state directly via the Soar Input. No attention operators are used for retrieving this information from the visual store. There is of course also motor feedback, which keeps track of the internal model of the body and hence also of the position of the hands on the steering wheel.

4 Since DRIVER can only perceive these cues when it is looking to the front, provision is made for an eye-move operator to be generated regularly to make it look to the front.

5 The steering movements made by DRIVER are initiated by a simple motor operator. In the current version of DRIVER this operator is generated when the lateral deviation from the ideal course becomes too great (for an explanation of the concept of *ideal course* see Section 3.2). It is however also possible to use the lateral speed and the TLC to generate motor operators.

6 The arm movements, and hence the steering movements, are simulated via the Soar Output and the Lower Level Motor Module.

7 Finally, the Car model ensures that the changing steering angles also result in a new position on the road.

12.3 Discussion

The following discusses some aspects of the mechanism for steering and lane keeping. In Section 12.3.1 we will deal with the cues that DRIVER perceives and uses. Section 12.3.2 covers how DRIVER performs left- and right-hand turns at intersections, since the previous section actually dealt only with maintaining course on a straight road. Finally, Section 12.3.3 describes the difference compared to the speed-control mechanism.

12.3.1 Cues used

DRIVER perceives four cues that it may use in lane keeping: heading angle, lateral position, lateral speed and time-to-line-crossing. The use of these cues is discussed in a number of recent dissertations relating to cognitive control strategies and visual cues in steering (Godthelp, 1984; and Riemersma, 1987). The general questions in the research on control strategies in steering involve (1) the frequency with which the driver visually checks that he is on course, (2) the accepted path error before corrective actions are taken and (3) the cues involved in the perception of path errors.

Time-to-Line-Crossing (TLC)

One covering all these general research questions is Godthelp (1984). Godthelp notes that most of the available vehicle control models are based on the fundamental assumption that drivers steer their vehicle in a continuous error-correction mode with permanent visual feedback, i.e. closed-loop. However, he continues, it is now also commonly accepted that driving cannot be considered as a continuous closed-loop task of this kind. Under many circumstances driving does not require permanent path error control or the driver may be forced, temporarily, to pay (visual) attention to other aspects of the driving task. Two important sets of distinctions that he makes are *open* versus *closed-loop* visual control and *error-correction* versus *error-neglection* strategies.

Open-loop versus closed-loop control

From a servo-theoretical point of view, a system operates in an open-loop mode whenever information about actual system behaviour is unavailable. Although in humans cues other than visual ones play a role in steering, his study uses the terms open and closed loop in relation to the availability of visual feedback: steering during the absence of visual information is referred to as open-loop control, whereas the term closed loop is used for steering with visual feedback. Note that drivers may use their internal representations of their position on the road in steering without visual feedback.

Error-neglection versus error-correction

A driver will not always react immediately to momentary path errors. This strategy may be the result of a voluntary decision to ignore path errors or the complexity of the driving task may force the driver to do so. Both situations will result in passive, no-steering periods. In Godthelp's study a driver's failure to act upon path errors is referred to as error-neglection¹, whereas the strategy to minimize path errors is defined as error-correction.

Godthelp's thesis studies the potential role of visually open-loop strategies and error-neglection in vehicle control. He shows that the time available for a driver to control his vehicle in an open-loop mode, i.e. without immediate visual feedback, largely depends on (1) the accuracy of the open-loop-generated steering-wheel action and (2) the time available for error-neglection.

The time available for error-neglection can be analysed by application of the path prediction techniques as commonly used in preview-predictor models. Based on this technique the Time-to-Line-Crossing (TLC) can be calculated, representing the time available for the driver to neglect path errors, until the moment in time at which any part reaches one of the lane boundaries. The concept of TLC quantifies the potential of neglecting path error in driving and can be used to answer two important questions: first, how long are drivers actually *willing to risk controlling* their vehicle without immediate visual feed-

back and, second, how long are drivers ultimately *allowed* to wait before switching over to the error-correction mode.

Godthelp investigated the first question by measuring drivers' self-chosen occlusion times in a straight lane-keeping task with constant speeds varying between 20 and 120 km/h.² Occlusion times appeared to correspond closely with TLC. In the same analysis it was shown that drivers choose occlusion times that can be described as a constant fraction, i.e. 40 %, of the available time. In another experiment Godthelp asked the subjects to correct course only if the vehicle motion could still comfortably be corrected to prevent a crossing of the lane boundary (driver should neglect path errors). The strategy adopted by drivers in this task is to switch over to error-correction at a roughly constant TLC distance (1.3 seconds) from the lane boundary (in the range from 20 to 120 km per hour).

Although TLC is a descriptive measure, it may be argued that it is also a perceivable cue used in steering control, just as TTC or TTI are possible direct perceivable cues (van der Horst, 1990). Though TLC might be considered to be a cue it is not explained how people perceive or compute TLC from other cues.

Lateral position, lateral speed, heading angle

Riemersma (1987), in his thesis on visual cues in straight road driving, describes how the optical correlates of lateral speed, lateral position and heading rate³ are used in course control. However, it appears likely that the heading angle itself is not controlled directly but remains within a suitable small range as a consequence of controlling the other cues. The main outcome of his study is that experts use lateral position and lateral speed (which might be used in the computation of TLC). Beginners, however, seem to be unable to use lateral speed.

Cues that DRIVER uses

By providing DRIVER with the perception of all the aforementioned cues and the mechanism discussed in Section 2, we gave DRIVER the hooks to implement various strategies that experiment with (1) the frequency with which DRIVER visually checks that it is on course, (2) the accepted path error before corrective actions are taken and (3) the cues involved in the perception of path errors.

In Section 2 we outlined the simplest strategy for reacting only when the lateral position exceeds a particular standard value. It is almost as simple, however, to use TLC as the most important cue. In order to prevent DRIVER from deviating too much from its course there are a number of rules that ensure that DRIVER regularly looks in the driving direction (see the default orientation rules in Chapter 10 on visual orientation).

Even with the simple strategy described in Section 2, DRIVER does not work in a continuous error-correction, closed-loop mode (with permanent visual feedback). Close to the intersection DRIVER is forced to look at other things as well and thus operates in an open-loop, error-neglection mode.

12.3.2 Handling curves

What was not discussed in Section 2 is how DRIVER handles curves in a left turn, right turn or in obstacle avoidance.

DRIVER has an internal representation of an ideal course which is represented both in DRIVER's WM and in its lower-level perception module. This ideal course is a course plan that consists of a linked list of course points to aim for. This ideal course is one of the factors used to determine the path error (lateral position). Figure 12-2a shows DRIVER's ideal course in a left-hand manoeuvre. The manoeuvre involves not only turning left, but also getting in lane. In Figure 12-2b we see an abstraction of the linked list in DRIVER's WM. Figure 12-2c shows one of the advantages of this approach. If the local situation forces DRIVER to change its ideal course (e.g. to avoid a parked car), then it is fairly simple to insert a number of new course points in this list. This ideal course is also represented in DRIVER's lower-level perception module, so that it can keep constant track of what the deviation from the ideal position is.

We have no psychological pretensions regarding our approach. Many of the models of steering behaviour relating to steering round bends have been formulated very mathematically and offer few, if any points of departure for the design of a more procedural, cognitive model. One point that these models do seem to offer, however, is how people look when handling bends. A systematic study of driver steering behaviour by Reid and Solowka (1981) shows how people, when avoiding (sudden) obstacles or when steering along a winding road display looking behaviour as though they are following an ideal course with ideal course points.

12.3.3 Differences between the mechanisms for speed control and steering

The main difference for the *perceive – decide – act* cycle is that in steering and lane keeping attend operators are not used to obtain the most important cues from the functional field. The reason for using attend operators in speed control was the fact that checking speed is a mentally loading task. This is not to suggest that checking the lateral deviation or the lateral speed (or the TLC) is not a mental loading task. There are many studies demonstrating that steering in itself is a mental loading task⁴. DRIVER's steering is in itself also a loading task in the sense that the more often a steering movement (and hence a motor command operator) is executed, the more often another task is interrupted.

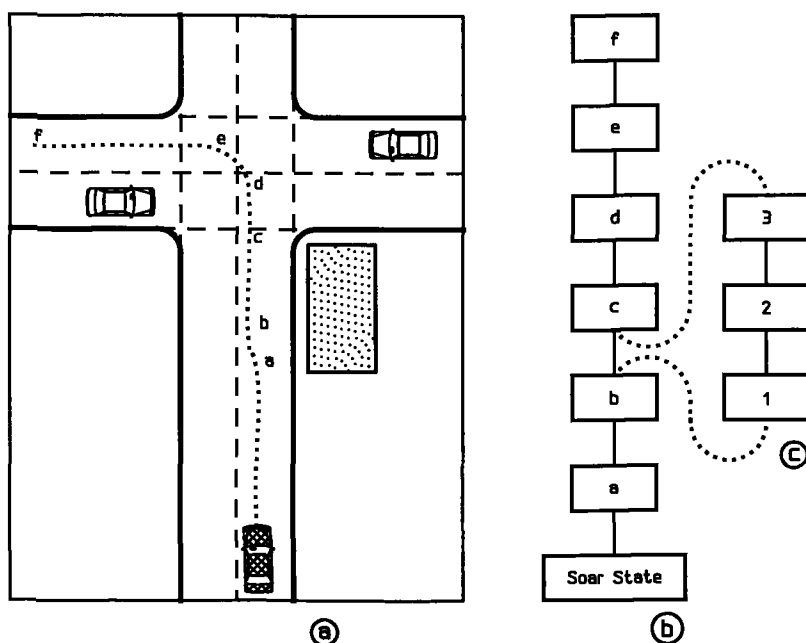


Figure 12-2. Figure 2a shows DRIVER's ideal course in a left-turn manoeuvre, Figure 2b shows an abstracted version of this list in DRIVER's working memory. Figure 2c shows how new course points are inserted in this list when DRIVER is forced to avoid an obstacle.

We decided, however, to have speed control performed as automatically as possible without attend operators. One aspect that is the same as the speed control mechanism is the use of so-called monitor productions. These productions constantly check a particular parameter and only become active when a particular standard value is exceeded. In speed control this is a change-speed operator, in lane keeping and steering a steering movement (by means of a motor command).

12.4 Notes

¹ We will use Godthelp's term *error-neglection* although it is not a proper noun.

² In this occlusion experiment subjects wear an electromagnetically driven visor mounted on a lightweight bicycle helmet. The visual field is occluded by a sheet of translucent drawing paper mounted on a frame, which can be raised or lowered on command by the subject.

³ Heading rate is defined as the change in heading angle, where heading angle is defined as the difference between the road axis and the longitudinal axis of the car.

⁴ Many researchers have focused on the *steering wheel reversal rate* (SRR), which is a measure of the number of times per minute that the direction of the steering wheel movement is reversed through a small, finite angle. There seems to be a positive relationship between SRR and driving

task demand. McDonald (1979) provides a list of regularities that have been reported in the literature. For example, SRR increases with an index of traffic-event density; SRR increases during encounters with oncoming vehicles; SRR is higher with decreasing lane width, increasing speed, and reduced preview. When task difficulty remains constant, it has been found that lower levels of driver capability are often associated with higher SRR. Thus, inexperienced drivers have been found to have higher rates than experienced drivers. A similar pattern has been shown for high-accident drivers and fatigued drivers. When the total task situation becomes very complex, for example because of the traffic situation or the addition of a (non-driving-related) secondary task, then SRR decreases.

There is also a number of studies that report research on the effect of mental load on the quality of driving performance (both for driving a car or driving a bicycle.) A complex trade-off was found between steering performance, speed, performance on a mental loading task (e.g. counting or continuous loading tasks such as the pasat) and signal detection parameters (Noordhof, 1989; Wierda et al., 1987; Brookhuis et al., 1989; Maring, 1988). The general finding was that the addition of a complex mental task to the driving task increases the steering amplitudes and signal detection errors and decreases the speed. Though steering performance almost always degrades with the addition of a mental loading task there was usually also a slowdown in mental task performance due to driving. These effects are more pronounced in young children and elderly drivers.

13 Navigation

Summary: Chapter 2 showed that external interaction interferes with the execution of goal-stack intensive search tasks such as navigation. We blamed Soar's default rules for this. Chapter 3 discussed the problems with Soar's default rules and proposed alternative default rules. The present chapter discusses navigation and the alternative default rules that enable DRIVER to navigate while performing external interaction.

13.1 Introduction

Route-finding or navigation is included in DRIVER because it is a main driver sub-task. However, the inclusion of the navigation task serves more purposes than just route-finding. First, as in our earlier model described in Part I, it provides a task for studying multitasking issues such as task interruption, task resumption, and the efficiency of default search rules. We saw in our earlier model that Soar's default rules prohibited efficient task-switching. Chapter 3 addressed this problem and discussed alternative default rule sets that (a) limit both the size of working memory and the depth of the goal stack and (b) are far more resistant to task interruption and resumption. The alternative rule set employed in DRIVER is the so-called 'what-if one-level' approach. In this version we built in a deliberate reversal and rejection mechanism that provides Soar with both progressive deepening and a simple form of error recovery.

A second and related reason for including navigation is that it provides a measure of the cognitive load of the perceptual and motor tasks. Both navigation and the perceptual and motor tasks involve the use of operators in the base level space. The addition of the navigation task in critical situations, for example the approach to an intersection, will have its consequences for motor and perceptual task performance or the performance on the navigation task itself.

Section 13.2 discusses navigation in human drivers even though the emphasis in this chapter is more on navigation as a problem-solving task than on navigation itself. We will describe one current prominent theory of human navigation that can fairly easily be mapped onto an architecture like Soar. Section 13.3 discusses the implementation of navigation in DRIVER. This section describes the details of the internal map and summarises the default search mechanisms. In this section we again encounter an error-recovery issue in the context of external interaction. This time, however, learning from external interaction can be handled by an extension of the default rules. This chapter also discusses how the navigation task may be interrupted with a minimum of performance degradation.

13.2 Navigation in human drivers

People are not perfect navigators, as indicated by the current research into electronic route guidance systems. King and Lunenfeld (1974) investigated real-life navigation in slightly unfamiliar territory and found that 30 percent of all subjects in their study had got lost during their last trip and 50 percent had thought they were lost. Not only are people bad navigators but navigation is also a mentally taxing task. Van Winsum (1987) showed in a field study that navigation is a task which demands considerable controlled attention and Färber and Färber (1986) presented evidence that a driver who is lost displays more insecure and dangerous driving behaviour.

Navigation is a standard example of a search task in technically oriented AI textbooks and algorithm courses. Unfortunately, less is known about the implementation of navigation in humans down to what Anderson (1990) calls the algorithmic level. However, important cognitively oriented research has been done into the cues that people use in navigation¹ (Lynch, 1960; Evans, 1980), the reference systems people use in real-life navigation² (Alm, 1990) or the types of knowledge involved in navigation. Kuipers' theory of navigation and mapping in large-scale space (Kuipers and Levitt, 1988; Kuipers, 1978; Kuipers, 1982) distinguishes between four types of spatial knowledge:

- *Sensorimotor knowledge* supports recognition of landmarks from a strictly egocentric point of view (e.g. go left at this traffic light)
- *Procedural knowledge* is knowledge that specifies how to find and follow routes. The knowledge is stored in so-called "travel-plans".
- *Topological knowledge* is a description of the environment in terms of fixed entities, such as places, paths, landmarks, and regions, linked by topological relations, such as connectivity, containment, and order. At this level of description the traveller is able to go beyond strictly egocentric sensorimotor experience. The traveller is able to recognise places as being the same, despite different viewpoints; identify places as being on a single path, in a particular order; define boundary regions to the left or right of a path.

- *Metric knowledge* is a description of the environment in terms of fixed entities, such as places, paths, landmarks, and regions, linked by metric relations, such as relative distance, relative angle, and absolute angle and distance with respect to a frame of reference. Using metric knowledge a driver is able, for instance, to infer that place A is south of place B, that a turn should be made at a sharp angle and that a particular route is two kilometres long.

The above summary of Kuipers' knowledge types is borrowed from Schraagen (1990), who found in his study on the use of different types of map information for route-following in unfamiliar cities that subjects are able to use topological knowledge but are less able, or less willing, to use metric knowledge. In a related study Alm (1990) found that subjects primarily use egocentric or mixed references and that the most important cues that subjects use are local landmarks such as traffic lights, shops, bridges and petrol stations.

Since Kuipers' theory forms the basis of a computational theory of navigation we shall take a look at how it could fit into a computational framework such as ACT* or Soar. At the lower level *sensorimotor knowledge* provides the basic information that we are at a certain path or node in an internal map of the environment. Without this type of knowledge navigation would clearly be impossible. *Procedural knowledge* translates into Anderson's compiled procedures or Soar's chunked preference schemes. Drivers going from home to their work undoubtedly possess these overlearned travel plans. Kuipers' theory becomes unclear at the level of *topological knowledge*. He does not make a distinction between the structural components of this knowledge and the procedures that operate on these components. However, with some good will these terms can be translated into declarative structures (representing places, paths, etc.) and knowledge (connectivity, transitivity rules), which operate on these structures in order to derive the so-called "travel-plans". The *metric knowledge* in Kuipers' theory implies that people are able to use very specific metric information in their spatial reasoning, suggesting that metric representations should be incorporated in a model of navigation. Schraagen's research, however, indicates that we probably do not need to do this, but should rely on more relative and procedural relations within an internal map.

13.3 Implementation of navigation in DRIVER

Although for people navigation is a combination of (1) looking for information on which a "travel plan" can be based, (2) the planning itself and (3) the implementation of a plan, in this section we shall deal with the various parts separately. First we will discuss the so-called "internal navigation", i.e., in Soar terms, the problem-solving required to arrive at a plan. Then we will deal very briefly with how an internal plan relates to the implementation of a plan in the external world. And, finally, we will show how DRIVER is sufficiently flexible to adapt if an intersection in the external world is found to be blocked.

13.3.1 "Internal" navigation

The task environment that DRIVER must be able to navigate is obviously **WORLD**. As mentioned earlier, this world currently contains intersections and T-junctions. DRIVER knows this environment, though it is often surprised that intersections or T-junctions are shut off due to construction. Houses and buildings, traffic lights and traffic signs may function as landmarks.

Task representation

The problem-space *Navigate* defines DRIVER's internal map of the environment and the procedures for searching through it. The internal map of its environment is basically a graph on which *nodes* are connected by *paths*. Nodes represent intersections or T-junctions and are stored on the state (even within the internal state there can only be one node stored on the state). Paths are generated from nodes and stored on operators. This means that if DRIVER knows which paths lead from a certain node it will generate operators to follow that path. Connectivity relations between distant nodes (A can be reached from B) or containment relations (C lies on the path from A to B) can be established by searching for a path between the nodes. Once the search has been successful the connectivity or containment may be stored in a chunk.

The search heuristic that DRIVER employs is a mix of a global and an egocentric reference system. In other words the surrounding nodes are firstly evaluated with regard to their position in an absolutely 2-dimensional coordinate system and secondly according to whether they are located to the left, to the right, in the same direction or behind the present position. One interesting aspect of the *Navigate* space is that it is a *shared* space; the Base-level problem space which is in general the space where motor control and perception are handled is also the base space for navigation. In practical (Soar) terms a shared space is a problem space that has multiple names, enabling multiple rule sets from different problem spaces to fire within the same goal context. The advantage of shared spaces is that navigation operators can be applied, intermingled with motor and perceptual operations. Only when insufficient knowledge is available to decide between paths will Soar impasse and go into a sub-space³.

The Default Rules

The reasons for rejecting Soar's default rules were discussed both in the chapter on our earlier model and in Chapter 3, on flattening goal hierarchies. This chapter extensively discussed various alternatives to Soar's default search rules. DRIVER employs the *what-if* method that restricts the depth of the goal stack to one level. Although all the alternatives given in Chapter 3 in principle have the same functionality as Soar's original default rules, the *what-if* method is found to be the most flexible for dealing with progressive deepening and error recovery. The general algorithm behind this rule set is summarised here again because it provides the introduction to the section in which we discuss

what happens when a learned path does not match reality (i.e. an intersection is blocked or a road is no longer there).

(1) Soar's preferences are now generated from explicit preferences which are stored on the operators themselves. (2) When multiple operators for the same task are in working memory then so-called *what-if* operators are generated for each operator that does not have an explicit preference. What-if operators are mutually indifferent but have a higher preference than the original task operators. (3) A random what-if operator is chosen, resulting in a no-change impasse. In the resulting sub-goal the super-space is proposed independently of the what-if operator and then preferred because of the what-if operator. This scheme enables chunks to be learned that do not contain the what-if operator in their left-hand side. (4) A copy of the current task state (which resides on the top state) is made. (5) The technique described in (3) is used to propose the original task operators in the sub-space. The operator on the what-if operator in the super-operator slot is preferred and applied. (6) The evaluation of this evaluation is copied up to this task-operator, resulting in a chunk that has the following form:

If	problem space P has name PNAME and state S has properties P1 to Pn and proposed operator O has name ONAME and arguments arg 1 to arg n
Then	add evaluation E with value V to O

In Chapter 3 we also described the price that has to be paid in the approaches that limit the depth of the goal stack. The first price is the loss of magical backtracking as provided in the original default rules. By limiting the depth of the goal stack it is easy to end up in a dead branch of the search tree. Soar (and thus DRIVER) needs to invest effort in storing operators in an operator list and in creating and applying reversal operators. The second price is that Soar has to invest more effort to learn and is obliged to engage in explicit learning, which in DRIVER is at present mainly rejection, though other evaluations are also possible. The third price, finally, is that the detection of duplicate states requires more elaborate mechanisms. These three issues were discussed in detail in Chapter 3, Section 1, and will not be repeated here.

A part of the street network that DRIVER lives in is shown in the left half of Figure 13-1. The dotted line gives the search path from origin N1 to destination N9. The search path is not an optimal one as a result of the one-level look-ahead and the heuristics used. Trace 1 in the appendix provides a trace of DRIVER planning its route from node N1 to N9. Notice how the stack never goes deeper than one level. Trace 2 shows the same run after DRIVER has learned the search control rules.

13.3.2 "External" navigation:

The navigational problem-solving described above is entirely *internal*, i.e. it is all done "in the head." *External* navigation in DRIVER amounts to the execution of the travel-plan in WORLD. The literature often does not make this external/internal distinction. Navigation generally implies both the external search for clues or landmarks and the internal problem-solving that uses these clues.

The execution of travel plans is fairly simple. DRIVER passively recognises intersections and relates these to nodes in its internal map. DRIVER currently does not actively search for information. One interesting aspect of navigation in DRIVER, reflecting navigation in human drivers, is that the travel-plan does not have to be finished before a decision is made. If the internal search for a destination is not finished before the next intersection, then DRIVER will just take the first node of the travel-plan which is currently the best one (though it might get worse upon further search).

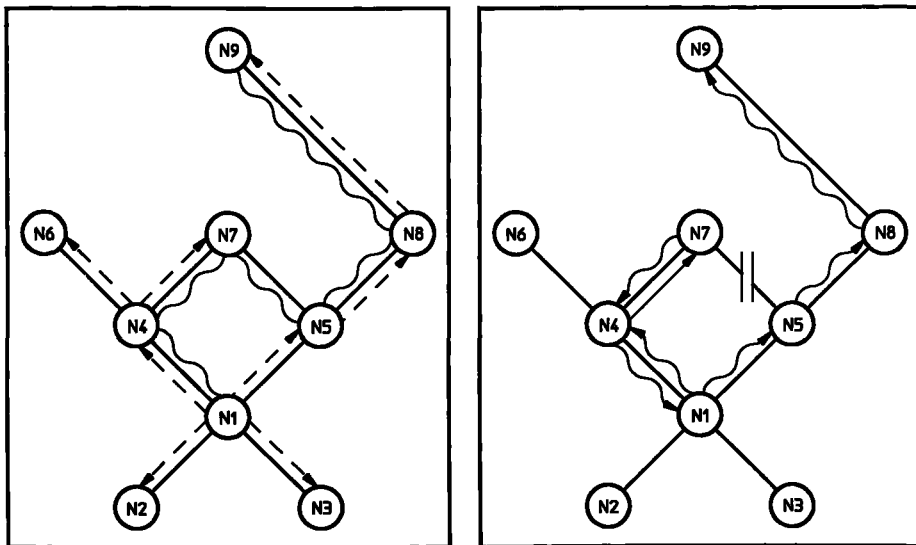


Figure 13-1. The street network and DRIVER's search path. Figure 1a shows the search when the connection between N7 and N5 is intact. Dotted arrows represent visits in sub-spaces, the solid lines represent the actual route taken. In Figure 1a a non-optimal path is chosen as a result of the one-level look-ahead and heuristics used. Figure 1b shows how DRIVER is forced to replan its route when it finds that the road between N7 and N5 is blocked. The traces included in this chapter provide far more detail (see also following text).

13.3.3 Plan repair or error recovery in Navigation.

DRIVER drives in a familiar environment where all the relevant intersections are known. It might be thought that in a way this makes navigation in DRIVER fairly uninteresting were it not that DRIVER's navigation provides a demon-

stration of the default rules and an example of a task that may be interrupted without severe penalties. However, there is one other feature in DRIVER's navigation that makes navigation an interesting task and that is the capability of repairing learned plans when during external navigation it finds out that paths in that learned plan do not exist or are blocked.

Error recovery in navigation does not require additional complex mechanisms. The same alternative set of default rules can be used without modification. The annotated Trace 3 in the appendix and the right half of Figure 13-1 shows what happens when DRIVER is forced to replan when one of the nodes on the learned path is missing (in our example the path from N7 to N5). First the old chunks will guide the search along the old path, then DRIVER will notice that the path from N7 to N5 no longer exists and it will learn to reject that path using the deliberate rejection technique described in 3.1. From that point on DRIVER will search for other possibilities and find another route. Note how halfway through the search DRIVER picks up the old path again (from N5 to N8) and goes directly to its destination.

13.4 Task interruption and resumption in navigation

The model of driver behaviour described in Chapter 2 centred almost entirely on the multitasking aspects of driving. We concluded that the use of Soar's standard default rules degraded task performance in navigation due to excessive times for rebuilding the goal stack after interrupts. A second conclusion was that the overhead of task-switching at the top level was overly expensive because of Soar's restriction that data structures on the state may not be destructively modified⁴. It will be clear by now that both these problems are addressed in DRIVER. DRIVER is implemented in a Soar version that allows for destructive modification and DRIVER's default rules were especially designed to overcome the problems of task-switching. Trace 5b displays a search that is continuously interrupted by eye and body movements.

Trace 5a shows how a trace looks when the navigation operator is protected against interrupts such as eye or movement operators. This does not mean that no eye movements or body movements are possible during navigation, just that these movements must be initiated before the navigation operator. The protection is achieved by giving navigation a higher preference than move operators *once* it is in the operator slot. Thus, before the navigation operator is installed it has equal rights when compared with other operators. The slowdown in navigation due to task-switching is obviously minimal in this approach, as the evaluation of one navigation operator (and the learning of a chunk) is allowed to finish before other operators are allowed.

In Trace 5b we see what happens when the navigation operators are not protected against interrupts. In this approach movement operators have

higher priority than navigation operators (plus the power to terminate the navigation operator and thus the navigate problem space). We see that both performance and learning are seriously hindered because the evaluation of a navigation operator may be interrupted before the chunk is built. This alternative is probably the safest in critical situations, but it is also the most detrimental to (navigation) task performance. The previous two traces suggest that only fixed strategies are possible (i.e. only favour navigation, only favour movements), but other strategies are also possible.

A third alternative would be a combination of one and two. We install both the protection and DRIVER's active control. In this scheme DRIVER determines at run time⁵ what the primary and secondary tasks are. The primary task will then be protected as long it is safe or useful to do so. However, this is only possible in situations that are not too dangerous, otherwise accidents may occur⁶.

13.5 Discussion and future research

It will be clear that the implementation of navigation in DRIVER is not primarily meant to reflect human navigation to the last detail. Not enough is known about human task performance in navigation to allow this to be done.

Another caveat is that navigation in this chapter is almost entirely restricted to search in the "map in the head" (Kuipers, 1982). Search in an unfamiliar environment is an entirely different task and not covered in DRIVER. The latter type of search involves for example the visual search for clues and landmarks. It is clear that in human navigation internal and external navigation are combined. A description of how people might combine various external and internal search strategies is given by Pailhous (1970). Pailhous presents a cognitively oriented account of how taxi drivers find their routes in Paris. He discusses how these drivers seem to mix search on an internal map with cues from the major landmarks of Paris and with dead-reckoning. One of the positive achievements of navigation in DRIVER is that internal and external search are already combined to some extent. Internally DRIVER plans complete routes but if during the execution of a plan it finds out that an intermediate node is not accessible DRIVER is flexible enough to replan on the fly.

13.6 Notes

¹ Lynch (1960) carried out some studies to determine what aspects of a large-scale environment people use and learn. Using different methods, Lynch concluded that people seem to classify large-scale environments into five basic categories.

- Landmarks: Objects that can easily be seen from a distance: monuments, towers, certain buildings, etc.
- Paths were defined as routes people travel along. Streets, railways, canals are examples of paths.

- Nodes were defined as points where several paths meet. Examples are intersections and roundabouts.
- Districts were defined as regions similar in one or many aspects. The similarities may be of a perceptual nature (colour, form) or abstract (for instance cultural similarity).
- Edges were defined as boundaries of districts or other areas. Examples of edges are rivers, walls and motorways.

² In this context it is possible to make a distinction between global, local and egocentric reference systems (Gärting and Golledge, 1989). A global reference system is one that can be applied all over the world, such as the north-south axis. A local reference system can be used on a smaller, local scale. The position of different categories can be defined in terms of a local referent, for instance a highly visible landmark. An egocentric reference system can be used on an even smaller scale, since it starts with the position of the observer and defines the position of objects in relation to the observer's position. For instance left-right in relation to the observer.

³ Within this sub-space navigation is, of course, no longer shared. The concept of 'shared spaces' was until recently a heretical one in the Soar research community. The multitasking chapter will discuss this issue in more detail.

⁴ The only way to change data structures was to build a new state and copy parts of the old structure along with new substructures.

⁵ on the fly, by problem-solving. At run time possibly sounds too compiler-like

⁶ A fourth alternative would be to have bursts of multiple movements by using DRIVER's buffered motor input. Remember that motor commands for the same extremity might be given before the old one is finished. Discussing the psychological evidence for these alternatives would lead us right into a discussion on mental strain in driving and multitasking, so we will postpone this discussion until chapters 14 and 15.

Appendix

Trace 1. Learning the path from N1 to N9 using the what-if default rules.

```

0 G: G1
1 P: P2
2 S: S3
3 O: O5 (INIT-TASK)
4 O: Q19 (WHAT-IF)
5 -- > G: G31 (OPERATOR NO-CHANGE)
6 P: P2 (NETWORK)
7 S: S33
8 O: O11 (NETWORK-MOVE)
   What-if: going from N1 to N2 at (-100,-100)
9 O: Q35 (EVAL-NETWORK-MOVE)
Build:P118
10 O: Q22 (WHAT-IF)
11 -- > G: G37 (OPERATOR NO-CHANGE)
12 P: P2 (NETWORK)
13 S: S39
14 O: O13 (NETWORK-MOVE)
   What-if: going from N1 to N3 at (100, -100)
15 O: Q41 (EVAL-NETWORK-MOVE)
Build:P124
16 O: Q25 (WHAT-IF)
17 -- > G: G43 (OPERATOR NO-CHANGE)
18 P: P2 (NETWORK)
19 S: S45
20 O: O15 (NETWORK-MOVE)
   What-if: going from N1 to N4 at (-100,100)
21 O: Q47 (EVAL-NETWORK-MOVE)
Build:P140
22 O: Q28 (WHAT-IF)
23 -- > G: G49 (OPERATOR NO-CHANGE)
24 P: P2 (NETWORK)
25 S: S51
26 O: O17 (NETWORK-MOVE)
   What-if: going from N1 to N5 at (100,100)
27 O: Q53 (EVAL-NETWORK-MOVE)
Build:P145
28 O: O15 (NETWORK-MOVE)
TOP-LEVEL: going from N1 to N4 at (-100,100) evaluations: 0
29 O: Q61 (WHAT-IF)
30 -- > G: G70 (OPERATOR NO-CHANGE)
31 P: P2 (NETWORK)
32 S: S72
33 O: O55 (NETWORK-MOVE)
   What-if: going from N4 to N6 at (-200,200)
34 O: Q74 (EVAL-NETWORK-MOVE)
Build:P146
35 O: Q64 (WHAT-IF)
36 -- > G: G76 (OPERATOR NO-CHANGE)
37 P: P2 (NETWORK)
38 S: S78
39 O: O57 (NETWORK-MOVE)
   What-if: going from N4 to N7 at (0,200)
40 O: Q80 (EVAL-NETWORK-MOVE)
Build:P147
41 O: O57 (NETWORK-MOVE)
TOP-LEVEL: going from N4 to N7 at (0,200) evaluations: 2
42 O: Q89 (WHAT-IF)
43 -- > G: G92 (OPERATOR NO-CHANGE)
44 P: P2 (NETWORK)
45 S: S94
46 O: Q84 (NETWORK-MOVE)
   What-if: going from N7 to N5 at (100,100)
47 O: Q96 (EVAL-NETWORK-MOVE)
Build:P149
48 O: Q84 (NETWORK-MOVE)
TOP-LEVEL: going from N7 to N5 at (100,100) evaluations: -2
49 O: Q104 (WHAT-IF)
50 -- > G: G113 (OPERATOR NO-CHANGE)
51 P: P2 (NETWORK)
52 S: S115
53 O: Q98 (NETWORK-MOVE)
   What-if: going from N5 to N1 at (0,0)
54 O: Q117 (EVAL-NETWORK-MOVE)
Build:P171
55 O: Q110 (WHAT-IF)
56 -- > G: G119 (OPERATOR NO-CHANGE)
57 P: P2 (NETWORK)
58 S: S121
59 O: Q102 (NETWORK-MOVE)
   What-if: going from N5 to N8 at (200,200)
60 O: Q123 (EVAL-NETWORK-MOVE)
Build:P172
61 O: Q102 (NETWORK-MOVE)
TOP-LEVEL: going from N5 to N8 at (200,200) evaluations: 0
62 O: Q132 (WHAT-IF)
63 -- > G: G135 (OPERATOR NO-CHANGE)
64 P: P2 (NETWORK)
65 S: S137
66 O: Q127 (NETWORK-MOVE)
   What-if: going from N8 to N9 at (0,300)
67 O: Q139 (EVAL-NETWORK-MOVE)
Build:P173
68 O: Q127 (NETWORK-MOVE)
TOP-LEVEL: going from N8 to N9 at (0,300) evaluations: 2
End -- Explicit Halt

```

Trace 2. The same run after learning

0 G: G1
 1 P: P2
 2 S: S3
 Firing P118 P124 P140 P145
 3 O: O13 (NETWORK-MOVE)
 TOP-LEVEL: going from N1 to N4 at (-100,100) evaluations: 0
 Firing P146 P147
 4 O: O37 (NETWORK-MOVE)
 TOP-LEVEL: going from N4 to N7 at (0,200) evaluations: 2
 Firing P149
 5 O: O49 (NETWORK-MOVE) TOP-LEVEL: going from N7 to N5 at (100,100) evaluations: -2
 Firing P171 P172
 6 O: O72 (NETWORK-MOVE) TOP-LEVEL: going from N5 to N8 at (200,200) evaluations: 0
 Firing P173
 7 O: O79 (NETWORK-MOVE) TOP-LEVEL: going from N8 to N9 at (0,300) evaluations: 2
 End -- Explicit Halt

Trace 3. A trace after breaking the connection between N7 and N5.

