J Sign Process Syst (2017) 87:157–175
DOI 10.1007/s11265-016-1193-7

Performance Analysis of Weakly-Consistent Scenario-Aware Dataflow Graphs

Marc Geilen[1] · Joachim Falk[2] · Christian Haubelt[3] · Twan Basten[1,4] · Bart Theelen[4] · Sander Stuijk[1]

Received: 24 March 2015 / Revised: 28 September 2016 / Accepted: 12 October 2016 / Published online: 9 November 2016

Abstract The timed dataflow model of computation is a useful performance analysis tool for electronic system level design automation and embedded software synthesis. Its determinism gives it strong analyzability properties. Its monotonic temporal behavior provides hard real-time guarantees on throughput and latency. It is expressive enough to cover a large class of applications and platforms. The trend however, in both embedded applications and their platforms is to become more dynamic, reaching the limits of what the model can express and analyze with tight performance guarantees. Scenario-aware dataflow (SADF) allows more dynamism to be expressed, introducing a controlled amount of non-determinism into the model to represent different scenarios of behavior. We investigate so-called weakly consistent graphs in which the scenario changes are not tightly coupled with periods of repetitive behavior of the static dataflow behavior in scenarios as in previous methods. We define the semantics of such graphs in terms of (max, +)-algebra and we introduce a method to analyze throughput using a generalization of (max, +)-automata. We show that weakly-consistent SADF generalizes many of the existing analyzable dynamic dataflow models, such as CSDF, PDF and CFDF and we present an algorithm to convert CSDF graphs to SADF.

Keywords Performance analysis · Synchronous dataflow · Dynamic dataflow · (max ,+)-algebra

✉ Marc Geilen
M.C.W.Geilen@tue.nl

[1] Department Electrical Engineering, Eindhoven University of Technology, Eindhoven, The Netherlands

[2] Department Computer Science, Friedrich-Alexander Universität Erlangen-Nürnberg, Erlangen, Germany

[3] Department Computer Science and Electrical Engineering, University of Rostock, Rostock, Germany

[4] Embedded Systems Innovation by TNO, Eindhoven, The Netherlands

1 Introduction

To develop concurrent embedded software applications and the platforms on which they execute, it is important to be able to efficiently assess whether or not performance requirements will be met. The parallel tasks and resource arbitrations create synchronization dependencies between tasks and time delays for processing and arbitration. Such behavior is captured well by performance models that build upon the (max, +)-semi-ring [1], such as Network Calculus [2], Real-Time Calculus [3], timed Petri-nets [4, 5], max-plus automata [6], and the timed dataflow models. An example is the MP3 decoder graph shown in Fig. 1. Figure 1a shows its structure. It has a File Reader component that reads the encoded audio stream from a source, an Entropy Decoder component that decompresses the bit-stream and the MP3 Synthesis filter banks that transform the encoded audio into samples that can be played by an analog-to-digital converter. Figure 1b shows the structure of the dataflow model of the decoder. The MP3 Synthesis component itself is refined into a complex dataflow graph, not shown in detail. Edges between tasks show dependencies that lead to synchronizations and scheduling constraints. The Huffman entropy decoder cannot start before the data is read from the file stream. Similarly, the dataflow graph models that the activities in the graph take time. This time

is a consequence of both the amount of work that needs to be done by a task as well as the processing speed due to the amount of resources from the platform that have been reserved to do it. The dataflow model will typically assume an execution time that is the worst case (or any upper bound) for the actual execution time, which may vary. The example is discussed in more detail below. An important feature of timed dataflow models for performance analysis is their determinacy. In the most restricted dataflow models, such as timed Synchronous Dataflow (SDF) [7, 8], task dependencies must be independent of input data, while some more dynamic models (for instance Dynamic Dataflow [9]) allow such data-dependent dependencies. The growing challenge is that the static structures of data dependencies and regular execution times with limited variation are becoming more and more exceptional as both applications and platforms are becoming more dynamic. Applications are becoming more dynamic, for instance, because of complex data reduction schemes, which introduce strong data-content dependencies. Hand-held devices need to support a wide range and diversity of communication protocols. More and more is handled in software by software-defined radio implementations. Novel cognitive radio protocols have strong adaptivity to environmental conditions. For MP3 compression, different parts of the audio, called frames, may be encoded using different methods. These methods cannot be accurately captured in a static dataflow model. Besides the application, also the platforms are becoming more dynamic. They need to dynamically handle various use-case scenarios of applications and use dynamic QoS management to match available resources with applications. Moreover, variability in the production process of integrated circuits makes that performance can vary from processor to processor, even on the same die, or vary over time with thermal changes or the aging process.

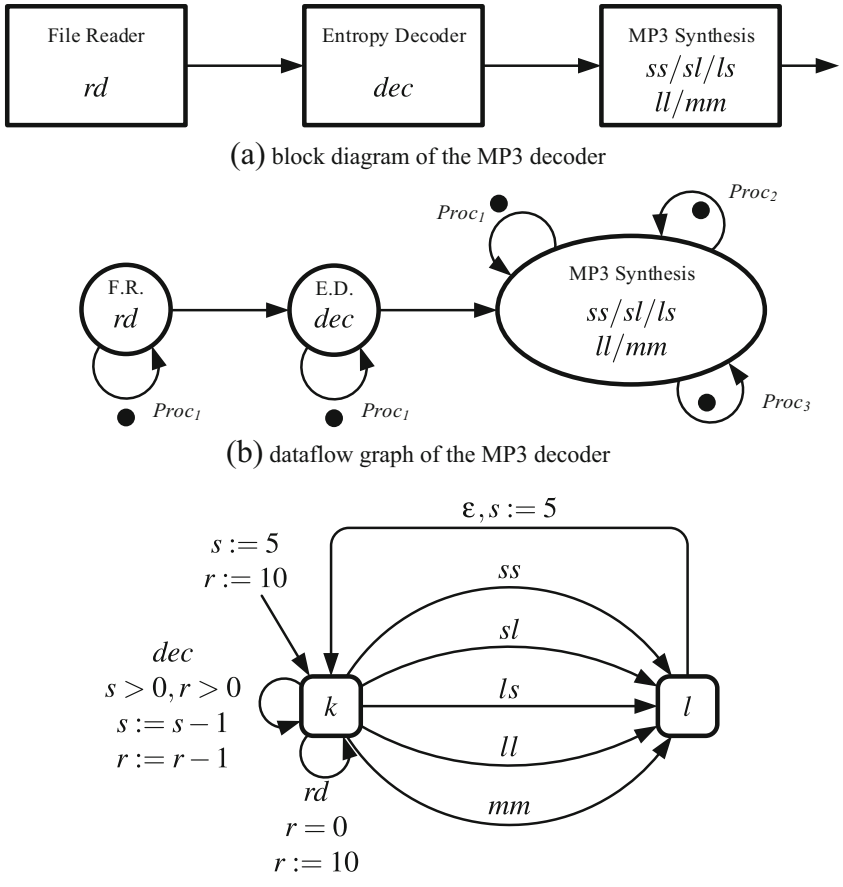
To deal with the increasing amount of dynamic behavior in applications and platforms, there is a growing need for performance models that can deal with more dynamic behavior and can still provide tight performance guarantees. The Scenario-Aware Dataflow (SADF) timed dataflow model [10–13] tries to maintain as much as possible of the determinacy of dataflow behavior, while introducing the possibility for non-deterministic variations in the form of *scenarios*. In MP3 decoding for instance, there are five individual coding schemes for audio frames. Each of these schemes can be represented accurately by a static dataflow graph, while the types of frames may occur non-deterministically (picked by the encoder depending on the sound content) in arbitrary orders. The SADF model and analysis techniques exploit the determinacy in behavior within a single scenario, while allowing for non-deterministic selection of the scenarios that occur. A crucial aspect is the concurrency among scenarios. Concurrent implementations of streaming applications are often pipelined. For the MP3 decoder this means that different frames in different scenarios may simultaneously be decoded. Yet, the analysis of scenario behaviors can be separately handled; they can be analyzed sequentially, despite their overlap in time when the application executes.

Analysis methods for the Synchronous Dataflow model based on spectral analysis techniques in the linear algebra on the $(\max, +)$-semi-ring have been introduced [12, 13]. These papers address the analysis of a particular subset of SADF models that are called *strongly consistent*, which means that every individual scenario behavior corresponds to a complete iteration of an SDF graph. For the analysis of the combination of non-deterministic sequences of scenarios which are modelled as SDF behaviors, the theory of $(\max, +)$-automata [6] has been used [13].

The contribution of this paper is to generalize the performance analysis approach of [13] to the case of [10], in which scenarios may occur at a finer granularity than complete SDF iterations. This class is called *weakly consistent* SADF, as opposed to the *strongly consistent* case, in which every scenario corresponds to a full iteration. For weakly consistent graphs, this is not necessarily the case, although in the long run, they need still be consistent to guarantee boundedness and freedom of deadlock. This generalization is important because it allows us to use non-determinism to model dataflow graphs that are not globally synchronous. This type of behavior is observed, for instance, in the MP3 example where the file reading front-end operates asynchronously from the sound decoding back-end, i.e., the amount of data that needs to be read to produce a new audio frame may vary, due to data-dependent levels of data reduction. We introduce methods to determine the worst-case throughput of a weakly consistent SADF and a compact state-space from which latency type of properties can be determined. The generalization also makes the Finite State Machine (FSM) based SADF model a proper generalization of the Cyclo-Static Dataflow (CSDF) [14] model and many other analyzable dynamic dataflow models. In CSDF, actors can have dynamically varying communication rates and varying execution times, but these variations are restricted to be deterministically varying in periodic patterns.

An example of the type of system that we are addressing is the MP3 decoder of Fig. 1. The decoder gets its input from some data source, e.g., a file, in an input buffer that is being refilled when it becomes empty. The input data is decompressed by an entropy decoder based on a Huffman code. Every now and then, depending on the input data, an audio frame is completed and can be processed by the synthesis filter banks. The blocks in the block diagram and the circles/ellipses in the dataflow model are

Figure 1 An MP3 decoder.



annotated with a name (in roman font) and with the scenarios in which these components may execute (in italics). The bigger ellipse in Fig. 1b, labelled 'MP3 Synthesis' is in fact a large dataflow graph, which depends on the audio frame type and consists of up to 25 separate actors taking hundreds of actor firings to complete the synthesis of the frame. The three tokens shown on the edges sticking out of the synthesis graph represent the dependencies carried over from one frame to the next. In this example, we have mapped the decoder onto three processors. The three tokens $Proc_i$ represent these resource dependencies. Each processor individually, but independently from each other, needs to complete a frame before starting the next frame. Note that in Fig. 1b there are three tokens labelled $Proc_1$; these, in fact, represent the same token modeling the processor 1 dependency, but in different scenarios. This is formalized in Section 6. The MP3 dataflow graph can operate in seven different scenarios: reading from file, performing entropy decoding, and five different types of audio frame synthesis. The finite state automaton (on infinite words) in Fig. 1c is our specification of the possible orders in which these scenarios can occur. We have used counters (r and s) to provide a more compact representation of the automaton, but it can easily be unfolded to a regular FSM. Edges are labeled with guards on the counters and assignments to counters, but, most importantly, with the scenario that is executed when the edge is taken. *dec* denotes execution of the entropy decoder, *rd* of the file reader and *ss*, *sl*, *ls*, *ll* and *mm* represent decoding of any of the five different types of audio frame. The file reader needs to be run exactly once every ten firings of the decoder and the decoder may non-deterministically produce a complete frame or not, but needs to complete a frame at least once in every five subsequent executions.

