Doctoral theses at NTNU, 2016:245

Mladen Skelin

Worst-case Performance Analysis of Scenario-aware Real-time Streaming Applications

Doctoral Thesis

Mladen Skelin

ISBN 978-82-326-1828-6 (printed version)
ISBN 978-82-326-1829-3 (electronic version)
ISSN 1503-8181

Doctoral theses at NTNU, 2016:245

NTNU
Norwegian University of
Science and Technology
Faculty of Information Technology,
Mathematics and Electrical Engineering
Department of Engineering Cybernetics

NTNU
Norwegian University of
Science and Technology

NTNU

NTNU
Norwegian University of
Science and Technology

Mladen Skelin

Worst-case Performance Analysis of Scenario-aware Real-time Streaming Applications

Thesis for the degree of Philosophiae Doctor

Trondheim, April 2016

Norwegian University of Science and Technology
Faculty of Information Technology,
Mathematics and Electrical Engineering
Department of Engineering Cybernetics

KU Leuven
Faculty of Engineering Science
Department of Electrical Engineering (ESAT)

NTNU
Norwegian University of
Science and Technology

KU LEUVEN

NTNU and KU Leuven

Thesis for the degree of Philosophiae Doctor

Norwegian University of Science and Technology
Faculty of Information Technology, Mathematics
and Electrical Engineering
Department of Engineering Cybernetics

KU Leuven
Faculty of Engineering Science
Department of Electrical Engineering (ESAT)

ISBN 978-82-326-1828-6 (printed version)
ISBN 978-82-326-1829-3 (electronic version)
ISSN 1503-8181
ITK-report 2016-13-W

Doctoral theses at NTNU, 2016:245
Doctoral theses at KU Leuven

Printed by Skipnes Kommunikasjon as

Abstract

An embedded system is a combination of hardware and software designed to perform a dedicated function. Embedded systems typically come with stringent performance constraints such as throughput, latency, memory demands and resource usages. Deriving (or proving) that these performance constraints are satisfied in all possible circumstances is a challenging task due to increasing complexity of such systems.

In this thesis, we focus on formal method-based performance analysis techniques for dynamic streaming applications using dataflow models of computation (MoC). Streaming applications are applications that transform input streams of data of indefinite length to output streams of data. They are considered dynamic if their computational and communicational requirements vary at run-time. Dataflow formalisms have been widely used to model and analyze streaming applications. Finite-state machine-based scenario-aware dataflow (FSM-SADF) is one of better-known MoCs used for modeling of dynamic streaming applications. In particular, FSM-SADF models the execution of a dynamic application as a sequence of fairly static modes of operation called *scenarios*. Each scenario is in turn modeled by a synchronous dataflow (SDF) graph.

FSM-SADF allows rigorous design-time analysis and comes equipped with various worst-case performance analysis techniques.

However, as the number of scenarios grows, FSM-SADF experiences compactness problems which in turn render it incapable of capturing applications exposing fine-grained data-dependent dynamic behavior. As the first contribution of this thesis we identify the semantic link between FSM-SADF and parameterized dataflow models based on SDF we refer to as SDF-based parameterized dataflow (SDF-PDF) by relating the concepts of FSM-SADF scenario and SDF-PDF configuration/instance. SDF-PDF, by using dynamic parameters has the ability of capturing fine-grained data-dependent application dynamics. By exploiting the semantic link with FSM-SADF we adapt the worst-case throughput and latency analysis techniques of FSM-

SADF to be used for the analysis of SDF-PDF models. Thus, we have indirectly, by introducing the SDF-PDF concept and recasting it into the FSM-SADF context enabled worst-case performance analysis of applications exhibiting fine-grained data-dependent dynamism.

SDF-PDF as a parameterized dataflow model considers data-dominated applications with fine-grained data-dependent dynamics. However, many streaming applications can in addition have intricate control requirements. Therefore, the second contribution of the thesis combines parameterized dataflow and FSM-SADF into a novel model called FSM-based parameterized scenario-aware dataflow (PFSM-SADF). Combining the properties of the two, as a FSM/dataflow hybrid, PFSM-SADF is able to capture applications with both fine-grained data-dependent dynamics and intricate control requirements. For a flavor of PFSM-SADF based on SDF called SDF-based PFSM-SADF (SDF-PFSM-SADF), we in addition propose worst-case throughput and latency analysis techniques.

As the third contribution of the thesis we consider SDF-PFSM-SADF where parameters are deemed static or change infrequently. This is often the case for many applications is practice. Under such restricted semantics (normally, SDF-PFSM-SADF parameters are dynamic), we develop a technique that expresses the worst-case throughput of the graph as a function of the graph's parameters. Such expressions can be efficiently evaluated at both design-time and run-time and therefore can be used to perform both offline and online optimizations.

The available analysis methods for FSM-SADF are implemented in the SDF^3 tool. However, SDF^3 is of limited scope in the sense that it supports only a predefined set of properties. As the fourth contribution of the thesis, we report on the translation of the FSM-SADF formalism to UPPAAL timed automata that enables a more general verification of FSM-SADF models than currently supported by existing tools, i.e. by SDF^3.

Finally, the fifth and the last contribution of the thesis extends FSM-SADF to enable it to capture systems that make use of event-driven mechanisms to control the operation of its data-intensive parts. For the extension, translated to UPPAAL timed automata, we propose a schedulability analysis technique formulated as a reachability problem for (ordinary) timed automata.

All of the analysis techniques presented in this thesis are evaluated on realistic case studies from the multimedia domain and/or on representative artificial case studies.

Preface

This thesis has been submitted in partial fulfillment of the requirements for the degree of PhD at the Department of Engineering Cybernetics (ITK), Norwegian University of Science and Technology (NTNU) in Trondheim, Norway and at the Department of Electrical Engineering (ESAT), KU Leuven in Leuven, Belgium. The thesis is a part of a dual PhD program between NTNU and KU Leuven, in cooperation with Interuniversity Microelectronics Center (IMEC) in Leuven, Belgium. The work herein was performed at the ITK, NTNU and ESAT, KU Leuven. The work was performed under the supervision of Professor Francky Catthoor (KU Leuven) and Associate professor Sverre Hendseth (NTNU).

Acknowledgements

I would like to thank my supervisors Sverre Hendseth and Francky Catthoor for their guidance and support through the long process that eventually became this thesis. I also extend my gratitude to my co-supervisor Per Gunnar Kjeldsberg for friendly attitude and advice over the years. Last but not least, I would like to thank Marc Geilen for providing in-depth discussions about the subject matter and motivation at a key stage of the process.

Furthermore, I'm very grateful to Iason Filippopoulos for sharing office space with me and for companionship during the entire duration of our PhD studies.

Finally, I would like to thank my parents and the people close to me for their support.

Mladen Skelin
April 2016

Contents

Chapter 1

Introduction

1.1 Embedded systems

The literature [82][118][75] defines embedded systems as electronic systems that use a computer to perform a specific function. They are a combination of hardware and software and additional mechanical, optical and/or other parts. Although they make use of a computer to perform their function, they are not perceived as computers themselves. Here, we think of computers as devices that serve various functions like our desktop or laptop computers.

Examples of embedded systems span across numerous domains such as biomedical, automotive, aerospace, consumer, etc. A good example from the biomedical domain is a pacemaker, from the automotive an anti-lock braking system in a car, from the aerospace domain a flight computer, from the consumer a cell phone, washing machine, microwave oven etc. Thus, our everyday lives are unimaginable without embedded systems.

The statistics confirm pervasiveness of embedded systems in today's society by saying that the worldwide market for embedded systems in 2009 was 160 billion euros, with an annual growth of 9 percent [36]. Which such a growth, we will reach a staggering 275 billion of euros in 2016. Furthermore, in 2009, embedded microprocessors account for more than 98 percent of all produced microprocessors, thus vastly surpassing computing power in the IT industry [36].

1.2 Challenges in embedded system design

Embedded systems are pervasive. That is a self-evident fact. What is less evident is the number of problems faced by the industry in the process of embedded system development due to increased complexity. The increase

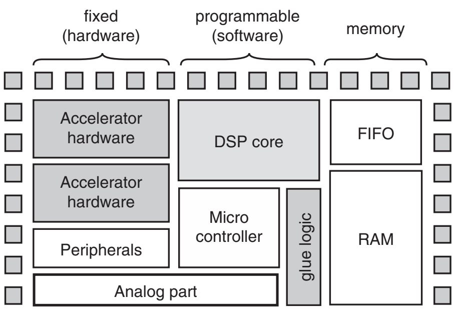


Fig. 1.1: A typical SoC device (source [118]).

in complexity is equally due to quite a few aggravating circumstances and due to constraints the designs must comply to. We list the most important ones.

The first aggravating circumstance is the market need for more and more functionality to be implemented by embedded systems (large number of use-cases) with lower cost and reduced time to market.

With regard to the degree of integration, the second aggravating factor influencing design complexity is the system on chip (SoC) era. In particular, the complexity is growing due to exponentially increasing transistor count which give rise to the SoC concept that enables integration of complete systems on a single chip. In practice this means that in SoC, entities like processors, digital signal processors (DSPs), sensors, memories, FPGAs (field programmable gate arrays), actuators and others are placed on a single chip. Fig. 1.1 shows a typical example of such a SoC device. Next to the software-driven components (DSP core and the microcontroller), the device contains specialized data paths in form of accelerators and various memory organizations (RAM and FIFO). Although heterogeneity allows for achieving lower power consumption and higher performance (e.g. signal processing on a DSP is more efficient with respect to power dissipation and chip area than on a microprocessor, while a microprocessor is more efficient in handling control-flow oriented code [75]), it significantly impedes the design process because it enlarges the design-space to be explored in search for a system configuration that satisfies input design constraints.

The third aggravating circumstance leading to increased design complexity is the dynamic nature of applications running on embedded SoC platforms themselves. These applications change their computational and communicational requirements at run-time depending on the characteristics of input data, which from the system perspective results in a dynamically changing workload.

At the same time, systems must be constraint compliant. E.g., real-time systems have constraints that concern various temporal properties like the minimal throughput, maximal latency, deadline satisfaction, etc. Multimedia systems on the other hand must achieve certain quality of service (QoS) levels. A good example is video quality in present-day streaming services available to us through our smart-phones that is being constantly enhanced. This leads to the increase of computational power needed to assure the desired QoS level. Furthermore, portable systems have constraints concerning power consumption and chip area. Here it is interesting to notice that QoS and limited power consumption are two constraints that adversely affect each other making the design problem even more difficult.

To summarize, these design constraints in interaction with the aforementioned aggravating circumstances lead to vast design spaces. The formulation and exploration of these design spaces with the overall goal of optimizing the trade-offs between several design objectives while meeting all the constraints is a difficult problem that impedes the design process of an embedded system.

1.3 Embedded system design methodologies

This section briefly introduces the existing strategies that successfully address the design challenges listed previously. It is important to emphasize that these methodologies are not to be always considered independent of each other but rather as being complementary to each other.

1.3.1 Platform-based design

The *Taxonomies for the Development and Verification of Digital Systems* [8] defines platform-based design as an integration oriented design approach emphasizing systematic reuse for developing complex products based upon platforms and compatible hardware and software components, intended to reduce development risks, costs, and time to market.

Simply put, in consideration of increasing non-recurring engineering (NRE) and design costs, platform-based design advocates for reuse of predesigned intellectual property (IP) components while developing a platform that will be suitable for some application domain [91]. This way, NRE and design costs are reduced as the need for the development of new IPs is avoided. Therefore, the company can focus on its core competency rather than investing money and time in IP development. Typically, IP creation, design assembly and manufacturing, for the most part, no longer take place in the same organization. Thus, platform-based approach defines a "meeting-in-the-middle" design style that lies between the full custom and fully programmable styles.

1.3.2 Model-based design

Model-based design flows [71][92], as the word suggests, focus on models of computation (MoCs) as the core design artifact. In particular, they are based on the use of models with a mathematical backing and a strong semantics characterization. The entire system (applications and the platform) is represented at an abstract level. Therefore, model-based design flows can be combined with platform-based design flows. Typically, systems described by models can be directly used as inputs into the synthesis trajectory. Within the trajectory, the model enables a hierarchical design process that iteratively refines the design and includes details necessary to implement the desired functionality [82]. Inside the process, different properties of the system can be determined by analyzing its model.

In the model-based design approach the platforms should be as composable as possible [91] and models relatively static so that analyzability is ensured. However, as mentioned previously, modern applications are no longer static and cannot be captured by static models without incurring significant amounts of pessimism in the design process. This pessimism has the effect of reducing the optimization margin a designer has at his/hers disposal.

1.3.3 Scenario-based design

Scenario-based design approach [52][122][76][79] targets dynamic applications. Within the approach, the dynamic behavior of an application is viewed as a collection of different *modes* or *scenarios*. Scenarios are derived by bounded clustering of behaviors of the application and the application mapping on the platform in such a way that these behaviors are similar from some multidimensional cost perspective such as delay or energy consumption. Internally, scenarios are fairly static and in isolation can be treated using model-based design methodologies in consideration of a particular platform. This shows that scenario-based design can be complemented with platform-based and model-based design.

Within the scenarios methodology, clustering of behaviors in scenarios is done in such a way that the system can be configured to exploit this cost similarity [52].

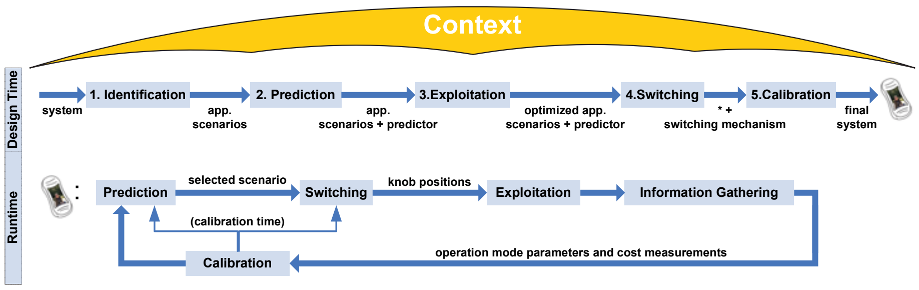


Fig. 1.2: System scenario methodology overview (source [52]).

This approach leads to a five step methodology shown in Fig. 1.2, where each of the steps has a design-time and a run-time phase. To clarify, consider a block-based video decoder application, e.g. H.264 [84] that is to be run on an battery-powered device supporting dynamic voltage and frequency scaling (DVFS) [64]. Assume that the video decoder has an associated throughput constraint, i.e. it has to deliver a certain number of video frames per unit of time. In H.264, frames are composed of a certain number of macroblocks the number of which depends on the frame resolution. Each macroblock can be encoded using several different encoding schemes. On the decoder's side each of these scheme's entails a different computational effort. Therefore, the time (and energy) needed to decode a particular frame will depend on the exact breakup, i.e. how many macroblocks of the frame belong to a particular encoding scheme. Now, the decoder throughput constraint implies that every frame needs to be encoded within a fixed period of time called the available decoding time or simply the frame budget. However, due to dynamism that the input video stream exhibits, some frames require all the available decoding time, while some do not. This fact can be used in conjunction with DVFS to achieve energy savings and so prolong the battery life. With DVFS, when scaling the voltage, the processor's frequency and the execution time of a task scale linearly ($f_{\mathrm{clk}} \sim V_{\mathrm{DD}}$), while the energy consumption scales approximately quadratically ($E \sim V_{\mathrm{DD}}^2$). Therefore, it is advantageous from energy saving perspective to lower the processor's voltage as long as the new frame decoding time is less or equal to the frame budget. This way, energy will be saved, throughput constraints met and the battery life prolonged.

However, considering every frame breakup to tune the system at runtime would involve too much overhead. In particular, for an H.264 decoder, while decoding CIF images up to $6.22 \cdot 10^{23}$ frame breakups would need to be considered [52] and for each one we would have to store one voltage/frequency setting. This is obviously impossible. Even if it was, i.e. if memory was infinite, changing voltage/frequency settings of a processor incurs a significant amount of overhead in time that would undermine the purpose of the entire optimization.

Therefore, in the system scenario-based design methodology, it is necessary to cluster frames with similar breakups and merge them into one system scenario. This step corresponds to the *identification* step of Fig. 1.2. At run-time, depending on the input frame characteristics the working scenario needs to be predicted to scale the voltage and frequency values to the ones determined at design-time. However, scenario prediction may incur a significant amount of overhead. This overhead can be accounted at design time to further refine the scenario set. This corresponds to the *prediction* step of Fig. 1.2. After the scenario had been predicted it can actually be run under a configuration determined at design time (DVFS setting, scheduling, etc.). This corresponds to the *exploitation* step of Fig. 1.2.

Once the working scenario changes, i.e. once a frame belonging to another scenario is presented to the decoder, the DVFS processor settings are to be changed. However, this implies some overhead which may be large. Therefore, even when a scenario different from the current one is predicted, it is not always a good idea to switch scenarios, because the overhead may be larger than the gain. At run-time, decision on whether to switch scenario or not are made by the *switching* step of the methodology. At design-time, the switching overhead is used to further refine the scenario set.

Finally, the *calibration* step of the methodology helps with cases when it is hard or even impossible to account for the actual run-time environment at design-time. Therefore, the system is infrequently (otherwise, the overhead would obviously become too large) calibrated to optimize the average-case behavior.

1.4 System-level performance analysis

The design approaches outlined in the previous section can be fitted into the Y-char approach of Fig. 1.3.

Given an application and architectural options, the goal of the design is to by exploring different allocations of resources and binding and scheduling options produce a system of desired characteristics. This is an iterative process, well-known as *design-space exploration* (DSE) in which performance analysis plays a crucial role, as can be seen in Fig. 1.3. In each iteration of the process, different revisions of the application, architecture, allocation,

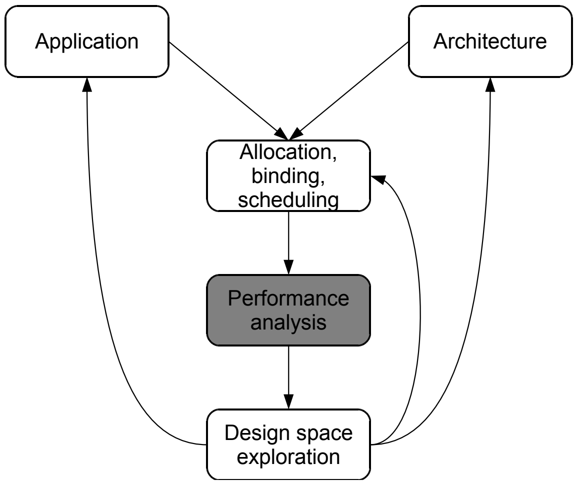


Fig. 1.3: Performance analysis in the design space exploration cycle (source [82]).

binding and scheduling decisions are considered, until performance requirements are met with minimized costs.

Therefore, providing the designer with performance indicators of the final system implementation at early design stages is of utmost importance. We refer to this analysis as *system-level performance analysis*, or shortly *performance analysis*. The performance indicators may include throughput, latency, memory demands and resource usages.

Most of the approaches for performance analysis can be broadly divided into two classes: simulation-based approaches and formal method-based approaches.

Simulation-based approaches are widely used in the industry. Moreover, several tools exist that support cycle-accurate co-simulation of complete HW/SW systems. There exist free simulation frameworks too, such as the broadly-known SystemC [1].

Simulation-based approaches can capture complex interactions and correlations in systems. Therefore, they have a large and easily customizable scope that renders them an attractive tool for performance analysis in everyday engineering practice.

However, in the parlance of [82], most of simulation-based techniques suffer from insufficient corner-case coverage. This means that they are, in most cases, unable to provide worst-case performance guarantees that is of critical importance for many embedded systems. Typically, embedded systems are very different from general purpose computing systems. This is primarily due to their dedication to control of systems and because they interact with the physical environment. Therefore, they are typically imposed with strict performance constraints. One of the most important constraints is the maximal allowed time to perform a computation which must be proven to have an upper bound in all possible circumstances. Therefore, simulation-based techniques, although very useful in system prototyping, cannot be used to attest performance properties of hard-real time (embedded) systems.

That is where formal method-based analysis techniques come into action. They do provide hard performance bounds. However, there is a tradeoff to be taken into account. In particular, as formal method-based techniques are typically not able to take into account all the complex interactions and correlations in the system they capture, the performance bounds obtained may be somewhat pessimistic. Therefore, a significant amount of effort is put into developing techniques being able to produce tighter and tighter performance bounds while maintaining computational tractability.

1.5 Problem statement

Streaming applications are applications that transform input data streams of arbitrary length to output data streams by performing some deterministic transformation. Examples are applications for video stream processing, digital communications, image processing, etc.

Modern streaming applications expose increasing levels of dynamic behavior which in essence means that their computational and communicational loads vary during run-time. Examples of such dynamic streaming applications can be, e.g., found in the domains of multi-media [97] and wireless signal processing [2] which are not static any longer, in contrast to earlier generations of algorithms and standards. These applications often require real-time processing capabilities. For streaming applications, these real-time demands are expressed as throughput and latency constraints that are considered the most important performance characteristics. Because of that when using the terms "performance analysis" or "performance metrics/characteristics" we will be typically referring to throughput and latency, unless stated otherwise. Other performance characteristics may include memory demands, resource usage, etc.

In this thesis we will primarily focus on formal method-based performance analysis for dynamic real-time streaming applications by exploiting the concept of system scenarios. We say primarily because in some contexts we will also consider some functional aspects of systems as absence of deadlock, boundedness, execution patterns of enclosed subsystems etc.

Dataflow models of computation (MoCs) have been widely used to model, design, analyze, implement and optimize such applications. This is due to their relatively simple graphical representation and their effectiveness in exposing task-level or data-level parallelism that enable achieving high performance implementations [92]. The most-well known, most used, stable and mature flavor of dataflow is synchronous dataflow (SDF) [70]. In SDF, applications are represented as SDF graphs (SDFG). SDF in its timed version comes with a rich analysis portfolio including throughput analysis [50], latency analysis [51] and buffer requirements analysis [123][43][108].

However, SDF is a static dataflow MoC, which means that properties of SDFGs are fixed and known at compile time. Therefore, SDF is incapable of tightly capturing the dynamic behavior of modern streaming applications, the emphasis being on the word tightly. Of course, in many cases one can construct an SDF model that captures the worst-case across all application behaviors. However, such an abstraction will in many cases lead to overly pessimistic worst-case performance bounds. Furthermore, the use of such pessimistic results in the design process of Fig. 1.3 leads to overallocation of resources on already resource scarce embedded processing platforms that could have been used to host another application.

Thus, different dataflow models were proposed to increase the expressive power of SDF [19][16][121][53][120][21][65][45]. However, for many of these models key analysis and verification problems are undecidable or they do not have known performance analysis techniques.

FSM-based scenario-aware dataflow (FSM-SADF) [46] is a dataflow MoC that is able to capture dynamic applications and that is design-time analyzable. Inspired by the scenario-based design methodology, FSM-SADF clusters the dynamic application data processing behaviors into a collection of *modes* or *scenarios* that are internally static and captured by SDFGs. Scenario occurrence patterns are given by a nondeterministic FSM. FSM-SADF comes equipped with various performance analysis techniques: worst-case throughput [46][98][113], worst-case latency [46][95][113] and storage space analysis techniques [109][113].

Still, FSM-SADF suffers from succinctness (compactness [21]) problems when the number of scenarios increases. In particular, FSM-SADF does not provide us with compact constructs to envelop large numbers of operation modes, i.e. scenarios. Indeed, an enumeration of these modes where every mode would be captured by a single SDFG is in theory possible but in practice it will render the size of the model unmanageable and the run-time of performance analysis prohibitive. This comes to the fore in modeling of dynamic streaming applications that expose fine-grained data-dependent behavior. In particular, such applications may have thousands or even millions of possible behaviors that render bottom-up enumeration-based clustering impractical. A representative example of such an application is the epileptic seizure predictor based on Lyapunov exponent calculator that was presented in [60][59] with up to ~ 2000000 possible behaviors. Further examples are various block-based video coding applications like the VC-1 decoder presented in [12][13] that we use as a case-study in this thesis with ~ 25000 behaviors only in case of SDTV input format. For formats of higher video quality this number would further increase.

Therefore, to be able to prove worst-case performance bounds for such applications, we need to alleviate the compactness problems of the original FSM-SADF formalism. To that end, we consider the use of parameterized dataflow based on SDF by defining a concept called SDF-based parameterized dataflow (SDF-PDF). By integrating dynamic parameters into SDF we can capture dynamic applications without the loss of resolution in a succinct manner. Furthermore, at design time, parameters help keep the size of model sizes manageable and allow models to be quickly modified or tuned for performance. Furthermore, parameterization allows to replace successive analysis of all behaviors with a single parametric analysis. At run-time, actor parameters allow for dynamic reconfiguration of models, i.e. while a model is running. For the proposed SDF parameterization we identify the semantic link between it and FSM-SADF. Using the link, we propose a parametric worst-case throughput and latency analysis scheme for SDF-PDF that is able to render significantly tighter worst-case throughput and latency bounds than the approaches that rely on construction of “SDF worst-case abstractions” of parameterized specifications.

However, applications in addition to exposing fine-grained data-dependent behavior that we capture with parameterized dataflow models may also have intricate control requirements. SDF-PDF does not provide constructs to capture these requirements which may adversely affect the tightness of SDF-PDF analysis. To alleviate this problem, we propose a refinement to SDF-PDF that is in essence generalization of the original FSM-SADF formalism where we use parameterized scenarios to capture the fine-grained data-dependent application dynamics within a scenario. In the model, application control requirements expressed in terms of allowed transitions between scenarios are captured by a nondeterministic FSM. We refer to the novel formalism as FSM-based parameterized scenario-aware dataflow (PFSM-SADF). For an SDF-based flavor of PFSM-SADF called SDF-based PFSM-SADF (SDF-PFSM-SADF) we develop corresponding worst-case throughput and latency analysis techniques.

SDF-PFSM-SADF allows us to analyze worst-case performance of applications exposing fine-grained data-dependent behavior within a superimposed control structure by means of parameterization. Parameters involved are defined and bounded across scenarios and they are allowed to change (within their bounds) every time a scenario is re-visited. Therefore, such parameters can be called *dynamic*. However, for many applications in practice values of these parameters will still be unknown *a priori*, but once set they will remain the same or will change infrequently often. We call such parameters *static*. Therefore, what we now need to analyze is worst-case performance but for given parameter values. Of course, such an analysis could be performed enumeratively by analyzing all possible parameter settings. However, parameters spanning wide intervals render the enumeration impractical or impossible. Therefore, a parametric analysis approach is needed. To that end, we propose an analysis approach that is able express the worst-case throughput of the application as a collection simple functions defined in graph parameters that can be evaluated in no-time when that is needed. E.g., a run-time manager, given the application parameter values we may need to decide whether the application can meet its throughput constraints or not. If not, it can choose not to admit the application to the system and inform higher layer protocols about the occurrence for further (possibly error) handling.

In previous problems we have dealt with performance analysis in terms of throughput and latency in context of parameterized scenarios. However, other performance and functional properties such as memory requirements and requirements on execution order of actors are oftentimes also vital parts of a typical design trajectory. The original FSM-SADF had been analyzed for some of those properties too. All of those analysis techniques are implemented in the SDF^3 tool [112]. However, the analysis of SDF^3 tool is limited to a set of predefined properties, which limits the user in defining new properties unless he/she wants to get involved into a long-lasting software development process with the goal of extending the capabilities of SDF^3. Therefore, a more general verification framework than SDF^3 is needed. Thus, translation of FSM-SADF to timed automata (TA) is proposed where the UPPAAL model checker is used to perform the actual verification. The gain is twofold. First, the analysis of user-defined performance metrics is supported. Second, by presenting FSM-SADF through a widely used and user-friendly model-checker as UPPAAL, we make it available to a wider rande of users [21].

FSM-SADF efficiently captures dynamic streaming applications by combining dataflow and finite-state control. The control in FSM-SADF is data driven, i.e. the activation of a new scenario is triggered by input data availability. An example of such an application is a typical block-based video encoder/decoder. However many systems rely on event-driven mechanism to achieve control of data-intensive system parts. E.g., these mechanisms may include handling non-synchronized event inputs from users. A typical example may be a press to a button of a remote controller that changes the TV channel and requires the video decoder to start processing a new and different video stream. To capture such systems we introduce events to FSM-SADF. Using the UPPAAL timed automata translation of the new flavor of FSM-SADF we show how to perform schedulability analysis of such systems.

1.6 Contributions

The main contributions of the thesis are listed below:

- We propose a framework for worst-case throughput and latency analysis of SDF-PDF. We base our techniques on the techniques of FSM-SADF adapted to the parametric case by exploiting the semantic link between SDF-PDF and FSM-SADF.

- We generalize the concept of FSM-SADF to the concept of PFSM-SADF to be able to succinctly capture applications exposing both fine-grained data-dependent dynamics and intricate control requirements. In addition, we develop corresponding worst-case throughput and latency analysis techniques for an SDF-based flavor of PFSM-SADF called SDF-PFSM-SADF.

- In consideration of restricted operational semantics of SDF-PFSM-SADF where parameters once set remain fixed or change infrequently, we propose a throughput analysis scheme that is able to express graph's throughput as a function of graph parameters.

- We propose a translation of FSM-SADF formalism to UPPAAL timed automata to enable a more general verification than currently supported by existing tools.

- We briefly discuss possibilities for integration of event-driven control with the FSM-SADF formalism and schedulability analysis of the combination using the UPPAAL model checker.

1.7 Thesis overview

The remainder of this thesis is organized as follows.

Chapter 2 introduces the dataflow concepts used throughout this thesis.

Chapter 3 addresses the first contribution of the thesis, that is the worst-case performance analysis of SDF-PDF. We use SDF-PDF to capture and analyze applications with fine-grained data-dependent dynamics.

In Chapter 4 we introduce PFSM-SADF and its specialization SDF-PFSM-SADF with associated worst-case performance analysis techniques. SDF-PFSM-SADF can be considered a refinement of SDF-PDF that is achieved by integrating the information on the control requirements of the application (if available) into the SDF-PDF model in form of an FSM.

For a restricted operational semantics of SDF-PFSM-SADF where parameters are deemed static (change or change infrequently often which may happen oftentimes in practice), Chapter 5 presents a throughput analysis scheme where throughput of the model is expressed as a function of parameters.

Chapter 6 leaves the parametric world of previous chapters and considers the translation of FSM-SADF to TA to enable more general verification, i.e. it goes beyond the throughput and latency analysis of previous chapters but in a nonparametric context. Therefore, Chapter 6 is not in a narrow sense related to previous chapters and can be read separately.

Finally, Chapter 7 concludes and sets directions to future work.

The last contribution concerning the modeling and schedulability analysis of FSM-SADF combined with event-driven control is briefly presented in Appendix A [1]. It is strongly correlated to Chapter 6 because as the basic modeling concept it uses the translation of FSM-SADF to TA presented in the same chapter.

Fig. 1.4 shows dependencies between chapters. In particular, if there exists an edge between Chapter X and Chapter Y, Chapter Y is said to be dependent on Chapter X. This in turn means that Chapter X should be read prior to reading Chapter Y.

[1]To present this contribution, we choose to use an appendix rather than a chapter to indicate that this is work in progress and that a significant amount of research effort is needed to complete it.

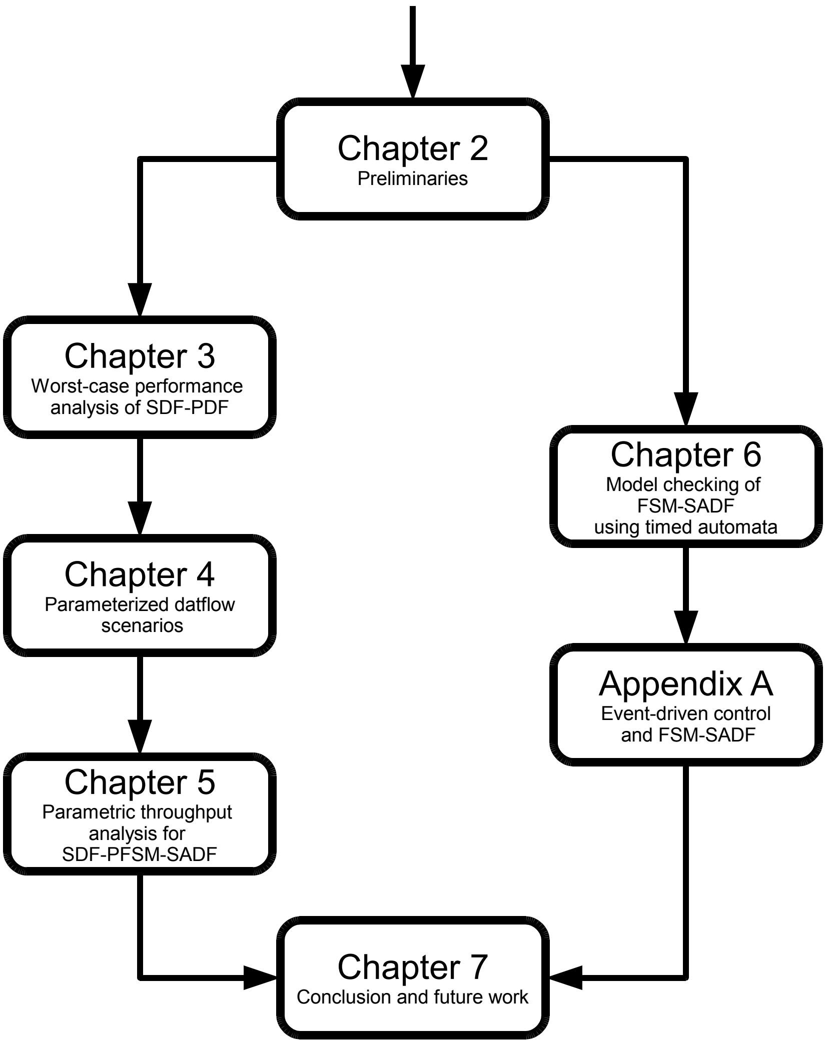

Chapter 2
Preliminaries
Chapter 3
Worst-case performance analysis of SDF-PDF
Chapter 4
Parameterized datflow scenarios
Chapter 5
Parametric throughput analysis for SDF-PFSM-SADF
Chapter 6
Model checking of FSM-SADF using timed automata
Appendix A
Event-driven control and FSM-SADF
Chapter 7
Conclusion and future work

Fig. 1.4: Chapter dependency graph.

Chapter 2

Preliminaries

This chapter introduces the dataflow modeling concepts used throughout this thesis. It starts by discussing Max-plus algebra. Max-plus algebra is a mathematical tool that will be used to capture the temporal behavior of SDF-based dataflow MoCs covered in this thesis.

2.1 Max-plus algebra

2.1.1 Definition

Define $\varepsilon \overset{\text{def}}{=} -\infty$ and $e \overset{\text{def}}{=} 0$, and denote $\mathbb{R}_{\max}$ the set $\mathbb{R} \cup \{-\infty\}$, where $\mathbb{R}$ is the set of real numbers. For elements $a, b \in \mathbb{R}_{\max}$, we define operations $\oplus$ (pronounced "o-plus") and $\otimes$ (pronounced "o-times") by

$$a \oplus b \overset{\text{def}}{=} \max(a, b) \qquad \text{and} \qquad a \otimes b \overset{\text{def}}{=} a + b. \tag{2.1}$$

Clearly, for any $a \in \mathbb{R}_{\max}$,

$$a \oplus \varepsilon = \varepsilon \oplus a = a, \quad a \otimes \varepsilon = \varepsilon \otimes a = \varepsilon, \tag{2.2}$$

and

$$a \otimes e = e \otimes a = a. \tag{2.3}$$

By Max-plus algebra [7][61] we understand the analogue of linear algebra developed for the pair of operations $(\oplus, \otimes)$ extended to matrices and vectors and denoted by $\mathcal{R}_{\max} = \{\mathbb{R}_{\max}, \oplus, \otimes, \varepsilon, e\}$.

Note that $\mathcal{R}_{\max}$ is a dioid, i.e. a semiring[1] with idempotent addition $(a \oplus a = a)$.

[1]A set K equipped with two operations $\oplus$ and $\otimes$ is a semiring if $\oplus$ is associative and commutative, $\otimes$ is associative and distributive with respect to $\oplus$, there is a zero element ε $(a \oplus \varepsilon = \varepsilon \oplus a = a, a \otimes \varepsilon = \varepsilon \otimes a = \varepsilon)$ and a unit element e $(a \otimes e = e \otimes a = a)$ [41].

2.1.2 Vectors and matrices in Max-plus

The set of n dimensional Max-plus vectors is denoted $\mathbb{R}^n_{\max}$, while $\mathbb{R}^{n\times n}_{\max}$ denotes the set of $n \times n$ Max-plus matrices. The sum of matrices A, $B \in \mathbb{R}^{n\times n}_{\max}$, denoted by $A \oplus B$ is defined by

$$[A \oplus B]_{i,j} = [A]_{i,j} \oplus [B]_{i,j}, \tag{2.4}$$

while the matrix product $A \otimes B$ is defined by

$$[A \otimes B]_{i,j} = \bigoplus_{k=1}^{n} [A]_{i,k} \otimes [B]_{k,j}, \tag{2.5}$$

where $[A \oplus B]_{i,j}$, $[A \otimes B]_{i,j}$, $[A]_{i,j}$ and $[B]_{i,j}$ are entries of matrices $A \oplus B$, $A \otimes B$, A and B, respectively, with indices i and j. For a vector $a = [a_1, \ldots, a_n]^T \in \mathbb{R}^n_{\max}$ and scalar c we use $c \otimes a$ or $a \otimes c$ to denote a vector with entries identical to entries of a with c added to each of them, i.e. $c \otimes a = a \otimes c = [a_1 + c, \ldots, a_n + c]^T$. For a vector $a \in \mathbb{R}^n_{\max}$, $||a||$ denotes the vector norm, defined as

$$||a|| = \bigoplus_{i=1}^{n} a_i. \tag{2.6}$$

For a vector $a \in \mathbb{R}^n_{\max}$ with $||a|| > -\infty$, we use a^{norm} to denote $a - ||a|| = [a_1 - ||a||, \ldots, a_n - ||a||]^T$, i.e. the normalized vector a, so that $||a^{\text{norm}}|| = 0$. With $A \in \mathbb{R}^{n\times n}_{\max}$ and $c \in \mathbb{R}_{\max}$, we use notation $A \otimes c$ or $c \otimes A$ for a matrix where $[A \otimes c]_{i,j} = [c \otimes A]_{i,j} = [A]_{i,j} + c$. The $\otimes$ symbol in the exponent indicates a matrix power in Max-plus algebra. For $A \in \mathbb{R}^{n\times n}_{\max}$ and $k \in \mathbb{N}_{>0}$,

$$A^{\otimes k} = \bigotimes_{i=1}^{k} A. \tag{2.7}$$

For scalars $c \in \mathbb{R}_{\max}$ and $\alpha \in \mathbb{R}$, $c^{\otimes \alpha} = \alpha \cdot c$. Furthermore, it is easy to verify that Max-plus matrix multiplication is linear, i.e.

$$M \otimes (a \oplus b) = M \otimes a \oplus M \otimes b \quad \text{and} \quad M \otimes (c \otimes a) = c \otimes M \otimes a \tag{2.8}$$

for all $M \in \mathbb{R}^{n\times n}_{\max}$, $a, b \in \mathbb{R}^n_{\max}$ and $c \in \mathbb{R}_{\max}$. In addition, the matrix multiplication is monotone, which means that if $a \preceq b$, then

$$M \otimes a \preceq M \otimes b. \tag{2.9}$$

Note that in (2.9), for $M, N \in \mathbb{R}^{n\times n}_{\max}$, we write $M \preceq N$ if $[M]_{i,j} \leq [N]_{i,j}$ for all $i \in 1, \ldots, n$ and $j \in 1, \ldots, n$.

2.2 Dataflow models of computation

2.2.1 The basics

Dataflow models of computation (MoC) are widely used for modeling, analyzing and implementing streaming applications. This is thanks to their simple graphical representation, compactness and the ability to expose parallelism contained in the considered application. Furthermore, the use of dataflow in a design process encourages good software engineering practices as modularity and code reuse.

Dataflow MoCs are instantiated as dataflow graphs. In such graphs, nodes are called *actors* while edges are called *channels*. Actors represent computational kernels, while channels capture the flow of streams of data values between actors. These data values are called *tokens*. The essential property of dataflow is that of an actor *firing*. Simply put, actor firing denotes the execution of an actor. Actor firing is an atomic action during which the actor consumes a certain number of tokens from input channels through its input ports, executes some behavior and produces a certain number of tokens at its output ports that are put on its output channels [124]. Firings are controlled by *firing rules* that specify the conditions for the execution of these firings [96]. These conditions are typically specified in terms of availability of input tokens, the values of input tokens and the state of the enclosing actor [57]. In timed dataflow [107] under consideration in this thesis, actor firing takes a finite amount of time called the *actor firing delay*.

In dataflow graphs, actors communicate by sending tokens along graph channels. On a channel, these tokens form *token sequences* that we define similarly as the concept of signals is defined in [71].

Definition 2.1 (Token sequence)**.** *Let V be a set of values and let T be a set of tags originating from some totally ordered continuous time domain. Let V and T include special values $\perp$ and $*$ which indicate the absence of value and an arbitrary value, respectively. We define a token sequence as a total mapping $\sigma : \mathbb{N}_{>0} \to V \times T$ denoted using square bracket and commas as follows* $[\sigma(1), \sigma(2), \ldots, \sigma(n), \ldots] = [\sigma(n)]_{n=1}^{\infty}$.

We call the set of all finite and infinite token sequences Σ where, of course, $\Sigma = 2^{V \times T}$. We denote the tuple of N token sequences as $\boldsymbol{\sigma}$ where $N \in \mathbb{N}_{>0}$. The tuple of token sequences will be denoted using parentheses, as in $([\sigma_1(n)]_{n=1}^{\infty}, [\sigma_2(n)]_{n=1}^{\infty}, \ldots, [\sigma_N(n)]_{n=1}^{\infty})$, an N-tuple with N sequences of infinite length. The set of all such tuples will be denoted Σ^N. A set of tuples will be denoted using the usual braces for sets.

Through the concept of firing, actors transform finite or infinite sequences of input tokens to finite or infinite sequences of output tokens. Input and output sequences are communicated through actor input and output ports, while the transformation between them is given by actor *firing function.*

We define a dataflow actor as follows.

Definition 2.2 (Dataflow actor). *A dataflow actor $A = (P, Q, R, f)$ is a tuple, where P is the set of actor input ports, Q is the set of actor output ports, $R \subset \Sigma^U$ is a set of finite sequences called the firing rules and $f : \Sigma^U \to \Sigma^V$ is a mapping called the firing function, where $U = |P|$ and $V = |Q|$.*

However, actors in isolation are of little use in modeling of complex systems. Therefore, we typically consider compositions of dataflow actors, i.e. dataflow graphs that we define next in accordance with [21].

Definition 2.3 (Dataflow graph). *A dataflow graph $G = (\mathcal{A}, C)$ is a directed graph, where $\mathcal{A}$ is the set of vertices representing actors while $C \subseteq \mathcal{A} \times \mathcal{A}$ is a multiset*[2] *of edges representing channels.*

According to Definition 2.3, in the most general sense, a dataflow graph is a directed graph with actors (cf. Definition 2.2) represented by vertices and channels represented by edges. Channels convey values known as tokens between the actors. Channels are conceptually FIFO queues that are defined by source and destination actors. For that matter, given a dataflow graph G, let for each $c \in C$, functions $src : C \to \mathcal{A}$ and $dst : C \to \mathcal{A}$ return the source and destination actor of c, respectively.

2.2.2 Flavors of dataflow MoCs

Dataflow MoCs can be divided into two classes: static dataflow MoCs [58] and dynamic dataflow MoCs [18].

Static dataflow MoCs are in wide use due to their predictability, strong formal properties and amenability to powerful optimization techniques [18].

Most well-known representatives of static dataflow are homogeneous synchronous dataflow (HSDF) [69], synchronous dataflow (SDF) [70] and cyclo-static dataflow (CSDF) [19]. The firing rules for these MoCs are specified in terms of availability of input tokens, regardless of their value. A firing results in the consumption of these tokens from input channels and production of tokens on output channels. These token production and consumption numbers are called *actor port rates* or simply *rates* and they form the actor

[2]Parallel channels are allowed.

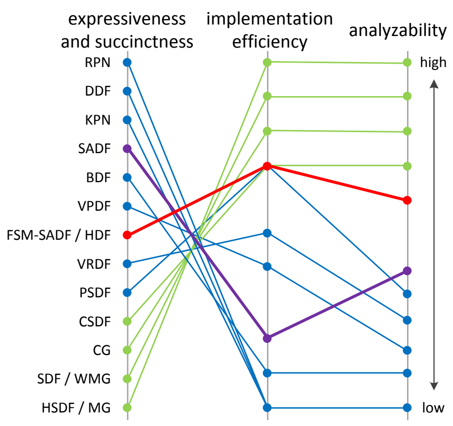


Fig. 2.1: Comparison of dataflow MoCs (source [110]).

type signature [53][63]. In HSDF and SDF, actor type signatures are fixed and known at design-time. Actually, in HSDF all rates are equal to one. In CSDF, actor type signatures can vary between actor firings as long as the variation complies to a certain type of a periodic pattern.

Static dataflow MoCs owe their "nice" properties to their restricted semantics. This restricted semantics, however, makes them an inadequate design tool choice for capturing the dynamic behavior inherent to modern streaming applications. The need for expressive power beyond that offered by static dataflow MoCs announced the dawn of the class of dataflow MoCs we call dynamic dataflow MoCs.

Most prominent dynamic dataflow models are parameterized SDF or shortly PSDF [16], variable-rate dataflow (VRDF) [121], heterochronous dataflow (HDF) [53], FSM-based scenario-aware dataflow (FSM-SADF) [46], variable-rate phased dataflow (VPDF) [120], boolean dataflow (BDF) [21], scenario-aware dataflow (SADF) [113], Kahn process networks [65], dynamic dataflow (DDF) [21] and reactive process networks (RPN) [45]. However, the increase in expressive power comes at the price of reduced analyzability and implementation efficiency as shown by Fig. 2.1 borrowed from [110]. Furthermore, many of these MoCs are either not sufficiently analyzable or do not have known performance analysis techniques at all. E.g., Buck [21] had shown that BDF and DDF are Turing complete. Consequently, it is impossible to realize an exact performance analysis for such models at design-time. Models such as PSDF and HDF although not Turing complete are untimed and therefore are not equipped with any type of analysis that would address their real-time performance properties.

At this point we do not go any further into the intrinsics of all mentioned dataflow MoCs (we will address many of them in the related work parts of the chapters to come). Instead, we focus on SDF and FSM-SADF as these models are the cornerstones of the work we present in this thesis.

2.3 Synchronous dataflow (SDF)

2.3.1 The model

SDF is the most widely used, stable and mature dataflow MoC [16]. In timed SDF actor firing delays and rates are fixed and known at design-time. SDF is a uninterpreted dataflow MoC, which means that the actual meaning of the computations and semantics of data tokens are not relevant [67]. Furthermore, the firing rules of SDF are conjunctive which implies that all actor input channels must contain sufficient quantities of input tokens for the firing to be enabled. These quantities are given by *port rates.*

Now, given an SDF actor $A = (P, Q, R, f)$ where $P = \{p_1, \ldots, p_U\}$ and $Q = \{q_1, \ldots, q_V\}$ let function $r_A : (P \cup Q) \to \mathbb{N}_{>0}$ return the rate value for a given actor port. Then the firing rule of A takes the form

$$R = \{([\sigma_{p_1}(n)]_{n=1}^{r_A(p_1)}, \ldots, [\sigma_{p_U}(n)]_{n=1}^{r_A(p_U)})\} \tag{2.10}$$

where $\sigma_{p_i}(n) = (*, \perp)$ for all $i = 1, \ldots, U$.

Firing rule of (2.10) says that in every firing A consumes $r_A(p_1)$ input tokens from port p_1 and so on until the last input port p_U from which it consumes $r_A(p_U)$ tokens regardless of their value (notation $*$). The firing rules do not depend on the availability times of input tokens, and therefore the notation $\perp$ is used.

Values $r_A(p_i)$ are fixed and known at design-time and so are the firing rules. This renders SDF a static dataflow MoC.

In consideration of the firing function of A, as SDF is an uninterpreted dataflow MoC, we abstract from the token content and consider only the timed part of the firing function given as the mapping $f^T : T^U \to T^V$ such that $f^T(n) = (\tau(q_1)(n), \ldots, \tau(q_V)(n))$ where $\tau(q_i)(n) = (\pi_r \circ \sigma_{q_i})(n) = \pi_r(\sigma_{q_i}(n))$ and π_r is the right projection function, i.e. $\pi_r((v, t)) = t$ for any $(v, t) \in V \times T$.