0 G: G1
 1 P: P2
 2 S: S3
 3 O: O5 (INIT-TASK)
 Firing P118 P124 P140 P145
 4 O: O15 (NETWORK-MOVE)
 TOP-LEVEL: going from N1 to N4 at (-100,100) evaluations: 0
 Firing P146 P147
 5 O: O41 (NETWORK-MOVE)
 TOP-LEVEL: going from N4 to N7 at (0,200) evaluations: 2
 6 O: O6 (WAIT)
No chunks fire to propose an operator for N5 because the connection is broken. A wait operator is installed instead.
 7 O: O53 ((NETWORK-MOVE) NETWORK-MOVE)
 TOP-LEVEL: going back from N7 to N4
 Firing P146 P147
Soar is forced to go back from N7 to N4 and immediately the chunk for N4 fires again. However, this time the what-if default rules force the way to N7 to be unlearned.
 8 O: O74 (LEARN-REJECT)
 9 ==>G: G31 (OPERATOR NO-CHANGE)

10 P: P32 (LEARN-REJECT)
 11 S: S94
 12 O: O98 (REJECT-OPERATOR)
 13 O: O127
 (RECOGNISE-STATE-AND-OPERATOR)
 14 O: O144
 (LEARN-REJECT*BUILD-CHUNK)
 Build:P213, P215, P221 *These three chunks express that going from N4 to N7 is wrong.*
 Firing P213
 15 O: Q107 (WHAT-IF)
 16 ==>G: G37 (OPERATOR NO-CHANGE)
 17 P: P2 (NETWORK)
 18 S: S105
 19 O: O59 (NETWORK-MOVE)
 What-if: going from N4 to N1 at (0,0)
 20 O: O148 (EVAL-NETWORK-MOVE)
 Build:P222
 21 O: O55 (NETWORK-MOVE)
 TOP-LEVEL: going from N4 to N6 at (-200,200) evaluations: 0
N6 now has the best evaluation so Soar goes to N6
 22 O: O6 (WAIT)
However, from here no other nodes can be reached
 23 O: O150 ((NETWORK-MOVE) NETWORK-MOVE)
 TOP-LEVEL: going from N6 to N4
And Soar retracts to N4 again.
 Firing P146 P147 P213 P222 P146
Soar learns that in the future N6 should be avoided.
 24 O: O155 (LEARN-REJECT)
 25 ==>G: G43 (OPERATOR NO-CHANGE)
 26 P: P44 (LEARN-REJECT)
 27 S: S134
 28 O: O161 (REJECT-OPERATOR)
 29 O: O167
 (RECOGNISE-STATE-AND-OPERATOR)
 30 O: O170
 (LEARN-REJECT*BUILD-CHUNK)
 Build:P223, P36, P384
 Firing P181 P223
 31 O: O6 (WAIT)
Soar is now back in N1 and decides to start all over again.
 32 O: O174 (INIT-TASK)
 Firing P118 P124 P140 P145
 33 O: O182 (NETWORK-MOVE)
 TOP-LEVEL: going from N1 to N4 at (-100,100) evaluations: 0

*DRIVER goes to N4 again because it hasn't yet
learned that N4 leads only to dead branches.*

34 O: O6 (WAIT)

35 O: O273 ((NETWORK-MOVE)

NETWORK-MOVE)

TOP-LEVEL: going back from N4 to N1

Firing P118 P124 P140 P145 P140

Learn that going from N4 to N1 is wrong

36 O: O279 (LEARN-REJECT)

37 ==>G: G76 (OPERATOR NO-CHANGE)

38 P: P77 (LEARN-REJECT)

39 S: S327

40 O: O285 (REJECT-OPERATOR)

41 O: O344

(RECOGNISE-STATE-AND-OPERATOR)

42 O: O347

(LEARN-REJECT*BUILD-CHUNK)

Build: P97, P399, P400

Firing P97

43 O: O278 (NETWORK-MOVE)

TOP-LEVEL: going from N1 to N5 at (100,100)

evaluations: 0

Firing P171 P172

And from here on the path is known

44 O: O349 (NETWORK-MOVE)

TOP-LEVEL: going from N5 to N8 at (200,200)

evaluations: 0

Firing P173

45 O: O351 (NETWORK-MOVE)

TOP-LEVEL: going from N8 to N9 at (0,300)

evaluations: 2

End -- Explicit Halt

Figure 13.2. A typical evaluation chunk

```

(SP P118
  (GOAL <G1> ^OBJECT NIL ^STATE <S1>
 ^OPERATOR <O1> + ^PROBLEM-SPACE <P1>)
  (OPERATOR <O1> ^NAME NETWORK-MOVE ^TO
  <I2>)
  (PROBLEM-SPACE <P1> ^NAME NETWORK
 ^TASK-OPERATOR-NAMES NETWORK-MOVE)
  (O <I2> ^NAME N2 ^X -100 ^Y -100)
  (STATE <S1> ^NETWORK-STATE <N1>)
  (O <N1> ^DESTINATION-INTERSECTION <D1>
 ^CURRENT-INTERSECTION <C1>
 ^INITIAL-INTERSECTION <I1>)
  (O <D1> ^NAME N9 ^X 0 ^Y 300)
  (O <C1> ^NAME N1 ^Y 0)
  (O { <> <D1> <C1> } ^X 0)
  ..>
  (OPERATOR <O1> ^NUMBER-VALUE -2 &, -2 +
 ^EVALUATION NUMBER &, NUMBER +))

```

Trace 4. A small sample trace and specimen chunk learned in error correction.

```

44 O: O100 (LEARN-REJECT)
45 --> G: G92 (OPERATOR NO-CHANGE)
46 P: P93 (LEARN-REJECT)
47 S: S112
48 O: O127 (REJECT-OPERATOR)
49 O: O151 (RECOGNISE-STATE-AND-OPERATOR)
50 O: O155 (LEARN-REJECT*BUILD-CHUNK)
Build:P149
Build:P171
Build:P172
Firing P149
(SP P149
  (GOAL <G1> ^OPERATOR <O1> + ^STATE
  <S1>)
  (OPERATOR <O1> ^NAME NETWORK-MOVE ^TO
  <I2>)
  (STATE <S1> ^NETWORK-STATE <N1>)
  (O <N1> ^DESTINATION-INTERSECTION <D1>
 ^CURRENT-INTERSECTION <I1>)
  (O <D1> ^NAME N9)
  (O <I1> ^NAME N4)
  (O <I2> ^NAME N1)
  ..>
  (OPERATOR <O1> ^EVALUATION REJECT &,
  REJECT +))

```

Trace 5a. Protection against interrupts

```

0 G: G1
1 P: P2
2 S: S3
3 O: O5 (INIT-TASK)
4 O: O19 (WHAT-IF)
5 --> G: G31 (OPERATOR NO-CHANGE)
6 P: P2 (NETWORK)
7 S: S33
8 O: O11 (NETWORK-MOVE)
  What-if: going from N1 to N2 at (-100,-100)
9 O: O35 (EVAL-NETWORK-MOVE)
Build:P118
The move-eye operator has to wait until the problem-solving episode has ended.
10 O: MOVE-EYE (Auto-4)
11 O: O22 (WHAT-IF)
12 --> G: G37 (OPERATOR NO-CHANGE)
13 P: P2 (NETWORK)
14 S: S39
15 O: O13 (NETWORK-MOVE)
  What-if: going from N1 to N3 at (100,-100)
16 O: O41 (EVAL-NETWORK-MOVE)
Build:P124
17 O: MOVE-ATTENTION (Auto-4)
The move-attention operator also has to wait.

```

Trace 5b. No protection against interrupts

```

0 G: G1
1 P: P2
2 S: S3
3 O: O5 (INIT-TASK)
4 O: O19 (WHAT-IF)
5 --> G: G31 (OPERATOR NO-CHANGE)
6 P: P2 (NETWORK)
7 S: S33
8 O: O11 (NETWORK-MOVE)
  What-if: going from N1 to N2 at (-100,-100)
9 O: MOVE-EYE (Auto-4)
When there is no protection against interrupts the move-eye operator does not have to wait until the problem-solving episode has ended. No chunk is learned.
10 O: MOVE-ATTENTION (Auto-4)
DRIVER will have to start all over again with O19.
11 O: O19 (WHAT-IF)
12 --> G: G37 (OPERATOR NO-CHANGE)
13 P: P2 (NETWORK)

```

14 S: S39

15 O: O13 (NETWORK-MOVE)

What-if: going from N1 to N2 at (-100, -100)

16 O: O41 (EVAL-NETWORK-MOVE)

Build:P124

And this time it succeeds.

14 Integration and multitasking

Summary: This chapter discusses the integration of the tasks presented in earlier chapters. Soar traces are provided of DRIVER at work, using the machinery discussed so far. The traces are used to evaluate the fit between DRIVER and the human drivers from Chapter 4. We also use the traces to explain DRIVER's multitasking mechanisms.

14.1 Introduction

The purpose of this chapter is to show how all the sub-tasks covered in the previous chapters are eventually integrated in DRIVER. Figure 14-1 shows how the previous chapters relate to the present chapter. The chapters on lower-level motor control, motor planning, lower-level perception and visual orientation are clearly the pillars upon which DRIVER rests. The first form of integration of visual orientation and motor planning was performed in the chapters that discussed speed control and steering. In this chapter we discuss the final integration step, namely the combination of speed control, steering and navigation. Chapter 3 is included in the figure because the navigation task would not be possible without the alternative default rules. Chapter 4 is included because the ultimate purpose of DRIVER is for its behaviour to fit with our empirical data.

The layout of this chapter is as follows. To give the reader an impression of how DRIVER performs several tasks simultaneously, we begin in Section 14.2 with a Soar trace of DRIVER at work. In Section 14.3 we then describe how DRIVER's behaviour fits with the behaviour of the subjects described in Chapter 14.4. In Section 4 we describe how DRIVER's multitasking works. As in Chapter 2, DRIVER's multitasking is a simple form of *task-switching* in Soar's base level space. Section 14.4 will discuss the rules (and heuristics) that are responsible for *switching* to the appropriate *task operator* at the right time. In this section several learning aspects of multitasking will also be discussed. In Section 14.5, the discussion, we will look again at the differences between DRIVER and the model in Chapter 2. A full discussion of the psychological

validity of this approach is postponed until the next chapter, in which our multitasking mechanism will be evaluated in a broader context.

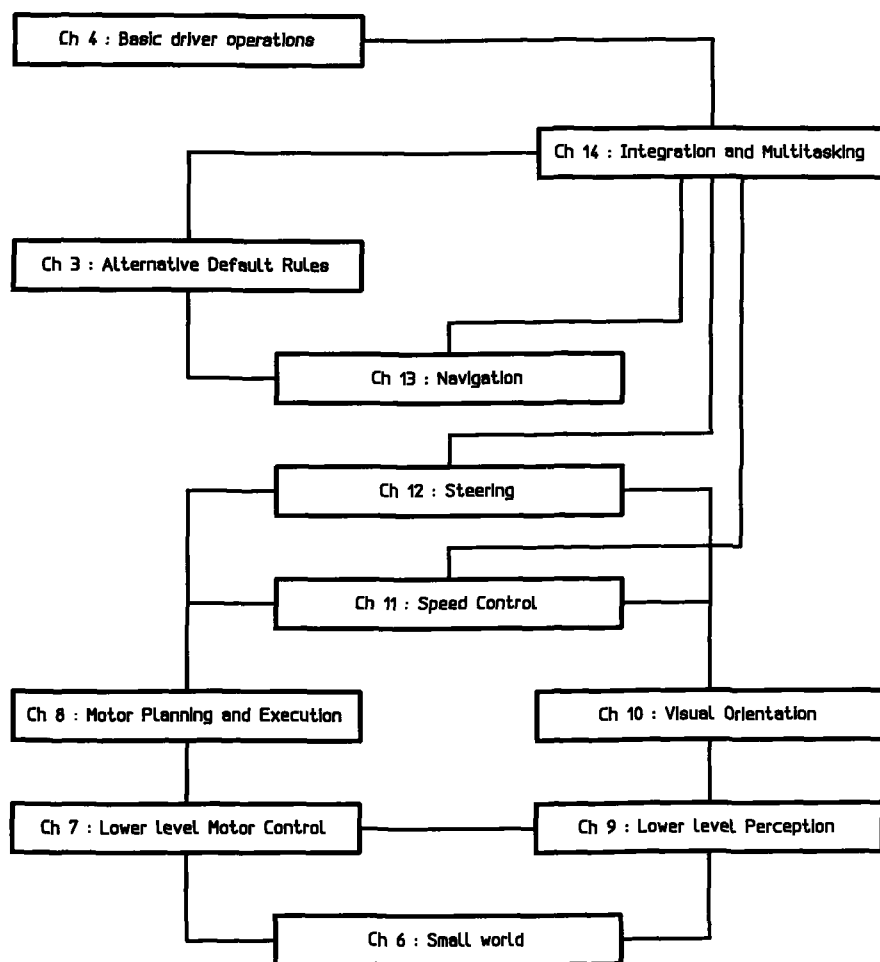


Figure 14-1. How the foregoing chapters lead to this chapter.

14.2 A trace of DRIVER's behaviour

Trace 1 is a Soar trace of DRIVER's overall behaviour in the approach to an intersection. It is a rather lengthy trace but I would urge readers to work through it in order to get an impression of the various activities that DRIVER handles simultaneously. The reader will recognise most of the trace, as it does

not in itself contain any new elements. All elements have figured in earlier traces. The only difference is that the elements are now integrated.

Summarizing briefly, what happens is the following: DRIVER sees an intersection and commences the navigation task. However, before it can start to think properly about the choice at the next intersection, DRIVER is forced to look around a little and adjust its speed. From cycle 22 onwards DRIVER starts the navigation task, which is only occasionally interrupted by the speed adjustment (cycles 27, 28) or by eye movements and attention operators (such as cycles 42 to 47). In cycle 55 DRIVER finally decides to turn left and also begins the preparations for a left-turn manoeuvre. In cycle 66 a change-gear manoeuvre begins. The difference from the trace of the change-gear manoeuvre in Chapter 8 is that a large number of steering movements and eye movements now take place in between.

Trace 1. DRIVER making a right turn.

0 G: G1

1 P: P2

2 S: S3

3 O: 049 ((TREE2) ATTEND)

4 O: 074 ((INIT-NAVIGATION-TASK))

After recognizing that an intersection is approaching, DRIVER starts the navigation task. In the following trace we will see that DRIVER does not have the heuristics to choose the right direction immediately, so it will need to sub-goal.

5 O: 093 (CHECK-SPEED)

However, before DRIVER can start the first what-if operator it checks its speed and finds that it has to change its desired speed.

6 O: 084 (CHANGE-DESIRED-SPEED)

Before changing the desired speed it attends to its surroundings.

7 O: 052 ((TREE1) ATTEND)

8 O: 055 ((INTERSECTION1) ATTEND)

9 O: 058 ((SIGN1) ATTEND)

10 O: 061 ((HOUSE4) ATTEND)

11 O: 064 ((HOUSE1) ATTEND)

12 O: 0119 (CHECK-SPEED)

13 O: 0127 (M86 MOTOR-COMMANDS)

DRIVER can achieve the new desired speed simply by pressing the accelerator a little more. Note that it now executes the command given in step 5.

Moving RIGHTFOOT from ACC-DOWN to ACC-DOWN, dtm = 10, dm = 6

Moving RIGHTFOOT from ACC-DOWN to ACC-DOWN, dtm = 10, dm = 10

14 O: 0129 (M143 MOTOR-COMMANDS)

And some more.

0129 moves RIGHTFOOT from ACC-DOWN to ACC-DOWN

Moving RIGHTFOOT from ACC-DOWN to ACC-DOWN, dtm = 10, dm = 6

Moving RIGHTFOOT from ACC-DOWN to ACC-DOWN, dtm = 10, dm = 10

15 O: 067 ((AUTO1A) ATTEND)

16 O: 0116 ((AUTO2) ATTEND)

17 O: 037 ((AUTO4) EYE-MOVEMENT-COMMAND)

Moving eye from 1.57 to 1.92 dtm = 0.35 dm = 0.35

18 O: 0137 (CHECK-SPEED)

The speed is OK now.

19 O: 0264 ((AUTO3C) ATTEND)

20 O: 0266 ((AUTO4) ATTEND)

And finally there is time for the what-if operator for the navigation task. Notice that in this trace the sub-goaling is protected against interrupts.

21 O: Q102 (WHAT-IF)

22 -- > G: G122 (OPERATOR NO -CHANGE)

23 P: P2 (NETWORK)

24 S: S124

25 O: 094 (NETWORK-MOVE) What-if: going from N1 to N2 at (-100,-100)

26 O: 0269 (EVAL-NETWORK-MOVE)

Build: P130

DRIVER has evaluated the first option and built a chunk that remembers the evaluation for that move.

27 O: 0270 (CHECK-SPEED)

DRIVER evaluates speed and finds that it is still too slow.

28 O: 0271 (M253 MOTOR-COMMANDS)

Moving RIGHTFOOT from ACC -DOWN to ACC -DOWN, dtm = 10, dm = 6

Moving RIGHTFOOT from ACC -DOWN to ACC -DOWN, dtm = 10, dm = 10

After this motor action there is time for the evaluation of the second option.

29 O: Q105 (WHAT-IF)

30 -- > G: G131 (OPERATOR NO -CHANGE)

31 P: P2 (NETWORK)

32 S: S133

33 O: 096 (NETWORK-MOVE) What-if: going from N1 to N3 at (100, -100)

34 O: 0277 (EVAL-NETWORK-MOVE)

Build: P138

35 O: 0279 (CHECK-SPEED)

36 O: Q108 (WHAT-IF)

37 -- > G: G139 (OPERATOR NO -CHANGE)

38 P: P2 (NETWORK)

39 S: S141

40 O: 098 (NETWORK-MOVE) What-if: going from N1 to N4 at (-100,100)

41 O: 0280 (EVAL-NETWORK-MOVE)

Build: P146

42 O: 0283 (CHECK-SPEED)

43 O: 0291 ((AUTO1A) EYE-MOVEMENT-COMMAND)

Moving eye from 1.92 to 1.57 dtm = 0.35 dm = 0.35

44 O: 0294 ((AUTO1A) ATTEND)

45 O: 0322 ((AUTO2) ATTEND)

46 O: 0330 ((HOUSE1) ATTEND)

47 O: 0332 ((HOUSE4) ATTEND)

After a rather long interrupt from its navigation task, in this case by eye movements and attention, there is finally time to evaluate the last option.

48 O: Q111 (WHAT-IF)

49 -- > G: G147 (OPERATOR NO -CHANGE)

50 P: P2 (NETWORK)

51 S: S149

52 O: 0100 (NETWORK-MOVE) What-if: going from N1 to N5 at (100,100)

53 O: 0286 (EVAL-NETWORK-MOVE)

Build: P154

The last option has been evaluated.

54 O: 0289 (CHECK-SPEED)

55 O: 098 (NETWORK-MOVE) Going from N1 to N4

And DRIVER can make a move at the top level.

56 O: 0351 (CHECK-SPEED)

57 O: 0333 ((SIGN1) ATTEND)
58 O: 0336 ((INTERSECTION1) ATTEND)
59 O: 0339 ((TREE1) ATTEND)
60 O: 0342 ((TREE2) ATTEND)
61 O: 0348 ((AUTO4) ATTEND)
62 O: 0357 ((AUTO3A) ATTEND)

At this point driver is 100 meters from the intersection

63 O: 049 ((TREE2) ATTEND)
64 O: 076 (CHECK-SPEED)
65 O: 077 (CHANGE-DESIRED-SPEED)

DRIVER is now so close to the intersection that it reduces its desired speed enough to turn right at the intersection.

66 O: 074 (CHANGE-GEAR)

A change of gears is required. The chunk that installs the right motor program fires (see chapter on motor control).

67 O: 0102 (M106 MOTOR-COMMANDS) RIGHTFOOT from ACC -DOWN to ACC-UP
Moving RIGHTFOOT from ACC -DOWN to ACC-UP, dtm = 60, dm = 15
Moving RIGHTFOOT from ACC -DOWN to ACC-UP, dtm = 60, dm = 30
68 O: 0103 (M108 MOTOR-COMMANDS) LEFTFOOT from FLOOR to CLUTCH -UP
Moving RIGHTFOOT from ACC -DOWN to ACC-UP, dtm = 60, dm = 45
Moving RIGHTFOOT from ACC -DOWN to ACC-UP, dtm = 60, dm = 60
Moving LEFTFOOT from FLOOR to CLUTCH -UP, dtm = 100, dm = 30
Moving LEFTFOOT from FLOOR to CLUTCH-UP, dtm = 100, dm = 60
Moving LEFTFOOT from FLOOR to CLUTCH -UP, dtm = 100, dm = 90
69 O: 0115 (M117 MOTOR-COMMANDS) TURN WHEEL
Moving LEFTFOOT from FLOOR to CLUTCH -UP, dtm = 100, dm = 100
Moving LEFTHAND from WHEEL to WHEEL, dtm = 6, dm = 6
Moving RIGHTHAND from WHEEL to WHEEL, dtm = 6, dm = 6
70 O: 0126 (M127 MOTOR-COMMANDS) RIGHTFOOT from ACC -UP to BRAKE-UP.
Moving RIGHTFOOT from ACC -UP to BRAKE-UP, dtm = 100, dm = 30
Moving RIGHTFOOT from ACC -UP to BRAKE-UP, dtm = 100, dm = 60
Moving RIGHTFOOT from ACC -UP to BRAKE-UP, dtm = 100, dm = 90
71 O: 0132 (CHECK-SPEED) RIGHTFOOT from ACC -UP to BRAKE-UP
Moving RIGHTFOOT from ACC -UP to BRAKE-UP, dtm = 100, dm = 100
72 O: 052 ((TREE1) ATTEND)
73 O: 055 ((INTERSECTION1) ATTEND)
74 O: 058 ((SIGN1) ATTEND)
75 O: 0123 ((AUTO2) ATTEND)
76 O: 037 ((AUTO4) EYE-MOVEMENT-COMMAND)
Moving eye from 1.57 to 2.32 dtm = 0.75 dm = 0.47
Moving eye from 1.57 to 2.32 dtm = 0.75 dm = 0.75
77 O: 0164 ((AUTO3C) ATTEND)
78 O: 0174 (CHECK-SPEED)
79 O: 0170 ((AUTO4) ATTEND)
80 O: 0152 ((AUTO2) EYE-MOVEMENT-COMMAND)
Moving eye from 2.32 to 1.63 dtm = 0.69 dm = 0.47
Moving eye from 2.32 to 1.63 dtm = 0.69 dm = 0.69
81 O: 0205 ((AUTO2) ATTEND)
82 O: 0208 ((HOUSE3) ATTEND)
83 O: 0211 ((SIGN1) ATTEND)
84 O: 0214 ((INTERSECTION1) ATTEND)
85 O: 0228 (CHECK-SPEED)
86 O: 0190 ((AUTO3A) EYE-MOVEMENT-COMMAND)
Moving eye from 1.63 to 0.83 dtm = 0.79 dm = 0.47
Moving eye from 1.63 to 0.83 dtm = 0.79 dm = 0.79
87 O: 0238 ((AUTO2) EYE-MOVEMENT-COMMAND)

- Moving eye from 0.83 to 1.67, dtm = 0.83, dm = 0.47
 Moving eye from 0.83 to 1.67, dtm = 0.83, dm = 0.83
- 88 O: 0133 (M134 MOTOR-COMMANDS) RIGHTFOOT from BRAKE -UP to BRAKE-DOWN
 Moving RIGHTFOOT from BRAKE -UP to BRAKE-DOWN, dtm = 55, dm = 15
 Moving RIGHTFOOT from BRAKE -UP to BRAKE-DOWN, dtm = 55, dm = 30
 Moving RIGHTFOOT from BRAKE -UP to BRAKE-DOWN, dtm = 55, dm = 45
- 89 O: 0129 (M130 MOTOR-COMMANDS) LEFTFOOT from CLUTCH -UP to CLUTCH-DOWN
 Moving RIGHTFOOT from BRAKE -UP to BRAKE-DOWN, dtm = 55, dm = 55
 Moving LEFTFOOT from CLUTCH -UP to CLUTCH-DOWN, dtm = 60, dm = 15
 Moving LEFTFOOT from CLUTCH -UP to CLUTCH-DOWN, dtm = 60, dm = 30
 Moving LEFTFOOT from CLUTCH -UP to CLUTCH-DOWN, dtm = 60, dm = 45
- 90 O: 0104 (M110 MOTOR-COMMANDS) RIGHTHAND from WHEEL to STICK3
 Moving LEFTFOOT from CLUTCH -UP to CLUTCH-DOWN, dtm = 60, dm = 60
 Moving RIGHTHAND from WHEEL to STICK3, dtm = 100, dm = 30
 Moving RIGHTHAND from WHEEL to STICK3, dtm = 100, dm = 60
 Moving RIGHTHAND from WHEEL to STICK3, dtm = 100, dm = 90
 Moving RIGHTHAND from WHEEL to STICK3, dtm = 100, dm = 100
- 91 O: 0286 (M284 MOTOR-COMMANDS) RIGHTHAND from STICK3 to NEUTRAL
 Moving RIGHTHAND from STICK3 to NEUTRAL, dtm = 100, dm = 60
 Moving RIGHTHAND from STICK3 to NEUTRAL, dtm = 100, dm = 90
 Moving RIGHTHAND from STICK3 to NEUTRAL, dtm = 100, dm = 100
- 92 O: 0287 (CHECK-SPEED)
- 93 O: 0290 (M288 MOTOR-COMMANDS) RIGHTHAND from NEUTRAL to STICK2
 Moving RIGHTHAND from NEUTRA L to STICK2, dtm = 100, dm = 30
 Moving RIGHTHAND from NEUTRAL to STICK2, dtm = 100, dm = 60
- 94 O: 0298 (M291 MOTOR-COMMANDS)RIGHTHAND from STICK2 to WHEEL
 Moving RIGHTHAND from NEUTRAL to STICK2, dtm = 100, dm = 90
 Moving RIGHTHAND from NEUTRAL to STICK2, dtm = 100, dm = 100
 Moving RIGHTHAND from STICK2 to WHEEL, dtm = 100, dm = 30
 Moving RIGHTHAND from STICK2 to WHEEL, dtm = 100, dm = 60
 Moving RIGHTHAND from STICK2 to WHEEL, dtm = 100, dm = 90
 Moving RIGHTHAND from STICK2 to WHEEL, dtm = 100, dm = 100
- 95 O: 0304 (M301 MOTOR-COMMANDS) TURN WHEEL (BOTH HANDS)
 Moving LEFTHAND from WHEEL to WHEEL, dtm = 5, dm = 5
 Moving RIGHTHAND from WHEEL to WHEEL, dtm = 5, dm = 5
- 96 O: 0277 ((INTERSECTION1) ATTEND)
- 97 O: 0256 ((AUTO3C) EYE-MOVEMENT-COMMAND)
 Moving eye from 1.67 to 2.39 dtm = 0.71 dm = 0.47
 Moving eye from 1.67 to 2.39 dtm = 0.71 dm = 0.71
- 98 O: 0325 ((HOUSE4) ATTEND)
- 99 O: 0328 (CHECK-SPEED)
- 100 O: 0321 ((AUTO4) EYE-MOVEMENT-COMMAND)
 Moving eye from 2.39 to 2.05 dtm = 0.33 dm = 0.33
- 101 O: 0353 ((AUTO4) ATTEND)
- 102 O: 0356 ((HOUSE3) ATTEND)
- 103 O: 0341 ((AUTO3A) EYE-MOVEMENT-COMMAND)
 Moving eye from 2.05 to 1.26 dtm = 0.79 dm = 0.47
 Moving eye from 2.05 to 1.26 dtm = 0.79 dm = 0.79
- 104 O: 0380 ((AUTO3A) ATTEND)
- 105 O: 0383 ((HOUSE2) ATTEND)
- 106 O: 0389 (CHECK-SPEED)
- 107 O: 0404 (M407 MOTOR-COMMANDS) TURN WHEEL (BOTH HANDS)
 Moving LEFTHAND from WHEEL to WHEEL, dtm = 8, dm = 6
 Moving RIGHTHAND from WHEEL to WHEEL, dtm = 8, dm = 6

- Moving LEFTHAND from WHEEL to WHEEL, dtm = 8, dm = 8
 Moving RIGHTHAND from WHEEL to WHEEL, dtm = 8, dm = 8
- 108 O: 0280 (M275 MOTOR-COMMANDS) RIGHTFOOT from BRAKE -DOWN to BRAKE-UP
 Moving RIGHTFOOT from BRAKE -DOWN to BRAKE-UP, dtm = 60, dm = 15
 Moving RIGHTFOOT from BRAKE -DOWN to BRAKE-UP, dtm = 60, dm = 30
- 109 O: 0406 (M411 MOTOR-COMMANDS) RIGHTFOOT from BRAKE -UP to ACC-UP
 Moving RIGHTFOOT from BRAKE -DOWN to BRAKE-UP, dtm = 60, dm = 45
 Moving RIGHTFOOT from BRAKE -DOWN to BRAKE-UP, dtm = 60, dm = 60
 Moving RIGHTFOOT from BRAKE -UP to ACC-UP, dtm = 100, dm = 30
 Moving RIGHTFOOT from BRAKE -UP to ACC-UP, dtm = 100, dm = 60
- 110 O: 0386 ((SIGN1) ATTEND)
 Moving RIGHTFOOT from BRAKE -UP to ACC-UP, dtm = 100, dm = 90
 Moving RIGHTFOOT from BRAKE -UP to ACC-UP, dtm = 100, dm = 100
- 111 O: 0367 ((AUTO4) EYE-MOVEMENT-COMMAND)
 Moving eye from 1.26 to 2.04 dtm = 0.77 dm = 0.47
 Moving eye from 1.26 to 2.04 dtm = 0.77 dm = 0.77
- 112 O: 0432 ((AUTO3A) ATTEND)
- 113 O: 0444 (CHECK-SPEED)
- 114 O: 0441 ((HOUSE3) ATTEND)
- 115 O: 0458 (LEFT EYE-MOVEMENT-COMMAND)
 Moving eye from 2.04 to 2.02 dtm = 0.02 dm = 0.02
- 116 O: 0478 ((HOUSE3) ATTEND)
- 117 O: 0464 ((AUTO3A) EYE-MOVEMENT-COMMAND)
 Moving eye from 2.02 to 2.66 dtm = 0.64 dm = 0.47
 Moving eye from 2.02 to 2.66 dtm = 0.64 dm = 0.64
- 118 O: 0510 (HEAD-MOVEMENT-COMMAND) MOVE HEAD
 Moving head from 1.57 to 2.75 dtm = 1.18 dm = 0.18
 Moving head from 1.57 to 2.75 dtm = 1.18 dm = 0.37
 Moving head from 1.57 to 2.75 dtm = 1.18 dm = 0.56
- 119 O: 0283 (M281 MOTOR-COMMANDS) LEFTFOOT from CLUTCH -DOWN to CLUTCH-UP
 Moving head from 1.57 to 2.75 dtm = 1.18 dm = 0.75
 Moving head from 1.57 to 2.75 dtm = 1.18 dm = 0.94
 Moving head from 1.57 to 2.75 dtm = 1.18 dm = 1.13
 Moving LEFTFOOT from CLUTCH -DOWN to CLUTCH-UP, dtm = 60, dm = 3/2
 Moving head from 1.57 to 2.75 dtm = 1.18 dm = 1.18
 Moving LEFTFOOT from CLUTCH -DOWN to CLUTCH-UP, dtm = 60, dm = 3
- 120 O: 0513 (CHECK-SPEED)
 Moving LEFTFOOT from CLUTCH -DOWN to CLUTCH-UP, dtm = 60, dm = 9/2
 Moving LEFTFOOT from CLUTCH -DOWN to CLUTCH-UP, dtm = 60, dm = 6
 Moving LEFTFOOT from CLUTCH -DOWN to CLUTCH-UP, dtm = 60, dm = 15/2
 Moving LEFTFOOT from CLUTCH -DOWN to CLUTCH-UP, dtm = 60, dm = 9
 Moving LEFTFOOT from CLUTCH -DOWN to CLUTCH-UP, dtm = 60, dm = 21/2
 Moving LEFTFOOT from CLUTCH -DOWN to CLUTCH-UP, dtm = 60, dm = 12
 Moving LEFTFOOT from CLUTCH -DOWN to CLUTCH-UP, dtm = 60, dm = 27/2

Look in forward direction if forced this time because DRIVER is so close to intersection; note that there is no object that the eye movement is directed at.

- 121 O: 0530 (FORWARD EYE-MOVEMENT-COMMAND)
 Moving LEFTFOOT from CLUTCH -DOWN to CLUTCH-UP, dtm = 60, dm = 15
 Moving LEFTFOOT from CLUTCH -DOWN to CLUTCH-UP, dtm = 60, dm = 33/2
 Moving eye from 2.66 to 1.32 dtm = 1.34 dm = 0.47
 Moving LEFTFOOT from CLUTCH -DOWN to CLUTCH-UP, dtm = 60, dm = 18
 Moving eye from 2.66 to 1.32 dtm = 1.34 dm = 0.94
 Moving LEFTFOOT from CLUTCH -DOWN to CLUTCH-UP, dtm = 60, dm = 39/2
 Moving eye from 2.66 to 1.32 dtm = 1.34 dm = 1.34

- Moving LEFTFOOT from CLUTCH -DOWN to CLUTCH -UP, dtm = 60, dm = 21
- 122 O: 0410 (M417 MOTOR-COMMANDS) RIGHTFOOT from ACC -UP to ACC-DOWN
- Moving LEFTFOOT from CLUTCH -DOWN to CLUTCH -UP, dtm = 60, dm = 45/2
- Moving LEFTFOOT from CLUTCH -DOWN to CLUTCH -UP, dtm = 60, dm = 24
- Moving LEFTFOOT from CLUTCH -DOWN to CLUTCH -UP, dtm = 60, dm = 51/2
- Moving RIGHTFOOT from ACC -UP to ACC-DOWN, dtm = 60, dm = 3
- Moving LEFTFOOT from CLUTCH -DOWN to CLUTCH -UP, dtm = 60, dm = 27
- Moving RIGHTFOOT from ACC -UP to ACC-DOWN, dtm = 60, dm = 6
- 123 O: 0531 ((AUTO3C) EYE-MOVEMENT-COMMAND)
- Moving LEFTFOOT from CLUTCH -DOWN to CLUTCH -UP, dtm = 60, dm = 57/2
- Moving RIGHTFOOT from ACC -UP to ACC-DOWN, dtm = 60, dm = 9
- Moving LEFTFOOT from CLUTCH -DOWN to CLUTCH -UP, dtm = 60, dm = 30
- Moving RIGHTFOOT from ACC -UP to ACC-DOWN, dtm = 60, dm = 12
- Moving eye from 1.32 to 2.55 dtm = 1.23 dm = 0.47

Example of moving extremities and eyes at the same time.

- Moving LEFTFOOT from CLUTCH -DOWN to CLUTCH -UP, dtm = 60, dm = 63/2
- Moving RIGHTFOOT from ACC -UP to ACC-DOWN, dtm = 60, dm = 15
- Moving eye from 1.32 to 2.55 dtm = 1.23 dm = 0.94
- Moving LEFTFOOT from CLUTCH -DOWN to CLUTCH -UP, dtm = 60, dm = 33
- Moving RIGHTFOOT from ACC -UP to ACC-DOWN, dtm = 60, dm = 18
- Moving eye from 1.32 to 2.55 dtm = 1.23 dm = 1.23

Over multiple elaboration cycles.

- Moving LEFTFOOT from CLUTCH -DOWN to CLUTCH -UP, dtm = 60, dm = 69/2
- Moving RIGHTFOOT from ACC -UP to ACC-DOWN, dtm = 60, dm = 21
- Moving LEFTFOOT from CLUTCH -DOWN to CLUTCH -UP, dtm = 60, dm = 36
- Moving RIGHTFOOT from ACC -UP to ACC-DOWN, dtm = 60, dm = 24
- Moving LEFTFOOT from CLUTCH -DOWN to CLUTCH -UP, dtm = 60, dm = 75/2
- Moving RIGHTFOOT from ACC -UP to ACC-DOWN, dtm = 60, dm = 27
- Moving LEFTFOOT from CLUTCH -DOWN to CLUTCH -UP, dtm = 60, dm = 39
- Moving RIGHTFOOT from ACC -UP to ACC-DOWN, dtm = 60, dm = 30
- 124 O: 0484 (M486 MOTOR-COMMANDS) TURN WHEEL
- Moving LEFTFOOT from CLUTCH -DOWN to CLUTCH -UP, dtm = 60, dm = 81/2
- Moving RIGHTFOOT from ACC -UP to ACC-DOWN, dtm = 60, dm = 33
- here we edited out many cycles
- Moving RIGHTFOOT from ACC -UP to ACC-DOWN, dtm = 60, dm = 60
- Moving LEFTFOOT from CLUTCH -DOWN to CLUTCH -UP, dtm = 60, dm = 111/2
- Moving LEFTFOOT from CLUTCH -DOWN to CLUTCH -UP, dtm = 60, dm = 57
- Moving LEFTFOOT from CLUTCH -DOWN to CLUTCH -UP, dtm = 60, dm = 117/2

- 125 O: 0586 ((INTERSECTION1) ATTEND)
- Moving LEFTFOOT from CLUTCH -DOWN to CLUTCH -UP, dtm = 60, dm = 60
- 126 O: 0545 ((AUTO3A) EYE-MOVEMENT-COMMAND)
- Moving eye from 2.55 to 3.04 dtm = 0.48 dm = 0.47
- Moving eye from 2.55 to 3.04 dtm = 0.48 dm = 0.48

- 127 O: 0622 (CHECK-SPEED)

This is where the turn manoeuvre starts. First DRIVER starts to look to the right for the last time; note that the eye-movement operator does not specify an object to look at. It is forced there by top-down control rules.

- 128 O: 0641 (RIGHT EYE-MOVEMENT-COMMAND)
- Moving eye from 3.04 to 1.19 dtm = 1.85 dm = 0.47
- Moving eye from 3.04 to 1.19 dtm = 1.85 dm = 0.94
- Moving eye from 3.04 to 1.19 dtm = 1.85 dm = 1.41

Then starts to turn the wheel

- 129 O: 0602 (M533 MOTOR-COMMANDS) TURN WHEEL
- Moving eye from 3.04 to 1.19 dtm = 1.85 dm = 1.85
- Moving LEFTHAND from WHEEL to WHEEL, dtm = 60, dm = 6

Moving RIGHTHAND from WHEEL to WHEEL, dtm = 60, dm = 6

Moving LEFTHAND from WHEEL to WHEEL, dtm = 60, dm = 12

Moving RIGHTHAND from WHEEL to WHEEL, dtm = 60, dm = 12

While turning the wheel its eyes move to the front.

130 O: 0644 ((AUTO1A) EYE-MOVEMENT-COMMAND)

Moving LEFTHAND from WHEEL to WHEEL, dtm = 60, dm = 18

Moving RIGHTHAND from WHEEL to WHEEL, dtm = 60, dm = 18

Moving LEFTHAND from WHEEL to WHEEL, dtm = 60, dm = 24

Moving RIGHTHAND from WHEEL to WHEEL, dtm = 60, dm = 24

Moving eye from 1.19 to 1.69 dtm = 0.50 dm = 0.47

Moving LEFTHAND from WHEEL to WHEEL, dtm = 60, dm = 30

Moving RIGHTHAND from WHEEL to WHEEL, dtm = 60, dm = 30

Moving eye from 1.19 to 1.69 dtm = 0.50 dm = 0.50

Moving LEFTHAND from WHEEL to WHEEL, dtm = 60, dm = 36

Moving RIGHTHAND from WHEEL to WHEEL, dtm = 60, dm = 36

....

....

Moving LEFTHAND from WHEEL to WHEEL, dtm = 60, dm = 60

Moving RIGHTHAND from WHEEL to WHEEL, dtm = 60, dm = 60

Head follows the eye and also turns to the front.