Interleaving of non-deterministic choices of the FSM (which, after unfolding the counters, has 65 states) with the execution of this dataflow graph (792 firings in the largest scenario, *ss*, as well as pipelining of multiple frames) can easily lead to state-space explosion in a naive state-space model. Moreover, in this example there are two independent, unsynchronized, sources of non-deterministic behavior. On the one hand, the entropy decoder occasionally produces a frame to be synthesized, but how often depends on the compression achieved for the particular piece of

music. On the other hand, the decompressed frames occur in different types which are decoded differently. Making this distinction leads to tighter estimates of the performance compared to synchronous dataflow models without scenarios. The results of this paper make it easier to model the individual scenarios independently and to model them more accurately. Forcing this behavior into a strongly consistent behavior, or even a static synchronous model may require abstractions that lead to less precise performance estimates. We illustrate this in the experimental section with a static abstraction of a channel equalizer model.

This paper is an extended version of [15]. It provides more details and experiments and it discusses the relationship with existing dynamic dataflow models including an algorithm to transform a CSDF graph into an SADF graph. The paper is organized as follows. The next section discusses related work. Section 3 introduces various preliminaries and notations. Section 4 introduces the SADF model and semantics by an example. Section 6 formalizes the general model and semantics. The new analysis methods are introduced in Section 7. Section 8 relates the weakly consistent SADF model to other existing analyzable dynamic dataflow models. Experimental evaluations are presented in Section 9. It is followed by conclusions in Section 10.

2 Related Work

Within the broad scope of performance analysis and schedulability analysis, we focus on a particular class of systems with repetitive behavior, processing on streams of data leading to strong dependencies between the individual tasks. The $(\max, +)$-semi-ring and its linear algebra [1] are very suitable to express the behavior and semantics of such models and it forms the foundation of many popular performance analysis models, such as Network Calculus [2], Real-Time Calculus [3], timed Petri-nets [4, 5], and the family of timed dataflow models of which SADF is a member.

Dataflow models of computation range from very static through more dynamic and (partially) analyzable models, to very dynamic, but also very hard to analyze models [11]. The static models include (homogeneous, cyclo-static) synchronous dataflow [8, 14, 16] and Computation Graphs [17]. Heterochronous Dataflow (HDF) [18] introduces dynamism by combining a finite state automaton with synchronous dataflow graphs in the individual states. The model is restricted to executing complete iterations per state transition of the automaton and the model does not have a timed version for performance analysis. Parameterized Synchronous Dataflow (PSDF) [19] considers a static structure of a dataflow graph, where one or more of the port rates are parameters. It is possible to find parameterized schedules and appropriate buffer sizes, but the possibilities for expressing dynamism are limited. In the variable rate dataflow model (VRDF) [20] communication rates may vary arbitrarily and are not necessarily constant over a complete iteration. Analysis methods for this model are efficient, but restricted to (conservative) buffer sizing under throughput constraints. More dynamic variations on dataflow models have been defined, but they introduce serious difficulties in the analysis. Examples include Dynamic Dataflow (DDF) and Boolean Dataflow (BDF) [21], which are models with data dependent firing rules. Their buffer sizing and throughput analysis problems are undecidable. Kahn Process Networks [22, 23] are also a form of dynamic dataflow model, but not based on actors with firing rules. (Lee and Matsikoudis discuss the relationship between Kahn's semantics and dataflow with firing rules [16].) Its dynamism and data-dependent behavior make the relevant analysis problems undecidable.

Network Calculus was introduced for the analysis of network processing on streams of network traffic. It abstracts concrete streams into worst-case bounds on amounts of traffic observed in any interval of a particular duration. Real-Time Calculus is a specialization of the Network Calculus approach to schedulability analysis of real-time embedded systems, in particular towards modeling of arbitration of shared resources and resource composition. The abstraction into the time-interval domain makes it harder to model dependencies, in particular circular dependencies, although extensions have been made to make handling such dependencies feasible or more accurate [24].

Petri-nets [25], with its many variants is also a model of computation that can express deterministic dataflow behavior, as well as non-deterministic behavior. Timed Marked Graphs [4] are in fact a class of Petri-nets equivalent to timed synchronous dataflow graphs. Determinism and consistency can be expressed as network invariants. There are no Petri-net analysis techniques that combine large aggregations of deterministic dataflow behavior with only the essential non-deterministic choices like in this paper.

An appropriate semantic domain for timed synchronous dataflow behavior is $(\max, +)$-linear algebra [1]. Spectral analysis in this linear algebra is intimately related to throughput and latency analysis. $(\max, +)$-automata [6] combine $(\max, +)$-linear behavior with non-deterministic choice. We use this combination to model non-deterministic scenario transitions. The Heaps of Pieces model [26, 27] is a specialization of $(\max, +)$-automata, used in literature to study the behavior of discrete event systems and in particular Safe Timed Petri-nets (see for instance [27]). It is important to observe the difference between the $(\max, +)$-automaton model and the Heaps of Pieces model, namely that Pieces cannot accurately capture a larger collection of

dataflow actor firings as a single Piece, since Pieces have fixed relative starting times (the 'lower contour') and fixed completion times (the 'upper contour'). As such, a Piece is 'rigid', while an iteration in a dataflow graph is more 'flexible' as it consists of a collection of independent actor firings, i.e., each actor firing can be modeled as a piece. Thus an iteration forms an aggregated stack of Pieces; the resulting upper contour may depend on the starting lower contour. By modelling only individual firings, the Heaps of Pieces model is too fine-grained to efficiently represent complex graphs.

We show how our analysis problem is ultimately mapped on a Maximum Cycle Ratio (MCR) problem on a directed multigraph, derived from a $(\max, +)$-automaton. A generalization of cycle ratio analysis is provided by spectral analysis of $(\max, +)$-linear systems [28, 29]. Spectral analysis gives not only the cycle ratio (eigenvalue), but also an eigenvector, which relates to the relative firing times of actors, or latency. A good overview and comparison of cycle mean, cycle ratio and spectral analysis methods can be found in [30].

In this paper we use synchronous dataflow graphs in the individual scenarios. However, we do not consider only complete iterations [13, 18], but allow partial iterations after which the graph may not return to the initial state. A special case of grouping firings results from clustering of SDF actors [31–33]. The result can be a quasi-statically scheduled system, in which the clustered actors can be modelled as scenarios of a weakly-consistent SADF. Hence, the proposed analysis can also be applied to such systems. Another work exploring this aspect is [34], which considers a modular implementation of SDF, where firings of an SDF iteration are grouped together. These may be individually scheduled depending on the presence of input data.

3 Preliminaries

We give brief introductions to SDF, the extension of dataflow with scenarios, the timing-semantics of SDF formulated in $(\max, +)$-algebra, and $(\max, +)$-automata as an analysis tool to the minimal extent required for this paper.

3.1 Synchronous Dataflow Graphs

Synchronous Dataflow Graphs are directed (multi-)graphs in which the nodes represent *actors*, entities that model computations or other events that take time, such as a computation task on a processor or resource arbitration delays. Actors perform their events or actions repeatedly. A single action is called a *firing* of the actor. The directed edges, connecting actors, are called *channels* and represent dependencies between actor firings. Dependencies in the model may have different origins in reality. They may be data dependencies, but they may also represent resource dependencies, for instance when an activity requires a resource that first needs to be released by another activity. Concrete dependencies are incarnated by *tokens* (sometimes also called *delays*) that are being communicated across the channels. In SDF, actors producing or consuming tokens can do so with constant *rates*. Each firing may consume or produce multiple tokens, but the number needs to remain constant across firings. Channels may initially already contain some tokens, which are called *initial tokens*.

Because of the constant rates with which tokens are communicated, actor firings occur in repetitive patterns called *iterations*. An iteration defines a (smallest, positive) number of firings for each actor which is such that the number of tokens on channels remains unchanged. From this invariant, it is clear that this pattern or iteration can be repeated indefinitely to obtain a *streaming* execution of the dataflow graph.

3.2 (max, +) semantics of SDF

Semantics of SDF graphs comes in two flavours. Some focus on functional behavior of actors and graphs. Others focus on the performance of SDF graphs to predict their throughput or latency. In this paper, we investigate the second kind, timed SDF [8]. SDF graphs can be translated into equivalent event graphs [1, 8], although this may involve a considerable increase in the size of the graph. From this it does follow however that the timing behavior of SDF graphs follows similar patterns as event graphs. In particular, their behavior is deterministic and eventually becomes periodic. This behavior can be captured efficiently by means of $(\max, +)$- algebra [1], a linear algebra based on the operations of maximum and addition.

An interesting feature of timed SDF graphs is that although the semantics assumes fixed, deterministic execution times and therefore has a deterministic behavior, it can faithfully capture systems in which the execution times are non-deterministic, yet bounded from above, by deterministic execution times. Throughput results of the deterministic SDF graph provide guaranteed lower bounds on the actual throughput of the system [1, 8]. The timing is encoded by *dater functions* which assign to tokens alive in the graph at a given state, a time stamp of their first occurrence. We illustrate this with an example in Section 4.

We now briefly introduce some notation related to $(\max, +)$-algebra (see [1] for background on $(\max, +)$-algebra). $(\max, +)$-algebra defines the operations of the maximum of numbers and addition over the set $\mathbb{R}^{-\infty} =$

$\mathbb{R} \cup \{-\infty\}$, the real numbers extended with a smallest element called $-\infty$. For readability we use the standard notation for the max and addition operations instead of the $\oplus$ and $\otimes$ notation mostly used in $(\max, +)$ literature. Moreover, for scalars x and y, $x \cdot y$ (with shorthand xy) denotes ordinary multiplication, not the $(\max, +) \otimes$ operator. The max and $+$ operators are defined as usual with the additional convention that $-\infty$ is the zero element of addition: $-\infty + x = x + -\infty = -\infty$ and the unit element of max, $\max(-\infty, x) = \max(x, -\infty) = x$. $(\max, +)$ is a *linear algebra*: $x + \max(y, z) = \max(x + y, x + z)$.

The algebra is additionally extended to a linear algebra of matrices and vectors in the usual way. Note that any matrix-vector multiplication in this paper denotes a $(\max, +)$ matrix-vector multiplication and not a traditional matrix-vector multiplication. For a matrix $\mathbf{M}$ and vector x, we use $\mathbf{M}x$ to denote the $(\max, +)$ matrix multiplication. If $a = [a_i]$ and $b = [b_i]$ with $a_i, b_i \in \mathbb{R}^{-\infty}$ are vectors of size k, then we write $a \preceq b$ to denote that for every $1 \leq i \leq k$, $a_i \leq b_i$. With a a vector and c a scalar, we use $c + a$ or $a + c$ to denote a vector with entries identical to the entries of a with c added to each of them: $c + a = a + c = [a_i + c]$. We use 0 to denote a vector with all zero-valued entries. The size of 0 is derived from the context. We use $\max(a, b)$, defined as $[\max(a_i, b_i)]$ as a max operator on vectors and $a + b$, defined as $[a_i + b_i]$ as addition of vectors. $||a||$ denotes a vector norm, defined as: $||a|| = \max_i a_i$, i.e., the maximum element. It is a proper vector norm in the algebra, because (i) $||a|| = -\infty$ iff $a_i = -\infty$ for all i; (ii) $||c + a|| = c + ||a||$; (iii) $||\max(a, b)|| \leq \max(||a||, ||b||)$. For a vector a with $||a|| > -\infty$, we use a^{norm} to denote $a - ||a||$, the normalized vector a, so that $||a^{norm}|| = 0$. We use a^T to denote transposition of a vector, to turn a column vector into a row vector and vice versa. An inner product is defined as follows: $a^T b := \max_i(a_i + b_i)$. If matrix $\mathbf{M} = [m_j]$ (i.e., has column vectors m_j), then $\mathbf{M}x := \max_j(m_j + x)$ and $\mathbf{M}^T x := [m_j^T x]$. It is easy to verify that also matrix multiplication is linear: $\mathbf{M}(\max(x, y)) = \max(\mathbf{M}x, \mathbf{M}y)$ and $\mathbf{M}(c + x) = c + \mathbf{M}x$. Moreover, matrix multiplication is *monotone*: if $x \preceq y$, then $\mathbf{M}x \preceq \mathbf{M}y$.