For a particular output port $q_i \in Q$ of actor A with firing delay d the following equation holds under self-timed execution [47]

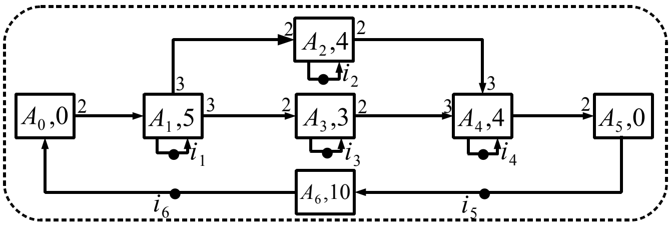
$$\begin{aligned}\tau(q_i)(n) &= d + \max_{p_i \in P} \tau(p_i)(\lfloor \frac{n}{r_A(q_i)} \rfloor \cdot r_A(p_i)) \\ &= d \otimes \bigoplus_{p_i \in P} \tau(p_i)(\lfloor \frac{n}{r_A(q_i)} \rfloor \cdot r_A(p_i)),\end{aligned} \tag{2.11}$$

where $\tau(p_i)(n) = (\pi_r \circ \sigma_{p_i})(n) = \pi_r(\sigma_{p_i}(n))$ and $\sigma_{p_i}(n)$ is the token sequence of input port $p_i \in P$. Equation (2.11) defines the Max-plus semantics of *self-timed execution* of SDF. In particular, the production time of port q_i's nth token equals to the availability time of the most delayed token needed to perform the $\lfloor \frac{n}{r_A(q_i)} \rfloor$th actor firing (max operator) increased by the firing delay d of the actor (+ operator). Self-timed execution is a schedule where every actor fires as soon as possible. The self-timed execution is of special importance as it defines the tightest bound that can be given on the temporal behavior of a system captured by an SDF model. With regard to Max-plus algebra, two fundamental concepts that determine the self-timed execution of an SDF actor are *synchronization* and *delay.* Synchronization manifests itself when an actor waits for all input tokens to become available (max operator). The delay manifests itself through the fact that the tokens that are the result of an actor firing will be available after an amount of time following the firing start time (+ operator). This amount of time is equal to the actor firing delay.

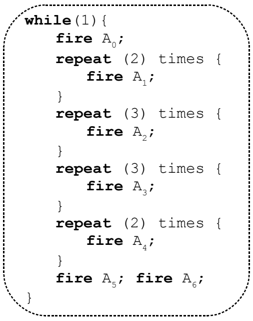
Of course, (SDF) actors operating in isolation are of limited use for modeling of complex systems. Therefore, we must consider compositions of (SDF) actors, i.e. SDF graphs (SDFGs) that we formally define in Definition 2.4.

Definition 2.4 (SDFG). *An SDFG $G = (\mathcal{A}, C, d, r, i)$ is a tuple where $\mathcal{A}$ is the a of actors, $C \subseteq \mathcal{A} \times \mathcal{A}$ is a multiset of channels, $d : \mathcal{A} \to \mathbb{R}_{\geq 0}$ returns for each actor its associated firing delay, $r : \mathcal{A} \times C \to \mathbb{N}_{>0}$ returns for each actor port its associated rate and $i : C \to \mathbb{N}_0$ returns for each channel its number of initial tokens.*

Definition 2.4 refines Definition 2.3 by introducing the concept of rates, firing delays and initial tokens. Existence of feedback loops in a dataflow graph will cause deadlock unless initial tokens are appropriately placed on graph channels forming feedback loops. The concept of initial tokens in dataflow corresponds to the concept of *initial marking* in a related MoC known as Petri nets. For more details, we refer to [80]. In this thesis as in [90], we think of initial tokens as initial conditions for the execution rather than a part of the execution itself.



(a) Example SDFG.



```
while(1){
    fire A0;
    repeat (2) times {
        fire A1;
    }
    repeat (3) times {
        fire A2;
    }
    repeat (3) times {
        fire A3;
    }
    repeat (2) times {
        fire A4;
    }
    fire A5; fire A6;
}
```

(b) Schedule of the example SDFG.

Fig. 2.2: SDF.

Fig. 2.2a shows an example of an SDFG. Actors are depicted by rectangles while port rates are annotated next to channel ends. If the value is omitted, a rate value of 1 is assumed. Actor firing delays are denoted alongside actors names. Initial tokens are depicted using black dots.

SDFGs can be scheduled at compile-time and thus implemented with minimal run-time overhead. Schedule for an SDFGs is a loop over a series of actor firings completing an iteration. Port rates can be used to unambiguously define a graph iteration, or a minimal set of actor firings that has no net-effect on the token distribution of the graph. The schedule for the running example can be denoted using the term $A_0^1 A_1^2 A_2^3 A_3^3 A_4^2 A_5^1 A_6^1$ where exponents represent actor repetition counts or as an infinite loop shown in Fig. 2.2b.

We consider SDFGs that are consistent and deadlock-free. The graph that is inconsistent may deadlock or be unbounded which means that it has no unbounded execution with bounded buffers [90]. The existence of a repetition vector implies consistency. The repetition vector of an SDFG reveals how many times a particular graph actor needs to be fired in a valid schedule/iteration. It is computed using the set of so-called balance equations [70]. We define it as a map $\Gamma : \mathcal{A} \to \mathbb{N}_{>0}$. With the abuse of notation, for the running example, $\Gamma(A_0, A_1, A_2, A_3, A_4, A_5, A_6) = (1, 2, 3, 3, 2, 1, 1)$. Nevertheless, consistency does not imply that a valid schedule exists. If a graph contains cycles, it may deadlock although consistent. That is why sufficient numbers of initial tokens must be placed in feedback channels. Checking the deadlock-freedom of an SDFG is performed by computing an iteration by abstract execution [70].

2.3.2 Max-plus algebra for SDF

Equation (2.11) shows how Max-plus algebra [7] captures the semantics of self-timed execution of SDF. In particular, (2.11) defines the Max-plus algebraic semantics of an SDF actor. However, we are interested in the Max-plus semantics of SDF at a graph level. Because SDFGs evolve in iterations, the beginning and the end time of any SDFG iteration are fully determined by the availability times of initial tokens. As mentioned, in the SDF domain, initial tokens represent initial conditions for execution [90]. If the production timestamps of initial tokens after the kth graph iteration are collected in the vector $\gamma(k) \in \mathbb{R}_{\max}^{|I|}$ the evolution of an SDFG G is given by the following recursive Max-plus linear equation

$$\gamma(k+1) = M_G \otimes \gamma(k), \tag{2.12}$$

for all $k \in \mathbb{N}_0$. In (2.12), $M_G \in \mathbb{R}_{\max}^{|I|\times|I|}$ is the SDFG Max-plus matrix, I is the set of initial tokens of the SDFG and $\gamma(k)$ is the timestamp vector of the kth SDFG iteration. Matrix M_G is a square matrix, which follows from the fact that each initial token has one entry in $\gamma(k+1)$, i.e. $\gamma(k)$.

For initial tokens, throughout this article, we use the notation i_l where $l \in \{1, \ldots, |I|\}$, so that l specifies the position of the initial token's timestamp in the timestamp vector and notation I is used for the set of graph's initial tokens.

From the recursion of (2.12), we can derive an explicit function for $\gamma(k)$ as follows

$$\gamma(k) = {M_G}^{\otimes k} \otimes \gamma(0), \tag{2.13}$$

for all $k \in \mathbb{N}_{>0}$. Matrix M_G of (2.12) and (2.13) can be derived by symbolically executing one iteration of the corresponding SDFG with the intention of relating the entries of $\gamma(k+1) = [t'_{i_1}, \ldots, t'_{i_{|I|}}]$ and $\gamma(k) = [t_{i_1}, \ldots, t_{i_{|I|}}]$ where t'_{i_l} and t_{i_l} are the timestamps of the corresponding initial tokens after the $(k+1)$st and the kth SDFG iteration embodied into the timestamp vectors of the $(k+1)$st and the kth iteration, respectively.

First, consider the following. It was shown in [44], that the production timestamp t of any graph token can be represented as a Max-plus scalar product

$$t = \bigoplus_{i_j \in I} m_j \otimes t_{i_j} = [m_1, \ldots, m_{|I|}] \otimes \gamma(k). \tag{2.14}$$

between a vector of suitable constants called the *initial token dependency vector* or shortly the dependency vector and the timestamp vector of the

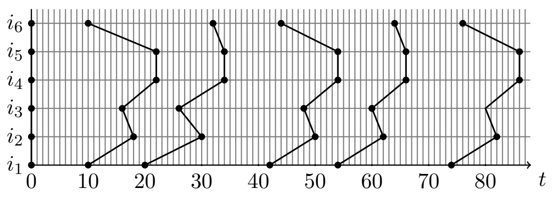


Fig. 2.3: Execution of the SDFG of Fig. 2.2a.

kth iteration. Then, also the entries of $\gamma(k+1)$ can be written as linear combinations of entries of $\gamma(k)$ as follows

$$t'_{i_l} = \bigoplus_{i_j \in I} m_{l,j} \otimes t_{i_j} = [m_{l,1}, \ldots m_{l,|I|}] \otimes \gamma(k). \tag{2.15}$$

It straightforwardly follows from (2.12) and (2.15) that dependency vectors $[m_{l,1}, \ldots m_{l,|I|}]$ define the rows of M_G. These vectors are determined by symbolic execution of one iteration of the graph as proposed by Algorithm 1 of [44].

The Max-plus matrix of the SDFG of Fig. 2.2a is given in (2.16).

$$M_G = \begin{bmatrix} 10 & -\infty & -\infty & -\infty & -\infty & 10 \\ 18 & 12 & -\infty & -\infty & -\infty & 18 \\ 16 & -\infty & 9 & -\infty & -\infty & 16 \\ 22 & 16 & 14 & 8 & -\infty & 22 \\ 22 & 16 & 14 & 8 & -\infty & 22 \\ -\infty & -\infty & -\infty & -\infty & 10 & -\infty \end{bmatrix} \tag{2.16}$$

Fig. 2.3 shows the evolution of the timestamp vector for the first five iterations of the running example SDFG.

Time is depicted horizontally and the six tokens of the timestamp vector are depicted vertically so that contours visualize the timestamp vectors of (2.12), i.e. (2.13).

An important property of SDF is monotonicity. This means that an earlier or shorter firing of an actor cannot lead to another actor firing occurring later. This property straightforwardly follows from (2.12), (2.13) and (2.9). Therefore, in worst-case temporal analysis, if actors are implemented in software, one can use their worst-case execution times (WCET) to get from the model an upper bound on the actual firing times and data production times of the real application in practice [46].

2.3.2.1 Deriving M_G

We exemplify how to derive M_G using the example SDFG of Fig. 2.2a. This graph has six initial tokens. We represent the timestamp vector of the kth graph iteration as

$$\gamma(k) = [t_{i_1}, t_{i_2}, t_{i_3}, t_{i_4}, t_{i_5}, t_{i_6}]^T. \tag{2.17}$$

Similarly, the timestamp vector of the $(k+1)$st iteration is represented as

$$\gamma(k+1) = [t'_{i_1}, t'_{i_2}, t'_{i_3}, t'_{i_4}, t'_{i_5}, t'_{i_6}]^T. \tag{2.18}$$

Recall that the iteration of the graph is given by the minimal periodic schedule of Fig. 2.2b.

According to that schedule actor A_0 fires first. In order to fire, A_0 must consume token i_6 the timestamp of which is expressed using the following Max-plus scalar product

$$t_{i_6} = [-\infty, -\infty, -\infty, -\infty, -\infty, 0] \otimes \gamma(k). \tag{2.19}$$

Therefore, according to the Max-plus semantics of SDF of (2.11), the tokens produced by its firing are determined by the timestamp vector

$$t_{i_6} \otimes 0 = [-\infty, -\infty, -\infty, -\infty, -\infty, 0] \otimes \gamma(k). \tag{2.20}$$

Thereafter, actor A_1 fires two times. In a firing, A_1 consumes one token from channel (A_0, A_1) and a token from its self-edge. The timestamp of i_1 is expressed as

$$t_{i_1} = [0, -\infty, -\infty, -\infty, -\infty, -\infty] \otimes \gamma(k). \tag{2.21}$$

The timestamps of tokens of channel (A_0, A_1) are given by (2.20). By consuming i_1 and one token from channel (A_0, A_1), the first firing of A_1 produces three tokens determined by the timestamp vector

$$\begin{aligned} &([0, -\infty, -\infty, -\infty, -\infty, -\infty] \otimes \gamma(k) \\ &\oplus [-\infty, -\infty, -\infty, -\infty, -\infty, 0] \otimes \gamma(k)) \otimes 5 \\ &= [5, -\infty, -\infty, -\infty, -\infty, 5] \otimes \gamma(k). \end{aligned} \tag{2.22}$$

Similarly, the second firing consumes the self-edge token and the remaining token from (A_0, A_1) whose timestamp is given by (2.20). However, now the self-edge token carries the timestamp of (2.22) as it was produced in the first firing of A_1. Therefore, the next three tokens produced by A_1 carry the timestamp

$$\begin{aligned}&([5, -\infty, -\infty, -\infty, -\infty, 5] \otimes \gamma(k)\\ &\oplus [-\infty, -\infty, -\infty, -\infty, -\infty, 0]) \otimes 5\\ &= [10, -\infty, -\infty, -\infty, -\infty, 10] \otimes \gamma(k)\end{aligned} \tag{2.23}$$

The second firing of A_1 restores i_1 because this is also the last firing of A_1 within the iteration. Therefore,

$$t'_{i_1} = [10, -\infty, -\infty, -\infty, -\infty, 10] \otimes \gamma(k). \tag{2.24}$$

By continuing the symbolic execution until the completion of the iteration we similarly obtain the new timestamps of remaining initial tokens as follows

$$\begin{aligned}t'_{i_2} &= [18, 12, -\infty, -\infty, -\infty, 18] \otimes \gamma(k),\\ t'_{i_3} &= [16, -\infty, 9, -\infty, -\infty, 16] \otimes \gamma(k),\\ t'_{i_4} &= [22, 16, 14, 8, -\infty, 22] \otimes \gamma(k),\\ t'_{i_5} &= [22, 16, 14, 8, -\infty, 22] \otimes \gamma(k),\\ t'_{i_6} &= [-\infty, -\infty, -\infty, -\infty, 10, -\infty] \otimes \gamma(k),\end{aligned} \tag{2.25}$$

By organizing the dependency vectors of (2.24) and (2.25) into matrix rows, we obtain M_G of (2.16).

2.4 FSM-based scenario-aware dataflow

2.4.1 Basic concepts

As already stated, the concept of synchronous dataflow scenarios [44] extends the expressive power of SDF by combining streaming data and finite control into a MoC called FSM-SADF [46]. More precisely, application behaviors are clustered into a group of static modes of operation called *scenarios* each modeled by an SDFG, while scenario occurrence patterns are constrained by a nondeterministic FSM. Consequently, an FSM-SADF graph (FSM-SADFG) evolves in iterations of its scenario SDFGs.

From the perspective of an FSM-SADF actor this means that an actor, within the execution of an FSM-SADFG it is a part of, operates in different scenarios. In each scenario actor may attain a different type signature and a different firing delay. We formalize this concept as follows.

Let $A = (P, Q, R, f)$ be an FSM-SADF actor where $P = \{p_1, \ldots, p_U\}$ and $Q = \{q_1, \ldots, q_V\}$. Let $\mathcal{S}_A = \{s_{A1}, \ldots, s_{AZ}\}$ be the set of scenarios of A.

where $Z = |\mathcal{S}_A|$. Let function $r_A : (P \cup Q) \times \mathcal{S}_A \to \mathbb{N}_{>0}$ give the rate value for a given actor port and a given scenario. Let function $d_A : \mathcal{S}_A \to \mathbb{R}_{\geq 0}$ return the firing delay of an actor for a given scenario. Then, the general firing rule of an FSM-SADF actor is as follows

$$R = \{([\sigma_{p_1}(n)]_{n=1}^{r_A(p_1, s_{Ai})}, \ldots, [\sigma_{p_U}(n)]_{n=1}^{r_A(p_U, s_{Ai})})_{i=1}^{Z}\}, \tag{2.26}$$

where $\sigma_{p_i}(n) = (*, \bot)$ for all $i = 1, \ldots, U$. Unlike an SDF actor that has only one firing rule (cf. (2.10)), an FSM-SADF actor has as many firing rules as there are scenarios (notice that R of (2.26) is composed of Z tuples). As inherited from SDF, in FSM-SADF tokens are uninterpreted and the firing rules do not depend on token arrival times.

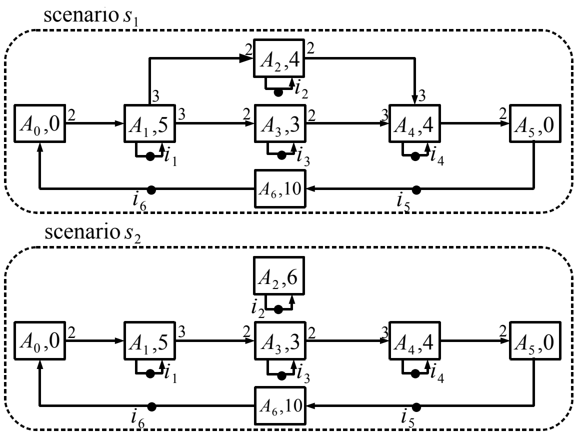
Correspondingly, we proceed by defining the timed firing function of FSM-SADF for an arbitrary $q_i \in Q$ as follows

$$\begin{aligned} \tau(q_i)(n) =& d_A(\overline{s}_Q(q_i, L)) \\ & \otimes \bigoplus_{p_i \in P} \tau(p_i)(\sum_{j=1}^{L} r_A(p_i, \overline{s}_Q(q_i, j))). \end{aligned} \tag{2.27}$$

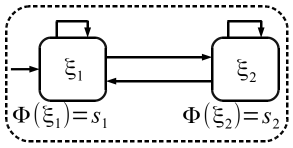
In (2.27), $\overline{s}_Q : Q \times [1, \ldots, L] \to \mathcal{S}_A$ where $L \in \mathbb{N}_{>0}$ defines the scenario sequence that lead to the production of token $\sigma_{q_i}(n) = (*, \tau(q_i)(n))$. Admissible sequences, i.e. scenario occurrence patterns are given by the scenario FSM. Unlike SDF actors (cf. (2.10), (2.11)), FSM-SADF actors (cf. (2.26), (2.27)) across different firings in different scenarios consume and produce different numbers of tokens. Similarly, their firing delays differ from one scenario to the other. Thus is FSM-SADF a dynamic dataflow MoC. Note that different scenario sequences will result in different evaluations of (2.27) for the same n. Therefore, it would be mathematically correct to call τ, i.e. $\tau(q_i)(n)$ a relation rather than a function with $\overline{s}_Q$ as its parameter. But to stay in correspondence with the existing dataflow literature [72] we abuse the notion of a function.

Naturally, FSM-SADF actors are composed into graphs. An example of an FSM-SADFG is shown in Fig. 2.4. The graph has two scenarios: s_1 and s_2 modeled by two SDFGs. The difference between the scenarios[3] is that scenario s_2 misses channels (A_1, A_2) and (A_2, A_4), while the firing delay of actor A_2 is different than the firing delay of the same actor in s_1. The scenario FSM has two states where each of the states corresponds to one scenario. In the figure, state ξ_1 corresponds to s_1, while ξ_2 corresponds to s_2. The scenario FSM defines admissible scenario sequences. The operational semantics of the model is as follows: every transition in the scenario

[3]We will be using the terms scenario and the scenario SDFG interchangeably



(a) Scenario SDFGs.



(b) Scenario FSM.

Fig. 2.4: Example FSM-SADFG.

FSM schedules the execution of one iteration of the SDFG that models the scenario corresponding to the transition's destination state. From the perspective of FSM-SADF actors, this means that their execution within one scenario is governed by one firing rule and one firing function. In that case, (2.26) and (2.27) reduce to (2.10) and (2.11), i.e. within an FSM-SADFG iteration, an FSM-SADF actor reduces to an SDF actor. The dynamic behavior of FSM-SADF is defined across iterations, while within one the behavior is static. Furthermore, an FSM-SADF actor where $|\mathcal{S}_A| = 1$ is an SDF actor. Therefore, FSM-SADF generalizes SDF. We give the formal definition of FSM-SADF as adopted from [98]. First we define the scenario FSM as follows.

Definition 2.5 (Scenario FSM). *Given a set $\mathcal{S}$ of scenarios, a scenario FSM F on $\mathcal{S}$ is a tuple $F = (\Xi, \xi_0, \mathbb{T}, \Phi)$, where Ξ is the set of states, ξ_0 is the initial state, $\mathbb{T} : \Xi \to 2^{\Xi}$ is the transition function and $\Phi : \Xi \to \mathcal{S}$ is the scenario labeling.*

Thereafter, we define FSM-SADF in Definition 2.6.

Definition 2.6 (FSM-SADF). *FSM-SADF F is a tuple $\mathsf{F} = (\mathcal{S}, F)$ where $\mathcal{S}$ is the set of scenarios and F is an FSM on $\mathcal{S}$.*

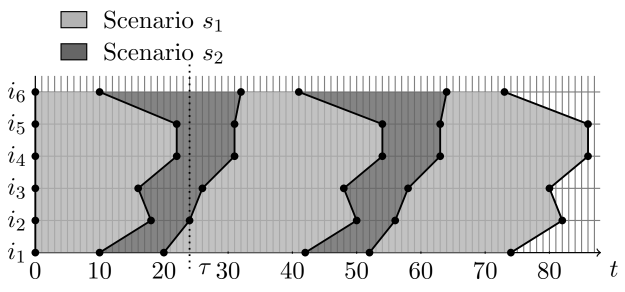


Fig. 2.5: Execution of FSM-SADFG of Fig. 2.4.

Note that in Definition 2.6 scenario SDFGs are included in the FSM-SADF (although not explicitly mentioned) through the set $\mathcal{S}$, i.e. we make no difference between the terms scenario and scenario SDFG.

2.4.2 Max-plus algebra for FSM-SADF

Because an FSM-SADFG evolves in iterations of its SDF constituents, i.e. scenario SDFGs, the Max-plus algebraic semantics of SDF is naturally carried over to FSM-SADF. In particular, a sequence of scenarios can be associated with a sequence of timestamp vectors $\gamma(0), \gamma(1), \ldots$ where

$$\gamma(k+1) = \mathcal{M}_{\mathsf{F}}(\zeta_{\mathsf{F}}(k+1)) \otimes \gamma(k). \tag{2.28}$$

In (2.28), $\mathcal{M}_{\mathsf{F}} : \mathcal{S} \to \mathbb{R}_{\max}^{|I|\times|I|}$, returns the Max-plus matrix of the scenario SDFG, $\zeta_{\mathsf{F}} : \mathbb{N}_{>0} \to \mathcal{S}$ returns the scenario of the $(k+1)$st FSM-SADFG iteration. Fig. 2.5 shows the execution of the running example FSM-SADFG. As before, time is depicted horizontally and the six tokens of the timestamp vector are depicted vertically so that contours visualize the timestamp vectors of (2.28). Let

$$\overline{s} = s_1, \ldots, s_k \in \mathcal{S}^* \cap L \tag{2.29}$$

denote a sequence of scenarios where L defines a restriction of $\mathcal{S}^*$ determined by the scenario FSM. Note that * stands for Kleene star. It had been shown in [46] that the completion time of (2.29) can be defined as follows

$$\mathcal{A}(\overline{s}) = \alpha^T \otimes \mu(\overline{s}) \otimes \beta, \tag{2.30}$$

where α is the final delay, $\mu : \mathcal{S}^* \to \mathbb{R}_{\max}^{|I|\times|I|}$ is the morphism that associates sequences of scenarios with Max-plus matrices as follows

$$\mu(\overline{s}) = \mathcal{M}_{\mathsf{F}}(s_k) \otimes \ldots \otimes \mathcal{M}_{\mathsf{F}}(s_1) \tag{2.31}$$

and β is the initial delay. The initial delay β specifies the initial enabling time of initial tokens and typically $\beta = \mathbf{0}$, while the final delay α serves as a mean to specify the metrics we are interested in. E.g., if we are interested in the makespan of a sequence of scenarios, we set $\alpha = \mathbf{0}$. The triple $\mathcal{A} = (\alpha, \mu, \beta)$ defines a Max-plus automaton [40].

In Max-plus automata, the concurrency features of a system are modeled by the possible choices between the letters that represent the static subschedules the system evolves in, and the synchronization features are implemented by the Max-plus algebra. In particular, Max-plus automata covers a class of systems with synchronization phenomena and variable schedules. Therefore, the completion time of a variable schedule can be given by a product of matrices describing the synchronization between particular static subschedules and vectors specifying the initial state of the system and the metrics we are interested in (cf. (2.30)).

On the other hand, FSM-SADF evolves in iterations of its nondeterministically chosen scenario SDFGs which are internally static. These scenario SDFGs are finite dimensional causal stationary recurrent Max-plus systems (cf. (2.12) and (2.13)). The schedule of an FSM-SADFG is determined by the driving scenario sequence. The synchronization between scenarios is determined by the availability of initial tokens that exist between them.

Therefore, Max-plus automata can be used to capture the temporal behavior of FSM-SADF as letters that drive the Max-plus automata correspond to FSM-SADFG scenarios. Synchronization between scenarios is captured by scenario matrix products. Each matrix “implements” the self-timed schedule of one scenario SDFG iteration that is in turn a subschedule of the enclosing FSM-SADFG.

If we take another look at Fig. 2.5, in the parlance of Max-plus automata theory, we say that the contours in the figure are recognized by a Max-plus automaton of 2.30 driven by (2.29).

The theory of Max-plus automata had been used in [46] to analyze worst-case performance of FSM-SADF. We leave this matter aside now and explain it in detail in the chapter to come when we reformulate the FSM-SADF results and apply them to SDF-PDF.

2.5 Summary

In this chapter, preliminary dataflow concepts were presented. These concepts will be used throughout this thesis. Special focus was put on the Max-plus algebraic semantics of SDF and FSM-SADF as we will heavily depend on those when elaborating the Max-plus algebraic semantics of SDF-PDF in Chapter 3 and SDF-PFSM-SADF in Chapter 4. Furthermore, the Max-plus algebraic semantics of FSM-SADF will be used to define a policy that assures determinacy of the FSM-SADF in the model-checking context of Chapter 6.

Chapter 3

Worst-case performance analysis of SDF-based parameterized dataflow

As discussed both in Chapter 1 and Chapter 2, dynamic dataflow MoCs have been introduced to provide designers with sufficient expressive power to capture increasing levels of dynamism in modern streaming applications.

When comparing these models, one must take into account their expressiveness, compactness, analyzability (ease of analysis) and intuitive appeal [21].

When compared to other dynamic dataflow MoCs, FSM-SADF offers a good tradeoff between expressiveness and analyzability. However, in consideration of streaming applications with fine-grained data-dependent dynamics, FSM-SADF suffers from compactness problems. Simply put, if FSM-SADF is to be used to tightly capture the behavior of such applications, an enormous number of scenarios needs to be foreseen. This in turn makes the graph explode is size (renders the model size unmanageable) and makes the analysis run-time prohibitive. Therefore, to treat such applications, we need to look for a more compact model than FSM-SADF.

Among dynamic dataflow MoCs, parameterized dataflow MoCs hold an important place. This is due to the fact that they by integrating dynamic parameters and run-time adaptation of parameters in a structured way allow for a compact representation of fine-grained data-dependent dynamics inherent to many modern streaming applications.

However, these models have been primarily analyzed for functional behavior and correctness, while the analysis of their temporal behavior has attracted less attention.

This chapter analyzes worst-case performance metrics (throughput and latency) of an important class of parameterized dataflow MoCs based on synchronous dataflow where we allow for an explicit representation of the dependencies between parameters. We refer to such models as SDF-based parameterized dataflow (SDF-PDF).

To achieve this, we first introduce the Max-plus algebraic semantics of SDF-PDF. Thereafter, we model run-time adaptation of parameters using the theory of Max-plus automata by exploiting the semantic link between FSM-SADF and SDF-PDF based on one-to-one correspondence between FSM-SADF scenarios and SDF-PDF configurations/instances where an adaptation of parameters in SDF-PDF corresponds to a scenario transition in FSM-SADF. Finally, we show how to derive the worst-case performance metrics from the resulting Max-plus automaton structure.

We evaluate our approach on a representative case study from the multimedia domain.

Different parts of this chapter have been published in [105][101][99] and are being considered for publication in [100].

3.1 Introduction

In the strictest sense, making a comparison of different flavors of dynamic dataflow MoCs is not an easy task. The work of [110] compares dataflow models based on their expressiveness and succinctness, analyzability and implementation efficiency (cf. Fig. 2.1). Prior to [110], Buck [21] mentioned similar criteria: expressive power, compactness[1], ease of analysis and intuitive appeal.

As discussed in Chapter 1, FSM-SADF lies well-positioned in the tradeoff space defined by expressiveness and analyzability. In particular, in contrast to many other dynamic dataflow MoCs it nourishes a high level of design-time analyzability. However, if one wishes to use FSM-SADF for modeling of streaming applications that expose fine-grained data-dependent dynamics, he/she will soon face compactness problems. Such applications may have thousands or even millions of possible behaviors which makes bottom-up clustering (or even worse enumeration) of these into scenarios impractical [60]. In particular, due to a large number of scenarios, the model size will soon become unmanageable and the analysis run-time prohibitive. This is because the symbolic simulation of a scenario SDFG to generate its Max-plus matrix (cf. Algorithm 1 of [44]), especially for graphs with large

[1]We will be using terms compactness and succinctness interchangeably.

repetition vectors, is both time and memory demanding (cf. Fig. 11 of [46]). On the other hand, if one tries to keep the scenario number moderate, the analysis may be overly pessimistic.

Therefore, in the context of streaming applications with fine-grained data-dependent dynamics an alternative to FSM-SADF needs to be considered.

In particular, we consider parameterized dataflow as a meta-modeling technique that integrates parameters and run-time adaptation of parameters into a certain class of dataflow MoCs we refer to as *base models* [18]. This way, using parameters, one is able to express fine-grained data-dependent dynamics of modern streaming applications in a compact way. This has repercussions both at design- and run-time. At design time, parameters help keep the size of models/libraries manageable and allow models to be quickly modified or tuned for performance. At run time, parameters allow for dynamic reconfiguration of the model while it is running [81].

In this work, we are interested in parameterized dataflow MoCs where SDF serves as the base model. Such models are of special importance, as SDF is considered the most stable and mature dataflow MoC.

Examples of such MoCs are parameterized SDF (PSDF) [16], schedulable parametric dataflow (SPDF) [38], boolean parametric dataflow (BPDF) [12] and variable rate dataflow (VRDF) [121].

We refer to such models, obtained by parameterization of SDF (in terms of rates and actor firing delays in the timed dataflow context) as SDF-based parameterized dataflow (SDF-PDF). Furthermore, in SDF-PDF we allow for an explicit representation of the dependencies between parameters. Although such models have been extensively analyzed for functional behavior and correctness, the analysis of their temporal behavior, in particular analysis of their performance metrics such as throughput and latency, had received far less attention. One reason is that all MoCs mentioned above except VRDF are untimed and therefore inherently not accompanied by temporal behavior analysis techniques. The second reason is that in case where only firing delays are parameterized, one can easily imagine an implied analysis for such models. This analysis would include an initial step where a certain "SDF-based worst-case abstraction" of the original parameterized specification is constructed and subjected to standard SDF performance analysis techniques [50][51]. The information needed to construct such a "worst-case SDF abstraction" would include the upper endpoints of parameterized actor firing delay intervals (assuming these are initially box constrained). The validity of such an abstraction follows from the monotonicity property of SDF [44] that SDF-PDF inherits.

However, using these endpoints will often incur significant amounts of pessimism. E.g., if actors are implemented in software their firing delays correspond to worst-case execution times (WCETs) of associated software modules. It is often the case that these WCETs depend on the module inputs in very complex ways. Paper [4] lists a few examples of applications where WCETs are expressed as polynomial functions of application inputs. Therefore, taking the upper endpoints of default parameter intervals and not considering these dependencies will most definitely incur a significant amount of pessimism which results in a decrease of the optimization margin a designer has at hers/his disposal.

The case of graphs containing parameterized rates is even more complicated, as these necessarily do not influence the temporal behavior of the model in a "monotonic" way. By this we mean that increasing the value of a rate in the graph does not necessarily need to lead to a later completion of the graph's iteration. Quite the contrary, this may lead to an earlier completion of an iteration. To the best of our knowledge, there exists no systematic approach that would determine the "worst-case SDF abstraction" of a SDF-PDF-like structure with parameterized rates. The problem gets even more complicated if these rates show functional dependence on other rates, design environment parameters or input signal parameters what we indeed do allow to occur in SDF-PDF.

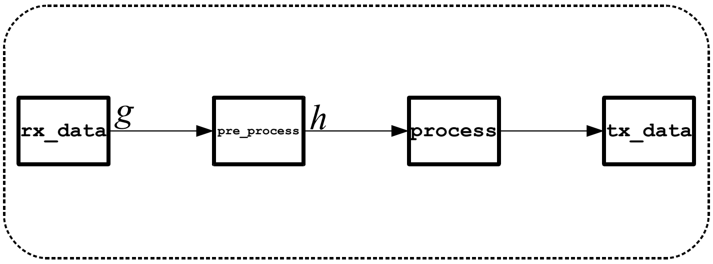
The aforementioned justifies the need for novel worst-case parametric performance analysis techniques that by operating directly on parameterized graphs (with interdependent parameters) remove the need for the touchy construction process of "SDF-based worst-case abstractions" of original parameterized specifications is such exist at all.

In this work we present such a worst-case performance analysis framework for SDF-PDF specifications in consideration of certain technical constraints we impose on the input graph structures. Within the framework, we consider self-timed execution of SDF-PDF structures. We base our approach on the Max-plus algebraic [7] semantics of self-timed execution of SDF that SDF-PDF inherits. We model parameter reconfigurations using the theory of Max-plus automata [40] by exploiting the Max-plus semantic equivalence of SDF-PDF parameter reconfigurations and scenario transitions in FSM-SADF. By subjecting the derived Max-plus automaton structure to appropriate analysis, we are able to derive the relevant worst-case throughput and latency metrics for SDF-PDF.

```
extern int rx_data(uint*,uint*)
extern int pre_process(int, uint);
extern int process(int, uint);
extern void tx_data(int);

void main(void){
    uint g, h;
    int res1, res2, res3;
    while(1){
        res1 = rx_data(&g, &h);
        for(uint i=0; i < g; i++){
            res2 = pre_process(res1, i);
            for(uint j=0; j < h; j++){
                res3 = process(res2, j);
                tx_data(res3);
            }
        }
    }
}
```

(a) C specification.



(b) Dataflow specification.

Fig. 3.1: Motivational example.

3.2 Motivational example

Parameterized dataflow MoCs are important as they through the use of dynamic parameters allow for a succinct representation of applications exposing fine-grained data-dependent dynamics. Furthermore, they define precise semantics for parameter reconfiguration across application activations.

A motivational example of a synthetic DSP application that illustrates the importance of parameterized dataflow as a modeling and analysis concept is shown in Fig. 3.1. The `C` specification of the example application is shown in Fig. 3.1a. The application consists of two nested loops with bounds `g` and `h`. The loop bound values are input data-dependent as computed in the `rx_data` module. The actual implementation of the `rx_data` module is assumed to involve complex input data processing. The derived bounds are assumed to depend on some characteristics of the input signal. Across different application activations, parameters take different values. Assume that bound `g` can be assigned with a value originating from the interval $[0, m/2]$ while `h` can be assigned with a value from $[0, n/2]$. It this case, the application will attain as many behaviors as there are integer points in the rational 2-polytope $P_{m,n}$ given by the set of constraints $\{0 \leq m/2, 0 \leq n/2\}$. With $m = 2001$ and $n = 4500$ the specification of Fig. 3.1a abstracts $2,252,126$ application behaviors [27] and defines $2,252,126^2$ possible transitions between pairs $(\texttt{g}, \texttt{h})$ from one application activation to the other. Therefore, we can say that the application exposes fine-grained data-dependent dynamism. In FSM-SADF, the application could be represented as an FSM-SADF with an unacceptably large number of scenarios, i.e. $2,252,126$. The analysis run-time for such a structure would be prohibitively large because Algorithm 1 of [44] would have to be run $2,252,126$ times, i.e. once per scenario SDFG. Furthermore, if we assume that the SDF^3 tool [112] uses 1kB to store the configuration of a scenario, the complete configuration file size only for this example would be cca. 2GB.

Luckily, data-dependent behavior of the application can be succinctly expressed (as a single entity) using the parameterized dataflow structure of Fig. 3.1b where loop bounds are denoted using parameterized graph rates (actor firing delays are implied). This way, at design-time, we keep the model size manageable. In particular, we avoid the need for enumeration of $P_{m,n}$ that would result in $2,252,126$ non-parameterized dataflow structures accounting for all $(\texttt{g}, \texttt{h})$ pairs. At run-time, parameters express the application reconfiguration in a natural and intuitive fashion, i.e. as a (simple) change of parameter values.

Assuming now that each module is mapped to a separate processing element the worst-case performance analysis for the motivational example is difficult due to several reasons.

First, the application is dynamic, i.e. in every activation the values of loop bounds `g` and `h` differ from those in the previous activation.

Second, the consecutive activations of the application will be pipelined, i.e. they may be active at the same time.

Third, consecutive activations are inter-dependent as a module in the current activation cannot commence execution before all executions of the same module of the previous activation have completed because they share the same processing element.

The outlined importance of parameterized dataflow in modeling of dynamic streaming applications justifies the need for a framework that can address the worst-case performance analysis for such applications. This is a challenging problem the solving of which we dedicate the remainder of this chapter.

3.3 Related work

We begin by providing more insight into the class of parameterized dataflow MoCs based on SDF whose main members were listed in Section 3.1.

We start with PSDF [16]. PSDF introduces parameters to SDF actors that control their functionality and/or their dataflow behavior. PSDF concept enables hierarchical integration where PSDF graphs can be abstracted into actors in higher PSDF levels. It is of vital importance, that the interface dataflow of a hierarchical actor remains unchanged throughout any iteration of its hierarchical parent actor. This way, one maintains a level of predictability and permits efficient quasi-static scheduling at least for a class of PSDF specifications that satisfy certain technical constraints regarding the number of initial tokens placed on feedback channels.

SPDF [38] is a MoC closely related to PSDF. SPDF explicitly define requirements that a parameterized dataflow specification must satisfy so that questions about liveness (deadlock freedom), boundedness and schedulability can be answered at compile-time. In contrast to SPDF, PSDF employs run-time mechanisms that check the consistency and bounded memory consistency of a specification.

BPDF [12] is a syntactical extension of SPDF developed to elegantly treat cases where actor port rates may be 0. This is achieved by the introduction of conditional channels annotated with boolean expression. Depending on the value the expression attains at run-time, channel is to be activated or deactivated. Deactivation infers that no consumption or production can take place at that channel.

VRDF [121] introduces facilities for frequent changes of actor port rates by means of parameterization. However, VRDF defines strong structural constraints that must be satisfied for achieving boundedness. More precisely, every production of p tokens must be coupled by exactly one consumption of p tokens. In addition, these pairs must be well-parenthesized in the graph.

As originally proposed, all these models, except VRDF are untimed and consequently not accompanied by any kind of temporal analysis. Paper [121] discusses the temporal aspects of VRDF. However, it is of limited scope as it considers conservative buffer size computations under a given throughput constraint.

Some results exist, though, on parametric throughput analysis of SDF and HSDF. In particular, the authors of [48] add the notion of parameterized actor firing delays to SDF. Consequently, via state-space analysis embedded in a divide and conquer algorithm, one can obtain expressions for the throughput of the graph expressed as functions of parameters. However, the parametric analysis is limited only to actor firing delays, while the rates are kept constant. The second and even more important difference between the work of [48] and ours is in the semantics of parameterization. In particular, the purport of parameterization in [48] is syntactical because parameters are static, i.e. once set they do not change. Therefore, the static nature of SDF is preserved. Nevertheless, parameterization renders the analysis more complex as seen in the same work. In our work, parameterization of SDF implies dynamic parameters, i.e. parameters whose values change at run-time. Therefore, SDF-PDF does not inherit the static nature of SDF, but is a dynamic dataflow MoC. Poplavko et al. [88] use parametric expressions to derive upper bounds for firing delays of actors of HSDF graphs. In the same work, these graphs are used to define exact timing models for capturing computation and communication of independent jobs running on a multiprocessor network-on-chip platform. However, as in [48] these parameters are static and moreover, they are even known at design-time which means that in contrast to [48] the analysis of [88] at HSDF graph level is not parametric. The work of [33] applies the technique of [48] to FSM-SADF which is a dynamic dataflow MoC by adding parameterized actor firing delays to FSM-SADF. However, rates are left constant within scenarios. Furthermore, FSM-SADF implies a reasonably sized set of application scenarios/modes/behaviors and is not appropriate for capturing the fine-grained data-dependent application dynamics. Authors of [87] also consider scenarios of SDF behavior, but in their case only HSDF graphs are considered (graphs in which all consumption and production rates are equal to one, i.e. constant). Parameters denoting actor firing delays are used to derive timing overlap between scenarios. However, the timing model of [87] as the analysis of FSM-SADF will also experience compactness issues as the number of scenarios grows.

3.4 SDF-based parameterized dataflow

In this section we formally present SDF-PDF as a dynamic dataflow MoC obtained by applying parameterization to SDF. We define SDF-PDF as a model that can be used to capture the existing parameterized dataflow MoCs based on SDF such as PSDF, SPDF, BPDF and VRDF. This way these models can be subjected to the performance analysis of SDF-PDF we define in Section 3.7 if they comply to certain restrictions we put on the input graph structure.

We proceed by first focusing on SDF-PDF actors in Subsection 3.4.1, that we compose into SDF-PDF graphs (SDF-PDFGs) in Subsection 3.4.2. Thereafter in Subsection 3.4.3 we define the subset of SDF-PDF we will use for performance analysis of systems. Finally, in Subsection 3.4.4 we discuss the semantic link between our analysis model and FSM-SADF that is the foundation of the Max-plus algebraic semantics of SDF-PDF we present in Section 3.5.

3.4.1 SDF-PDF actors

We start by defining our parameterization scope. Having in mind SDF as an uninterpreted dataflow MoC with conjunctive firing rules, we consider parameterization of SDF rates and actor firing delays. As we wish to produce a dynamic dataflow MoC, we declare our parameters dynamic, i.e. we allow for parameter values to change at run-time.

Rate parameterization concerns data-dependent dynamics that the designer wishes to expose at a module, i.e. actor level. Therefore, parameterized rates capture data-dependent variation of communication patterns between actors. On the other hand, parameterized firing delays are used to capture the temporal effect of data-dependent dynamics within modules represented by actors on the overall composition, i.e. the graph. Moreover, we may say that by using parameterized firing delays the designer hides module implementation details irrelevant at a particular abstraction level. E.g., if actors are implemented in software, their firing delays correspond to worst-case execution times (WCET) of modules they represent. These in turn may be represented as parametric expressions defined in terms of input parameters to the module or maximal iteration counts of module loops [73][4].

We now define the concept of an SDF-PDF actor and the semantics of the parameterization employed. Let $A = (P, Q, R, f)$ be an SDF-PDF actor. Actor port rates are parameterized using parametric arithmetic expressions. We let the production of these expressions be governed by an arbitrary grammar $\mathcal{R}_A$ defined over a set of symbolic variables $\mathcal{P}_{Ai}$ by default constrained to the set of nonnegative integers. To exemplify, consider the SDF-PDF actor A shown in Fig. 3.2. The actor has one input and one output port annotated with rates $1 + u$ and $u + 2v$, respectively, that are

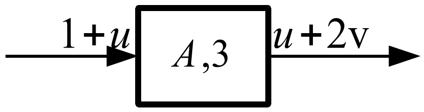


Fig. 3.2: SDF-PDF actor.

generated by the grammar

$$\mathcal{R}_A := k \mid k \cdot p \mid \mathcal{R}_{A1} + \mathcal{R}_{A2} \tag{3.1}$$

where $k \in \mathbb{N}_{>0}$ and $p \in \mathcal{P}_{Ai} = \{u, v\}$. The definition of $\mathcal{R}_A$ of (3.1) allows for rates that are linear combinations of parameters u and v.

As we will witness soon, parameterization of actor port rates has repercussions on both the actor firing rules and the actor firing function. Similarly, we let actor firing delay be parameterized by a parametric expression generated by an arbitrary grammar $\mathcal{D}_A$ defined over a set of symbolic variables $\mathcal{P}_{Ad}$ by default constrained to the set of nonnegative real numbers. Parameterization of firing delays will influence the (timed) actor firing function. For actor A of Fig. 3.2,

$$\mathcal{D}_A := k, \tag{3.2}$$

where $k \in \mathbb{R}_{\geq 0}$ and $\mathcal{P}_{Ad} = \emptyset$, i.e. we only allow for constant firing delays.

Now, given sets of parameters $\mathcal{P}_{Ai}$ and $\mathcal{P}_{Ad}$ let x^A denote a *configuration* of A that is obtained by assigning values to all parameters of $\mathcal{P}_{Ai}$ and $\mathcal{P}_{Ad}$. Simply put, a configuration is a set of parameter values (one value for each parameter). Furthermore, let X_A denote the *domain* of A that is the set of all configurations of A with $Z = |X_A|$.

For an actor A, let function $r_A : (P \cup Q) \times X_A \to \mathbb{N}_0$ given an actor port and an actor configuration return its rate. It does so by evaluating the parametric arithmetic expression generated by $\mathcal{R}_A$ valid for that port at a specific $x^A \in X_A$. In particular, parameters in the parametric arithmetic expressions are replaced by their values that are specified in the configuration. This gives us a concrete arithmetic expression that when evaluated yields the rate value of the port as nonnegative integer. Similarly, let $d_A : X_A \to \mathbb{R}_{\geq 0}$ given a configuration return the firing delay of the actor. It does so by evaluating the parametric arithmetic expression generated by $\mathcal{D}_A$ that gives the parameterized firing delay of A for $x^A \in X_A$. In particular, parameters in the parametric arithmetic expressions are replaced by their values that are specified in the configuration. This gives us a concrete arithmetic expression that when evaluated yields the firing delay of the actor as a nonnegative real number.

With the notations above, the general firing rule of an SDF-PDF actor A is as follows

$$R = \{([\sigma_{p_1}(n)]_{n=1}^{r_A(p_1,x_i^A)}, \ldots, [\sigma_{p_U}(n)]_{n=1}^{r_A(p_U,x_i^A)})_{i=1}^{Z}\}, \tag{3.3}$$

where $\sigma_{p_i}(n) = (*, \perp)$ for all $i = 1, \ldots, U$. From (3.3) it follows that an SDF-PDF actor has as many firing rules as it has configurations. Configurations may change from one actor firing to the next which makes SDF-PDF a dynamic dataflow MoC.

We proceed by defining the firing function of an arbitrary SDF-PDF actor defined for its arbitrary output port $q_i \in Q$ as follows

$$\begin{aligned}\tau(q_i)(n) =& d_A(\overline{x}_Q(q_i, L)) \\ & \otimes \bigoplus_{p_i \in P} \tau(p_i)(\sum_{j=1}^{L} r_A(p_i, \overline{x}_Q(q_i, j))).\end{aligned} \tag{3.4}$$

An SDF-PDF actor evolves in firings under different configurations. This means that if configurations differ from one firing to the next one, so will the actor firing rules differ as well as the firing delays. Therefore, in consideration of (3.4) where we compute the production time of token $\sigma_{q_i}(n)$ that equals to $\tau(q_i)(n)$ we must consider the configuration sequence that lead to the production of $\sigma_{q_i}(n)$. This sequence is given as a mapping $\overline{x}_Q : Q \times [1, \ldots, L] \to X_A$ where $L \in \mathbb{N}_{>0}$. The sequencing of configurations is arbitrary and therefore it can be modeled as a nondeterministic choice and the (timed) firing function of (3.4) can take many different values for the same value of n depending on all possible interleavings of configurations that can produce output sequences of length n. Therefore, as in the case of FSM-SADF, it would be mathematically correct to call it a firing relation rather than a function but we deliberately leave the term function for the reason explained while discussing the firing function of an FSM-SADF actor. Given a configuration, SDF-PDF actor acts like an "ordinary" SDF actor, i.e. for an X_A with a single configuration, it is trivial to check that (3.3) and (3.4) reduce to (2.10) and (2.11). This shows that SDF-PDF generalizes SDF.

3.4.2 SDF-PDF graphs

SDF-PDF actors are composed into SDF-PDF graphs (SDF-PDFGs). From the considerations of the previous subsection on SDF-PDF actors and the definition of SDFG (cf. Definition 2.4), with a minimal change in notation, it is rather straightforward to induce a definition for the composition of SDF-PDF actors, i.e. an SDF-PDFG.

Let graph rates be expressed as parametric arithmetic expressions. Let the expression production be governed by an arbitrary grammar $\mathcal{R}$. Similarly, we let actor firing delays be parameterized by an arbitrary grammar $\mathcal{D}$. Then, an SDF-PDFG is to be defined as follows.

Definition 3.1 (SDF-PDFG). *An SDF-PDF graph is defined as a tuple $G = (\mathcal{A}, C, \mathcal{P}_i, \mathcal{P}_d, r, d, i, X_G)$, where $\mathcal{A}$ is the set of actors, $C \subseteq \mathcal{A} \times \mathcal{A}$ the multiset of channels, $\mathcal{P}_i$ is the set of nonnegative integer parameters, $\mathcal{P}_d$ is the set of nonnegative real parameters, $r : \mathcal{A} \times C \to \mathcal{R}$ returns for each port its (possibly symbolic) rate, $d : \mathcal{A} \to \mathcal{D}$ returns for each actor its associated (possibly symbolic) firing delay, $i : C \to \mathbb{N}_0$ returns for each channel its associated number of initial tokens while X_G is the domain of the graph.*

Aside the typical dataflow graph constituents such as actors, channels, rates, firing delays and initial tokens, Definition 3.1 introduces the concept of SDF-PDFG domain that extends the concept of SDF-PDF actor domain discussed earlier. The domain X_G of an SDF-PDFG G is the set of all configurations of G. A configuration of an SDF-PDFG is determined by assigning concrete values to all parameters defined by the sets $\mathcal{P}_i$ and $\mathcal{P}_d$. We denote a configuration of G with $x_i^G \in X_G$ where $i \in \{1, \ldots, |X_G|\}$. Once a configuration is routed through the grammars $\mathcal{R}$ and $\mathcal{D}$ and applied to the SDF-PDFG, an instance of that graph emerges, denoted $\iota_G(x_i^G)$. An instance of an SDF-PDFG is nothing but an SDFG.