131 O: 0659 (HEAD-MOVEMENT-COMMAND)

Moving head from 2.75 to 1.78 dtm = 0.96 dm = 0.18

Moving head from 2.75 to 1.78 dtm = 0.96 dm = 0.37

Moving head from 2.75 to 1.78 dtm = 0.96 dm = 0.56

132 O: 0671 ((AUTO1A) ATTEND)

Moving head from 2.75 to 1.78 dtm = 0.96 dm = 0.75

Moving head from 2.75 to 1.78 dtm = 0.96 dm = 0.94

Moving head from 2.75 to 1.78 dtm = 0.96 dm = 0.96

133 O: 0674 ((AUTO3A) EYE-MOVEMENT-COMMAND)

Moving eye from 1.69 to 0.10 dtm = 1.58 dm = 0.47

Moving eye from 1.69 to 0.10 dtm = 1.58 dm = 0.94

Moving eye from 1.69 to 0.10 dtm = 1.58 dm = 1.41

134 O: 0680 (CHECK-SPEED)

135 O: 0706 (M707 MOTOR-COMMANDS) RIGHTFOOT from ACC-DOWN to ACC-DOWN

Moving RIGHTFOOT from ACC-DOWN to ACC-DOWN, dtm = 10, dm = 6

Moving RIGHTFOOT from ACC-DOWN to ACC-DOWN, dtm = 10, dm = 10

136 O: 0710 (M712 MOTOR-COMMANDS) RIGHTFOOT from ACC-DOWN to ACC-DOWN

Moving RIGHTFOOT from ACC-DOWN to ACC-DOWN, dtm = 10, dm = 6

Moving RIGHTFOOT from ACC-DOWN to ACC-DOWN, dtm = 10, dm = 10

137 O: 0700 ((LEFT-ROAD) ATTEND)

138 O: 0703 ((INTERSECTION1) ATTEND)

139 O: 0697 ((LEFT-ROAD) EYE-MOVEMENT-COMMAND)

Moving eye from 0.10 to 0.10; (*looking in mirror*)

140 O: 0734 ((LEFT-ROAD) ATTEND)

141 O: 0741 (CHECK-SPEED)

142 O: 0746 (M751 MOTOR-COMMANDS)

Moving RIGHTFOOT from ACC-DOWN to ACC-DOWN, dtm = 10, dm = 6

Moving RIGHTFOOT from ACC-DOWN to ACC-DOWN, dtm = 10, dm = 10

143 O: 0737 ((INTERSECTION1) ATTEND)

144 O: 0743 ((SIGN2) ATTEND)

145 O: 0748 ((HOUSE3) ATTEND)

146 O: 0731 ((LEFT-ROAD) EYE-MOVEMENT-COMMAND)

Moving eye from 0.10 to 0.10 dtm = 0.0, dm = 0.0 (*looking in mirror*)

147 O: 0781 ((LEFT-ROAD) ATTEND)

148 O: 0794 (CHECK-SPEED)
149 O: 0784 ((INTERSECTION1) ATTEND)
150 O: 0787 ((HOUSE3) ATTEND)
151 O: 0790 ((SIGN2) ATTEND)
152 O: 0778 ((AUTO6A) EYE-MOVEMENT-COMMAND)
 Moving eye from 0.10 to 0.10 dtm = 0.0, dm = 0.0
153 O: 0821 ((AUTO6) ATTEND)
154 O: 0824 ((INTERSECTION1) ATTEND)
155 O: 0835 (CHECK-SPEED)
156 O: 0827 ((HOUSE3) ATTEND)
157 O: 0830 ((SIGN2) ATTEND)
158 O: 0818 ((LEFT-ROAD) EYE-MOVEMENT-COMMAND)
 Moving eye from 0.10 to 0.10 dtm = 0.0, dm = 0.0
159 O: 0861 ((AUTO6A) ATTEND)
160 O: 0864 ((AUTO6B) ATTEND)
161 O: 0867 ((HOUSE3) ATTEND)
162 O: 0879 (CHECK-SPEED)
163 O: 0870 ((SIGN2) ATTEND)
164 O: 0875 ((HOUSE4) ATTEND)
165 O: 0858 ((AUTO6A) EYE-MOVEMENT-COMMAND)
 Moving eye from 0.10 to 0.10 dtm = 0.0, dm = 0.0
166 O: 0908 ((AUTO6A) ATTEND)
167 O: 0911 ((INTERSECTION1) ATTEND)
168 O: 0914 ((HOUSE3) ATTEND)
169 O: 0926 (CHECK-SPEED)
170 O: 0917 ((SIGN2) ATTEND)
171 O: 0920 ((HOUSE4) ATTEND)
172 O: 0905 ((AUTO6A) EYE-MOVEMENT-COMMAND)
 Moving eye from 0.10 to 0.10 dtm = 0.0, dm = 0.0
173 O: 0955 ((AUTO6A) ATTEND)
174 O: 0958 ((INTERSECTION1) ATTEND)
175 O: 0961 ((HOUSE3) ATTEND)
176 O: 0973 (CHECK-SPEED)
177 O: 0964 ((SIGN2) ATTEND)

14.3 DRIVER's performance compared to human drivers

14.3.1 Fitting observable behaviour

An obvious requirement for DRIVER is that its behaviour must resemble humans' traffic behaviour. In our case we are interested in the extent to which a trace like the one presented above models the intersection behaviour of the young experts that were described in Chapter 4. A minimum requirement is that DRIVER should display a qualitative fit; in other words we want DRIVER to display the same sequence for those actions as we observed in the experts. In addition, we would of course like DRIVER to perform the operations in the same time period and the timing of its actions to match that of the experts¹. The actions about which we can say something are of course only the motor actions and the visual actions that were observed among the young experts.

We shall deal with the fit between DRIVER and the behaviour of the young experts on the basis of a turn-right manoeuvre.

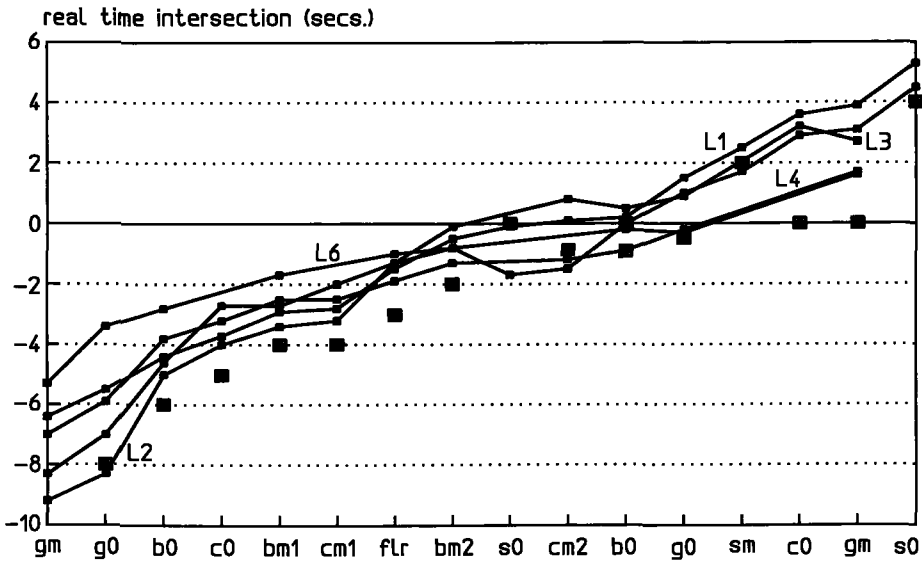


Figure 14-2. The y-axis shows the real time to intersection. Negative numbers refer to the period before the intersection. The x-axis shows a number of observed events. The accelerator events before the intersection are accelerator maximum (gm) and release of accelerator (g0). The brake is represented by the start (b0), first maximum (bm1), second maximum (bm2), and the final release (b0). The clutch is represented by the start (c0), first maximum (cm1), second maximum (cm2), and the final release (c0) and the steering wheel is represented by start of turning (s0), maximum angle (sm) and return of wheel to original position (s0). Flr stands for first look to the right. This figure is in fact a copy of Figure 6 from Chapter 4. Each line represents the average of all subjects in a particular manoeuvre. The only deviation from the figure from Chapter 4 are the unconnected dots. These represent the events for driver in a turn-right manoeuvre and are derived from Trace 1.

The continuous lines in Figure 14-2 represent the average timing of events for the experienced drivers described in Chapter 4. The unconnected dots in this figure represent DRIVER's events in a turn-right manoeuvre. These events, derived from Trace 1, match the data from the experienced drivers reasonably well. With the exception of the steering behaviour we see that, as with the experts, DRIVER displays a steadily increasing line for all its actions. This shows that DRIVER performs most of the actions in the same order. The exception of steering behaviour can be explained by the fact that in this specific approach DRIVER braked a little too much before starting the curve. We also see that DRIVER handles the entire manoeuvre in approximately the same time period and that the timing of the individual action is roughly the same.

We must make a number of remarks regarding the fit found. In the first place the question arises how surprising it is that we found this fit. This question is discussed in detail in section 5 of this chapter. A second remark is that the trace is far richer than the observed behaviour for the young experts. For the

motor behaviour, for example, the trace also shows actions such as lifting the foot so as to place it on the clutch. In the visual orientation it is even more difficult to ascertain a fit, since in De Velde Harsenhorst and Lourens' data only the main directions in which a driver looked were examined, while we also have eye movements at our disposal. Nevertheless we see in the trace that in the left-turn manoeuvre DRIVER, as with the experts, looks consciously once to the right, forward and then to the left when starting the bend.

A third remark is that it must not be forgotten that figure 14-2 shows average driver behaviour. We saw in Chapter 4 that there are not many, but certainly some, individual differences, primarily in the steering movements. We also see the discrepancy with our fit with regard to the steering movements among some individual experts.

14.3.2 Additional fits

In the previous section the fit for the observable behaviour was dealt with. A more detailed analysis of the traces gives rise to another, more indirect input for comparing driver's behaviour with human behaviour. Our analysis begins with some basic statistics describing Soar's internal activity during the entire right-turn manoeuvre (See Table 1).

Table 1. Measures describing Soar's internal activity during right turn.

Total duration of right-turn manoeuvre in seconds	15.75
Operators per second	7.3
Elaboration cycles per operator	4.6
Production firings per elaboration cycle	7.6

Total duration of manoeuvre. DRIVER's right-turn manoeuvre starts at decision cycle 62 in the trace (excluding navigation). From cycle 62 to the end of the trace DRIVER takes 525 elaboration cycles. Remember that an elaboration cycle served as the clock tick of WORLD and that the length of a clock tick in DRIVER was set at 30 milliseconds. This results in a total length of 525 times .03 seconds equals 15.75 seconds for the total manoeuvre (Fig. 2a shows that this duration fits with the total manoeuvre time of our young experts).

Number of operators per second. We count 121 decision cycles in this period, so the mean number of decision cycles per second is 7.7. However, not every decision cycle results in an operator, so there are slightly fewer operators in this period (114); the average number of operators per second is 7.3.

Elaboration cycles per operator. During the turn-right manoeuvre we count 525 elaboration cycles. This equals 4.60 elaboration cycles per operator. Figure 14-3 shows that the distribution of the elaborations over the decision cycles is not entirely flat. The sharp increases in activity are mainly due to the initiation of check speed operators and motor control actions.

Production firings per elaboration cycle. Altogether 3993 productions fired, from cycle 62 to the end. This represents 7.6 production firings per elaboration cycle.

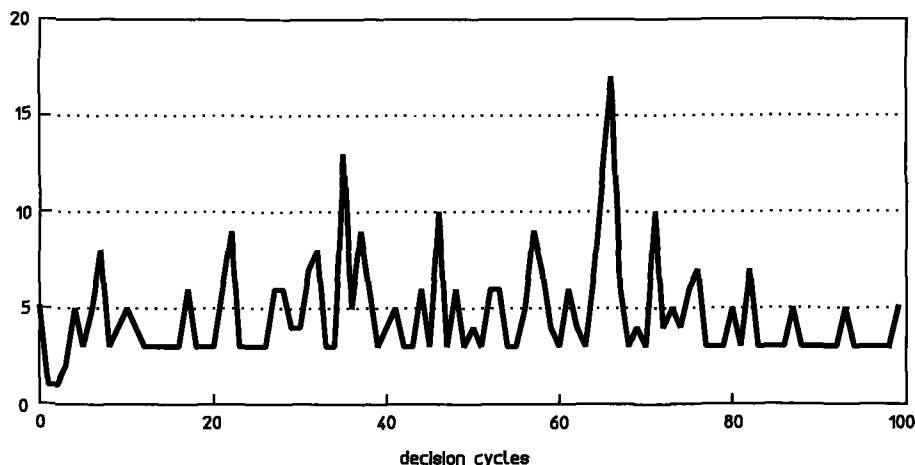


Figure 14-3. Number of elaboration cycles per decision cycle. Decision cycle 0 in this figure relates to decision cycle 62 in Trace 1.

An initial remark arising from this analysis relates to DRIVER's cognitive load. We have seen that the times for handling the entire manoeuvre and the number of observable actions are approximately the same for DRIVER. But what we are also interested in is the load on DRIVER's cognitive processor in comparison with the human drivers. In Soar terms we define this load as the number of (Soar) operators required per unit of time to perform a given task². The higher this load, the less room there is for other task operators.

The problem is of course that whereas in Soar one can count the number of operators required for a particular task, with people one has to make an estimate in some way. Figure 14-4 provides an overview of a left turn manoeuvre for the young experts from chapter 4. The figure also includes the load of the cognitive apparatus, assuming that all motor actions, including the eye movements, are initiated by cognition. What this figure adds to previous information is that the hand movements from the steering wheel to the gear-stick and the various actions involving the gear-stick are now included. We see that close to the intersection the number of cognitive actions rises to 5 per second. This however is still lower than the 7.3 operators per second in Table 1. The explanation is that for DRIVER we also count the eye movements (not only the main looking direction), the attention operators, steering movements and speed control checks. Given this explanation there is again evidence that DRIVER's internal load is comparable to the load in human drivers.

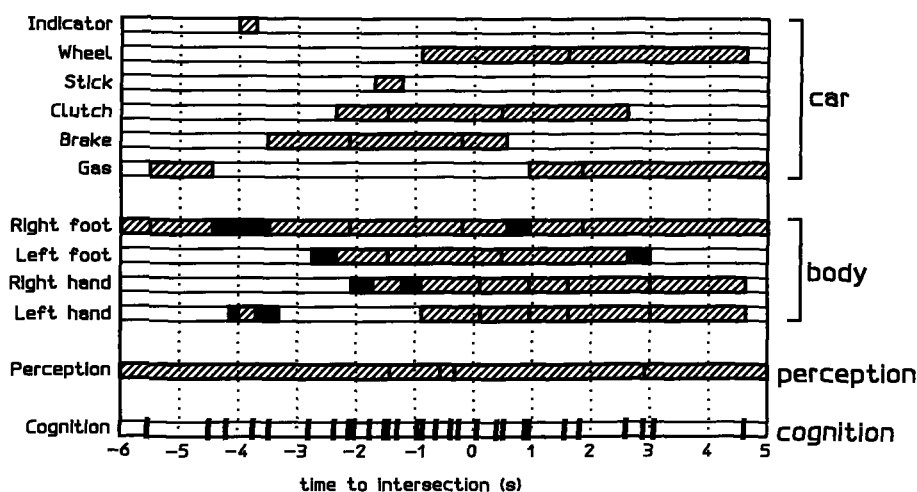


Figure 14-4. This figure shows an average left turn manoeuvre of the young experts from chapter 4. The upper boxes show the activity of the car devices. The hatched parts show when the device is used. The vertical lines that divide the hatched parts are transitions from one phase to another. For example, in the box for the brake pedal we see the three phases that were described in chapter four. The following four boxes show the activity for the body parts. The hatched parts denote activity at the devices; the black areas show when an extremity moves between two devices; and the white areas denote rest. The perception box shows the transitions between the main gaze directions (but no eye movements). Finally, the lower box is derived by copying down all the transitions from the body and perception, based upon the assumption that all body actions start with a motor operator in cognition.

A second remark is that we see that DRIVER is able to perform the manoeuvre properly with the rather slow elaboration cycle of 30 milliseconds³. In the Soar literature estimates are given for 10 to 20 milliseconds for an elaboration cycle (Newell, 1990). If we perform the same simulations of manoeuvres with 20 milliseconds we see hardly any differences at the global level, although 50 percent more operators may be used. The extra operators that DRIVER uses are then mainly used for more attention operators and eye movements. If we carry out the same simulations with 40 millisecond elaboration cycles DRIVER really gets into difficulties because we then find that there is not enough room for speed checks or attention operators.

Our final remark relates to the high degree of parallel activity in DRIVER. Table 1 shows that about 35 productions fire per operator cycle (multiplying elaboration cycles per operator by production firings per elaboration cycle). Only some of these productions are dedicated to applying the current operator, while most of them are dedicated to generating and selecting operators for other, relevant tasks. Trace 2 shows two decision cycles from Trace 1 in more detail. The trace is impossible to understand without the full source code. However, just look at the names of the productions and the italicised comments on the right-hand side in order to get a feeling of the other tasks that DRIVER works on while one task operator is in place.

Trace 2. A more detailed Soar trace of decision cycles 68 and 69 in Trace 1. The trace has been considerably shortened by editing out all the multiple firings of productions. The comments in the right-hand margin indicate which task or process DRIVER is dealing with. Note that all this activity occurs in only a few hundred milliseconds.

68 O: 0636 (RIGHT EYE-MOVEMENT-COMMAND)

--Preference Phase--

Firing 69:733 ATTEND*GENERAL*UPDATE*CHECK -AGAIN-COUNT

Attention

Firing 69:733 UPDATE-LOOKING-COUNT

Eye movements

Firing 69:733 CHECK-SPEED*UPDATE-CHECK-COUNT-EVERY-CYCLE

Speed control

Firing 69:733 VISOR*EMC*APPLY

Visual orientation

Firing 69:733 APPROACH-INTERSECTION*APPLY*LOOK-RLF-EYEMOVEMENTS

Eye movements

Firing 69:733 CHECK-SPEED*APPLY*REMOVE-NEWS

Speed control

Retracting 69:733 CHECK-SPEED*TERMINATE

Speed control

--Working Memory Phase --

--Preference Phase--

Firing 69:735 VISOR*EMC*TERMINATE

Visual orientation

Firing 69:735 PRINT-EYE-DIRECTION

Retracting 69:735 APPROACH-INTERSECTION*TURN-RIGHT*LOOK-RIGHT

Eye movements

--Working Memory Phase --

--Preference Phase--

Retracting 69:737 VISOR*PROPOSE*ATTEND*ALL -OBJECTS-IN-FUNCTIONAL-FIELD

Attention

Retracting 69:737 VISOR*PROPOSE*EMC*LOOK -AT-MOVING-OBJECTS

Attention

Retracting 69:737 VISOR*PROPOSE*EMC*LOOK -AT-MOVING-OBJECTS*MIRROR

Visual orientation

Retracting 69:737 GEN-PREFS-FROM-EVALUATIONS*GT-ZERO

Multitasking

Retracting 69:737 GEN-PREFS-FROM-EVALUATIONS*MULTIPLE-OPS*DIFF-EVALS

--Working Memory Phase --

--Preference Phase--

Retracting 69:739 GEN-PREFS-FROM-EVALUATIONS*GT-ZERO

Multitasking

Retracting 69:739 VISOR*PREFER*MOTOR -COMMANDS

Motor control

Retracting 69:739 VISOR*PREFER*STEER-OPERATOR

Steering

Retracting 69:739 GEN-PREFS-FROM-EVALUATIONS*GT-ZERO

Multitasking

Retracting 69:739 GEN-PREFS-FROM-EVALUATIONS*MULTIPLE-OPS*DIFF-EVALS

Multitasking

--Working Memory Phase --

69:741 DECIDE OPERATOR 0594

69 O: 0594 (M596 MOTOR-COMMANDS)

Steering

--Preference Phase--

Firing 70:742 ATTEND*GENERAL*UPDATE*CHECK -AGAIN-COUNT

Attend

Firing 70:742 UPDATE-LOOKING-COUNT

Eye movements

Firing 70:742 CHECK-SPEED*UPDATE-CHECK-COUNT-EVERY-CYCLE

Speed control

Firing 70:742 MONITOR*MOTOR-COMMAND*INTERNALLY

Motor control

Firing 70:742 APPLY-MOTOR-COMMANDS*DOUBLE-ACTION

Motor control

Retracting 70:742 VISOR*EMC*TERMINATE

--Working Memory Phase --

--Preference Phase--

Firing 70:744 VISOR*PROPOSE*EMC*LOOK -AT-MOVING-OBJECTS

Attend

Firing 70:744 VISOR*PROPOSE*EMC*LOOK -AT-MOVING-OBJECTS*MIRROR

Attend

Firing 70:744 MONITOR-BODY-ACTION-COMMANDS*COMMAND-ACTIVE*NOT-STICKY

Motor control

Firing 70:744 TERMINATE-MOTOR-COMMANDS

Motor control

Firing 70:744 MONITOR-VEHICLE-ACTION-COMMANDS*COMMAND-GIVEN*STICKY

Motor control

Retracting 70:744 PROPOSE-MOTOR-OPS-FROM-BODY-PLAN*FIRST-OP

*Motor control
(changing gear)*

Retracting 70:744 PROPOSE -MOTOR-OPS-FROM-BODY-PLAN*STEER*BOTH-HANDS

*Motor control
(steering)*

--Working Memory Phase--

Moving LEFTHAND from WHEEL to WHEEL, dtm = 60, dm = 6

Moving RIGHTHAND from WHEEL to WHEEL, dtm = 60, dm = 6

--Preference Phase--

Firing 70:746 GEN-PREFS-FROM-EVALUATIONS*GT-ZERO

Multitasking

Firing 70:746 MONITOR-VEHICLE-ACTION-COMMANDS*COMMAND-ACTIVE*NOT-STICKY

Retracting 70:746 PREFER-DOUBLE-OPS-IN-STEERING

--Working Memory Phase--

Moving LEFTHAND from WHEEL to WHEEL, dtm = 60, dm = 12

Moving RIGHTHAND from WHEEL to WHEEL, dtm = 60, dm = 12

--Quiescence Phase--

70:748 DECIDE OPERATOR 0639

70 0: 0639 ((AUTO1A) EYE-MOVEMENT-COMMAND)

14.4 DRIVER's multitasking mechanism

Having discussed DRIVER's overall behaviour it is now time to discuss the mechanism for DRIVER's multitasking. To start with, the mechanism is exceedingly simple. DRIVER switches tasks by switching operators in the base level space. It is clear from Trace 1 that most tasks are not carried out in their own problem space. DRIVER will not, as in the Chapter 2 model, go into the change-course problem space for correcting the course and then, say, into the change-speed problem space to correct speed. Instead it will execute most of the operators in the top level space. The exception in Trace 1 is the navigation task (and in earlier chapters we saw that DRIVER drops into problem spaces when learning to change gear.) The trace thus displays mainly expert behaviour for switching operators. The following section discusses the switching mechanism and the preference knowledge for making sure that the right operator is generated and selected at the right moment.

14.4.1 Mechanisms and knowledge involved in multitasking

Properly speaking, there are no additional representational structures or additional mechanisms involved in DRIVER's multitasking. The multiple goals or tasks that are active in driving are stored in the base level state, albeit in a non-uniform fashion. This is in contrast to the Chapter 2 model, where all tasks were represented in a standard, uniform fashion on special task or process structures (see Figs. 2.2 – 2.4 in Chapter 2).

The operators that are generated in DRIVER's base level space are chosen via the normal Soar preferences and do not require special operators to change task contexts (again in contrast to the older model). The only, slightly heretical, deviation from the normal Soar practice is that explicit and unique preferences are used in addition to Soar's normal preferences. We have already encountered these explicit structures in earlier chapters (variations on Soar's

default rules, the motor control chapters and the chapter on visual orientation).

```
(operator <0> ^evaluation <ev> )
(evaluation <ev> ^value 3 ^id x190)
```

The main reason for choosing these explicit structures is that they make *reasoning about preferences* possible. Normally in Soar, preferences are invisible to the left-hand side of productions, with the exception of an acceptable preference. These explicit structures are particularly useful in *error correction about preferences*. For example, using Soar's normal preferences it is impossible to learn that operator X is first 'best', then 'worst' and then 'best' again. By using explicit and unique preferences one can (conditionally) reject faulty preferences. In the concluding chapter of this study and in Appendix A we discuss error correction and learning from external interaction in more detail.

Looking at the traces we see that DRIVER chooses more or less the right action at the right moment. But how does DRIVER know *when* to generate and choose the right action? Table 2 lists DRIVER's preferences for different task operators. The numbers correspond to the explicit evaluations that are stored on the operator. General default productions translate these explicit evaluations into preferences. Operators with higher evaluations have a higher preference. Operators with the same evaluation are mutually indifferent.

Table 2. Preference values for choosing operators.

5	explicit eye and head movement close to the intersection
4	check and correct speed check and correct course
3	motor control operator
2	navigation operators well before intersection (> 60 m) attend
1	eye movements head movements
0	navigation operators close to the intersection (< 60 m).

After some experimentation this preference scheme turned out to give the most reasonable results. Moreover, it can also be argued fairly well:

The highest preference for explicit eye and head movements close to the intersection ensures that bottom-up control does not make DRIVER look in the wrong directions just before the intersection. The operators for checking speed or correcting course have the second highest preference in DRIVER. However, there are built-in mechanisms that make sure that the 'check' or 'monitor' operators are not selected too often. The generation and selection of a check-speed operator is regulated by a clock. The generation of steering operators is controlled by autonomous productions that only become active when a certain lateral deviation is exceeded. Motor control actions have a

relatively high preference because in most cases it is more important to carry out a motor plan to reduce speed than to look in the right direction at the right time. The assumption here is that DRIVER had already looked in the right direction and found that speed needed to be reduced. Of course there are always pauses in the execution of a motor plan, so DRIVER may look again.

We see that navigation has two evaluations. A two when DRIVER is far enough from the intersection, but a zero when it is too close. So, if DRIVER is more than 5 seconds from the intersection, navigation may compete with other operators. Close to the intersection all other tasks are more important than navigation. Attend, eye movements and head movements have a very low priority but they occur very frequently in the trace, as they use up all the idle time. (Soar has a default 'wait' operator that is active whenever there is nothing else to do. However, close to an intersection this operator is never active in DRIVER.)

14.5 Discussion

In this discussion three further points are covered. First, the question of to what extent DRIVER's behaviour is pre-programmed and to what extent it is spontaneous. Second, this chapter has not mentioned how DRIVER could learn the preference values for performing the correct task at the right moment. Third, we will compare DRIVER's multitasking with the multitasking of the model that was discussed in Chapter 2.

14.5.1 The spontaneity and the specificity of DRIVER's driving behaviour

It was argued above that DRIVER's driving behaviour shows a reasonable resemblance to human drivers' behaviour. The two questions that could then be asked are questions that should in fact be asked of any cognitive model. The first question is whether the behaviour displayed by the model is mainly pre-programmed or whether it occurs spontaneously in some way. In other words, does DRIVER follow a fixed algorithm when approaching a crossing or is the behaviour an emergent property of much simpler underlying rules? The second, related question is to what extent this system is so flexible that it would also display adequate behaviour in other situations. In other words, to what extent is DRIVER generally and to what extent is it specifically directed towards crossings?

Often these questions are not asked because the emphasis is less on fitting the model to the data – it is assumed anyway that this must be right – than on the underlying architecture or on the rules that generated the behaviour. If the emphasis is on the architecture, it is pointed out that this architecture meets all the conditions for specifying this type of behaviour. If the emphasis is on the underlying (sets of) rules it is often pointed out that this body of rules is sufficient to reproduce this type of behaviour. Even more frequently, however,

the emphasis will be on the combination of architecture and the rules. In this study too the emphasis is on the combination of architecture and rules. We have built upon the Soar architecture and used both constraints from the perception and motor control literature and empirical constraints to build the DRIVER architecture. The integration of the different subtasks then takes only a small number of preference rules.

But to answer the questions asked above: yes, of course DRIVER's behaviour is pre-programmed, particularly since learning takes place only to a limited extent. And even if learning was more ubiquitous, we would have to have set the system up in such a way that it would automatically learn the rules we want. The spontaneous and flexible aspects of the DRIVER system arise because most behaviour is generated bottom-up (and is thus determined by the external world). Only in specific situations are there a number of preference rules that override the bottom-up rules and force more situation-specific behaviour.

As regards the separation between the general and the specific, we can say that most of DRIVER is set up on a general basis (within the context of the driving task, of course). The bottom-up rules for all kinds of behavioural aspects are somewhat more specific, but still fairly general. It is only when we come to multitasking that we see that there are a number of highly intersection-specific rules that determine behaviour in critical situations.

We thus feel that we can safely say that DRIVER would be suitable for many other traffic situations. It is only logical, though, that DRIVER would have to learn the more situation-specific rules for different situations.

14.5.2 Learning to multitask

An early version of DRIVER incorporated a simplistic form of learning absolute evaluations and relative preferences. The evaluations and the relative preferences between operators were stored in a separate problem space called TASK-TIES. Whenever a tie arose between task operators in the base level this problem space was chosen to decide between the operators. Within this problem space there were always two options. Either Soar would ask the programmer, i.e. the simulated instructor, to type in the evaluations for the operators in this situation or there would already be knowledge in this space that knew how to evaluate the operators. After the evaluations have been made one way or another, the original impasse in the base level is solved and chunks are learned that will automatically apply these evaluations next time. This approach to solving impasses and learning is applied, for example, by Golding, Rosenbloom, and Laird (1987) or Laird (1990) or Laird, Hucka, Yager, and Tuck, (1990). However, this approach was rejected in DRIVER for a number of reasons:

(1) Learning by advice as described is rather unrealistic because input is given in a sub-space, whereas the Soar philosophy adopted in Newell (1990) is that all input arrives at the top level. By giving advice in the sub-space the problem of data-chunking is avoided. If advice is given in the top space then the connection between multiple unrelated data structures in the top space (at least two operators and an instruction) can only be learned by data-chunking (Rosenbloom and Aasman, 1992; see also Chapter 3 and the concluding chapter of this study).

(2) If no advice is given, but available knowledge in the TIE-TASK problem space is used to assign evaluations to the tying operators, then the question arises of how this knowledge got there in the first place. One might conceive that for certain types of ties this approach is a valid one. For example, one might have learned the general knowledge that while driving, driving-related operators have higher preference than non-driving-related operators. In tie situations, and hence in the tie-task problem space, the operators are then evaluated with regard to their relatedness to driving and an evaluation can be made.

However, it seems unrealistic that for example coordination of motor control operators and eye movements is learned by sub-goaling in time-critical situations. Just before the interaction there is no time to explore all the ramifications of making an eye movement or a hand movement in sub-goals when immediate reactions are required.

The above reasons were sufficient (for us) not to focus primarily on learning, but on what should be learned. We thus again come up against the problem of learning from external interaction. The concluding chapter of this study will discuss this issue in more detail.

14.5.3 DRIVER compared to the earlier model

Chapter 2 of this study described a *naive* version of multitasking in driving in an earlier version of Soar. Naive in the sense that task operators are carried out in their own problem spaces and that the standard default rules are applied when operators are tied. It was also demonstrated that Soar features such as destructive state modification and the truth maintenance principle are indispensable. The main criticisms of the earlier version were given in Chapter 3, where it was argued that the overhead for switching tasks is too high. Task-switching proved to be very impractical if Newell's estimates for the duration of elaboration cycles, decision cycles and operators were taken seriously. It was found that real-time task-switching was nearly impossible. Table 3 summarises the differences between the older model and DRIVER.

Finally: we have touched upon many issues that are relevant to a discussion of the psychological validity of multitasking. For example, the distinction between top-down and bottom-up processing, automatic and controlled process-

ing, learning, etc. However, a full discussion of these issues is postponed until the next, concluding chapter of this study. The mechanisms implemented will be evaluated in a broader context in that chapter.

Table 3. Multitasking in DRIVER and the Chapter 2 version.

Chapter 2 version	DRIVER
Uniform representation of processes, a handle for each task.	No uniform representation of processes. The representation of the body, the car and the internal model on the state is all that is needed.
Uniform mechanism for task-switching and resumption.	No uniform interrupt mechanism and no interrupt operator; attention (as a replacement for the interrupt operator) competes with move-eye, move-attention, move-extremity, navigation, check speed and all other operators for the operator slot. Interruption via preferences.
Soar 4 used operators to switch tasks; this is necessary when destructive state modification is available.	No (special) operators required to switch tasks.
Perception and motor control do not compete with main driving tasks. Large dependency on monitor productions (no active strategies), so no active search for information.	Perception and motor control compete with all other tasks. Moreover, DRIVER has active strategies for obtaining information.
All tasks are carried out in their own dedicated problem space.	Most tasks are carried out in base level space.
Soar's default rules for resolution of impasses.	DRIVER uses an alternative set of default rules that avoid deep goal stacks. Learning is not possible when approaching an intersection. System is too busy handling intersection.

14.6 Notes

¹ John (1988) and Newell (1990) discuss several ways in which a (Soar) model can fit empirical data. The best models and theories are able to deal with the full *parametric* detail. That is, variations in the data can be explained with experimental parameters. For example, it explains how the time required to extract an item from the visual store depends on the number of items in the store, the saliency of the items, the size and position, etc. Less strong, but still good, theories find a *quantitative* fit between the model and the data. These theories can for example explain mean times and variations. Even weaker theories find only a *qualitative* agreement. These theories predict the presence or absence of an effect or the order of effects. For example, they predict that under certain circumstances a variable will be larger than under others or it can predict the order of steps for solving a certain problem. For DRIVER a parametric fit is currently out of the question, but a qualitative and, to some extent, a quantitative fit can be found.

² The term *cognitive processor* is part of the *Model Human Processor* (MHP), an engineering model of human information processing by Card, Moran and Newell (1983, 1986). The model describes the human information processing system as a system consisting of three interacting

subsystems: (1) the *perceptual system*, (2) the *motor system*, and (3) the *cognitive system*, each with its own memories and processors. The three sub-systems work in parallel, but all motor actions, eye movements and attention are initiated by the cognitive processor, thereby enforcing *serialism* in information processing. The memory of each sub-system is described by three parameters, namely storage capacity, a decay constant and the main code type. The important parameter of a processor is the cycle time, i.e. the time required to process a minimum unit of information. With these parameters and a good task analysis this model is able to make qualitative, quantitative and sometimes even parametric predictions. This model and derivatives like GOMS are used particularly often in human computer interaction (see for an example application John, 1988). The MHP is a close relative of Soar (Newell, 1990) and in a sense Soar is a dynamic version of the MHP.

¹ Remember that Soar's elaboration cycle is the clock tick in WORLD. Thus, every clock tick the other semi-intelligent agents advance 30 milliseconds. By reducing the time per clock tick the non-Soar agents will advance more slowly, giving DRIVER the opportunity to think more, i.e. apply more operators. By increasing the time per clock tick the world will move faster with respect to DRIVER, thereby giving DRIVER less time to think.

15 Discussion

15.1 Introduction

The prime objective of this study was the evaluation of the theoretical and practical suitability of Soar for modelling complex dynamic task behaviour, whilst approaching and handling unordered crossings has been taken as a specimen task. The preceding chapters have dealt primarily with design choices and the implementation of *aspects* of DRIVER. In this chapter we evaluate Soar's suitability in a broader perspective.

This evaluation will be performed as follows. We start by examining the extent to which we succeeded in mapping the various sub-tasks comprising the driving task onto Soar. First, in Section 15.2, we shall deal with those aspects of the driving task that can easily be fitted into Soar. We shall also look at the more general behavioural principles that do not relate specifically to the driving task but are nevertheless essential to enable DRIVER to function properly. In section 15.3 we shall then deal with a number of problems we encountered when implementing DRIVER. Here a distinction is made between fundamental and non-fundamental problems. Examples of non-fundamental problems are problems in which either further research is required or in which only small, non-essential modifications to the Soar mechanisms are needed. Examples of fundamental problems are problems that really require essential modifications to Soar. In Section 15.4 a number of research fields are described which would make both Soar and DRIVER more complete. Finally, Section 15.5 summarises of our evaluation.

In evaluating Soar's suitability we are also indirectly assessing the completeness and extendibility of DRIVER as our cognitive model of driving behaviour. Remember that we also attach great importance to such a model of driving behaviour.

Before starting this evaluation it seems sensible to reiterate how we went about implementing DRIVER. First of all the entire implementation process for DRIVER is a pure bottom-up and incremental-design process. Although Chapter 5 may suggest otherwise, we did not start with a grand picture of how

DRIVER would look in its entirety, but by implementing the perceptual and motor components and the various sub-tasks in DRIVER. The order of chapters in Part 2 corresponds roughly to the chronological order of implementation. This also means that in some cases the implementation of a task or component is determined by design choices in earlier components. For example, the mechanism of eye movements and attention was to some extent determined by the motor-control mechanisms.

There is thus some effect from the bottom-up and incremental-design process on the eventual implementation of DRIVER. In this evaluation we must also realise that the implementation of these sub-parts and sub-tasks was also primarily determined by the different types of constraints that we had imposed upon ourselves beforehand. These constraints are summarised below.

1. Soar constraints. It will be clear that the implementation of DRIVER is primarily determined by the Soar architecture itself. The use of such mechanisms as productions, sub-goaling and chunking undeniably determines the shape of the model. In addition, an architecture such as Soar forces one into a certain type of decision. For example, (1) the balance between what is achieved by productions (within elaboration cycles) alone and what must be achieved by operators or (2) whether or not to use the default search control rules or (3) which structures should be fixed by operator application and what should be handled by Soar's intrinsic truth-maintenance system.

2. Soar as a UTC. The second group of constraints derives from Soar as a unified theory of cognition and in particular Newell's theory of immediate behaviour. Remember that there is no one-to-one relation between Soar as an implementation and Soar as a UTC (see Chapter 1). In implementing DRIVER we adhered to a number of constraints. First, we situated the use of motor operators, eye-movement operators and the use of attention operators at the top level, i.e. in the base level space. Second, we did not allow parallel access to all objects in visual memory store. And, third, we used Newell's time estimates for elaboration cycles and operators.

3. Basic constraints for perception and motor control. In implementing the model we focused primarily on the breadth of the driving task. We tried to include all relevant components and sub-tasks in the model, the most important criterion being that everything eventually had to fit together. The down-side of this approach is that we did not really delve into each component and sub-task in full depth. This is particularly evident in the implementation of perception and motor control. We have included only those constraints that we considered to be functional for the implementation of DRIVER.

4. Use of empirical data. The final source of constraints relates to the empirical data available to us. De Velde Harsenhorst and Lourens's data were initially used to translate regularities in drivers' behaviour into behavioural rules for DRIVER. These data were also used to test DRIVER's final behaviour. As a matter of fact, these data were not used until late in the implementation

process. It was only during the integration of all the components and sub-tasks (in Chapter 14) that the regularities in Chapter 4 were translated into multitasking rules. Most of DRIVER was implemented independently of our empirical data.

15.2 Mapping tasks and behaviours onto Soar

The evaluation starts with the positive achievements of DRIVER. In the following sections we will illustrate that Soar provides an appropriate framework and an appropriate set of mechanisms for experimenting with the integration of problem-solving, learning, multitasking, high-level motor control and visual orientation in a complex real-time environment.

Table 1 lists the driver sub-tasks and aspects of driver behaviour that can be captured using Soar's intrinsic mechanisms. In addition, this table lists behavioural phenomena not related to driving but still interesting in themselves. This section will cover items from this list one by one. In doing so we will occasionally encounter fundamental problems, but these will be postponed until Section 3, a section devoted entirely to the more fundamental problems existing in Soar.

Table 1. Task-related and behavioural phenomena that map onto Soar and onto DRIVER

-
- basic driver tasks: speed control, lane keeping, car control, navigation
 - attention and visual orientation
 - motor control: planning and execution
 - multiple types of goals
 - parallel processing and multitasking
 - external memory and situated action
 - a mix of top-down and bottom-up controlled behaviour.
 - a mix of controlled and automatic behaviour
 - open loop vs controlled loop behaviour
 - use of declarative vs procedural knowledge
-

15.2.1 Basic driver tasks

One of the prime aims of this study was to implement the basic driver sub-tasks, using Soar's intrinsic mechanisms as much as possible. The foregoing chapters showed that we have mainly succeeded in this aim as regards the cognitive aspects of these tasks. For the perceptual and motor aspects of these tasks we were forced to implement both low-level perception and motor control.

15.2.2 Attention and visual orientation

Wiesmeyer (1992) shows how covert visual attention can be modelled to a detailed level in a symbolic architecture such as Soar. Covert visual attention

is also a full-fledged part of DRIVER, though in its current form it is relatively primitive compared to Wiesmeyer's model. The main function of applying an attention operator in DRIVER is to prevent it from accessing all objects in the visual store in parallel. One of the desired side-effects of this mechanism is that it also made it possible to model the length of time required for inspecting the field of vision in various traffic situations.

Attention thus plays a role in the visual orientation process that also incorporates eye and head movements. Where Wiesmeyer showed that Soar has the right mechanisms to implement a covert visual attention mechanism, this study shows that Soar is also able to model the larger (overt) attention mechanism involving eye and head movements.

15.2.3 Motor control: planning and execution

DRIVER successfully learns and executes complex motor programmes. To achieve this, however, we had to add to Soar a very crude simulation of the low-level motor mechanism in Lisp. Crude in the sense that only the time required to move from one location to another was modelled. We therefore agree with Newell's statement that Soar in its current state offers only the right features to implement the interface between cognition and lower-level motor control (Newell, 1990, p. 259).