3.3 Scenario-Aware Dataflow Graphs

Scenario-Aware Dataflow graphs [10] are a variant of dataflow models that try to occupy a sweet spot in the trade-off between analyzability and expressiveness [11], in particular to express more dynamic behavior. It can be seen as a timed extension of the Heterochronous Dataflow (HDF) model [18]. It combines Synchronous Dataflow behavior with finite state machines (FSMs). However, it extends the HDF model with time and optionally stochastic behavior, by using Markov chains instead of FSMs. It allows the FSM transitions to occur not only at the borders of complete iterations of the SDF behaviors, but also at intermediary stages of the data flow. Note that some of the earlier analysis methods have also adopted the constraint that FSM transitions may only occur at iteration boundaries [11–13]; this paper frees the analysis from this restriction. An important element of the timed model is that even though the FSM transitions occur in-between pieces of deterministic dataflow behavior, this does *not* mean that such pieces cannot overlap in time. They can be pipelined. If that were not allowed, no tight performance predictions could be made. In this paper, we exploit the fact that although they are pipelined, they are independent and their analysis can be sequentialized. An important strength of the (timed) synchronous dataflow model is its determinism. An important goal of the SADF model is to preserve the benefits of the deterministic behavior *within* scenarios for efficient analysis, despite the addition of non-deterministic scenario changes.

The semantics of SADF can be captured by a combination of classical FSM semantics and $(\max, +)$-based semantics of the scenarios of determinate synchronous dataflow behavior. The precise semantics is worked out in more detail in Section 6. It uses a combination of state machines and $(\max, +)$-matrix multiplication that is called a $(\max, +)$-*automaton* and is briefly introduced in the next subsection.

3.4 (max, +)-automata

A $(\max, +)$-automaton [6] is a generalization of a regular finite state automaton with time durations on its edges. The representation of automata using their characteristic matrices is the most convenient for our purposes. We use it instead of the more common, equivalent, representation as states and transitions. It is defined as a tuple $\mathcal{A} = (\Sigma, \mathbf{M}, \mathcal{M})$, of a finite set Σ of scenarios, a mapping $\mathbf{M}$, which assigns to every scenario $\sigma \in \Sigma$ a $(\max, +)$-matrix $\mathbf{M}(\sigma)$ and a morphism $\mathcal{M}$ on finite sequences of scenarios, mapping such sequences to a $(\max, +)$-matrix such that

$$\mathcal{M}(\sigma_1 \ldots \sigma_k) = \mathbf{M}(\sigma_k) \ldots \mathbf{M}(\sigma_1).$$

For a given finite sequence of scenarios, the automaton defines the *completion time* as follows:

$$\mathcal{A}(\sigma_1 \ldots \sigma_k) = ||\mathcal{M}(\sigma_1 \ldots \sigma_k)0|| = ||\mathbf{M}(\sigma_k) \ldots \mathbf{M}(\sigma_1)0||.$$

Then, $\mathcal{M}(\bar{\sigma})0$ captures the production times of the tokens of the SADF after the sequence $\bar{\sigma}$ of scenarios. The time when the final token is produced is captured by taking the $(\max, +)$-norm (maximum entry) of the resulting vector. We are often interested in the worst-case throughput for any possible sequence of scenarios. Gaubert shows [6] how

this maximum growth rate of the completion time (minimum throughput) can be computed as the maximum cycle mean of the equivalent timed event graph [1] of the matrix $\mathbf{M} = \max_{\sigma\in\Sigma} \mathbf{M}(\sigma)$. It also shows how, given an (infinite) regular sub language of Σ^*, the set of all finite scenario sequences, the maximum growth rate can be determined using a classical product automaton construction. Its worst-case behavior can then be analyzed using spectral analysis of a corresponding matrix, or if we are only interested in throughput by maximum cycle mean analysis directly on the automaton graph (R, ϵ), where the nodes R contain a node for each row/column of the matrices in $\mathbf{M}$. The labeled edges $\epsilon \subseteq R \times \mathbb{R}^{-\infty} \times R$ contain an edge (k, v, m) for each $v = \mathbf{M}(m, k) > -\infty$ [6].

For this paper, we need to generalize this concept. In particular, instead of studying the average growth rate *per step* of the automaton, we study the ratio of the growth rate of the completion time relative to another quantity of progress expressed as the sum of a certain benefit or *reward* per scenario. This amounts to application of the *generalized spectral problem* [28] to (max, +)-automata. In this case, the worst-case throughput can be determined as an MCR of the automaton where edges have two labellings, delays and rewards. We also generalize the model by associating non-square matrices $\mathbf{M}(\sigma)$ with scenarios. We assume that we use a specification of legal scenario sequences that is consistent with the matrix sizes, i.e., such that the morphism $\mathcal{M}$ is well-defined.

4 A Semantic Model of Weakly Consistent SADF

In strongly consistent SADF, every transition of the FSM corresponds to a *full iteration* of the SDF graph for the particular scenario. It is therefore a piece of behavior that can be repeated forever. Moreover, switches between scenarios are always possible in such states of the dataflow graphs, because their initial states are identical. *State* in this case refers to the tokens present in the graph, as the graph's actors and channels may be different in different scenarios, but the initial tokens are found in all scenarios. Specifically 'state' of those tokens refers to a dater function on the tokens, time stamps indicating the time of their availability to be used (consumed).

Although for many applications of this type of scenario-aware dataflow behavior, scenarios align well with pieces of behavior that constitute iterations, there exist also situations in which it is convenient to consider units at smaller granularity, as explained above. We generalize the model to allow for edges of the FSM to correspond to arbitrary (but fixed) collections of firings. In contrast with the strongly consistent case, the starting and ending state of a scenario dataflow graph are not necessarily the same. Such weakly consistent graphs can still be fully and precisely characterized by a (max, +)-matrix, which is not necessarily square in this case.

5 Example

As an example we first establish the (max, +)-automaton model of the running example graph shown in Fig. 2. We give a precise general definition in the next section. The initial state of the FSM is k with an edge to itself labelled with the scenario α, which has one firing of P and one firing of Q in the 'mode' of scenario a in which the execution times of both actors P and Q are 2 and the output rate of actor Q on the channel to R is 0. This combination of two firings has no net effect on the distribution of the three tokens on positions 1, 2, and 4 (indicated by the labels inside the tokens). The relevant part of the starting state, for these two firings, is defined by the two tokens 1 and 2 in the figure, with time stamps t_1 and t_2. This is captured in a (max, +)-vector $[t_1 \;\; t_2]^T$. In scenario α P needs to fire before Q_a (a firing of actor Q in mode a) and consumes both tokens. Hence, its earliest starting time is $\max(t_1, t_2)$. The firing takes 2 time units and completes at time $\max(t_1, t_2) + 2$, which in (max, +)-sum-of-product form is equal to $\max(t_1 + 2, t_2 + 2)$, or in vector inner-product notation: $[2 \;\; 2] \cdot [t_1 \;\; t_2]^T$. This is the time stamp of the new token produced at position 1. Next, Q_a fires and consumes the token just produced by P on the edge from P to Q. Its firing takes also 2 units of time and completes at $\max(t_1+4, t_2+4)$, or: $[4 \;\; 4] \cdot [t_1 \;\; t_2]^T$. At this time the token at position 2 is reproduced. Combining the two symbolic states into a matrix-vector equation, we get the following relation between the starting state vector and the end state vector.

$$\begin{bmatrix} t_1' \\ t_2' \end{bmatrix} = \begin{bmatrix} 2 & 2 \\ 4 & 4 \end{bmatrix} \begin{bmatrix} t_1 \\ t_2 \end{bmatrix}$$

This matrix characterizes the collective effect of the (two) firings in this scenario. Note however that, considering the whole graph, there is a third token present at position 4 that is neither consumed nor produced in this scenario. It is however part of the state and needs to be accounted for in state and matrix. It is easy to see that this is done by adding the following row and column.

$$\begin{bmatrix} t_1' \\ t_2' \\ t_4' \end{bmatrix} = \begin{bmatrix} 2 & 2 & -\infty \\ 4 & 4 & -\infty \\ -\infty & -\infty & 0 \end{bmatrix} \begin{bmatrix} t_1 \\ t_2 \\ t_4 \end{bmatrix}$$

The transition $\alpha = P\,Q_a$ in the FSM can be considered a complete iteration in the sense that it has no net effect on the

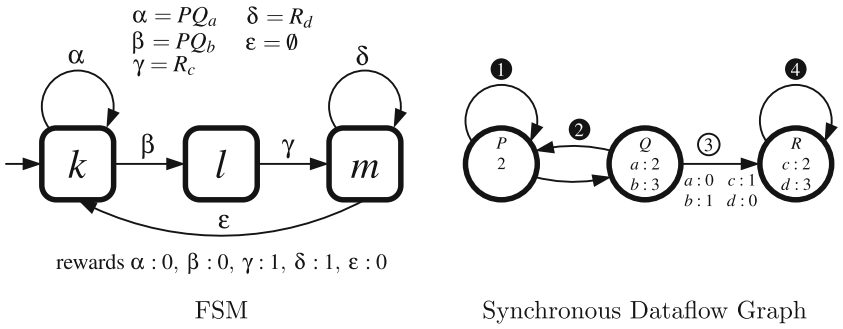


Figure 2 An example weakly-consistent SADF.

distribution of the tokens, although actor R is not involved in the firings. This is not true for all transitions however. Another transition, from state k to state l, is labeled with scenario β consisting of the firings P, Q_b. These firings produce an additional token on position 3 on the channel from Q to R in the SDF graph. The combined effect can be represented by the following matrix vector equation.

$$\begin{bmatrix} t'_1 \\ t'_2 \\ t'_3 \\ t'_4 \end{bmatrix} = \begin{bmatrix} 2 & 2 & -\infty \\ 5 & 5 & -\infty \\ 5 & 5 & -\infty \\ -\infty & -\infty & 0 \end{bmatrix} \begin{bmatrix} t_1 \\ t_2 \\ t_4 \end{bmatrix}$$

Note that the matrix is no longer square (4 by 3), because the end state has four tokens while the starting state has three.