Simply put, SDF-PDFGs are SDFGs with rates and actor firing delays parameterized by parametric arithmetic expressions. A configuration is a set of parameter values appearing in those expressions (one value for each parameter). When parameters in these parametric arithmetic expressions are replaced by their values that are specified in the configuration, we obtain concrete expressions. By evaluating these concrete expressions an SDF-PDFG instantiates to an SDFG called its instance.

To ensure that all graph rates are well-defined at all times, we require that all parameters remain constant within an SDF-PDFG iteration, i.e. reconfigurations are only allowed in-between iterations. Across iterations, configurations change arbitrarily through reconfiguration, which can be modeled as a nondeterministic choice. Therefore, SDF-PDF supports nondeterminism. The SDF-PDF model of Definition 3.1 offers a high modeling flexibility as actor port rates and actor firing delays can be expressed as arbitrary expressions of parameters. Furthermore, the concept of domain adopted from [16] allows to explicitly represent the dependencies between parameters by specifying X_G as a set of (nonlinear) constraints. These dependencies may originate from a variety of sources such as arithmetic expressions that express the values of parameters in terms of values of other parameters [81].

3.4.3 Our analysis model

The key technique to execute an SDF-PDFG as any other dataflow graph is scheduling. In our performance analysis to be disclosed soon, we consider self-timed scheduling. Because SDF-PDF is a dynamic dataflow models, it will be dynamically scheduled, i.e. at run-time once all parameter values are known.

However, to give compile-time worst-case performance guarantees for SDF-PDF programs we need to have a degree of knowledge how the program is scheduled. Therefore, we revert to a scheduling concept called quasi-static scheduling, i.e. we consider only SDF-PDF specifications for which a quasi-static schedule is available. Quasi-static scheduling is a middle-ground between static and dynamic scheduling where most of the schedule is known at compile-time while only some scheduling decisions are made at run-time [93].

Furthermore, we focus on consistent/bounded and deadlock-free SDF-PDF specifications because the ones that are not are of little practical importance.

However, for programs modeled in the general SDF-PDF abstraction of Definition 3.1 it is not decidable (at compile-time) whether they are consistent/bounded, deadlock-free and quasi-static schedulable.

To ensure consistency/boundedness, deadlock-freedom and quasi-static schedulability we need to restrict parameterization patterns. In particular, we need to restrict $\mathcal{R}$ so that design-time guarantees regarding consistency, boundedness and deadlock-freedom can be provided and that a quasi-static schedule for the specification can be computed. Furthermore, we need to restrict the parameter change intervals so that at all times all rates in the graph are well-defined. As we are interested in performance metrics of SDF-PDF, this type of functional analysis is outside the scope of this paper. Therefore, we revert to the existing results. To the best of our knowledge, only SDF-PDF of [38] and its relative BPDF of [12] define precise criteria under which consistency/boundedness and deadlock-freedom are decidable at compile-time. The algorithms for deciding on consistency/boundedness through the derivation of the repetition vector and generation of the quasi-static schedule are parametric extensions of SDF algorithms [9]. Therefore, we take the definition of $\mathcal{R}$ from [38] for our analysis model as follows

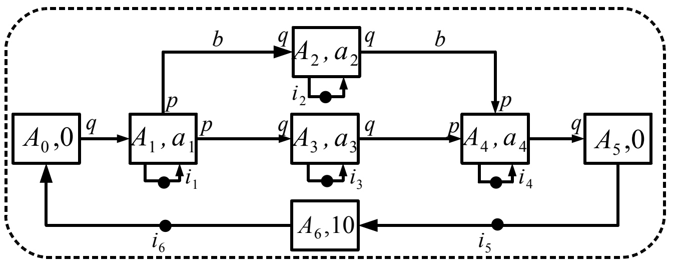
$$\mathcal{R} := k \mid p \mid \mathcal{R}_1 \cdot \mathcal{R}_2. \tag{3.5}$$

In (3.5), $k \in \mathbb{N}_{>0}$ and $p \in \mathcal{P}_i$ with $\mathcal{P}_i$ a set of symbolic variables, i.e. rates are defined as products of positive integers and/or symbolic variables by default constrained to $\mathbb{N}_{>0}$. Here we notice that we exclude 0 from the value set of rate parameters while Definition 3.1 allows it. This is because a rate of 0 in the context of a channel balance equation of an SDF-PDFG has an ambiguous semantics. For a balance equation to be satisfied, if one channel rate is 0 and the other is parametric, either the parametric rate must be 0 too or the repetition vector entry of one of the channel actors must be 0. We go for the former, i.e. if one rate of the channel is 0, we require that the other rate must be 0 too. We include this in the analysis model, using the notion of conditional channels borrowed from BPDF. There, every channel is annotated with a boolean expression. At run-time, whether the boolean expression evaluates to `true` (`tt`) or `false` (`ff`) is the channel enabled or disabled, respectively. Disabled means that no production or consumption will take place at that channel. In particular, let $\mathcal{P}_b$ be a set of boolean parameters. Let

$$\mathcal{B} := \mathtt{tt} \mid \mathtt{ff} \mid b \mid \neg\mathcal{B} \mid \mathcal{B}_1 \wedge \mathcal{B}_2 \mid \mathcal{B}_1 \vee \mathcal{B}_2, \tag{3.6}$$

where $b \in \mathcal{P}_b$ be a grammar defined over $\mathcal{P}_b$. Let $\beta : C \to \mathcal{B}$ return for a channel its condition. As mentioned, if $\beta(c)$ evaluates to `ff`, the channels is disabled, which translates to both its source and destination rate being equal to 0. This way, dynamic change of graph topology is achieved. Conditional channels have no bearing on the repetition vector of the graph, i.e. its entries are obtained by solving the balance equations using the default rates, that is the ones defined by $\mathcal{R}$. Consequently, actors fire the designated number of times regardless of the fact whether they actually produce/consume tokens or not. Therefore, conditional channels must not be confused with the notion of conditional execution of e.g. BDF [22] as they serve as syntactical sugar to account for the possibility of a rate being equal to 0.

Fig. 3.3a shows an example SDF-PDFG. For the example, $\mathcal{P}_i = \{p, q\}$, $\mathcal{P}_d = \{a_1, a_2, a_3, a_4\}$ and $\mathcal{P}_b = \{b\}$. Channels (A_1, A_2) and (A_2, A_4) are made conditional using boolean parameter b via (3.6). Rates are parameterized via expressions generated by (3.5) and the firing delays via $\mathcal{D} := d$ where $d \in \mathcal{P}_d$. The graph's quasi-static schedule is shown in Fig. 3.3b and defines the string $A_0^1 A_1^q A_2^p A_3^p A_4^q A_5^1 A_6^1$. The repetition vector of the graph equals to $\Gamma(A_0, A_1, A_2, A_3, A_4, A_5, A_6) = (1, q, p, p, q, 1, 1)$. The domain of the graph is given as $X_G = \{p \in [1, 20], q \in [1, 30], a_1 = a_2 = a_3 = a_4 = 3\}$.



(a) Example SDF-PDFG.

```
while(1){
   fire A0;
   repeat (q) times {
      fire A1;
   }
   repeat (p) times {
      fire A2;
   }
   repeat (p) times {
      fire A3;
   }
   repeat (q) times {
      fire A4;
   }
   fire A5; fire A6;
}
```

(b) Schedule of the example SDF-PDFG.

Fig. 3.3: SDF-PDF.

3.4.4 SDF-PDF and FSM-SADF

After having elaborated our SDF-PDF analysis model, the crucial point being that parameters are allowed to change in-between graph iterations, we establish a semantic link between our analysis model and FSM-SADF by showing that the concept of configuration/instance corresponds to the concept of scenario in FSM-SADF and that every SDF-PDF can be unfolded to an FSM-SADF that captures its temporal behavior if every configuration/instance is declared a scenario. Consequently, the concept of reconfiguration in SDF-PDF corresponds to the concept of scenario transition in FSM-SADF.

We start the discussion from the perspective of SDF-PDF and FSM-SADF actors. If we compare the firing rules and the firing functions of FSM-SADF actors of (2.26) and (2.27) to that of SDF-PDF actors of (3.3) and (3.4) we observe a striking resemblance. Actually, they are fully correspondent in the sense that the notion of scenario in FSM-SADF corresponds to the notion of configuration in SDF-PDF. This follows from the fact that once the scenario information is accounted for by an FSM-SADF actor, that actor instantiates to a single SDF actor. In the context of SDF-PDF, once the configuration is applied to an SDF-PDF actor, a single SDF actor emerges.

Consider now an SDF-PDF actor and its domain. By applying each and every domain configuration to the actor, we will obtain as many SDF actors as is the cardinality of the actor domain. If we now group all those SDF actors with different type signatures and firing delays and call them scenarios, this group forms an FSM-SADF actor. From the observer's perspective the original SDF-PDF actor and the so constructed FSM-SADF actor will temporally (in terms of their timed firing functions) behave the same.

This informally presented equivalence in temporal behavior of SDF-PDF and FSM-SADF actors straightforwardly applies to corresponding actor compositions, i.e. SDF-PDFGs and FSM-SADFGs.

In particular, SDF-PDF evolves in iterations of its instances, $\iota_G(x_i^G)$. Instances are defined by configurations that are arbitrarily selected (nondeterminism) from one SDF-PDFG iteration to the other. These instances are nothing but SDFGs. On the other hand, FSM-SADFG evolves in iterations of its scenario SDFGs, where scenario occurrence patterns are given by the scenario FSM. Now, given a SDF-PDFG and its domain, for each domain configuration an instance SDFG can be obtained. If we group these instances and call them scenarios, we have obtained an FSM-SADFG. As the instance occurrence pattern is arbitrary, so will the newly constructed FSM-SADFG have a fully connected FSM where each state corresponds to one scenario and vice-versa (scenario labeling is a bijection). From the observer's perspective, in terms of the timestamp vector sequence $\gamma(0), \gamma(1), \gamma(2), \ldots$ of production times of initial tokens after the kth graph iteration, the two graphs behave the same for a properly paired instance/scenario sequence.

E.g., consider the example SDF-PDFG of Fig. 3.3. Assume that the graph has only two configurations, namely $x_1^G = \{b = \texttt{tt}, p = 3, q = 2, a_1 = 5, a_2 = 4, a_3 = 4, a_4 = 4\}$ and $x_2^G = \{b = \texttt{ff}, p = 3, q = 2, a_1 = 5, a_2 = 6, a_3 = 4, a_4 = 4\}$. If we evaluate these configurations we obtain two SDFG instances of the original SDF-PDFG, namely $\iota_G(x_1^G)$ and $\iota_G(x_2^G)$. These in turn correspond to scenario s_1 and s_2 SDFGs of the FSM-SADFG of Fig. 2.4a. Because the same FSM-SADFG has a fully connected FSM, using the correspondence $x_1^G \equiv s_1$ and $x_2^G \equiv s_2$, any configuration/instance[2] sequence of the SDF-PDFG has an "equivalent" FSM-SADF scenario sequence. The difference exists however in the repetition vector entries for actor A_2 in s_2 and $\iota_G(x_2^G)$ due to the convention that the repetition vector of an SDF-PDFG depends not on the values of channel conditions. In particular, with $\iota_G(x_2^G)$, $\Gamma(A_2) = 3$ while for s_2, $\Gamma(A_2) = 1$. But, this is only a matter of convention and not semantics. If we set $\Gamma(A_2) = 3$ in s_2, the completion times of SDF-PDFG and FSM-SADFG iterations expressed by means of $\gamma(k)$ are equal as shown in Fig. 3.4 for the configuration/scenario sequence $\langle x_1^G \equiv s_1, x_2^G \equiv s_2, x_1^G \equiv s_1, x_2^G \equiv s_2, x_1^G \equiv s_1 \rangle$.

Not mentioned so far, but worth pointing out is that both SDF-PDF and FSM-SADF defined precise semantics for the delivery of configurations and scenario information to the implementing actors. These mechanisms

[2]We will be using the terms configuration and instance interchangeably.

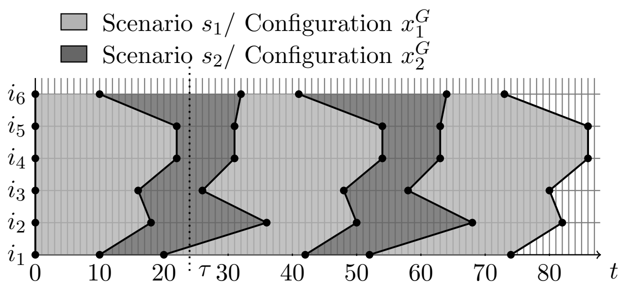


Fig. 3.4: Execution of SDF-PDFG of Fig. 3.3.

include addition of extra channels to the original specifications that carry tokens with values of parameters and scenarios. However, these elements are added in a purely dataflow manner and do not influence the semantics of the models. Therefore, in our presentation they are left out for simplicity and without loss of generality because the temporal analysis needs not to make a difference between data token channels and such parameter delivery channels as they are added in a purely dataflow manner. For more details we refer the interested reader to the concepts of parameter distribution network of SPDF [38], the concept of *initflow* of PSDF [16]. For FSM-SADF we will be discussing scenario information delivery mechanisms in Chapter 6 when we introduce the traditional operational semantics of FSM-SADF.

3.5 Max-plus algebra for SDF-PDF

After having formally defined our SDF-PDF analysis model in the previous section, we proceed by defining the Max-plus algebra-based tools needed to capture its temporal behavior. This is a crucial milestone on the path towards our final goal, i.e. the performance analysis for SDF-PDF.

3.5.1 Max-plus algebraic semantics of SDF-PDF

With regard to the Max-plus algebraic semantics of FSM-SADF of (2.28) and the semantic link of SDF-PDF and FSM-SADF explained in the previous section where reconfiguration in SDF-PDF corresponds to a scenario transition in FSM-SADF, the evolution of an SDF-PDFG G can be given as a recursive Max-plus linear equation relating the timestamps vectors $\gamma(k+1)$ and $\gamma(k)$ of initial tokens after the $(k+1)$st and the kth SDF-PDFG iteration, respectively, as follows

$$\gamma(k+1) = \mathcal{M}_G(\zeta_G(k+1)) \otimes \gamma(k). \tag{3.7}$$

In (3.7), $\mathcal{M}_G : X_G \to \mathbb{R}_{\max}^{|I|\times|I|}$ denotes a mapping that for each $x^G \in X_G$ returns the associated Max-plus matrix of the instance SDFG, i.e. $M_{\iota_G(x^G)}$. Mapping $\zeta_G : \mathbb{N}_{>0} \to X_G$ returns the configuration that determines the instance $\iota_G(x^G)$ that is executed as the $(k{+}1)$st iteration of the SDF-PDFG. Therefore, the temporal behavior of an SDF-PDFG can be fully described by a set of Max-plus instance matrices. The number of such matrices equals to the cardinality of X_G, i.e. $|X_G|$. However, $|X_G|$ is typically very large and proportional to the cardinality of the product set of parameter ranges. This renders the generation of this set via enumeration of X_G often unachievable in a reasonable amount of time or even impossible if the firing delays are defined in $\mathbb{R}_{\geq 0}$.

Instead of enumeration, with the overall goal of compacting the representation while retaining relevant information, we advocate for the characterization of temporal behavior of SDF-PDF models using a set of parameterized Max-plus matrices, i.e. matrices whose entries will be parameterized expressions in $\mathcal{P}_i$ and $\mathcal{P}_d$ ($\mathcal{P}_b$ will be handled separately). In the light of the aforementioned, the evolution of an SDF-PDFG can be described via

$$\gamma(k+1) = \underbrace{\left(\mathcal{M}_G^{\text{par}}(\zeta_G(k+1))\right)(\zeta_G(k+1))}_{\mathcal{M}_G(\zeta_G(k+1))} \otimes \gamma(k). \tag{3.8}$$

In (3.8), $\mathcal{M}_G^{\text{par}} : X_G \to E^{I\times I}$ denotes a mapping that for each $x^G \in X_G$ returns the associated parameterized Max-plus matrix ($(\mathcal{M}_G^{\text{par}}(x^G))(\cdot)$ notation) that when evaluated for that x^G ($(\cdot)(x^G)$ notation) is nothing but the Max-plus matrix of the instance SDFG, i.e. $\mathcal{M}_G(\zeta_G(k+1)) = M_{\iota_G(x^G)}$ when $\zeta_G(k+1) = x^G$. Notation E defines the set of all arithmetic expressions defined on $(\mathcal{P}_i \cup \mathcal{P}_d \cup \mathbb{R}_{\max})$ which in turn is used to define $E^{I\times I}$ the set of all I by I Max-plus matrices with entries in E. By using parameterized matrices, one needs not to perform an enumeration of X_G. The difficulty is moved, however, to determining the mapping $\mathcal{M}_G^{\text{par}}$ defining the collection of parameterized matrices as constituents of its codomain. It is a collection (and not a single parameterized matrix) because in a parametric (general) setting, the partitioning of X_G occurs naturally due to the max operator in Max-plus. In itself, as we will explain later, it is not clear how to derive $\mathcal{M}_G^{\text{par}}$ and therefore, in this thesis, we use $\mathcal{M}_G^{\text{par}}$ as an abstract concept. What we will show how to derive, is the mapping $\mathcal{M}_G^{\text{par}\prime} : X_G \to E^{I\times I}$ that for each $x^G \in X_G$ returns a parameterized matrix that is a conservative approximation of $\mathcal{M}_G^{\text{par}}(x^G)$, or formally

$$\mathcal{M}_G^{\text{par}}(x^G) \preceq \mathcal{M}_G^{\text{par}\prime}(x^G) \tag{3.9}$$

for all $x^G \in X_G$. This is a sound approximation as the relative approximation error goes to 0 with growing entries of the repetition vector of the graph.

We now discuss how to compose this matrix. In particular, as SDF is the base model of SDF-PDF, the timestamp t of any token produced within the $(k+1)$st SDF-PDFG iteration can be written as a Max-plus scalar product

$$t = \bigoplus_{i_j \in I} m_j^{\text{par}} \otimes t_{i_j} = [m_1^{\text{par}}, \ldots, m_{|I|}^{\text{par}}] \otimes \gamma(k), \tag{3.10}$$

where t_{i_j} are the timestamps of initial tokens after the kth graph iteration and m_j^{par} are now parameterized expressions in E. Therefore, the timestamps of initial tokens at the end of the $(k+1)$st iteration embedded in $\gamma(k+1)$ can be computed as follows

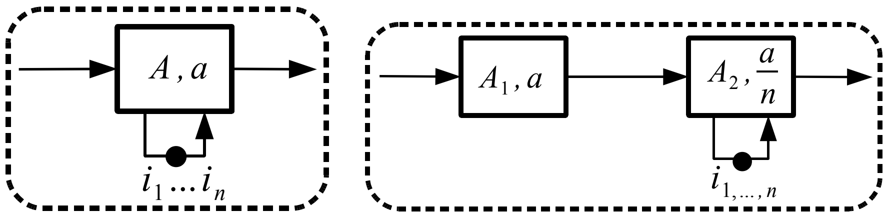
$$t'_{i_l} = \bigoplus_{i_j \in I} m_{l,j}^{\text{par}} \otimes t_{i_j} = [m_{l,1}^{\text{par}}, \ldots, m_{l,|I|}^{\text{par}}] \otimes \gamma(k). \tag{3.11}$$

In this case, dependency vectors $[m_{l,1}^{\text{par}}, \ldots, m_{l,|I|}^{\text{par}}]$ where $l = 1, \ldots, |I|$ will form the rows of a parameterized Max-plus SDF-PDFG matrix that represents an element of the codomain of $\mathcal{M}_G^{\text{par}}$, i.e. $\mathcal{M}_G^{\text{par}\prime}$. Thus, the challenge lies in determining expressions of type (3.11)[3]. In the remainder of this section we show how to do this for a type of graphs that in addition to being consistent, deadlock free and quasi-static schedulable satisfy the following two requirements.

Requirement 3.1. *For all SDF-PDFG channels $c \in C$ such that $src(c) \neq dst(c)$ and $i(c) > 0$, $i(c) > \Gamma(dst(c)) \cdot r(dst(c), c)$ must hold, i.e. if c has initial tokens, there must be enough of them for actor $dst(c)$ to complete all its firings within the iteration. Functions $src : C \to \mathcal{A}$ and $dst : C \to \mathcal{A}$ return for each channel its source and destination actor, respectively.*

With this requirement, we limit our attention to feed-forward structures where initial tokens in graph channels (other than self-edges) are not reproduced more than once within an iteration. This way, in cyclic graphs, within one iteration, feedback loops can be broken resulting in acyclic specifications from the perspective of a single iteration. Across multiple iterations, the cyclicity is effectively restored. Fortunately a large number of streaming applications fall under this requirement that is typically enforced in literature to enable effective quasi-static scheduling [17][38][16]. In the context of our Max-plus analysis we impose this requirement as it is not clear how

[3]Actually, their conservative approximations in the context of (3.9)



(a) Actor with auto-concurrency bounded to n. (b) Latency-rate abstraction of the actor above.

Fig. 3.5: Latency-rate abstraction

to deal with schedule loops of length greater than one [9] with parametric repetition counts.

Requirement 3.2. *For all SDF-PDFG channels $c \in C$ such that $src(c) = dst(c)$, $i(c) = 1$ must hold.*

This requirement disables the bounding of auto-concurrency. Auto-concurrency of actors can be bounded by inserting a particular number of tokens on their self-edges. With Requirement 3.2 we allow either full auto-concurrency for an actor or no auto-concurrency at all. This is because we during the process of determining $\mathcal{M}_G^{\text{par}}$, i.e. $\mathcal{M}_G^{\text{par}}$, with regard to (3.10) wish to avoid situations where tokens produced by the actor depend on different self-edge tokens from one actor firing to the next. This requirement is not restrictive in practice as any such actor in the graph can be replaced by its latency-rate abstraction [119] that conservatively captures its temporal behavior. Fig. 3.5b shows such a conservative latency-rate based abstraction of an actor with auto-concurrency bounded to n displayed in Fig. 3.5a. Note that the collection $i_1, \dots, i_n$ of Fig. 3.5a is collapsed into a single token $i_{1,\dots,n}$ of Fig. 3.5b. Actor A itself is expanded into two actors A_1 and A_2 with firing delays a and $\frac{a}{n}$, respectively. We believe the same principle could be straightforwardly applied to cyclic graph substructures with channels not compliant to Requirement 3.1 using the notion of local iterations [38]. The "problematic" subgraphs would then be replaced by their latency-rate abstractions. The procedure could be recursively repeated in a bottom-up fashion in line with different levels of substructure nesting. This is, however, a subject of future work.

3.5.2 Max-plus model of SDF-PDF execution

3.5.2.1 The basics

To determine $\mathcal{M}_G^{\text{par}}$, i.e. $\mathcal{M}_G^{\text{par}\prime}$, as with SDF [44], we need to compute one iteration of the considered SDF-PDFG, i.e. the production times of restored initial tokens after one iteration of the graph expressed via the scalar product of (3.11). Recall that within an iteration, graph parameters do not change, i.e. they are static and an SDF-PDF actor can be treated as an SDF actor.

With SDF it is straightforward to keep track of timestamps of tokens produced by actor firings on channels within a simple FIFO container. This is due the fact the channel quantities are finite and known. Each FIFO element stores the dependency vector of the token it refers to (cf. (2.14)). With parameterized rates, the situation is more subtle. The channel quantities will still be finite but unknown as they are determined by parameters. Therefore, it would become cumbersome to define such a FIFO structure. Instead, we capture the ordering of tokens using the firing indices of their producing actors as a mapping $\tau : \mathcal{A} \times \mathbb{Z} \to E^{|I| \times 1}$. The nonpositive firing indices are reserved for the initial tokens themselves. As initial tokens represent the initial conditions for the execution of the graph, they are therefore assumed to be produced by some past actor firing. We give the following definition of τ as follows from the Max-plus algebraic semantics of self-timed execution of SDF (cf. (2.11)) that SDF-PDF inherits (cf. (3.4)):

$$\tau(A_j, n) = \bigoplus_{A_i | (A_i, A_j) \in C} \varepsilon((A_i, A_j)) \tag{3.12a}$$

$$\otimes\, \tau\left(A_i, \left\lceil \frac{n \cdot r(A_j, (A_i, A_j)) - i((A_i, A_j))}{r(A_i, (A_i, A_j))} \right\rceil\right) \tag{3.12b}$$

$$\otimes\, d(A_j). \tag{3.12c}$$

Equation (3.12) encodes the Max-plus semantics of self-timed execution of an SDF-PDF actor within an iteration of the superordinate SDF-PDFG. Its graphical interpretation is shown in Fig. 3.6. Synchronization is expressed via (3.12b) where an actor waits for the last required input token needed to perform the nth firing. The timestamps of these tokens are determined using the firing indices of their production actors where the number of initial tokens on the considered channel must be taken into account. After all input tokens are in place, the firing commences and finishes after an amount of time equal to the firing delay of that actor. This is the delay part of Max-plus and is expressed via (3.12c). However, in presence of conditional

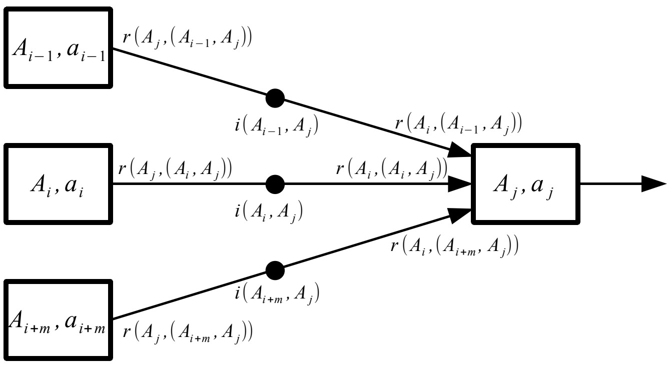


Fig. 3.6: Illustration of (3.12).

channels, some input channels are disabled and therefore do not influence the production times of output tokens. This concept of conditional channels is accounted for in the Max-plus semantics of an SDF-PDF actor by (3.12a), where $\varepsilon : C \to \{0, -\infty\}$ is defined as follows

$$\varepsilon(c) = \begin{cases} 0 & \text{if } \beta(c) = \texttt{tt} \\ -\infty & \text{if } \beta(c) = \texttt{ff} \end{cases} \tag{3.13}$$

for all $c \in C$.

3.5.2.2 Computation of the actor response

We show now how to compute $\tau(\cdot, n)$ for an actor within an SDF-PDFG iteration. Equation (3.12) reveals that in computing the response of an actor, different inputs (channels) can be treated in isolation. Ultimately, particular contributions need to be superimposed. This corresponds to the Max-plus superposition principle [7]. We first show how to apply (3.12) to one input channel and compute the channel's contribution to the actor's response using the concept of Max-plus convolution.

3.5.2.2.1 Max-plus convolution Consider a general SDF-PDF channel structure of Fig. 3.7. Channel (X, Y) is defined with parameterized rates p and q, actors X and Y and their parameterized firing delays x and y, respectively, while the number of initial tokens on the channel equals to $i(X, Y)$. By default, it is enabled i.e. the implied channel condition always evaluates to `tt`. We compute the output of actor Y using (3.12). We only treat the case when in the figure, $i(X, Y) = 0$. The case where $i(X, Y) > 0$ is trivial due to Requirement 3.1. More precisely, within one iteration of the

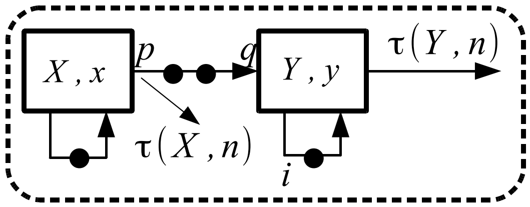


Fig. 3.7: SDF-PDFG channel.

graph, actor's demand for input tokens on that channel will always refer to one of those $i(X,Y)$ initial tokens, i.e. no firings of X within the iteration need to be considered. For actor Y with $i(X,Y)=0$, (3.12) transforms to

$$\tau(Y,n) = \left(\tau(Y,n-1) \oplus \tau(X, \lceil \frac{n \cdot q}{p} \rceil)\right) \otimes y. \tag{3.14}$$

We treat (3.14) using *backward substitution*. Backward substitution is a well-known method for solving recurrence equations and it works exactly as the name implies. In particular, starting from the equation itself, we work backwards substituting the values of the recurrence for previous ones.

If we unfold (3.14) for k times and substitute it back, we obtain

$$\tau(Y,n) = \tau(Y,n-k) \otimes y^{\otimes n} \oplus \bigoplus_{i=1}^{k} \tau(X, \lceil \frac{(n-i+1) \cdot q - \iota}{p} \rceil) \otimes y^{\otimes i}. \tag{3.15}$$

We obtain the base case when $k=n$ from (3.15) as follows

$$\tau(Y,n) = \tau(Y,0) \otimes y^{\otimes n} \oplus \underbrace{\bigoplus_{i=1}^{n} \tau(X, \lceil \frac{(n-i+1) \cdot q - \iota}{p} \rceil) \otimes y^{\otimes i}}_{conv(\tau(X, \lceil \frac{n \cdot q - \iota}{p} \rceil), y^{\otimes n})}. \tag{3.16}$$

In the second term of the Max-plus summation of (3.16) we recognize the Max-plus convolution of the input token timestamp sequence and the impulse response of actor Y, denoted $h(Y,n)$ where $h : \mathbb{N}_{>0} \to \mathbb{R}_{\max}$ is the timestamp sequence belonging to tokens produced by the actor in response to the impulse input token timestamp sequence

$$u(n) = \begin{cases} 0 & \text{if } n = 1 \\ -\infty & \text{otherwise} \end{cases}, \text{for all } n \in \mathbb{N}_{>0}. \tag{3.17}$$

For a complete presentation we refer to [42]. When the actor has a self-edge with one initial token, its impulse response takes the form

$$h(Y,n) = y^{\otimes n} \text{ , for all } n \in \mathbb{N}_{>0}. \tag{3.18}$$

When an actor is without a self-edge,

$$h(Y,n) = \begin{cases} y & \text{if } n = 1 \\ -\infty & \text{otherwise} \end{cases} \text{ , for all } n \in \mathbb{N}_{>0}. \tag{3.19}$$

An actor without a self-edge can be interpreted as an actor with a self-edge with an infinite stock of initial tokens all available at $t = -\infty$. Then, when (3.19) is applied to (3.16), (3.16) reduces to

$$\tau(Y,n) = \tau(X, \lceil \frac{n \cdot q}{p} \rceil) \otimes y. \tag{3.20}$$

We formally define the previously used concept of Max-plus convolution.

Definition 3.2. *Let $\varsigma_1(n)$ and $\varsigma_2(n)$ be two sequences in $\mathbb{R}_{\max}$, i.e. $\varsigma_{1,2} : \mathbb{N}_{>0} \to \mathbb{R}_{\max}$. The convolution of the two, denoted $conv(\sigma_1, \sigma_2)$ is defined as*

$$conv(\varsigma_1, \varsigma_2)(n) = \bigoplus_{i=1}^{n} \varsigma_1(n - i + 1) \otimes \varsigma_2(i). \tag{3.21}$$

In case of an actor with a self-edge, we wish to obtain a closed form solution for (3.16), i.e. a solution without the recursive part defined by the convolution. Note that, for the case of an actor without a self-edge, (3.20) immediately provides such a solution.

Before we proceed we define the concept of an eventually periodic sequence as follows.

Definition 3.3 (Eventually periodic sequence [26]). *We say that the sequence $\varsigma(n)$ is eventually periodic (or shortly periodic) if $\exists K \in \mathbb{N}_{>0}, \exists c \in \mathbb{N}_0, \exists \pi \in \mathbb{R}_{\max}$ such that $\forall n \geq K$:*

$$\varsigma(n + c) = \pi^{\otimes c} \otimes \varsigma(n). \tag{3.22}$$

The term "periodic" has to be understood in the Max-plus-algebraic sense: the terms of the sequence are Max-plus multiplied by a constant factor $c \cdot \pi$. If $\varsigma(n)$ is an eventually periodic sequence then the smallest possible c for which (3.22) holds is called the *period* of $\varsigma(n)$. The smallest possible corresponding π is then called the *ratio* of $\varsigma(n)$.

An SDF/SDF-PDF actor is a Max-plus linear system that can be described with (2.13). Consequently, according to the fundamental theorem in Max-plus algebra [26][29], SDF/SDF-PDF actors are eventually periodic and transitively are SDFGs/SDF-PDFGs as their compositions.

Therefore, the convolution of (3.16) is an eventually periodic sequence of Definition 3.3. In the parametric context, representing the convolution of (3.16) in terms of Definition 3.3 is challenging. In particular, it is intuitively clear from the context that there would not exist a single representation but quite a few depending on the relationships of involved channel rates and actor firing delays. Furthermore, even if we knew a way to derive these representations, it follows from Definition 3.3, that these representations would again be recursive, and recursiveness is a thing we wanted to get rid of in the first place. Therefore, we need a simpler representation. In particular, we take advantage of the periodicity property to simplify the analysis into an entirely (in contrast to eventually) periodic pattern where the period equals to 1, i.e. $c = 1$. "Entirely" means that (3.22) holds for all $n \in \mathbb{N}_{>0}$. The idea is to find a delay δ and a ratio π such that term $\delta \otimes \pi^{\otimes n}$ is a **conservative linear upper bound** to the convolution in (3.16). This (conservative) linear bound is, in terms of Definition 3.3, an entirely periodic sequence with $c = 1$. We show how we can to this for SDF-PDF structures compliant to Requirement 3.1 and to Requirement 3.2.

SDF-PDFG graphs satisfying Requirement 3.1 are acyclic within an iteration because their feedback channels always host enough initial tokens to account for all firings of the consuming actor within an iteration. Therefore, to compute one iteration of the graph, we can consider it acyclic, i.e. compute one iteration of the acyclic graph by starting from its source actors. If the source actor (X in Fig. 3.7) has a self-edge[4], the timestamps of output tokens of that actor can be described as an entirely periodic sequence where the ratio equals to the firing delay of the actor. This is because the actor has no other dependencies than the one given by its self-edge, and the self-edge token is restored every firing delay time-units. Once such a sequence is consumed by another actor (Y in Fig. 3.7) that has a self-edge too, the output of that actor is then defined by (3.16). The problematic part of (3.16) is the convolution of the input sequence (normalized to the input of the considered actor by channel rate fraction, i.e. $\frac{q}{p}$ in Fig. 3.7) and the impulse response of the considered actor.

These sequences can be represented as lines with rational slopes in $\mathbb{N}_{>0} \times \mathbb{R}$ plane (where the x-axis is the index domain and y-axis is the time domain) as shown by Fig. 3.8 for arbitrary sequences ς_1 and ς_2. Their

[4]The case of an actor without a self-edge is trivial.

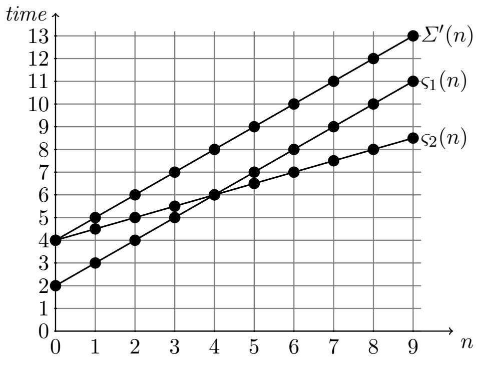


Fig. 3.8: Periodic sequences in $\mathbb{R}_{\max}$.

slopes equal to their ratios and represent the asymptotic growth rates of the sequences. Their counterparts in conventional linear system theory are sine functions, while their slopes are counter-parted with the frequencies of the sine waves [30]. If the input slope is strictly smaller than the slope of the impulse response (smaller slope means smaller ratio), then the output of the actor will be a periodic sequence that inherits the slope/ratio of the impulse response and this is a kind of the "low-pass" effect. If opposite, the slope/ratio of the output will attain the slope/ratio of the input.

To summarize, the ratio of the output of an SDF-PDF actor with one input channel is defined by the larger of the ratios of the input token sequence (normalized by the fraction of the consumption and production rate of the channel defining the dependence as accounted for by the ratio of (3.12b)) and the ratio of the actor's impulse response. In addition, the output sequence will incur a delay that initially accounts for the number of firings that the producing actor needs to perform to enable the first firing of the consuming actor. This number of firings equals to the ceiling of the channel rate fraction (cf. (3.12b)). Finally, as follows from the discussion above, the output token sequence given by the convolution of (3.16) can be conservatively bounded using the following result.

Proposition 3.1. *Let $\varsigma_1(n)$ and $\varsigma_2(n)$ be two entirely periodic sequences in $\mathbb{R}_{\max}$ with period equal to 1, defined by $\varsigma_1(n) = \delta_1 \otimes {\pi_1}^{\otimes\lceil r\cdot n\rceil}$ and $\varsigma_2(n) = {\pi_2}^{\otimes n}$, where $n \in \mathbb{N}_{>0}$, $r \in \mathbb{Q}_{\geq 0}$ and $\delta_1, \pi_1, \pi_2 \in \mathbb{R}_{\max}$. Then, the following inequality holds:*

$$conv(\varsigma_1, \varsigma_2)(n) < \begin{cases} \delta_1 \otimes {\pi_1}^{\otimes(1+r)} \otimes {\pi_2}^{\otimes n} = & \text{if } \pi_2 \geq r \cdot \pi_1 \\ \delta_1 \otimes \pi_2 \otimes {\pi_1}^{\otimes(1+r\cdot n)} & \text{if } \pi_2 \leq r \cdot \pi_1 \end{cases} \tag{3.23}$$

Proof. We prove this by induction using the argument

$$x \leq \lceil x \rceil < x + 1. \tag{3.24}$$

First, we consider the case where $\pi_2 \geq r \cdot \pi_1$. We prove the induction base case, i.e. when $n = 1$. By substituting $\varsigma_1(n) = \delta_1 \otimes {\pi_1}^{\otimes \lceil r \cdot n \rceil}$ and $\varsigma_2(n) = {\pi_2}^{\otimes n}$ into (3.21), we obtain

$$conv(\varsigma_1, \varsigma_2)(n) = \delta_1 \otimes \bigoplus_{i=1}^{n} {\pi_1}^{\otimes \lceil r \cdot (n-i+1) \rceil} \otimes {\pi_2}^{\otimes i}. \tag{3.25}$$

For $n = 1$, (3.25) reduces to

$$conv(\varsigma_1, \varsigma_2)(1) = \delta_1 \otimes {\pi_1}^{\otimes \lceil r \rceil} \otimes \pi_2. \tag{3.26}$$

By combining (3.24) and (3.26) we obtain the following inequality

$$conv(\varsigma_1, \varsigma_2)(1) = \delta_1 \otimes {\pi_1}^{\otimes \lceil r \rceil} \otimes \pi_2 < \delta_1 \otimes {\pi_1}^{\otimes (1+r)} \otimes \pi_2 \tag{3.27}$$

that proves the base case. We continue with the induction step, i.e. evaluate (3.25) for $(n+1)$ with the induction hypothesis of (3.23) where $\pi_2 \geq r \cdot \pi_1$. We obtain

$$\begin{aligned} conv(\varsigma_1, \varsigma_2)(n+1) &= \delta_1 \otimes \bigoplus_{i=1}^{n+1} {\pi_1}^{\otimes \lceil r \cdot (n-i+2)) \rceil} \otimes {\pi_2}^{\otimes i} \\ &= \delta_1 \otimes \bigoplus_{i=1}^{n} {\pi_1}^{\otimes \lceil r \cdot (n-i+1) \rceil} \otimes {\pi_2}^{\otimes i} \\ &\oplus \delta_1 \otimes {\pi_1}^{\otimes \lceil r \rceil} \otimes {\pi_2}^{\otimes (n+1)} \\ &= conv(\varsigma_1, \varsigma_2)(n) \\ &\oplus \delta_1 \otimes {\pi_1}^{\otimes \lceil r \rceil} \otimes {\pi_2}^{\otimes (n+1)}. \end{aligned} \tag{3.28}$$

By substituting the induction hypothesis into (3.28) we obtain the following inequality

$$\begin{aligned} conv(\varsigma_1, \varsigma_2)(n+1) < \delta_1 \otimes {\pi_1}^{\otimes (1+r)} \otimes {\pi_2}^{\otimes n} \\ \oplus \delta_1 \otimes {\pi_1}^{\otimes \lceil r \rceil} \otimes {\pi_2}^{\otimes (n+1)}. \end{aligned} \tag{3.29}$$

If we use (3.24) to get rid of the ceiling in (3.29), we obtain

$$\begin{aligned} conv(\varsigma_1, \varsigma_2)(n+1) &< \delta_1 \otimes {\pi_1}^{\otimes (1+r)} \otimes {\pi_2}^{\otimes n} \\ &\oplus \delta_1 \otimes {\pi_1}^{\otimes (1+r)} \otimes {\pi_2}^{\otimes (n+1)} \\ conv(\varsigma_1, \varsigma_2)(n+1) &< \delta_1 \otimes {\pi_1}^{\otimes (1+r)} \otimes {\pi_2}^{\otimes (n+1)}. \end{aligned} \tag{3.30}$$

Inequality (3.30) shows that the induction hypothesis holds for $(n+1)$ too which completes the proof for the case where $\pi_2 \geq r \cdot \pi_1$.

Now we consider the case where $\pi_2 \leq r \cdot \pi_1$. The base case is simple and equal to that of (3.26). We proceed with the induction step. By substituting the induction hypothesis into (3.28) we obtain the following inequality

$$\begin{aligned} conv(\varsigma_1, \varsigma_2)(n+1) < {} & \delta_1 \otimes \pi_2 \otimes {\pi_1}^{\otimes(1+r\cdot n)} \\ & \oplus \delta_1 \otimes {\pi_1}^{\otimes\lceil r\rceil} \otimes {\pi_2}^{\otimes(n+1)}. \end{aligned} \tag{3.31}$$

If we get rid of the ceiling function and rearrange, we obtain

$$\begin{aligned} conv(\varsigma_1, \varsigma_2)(n+1) < {} & \delta_1 \otimes \pi_2 \otimes ({\pi_1}^{\otimes(1+r\cdot n)} \\ & \oplus {\pi_1}^{\otimes(1+r)} \otimes \underbrace{{\pi_2}^{\otimes n}}_{{\pi_1}^{\otimes(r\cdot n)}}). \end{aligned} \tag{3.32}$$

Because we are considering the case where $\pi_2 \leq r \cdot \pi_1$, we can replace inside the bracket the term π_2 with $r \cdot \pi_1$ (cf. underbrace of (3.32)) so that the (3.32) still holds. We obtain

$$\begin{aligned} conv(\varsigma_1, \varsigma_2)(n+1) < {} & \delta_1 \otimes \pi_2 \otimes ({\pi_1}^{\otimes(1+r\cdot n)} \\ & \oplus {\pi_1}^{\otimes(1+r)} \otimes {\pi_1}^{\otimes(r\cdot n)}) \\ conv(\varsigma_1, \varsigma_2)(n+1) < {} & \delta_1 \otimes \pi_2 \otimes ({\pi_1}^{\otimes(1+r\cdot n)} \\ & \oplus {\pi_1}^{\otimes(1+r\cdot(n+1))}). \end{aligned} \tag{3.33}$$

If follows from (3.33) that

$$conv(\varsigma_1, \varsigma_2)(n+1) < \delta_1 \otimes \pi_2 \otimes ({\pi_1}^{\otimes(1+r\cdot(n+1))}), \tag{3.34}$$

which completes the proof. □

In the right hand side of (3.23) the member 1 in the exponents stems from the fact that $x+1$ is used to bound $\lceil x\rceil$ as $x \leq \lceil x\rceil < x+1$. Member δ_1 accounts for some initial delay of the input sequence $\varsigma_1(n)$, sequence $\varsigma_2(n)$ represents the impulse response of the considered actor, while r stands for the fraction of the channel destination and source rate. The result of (3.23) conservatively bounds $conv(\varsigma_1, \varsigma_2)(n)$. In particular, (3.23) defines a linear upper bound for $conv(\varsigma_1, \varsigma_2)(n)$ as a entirely periodic sequence with $c = 1$, i.e. one that can be expressed by $\delta \otimes \pi^{\otimes n}$. When $\pi_2 \geq r \cdot \pi_1$, $\delta = \delta_1 \otimes {\pi_1}^{\otimes(1+r)}$ and $\pi = \pi_2$, while when $\pi_2 \leq r \cdot \pi_1$, $\delta = \delta_1 \otimes \pi_2 \otimes \pi_1$ and $\pi = r \cdot \pi_1$. The bound of (3.23) is exact in the "ratio part", i.e. it correctly captures the asymptotic growth rate of $conv(\varsigma_1, \varsigma_2)(n)$. It only

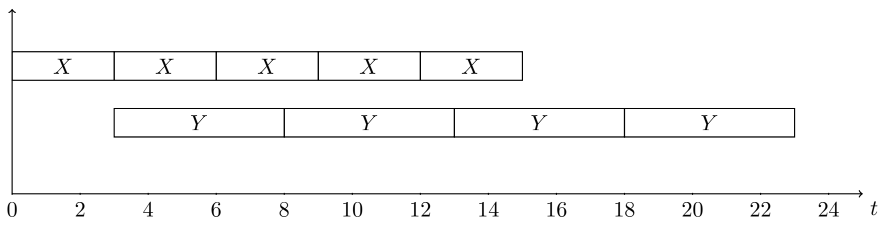


Fig. 3.9: Self-timed execution of the SDF-PDF structure of Fig. 3.7 with $p = 3, q = 2, x = 3, y = 5$ and $i(X, Y) = 0$.

adds a finite amount of delay to conservatively shift the estimate over the actual values of $conv(\varsigma_1, \varsigma_2)(n)$. This shift is derived from the ratios of the involved sequences as specified by (3.23). Therefore, when $n \to \infty$, the relative approximation error of (3.23) will asymptotically reach 0.

We demonstrate the application of Proposition 3.1 to the structure of Fig. 3.7 with, of course, $i(X, Y) = 0$. With regard to Proposition 3.1, in the figure, $\delta_1 = 0, \pi_1 = x, \pi_2 = y, r = \frac{q}{p}$. Fig. 3.9 shows the execution of the structure with: $p = 3, q = 2, x = 3, y = 5$. From the figure we see that actor Y is the bottleneck and its firing delay defines the ratio of the output sequence $\tau(Y, n)$. In particular,

$$\tau(Y, n) = [8, 13, 18, 23, \ldots]. \tag{3.35}$$

On the other hand, with Proposition 3.1, $\pi_2 \geq r \cdot \pi_1$ and

$$\tau(Y, n) \leq 3^{\otimes(1+\frac{2}{3})} \otimes 5^{\otimes n} = [10, 15, 20, 25, \ldots]. \tag{3.36}$$

By comparing (3.35) and (3.36), we see that (3.36) conservatively bounds (3.35) and that for $n \to \infty$ the relative approximation error goes to 0 because both sequences attain the ratio (asymptotic growth rate) of 5 time-units.

Now, consider the execution of the structure when $p = 2, q = 3, x = 3, y = 1$ that is shown in Fig. 3.10. From the figure, it follows that

$$\tau(Y, n) = [7, 10, 16, 19, \ldots]. \tag{3.37}$$

In this case, with Proposition 3.1, $\pi_2 \leq r \cdot \pi_1$, i.e. actor X is the bottleneck and its firing delay normalized to r defines the ratio of the output token sequence (defined by Y's output channels). In particular, according to Proposition 3.1,

$$\tau(Y, n) \leq 1 \otimes 3^{\otimes(1+\frac{3}{2} \cdot n)} = [8.5, 13, 17.5, 22, \ldots]. \tag{3.38}$$

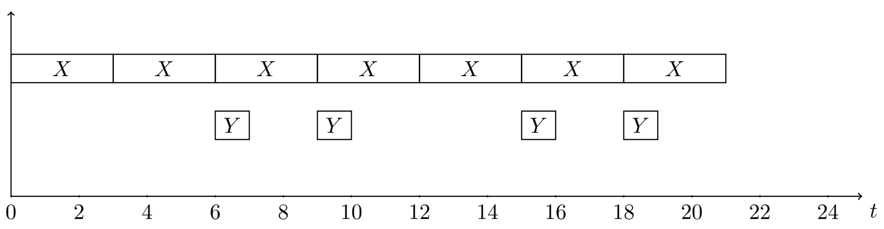


Fig. 3.10: Self-timed execution of the SDF-PDF structure of Fig. 3.7 with $p = 2, q = 3, x = 3, y = 1$ and $i(X, Y) = 0$.

By comparing (3.37) and (3.38), again we see that for (3.38) conservatively bounds (3.37) and that for $n \to \infty$ the relative approximation error goes to 0 because the sequences of (3.37) and (3.38) attain the same ratio (asymptotic growth rate) of 4.5 time-units.

So far we treated only two actors in isolation. We have shown how we can construct a linear upper-bound on their responses using the result of Proposition 3.1. In a graph, these responses will represent inputs to other actors. Therefore, Proposition 3.1 is to be repetitively used to bound the responses of remaining graph actors. From this discussion it follows that the response of a arbitrary actor A_k of an SDF-PDFG compliant to Requirements 3.1 and 3.2 within an iteration can be conservatively bounded by a delay-ratio (δ, π) abstraction as follows

$$\begin{aligned}\tau(A_k, n) &= \bigoplus_{i_j} (\delta_{k,j} \otimes {\pi_{k,j}}^{\otimes n}) \otimes t_{i_j} \\ &= [(\delta_{k,1} \otimes {\pi_{k,1}}^{\otimes n}), \ldots, (\delta_{k,|I|} \otimes {\pi_{k,|I|}}^{\otimes n})] \otimes \gamma(k).\end{aligned} \tag{3.39}$$

A delay-ratio (δ, π) abstraction defines the dependency vector entries as linear functions of n, i.e. the actor firing index. More specifically, given a dependency vector entry that represents the minimal temporal distance of some arbitrary token and an initial token, its ratio will be determined by the scaled (via rate fractions) firing delay of the slowest actor in the path defined by the producing actors of the two tokens.