15.2.4 Multiple types of goals

The model presented in Chapter 2 demonstrated that Soar is able to handle various types of goals. In DRIVER too different types of goals can be active at the same time. As in Chapter 2, we deal with these goals on the basis of Covrigaru's (1989) classification:

- goals and sub-goals (in goal hierarchies)
- single vs multiple top goals.
- long-term vs short-term
- endogenous vs exogenous
- achievable vs homeostatic

The fact that DRIVER employs Soar's *goals and sub-goals in goal hierarchies* is trivial. Distinctly not trivial, and a deviation from the standard Soar practice, is the use of *multiple top goals* in the base level space. In the model in Chapter 2 formal structures were used to represent multiple top goals in the base level space. In DRIVER these goals are *implicitly represented* in the representation of tasks at the top level. An example of this is lane keeping. The lateral position on the road serves as a cue for generating an operator to correct course if some limit is exceeded. There is no explicit goal stored on the state or in the goal slot to correct the course. The combination of lateral position, the fact

that DRIVER is driving and productions in memory keep DRIVER in its lane and only an observer will ascribe to DRIVER the goal of staying in its lane.

The coexistence of both *long-term* and *short-term goals* in DRIVER is also clear. An example of a long-term goal is navigation from one place to another, an example of a short-term goal is the goal of reducing the difference between the current speed and the desired speed. *Endogenous* goals in DRIVER are the goals that are built in by the implementer of the model. Acquired or *exogenous* goals are derived from input from the external environment. Actually all driving-related top goals in DRIVER should be derived from external input and existing older goals (as driving is something learned later in life).

The final distinction is that between *achievable* and *homeostatic* goals. DRIVER incorporates both types of goals simultaneously at the top level. Lane keeping and controlling the current speed are typical examples of homeostatic goals. Navigation is a typical achievable goal. As Covrigaru remarks, within every homeostatic goal there are achievable goals. In DRIVER these scarcely achievable goals are not represented as goals in goal contexts but represented implicitly on the state and achieved by operators.

The diversity of goals is a result of the many ways in which Soar mechanisms can be combined. The following summarises how multiple goals are represented and how they are kept active. Goals are represented in both working memory and long-term memory as:

- *Goal symbols in the goal context.* This is the standard Soar practice. The navigation goal in the navigation space and change-gear goal in the change-gear problem space are examples of ordinary Soar goals.
- *Productions in combination with representations and parameters in working memory.* In DRIVER, goals can be stored in long-term memory in the form of productions and be triggered by the right cues (state attributes) in working memory. The representation of the desired lateral position (in lane keeping) is an example of an indirect goal that is only represented by a parameter and where productions are triggered only when the lateral deviation exceeds a certain value. This value can of course itself also be stored in a production (long-term memory) or in a working-memory representation.
- *Productions in combination with cues from the external world.* In a sense the external world helps us to remember our current goals. For example: I have my hands on the steering wheel and the world is passing by so I must stay in the middle of the road and avoid other objects. However, productions are always triggered by objects in working memory, so this option partly overlaps the previous one.

15.2.5 Parallel processing and multitasking

In Chapter 14 we discussed why the multitasking mechanism in the earlier model in Chapter 2 was much too inefficient to allow real-time processing. We showed that Soar is capable, without formal multitasking mechanisms, of far more flexible and efficient forms of multitasking. Because of the importance of multitasking for this study we will again summarise here how Soar is primarily a parallel system, facilitating parallelism at all levels:

- *Parallel activity in elaboration cycles.* The generation, selection, application and termination of operators is achieved by productions that fire in parallel. Generating and selecting operators in parallel facilitates flexibility; while one operator is active many other operators from different tasks might be generated and preselected. It then depends on the situation which of these is selected for execution.
- *Autonomous productions.* Related to the above is the existence of so-called autonomous productions, which fire independently of the current goals (that is, these productions do not test for a goal symbol). This enables preprocessing of perceptual information and monitoring productions. An example of preprocessing is found in the operators generated for all moving objects in the functional field. An example of a monitoring production is seen in speed control, in which a check-speed operator is generated when the sound level is too high.
- *Multiple top goals.* Multiple productions fire in parallel in response to the multiple top goals represented in the base level space and in Soar's goal hierarchy. As a result of this multiple goals can be processed in parallel.
- *Asynchronous external input.* The Soar input mechanism is able to place many objects simultaneously in working memory. In addition, this happens asynchronously from all the other activity in working memory. In DRIVER we saw how an abundance of objects enters DRIVER's visual store each elaboration cycle, independently of all other activities in working memory.

Soar thus allows parallel processing in a variety of ways. The only reason why in spite of this Soar is largely a sequential system is due to the important constraint that at any given moment there may only be one active goal context and hence only one current operator.

In view of the above we can say that DRIVER's multitasking actually takes place on two levels. If we look only at the Soar trace, meaning the stream of operators that are active while a task is being performed, then DRIVER's multitasking involves only switching task operators derived from a set of tasks that are relevant at that moment. In this sense DRIVER's multitasking resembles that of a computer that performs several tasks with just one CPU and an operating system to ensure that every task is always allocated an amount of time in which to use the CPU¹. However, if we consider instead the detailed activities that take place in working memory, then this computer metaphor

ceases to apply. We then see that for every elaboration many productions can fire simultaneously and may relate to several tasks. This too, then, could be described as *multitasking*.

15.2.6 Soar enables both controlled and automatic behaviour in DRIVER

Related to the issue of parallelism and multitasking is the well-known distinction between automatic and controlled behaviour (Shiffrin & Schneider, 1977). In the Soar literature there are two, partially overlapping interpretations for the term 'automatic'. In one interpretation the term refers to the situation in which Soar has learned to perform a task and no sub-goaling is required for choosing the successive operators within the task. Soar thus 'automatically' chooses the next operator. Another interpretation refers to the 'automatic' activity within elaboration cycles². It is in this latter sense that Newell (1990, p. 136) describes how Soar has the right features to model both controlled and automatic behaviour. In fact, he stresses that if Soar were not able to do so it would fail in its ultimate purpose. The following list, derived from Shiffrin and Schneider (1977), describes the characteristics of both forms of processing.

Automatic	Controlled
Parallel	Serial
Can't inhibit	Can inhibit
Fast	Slow
Load-independent	Load-dependent
Exhaustive search	Self-terminating search
Unaware of processing	Aware of processing
Target pops up	No pop-up

Automatic behaviour

The above description of automatic behaviour relates in Soar to all activity in working memory between two decision cycles or two operators (i.e. within an elaboration cycle or a sequence of elaboration cycles). Take a look at the following list for the similarities:

- productions fire in parallel
- there is no way to inhibit productions between decision cycles
- choices for the next operator can only be made at the decision cycle
- tasks done in elaboration cycles are done fast
- requiring a limited number of 20-millisecond episodes
- activity between decision cycles is usually load-independent
- exhaustive search because it cannot be inhibited
- in some cases the targets pop out, in DRIVER for example for all moving objects in the functional field.

In DRIVER many tasks are automatic. Operators are generated, selected and applied in *parallel elaboration cycles*. Generation *cannot be inhibited*, operators

can only be rejected (and hence inhibited) in the decision phase. Once an operator is installed (and no sub-goaling is required to apply the operator), then execution cannot be inhibited either. For example, once DRIVER installs a motor operator the execution of the movement will commence. All driver operations that take place between two decision cycles are *fast*, as they involve just a *limited number of 20-millisecond* episodes. The generation of operators for moving objects in the functional field is in principle *load-independent* and, finally, the generation of operators for the moving objects in the functional field is a *pop-up* or *pop-out* effect.

Controlled activity

The above description of *controlled behaviour* relates to all tasks that require a number of operators in order to succeed. The following list presents the matches between Soar and Shiffrin and Schneider's description of controlled behaviour.

- task performance is serial; operators occur one after another
- therefore slow (roughly the number of operators times 100 milliseconds)
- individual operators can be inhibited
- in some cases tasks are load-dependent (as when searching through the functional field)
- self-terminating search (if the first car from the right on collision course is found you can stop searching)

Many tasks in DRIVER are *serial* tasks. Executing a motor plan is necessarily a serial activity requiring many 50 to 200-millisecond operator steps and is therefore *slow*. The execution of the motor plan can be *inhibited* simply by rejecting the next operator in the plan. There is *load-dependency* in DRIVER: the more moving objects there are in the visual store, the longer it takes to attend to them all. Search is *self-terminating* when only a moving car from the right is sought.

One of the key problems in implementing a cognitive (Soar) model is that a choice has to be made between solving tasks via controlled or automatic processing, i.e. in many situations there is a choice of implementing a task within an elaboration cycle, using only productions or using multiple operators to achieve the task. An example is searching the functional field. It would be possible to try to cycle through all moving objects within an elaboration cycle; search would then be fast and exhaustive; or operators could be used to attend to all objects; search can then be self-terminating. In this particular example it was decided to use operators, as Wiesmeyer's (1992) dissertation shows that operators model the data in laboratory experiments best.

15.2.7 External memory and situated action

The external environment has been qualified as additional *external memory* by Newell and Simon (1972). Similarly, Suchman (1988) argues that the external environment is as much a source of what a human does as long-term

memory is. The importance of the external environment in DRIVER's decision-making is clear; she relies on it almost completely to provide it with the cues for its decision-making. DRIVER does not build large models of its entire environment, but only samples parts of the world that are relevant to its current decision-making. So in the approach to an intersection DRIVER only represents the most important car (and in some cases only the fact that there is or was a car), but certainly does not build up an entire representation of everything that is going on at the intersection. It is intuitively clear that building up a representation of the environment by using a large number of attend operators is far too time-consuming, certainly in a situation that changes as fast as a traffic situation.

The idea of the high degree of dependency of behaviour on external memory is currently also being suggested under the term *situatedness* (Suchman, 1988; Chapman, 1990). Chapman argues that in most situations humans can do without deep reasoning, in other words humans build up simple procedures that are sufficient in the everyday interaction with the external world. To illustrate his idea about situatedness, Chapman built a computer model of an amazon, called Sonya, who survives in a (simulated) complex dynamic world where dangers lurk in every corner. With some basic natural language-understanding routines, visual routines and routines for moving and fighting she survives in her dangerous world (using some natural-language guidance from the modeller). Sonya rarely engages in real planning.

DRIVER is also a highly situated system. Like Sonya, it also survives in a complex dynamic world where dangers lurk around every corner. Second, DRIVER, like Sonya, is mainly data-driven and also survives by reacting appropriately to cues from the external world. Third, note that the routines in Chapman are essentially the same as the operators in DRIVER and, finally, DRIVER almost never display goal-stack intensive search behaviour.

15.2.8 Bottom-up and top-down control

An essential aspect in the concept of situatedness is that the external environment determines to a large extent every next move; behaviour is therefore to a large extent *bottom-up* controlled. We have demonstrated that DRIVER displays both bottom-up and top-down or conceptually driven behaviour. In DRIVER, bottom-up and top-down processing are closely linked and sometimes it is impossible to distinguish between the two. The best example comes from the visual orientation strategies: *What DRIVER sees determines the goals it will set, but the goals then largely determine where DRIVER will look.* An example that shows how they are linked together is given in the following episode close to the intersection.

Data driven	Concept driven
1) Detect intersection	2) Annotate mental model with <i>approaching-intersection</i>
4) and also detect other objects	3) Determine <i>manoeuvre-at-intersection</i> by navigation
6) When looking right, generate objects for every moving object	5) Search for traffic signs
8) Detect car on collision course	7) Close to intersection generate look right, left and forward operator
	9) Propose change desired speed, install change gear.

The classification into data-driven and top-down is a debatable one. On the left we have listed the more passive actions, on the right the actions for which (in the past) goals and learning were responsible. However, in DRIVER's current form all actions take place in the base level space; in a sense, therefore, they are all automatically generated and selected as a response to actions in the outside world. In this example the distinction between data-driven and conceptually driven is therefore largely a descriptive one.

15.2.9 Declarative vs procedural knowledge, knowledge compilation

Anderson's (1983) well-known distinction between declarative knowledge and procedural knowledge in ACT* and the transition from declarative knowledge to procedural knowledge can be mapped onto Soar's native mechanisms and bears some relevance to the distinction between automatic and controlled behaviour.

Declarative knowledge is often described as "what to do" knowledge or passive knowledge that has to be interpreted before it can have an effect. Procedural knowledge is often described as "how to do it" knowledge or directly applicable knowledge. If declarative knowledge is interpreted successfully into an internal or external action, then the interpretation may be stored as procedural knowledge in procedural memory. The transition from declarative to procedural knowledge is called knowledge compilation. Rosenbloom, Newell and Laird (1989) argue that there is a fairly direct mapping from these concepts onto Soar's mechanisms, notably sub-goaling and chunking.

A fine example of knowledge compilation is DRIVER's planning of a gear-change operation. Before DRIVER learns how to change gear it only has a declarative representation of the car and the body (including of course the dynamic representations for its body and the state of the car), but DRIVER does not possess the procedural knowledge of how to change gear. It carries out its first gear-change procedure in 80 decision cycles and about 20 sub-goals. In the process it learns 30 procedural chunks that will allow it to use only 12 decision cycles for a gear-change manoeuvre next time.

15.3 Fundamental problems with Soar and DRIVER

In our attempt to evaluate Soar's practical and theoretical suitability, this section will now discuss some more or less fundamental problems that we encountered in implementing DRIVER. 'More or less' refers to the fact that in some cases a problem seems more fundamental than in other cases. For example, in some cases an essential mechanism is missing but it is possible to imagine how this mechanism might be incorporated. Incorporating a new mechanism in Soar is a complicated affair because the new mechanism must work together with all the other available mechanisms. If a new mechanism can be implemented by productions only then there is a higher chance that it will integrate smoothly. If a mechanism requires that the fundamental architecture be changed the chances that it will conflict with the mechanisms already available are greater.

15.3.1 Soar's default search control rules

Soar's standard default rules are, at the level of immediate behaviour, inefficient, too slow and not resistant to interrupts.

There are forces within the Soar community to keep the default rules fixed (Newell, 1990). The advantages of doing this are of course that code remains portable and can easily be integrated into other researchers' applications. In addition, the default rules offer a way to experiment with various types of search and it is shown that most types of search in human problem-solving can be modelled using the Soar default rules as the basic layer. If, however, we examine the psychological validity of the default rules we look mainly at the qualitative fit between human search behaviour and the search strategies of the models constructed (Laird, Rosenbloom & Newell, 1986; Newell, 1990) and only superficially at the fit for solution times.

We saw in Chapter 2 that timing problems in particular can arise if Soar's default rules are used while several tasks are being performed in interaction with the external world. We do not regard this problem as a fundamental one, however, since we saw in Chapter 3 that variations on the default rules are possible that result in the same qualitative behaviour but that are far more time-efficient and result in a type of problem-solving that is more resistant to interrupts.

15.3.2 Problems with Soar's learning mechanisms

Soar is capable of handling a great variety of learning phenomena using chunking as the basic mechanism (Steier et al., 1987; Rosenbloom & Aasman, 1992; Newell, 1990). Although in DRIVER we have successfully used Soar's learning mechanisms on a number of occasions – such as learning motor programmes or navigation plans – we nevertheless encountered a number of essential problems during the implementation process.

Data-chunking

The first problem relates to data-chunking. Data-chunking is a special mechanism for learning associations between representations that reside in the same problem space. We come across data-chunking several times in DRIVER, both in the default rules we used and in learning external interaction. The data-chunking problem and its solutions are discussed in detail in Rosenbloom and Aasman (1992) and Newell (1990) and were also discussed to some extent in Chapter 3. We will again briefly summarise the data-chunking problem here because we feel that it is essential for an understanding of the following sections about error recovery and learning from external interaction.

Normal chunking

Before we discuss data-chunking we will first summarise briefly how Soar normally learns. Remember that Soar learns whenever it solves an impasse. A solution to an impasse (in the form of a Soar object or preference) is passed up from a sub-goal to a higher goal; the consequence is that a chunk is built and the result becomes part of the action component of the chunk. The condition side of the chunk contains those data structures that were accessed directly or indirectly when generating the results in the sub-space. Note that only the structures that existed before the sub-goal arose are included in the condition part of the new rule. The left half of Figure 15-1 illustrates the process of building up a chunk.

The data-chunking problem

Now imagine what happens if there are two representations R1 and R2 in the base level space and you want to learn the association *if R1 then R2* (for example, R1 is the perception car-from-the-right and R2 is the verbal instruction: stop). The naive approach would be to go into a sub-goal, test for R1 and R2 and then pass R2 up. However, the chunk that is then obtained will be: *if R1 and R2 then R2*. The reason that R2 appears in the condition side of the chunk is that in order to pass up R2 from the sub-goal Soar had to access R2 from the top level.

A solution

There is a fairly technical solution to the data-chunking problem, which in simple terms can be expressed as follows: in the sub-goal context all possible representations R_i are proposed. All R_i 's that are not R2 are deleted and only then is R2 passed up and the appropriate chunk be learned. The reason why R2 will not show up in the condition side of the chunk is that it was generated before the R2 in the higher space was accessed³.

A criticism of the data-chunking solution

Although the basic problem of data-chunking has been solved, a number of problems prohibit it from being effectively used in normal Soar practice, including DRIVER. The first problem is that the number of representations

that must be generated in a sub-goal context is very large and it is a time-consuming process for the production matcher to regenerate structures by the generate-and-test process. Note that R2 in the explanation above stands for a complex data structure. In the data-chunking process R2 will therefore not be generated as a whole but will be built up from a selected set of primitives⁴.

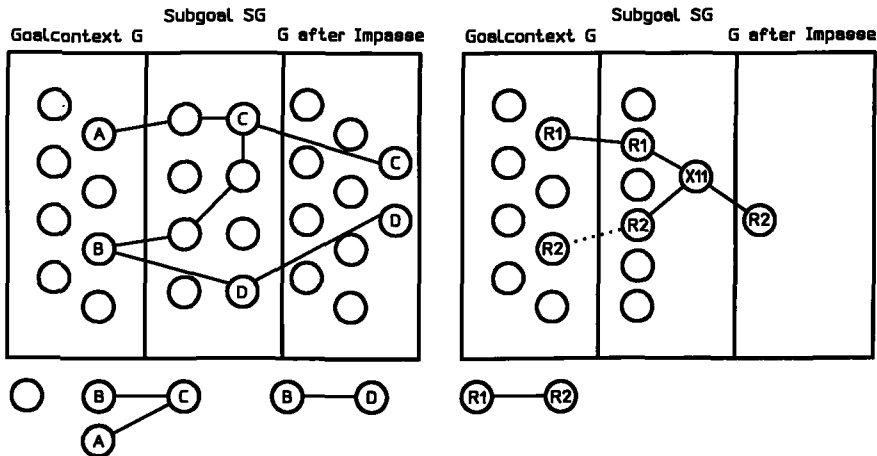


Figure 15-1. Figure (a) shows an abstraction of three goal contexts. The left part shows the contents of working memory in the goal context G. Note that each node stands or may stand for a complex object. In this goal context G an impasse has arisen and the middle part shows the sub-goal context SG. The nodes in the middle show all objects that were created in SG. Nodes that are connected to an object in the super-goal were created while testing for the object in the super-goal context. The right part shows the moment when the impasse for G is resolved by sending the objects C and D to the super-goal. Note how two chunks are built. Chunk 1 demonstrates that C was indirectly dependent on A and B. Chunk 2 demonstrates how D was generated only because of B.

Figure (b) shows the data-chunking solution. In this figure three goal contexts are again shown. A node in the sub-goal that is connected by a solid line to a node in the super-goal again stands for a node that is generated while testing for the node in the super-goal. In this example an intermediate object X11 is built that depends on all the features of R1. Note how nearly all the other objects are generated without testing for the super-context. The dotted line signifies that R2 is *selected* by testing for R2 in the super-context (thus not *generated* while testing for R2 in the super-goal). The production that sends R2 up to the super-goal also tests for X11 and thus the chunk if R1 then R2 is built.

The second problem is that Soar's current data-chunking is *too good*. That is, the solution seems more technically than psychologically valid. In the current version no intelligence is required to rebuild an R2 in the sub-goal context. It is a purely syntactic process; anything can in principle be built. Just give Soar the right set of primitives and enough time and it will learn the data chunk. By too good we also mean that the chunks that are learned can be excessively big. This is not really part of the data-chunking problem. There is currently no limit in Soar to the size that a chunk can be (see also Section 3.3 for more on this issue).

In the following sub-sections we will discuss learning problems that require data-chunking for their solution.

Learning from external interaction

Soar lacks an adequate theory of learning from external interaction. We will try to explain this on the basis of learning an evaluation of an operator that is being used.

The standard practice in Soar is to learn an evaluation of an operator by applying the operator in a sub-goal on (a copy of) the super-state. After the application of the operator the newly created state is evaluated within this sub-goal. When the evaluation is finished it will be passed up to a higher state and an evaluation chunk is built. The following shows the prototype of an evaluation chunk.

If	state S and operator O is proposed
Then	operator O receives evaluation E.

Unfortunately this standard practice does not always work with a system that interacts with the external world. For example, say S represents a state in which DRIVER is close to the crossing and in which a child on a bicycle is approaching the intersection from the left and is on a collision course. Say that O is the operator that proposes an acceleration, since formally DRIVER has right of way, and that O is then indeed performed. Say then that E is the instructor's very negative evaluation. The rule that DRIVER ought to learn is that one must not accelerate when children from the left are on a collision course.

Rather more formally, this situation can be represented in Soar terms as follows: S_{t1} is the base level state in the base level space base on time $t1$; O is a motor operator in the base level space; remember that all motor commands take place in this space; E is an evaluation that came about in the base level space via perception. Moreover, E is an evaluation of a state S on time $t2$ (S_{t2}) that was changed by an external action from O. S_{t1} has thus probably already disappeared from the working memory. The evaluation chunk to be learned is again:

If	state S_{t1} and operator O is proposed
Then	operator O receives evaluation E.

Because S, O and E are all in the base level space it is clear that here we need data-chunking. The additional problem is here that we want to learn an association between (1) state S on $t1$, i.e. before it was changed by the application of the external operator O, (2) operator O in its proposed form, i.e. before it was applied and (3) the evaluation E of state S on $t2$, i.e. after O has been applied.

Soar thus has to solve two problems before it can invoke the data-chunking process. The first problem to be solved is the operator assignment problem, i.e. Soar must find an answer to the following question: can the evaluation *E* of the current state be assigned to one particular operator? The other problem is that it must also retrieve the state at the moment that this operator was generated. There are a few suggestions for solving these problems, but each has its own drawbacks. The reader will notice that the following solutions were also mentioned in Chapter 3 in the flat-stack variation of the default rules. There too data-chunking was used in an almost identical problem.

1) *Pre-store* the main features: before *O* is applied the main features of *S* that seem relevant as future cues for operator generation and selection are set aside. The problem here is that Soar knows beforehand that the following external operator will or should be evaluated.

2) *Retrieve* the main features: Soar puts the main features of state *S* in a chunk and creates retrieval cues so that the state may be retrieved when necessary. It implies again that Soar knows beforehand which operator will be evaluated. An additional problem now is triggering the chunk that retrieves the retrieval cues.

3) *Reconstruct* the main features: in other words, use reasoning to reconstruct the state obtaining when the operator was generated. Right, I have performed *O* and I know that the general effect of *O* is *X*. The current state is *S*₂, so the previous state must have been *S*₁. This requires that we know the general effects of operators (the semantics of operator application) in the external world.

Each of these solutions will however in many situations lead to errors, that is, to wrong conditions so we therefore also need to discuss the following problem.

Error recovery in external interaction

An additional problem in learning from external interaction is that we cannot expect that a system interacting with the external world will learn all the appropriate rules at once, indeed in some case what has been learned will be over-general or over-specific or even just plain wrong. Such a system must be able to recover from incorrectly learned knowledge. If we just look back at our previous example we see that recovery from an over-general rule actually took place there. The over-general rule that traffic from the left should yield is now corrected by a rule that states that this does not apply to children on a bike and on a collision course. A problem that we did not deal with in the previous version is that we now have two conflicting evaluations for the same operator, a positive evaluation and a rejection.

A possible solution

The solution to this problem and a strategy that we used in an experimental version of *DRIVER* involves adding two new elements to our data-chunking mechanism: (1) evaluations are made unique and (2) while learning an

evaluation Soar learns a second chunk that invalidates earlier evaluations. Appendix 1 provides a simple Soar program for learning from external interaction and error recovery for evaluations. The reason for presenting this program in the appendix and not in a regular chapter is mainly that this technique is not yet an integral part of DRIVER.

The appendix will show (1) how Soar learns the evaluation of an external operator via data-chunking, (2) how this data-chunking is selective, i.e. in the data-chunking space only the relevant structures are generated, (3) how Soar can do error recovery by learning another evaluation for the same external operator and in the process (4) unlearn the older evaluation, and finally (5) how preferences are generated from the evaluations¹.

Fundamental problems to be solved in error recovery

The method presented in the appendix is only a technique for the basics of error recovery. Most of the intelligent work must still be done by the user of the technique. The fundamental problems to be solved are:

- 1) The *operator assignment* problem: which operator or which combination of operators is to blame for the new situation?
- 2) The problem of *pre-storing, retrieving or reconstructing* the appropriate state and operator cues after the external operator has been applied.
- 3) Even if we know the situation in which an external operator is created and we have the situation, then we still have the problem of how to *abstract* and *select* from the concrete situation before the evaluation is data-chunked.
- 4) This mechanism works for evaluations that arrive at the top space. If we already have the evaluation stored in a problem space then the data-chunking mechanism is still needed, because an evaluation of an external action is always *about* the *outcome* of an action. Evaluations in sub-space thus always need to be generated independently of these outcomes.
- 5) The problem of the *evaluation semantics*. It is fairly easy to copy Soar's preference scheme by generating it from evaluations (the method used in the appendix). The explicit preferences from Soar are thus introduced via the back door. And, even worse, they are now made specific. This has many consequences for Soar in general and this issue needs to be studied.

In this section a great deal of space is devoted to the problems of external learning in the context of Soar. This is currently an important point for attention in the Soar community and there seems to be some progress. See for example Howes (1994) for the use of *recognition chunks* in datachunking or Huffman (1994) for the use of natural language and recognition based datachunking in external learning tasks.

However, the method proposed above has so far scarcely been described. For example, if we look at the integration of external interaction, planning and learning in the Robo-Soar project (Laird, Hucka, Yager, & Tuck, 1990a) or at

the correction and expansion of domain know-how using outside guidance (Laird, Hucka, Yager & Tuck, 1990b), we see that the evaluations do not appear in the top goal but in deeper sub-goals, thus avoiding the problem of data-chunking, but also a number of other problems mentioned above. In itself this is not of course a problem in the context of learning AI systems, but if we continue to assume the UTC idea that perception appears in the base level space, we do encounter all of the problems referred to above.

15.3.3 Problems with Soar's working memory

In implementing DRIVER we encountered two problems that were directly related to Soar's working memory. The first problem is a more theoretical one and relates to the size of Soar's working memory; the second problem is both a theoretical and a practical problem and relates to the fact that Soar does not have an intrinsic decay or forgetting mechanism for objects in working memory.

Working memory size

One general criticism of Soar is that it lacks a constraint on the size of working memory. In principle Soar can fit an infinite number of objects in working memory. We thus see that DRIVER stores many representations simultaneously in working memory: motor plans, objects extracted from the visual store, navigational information, speed and steering information and timing information, totalling in some situations more than 500 Soar working memory elements (wme's).

Five hundred elements seems very large compared to the generally accepted 'seven plus or minus two' chunks for working memory (Miller, 1956). However, we cannot compare a Soar working memory object to a chunk. One object or chunk of information in Soar in general consists of many working memory elements. For example, a motor program in DRIVER may consist of 200 wme's. Here, then, we are in the middle of the 'how big is a chunk' debate (Simon, 1974)⁶.

Whatever the size of a chunk may be, the number of items that may reside in Soar's (and DRIVER's) memory seems too high. A simple way to limit the number of objects in working memory would be to set arbitrary limits. For example limits to the number of wme's that may reside in WM or the number of links that may be attached to the state. However, in the light of the previous discussion one can imagine that finding this limit will be almost impossible. Another way to constrain the number of objects in working memory is to allow only sequential access to objects in working memory. In principle all objects in working memory are accessible to productions in LTM. However, an operator-based attention mechanism might be built that allows inspection of an item only when it has been 'attended to'. This approach has been tried by Polk and Newell (1988) for verbal reasoning tasks. In DRIVER we did not

try this for working memory, though we did use it for extracting items from the visual store.

The lack of a decay mechanism in Soar's working memory

When objects in human working memory are not attended to they will decay from working memory. The decay half-life of elements in working memory is estimated to run from 5 to 226 seconds (see Card, Moran and Newell, 1986). People must invest effort, for example by using *maintenance rehearsal*, to keep the relevant objects in working memory.

A problem with Soar is the lack of a *decay* mechanism. This lack is a serious problem for DRIVER, as its working memory rapidly fills up in complex traffic situations. This problem was extensively discussed in the chapter on speed control (Chapter 11). We saw that a Soar programmer can keep the number of elements in the working memory under control in two ways:

The first way is to keep complete control of the book-keeping of what does and what does not belong in working memory. This involves, among other things, writing productions that remove objects from the working memory. This method only makes sense for objects that are fixed on the Soar state by means of operator-applications. An example of this would be to remove all intersection-related objects from the working memory after an intersection has been crossed. There are two disadvantages to this method. The first one is that in a complex task you can never foresee all the situations and so it will be very difficult to keep your books properly balanced. The second disadvantage was referred to earlier, in Chapter 11, namely the chance of the frame problem occurring. Many objects in Soar's working memory are dependent on other objects in the working memory. This means that if you remove objects from the working memory you really have to test whether all other objects in the working memory are still relevant.

The second way of doing the book-keeping for what does and what does not belong in the working memory is to rely entirely on Soar's built-in truth-maintenance system. Roughly speaking, in this system objects only remain in the working memory if they are directly (or indirectly) based on perceptual input. If a perceptual input alters, all objects that directly or indirectly depend on it will be removed from the memory. In Chapter 11 we showed that this method is impossible for the driving task owing to the fact that perception in DRIVER often changes drastically (due to the eye movements).

It is clear that intrinsic decay mechanism and mechanisms for maintenance rehearsal belong on the Soar research agenda.

15.3.4 Perception and motor control

The chapters on perception and visual orientation discussed how Soar lacks low-level perception and motor control. If we look at perception we see that

Soar (and DRIVER) provide only the basic interface functions for perception. Basic issues such as object recognition are currently not addressed in Soar. The same applies to motor control. Soar and DRIVER provide only the interface, everything below that is currently not addressed. The important question to ask in the context of this study is whether the low-level mechanisms are adequately simulated so that at the cognitive level a functional model of driving could be made. The answer to this question is that the simulations of the low-level mechanisms are in any case sufficient to support DRIVER's eventual behaviour, as described in Chapter 14.

15.3.5 Time and timing problems

Michon (1993) argues for a distinction between an implicit and an explicit temporality of human action. The implicit mode deals with direct tuning of action to the dynamics of the surrounding world. The explicit, reflective mode deals with time viewed as past, future and order of duration. Jackson, Akyürek, and Michon (1993) argue that most models of cognition that have reached a sufficient degree of formalization are not endowed with a sense of time; moreover, these models are certainly not capable of also dealing with the difference between an implicit and an explicit mode of temporality.

Unfortunately this criticism also applies to Soar: Soar too lacks a basic model of both the implicit and the explicit mode of temporality. When implementing DRIVER this constantly gave rise to problems. In almost every part of DRIVER we struggled with timing problems and estimations of duration. We came up against the problem, for example, in speed control (the duration between speed checks), visual orientation (how far am I from the intersection) and memory decay (removing elements from DRIVER's mental model).

The two solutions that we came up with in DRIVER are rather half-hearted. The first solution is to put timing in perception; for example, by letting DRIVER perceive time-to-intersection we introduce time information via the back door. The second solution is by explicitly building clocks in Soar. The clocks that we built in Soar just count the number of elaborations that go by, so that the resolution of DRIVER's clock is at least 20 to 30 milliseconds.⁷

15.3.6 Summary of our problems

All the problems described above were encountered while implementing DRIVER but the problems differ in their degree of seriousness. The problems with the default rules and those with learning with external interaction and the error recovery can probably be solved without Soar itself having to be fundamentally changed. It may also be possible to fit the problem of memory size and some form of temporality into Soar in some way.

Memory decay will be more difficult to fit into Soar because this requires probably changes into the heart of Soar's production-system mechanism. For example, a decay mechanism that is used in ACT* would drastically change Soar's nature. In fact Soar is now a fully deterministic system. If working-memory elements were removed in a stochastic way Soar would lose this deterministic nature.

The main problem we come up against, and the one whose solution would require the most changes in Soar, is of course perception and motor control. In the preceding chapters we discussed these problems extensively, but we would like to reiterate once more: Soar in this form is only the interface to perception and motor control and thus only partially implements a UTC.

15.4 Suggestions for future research

Having arrived at the end of this study we would like to list a number of other issues that might be covered in future versions of a cognitive model of driving behaviour. The reader will notice that a number of these issues are in fact fundamental problems disguised as suggestions for future research. A complete list of topics for future research would be exceedingly large because a human driver participating in traffic makes use of most of the human skills and capabilities. We have selected a number of these that we ourselves regard as important for the next version of DRIVER. Several of these topics also appeared on Newell's research agenda for Soar (Newell, 1990, pp. 434).

15.4.1 Extend the driving task domain

The very first step is of course to expand DRIVER'S task domain. At the moment DRIVER does only unordered intersections, with navigation and lane following as secondary tasks. The next step would be to include other types of manoeuvres. For us, it would be most sensible to aim for the tasks that are addressed in the small world simulation in GIDS, the European project that we referred to in the introduction of this study and which is also coordinated at the Traffic Research Centre (Michon, 1993). The GIDS small world is far more advanced than the world used by DRIVER and covers, in addition to handling intersections and lane following the following tasks.

- car following
- overtaking
- negotiating a roundabout
- merging into a traffic stream
- exiting from a traffic stream

15.4.2 Representations and mechanisms for working with quantities

Soar has no theory of how numbers and quantities are represented and processed. Newell (1988) calls this problem the problem of "the basic quantitative

code"; he argues that an architectural mechanism is required, since the issue is the transduction from perceptual signals, which are quantitative (intensity, direction), to symbols in Soar's representation and from such symbols to motor signals, which are again quantitative (force, direction). This transduction must not only be built in, it must lead to a representation code that is built in (Newell, 1990, p. 437).

The lack of such a theory is a problem in DRIVER. Floating-point numbers and integers are used in DRIVER to represent quantities and magnitudes (angles to other drivers, distances, speed, etc.). Most of these numbers stem from perception (and thus only simulate the signal/symbol transduction).

15.4.3 Memory for frequency of occurrence

People are reasonably good in estimating how often a certain event occurred and it is clear that this capability is used in driving. For example, the speed at which drivers approach an intersection seems to be a function of the frequency of other traffic arriving at the intersection. *Memory for frequency of occurrence* is a research topic in psychology but also an issue in driving.

It is generally assumed that several fundamental aspects of experience are stored in memory by an implicit or automatic encoding process. Hasher and Zacks (1984) review the evidence that suggests that information about frequency of occurrence is encoded in such a manner. They show that frequency information is stored for a wide variety of naturally occurring events. Laboratory research shows that normally powerful task variables (for example instructions, practice) and subject variables (for example age, ability) do not influence the encoding process.

Hendrickx (1991) studied the role of *frequency information* and *scenario information* in risk judgement and risky decision-making. The problem that he addressed is whether people in risky situations base their decisions on imaginable scenarios of "how things might go wrong in this situation" or on their knowledge of "how often things tend to go wrong in this type of situation". Not surprisingly, he found that both forms of information play a role in for example decision-making in risky traffic situations.

Of course, Soar can store frequency information.⁸ However, it will require considerable research to get DRIVER to pay attention to all possibly relevant (traffic) situations, find the right cues under which this situation should later be retrieved and then chunk this information into memory.

15.4.4 Emotions

The trend in the last decades of emotion research has been for cognition to play an important role in starting and maintaining the physiological correlates of emotions. For a review of the importance of cognition in emotion see Frijda

(1986). Emotion is currently not included in Soar, but Soar should address at least the cognitive side of emotions (Newell, 1990) if it is ever to earn the UTC classification.

The important role of motivational and emotional factors in driving has been described by, for example, Näätänen and Summala (1974, 1976). However, most readers do not need the literature to realise the role of emotion in driving. Everyone will be familiar with the stories that drivers nowadays are becoming more and more aggressive and those readers who are car drivers may recognise the effect of emotions on their own driving behaviour.

To give an impression of the content of the emotional factors that can affect driving, we will briefly discuss here a study by Labiale (1988). Labiale found, in a survey of 1006 drivers, strong correlations between patterns of driving behaviour and the cognitions and attitudes towards the car and car driving. The following factors from a factor analysis, listed in decreasing order of importance, correlate most strongly with reported and actual driving behaviour⁹:

- feeling of urgency
- experience in driving (the fun and fears one has in driving)
- anticipatory and careful behaviour
- emotional reactions to traffic problems
- search for strong sensations in driving.

Using these factors, Labiale comes up with a general driver taxonomy. He distinguishes five classes; the most extreme ends of his categorisation are Class I (26.3 %), calm, disciplined and fuel-saving drivers, and Class V (7.7%), aggressive, flamboyant, pushy drivers. The following is Labiale's description of Class V drivers:

... let us consider how group V scores on the first aspects (feeling of urgency). He often drives in haste and is never relaxed when driving; he drives fast, steps hard on the accelerator when starting, likes to strain the engine and shift gears with a cold engine; he has very bad lane discipline, never allows vehicles to merge in front of him, drives very close to vehicles in front, never decelerates in advance when approaching a curve; he accelerates and overtakes through amber lights; he is unable to wait patiently behind a slowing down or stationary vehicle; he becomes very angry when other drivers make errors and he hoots at drivers who force him to slow down. He regards his driving as smooth, fast, sportive, vigorous, competent and his behaviour on the road as impulsive, aggressive, energetic, time-pressed, competitive, impatient and commanding respect. He considers that his driving style is true to his nature and unlikely to change (Labiale, 1988).

Note how nearly all of the above sentences in this quote bear a strong cognitive load, suggesting that at least some of the emotions involved in driving can be modelled in Soar.

15.4.5 Individual differences

The previous section on emotion outlined the large differences that may exist between drivers. But human drivers are also remarkable for their consistency. We saw in Chapter 4 that expert drivers largely develop the same speed control and visual orientation strategies in the approach to several types of intersections. It thus appears that on the one hand external constraints force drivers to learn similar strategies, while on the other hand there also appears to be considerable scope for individual variation.