After this, a single firing R_c will take place in scenario γ, moving from state l to m. This consumes tokens 3 and 4 and produces token 4, according to $t'_4 = \max(t_3 + 2, t_4 + 2)$. Token 3 disappears in this process and tokens 1 and 2 remain untouched. The full matrix thus has size 3 by 4.

$$\begin{bmatrix} t'_1 \\ t'_2 \\ t'_4 \end{bmatrix} = \begin{bmatrix} 0 & -\infty & -\infty & -\infty \\ -\infty & 0 & -\infty & -\infty \\ -\infty & -\infty & 2 & 2 \end{bmatrix} \begin{bmatrix} t_1 \\ t_2 \\ t_3 \\ t_4 \end{bmatrix}$$

From state m, an arbitrary number of scenarios δ are possible, a single firing of R_d, each of which involves only token 4, according to $t'_4 = t_4 + 3$. The full matrix equation is as follows.

$$\begin{bmatrix} t'_1 \\ t'_2 \\ t'_4 \end{bmatrix} = \begin{bmatrix} 0 & -\infty & -\infty \\ -\infty & 0 & -\infty \\ -\infty & -\infty & 3 \end{bmatrix} \begin{bmatrix} t_1 \\ t_2 \\ t_4 \end{bmatrix}$$

At some point,[1] a transition ϵ is taken, back to state k. It is labelled with an empty set of firings and therefore leaves all tokens at rest. The matrix representation is an identity matrix. Note that every *cycle* in the FSM constitutes a collection of firings that has no net effect on the token distribution, i.e., is consistent. This makes the graph *weakly consistent*. As another a-priori sanity check on the graph, it can be verified by a straightforward reachability analysis whether or not a graph is deadlock free.

Equipped with the scenario matrices that characterise the effect on a system state of individual scenarios, we can determine the evolution of the system state for any given scenario sequence. Assume we have the following sequence $\alpha\alpha\beta\gamma\delta\delta\varepsilon\alpha\beta\gamma$ and the initial state $t_0 = [0\ \ 0\ \ 0]^T$ is such that all tokens are present at time $t = 0$. t_1, the state after the first scenario α is: $t_1 = \mathbf{M}(\alpha)t_0 = [2\ \ 4\ \ 0]^T$. Continuing, the sequence of states evolves as follows.

$$\begin{aligned}
t_1 &= \mathbf{M}(\alpha)t_0 &&= [2\ \ 4\ \ 0]^T \\
t_2 &= \mathbf{M}(\alpha)t_1 &&= [6\ \ 8\ \ 0]^T \\
t_3 &= \mathbf{M}(\beta)t_2 &&= [10\ \ 13\ \ 13\ \ 0]^T \\
t_4 &= \mathbf{M}(\gamma)t_3 &&= [10\ \ 13\ \ 15]^T \\
t_5 &= \mathbf{M}(\delta)t_4 &&= [10\ \ 13\ \ 18]^T \\
t_6 &= \mathbf{M}(\delta)t_5 &&= [10\ \ 13\ \ 21]^T \\
t_7 &= \mathbf{M}(\varepsilon)t_6 &&= [10\ \ 13\ \ 21]^T \\
&\ldots
\end{aligned}$$

This sequence is illustrated in Fig. 3. The horizontal axis shows time and the vertical axis the four tokens in the graph. Every scenario takes a starting vector, represented by the connected tokens at their given time stamps, to a final vector according to multiplication with the scenario matrix. The corresponding actor firings are shown in a Gantt chart below. The shading of the actor firings matches the scenarios in which they occur.

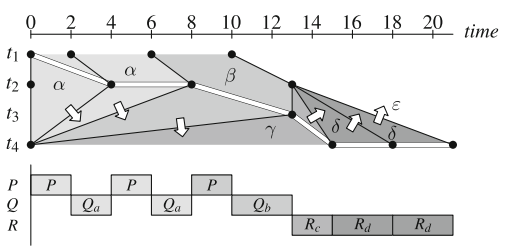


Figure 3 Execution of a sequence of scenarios.

[1] Note that we could add Büchi acceptance conditions to the automaton to enforce that progress is made eventually. However, such requirements typically have no impact on the worst-case performance.

6 Model and Semantics

We now formalize the weakly consistent SADF model. We make precise how we specify the graph and what we mean by its throughput. A weakly consistent SADF graph is defined by a tuple $(\Sigma, G, \rho, i, f, \pi, \mathcal{A})$. It has a finite set Σ of scenarios and every scenario $\sigma \in \Sigma$ has an associated SDF graph $G(\sigma)$ and a partial repetition vector $\rho(\sigma)$, which maps every actor of $G(\sigma)$ to a non-negative number specifying how often the actor fires in the scenario. In contrast with the habit in SDF analysis, this *partial* repetition vector does not need to be a multiple of the usual repetition vector of an SDF graph [7]. The graph $G(\sigma)$ has a collection of $i(\sigma) \in \mathbb{N}$ initial tokens, which we assume to be indexed $0 \leq n < i(\sigma)$. (Note that in Fig. 2 we have used labels 1-4 in tokens to refer to their location in the graph. These are not to be confused with the indices introduced here.) After execution of the partial repetition vector, the graph $G(\sigma)$ has a collection of $f(\sigma) \in \mathbb{N}$ 'final' tokens, which are indexed $0 \leq n < f(\sigma)$. (f can be determined from i, G and ρ, but it is convenient to make it explicit.) We use the $(\max, +)$-semantics of the SDF graphs [1] to associate with every graph $G(\sigma)$, a $(\max, +)$-matrix $\mathbf{M}(G(\sigma)) \in (\mathbb{R}^{-\infty})^{f(\sigma)\times i(\sigma)}$, or in short $\mathbf{M}(\sigma)$, that precisely characterizes the relationship between the time stamps of the initial and final tokens in the graph in that scenario as illustrated in Section 4. The FSM $\mathcal{A}$ is a tuple (Q, q_0, δ) with a set Q of states, an initial state q_0 and a labelled transition relation $\delta \subseteq Q \times \Sigma \times Q$. The scenario labels in the edges must be consistent in the sense that for any state $q \in Q$, any incoming edge labelled with scenario σ_1 and outgoing edge labelled with scenario σ_2, $f(\sigma_1) = i(\sigma_2)$, i.e., the number of final and initial tokens of subsequent scenarios must match. (We implicitly assume the tokens with the same index to be coupled, its time stamp at the end of scenario σ_1 is the initial time stamp for scenario σ_2.) We denote this number of tokens for a state q: $n(q)$. The FSM defines infinite sequences of scenarios in the usual way. $\mathcal{A}$ accepts the sequence $\bar{\sigma}$ of scenarios if and only if $\bar{\sigma}$ is in the language $\mathcal{L}(\mathcal{A})$ of the FSM, i.e., there exists a sequence $\bar{q}$ of states such that $\bar{q}(0) = q_0$ and for every $n \geq 0$, there exists an edge $(\bar{q}(n), \bar{\sigma}(n), \bar{q}(n+1)) \in \delta$. With sequence $\bar{\sigma}$, we associate the timing behavior, a sequence of $(\max, +)$-vectors, such that $t_0 = 0$ and for all $n \geq 0$, $t_{n+1} = \mathbf{M}(\bar{\sigma}(n))t_n$. We can now clearly recognise the structure of a $(\max, +)$-automaton.

It is important to recall that, as is common in the timed dataflow performance analysis [8], it may be assumed that the (per scenario) constant execution times given in the model are in fact upper bounds for the real system behavior and may in reality be non-deterministically smaller. Execution times may vary in a realization due to variations in workload caused by the concrete data being processed, or by influences from its environment, for instance the amount of interference from arbitration of shared resources and other tasks in the system. Monotonicity of timing behavior in dataflow graphs (and hence also in SADF graphs) which follows immediately from monotonicity of the $(\max, +)$-operators, ensures that performance guarantees derived for the model are in fact also guaranteed for such implementations [35].

For synchronous dataflow analysis, it is common to quantify throughput by measuring the number of iterations completed per time unit. In our case, it depends on the model how much actual, 'real-world' progress is made per scenario. We therefore assume that we explicitly quantify the amount of progress per scenario. For instance, for the example graph, we may be primarily interested in the number of firings of actor R. In this case the progress is 1 for scenarios γ and δ and 0 for any other scenario. For the MP3 example we may count the number of completed audio frames, by assigning progress of 1 (frame) to the scenarios *ss*, *sl*, *ls*, *ll* and *mm*, and 0 to the others. In general, we define a *reward* function $\pi : \Sigma \to \mathbb{R}^{\geq 0}$, which quantifies the amount of progress per scenario σ as $\pi(\sigma)$. The throughput obtained from a scenario sequence $\bar{\sigma}$ can hence be defined as follows.

$$\tau(\bar{\sigma}) = \lim_{k\to\infty} \sup \frac{\sum_{n=0}^{k-1} \pi(\bar{\sigma}(n))}{||t_k||}$$

I.e., throughput is defined as the average amount of progress per unit of time. The analysis question we answer in this paper is to determine the worst-case throughput of an SADF graph:

$$\tau = \inf_{\bar{\sigma}\in\mathcal{L}(\mathcal{A})} \tau(\bar{\sigma}).$$

We can define an explicit state space semantics of the model. The states consist of pairs (q, t) consisting of a state $q \in Q$ of the FSM and a *normalized* vector t. The initial state is $(q_0, 0)$, having the initial state of the FSM and the zero-vector for token time stamps. The transitions of the state-space are determined by the scenarios allowed by the FSM state. The FSM moves to the new state and the time-stamp vector is updated according to the scenario behavior and then normalized to keep the state space finite and capture only the relevant information, the relative differences between the time stamps. For a state (q, t) in the state space, consider every edge (q, σ, q') in the FSM. Then the state space has a labelled transition

$$\left((q, t), ||u|| - ||t||, \pi(\sigma), (q', u^{norm})\right),$$

where $u = \mathbf{M}(\sigma)t$. The transitions in the state space are decorated with two labels. The first is the amount of time progress $||u|| - ||t||$, i.e., the amount of time difference between the normalized vectors. The second is the progress reward, $\pi(\sigma)$, that is associated with the scenario of the FSM transition.

7 Throughput Analysis

To determine the infimum of the throughput values for all possible scenario sequences on the graph, we need to find the worst-case scenario sequence. In any scenario sequence, both time and total reward progress with the scenarios being executed. Progress of time is measured as the (max, +)-norm of the state vectors t_k, the maximum element of the vector. Since $t_{k+1} = \mathbf{M}(\bar{\sigma}(k))t_k$, every element of t_{k+1} is determined by some element of t_k and offset by the corresponding dependency in the matrix. This element in t_k can in turn be traced back to a single element in t_{k-1} and so forth back to t_0. In Fig. 3 these critical dependencies are illustrated by the thicker white line. The norm of state vector t_7 is equal to 21 because of token t_4 (its fourth element). Token t_4 in t_7 is determined by token t_4 in vector t_6, which in turn is determined by t_4 in t_5, t_4 in t_4, t_3 in t_3, t_2 in t_2, t_2 in t_1 and finally $t_1 = 0$ in t_0. Therefore, to study the relation between time progress and scenario sequences, we need not look at complete vectors, but we can concentrate on individual elements (initial / final tokens) and their individual dependencies as expressed by the entries in the matrices.

Figure 4 shows a structure which encodes these dependencies for the example of Fig. 2. The nodes in this graph represent the initial/final tokens (horizontally) in each of the states of the FSM (vertically). For every edge of the FSM, we take the matrix $\mathbf{M}(\sigma)$, with σ the label of the edge, and for every finite (non $-\infty$) element in the matrix we draw an edge between the corresponding initial/final tokens and label it with the value of that element and with the reward $\pi(\sigma)$ of σ. For clarity we have labeled it with the scenario σ itself in the figure instead of the reward. The dashed boxes are not actual nodes. They merely illustrate where the nodes for token t_3 would have been, had this token existed in that state, in order to make the structure more clear. The precise definition of the (max, +)-automaton corresponding to the SADF graph is as follows.