3.5.2.2.2 Max-plus superposition For an actor with multiple input channels, once we have computed how different inputs contribute to the output of the considered actor, we ultimately need to superimpose the contributions to fully determine the output (sequence). Across corresponding dependency vector entries of different contributions expressed via (3.39), the following propositions defines a conservative linear upper bound for the corresponding output dependency vector entry.

Proposition 3.2. *Let $\Upsilon = \{\varsigma_1(1), \ldots, \varsigma_N(n)\}$ be a set of entirely periodic sequences in $\mathbb{R}_{\max}$ such that $\varsigma_i(n) = \delta_i \otimes {\pi_i}^{\otimes n}$. Let $\delta = \max(\delta_1, \ldots, \delta_N)$ and $\pi = \max(\pi_1, \ldots, \pi_N)$ and let $\varSigma(n) = \bigoplus_{i=1}^{N} \varsigma_i(n)$. Then, $\varSigma(n)$ attains the following conservative linear upper-bound*

$$\varSigma'(n) = \delta \otimes \pi^{\otimes n}. \tag{3.40}$$

Proof. By taking π for the ratio of the bound and 0 for the initial delay, the bound will eventually (for some $n_0 \in \mathbb{N}_{>0}$) compensate for the initial delay difference of particular sequences, i.e. the sequence with the largest ratio will dominate. By taking δ for the delay of the estimate, the estimate will dominate for all $n \in \mathbb{N}_{>0}$. In Fig. 3.8, sequence $\varSigma'(n)$ defines such a bound on $\varsigma_1(n)$ and $\varsigma_2(n)$. If for $\varSigma'(n)$ we set $\pi = \max(\pi_1, \pi_2) = \pi_1 = 1$ and $\delta = 0$, $\varSigma'(n)$ starts to dominate after $n = 4$. However, by setting $\delta = \max(\delta_1, \delta_2) = \delta_2 = 4$, $\varSigma'(n)$ dominates over $\varSigma(n) = \varsigma_1(n) \otimes \varsigma_2(n)$ for all $n \in \mathbb{N}_{>0}$. $\square$

The result of (3.40) defines a entirely periodic sequence with $c = 1$ that attains the ratio of the slowest input sequence and the delay of the most delayed sequence. Sequence $\varSigma(n)$ is piecewise linear. However, in our computations we choose to use a conservative estimate that is entirely periodic. This way we avoid dealing with the cumbersome problem of enumeration of $\varSigma(n)$'s pieces in a parametric context. With such an enumeration, which would split the firing index domain it is not clear how to construct the mapping $\mathcal{M}_G^{\text{par}}$, i.e. $\mathcal{M}_G^{\text{par}\prime}$. However, the bound of (3.40) is exact in the "ratio part" and when $n \to \infty$, the relative approximation error goes to 0.

By using (3.16) and (3.20) in conjunction with Proposition 3.1 and Proposition 3.2 we can compute the response of any graph actor within an iteration expressed in the form of (3.39) that we use to render $\mathcal{M}_G^{\text{par}}$, i.e. $\mathcal{M}_G^{\text{par}\prime}$. We show how to do this in on an example in the subsection to come.

3.5.3 Example

We exemplify the parameterized matrix generation process using the SDF-PDF specification of Fig. 3.3. This is an artificial but an illustrative example that covers all the relevant cases in actor response computation elaborated prior to this subsection. Note that we discuss the application of our techniques to a realistic case study later on in Section 3.8.

Here, for the running example, we will compute the iteration of the graph using its quasi-static schedule that is given as $A_0^1 A_1^q A_2^p A_3^p A_4^q A_5^1 A_6^1$.

The timestamp vector of the kth SDF-PDFG iteration is specified as

$$\gamma(k) = [t_{i_1}, t_{i_2}, t_{i_3}, t_{i_4} t_{i_5}, t_{i_6}]. \tag{3.41}$$

Furthermore, every entry of $\gamma(k)$ can be written in terms of (3.12) (by associating it with the corresponding producing actor) and onwards as the Max-plus scalar product of (3.10).

$$\begin{aligned}
t_{i_1} &= \tau(A_1, 0) = [0, -\infty. -\infty, -\infty, -\infty, -\infty] \otimes \gamma(k),\\
t_{i_2} &= \tau(A_2, 0) = [-\infty, 0, -\infty, -\infty, -\infty, -\infty] \otimes \gamma(k),\\
t_{i_3} &= \tau(A_3, 0) = [-\infty, -\infty, 0, -\infty, -\infty, -\infty] \otimes \gamma(k),\\
t_{i_4} &= \tau(A_4, 0) = [-\infty, -\infty, -\infty, 0, -\infty, -\infty] \otimes \gamma(k),\\
t_{i_5} &= \tau(A_5, 0) = [-\infty, -\infty, -\infty, -\infty, 0, -\infty] \otimes \gamma(k),\\
t_{i_6} &= \tau(A_6, 0) = [-\infty, -\infty, -\infty, -\infty, -\infty, 0] \otimes \gamma(k).
\end{aligned} \tag{3.42}$$

According to the quasi-static schedule of the specification, actor A_0 fires first. It has only one input channel (A_6, A_0), i.e. we need to consider only one contribution. It fires a nonparametric number of times within iteration and does not have a self-edge. Channel (A_6, A_0) forms a directed cycle (feedback loop) and due to Requirement 3.1 it must host a sufficient number of initial tokens to fire A_0 at least $\Gamma(A_0)$ times. This is the case because $\Gamma(A_0) = 1$ and $i((A_6, A_0)) = 2$. Thus, we use (3.12) to compute $\tau(A_0, n)$ for all $n = 1, \ldots, \Gamma(A_0)$ one-by-one. In (3.12), the right hand side will always refer to one of the timestamps of initial tokens. This way the feedback is effectively broken. We evaluate (3.12) for A_0 with $n = 1$ as follows

$$\begin{aligned}
\tau(A_0, 1) &= \tau(A_6, 0) \otimes 0\\
&= [-\infty, -\infty, -\infty, -\infty, -\infty, 0] \otimes \gamma(k).
\end{aligned} \tag{3.43}$$

A more interesting case is that of actor A_1 which fires a parametric number of times within an iteration, i.e. $\Gamma(A_1) = q$ and has a self-edge. Then $\tau(A_1, n)$ cannot be derived in a one-by-one fashion, but we need to express the analytical relation between $\tau(A_1, n)$ and n. We use (3.16) in the context of A_1. The input token sequence to A_1 is defined by (3.43). By substituting (3.43) and t_{i_1} of (3.42) into (3.16) we obtain

$$\begin{aligned}\tau(A_1,n) &= [0,-\infty,-\infty,-\infty,-\infty,-\infty]\\ &\otimes \gamma(k)\otimes {a_1}^{\otimes n}\\ &\oplus [-\infty,-\infty,-\infty,-\infty,-\infty,0]\\ &\otimes \gamma(k)\otimes {a_1}^{\otimes n}\\ &= [{a_1}^{\otimes n},-\infty,-\infty,-\infty,-\infty,{a_1}^{\otimes n}]\\ &\otimes \gamma(k).\end{aligned} \tag{3.44}$$

Even a more intriguing case arises in the consideration of the next actor in the quasi-static schedule, i.e. A_2 with $\Gamma(A_2) = p$. Actor A_2 has a self-edge and one input dependency defined by tokens produced by A_1 on channel (A_1, A_2). However, channel (A_1, A_2) is conditional and whether we consider it in the computation of the response of A_2 depends on the evaluation of expression b. We need to consider both cases, i.e. when $b = \mathtt{tt}$ and $b = \mathtt{ff}$. The consideration splits our exploration into two exclusive parts, one where $b = \mathtt{tt}$ and the other where $b = \mathtt{ff}$. We proceed with

$$C_0 \equiv b = \mathtt{tt}, \tag{3.45}$$

which says that (A_1, A_2) is enabled, i.e. it needs to be accounted for when computing the response of A_2. Thus, using (3.16), (3.42) and (3.44) we derive

$$\begin{aligned}\tau(A_2,n) &= [-\infty,0,-\infty,-\infty,-\infty,-\infty]\\ &\otimes \gamma(k)\otimes {a_2}^{\otimes n}\\ &\oplus [conv({a_1}^{\otimes\lceil\frac{q}{p}\cdot n\rceil},{a_2}^{\otimes n}),-\infty,-\infty,-\infty,-\infty,\\ &conv({a_1}^{\otimes\lceil\frac{q}{p}\cdot n\rceil},{a_2}^{\otimes n})]\otimes\gamma(k).\end{aligned} \tag{3.46}$$

Every entry of the dependency vector of (3.46) expressed as a convolution needs to be treated by Proposition 3.1 in a one-by-one manner. For an entry, Proposition 3.1 gives rise to two cases that split the original parameter space (the graph domain) into two exclusive parts. For each of these parts, the iteration computation continues in a separate branch. In case of actor A_2 only one split will occur as for the two dependency vector entries of (3.46) defined by a convolution, these convolutions are identical. The parts of the parameters space to be considered are defined by inequalities $p \cdot a_2 \geq q \cdot a_1$ and $p \cdot a_2 \leq q \cdot a_1$. We continue the computation in the part of the space defined with

$$C_1 \equiv p \cdot a_2 \geq q \cdot a_1. \tag{3.47}$$

With (3.47), using (3.23), (3.46) transforms to

$$\begin{aligned}\tau(A_2, n) \preceq [&{a_1}^{\otimes(\frac{q}{p}+1)} \otimes {a_2}^{\otimes n}, {a_2}^{\otimes n}, -\infty, -\infty, -\infty,\\ &{a_1}^{\otimes(\frac{q}{p}+1)} \otimes {a_2}^{\otimes n}] \otimes \gamma(k).\end{aligned} \tag{3.48}$$

The right-hand side of (3.48) computed using Proposition 3.1 is a conservative estimate of the actual $\tau(A_2, n)$ and therefore the $\preceq$ notation. A similar case to that of actor A_2 is the one of actor A_3 with $\Gamma(A_3) = p$. From (3.16) using (3.42) and (3.44) we derive

$$\begin{aligned}\tau(A_3, n) = &[-\infty, -\infty, 0, -\infty, -\infty, -\infty]\\ &\otimes \gamma(k) \otimes {a_3}^{\otimes n}\\ &\oplus [conv({a_1}^{\otimes\lceil\frac{q}{p}\cdot n\rceil}, {a_3}^{\otimes n}), -\infty, -\infty, -\infty, -\infty,\\ &conv({a_1}^{\otimes\lceil\frac{q}{p}\cdot n\rceil}, {a_3}^{\otimes n})] \otimes \gamma(k).\end{aligned} \tag{3.49}$$

Equation (3.49) mandates a further split of the parameter space with $p \cdot a_3 \geq q \cdot a_1$ and $p \cdot a_3 \leq q \cdot a_1$. We proceed with

$$C_2 \equiv p \cdot a_3 \leq q \cdot a_1. \tag{3.50}$$

With (3.50),using (3.23), (3.49) reduces to

$$\begin{aligned}\tau(A_3, n) \preceq [&a_1 \otimes a_3 \otimes {a_1}^{\otimes(\frac{q}{p}\cdot n)}, -\infty, {a_3}^{\otimes n}, -\infty, -\infty,\\ &a_1 \otimes a_3 \otimes {a_1}^{\otimes(\frac{q}{p}\cdot n)}] \otimes \gamma(k).\end{aligned} \tag{3.51}$$

We proceed with actor A_4 with $\Gamma(A_4) = q$. This actor has two input channels, and therefore to compute its outputs we apply the Max-plus superposition principle. First we consider the contribution of channel (A_2, A_4) given by (3.48). This is a conditional channel annotated with Boolean expression b. Therefore we need to consider two cases: one where $b = \texttt{tt}$ and the other where $b = \texttt{ff}$. The latter case conflicts with (3.45) and therefore is not satisfiable. This branch of exploration is immediately dropped. We proceed with the former, i.e. when $b = \texttt{tt}$. In that case channel (A_2, A_4) is enabled and contributes to the output of A_4 as follows (we substitute (3.48) and t_{i_4} of (3.42) into (3.16))

$$\begin{aligned}\tau^{(A_2,A_4)}(A_4,n) \preceq\ & [-\infty,-\infty,-\infty,0,-\infty,-\infty] \\ & \otimes \gamma(k) \otimes {a_4}^{\otimes n} \\ & \oplus [{a_1}^{\otimes(1+\frac{q}{p})} \otimes conv({a_2}^{\otimes\lceil\frac{p}{q}\cdot n\rceil}, {a_4}^{\otimes n}), \\ & conv({a_2}^{\otimes\lceil\frac{p}{q}\cdot n\rceil}, {a_4}^{\otimes n}), -\infty, -\infty, -\infty, \\ & {a_1}^{\otimes(1+\frac{q}{p})} \otimes conv({a_2}^{\otimes\lceil\frac{p}{q}\cdot n\rceil}, {a_4}^{\otimes n})] \otimes \gamma(k).\end{aligned} \tag{3.52}$$

The convolutions of (3.52) split the parameter space with $q \cdot a_4 \geq p \cdot a_2$ and $q \cdot a_4 \leq p \cdot a_2$. We arbitrarily choose to proceed with

$$C_3 \equiv q \cdot a_4 \leq p \cdot a_2. \tag{3.53}$$

With (3.53), via (3.23), (3.52) becomes

$$\begin{aligned}\tau^{(A_2,A_4)}(A_4,n) \preceq\ & [{a_1}^{\otimes(1+\frac{q}{p})} \otimes a_2 \otimes a_4 \otimes {a_2}^{\otimes(\frac{p}{q}\cdot n)}, \\ & a_2 \otimes a_4 \otimes {a_2}^{\otimes(\frac{p}{q}\cdot n)}, \\ & -\infty, {a_4}^{\otimes n}, -\infty, \\ & {a_1}^{\otimes(1+\frac{q}{p})} \otimes a_2 \otimes a_4 \otimes {a_2}^{\otimes(\frac{p}{q}\cdot n)}] \otimes \gamma(k).\end{aligned} \tag{3.54}$$

Similarly, we calculate the contribution of channel (A_3, A_4).

$$\begin{aligned}\tau^{(A_3,A_4)}(A_4,n) \preceq\ & [-\infty,-\infty,-\infty,0,-\infty,-\infty] \\ & \otimes \gamma(k) \otimes {a_4}^{\otimes n} \\ & \oplus [conv(a_1 \otimes a_3 \otimes {a_1}^{\otimes(\frac{q}{p}\cdot\lceil\frac{p}{q}\cdot n\rceil)}, {a_4}^{\otimes n}), \\ & -\infty, conv({a_3}^{\otimes(\lceil\frac{p}{q}\cdot n\rceil)}, {a_4}^{\otimes n}), \\ & -\infty, -\infty, \\ & conv(a_1 \otimes a_3 \otimes {a_1}^{\otimes(\frac{q}{p}\cdot\lceil\frac{p}{q}\cdot n\rceil)}, {a_4}^{\otimes n})] \otimes \gamma(k)\end{aligned} \tag{3.55}$$

In (3.55), two convolutions define the further split in the parameter space. The first one (first and last entries of the dependency vector) proposes the following split: $a_4 \geq a_1$ and $a_4 \leq a_1$. We proceed with the option

$$C_4 \equiv a_4 \geq a_1. \tag{3.56}$$

With (3.56), (3.55) becomes

$$\begin{aligned}\tau^{(A_3,A_4)}(A_4,n) \preceq\ & [{a_1}^{\otimes(1+\frac{q}{p})} \otimes a_3 \otimes a_4 \otimes {a_1}^{\otimes n}, -\infty, \\ & conv({a_3}^{\otimes(\lceil\frac{p}{q}\cdot n\rceil)}, {a_4}^{\otimes n}), {a_4}^{\otimes n}, \\ & -\infty, {a_1}^{\otimes(1+\frac{q}{p})} \otimes a_3 \otimes a_4 \otimes {a_1}^{\otimes n}] \otimes \gamma(k).\end{aligned} \tag{3.57}$$

The remaining convolution of (3.57) defines a further split along the current branch of exploration via options $q \cdot a_4 \geq p \cdot a_3$ and $q \cdot a_4 \leq p \cdot a_3$. By selecting

$$C_5 \equiv q \cdot a_4 \leq p \cdot a_3, \tag{3.58}$$

from (3.57) we derive

$$\begin{aligned}\tau^{(A_3,A_4)}(A_4,n) \preceq [&{a_1}^{\otimes(1+\frac{q}{p})} \otimes a_3 \otimes a_4 \otimes {a_1}^{\otimes n}, -\infty,\\ &a_3 \otimes a_4 \otimes {a_3}^{\otimes(\frac{p}{q}\cdot n)}, {a_4}^{\otimes n}, -\infty,\\ &{a_1}^{\otimes(1+\frac{q}{p})} \otimes a_3 \otimes a_4 \otimes {a_1}^{\otimes n}] \otimes \gamma(k).\end{aligned} \tag{3.59}$$

To finalize the computation of response of A_4 now we need to superimpose the contributions of A_2 and A_4. Thus

$$\tau(A_4,n) = \tau^{(A_2,A_4)}(A_4,n) \oplus \tau^{(A_3,A_4)}(A_4,n). \tag{3.60}$$

The computation of (3.60) involves an iterative approach where each entry of the dependency vector of the first contribution is paired with the corresponding entry of the second contribution and the combination is treated by Proposition 3.2. By considering all combinations of maximal delays and periods between corresponding dependency vector entries, Proposition 3.2 further splits the parameter space. For the concrete examples of (3.54) and (3.59) only the first and the sixth entry of dependency vectors are to be treated as the others are either equal (the fourth and the fifth entry) or one immediately dominates because the other equals to $-\infty$ (second and third entry). For the first and the sixth entry (note these are equal in the respective dependency vectors of (3.54) and (3.59)) in the light of Proposition 3.2, we define the corresponding delay-ratio abstractions as follows: $\delta_1 = {a_1}^{\otimes(1+\frac{q}{p})} \otimes a_2 \otimes a_4$, $\pi_1 = {a_2}^{\otimes(\frac{p}{q})}$, $\delta_2 = {a_1}^{\otimes(1+\frac{q}{p})} \otimes a_3 \otimes a_4$ and $\pi_2 = a_1$. Now, Proposition 3.2 mandates four splits in the parameter space defined by the following sets of inequalities: $\{\delta_1 \geq \delta_2, \pi_1 \geq \pi_2\}$, $\{\delta_1 \geq \delta_2, \pi_1 \leq \pi_2\}$, $\{\delta_1 \leq \delta_2, \pi_1 \geq \pi_2\}$ and $\{\delta_1 \leq \delta_2, \pi_1 \leq \pi_2\}$. However, in the current part of the parameter space/domain, constrained by (3.45), (3.47), (3.50), (3.53), (3.56) and (3.58) it is straightforward to see that only the constraint $\{\delta_1 \geq \delta_2, \pi_1 \geq \pi_2\}$ does not conflict with the previously made assumptions and is, furthermore, redundant. Therefore, the computation only continues for $\delta_1 \geq \delta_2, \pi_1 \geq \pi_2$, with regard to which, (3.60) transforms to

$$\begin{aligned}\tau(A_4,n) \preceq [&{a_1}^{\otimes(1+\frac{q}{p})} \otimes a_2 \otimes a_4 \otimes {a_2}^{\otimes(\frac{p}{q}\cdot n)},\\ &a_2 \otimes a_4 \otimes {a_2}^{\otimes(\frac{p}{q}\cdot n)}, a_3 \otimes a_4 \otimes {a_3}^{\otimes(\frac{p}{q}\cdot n)}, {a_4}^{\otimes n},\\ &-\infty, {a_1}^{\otimes(1+\frac{q}{p})} \otimes a_2 \otimes a_4 \otimes {a_2}^{\otimes(\frac{p}{q}\cdot n)}] \otimes \gamma(k).\end{aligned} \tag{3.61}$$

Now according to the quasi-static schedule we consider A_5, with $\Gamma(A_5) = 1$. As for all actors with a constant repetition count, we evaluate (3.12) for all n up to the repetition vector entry for the actor. We obtain

$$\begin{aligned}\tau(A_5, 1) = \tau(A_4, q) \preceq [&{a_1}^{\otimes(1+\frac{q}{p})} \otimes {a_2}^{\otimes(1+p)} \otimes a_4, \\ &{a_2}^{\otimes(1+p)} \otimes a_4, {a_3}^{\otimes(1+p)} \otimes a_4, \\ &{a_4}^{\otimes q}, -\infty, {a_1}^{\otimes(1+\frac{q}{p})} \otimes {a_2}^{\otimes(1+p)} \otimes a_4] \otimes \gamma(k).\end{aligned} \tag{3.62}$$

Actor A_6 fires only once and produces the tokens carrying the timestamp

$$\begin{aligned}\tau(A_6, 1) &= \tau(A_5, 0) \otimes 10 \\ &= [-\infty, -\infty, -\infty, -\infty, 10, -\infty] \otimes \gamma(k).\end{aligned} \tag{3.63}$$

The computation of (3.63) completes the iteration. What remains to be done is to determine the entries of $\gamma(k+1) = [t'_{i_1}, t'_{i_2} t'_{i_3}, t'_{i_4}, t'_{i_5}, t'_{i_6}]$ from computed actor responses and compose the corresponding dependency vectors row-by-row into a matrix. These entries are determined from evaluations of responses of initial token producing actors at the iteration boundary, i.e. for values of n that equal to their repetition vector entries because for those values of n initial tokens are restored. For the example graph, with

$$\begin{array}{lll} t'_{i_1} = \tau(A_1, q) & t'_{i_2} = \tau(A_2, p) & t'_{i_3} = \tau(A_3, p) \\ t'_{i_4} = \tau(A_4, q) & t'_{i_5} = \tau(A_5, 1) & t'_{i_6} = \tau(A_6, 1), \end{array} \tag{3.64}$$

we obtain the parameterized matrix of (3.65)

$$\mathcal{M}_G^{\mathrm{par}}(x^G) \preceq \mathcal{M}_G^{\mathrm{par}\prime}(x^G) = \left[\begin{array}{cc} {a_1}^{\otimes q} & -\infty \\ {a_1}^{\otimes(1+\frac{q}{p})} \otimes {a_2}^{\otimes p} & {a_2}^{\otimes p} \\ {a_1}^{\otimes(1+q)} \otimes a_3 & -\infty \\ {a_1}^{\otimes(1+\frac{q}{p})} \otimes {a_2}^{\otimes(1+p)} \otimes a_4 & {a_2}^{\otimes(1+p)} \otimes a_4 \\ {a_1}^{\otimes(1+\frac{q}{p})} \otimes {a_2}^{\otimes(1+p)} \otimes a_4 & {a_2}^{\otimes(1+p)} \otimes a_4 \\ -\infty & -\infty \end{array}\right.$$

$$\left.\begin{array}{cccc} -\infty & -\infty & -\infty & {a_1}^{\otimes q} \\ -\infty & -\infty & -\infty & {a_1}^{\otimes(1+\frac{q}{p})} \otimes {a_2}^{\otimes p} \\ {a_3}^{\otimes p} & -\infty & -\infty & {a_1}^{\otimes(1+q)} \otimes a_3 \\ {a_3}^{\otimes(1+p)} \otimes a_4 & {a_4}^{\otimes q} & -\infty & {a_1}^{\otimes(1+\frac{q}{p})} \otimes {a_2}^{\otimes(1+p)} \otimes a_4 \\ {a_3}^{\otimes(1+p)} \otimes a_4 & {a_4}^{\otimes q} & -\infty & {a_1}^{\otimes(1+\frac{q}{p})} \otimes {a_2}^{\otimes(1+p)} \otimes a_4 \\ -\infty & -\infty & 0 & -\infty \end{array}\right] \tag{3.65}$$

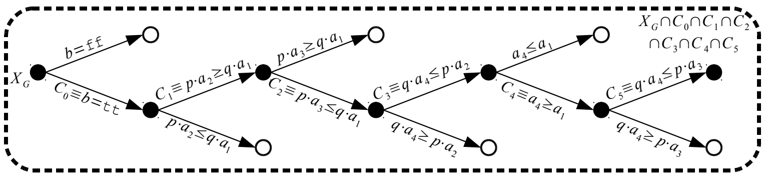

X_G
$b=\mathtt{ff}$
$C_0\equiv b=\mathtt{tt}$
$C_1\equiv p\cdot a_2\geq q\cdot a_1$
$p\cdot a_2\leq q\cdot a_1$
$p\cdot a_3\geq q\cdot a_1$
$C_2\equiv p\cdot a_3\leq q\cdot a_1$
$C_3\equiv q\cdot a_4\leq p\cdot a_2$
$q\cdot a_4\geq p\cdot a_2$
$a_4\leq a_1$
$C_4\equiv a_4\geq a_1$
$C_5\equiv q\cdot a_4\leq p\cdot a_3$
$q\cdot a_4\geq p\cdot a_3$
$X_G\cap C_0\cap C_1\cap C_2$
$\cap C_3\cap C_4\cap C_5$

Fig. 3.11: Exploration tree.

where $x^G \in X_G \cap (b = \mathtt{tt}) \cap (p\cdot a_2 \geq q\cdot a_1) \cap (p\cdot a_3 \leq q\cdot a_1) \cap (q\cdot a_4 \leq p\cdot a_2) \cap (a_4 \geq a_1) \cap (q\cdot a_4 \leq p\cdot a_3)$, i.e. x^G belongs to the part of the original graph domain refined by the set of constraints of (3.45), (3.47), (3.50), (3.53), (3.56) and (3.58). Furthermore, there exists a finite number of such partitions $X_G = \bigcup_{i=1}^{n} X_{Gi}$ that we call natural SDF-PDFG subdomains. Each subdomain $X_{Gi} \subseteq X_G$ defines one parameterized matrix. Collected, matrices form the codomain of $\mathcal{M}_G^{\mathrm{par}\prime}$. Recall that, matrix $\mathcal{M}_G^{\mathrm{par}\prime}(x^G)$ defines a conservative approximation of the theoretical concept of $\mathcal{M}_G^{\mathrm{par}}(x^G)$ first presented in (3.8). The conservativity is due to Propositions 3.1 and 3.2 used in deriving of $\mathcal{M}_G^{\mathrm{par}\prime}(x^G)$.

Fig. 3.11 illustrates the partitioning of the parameter space (domain) X_G for the running example. The matrix of (3.65) is defined by the path determined by the black nodes of the exploration tree.

At this point it is opportune to discuss the semantics of an entry of the parameterized SDF-PDFG matrix that is same as the semantics of the entry of a Max-plus SDFG matrix. In particular, $[\mathcal{M}_G^{\mathrm{par}}(x^G)]_{m,n}$ represents the minimal time distance between token i_m of the $(k+1)$st SDF-PDFG iteration and token i_n of the kth SDF-PDFG iteration. The parametric representation of the matrix entries gives clear insight into the structure of the graph and temporal relationships of actors in the graph. Basically, $[\mathcal{M}_G^{\mathrm{par}}(x^G)]_{m,n}$ defines the delay of the slowest path in the graph connecting the producing actors of initial tokens i_m and i_n. This path is determined by the delay that all actors along the path contribute to and by the ratio of the slowest actor in the path. E.g. if we consider $[\mathcal{M}_G^{\mathrm{par}\prime}(x^G)]_{4,1}$ as obtained from (3.61), we see that actor A_2 is the bottleneck of the path from i_1 to i_4, i.e. it has the highest ratio (or in the light of conventional linear system theory it has the lowest cutoff frequency). On the other hand, such relationships cannot be studied from a concrete Max-plus matrix.

3.5.4 Algorithm for computation of an SDF-PDFG iteration

Algorithm 3.1 specifies the previously described procedure for the computation of one iteration of an SDF-PDFG.

It is defined by the recursive function `ExploreGraph` that explores the tree-like structures like that of Fig. 3.11 in a depth first-search manner. The inputs to the function are `G` the SDF-PDFG itself with all associated meta-data like the quasi-static schedule of the structure, `T` the set of dependency vectors of all graph channels, `curr_actor` the structure containing all required meta-data for the actor being currently evaluated, `curr_actor_ndx` the index of the currently processed actor in the quasi-static schedule, `curr_in_chan_ndx` the index of the currently processed input of the currently processed actor, `curr_init_tok_dep_ndx` the index of the currently processed entry of the dependency vector either within a Max-plus convolution or Max-plus superposition context, `curr_init_tok_dep_delay_ndx` the index of the currently set maximal delay among all the delays observed for the currently processed dependency vector entry, `curr_init_tok_dep_ratio_ndx` the index of the currently set maximal ratio among all the ratios observed for the currently processed dependency vector entry, two-valued variable `in_contr_comput_completed` is a flag denoting whether or not all input channel contributions have been considered for the currently processed actor, `constraints` the set of constraints defining the parameter space partition of the current exploration path and `res` (passed by reference) the result set containing parameterized matrices governing the behavior of the SDF-PDFG in a partition of the initial domain defined by all the constraints encountered along the exploration path.

In the function, actors are processed as ordered in the quasi-static schedule (cf. Line 6). Once the last actor had been processed, the dependency vectors are composed into the related parameterized matrix and along with the constraints encountered added to the result set (cf. Line 49). In the processing of a particular actor the contributions stemming from all its input channels are processed one by one (cf. Line 9). If the channel is conditional, i.e. annotated by a boolean expression (cf. Line 10) we need to continue the search recursively in two exclusive parts of the domain. One is defined by the case when the considered boolean expression evaluates to `tt` (cf. Line 12) and the other when the expression evaluates to `ff` (cf. Line 13). Note that in the next invocation of the function, this conditional channel is no longer conditional, i.e. the boolean condition has been replaced by a concrete boolean value, i.e. `tt` and/or `ff`. Of course, with regard to previous assumptions made on the values of boolean expressions, the current assumption may conflict with those. Therefore a satisfiability check is re-

ALGORITHM 3.1: Compute iteration of an SDF-PDFG.

```
1  Function ExploreGraph(G, T, curr_actor, curr_actor_ndx, curr_in_chan_ndx, curr_init_tok_dep_ndx,
     curr_init_tok_dep_delay_ndx, curr_init_tok_dep_ratio_ndx, in_contr_comput_completed, constraints, ref res)
2      if ((Feasible(constraints) == false) or (Satisfiable(constraints) == false)) then /* check
         feasibility and satisfiability of constraints encountered so far                        */
3          return;
4      end if
5      if curr_actor == null then /* pick the next actor from the Qss we have finished with the previous
         one                                                                                     */
6          curr_actor = G.Qss[curr_actor_ndx];
7      end if
8      if (curr_actor) then /* process actor, by first considering input contributions           */
9          if (curr_input_chan = curr_actor[curr_in_chan_ndx]) then
10             if (IsConditional(curr_input_chan, constraints)) then
11                 expression = GetExpression(curr_input_chan);
                   /* Consider both options, i.e.  expression = tt and expression = ff           */
12                 ExploreGraph(G, T, curr_actor, curr_actor_ndx, curr_in_chan_ndx, 0, 0, 0, false,
                     constraints+(expression = tt), res);
13                 ExploreGraph(G, T, curr_actor, curr_actor_ndx, curr_in_chan_ndx, 0, 0, 0, false,
                     constraints+(expression = ff), res);
14             else
15                 if  ( IsEnabled(curr_input_chan) and curr_input_chan[curr_init_tok_dep_ndx] ) then
16                     options = compute_output(T[curr_in_chan_ndx][curr_init_tok_dep_ndx],
                         curr_actor.impulse_response);
17                     i = 0;
18                     while (i < options.num_options) do
19                         curr_actor.contributions[curr_in_chan_ndx][curr_init_tok_dep_ndx] =
                             options[i].solution;
20                         ExploreGraph(G, T, curr_actor, curr_actor_ndx, curr_in_chan_ndx,
                             curr_init_tok_dep_ndx + 1, 0, 0, false, constraints +
                             options[i].constraint, res);
21                         i++;
22                     end while
23                 else
                       /* completed one input contribution, go to the next                       */
24                     ExploreGraph(G, T, curr_actor, curr_actor_ndx, curr_in_chan_ndx + 1, 0, 0, 0,
                         false, constraints, res);
25                 end if
26             end if
27         else
               /* completed all contributions, proceed with Max-Plus superposition of contributions by
                  combining all delay and ratio relationships over all initial token dependencies    */
28             if (in_contr_completed == false) then
29                 curr_init_tok_dep_ndx = 0;
30                 curr_actor.dp = sort_delays_and_ratios(curr_actor.contributions);
31             end if
32             if (curr_actor.dp[curr_init_tok_dep_ndx]) then
33                 if (curr_delay =
                     curr_actor.dp[curr_init_tok_dep_ndx].delays[curr_init_tok_dep_delay_ndx]) then
34                     if (curr_ratio =
                         curr_actor.dp[curr_init_tok_dep_ndx].ratios[curr_init_tok_dep_ratio_ndx])
                         then
35                         T[curr_actor_output_chan_ndx_all][curr_init_tok_dep_ndx].delay =
                             curr_delay.value;
36                         T[curr_actor_output_chan_ndx_all][curr_init_tok_dep_ndx].ratio =
                             curr_ratio.value;
                           /* next ratio for current delay                                       */
37                         ExploreGraph(G, T, curr_actor, curr_actor_ndx, 0, curr_init_tok_dep_ndx,
                             curr_init_tok_dep_delay_ndx, curr_init_tok_dep_ratio_ndx + 1, true,
                             constraints + options[i].constraint);
38                     else
                           /* next delay                                                         */
39                         ExploreGraph(G, T, curr_actor_ndx, 0, 0, curr_init_tok_dep_ndx, 0,
                             curr_init_tok_dep_delay_ndx + 1, true, constraints, res);
40                     end if
41                 else
                       /* next initial token dependency                                          */
42                     ExploreGraph(G, T, curr_actor, curr_actor_ndx, 0, curr_init_tok_dep_ndx + 1, 0,
                         0, true, constraints, res);
43                 end if
44             else
                   /* done with this actor, do the next one                                      */
45                 ExploreGraph(G, T, null, curr_actor_ndx + 1, 0, 0, 0, 0, false, constraints, res);
46             end if
47         end if
48     else
           /* no more actors in the qss, this is a leaf node - build the matrix associated with a set of
              constraints                                                                        */
49         res += process(G,T,constraints);
50     end if
51     return;
52 end
```

quired at the very beginning of the function (cf. Line 2). If it fails, this branch of exploration is left altogether.

If it does not fail and the channel is deemed enabled (cf. Line 15), we continue by considering particular dependency vector entries of that channel. This may incur as many Max-plus convolutions as there are entries in the input dependency vector. As a convolution (cf. Line 16) incurs splitting of the parameter space, the search is recursively continued for each splitting option (cf. Line 20) while proceeding with the next dependency vector entry (note the increment of `curr_init_tok_dep_ndx` in the recursive call). Note that there are maximally two options per dependency vector entry. Of course, newly added constraints should not conflict with the previous ones, i.e. their feasibility needs to be verified (cf. Line 2). If the combination is not feasible this branch of exploration is left all together (cf. Line 3). Once all entries of the dependency vector of one input channel have been processed, the algorithm continues with the next input channel (cf. Line 24).

Once the contributions of all inputs have been computed, the algorithm performs the bounding of delay-ratio pairs over all corresponding entries of dependency vectors of different contributions according to Proposition 3.2. This is marked by resetting `curr_init_tok_dep_ndx` (cf. Line 29) if the flag `in_contr_comput_completed` is set to `false` (cf. Line 28) to denote that the actor had not yet been treated by Proposition 3.2. Note that in later recursive calls, the flag `in_contr_comput_completed` will be set to `true` (cf. Lines 37, 39 and 42). All combinations are considered, i.e. for each dependency vector entry (cf. Line 32) a different pivot delay (cf. Line 35) and a pivot ratio (cf. Line (36)) are set and the search is continued for the next ratio that is to be deemed maximal (cf. Line 37). Once all periods have been exhausted (cf. Line 34), we proceed with the next pivot delay (cf. Line 39). Once all delays have been exhausted, we proceed with the next dependency vector entry (cf. Line 42). Note that after every recursive call the feasibility is verified with the newly added constraint. The search is aborted in the current branch if the feasibility check fails (cf. Line 3). Once ending with the current actor, we proceed to the next one (cf. Line 45) until the quasi-static schedule has been entirely processed and the matrix added to the solution set (cf. Line 49).

The algorithm has exponential time complexity in the worst-case. However, practical application may be expected to have only a few critical parameters while the graph structures can expose sparsity in the sense that there will be no dependencies between many initial tokens in the graph. Furthermore, the definitions of the domains may be such that many exploration paths will be pruned out due to infeasibility. All this will make the computational effort of Algorithm 3.1 reasonable and render it applicable in many cases in practice.

The set of matrices obtained via Algorithm 3.1 when evaluated at a corresponding $x^G \in X_G$ are actually conservative estimates of the corresponding Max-plus instance matrices. This is due to the conservativity entailed by Propositions 3.1 and 3.2. Therefore, the following inequality holds

$$\left(\mathcal{M}_G^{\mathrm{par}\prime}(x^G)\right)(x^G) \succeq \mathcal{M}_G(x^G) = M_{\iota_G(x^G)} \tag{3.66}$$

for all $x^G \in X_G$. The approximation of (3.66) per matrix entry only incurs an added element of delay while the ratio of the sequence used to obtain the actual matrix value is exact and captures the slowest actor in the path between two tokens (cf. Propositions 3.1 and 3.2). We show this by example. E.g., we mentioned in the prelude that when we evaluate SDF-PDFG of Fig. 3.3 at the configuration $x^G = \{p = 3, q = 2, a_1 = 5, a_2 = 4, a_3 = 3, a_4 = 4\}$ its instance SDFG emerges. This instance is equal to the scenario s_1 SDF-PDFG of Fig. 2.4a whose Max-plus matrix is given by (2.16). We consider an arbitrary entry of the left and right-hand side matrices of (3.66) for the example graph. With (3.65), $\left[\left(\mathcal{M}_G^{\mathrm{par}\prime}(x^G)\right)(x^G)\right]_{4,1} = 28.3$, while with (2.16), $[M_{\iota_G(x^G)}]_{4,1} = 22$. For growing values of p and q, i.e. for growing repetition vector entries the ratio component becomes dominant and the relative error shrinks and for $\{p, q\} \to \infty$ it reaches 0. E.g, with $x^G = \{p = 300, q = 200, a_1 = 5, a_2 = 4, a_3 = 3, a_4 = 4\}$, $\left[\left(\mathcal{M}_G^{\mathrm{par}\prime}(x^G)\right)(x^G)\right]_{4,1} = 1216.3$, while $[M_{\iota_G(x^G)}]_{4,1} = 1210$.

3.6 Problem definition

In this section we define the performance metrics of interest for SDF-PDF, i.e. worst-case throughput and worst-case latency.

3.6.1 Worst-case throughput

We define throughput of an SDF-PDFG in terms of numbers of iterations per time-unit, which is in accordance with the definition of the throughput used for SDF [50] and FSM-SADF [46] and the practical consideration that an iteration typically represents a coherent set of calculations, e.g. decoding a video frame.

An SDF-PDFG evolves in iterations of its nondeterministically sequenced instances. Therefore, a particular execution of an SDF-PDFG can be associated with a sequence of Max-plus timestamp vectors $\overline{\gamma} = \gamma(0), \gamma(1), \gamma(2), \ldots$

The completion time of the kth SDF-PDFG iteration is given by the norm of $\gamma(k)$, i.e. $||\gamma(k)||$. Therefore, we define the worst-case throughput of an SDF-PDFG by adopting the definition of worst-case throughput for FSM-SADF of [46] as follows.

Definition 3.4 (Worst-case throughput)**.** *Worst-case throughput of an SDF-PDFG G is defined as the largest value $Th_G \in \mathbb{R}$ such that for every possible instance sequence and its associated Max-plus timestamp vector sequence $\overline{\gamma} = \gamma(1), \gamma(2), \ldots$, for every $\epsilon \in \mathbb{R}$ such that $\epsilon > 0$, there is some $K \in \mathbb{N}_{>0}$ s.t. for all $L \in \mathbb{N}_{>0}$, $L > K$,*

$$\frac{L}{||\gamma(L)||} > Th_G - \epsilon. \tag{3.67}$$

Simply put, Definition 3.4 says that the throughput is the worst-case long-run average of completed iterations per time-unit. However, such a long-run average does not necessarily exist for all instance/configuration sequences. Instead it may bounce between superior and inferior limiting bounds [44][98] and therefore the need for a somewhat cumbersome formulation of Definition 3.4.

3.6.2 Worst-case latency

Similarly as proposed by [46][110] for FSM-SADF, we define the worst-case latency of an SDF-PDFG G relative to a desired period $\rho \in \mathbb{R}$.

Definition 3.5 (Worst-case latency)**.** *Worst-case latency of an SDF-PDFG G relative to a desired period $\rho \in \mathbb{R}$ is defined as the smallest vector L_G such that for every possible instance sequence and its associated Max-plus timestamp vector sequence $\overline{\gamma} = \gamma(1), \gamma(2), \ldots$, for every $k \geq 0$,*

$$\gamma(k) \preceq k \cdot \rho + L_G. \tag{3.68}$$

Definition 3.5 is a common definition of worst-case latency as a linear bound on actor firings of the form (3.68) where typically $\rho = \frac{1}{Th_G}$ [110]. Note that in contrast to throughput, where we are only interested in the asymptotic growth rate of $\overline{\gamma}$, to determine the worst-case latency we need to know at what time particular actor firings start or complete their firings within an iteration.

3.7 Performance analysis for SDF-PDF

3.7.1 Introductory remarks

The problem of worst-case performance analysis for SDF-PDFG is challenging due to four reasons: 1) SDF-PDFG actors execute in parallel within a graph iteration; 2) SDF-PDFG iterations overlap, i.e. they are pipelined (in Fig. 3.4, at $t = \tau$ two instances are concurrently active); 3) SDF-PDFG iterations are inter-dependent, i.e. synchronized by the availability of the initial tokens; and 4) SDF-PDFG is a dynamic dataflow structure, i.e. properties of consecutive iterations may drastically differ (cf. Fig. 3.4).

The definitions for worst-case throughput and latency for an SDF-PDFG (cf. Definitions 3.4 and 3.5) reveal that we need to consider the completion times of iterations. An SDF-PDFG evolves in iterations of its instances defined by configurations. Therefore, using the semantic link between SDF-PDF and FSM-SADF explained in Section 3.4.4 and the definition of the completion time (cf. (2.30)) of a sequence of FSM-SADF scenarios of (2.29) we can define the completion time of a sequence of configurations

$$\overline{x} = x_1^G, \ldots, x_k^G \in X_G^* \tag{3.69}$$

of an SDF-PDFG as follows

$$\mathcal{A} = \alpha^T \otimes \mu(\overline{x}) \otimes \beta, \tag{3.70}$$

where α is the final delay, $\mu : X_G^* \to \mathbb{R}_{\max}^{|I|\times|I|}$ is the morphism that associates sequences of configurations with Max-plus matrices as follows

$$\mu(\overline{x}) = \mathcal{M}_G(x_k^G) \otimes \ldots \otimes \mathcal{M}_G(x_1^G) \tag{3.71}$$

The structure $\mathcal{A} = (\alpha, \mu, \beta)$ defines the Max-plus automaton tuple of an SDF-PDFG G. Note that in (3.69), $\overline{x} \in X_G^*$ which means that configurations are sequenced nondeterministically/arbitrarily.

3.7.2 Worst-case throughput

The Max-plus automaton structure of (3.70) with the morphism μ of (3.71) fully captures the temporal behavior of a given SDF-PDFG. We use this structure to study the performance of SDF-PDF in a similar fashion it had been used to study the performance of FSM-SADF [46][98][95]. In particular, we focus on the results obtained for an FSM-SADFG with a fully connected FSM. This is because, as discussed in Section 3.4.4, the temporal behavior of an SDF-PDFG in terms of vectors $\gamma(k)$ is equal to that of an

FSM-SADFG obtained by calling all SDF-PDFG configurations/instances scenarios and allowing an arbitrary occurrence pattern between them. An arbitrary scenario occurrence pattern in terms of FSM-SADF is defined by using a fully connected FSM where the scenario labeling function is a bijection.

Let G be an SDF-PDFG. Define the worst-case evaluation Max-plus matrix of SDF-PDFG G as follows

$$M_G^{\mathrm{w-c}} = \bigoplus_{x_i^G \in X_G} \mathcal{M}_G(x_i^G). \tag{3.72}$$

Furthermore, let $M \in \mathbb{R}_{\max}^{n \times n}$ be a Max-plus matrix. The communication graph of $M \in \mathbb{R}_{\max}^{n \times n}$, denoted $\mathcal{G}(M) = (\mathcal{N}, \mathcal{E})$, is a graph with the set of nodes given by $\mathcal{N} = \{1, \ldots, n\}$ where a pair $(i, j) \in \mathcal{E} \subseteq \mathcal{N} \times \mathcal{N}$ is an edge of the graph if $[m]_{j,i} \neq -\infty$ and $[m]_{j,i}$ is the weight of that edge.

In analogy to the results obtained for an FSM-SADFG with a fully connected FSM (cf. Proposition 5.2 of [46] that directly follows from Theorem 2 of [40]), the worst-case throughput of SDF-PDFG G corresponds to the inverse of the maximum cycle mean (MCM) [34] of the communication graph of $M_G^{\mathrm{w-c}}$. Formally,

$$\frac{1}{Th_G} = mcm(\mathcal{G}(M_G^{\mathrm{w-c}})), \tag{3.73}$$

Where $mcm(\mathcal{G}(M))$ of a communication graph of an arbitrary Max-plus matrix M is defined as

$$mcm(\mathcal{G}(M)) = \max_c \frac{\sum_{e \in c} w(e)}{\sum_{e \in c} 1}, \tag{3.74}$$

where max is taken over all circuits c of $\mathcal{G}(M)$ and the sums are taken over all edges e of c. Map $w : \mathcal{E} \to \mathbb{R}$ returns the edge weight where $w((i, j)) = [m]_{j,i}$.

The intuitive explanation is as follows. The weights of the edges correspond the entries of $M_G^{\mathrm{w-c}}$ and represent minimal timing distances between tokens across consecutive iterations. As we consider executions composed of arbitrary large number of iterations, all such distances must reside within the cycles of the communication graph. Therefore, the inverse of the MCM of the communication graph equals to the worst-case throughput.

The problem now lies in determining $M_G^{\mathrm{w-c}}$ of (3.72). Enumeration of X_G imposes itself as a solution to the problem. However, as we already stated in Section 3.5.1 enumeration of X_G may not be feasible in practice due to size of X_G. Even with relatively small domains, if the repetition vectors of instances are relatively large, the generation of all $M_{\iota_G(x^G)}$ will take a significant amount of time as the instance needs to be simulated, i.e. every actor's firing needs to be symbolically performed. Actually, the experimental results of [46] show that the time needed to produced $M_{\iota_G(x^G)}$ scales "more than linearly" with the repetition vector entries.

However, the enumeration problem can be avoided by using the set of parameterized SDF-PDFG matrices as obtained by Algorithm 3.1. In that case, the problem of (3.72) transforms to

$$\begin{aligned} M_G^{\text{w-c}} &\preceq M_G^{\text{w-c}\prime} \\ &= \bigoplus_{M^{\text{par}} \in cod(\mathcal{M}_G^{\text{par}\prime})} \underbrace{\bigoplus_{x^G \in dom(\mathcal{M}_G^{\text{par}\prime}) \text{ s.t. } \mathcal{M}_G^{\text{par}\prime}(x^G) = M^{\text{par}}} M^{\text{par}}(x^G)}_{M^{\text{opt}}} \end{aligned} \tag{3.75}$$

In (3.75), $cod(\mathcal{M}_G^{\text{par}\prime})$ denotes the codomain of the mapping $\mathcal{M}_G^{\text{par}\prime}$. Simply put $cod(\mathcal{M}_G^{\text{par}\prime})$ is the set of parameterized matrices obtained by running Algorithm 3.1. Notation $dom(\mathcal{M}_G^{\text{par}\prime})$ is used for the domain of the mapping $\mathcal{M}_G^{\text{par}\prime}$. Then the notation $x^G \in dom(\mathcal{M}_G)$ s.t. $[\mathcal{M}_G(x^G)] = M^{\text{par}}$ denotes a particular natural subdomain of G. Note that the right hand side of (3.75) defines a conservative estimate of $M_G^{\text{w-c}}$ denoted $M_G^{\text{w-c}\prime}$ due to (3.66).

To determine the worst-case throughput of an SDF-PDFG we use $M_G^{\text{w-c}\prime}$ or formally,

$$\frac{1}{Th'_G} = mcm(\mathcal{G}(M_G^{\text{w-c}\prime})). \tag{3.76}$$

In this case, Th'_G may be a conservative estimate of the actual value denoted

$$Th'_G \leq Th_G. \tag{3.77}$$

We say "may" because if the critical cycle of $\mathcal{G}(M_G^{\text{w-c}\prime})$ has weights corresponding to entries (i, j) of $M_G^{\text{w-c}}$ where $[M_G^{\text{w-c}\prime}]_{i,j} = [M_G^{\text{w-c}}]_{i,j}$, then the inverse of the MCM of $\mathcal{G}(M_G^{\text{w-c}\prime})$ will be equal to the actual throughput value. Nevertheless, when it is a conservative estimate, for growing repetition vector entries of the graph, the relative estimation error moves towards 0.