Soar is by its nature suited to modelling differences in cognitive strategies in problem-solving and thus in driving. Soar is also a tool for studying the acquisition of these strategies. The following lists some factors that may contribute to differences in learned strategies in human drivers. For each of these factors it is relatively easy to visualise how this factor can be manipulated in Soar or in the world that DRIVER lives in.

- *Instructor.* What are his instructions, does he teach by positive feedback or corrections only? The Groeger et al. (1990) study showed that some instructors provide up to three times as many comments as others, irrespective of their pupils.
- *Actual driving experiences.* Drivers with experience in only rural areas will be different from city drivers. The car-following distance between car drivers on motorways is shorter in more populated areas. Drivers from less populated areas notice this when driving in more populated areas.
- *Physical properties.* Drivers who have problems turning their necks probably look over their shoulder less often and may compensate by relying more on their mirrors.
- *Emotional factors.* (see previous section)
- *Speed and efficiency of information processing.* Information processing is faster in young drivers and that also results in differences in driver behaviour, for example in gap acceptance in traffic merging (Van Wolffelaar et al. 1991). On the other hand, older people who know that they are slower avoid situations where overload is likely to arise¹⁰
- *Car properties.* It is possible to brake later in a car with a better braking system. For example, it seems that Mercedes drivers who have ABS in their car have more collisions from behind. Wilde's theory of risk homeostasis predicts that this is something to be expected (Wilde, 1982).

15.4.6 Incorporate natural language

The incorporation of natural language in DRIVER on our list of future research topics may seem exotic at first but in our opinion the ability to employ natural language in DRIVER would be a very valuable addition. Looking at the driver lessons in De Velde Harsenhorst and Lourens (1987, 1988) and Groeger et al. (1990), we see that drivers learn to a large extent from verbal advice and corrections. In the De Velde Harsenhorst and Lourens study we find up to one remark per 30 seconds. The instructor tells the novice beforehand what he should do and afterwards what he should have done. In both cases extensive problem-solving is required to translate natural language into rules that are effective the next time the driver is in the same situation.

The incorporation of some form of natural language in DRIVER will be easier than one might at first expect. Great efforts are being made to incorporate natural language in Soar (Lehman, Polk & Newell, 1991) and some Soar researchers are already trying to incorporate existing natural-language capabilities into their Soar systems (Huffman, 1994). The ability to employ natural language in DRIVER would be a very valuable addition. Learning from external interaction in driving will become more effective and 'natural' if we include natural language.

15.4.7 Soar and connectionism

Another, partially practical way of getting Soar under way (that is, on the road) would seem to be to build a hybrid system by connecting Soar to a connectionist system that does the basic perception. Soar is a symbolic system and (therefore) good at multi-step problem-solving. However, Soar is computationally too expensive for object recognition. Connectionist systems are very good at recognizing objects, but very bad at multi-step problem-solving and multitasking (See Levelt, 1989; Quinlan, 1991). What could be better than to hook up a connectionist system to Soar? Efforts are also being made to reimplement Soar itself, i.e. the Soar matcher, with a connectionist matcher (Cho et al., 1991; Rosenbloom, 1989).

15.5 Final conclusions

We will now give a brief summary of our final evaluation of Soar's practical and theoretical suitability for modelling a complex dynamic task such as driving is.

First of all let us deal with its theoretical suitability. A positive aspect is the way Soar provides an appropriate framework for experimenting with the integration of problem-solving, high-level motor control and high-level visual orientation, multitasking and (to some extent) learning in a complex real-time environment. We saw that it is possible to use Soar to create a reasonably functional model of driving behaviour in which the main sub-tasks of the

driving task can be mapped onto Soar. We also saw that Soar offers several important, non-driving-specific behaviour principles that are essential for the functioning of DRIVER.

On the negative side of our evaluation of DRIVER's theoretical suitability is the fact that Soar lacks a number of important mechanisms. The most serious problems that we encountered in implementing DRIVER are learning from external interaction, error recovery, the low-level perception and motor control, working-memory decay, the representation and use of time information and, finally, working with quantities.

Most of the above-mentioned problems have been pointed out elsewhere. Newell (1990) refers to almost all of these problems. The problems that are unique to this study, however, are those that have arisen as a result of our intention to comply with the minimal scheme of immediate responses, that is, the starting-point of having all perception and motor commands taking place in the base level space. We saw in this study how our problems with multitasking, learning from external interaction and error recovery are all coloured by this starting-point.

Another issue is Soar's practical suitability for modelling driving behaviour. We distinguish two aspects. The first relates to the question of whether Soar is a practical system for modelling driver behaviour. If we look at Soar purely as a programming tool the answer is "Yes, if you are a happy hacker; No for all others". For most people Soar is a terrible system to work with. The OPS syntax for productions, the difficult debugging process and a chunker that interferes with your programming take months to get used to. Fortunately, at last efforts are being made to provide Soar with a more user-friendly graphical user interface.

The second aspect of Soar's practical suitability is that it is rather difficult to communicate with the scientific community about the models made in Soar. There are several reasons for this. First, the user-unfriendliness obstructs Soar's expansion. Another problem is that in the literature Soar is presented under a variety of guises. Cognitive psychologists mainly treat Soar as a cognitive modelling tool or unified theory of cognition, scientists from the field of AI treat it sometimes as a problem-space computational model and sometimes as an implementation language.

There is, however, more to be said in reply to the question of whether Soar is suitable as a cognitive modelling tool. To start with it can be said that the value of working with Soar lies in the value of modelling itself. In Posner's *Foundations of Cognitive Science* (1989), Simon and Kaplan look at the role and value of computer simulations of cognitive behaviour.

They argue that the general value of building a computational model is that in attempting to find a parsimonious set of mechanisms and the appropriate knowledge to get the mechanisms to work the modeller (a) uncovers interesting and unexpected interactions between the mechanisms, (b) finds new and unexpected behavioural patterns that arise naturally from his model and (c) learns to ask the right questions in the domain that the computational model is tackling. We find that Soar is an excellent facilitator for these positive consequences of modelling.

We conclude this evaluation with a statement from Newell that is an extension of what Simon asserts above, namely that "*in AI progress is made by building systems that perform: synthesis comes before analysis*" (Newell, 1977, p. 970). That this does not apply to AI alone, but also to cognitive science in general, is demonstrated by all his Soar work. And I hope it is demonstrated again in my work.

15.6 Notes

¹ The comparison extends even further, since it depends on a variety of circumstances how much time each task is allocated. In the case of computers it is sometimes the case that difficult, though not important tasks are postponed until the night and that other tasks, such the interaction with users, are given higher priority. In DRIVER too we see that less important tasks sometimes have to wait until after the intersection.

² There is overlap because the selection of the next operator is performed within elaboration cycles.

³ The representations or objects R_i are usually composite objects, built up from lower-level symbolic primitives. Generating all representations R_i is thus impossible because by combining all the lower-level symbols an infinitely large set is obtained. In the data-chunking mechanism the composite objects R_i are slowly built up in several phases. Suppose R_2 , from our example, resides on the state and is the change-gear action:

```
(state <s> ^action <a>)
(action <a> ^name change-gear ^from <f> ^to <t>)
(number <f> ^name three)
(number <t> ^name two)
```

In the first phase the attribute ^action is built, i.e. all basic symbols are proposed and only "action" is chosen. In the second phase the attributes ^name, ^from and ^to are added onto action (and the value for ^name is filled in). In the following phases the remaining objects are filled in one by one.

⁴ A large number of questions relating to these primitives have not yet been resolved. Some of these questions are: (1) What is the basic set of primitives? Are we born with them or do we acquire them? (2) Do people enhance their data-chunking efficiency by generating more complex representations in the data-chunking space? By more complex representations we mean representations that are built up from the lower primitives. This might explain expert behaviour in all kinds of fields. Experts might have learned to generate better (higher-order) primitives so that they do not have to build up the representations from the lowest primitive every learning episode. (3) Do people enhance their data-chunking efficiency by employing *selective data-chunking*? In

other words, do people have special data-chunking spaces for task X where only primitives relevant to task X are generated?

⁶ The specimen task in the appendix is to learn whether to look first right or first left when DRIVER is close to the intersection and the intended manoeuvre is a right turn. The appendix contains three runs which we will summarize briefly here. In the first run the look-left and look-right operators are available and the look-right operator is selected at random and applied to the external world. The operator receives a positive evaluation from the external world (via the Soar input) at the top level. The positive evaluation is learned via data-chunking. In the second run the same operator is chosen again but this time is not random (note that it now has a positive evaluation). However, this time it receives a very negative evaluation after it has been applied. This negative evaluation is also data-chunked and, in addition, the previously learned positive chunk is unlearned by making the earlier evaluation *invalid*. Finally, in the third run we see that the other external operator (for looking to the left) is given a chance of being applied.

⁶ Simon argues that experts do have bigger and probably hierarchically organized chunks, *but not more chunks* in WM.

⁷ The lack of timing is not always a problem. As long as a sequence of actions unfolds according to a scenario then timing is of lesser importance. A good example is shifting gear: the individual motor actions are far more determined by their mutual constraints and the speed (or rather slowness) of arms and legs than by timing cues.

⁸ There seems to be a relation with the issue of episodic memory versus semantic memory (Tulving, 1969). Newell (1990) argues that Soar's memory can be used for both types of memory, but the mechanisms for storing episodes is an issue on the research agenda.

⁹ One may disagree with the methodological approach taken by Labiale from a cognitive psychological point of view but keep in mind that his research is only presented to illustrate that (a) emotional factors play a role and (b) each of these factors has a cognitive load.

¹⁰ Wiebo Brouwer, personal communications.

List of Abbreviations

Abbreviation		Chapter of first occurrence
BLS	Base Level Space	1
DTI	Distance to Intersection	4
FVF	Functional Visual Field	5,9
LLMM	Lower Level Motor Module	7
LLPM	Lower Level Perception Module	9
LTM	Long Term Memory	1
MSIR	Minimal Scheme for Immediate Responses	1
PMS	Process Manager Space	2
PVF	Peripheral Visual Field	5,9
TRC	Traffic Research Centre	
TTC	Time to Collision	12
TTI	Time To Intersection.	4
UTC	Unified Theory of Cognition	1
SOAR I/O	Soar Output/Input Module	1,7,9
WM	Working Memory	1

References

- Aasman, J. (1986). Computersimulatie en modelleren van rijgedrag. In P.F. Lourens (Ed.), *Jaarverslag 1986 Verkeerskundig Studiecendum Rijksuniversiteit Groningen*.
- Aasman, J. (1988). Implementations of car driver behaviour and psychological risk models. In J. A. Rothengatter & R. A. de Bruin (Eds.), *Road user behaviour: Theory and research* (pp. 106-117). Assen, The Netherlands: Van Gorcum.
- Aasman, J., and Lourens, P. F. (1991). Timing of basic driver operations in the negotiation of general rule intersections. Unpublished manuscript, University of Groningen, Traffic Research Center, Haren, The Netherlands.
- Aasman, J. and Akyürek, A. (1992). Flattening goal hierarchies. In J.A. Michon & A. Akyürek (Eds.), *Soar, a cognitive architecture in perspective*. Dordrecht, The Netherlands: Kluwer.
- Aasman, J. and Michon, J.A. (1991). Soar as an environment for driver behavior modeling. In L.J.M. Mulder, F.J. Maarse, W.P.B. Sjouw, and A.E. Akkerman (Eds.), *Computers in Psychology: Applications in education, research, and psychodiagnostics* (pp.129-226). Amsterdam: Swets & Zeitlinger.
- Aasman, J. and Michon, J.A. (1992). Multitasking in driving. In J.A. Michon & A. Akyürek (Eds.), *Soar, a cognitive architecture in perspective*. (pp. 169-198). Dordrecht, The Netherlands: Kluwer.
- Ajzen, I., and Fishbein, M. (1977). Attitude-behaviour relations: a theoretical analysis and review of empirical research. *Psychological Bulletin*, 84, 888-918.
- Akyürek, A. (1991). Means-ends planning, operator subgoalng, and operator valuation: An example Soar program. In J.A. Michon & A. Akyürek (Eds.), *Soar, a cognitive architecture in perspective*. Dordrecht, The Netherlands: Kluwer.
- Akyürek, A. (1992). Means-ends planning: An example Soar system. In J. A. Michon & A.Akyürek (Eds.), *Soar: A cognitive architecture in perspective* (pp. 109-167). Dordrecht, The Netherlands: Kluwer.
- Akyürek, A. (forthcoming) Planning, acting, and the frame problem.

- Alm, H. (1990). Driver's cognitive models of routes. In W. van Winsum, H. Alm, J. Schraagen, and T. Rothengatter (Eds.), *Task reports on laboratory studies on route representation and navigation and on cognitive navigation models*. Report 1041/GIDS:NAV2 to the Commission of the European Community. Haren: Traffic Research Centre, University of Groningen.
- Anderson, J.R. (1983). *The architecture of cognition*. Cambridge, MA: Harvard University Press.
- Anderson, J.R. (1990). *The adaptive character of thought*. NJ: Lawrence Erlbaum Associates.
- Anderson, J.R. (1993). *Rules of the mind*. NJ: Lawrence Erlbaum Associates.
- Anderson, J.R., and Bower, G.H., (1973). *Human associative memory*. Washington: D.C. Winston.
- Bartmann, A., Spijkers, W., and Hess, M. (1991). Street environment, driving speed and field of vision. In A.G. Gale, A.G., Brown, I.D., Haslegrave, C.M., Morehead, I., and Taylor, S. (Eds). *Vision in Vehicles-III*, (pp 381-389). Elsevier Science Publishers B.V. North Holland.
- Behavioral and Brain Sciences*, (1992). 15, 425-492. Cambridge University Press.
- Bobrow, D. G., and Norman, D. A. (1975). Some principles of memory schemata. In D.G.Bobrow & A. Collins (Eds.), *Representation and understanding: Studies in cognitive science* (pp. 131-149). Orlando, FL: Academic Press.
- Boff, K.R., Kaufman, L., and Thomas, J.P. (1986). *Handbook of perception and human performance Volume I*. New York: Wiley.
- Boff, K.R., Kaufman, L., and Thomas, J.P. (1986). *Handbook of perception and human performance Volume II*. New York: Wiley.
- Carbonell, J.G., Knoblock, C.A., and Minton, S.N. (1989). The PRODIGY project: An integrated architecture for planning and learning. In K. VanLehn (Ed.), *Architectures for intelligence*. Hillsdale, NJ: Lawrence Erlbaum Associates.
- Card, S.K., Moran, T.P. and Newell, A. (1983). *The psychology of human-computer interaction*. Hillsdale, NJ: Lawrence Erlbaum Associates.
- Card, S.K., Moran, T.P. and Newell, A. (1986). The model human processor, an engineering model of human performance. In R.Boff, L. Kaufman and J.Thomas (Eds.), *Handbook of perception and human performance Volume I*. New York: Wiley.
- Chapman, D. (1990). *Vision, instruction and action*. (Tech. Rep. 1204) MIT.
- Cho, B., Rosenbloom, P.S., & Dolan, C.P. (1991). Neuro-Soar: A neural-network architecture for goal-oriented behaviour. In *Proceedings of the Thirteenth Annual Conference of the Cognitive Science Society* (pp. 673-677). Chicago: Lawrence Erlbaum Associates.
- Churchill, E.F., and Young, R.M. (1991). Modelling representations of device knowledge in Soar. In L. Steels and B. Smith (Eds.), *AISB-91 Artificial Intelligence and Simulation of Behaviour* (pp. 247-255). Berlin: Springer-Verlag.
- Covrigaru, A. (1989). The goals of autonomous systems. Unpublished manuscript, University of Michigan, Electrical Engineering and Computer Science Department, Ann Arbor.

- Covrigaru, A. (1990, October). Merging interrelated tasks. Paper presented at the Eighth Soar Workshop, Information Sciences Institute, University of Southern California, Los Angeles, CA.
- Cremer, R., Snel, J. and Brouwer, W.H. (1990). Age-related differences in timing of position and velocity identification. *Accident Analysis and Prevention*, 22, 467-474.
- De Kleer, J. (1993). A perspective on assumption-based truth maintenance. *Artificial Intelligence*, 59, 63-67.
- De Velde Harsenhorst, J. J., and Lourens, P. L. (1987). *Classificatie van rijtaakfouten en analyse van rijtaakverrichtingsparameters* [Classification of driving errors and analysis of driving performance parameters]. (Tech. Rep. VK 87-17) Haren, The Netherlands: University of Groningen, Traffic Research Center.
- De Velde Harsenhorst, J.J., and Lourens, P.F. (1988). *Het onderwijs leerproces bij een leerlingautomobiliste en specifieke rijgedrag van jonge automobilisten*. [The educational learning process of a novice driver and specific driving behavior of young car drivers]. (Tech. Rep. VK 87-25). Haren, The Netherlands: University of Groningen, Traffic Research Center.
- Denet, D.C. (1978). *Brainstorms: philosophical essays on mind and psychology*. Hassocks, Sussex: Harvester Press.
- Denet, D.C. (1987). *The intentional stance*. Cambridge, MA: MIT Press.
- Denton, G.G. (1976). The influence of adaptation of subjective velocity for an observer in simulated rectilinear motion. *Ergonomics*, 19, 409-430.
- Doyle, J. (1979). A truth maintenance system. *Artificial Intelligence*, 12, 231 - 272.
- Drury, C. (1975). Application of Fitts's Law to foot-pedal design. *Human Factors*, 17, 368-373.
- Elgot-Drapkin, J., Miller, M., & Perlis, D. (1987). Life on a desert island: Ongoing work onreal-time reasoning. In F. M. Brown (Ed.), *The frame problem in artificial intelligence*(pp. 349-357). Los Altos, CA: Morgan Kaufmann.
- Ellis, R.E., and Smith, J.D. (1985). Patterns of statistical dependency in visual scanning. In R. Groner, G.W. McConckie and C. Menz (Eds.), *Eye movements and human informatin Processing*. Amsterdam: Elseiver Science Publishers.
- Ericsson, K.A., and Simon, H.A. (1993). *Protocol analysis, verbal reports as data*. Cambridge, MA: Bradford.
- Erikson, W.E., and Yei-Yu Yeh. (1986). Allocation of attention in the visual field. *Journal of experimental psychology: Human perception and performance*, 11, 583-597.
- Evans, G. (1980). Environmental cognition. *Psychological Bulletin*, 88, 259-287.
- Färber, B., Färber, B., and Popp, M.M. (1986). Are oriented drivers better drivers? Fifth International Congress ATEC 86, Paris. [Incompleet]
- Fitts, P.M. (1954). The information capacity of the human motor system in controlling the amplitude of movement. *Journal of Experimental Psychology*, 47, 381-391.

- Fleishman, E.A. (1967). Performance assessment based on empirically derived task taxonomy. *Human Factors*, 9, 349-366.
- Frijda, N.H. (1986). *The emotions*. CA: Cambridge University Press.
- Fuller, R. (1984). A conceptualization of driver behaviour as threat avoidance. *Ergonomics*, 27, 1139-1155.
- Gale, A.G., Brown, I.D., Haslegrave, C.M., Morehead, I., Taylor, S. (1991). *Vision in vehicles-III*. Elsevier Science Publishers B.V. North Holland.
- Gale, A.G., Freeman, M.H., Haslegrave, C.M., Smith, P., and Taylor, S.P. (1986). *Vision in vehicles*. Proceedings of the Conference on Vision in Vehicles. Elsevier Science Publishers B.V. North Holland.
- Gale, A.G., Freeman, M.H., Haslegrave, C.M., Smith, P., and Taylor, S.P. (1988). *Vision in vehicles-II*. Proceedings of Second International Conference on Vision in Vehicles. Elsevier Science Publishers B.V. North Holland.
- Gärling, T., and Golledge, R.G. (1989). Environmental perception and cognition. In E.H. Zube and G.T. Moore (Eds.), *Advances in Environment, Behaviour, and Design* (Vol. 2, pp. 203-236).
- Georgeff, M.P., and Lansky, A.L. (1987). Reactive reasoning and planning. In *Proceedings of the Sixth National Conference on Artificial Intelligence* (pp 677-682). Los Altos, CA: Morgan Kaufman Publishers.
- Godthelp, J. (1984). *Studies on human vehicular control*. Thesis. Soesterberg: Instituut voor Zintuigfysiologie TNO.
- Golding, A., Rosenbloom, P.S., and Laird, J.E. (1987). Learning general search control from outside guidance. In John McDermott (Ed.), *Proceedings of IJCAI-87* (pp. 334-337). , Los Altos, CA: Morgan Kaufman Publishers.
- Gordon, P.C. and Meyer, D.E. (1987). Control of serial order in rapidly spoken syllable sequences. *Journal of Memory and Language*, 26, 300-321.
- Groeger, J., Kuiken, M., Grande, G., Miltenburg, P., Brown, I., and Rothengatter, T. (1990). *Preliminary design specifications for appropriate feedback provision to drivers with differing levels of traffic experience*. Drive Project V1041, Traffic Research Centre, University of Groningen.
- Groner, R. (1988). Eye movements, attention and visual information processing: some experimental results and methodological consideration. In G. Lüer, U. Lass & J. Shallo-Hoffmann (Eds.), *Eye movement research*. Toronto: Hogrefe.
- Groner, R., and Menz, C. (1985). The effect of stimulus characteristics, task requirements and individual differences on scanning patterns. In R. Groner, G.W. McConkie and C. Menz (Eds.), *Eye movements and Human Information Processing*. Elsevier Science Publishers B.V.
- Groner, R., Walder, F., and Groner, M. (1984). Looking at faces: local and global aspects of scanpaths. In A.G. Gale and F. Johnson (Eds.), *Theoretical and applied aspects of scanpaths*. Amsterdam: Elsevier.

- Halasz, F.G., and Moran, T.P. (1983). Mental models and problem solving in using a calculator. *Proceedings of ACM SIGCHI*. Boston, M.A.
- Hale, A. R., Stoop, J., and Hommels, J. (1990). Human error models as predictors of accident scenarios for designers in road transport systems. *Ergonomics*, 33, 1377-1387.
- Hasher, L., and Zacks, R.T. (1984). Automatic processing of fundamental information. *American Psychologist*, 39, 1372-1388.
- Hayes-Roth, B. (1984). *BB1: An architecture for blackboard systems that control, explain, and learn about their own behaviour*. (Tech. Rep. HPP 84-16). Knowledge Systems Laboratory, Stanford University.
- Hendrickx, L.C.W.P. (1991). *How versus how often*. Thesis, Rijksuniversiteit Groningen.
- Hills, B.L. (1980). Vision, visibility, and perception in driving. *Perception*, 9, 183-216.
- Horst, A.R.A. van der (1990). *A time-based analysis of road user behaviour in normal and critical encounters*. Dissertation. TNO Institute for Perception, Soesterberg, The Netherlands.
- Hucka, M. (1989). *Planning, interruptibility, and learning in Soar*. Unpublished manuscript. University of Michigan, Electrical Engineering and Computer Science Department, Ann Arbor.
- Hucka, M. (1991). *Interruption and resumption in integrated intelligent agents*. Unpublished Manuscript. University of Michigan, Electrical Engineering and Computer Science Department, Ann Arbor.
- Huffman, S. B. (1994). *Instructable autonomous agents*. Report CSE-TR-193-94. PhD thesis, Computer Science & Engineering Division, University of Michigan at Ann Arbor.
- Howes, A. (1994). A model of acquisition of menu knowledge by exploration. In B. Adelson, S. Dumais, J. Olson (Eds.), *Proceedings of Human Factors in Computing Systems CHI '94* (pp. 445-451). Boston: MA, ACM Press.
- Jackendoff, R.S. (1987). *Consciousness and the computational mind*. Cambridge, Mass.: MIT Press, Bradford Books.
- Jackson, J.L., Akyürek, A., and Michon, J.A. (1993). Symbolic and other cognitive models of temporal reality. *Time & Society*, 2, 241-256.
- Janssen, W.H. (1985). Het beoordelen van dwarsverkeer tijdens het naderen in een boog. In *Cahier 4, Verkeerskunde*. Published by ANWB, the Netherlands.
- Janssen, W.H. (1984). *The detection of impending collision in curved intersection approaches*. Rapport IZF 1984 C-3. Instituut voor Zintuigfysiologie TNO, Soesterberg.
- Janssen, W.H., Michon, J.A., and Harvey, L.O. (1976). The perception of lead vehicle movement in darkness. *Accident Analysis and Prevention*, 8, 151-166.
- Jesserun, M., Steyvers, F.J.J.M., de Waard, D., Dekker, K., and Brookhuis, K.A. (1990). *Beleving, waarneming en activatie tijdens het rijden over een deel van de A2*. [Subjective perception, visual perception and activation during driving on a part of the A2 highway] (Tech. Rep. VK 90-18). Haren, The Netherlands: University of Groningen, Traffic Research Center.

John, B.E. (1988). *Contributions to engineering models of human-computer interaction*. Unpublished doctoral dissertation, Carnegie Mellon University, Pittsburgh, Pennsylvania.

Johnson-Laird, P. (1983). *Mental Models*. Cambridge, MA: Harvard University Press.

Jordan, M.I., and Rosenbaum, D.A. (1989). Action. In M.I. Posner (Ed.), *Foundations of cognitive science*. Cambridge: M.I.T. Press.

Justy, M.A., and Carpenter, P.A. (1987). *The Psychology of Reading and Language Comprehension*. Boston: Allyn and Bacon.

Keele, S.W. (1987). Motor Control. In K.R. Boff, L. Kaufmann & J.P. Thomas (Eds.), *Handbook of perception and human performance*. New York: Wiley.

Kidd, E.A., and Laughery, K.R. (1964). *A computer model of driving behavior: The highway intersection situation*. Buffalo, NY: Cornell Aeronautical laboratories. Report nr. VI-1843-V-1, 1964.

Kieras, B., and Bovair, S. (1984). The role of a mental model in learning to operate a device. *Cognitive Science*, 8, 255-273.

King, G.F., and Lunenfeld, H. (1974). Urban guidance: perceived needs and problems. *Transportation Research Record*, 503, 25-37.

Klebsberg, D. (1971). Subjective und objektive Sicherheit in Strassenverkehr als Aufgabe für die Verkehrssicherheitsarbeit. *Schriftenreihe der Deutschen Verkehrswacht*, 51, 3-12.

Krammer, T. (1990). Modellen voor de cognitie van motorcontrole. Scriptie, Traffic Research Centre, The Netherlands.

Kuipers, B.J. (1978). Modelling spatial knowledge. *Cognitive Science*, 2, 129-153.

Kuipers, B.J. (1982). The "map in the head" metaphor. *Environment and Behaviour*, 14, 202-220.

Kuipers, B.J. and Levitt, T.S. (1988). Navigation and mapping in large-scale space. *AI Magazine*, 9 (2), 25-43.

Kuokka, D. R. (1990). *The deliberative integration of planning, execution, and learning*. (Tech. Rep. CMU-CS-90-135). Pittsburgh, PA: Carnegie Mellon University, School of Computer Science.

Kuokka, D.R. (1988). Integrating planning, execution, and learning. Thesis proposal. Carnegie Mellon University

Labiale, G. (1988). Survey on driver behaviour. In J.A. Rothengatter & R.A. de Bruin (Eds.), *Road user behaviour: Theory and research* (pp 249-259). Assen, The Netherlands: Van Gorcum.

Laird, J. E. (1986). Universal subgoalting. In J. Laird, P. Rosenbloom, and A. Newell. *Universal subgoalting and chunking: The automatic generation and learning of goal hierarchies* (pp. 1-131). Boston, MA: Kluwer.

- Laird, J.E. (1988). Recovery from incorrect knowledge in Soar. In Reid G. Smith (Ed.), *Proceedings of the AAAI-88* (pp. 618-623). San Mateo, CA: Morgan Kaufmann Publishers
- Laird, J.E. (1989, May). Why we don't need P in GPSO. Paper presented at the Sixth Soar Workshop, University of Michigan, Ann Arbor, MI.
- Laird, J.E., Rosenbloom, P.S., and Newell, A. (1986). *Universal subgoalting and chunking: The automatic generation and learning of goal hierarchies*. Boston, MA: Kluwer.
- Laird, J.E. (1990). Integrating planning and execution in Soar. In *Proceedings of the 1990 AAAI Spring Symposium on Planning in Uncertain, Unpredictable, and Changing Environments*.
- Laird, J. E., and Rosenbloom, P. S. (1990). Integrating execution, planning, and learning in Soar for external environments. In *Proceedings of the Eighth National Conference on Artificial Intelligence* (pp. 1022-1029). San Mateo, CA: Morgan Kaufmann.
- Laird, J. E., Congdon, C. B., Altmann, E., and Swedlow, K. (1990). *Soar User's Manual: Version 5.2*. (Tech. Rep. CMU-CS-90-179). Pittsburgh, PA: Carnegie Mellon University, School of Computer Science.
- Laird, J.E., Hucka, M., Yager, E.S., and Tuck, C.M. (1990a). Correcting and extending domain knowledge using outside guidance. Unpublished manuscript.
- Laird, J.E., Hucka, M., Yager, E.S., and Tuck, C.M. (1990b). Robo-Soar: An integration of external interaction, planning, and learning using Soar. In W. van der Velde (Ed.), *Machine learning for autonomous agents*.
- Laird, J. E., Newell, A., and Rosenbloom, P. S. (1987). SOAR: An architecture for general intelligence. *Artificial Intelligence*, 33, 1-64.
- Laird, J., and Newell, A. (1983). *A universal weak method* (Tech. Rep. CMU-CS-83-141). Pittsburgh, PA: Carnegie Mellon University, Department of Computer Science.
- Lashley, K.S. (1951). The problem of serial order in behaviour. In L.A. Jeffress (Eds.), *Cerebral Mechanisms in Behaviour*. New York: Wiley.
- Lee, D.N. (1976). A theory of visual control of braking based on information about time-to-collision. *Perception*, 5, 437-459.
- Lehman, J.F., Lewis, R.L., and Newell, A. (1991). *Natural language comprehension in Soar: Spring 1991*. (Tech. Rep. CMU-CS-91-17). Carnegie Mellon University, Pittsburgh PA.
- Levelt, W.J.M. (1989). De connectionistische mode. In C. Brown, P. Hagoort, and T. Meijering (Eds.), *Vensters op de geest* (pp. 202-251).
- Lewis, R.L., Newell, A., and Polk, T.A. (1989). Towards a Soar theory of taking instructions for immediate reasoning tasks. In *Proceedings of the Eleventh Annual Conference of the Cognitive Science Society* (pp. 514-521) Hillsdale, NJ: Lawrence Erlbaum Associates.
- Lewis, R. L., Huffman, S. B., John, B. E., Laird, J. E., Lehman, J. F., Newell, A., Rosenbloom, P. S., Simon, T., and Tessler, S. G. (1990). Soar as a unified theory of cognition: Spring 1990. In *Proceedings of the Twelfth Annual Conference of the Cognitive Science Society* (pp. 1035-1042). Hillsdale, NJ: Erlbaum.

Lewis, C.F., Blakely, W.R., Swaroop, R., Masters, R.L., and McMurty, T.C. (1973). Landing performance by low-time private pilots after the sudden loss off binocular vision. *Aerospace Medicine*, 44, 1241-1245.

Lourens, P.F., and van der Molen, H.H. (1986). *De psychogenese van incorrect rijgedrag*. [The psychogenesis of incorrect driver behaviour]. (Tech. Rep. VK 86-12). Haren, The Netherlands: University of Groningen, Traffic Research Center.

Lynch, K. (1960). *The image of the city*. Cambridge, MA: MIT Press.

MacKay, D.G. (1982). The problem of flexibility, fluency, and speed-accuracy trade-off in skilled behaviour. *Psychological review*, 89, 483-506.

Marr, D. (1982). *Vision*. San Francisco: Freeman.

McCarthy, J., and Hayes, P.J. (1969). Some philosophical problems from the standpoint of artificial intelligence. *Machine Intelligence*, 4, 463-502.

MacDonald, W.A. (1977). *The measurement of driving task demand*. Thesis, University of Melbourne.

McKnight, A.J., and Adams, B.B. (1970a). *Driver education tasks analysis. Volume I: Task descriptions*. Alexandria, VA: Human Resources Research Organization, Final Report, Contract No. FH 11-7336.

McKnight, A.J., and Adams, B.B. (1970b). *Driver education tasks analysis. Volume II: Task analysis methods*. Alexandria, VA: Human Resources Research Organization, Final Report, Contract No. FH 11-7336.

Michon, J.A. (1971). *Psychonomie onderweg* (Inaugural Lecture). Groningen: Wolters-Noordhof.

Michon, J. A. (1976). The mutual impacts of transportation and human behaviour. In P. Stringer and H. Wenzel (Eds.), *Transportation planning for a better environment* (pp. 221-235). New York: Plenum Press.

Michon, J.A. (1979). Dealing with danger: Report of the European Commission MRC Workshop on Physiological and psychological factors in performance under hazardous conditions, Gieten, The Netherlands, 23-25 may, 1978. Haren (The Netherlands): Traffic Research Center, University of Groningen, Report, VK 79-01, 1979.

Michon, J. A. (1985). A critical review of driver behaviour models: What do we know, what should we do? In L. A. Evans & R. C. Schwing (Eds.), *Human behaviour and traffic safety* (pp. 487-525). New York: Plenum Press.

Michon, J.A. (1989). Explanatory pitfalls and rule-based driver models. *Accident Analysis and Prevention*, 21, 341-353.

Michon, J.A. (1993). The seven pillars of time psychology. *Psychologica Belgica*, 33, 329-345.

Michon, J.A. (1993). *Generic Intelligent Driver Support*. London: Taylor & Francis.

- Michon, J.A. and Akyürek, (1993). *Soar, a cognitive architecture in perspective*. Dordrecht, The Netherlands: Kluwer.
- Michon, J.A., Smiley, A. and Aasman, J. (1990). Errors and driver support systems. *Ergonomics*, 33, 1215-1229.
- Miller, G.A. (1956). The magical number seven, plus or minus two: Some limits on our capacity for processing information. *Psychological Review*, 63, 81-97.
- Minton, S. (1988). *Learning search control knowledge: an explanation-based approach*. Boston, MA: Kluwer.
- Mitchell, T.M., Allen, J., Chalasani, P., Cheng, J. Etzioni, O., Ringuette, M., and Schlimmer, J.C. (1989). Theo: a framework for selfimproving systems. In K. VanLehn (Ed.), *Architectures for Intelligence*. Hillsdale, NJ: Lawrence Erlbaum Associates.
- Miura, T. (1986). Coping with situational demands: A study of Eye Movements and Peripheral Vision Performance. In A.G Gale et al. (Eds.), *Vision in Vehicles*. Elsevier Science Publishers B.V. North Holland.
- Molen, H. H. Van der (1984). *Pedestrian ethology: unobtrusive observations of child and adult road crossing behaviour in the framework of the developmnet of a child pedestrian training program*. Thesis, University of Groningen.
- Molen, H.H. Van der, and Böttcher, A.M.T. (1988). A hierarchical risk model for traffic participants. *Ergonomics*, 31 (4), 537-555.
- Moraal, J. (1980). De studie van verkeersgedrag. *Ergonomie*, 5, 1-8.
- Mourant, R.R., Rockwell, T.H. (1970). Mapping eye-movement patterns to the visual scene in driving: an exploratory study. *Human Factors*, 12, 81-87.
- Mourant, R.R. and Rockwell, T.H. (1972). Strategies of visual search by novice and experienced drivers. *Human Factors*, 14, 325-335.
- Näätänen, R. and Summala, H. (1974). A model for the role of motivational factors in drivers' decision making. *Accident Analysis and Prevention*, 6, 243-261.
- Näätänen, R., and Summala, H. (1976), *Road user behaviour and traffic accidents*. Amsterdam: North Holland Publishing.
- Newell, A. (1977). Reflections on obtaining science through building systems. In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence* (pp. 970-971). MIT.
- Newell, A. (1988, September). The basic quantitative code. Paper presented at the Fifth Soar Workshop, Carnegie Mellon University, Pittsburgh, PA.
- Newell, A. (1990). *Unified theories of cognition*. Cambridge, MA: Harvard University Press.
- Newell, A., and Rosenbloom, P.S. (1981). Mechanisms of skill acquisition and the law of practice. In J.R. Anderson (Ed.), *Cognitive Skills and their Acquisition*. Hillsdale, NJ: Erlbaum.

- Newell, A., and Simon, H. A. (1972). *Human problem solving*. Englewood Cliffs, NJ: Prentice-Hall.
- Newell, A., Rosenbloom, P. S., and Laird, J. E. (1989). Symbolic architectures for cognition. In M. I. Posner (Ed.), *Foundations of cognitive science*, (pp. 93-131). Cambridge, MA: MIT Press.
- Newell, A., Shaw, J.C., and Simon, H.A. (1962). The processes of creative thinking. In H.A. Simon, *Models of thought*. New Haven, Yale University Press.
- Newell, A., Yost, G.R., Laird, J.E., Rosenbloom, P.S., and Altmann, E. (1990). Formulating the problem space computational model. In R.F. Rashid (Ed.), *CMU computer science: A 25th Anniversary Commemorative* (pp. 255-293). New York: ACM Press.
- Nilsson, N. J. (1971). *Problem-solving methods in artificial intelligence*. New York: McGraw-Hill.
- Nodine, C.F., Kundel, H.L. (1987). The cognitive side of visual search in radiology. In J.K. O'Regan and A.Levy-Schoen (Eds.), *Eye movements: from physiology to cognition*. Amsterdam: Elsevier science publishers.
- Nooteboom, S.G. (1985). A functional view of prosodic timing in speech. In J.A. Michon and J.L. Jackson (Eds.), *Time, mind and behaviour*. Berlin: Springer-Verlag.
- Norman, D. A., and Bobrow, D. G. (1975). On data limited and resource limited processes. *Cognitive Psychology*, 7, 44-64.
- Pailhous, J. (1970). *La représentation de l'espace urbain* [The representation of urban space]. Paris: Presses Universitaires de France.
- Polk, T.A., and Newell, A. (1988). Modeling human syllogistic reasoning in Soar. In *Proceedings of Tenth Annual Conference of the Cognitive Science Society* (pp. 181-187). Hillsdale, NJ: Erlbaum.
- Posner, M.I. (1980). Orienting of attention. *Quarterly Journal of Experimental Psychology*, 32, 3-25.
- Posner, M.I. (1989). *Foundations of Cognitive Science*. Cambridge, MA: MIT Press.
- Pylyshyn, Z.W. (1984). *Computation and cognition: towards a foundation for cognitive science*. Cambridge, Mass.: MIT Press, Bradford Books.
- Quinlan, P.T. (1991). *Connectionism and psychology: a psychological perspective on new connectionist research*. New York: Harvester.
- Rasmussen, J. (1985). Trends in human reliability analysis. *Ergonomics*, 28, 1185-1195.
- Rasmussen, J. (1987). The definition of human error and a taxonomy for technical system design. In J. Rasmussen, K. Duncan, and J. Leplat (Eds.), *New technology and human error* (pp. 23-31). New York: Wiley.
- Reason, J. (1987). Generic error-modelling system (GEMS): A cognitive framework for locating common human error forms. In J. Rasmussen, K. Duncan, & J. Leplat (Eds.), *New technology and human error* (pp. 63-85). New York: Wiley.