Definition 1 For an SADF graph $(\Sigma, G, \rho, i, f, \pi, \mathcal{A})$, the analysis (max, +)-automaton is defined in the form of a graph (R, E) with vertices R and edges E, as follows.

- $R = \{(q, i) \mid q \in Q, 1 \leq i \leq n(q)\}$
- $E = \{((q_1, i), \mathbf{M}(\sigma)_{i,j}, \pi(\sigma), (q_2, j)) \mid (q_1, \sigma, q_2) \in \delta, 1 \leq i \leq n(q_1), 1 \leq j \leq n(q_2), \mathbf{M}(\sigma)_{i,j} \neq -\infty\}$

The worst-case throughput of the graph can be determined from a *maximum cycle ratio* analysis of the corresponding (max, +)-automaton, i.e., by finding the cycle in the graph with the worst-case ratio of total reward over time progress.

Theorem 1 *Let* $\mathcal{G} = (\Sigma, G, \rho, i, f, \pi, \mathcal{A})$ *be an SADF graph and* (R, E) *be the corresponding* (max, +)*-automaton graph, then* $\inf_{\bar{\sigma} \in \mathcal{L}(\mathcal{A})} \tau(\bar{\sigma}) = \mathrm{MCR}(R, E)$ *is the worst-case throughput of* $\mathcal{G}$.

Proof (Sketch) Analogous to the results of Section VI of Gaubert [6], in particular Proposition 2, but generalized to the case of time progress divided by reward progress. □

Note that the example graph (Fig. 4) has cycles of zero reward (α self-loops) and hence the worst-case throughput is zero. Indeed, actor Q may never produce any output to R in which case, R will never fire. In a refined model we may limit the number of firings of Q in mode a to two, before it must fire in mode b. And similarly we bound the number of firings of actor R in mode d to three. This can be modeled by introducing extra states in the FSM that count the number of firings. Note that the behavior of this example is characteristic for the asynchronous file reader front-end of the MP3 example. After a predictable number of reads it must have data available for decoding. This leads to the automaton shown in Fig. 5, in which counters have been used to enforce the above constraints. Counter x limits the number of subsequent α's to two, counter y the δ's to three. After unfolding the counters to a plain FSM, it has 8 states.

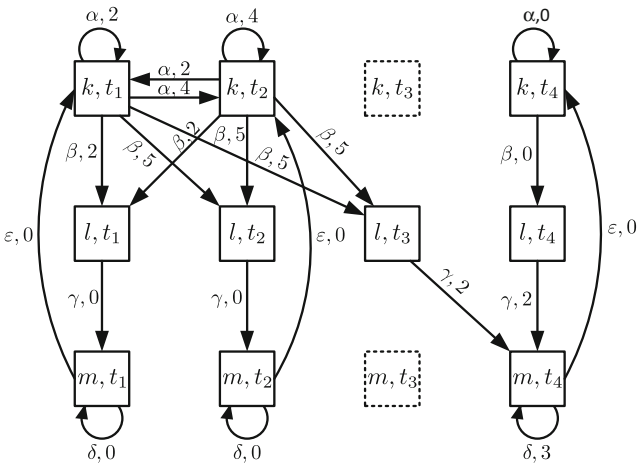


Figure 4 (max, +)-automaton of the example model.

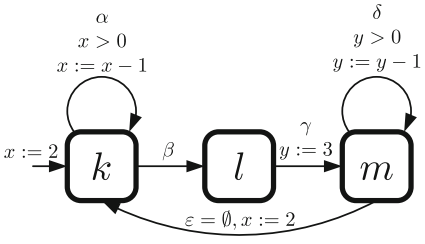


Figure 5 Refined example SADF.

The corresponding (max, +)-automaton then becomes the graph depicted in Fig. 6.

We have analyzed the graph of Fig. 5 using the conversion and MCR analysis. The worst-case throughput is $\frac{1}{13}$. The critical cycle also tells us the scenario sequence that leads to this worst-case performance. In this case, it is determined to be the cycle of edges that is shown in bold in Fig. 6. It corresponds to a repetition of the scenarios $\alpha\alpha\beta\gamma\varepsilon$, which takes 13 units of time and involves only one firing of R. The critical dependencies involve only t_2 in this case.

8 Relation to Other Dynamic Dataflow Models

An important advantage of the relaxation of the consistency constraint in the SADF model is that it makes the model a proper generalization of the popular Cyclo-Static Dataflow model (CSDF) [14]. As such it is an analyzable dataflow model that can serve as a semantic basis for many of the popular analyzable dataflow models, including the static HSDF and SDF [7], and dynamic models such as HDF [18], CSDF [14] and parametric models such as Parameterized Dataflow (PDF) [19] and Schedulable Parametric Dataflow (SPDF) [36], (Structured) Variable-Rate Phased Dataflow ((S)VRPDF) [37, 38] and Mode Controlled Data Flow (MCDF) [39]. Parametric execution time analysis [40–42] can also straightforwardly be generalized to weakly consistent FSM-SADF (WC-FSM-SADF), because the concept of convex throughput regions is preserved. In this section we show in detail how CSDF graphs can be represented as weakly consistent SADF. We then discuss the relationship of SADF to a number of existing dataflow models.

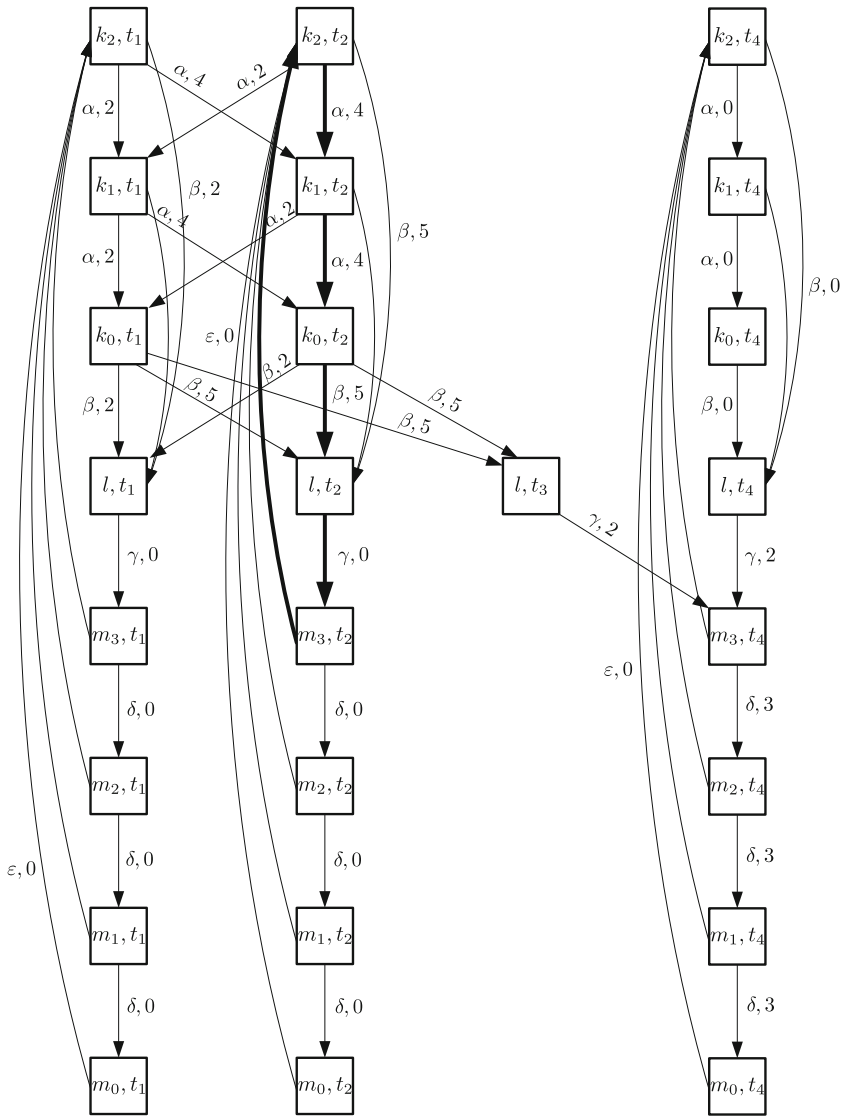
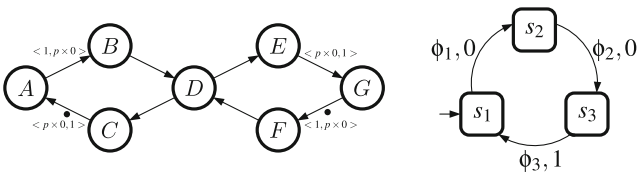


Figure 6 Extended (max, +)-automaton of the example graph.

A CSDF graph is a dataflow graph in which the actors have token production and consumption rates that can dynamically change with subsequent firings. These dynamic changes are however restricted to occur in strictly periodic patterns. An example of such behavior is a down-sampling actor in which the number of new input samples needed to compute an output sample varies periodically depending on the sample rates. Figure 7 shows an example of a CSDF graph representing a typical fragment of a model of a data item that is written once and subsequently used $p + 1$ times, including its buffer, taken from [38]. We assume that p is some arbitrary constant, rather than an explicit parameter. The repetition vector of the CSDF is $\rho = \{(A, 1), (B, p + 1), (C, p + 1), (D, p + 1), (E, p + 1), (F, p + 1), (G, 1)\}$. All actors have a self-edge with one token, not shown in the figure for clarity, to prevent auto-concurrency and out-of-order execution of the actor firings. Actors B, C, E and F cycle through different phases with different port rates. To capture this in SADF we need to separate those firings into different scenarios. We need however not make every actor firing a different scenario, or have a different scenario for every combination of actor phases. We group as many actor firings as possible into a single scenario. The resulting FSM for the SADF is also shown in Fig. 7. It has three phases that are executed in the characteristic cyclic fashion of CSDF graphs. We need three scenarios and the partial repetition vectors of those scenarios are also listed. In phase ϕ_1, all actors except G fire once. In the second phase the actors B, C, D, E and F fire according the vector ϕ_2. Note that only this phase depends on the value of p. The last phase completes the CSDF iteration. A reward of 1 is assigned to the last scenario ϕ_3 to define throughput as the number of iterations per time unit. Note that in parametric extensions of CSDF, such as VPDF, p can be interpreted as an explicit parameter. If the parameter has a finite domain, this can be modelled by having a copy of the ϕ_2 state for each valuation of p and have the FSM make a non-deterministic choice to enter any of them after ϕ_1. Alternatively, symbolic methods may be used, as in [42].



$$\phi_1 = \{(A,1),(B,1),(C,1),(D,1),(E,1),(F,1)\}$$
$$\phi_2 = \{(B,p),(C,p-1),(D,p),(E,p-1),(F,p)\}$$
$$\phi_3 = \{(C,1),(E,1),(G,1)\}$$

Figure 7 A CSDF graph and its SADF scenarios.