In (3.75), matrix M^{opt} as a concrete matrix can be obtained by solving a series of optimization problems as follows:

foreach (i, j) s.t. $[M^{\text{par}}]_{i,j} \neq -\infty$ **do**

$$\begin{aligned} &\underset{x^G}{\text{maximize}} && [M^{\text{par}}(x^G)]_{i,j} \\ &\text{subject to} && x^G \in dom(\mathcal{M}_G^{\text{par}\prime}) \text{ s.t. } \mathcal{M}_G^{\text{par}\prime}(x^G) = M^{\text{par}}. \end{aligned} \tag{3.78}$$

ALGORITHM 3.2: Conservative approximation of the worst-case evaluation matrix of an SDF-PDFG.

```
Data:   Mapping $\mathcal{M}_G^{\mathrm{par}\prime}$
Result: Conservative approximation of the worst-case evaluation
        matrix of an SDF-PDFG, $M_G^{\mathrm{w-c}\prime}$
for $i = 1$ to $|I|$ do
    for $j = 1$ to $|I|$ do
        $[M_G^{\mathrm{w-c}\prime}]_{i,j} = -\infty$;
        foreach $M^{\mathrm{par}} \in cod(\mathcal{M}_G^{\mathrm{par}\prime})$ do
            if $[M^{\mathrm{par}}]_{i,j} \neq -\infty$ then
                $$tmp = \underset{x^G}{\texttt{maximize}} \quad [M^{\mathrm{par}}(x^G)]_{i,j}$$
                $$\texttt{subject to} \quad x \in dom(\mathcal{M}_G^{\mathrm{par}\prime}) \texttt{ s.t. } \mathcal{M}_G^{\mathrm{par}\prime}(x^G) = M^{\mathrm{par}} \quad (3.79)$$
                $[M_G^{\mathrm{w-c}\prime}]_{i,j} = [M_G^{\mathrm{w-c}\prime}]_{i,j} \oplus tmp$
            end if
        end foreach
    end for
end for
```

We summarize the procedure for obtaining $M_G^{\mathrm{w-c}\prime}$ in Algorithm 3.2. The input to the algorithm is the mapping $\mathcal{M}_G^{\mathrm{par}\prime}(x^G))$, while the output is the desired $M_G^{\mathrm{w-c}\prime}$ of (3.75). Each entry of $M_G^{\mathrm{w-c}\prime}$ corresponds to the maximal entry among all corresponding maximal entries of parameterized matrices defining the codomain of $\mathcal{M}_G^{\mathrm{par}\prime}$ (cf. Line 4). These entries on the other hand correspond to the maximum value an entry of the parameterized matrix attains when evaluated for all configurations within the subdomain the matrix is defined in. It is obtained by solving the optimization problem of (3.79), i.e. (3.78) where the objective function is the entry of the considered parameterized matrix, i.e. a parameterized expression. Recall that Algorithm 3.1 returns a set of parameterized matrices valid in different subregions of the SDF-PDFG domain. The type of optimization problems encountered in (3.78) depends on the formulations of $\mathcal{R}$ and $\mathcal{D}$ in the definition of our analysis model as well as on the specification of the SDF-PDFG domain. In the definition of $\mathcal{R}$ we were constrained by decidability of boundedness, deadlock-freedom and schedulability. If only rates were subject to parameterization, in the context of (3.78) we would be facing rational functions of polynomials. These problems can be converted to polynomial programming problems [23] and solved using the techniques of [94]. When it comes to the definition of $\mathcal{D}$, no restrictions regarding the functional behavior of SDF-

PDF exist. However, technical restrictions exist regarding the availability of optimization techniques needed to give global solutions to (3.78). To stay in the scope of polynomial programming, we limit $\mathcal{D}$ to polynomial functions of parameters or more formally

$$\mathcal{D} := k \mid k \cdot d \mid \mathcal{D}_1 \cdot \mathcal{D}_2 \mid \mathcal{D}_1 + \mathcal{D}_2 \tag{3.80}$$

where $d \in \mathcal{P}_d$ and $k \in \mathbb{R}_{\geq 0}$. When it comes to the definition of the X_G that with the definitions of $\mathcal{R}$ and $\mathcal{D}$ determines the type of the optimization problems we encounter, it is the designer's responsibility to specify the domain in a way that a global solver for the problem of (3.78) exists. For the solver of [94] we require that the domain is compact in the Euclidean sense, i.e. bounded and closed.

We exemplify using the SDF-PDFG of Fig. 3.3. For simplicity, assume that its domain is given by

$$\begin{aligned} X_G = & C_0 \cap C_1 \cap C_2 \cap C_3 \cap C_4 \cap C_5 \cap \\ & \big\{p = w_1 \cdot w_2, w_1 + w_2 = 2 \cdot x_1 - x_2, \\ & p \in [1,10], q \in [1,10], w_1 \in [1,3], w_2 \in [1,4], \\ & x_1 \in [1,3], x_2 \in [1,5], a_1 \in [1,7], a_2 = 4, \\ & a_3 \in [1,5], a_4 = 4\big\} \end{aligned} \tag{3.81}$$

Equation (3.81) is a very illustrative example of a domain specification because it shows how graph parameters (rates and actor firing delays) may exhibit arbitrary dependence on parameters not present in the graph itself in a nested fashion. E.g. parameterized rate p nonlinearly depends on parameters w_1 and w_2 which in turn depend on parameters x_1 and x_2. The domain (3.81) in addition defines a default parameter interval for each parameter, e.g $p \in [1, 10]$. For illustration purposes, in (3.81) we assume that the SDF-PDFG domain is a subset of the natural scenario subdomain defined by the constraints encountered while producing the matrix of (3.65). This way, during the generation of $M_G^{\text{w-c}\prime}$, (3.75) needs only to maximize over the entries of (3.65), i.e. run (3.78) only once. After running (3.78), we obtain the worst-case evaluation matrix specified by (3.82).

$$M_G^{\text{w-c}\prime} = \begin{bmatrix} 24 & -\infty & -\infty & -\infty & -\infty & 24 \\ 34.5 & 24 & -\infty & -\infty & -\infty & 34.5 \\ 34 & -\infty & 24 & -\infty & -\infty & 34 \\ 42.5 & 32 & 32 & 24 & -\infty & 42.5 \\ 42.5 & 32 & 32 & 24 & -\infty & 42.5 \\ -\infty & -\infty & -\infty & -\infty & 0 & -\infty \end{bmatrix} \tag{3.82}$$

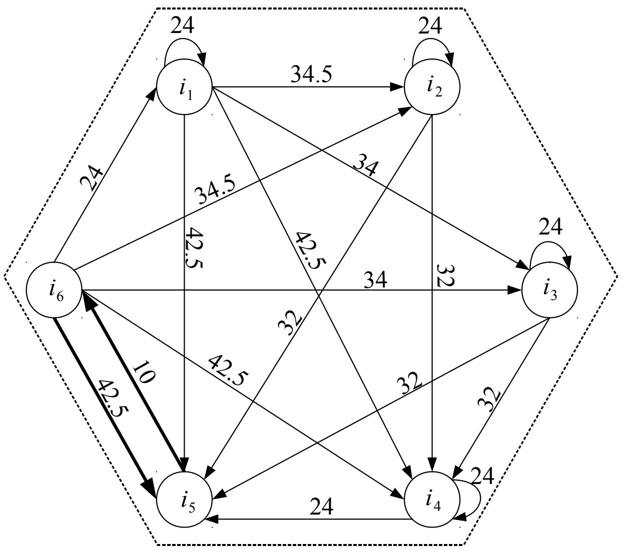


Fig. 3.12: Communication graph of Max-plus matrix of (3.82).

The communication graph of $M_G^{\text{w-c}\prime}$ is shown in Fig. 3.12. In the graphical representation of $\mathcal{G}(M_G^{\text{w-c}\prime})$, rather than using numerical values given by the set $\{1, \ldots, n, \ldots, |I|\}$ for node designators, we use the names of the initial tokens the numerical values refer to, e.g. value n corresponds to i_n. The critical cycle of the graph is marked with bold edges. The MCM of the graph is $mcm(\mathcal{G}(M_G^{\text{w-c}\prime})) = \frac{1}{2} \cdot (42.5 + 10) = 26.25$, and therefore $Th_G' = \frac{1}{26.25}$ iterations per time-unit.

The same specification can be analyzed for worst-case throughput by constructing a worst-case SDF abstraction of the parameterized specification by using parameter upper bounds for rates and firing delays and enabling all conditional channels. In that case, for the running example we would obtain the throughput value of $Th_G' = \frac{1}{70}$ time-units per iteration which is clearly a significant overestimation of the result obtained using our approach although our result is a conservative estimate itself. This is because in this case, the "worst-case" SDF abstraction cannot take into account the complex parameter interdependencies of (3.81).

3.7.3 Worst-case latency

From Definition 3.5 it follows that determining the worst-case latency equals to finding the smallest L_G such that $\gamma(k) \preceq L_G + k \cdot \rho$ holds for every possible sequence $\overline{\gamma}$. Therefore,

$$L_G = \bigoplus_k (\gamma(k) - k \cdot \rho). \tag{3.83}$$

It follows from (3.83) that given ρ we are interested in the maximal value $\gamma(k)$ can attain for all possible instance sequences. As follows from the Max-plus algebraic semantics of SDF-PDFG defined by (3.70) and (3.71) and the monotonicity of SDF-PDFG, $\gamma(k)$ can be conservatively bounded as follows.

$$\gamma(k) \preceq \gamma^{\mathrm{w-c}}(k). \tag{3.84}$$

In (3.84),

$$\gamma^{\mathrm{w-c}}(k) = M_G^{\mathrm{w-c}} \otimes \gamma^{\mathrm{w-c}}(k-1), \tag{3.85}$$

where $\gamma^{\mathrm{w-c}}(k)$ is the worst-case evaluation timestamp vector of the kth graph iteration. Matrix $M_G^{\mathrm{w-c}\prime}$ is used to define a bound on $\gamma^{\mathrm{w-c}}(k)$. Formally,

$$\gamma^{\mathrm{w-c}\prime}(k) \preceq \gamma^{\mathrm{w-c}}(k), \tag{3.86}$$

where

$$\gamma^{\mathrm{w-c}\prime}(k) = M_G^{\mathrm{w-c}\prime} \otimes \gamma^{\mathrm{w-c}\prime}(k-1) \tag{3.87}$$

for all $k > 0$. Therefore, the worst-case latency computation problem for SDF-PDF of (3.83) transforms to

$$L_G \preceq L'_G = \bigoplus_k (\gamma^{\mathrm{w-c}\prime}(k) - k \cdot \rho). \tag{3.88}$$

for the given enabling vector $\gamma^{\mathrm{w-c}\prime}(0)$. Typically, $\gamma^{\mathrm{w-c}\prime}(0) = \mathbf{0}$. The right hand side expression of (3.88) now represents a conservative bound on L_G that we want to determine, denoted L'_G. Formally,

$$L'_G \succeq L_G. \tag{3.89}$$

At first glance, this bound seems hard to compute because we need to consider all $\gamma^{\mathrm{w-c}\prime}(k)$ vectors up to an arbitrary large k as we consider finite SDF-PDFG executions of arbitrary length. However, the sequence $\gamma^{\mathrm{w-c}\prime}(k)$ as defined by (3.87) has a very nice property. In particular, it follows from the Max-plus spectral theory [61][7] that the sequence of vectors given by $\gamma(k+1) = M \otimes \gamma(k)$ where $M \in \mathbb{R}_{\max}^{n \times n}$ for $k \geq t$ where $t \in \mathbb{N}_{>0}$ will show a periodic behavior of type

$$\gamma(k+c) = \gamma(k) \otimes c \otimes \eta. \tag{3.90}$$

In (3.90), $\eta \in \mathbb{R}_{\max}^{m}$ is the cycle-time vector of M that is computed from the MCMs of the maximal strongly connected subgraphs of the communication graph of M. The value of c can be computed from the cyclicities of the maximal strongly connected subgraphs of M. The MCM of the communication graph of M will equal to the maximal among cycle-time vector entries. For more details we refer to [28] and [61].

Therefore, in the context of our $M_G^{\mathrm{w-c}\prime}$ where the inverse of its MCM corresponds to the throughput of the graph Th_G', by setting

$$\rho = \frac{1}{Th_G'}, \tag{3.91}$$

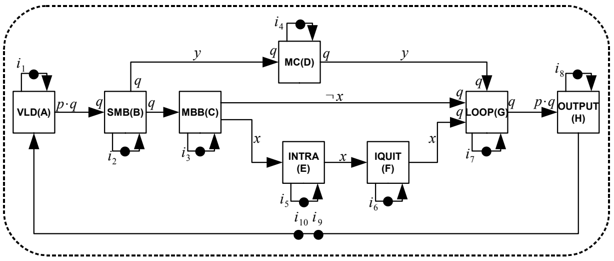
the periodicity property of (3.90) allows us to solve the problem of (3.88) by only determining the first $t + c$ timestamp vectors of (3.85), i.e. for $k = 1, \ldots, t + c$. This is because for the values of k beyond $t + c$, the growth rate of $\gamma^{\mathrm{w-c}}(k)$ cannot be faster than determined by the cycle-time vector and consequently the inverse of Th_G' that is the maximal among all cycle time vector entries and will not lead to a larger L_G'.

We compute the conservative bound to L_G for the running example SDF-PDFG with $M_G^{\mathrm{w-c}\prime}$ of (3.82), $\gamma^{\mathrm{w-c}\prime}(0) = [0,0,0,0,0,0]^T$ and $\frac{1}{Th_G'} = 26.25$ time-units per iteration. For $M_G^{\mathrm{w-c}\prime}$, $\eta = [26.25]^T$, $c = 2$ and $t = 2$ and therefore

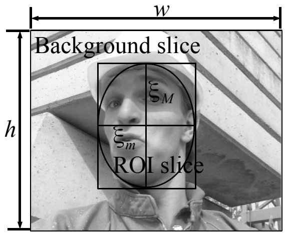
$$\begin{aligned} L_G' = \bigoplus \{ &[0,0,0,0]^T, [24, 34.5, 34, 42.5, 42.5, 10]^T - 26.25, \\ &[48, 58.5, 58, 66.5, 66.5, 52.5]^T - 52.5, \\ &[76.5, 87, 86.5, 95, 95, 76.5] - 78.75 \} \\ = &[0, 8.25, 7.75, 16.25, 16.25, 0]^T. \end{aligned}$$

3.8 Evaluation

In this section, we evaluate our analysis techniques on a realistic case study from the multimedia domain. In particular, we consider the case of a VC-1 video decoder used in a region of interest (ROI) coding scheme. We show how graph parameters can exhibit complex dependencies on the decoder's input signal parameters. Furthermore, we demonstrate that in the presence of such complex parameter dependencies, using the "worst-case SDFG" constructed from parameter interval endpoints in the worst-case throughput analysis will lead to a very pessimistic end result. With regard to that result, we show that our technique can give a significantly tighter but still a conservative estimate. Thereafter, we discuss the tightness of the performance bound and technical aspects of the analysis.



(a) VC-1 decoder captured in SDF-PDF.



(b) ROI coding.

Fig. 3.13: Case study.

3.8.1 VC-1 decoder

ROI coding [55] is a feature of modern video codecs that allows to independently store and transmit a video in a variety of regions of interest. This feature is useful for achieving higher error resilience as errors cannot cross ROI boundaries or for saving bandwidth as a ROI can be coded with more bits to obtain a much higher-quality than that of the non-ROI which is coded with fewer bits. Typical way of representing ROIs in a video picture is by the use of a rectangular region that corresponds to a picture slice. Slice on the other hand is a group of macroblocks. We exemplify using the picture from the *Foreman* sequence shown in Fig. 3.13b. In the sequence, the region of interest is the foreman's face represented by the rectangular "ROI slice", while the background is represented by the "Background slice".

The VC-1 decoder shown in Fig. 3.13a adopted from [13] is used to decode only ROI slices, i.e. the foreman's face. The decoder has two main pipelines: the *intra* pipeline (actors $MBB, INTRA$ and $IQUIT$) and the *inter* pipeline (actor MC). VLD performs variable-length decoding, actor SMB splits slices into macroblocks, actor $LOOP$ implements the deblocking filter, while actor $OUTPUT$ stores the decoded slice into the output frame buffer. One iteration of the SDF-PDFG of Fig. 3.13a corresponds to decoding of one picture slice. Boolean expressions defined over boolean parameters x and y are used to adjust the topology of the graph according to the type of slice subject to processing. In particular, we differentiate between three types of slices: 1) intra-coded only ($x \wedge \neg y$); 2) inter-coded only ($\neg x \wedge y$); and 3) both intra- and inter-coded ($x \wedge y$).

We assume the ROI (foreman's face) can be abstracted into an ellipse of known characteristics, i.e. of known circumference o and eccentricity ϵ where ξ_M and ξ_m are the major and minor axis of the ellipse, respectively. The ellipse abstraction is a natural representation for a face where eccentricity can be thought of as a characteristic of a particular face (some faces are more oval than the others) while the circumference models the distance of the face from the capturing device. The bounding rectangle of the ellipse defines the actual slice to be decoded. These considerations lead to the definition of X_G as follows

$$\begin{aligned}
X_G = \{ & p = (2 \cdot \xi_M \cdot 2 \cdot \xi_m)/(16 \cdot 16), p \in [1, P], && (3.92a)\\
& q \in [1, 16] && (3.92b)\\
& p' \geq \mu \cdot P, p + p' \leq P && (3.92c)\\
& o^2 = 4 \cdot \pi^2 (\xi_M^2 + \xi_m^2), o \geq O && (3.92d)\\
& \epsilon^2 \cdot \xi_M^2 = \xi_M^2 - \xi_m^2, \epsilon = E, 2 \cdot \xi_M \leq w, && (3.92e)\\
& 2 \cdot \xi_m \leq h && (3.92f)\\
& a = a_{\text{ref}}, b = b_{\text{ref}}, c = c_{\text{ref}}, d = d_{\text{ref}}, && (3.92g)\\
& e = e_{\text{ref}}, f = f_{\text{ref}}, g = q_{\text{ref}}, h = h_{\text{ref}} \}. && (3.92h)
\end{aligned}$$

The number of macroblocks p within the slice is given by the area of the ellipse's bounding rectangle (cf. (3.92a)). Note that the size of a macroblock is 16×16 pixels. Depending on resolution, the picture/frame consists of maximally P macroblocks (cf. (3.92a)). The number of blocks within a macroblock q is constrained by (3.92b). It is known that o is always greater than a certain predefined constant O (cf. (3.92d)), i.e. O defines the maximal distance from the face to the camera. Furthermore, ϵ is equal to a constant E and the ellipse is entirely contained inside the picture/frame (cf. (3.92e) and (3.92f)). Within a picture, it is assumed that the background always occupies the portion μ of the picture/frame comprising p' macroblocks (cf. (3.92c)). Referent actor execution times (cf. (3.92g) and (3.92h)) were taken from [13] and are expressed in cycles of the STMicroelectronics STxP70 processor.

From the case study we see the modeling flexibility the SDF-PDF offers. In particular, it allows to express fine-grained data dependent behavior using parameters. The value that parameters attain at run-time may in turn depend on the characteristics of the input data (the input signal). In the case study, these are the relative displacement of the tracked object (face) and the camera and the ovality of the face.

In the exercise, we assume SDTV input format with signal type 480i 16:9 and resolution 720x480 pixels. Thus, $w = 720$, $h = 240$ and $P = 1620$. Furthermore, $O = 700$, $E = 0.6$ and $\mu = 30$. For these values using our performance analysis technique presented in Section 3.7 we obtain a conservative throughput estimate of $1.74727 \cdot 10^{-7}$ slices per referent processor cycle. By using SDF in an upper endpoint manner (taking the maximal default values for all parameters) we obtain the guaranteed throughput value of $1.05699 \cdot 10^{-7}$ slices per cycle. Both values are conservative approximations of the actual value but our technique tightens the SDF result by 39.65 percent.

3.8.2 Tightness of performance bound and technical aspects of the analysis

The performance bounds derived by our analysis techniques are conservative.

For throughput, the conservativeness is solely incurred by Propositions 3.1 and 3.2 that are vital in deriving Max-plus algebraic abstraction of the worst-case behavior of the input SDF-PDF structure. This straightforwardly follows from (3.75). Still, as we have shown, the relative approximation error goes to 0 with the growing repetition vector entries of the graph. An exact result could be obtained by performing domain enumeration, were every configuration would be treated as a scenario of the implied FSM-SADFG. Still, for large domain as in the case of VC-1 decoder presented above, the analysis time of such an approach would be prohibitive. To illustrate, obtaining the worst-case throughput estimate using "worst-case SDF abstraction" of the VC-1 decoder case study in the SDF^3 tool [112] took 79.07 seconds on an Intel Core i5-750 CPU running at 2.67 GHz with 8GB main memory. Furthermore, 99% of that time was used to produce the SDFG Max-plus matrix (Algorithm 1 of [44]). This is because symbolic execution (simulation) of graphs with large repetition vector entries is costly in time (and memory). In particular, as mentioned, the time needed to produce the matrix scales "more than linearly" with the growing repetition vector entries [46]. Therefore, it would take all too long to process the entire domain of the VC-1 decoder case study and obtain the exact throughput value as the size of the domain is of the order of magnitude of the product set of parameter ranges, i.e. $\sim 16 \cdot 1620$.

For latency, the conservativeness is incurred by Propositions 3.1 and 3.2 as well as the fact that we use the matrix $M_G^{\mathrm{w-c}}$, i.e. $M_G^{\mathrm{w-c}\prime}$ to derive it. To derive an exact bound on latency one indeed needs to perform domain enumeration and appliy the state-space analysis of [46], but as we argue here, enumeration is not an option for applications with fine-grained data-dependent dynamics.

Our analysis is not fully automated. In particular, the Max-plus algebraic characterization of SDF-PDF specifications specified by Algorithm 3.1

is performed manually. Solving the problem of Algorithm 3.2 and the resulting MCM problem is automated. Because the core of the analysis is performed manually, we cannot provide relevant analysis time statistics for the current revision of the work.

Still, the results are promising. In particular, the manual part of the analysis for graphs similar in size (both in terms of actors, channels and the number of parameters) to the VC-1 decoder above, typically takes up to a few hours. This being the case, while taking into account the SDF^3 analysis time per configuration/instance of the SDF-PDFG, implies that a fully automated version of our analysis would outcompete the SDF^3 analysis in terms of run-time for applications with *large* domains. By "large" here we mean both in terms of cardinality and size of the repetition vector entries of the SDFGs that the domain configurations define. The automation of the manual part of the analysis would consist of the implementation of Algorithm 3.1 on top of the existing symbolic simulation algorithm of SDF^3 (Algorithm 1 of [44]). In the mere technical sense, this would entail the integration of a symbolic computation framework as GiNaC [10][11] and and a nonlinear constraint solver such as RealPaver [54] with the SDF^3 tool. A conservative estimate of the development time needed to accomplish this would amount up to a few months.

3.9 Summary

In this chapter we considered the worst-case performance analysis problem for dynamic streaming applications exhibiting fine-grained data-dependent dynamism that can be captured using SDF-PDF where application/design-space parameters expose complex interdependencies. So far, the problem was coarsely treated using the existing SDF techniques that typically incur too pessimistic performance estimates due to working with upper parameter interval endpoints. However, in practice there even may be cases where taking the upper parameter interval endpoints results in under-approximations because in a parameterized context it is not clear how to account for the "non-monotonic" effect of parameterized rates to the temporal behavior of the graph. From a real-time perspective the latter case is unacceptable. Therefore, in our work, using the Max-plus algebraic semantics of SDF-PDF we developed an analysis technique that works directly on parameterized graphs and thus avoids the pitfalls of approaches that try to "worst-case abstract" the parameterized specifications with SDF. Therefore, our technique is able to produce significantly tighter but still conservative performance estimates than the existing techniques. To show that our technique is not only of mere theoretical interest but also applicable in practice, we validated it on a realistic case-study from the multimedia domain for which we were able to produce tighter worst-case performance estimates than by using the existing techniques. This shows that in spite of the exponential worst-case complexity of our technique for Max-plus algebraic characterization of SDF-PDFGs, our technique is applicable in everyday engineering practice as for many applications only a few (critical) parameters will need to be accounted for.

Chapter 4

Parameterized dataflow scenarios

In Chapter 3 we presented SDF-PDF as a parameterized dataflow model based on SDF which specializes in capturing applications with fine-grained data-dependent dynamics.

SDF-PDF does not deviate from the dataflow framework, i.e. it is a "full-blooded" dataflow MoC. Although well-suited for capturing concurrency in streaming applications, purely dataflow-based models of computation are lacking in expressing intricate control requirements that many modern streaming applications have. Consequently, a number of modeling approaches combining dataflow and finite-state machines has been proposed. However, existing FSM/dataflow hybrids struggle with capturing the fine-grained data-dependent dynamics of modern streaming applications.

In this chapter, we enrich the set of such FSM/dataflow hybrids with a novel formalism that uses parameterized dataflow as the concurrency model. We call the model FSM-based parameterized scenario-aware dataflow (PFSM-SADF). Through the use of parameterized dataflow, the formalism can capture application's fine-grained data-dependent dynamics while the enveloping FSM enables the capturing of the application control flow. We demonstrate the application of our modeling framework to SDF, for which we propose a worst-case performance analysis framework based on the Max-plus algebraic semantics of SDF and the theory of Max-plus automata. We show that by using the novel hybrid one can give tighter bounds on worst-case performance metrics such as throughput and latency for streaming applications exposing fine-grained dynamic behavior embedded inside a control-flow structure than by using the existing hybrids. We evaluate our approach on a realistic case-study from the multimedia domain.

Different parts of this chapter have been published in [106] and are being considered for publication in [102].

4.1 Introduction

With regard to the concept used to represent the dataflow dynamics, the class of dynamic dataflow MoCs can be further refined into two subclasses [18].

The first subclass is composed out of dataflow MoCs that are developed around an interacting combination of finite-state machines (FSM) and dataflow graphs. Models such as heterochronous dataflow (HDF) [53] and FSM-based scenario-aware dataflow (FSM-SADF) [46] are well-known examples of such FSM/dataflow hybrids where an FSM is used to decouple control from concurrency. We bring further examples of such formalisms in Section 4.3.

In the second subclass, dataflow dynamics are represented by alternative means [18]. Examples of such models are BDF [22], DDF [22] and different parameterized dataflow models including our recently disclosed SDF-PDF (cf. Chapter 3).

The dichotomy between the two subclasses lies in the notion of state. The MoCs of the first subclass maintain the notion of state, while the MoCs belonging to the second subclass do not. In particular, the second subclass considers models that involve different kinds of modeling abstractions, apart from state transitions, as the key mechanisms for capturing dataflow behaviors and their potential for run-time variation [18].

The need for the first subclass is justified by the existence of streaming applications with both intricate control requirements and concurrency [53]. FSMs have long been used to describe and analyze control requirements. This is justified by their finite nature and strong formal properties.

The need for the second subclass, in particular for parameterized dataflow models that we focus on here, stems from the fact that there exist dynamic applications in which dynamism cannot be captured using a well-defined state structure. Examples are DSP applications whose behavior will depend on the results of some complex transformations performed on input signal. Due to the fact that the results of these transformations can span very large intervals, the existing FSM/dataflow hybrids with their finite nature inherited from the FSM are not an efficient abstraction for capturing fine-grained reconfiguration processes taking place in such applications. In particular, in this context, we may expect the hybrid as a whole to explode in size due to the size explosion of the underlying FSM.

Therefore, to keep the size of the problem manageable, other mechanism to capture run-time variation need to be employed. In case of applications exposing fine-grained data-dependent dynamics, a good modeling choice are parameterized dataflow models.

However, there exist many applications that combine the two, i.e. which have both intricate control requirements and that expose fine-grained data-dependent dynamics.

For this type of dynamic applications, to the best of our knowledge, none of the MoCs belonging to the two dynamic dataflow subclasses provide a natural and intuitive representation. Therefore, we argue that a combination of the two is needed.

In this chapter, we investigate an interacting combination of FSMs and parameterized dataflow. We use FSMs to capture the application control logic thanks to their finiteness, strong formal properties and an intuitive state abstraction that serves well for modeling control-oriented parts of the application. We use parameterized dataflow to express fine-grained data-dependent dynamics of data-dominated parts of the application thanks to its ability to combine dynamic parameters and run-time adaptation of parameters in a structured way.

We base the novel model on the concept of scenarios adopted from [46]. Consequently, we model the execution of an application as a sequence of modes called scenarios, each of which is represented by a parameterized dataflow structure while the scenario occurrence patterns are given by the superordinate FSM. We refer to the novel model as FSM-based parameterized scenario-aware dataflow (PFSM-SADF). We demonstrate the application of PFSM-SADF concept to SDF as it is arguably the most used, mature and stable dataflow formalism. We refer to this specialization of PFSM-SADF as SDF-based PFSM-SADF (SDF-PFSM-SADF) for which we develop novel parametric worst-case performance analysis techniques based on the Max-plus algebraic [7] semantics of SDF and the theory of Max-plus automata [40].

4.2 Motivational example

In this section we use the opportunity to motivate the need for a FSM/dataflow hybrid that can capture both application's intricate control requirements and its fine-grained data-dependent dynamics. A synthetic example of an application that has both intricate control requirements and that exposes fine-grained data-dependent dynamics is shown in Fig. 4.1. The application is composed out of three modules: a control module and two data processing modules `f1` and `f2`. The C specification of the control module is

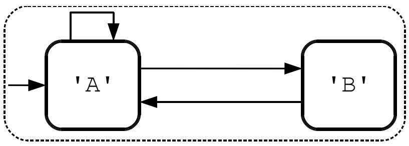
```
extern void f1(void);
extern void f2(void);
extern bool input(void);
void main(void){
   char state = 'A'; bool in;
   while(1){
      in = input();
      switch(state){
         case 'A':
            f1(); state = in ? 'A' : 'B';
            break;
         case 'B':
            f2(); state = 'A'; break;
         default:
            break;
}  }  }
```

(a) C specification of the control module.

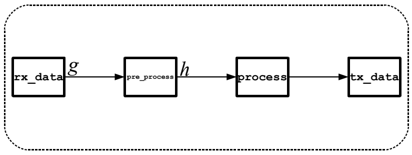
```
extern int rx_data(uint*,uint*)
extern int pre_process(int, uint);
extern int process(int, uint);
extern void tx_data(int);

void f1(void){
   uint g, h;
   int res1, res2, res3;
   res1 = rx_data(&g, &h);
   for(uint i=0; i < g; i++){
      res2 = pre_process(res1, i);
         for(uint j=0; j < h; j++){
            res3 = process(res2, j);
            tx_data(res3);
}        }        }
```

(b) C specification of f_1 module.



(c) FSM specification of the control module.



(d) Dataflow specification of f_1 module.

Fig. 4.1: Motivational example.

shown in Fig. 4.1a, while its FSM specification is shown in Fig. 4.1c. The control structure is simple and involves transitions between states `'A'` and `'B'` depending on the current state and the value of the control input `in`. Within a state, the execution of a data processing module is invoked. C specification of one of the data modules, namely `f1` is shown in Fig. 4.1b. A careful reader may notice that module `f1` fully corresponds to the application of Fig. 3.1a. In any case, the module consists of two nested loops. The loop bounds `g` and `h` are input data-dependent and computed within the `rx_data` submodule that implements some complex data transformation. Assume, as in Section 3.2, that parameter `g` is assigned with a value originating from the interval $[0, m/2]$, while `h` is assigned with a value from $[0, n/2]$. It this case, module `f1` will attain as many behaviors as there are integer points in the rational 2-polytope $P_{m,n}$ given by the set of constraints $\{0 \leq m/2, 0 \leq n/2\}$. With $n = 4500$ and $m = 2001$ the specification of Fig. 4.1b abstracts $2,252,126$ system behaviors [27]. Therefore, we can say that module `f1` and consequently the application as a whole expose fine-grained data-dependent dynamics recapitulated within the superordinate control structure. The data-dependent behavior of module `f1` can be succinctly expressed using the parameterized dataflow structure of Fig. 4.1d where loop bounds are abstracted into graph rates.

As mentioned in the Section 4.1, for this type of dynamic applications,to the best of our knowledge, none of the MoCs belonging the two dynamic dataflow subclasses provide a natural and intuitive representation. The models from the first subclass can express the control structure of Fig. 4.1a through the notion of state, but cannot express fine-grained data-dependent dynamics of the subordinate data processing module of Fig. 4.1b. The models from the second subclass (in particular parameterized dataflow models) can express the latter, but not the former, i.e. their design interface cannot expose the control structure directly to the programmer [18]. Therefore, a combination of the two is needed.

4.3 Related work

Parameterized dataflow as a meta-modeling technique was first introduced by [16]. The concept is applicable to any dataflow MoC that possess a well-defined notion of an iteration. The application of the concept to a target dataflow MoC, called the *base* model, extends the semantics of the base model by introducing arbitrary parameters that can be modified at run-time.

Following the publication of [16] various flavors of parameterized dataflow based on SDF MoC were introduced. These models include SPDF [38], BPDF [12] and VRDF [121]. Each of them was briefly described in Section 3.3.

In addition, there exist other, more general models such as parameterized and interfaced dataflow meta-model or shortly PiMM [35] and variable-rate phased dataflow (VPDF) [120].

PiMM is obtained by enriching the meta-modeling techniques of [16] with the notion of interfaces as introduced in interface based synchronous dataflow (IBSDF) [86]. This way, PiMM inherits the well-establish reconfiguration concepts of [16], while through the use of interfaces of IBSDF it enables design reuse, i.e. the design of independent graphs that can be instantiated in an entirely different design layout. PiMM as the parameterized dataflow of [16] can be applied to various base models.

VPDF is a CSDF-inspired generalization of VRDF where actors operate through sequences of phases. In each phase, the number of actor firings is parameterized along with the rates (or token transfer quantas in the parlance of [120]). The distinction is made to model loops for which no upper bound on their number of iterations is known. In contrast to VRDF, rates are allowed to have zero value and VPDF can model conditional execution.

To summarize, various flavors of parameterized dataflow models have been introduced as to support the growing need for efficient modeling tools that can capture both coarse and fine-grained reconfiguration phenomena present in modern streaming application. However, in their pure form, they are all inadequate for abstracting application control requirements as they do not depart from the dataflow framework and consequently do not provide interfaces to present application control requirements directly to the programmer. In addition, all of the models described expect VPDF and VRDF support no notion of time, i.e. they are untimed and therefore not accompanied by techniques for the analysis of their temporal behavior.

Next, we list models that do foster provision for expressing intricate control logic by defining precise semantics for integration of FSMs and dataflow.

Article [53] advocates for the use of a combination of hierarchical state machines and various concurrent MoCs to decouple control from concurrency. The approach is referred to as *charts (pronounced *starcharts*). When SDF is used in conjunction with FSMs, the resulting model is referred to as heterochronous dataflow (HDF). In HDF, two structural patterns are viable. First, an FSM can refine an SDF actor. In this case, the FSM must obey the SDF semantics externally. Second, an SDF can be used to refine an FSM state. SDF actor type signature (the number of tokens consumed and produced on each input and output) changes can occur only at iteration boundaries. This is ensured by not allowing the FSM components to change state until the last actor firing within an iteration had completed.

Scenario-aware dataflow (SADF) MoC introduced in [113] enables modeling and analysis of dynamic systems by allowing actors to operate in different *modes* or *scenarios* across firings. In different scenarios, actors have different execution times and rates. SADF uses a stochastic approach to model scenario occurrence patterns. The operational semantics is defined in terms of a labeled transition system that can be analyzed to obtain both long-run average and worst-case performance metrics.

FSM-SADF [46], in detail described in Section 2.4, is a model that is from the expressiveness point of view equivalent to HDF [110] but unlike HDF has known performance analysis techniques and allows iteration pipelining whereas HDF favors sequential schedules. FSM-SADF was introduced as a restriction of SADF in the sense that with FSM-SADF scenarios can change only between complete iterations of SDF models of the respective scenarios, while with SADF scenario changes are allowed within an iteration. Furthermore, the Markov automata of SADF is restricted to a (nondeterministic) FSM in FSM-SADF. On the other hand, FSM-SADF extends SADF because it allows auto-concurrent actor firings. The overall reduction in expressive power compared to SADF is advantageous from the analysis perspective. As in FSM-SADF a clearer separation between nondeterministic control flow and determinate dataflow computations can be made, the max-plus spectral analysis-based algorithms for determining performance numbers for FSM-SADF avoid the state space explosion problems that the original SADF analysis is prone to.

The DF* (pronounced “DFstar”) modeling framework of [31] is another dataflow MoC in the family of FSM/dataflow hybrids. A DF* graph is a network of blocks where each block consists of a set of code segments and a block controller. Each code segment specifies an alternative behavior of the block. The block controller is captured by a nondeterministic FSM. DF* is similar to HDF and consequently to FSM-SADF in the sense that code segments correspond to actor type signatures/scenarios.

The FunState MoC introduced in [117] defines precise semantics for separating dataflow from control in terms of functions driven by state machines.

Article [85] adds control flow provisions to bounded dynamic dataflow (BDDF) introduced in the same work yielding another FSM/dataflow hybrid. BDDF allowing varying port rates with a requirement that the upper bound of each data rate must be specified. The control flow is specified as an FSM. Each state is defined by a network of blocks and it is executed repeatedly until a combination of multiple events causes it to be stopped in a non-preemptive manner and another state entered.

The modeling and simulation framework called El Greco [20] provides facilities to dynamically change specification parameters. It supports specifications given as combinations of dataflow graphs and hierarchical FSMs. Data-dependent dynamics can be captured using the limited support for parameterized data rates. However, the framework is tailored for rapid simulation-based algorithm exploration. Therefore, it is not clear from [20] how to analytically analyze the model (in the presence of parameters).

All the aforementioned hybrids are inadequate for representing fine-grained dynamism and fine-grained reconfigurations of data-dominated application parts. This is mainly due to compactness issues because the finite nature of FSM limits its resolution and renders it inadequate for capturing fine-grained application dynamics. Furthermore, all of the listed models except SADF and FSM-SADF are untimed and in their current form cannot be used for performance analysis of systems.

4.4 Parameterized dataflow

4.5 Preliminary remarks

In this section we elaborate our parameterized dataflow modeling framework (refered to as PDF) which we use to define the parameterized dataflow scenarios concept of Section 4.7.

As mentioned in Section 4.3, the concept of parameterized dataflow was defined in [16]. Through a hierarchical discipline, it introduces dynamic parameters to dataflow MoCs called base models that have a well-defined notion of an iteration and by doing so it significantly increases their expressive power.

In parameterized dataflow of [16], each hieararchical actor is composed of 3 subgraphs, namely the *init* ϕ_i, the *subinit* ϕ_s and the *body* ϕ_b subgraph.

The *body* subgraph models the main functional behavior of the specification, while the *init* and *subinit* subgraphs control the behavior of the body graph by performing reconfiguration activity. This activity boils down to setting parameter values. These reconfiguration mechanisms are referred to as *initflow* to distinguish them from dataflow.

To assure run-time integrity of the application, parameters that influence the dataflow interfaces of subsystems (e.g. port rates) are only allowed to change once per iteration of the enclosing subsystem. Parameters that do not (e.g. parameters that concern functionality) influence the dataflow behavior of subsystems are allowed to change even more frequently.

To summarize, parameterized dataflow modeling framework of [16] provides reconfiguration facilities that can be exploited to achieve run-time control of actor's dataflow interfaces as well as the control of actor's functionality. Furthermore, parameterized dataflow concept of [16] by defining a modular decomposition of parameterized dataflow specifications into 3 subgraph types is an implementation-oriented, untimed and an architecture independent model. In particular, it favors development of portable application for heterogenous multiprocessor SoCs [35].

However, in our work, it is analysis that is in the focus of our interest, not synthesis. Therefore, from the performance analysis point of view, the parameterization of [16] is rather impractical because the addition of auxiliary graphs (init and subinit) increases the model complexity and reduces its intuitive appeal.

Therefore, to build a more analysis-oriented flavor of parameterized dataflow concept we favor a more abstract definition. Still, we reuse many of the concepts of [16]. More precisely, we think of PDF as a meta-modeling concept for integrating dynamic parameters and run-time adaptation of parameters in a structured way into a certain class of dataflow MoCs called the *base models* that have a well-defined concept of a graph iteration [18].

4.6 Our parameterization model

In PDF, actor firing rules can be reconfigured in-between actor firings using a set of parameters. More formally, a PDF actor is a dataflow actor $A = (P, Q, R, f)$ whose firing rules R depend on the set of parameters $\mathcal{P}_A$. In particular, these parameters control the firing of an actor by parameterizing conditions that need to be satisfied for an actor to fire. These conditions typically concern availability of input tokens, their values and the enclosing actor state. A PDF actor A admits the notion of configuration x^A that is obtained by assigning values to all parameters. Collected, configurations define the domain of A, denoted X_A. Once a configuration is applied to an actor, an instance of the parameterized actor emerges, denoted $\iota_A(x^A)$ that is nothing but a base model actor. Configurations can se applied to an actor in-between actor firings.

A parameterized dataflow graph (PDFG) is, of course, a composition of PDF actors. We define it by refining Definition 2.3 as follows.

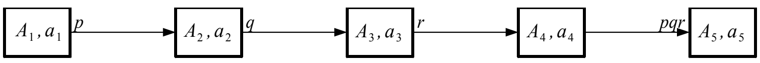
Definition 4.1 (PDFG). *A PDFG graph is a tuple $G = (\mathcal{A}, C, \mathcal{P}, X_G)$, where $\mathcal{A}$ is the set of actors, $C \subseteq \mathcal{A} \times \mathcal{A}$ the multiset of channels, $\mathcal{P}$ is the set of parameters, while X_G is the domain of the graph.*

In Definition 4.1, $\mathcal{P}$ is a set of parameters used to reconfigure the firing rules of the graph actors, while X_G is a set that collects all the configurations x^G of G.

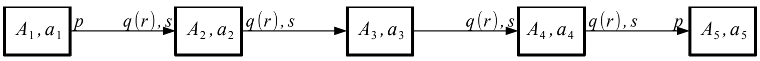
The concept of domain allows to explicitly represent the dependencies between parameters. These dependencies may arise from a variety of sources. One source may be the design environment itself. In particular, it may expresses the value of a parameter in terms of another.

Once x^G is applied to G, an instance $\iota_G(x^G)$ of G emerges. This instance is nothing but a base model graph.

To ensure the run-time integrity of the model, in PDF we allow reconfigurations only in-between graph iterations. However, this does not mean that reconfigurations of all actors must take place at the same time. Instead, they take place once the considered actor had completed all its firings withing the graph iteration. Therefore, a PDFG typically evolves in overlapped (pipelined) iterations defined by different configurations/instances.



(a) SDF-PDF specification of an artificial application.



(b) CSDF-PDF specification of an artificial application.

```
while(1){
   fire A1;
   repeat (p) times {
      fire A2;
      repeat (q) times {
         fire A3;
         repeat (r) times {
            fire A4;
         }
      }
   }
   fire A5;
}
```

(c) Schedule for the SDF-PDF specification.

```
while(1){
   fire A1;
   fire A2;
   repeat (q) times {
      repeat (q) times {
         fire A3;
      }
   }
   fire A2;
   repeat (s) times {
      fire A3;
   }
   fire A4;
   fire A5;
}
```

(d) Schedule for the CSDF-PDF specification with $p = q \cdot r + s$.

Fig. 4.2: SPDF and PCSDF specifications.

In our PDF flavor, we permit initial tokens on channels just as Petri nets have initial markings [80]. Furthermore, initial tokens are considered initial conditions for execution and their placement defines the initial system state. Now, using the concept of initial tokens, we define an iteration as a minimal non-empty set of actor firings that does not have a net effect on the (initial) system state. Therefore, if reconfiguration takes place in-between iterations, the system integrity is ensured. That is why PDF focuses on base models that have a well-defined notion of an iteration. Examples of such models are SDF and CSDF that originally host the iteration concept as defined above and even BDF where the *complete cycle*, if it can be found, corresponds to the notion of iteration. The ordering of configurations is nondeterministic, which renders PDF a nondeterministic MoC.

By applying it to SDF, the concept of PDF refines to the concept of SDF-PDF covered in detail in Chapter 3. For completeness, an example of an SDF-PDF structure is given here too in Fig. 4.2a along with its quasi-static schedule shown in Fig. 4.2c. Generally speaking, given a base model, it is then convenient to refine Definition 4.1 to a one more appropriate in the context of the considered base model. For SDF, Definition 4.1 refines to Definition 3.1. The key addition in Definition 4.1 is the introduction of rates. In particular, SDF is a uninterpreted dataflow MoC with conjunctive firing rules (cf. Section 2.3). Therefore, the firing conditions of SDF actors only concern the availability numbers of tokens, i.e. rates.

Moreover, although initially developed for SDF, the refinement of Definition 4.1 can also be used to define other uninterpreted refinements of PDF. We exemplify using CSDF-based PDF (CSDF-PDF). Recall that in CSDF, actor port rates exhibit a variation that forms a certain type of periodic pattern [19]. In the context of Definition 3.1, this can be modeled by setting

$$\mathcal{R} := k \mid p \mid \mathcal{R}_1(\mathcal{R}_2) \mid \mathcal{R}_1, \mathcal{R}_2, \tag{4.1}$$

where $k \in \mathbb{N}_{>0}$ and $p \in \mathcal{P}_i$ with $\mathcal{P}_i$ a set of symbolic variables by default constrained to $\mathbb{N}_{>0}$.

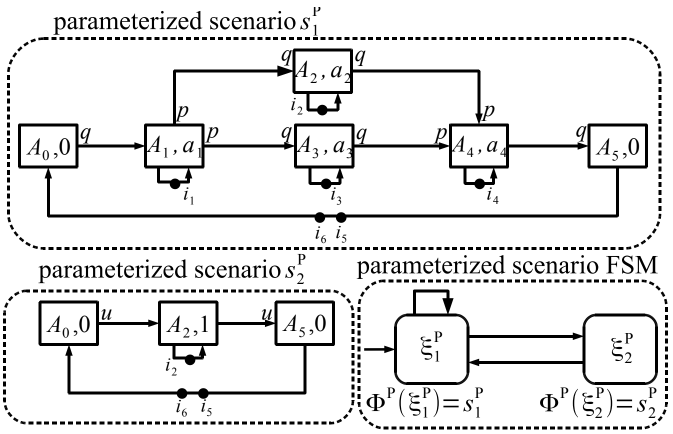
We exemplify using the CSDF-PDF structure of Fig. 4.2b. In the figure, notation $q(r), s$ generated with (4.1), denotes the parameterized cyclo-static dataflow sequence $r, r, r, \ldots, r, s$. In the sequence, r is repeated q times. This means that in the first q firings within an iteration, an actor awaits for the availability and eventually consumes r tokens, while in the last firing within an iteration it awaits and consumes s tokens. Note that with the definition of $\mathcal{R}$ of (4.1) we both parametrize the phase (length) and particular phase rates.

Furthermore, to complete the example, we define an arbitrary domain for the example graph that we specify with $X_G = \{p \in [50, 100], r \in [3, 20], q \in \mathbb{N}_{>0}, s \in \mathbb{N}_{>0}, p = q \cdot r + s, a_1 = a_2 = a_3 = a_4 = a_5 = 1\}$. With the requirement $p = qr + s$ stemming from the graph domain definition, it is relatively straightforward to derive a quasi-static schedule that defines the iteration of the structure. Such a schedule is shown in Fig. 4.2d.

4.7 Integration of parameterized dataflow and finite-state machines

4.7.1 The basics

PDF model defined in Section 4.4 introduces dynamic parameters to dataflow MoCs that have a well-defined concept of iteration. This way, it increases their expressive power by rendering them capable of modeling applications with fine-grained data dependent dynamics while keeping the model size manageable. However, PDF does not provide facilities to capture intricate control requirements that often accompany modern streaming applications because with PDF one cannot constrain reconfiguration patterns. Recall that in PDF reconfigurations happen at iteration boundaries where



(a) Example PFSM-SADFG

```
while(1){
    fire A0;
    repeat (q) times {
        fire A1;
    }
    repeat (p) times {
        fire A2;
    }
    repeat (p) times {
        fire A3;
    }
    repeat (q) times {
        fire A4;
    }
    fire A5;
}
```

(b) Scenario s_1^{P} Qss

```
while(1){
    fire A0;
    repeat (u) times {
        fire A2;
    }
    fire A5;
}
```

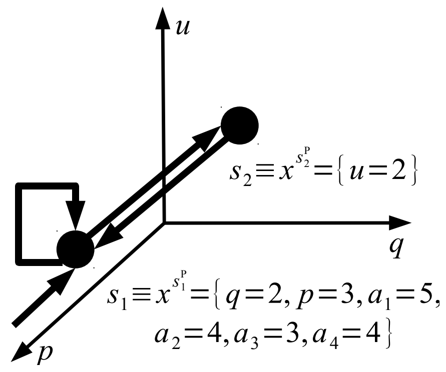
(c) Scenario s_1^{P} Qss

Fig. 4.3: PFSM-SADF.

the configuration to be applied next is nondeterministically chosen from the graph's domain, i.e. regardless of the current configuration.