- Reece, D.A. (1992). *Selective perception for robot driving*. (Tech. Rep. CMU-CS-92-139). Carnegie Mellon University, Pittsburgh.
- Reece, D., and Shafer, S. (1988). An overview of the Pharos traffic simulator. In J. A. Rothengatter and R. A. de Bruin (Eds.), *Road user behaviour: Theory and research* (pp. 285-293). Assen, The Netherlands: Van Gorcum.
- Regan, D.M., Kaufman, L., and Lincoln, J. (1986). Motion in depth and visual acceleration. In R. Boff, L. Kaufman and J. Thomas (Eds.), *Handbook of perception and human performance*, Volume I. New York: Wiley.
- Reid, L.D., and Solowka, E.N. (1981). A systematic study of driver steering behaviour. *Ergonomics*, 24, 447-462.
- Reitman, W. R. (1965). *Cognition and thought: An information-processing approach*. New York: Wiley.
- Riemersma, J.B.J. (1987). *Visual cues in straight road driving*. Thesis, University of Groningen, The Netherlands.
- Rich, E. (1983). *Artificial intelligence*. New York: McGraw-Hill.
- Rosenbaum, D.A. (1975). Perception and extrapolation of velocity and acceleration. *Journal of experimental psychology: Human perception and Performance*, 1, 305-403.
- Rosenbaum, D.A., Kenny, S., and Derr, M.A. (1983). Hierarchical control of rapid movement sequences. *Journal of Experimental Psychology: Human Perception and Performance*. 9, 86-102.
- Rosenbloom, P.S. (1989). A symbolic goal-oriented perspective on connectionism and Soar. In R. Pfeifer, Z. Schreter, F. Fogelman-Soulie, & L. Steels (Eds.), *Connectionism in perspective* (pp 245-263). Elsevier (North-Holland).
- Rosenbloom, P., and Aasman, J. (1992). Knowledge level and inductive uses of chunking. In J.A. Michon and A. Akyürek (Eds.), *Soar, a cognitive architecture in perspective*. Dordrecht. The Netherlands: Kluwer.
- Rosenbloom, P. S., Laird, J. E., Newell, A., and McCarl, R. (1991). A preliminary analysis of the Soar architecture as a basis for general intelligence. *Artificial Intelligence*, 47, 289-325
- Rosenbloom, P.S., Newell, A., and Laird, J.E. (1989). Towards the knowledge level in Soar: the role of the architecture in the use of knowledge. In VanLehn, K. (Ed.), *Architectures for Intelligence*. Hillsdale, NJ: Lawrence Erlbaum Associates.
- Rumelhart, D.E., and Norman, D.A. (1982). Simulating a skilled typist: A study of skilled cognitive-motor performance. *Cognitive science*, 6, 1-36.
- Runeson, S. Visual prediction of collision with natural and nonnatural motion funtions. *Perception and Psychophysics*, 18, 261-266.
- Russo, J.E. (1978). Adaptation of cognitive processes to the eye-movement system. In J. W. Senders, D.F. Fisher, and R.A. Monty (Eds.), *Eye movements and the higher psychological functions*. Hillsdale, NJ: Erlbaum.

Sabey, B.E., and Staughton, G.C. (1975). Interacting roles of road environment, vehicle and road user in accidents. 5th International Conference of the International Association for Accident and Traffic Medicine. London.

Sanders, A.F. (1963). *The selective process in the functional visual field*. Assen, The Netherlands: van Gorcum.

Schmidt, R.A. (1975). A schema theory of discrete motor skill learning. *Psychological review*, 82, 225-260.

Schmidt, R.A. (1980). On the theoretical status of time in motor program representations. In G.E. Stelmach and J. Requin (Eds.), *Tutorials in motor behaviour*. North Holland Publishing Company.

Schmidt, R.A. (1982). *Motor control and learning*. Champaign, IL: Human Kinetics Publishers.

Schraagen, J.M.C. (1990). Use of different types of map information for route following in unfamiliar cities. In W. van Winsum, H. Alm, J. Schraagen, and T. Rothengatter. *Task reports on laboratory studies on route representation and navigation and on cognitive navigation models*. Report 1041/GIDS:NAV2 to the Commission of the European Community. Haren: Traffic Research Centre, University of Groningen.

Scialfa, C.T., Lawrence, T.G., Leibowitz, H.W., Garvey, P.M., and Tyrrell, R.A. (1991). Age differences in estimating vehicle velocity. *Psychology and Aging*, 6, 60-66.

Shaffer, L.H. (1985). Timing in the motor programming of typing. *Quarterly Journal of Experimental Psychology*, 30, 333-345.

Shiffrin, R.M., and Schneider, W. (1977). Controlled and automatic human information processing: II. Perceptual learning, automatic attending, and a general theory. *Psychological Review*, 84, 127-190.

Shoham, Y. (1987). What is the Frame Problem. In M.P. Georgeff and A.L. Lansky (Eds.), *Proceedings of the 1986 Workshop on Reasoning about Actions and Plans* (pp. 83-98). Los Altos, CA: Morgan Kaufmann Publishers.

Simon, H.A. (1974). How big is a chunk? *Science*, 183, 482-488

Simon, H.A., and Kaplan, C.A. (1989). Foundations of cognitive science. In M.I. Posner (Ed.), *Foundations of Cognitive Science*. Cambridge, MA: MIT Press.

Smiley, A. (1989). Cognitieve vaardigheden van autobestuurders. (cognitive abilities of drivers). In C.W.F. van Knippenberg, J.A. Rothengatter & J.A. Michon (eds.), *Handboek sociale verkeerskunde*. Assen: Van Gorcum.

Staughton, G.C., and Storie, V.J. (1977). *Methodology of an in-depth accident investigation survey report*. (Tech. Rep. LR762), Department of the Environment/Department of Transport, TRRL, Crowthorne, Berks.

Steier, D. M., Laird, J. E., Newell, A., Rosenbloom, P. S., Flynn, R. A., Golding, A., Polk, T. A., Shivers, O. G., Unruh, A., and Yost, G. R. (1987). Varieties of learning in Soar: 1987. In *Proceedings of the Fourth International Workshop on Machine Learning* (pp. 300-311). San Mateo, CA: Morgan Kaufmann Publishers.

- Stevens, S.S. (1975). *Psychophysics: Introduction to its perceptual, neural, and social prospects*. New York: Wiley.
- Storie, V.J. (1977). *Male and female car drivers: differences observed in accidents*. (Tech. Rep. Report LR761). Department of the Environment/Department of Transport, TRRL, Crowthorne, Berks.
- Suchman, L.A. (1988). *Plans and situated action: the problem of human machine communication*. New York: Cambridge University Press.
- Tambe, M., and Newell, A. (1988). Some chunks are expensive. In *Proceedings of the Fifth International Workshop on Machine Learning* (pp. 451-458). San Mateo, CA: Morgan Kaufmann.
- Tambe, M., and Rosenbloom, P. S. (1989). Eliminating expensive chunks by restricting expressiveness. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence* (pp. 731-737) San Mateo, CA: Morgan Kaufmann.
- Taylor, D.H. (1964). Driver's galvanic skin response and the risk of accident. *Ergonomics* 7, 439-451.
- Theeuwes, J. (1989). *Conspicuity is task dependent; Evidence from selective search*. Report IZF 1989, C-8 Soesterberg: Institute for Perception, The Netherlands.
- Treisman, A., and Gelade, G. (1980) A feature integration theory of attention. *Cognitive Psychology*, 12, 97-136.
- Tulving, E. (1969). Episodic and semantic memory. In E. Tulving and W. Donaldson (eds.), *Organization of Memory*. New York: Academic press.
- Van Winsum, W., Alm, H., Schraagen, J.M.C., and Rothengatter, T. (1990). *Task reports on laboratory studies on route representation and navigation and on cognitive navigation models*. Report 1041/GIDS:NAV2 to the Commission of the European Community. Haren: Traffic Research Centre, University of Groningen.
- Van der Heijden, A.H.C., and Joutstra, A.J. (1992). Selectieve aandacht in de visuele waarneming. *Nederlands tijdschrift voor de psychologie*, 47, 49-72.
- Van Berkum, J. A. (1988). *Cognitive modelling in Soar* (WR 88-01). Haren, The Netherlands: University of Groningen, Traffic Research Center.
- Wagenaar, W.A., and Reason, J.T. (1990). Types and tokens in road accident causation. *Ergonomics*, 33, 1365-1375.
- Van Winsum, W. (1987). De mentale belasting van het navigeren in het verkeer. [The mental load of navigation in driving]. (Tech. Rep. VK 87-30.) Haren: Traffic Research Centre, University of Groningen.
- Van Winsum, W., and Wolffelaar, P.C. (1993). GIDS Small World Simulation. In J.A. Michon (Ed.), *Generic Intelligent Driver Support* (pp 175-191). London: Taylor & Francis.

Van Wolfelaar, P.C., Rothengatter, T., and Brouwer, W. Elderly drivers' traffic merging decisions. In A.G. Gale, A.G., Brown, I.D., Haslegrave, C.M., Morehead, I., and Taylor, S. (Eds). *Vision in Vehicles-III*, (pp 247-256). Elsevier Science Publishers B.V. North Holland.

Wickelgren, W.A. (1969). Context-sensitive coding, associate memory, and serial order in (speech) behaviour. *Psychological Review*, 76, 1-15.

Wickens, C.D. (1986). The effects on control dynamics and performance. In R.Boff, L. Kaufman and J.Thomas (Eds.), *Handbook of Perception and Human Performance Volume I*. New York: Wiley.

Wierda, M., and Aasman J. (1988). Expertsystemen en computers in verkeersopvoeding in het voortgezet onderwijs [Expert systems and computers in traffic education in secondary schools]. (Tech. Rep. VK 88-24). Haren, The Netherlands: University of Groningen, Traffic Research Center.

Wierda, M., and Aasman, J. (1991). Seeing and driving: computation, algorithms and implementation. (Tech. Rep. VK-91-06). Haren, The Netherlands: University of Groningen, Traffic Research Center.

Wierda, M., and Maring, W. (to appear). Interpreting eye movements of traffic participants. In D. Brogan (Ed.), *Proceedings of the second international conference on eye movements*. London: Taylor and Francis.

Wierda, M., Brookhuis, K.A., and Van Schagen, I.N.L.G. (1987). *Elementaire fietsvaardigheden en mentale belasting; empirisch onderzoek*. [elementary bicycle riding skills and mental load; empirical research]. (Tech. Rep. VK-87-08). Haren, The Netherlands: University of Groningen, Traffic Research Center.

Wierda, M., Van Schagen, I.N.L.G., and Brookhuis, K.A. (1990). *Waarnemingsstrategieën van fietsers*. [Visual orientation strategies in bicyclists]. (Tech. Rep. VK-90-13). Haren, The Netherlands: University of Groningen, Traffic Research Center.

Wiesmeyer, M.D. (1992). *An operator-based model of human covert visual attention*. Thesis University of Michigan CSE-TR-123-92.

Wiesmeyer, M., and Laird, J. (1990). A computer model of 2D visual attention. In *Proceedings of the Twelfth Annual Conference of the Cognitive Science Society* (pp. 582-589). Hillsdale, NJ: Erlbaum.

Wilde, G.J.S. (1982a). Critical issues in risk homeostasis: implications for safety and health. *Risk Analysis*, 2, 209-225.

Wolf, J.D., and Barrett, M.F. (1978a). *Driver-vehicle effectiveness model*; volume I: Final report. Washington, DC: Department of Transportation. Report No DOT HS 804 337.

Zwahlen, H. T. (1992). Eye scanning rules for drivers; how do they compare with actual observed eye scanning behaviour. *Proceedings of the strategic highway research program and traffic safety on two continents, Gothenburg, 1991*, Linköping: VTI.

Appendix 1: Learning and error correction for external operators

The goal of this appendix is to demonstrate a simplified model of learning from external interaction and error correction for evaluations. What is shown in this appendix is (1) how Soar learns via selective data-chunking the evaluation of an external operator and (2) how Soar can perform error correction by learning another evaluation for the same external operator and in the process unlearn the older evaluation.

The basic principles of learning and error correction for external operators were explained in the concluding chapter of this study. This appendix provides a far more detailed treatment for Soar aficionados. First, a few runs demonstrating the general behaviour of Soar in external learning mode are discussed. Second, the Soar productions that generate this behaviour are presented.

The task

Before we get into the runs and the productions, we need to explain the task. The task in this example has been kept very simple. On the state resides an object `^model <m>`, which stands for a representation of the external world. In this case it is the situation that DRIVER's distance-to-intersection (dti) is 10 metres and its intended manoeuvre is a right turn.

```
(state <s> ^model <m>)  
(model <m> ^dti 10 ^manoeuvre turn-right)
```

There are two operators, one for looking to the right, one for looking to the left.

```
(operator <O1> ^name external-op ^look right)  
(operator <O2> ^name external-op ^look left)
```

The task is to learn whether to look first right or first left. The application of an operator leads via external interaction to the addition of the `^look <x>` annotation on the model. After the application of the operator an evaluation is

immediately given. +3 means good, -3 means not-so-good. Soar must learn to learn these evaluations and remember them next time.

The runs and the productions

In the first run, on the following page, two external operators are available and one is selected at random and applied to the external world. The operator receives a positive evaluation from the external world (via the Soar input) at the top level. The positive evaluation is learned via data-chunking. In the second run the same operator is chosen again, but now because it has a very positive evaluation. However, this time it receives a very negative evaluation. This negative evaluation is also data-chunked and in addition the previously learned positive chunk is unlearned by making the earlier evaluation *invalid*. Finally, in the third run we see that the other external operator gets a chance of being applied.

The productions in the second half of this appendix are ordered according to their activity in the trace. I would like to advise the reader to study the run together with the productions that generated this behaviour.

A.1 Specimen runs

A.1.1 First run

0	G: G1	
1	P: P2 (BASE-LEVEL)	
2	S: S3 (IO-STATE)	
3	O: O5 (ADD-FIRST)	This operator adds ^model <m> as described above to S3.
4	O: X7 (EXTERNAL-OP) Look to the right and add the annotation ^look right to M.	Two operators are available and this one is chosen at random. The application of the operator leads, via simulated external interaction, to the addition of ^look right to model <m>.
5	O: O10 (GET-EVALUATION) evaluation: 3	Soar is in a so-called <i>evaluate external operators</i> mode and asks the outside world for an evaluation. The user answers with 3, a positive evaluation.
6	O: O13 (LEARN-EVALUATION)	As Soar is in the <i>evaluate external operators</i> mode, data-chunking of an evaluation chunk begins.
7	--> G: G16 (OPERATOR NO-CHANGE)	
8	P: P17 (LEARN-EVALUATION)	Whenever Soar is in a no-change impasse, this problem space is proposed without testing for the super-operator. It is selected because the learn-evaluation operator is in place. This way we are sure that the learn-evaluation operator will not show up in the chunk and the chunk can fire in response to the external operator that is to be evaluated.
9	S: S18	

10	O: X7 (EXTERNAL-OP)	All operators in the top space are copied down to the sub-space without testing for the learn-evaluation super-operator. The external operator to be evaluated is then selected.
11	O: O25 (POSSIBLE-EVAL) (SP P28 (GOAL <G1> ^OBJECT NIL ^STATE <S1> ^OPERATOR <X1> +) -{(OPERATOR <X1> ^EVALUATION <E1>) (O <E1> ^ID J34)} (OPERATOR <X1> ^NAME EXTERNAL-OP ^LOOK RIGHT) (STATE <S1> ^MODEL <L1>) (O <L1> ^DTI 10 ^MANOEUVRE TURN-RIGHT) --> (OPERATOR <X1> ^EVALUATION <E2> &, <E2> +) (O <E2> ^VALUE 3 + ^ID J34 +)) Firing 12:62 P28 Firing 12:62 P28	Internally Soar generates all possible evaluations and then selects the right evaluation by testing for the evaluation in the top space. Soar builds an evaluation structure <e2> and then adds this evaluation structure to the external operator X7. Because X7 is an operator at the top-level, a chunk is built. The chunk reads in English. // a model <m> with ^dti 10 and ^manoeuvre turn-right an external operator <x1> ^look right proposed Then add to operator <x1> an evaluation <e> with an evaluation 3 and a unique id.
12	O: X29 (EXTERNAL-OP) operator for ^looking left	and now the other external operator may be applied.

A.1.2 Second run

0	G: G1	
1	P: P2 (BASE-LEVEL)	
2	S: S3 (IO-STATE)	
3	O: O5 (ADD-FIRST)	
	Firing 12:62 P28 Firing 12:62 P28	This time X7 is not chosen at random. Note that the chunk learned in the previous trace applied and had a positive evaluation.
4	O: X7 (EXTERNAL-OP) Look to the right and add the annotation ^look right to M.	
5	O: O10 (GET-EVALUATION) evaluation: 3	Soar is still in the so-called <i>evaluate external operators</i> mode and asks the outside world for an evaluation. This time the user answers with -3, a negative evaluation.
6	O: O13 (LEARN-EVALUATION)	As Soar is in the <i>evaluate external operators</i> mode, the data-chunking of an evaluation chunk is again attempted.
7	--> G: G16 (OPERATOR NO-CHANGE)	
8	P: P17 (LEARN-EVALUATION)	
9	S: S18	
10	O: X7 (EXTERNAL-OP)	
11	O: O25 (POSSIBLE-EVAL) (SP P55 (GOAL <G1> ^OBJECT NIL ^STATE <S1> ^OPERATOR <X1> +) -{(OPERATOR <X1> ^EVALUATION <E1>)	Again a data-chunk is learned, this time with a negative evaluation. // a model <m> with ^dti 10 and ^manoeuvre turn-right and

<pre>(O <E1> ^ID J58)}) (OPERATOR <X1> ^NAME EXTERNAL-OP ^LOOK RIGHT) (STATE <S1> ^MODEL <L1>) (O <L1> ^DTI 10 ^MANOEUVRE TURN-RIGHT) --> (OPERATOR <X1> ^EVALUATION <E2> &, <E2> +) (O <E2> ^VALUE -3 + ^ID J58 +)) (SP P60 (GOAL <G1> ^OBJECT NIL ^STATE <S1> ^OPERATOR <X1> +) (OPERATOR <X1> ^EVALUATION <E1>) (O <E1> ^ID J34) --> (O <E1> ^INVALID T +, ^INVALID T &))</pre>	an external operator <x1> ^look right is proposed <i>Then</i> add to operator <x1> an evaluation <e> with an evaluation -3 and a unique id. But, more interestingly, the chunk learned in the previous trace is unlearned. In English: <i>If</i> an operator <x1> is proposed with evaluation <e> with the unique id J34 <i>Then</i> add the invalid attribute to this evaluation. The addition of an invalid attribute ensures that no preferences are generated for this evaluation.

A.1.3 Third run

0 G: G1	
1 P: P2 (BASE-LEVEL)	
2 S: S3 (IO-STATE)	
3 O: O5 (ADD-FIRST)	
Firing P28 Firing P55 ----- Firing P60	P28 adds a +3 evaluation to X7 while P55 adds a -3 evaluation. In the next cycle P60 <i>invalidates</i> the +3 evaluation.
4 O: X10 (EXTERNAL-OP) Look left and add ^look left to M.	Because the evaluations are translated into preferences, only the -3 evaluation takes effect and thus X10, look left, is chosen.
5 O: O10 (GET-EVALUATION)	and the story continues....

A.2 Productions for learning from external interaction and error correction

The following productions go with the traces. Note that we use the spe notation for productions. In this shorthand notation the symbols *goal*, *problem-space*, *state* and *operator* are omitted. Instead, special variables are assigned a semantic role. The following lists these variables.

symbol	expands to...
<g> <sg>	goal <g>
<p> <sp>	problem-space <p>
<s> <ss>	state <s>
<o> <o1> <o2> <so>	operator <o>

For the sake of readability the variables <sg>, <sp>, <ss> and <so> are consistently used to denote super-goal, super-state, super-problem-space and super-operator. They expand, however, as listed above.

The productions are roughly ordered according to their place in the entire problem-solving process. This way they accompany the runs above. I realise that the trace can only be understood well by Soar aficionados but I have tried to provide enough comments in simple English to enable the reader to have a try.

0. Starting

Start the demo task by installing the base-level problem space and io-state.

```
(spe start
 (<g> ^object nil)
 -->
 (<g> <p> <s> )
 (<p> ^name base-level)
 (<s> ^name io-state))
```

1. Task Knowledge

1.1 ADD-FIRST builds up the task state

One of the objectives of this demo is to show how a data-chunk is learned with on the left-hand side a model of the external world and a proposal for an external operator and on the right-hand side the evaluation for the external operator. In this demo the building-up of the external model via the Soar I/O is faked. The object ^model <m> is simply conceived as the representation of the external world.

The operator ADD-FIRST initiates the task by adding the operator.

1.1.1 Proposal

```
(spe base-level*propose*add-first
 (<g> <s> )
 (<s> ^model)
 -->
 (<g> <o> +)
 (<o> ^name add-first ^model <m> )
 (p <m> ^dti 10 ^manoeuvre turn-right))
```

1.1.2 Application

Note that in this demo Soar is in the so-called Evaluate-External-Ops mode (EEO mode). The effect of this flag is that Soar will try to get an evaluation from the outside world every time an external operator has been applied. The flag can of course be turned off.

```
(spe base-level*apply*add-first
 (<g> <s> <o> )
 (<o> ^name add-first ^model <m> )
 -->
 (<s> ^model <m> ^mode evaluate-external-ops))
```

1.2.3 Termination

```
(spe base-level*terminate*add-first
 (<g> <s> <o> )
 (<s> ^model <m> )
 (<o> ^name add-first)
 -->
 (<g> <o> @))
```

1.2 The external operators

1.2.1 Proposal

One of the basic ideas of this demo is that Soar does external learning and error correction regarding evaluations (and indirectly preferences) of operators. We therefore need at least two operators to choose from.

```

(spe base-level*propose*external-op1
  (<g> <s> <s> ^object nil)
  (<s> ^model <m> )
  (o <m> ^dti 10 ^manoeuvre turn-right ^look right)
  ->
  (<g> ^operator <x> +)
  (operator <x> ^name external-op ^look right))

(spe base-level*propose*external-op2
  (<g> <s> <s> ^object nil)
  (<s> ^model <m> )
  (o <m> ^dti 10 ^manoeuvre turn-right ^look left)
  ->
  (<g> ^operator <x> +)
  (operator <x> ^name external-op ^look left))

(spe simulate*base-level*apply*external-op1
  (<g> <s> <o> <o> ^object nil)
  (<s> ^model <m> )
  (o <m> ^manoeuvre turn-right)
  (<o> ^name external-op ^look right ^copied-old-model T)
  ->
  (o <m> ^look right))

(spe simulate*base-level*apply*external-op2
  (<g> <s> <o> <o> ^object nil)
  (<s> ^model <m> )
  (o <m> ^manoeuvre turn-right)
  (<o> ^name external-op ^look left ^copied-old-model T)
  ->
  (o <m> ^look left))

```

The operators are made indifferent so that one can be chosen at random.

1.2.3 Termination

```

(spe base-level*external-operators-default-indifferent
  (<g> <s> <o1> + <o2> + ^object nil)
  (<o1> ^name external-op)
  (operator { <> <o1> <o2> } ^name external-op)
  ->
  (<g> <o1> = <o2> ))

(spe base-level*terminate*external-op*step1
  (<g> <s> <o> <o> ^object nil)
  (<o> ^name external-op)
  ->
  (<o> ^applied T))

(spe base-level*terminate*external-op*step2
  (<g> <s> <o> <o> + ^object nil)
  (<o> ^name external-op ^applied T)
  ->
  (<g> <o> <o> @ -))

```

1.2.2 Application

The application of the external operators is faked. For the sake of the demo it is irrelevant that the addition to the mental model goes through the Soar I/O. What is relevant is that the application happens at the top level in such a way that no direct evaluation chunk can be learned from a sub-goal.

2. Tracing operators and states

2.1 Operator tracing

In the evaluate-external-ops mode (EEO mode) you must store the operators that you applied, otherwise there will be nothing to learn after the evaluation operator has done its work. The following two productions remember at all times the external op last applied.

```
(spe base-level"remember-external-op"first-time
(<g> <p> <s> <o>)
(<s> ^mode evaluate-external-ops ^last-external-op)
(<o> ^name external-op)
-->
(<s> ^last-external-op <o>))
```

```
(spe base-level"remember-external-op"later-times
(<g> <p> <s> <o>)
(<s> ^mode evaluate-external-ops ^last-external-op {<o> <o> <o1>})
(<o> ^name external-op)
-->
(<s> ^last-external-op <o1> . <o> +))
```

2.2 State tracing

Just as with the operators above, you must remember the situation in which an external operator was generated. The following two productions remember the state of model <m> before the external operator was applied. Again, this is very simplistic but it illustrates that somehow the right cues must be remembered, otherwise you will have to regenerate them later by more complex problem solving.

```
(spe base-level"remember-last-state"step-one
(<g> <p> <s> <o>)
(<s> ^mode evaluate-external-ops ^model <m>)
(<o> ^name external-op ^copied-old-model)
-->
(<s> ^last-model <lm>))

(spe base-level"remember-last-state"second-step
(<g> <p> <s> <o>)
(<s> ^mode evaluate-external-ops ^model <m> ^last-model <lm>)
(<o> ^name external-op ^copied-old-model)
(o <m> ^<feature> <value>)
-->
(o <lm> ^<feature> <value>)
(<o> ^copied-old-model T))
```

3. GET-EVALUATION does external evaluation

3.1 Proposal

In the EEO mode Soar evaluates an external operator immediately after it is applied.

```
(spe base-level"propose"get-evaluation
(<g> <p> <s>)
(<s> ^last-external-op <o2> ^mode evaluate-external-ops)
-->
(<g> <o> + <o2> +)
(<g> <o> > <o2>)
(<o> ^name get-evaluation ^operator <o2>))
```


3.2 Selection

And made better than all other external operators

```
(spe base-level*get-evaluation-better-than-external
 (<g> <p> <s> <o1> + <o2> + ^object nil)
 (<o1> ^name external-op)
 (<o2> ^name get-evaluation)
 ->
 (<g> <o2> > <o1>))
```

3.3 Application

The user is requested to enter an evaluation. He or she can choose an integer from -3 to +3. -3 stands for a very low evaluation, +3 for the highest (see also Section 6).

```
(spe task*base-level*apply*get-evaluation*first-time
 (<g> <p> <s> <o>))
 (<s> ^model <m> ^evaluation)
 (<o> ^name get-evaluation ^operator <o2>)
 ->
 (<s> ^evaluation <e>)
 (o <e> ^operator <o2> ^last-model <lm> ^value (accept)))

(spe task*base-level*apply*get-evaluation*later-times
 (<g> <p> <s> <o>))
 (<s> ^model <m> ^evaluation <eo>)
 (o <eo> ^operator <e> <o2>)
 (<o> ^name get-evaluation ^operator <o2>)
 ->
 (<s> ^evaluation <eo> . <e> +)
 (o <e> ^operator <o2> ^last-model <lm> ^value (accept))).
```

3.4 Termination

```
(spe base-level*terminate*get-evaluation
 (<g> <p> <s> <o> +)
 (<o> ^name get-evaluation ^operator <o2>)
 (<s> ^evaluation <e>)
 (o <e> ^operator <o2>)
 ->
 (<g> <o> @ -))
```

4. LEARN-EVALUATION via data-chunking

Because Soar is in the EEO mode, the learn-evaluation operator is proposed. The goal of this operator is to learn the association between (1) the model before the operator was applied and the external operator was proposed and (2) the evaluation of this operator.

4.1 Proposal

```
(spe simulate*base-level*learn-evaluation
 (<g> <p> <s>))
 (<s> ^last-external-op <o2> ^mode evaluate-external-ops ^evaluation <e>)
 (o <e> ^operator <o2>)
 ->
 (<g> <o> +)
 (<g> <o> > <o2>)
 (<o> ^name learn-evaluation ^operator <o2>))
```

4.2 Selection

```
(spe base-level "learn-evaluation-better-than-external
(<g> <p> <s> <o1> + <o2> + ^object nil)
(<o1> ^name external-op)
(<o2> ^name learn-evaluation ^finished)
-->
(<g> <o2> > <o1> >))
```

4.3a Application (part 1)

The model that was stored in ^last-model is now temporarily switched to the ^model, whereas the current model is stored in ^save-model. What you want in the future is for productions to be automatically triggered by the current model. While data-chunking, the old model should thus be the current model.

```
(spe base-level "learn-evaluation" switch-models
(<g> <p> <s> <o> >)
(<p> ^name base-level)
(<s> ^last-model <lm> ^model <m> >)
(<o> ^name learn-evaluation ^switch-models)
-->
(<o> ^switch-models 1)
(<s> ^model <m> - <lm> + ^save-model <m> >))
```

4.3a Application (part 2)

Once the data-chunk is learned you then switch back to the model that was saved in ^save-model. Please note that the following two productions take place after the complete data-chunking episode.

```
(spe base-level "switch-models-again
(<g> <p> <s> <o> + ^object nil)
(<o> ^name learn-evaluation ^finished T)
(<s> ^model <m> + ^save-model <sm> ^last-model <lm> >)
-->
(<s> ^model <m> - <sm> + ^save-model <sm> ^last-model <lm> > -))
```

4.4 Termination

```
(spe base-level "terminate-learn-evaluation
(<g> <p> <s> <o> + ^object nil)
(<o> ^name learn-evaluation ^finished T)
-->
(<g> <o> > @ -))
```

5. LEARN-EVALUATION Problem Space

5.1 Preliminaries

In a no-change impasse the learn-evaluation problem space is always proposed.

```
(spe learn-evaluation "always-propose-ps
(<g> <sg> ^impasse no-change ^attribute operator)
-->
(<g> <p> >)
(<p> ^name learn-evaluation))
```

Only after it is proposed does the following production look to see whether it must be selected. This way the problem space becomes INDEPENDENT of the super-operator and a chunk can be built that does not check for the learn-evaluation super-operator. If a chunk were to test for this operator it would

not be possible for it to react spontaneously to a model + proposed external operator.

```
(spe learn-evaluation*prefer-ps
(<g> <p> + <sg> ^impassé no-change ^attribute operator)
(<p> ^name learn-evaluation)
(<sg> <o> +)
(<o> ^name learn-evaluation)
-->
(<g> <p> > ||)

(spe learn-evaluation*new-state*pointer-to-super-model
(<g> <p> <p> <sg> )
(<p> ^name learn-evaluation)
(<sg> <ss> )
(<ss> ^model <m> )
-->
(<g> <s> )
(<s> ^model <m> ||)
```

5.2 Selecting the right external super-operator

The next thing we want to do is to test for the external operator that eventually will wind up in the data-chunk. Note, however, that again we do not want the learn-evaluate super-operator in the chunk. The mechanism to ensure this is:

- propose all super-operators
- select, from this set of super-operators, the right external operator in the learn-evaluation space by testing for the right external operator in the top space. However, because of our selection technique, the external operator will now also not show up in the chunk. We are thus forced to the following:

- generate a unique symbol on the state that is dependent on all the features of the external operator. This ensures that the operator will appear correctly in the chunk. See Section 7.
- generate a unique symbol on the state that is dependent on all the features of the model to be learned. This ensures that the model will appear correctly in the chunk. See Section 7.

5.2.1 Proposal

```
(spe learn-evaluation*propose-all-super-ops
(<g> <p> <p> <s> <sg> )
(<p> ^name learn-evaluation)
(<s> ^desired-super-op)
(<sg> <so> +)
-->
(<g> <so> + ||)
```

5.2.1 Selection

```
(spe learn-evaluation*select-desired-super-op
(<g> <p> <p> <s> <o> + <sg> )
(<p> ^name learn-evaluation)
(<sg> <ss> )
(<ss> ^evaluation <e> )
{o <e> ^operator <o> }
-->
(<g> <o> > ||)
```

5.2.2 Application

```
(spe learn-evaluation* apply-desired-super-op
 (<g> <p> <s> <o> >)
 (<p> ^name learn-evaluation)
 (<s> ^desired-super-op)
 (<o> ^name external-op)
 ->
 (<s> ^desired-super-op <o> >))
```

5.2.3 Termination

```
(spe learn-evaluation* terminate-desired-super-op
 (<g> <p> <s> <o> > <sg> >)
 (<p> ^name learn-evaluation)
 (<s> ^desired-super-op <o> >)
 (<sg> <o> > +)
 ->
 (<g> <o> > @!))
```

5.3 POSSIBLE-EVAL: the final data-chunking mechanism

By now we are sure that the operator and the state will appear in the left-hand side of the rule. What we want next is to learn the evaluation that is stored on the external operator. The data-chunking mechanism here is to generate all the possible evaluations (in our case, stored on the possible-eval operators) before looking at the evaluation on the super-operator, thereby preventing the evaluation from also winding up in the left-hand side of the chunk to be learned. Note that the data-chunking in this example is *selective*. In general data-chunking all possible basic primitives are generated. In this example only the relevant composite objects (i.e. built up from lower primitives) are generated.

5.3.1 Proposal

```
(spe learn-evaluation* generate-all-possible-evaluations
 (<g> <p> <s> <o> >)
 (<p> ^name learn-evaluation)
 (<s> ^desired-super-op)
 ->
 (<g> ^operator <oe-3> + <oe-2> + <oe-1> > + <oe+0> > +
 <oe+1> > + <oe+2> > + <oe+3> > +)
 (operator <oe-3> ^name possible-eval ^value -3)
 (operator <oe-2> ^name possible-eval ^value -2)
 (operator <oe-1> ^name possible-eval ^value -1)
 (operator <oe+0> ^name possible-eval ^value 0)
 (operator <oe+1> ^name possible-eval ^value 1)
 (operator <oe+2> ^name possible-eval ^value 2)
 (operator <oe+3> ^name possible-eval ^value 3))
```

5.3.2 Selection

Only now look at top level

```
(spe learn-evaluation* prefer-possible-eval
 (<g> <p> <sg> <o> > +)
 (<sg> <ss> >)
 (<ss> ^evaluation <e> >)
 (o <e> ^value <v> >)
 ( <o> ^name possible-eval ^value <v> >)
 ->
 (<g> <o> > ||))
```

5.3.3 Application: building the data-chunk

The application of the possible-eval operator means that an evaluation structure is added to the evaluation. The evaluation consists of (1) the evaluation to be learned and (2) in addition to that a unique symbol.

```
(spe learn-evaluation* apply-possible-eval* step1
  (<g> <p> <s> <o>)
  (<s> ^state-symbol <ssym> ^operator-symbol <osym>)
  (<o> ^name possible-eval ^value <v>)
  -->
  (<o> ^evaluation <e> + &))
(o <e> ^id (call2 mysymbol) ^value <v> +))
```

(spe learn-evaluation* apply-possible-eval* step2

```
(<g> <p> <s> <o> <sg>)
(<s> ^desired-super-op <o1>)
(<sg> ^object nil)
(<o> ^name possible-eval ^evaluation <e>)
(o <e> ^id <id> ^value <v>)
-{ (operator <o1> ^evaluation <eo>)
  (o <eo> ^id <id> ) }
-->
(<o1> ^evaluation <e> + &))
```

5.3.4 Unlearning older evaluations

Yes, only one production is required to unlearn the older chunk. If the external operator to be evaluated already had an evaluation, that evaluation will be made effectively invalid by adding an 'invalid attribute to the evaluation. The traces and Section 6 of the program show the effect of this attribute.

```
(spe learn-evaluations* apply-possible-eval* reject-older-evaluations
  (<g> <p> <s> <o> <sg>)
  (<s> ^desired-super-op <o1>)
  (<sg> ^object nil) ;; prevent firing in learn evaluation space
  (<o> ^name possible-eval ^evaluation <e>)
  (o <e> ^id <id> ^value <v>)
  (operator <o1> ^evaluation <eo>)
  (o <eo> ^id <id> <id> ^invalid)
  -->
  (o <eo> ^invalid T + &))
```

5.3.5 Termination

Terminate the sub-goal after building chunks by adding a finished attribute to the external operator. [The 'finished attribute will not show up in the data-chunk because of the 'quiescence attribute in the first clause.]

```
(spe test
  (<g> <p> <s> <o> <sg> ^quiescence T)
  (<o> ^name possible-eval ^evaluation <e>)
  (o <e> ^id <id>)
  (<s> ^desired-super-op <o1>)
  (<o1> ^evaluation <eo>)
  (o <eo> ^id <id>)
  (<sg> <so>)
  (<so> ^name learn-evaluation)
  -->
  (<so> ^finished T))
```

6. Deriving Soar preferences from evaluations

The evaluations that are learned are only "indirect" preferences. In order to have any effect they must be translated into Soar preferences. The following

productions give a simple example of a possible scheme. Note that preferences are only created when an evaluation has NO ^invalid attribute.

```
(spe gen-preferences-from-evaluations*gt-zero
 (<g> <p> <s> <o> + ^object nil)
 (<o> ^evaluation <e>)
 (o <e> .^invalid ^value > 0)
 -->
 (<g> <o> > ))

(spe gen-preferences-from-evaluations*lt-zero
 (<g> <p> <s> <o> + ^object nil)
 (<o> ^evaluation <e>)
 (o <e> .^invalid ^value < 0)
 -->
 (<g> <o> < ))

(spe gen-preferences-from-evaluations*zero
 (<g> <p> <s> <o> + ^object nil)
 (<o> ^evaluation <e>)
 (o <e> .^invalid ^value 0)
 -->
 (<g> <o> = ))

(spe gen-preferences-from-evaluations*multiple-ops*different-evals
 (<g> <p> <s> <o1> + <o2> + ^object nil)
 (<o1> ^evaluation <e1>)
 (o <e1> .^invalid ^value <v>)
 (<o2> ^evaluation <e2>)
 (o <e2> .^invalid ^value > <v>)
 -->
 (<g> <o1> < <o2> ))
```

7. Recognition

It was noted above that in order to get the right operator and state in the left-hand side of the data-chunk we must generate symbols in the sub-space that are entirely dependent on all features of the state and operator. The following provides a very simplistic solution. In Chapter 3 we describe a generic solution for arbitrary deeply nested objects.

```
(spe simulation*recognize-model
 (<g> <p> <s> <s>)
 (<p> ^name learn-evaluation)
 (<s> ^model <m>)
 (o <m> ^diti <f1> ^manoeuvre <t2>)
 -->
 (<s> ^state-symbol (call2 mysymbol)))

(spe simulation*recognize-desired-super-operator
 (<g> <p> <s> <s> <sg>)
 (<p> ^name learn-evaluation)
 (<s> ^desired-super-op <o>)
 :: (<sg> <o> +)
 :: (<o> ^name external-op ^look <x>)
 -->
 (<s> ^operator-symbol (call2 mysymbol)))
```

Appendix 2: Basic driver operations in more detail

B.1 Introduction

In Chapter 4 the observed regularities were not extensively validated. This appendix provides a rather more detailed analysis of the data and contains the tables on which the regularities described earlier are based. The analysis is primarily intended for readers interested in the details of driver behaviour close to intersections.