Algorithm 1 shows the pseudocode of an algorithm to convert a CSDF graph into a weakly consistent SADF. It computes the repetition vector in Line 1, which counts all the actor firings that need to be placed into scenarios. The variable σ, initialized in Line 2, keeps track of the locations of the tokens in the CSDF graph after the actor firings that have been processed, to be able to determine which next of the remaining actor firings are enabled. Variable **sf** is used to keep track of all firings that will be assigned to a new scenario to be created. In Line 4 the data structure is initialized that will eventually contain the SADF that is the result of the conversion. Initially, it is empty, has no scenarios. The scenarios are added later in the algorithm. Its FSM is initialized to have a single state only, with no transitions yet. Those will be added later. In the loop that starts on Line 6 actor firings are added to scenarios. To prevent the generation of an excessive number of states the algorithm combines maximal collections of actor firings into a single scenario. When two subsequent actor firings use the same port rates, they will be combined into one scenario (Lines 8–11). When the rates change, a new state is created in the SADF for a new scenario, because all actor firings within one scenario must have identical port rates. This is done in Lines 13–17. With "static in **sf**" in the condition in Line 7 we mean that the firing uses the same port rates as firings of the same actor in **sf**, if there are any. Note that if the input CSDF graph is, in fact, an SDF graph, then the constructed SADF graph will have only one state and one scenario and will therefore also be strongly consistent. When the loop is complete, all actor firings have been processed. A final scenario is created for the actor firings remaining in **sf** with a corresponding state in the FSM returning to the initial state in Lines 20 and 21. Finally, Lines 22 and 23 set the appropriate rewards so that one iteration through all phases yields a total reward of 1.

The complexity and run-time of the algorithm are chiefly influenced by the while loop on Line 6. The number of iterations of the loop is equal to the sum of the entries in the repetition vector of the CSDF graph. This number is typically in the order of the least common multiple of the actor firing rates in the graph and in the worst case it is equal to the product of all the firing rates. This makes the algorithm scale exponentially, in the worst-case, with the size of the graph and (pseudo-)polynomially with the firing rates themselves. Note however, that there are no known exact performance analysis algorithms for CSDF (or SDF) that scale better than with the size of the repetition vector, although there exist approximations [43]. The computation of the repetition vector itself (Line 1) is only linear in the number of actors in the graph. The additional operations inside or outside the loop can all be realized with complexity linear in the size of the number of actors in the graph.

Algorithm 1 Conversion from CSDF to WC-FSM-SADF

Data: deadlock-free CSDF graph G
Result: An SADF graph H equivalent to G

```
ρ_G ← repetition vector of G;
σ ← initial symbolic state of G;
sf ← ∅;
H ← empty SADF with single, initial, state;
s' ← initial state of H;
while ρ_G has actor firings do
    if there is an actor a, with ρ_G(a) > 0 and its next firing is enabled
    in σ and static in sf then
        φ ← next firing of actor a;
        add φ to sf;
        decrease ρ_G(a) by 1;
        apply firing φ to σ;
    else
        create scenario graph 𝒢 from sf;
        s ← new state in FSM of H;
        add edge from s' to s labelled 𝒢 in FSM of H;
        sf ← ∅;
        s' ← s;
    end
end
create scenario graph 𝒢 from sf;
add edge from s' to the initial state, labelled 𝒢 in FSM of H;
mark all scenarios of H with reward 0;
mark the last scenario added to H with reward 1;
```

Theorem 2 *Weakly consistent FSM-SADF generalizes CSDF. I.e., for any CSDF graph one can create an equivalent weakly consistent FSM-SADF with the same throughput and having a one-to-one mapping between actors of the CSDF and the SADF with the same actor firing times.*

Proof Assume (without loss of generality) that the CSDF graph is deadlock free. Algorithm 1 computes a weakly consistent SADF with the required properties. It is straightforward to demonstrate that dependencies between actor firings are preserved in the transformation. In the deterministic FSM that is generated, all states form a single cycle of states and a reward of exactly one is obtained for every passing of a complete cycle, which corresponds to a complete iteration of the CSDF graph. □

We briefly discuss some of the other popular analyzable dataflow models that weakly consistent SADF generalizes.

Static SDF , including HSDF, Computation Graphs and Weighted Marked Graphs are clearly generalized by SADF. They operate in only one scenario and are therefore strongly consistent.

HDF is generalized by SADF in the sense that it uses the same concept of scenarios or modes, but the model does not include a timing model. Its implementation in Ptolemy II does not allow pipelining of scenarios. It is also restricted to strongly consistent models.

PDF is a rate-parameterized dataflow model in which parameters may assume different values for each iteration. For each given value of the parameter the graph is a consistent SDF graph. It is therefore a strongly consistent SADF. When the number of possible parameter values is finite, it can be represented as a strongly consistent FSM-SADFG, otherwise parametric representations are required [42].

SPDF graphs are rate-parametric dataflow graphs with additional structural constraints on how the parameters are used that make them schedulable. It is a subclass of PDF and therefore of SADF.

(S)VPDF combines the actor phases of CSDF with parametric rates similar to SPDF. Those features can all be expressed in SADF. The example of Fig. 7, is a SVPDF when p is interpreted as a parameter that can vary per iteration.

MCDF, Mode Controlled Data Flow, is a dynamic dataflow model with a very similar semantics to SADF. Syntactically, it adds actors that explicitly control the modes or scenarios and switch and select actors that introduce mode-dependent behavior in a conceptually similar way it was done in the original SADF formulation [10]. MCDF models can be directly translated to SADF models. Constraints on scenario sequences are implicit in the functional behavior of the mode controller actors. If this information can be extracted by static analysis of the actor, it can be captured in an FSM.

Execution-time-parametric dataflow models [40–42, 44] take any of the existing models and replace constant actor execution times with parameters that may or may not be able to change values between iterations. Combined valuations of the execution time parameters can be seen as scenarios of an SADF [42] or regions of similar parameter values can be grouped into a single scenario representing its worst-case behavior [45]. Whether a parameter receives a value once, or repeatedly in every iteration can be expressed by the structure of the FSM. If the parameters take values from an infinite or continuous domain, there may also be an infinite number of states and a symbolic or implicit analysis is required. [44] explores such an approach, in which also constraints on dependencies between parameters are taken into account.

Core Functional Dataflow (CFDF) [46] is an analyzable version of the Enable-Invoke Data Flow [46] model, and Parameterized Set of Modes (PSM) [47] is a syntactic model

to allow their parametric specification. In CFDF, actors may fire according to different modes. This can be expressed in WC-FSM-SADF using scenarios for the individual actor firings. This may lead however to a too fine-grained model with a very large FSM. Static analysis of the enabling conditions and dependencies between actor modes could possibly be used to reduce the complexity of the SADF representation.

Boolean Parametric Data Flow (BPDF) [48] is a parametric dynamic dataflow model with strong analysis possibilities that combines boolean conditions in dependencies with parametric rates. Similar to other models, both aspects can be modeled with SADF scenarios. For boolean conditions this is very natural. The parametric rates can be enumerated in scenarios, but a parametric specification and analysis are preferred.

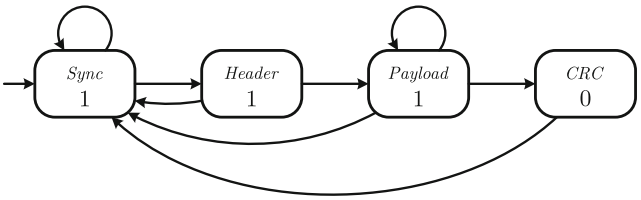
9 Experimental Evaluation

We have implemented the worst-case throughput analysis method in the SDF^3 tool set [49] as an extension to the available scenario-aware dataflow analysis. To get the performance analysis results, a maximum cycle ratio analysis is performed on the (max, +)-automaton using the algorithm of Young et al. [50]. In this section we first illustrate the method and the tool with two use cases. We then investigate how it compares to other approaches and then we take a closer look at the conversion algorithm from CSDF to SADF.

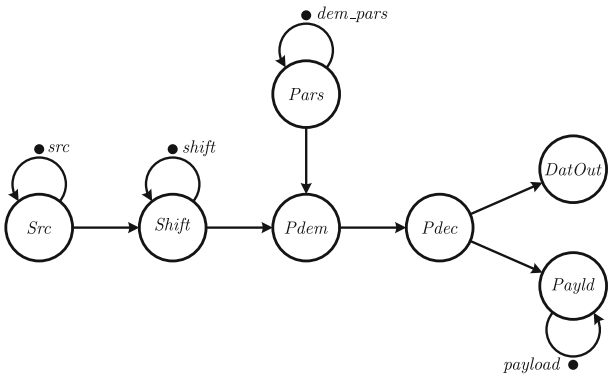
9.1 Use Cases

We have used the tool to analyze an MP3 decoder model with a file reader front-end (Fig. 1). Weak consistency is needed for this model to be able to express the asynchronous operation of the file reader and the decompression of the MP3 decoding. The specification consists of seven dataflow graphs for the five coding scheme scenarios and two additional file processing scenarios. The graphs of the frame decoding scenarios are fairly large (up to 25 actors). The specified FSM has 65 states. The (max, +)-matrices extracted from the scenario dataflow graphs are 3 by 3 matrices, where the rows/columns each represent one of the three processors on which the decoder is presumably mapped. The decoding process itself is stateless; therefore there are no additional initial or final tokens. The determinate behavior of the large scenario dataflow graphs, with many firings, can thus be very compactly represented with a small matrix. The (max, +)-automaton that is constructed from the FSM and the matrices has 195 nodes (one for every combination of the three initial tokens and one of the 65 FSM states) and it has 2745 edges. The MCR analysis of this graph tells us the maximal throughput which is guaranteed to be attainable. The computation time on a standard PC is around 35ms. As a result we get the worst-case throughput as well as a critical scenario sequence from a critical cycle of the MCR analysis. This is the sequence $ss \cdot (dec)^5 \cdot ss \cdot (dec)^4 \cdot rd \cdot dec$ (there may be other sequences with the same throughput). We see that the worst-case situation is that the decoder needs its maximum number of firings (5) to produce an audio frame, combined with the (apparently) hardest of the frame synthesis scenarios, *ss*.

We have also used SADF to model a WLAN receiver [39, 51]. It is intended to demonstrate that the scenario model of dynamic dataflow behavior fits very naturally with this type of application. In the WLAN receiver the reception of a data frame consists of phases of synchronization, header processing, payload processing and transmitting an acknowledgement. The internal data processing can be conveniently modelled with deterministic dataflow behavior, but the changes from synchronization mode to header processing mode and the length of the payload are unknown. This behavior cannot accurately be modeled with static dataflow models such as SDF or CSDF, but it can be modelled with non-deterministic scenario changes in SADF. Figure 8a shows the FSM expressing the scenario sequences. Figure 8b shows the dataflow graph of one of the scenarios (*Payload*). Note that for readability we have chosen an FSM representation where scenarios and rewards are associated with states instead of edges, but one representation can easily be converted in the other. Inside SDF^3 a



(a) FSM of the WLAN SADF



(b) SDFG of the *Payload* scenario

Figure 8 SADF model of a WLAN receiver.

representation is used in which edges are labelled, because this typically leads to fewer states. In the specification language, states can be labelled with scenarios and the tool converts and minimizes the automaton before analysis. Each scenario in the WLAN model, except the *CRC* scenario, corresponds to the processing of a single input symbol (OFDM) of 4μs. Recall that although a new scenario starts every 4μs, the duration of a single scenario may be longer when the executions are pipelined. The model starts with synchronization looking for a WLAN frame preamble in scenario *Sync*. While searching it will continue in the *Sync* scenario. When a frame is detected, it continues with the *Header* scenario in which additional processing takes place for demodulation and decoding. From any scenario, the application may fall back to the *Sync* scenario when synchronization is lost. When header processing succeeds, it continues in the *Payload* scenario, which is slightly different from the header processing scenario in terms of the processing of the payload. Although a frame consists of at most 256 payload symbols, the model conservatively allows arbitrarily long frames, which leads to a smaller FSM, since it does not need to count till 256. When the payload symbols have been processed, the frame is completed with computation of CRC error detection and the transmission of an acknowledgement in the *CRC* scenario. As rewards we assigned the number of symbols processed by a scenario, which is 1 except for the *CRC* scenario, where it is 0. This way the throughput computation determines the worst-case average number of symbols that can be processed per time unit. In terms of analysis it is a fairly simple model. The execution time of the throughput analysis is below 1ms.