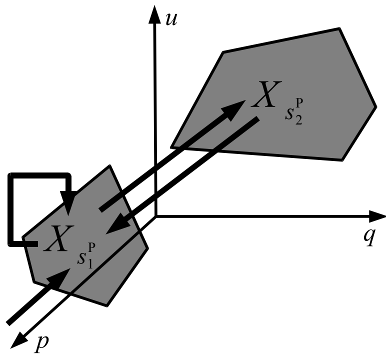
In this section we investigate the integration of PDF and FSMs in a concept we refer to as parameterized dataflow scenarios. More precisely, we propose a hybrid framework that extends PDF of Definition 4.1 with finite state control expressed via an FSM. We achieve this by generalizing the concept of FSM-SADF (HDF). In particular, we model the execution of an application as a sequence of parameterized scenarios or shortly scenarios. The sequencing of scenarios is dictated by the control structure captured by the scenario FSM while a particular scenario is represented by a PDFG. Scenario PDFGs may be entirely different (different parameters and graph structures), or can only differ in their respective domains. From now on, we use the terms scenario, PDFG and scenario PDFG interchangeably. We refer to the new model as FSM-based parameterized scenario-aware dataflow (PFSM-SADF).

We illustrate the concept using the structure of Fig. 4.3a where SDF is used as a base model of PDF. The composite PFSM-SADF graph (PFSM-SADFG) in the figure is defined over two parameterized scenarios s_1^{P} and s_2^{P}, each of which is modeled by a scenario PDFG in this case being an SDF-PDFG. Ordering of scenarios is dictated by the parameterized scenario FSM or shortly scenario FSM, where each state corresponds to one scenario. The example scenario FSM has two states: ξ_1^{P} and ξ_1^{P}. State ξ_1^{P} corresponds to



u
q
p
$s_2 \equiv x^{s_2^P} = \{u=2\}$
$s_1 \equiv x^{s_1^P} = \{q=2, p=3, a_1=5,$
$a_2=4, a_3=3, a_4=4\}$

(a) Operational semantics of FSM-SADF.



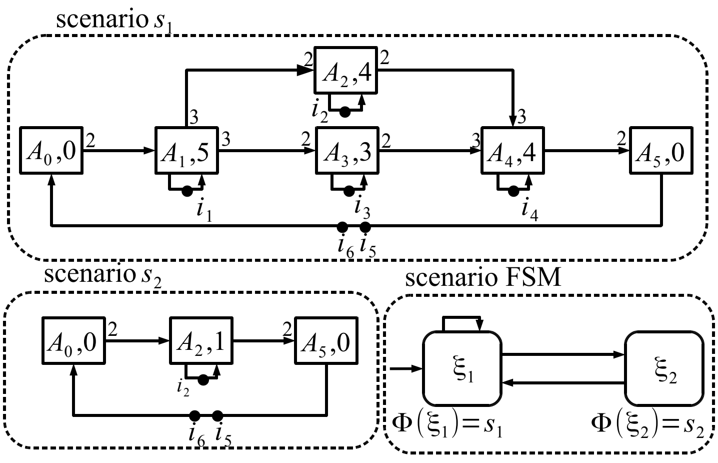
(b) Operational semantics of PFSM-SADF.

Fig. 4.4: Comparison of operational semantics of FSM-SADF and PFSM-SADF.

s_1^P and state ξ_2^P corresponds to s_2^P.

The operational semantics of PFSM-SADF is as follows. Each reaction/transition of the scenario FSM incurs the execution of one iteration of an arbitrary instance of the scenario PDFG that the transition destination state corresponds to. In consideration of an FSM/dataflow hybrid it is natural to link the dataflow graph iteration with the reaction of the FSM. This is because the reaction of the FSM will usually take a finite amount of time within which the dataflow subsystem needs to perform some computation and possibly emit output tokens as a result. This implies finiteness of computation that is not intrinsic to may models of computation [53]. In dataflow, the solution to the finiteness problem is simple. In particular, the notion of an iteration is used to assure finiteness of computation because a nonterminating execution of a dataflow graph can be divided into a set of iterations [53].

In PSDF context, the execution of one iteration of the scenario s_i^P PDFG, translates to the execution of one iteration of an arbitrary instance of the scenario s_i^P PDFG defined by some configuration $x^{s_i^P}$ originating from the scenario PDFG domain $X_{s_i^P}$ and denoted $\iota_{s_i^P}(x^{s_i^P})$. Therefore, the model fosters nondeterminism at two levels. The inter-scenario level, where the parameterized scenario to be activated next is nondeterministically chosen and at the intra-scenario level where one of the instances of that scenario is nondeterministically chosen to carry out the actual execution of the scenario. Fig. 4.4b illustrates the operational semantics of the PFSM-SADF of Fig. 4.3a. Scenario PDFG domains are depicted as 2-dimensional planes



(a) Example FSM-SADFG.

Fig. 4.5: Example FSM-SADFG.

in the $p-q-u$ space (we omit actor firing delay parameters). E.g., every time a transition $\xi_1^{\mathrm{P}} \to \xi_2^{\mathrm{P}}$ is taken, one iteration of the scenario s_2^{P} PDFG is executed. This corresponds to the execution of one iteration of an arbitrary instance of scenario s_2^{P} PDFG defined by the configurations found in the $X_{s_2^{\mathrm{P}}}$ hyperplane.

We use the opportunity to compare the operational semantics of PFSM-SADF to that of FSM-SADF and show that PFSM-SADF generalizes FSM-SADF. For the PFSM-SADFG of Fig. 4.3, let $X_{s_1^{\mathrm{P}}} = \{x^{s_1^{\mathrm{P}}}\}$ and let $X_{s_2^{\mathrm{P}}} = \{x^{s_2^{\mathrm{P}}}\}$ where $x^{s_1^{\mathrm{P}}} = \{q = 2, p = 3, a_1 = 5, a_2 = 4, a_3 = 3, a_4 = 4\}$ and $x^{s_2^{\mathrm{P}}} = \{u = 2\}$. Thus, our PFSM-SADFG contains only one configuration per parameterized scenario. If we apply these configurations to the scenario graphs that are SDF-PDFGs, we obtain two instance SDFGs. As usual, the scenario FSM defines the ordering of the activations of the two at run-time.

This structure is displayed in Fig. 4.5. This is nothing but an FSM-SADFG with two scenarios, namely s_1 and s_2 that correspond to instances $\iota_G(x^{s_1^{\mathrm{P}}})$ and $\iota_G(x^{s_2^{\mathrm{P}}})$ of the parameterized scenarios of the PFSM-SADFG of Fig. 4.3, respectively. Fig. 4.4a illustrates its operational semantics in the context of operational semantics of the PFSM-SADFG of Fig. 4.3. E.g., transition $\xi_1 \to \xi_2$ is refined by the execution of one iteration of the scenario s_2 SDFG (which can be obtained by applying the configuration $\{u = 2\}$ to the s_2^{P} PDFG of the PFSM-SADFG of Fig. 4.3). Consequently, Fig. 4.4a can be obtained by collapsing the hyperplanes of Fig. 4.4b into two configurations, namely $x^{s_1^{\mathrm{P}}}$ and $x^{s_2^{\mathrm{P}}}$.

This informally proves that PFSM-SADF generalizes FSM-SADF (HDF) because an FSM-SADFG is a special case of an SDF-based PFSM-SADFG with a single configuration per parameterized scenario. A reader may be skeptical towards the claim that the concept of parameterized dataflow scenarios also generalizes HDF because HDF unlike FSM-SADF (where the FSM is flat and sequential) enforces a hierarchical FSM discipline. This is not an issue because as stated in [53], hierarchy adds nothing to the model of computation but is used to reduce the number of transitions and makes the FSM more intuitive and easier to understand.

Furthermore, PFSM-SADF via PDF of Definition 4.1 allows scenario representations in base models other than SDF. A good example is CSDF, that when used as the base model of PFSM-SADF is at run-time instantiated into a CSDF graph.

We formally define the new model. First, we define the parameterized scenario FSM in Definition 4.2.

Definition 4.2 (Parameterized scenario FSM). *Given a set $\mathcal{S}^{\mathrm{P}}$ of parameterized scenarios, a parameterized scenario FSM F^{P} over $\mathcal{S}^{\mathrm{P}}$ is a tuple $F^{\mathrm{P}} = (\Xi^{\mathrm{P}}, \xi_0^{\mathrm{P}}, \mathbb{T}^{\mathrm{P}}, \Psi^{\mathrm{P}})$, where Ξ^{P} is the set of states, ξ_0^{P} is the initial state, $\mathbb{T}^{\mathrm{P}} : \Xi^{\mathrm{P}} \to 2^{\Xi^{\mathrm{P}}}$ is the transition function and $\Psi^{\mathrm{P}} : \Xi^{\mathrm{P}} \to \mathcal{S}^{\mathrm{P}}$ is the scenario labeling.*

Thereafter, we expose the definition of PFSM-SADF.

Definition 4.3 (PFSM-SADF). *PFSM-SADF F^{P} is a tuple $\mathsf{F}^{\mathrm{P}} = (\mathcal{S}^{\mathrm{P}}, F^{\mathrm{P}})$ where $\mathcal{S}^{\mathrm{P}}$ is the set of PDF scenarios and F^{P} is an FSM on $\mathcal{S}^{\mathrm{P}}$.*

4.7.2 Functional properties of PFSM-SADF

We briefly discuss the consistency, deadlock freedom and scheduling properties of PFSM-SADF. A PFSM-SADF progresses in iterations of its PDFG scenario instances like an FSM-SADF model progresses in iterations of its scenario SDFGs. It is natural to define scenarios at dataflow graph iteration granularity due to the need for the finiteness of the computation that refines an FSM reaction as well as due to a rather practical consideration where an iteration typically represents a coherent set of computations, e.g. decoding a video frame. The iteration boundaries in PDF are defined by points in time at which graph's initial tokens are restored. Recall that in the PDF domain, these tokens represent initial conditions for graph execution. Therefore, dependencies between two consecutive (parameterized) scenarios are captured by these initial tokens that we consider as inter-scenario synchronization data.

Given the operational semantics of PFSM-SADF and its synchronization principles described above, the same criteria for the consistency of applications modeled by PFSM-SADF as for those modeled by FSM-SADF applies, i.e. consistency of the application is guaranteed if all individual scenarios are consistent [96]. Deadlock freedom is a more subtle concept. Since our modeling targets streaming applications, we consider infinite traces of the scenario FSM. Therefore, an application modeled by PFSM-SADF is deadlock-free if all individual scenarios are deadlock-free and there exists no state in the scenario FSM with no outgoing transitions. As PFSM-SADF is a dynamic dataflow model which means that it cannot be statically scheduled. Still, as PFSM-SADFG progresses in scenario sequences, given a trace of the scenario FSM a schedule for the trace can be constructed by concatenating schedules of particular scenario graphs. Therefore, we deem a PFSM-SADFG schedulable if for all its scenario graphs considered in isolation a (quasi)-static schedule can be found.

Determining whether a particular scenario PDFG is consistent, deadlock-free and schedulable is not an easy task. In general, no general decidability criteria can be derived for the structure of Definition 4.1 and each refinement of Definition 4.1 needs to be analyzed separately. Furthermore, this type of functional analysis is outside the scope of this work and we will not consider it further. Instead, in the context of performance analysis of systems, we focus on a refinement of Definition 4.1 for which decidability criteria exists. In particular, we focus on SDF-PDF of Definition 3.1 described in detail in Section 3.4.3.

4.8 Parameterized synchronous dataflow scenarios

4.8.1 Basic remarks

We continue with further formalization of PFSM-SADF concept in the context of SDF because SDF has emerged as the most stable and mature dataflow MoC for representing streaming applications and systems [16]. By applying the concept of parameterized dataflow scenarios to SDF, we obtain parameterized dataflow scenarios or shortly SDF-based PFSM-SADF (SDF-PFSM-SADF).

In particular, we show how to capture the self-timed execution of SDF-PFSM-SADF by extending the Max-plus apparatus of FSM-SADF. Consequently, we propose a worst-case performance analysis framework for SDF-PFSM-SADF based on the theory of Max-plus automata.

4.8.2 Max-plus algebraic semantics of SDF-PFSM-SADF

SDF-PFSM-SADF is an SDF-based specialization of PFSM-SADF. Therefore, in SDF-PFSM-SADF, every parameterized scenario $s_j^{\mathrm{P}} \in \mathcal{S}^{\mathrm{P}}$ is represented by a parameterized scenario SDF-PDFG[1] of Definition 3.1. We denote the domain of scenario s_j^{P} SDF-PDFG with $X_{s_j^{\mathrm{P}}}$.

An SDF-PFSM-SADF graph (SDF-PFSM-SADFG) evolves in iterations of instances of the parameterized scenarios it encloses. As discussed in Section 4.7, synchronization between different scenarios is achieved by the set of tokens that exist in scenario PDFG, i.e. SDF-PDFG channels in-between iterations. These tokens correspond to the initial tokens of parameterized scenarios.

With SDF-PDF as the basic building block of SDF-PFSM-SADF, as shown in Section 3.5, the availability times of initial tokens can be captured using the timestamp vector of the kth graph iteration, i.e. $\gamma(k)$. Therefore, our first objective is to mathematically formulate the evolution of $\gamma(k)$ for an SDF-PFSM-SADFG. Following that, our second objective is to mathematically capture the completion time of a sequence of parameterized scenarios.

Let

$$\mathsf{M}_{\mathsf{F}^{\mathrm{P}}} = \{\mathcal{M}^{\mathrm{par}}_{s_1^{\mathrm{P}}}, \ldots, \mathcal{M}^{\mathrm{par}}_{|\mathcal{S}^{\mathrm{P}}|}\} \tag{4.2}$$

be the set of all mappings of that per particular scenario given the scenario SDF-PDFG configuration return the associated parameterized scenario matrix (recall the semantics of $\mathcal{M}^{\mathrm{par}}_{G}$ from (3.8)). Now, the operational semantics of SDF-PFSM-SADF says that an SDF-PFSM-SADFG evolves in iterations of its scenario SDF-PDFG instances where the scenario occurrence patterns are given by the scenario FSM. Scenarios are synchronized by the set of initial tokens whose production times are in turn captured by the initial token timestamp vectors.

Therefore, the timestamp vectors of initial tokens after the $(k+1)$st SDF-PFSM-SADFG iteration can be related to the timestamp vector of initial tokens after the kth iteration as follows

$$\gamma(k+1) = \underbrace{\big((\mathbb{M}_{\mathsf{F}^{\mathrm{P}}}(\pi_l(\zeta_{\mathsf{F}^{\mathrm{P}}}(k+1))))(\pi_r(\zeta_{\mathsf{F}^{\mathrm{P}}}(k+1)))\big)(\pi_r(\zeta_{\mathsf{F}^{\mathrm{P}}}(k+1))}_{M_{\iota_{\pi_l(\zeta_{\mathsf{F}^{\mathrm{P}}}(k+1))}}(\pi_r(\zeta_{\mathsf{F}^{\mathrm{P}}}(k+1)))} \otimes \gamma(k). \tag{4.3}$$

In (4.3), mapping $\zeta_{\mathsf{F}^{\mathrm{P}}}(k+1) = (s_j^{\mathrm{P}}, x^{s_j^{\mathrm{P}}})$ returns the active scenario of the $(k+1)$st SDF-PFSM-SADFG iteration as well as its configuration, while

[1] We will be using the terms parameterized scenarios SDF-PDFG, scenario SDF-PDFG and scenario interchangeably.

$\mathbb{M}_{\mathsf{F}^{\mathrm{P}}} : \mathcal{S}^{\mathrm{P}} \to \mathsf{M}_{\mathsf{F}^{\mathrm{P}}}$ returns the configuration to parameterized Max-plus matrix mapping of a particular scenario and π_l and π_r are the left and right projection functions, respectively.

Finally, the matrix of the underbrace of (4.3) is the Max-plus matrix of the SDFG instance running as the $(k+1)$st iteration of the SDF-PFSM-SADFG. To explain the cumbersome notation of (4.3), we use (4.4).

$$\begin{aligned}(k+1) &\xrightarrow{\zeta_{\mathsf{F}^{\mathrm{P}}}(k+1)} (s_j^{\mathrm{P}} \in \mathcal{S}^{\mathrm{P}}, x^{s_j^{\mathrm{P}}} \in X_{s_j^{\mathrm{P}}}) \xrightarrow{\mathbb{M}_{\mathsf{F}^{\mathrm{P}}}(s_j^{\mathrm{P}})} \mathcal{M}^{\mathrm{par}}_{s_j^{\mathrm{P}}} \xrightarrow{\mathcal{M}^{\mathrm{par}}_{s_j^{\mathrm{P}}}(x^{s_j^{\mathrm{P}}})} \left(\mathcal{M}^{\mathrm{par}}_{s_j^{\mathrm{P}}}(x^{s_j^{\mathrm{P}}})\right) \\ &\xrightarrow{\left(\mathcal{M}^{\mathrm{par}}_{s_j^{\mathrm{P}}}(x^{s_j^{\mathrm{P}}})\right)(x^{s_j^{\mathrm{P}}})} M_{\iota_{\pi_l(\zeta_{\mathsf{F}^{\mathrm{P}}}(k+1))}(\pi_r(\zeta_{\mathsf{F}^{\mathrm{P}}}(k+1)))}\end{aligned} \tag{4.4}$$

Starting from the SDF-PFSM-SADFG iteration indexed with $(k+1)$ by applying $\zeta_{\mathsf{F}^{\mathrm{P}}}$, the running parameterized scenario and its running configuration are determined as a pair $(s_j^{\mathrm{P}} \in \mathcal{S}^{\mathrm{P}}, x^{s_j^{\mathrm{P}}} \in X_{s_j^{\mathrm{P}}})$. Thereafter s_j^{P} is used as an argument to $\mathbb{M}_{\mathsf{F}^{\mathrm{P}}}$ to determine the mapping from configurations to parameterized matrices within the scenario (cf. (4.2)), i.e. $\mathcal{M}^{\mathrm{par}}_{s_j^{\mathrm{P}}}$. After the mapping had been determined, it is used to determine the parameterized matrix corresponding to the configuration $x^{s_j^{\mathrm{P}}}$ from the initial pair. This matrix is denoted $(\mathcal{M}^{\mathrm{par}}_{s_j^{\mathrm{P}}}(x^{s_j^{\mathrm{P}}}))$. Finally, this matrix is evaluated at $x^{s_j^{\mathrm{P}}}$ which gives the Max-plus matrix of the running scenario SDFG instance, denoted $M_{\iota_{\pi_l(\zeta_{\mathsf{F}^{\mathrm{P}}}(k+1))}(\pi_r(\zeta_{\mathsf{F}^{\mathrm{P}}}(k+1)))}$.

We use the opportunity here to graphically exemplify $\gamma(k)$. Consider the SDF-PFSM-SADFG of Fig. 4.3. Let $X_{s_1^{\mathrm{P}}} = \{x^{s_1^{\mathrm{P}}}\}$ and let $X_{s_2^{\mathrm{P}}} = \{x^{s_2^{\mathrm{P}}}\}$ where $x^{s_1^{\mathrm{P}}} = \{q = 2, p = 3, a_1 = 5, a_2 = 4, a_3 = 3, a_4 = 4\}$ and $x^{s_2^{\mathrm{P}}} = \{u = 2\}$. Then the evolution of $\gamma(k)$ for scenario sequence $\overline{s}^{\mathrm{P}} = s_1^{\mathrm{P}}, s_2^{\mathrm{P}}, s_1^{\mathrm{P}}, s_2^{\mathrm{P}}, s_1^{\mathrm{P}}, s_1^{\mathrm{P}}$ is shown in Fig. 4.6.

We now dedicate to our second objective, i.e. the computation of the completion time of a sequence of parameterized scenarios

$$\overline{s}^{\mathrm{P}} = s_1^{\mathrm{P}}, \ldots, s_k^{\mathrm{P}} \in \mathcal{S}^{\mathrm{P}*} \cap L \tag{4.5}$$

that we consider a part of the scenario FSM trace (if scenarios FSM transitions were labeled with scenarios defining the transition destination states, and all the states were declared accepting, L would be considered a language the scenario FSM accepts).

Due to the fact that the activation of a scenario entails the activation of an arbitrary scenario SDFG instance (nondeterministic choice), it is conve-

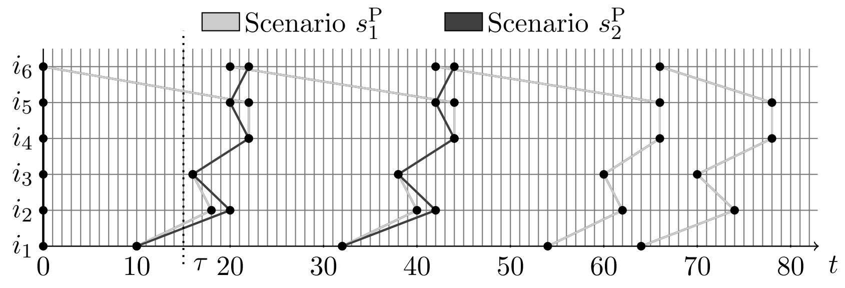


Fig. 4.6: FSM-SADF.

nient to expand (4.5) as follows

$$\overline{s}^{\mathrm{P}} = (s_1^{\mathrm{P}}, x_1^{s_1^{\mathrm{P}}} \mid \ldots \mid x_{|X_{s_1^{\mathrm{P}}}|}^{s_1^{\mathrm{P}}}), \ldots, (s_k^{\mathrm{P}}, x_1^{s_k^{\mathrm{P}}} \mid \ldots \mid x_{|X_{s_k^{\mathrm{P}}}|}^{s_k^{\mathrm{P}}}), \tag{4.6}$$

where bar $|$ denotes a nondeterministic choice. Then, from the recursion of (4.3), we can derive an explicit function for the completion time of (4.6) as follows

$$\mathcal{A}^{\mathrm{P}}(\overline{s}^{\mathrm{P}}) = \alpha^{\mathrm{P}^T} \otimes \mu^{\mathrm{P}}(\overline{s}^{\mathrm{P}}) \otimes \beta^{\mathrm{P}} \tag{4.7}$$

where α^{P} is the final delay, β^{P} is the initial delay and $\mu^{\mathrm{P}} : \mathcal{S}^{\mathrm{P}*} \to \mathbb{R}_{\max}^{I \times I}$ is a morphism that associates sequences of parameterized scenarios $\overline{s}$ with Max-plus matrices as follows

$$\begin{aligned} \mu^{\mathrm{P}}(\overline{s}^{\mathrm{P}}) &= \mu^{\mathrm{P}}((s_1^{\mathrm{P}}, x_1^{s_1^{\mathrm{P}}} \mid \ldots \mid x_{|X_{s_1^{\mathrm{P}}}|}^{s_1^{\mathrm{P}}}), \ldots, (s_k^{\mathrm{P}}, x_1^{s_k^{\mathrm{P}}} \mid \ldots \mid x_{|X_{s_k^{\mathrm{P}}}|}^{s_k^{\mathrm{P}}})) \\ &= \left((\mathbb{M}_{\mathsf{F}^{\mathrm{P}}}(s_1^{\mathrm{P}}))\, (x_1^{s_k^{\mathrm{P}}} \mid \ldots \mid x_{|X_{s_k^{\mathrm{P}}}|}^{s_k^{\mathrm{P}}}) \right) (x_1^{s_k^{\mathrm{P}}} \mid \ldots \mid x_{|X_{s_k^{\mathrm{P}}}|}^{s_k^{\mathrm{P}}}) \otimes \cdots \\ &\otimes \left((\mathbb{M}_{\mathsf{F}^{\mathrm{P}}}(s_1^{\mathrm{P}}))\, (x_1^{s_1^{\mathrm{P}}} \mid \ldots \mid x_{|X_{s_1^{\mathrm{P}}}|}^{s_1^{\mathrm{P}}}) \right) (x_1^{s_1^{\mathrm{P}}} \mid \ldots \mid x_{|X_{s_1^{\mathrm{P}}}|}^{s_1^{\mathrm{P}}}). \end{aligned} \tag{4.8}$$

The triple $\mathcal{A}^{\mathrm{P}} = (\alpha^{\mathrm{P}}, \mu^{\mathrm{P}}, \beta^{\mathrm{P}})$ defines the Max-plus automaton structure of [40]. In (4.7), β^{P} captures the initial enabling times of graph's initial tokens, i.e. $\beta^{\mathrm{P}} = \gamma(0)$ and typically $\gamma(0) = \mathbf{0}$. On the other hand α^{P} specifies the metrics we are interested in. E.g., if we are interested in the makespan of the scenario sequence, we set $\alpha^{\mathrm{P}} = \mathbf{0}$.

The Max-plus automaton structure of (4.7) with (4.8) can be used to study the performance of SDF-PFSM-SADF in a similar fashion as it had been used to study the performance of FSM-SADF. By comparing the Max-plus automata structure of FSM-SADF (cf. (2.30) and (2.31)) and SDF-PFSM-SADF (cf. (4.7) and (4.8)) we observe a striking resemblance. Both in FSM-SADF and PFSM-SADF the Max-plus automata structures are defined on a finite number of scenarios. However the crucial difference lies in the definition of automata morphisms of (2.31) and (4.8). In FSM-SADF, the Max-plus automata morphism μ returns for a scenario the corresponding scenario SDFG matrix and for each scenario there is only one such matrix. With SDF-PFSM-SADF, the mapping μ^{P} for a parameterized scenario returns the Max-plus matrix of an arbitrarily chosen parameterized scenario SDFG instance. Further comparison of the morphisms reveals that (4.8) can be unfolded into (2.31) in a way that every parameterized scenario instance would become a scenario in an equivalent FSM-SADF. This follows straightforwardly from the discussion on operational semantics of PFSM-SADF compared to that of FSM-SADF of Section 4.7. Then the equivalent structure could be used to analyze the original parameterized specification for worst-case performance. However, in practice, this is not feasible because Max-plus automata-based techniques for throughput analysis of FSM-SADF rely on the automata product structure which would explode in size due to unfolding. The same effect would incapacitate the use of state-space-based latency analysis techniques.

However, as we show in the section to come, for worst-case performance analysis, such an unfolding is not even necessary.

4.9 Worst-case performance analysis for parameterized synchronous dataflow scenarios

SDF-PFSM-SADF has a well-defined concept of iteration it inherits from SDF-PDF. Furthermore, timestamp vector $\gamma(k)$ of (4.3) is used to capture the evolution of SDF-PFSM-SADF across iterations in a similar manner it was used to capture the evolution of SDF (cf. (2.12)), FSM-SADF (cf. (2.28) and SDF-PDF (cf. (3.7)).

Therefore, it is only natural to use the concept of iteration to define performance metrics for PFSM-SADF similarly as done for SDF, FSM-SADF and SDF-PDF.

4.9.1 Worst-case throughput

In particular, for worst-case throughput, we adopt the following definition (cf. Definition 3.4 and Definition 2 of [46]).

Definition 4.4 (Worst-case throughput)**.** *Worst-case throughput of an SDF-PFSM-SADF* F^{P} *is defined as the largest value* $Th_{\mathsf{F}^{\mathrm{P}}} \in \mathbb{R}$ *such that for every* *possible instance sequence and its associated Max-plus timestamp vector sequence $\overline{\gamma} = \gamma(1), \gamma(2), \ldots$, for every $\epsilon \in \mathbb{R}$ such that $\epsilon > 0$, there is some $K \in \mathbb{N}_{>0}$ s.t. for all $L \in \mathbb{N}_{>0}$, $L > K$,*

$$\frac{L}{||\gamma(L)||} > Th_{\mathsf{F}^{\mathrm{P}}} - \epsilon. \tag{4.9}$$

FSM-SADF uses the Max-plus automaton structure of 2.30 for worst-case throughput analysis. Furthermore, Section 4.8 explaines how the SDF-PFSM-SADF Max-plus automaton structure of (4.7) could be unfolded into an equivalent structure of 2.30. However, the unfolding will lead to an explosion in the problem size for scenarios with large domains. Therefore, the problem needs to be approached from another angle.

In worst-case throughput analysis of FSM-SADF, one is interested in the worst-case increase of $\mathcal{A}(\overline{s})$ of 2.30 for a growing length of $\overline{s}$. This inverse of this worst-case increase represents the worst-case throughput of the graph. The increase is in turn computed as the MCM of the throughput graph of the FSM-SADF [46][98].

With SDF-PFSM-SADF, in consideration of its operational semantics where execution of a parameterized scenario corresponds to the execution of one of its arbitrarily chosen instances, the worst-case increase of $\mathcal{A}^{\mathrm{P}}(\overline{s}^{\mathrm{P}})$ will be defined using worst-case evaluation matrices $M^{\mathrm{w-c}}_{s^{\mathrm{P}}_j}$ of enclosed parameterized scenarios $s^{\mathrm{P}}_j \in \mathcal{S}^{\mathrm{P}}$. For a parameterized scenario, a conservative estimate of this matrix can be obtained using Algorithm 3.2.

For that matter, given a PFSM-SADFG F^{P} let $\overline{s}^{\mathrm{P}} = s^{\mathrm{P}}_1, \ldots, s^{\mathrm{P}}_k \in \mathcal{S}^{\mathrm{P}} \cap L$. Then, in consideration of (4.3) and (4.7), the following inequality holds

$$\mathcal{A}^{\mathrm{P}}(\overline{s}^{\mathrm{P}}) \preceq \alpha^{\mathrm{P}^T} \otimes \underbrace{M^{\mathrm{w-c}}_{s^{\mathrm{P}}_k} \otimes \ldots \otimes M^{\mathrm{w-c}}_{s^{\mathrm{P}}_1}}_{\succeq \mu^{\mathrm{P}}(\overline{s}^{\mathrm{P}})} \otimes \beta^{\mathrm{P}}. \tag{4.10}$$

The right hand side of (4.10) establishes a conservative upper bound on $\mathcal{A}^{\mathrm{P}}(\overline{s}^{\mathrm{P}})$ of the left hand size. However, as immediately follows from Theorem 2 of [40], the worst-case increases of both sides are equal.

Therefore, the representation of the right-hand side of (4.10) thanks to its relatively small size that measures in the number of parameterized scenarios can now be used for worst-case throughput analysis of SDF-PFSM-SADF. We may think of it as a “reduced Max-plus automaton” of the SDF-PFSM-SADF that abstracts an entire parameterized scenario into one matrix.

The worst-case throughput of the considered SDF-PFSM-SADFG will equal to the inverse of the MCM of its throughput graph. We now show how to construct this throughput graph.

In the construction of the throughput graph we will actually be using the conservative estimates of $M^{\mathrm{w-c}}_{s^{\mathrm{P}}_j}$ matrices obtained using Algorithm 3.2, i.e. $M^{\mathrm{w-c}\prime}_{s^{\mathrm{P}}_j}$. Therefore, the computed throughput $Th'_{\mathsf{F}^{\mathrm{P}}}$ will be a conservative estimate of $Th_{\mathsf{F}^{\mathrm{P}}}$. However, here as in the case of SDF-PDF, the relative approximation error goes to 0 with growing scenario graph repetition vector entries.

We specify now the throughput graph construction process. Traverse over all scenario FSM states $\xi^{\mathrm{P}}_u \in \Xi^{\mathrm{P}}$ and add a node to the throughput graph for each initial token $i_v \in I$ of the SDF-PFSM-SADFG and label the node with $(\xi^{\mathrm{P}}_u, i_v)$. Then for every transition $(\xi^{\mathrm{P}}_r, \xi^{\mathrm{P}}_s)$ add an edge from node $(\xi^{\mathrm{P}}_r, i_m)$ to node $(\xi^{\mathrm{P}}_s, i_n)$ if $[M^{\mathrm{w-c}\prime}_{\Psi^{\mathrm{P}}(\xi^{\mathrm{P}}_s)}]_{n,m} \neq -\infty$ and set the weight of the edge to $[M^{\mathrm{w-c}\prime}_{\Psi^{\mathrm{P}}(\xi^{\mathrm{P}}_s)}]_{n,m}$.

These weights represent minimal distances between initial tokens across two consecutive scenarios. Over infinite sequences of scenarios these distances will be part of the throughput graph cycles. Therefore, the inverse of the MCM of the throughput graph defines the worst-case throughput value, i.e. its conservative estimate because we are using $M^{\mathrm{w-c}\prime}_{s^{\mathrm{P}}_j}$ matrices.

We exemplify using the running example SDF-PFSM-SADF of Fig. 4.3. Assume that the respective scenario SDF-PDFG domains are given as follows

$$\begin{aligned}
X_{s^{\mathrm{P}}_1} =& (p \cdot a_1 \geq q \cdot a_1) \cap (p \cdot a_3 \leq q \cdot a_1) \\
& \cap (q \cdot a_4 \leq p \cdot a_2) \cap (a_4 \leq a_4) \cap (q \cdot a_4 \leq p \cdot a_3) \\
& \cap \big\{p = w_1 \cdot w_2, w_1 + w_2 = 2 \cdot x_1 - x_2, \\
& p \in [1,10], q \in [1,10], w_1 \in [1,3], w_2 \in [1,4], \\
& x_1 \in [1,3], x_2 \in [1,5], a_1 \in [1,7], a_2 = 4, \\
& a_3 \in [1,5], a_4 = 4\big\}
\end{aligned} \tag{4.11}$$

and

$$X_{s^{\mathrm{P}}_2} = \{u = 30\} \tag{4.12}$$

Equation (4.11) is a very illustrative example of a domain specification because it shows how the concept of domain explicitly represents the dependencies between parameters of the graph $\{p, q, a_1, a_2, a_3, a_4\}$ and design environment parameters $\{x_1, x_2, w_1, w_2\}$. After running Algorithm 3.2, we obtain the conservative approximations of worst-case evaluation matrices of the respective parameterized scenarios as specified by (4.13).

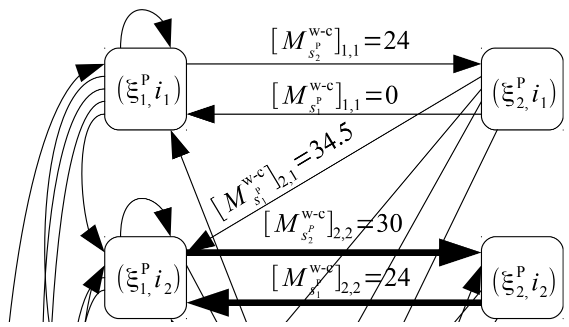

$[M^{\text{w-c}}_{s_2^P}]_{1,1}=24$
$[M^{\text{w-c}}_{s_1^P}]_{1,1}=0$
$[M^{\text{w-c}}_{s_1^P}]_{2,1}=34.5$
$[M^{\text{w-c}}_{s_2^P}]_{2,2}=30$
$[M^{\text{w-c}}_{s_1^P}]_{2,2}=24$
$(\xi^P_{1,} i_1)$
$(\xi^P_{2,} i_1)$
$(\xi^P_{1,} i_2)$
$(\xi^P_{2,} i_2)$

Fig. 4.7: Throughput graph of SDF-PFSM-SADFG of Fig. 4.3.

$$
\begin{aligned}
M^{\mathrm{w-c}\prime}_{s_1^{\mathrm{P}}} &= \begin{bmatrix} 24 & -\infty & -\infty & -\infty & -\infty & 24 \\ 34.5 & 24 & -\infty & -\infty & -\infty & 34.5 \\ 34 & -\infty & 24 & -\infty & -\infty & 34 \\ 42.5 & 32 & 32 & 24 & -\infty & 42.5 \\ 42.5 & 32 & 32 & 24 & -\infty & 42.5 \\ -\infty & -\infty & -\infty & -\infty & 0 & -\infty \end{bmatrix} \\
M^{\mathrm{w-c}\prime}_{s_2^{\mathrm{P}}} &= \begin{bmatrix} 0 & -\infty & -\infty & -\infty & -\infty & -\infty \\ -\infty & 30 & -\infty & -\infty & -\infty & 30 \\ -\infty & -\infty & 0 & -\infty & -\infty & -\infty \\ -\infty & -\infty & -\infty & 0 & -\infty & -\infty \\ -\infty & 30 & -\infty & -\infty & -\infty & 30 \\ -\infty & -\infty & -\infty & -\infty & 0 & -\infty \end{bmatrix}
\end{aligned}
\tag{4.13}
$$

Using the matrices of (4.13) we now construct the throughput graph which is partially displayed in Fig. 4.7. The critical cycle of the throughput graph is depicted using bold arrows. The critical cycle defines the MCM of the throughput graph which attains the value $(30 + 24)/2 = 27$ time units and therefore $Th'_{\mathsf{FP}} = 1/27$ iterations per time unit.

At this point one might argue, that given the relatively small cardinality of the scenario domains of (4.11) one could enumerate the domains and use techniques of FSM-SADF to obtain the worst-case throughput value. This claim holds for the running example, but in practice the respective domains can be vast and enumeration infeasible. Another might argue that instead of computing the worst-case evaluation matrices of parameterized scenarios via Algorithms 3.2 and 3.1, one could simply construct the worst-case SDFG of a parameterized scenario graph by taking the upper endpoints of default parameter intervals which would in turn define a worst-case FSM-SADF that can be analyzed. In response to this, we argue that such an approach (although straightforward) which disregards complex parameter dependencies defined by the parameterized scenario domain specifications might incur to much pessimism. For the running example, by merely taking $p = 10$, $q = 10$ and $u = 30$ with $a_1 = 7$, $a_2 = 4$, $a_3 = 5$ and $a_4 = 4$ we obtain an FSM-SADFG with the worst-case throughput value of 1/70 iterations which compared to the estimate obtained using our technique incurs too much unnecessary pessimism. Furthermore, in some case it may happen that due to the fact that rates may influence the temporal behavior of the structure in a "non-monotonic way", the result obtained using default parameter interval endpoints may even be an underestimation of the actual throughput which is an unacceptable side-effect in a real-time setting.

4.9.2 Worst-case latency

The remaining performance metric to be discussed is latency. As for throughput we adopt the definition of SDF-PDF (cf. Definition 3.5), i.e. FSM-SADF (cf. Definition 3 of [46]).

Definition 4.5 (Worst-case latency)**.** *Worst-case latency of an SDF-PFSM-SADFG F^{P} relative to a desired period $\rho \in \mathbb{R}$ is defined as the smallest vector $L_{\mathsf{F}^{\mathrm{P}}}$ such that for every possible instance sequence and its associated Max-plus timestamp vector sequence $\overline{\gamma} = \gamma(1), \gamma(2), \ldots$, for every $k \geq 0$,*

$$\gamma(k) \preceq k \cdot \rho + L_{\mathsf{F}^{\mathrm{P}}}. \tag{4.14}$$

Unlike with throughput where we were not interested in firing times of individual actors but only the rate at which the iterations of the graph are executed, to derive $L_{\mathsf{F}^{\mathrm{P}}}$ we do need to consider what time do individual actor firings take place or complete within an iteration. We need to consider all possible scenario sequences. To do this we construct the state space of scenario sequence executions using the conservative approximations of worst-case evaluation matrices of the parameterized scenarios obtained via Algorithm 3.2 and the scenario FSM. Using these matrices we are able to conservatively bound vectors $\gamma(k)$ of (4.5) and that is why we call this state space also the worst-case evaluation state space of the SDF-PFSM-SADFG. The latency derived from this state space is denoted with $L'_{\mathsf{F}^{\mathrm{P}}}$ and is a conservative estimate of $L_{\mathsf{F}^{\mathrm{P}}}$.

The space is constructed incrementally in a bread-first search manner from the parameterized scenario FSM. The state itself is defined as a tuple

$$(\Psi^{\mathrm{P}}(\xi^{\mathrm{P}}), \gamma^{\mathrm{w-c}\prime}, w),, \tag{4.15}$$

where $\xi^{\mathrm{P}} \in \Xi^{\mathrm{P}}$, $\gamma^{\mathrm{w-c}'}$ is a Max-plus timestamp vector which is used to initialize the next scenario execution and w is the state weight.

Let tuple

$$(\Psi^{\mathrm{P}}(\xi^{\mathrm{P}\sharp}), \gamma^{\mathrm{w-c}'\sharp}, w^{\sharp}) \tag{4.16}$$

define a state that is directly reachable from state (4.15). In that case,

$$\gamma^{\mathrm{w-c}'\sharp} = (M^{\mathrm{w-c}'}_{\Psi^{\mathrm{P}}(\xi^{\mathrm{P}'})} \otimes \gamma^{\mathrm{w-c}'})^{\mathrm{norm}} \tag{4.17}$$

and

$$w^{\sharp} = ||M^{\mathrm{w-c}'}_{\Psi^{\mathrm{P}}(\xi^{\mathrm{P}'})} \otimes \gamma^{\mathrm{w-c}'}||. \tag{4.18}$$

Continuation of the state-space construction will eventually result in revisiting an already existing state if the reachable part of the state space is finite. The exploration terminates, when there are no more new states. For any path of length k leading to state $(\Psi^{\mathrm{P}}(\xi^{\mathrm{P}}), \gamma^{\mathrm{w-c}'}, w)$, the actual $\gamma^{\mathrm{w-c}'}(k)$ of the associated parameterized scenario sequence is given by $T \otimes \gamma^{\mathrm{w-c}'}$ where T equals to the sum of the weights of the path states.

By setting $\rho = \frac{1}{Th'_{\mathsf{FP}}}$, we can conservatively bound the latency in a single traversal of state space by finding the smallest vector L'_{FP} such that $\gamma^{\mathrm{w-c}'}(k) \leq L'_{\mathsf{FP}} + \frac{k}{Th'_{\mathsf{FP}}}$. This equals to determining the maximal value of $\gamma^{\mathrm{w-c}'}(k) - \frac{k}{Th'_{\mathsf{FP}}}$ observed. The exploration needs to consider only acyclic paths in the state space as any cycle will not be faster than determined by the throughput. We demonstrate this for the running example in (4.19) over the state space path of Fig. 4.9.2.

$$\begin{aligned} L'_{\mathsf{FP}} = \bigoplus \{&[0,0,0,0,0,0]^T, [0,30,0,0,30,0]^T - 27, \\ &[24,54,35,62,62,30]^T - 54, [24,84,35,62,84,62]^T - 81, \\ &[86,108,97,116,116,84]^T - 108, \\ &[86,138,97,116,138,116]^T - 135\} = [0,3,0,8,8,0]^T \end{aligned} \tag{4.19}$$

A necessary condition to determine the latency in this way is that the reachable part of the state space is finite. For the example SDF-PFSM-SADFG of Fig. 4.3 with (4.13) this is indeed the case. However, in general, this may not be so. Therefore, we need to define at least a sufficient condition under which the reachable part of the state space is finite.

For FSM-SADF, paper [46] in Proposition 4.1 gives a practical condition under which the reachable part of the state space is finite. This result can be recasted in the context of SDF-PFSM-SADF.

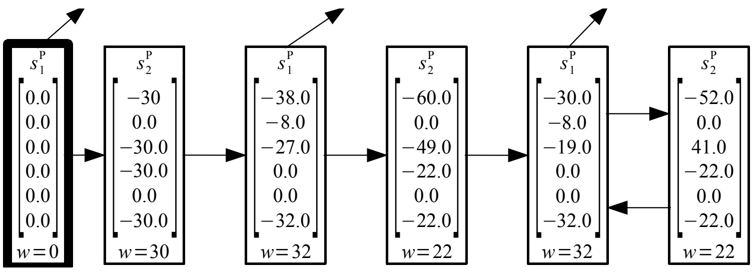


Fig. 4.8: State space of SDF-PFSM-SADFG of Fig. 4.3.

In particular, we say that the reachable part of the state space is finite if for every possible scenario sequence $\overline{s}^{\mathrm{P}} = s_1^{\mathrm{P}}, \ldots, s_p^{\mathrm{P}}$ allowed by the FSM and any $k > 0$ there is some $m > 0$ such that the matrix

$$M = \bigotimes_{k \le l \le m} M_{s_l^{\mathrm{P}}}^{\mathrm{w-c\prime}} \tag{4.20}$$

contains no entries $-\infty$.

We argue that this condition is too restricting. We show this by a simple example. Imagine a SDF-PFSM-SADFG with only one scenario defined by an actor with an self-edge hosting 2 initial tokens. The Max-plus representation of such a scenario is a 2 by 2 Max-plus matrix with $-\infty$ on the diagonal. Therefore, any power of this matrix has entries $-\infty$ on the diagonal too and this matrix cannot satisfy (4.20) although the state-space of such a specification is finite (we leave it to the interested reader as a small exercise to generate the state space of this simple structure). We give a new and less restricting condition for fitness of the reachable part of the state space but only after we define the notion of matrix irreducibility in Max-plus.

A matrix $M \in \mathbb{R}_{\max}^{n \times n}$ is called irreducible if its communication graph $\mathcal{G}(M)$ is strongly connected [61].

We now give a sufficient condition for finiteness of the reachable part of the worst-case evaluation state space of an SDF-PFSM-SADFG.

Proposition 4.1. *Let $\mathsf{F}^{\mathrm{P}} = (\mathcal{S}^{\mathrm{P}}, F^{\mathrm{P}})$ be an SDF-PFSM-SADFG. Let $C = \{c_i\}$ by the set of all simple cycles of F^{P}. If the matrix $M_{c_i} = \bigoplus_{n=1}^{length(c_i)} M_{\Psi^{\mathrm{P}}(c_i(n))}^{\mathrm{w-c\prime}}$ is irreducible for every $c_i \in C$ then the reachable part of the state space is finite.*

Proof. Notice that any scenario sequence can be formed by concatenation of cycles of the scenario FSM [98]. Now, irreducibility of a Max-plus matrix

M_{c_i} implies that its eigenvalue is unique [61]. The eigenvalue on the other hand specifies the asymptotic growth rate of a timestamp vector produced by this matrix. Therefore, entries of the normalized timestamp vectors generated by M can only take values from a bounded range because the growth rate is the same for all the entries. Consequently, in consideration of a scenario sequence as a repetitive pattern consisting only of one FSM cycle, there will be a finite number of timestamp vectors within the sequence which implies the finiteness of the state space over one cycle. The argument straightforwardly carries over to concatenations of different cycles as they are all individually bounded, i.e. no token timestamp can diverge. $\square$

If we verify the example with an actor with a self-edge hosting two initial tokens where the finiteness criteria of [46] fails to give an answer against Proposition 4.1, we see that Proposition 4.1 gives a positive answer to the question, which is the correct one.

4.10 Evaluation

In this section, we demonstrate the application of our parameterized scenario modeling and analysis techniques to a realistic case study from the multimedia domain. We consider the case of a VC-1 video decoder used in a region of interest (ROI) coding scheme. Thereafter, we discuss the tightness of the performance bound and technical aspects of the analysis.

4.10.1 VC-1 decoder

The experimental setup is somewhat similar to that of Section 3.8.1. In particular, we will be using the *Foreman* video sequence (cf. Fig. 3.13b) as the input to the decoder with the difference that we will be decoding not only the "ROI slice" that captures the foreman's face but also the “background slice”. The slices are transmitted together and the decoder processes them in an interleaved fashion to be able to reconstruct the entire picture, i.e. frame.

In VC-1 coding, three different types of slices are supported: I, i and Ii slices. In an I-slice all macroblocks are encoded in the *Intra* mode. In an i-slice all macroblocks are encoded in the *Inter* mode. In an Ii-slice all macroblocks are both *Intra* and *Inter* coded. The types of slices naturally represent three modes of operation of the decoder shown in Fig. 4.9 adopted from [13]. Each mode is represented by a different SDF-PDFG according to our parameterized scenario modeling technique. Each SDF-PDFG iteration corresponds to decoding of one slice.

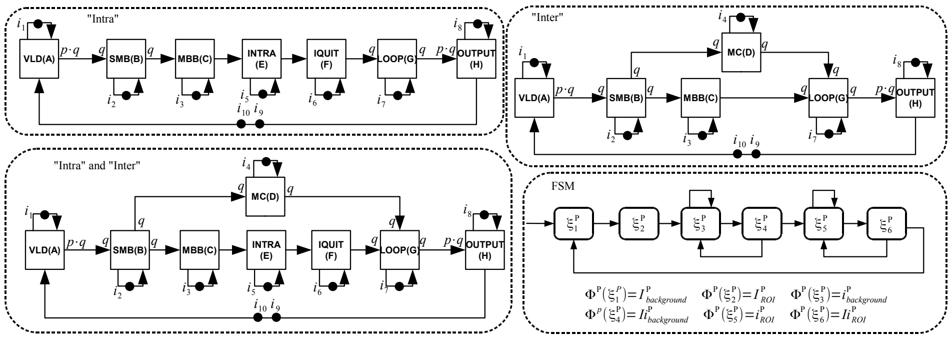

"Intra"
VLD(A)
SMB(B)
MBB(C)
INTRA (E)
IQUIT (F)
LOOP(G)
OUTPUT (H)
"Inter"
MC(D)
"Intra" and "Inter"
FSM
$\Phi^P(\xi_1^P)=I^P_{background}$
$\Phi^P(\xi_2^P)=I^P_{ROI}$
$\Phi^P(\xi_3^P)=i^P_{background}$
$\Phi^p(\xi_4^P)=Ii^P_{background}$
$\Phi^P(\xi_5^P)=i^P_{ROI}$
$\Phi^P(\xi_6^P)=Ii^P_{ROI}$

Fig. 4.9: VC-1 decoder captured in SDF-PFSM-SADF.

The functions of particular actors are already described in Section 3.8.1.

In contrast to the VC-1 decoder of Section 3.8.1, our decoder here also decodes the background slices. Therefore, it must not only differentiate slices based on the encoding scheme used, but also based on their affiliation to either ROI or background.

Therefore, with slices being either ROI or background slices each of which can be encoded in 3 different ways, the dynamic behavior of the decoder can be captured using 6 scenarios: $I^{\mathrm{P}}_{\mathrm{ROI}}$, $i^{\mathrm{P}}_{\mathrm{ROI}}$, $Ii^{\mathrm{P}}_{\mathrm{ROI}}$, $I^{\mathrm{P}}_{\mathrm{background}}$, $i^{\mathrm{P}}_{\mathrm{background}}$ and $Ii^{\mathrm{P}}_{\mathrm{background}}$. E.g., scenario $I^{\mathrm{P}}_{\mathrm{ROI}}$ models the decoding of an I slice capturing the ROI, i.e. the foreman's face and captures as many behaviors as there are possible *ROI* slice sizes. The slice size expressed in the number of enclosed macrobolocks (parameterized rate p) basically depends on the given frame resolution, the displacement of the foreman's face from the capturing device (camera) and the ovality of the foreman's face. These relationships are encoded in the definition of the scenario domain.