B.1.1 Building the events database

The analysis performed on the data from the De Velde Harsenhorst and Lourens study was started by making two databases. The first database contains the continuous data sampled at 5 Hz. The second database contains the events extracted from the first database.

Database with continuous data: The first step consisted of transferring the information recorded on the audio track of the video tape to a PC. The audio track contained data relating to the speed, steering-wheel angle, brake, accelerator and clutch pressure and the use of the gear-stick. The exact entrance times for the six crossings selected were ascertained by means of the video recordings and then the 15 seconds before the intersection to the 7 seconds after the intersection were selected for each crossing. Given the sampling rate of 5 Hz we obtained 110 records per block. Table 1 shows the raw data obtained from the video tape. Table 2 shows which variables we were then able to derive from this first group.

Table 1. Continuous data

-
- time (T) real time to entrance point in seconds
 - speed (V) in meters per second.
 - steering wheel angle
 - brake pressure
 - accelerator pressure
 - clutch pressure
 - gear-stick state
-

Table 2. derived data

- distance-to-intersection (DTI) in meters to the entrance point obtained by integrating the speed signal)
- acceleration (ACC) in m/s^2 (obtained by differentiating speed signal)
- time-to-intersection (TTI) in seconds, computed as DTI/V .
- mean acceleration to intersection : the difference between current speed and speed at intersection divided by the absolute time to intersection.

Table 3: Events database

Vmax	Maximum speed before intersection (in the 15-second period)
Vint	Speed at intersection
GasMax1	Release of accelerator before intersection
Gas0	Accelerator completely released before intersection
Gas0*	Accelerator in after intersection
GasMax2	First maximum accelerator pressure after intersection
Brake0	Start of braking manoeuvre before intersection
BrakeMax1	Brake reaches first maximum
BrakeMax2	Start of release
Brake0*	Brake completely released
Clutch0	Clutch pressed for gear-change manoeuvre
ClutchMax1	Clutch reaches first maximum
ClutchMax2	Start of release
Clutch0*	Clutch completely released
Gear	Gear-stick is used
Steer0	Beginning of steering for curve
SteerMax	Maximum steering-wheel angle in curve
Steer0*	End of steering manoeuvre
LL	Looking left
LR	Looking right
LF	Looking forward
LRM	Looking rear mirror
LLM	Looking left mirror
LRS	Looking right shoulder
FL-right	First glance to the right
FL-left	First glance to the left.

Table 4 : Time, distance and pressure variables for defined events.

T	real time to intersection at time of event
DTI	distance to intersection at time of event
V	speed at time of event
TTI	time to intersection, computed from DTI/V at time of event
Acc0	average deceleration from event to intersection
press	the pressure on a pedal.

The events database. In the second step events are derived from the first database. A number of these events were derived automatically from these data, for example maximum speed before intersection, speed at intersection and minimum and maximum acceleration. All other events were collected by means of a graphics analysis tool. This tool worked as follows: an entire block of data (from 15 seconds before to 7 seconds after the location) is displayed in three diagrams on a computer screen. One diagram for speed and acceleration, one for the brake, clutch and accelerator pedals and one diagram for the steering-wheel angle. A vertical ruler, controlled by the analyst, marks the relevant events. Table 3 shows which events we were able to abstract from our continuous data.

A number of remarks are required regarding this table. The first remark relates to the reliability of the various events. In order to test the reliability of the analyst and the reliability of the precise timing of the events, we also calculated algorithmically all the car-device events from Table 3 on the basis of the continuous data. We find few to no differences for the onsets and offsets of the accelerator (Gas0 and Gas0*) and brake pedal (Brake0 and Brake0*). This is only logical, since these are curves that start from zero or go back to zero. The first and second maximum for brake and the first maximum of the clutch also showed high correlations between manual and automatic collection. However, the release of the accelerator pedal and the first maximum after the intersection (GasMax1 and GasMax*), as well as the release of the clutch pedal, can be extracted with rather less reliability by means of automatic collection and therefore had to be marked by an analyst. The second remark is that the looking directions were directly obtained from video tape and inserted in the events database.

The third remark is that we coded all events in time, distance and speed-based measures. The measures are listed in Table 4. In addition, for some events we computed the pedal pressure for that event. Note that the term pressure is slightly misleading here as the pressure values do not really indicate the pressure but the distance that the brake, accelerator or clutch was pressed as a percentage of the total possible distance.

B.1.2 Results

These results have been organised rather differently from in Section 4. We will first discuss the aspects of the speed profile and the car-device manipulations that determine the speed profile. We will then discuss the car-control events as a whole and conclude with the visual orientation.

Events related to speed

All subjects reduce speed in the approach to an intersection, with the exception of one subject crossing location 6. Table 5 presents the important events in the speed profile (Vmax and Vint) and the main car-manipulation events that determine that profile: Gas0, Brake0, BrakeMax1 and BrakeMax2 (see

also Figure 4 in Section 4). For each speed-related event V, DTI, T and TTI are listed. Table 6 presents the significant contrasts between locations (with a $p < .05$). Figure 5 in Section 4 provides a visual overview of the significant contrasts for DTI and TTI. The number of cases (N) used for Vmax and Vint is 24, the Ns for the other events can be obtained from Table 7. Note that for L6 both the release of the accelerator and use of the brake are significantly lower than for the other locations. A '+' in Table 6 indicates that the contrast is significant. A number 1, 2 and/or 3 in the 'L123-L4/6' columns means that the location differs significantly from location 4 or 6. For a few events in these tables we have added Acc0 (the average deceleration from event to intersection) and the pedal pressure (press) at that event.

Table 5. Speed and car-control events for all locations

		location 1	location 2	location 3	location 4	location 6
Vmax	V	35.5 (4.6)	34.6 (3.7)	36.7 (6.1)	36.5 (5.45)	39.4 (3.28)
	DTI	55.2 (33)	57.0 (19)	76.9 (28)	64.0 (18)	30.8 (14)
	T	7.0 (3.8)	7.3 (2.1)	9.1 (2.7)	7.3 (1.7)	3.0 (1.4)
	TTI	5.7 (2.9)	5.9 (1.9)	7.2 (2.2)	6.3 (1.5)	2.8 (1.2)
Gas0	V	32.5 (4.1)	34.7 (3.6)	36.3 (6.2)	35.7 (5.6)	39.7 (3.2)
	DTI	41.5 (19)	55.7 (12.6)	69.4 (28.9)	50.3 (16.8)	35.6 (8.7)
	T	5.5 (2.2)	7.0 (1.4)	8.3 (2.5)	5.9 (1.5)	3.4 (1.0)
	TTI	4.5 (1.7)	5.8 (1.3)	6.7 (2.1)	5.0 (1.3)	3.2 (0.8)
Brake0	V	31.7 (4.0)	33.2 (4.2)	33.5 (6.3)	34.2 (5.3)	39.2 (3.2)
	DTI	30.6 (11.3)	33.4 (8.0)	37.4 (14.8)	30.5 (9.1)	27.1 (8.3)
	T	4.4 (1.5)	4.6 (1.0)	5.0 (1.5)	3.8 (0.9)	2.8 (1.0)
	TTI	3.4 (1.0)	3.6 (0.6)	3.9 (1.0)	3.2 (0.6)	2.5 (0.8)
	Acc0	-0.9 (0.2)	-1.0 (0.3)	-0.9 (0.3)	-0.7 (0.3)	-0.7 (0.3)
BrakeMax1V	V	28.5 (4.7)	29.1 (4.1)	29.8 (6.4)	31.2 (5.4)	37.1 (3.8)
	DTI	18.4 (10.0)	17.2 (7.0)	22.5 (12.6)	18.8 (7.7)	16.2 (5.1)
	T	2.9 (1.4)	2.7 (0.9)	3.4 (1.4)	2.5 (0.9)	1.7 (0.6)
	TTI	2.2 (1.0)	2.1 (0.6)	2.6 (1.0)	2.1 (0.7)	1.6 (0.5)
	Acc0	-1.1 (0.4)	-1.3 (0.4)	-1.0 (0.3)	-0.8 (0.4)	-0.8 (0.5)
	Pres	64.2 (9.3)	69.2 (5.6)	66.8 (5.6)	56.4 (18)	52.2 (14)
BrakeMax2V	V	18.2 (4.2)	15.9 (6.4)	19.1 (2.2)	26.4 (5.3)	33.7 (4.9)
	DTI	2.4 (3.5)	1.0 (3.6)	3.6 (2.9)	9.0 (4.5)	6.9 (4.8)
	T	0.5 (0.8)	0.1 (0.8)	0.8 (0.6)	1.3 (0.6)	0.8 (0.5)
	TTI	0.4 (0.9)	0.0 (0.9)	0.7 (0.5)	1.2 (0.5)	0.8 (0.5)
	Pres	67.4 (6.4)	72.4 (6.8)	67.5 (6.8)	57.2 (18)	56.3 (14)
Vint	V	17.4 (4.2)	15.8 (5.2)	16.8 (3.1)	24.2 (5.6)	35.4 (4.8)

Table 6. Differences in speed control events for F with $p < 0.5$. The + 's in the table refers to a significant difference between two locations. The numbers in the table refer to the location that significantly differs from the location behind the '+. So a 3 in L123-L4 means that only location 3 differs from L4.

contrast:		L1-L2	L1-L3	L2-L3	L123-L4	L123-L6	L4-L6
Vmax	V					1, 2	+
	DTI		+	+		1, 2, 3	+
	T		+	+	3	1, 2, 3	+
	TTI		+	+		1, 2, 3	+
Gas0	V		+		1	1, 2, 3	+
	DTI	+	+	+	3	1, 2, 3	
	T	+	+	+	2, 3	1, 2, 3	+
	TTI	+	+		2, 3	1, 2, 3	+
Brake0	V					1, 2, 3	+
	DTI					2, 3	
	T				2, 3	1, 2, 3	+
	TTI				2, 3	1, 2, 3	+
BrakeMax1	acc0				1, 2, 3	1, 2, 3	
	V					1, 2, 3	+
	DTI					3	
	T				2, 3	1, 2, 3	+
BrakeMax2	TTI			+	2, 3	1, 2, 3	+
	acc0	+			1, 2	1, 2	
	press	+			2, 3	1, 2, 3	
	V			+	1, 2, 3	1, 2, 3	+
BrakeMax2	DTI			+	1, 2, 3	1, 2, 3	
	T			+	1, 2, 3	2	+
	TTI			+	1, 2, 3	2	+
	press	+		+	1, 2, 3	1, 2, 3	
Vint	V				1, 2, 3	1, 2, 3	+

Table 7. Frequency of use of gas, brake and clutch pedals. Left half shows absolute frequencies, the right half of the table shows percentage of use corrected for missing cases. (2 in L2 and 3 in L3 due to conflicts and measurement problems)

	L1	L2	L3	L4	L6	L1	L2	L3	L4	L6
release gas	23	22	21	24	19	96	100	100	100	79
use brake	24	22	21	23	10	100	100	100	96	41
use clutch	18	16	13	8	3	75	72	61	33	13
	65	60	55	55	32	90	90	87	67	44

Table 8. Correlation coefficients between variables V, DTI, TTI and T at events Gas0, Brake0 and BrakeMax1. For example the first shows the correlations between V and DTI at location 1 at events Gas0, Brake0 and BMax1.

vars	loc	Gas0	Brake0	BMax1
V-DTI	L1	0.56	0.69	0.77
	L2	0.28	0.73	0.78
	L3	0.74	0.81	0.81
	L4	0.62	0.80	0.75
	L6	0.11	0.11	0.05
V-TTI	L1	0.39	0.48	0.65
	L2	-0.12	0.36	0.62
	L3	0.54	0.64	0.68
	L4	0.24	0.47	0.49
	L6	-0.13	-0.09	-0.22
DTI-TTI	L1	0.93	0.92	0.93
	L2	0.87	0.84	0.92
	L3	0.91	0.91	0.92
	L4	0.86	0.84	0.89
	L6	0.92	0.88	0.85
T-DTI	L1	0.93	0.88	0.89
	L2	0.84	0.80	0.86
	L3	0.90	0.90	0.91
	L4	0.85	0.79	0.84
	L6	0.90	0.86	0.83
T-TTI	L1	0.94	0.92	0.92
	L2	0.92	0.91	0.92
	L3	0.94	0.93	0.94
	L4	0.92	0.89	0.90
	L6	0.93	0.89	0.89
T-V	L1	0.44	0.46	0.59
	L2	0.11	0.35	0.52
	L3	0.59	0.67	0.70
	L4	0.29	0.43	0.46
	L6	0.13	0.11	0.26

Events related to speed: intermediate intersection (L4) versus minor intersection (L6)

The results are fairly straightforward: Table 6 shows that L4 and L6 differ significantly for almost all events. Table 5 shows that in the approach to L6 Vmax occurs closer to the intersection in terms of T, DTI and TTI. Gas0, Brake0, BrakeMax1, and BrakeMax2 also occur closer to the intersection in L6 but for these events DTI does not differ significantly between L4 and L6. Speed is higher during all the events. Note also that Acc0 (the mean deceleration from brake0 to intersection) is the same for L4 and L6.

The general impression is that in L6 actions occur at the same distance to the intersection but due to the significantly higher speed events happen closer to the intersection in terms of time.

Events related to speed: major intersections (L123) versus the minor intersection (L6)

Nearly all events differ significantly between L123 (read: locations one, two, and three) and L6. All events at L6, up to BrakeMax2, occur closer to the intersection (in terms of DTI, TTI and T) and at a higher speed. BrakeMax2 however occurs closer (in DTI) to the intersection for L13, though not in T and TTI. We find no differences here between L2 and L6. Note that again DTI is an exception: DTI does not show any significant effect for Brake0 and BrakeMax1.

We can draw two conclusions from these data. First, the differences between L123 and L6 are even more marked than in L4 – L6 and, second, we see again that DTI at events differs less, but time measures (T and TTI) do differ, due to higher speed in L6.

Speed-related events: major intersection (L123) versus intermediate intersection (L4)

The most consistent differences between L123 and L4 are found for Vint, BrakeMax2 and Acc0. In other words, speed at intersection is found to be higher at L4, probably because subjects release the brake pedal earlier and choose a lower deceleration (-0.7 in L4 vs -0.9 in L123). This lower deceleration is also illustrated by the differences between brake pressure (see Table 5). The remaining differences are slightly harder to report/interpret. Before BrakeMax2 there are no differences between L1 and L4, while in the time measures TTI and T for Gas0 and Brake0 there are some consistent differences between L23 and L4, all similar to the differences between L123 and L6 (as one might expect). Note that though we see differences in the time measures there are again no DTI differences.

The conclusion that we draw from these data is that after Vmax our intermediate intersection (L4) really does lie in between L123 and L6.

Events related to speed: comparing the crossing, turn-right and turn-left manoeuvres

First compare the variables Vmax and Gas0 with Brake0 and BrakeMax1 in Table 6. For the first two variables we observe that 16 out of 24 possible outcomes are significant. Contrast this with the time and distance-based variables for the events Brake0 and BrakeMax1, in which only one (L2L3) of the 24 contrasts is significant (temporarily disregarding acc0 and brake pressure). In more detail the picture is the following: L3 has a somewhat higher Vmax (not significant) further away from the intersection. The same applies to the moment at which subjects release the accelerator (Gas0). However, at the point where subjects start to brake, the three manoeuvres become remarkably similar. In the approach to L3 the gap between Vmax and Brake0 is about 40 metres and 4.1 seconds, in L12 about 25 metres and 2.6 seconds. In BrakeMax2 we see that subjects release the brake earlier in L3 than in L2. However, they release the brake slower than in L2 (see Table 10) so that the resulting Vint is about the same as in L2.

The overall use of accelerator, brake and clutch pedals in speed control

The left half of Table 7 shows the general use of accelerator, brake and clutch pedals. Note that in L2 two cases and in L3 three cases are missing due to conflicts and bad data recordings. In the right half of the table we corrected for the missing cases and expressed the usage numbers as percentages. The table shows that after this correction L1, L2 and L3 do not differ in the use of the accelerator and brake. (The one missing case for release of the accelerator in L1 is caused by the fact that the release of the accelerator occurred before the 15 seconds that were used in the analyses; the accelerator was of course released). It is clear that in L4 the clutch is used less than in L123. Because subjects reduce speed less in L4 some of them stay in third gear. In L123 almost all subjects changed down to second or first gear. L6 clearly differs from the rest. Five subjects kept their foot on the accelerator pedal while approaching the intersection. It is clear from the table that releasing the accelerator is the most important way of regulating speed in L6; the brake is used in less than half of the cases and the use of the clutch indicates that subjects rarely change down.

Timing cues in events related to speed

One of the topics in the introduction concerns the timing cues that drivers use in the timing of behaviour. This section deals primarily with the timing of brake events.

The close relation between speed (V) and distance (DTI) at the moment of the onset of braking (Brake0) is shown in Figure 9 in Chapter 4. In this figure speed is plotted against DTI for L1234. The line of best fit is described by: $DTI = -24.06 + 1.72V$ (correlation 0.76).

Table 8 lists several relevant relations between the timing parameters in this study for the speed-relevant events Gas0, Brake0 and BrakeMax1 (see also Figure 8 in Section 4). The first block in the table again shows the relation between speed and distance. Note the high correlations for L1234 and the low correlations in L6. In L1234 all events occur earlier (in DTI) when speed is higher. The correlations for Brake0 and BrakeMax1 seem to be somewhat higher than for Gas0.

The second and third blocks in Table 8 show the relations V -TTI and DTI-TTI. If we look at the V -TTI block we see that the correlations rise from left to right. However, if we look at the DTI-TTI block we see that, first, the correlations are invariably higher than in the V -TTI block and, second, there is no increase from left to right. In conclusion we might say that DTI determines TTI more than V ; however, the importance of V increases closer to the intersection.

The last three blocks of Table 8 show the relation between T and DTI, TTI and V . The first block states that if the subject is further away from the inter-

section (DTI) then it will take longer to reach the intersection. The degree of the correlation is not so surprising if we remember the relative constancy of the acceleration. The same applies for the relation T-TTI. Remember that TTI is nothing more than distance corrected for speed; it is therefore not surprising that correlations are even higher than in T-TTI. In the last block, T-V, we see the same phenomenon as in the V-TTI comparison: a slight increase in correlation for events closer to the intersection.

The relation between V, DTI, TTI and Brake Pressure at BrakeMax1

One issue we deal with in Section 4 is whether drivers manipulate their approach speed by varying the timing of braking or by varying brake pressure. Here we will give more details about the relation between timing of braking and brake pressure.

Table 9 shows the relation between brake pressure and speed, distance to intersection and TTI at the moment the brake reaches its first maximum (BrakeMax1).

Table 9. Correlation coefficients between brake pressure and V, DTI and TTI for all locations at brakemax1.

	V-Press	DTI-Press	TTI-Press
L1	-0.23	-0.46	-0.47
L2	-0.02	-0.10	-0.11
L3	-0.07	-0.08	-0.09
L4	-0.00	-0.00	-0.01
L6	-0.04	-0.24	-0.15

Only in L1 do we see what might be expected: when BrakeMax1 occurs closer to the intersection (DTI and TTI), subjects press the brake harder. The low correlation for V-Press (L1) is counter-intuitive: if subjects have higher speed at BrakeMax1 then they will press the brake less. [Note: closer examination of the data reveals that in most of those cases where speed is high and brake pressure is low, DTI is large. The reverse is also true: when speed is low and brake pressure is low DTI is small].

Car-control events

In the the previous section we discussed the main events related to speed (Vmax and Vint) and the car-manipulation events that determine the speed profile. This section looks at the overall manipulation of the car in the approach to and negotiation of an intersection from a broader perspective. Included now are the steering wheel in L1 and L3, the use of the accelerator after the intersection and the use of the clutch. Using real time as the base, Table 10 shows the order of actions for locations. If L1 is taken as the reference it can be seen that almost all actions at all locations are performed in the

same order. The only exception is in some cases the clutch and the steering wheel¹.

The best way to show the consistency of the order of actions for subjects and the distribution of these events is to present the individual data. However, this is rather prohibitive and the following figure (Fig. 18-1) offers an alternative.

Table 10. the basic car control events and the first look to the right for all locations in real time (T) to intersection.

	L1	L2	L3	L4	L6
GasMax1 :	-6.4 (2.2)	-8.3 (1.3)	-9.2 (2.9)	-7.0 (1.9)	-5.3 (1.4)
Gas0 :	-5.5 (2.2)	-7.0 (1.4)	-8.3 (2.5)	-5.9 (1.5)	-3.4 (1.0)
Brake0 :	-4.4 (1.5)	-4.6 (1.0)	-5.0 (1.5)	-3.8 (0.9)	-2.8 (1.0)
Clutch0 :	-3.7 (1.5)	-2.7 (1.5)	-4.0 (1.1)	-3.2 (1.0)	
BrakeMax1 :	-2.9 (1.4)	-2.7 (0.9)	-3.4 (1.4)	-2.5 (0.9)	-1.7 (0.6)
ClutchMax1 :	-2.8 (1.6)	-2.0 (1.5)	-3.2 (1.0)	-2.5 (1.0)	
Fl-right :	-1.5 (0.5)	-1.3 (0.8)	-1.3 (1.0)	-1.9 (0.4)	-1.0 (0.5)
BrakeMax2 :	-0.5 (0.8)	-0.1 (0.8)	-0.8 (0.6)*	-1.3 (0.6)	-0.8 (0.5)
Steer0 :	-0.1 (1.2)		-1.7 (0.7)*		
ClutchMax2 :	0.1 (2.4)	0.8 (1.1)*	-1.5 (1.7)*	-1.2 (0.9)	
Brake0* :	0.2 (0.9)	0.5 (0.7)	-0.0 (0.6)	-0.9 (0.6)	-0.2 (0.6)
Gas0* :	1.5 (1.1)	0.9 (0.7)	1.0 (1.0)	-0.2 (0.9)	-0.3 (1.1)
SteerMax :	2.5 (1.2)		1.7 (0.4)		
Clutch0* :	3.6 (2.0)	3.2 (1.3)	2.9 (1.9)		
GasMax2 :	3.9 (1.7)	2.7 (1.4)	3.1 (1.4)	1.7 (1.5)	1.6 (1.2)
Steer0* :	5.3 (0.9)		4.5 (0.6)		

Figure 18-1 presents a cumulative frequency distribution for some (but not all) relevant events in the approach to L1. What this picture shows (and what was already clear from the standard deviations in Tables 5 and 10) is how the slopes of the cumulative distribution are a function of the time to the intersection. As subjects get closer to the intersection the variance in timing decreases.

Visual orientation

This section will look at how visual orientation differs between intersections and manoeuvres. The main variables that we will consider are the first look to the right and left.

¹ The difference in the use of the steering wheel is of course that in the left turn (L1) the driver first has to cross one lane before he can start to turn while in the right turn (L3) the driver will have started his steering manoeuvre even before crossing the entrance line.

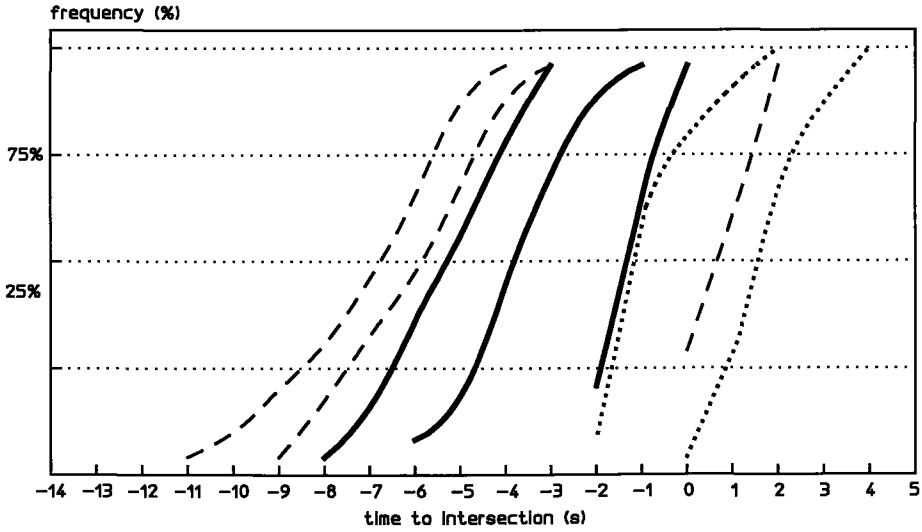


Figure 18-1. Cumulative frequency distribution for some relevant events for all the subjects in the approach to L1. The first two thin lines represent GasMax1 and Gas0. The three thick lines represent Brake0, BrakeMax1 and Brakemax2. The following dotted line represents steer0, the next thin line Gas0* and the dotted line SteerMax.

Table 11. First time looking to the right and left for V in km/hr, DTI in meters, and T and TTI in seconds

		L1	L2	L3	L4	L6
Fl-Right	V	23.8 (3.8)	21.6 (6.7)	21.2 (6.7)	28.3 (4.9)	36.2 (4.4)
	DTI	8.8 (4.0)	7.6 (5.7)	7.4 (7.3)	13.3 (3.5)	9.1 (4.6)
	T	1.5 (0.5)	1.3 (0.8)	1.3 (1.0)	1.9 (0.4)	1.0 (0.5)
	TTI	1.3 (0.4)	1.1 (0.7)	1.1 (0.8)	1.7 (0.3)	0.9 (0.4)
Fl-Left	V	20.3 (2.7)	21.9 (5.3)	23.2 (4.5)	28.1 (6.6)	
	DTI	3.4 (2.3)	6.4 (3.9)	10.4 (3.6)	16.6 (5.7)	
	T	0.6 (0.4)	1.2 (0.6)	1.9 (0.5)	2.4 (0.7)	
	TTI	0.6 (0.4)	1.0 (0.5)	1.6 (0.4)	2.1 (0.6)	

Table 12. Frequency of looking right and left in conflict free situations

	L1	L2	L3	L4	L6
Available Cases	19	21	20	23	23
Right	17	20	18	23	22
Left	18	17	17	9	0

Table 13. Differences in speed control events for F with $p < 0.5$. The **s in the table refers to a significant difference between two locations. The numbers in the table refer to the location that significantly differs from the location behind the ' '. So a 3 in L123-L4 means that location 3 differs from L4.

contrast:		L1-L2	L1-L3	L2-L3	L123-L4	L123-L6	L4-L6
Fl-Right	DTI				1, 2, 3		*
	T				1, 2, 3	1	*
	TTI				1, 2, 3	1	*
Fl-Left	DTI		*	*	1, 2, 3		
	T		*	*	1, 2, 3		
	TTI		*	*	1, 2, 3		

Table 11 lists the first looks to the right (Fl-right) and left (Fl-left) for all locations, in the same manner as Table 5 did for the speed-control events. Table 12 shows the number of cases for each cell in Table 11. The first row in Table 12 shows how many approaches could be used for our analysis. Note that these numbers differ from those in Table 5. In Table 5 all approaches were taken where it was obvious that speed control was not hindered by other traffic. In Table 12 more cases are missing because other traffic clearly attracted attention. The second and third rows in Table 12 show that in the remaining cases for L123 almost all subjects looked to the right and left, whereas in L4 only nine and in L6 (the T-junction) nobody looked to the left.

Table 13 lists the significant differences between the locations, in the same manner as in Table 6. The variable V is left out as we have already seen in previous sections that it was significant in all cases.

The first look to the right (Fl-right)

Tables 11 and 13 show that there is a consistent difference between Fl-right in terms of DTI, TTI and T for L123-L4 and L4-L6. That is, in L4 subjects look earlier to the right both in time and in space. It is interesting to see that subjects in both L123 and L6 look right at about the same distance from the intersection. Due to the higher speed in L6 we find a significant difference between L6 and L1 for T and TTI.

Table 14. stopping distance at first look to the right given reaction time = 0.5 and initial deceleration = -1 m/s²

a	L123	L4	L6
-1.0	19.8	34.0	54.9
-2.0	11.3	18.9	29.9
-3.0	8.5*	13.8*	21.5
-4.0	7.0	11.3	17.4
-5.0	6.2	9.8	14.9
-6.0	5.6	8.8	13.2

Table 14 shows hypothetical stopping distances for the speed at Fl-right, given several decelerations. The first column shows the accelerations. The following columns show the stopping distances for Fl-right at L1 (21s), L4 (28s) and L6 (36s). The stopping distances are computed on the basis of two assumptions. The first assumption is that reaction time is the usual 0.5 seconds [note: 0.5 is rather fast given the fact that the subject has to search for a car from the right, this in contrast to for example car following experiments where a subject is already looking forward. However, one thing that speeds up the reaction time is the fact that all subjects already have their right foot on the brake (see Table 7), this again in contrast to car following experiments where subjects have their right foot on the accelerator pedal]. The second assumption is that the initial deceleration is already -1 m/s^2 . This is certainly true for L123, somewhat optimistic for L4 (see Tables 5 and 7) and overly optimistic with respect to L6, as only a few drivers used their brake.

Table 14 shows that the deceleration required to come to a halt before the intersection is -3 m/s^2 for both L123 and L4 at Fl-right. The speed at Fl-right for L6 is so high that even a deceleration of -6 m/s^2 , a maximum for most cars, would result in a stopping distance that is larger than the DTI at Fl-right.

The first look to the left (Fl-left) and right (Fl-right)

Table 13 shows that for Fl-left *all* locations differ from each other (the two columns for L6 are empty because at L6 no one ever looked or needed to look to the left). The differences will be explained in the following paragraphs, but a good overview of differences is given in Figure 18-2. This figure, Table 11, Table 15 and Table 16 give some more information about the order and duration of looking directions.

Figure 18-2 standardises DTI, TTI and T with respect to L1. In this figure 100 represents 8.8 metres for DTI, 1.5 seconds for T and 1.3 seconds for TTI. It is clear from this picture that Fl-right is much more stable by location than Fl-left. Note also that for L1, L3 and L4 we find a clear difference between Fl-right and left. However, there is no such difference for L2.

Table 15 shows a simple (sequential) analysis performed on the individual eye and head-movement data. In this analysis I looked at the first few eye and head movements in the 5 seconds before the intersection. At L1 16 subjects showed a right-left or right-forward-left sequence. The latter category (r-f-l) is included in this table because 11 of those 16 had a clear 0.2 to 0.4s forward fixation between the right-left looks. At L2 R-(F)-L wins by 13 to 7. However, it is not an easy win compared to the other locations and that might explain why we see no differences in the time and distance-based parameters in Table 11. L3 is more clear-cut than L2; here left first wins by 15-4. In L4 14 subjects looked only to the right but where subjects looked to the left they looked left before right. L6 needs no explanation.

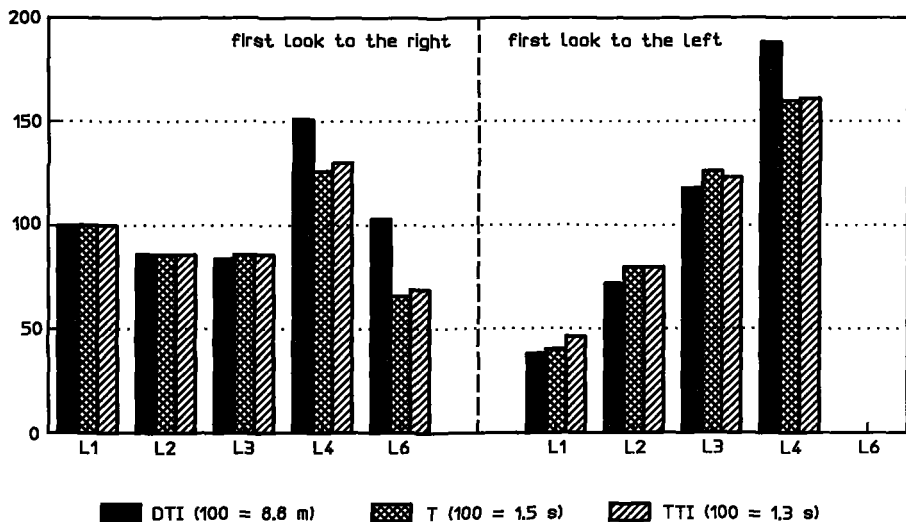


Figure 18-2. This figure standardises DTI, TTI and T using the first look to the right at location L1 as the basis. 100 represents 8.8 metres for DTI, 1.5 seconds for T and 1.3 seconds for TTI.

Table 15. counts of visual orientation patterns before the intersection

	L1	L2	L3	L4	L6
Right/Forward/Left	11	6	3		
Right/Left	5	7	1	1	
Left/Forward/Right		5	8	4	
Left/Right		2	7	3	
Only Right				14	22
Only Left	1				
None					1

Table 16 shows the durations for the looks (fixations) to the right and left during the negotiation of the intersection (from 5 seconds before to 3 seconds after the intersection). The numbers between parentheses show the average number of times that a subject looked in a certain direction (computed as total number of times subjects looked in a certain direction divided by number of subjects). So a rough interpretation of (1.23) in the first columns is that a quarter of the subjects looked more than once to the right, whereas (1.94) in the first columns means that most subjects had two or more fixations to the right. The total looking time (Fl-right + Fl-left) is lowest for L6 (the T-junction). The crossing manoeuvres L2 and L4 require less looking time than the right and left turns (L1, L3), mainly because making a turn requires looking at the curve for some time.

Table 16. Durations for fixations to right and left from 5 seconds before to 3 seconds after the intersection. Numbers between parenthesis shows the average number of times that a subject looked in a certain direction (computed as total number of times subjects looked in a certain direction divided by number of subjects.)

	L1	L2	L3	L4	L6
Duration R	0.78 (1.23)	1.62 (1.76)	2.17 (1.69)	1.37 (1.56)	1.04 (1.0)
Duration L	2.14 (1.94)	0.91 (1.65)	1.07 (1.66)	0.65 (1.44)	
	2.92	2.53	3.24	2.02	1.04

Samenvatting

Inleiding (hoofdstuk 1)

Deze studie beschrijft een cognitief model van het gedrag van een automobilist bij het naderen en afhandelen van kruisingen. Het model bestuurt een gesimuleerde auto in een gesimuleerde verkeerswereld waarin ook andere auto's en fietsers rondrijden. De belangrijkste rijtaken die het model uitvoert zijn het afhandelen van ongeordende kruisingen, het regelen van de snelheid, het koers houden op de rechte weg, het sturen door bochten, en het navigeren ofwel het vinden van een bestemming in de gesimuleerde wereld. Bij de beschrijving van het model komen daarbij drie onderwerpen uitgebreid aan de orde. In de eerste plaats de visuele oriëntatiestrategieën die nodig zijn om de rijtaak goed te kunnen uitvoeren. Dit wil zeggen dat we de oog- en hoofdbewegingen, maar ook de interne aandachtsmechanismen van een automobilist bij het naderen en afhandelen van een kruising, modelleren. In de tweede plaats behandelen we de motorische processen die nodig zijn om een auto te besturen. We beschrijven onder andere hoe dit model zijn ledematen leert aansturen bij het bedienen van de instrumenten van de auto. Tenslotte besteden we veel aandacht aan de *multitasking* mechanismen die nodig zijn om de vele taken die een automobilist in kritische verkeerssituaties tegelijkertijd moet doen, ook werkelijk uitgevoerd kunnen worden.

Het model van rijgedrag is geïmplementeerd met behulp van het productiesysteem Soar. Dit systeem is computer-implementatie van een algemene theorie van menselijk probleemoplossen en wordt als zodanig gebruikt door cognitief psychologen om vele soorten menselijke gedragingen te modelleren. Soar is ontwikkeld onder leiding van Allen Newell (Laird, Newell, & Rosenbloom, 1987) en bouwt voort op het werk van Newell en Simon op het gebied van menselijk probleemoplossen. Soar is daarbij in feite de belichaming van Newell en Simon's *Problem Space Hypothesis*, geïmplementeerd in een parallel productiesysteem. Om de niet in Soar ingewijde lezer enig houvast te geven bij het lezen van deze samenvatting worden de belangrijkste steekwoorden betreffende Soar hier kort toegelicht.

Productiesysteem. Een productiesysteem is een combinatie van *als-dan* regels en een werkgeheugen waarin tijdelijke informatie opgeslagen is. Voortdurend worden deze regels getoetst aan het werkgeheugen. Als een regel waar is, gezien de inhoud van het werkgeheugen, dan wordt de inhoud van het werkgeheugen gewijzigd. Soar is een *parallel* productiesysteem omdat meerdere regels tegelijkertijd waar kunnen zijn en daarmee ook tegelijkertijd de inhoud van het werkgeheugen kunnen wijzigen. Eén van de voordelen van het gebruik van productiesystemen is dat men vrij eenvoudig kan uitrekenen hoe lang een taak duurt. Hierdoor wordt het mogelijk om modellen te testen aan empirische gegevens.

Probleemruimte hypothese. Deze hypothese is, naast het formalisme van productiesystemen, een andere hoeksteen van Newell en Simon's theorie van menselijk probleemoplossen. De meest algemene formulering van de theorie is dat *alle menselijk gedrag beschreven kan worden als probleemoplossen en dat bovendien alle probleemoplossen beschreven kan worden als heuristisch zoeken in probleemruimtes*. In Soar bestaat een probleemruimte uit *toestanden* en *operatoren*. Een toestand is een representatie van de huidige probleemsituatie in Soar's werkgeheugen. Operatoren zijn datastructuren in Soar's werkgeheugen die specificeren hoe de huidige toestand veranderd zou kunnen of moeten worden. Probleemoplossen in Soar betekent dat de *begin-toestand*, door middel van het toepassen van een sequentie van operatoren, getransformeerd wordt tot de zogenaamde *doel-toestand*.

Universal subgoalng. Zolang Soar nog geen expert is voor een bepaalde taak, zullen er tijdens het probleemoplossingsproces zogenaamde *impasses* optreden. Er is bijvoorbeeld niet genoeg kennis aanwezig om te kiezen tussen meerdere operatoren of om een operator toe te passen. Soar zal voor zo'n impasse automatisch een nieuw subdoel creëren waarin deze impasse als nieuw probleem wordt aangepakt. Dit subdoel kan eenzelfde of een totaal andersoortige probleemruimte vereisen. Soar is een *recursieve* probleemoplosser omdat in elke probleemruimte weer nieuwe subdoelen gecreëerd kunnen worden.

Chunking. Het leermechanisme van Soar heet chunking. Dit mechanisme wordt geactiveerd als een impasse opgelost is en Soar naar het doel teruggaat waar de impasse ontstond. Bij het terugspringen worden nieuwe regels (chunks) aangemaakt die als het ware de oplossing van de impasse samenvatten. Moch Soar ooit weer in dezelfde situatie terecht dan heeft het nu de goede regels om wel een keuze te maken of een operator toe te passen.