9.2 Comparison to Other Dataflow Approaches

To illustrate how dynamic dataflow models such as CSDF or SADF can give more accurate bounds on performance than static models of the same application, we have experimented with the channel equalizer application dataflow model from Moonen et al. [52, 53]. The graph is shown in Fig. 9. It is modelled in [52] as a CSDF graph. The communication rates of all actors in Fig. 9 are equal to 1, except where an actor with a solid line connects to a dashed channel. There, a token is produced or consumed only on every 8th firing of the actor. Because of this the actors with a dashed line fire only once for every 8 firings of the solid actors. In our terminology, the graph exhibits two (strongly consistent) scenarios, one in which only the solid actors fire, and one in which all actors fire. The former scenario is repeated 7 times in a row, followed by one execution in the latter scenario and this pattern deterministically and periodically repeats. A static abstraction can be made by conservatively assuming that all actors execute in each scenario with their worst-case execution times or, alternatively, by unfolding 8

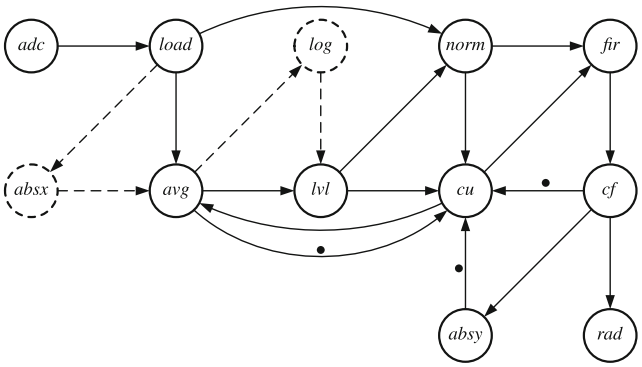


Figure 9 CSDF model of a channel equalizer [52].

subsequent scenarios into one large dataflow graph. In the latter case, for the channel equalizer, the number of actors goes up from 12 to 82, moreover the unfolding is, in general, only possible for models with deterministic sequences of scenarios. The unfolding approach is used in [52] to compute the throughput of the CSDF graph. We have applied the former approach to obtain a static model of a single scenario, the processing of a single sample in the channel equalizer. The analysis shows that the throughput guarantee provided by the equivalent static dataflow graph is $119kHz$, while the throughput guarantee from the CSDF/SADF model is $162kHz$. A difference, for this particular example, of 36 %.

We investigate the scalability benefits of the separation of the scenarios into matrix multiplications in the $(\max, +)$-automaton, by comparing the analysis with an explicit state-space analysis approach as presented in [54]. We have taken the MPEG-4 Simple Profile decoder use case from [10]. The scenarios in the model represent different types of encodings of the video frame that needs to be decoded. The tool of [54], which is also integrated in the SDF^3 tool set, creates the explicit state-space of the operational semantics of the model. Due to the pipelining of non-deterministic different scenario sequences and the interleaving of concurrent actor firings, the state-space is rather large. The tool reports a state-space of 6489 states after the state-space reduction techniques that are described in [54]. We have modelled the same application in our $(\max, +)$-automaton model. The $(\max, +)$-automaton graph structure created and analyzed by our method has 8 nodes, because the scenario graphs have 8 tokens and the FSM has a single state, because all scenarios are possible in arbitrary orders. The throughput analysis of [54] takes 185 seconds, while our analysis takes less than 15 milliseconds. Note that the tools analyze different properties. The tool of [54] computes expected long-run average throughput from a Markov model of scenario sequences, while our analysis computes worst-case throughput. In order to compute worst-case throughput however, the same state-space would be generated.

Table 1 Experiment with CSDF graphs.

Model	actors	phases	rep.vector	states	transf.	analysis
Channel equalizer [53]	12	8	656	24	<1ms	15ms
H.263 encoder [55, 56]	6	≤ 99	398	3	15ms	<1ms
MP3 playback [57]	4	≤ 39	180283	325	1015ms	1656ms

9.3 Conversion from CSDF to SADF

Algorithm 1, to convert a CSDF graph to SADF, has also been implemented in SDF^3. The conversion has been applied to three CSDF models. The first is the aforementioned channel equalizer [52, 53]. It has 12 actors, which have 8 phases each and a total number of 656 firings in the repetition vector. The second is an H.263 encoder [55, 56]. It has 6 actors, which have up to 99 phases each and a total number of 398 firings in the repetition vector. The third is an MP3 playback model [57] (a CSDF model different from the MP3 model of Fig. 1). It has 4 actors which have up to 39 phases each and a total number of 180283 firings. Table 1 summarizes the results of experiments with these CSDF models. It shows the number of actors in the models, the maximum number of phases of any of the actors, the number of firings in the repetition vector, the number of states in the FSM of the converted FSM-SADF, the time the conversion took in SDF^3 and the time the throughput analysis in the converted model took in SDF^3. We need to add that the throughput analysis directly on the CSDF models, using SDF^3, is still significantly faster than the analysis of the converted SADF models. The translation algorithm confirms the generalization results and allows us to apply other SADF analysis algorithms to CSDF graphs. Needless to say the throughput analysis yields the same results as direct throughput analysis on the CSDF.

10 Conclusion

We introduced an exact analysis method for a class of dynamic dataflow graphs, called weakly consistent scenario-aware dataflow in which the behavior may non-deterministically vary according to scenarios of behavior, yet within these scenarios behavior is deterministic and follows the synchronous dataflow paradigm which provides us with powerful analysis techniques. The model introduced in this paper generalizes a wide class of analyzable dynamic dataflow models, including CSDF, PDF and CFDF, and is therefore a good candidate for a semantic framework and analysis tool set. We have generalized the (max, +)-semantics of SADF to allow non-consistent scenario behavior to (a generalization of) (max, +)-automata and exploit existing spectral analysis techniques in (max, +)-algebra for performance analysis. We have implemented the techniques in a tool for performance analysis of dataflow models, made available as part of the SDF^3 tool set [49] at www.es.ele.tue.nl/sdf3. We see that it can effectively analyze various real-world models, such as the MP3 decoder and the WLAN receiver.

Acknowledgments This research is supported by the ARTEMIS joint undertaking through the ALMARVI project (621439) and by Catrene through the BENEFIC project (CA505).

References

1. Baccelli, F., Cohen, G., Olsder, G., & Quadrat, J.P. (1992). *Synchronization and linearity*: Wiley.
2. Boudec, J.L., & Thiran, P. (2003). *Network calculus: A theory of deterministic queuing systems for the internet, vol. 2050 of Lecture Notes in Computer Science*: Springer.
3. Thiele, L., Chakraborty, S., & Naedele, M. (2000). Real-time calculus for scheduling hard real-time systems. In *International Symposium on Circuits and Systems ISCAS 2000*, (Vol. 4 pp. 101–104). Geneva, Switzerland.
4. Campos, J., Chiola, G., Colom, J., & Silva, M. (1992). Properties and performance bounds for timed marked graphs. *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, *39*(386), 401.
5. Murata, T. (1989). Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, *77*(4), 541–580.
6. Gaubert, S. (1995). Performance evaluation of (max, +) automata. *IEEE Transactions Automatic Control*, *40*(12), 2014–2025.
7. Lee, E., & Messerschmitt, D. (1987). Synchronous data flow. *IEEE Proceedings*, *75*, 1235–1245.
8. Sriram, S., & Bhattacharyya, S.S. (2009). *Embedded Multiprocessors: Scheduling and Synchronization*. NY, USA: Marcel Dekker, Inc.
9. Buck, J.T. (1994). A dynamic dataflow model suitable for efficient mixed hardware and software implementations of dsp applications. In *Proceedings of the 3rd International Workshop on Hardware/Software Co-Design* (pp. 165–172).
10. Theelen, B.D., Geilen, M., Basten, T., Voeten, J., Gheorghita, S.V., & Stuijk, S. (2006). A scenario-aware data flow model for combined long-run average and worst-case performance analysis. In *Proceedings of the Fourth ACM and IEEE International*