For $I^{\mathrm{P}}_{\mathrm{ROI}}$, we define $X_{I^{\mathrm{P}}_{\mathrm{ROI}}}$ by abstracting ROI into an ellipse of known characteristics (cf. (4.21)).

$$
\begin{aligned}
X_{I^{\mathrm{P}}_{\mathrm{ROI}}} =\{ & p = (2 \cdot \xi_M \cdot 2 \cdot \xi_m)/(16 \cdot 16), p \in [1, P], && (4.21a)\\
& q \in [1, 16] && (4.21b)\\
& p' \geq \mu \cdot P, p + p' \leq P && (4.21c)\\
& o^2 = 4 \cdot \pi^2(\xi_M^2 + \xi_m^2), o \geq O && (4.21d)\\
& \epsilon^2 \cdot \xi_M^2 = \xi_M^2 - \xi_m^2, \epsilon = E, 2 \cdot \xi_M \leq w, 2 \cdot \xi_m \leq h && (4.21e)\\
& a = a_{\mathrm{ref}}, b = b_{\mathrm{ref}}, c = c_{\mathrm{ref}}, d = d_{\mathrm{ref}}, && (4.21f)\\
& e = e_{\mathrm{ref}}, f = f_{\mathrm{ref}}, g = q_{\mathrm{ref}}, h = h_{\mathrm{ref}}\} && (4.21g)
\end{aligned}
$$

Detailed description of parameters involved can be found in Section 3.8.1. The remaining 5 scenario domain definitions take a similar form and we do not list them here.

Every scenario is fully determined by the scenario SDF-PDFG and the respective domain. Of course, different scenarios may have the same graphs or domains. In our case study, each of the SDF-PDFGs is shared by 2 scenarios.

As mentioned, in contrast to the VC-1 decoder case study of Section 3.8.1, we here decode both the ROI and the background slices. But this is a difference that concerns only functionality and has no bearing on the modeling itself. The second and crucial difference is that here by using SDF-PFSM-SADFG we are able to express intricate control requirements that an application may have. This is not possible by using SDF-PDF of Chapter 3. In particular, in VC-1 or any other block-based coding scheme, slices are not sequenced arbitrarily but the next slice type depends on the previous one. The sequencing patterns are specified by *group of pictures* (GoP) structures that always start with I-slices. In such a setting, the control requirements of our decoder are specified by the FSM of Fig. 4.9. In particular, slices are sequenced as follows. First, I slices of both ROI and the background are decoded. Thereafter, a number of i and Ii slices forming i and Ii pictures are decoded. This is first done for ROI and thereafter for the background. In reality, the number of i frames following I and Ii frames is bounded by the Group of Pictures length. For simplicity, we approximate this conservatively by allowing an arbitrary long sequence of i slices that is always followed by one Ii slice for both ROIs. Finally, the FSM revisits the initial state.

In the exercise, we assume SDTV input format with signal type 480i 16:9 and resolution 720x480 pixels. Thus, $w = 720$, $h = 240$ and $P = 1620$. Furthermore, $O = 700$, $E = 0.6$ and $\mu = 30$. For these values using our performance analysis technique presented in Section 4.9 we obtain a conservative throughput estimate of $1.78516 \cdot 10^{-7}$ slices per cycle.

For comparison, if we were to build an FSM-SADF that is a conservative model of SDF-PFSM-SADF model of Fig. 4.9 by using the upper endpoints of default domain parameter intervals in each scenario we obtain a throughput estimate of $1.44252 \cdot 10^{-7}$. Therefore, compared to the capabilities of FSM-SADF our contribution is twofold. First, PFSM-SADF as a modeling concept can express fine-grained reconfiguration phenomena in dynamic systems as well as capture their intricate control requirements. Second, by accounting for complex dependencies between parameters the worst-case performance analysis techniques of SDF-PFSM-SADF tighten the results of

FSM-SADF. In some cases even, if we use domain parameter upper interval endpoints with FSM-SADF analysis, we may even underestimate the metric of interest because we cannot account the "non-monotonic" effect of rates to temporal behavior of the graph. By this we mean that increasing the value of a rate in the graph does not necessarily need to lead to a later completion of the graph's iteration. Quite the contrary, this may lead to an earlier completion of an iteration. For the case study the comparison shows that our results tightens the FSM-SADF throughput estimate by 19.19% while remaining on the safe side, i.e. conservative.

4.10.2 Tightness of performance bound and technical aspects of the analysis

In SDF-PFSM-SADF every scenario is represented by an SDF-PDFG. Each of these is analyzed in isolation using the techniques presented in Chapter 3 and the results are composed in the final SDF-PFSM-SADF analysis. Therefore, the conclusions of the discussion on tightness of performance bounds for SDF-PDF presented in Section 3.8.2 simply carry over into the SDF-PFSM-SADF context. More precisely, we are as conservative as the worst-case evaluation matrices of scenarios are. For growing scenario repetition vector entries the relative estimation error shrinks, i.e. it asymptotically reaches 0 (cf. Subsection 3.5.4).

Because the worst-case evaluation matrices of scenarios are obtained from parameterized matrices extracted by manually executing Algorithm 3.1 for each parameterized scenario, so is also the analysis partly manual.

Once the analysis of Chapter 3 has been fully automated, the additional effort involved to fully automate the analysis of this chapter is negligible. In particular, in the SDF^3 tool, the implementation would entail the development of a simple routine that would feed the worst-case scenario evaluation matrices and the scenario FSM to the existing FSM-SADF Max-plus automata analysis algorithms.

4.11 Summary

In this chapter, we introduced the PFSM-SADF formalism that combines FSM with parameterized dataflow as the underlying concurrency model. An FSM has the natural capability of expressing intricate control logic governing the application behavior, while parameterized dataflow is very fit for expressing fine-grained data-dependent application dynamics. In addition, the domain concept entails a modeling flexibility which allows to represent the dependencies between parameters. These dependencies may arise from a variety of sources, such as an expression in a design environment that expresses the value of parameter in terms of another. For the SDF-based specialization of PFSM-SADF we developed corresponding worst-case performance analysis techniques that are able to provide tighter but still conservative performance bounds for systems exposing fine-grained dynamism combined with control than the existing techniques do.

Chapter 5

Parametric throughput analysis for SDF-based parameterized scenario-aware dataflow graphs

SDF-based specialization of PFSM-SADF referred to as SDF-based PFSM-SADF (SDF-PFSM-SADF) MoC is an efficient tool for capturing the fine-grained data-dependent dynamics of modern streaming application with intricate control requirements (cf. Chapter 4). The model comes equipped with the worst-case throughput (and latency) analysis technique developed in correspondence with its operational semantics where parameter reconfigurations occur between activations of different parameterized scenarios as well as between consecutive activations of the same parameterized scenario.

However, not all applications will always utilize the full operational semantics of SDF-PFSM-SADF. In particular, there may be situations where parameters can be considered fixed across scenarios, i.e. they do not change at all or change infrequently. In this chapter, we consider the throughput analysis of SDF-PFSM-SADF in the presence of such, so called static parameters. In particular, we propose a parametric throughput analysis technique that computes the conservative estimate of the graph throughput as a simple function of SDF-PFSM-SADFG parameters, i.e. rates and actor firing delays. These functions can then be evaluated in a negligible amount of time for a given set of parameter values. We base our technique on the theory of Max-plus automata. We evaluate our approach on an artificial case study.

The contribution of this chapter has not been (yet) published.

5.1 Introduction

The concept of FSM-based parameterized scenario aware-dataflow (PFSM-SADF) introduced in Chapter 4 enables succinct modeling of streaming applications exposing both fine-grained data-dependent dynamics and intricate control requirements. The execution of an application captured in PFSM-SADF is interpreted as an execution of a sequence of parameterized scenarios each of which is modeled by a parameterized dataflow graph. The (parameterized) scenario occurrence patterns are specified by an nondeterministic FSM that abstracts application control requirements.

A particularly important specialization of PFSM-SADF is obtained when its paramterization concept is applied to SDF [70] as SDF is arguably the most stable and mature dataflow MoC [16]. We refer to this specialization as SDF-based parameterized dataflow (SDF-PFSM-SADF) (cf. Chapter 4). SDF-PFSM-SADF comes equipped with worst-case throughput (and latency) analysis techniques based on the Max-plus algebraic semantics of SDF [44] and the theory of Max-plus automata [40] that operate in full compliance with the operational semantics of PFSM-SADF, i.e. SDF-PFSM-SADF. In particular, over sequences of parameterized scenarios, parameters are allowed to change not only between different parameterized scenarios but also between consecutive activations of the same parameterized scenario. We call such parameters *dynamic* parameters. Recall that the execution of a parameterized scenario interprets as an execution of an arbitrarily chosen instance of that parameterized scenario. As mentioned, instances are determined by configurations, while configurations are constrained to belong to a subset of the parameter space referred to as the parameterized scenario domain.

However, in practice, many applications do not utilize the full expressiveness of PFSM-SADF, i.e. SDF-PFSM-SADF. In particular, we consider applications captured in SDF-PFSM-SADF for which, of course, parameter values are *a priori* unknown, but once set they remain fixed or change infrequently. We call such parameters *static* parameters. In context of such restricted semantics of SDF-PFSM-SADF we consider the problem of parametric throughput analysis. In particular, given an SDF-PFSM-SADFG, our objective is to express the throughput of the graph or possibly its conservative estimate as a simple function of graph parameters, i.e. graph rates and actor firing delays. This is possible because the graph parameters are deemed static. In case they are not, such a formulation is, of course, not possible. In particular, we then must resort to the techniques of Section 4.9.

We build on the on the work of [33], that considers the throughput computation problem for a flavor of FSM-SADF with parameterized actor firing delays expressed as linear combinations of parameters. Note that a SDF-PFSM-SADF with static parameters is equivalent to a FSM-SADF with parameterized rates and parameterized actor firing delays which justifies our starting point choice of [33]. Therefore, by extending the analysis to SDF-PFSM-SADF, we generalize the initial result of [33] as we are, in the parlance of FSM-SADF, able to treat FSM-SADF specifications with parameterized rates (parameterized rates are inherent to SDF-PFSM-SADF) and parameterized firing delays where these are not restricted to linear combinations of parameters but to polynomial functions of parameters. To summarize, with our analysis compared to that of [33] the gain is twofold.

First, we can consider parameterized rates. Rates have a dramatic impact on the dataflow and consequently temporal behavior of a dataflow graph. Thus, the information on the dependence of graph throughput on graph rates is of great significance to the designer. E.g., in scientific computing, the number of iterations of some numerical algorithm can be represented as a graph rate in the corresponding dataflow model. The more iterations the algorithm performs, the more precise is the result of the computation. However, in a throughput constrained system, the designer may need to determine what is the minimal number of such iterations that both the throughput constraint as well as any result precision constraints are satisfied.

Second, we can consider firing delays given as polynomial functions of parameters, while the works of [33] and [32] allow only linear functions. In particular, in the DVFS context of [32] parameters represent scale factors where the actor firing delay is represented as a product of some constant worst-case nominal firing delay and the scale factor that accounts for the processor DVFS setting. This is a linear expression. By allowing a less limiting pattern, i.e. polynomials, we can express firing delays as polynomial functions of input parameters to the module the actor implements and so refine the representation. We motivate by referring to matrix multiplication and insertion sort case studies of [4] the worst-case execution time of which is bounded by polynomial functions. This way we avoid the need for a constant and pessimistic worst-case nominal firing delay that accounts for the "worst-case input" and we can consequently provide a tighter throughput bound which results in the increase of the optimization margin at the designer's disposal.

We demonstrate our technique using an artificial case study that also serves as the running example of this chapter.

5.2 Motivational example

We motivate by showing the benefits our end result could have in the problem scope of optimal voltage scaling in power-limited systems with real-time constraints.

Minimizing energy consumption in low-power systems has become a critical design considerations, especially with the proliferation of portable and mobile embedded systems. Energy consumption E in CMOS processors depends linearly on the operating frequency f and the quadratically on the supply voltage V_{DD} as follows [24][32][78][77]

$$E = C_1 \cdot V_{\mathrm{DD}}^2. \tag{5.1}$$

The frequency f is given by

$$f = \frac{V_{\mathrm{DD}} - V_{\mathrm{t}}}{C_2 \cdot V_{\mathrm{DD}}} = \frac{1}{C_2} \cdot (1 - \frac{V_{\mathrm{t}}}{V_{\mathrm{DD}}}) \tag{5.2}$$

where C_1 and C_2 are some constants dependent on hardware characteristics and V_t is the threshold voltage so that $V_{\mathrm{DD}} \geq V_{\mathrm{t}}$. Equations (5.1) and (5.2) reveal that controlling the voltage and the operating frequency provide means to regulate the energy consumption leading to DVFS techniques [25] that are commonly used to develop low power and energy systems. Furthermore, (5.2) reveals that while lowering the operating voltage, the maximal possible operating frequency also reduces. Therefore, by reducing voltage we reduce energy but at the expense of longer delays which adversely affects performance [77]. In real-time systems this may have undesirable consequences.

Therefore the crucial step in the design process of power-limited real-time systems is to determine the DVFS settings of platform processors so that all real-time constraints are still met.

Consider a dynamic application with a throughput constraint captured by the SDF-PFSM-SADFG with static parameters shown in Fig. 5.1. The graph defines two scenarios and an FSM defining scenario ordering. The internal structure of scenarios is irrelevant here and therefore they are abstracted into hierarchical actors. We assume that all actors are mapped to separate processors supporting continuous DVFS. Per scenario, computational requirements of a particular actor are expressed in processor clock

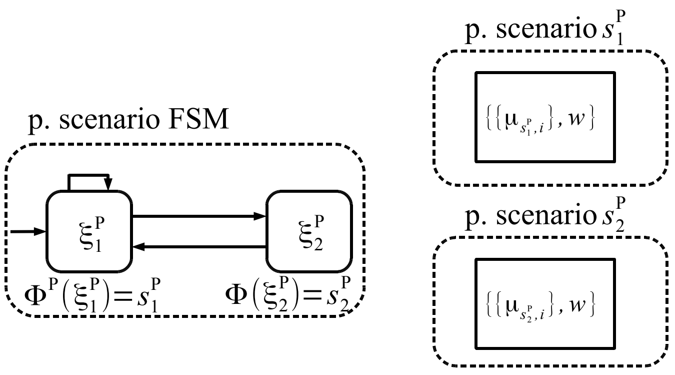


Fig. 5.1: Motivational example.

cycles (parameter sets $\{\mu_{s_1^P,i}\}$ and $\{\mu_{s_2^P,i}\}$) that are themselves (nonlinear) functions of some design environment parameters $\kappa_1, \ldots, \kappa_N$ and input signal parameters $\kappa_{N+1}, \ldots, \kappa_{N+M}$. More formally,

$$\mu_{s_j^P,i} = g_{s_j^P,i}(\kappa_1, \ldots, \kappa_{N+M}). \tag{5.3}$$

Furthermore, we assume that the considered application is an numerical algorithm the precision of which can be controlled by varying the number of times particular actors run. The variation is captured using the parameterized rate w present in both scenarios

The design task is as follows. Given a precision constraint (equivalently, the value w), the values of $\kappa_1, \ldots, \kappa_{N+M}$ and the values of $\{\mu_{s_j^P,i}\}$ find the voltage setting for each of the processors so that the energy consumption is minimized.

Now, if actor indexed by i operating in s_j^P runs $\mu_{s_j^P,i}$ cycles on a processor running at frequency $f_{s_j^P,i}$, then the firing delay of that actor equals

$$d_{s_j^P,i} = \frac{\mu_{s_j^P,i}}{f_{s_j^P,i}}. \tag{5.4}$$

By combining (5.1)-(5.2) with (5.4), we can express the energy consumption as a result of one firing of actor indexed by i in scenario s_j^P as a function of processor frequency as follows

$$E_{s_j^P,i} = C_1 \cdot \left(\frac{V_t^2}{1 - C_2 \cdot f_{s_j^P,i}} \right)^2. \tag{5.5}$$

Equation (5.5) reveals that lowering the operating frequency allows lowering of voltage which results in reduced energy consumption.

The question is now what the lowest possible frequencies are so that the application throughput and precision requirements are met.

Assume that the throughput of the application graph can be expressed as a function of actor firing delays $\{d_{s_j^{\mathrm{P}},i}\}$ and parameter w as follows

$$Th_{\mathsf{FP}} = h(d_{s_1^{\mathrm{P}},1} \ldots, d_{s_2^{\mathrm{P}},R}, w). \tag{5.6}$$

In this case determining the processor voltage settings that minimize the energy consumption equals to solving the optimization problem where the objective is to minimize the processor frequencies $\{f_{s_j^{\mathrm{P}},i}\}$ while assuring that the application throughput and precision constraint is met. The problem is formulated as follows

$$\begin{aligned} \text{minimize} \quad & \{f_{s_j^{\mathrm{P}},i}\} \\ \text{subject to} \quad & Th_{\mathsf{FP}} = h(d_{s_1^{\mathrm{P}},1} \ldots, d_{s_2^{\mathrm{P}},R}, w) \geq T \\ & w \geq W \\ & d_{s_j^{\mathrm{P}},i} = \frac{\mu_{s_1^{\mathrm{P}},i}}{f_{s_1^{\mathrm{P}},i}} \\ & \mu_{s_j^{\mathrm{P}},i} = g_{s_j^{\mathrm{P}},i}(\kappa_1, \ldots, \kappa_{N+M}). \end{aligned} \tag{5.7}$$

The problem of (5.7) is a typical (nonlinear) optimization problem with T being the throughput constraint and W being the precision constraint. Here, we assumed that the problem of (5.7) is solved at design-time. Nevertheless, one can easily imagine heuristics being developed to address flavors (5.7) at run-time too. The missing part in (5.7) is the formulation of the graph throughput as a function of parameters that can be quickly evaluated. To this formulation we dedicate the remainder of this chapter.

5.3 Related work

To the best of our knowledge the work on parametric throughput analysis[1] for dataflow MoCs is relatively scarce [2]. Paper [48] introduces a parametric throughput analysis technique for SDF where (only) actor firing delays can be parametric. The analysis yields a set of simple expression of these

[1]Within this chapter, when we use the term “parametric analysis” we mean analysis considering static parameters.

[2]Related work on analysis considering dynamic parameters is outlined in Section 4.3.

parameters, that when evaluated for a particular set of parameter values, give the throughput value of the input SDF specification. The core of the technique is a divide & conquer algorithm used in conjunction with state-space exploration of the graph [49]. However, SDF is a static dataflow MoC and cannot be used to efficiently capture dynamic streaming applications. Therefore, throughput analysis of dynamic applications using a SDF variant that conservatively captures application's temporal behaviour will incur a significant amount of pessimism. The work of [33] generalizes the analysis of [48] by considering the parametric throughput analysis problem for FSM-SADF as a generalization of SDF able to capture dynamic streaming applications. It adopts the divide & conquer strategy of [48] but, in addition, with the purpose of improving scalability, recasts it in the context of the compositional Max-plus-based analysis of SDF/FSM-SADF.

However, both techniques of [48] and [33] can only consider structures with parameterized actor firing delays, i.e. parameterized rates cannot be accounted for which significantly hampers the application domain of the technique. Furthermore, actor firing delays are limited to being constants or linear combinations of parameters.

The work of [15] reports on work in progress on throughput analysis of BPDF. Here, rates can be parametric. However, the analysis is applicable to only acyclic BPDF specifications. Furthermore, the analysis itself (not BPDF as a MoC) has limited support for dynamic behavior, i.e. no graph topology reconfiguration is allowed from one graph iteration to the other (in SDF-PFSM-SADF this is naturally supported by the scenario concept itself).

5.4 The analysis model

In this section we formally define our analysis model. We consider SDF-PFSM-SADF. In SDF-PFSM-SADF, every scenario is represented by an SDF-PDFG, while, as usual, admissible scenarios sequences are defined by the accompanying scenario FSM.

Furthermore, we allow actor firing delays to be represented as polynomial functions of parameters.

Fig. 5.2 shows such an SDF-PFSM-SADFG. The structure in the figure is composed out of two scenarios, namely s_1^{P} and s_2^{P}. Scenario s_1^{P} SDF-PDFG has two rates that are parameterized with parameter $x_1 \in \mathbb{N}_{>0}$ and two parameterized firing delays, namely those of actors A_1 and A_2, parameterized with parameter x_1 too.

In scenario s_2^{P} one firing delay is parameterized, namely that of actor

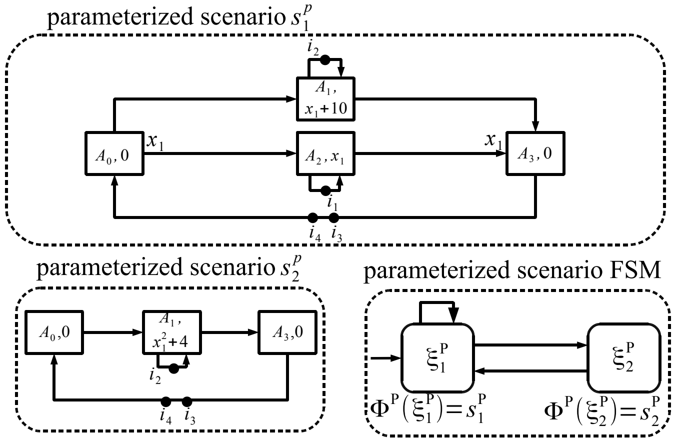


Fig. 5.2: Example SDF-PFSM-SADFG.

A_1 as a polynomial function of parameter x_1, denoted $x_1^2 + 4$.

Note that although this is typically not the case, for simplicity and without the loss of generality, here we used one parameter ($x_1 \in \mathbb{N}_{>0}$) to parameterize both rates and actor firing delays in both scenarios.

The scenario FSM is defined by two states, namely ξ_1^{P} and ξ_2^{P} and transitions between them. State ξ_1^{P} corresponds to s_1^{P}, while state ξ_2^{P} corresponds to s_2^{P}.

5.4.1 The domain of the analysis model

An SDF-PFSM-SADFG evolves in iterations of its parameterized scenario instances. Each scenario instance is an SDFG obtained by applying an arbitrary configuration from the particular domain to the scenario SDF-PDFG. Therefore, the concept of domain is crucial and in this subsection we formally define requirements on the domain of our analysis model, i.e. SDF-PFSM-SADF.

In the context of full operational semantics of SDF-PFSM-SADFG presented in Section 4.7, it was convenient to define domains per scenario. However, in the context of this work (as it will become clearer later) it is more convenient to treat the model through the notion of a unified (composite) SDF-PFSM-SADFG domain

$$X_{\mathsf{F}^{\mathrm{P}}} = X_{s_1^{\mathrm{P}}} \times \ldots \times X_{s_K^{\mathrm{P}}} \tag{5.8}$$

for an SDF-PFSM-SADFG with K parameterized scenarios.

For X_{FP}, we require that it is defined in terms of some polynomial functions, where a configuration $\mathbf{x} \in X_{\mathsf{FP}}$ of the SDF-PFSM-SADFG is denoted as follows:

$$\mathbf{x} = (x_1, \ldots, x_n). \tag{5.9}$$

Polynomials are a sound choice as approximation theory is centered on them, i.e. various complex functions can be approximated by polynomials of high degree [89]. Except that, we do not put any convexity or generalized convexity restrictions on these functions, but we do require that the feasible region is compact. A mathematical formulation (borrowed from [94]) is as follows:

$$X_{\mathsf{FP}} = Z \cap \Omega \tag{5.10}$$

where

$$\begin{aligned} Z = \{\mathbf{x} :& \varphi_r(\mathbf{x}) \geq \beta_r \text{ for } r = 1, \ldots, R_1, \\ & \varphi_r(\mathbf{x}) = \beta_r \text{ for } r = R_1 + 1, \ldots, R\}, \end{aligned} \tag{5.11}$$

and

$$\Omega = \{\mathbf{x} : 0 \leq l_j \leq x_j \leq u_j < \infty \text{ for } j = 1, \ldots, n\}, \tag{5.12}$$

and where

$$\varphi_r(\mathbf{x}) \equiv \sum_{t \in T_r} \alpha_{rt} \left[\prod_{j \in J_{rt}} x_j \right] \text{ for } r = 0, 1, \ldots, R. \tag{5.13}$$

In (5.13), T_r is an index set for the terms defining $\varphi_r(\cdot)$ and α_{rt} are real coefficients for the polynomial terms $(\prod_{j \in J_{rt}} x_j)$, $t \in T_r$, $r = 0, 1, \ldots, R$. Note that indices can repeat within each set J_{rt}. E.g., if $J_{rt} = \{1, 1, 2, 3\}$ then the corresponding polynomial term is $x_1^2 x_2 x_3$.

Simply put, X_{FP} is defined using a set of polynomial functions forming polynomial inequality and equality constraints of (5.11) with an attached set of bounding or box constraints (5.12).

We exemplify using the running example SDF-PFSM-SADFG of Fig. 5.2. In this case, parameter x_1 defines the configuration $\mathbf{x} = (x_1)$. The domain X_{FP} of the SDF-PFSM-SADFG is given as follows:

$$X_{\mathsf{FP}} = \{1 \leq x_1 \leq 5\}. \tag{5.14}$$

The domain specification of (5.14) contains (but not limited to) only one box constraint. Box constraints give lower and upper bounds of all domain parameters. Any additional equality and inequality constraints not present here would serve the purpose of nonlinearly shaping the Ω of (5.12). In (5.14), $x_1 \in \mathbb{N}_{>0}$ as x_1 is used to parameterize rates too, but for the analysis to follow we will use the continuous interval of (5.14) as it by default includes the feasible values of x_1, i.e. $1, 2, 3, 4$ and 5.

5.4.2 Operational semantics of the analysis model

Here we define the operational semantics of our analysis model. By definition, SDF-PFSM-SADF as a specialization of PFSM-SADF evolves in iterations of its scenario SDF-PDFG instances. The sequencing of scenarios is constrained by a sublanguage L of $\mathcal{S}^{\mathrm{P}^*}$ that is (in the parlance of automata theory) recognized by the scenario FSM. The execution of a scenario entails an execution of an arbitrary SDF-PDFG instance of that scenario where the set of instances is constrained by the SDF-PFSM-SADF domain definition.

Consider a parameterized scenario sequence $\overline{s}^{\mathrm{P}} \in \mathcal{S}^{\mathrm{P}^*} \cap L$ of length k

$$\overline{s}^{\mathrm{P}} = s_1^{\mathrm{P}}, \ldots, s_k^{\mathrm{P}}. \tag{5.15}$$

Recall that in consideration of the operational semantics of SDF-PFSM-SADF, (5.15) can be written as (4.6). Now using the concept of (composite) SDF-PFSM-SADFG configuration $\mathbf{x} \in X_{\mathsf{F}^{\mathrm{P}}}$, we can re-write (4.6) as follows

$$\overline{s}^{\mathrm{P}} = (s_1^{\mathrm{P}}, \mathbf{x}_{1,1} \mid \ldots \mid \mathbf{x}_{1,|X_{\mathsf{F}^{\mathrm{P}}}|}), \ldots, (s_k^{\mathrm{P}}, \mathbf{x}_{k,1} \mid \ldots \mid \mathbf{x}_{k,|X_{\mathsf{F}^{\mathrm{P}}}|}), \tag{5.16}$$

so that

$$\pi_{s_i^{\mathrm{P}}}(\mathbf{x}_{i,j}) \in X_{s_i^{\mathrm{P}}} \text{ for all } i = 1, \ldots, k \text{ and } j = 1, \ldots, |X_{\mathsf{F}^{\mathrm{P}}}|\,. \tag{5.17}$$

In (5.17), $\pi_{s_k^{\mathrm{P}}} : X_{\mathsf{F}^{\mathrm{P}}} \to X_{s_k^{\mathrm{P}}}$ is the scenario s_k^{P} projection function. This function, given a SDF-PFSM-SADFG configuration returns the particular scenario configuration, i.e. values of the composite domain parameters relevant for scenario s_k^{P}. Recall that parameterized scenarios do not have to share parameters. Therefore, it follows from the operational semantics of PFSM-SADF/SDF-PFSM-SADF that given two scenario sequences $\overline{s}_1^{\mathrm{P}}$ and $\overline{s}_2^{\mathrm{P}}$ of the same length k, such that

$$\pi_l(\overline{s}_1^{\mathrm{P}}(i))) = \pi_l(\overline{s}_2^{\mathrm{P}}(i))) \text{ for all } i = 1, \ldots, k, \tag{5.18}$$

it does not necessarily hold that

$$\pi_{\pi_l(\overline{s}_1^{\mathrm{P}}(i)))}(\pi_r(\overline{s}_1^{\mathrm{P}}(i)))) = \pi_{\pi_l(\overline{s}_2^{\mathrm{P}}(i)))}(\pi_r(\overline{s}_2^{\mathrm{P}}(i)))) \text{ for all } i = 1, \ldots, k. \tag{5.19}$$

This is a straightforward consequence of the intra-scenario nondeterminism, i.e. a parameterized scenario execution entails the execution of any of its SDF-PDFG instances. Worst-case performance analysis under the full operational semantics of SDF-PFSM-SADF has been addressed by Section 4.9.

In this article we consider a restricted operational semantics, where scenario parameters values are *a priori* unknown, i.e. they are determined at run-time, but once known they remain fixed or change infrequently.

Mathematically speaking, we require that for all scenario sequences $\overline{s}^{\mathrm{P}}_m, \overline{s}^{\mathrm{P}}_n \in \mathcal{S}^{\mathrm{P}^*} \cap L$ such that $k = \mathit{length}(\overline{s}^{\mathrm{P}}_m) = \mathit{length}(\overline{s}^{\mathrm{P}}_n)$,

$$\begin{aligned} &\text{if } \pi_l(\overline{s}^{\mathrm{P}}_m(i))) = \pi_l(\overline{s}^{\mathrm{P}}_n(i))) \text{ for all } i = 1, \ldots, k, \\ &\qquad \Rightarrow \pi_{\pi_l(\overline{s}^{\mathrm{P}}_m(i)))}(\pi_r(\overline{s}^{\mathrm{P}}_m(i)))) = \pi_{\pi_l(\overline{s}^{\mathrm{P}}_n(i)))}(\pi_r(\overline{s}^{\mathrm{P}}_n(i)))) \\ &\qquad \text{for all } i = 1, \ldots, k. \end{aligned} \tag{5.20}$$

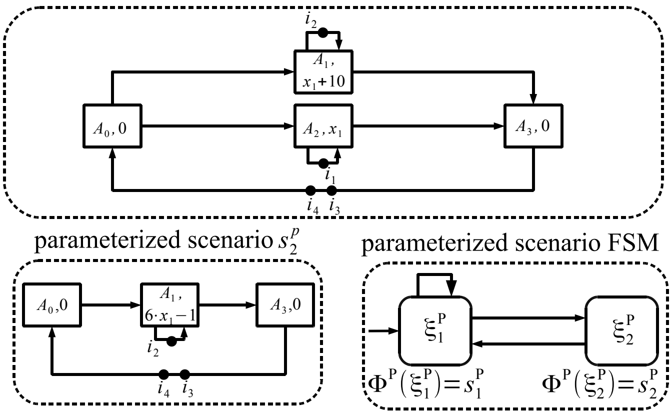
With (5.20), an SDF-PFSM-SADFG reduces to an FSM-SADFG with both parameterized rates and actor firing delays. For this case, we show how throughput of the specification can be expressed as a simple function of specification parameters by generalizing the result of [33] for FSM-SADF graphs including both parameterized rates and parameterized firing delays expressed as polynomial functions of SDF-PFSM-SADF domain parameters.

5.5 Parametric throughput analysis

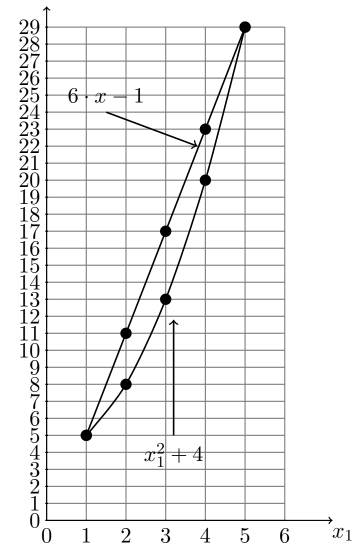
5.5.1 Baseline matter

The approach of [33] considers FSM-SADFGs with parameterized actor firing delays expressed as linear combinations of parameters originating from a parameter space (graph domain in the parlance of our work) constrained to a convex polytope.

To illustrate, consider our running example graph of Fig. 5.2 but modified in a manner that parameterized rates have been set to 1 and the firing delays expressed as polynomial functions have been upper-bounded by linear functions. In particular, the firing delay $x_1^2 + 4$ of A_1 in s^{P}_2 had been (conservatively) upper bounded on $X_{\mathsf{F}^{\mathrm{P}}}$ of (5.14) with $6 \cdot x_1 - 1$. For completeness, the structure is shown in Fig. 5.3 along with the linear upper bound (LUB) for the firing delay of A_1 in s^{P}_2. Domain $X_{\mathsf{F}^{\mathrm{P}}}$ of (5.14), is a 1-dimensional convex polytope in parameter x_1. It is shown in Fig. 5.4. Now, we show how to apply the analysis procedure of [33] developed around the divide & conquer strategy of [48]. Initially, a random configuration $\mathbf{x}_r$ inside $X_{\mathsf{F}^{\mathrm{P}}}$ is selected. For that configuration a maximum cycle mean expression (MCME) $\mathbf{M}_{\mathbf{x}_r}$ must be identified. This is achieved as follows. Given $\mathbf{x}_r$, for each scenario SDFG, its parameterized Max-plus matrix is determined by symbolic execution of the structure. During the execution, whenever the max operator of Max-plus is applied to two or more parametric dependency vector entries, only the largest among them propagates to the next step of the symbolic execution. It is straightforward to determine the largest by simply evaluating each one at $\mathbf{x}_r$. For the running example with



(a) SDF-PFSM-SADFG.



(b) LUB fo firing delay of A_2 in s_2^{P}.

Fig. 5.3: Example SDF-PFSM-SADFG for the analysis of [33].

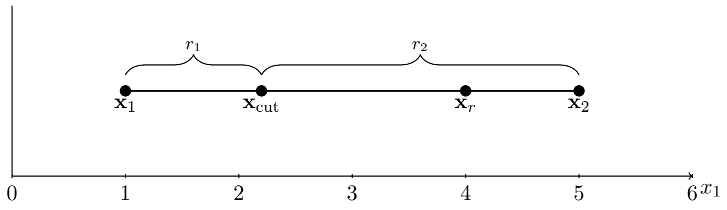


Fig. 5.4: Domain of SDF-PFSM-SADFG of Fig. 5.3a.

$\mathbf{x}_r = (x_1 = 4)$, we obtain the following matrices:

$$\mathcal{M}^{\mathrm{par}}_{s_1^{\mathrm{P}}}(\mathbf{x}_r) = \begin{bmatrix} x_1 & -\infty & -\infty & x_1 \\ -\infty & x_1+10 & -\infty & x_1+10 \\ x_1 & x_1+10 & -\infty & x_1+10 \\ -\infty & -\infty & 0 & -\infty \end{bmatrix} \tag{5.21}$$

and

$$\mathcal{M}^{\mathrm{par}}_{s_2^{\mathrm{P}}}(\mathbf{x}_r) = \begin{bmatrix} 0 & -\infty & -\infty & -\infty \\ -\infty & 6\cdot x_1-1 & -\infty & 6\cdot x_1-1 \\ -\infty & 6\cdot x_1-1 & -\infty & 6\cdot x_1-1 \\ -\infty & -\infty & 0 & -\infty \end{bmatrix}. \tag{5.22}$$

Once all parameterized scenario matrices have been computed, one needs

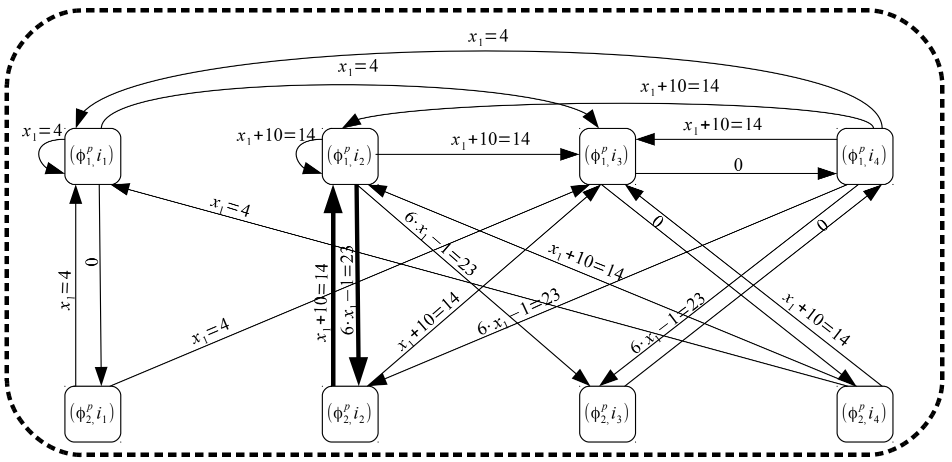


Fig. 5.5: Throughput graph of SDF-PFSM-SADFG of Fig. 5.3a.

to construct the parameterized throughput graph of the FSM-SADFG corresponding to configuration $\mathbf{x}_r$. The graph is constructed in the usual way as described in [46] using matrices of (5.21) and (5.22) and the scenario FSM of Fig. 5.2. It is displayed in Fig. 5.5. The parameterized throughput graph is then evaluated at $\mathbf{x}_r$ (right hand sides of the graph weight expressions in Fig. 5.5 are concrete values) to obtain a concrete throughput graph. Maximum cycle mean (MCM) analysis is performed on the concrete graph to determine the critical cycle (cycle with bold arrows in Fig. 5.5). Once the critical cycle is identified, using the relation between the edges in the parameterized and the concrete throughput graph, the MCM expression (MCME) $\mathbf{M}_{\mathbf{x}_r}$ of $\mathbf{x}_r$ can be extracted. The inverse of the $\mathbf{M}_{\mathbf{x}_r}$ defines the throughput expression for $\mathbf{x}_r$ as follows

$$\mathbf{Th}_{\mathbf{x}_r} = \frac{1}{\mathbf{M}_{\mathbf{x}_r}}. \tag{5.23}$$

For our $\mathbf{x}_r$,

$$\mathbf{M}_{\mathbf{x}_r} = \frac{1}{2} \cdot (7 \cdot x_1 + 9) \tag{5.24}$$

and

$$\mathbf{Th}_{\mathbf{x}_r} = \frac{2}{7 \cdot x_1 + 9}. \tag{5.25}$$

In any case, once $\mathbf{M}_{\mathbf{x}_r}$ had been found for $\mathbf{x}_r$, the divide & conquer algorithm evaluates the throughput $Th_{\mathsf{FP}}(\mathbf{x}_c)$ for each corner configuration $\mathbf{x}_c$ of the initial convex polytope (configurations $\mathbf{x}_1$ and $\mathbf{x}_2$ in Fig. 5.4). When the inverse of the throughput of the corner configuration is equal to the MCM value obtained when evaluating the MCME $\mathbf{M}_{\mathbf{x}_r}$ for $\mathbf{x}_c$, denoted $\mathbf{M}_{\mathbf{x}_r}(\mathbf{x}_c)$, then $\mathbf{x}_c$ belongs to the same throughput region as $\mathbf{x}_r$. If this holds for all corner points, the polytope is deemed a throughput region with throughput expression $\mathbf{Th}_{\mathbf{x}_r}$ defined by (5.23). If $\frac{1}{Th_{\mathsf{FP}}(\mathbf{x}_c)}$ is not equal to $\mathbf{M}_{\mathbf{x}_r}(\mathbf{x}_c)$, $\mathbf{x}_c$ belongs to some other throughput region. In that case, a new MCME $\mathbf{M}_{\mathbf{x}_c}$ is derived for $\mathbf{x}_c$ and the initial polytope is split into two subpolytopes via the cutting hyperplane $\mathbf{M}_{\mathbf{x}_c} = \mathbf{M}_{\mathbf{x}_r}$. For each subpolytope, the divide & conquer algorithm is recursively invoked until all throughput regions have been identified. We refer the interested reader to [48] for more details.

For our running example, the divide & conquer algorithm defines two throughput regions denoted with r_1 and r_2 in the Fig. 5.4. The line $\frac{1}{2} \cdot (7 \cdot x_1 + 9) = 10 + x_1$ divides the original domain into two throughput regions r_1 and r_2. As the initial domain is unidimensional, the configuration $\mathbf{x}_{\text{cut}} = (x_1 = 2.2)$ obtained as a solution of the equation $\frac{1}{2}\cdot(7\cdot x_1+9) = x_1+10$ determines the border between the two subregions. In r_1, $\mathbf{Th}_{\mathbf{x}} = 1/(x_1+10)$ for all $\mathbf{x} \in r_1$ and in r_2, $\mathbf{Th}_{\mathbf{x}} = 2/(7 \cdot x_1 + 9)$ for all $\mathbf{x} \in r_2$.

5.5.2 Problem transformation

The previously described analysis of FSM-SADF with paremeterized actor firing delays cannot be applied to SDF-PFSM-SADF structures with domain of type (5.10) due to several reasons.

First, the initial domain must be a convex polytope which is not the case with the formulation of (5.10). This is because the divide & conquer procedure of [48] does not apply to domain shapes other than convex polytope.

Second, in [33], actor firing delays are restricted to linear combinations of parameters and our analysis model allows these delays to be expressed as polynomial functions of parameters. If we try to consider actor firing delays posed differently than linear functions of parameters with [33], the underlying divide and & conquer algorithm would no longer work because now it would have to deal not with splitting hyperplanes but with splitting hypersurfaces. A recursive call to the divide & conquer including such a hypersurface would conflict with the initial requirement that the input domain must be a convex polytope.

Third, the analysis of [33] does not support parameterized rates that are inherent to SDF-PFSM-SADFG. In particular, in [33] if rates are kept constant and actor firing delays are linear combinations of parameters also the parameterized Max-plus matrix entries will be linear combinations of parameters. As the constructed throughput graph has its edge weights expressed in terms of those entries, they will also be linear combinations of parameters and so will the generated MCMEs. Because within the divide & conquer algorithm MCMEs define splitting hyperplanes, due to linearity of MCMEs are the domain subregions for the next call to divide & conquer also convex polytopes and divide & conquer can be continued until all throughput regions have been identified. However, as follows from the discussions of Chapters 3 and 4, in the presence of parameterized rates, entries of parameterized matrices are nonlinear functions of rates (and actor firing delays) while the divide & conquer of [33] in its current form accepts no nonlinearities.

To exemplify, we use Algorithm 3.1 to generate the conservative estimates of parameterized matrices of scenario graphs of SDF-PFSM-SADFG of Fig. 5.2. We obtain three matrices. Two for scenario s_1^{P} and one for s_2^{P}. These matrices are shown in (5.26), (5.27) and (5.28), respectively.

$$\mathcal{M}^{\mathrm{par}\prime}_{s_1^{\mathrm{P}}}(\pi_{s_1^{\mathrm{P}}}(\mathbf{x})) = \begin{bmatrix} x_1^2 & -\infty & -\infty & x_1^2 \\ -\infty & x_1+10 & -\infty & x_1+10 \\ x_1^2 & x_1+10 & -\infty & x_1^2 \\ -\infty & -\infty & 0 & -\infty \end{bmatrix}, \tag{5.26}$$

for all $\mathbf{x} \in X_{\mathsf{FP}} \cap (x_1^2 \geq x_1 + 10)$.

$$\mathcal{M}^{\mathrm{par}\prime}_{s_1^{\mathrm{P}}}(\pi_{s_1^{\mathrm{P}}}(\mathbf{x})) = \begin{bmatrix} x_1^2 & -\infty & -\infty & x_1^2 \\ -\infty & x_1+10 & -\infty & x_1+10 \\ x_1^2 & x_1+10 & -\infty & x_1+10 \\ -\infty & -\infty & 0 & -\infty \end{bmatrix}, \tag{5.27}$$

for all $\mathbf{x} \in X_{\mathsf{FP}} \cap (x_1^2 \leq x_1 + 10)$.

$$\mathcal{M}^{\mathrm{par}\prime}_{s_2^{\mathrm{P}}}(\pi_{s_2^{\mathrm{P}}}(\mathbf{x})) = \begin{bmatrix} 0 & -\infty & -\infty & -\infty \\ -\infty & x_1^2+4 & -\infty & x_1^2+4 \\ -\infty & x_1^2+4 & -\infty & x_1^2+4 \\ -\infty & -\infty & 0 & -\infty \end{bmatrix}, \tag{5.28}$$

for all $\mathbf{x} \in X$.

In case of scenario s_1^{P} SDF-PDFG, during the construction of mapping $\mathcal{M}^{\mathrm{par}\prime}_{s_1^{\mathrm{P}}}$, the initial domain needed to be partitioned into two exclusive parts defined by the exploration tree of Fig. 5.6. All these matrices contain entries which are nonlinear functions of domain parameters. Therefore, the analysis of [33] cannot be applied.

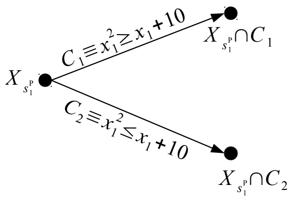

$X_{s_1^P}$
$C_1 \equiv x_1^2 \geq x_1 + 10$
$X_{s_1^P} \cap C_1$
$C_2 \equiv x_1^2 \leq x_1 + 10$
$X_{s_1^P} \cap C_2$

Fig. 5.6: Exploration tree for scenario s_1^{P} of the SDF-PFSM-SADFG of Fig. 5.2.

However, we do not give up on the divide & conquer approach of [33] and [48]. Instead we immerse ourselves into the task of problem linearization. If all matrix entries and constraints stemming from the definition of $X_{\mathsf{F}^{\mathrm{P}}}$ could be expressed as linear functions of some substitute variables spanning some higher-dimensional convex polytope $\tilde{X}_{\mathsf{F}^{\mathrm{P}}}$ derived from $X_{\mathsf{F}^{\mathrm{P}}}$, then the divide & conquer technique would be applicable in $\tilde{X}_{\mathsf{F}^{\mathrm{P}}}$. To perform the linearization, we will use the reformulation-linearization technique (RLT) introduced in [94]. Through the addition of variables called *lifting variables*, RLT is employed to define linear relaxations for polynomial programming problems where the objective function and constraint functions are polynomials with integer exponents. In particular, when applied, RLT gives an explicit algebraic characterization of the convex hull of the feasible region of the original problem.

RLT technique is developed in the context of polynomial programming. In our analysis model we allow parameterized rates and firing delays expressed as polynomial functions of parameters. These parameters are in turn defined over the domain $X_{\mathsf{F}^{\mathrm{P}}}$ composed of polynomial equalities and inequalities (cf. (5.11)). However, the entries of our parameterized matrices can be rational functions of domain parameters. This is a direct consequence of Proposition 3.1 used in calculating the actor responses which are then composed into a parameterized matrix. Therefore as a preceding step to RLT, we first need to represent entries of parameterized matrices as polynomial functions.

5.5.3 Problem polynomization

5.5.3.1 Approach outline

Across scenarios the respective parameterized matrices of the codomain of $\mathcal{M}^{\mathrm{par}}_{s_k^{\mathrm{P}}}$, where $s_k^{\mathrm{P}} \in \mathcal{S}^{\mathrm{P}}$ are entrywise represented using rational functions of parameters. To be able to apply RLT, we need to express the entries of parameterized matrices as polynomial functions. To do this, as a pre-step, we use the result of [23] which says that rational (objective) functions can be turned into polynomial ones by introducing additional variables we call called *polynomization* variables.

In particular, for a rational function of original parameters

$$\mathbf{x} \mapsto f_i(\mathbf{x}) := \frac{p_i(\mathbf{x})}{q_i(\mathbf{x})} \tag{5.29}$$

with given polynomials p_i, $q_i \in \mathbb{R}[\mathbf{x}]$, we introduce additional variables x_{n+i} called *polynomization* variables with associated constraints

$$\varphi_{n+i} \equiv \frac{p_i(\mathbf{x})}{q_i(\mathbf{x})} \leq x_{n+i}, i = 1, \ldots, M. \tag{5.30}$$

The original SDF-PFSM-SADF domain $X_{\mathsf{F}\mathrm{P}}$ is now expanded into a structure we call the polynomized domain $\hat{X}_{\mathsf{F}\mathrm{P}}$ that is represented in terms of polynomized configurations obtained by expanding (5.9) as follows

$$\hat{\mathbf{x}} = (x_1, \ldots, x_n, x_{n+1}, \ldots, x_{n+M}) \in \mathbb{R}^{n+M}. \tag{5.31}$$

Formally,

$$\hat{X}_{\mathsf{F}\mathrm{P}} = X_{\mathsf{F}\mathrm{P}} \cap \{\hat{\mathbf{x}} = (x_1, \ldots, x_n, x_{n+1}, \ldots, x_{n+M}) \in \mathbb{R}^{n+M} : \tag{5.32a}$$

$$\varphi_{n+i}(\hat{\mathbf{x}}) \equiv x_{n+i} \cdot q_i(\mathbf{x}) - p_i(\mathbf{x}) \geq 0 \text{ for } i = 1, \ldots, M \tag{5.32b}$$

$$l_{n+i} \leq x_{n+i} \leq u_{n+i}, \text{ for } i = 1, \ldots, M \tag{5.32c}$$

$$x_{n+i} \mapsto \frac{p_i(\mathbf{x})}{q_i(\mathbf{x})} \text{ for } i = 1, \ldots, M\}. \tag{5.32d}$$

Note that the polynomized domain in addition to the usual mathematical definition of the feasible region (cf. (5.32a), (5.32b) and (5.32c)), contains the information about the relationship of the polynomization variables and the original domain variables (cf. (5.32d)). We call this information polynomized domain metadata.