Hoewel Soar een intrinsiek probleemoplossende architectuur is, blijft de vraag of het ook een psychologische theorie van menselijke gedrag is. Newell neemt duidelijk stelling door Soar te presenteren als een voorbeeld van een zogenaamde "unified theory of cognition". Hij stelt dat "de tijd nu rijp is voor de psychologie om te streven naar geünificeerde theorieën van cognitie – dat zijn theorieën die hun kracht ontleen aan het beschikken over een enkelvoudig systeem van mechanismen dat samenwerkt om alle facetten van de menselijke

cognitie te reproduceren." Met 'alle facetten' bedoelt Newell dat (1) de architectuur in staat moet zijn om zowel routinetaken als zeer complexe problemen aan te pakken; (2) dezelfde uniforme representatie gebruikt wordt voor perceptuele, motorische en cognitieve taken; (3) alle mogelijke probleem oplossingsmethoden worden gebruikt en (4) leren een integraal deel van de architectuur vormt.

Hoewel Soar is gebaseerd op algemene principes van menselijk probleemoplossen is het mogelijk in Soar taken op zo'n manier te implementeren dat het resultaat alsnog niet psychologisch verantwoord is. In zijn boek *Unified Theories of Cognition* (UTC) stelt Newell daarom nog extra eisen aan de implementatie van taken in Soar. Deze eisen hebben voornamelijk te maken met de activiteiten (operatoren) die in Soar's basisruimte moeten plaatsvinden. Deze basisruimte is de probleemruimte die altijd in Soar's werkgeheugen aanwezig is en waar alle andere subdoelen en subruimtes aan ontspruiten. Twee van deze eisen hebben de implementatie van ons model van rijgedrag sterk beïnvloed. De belangrijkste is dat alle input en output vanuit de basisruimte gebeurt. De tweede is dat taak-operatoren in deze basisruimte plaatsvinden. Waarom deze eisen zo sterk domineren wordt in de volgende hoofdstukken uitgewerkt.

De twee onderwerpen die tot nu toe in deze inleiding apart aan de orde kwamen zijn het model van rijgedrag en het gebruik van Soar. In de rest van deze studie zullen we deze twee geïntegreerd behandelen, wat niet ongebruikelijk is als het één in het ander geïmplementeerd wordt. Toch moet men bij het lezen van deze studie beseffen dat we in feite twee doelen nastreven. In de eerste plaats is deze studie een poging om te komen tot een psychologisch valide model van menselijk rijgedrag, dat wil zeggen, een model dat rekening houdt met de belangrijkste menselijke vaardigheden en beperkingen. In de tweede plaats kan de studie gelezen worden als een evaluatie van de theoretische en praktische geschiktheid van het cognitieve modelleer-medium *Soar* voor het modelleren van complex dynamisch taakgedrag, waarbij het rijgedrag als exemplarisch voor dit type gedrag wordt beschouwd.

De studie bestaat uit twee delen. Deel I bevat een aantal hoofdstukken die weliswaar niet het uiteindelijke model van rijgedrag beschrijven maar die wel zeer van belang zijn voor het begrijpen van het uiteindelijke model dat in deel II behandeld wordt.

Deel I

Multitasking in driving: een eerste poging tot het modelleren van rijgedrag (hoofdstuk 2)

Dit hoofdstuk, eerder gepubliceerd als Aasman en Michon (1992), beschrijft een eerste poging om een aantal aspecten van het menselijk rijgedrag in Soar te modelleren. Er wordt een eenvoudig Soar model beschreven, dat in staat is een gesimuleerde kruising, waar ook een aantal andere semi-intelligente auto's en fietsers rijden, af te handelen. Het belangrijkste dat in dit hoofdstuk aan de orde komt is hoe Soar de vele deeltaken en sub-doelen, die tegelijkertijd actief kunnen zijn in kritische verkeerssituaties, kan afwikkelen.

De deeltaken die in dit model zijn geïmplementeerd zijn snelheidscontrole, koerscontrole, afhandelen van kruispunten en navigatie. Een eerste complicatie is dat deze taken in principe alle tegelijkertijd actief kunnen zijn. Een tweede complicatie is dat de aard van de doelen van deze taken niet eenduidig is. Kenmerkend voor de *multitasking* (of meervoudige taakuitvoering) in dit model is dat meerdere deeltaken op een uniforme wijze in de basisruimte gerepresenteerd worden. Een taak wordt gestart door een taak-operator in de basisruimte te installeren; de uitvoering van de taak vindt in de eerstvolgende subruimte plaats. Taken worden beëindigd als er een zogenaamde *interrupt-operator* voor een andere taak gegenereerd wordt.

Positieve aspecten van dit eerste eenvoudige model zijn dat we kunnen laten zien dat "multitasking door task-switching", het afhandelen van meerdere types doelen, taak-interruptie, automatische en gecontroleerde taakverwerking en *bottom-up* en *top-down* informatie verwerking op een redelijk eenvoudige wijze rechtstreeks in Soar te implementeren zijn.

De positieve aspecten blijken echter niet op te wegen tegen de negatieve aspecten van het model. In de eerste plaats is de *task-switching* (het taakwisselen) in dit model zeer inefficiënt. In het model kost het teveel operatoren om van taak te wisselen en daardoor blijkt het model amper tot real-time gedrag in staat te zijn. De belangrijkste oorzaak voor deze inefficiëntie is dat hele doelhiërarchie afgebroken moeten worden om een andere taak op te zetten in een andere taakruimte. De tweede reden is dat Soar's *default regels* (standaard zoekregels) voor het afhandelen van impasses veel te gevoelig zijn voor regelmatige interrupties. In hoofdstuk 3 laten we zien hoe door het veranderen van deze standaard zoekregels Soar wel in staat is om met interrupts om te gaan. Een tweede probleem van dit eerste eenvoudige model is dat de perceptie en motoriek bijna volledig ontbreken. Er is nog geen sprake van het modelleren van oogbewegingen en visuele oriëntatie of van het manipuleren van instrumenten in de auto.

Het afplatten van doelhiërarchieën (hoofdstuk 3)

Dit hoofdstuk, eerder gepubliceerd als Aasman en Akyürek (1992), beschrijft een aantal oplossingen voor het probleem dat probleemoplossen met gebruikmaking van Soar's default regels leidt tot veel te grote doelhiërarchieën en daardoor veel te gevoelig is voor externe interrupties.

Het is gebruikelijk dat Soar tijdens het probleemoplossen zeer diepe doelhiërarchieën ontwikkelt waarbij een taak-operator in de basisruimte de top van de hiërarchie vormt. Soar kan op alle niveaus van deze hiërarchie leren, maar het is gebruikelijk dat pas zeer laat regels geleerd worden die consequenties hebben voor de taak-operator in de basisruimte. Als nu tijdens het opzetten van zo'n doelhiërarchie een externe interruptie een taak-operator verdrijft uit het operator-slot in de basisruimte, dan gaat de hele doelhiërarchie verloren en zal er meestal nog bijna niets geleerd zijn. Als de taak-operator ooit weer geïnstalleerd wordt, zal het probleemoplossen weer van voren af aan moeten beginnen.

In dit hoofdstuk worden drie varianten van Soar's zoekregels beschreven die dezelfde functionaliteit hebben als Soar's eigen regels, maar die er voor zorgen dat het probleemoplossen veel minder gevoelig wordt voor interrupties en daardoor veel robuuster.²

Basis handelingen bij het naderen en afhandelen van ongeordende kruisingen (hoofdstuk 4)

Een van de eisen die we stellen aan het cognitieve model in deel II is dat het realistisch gedrag vertoont, zowel voor wat betreft kwantitatieve als kwalitatieve aspecten van de taakuitvoering. Voor de ijking van ons model maakten we zeer dankbaar gebruik van de data van twee veldexperimenten die op het Verkeerskundig Studiecentrum te Haren zijn uitgevoerd door Jaap de Velde Harsenhorst en Peter Lourens (De Velde Harsenhorst & Lourens, 1987, 1988).

In hun eerste experiment kreeg een leerling-automobiliste 25 rijlessen in een geïnstumenteerde auto. Aan het eind van elke les reed de leerling zo'n twintig minuten een vaste route door een woonwijk. Met behulp van een aantal camera's werden haar oog- en hoofdbewegingen en de verkeerswereld vóór haar opgenomen. Daarnaast werden het gebruik van de pedalen, de stuurhoek en de snelheid geregistreerd. Ook werd alles wat in de auto gezegd werd door de leerling en de instructeur opgenomen. In hun tweede experiment reden 24

² Voor de Soar-ingewijde: de variant die uiteindelijk in deel II gebruikt wordt heeft als kenmerk dat operatoren in de tie-set nooit een tie-impasse opleveren daar elke operator in deze set door een *what-if* operator in een no-change impasse wordt geëvalueerd. Pas als alle operatoren geëvalueerd zijn zal er een keuze gemaakt worden. Hierdoor hoeft de goal-stack nooit veel dieper dan één niveau te gaan, maar zal elke evaluatie wel meteen als chunk bewaard worden.

jonge mannen, nadat ze aan de auto gewend waren, dezelfde 20 minuten route als de leerling.³

Hoewel De Velde Harsenhorst en Lourens voor beide experimenten ook de nadersnelheid en de oog- en hoofdbewegingen voor een aantal verschillende kruispunten hebben geanalyseerd, bleek deze analyse voor onze vraagstelling te summier. Wij hebben daarom hun data nogmaals geanalyseerd en daarbij gekeken naar alle geobserveerde handelingen die uit hun materiaal af te leiden waren. Dit leidde tot een aanzienlijk aantal handelingen; dit wil zeggen dat we uiteindelijk beschikten over alle discrete manipulaties van de pedalen, de continue stuurhoek, oog- en hoofdbewegingen. Daarnaast werden al deze handelingen nog eens op vier manieren gecodeerd, namelijk in afstand-tot-kruising, snelheid, tijd-tot-kruising (berekend als afstand-tot-kruising/snelheid) en de echte, geobserveerde tijd tot kruising.

Onze analyse voor een aantal verschillende kruisingen, leidde tot een groot aantal regelmatigheden in rijgedrag die alle voor het model in deel II zeer interessant bleken. Het zou hier echter te ver leiden om deze alle te noemen. De belangrijkste uitkomsten echter zijn:

- Voor alle pedalen bleek dat het gebruik in lineaire termen valt te beschrijven. Het intrappen of loslaten van pedalen blijkt steeds zeer gelijkmatig te gaan. Het gebruik van de rem bij de nadering van een kruising is daarbij het meest interessant: naast de constantie in snelheid van intrappen en loslaten, zien we dat de rem na het intrappen op een vast niveau blijft. De bestuurder regelt de oprijnsnelheid (van het kruispunt) dus door het variëren van het moment van loslaten van de rem maar niet door het variëren van de remkracht.
- De patronen van oogbewegingen variëren nogal tussen kruisingen, maar per kruising zijn er zeer vaste, en ook goed te verklaren, visuele oriëntatiestrategieën waar te nemen. Een zeer interessante regelmatigheid is dat de eerste

³ In hun eigen analyse van het eerste experiment keken De Velde Harsenhorst en Lourens vooral naar de verschillende soorten instructies en opmerkingen die de instructeur maakte (Zie tabel 1 in hoofdstuk 4 voor een overzicht). De meest relevante gegevens met betrekking tot ons cognitieve model in deel II zijn de volgende. In de eerste plaats bleek dat ruim 66 procent van alle opmerkingen die een instructeur maakt *correcties* zijn. Dit duidt erop dat de hoofdcomponent van het leren 'trial and error' lijkt zijn. Een tweede gegeven is dat, gezien het aantal opmerkingen, het afhandelen van een kruispunt de moeilijkste taak is om te leren. Dit rechtvaardigt naar onze mening onze keuze om ons in deze studie vooral op deze manoeuvre te richten. Een laatste, zeer relevant gegeven is dat van alle basistaken de visuele orientatie veruit de moeilijkste taak bleek te zijn. Het op het op het juiste moment in de juiste richting kijken lijkt het moeilijkst aan te leren, tot aan de laatste lessen toe. Dit gegeven rechtvaardigt ook de grote aandacht die we aan visuele orientatie in ons model van rijgedrag besteden.

blik naar rechts bij ongeordende kruisingen waar het zicht naar rechts slecht is, altijd valt tussen het eerste en tweede remmaximum.

- Als we alle manipulaties van de pedalen en de oogbewegingen combineren dan blijkt dat de *volgorde* van handelingen opvallend overeenstemt over alle kruisingen heen. De *timing* van de handelingen verschilt, maar is binnen kruisingen weer spectaculair eenduidig.

Deel II

Inleiding tot deel II (hoofdstuk 5)

Na het voorwerk in Deel I kan nu de implementatie van het uiteindelijke cognitieve model van rijgedrag beschreven worden. De naam die we voor dit model gekozen hebben is DRIVER. Net als bij het eenvoudige model dat in hoofdstuk 2 beschreven wordt, zijn de belangrijkste taken van DRIVER de afhandeling van kruispunten, de interactie met het andere verkeer, het voortdurend bewaken en bijstellen van de juiste snelheid en koers, en het navigeren door de verkeerswereld. Hiermee houden echter dan ook alle overeenkomsten met het vorige model op.

In de eerste plaats worden in DRIVER de aansturing van armen, benen, en daarmee ook de manipulatie van de instrumenten in de auto, gemodelleerd. Omdat Soar geen motorische subsystemen bevat die de uitvoering van bewegingen kunnen modelleren hebben we zelf een eenvoudig motorisch substelsysteem in Soar geïmplementeerd. Dit model van motoraansturing wordt daarnaast ook gebruikt om de ogen en het hoofd aan te sturen. In de tweede plaats houden we in DRIVER rekening met een aantal beperkingen van het menselijk perceptuele systeem, zodat zowel de interne aandacht als de visuele oriëntatie gemodelleerd kunnen worden. In de derde plaats is er in DRIVER een meer efficiënte wijze van multitasken geïmplementeerd die onder meer gebruik maakt van de default regels die in hoofdstuk 3 beschreven worden.

De kleine wereld van DRIVER (hoofdstuk 6)

Het gebruik van gesimuleerde verkeersomgevingen als *testbed* voor intelligente architecturen is de laatste tijd sterk in opkomst. Kenmerkend voor al deze verkeersomgevingen is dat ze bewoond worden door andere semi-intelligente objecten (auto's en soms ook fietsers) die verbazingwekkend realistisch gedrag vertonen. De meest interessante, maar ook de allermooiste, is zonder enige twijfel de *small world* van het Verkeerskundig Studiecentrum te Haren. In deze kleine wereld in virtual reality uitvoering, kunnen proefpersonen, gezeten in een speciaal geprepareerde auto, op vrij realistische wijze rondrijden. In hoofdstuk 6 beschrijven wij een zeer eenvoudige voorloper van deze kleine wereld waarin wij DRIVER laten rondrijden. Kern van hoofdstuk 6 is de uitleg hoe de regels en het regelsysteem van deze semi-intelligente objecten realistisch gedrag kunnen genereren. De semi-intelligente objecten in onze kleine

wereld hebben twee soorten regels. De eerste zijn de officiële regels die we voorzien hebben van een aantal, op empirische onderzoek en eindeloos proberen, gebaseerde parameters. Deze regels worden elke kloktik van de simulatie gematched (getoetst) aan de huidige situatie. Het enige wat deze regels doen is voorstellen doen voor positieve of negatieve acceleraties. De tweede soort regel is de metaregel die er voor zorgt dat als alle andere regels gematched zijn, altijd de laagste acceleratie gekozen wordt.

Basis motorische vaardigheden van DRIVER (hoofdstuk 7)

DRIVER stuurt in het kader van de uitvoering van de rijtaken twee armen, twee benen, een hoofd en twee ogen aan. Deze aansturing vindt slechts gedeeltelijk binnen Soar plaats; alleen de motor-commando's worden in Soar's werkgeheugen gegenereerd. Het grootste gedeelte van de motorische processen vindt in een eenvoudige Lisp-simulatie van de ledematen plaats buiten Soar. Wij waren genoodzaakt deze simulatie van de motorische processen en vaardigheden zelf te implementeren omdat deze (nog niet) in Soar geïncorporeerd zijn. Bij de implementatie van deze motorische vaardigheden hebben wij ons laten leiden door inperkingen of constraints die de literatuur en de UTC-theorie ons leverden. De belangrijkste constraints uit de literatuur zijn: (1) in Soar's werkgeheugen maken de *motor-commando's* onderdeel uit van *motor-programma's* die geschakeld zijn in zogenaamde *verwarde hiërarchieën* (tangled hierarchies), (2) de motor-commando's zijn voorzien van een aantal *parameters* waarvan vaststaat dat ze daadwerkelijk gebruikt worden (o.a. snelheid, kracht, doellocatie etc.); (3) het leren gebeurt door chunking, (4) er wordt geen *time-keeper* gebruikt en (5) voor feedback wordt de *feedback-chaining hypothesis* gebruikt. De belangrijkste UTC constraint is dat de motor-commando's *in de basisruimte* gebeuren en dat het encoderen/decoderen door de input/output mechanismen gebeurt.

De belangrijkste eigenschap van onze implementatie van motor gedrag is dat motor operatoren alleen een actie kunnen initiëren. Onmiddellijk daarna verdwijnen zij uit het werkgeheugen. Wel wordt door middel van feedback vervolgens de beweging continu 'gevolgd'. Op deze manier kunnen meerdere ledematen tegelijkertijd bewegen en actief zijn en kan er toch ingegrepen worden als een beweging op een of andere wijze gefrustreerd wordt. Bovendien kunnen andere Soar processen gewoon doorgaan, ondanks het feit dat bewegingen nog niet af zijn.

Gebruik van de motorische vaardigheden bij het gebruik van de auto (hoofdstuk 8)

In het vorige hoofdstuk werden de motorische vaardigheden van DRIVER onafhankelijk van de rijtaak beschreven. In dit hoofdstuk beschrijven we het gebruik van deze vaardigheden bij de bediening van de instrumenten in de auto. We gebruiken daarbij een vrij moeilijke taak als voorbeeld, namelijk het schakelen van de ene naar de andere versnelling. Het blijkt dat er bij het schakelen ruwweg vier soorten kennis te onderscheiden zijn: kennis over de

functies van de instrumenten (pedalen, pook) en welk lichaamsdeel bij welk onderdeel hoort; kennis over de constraints van de instrumenten in de auto (gas en koppeling niet tegelijkertijd naar beneden, de versnellingspook met rust laten als het koppelingspedaal niet ingedrukt is, etc.); kennis over de volgorde waarin handelingen moeten plaatsvinden en tenslotte kennis die er voor zorgt dat de handelingen, afhankelijk van de omstandigheden (snelheid, afstand tot de kruising, etc.) op juiste moment geïnitieerd worden.

DRIVER als beginnende automobilist heeft slechts de eerste twee typen tot zijn beschikking en moet de volgorde en de timing van handelingen leren. Dit leerproces gaat in twee fasen. In de eerste fase leert DRIVER met behulp van de kennis over functies en constraints een zogenaamd "device-plan" en een gereleerd "lichaamsplan" aan. Het blijkt dat Soar het leren van deze plannen zonder problemen aan kan. De tweede fase, de uitvoering van het lichaamsplan in de werkelijkheid, levert veel meer moeite op voor DRIVER. Het moeilijkste blijkt daarbij de timing van de handelingen te zijn.

Basis perceptie in DRIVER (hoofdstuk 9)

Uit meerdere studies, waaronder die van De Velde Harsenhorst en Lourens, blijkt dat visuele oriëntatie de moeilijkste basistaak voor de automobilist is, en waarschijnlijk ook het grootste beslag op centrale cognitie legt. We hebben er daarom voor gekozen veel aandacht aan het modelleren van de visuele oriëntatie te besteden. Echter, net als voor de motorische vaardigheden ontbeert Soar een gedetailleerde theorie van perceptie en visuele oriëntatie. Soar biedt, in Newell's woorden, slechts de cognitieve interface naar de "vroege" perceptuele-stadia.⁴

Hoofdstuk 9 beschrijft het basis apparaat voor de perceptie zoals dat door ons gedeeltelijk in Soar en gedeeltelijk in Lisp geïmplementeerd is. De implementatie van dit basis apparaat werd bepaald door drie soorten constraints. In de eerste plaats constraints die de Soar en UTC theorie zelf leveren. De constraints uit deze eerste categorie zijn: (1) objecten kunnen alleen in de basisruimte van Soar terecht komen; (2) objecten worden *asynchroon* aan alle andere Soar processen in het werkgeheugen geplaatst waarbij zij oude objecten *destructief* overschrijven; (3) objecten worden alleen gered van destructieve overschrijving als Soar er op tijd *aandacht* aan besteedt door middel van een *aandachts-operator* en ze daarmee in het *interne model* van de wereld onderbrengt.

Het tweede type constraint betreft het *functionele* en het *perifere visuele veld*. Het functionele veld is een gebied van omstreeks 20 graden in het totale visuele veld waarbinnen objecten zonder oogbewegingen waargenomen kun-

⁴ Desondanks zijn er toch al een aantal Soar studies verschenen waarin blijkt dat door het goed kiezen van de menselijke perceptuele constraints zeer succesvolle modellen van vooral de interne aandacht te maken zijn. In navolging van deze studie modelleren wij ook de interne aandacht. Daarnaast hebben we de oogbewegingen gemodelleerd.

nen worden.⁵ Het functionele veld zoals in DRIVER geïmplementeerd, heeft een bufferfunctie voor tijdelijke objecten. Pas als er aan een object in deze buffer een aandachts-operator besteed is komt het object echt in Soar's werkgeheugen terecht.

Alles buiten dit functionele veld heet het perifere veld. De waargenomen kenmerken van de objecten zijn net toereikend om interrupties (voor bijvoorbeeld lateraal bewegende objecten) te genereren of de aansturing van de oogbewegingen te ondersteunen.

Het laatste type constraint heeft te maken met oog- en hoofdbewegingen. De belangrijkste constraints in deze categorie zijn: (1) gedurende een oogbeweging kunnen geen nieuwe objecten aan het werkgeheugen toegevoegd worden; (2) zowel data-driven als top-down control van oogbewegingen zijn mogelijk; (3) de spanning die staat op de oogspieren bij het draaien van de ogen ten opzichte van de oogkas is de belangrijkste parameter die bepaalt of het hoofd wel of niet gedraaid moet worden; en (4) oog- en hoofdbewegingen kosten tijd, afhankelijk van de snelheid en de hoek die overbrugd moet worden.

Het basisapparaat voor perceptie verenigt alle bovenstaande constraints in zich. Het basisapparaat in Lisp zorgt er voornamelijk voor dat, gegeven de positie en de kijkrichting van het oog, de potentiële waar te nemen objecten in de juiste visuele velden terechtkomen. Het basisapparaat in Soar bestaat in feite uit drie soorten operatoren die de hoofdrol spelen in visuele oriëntatie, te weten, de operatoren voor aandacht en operatoren voor oog- en hoofdbewegingen. Het belangrijkste wat in dit hoofdstuk besproken wordt is hoe deze operatoren gegenereerd en geselecteerd kunnen worden. Een discussiepunt daarbij is of we moeten kiezen voor een intelligente wijze van genereren van operatoren, zodat weinig selectiekennis nodig is, of dat er voor veel types objecten operatoren gegenereerd moeten worden zodat selectie heel belangrijk wordt.

Een interessant gegeven van onze implementatie is dat, omdat de ogen en het hoofd in dit model door het motor subsysteem aangestuurd worden, een 'beweeg-oog' operator wel een oogbeweging initieert maar niet wacht op de uitkomst. Dit betekent dat, hoewel er geen nieuwe informatie binnenkomt tijdens een oogbeweging, het Soar proces voor andere taken wel gewoon kan doorgaan.

Visuele oriëntatie in het verkeer (hoofdstuk 10)

Voortbouwend op het basis perceptieapparaat, beschrijft dit hoofdstuk de twee soorten regels die de visuele oriëntatie van DRIVER in het verkeer bepalen. De eerste soort zijn de zogenaamde standaard oriëntatieregels die in bijna alle situaties van toepassing zijn. Voorbeelden van deze regels zijn: bewegende

⁵ Voor de specialist op het gebied van perceptie mag dit wat veel lijken, maar het blijkt dat automobilisten de relevante objecten in dit gebied kunnen waarnemen zonder oogbewegingen.

objecten in het functionele veld hebben een hogere preferentie dan objecten in het perifere veld; binnen het functionele veld hebben bewegende objecten een hogere preferentie dan statische objecten in het functionele veld. Kenmerkend voor deze standaard regels is dat het *selectieregels* zijn voor operatoren die voornamelijk *data-driven* gegenereerd zijn.

De tweede soort regels zijn de situatie-specifieke regels die er, in geval van het afhandelen van een kruispunt, voor zorgen dat de standaard regels “overruled” worden door meer op kruispunten toegespitst gedrag. Een voorbeeld van zo’n regel is dat er, voordat de rem weer losgelaten wordt, een oogbewegingsoperator gegenereerd wordt met de hoogste preferentie. Kenmerken voor deze situatiespecifieke regels is dat ze een *top-down* karakter hebben en dat ze naast het selecteren van operators ook zelf operatoren genereren.

Deze twee soorten regels genereren tezamen het gedrag van de ervaren automobilisten van hoofdstuk 4. Nog veel interessanter is dat we met het basisperceptieapparaat en de visuele oriëntatieregels ook veel algemenere visuele oriëntatiefenomenen kunnen verklaren. Zo valt met DRIVER bijvoorbeeld goed te verklaren dat ervaren automobilisten meer, en eerder, kijken naar relevante objecten in hun omgeving; meer vertrouwen op informatie uit de perifere velden; beter intern de positie van bewegende objecten kunnen “updaten”; niet-normatieve of informele regels ontwikkelen naast de officiële regels; vaste strategieën gebruiken in het naderen van kruisingen; en hun visuele veld minder vaak wisselen in het afhandelen van kruisingen. Daarnaast kunnen we met DRIVER ook het “gekeken maar niet gezien” of de “gezien, maar niet gekeken” fenomeen verklaren.

Snelheidscontrole (hoofdstuk 11)

DRIVER is voortdurend bezig met het afstemmen van de snelheid op de omgeving. Op de rechte weg, ver van de kruising, is dit een vrij simpele neven-taak. Bij het naderen van een kruising wordt het een zware hoofdtak. Het vereist de integratie van de drie primaire processen die in de vorige hoofdstukken besproken werden: (1) het visuele oriëntatieproces dat er voor zorgt dat de voor de snelheidsbepaling relevante informatie uit de buitenwereld in het werkgeheugen terecht komt; (2) het cognitieve proces van het opbouwen van een mentaal model van de verkeersomgeving en het toepassen van de verkeersregels; (3) en tenslotte de motorische processen die er voor zorgen dat DRIVER de snelheidsbeslissingen goed uitvoert. In hoofdstuk 11 wordt op al deze processen ingegaan.

De zwaartepunt van dit hoofdstuk ligt in de eerste plaats bij de verschillende vormen van snelheidsperceptie waar een automobilist gebruik van kan maken en bij de manier waarop deze percepties in het werkgeheugen terechtkomen. Een tweede zwaartepunt betreft de opbouw van het *mentale model* van de verkeersomgeving op grond van de waarnemingen. Eén van problemen die we daarbij tegenkomen is dat Soar geen theorie van het *vergeten* bezit. Het blijkt

dat we zelf een vergeet-mechanisme moeten inbouwen om objecten automatisch uit het interne model te verwijderen. Een ander probleem is het mentale model consistent met de externe wereld te houden.

Sturen (hoofdstuk 12)

Ook het sturen op de rechte weg en het sturen door bochten vereisen de integratie van visuele, cognitieve en motorische processen. De informatieverwerkende cyclus die vereist is voor het sturen lijkt sterk op die voor de snelheidsregeling maar is wat simpeler uitgevoerd. In de eerste plaats omdat geen ingewikkelde modellen van de omgeving vereist zijn en in de tweede plaats omdat er voor het invoeren van koersdeviaties geen aandachtsoperatoren nodig zijn.

Een interessant punt in het hoofdstuk is hoe DRIVER bij het sturen in twee toestanden kan verkeren. In normale situaties is dit een *closed-loop error correction mode* met *permanente visuele feedback*. Als het visuele apparaat echter te zwaar belast is met andere taken kan DRIVER ook functioneren in een *open-loop error-neglecting mode*.

Navigatie (hoofdstuk 13)

Naast het afhandelen van kruisingen en het voorkomen van ongelukken heeft DRIVER ook nog de taak om op een bepaalde plaats in de gesimuleerde verkeerswereld aan te komen. Deze navigatietaak is de enige klassieke zoektaak in DRIVER waarbij diepe doelhiërarchieën zouden kunnen ontstaan. In hoofdstuk 2 werd echter aangetoond dat het bijna onmogelijk is om in een situatie waar veel interactie met de buitenwereld plaatsvindt, taken uit te voeren die diepe doelhiërarchieën vergen. We zagen dat dit vooral veroorzaakt wordt door Soar's default zoekregels. DRIVER maakt daarom gebruik van één van de alternatieve sets van default zoekregels die in hoofdstuk 3 besproken werden. Hierdoor wordt het mogelijk om probleemoplossen als het ware in korte *bursts* uit te voeren, terwijl toch zo snel mogelijk zo veel mogelijk geleerd wordt. Navigatie in DRIVER bestaat uit twee fasen. In de eerste fase wordt er, gegeven de kennis die DRIVER heeft van het netwerk van straten in zijn omgeving, een intern route-plan gebouwd en geleerd. De tweede fase is de uitvoering van dit plan. Om het leven van DRIVER nog wat meer te compliceren kunnen er echter onverwacht bepaalde wegen en kruispunten ineens geblokkeerd blijken. De gebruikte zoekregels blijken zo flexibel te zijn dat DRIVER ter plekke de bestaande route-plannen kan "afleren" en een nieuwe route bedenken.

Integratie en Multitasking (hoofdstuk 14)

In de voorgaande hoofdstukken werd steeds één rijtaak behandeld. Voor elke rijtaak zagen we een integratie van visuele, cognitieve en motorische processen. In hoofdstuk 14 komt aan de orde dat al deze rijtaken, inclusief de navigatietaak, ook nog eens gelijktijdig uitgevoerd moeten worden. Het multitas-

ken (of de meervoudige taakuitvoering) in DRIVER is net als in hoofdstuk 2 een kwestie van het wisselen van operatoren in de basis ruimte, echter nu niet met uniforme taakstructuren maar geheel afhankelijk van de taak. Ook gelijk aan hoofdstuk 2 is dat alle (actieve) taken in de basisruimte gerepresenteerd zijn. Welke taak in een bepaalde situatie de hoogste prioriteit heeft wordt in het algemeen afgehandeld door een kleine set van default regels. In sommige situaties, zoals bij de afhandeling van de kruispunten, blijken daarnaast een aantal situatie-specifieke regels nodig.

Het blijkt dat we met deze regels zowel de volgorde als de timing van de basishandelingen van de ervaren automobilisten uit hoofdstuk 4 kunnen reproduceren.

Een kernpunt in dit hoofdstuk is het gegeven dat Soar precies voldoende tijd lijkt te hebben om de belangrijkste taken in real-time uit te voeren. Daarmee lijkt de basis klok-tik van onze simulatie, namelijk Soar's elaboratiecyclus, met 30 milliseconden redelijk voorzichtig geschat te zijn. In de literatuur wordt deze tijd meestal nog wat korter genomen, zo rond de 10 tot 20 milliseconden. Er lijkt dus nog enige ruimte te zijn om de rijtaak nog wat complexer te maken, of andere taken toe te voegen.

Discussie (hoofdstuk 15)

De twee doelen van deze studie zijn de ontwikkeling van een psychologisch valide model van rijgedrag en een evaluatie van de praktische en theoretische geschiktheid van Soar voor het modelleren van complex dynamisch gedrag.

Wij achten de psychologisch validiteit van ons model hoog omdat de ontwikkeling van DRIVER stringent gestuurd is door een viertal psychologisch relevante constraints. In de eerste plaats hebben we gebruik gemaakt van de psychologische principes van menselijk informatieverwerking en probleemoplossen waarop Soar gebaseerd is en die uiteraard gevolg hebben voor taken die in Soar geprogrammeerd worden. DRIVER is voor een groot gedeelte gevormd door de eigenschappen zoals producties, probleemruimtes, sub-goaling en leren door chunking. In de tweede plaats hebben we ons gehouden aan de constraints die door Newell in zijn boek *Unified Theories of Cognition* aan de Soar constraints toegevoegd zijn. Ondanks alle moeilijkheden die ons dat in het praktische en theoretische vlak bezorgde, hebben we vastgehouden aan perceptie, aan motorische commando's en aan een aandachtsmechanisme in de basisruimte.

De derde groep constraints betreft de belangrijkste beperkingen van zowel de perceptuele als de motorische subsystemen. We hebben gebruik gemaakt van (1) gelimiteerde visuele velden, (2) aandachtsmechanismen die voorkomen dat Soar regels totale toegang tot de perceptuele input hebben, en (3) trage oog- en hoofdbewegingen en trage ledematen die niet sneller kunnen bewegen dan bij mensen. In het algemeen kunnen we stellen dat we DRIVER geen vaardigheden hebben gegeven die mensen niet hebben.

De vierde groep constraint betreft ons streven om empirische data te gebruiken bij het invullen maar ook het testen van DRIVER. De gehele ontwikkeling van DRIVER werd uiteindelijk gestuurd met als doel te komen tot realistisch gedrag, dat wil zeggen, een reproductie van het gedrag van de jonge ervaren automobilisten uit hoofdstuk 14.

De evaluatie van de theoretische geschiktheid van Soar voor het modelleren van complex dynamisch gedrag, waarbij rijgedrag als prototypisch voorbeeld van dit type gedrag is genomen, leidt niet tot een eenduidige uitspraak. We identificeren zowel een aantal zeer positieve aspecten van het gebruik van Soar als ook een aantal duidelijke gebreken. Positief is in de eerste plaats dat Soar krachtig en flexibel genoeg is om realistisch gedrag te genereren voor de basisrijtaken, de motorische taken en de visuele oriëntatie. Daarnaast stellen we zonder reserves dat Soar een fantastisch medium blijkt te zijn voor cognitief psychologen om te experimenteren met de taken die liggen in de "intermediate range", dat wil zeggen taken die plaatsvinden in het 1 tot 5 seconden bereik: Soar biedt de mogelijkheid om te werken met meerdere soorten doelen; vele vormen van parallelle informatieverwerking en multitasking. Het geeft inzicht in hoe gecontroleerd en automatisch verwerken en top-down en bottom up gestuurd gedrag in computationele termen gegoten kunnen worden. Tevens laat het zien hoe een computationeel systeem gebruik kan maken van het externe geheugen en hoe "situated action" kan ontstaan.

De lijst van beperkingen of gebreken van Soar is in dit hoofdstuk minstens even lang als het aantal positieve aspecten. Een aantal van deze gebreken is op zichzelf minder ernstig van aard omdat ze geen ingrijpende architecturale wijzigingen vereisen. Een voorbeeld is Soar's standaard zoekmethode die zeer inefficiënt bleek in het real-time multitasken. Een ander voorbeeld is Soar's probleem met het leren van externe interactie. In deze studie word echter al aangegeven hoe deze problemen aangepakt zouden kunnen worden zonder de Soar architectuur te wijzigen.

Een meer fundamenteel probleem heeft betrekking op Soar's werkgeheugen. Dit geheugen loopt in een dynamische situatie snel vol omdat Soar geen goede theorie van het "vergeten" heeft. Een probleem van dezelfde orde is dat de Soar architectuur niet goed met tijd om kan gaan: timingmechanismen ontbreken. Ook dat leverde in DRIVER een aantal praktische en theoretische problemen op.

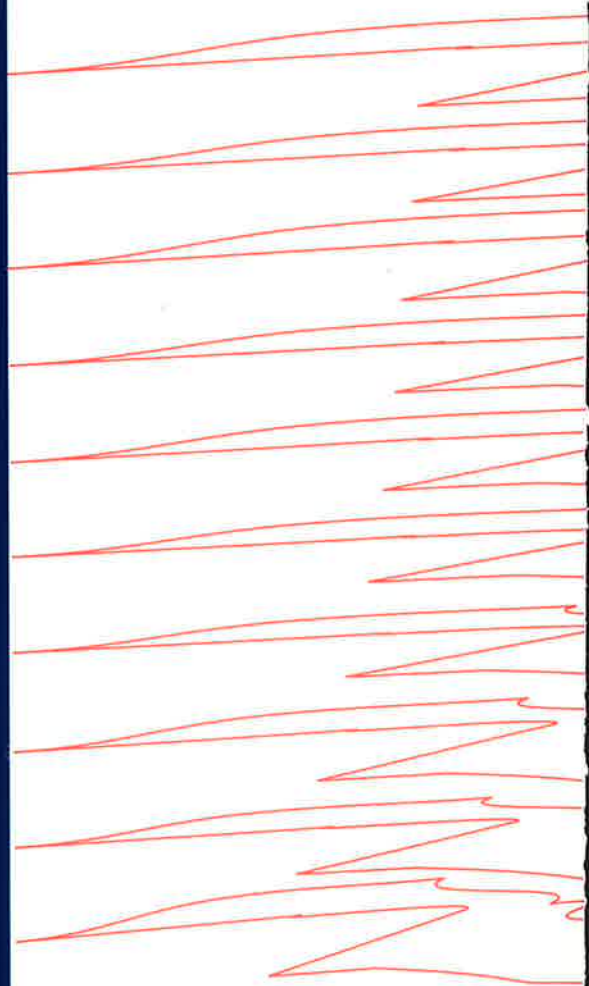
De grootste en meest fundamentele gebreken van Soar zijn al vaker genoemd en zij betreffen het ontbreken van een model van de perceptuele en motorische submodules. Soar biedt op dit moment niet meer dan de interface naar de modules en het wordt aan degene die modelleert overgelaten deze submodules zelf te simuleren.

Over de praktische geschiktheid van Soar valt veel minder te zeggen dan over de theoretische geschiktheid van Soar. Als men afziet van de gebruiksonvrien-

delijkheid van Soar en de problemen die er zijn om communiceren naar andere onderzoekers, is de praktische geschiktheid overwegend positief. De belangrijkste reden is dat Soar, in vergelijking met andere, door cognitief psychologen gebruikte hulpmiddelen, een volwaardige en universele programmeeromgeving biedt.

Er valt echter nog wel iets anders, iets meer algemeen, te zeggen over de geschiktheid van Soar en dat heeft te maken met de waarde van het modelleren zelf. In Posner's *Foundations of Cognitive Science* (1989) kijken Simon en Kaplan naar de rol en de waarde van computer simulaties van cognitief gedrag. Zij argumenteren dat de algemene waarde van het bouwen van een computationeel model er uit bestaat dat de zoektocht naar een zuinige set van mechanismen en de kennis om het model aan het werk te krijgen de onderzoeker een aantal belangrijke neveneffecten oplevert. In de eerste plaats ontdekt hij interessante, maar vooral onverwachte interacties tussen mechanismen, in de tweede plaats vindt hij nieuwe en onverwachte gedragspatronen die als natuurlijk uit zijn model voortvloeien, en in de derde plaats leert hij de juiste vragen stellen in het domein dat hij aanpakt.

Op grond van mijn persoonlijke ervaringen kan ik stellen dat Soar zeer geschikt is om deze zoektocht te ondersteunen en ik hoop dat de belangrijke neveneffecten van deze zoektocht voldoende in deze studie naar voren zijn gekomen.



ISBN 90-72125-50-9