Conference on Formal Methods and Models for Co-Design 2006 (MEMOCODE '06) (pp. 185–194).
11. Stuijk, S., Geilen, M., Theelen, B., & Basten, T. (2011). Scenario-aware dataflow: Modeling, analysis and implementation of dynamic applications. In *Proceedings of International Conference on Embedded Computer Systems (SAMOS), 2011* (pp. 404–411).
12. Geilen, M. (2011). Synchronous data flow scenarios. *Transactions on Embedded Computing Systems*, *10*, 16:1–16:31.
13. Geilen, M., & Stuijk, S. (2010). Worst-case performance analysis of synchronous data ow scenarios. In *International Conference on Hardware-Software Codesign and System Synthesis, CODES+ISSS 10* (pp. 125–134). Az, USA: Proceedings.
14. Bilsen, G., Engels, M., Lauwereins, R., & Peperstraete, J. (1996). Cyclo-static dataflow. *IEEE Transactions on signal processing*, *44*, 397–408.
15. Geilen, M., Falk, J., Haubelt, C., Basten, T., Theelen, B., & Stuijk, S. (2014). Performance analysis of weakly-consistent scenario-aware dataflow graphs. In *Asilomar Conference on Signals, Systems, and Computers, Asilomar 14, Pacific Grove*. CA, USA: IEEE Proceedings.
16. Lee, E.A., & Matsikoudis, E. The Semantics of Data ow with Firing. Cambridge University Press, 2007. Chapter from From Semantics to Computer Science: Essays in memory of Gilles Kahn.
17. Karp, R.M., & Miller, R.E. (1966). Properties of a model for parallel computations: Determinancy, termination, queueing. *SIAM Journal of Applied Mathematics*, *14*, 1390–1411.
18. Girault, A., Lee, B., & Lee, E. (1999). Hierarchical finite state machines with multiple concurrency models. *IEEE Transactions on Computer-aided Design of Integrated Circuits and Systems*, *18*, 742–760.
19. Bhattacharya, B., & Bhattacharyya, S. (2001). Parameterized dataflow modeling for DSP systems. *IEEE Transactions on Signal Processing*, *49*, 2408–2421.
20. Wiggers, M.H., Bekooij, M.J., & Smit, G.J. (2008). Buffer capacity computation for throughput constrained streaming applications with data-dependent inter-task communication. In *Proceedings of the 14th IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS'08* (pp. 183–194).
21. Buck, J., & Lee, E.A. (1994). Advanced Topics in Dataflow Computing and Multithreading, ch. The Token Flow Model. IEEE Computer Society Press.
22. Kahn, G. (1974). The semantics of a simple language for parallel programming, In Rosenfeld, J. (Ed.) *Information Processing 74: Proceedings of the IFIP Congress 74, Stockholm, Sweden* (pp. 471–475). North-Holland, Amsterdam, Netherlands.
23. Kahn, G., & MacQueen, D. (1977). Coroutines and networks of parallel programming, In Gilchrist, B. (Ed.) *Information Processing 77: Proceedings of the IFIP Congress 77, Toronto, Canada* (pp. 993–998). North-Holland.
24. Thiele, L., & Stoimenov, N. (2009). Modular performance analysis of cyclic dataflow graphs. In *EMSOFT '09: Proceedings of the Seventh ACM International Conference on Embedded Software* (pp. 127–136). NY, USA: ACM.
25. Petri, C. (1962). *Kommunikation mit Automaten. PhD thesis, Institut für instrumentelle Mathematik*. Germany: Bonn.
26. Viennot, G. (1986). *Combinatoire Énumérative, ch. Heaps of Pieces, I: Basic definitions and combinatorial lemmas*, (pp. 321–350): Springer.
27. Gaubert, S., & Mairesse, J. (1999). Modeling and analysis of timed petri nets using heaps of pieces. *IEEE Transactions Automatic Control*, *44*, 683–697.
28. Cochet-Terrasson, J., Cohen, G., Gaubert, S., Gettrick, M., & Quadrat, J.-P. (1998). Numerical computation of spectral elements in max-plus algebra. In *Proceedings of the IFAC Conference on System Structure and Control*: Nantes.
29. Dhingra, V., & Gaubert, S. (2006). How to solve large scale deterministic games with mean payoff by policy iteration. In *Proceedings of the 1st international conference on Performance evaluation methodolgies and tools, valuetools '06*. NY, USA: ACM.
30. Dasdan, A. (2004). Experimental analysis of the fastest optimum cycle ratio and mean algorithms. *ACM Transactions on Design Automation of Electronic Systems*, *9*, 385–418.
31. Bhattacharyya, S.S., Murthy, P., & Lee, E. (1997). APGAN and RPMC: Complementary heuristics for translating dsp block diagrams into efficient software implementations. *Journal of Design Automation for Embedded Systems*.
32. Falk, J., Keinert, J., Haubelt, C., Teich, J., & Bhattacharyya, S. (2008). A generalized static data flow clustering algorithm for MPSoC scheduling of multimedia applications. In *EMSOFT'08: Proceedings of the 8th ACM International Conference on Embedded Software*.
33. Meijer, S., Nikolov, H., & Stefanov, T. (2010). Throughput modeling to evaluate process merging transformations in polyhedral process networks. In *DATE* (pp. 747–752).
34. Tripakis, S., Bui, D., Geilen, M., Rodiers, B., & Lee, E.A. (2010). *Compositionality in synchronous dataflow: Modular code generation from hierarchical sdf graphs*. Berkeley: Technical Report UCB/EECS-2010-52, EECS Department, University of California.
35. Geilen, M., Tripakis, S., & Wiggers, M. (2011). The earlier the better: A theory of timed actor interfaces. In *Proceedings of the 14th International Conference on Hybrid Systems: Computation and Control, HSCC '11* (pp. 23–32). NY, USA: ACM.
36. Fradet, P., Girault, A., & Poplavko, P. (2012). Spdf: A schedulable parametric dataflow moc. In *Proceedings of the Conference on Design, Automation and Test in Europe, DATE '12* (pp. 769–774). CA, USA: EDA Consortium.
37. Wiggers, M. (2009). Aperiodic multiprocessor scheduling. PhD thesis, University of Twente.
38. Geuns, S.J., Hausmans, J.P., & Bekooij, M.J. (2013). Automatic dataflow model extraction from modal real-time stream processing applications. In *Proceedings of the 14th ACM SIGPLAN/SIGBED Conference on Languages, Compilers and Tools for Embedded Systems, LCTES '13* (pp. 143–152). NY, USA: ACM.
39. Moreira, O. (2012). Temporal analysis and scheduling of hard real-time radios running on a multi-processor. PhD thesis, Eindhoven University of Technology.
40. Ghamarian, A., Geilen, M., Basten, T., & Stuijk, S. (2008). Parametric throughput analysis of synchronous data flow graphs. In *Proceedings of the Design Automation and Test in Europe Conference (DATE)*, (Vol. 10-14 pp. 116–121). Munich, Germany.
41. Damavandpeyma, M., Stuijk, S., Geilen, M., Basten, T., & Corporaal, H. (2012). Parametric throughput analysis of scenario-aware dataflow graphs. In *2012 IEEE 30th International Conference on Computer Design (ICCD)* (pp. 219–226).
42. Skelin, M., Geilen, M., Catthoor, F., & Hendseth, S. (2014). Worst-case throughput analysis for parametric rate and parametric actor execution time scenario-aware dataflow graphs, (Vol. 2014 pp. 65–79).
43. De Groote, E. (2016). On the analysis of synchronous dataflow graphs: a system-theoretic perspective. PhD thesis, University of Twente.
44. Skelin, M., Geilen, M., Catthoor, F., & Hendseth, S. (2015). Worst-case latency analysis of sdf-based parametrized dataflow mocs. In *Proceedings of the 2015 Conference on Design and Architectures for Signal and Image Processing (DASIP)* (p. 2015).

45. Gheorghita, S., Basten, T., & Corporaal, H. (2008). Application scenarios in streaming-oriented embedded system design. *IEEE Design and Test of Computers*, *25*, 581–589.
46. Plishker, W., Sane, N., Kiemb, M., Anand, K., & Bhattacharyya, S.S. (2008). Functional dif for rapid prototyping. In *RSP'08. The 19th IEEE/IFIP International Symposium on Rapid System Prototyping* (pp. 17–23): IEEE.
47. Lin, S., Wang, L.-H., Vosoughi, A., Cavallaro, J., Juntti, M., Boutellier, J., Silvň, O., Valkama, M., & Bhattacharyya, S. (2014). Parameterized sets of dataflow modes and their application to implementation of cognitive radio systems. *Journal of Signal Processing Systems*, 1–16.
48. Bebelis, V., Fradet, P., Girault, A., & Lavigueur, B. (2013). BPDF: A statically analyzable data ow model with integer and boolean parameters. In *Proceedings of the International Conference on Embedded Software (EMSOFT)* (pp. 1–10): IEEE.
49. Stuijk, S., Geilen, M., & Basten, T. (2006). SDF^3: SDF for free. In *Application of Concurrency to System Design, 6th International Conference, ACSD 2006, Proceedings* (pp. 276–278). CA, USA: IEEE Computer Society Press.
50. Young, N.E., Tarjan, R., & Orlin, J. (1991). Faster parametric shortest path and minimum balance algorithms. *Networks*, *21*(2), 205–221.
51. Siyoum, F. (2014). Worst-case Temporal Analysis of Real-time Dynamic Streaming Applications. PhD thesis, Eindhoven University of Technology.
52. Moonen, A., Bekooij, M., Van den Berg, R., & Van Meerbergen, J.L. (2007). Practical and accurate throughput analysis with the cyclo static data ow model. In *15th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS 2007)* (pp. 238–245). Istanbul, Turkey.
53. Moonen, A. (2009). Predictable Embedded Multiprocessor Architecture for Streaming Applications, PhD thesis, Eindhoven University of Technology.
54. Theelen, B., Geilen, M., & Voeten, J. (2011). Performance model checking scenarioaware dataflow. In *Proceedings of the International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS), vol. 6919 of LNCS* (pp. 43–59): Springer.
55. Kim, D. (2003). System-level specification and cosimulation for multimedia embedded systems, PhD thesis, Seoul National University.
56. Oh, H., & Ha, S. (2004). Fractional rate dataflow model for efficient code synthesis. *Journal of VLSI Signal Processing Systems for Signal, Image and Video Technology*, *37*(1), 41–51.
57. Wiggers, M., Bekooij, M., & Smit, G.J.M. (2007). Efficient computation of buffer capacities for cyclo-static dataflow graphs. In *Proceeding of the Design Automation Conderence (DAC)* (pp. 658–663).

Marc Geilen is an assistant professor in the Department of Electrical Engineering at Eindhoven University of Technology. He holds an MSc in Information Technology and a Ph.D. from the Eindhoven University of Technology. His research interests include formal models-of-computation, model-based design processes, multi-processor systems-on-chip, networked embedded systems and cyber-physical systems, and multi-objective optimization and trade-off analysis. He has served as member or chair of several related technical program committees. He has been involved with national and international research projects and programs on the above topics with strong industrial connections.

Joachim Falk was born in 1977 and grew up around Erlangen, Germany. He started his study of electrical engineering in 1997 at the Georg Simon Ohm University of Applied Sciences in Nuremberg, Germany. There, he earned his degree Dipl.-Ing. (FH) in electrical engineering with a specialization in data processing in 2002. In 2003, he joined the chair of Hardware/Software Co-Design of Prof. Dr.-Ing. Jürgen Teich at the Friedrich-Alexander-Universität Erlangen-Nürnberg. From 2006 onward, he has been working as a PhD student in the project "Actor-Oriented Synthesis and Optimization of Digital Hardware/Software-Systems at the Electronic System Level" receiving his PhD degree in 2014 from the Friedrich-Alexander-Universität Erlangen-Nürnberg with a dissertation titled "A Clustering-Based MPSoC Design Flow for Data Flow-Oriented Applications". Joachim Falk has been a reviewer for several international journals. His main research focuses are data flow languages and transformations for data flow graphs in the context of embedded system design.

Christian Haubelt is a full professor of Embedded Systems at the University of Rostock, Germany. He received his diploma degree in Electrical Engineering from the University of Paderborn, Germany, in 2001. He finished his Ph.D. in Computer Science and his Habilitation (post-doctoral lecture qualification) in Computer Engineering at the University of Erlangen-Nuremberg, Germany, in 2005 and 2010, respectively. Since 2011, he is a full professor of Embedded Systems at the University of Rostock, Germany. His research interests include embedded and cyber-physical systems, electronic system level design automation, and multi-objective optimization.

Twan Basten is a professor in the Electrical Engineering department at Eindhoven University of Technology (TU/e), the Netherlands, where he chairs the Electronic Systems group. He is also a senior research fellow of TNO Embedded Systems Innovation in the Netherlands. His research interests include embedded, networked and cyber-physical systems, dependable computing and computational models. He (co)supervised 15 PhD degrees. Twan Basten is a senior member of the IEEE and a life member of the ACM.

Bart Theelen received his Master's degree in Information Technology Sciences in 1999 and his Ph.D. in 2004 from Eindhoven University of Technology. He is currently a Research Fellow at Embedded Systems Innovations by TNO where he leads the research program on system performance. His interests include industrial application of model-based methods, formalisms and techniques for the design, specification, analysis and synthesis of (embedded) high-tech systems, with particular focus on designing for system performance, virtual prototyping, model management and test. He conducted projects with various industrial partners including Alcatel Bell Antwerp (B), IBM Research Laboratory Zürich (CH), ASML Veldhoven (NL), TPVision Bruges (B), Philips HealthTech Best (NL) and Barco Kortrijk (B).

Sander Stuijk received his M.Sc. (with honors) in 2002 and his Ph.D. in 2007 from the Eindhoven University of Technology. He is currently an assistant professor in the Department of Electrical Engineering at Eindhoven University of Technology. He is also a visiting researcher at Philips Research Eindhoven working on bio-signal processing algorithms and their embedded implementations. His research focuses on modelling methods and mapping techniques for the design and synthesis of predictable systems with a particular interest into bio-signals.