We have now introduced new (polynomization) variables to the problem to deal with rational functions hampering the use of RLT.

Now we are tempted to simply generate the set $\mathsf{M}_{\mathsf{F}\mathrm{P}} = \{\mathcal{M}^{\mathrm{par}}_{s_1^{\mathrm{P}}}, \ldots, \mathcal{M}^{\mathrm{par}}_{|\mathcal{S}^{\mathrm{P}}|}\}$ of all configuration to parameterized scenario mappings for a given SDF-PFSM-SADFG by using Algorithm 3.1 and apply polynomization to each matrix of $cod(\mathcal{M}^{\mathrm{par}}_{s_k^{\mathrm{P}}})$ across all $s_k^{\mathrm{P}} \in \mathcal{S}^{\mathrm{P}}$. Needless to say, many polynomization variables will be the same across these matrices and are, of course, not reintroduced.

However, the process of M_{FP} generation is unnecessarily demanding in the context of SDF-PFSM-SADF with static parameters. Instead of generating M_{FP}, in the spirit of the divide & conquer algorithm of [33], we wish to construct these matrices on-the-fly, i.e. when needed given a particular configuration of the linearized domain obtained by application of RLT. However, to construct the linearized domain via RLT, we need to obviously know the polynomized domain which boils to knowing the polynomization variables. We now show how to determine polynomization variables without actually generating the parameterized scenario matrices.

5.5.3.2 Determining polynomization variables

To see how to determine the polynomization variables, we first need to remind ourselves of the semantics of the parameterized scenario matrix entry, i.e. $[\mathcal{M}^{\mathrm{par}}_{s^{\mathrm{P}}_k}(\pi_{s^{\mathrm{P}}_k}(\mathbf{x}))]_{m,n}$. This entry specifies the minimal timing distance between token i_m in the current iteration of the scenario s^{P}_k SDF-PDFG and token i_n in the previous iteration. This distance is determined by the most tardy path among the paths of the graph connecting the producing actors of i_n and i_m. The tardiness of the path can be expressed using a delay-ratio abstraction as we witnessed in Chapter 3. The delay is determined by the firing delays of path actors, while the ratio is determined by the firing delay of the slowest actor in the path. As seen in Chapter 3, the involved firing delays are scaled where the scale factors are defined by products of channel rate fractions.

We exemplify using the graph of Fig. 5.7a. The graph in the figure contains only one path, namely $(A_1, A_2) - (A_2, A_3)$. We compute graph actor responses in dependence on the availability times of initial tokens after the kth graph iteration that form the timestamp vector

$$\gamma(k) = [t_{i_1}, t_{i_2}, t_{i_2}]^T. \tag{5.33}$$

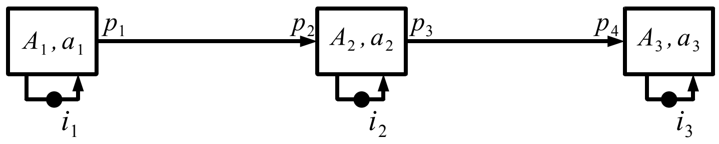
Actor A_1 has no input dependencies and can fire at a period determined by its own firing delay. Thus, the timestamps of tokens produces by A_1 are given as follows

$$\tau(A_1, n) = [{a_1}^{\otimes n}, -\infty, -\infty] \otimes \gamma(k). \tag{5.34}$$

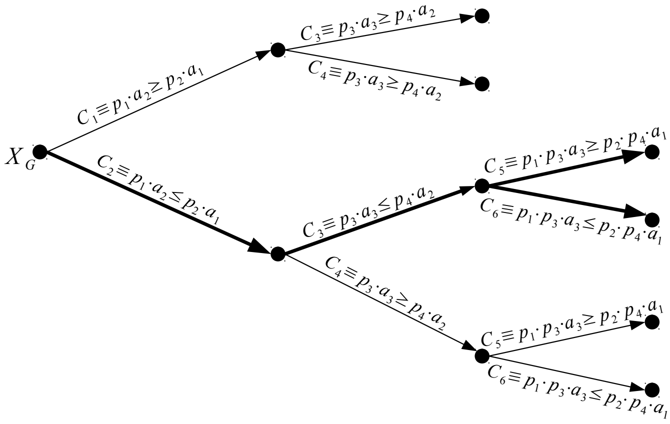
In consideration of actor A_2 (as follows from Proposition (3.1)), we need to differ between two cases

$$C_1 \equiv p_1 \cdot a_2 \geq p_2 \cdot a_1, \text{ and} \tag{5.35}$$

$$C_2 \equiv p_1 \cdot a_2 \leq p_2 \cdot a_1. \tag{5.36}$$



(a) Graph structure.


X_G
$C_1 \equiv p_1 \cdot a_2 \geq p_2 \cdot a_1$
$C_2 \equiv p_1 \cdot a_2 \leq p_2 \cdot a_1$
$C_3 \equiv p_3 \cdot a_3 \geq p_4 \cdot a_2$
$C_4 \equiv p_3 \cdot a_3 \geq p_4 \cdot a_2$
$C_3 \equiv p_3 \cdot a_3 \leq p_4 \cdot a_2$
$C_4 \equiv p_3 \cdot a_3 \geq p_4 \cdot a_2$
$C_5 \equiv p_1 \cdot p_3 \cdot a_3 \geq p_2 \cdot p_4 \cdot a_1$
$C_6 \equiv p_1 \cdot p_3 \cdot a_3 \leq p_2 \cdot p_4 \cdot a_1$
$C_5 \equiv p_1 \cdot p_3 \cdot a_3 \geq p_2 \cdot p_4 \cdot a_1$
$C_6 \equiv p_1 \cdot p_3 \cdot a_3 \leq p_2 \cdot p_4 \cdot a_1$

(b) Exploration tree for the structure.

Fig. 5.7: Structure used to explain polynomization and RLT.

Using (5.36) and (5.34), we obtain

$$\tau(A_2, n) \preceq [a_2 \otimes {a_1}^{\otimes(1+\frac{p_2}{p_1}\cdot n)}, {a_2}^{\otimes n}, -\infty] \otimes \gamma(k). \tag{5.37}$$

For A_3 using (5.37) we write

$$\begin{aligned}\tau(A_3, n) \preceq [a_1 \otimes a_2 \otimes conv({a_1}^{\otimes(\frac{p_2}{p_1}\cdot\lceil\frac{p_4}{p_3}\cdot n\rceil)}, {a_3}^{\otimes n}),\\ conv({a_2}^{\otimes\lceil\frac{p_4}{p_3}\cdot n\rceil}, {a_3}^{\otimes n}), {a_3}^{\otimes n}] \otimes \gamma(k).\end{aligned} \tag{5.38}$$

To obtain a closed form solution for the convolution defining the second entry of the dependency vector, we need to consider two cases

$$C_3 \equiv p_3 \cdot a_3 \geq p_4 \cdot a_2, \tag{5.39}$$

$$C_4 \equiv p_3 \cdot a_3 \leq p_4 \cdot a_2. \tag{5.40}$$

With (5.40), (5.38) transforms to

$$\begin{aligned}\tau(A_3, n) \preceq [a_1 \otimes a_2 \otimes conv({a_1}^{\otimes(\frac{p_2}{p_1}\cdot\lceil\frac{p_4}{p_3}\cdot n\rceil)}, {a_3}^{\otimes n}),\\ a_3 \otimes {a_2}^{\otimes(1+\frac{p_4}{p_3}\cdot n)}, {a_3}^{\otimes n}] \otimes \gamma(k).\end{aligned} \tag{5.41}$$

Now to account for the remaining convolution in the dependency vector of (5.41), we need to consider two cases

$$C_5 \equiv p_1 \cdot p_3 \cdot a_3 \geq p_2 \cdot p_4 \cdot a_1, \tag{5.42}$$

$$C_6 \equiv p_1 \cdot p_3 \cdot a_3 \leq p_2 \cdot p_4 \cdot a_1. \tag{5.43}$$

with (5.43), (5.41) transforms to

$$\begin{aligned}\tau(A_3, n) \preceq [&a_1 \otimes a_2 \otimes a_3 \otimes a_1{}^{\otimes \frac{p_2}{p_1} \cdot (1+\frac{p_4}{p_3} \cdot n)}, \\ &a_3 \otimes a_2{}^{\otimes (1+\frac{p_4}{p_3} \cdot n)}, a_3{}^{\otimes n}] \otimes \gamma(k) \\ = [&a_1 \otimes a_2 \otimes a_3 \otimes a_1{}^{\otimes \frac{p_2}{p_1}} \otimes a_1{}^{\otimes \frac{p_2}{p_1} \cdot \frac{p_4}{p_3} \cdot n}, \\ &a_3 \otimes a_2{}^{\otimes (1+\frac{p_4}{p_3} \cdot n)}, a_3{}^{\otimes n}] \otimes \gamma(k).\end{aligned} \tag{5.44}$$

On the other hand, with (5.42), (5.41) transforms to

$$\begin{aligned}\tau(A_3, n) \preceq [&a_1 \otimes a_2 \otimes a_1{}^{\otimes \frac{p_2}{p_1} \cdot (1+\frac{p_4}{p_3})} \otimes a_3{}^{\otimes n}, \\ &a_3 \otimes a_2{}^{\otimes (1+\frac{p_4}{p_3} \cdot n)}, a_3{}^{\otimes n}] \otimes \gamma(k) \\ = [&a_1 \otimes a_2 \otimes a_1{}^{\otimes \frac{p_2}{p_1}} \otimes a_1{}^{\otimes \frac{p_2}{p_1} \cdot \frac{p_4}{p_3}} \otimes a_3{}^{\otimes n}, \\ &a_3 \otimes a_2{}^{\otimes (1+\frac{p_4}{p_3} \cdot n)}, a_3{}^{\otimes n}] \otimes \gamma(k).\end{aligned} \tag{5.45}$$

Equations (5.44) and (5.45) produced by the exploration of bold paths of the exploration three of Fig. 5.7b reveal that the temporal distances between tokens are indeed defined by responses of actors along the graph path connecting the producing actors of those tokens that are scaled using rate fractions of channels defining the path. E.g. if we consider the firsts dependency vector entries of (5.44) and (5.45) that encode the temporal distance between i_3 and i_1, we see that this distance is determined by a delay-ratio abstraction, where the delays and rates are determined by firing delays of actors scaled by products of path channel rate fractions. In case of (5.44), A_1 is the slowest actor in the path and therefore the ratio equals to the firing delay of A_1 scaled by the product of rate fractions encountered in the path, i.e. $\frac{p_2}{p_1} \cdot \frac{p_4}{p_3}$. The delay is contributed to by all path actors with A_1's firing delay scaled with $\frac{p_2}{p_1}$. In case of (5.45), A_3 is the slowest and therefore the ratio equals to the firing delay of A_3 which needs not to be scaled as A_3 is the producing actor of i_3. The delay is contributed by all actors with the firing delay of A_1 scaled with $\frac{p_2}{p_1} \cdot \frac{p_4}{p_3}$.

These considerations reveal that no matter what the constellation of relative actor speeds is, the expressions for dependency vector elements will involve products of path channel rate fractions and actor firing delays.

ALGORITHM 5.1: Generation of the polynomized domain.

```
1  Function PolynomizeDomain($\mathcal{S}^{\mathrm{P}}$, $X_{\mathsf{FP}}$)
       input : Set $\mathcal{S}^{\mathrm{P}}$
       input : SDF-PFSM-SADFG domain $X_{\mathsf{FP}}$
       output: Polynomized SDF-PFSM-SADFG domain, $\hat{X}_{\mathsf{FP}}$
2      $\hat{X}_{\mathsf{FP}} \leftarrow X_{\mathsf{FP}}$ ;
3      $i \leftarrow 1$;
4      foreach $s_k^{\mathrm{P}} \in \mathcal{S}^{\mathrm{P}}$ do
5          foreach channel $\in C_{s_k^{\mathrm{P}}}$ do
6              chann_src_rate $\leftarrow$ SrcRate(channel) ;
7              chann_dst_rate $\leftarrow$ DstRate(channel) ;
8              if ExpressionType(chann_dst_rate/chann_src_rate) $=$ type_rational
                then
9                  if NotIncluded( chann_dst_rate / chann_src_rate) then
10                     $x_{n+i} \leftarrow$ chann_dst_rate / chann_src_rate;
11                     $l_{i+n}$ = LowerBound(chann_dst_rate / chann_src_rate);
12                     $u_{i+n}$ = UpperBound(chann_dst_rate / chann_src_rate);
13                     AddPolynomizationVar($\hat{X}_{\mathsf{FP}}, x_{n+i}$,
                        $x_{n+i} \cdot$ chann_src_rate $-$ chann_dst_rate $\geq 0$,
                        $l_{i+n} \leq x_{n+i} \leq u_{i+n}$) ;
14                     $i \leftarrow i + 1$;
15                 end if
16             end if
17         end foreach
18     end foreach
19     return $\hat{X}_{\mathsf{FP}}$;
20 end
```

Therefore, to polynomize entries of $\mathcal{M}^{\mathrm{par}}_{s_k^{\mathrm{P}}}(\pi_{s_k^{\mathrm{P}}}(\mathbf{x}))$ we need to introduce as many polynomization variables to the original domain as there are rational rate fractions in the s_k^{P} SDF-PDFG.

5.5.3.3 Domain polynomization

After determining polynomization variables for all scenarios graphs of an SDF-PFSM-SADFG, we can derive the desired parameterized version of the original domain. The overall procedure is specified by Algorithm 5.1. The input to the procedure are all scenario graphs of the SDF-PFSM-SADFG and the initial domain. The output of the algorithm is the polynomized SDF-PFSM-SADFG domain. We consider fractions of destination (cf. Line 7) and source rates (cf. Line 6) for all channels (cf. Line 5) of each scenario SDF-PDFG (cf. Line 4). If this fraction is rational in terms of parameters (cf. Line 8) and not already considered (cf. Line 9) a new polynomization variable is generated (cf. Line 10) for which lower and upper bounds are calculated (cf. Lines 11 and (12)). We must produce these bounds as RLT requires that all variables must be box constrained (note that box constraints are specified in the definition of the polynomized domain in (5.32c)). Finally, the polynomization variable x_{n+i} is added to the polynomized domain $\hat{X}_{\mathsf{F}\mathrm{P}}$ along with the associated constraints in Line 13. In the context of SDF-PFSM-SADF, where polynomization variables stand for fractions of rates that are always positive and represented as products of constant and parameters, we use the following simple relations to obtain the box constraints of a polynomization variable x_{n+i} of (5.30):

$$l_{n+1} = \frac{\min(p_i(\mathbf{x}))|_{\mathbf{x}\in X_{\mathsf{F}\mathrm{P}}}}{\max(q_i(\mathbf{x}))|_{\mathbf{x}\in X_{\mathsf{F}\mathrm{P}}}}, \qquad u_{n+1} = \frac{\max(p_i(\mathbf{x}))|_{\mathbf{x}\in X_{\mathsf{F}\mathrm{P}}}}{\min(q_i(\mathbf{x}))|_{\mathbf{x}\in X_{\mathsf{F}\mathrm{P}}}}. \tag{5.46}$$

For the example graph of Fig. 5.2 with its initial domain of (5.14), by introducing the polynomization variable $x_2 \mapsto \frac{1}{x_1}$, the polynomized domain takes the form

$$\begin{aligned}\hat{X}_{\mathsf{F}\mathrm{P}} = \{1 \le x_1 \le 5,& \\ x_1 \cdot x_2 - 1 \ge 0& \\ 0.2 \le x_2 \le 5,& \\ x_2 \mapsto \frac{1}{x_1}\}.&\end{aligned} \tag{5.47}$$

However, for the sake of simplicity of presentation, we take advantage of having performed the exploration of Algorithm 3.1 beforehand and obtained the matrices of (5.26), and (5.27). We observe that in (5.26) and (5.27) this fraction does not appear. This is due to the fact that actor A_0 in both scenario graphs fires only once within an iteration and therefore all tokens that A_2 consumes within an iteration are the result of a single A_0 firing and the fraction $\frac{1}{x_1}$ plays no role. Therefore, for the example SDF-PFSM-SADFG we can freely proceed with

$$\hat{X}_{\mathsf{F}\mathrm{P}} = X_{\mathsf{F}\mathrm{P}}. \tag{5.48}$$

5.5.4 Problem linearization

5.5.4.1 Approach outline

The polynomized domain mathematically formulated in (5.32) inherits all the properties of the initial domain defined in (5.10). In particular, it only extends the original domain by introducing the polynomization variables and their attached constraints to polynomize rational functions encountered as entries of parameterized scenario matrices.

Given an SDF-PFSM-SADFG and its polynomized domain, we now need to define a way how to linearize both the polynomized domain and all the parameterized matrices entries without knowing them beforehand.

What comes to our rescue is the RLT technique of [94] that is able to give an explicit algebraic characterization of the convex hull of the feasible region of the input space.

We define our linearization space as follows.

$$\begin{aligned}\mathcal{L} = \{\hat{\mathbf{x}} :& \varphi_m(\hat{\mathbf{x}}) \geq 0, m \in \mathbb{N}_0, \\ & \hat{\mathbf{x}} \in \hat{X}_{\mathsf{FP}}\}.\end{aligned} \tag{5.49}$$

In (5.49), $\varphi_m(\hat{\mathbf{x}})$ stands for a parameterized scenario matrix entry. Therefore, $\mathcal{L}$ consists of all entries of parameterized matrices of all SDF-PFSM-SADF scenarios and the associated polynomized domain itself. Note that after polynomization had been applied, $\varphi_m(\hat{\mathbf{x}})$ denotes a polynomial function in $\hat{X}_{\mathsf{FP}}$.

Now, denote $\Lambda = \{1, \ldots, n + M\}$ (the meaning of n and M stems from (5.31)) and let δ be the maximum degree of any polynomial appearing in $\mathcal{L}$ of (5.49) (note again that $\mathcal{L}$ includes $\hat{X}_{\mathsf{FP}}$). Define $\overline{\Lambda} = \{\Lambda, \ldots, \Lambda\}$ to be composed of δ replicates of Λ. Furthermore, let $J_l \subseteq \overline{\Lambda}$ be a set of indices for lower bound factors (note that the same index i may occur more than once in J_l). Similarly, let $J_u \subseteq \overline{\Lambda}$ be a set of indices for upper bound factors.

Then, in order to construct an explicit algebraic characterization of the convex hull of the feasible region of (5.49) the RLT procedure begins by generating *implied* constraints of the type

$$\prod_{j \in J_l} (x_j - l_j) \prod_{j \in J_u} (u_j - x_j) \geq 0, \tag{5.50}$$

where $J_u \subseteq \overline{\Lambda}$ is a set of indices for upper bounding factors and $J_l \subseteq \overline{\Lambda}$ is the set of indices for lower bounding factors where $|J_l \cup J_u| = \delta$. This addition of implied constraints to $\mathcal{L}$ of (5.49) is called the *reformulation* phase. The number of distinct constrains of type (5.50) is given by

$$T = \sum_{k=0}^{\delta} \binom{n + M + k - 1}{k} \cdot \binom{n + M + (\delta - k) - 1}{(\delta - k)}. \tag{5.51}$$

After the addition, let us substitute

$$X_J = \prod_{j \in J} x_j, \text{ for all } J \subseteq \overline{\Lambda}, \tag{5.52}$$

where the indices in J are assumed to be sequenced in nondecreasing order and where $X_{\{j\}} \equiv x_j$ for all $j \in \Lambda$ and $X_\emptyset \equiv 1$. This phase is called the linearization phase as with the substitution of (5.52) all inequalities and equalities contained in (5.49) expanded with (5.50) are now linear in the variables X_J that we call the *lifting* variables. The number of X-variables of (5.52) besides $X_{\{j\}}$ and $X_\emptyset$ is

$$L = \binom{n+M+\delta}{\delta} - (n+M+1). \tag{5.53}$$

To summarize, when RLT of (5.50) and (5.52) is applied to the space $\mathcal{L}$ of (5.49) defined over $\hat{X}_{\mathsf{FP}}$, $\mathcal{L}$ transforms to the linearized space $\tilde{\mathcal{L}}$ defined as follows

$$\begin{aligned}\tilde{\mathcal{L}} = \{\tilde{\mathbf{x}} :& \tilde{\varphi}_m(\hat{\mathbf{x}}) \geq 0, m \in \mathbb{N}_0,\\ & \tilde{\mathbf{x}} \in \tilde{X}_{\mathsf{FP}}\}\end{aligned} \tag{5.54}$$

Notation $\tilde{\mathbf{x}}$ is used to denote a linearized configuration defied as follows

$$\tilde{\mathbf{x}} = (\hat{\mathbf{x}}, X_{J_1}, \ldots, X_{J_L}). \tag{5.55}$$

The linearized configuration $\tilde{\mathbf{x}}$ belongs to the linearized domain $\tilde{X}_{\mathsf{FP}}$ that is derived from $\hat{X}_{\mathsf{FP}}$ by the application of RLT to $\mathcal{L}$ of (5.49) and is specified as follows

$$\begin{aligned}
\tilde{X}_{\mathsf{FP}} = \{\tilde{\mathbf{x}} = (\hat{\mathbf{x}}, X_{J_1}, \ldots, X_{J_L}) \in \mathbb{R}^{n+M+L} :& \qquad (5.56a)\\
\tilde{\varphi}_r(\tilde{\mathbf{x}}) \geq \beta_r \text{ for } r = 1, \ldots, R_1,& \qquad (5.56b)\\
\tilde{\varphi}_r(\tilde{\mathbf{x}}) = \beta_r \text{ for } r = R_1, \ldots, R,& \qquad (5.56c)\\
\tilde{\varphi}_{n+i}(\tilde{\mathbf{x}}) \geq 0 \text{ for } i = 1, \ldots, M,& \qquad (5.56d)\\
\tilde{\varphi}_t(\tilde{\mathbf{x}}) \geq 0 \text{ for } k = 1, \ldots, T& \qquad (5.56e)\\
l_i \leq x_i \leq u_i \text{ for } i = 1, \ldots, n,& \qquad (5.56f)\\
l_i \leq x_i \leq u_i \text{ for } i = n+1, \ldots, n+M,& \qquad (5.56g)\\
l_{J_j} \leq X_{J_j} \leq u_{J_j} \text{ for } j = 1, \ldots, L,& \qquad (5.56h)\\
x_{n+i} \mapsto \frac{p_i(\mathbf{x})}{q_i(\mathbf{x})} \text{ for } i = n+1, \ldots n+M,& \qquad (5.56i)\\
X_{J_j} \mapsto \prod_{i \in J_j} x_i, \text{ for } j = 1, \ldots, L,\ i = 1, \ldots, n+M\}.& \qquad (5.56j)
\end{aligned}$$

The linearized domain inherits all the constraints of $\hat{X}_{\mathsf{FP}}$ (cf. (5.56d) and (5.56g)) and transitively from X_{FP} (cf. (5.56b), (5.56c) and (5.56f)). The difference is that these constraints are now linearized and expressed in terms of $\tilde{\mathbf{x}}$. Together with the implied constrains (cf. (5.56e)) and the box constraints defined for lifting variables (cf. (5.56h)), $\tilde{X}_{\mathsf{FP}}$ now forms the algebraic characterization of the convex hull of the original feasible region of X_{FP}. The box constraints for the lifting variables are simply derived from the box constrains of the $\hat{X}_{\mathsf{FP}}$ variables defining the substitution of (5.52). In addition, the linearized domain, contains meta-data, i.e. the information of the substitutions made on the road from X_{FP} via $\hat{X}_{\mathsf{FP}}$ to $\tilde{X}_{\mathsf{FP}}$ (cf. (5.56i) and (5.56j)).

Now again one might be tempted to apply RLT after deriving all parameterized matrices using Algorithm 3.1. This way, the space of (5.49) is fully defined and RLT can be straightforwardly applied via (5.50) and (5.52). Recall however that within the divide & conquer algorithm we wish to construct these matrices on-the-fly and therefore, we do not *a priori* know how the linearization space of (5.49) looks like. Thus, we need to apply RLT on-the-fly too during the process of determining the parameterized matrix for a linearized configuration $\tilde{\mathbf{x}}$. The problem is however, that without knowing the maximum degree of polynomials appearing in $\mathcal{L}$ we cannot construct $\tilde{\mathbf{x}}$ either. Hence, the crux of the matter is determining δ.

5.5.4.2 Determining δ

We once again consider the structure of Fig. 5.7a and the actor responses computed for the structure that are expressed as scalar products of the dependency vector and the initial token timestamp vector. These vectors are used to construct the parameterized matrix by evaluating them for n which equals to the repetition vector entry for that actor. Dependency vector entries are polynomial functions. Therefore, we need to determine the maximal degree among these polynomials. Consider the response of actor A_3 of (5.44). In particular, consider the first entry of the dependency vector. This entry encodes the minimal temporal distance between token i_3 and initial token i_1. It is defined by the path $(A_1, A_2) - (A_2, A_3)$ and therefore takes into account all rate fractions encountered along the path. Thus the polynomial is at least of the degree of the product of all rate fractions along the path (we call this degree the *path degree*) increased by the degree of the (possibly polynomial) expression that defines the actor firing delay itself (we call this degree the *actor degree*). The other contributing factor is n which at the moment of the construction of the parameterized matrix takes the value of the repetition vector entry of A_3. Therefore, the degree of any entry of the dependency vector defining the response of an actor at the iteration boundary cannot be greater than the maximum degree of actor's repetition vector entry (we call this degree the *repetition vector* *entry degree*) entry increased by the maximal degree of any graph path leading to that actor increased by the actor degree.

With this in mind, we give the procedure to determine δ. It is specified in Algorithm 5.2.

The inputs to the algorithm are all scenario SDF-PDFGs of the SDF-PFSM-SADFG and the polynomized domain of the SDF-PFSM-SADFG. The output is the required maximal degree of any polynomial appearing in $\mathcal{L}$. The algorithm iterates over all scenario SDF-PDFGs (cf. Line 3). For each scenario graph, first all feedback loops are cut, rendering it acyclic (cf. Line 4). This can be done because we consider only graphs that are acyclic within an iteration (cf. Requirement 3.1). In such graphs, due to consistency, rates on feedback channels (channels with initial tokens) are constant, and therefore do not influence the degree of polynomials encountered during parameterized Max-plus matrix generation and can be disregarded in the scope of Algorithm 5.2. Now, within the acyclic scenario specification, we consider all paths (cf. Line 7). For a path, we consider all its channels (cf. Line 10). For each considered channel, we fix its source actor (cf. Line 11) and then consider how it influences all the actors along the path. First we check whether the considered actor has a self-edge or not (cf. Line 13). If it does not, it only influences the downstream actors by an amount of delay equal to its firing delay and therefore only the degree of its firing delay needs to be accounted for (cf. Line 27). If it does have a self-edge then it influences the responses of the downstream actors by its firing delay scaled by products of channel rate fractions. The influence needs to be accounted for each downstream actor separately (cf. Line 18). For each channel of the subpath defined by the currently considered actor (`SrcActor`) and the sink actor, we check whether the channel has parameterized rates or not (cf. Line 19). If it does, the degree of the subpath polynomials is increased by one (cf. Line 20) to account for one additional factor in the rate fraction product that scales the output of the considered actor to input of the currently considered downstream actor (`DstActor`). Here we also account the for the value of n for which particular dependency vectors are finally evaluated before they are collected into the parameterized matrix. For the `SrcActor` this n equals to the repetition vector entry of the considered downstream actor. Therefore, the path degree is adjusted accordingly (cf. Line 22). Initially the repetition vector entry of the considered actor itself needs to be accounted. This entry multiplying its firing delay defines the parameterized matrix diagonal entries (cf. Line 15).

Finally, after all paths of all scenario graphs have been accounted for, one needs to take also the maximal degree of the polynomial appearing in

ALGORITHM 5.2: Determination of maximal degree of any polynomial of $\mathcal{L}$.

```
Function MaxDegree($\mathcal{S}^{\mathrm{P}}, \hat{X}_{\mathrm{FP}}$)
    input : Set $\mathcal{S}^{\mathrm{P}}$
    input : Parameterized domain $\tilde{X}_{\mathrm{FP}}$
    output: Max degree $\delta$
    $\delta \leftarrow$ 0 ;
    foreach $s_k^{\mathrm{P}} \in \mathcal{S}^{\mathrm{P}}$ do
        G ← ConvertToAcyclic($s_k^{\mathrm{P}}$) ;
        Paths ← GetAllPaths(G);
        path ← null;
        while path ← GetNextPath(Paths, path) do
            channel ← null;
            pathdegree ← 0;
            while channel ← GetNextChannel(path, channel) do
                /* Consider the propagation of this actor along the
                   path                                                */
                SrcActor ← GetSrcActor(channel);
                subpathdegree ← 0 ;
                if HasSelfEdge(SrcActor) then
                    /* Account for diagonal matrix entries             */
                    tmp ← PolyDeg(GetFirDelay(SrcActor) ·
                     GetRVEntry(SrcActor)) ;
                    pathdegree ← max(pathdegree, tmp) ;
                    tmpchannel ← channel;
                    subpathdegree ← PolyDeg(GetFirDelay(SrcActor)) ;
                    repeat
                        if HasParametricRates(tmpchannel) then
                            subpathdegree ← subpathdegree + 1;
                            /* Add the degree of the repetition vector
                               entry of the dest                       */
                            DstActor ← GetDstActor(channel)
                            subpathdegree ← subpathdegree + PolyDeg(
                             GetRVEntry(DstActor))
                        end if
                    until tmpchannel ← GetNextChannel(path, tmpchannel);
                    /* Done with propagation of this actor             */
                    pathdegree ← max(pathdegree, subpathdegree) ;
                else
                    /* Account firing delay only                       */
                    pathdegree ← max(pathdegree,
                     PolyDeg(GetFirDelay(SrcActor))) ;
                end if
            end while
        end while
        $\delta \leftarrow$ max($\delta$, pathdegree);
    end foreach
    $\delta \leftarrow$ max($\delta$, PolyDeg($\hat{X}_{\mathrm{FP}}$));
    return $\delta$;
end
```

the polynomized domain $\hat{X}_{\mathsf{FP}}$ into account (cf. Line 33).

Algorithm 5.2 results in a degree that in reality might be lower than the actual maximal degree. In theory of RLT, using any higher degree is perfectly acceptable. Still the drawback is that then the relaxation size will be larger than necessary [74].

5.5.4.3 Domain linearization exemplified

We now apply RLT to the domain of our example SDF-PFSM-SADFG of Fig. 5.2. The application of Algorithm 5.2 yields $\delta = 2$. The linearized domain is given as follows:

$$\begin{aligned}\tilde{X}_{\mathsf{FP}} = \{1 \leq x_1 \leq 5,\\ &\tilde{\mathbf{x}} : \tilde{\varphi}_t(\tilde{\mathbf{x}}) \geq 0, \text{ for } t = 1, \ldots, 3\\ &1 \leq X_{11} \leq 25,\\ &X_{11} \mapsto x_1 \cdot x_1\}.\end{aligned} \tag{5.57}$$

where

$$\tilde{\mathbf{x}} = (x_1, X_{11}). \tag{5.58}$$

In (5.57), $\tilde{\varphi}_t$ stands for the set of linearized implied constraints obtained during the reformulation phase of (5.50) and afterwards linearized during the linearization phase of (5.52) using lifting variable X_{11}. There are in total 3 such constraints given as follows:

$$\begin{aligned}J_l = \{1, 1\}, J_u = \emptyset :&[(x_1 - 1)(x_1 - 1)] \geq 0\\ &\to X_{11} - 2 \cdot x_1 + 1\end{aligned} \quad , \tag{5.59}$$

$$\begin{aligned}J_l = \emptyset, J_u = \{1, 1\} :&[(5 - x_1)(5 - x_1)] \geq 0\\ &\to X_{11} - 10 \cdot x_1 + 25\end{aligned} \quad , \tag{5.60}$$

and

$$\begin{aligned}J_l = \{1\}, J_u = \{1\} :&[(x_1 - 1)(5 - x_1)] \geq 0\\ &\to -X_{11} + 6 \cdot x_1 - 5\end{aligned} \quad . \tag{5.61}$$

5.5.5 Divide and conquer

We now fully specify the procedure for parametric throughput analysis. We start from the top-level. In particular we adapt the divide & conquer algorithm of [48] to fit our purpose. Its specification is given by Algorithm 5.3. The inputs to the algorithm are the SDF-PFSM-SADFG itself, its linearized (sub)domain and the reference to the result set. First, a random configuration in $\tilde{X}_{\mathsf{FP}}$ is selected (cf. Line 3). For that configuration we then

ALGORITHM 5.3: Adapted divide & conquer algorithm of [48].

```
1  Function DivideAndConquer(G, $\tilde{X}_{\mathsf{FP}}$, ref Mcmes)
       input : SDF-PFSM-SADFG G
       input : Linearized (sub)domain $\tilde{X}_{\mathsf{FP}}$ of G
       output: Set of MCM expressions, denoted Mcmes
2      bBranchingNode ← false;
3      $\tilde{\mathbf{x}}_r$ ← GetRndConf($\tilde{X}_{\mathsf{FP}}$);
4      $\mathbf{M}_{\tilde{\mathbf{x}}_r}$ ← ComputeMCME(G, $\tilde{X}_{\mathsf{FP}}$,$\tilde{\mathbf{x}}_r$);
5      Vertices ← GetVertices($\tilde{X}_{\mathsf{FP}}$);
6      foreach $\tilde{\mathbf{x}}_{c_i}$ ∈ Vertices do
7          if $\mathbf{M}_{\tilde{\mathbf{x}}_r}(\tilde{\mathbf{x}}_c) \neq 1$/ComputeThroughput($\tilde{\mathbf{x}}_{c_i}$) then
8              $\mathbf{M}_{\tilde{\mathbf{x}}_{c_i}}$ ← ComputeMCME(G, $\tilde{X}_{\mathsf{FP}}$, $\tilde{\mathbf{x}}_{c_i}$) ;
9              bBranchingNode ← true;
10             $\tilde{X}'_{\mathsf{FP}}$ ← $\tilde{X}_{\mathsf{FP}} \cup \{\mathbf{M}_{\tilde{\mathbf{x}}_r} - \mathbf{M}_{\tilde{\mathbf{x}}_{c_i}}\}$ ;
11             $\tilde{X}''_{\mathsf{FP}}$ ← $\tilde{X}_{\mathsf{FP}} \cup \{\mathbf{M}_{\tilde{\mathbf{x}}_{c_i}} - \mathbf{M}_{\tilde{\mathbf{x}}_r}\}$
12             DivideAndConquer(G, $\tilde{X}'_{\mathsf{FP}}$, Mcmes);
13             DivideAndConquer(G, $\tilde{X}''_{\mathsf{FP}}$, Mcmes);
14         end if
15     end foreach
16     if bBranchingNode = false then
17         AddtoResultSet(Mcmes, $\mathbf{Th}_{\tilde{\mathbf{x}}_r} = \frac{1}{\mathbf{M}_{\tilde{\mathbf{x}}_r}}$, $\tilde{X}_{\mathsf{FP}}$);
18     end if
19 end
```

determine the corresponding MCME (cf. Line 4). If the throughput of all corner configurations (vertices) of the polytope $\tilde{X}_{\mathsf{FP}}$ is equal to the inverse of $\mathbf{M}_{\tilde{\mathbf{x}}_r}(\tilde{\mathbf{x}}_{c_i})$, then the input (sub)domain $\tilde{X}_{\mathsf{FP}}$ defines a throughput region and the inverse of $\mathbf{M}_{\tilde{\mathbf{x}}_r}$ is the desired throughput expression for that region, i.e. $\mathbf{Th}_{\tilde{\mathbf{x}}_r}$. Throughput expression $\mathbf{Th}_{\tilde{\mathbf{x}}_r}$ is added to the result set (cf. Line 17). If not, a splitting hyperplane between $\tilde{\mathbf{x}}_r$ and $\tilde{\mathbf{x}}_{c_i}$ is defined as $\mathbf{M}_{\tilde{\mathbf{x}}_r} = \mathbf{M}_{\tilde{\mathbf{x}}_{c_i}}$ where $\mathbf{M}_{\tilde{\mathbf{x}}_{c_i}}$ is the MCME valid for $\tilde{\mathbf{x}}_{c_i}$ and calculated by the call of Line 8. The splitting plane splits the original domain into two subdomains for which the search is recursively continued (cf. Lines 12 and 13) until all throughput regions are identified. The throughput for an arbitrary configuration $\tilde{\mathbf{x}}_r$ is computed by employing the techniques of Section 4.9 but in consideration of a single configuration, i.e. $\tilde{\mathbf{x}}_r$.

5.5.6 Deriving MCME of a configuration

The key part of the divide & conquer algorithm is the `ComputeMCME` function that returns the MCME for some configuration of the linearized SDF-PFSM-SADFG domain. Adopted from [33] and recast into the context

ALGORITHM 5.4: Adapted `ComputeMCME` algorithm of [33] for MCME computation.

```
1  Function ComputeMCME(G, $\tilde{X}_{\mathsf{FP}}$, $\mathbf{x}_r$)
       input : SDF-PFSM-SADFG G
       input : Domain $\tilde{X}_{\mathsf{FP}}$ of G
       input : Configuration $\tilde{\mathbf{x}}_r$ in $\tilde{X}_{\mathsf{FP}}$
       output: MCME $\mathbf{M}_{\tilde{\mathbf{x}}_r}$ for $\tilde{\mathbf{x}}_r$
2      foreach $s_k^{\mathrm{P}} \in \mathcal{S}^{\mathrm{P}}$ do
3          AddtoSet(SetofParameterizedMatrices, GetLinearizedParamMat($s_k^{\mathrm{P}}$, $\tilde{\mathbf{x}}_r$,
             $\tilde{X}_{\mathsf{FP}}$))
4      end foreach
5      PTG ← ConstructPTG(SetofParameterizedMatrices, G) ;
6      CTG ← EvaluatePTG(PTG);
7      criticalCycle ← MCM(CTG) ;
8      foreach cycle ∈ criticalCycle do
9          $\mathbf{M}_{\tilde{\mathbf{x}}_r}$ ← $\mathbf{M}_{\tilde{\mathbf{x}}_r}$ + GetEdgeWeight(cycle, PTG) ;
10     end foreach
11     $\mathbf{M}_{\tilde{\mathbf{x}}_r}$ ← $\mathbf{M}_{\tilde{\mathbf{x}}_r}$/ length(cycle);
12     return $\mathbf{M}_{\tilde{\mathbf{x}}_r}$;
13 end
```

of SDF-PFSM-SADF it is specified by Algorithm 5.4. The input to the algorithm is the SDF-PFSM-SADFG itself, its linearized domain and the configuration for which the MCME is to be determined. This MCME is the output of the algorithm. First, for all scenarios graphs their respective linearized parameterized Max-plus matrices are constructed (cf. Line 3). These matrices are valid for the configuration of consideration. Then, using the scenario FSM and these matrices, the linearized parameterized throughput graph of the considered SDF-PFSM-SADFG is constructed (cf. Line 5) and evaluated at $\tilde{\mathbf{x}}_r$ (cf. Line 6). The purpose of the step is to identify the critical cycle of the starting linearized parameterized throughput graph via MCM analysis of the concrete throughput graph (cf. Line 7). Thereafter, using the correspondence between edges in the linearized parameterized and concrete throughput graph, the weight of the critical cycle can be computed as a parametric expression in $\tilde{X}_{\mathsf{FP}}$ (cf. Line 9). The final $\mathbf{M}_{\tilde{\mathbf{x}}_r}$ is produced by dividing the result of Line 9 with the cycle length (cf. Line 11).

5.5.7 Deriving the parameterized scenario matrix for a parameter space point

The crucial part of Algorithm 5.4 is the routine `GetLinearizedParamMat` that given a configuration $\tilde{\mathbf{x}}_r$ returns the associated linearized parameterized matrix of the considered scenario SDF-PDFG. The specification of the routine is given by Algorithm 5.5.

The algorithm is a simplified version of Algorithm 3.1. Simplified because there is no need for recursive calls present in Algorithm 3.1. Here, the domain needs not to be split as using the configuration of interest we can immediately resolve all splitting options that emerge from the considerations related to Max-plus convolutions and Max-plus superposition.

The inputs to the algorithms are the scenario SDF-PDFG, the linearized domain of the superordinate SDF-PFSM-SADFG and the configuration under consideration. The output is the linearized parameterized matrix of the scenario graph valid for the given configuration. As in Algorithm 3.1, we compute one iteration of the parameterized graph using its quasi-static schedule (cf. Line 2). Actors are considered in the order they appear in the quasi-static schedule (cf. Line 3). According to the Max-plus superposition principle we can consider one input channel at a time (cf. Line 7). The input is represented, as usual, by the Max-plus dependency vector and a convolution needs to be performed between the impulse response of the considered actor and all entries of the input dependency vector (cf. Lines 10 and 11). As a Max-plus convolution gives two options to proceed with, in Algorithm 3.1 the initial domain is split into two exclusive parts and the exploration is recursively continued. Here, as we know the configuration of interest, there is no need for that. In particular, we determine which the option of interest using the configuration is and proceed only with that option (cf. Line 12). Once all the actor input channels have been considered in this manner, we now need to superpose the contributions of different input channels. In Algorithm 3.1 this step included further partitioning of the initial domain. Here the same argument holds as with the Max-plus convolution. We need not to partition, as by knowing the configuration of interest, we only proceed with the relevant case. In particular, having all the input contribution dependency vectors, we construct the output dependency vector as a vector whose entries are the maximal dependency vector entries across all corresponding input dependency vector entries. As in Algorithm 3.1, we use the delay-ratio abstraction, i.e. we choose the maximal delay and ratio across all input dependency vector entries (cf. Lines 18 and 19). After all actors in the quasi-static schedule have been processed, we simply evaluate and collect the responses of initial token producing actors into the matrix $\mathcal{M}_G^{\mathrm{par}\prime}$ (cf. Line 25). Thereafter, we polynomize it (cf. Line 26) so we can finally linearize it in terms of lifting variables of the linearized domain (cf. Line 27). During both steps, we make use of the domain metadata that specifies all the substitutions.

ALGORITHM 5.5: Derive parameterized Max-plus matrix of a parameterized scenarios for a linearized configuration.

```
1  Function GetLinearizedParamMat(G, $\tilde{X}_{\mathsf{FP}}$ , $\tilde{\mathbf{x}}_r$)
       input : Scenario SDF-PDFG G
       input : Linearized domain $\tilde{X}_{\mathsf{FP}}$
       input : Configuration $\tilde{\mathbf{x}}_r$
       output: $\tilde{\mathcal{M}}_G^{\mathrm{par}\prime}$ - RLT linearized version of matrix $\mathcal{M}_G^{\mathrm{par}\prime}(\tilde{\mathbf{x}}_r)$
2      Qss ← GetQss(G);
3      n ← NumofInitialTok(G);
4      while curr_actor ← GetNextActor(Qss, curr_actor) do
5          h ← GetImpuseResponse(curr_actor) ;
6          output_channels ← GetOutputChannels(curr_actor);
7          curr_actor_contributions ←
            CreateArrayofInputContributions(curr_actor);
8          while currinchan ← GetNextInChan(curr_actor, currinchan) do
9              for $i \leftarrow 1$ to n do
10                 currinchandep ← T(currinchan, $i$);
11                 response ← maxplus_convolution(currinchandep, h)
12                 response_pruned ← EvaluateAndPrune(response, $\tilde{X}_{\mathsf{FP}}$ , $\tilde{\mathbf{x}}_r$);
13                 curr_actor_contributions(currinchan,currinchandep) ←
                    response_pruned;
14                 $i \leftarrow i + 1$;
15             end for
16         end while
           /* Completed all input contributions, proceed with
              superposition                                          */
17         for $i \leftarrow 1$ to n do
18             delay ← PruneDelays(curr_actor_contributions(:,currinchandep)
                , $\tilde{\mathbf{x}}_r$) ;
19             ratio ←
                PruneRatios(curr_actor_contributions(:,currinchandep)), $\tilde{\mathbf{x}}_r$)
                ;
20             T(output_channels, $i$) ← (delay,ratio);
21             $i \leftarrow i + 1$;
22         end for
23         Update(T, output_channels) ;
24     end while
25     $\mathcal{M}_G^{\mathrm{par}\prime}$ ← ConstructMatrix(T, G, Qss);
26     $\hat{\mathcal{M}}_G^{\mathrm{par}\prime}$ ← PolynomizeMatrix($\mathcal{M}_G^{\mathrm{par}\prime}$, $\tilde{X}_{\mathsf{FP}}$);
27     $\tilde{\mathcal{M}}_G^{\mathrm{par}\prime}$ ← LinearizeMatrix($\hat{\mathcal{M}}_G^{\mathrm{par}\prime}$, $\tilde{X}_{\mathsf{FP}}$);
28     return $\tilde{\mathcal{M}}_G^{\mathrm{par}\prime}$;
29 end
```

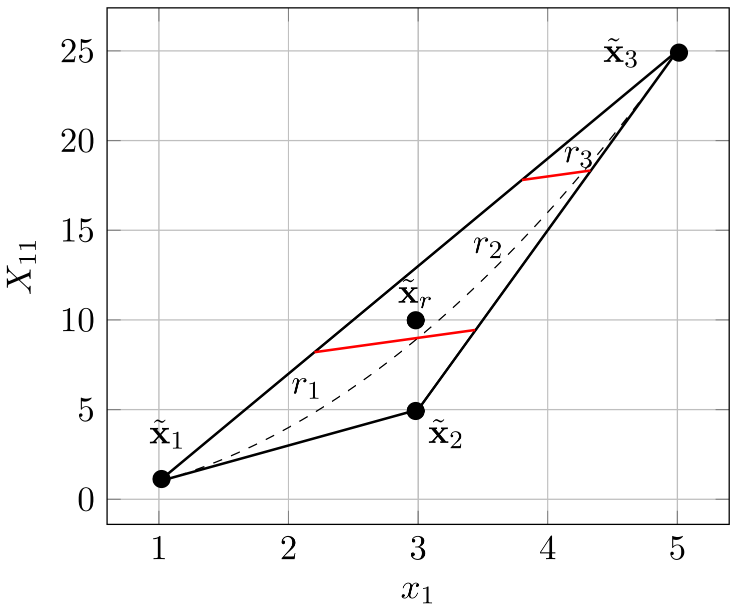


Fig. 5.8: Linearized domain $\tilde{X}_{\mathsf{FP}}$ (cf. (5.57)) of the example SDF-PFSM-SADFG of Fig. 5.2

5.6 Evaluation

5.6.1 Case study

We now show how do derive all the throughput regions and associated throughput expressions using an artificial (but representative) case study application captured by the SDF-PFSM-SADFG of Fig. 5.2[3].

To do so, we apply the divide & conquer procedure of Algorithm 5.3. The linearized domain of (5.57) that alongside the SDF-PFSM-SADFG itself is the input to the procedure is shown in Fig. 5.8.

From the figure, we see that $\tilde{X}_{\mathsf{FP}}$ defines a convex hull of the original feasible region (domain) of the SDF-PFSM-SADFG defined by positive integer points (recall that x_1 is used to parameterize rates too and thus is limited to a subset of positive integers) of the curve $X_{11} = x_1 \cdot x_1$ and depicted using a dashed line. All linearized configurations not being positive integer points of this curve are not feasible in the graph because for those configurations $X_{11} \neq x_1 \cdot x_1$ or $x_1, X_{11} \notin \mathbb{N}_{>0}$. However, the divide & conquer must consider the entire relaxation of the feasible region.

[3]Note that future work is needed to add more complete realistic case studies.

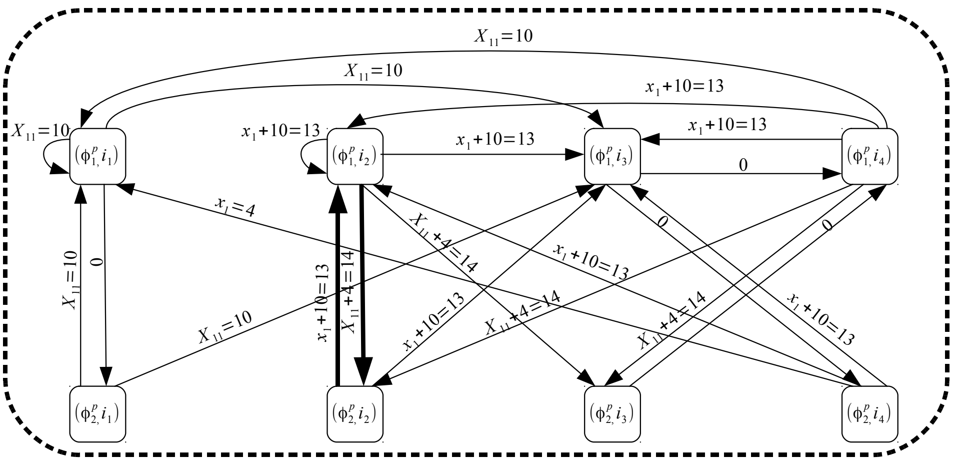


Fig. 5.9: Linearized parameterized throughput graph for $\tilde{\mathbf{x}}_r$.

It starts by selecting a random configuration $\tilde{\mathbf{x}}_r \in \tilde{X}_{\mathsf{F}^{\mathrm{P}}}$. Let $\tilde{\mathbf{x}}_r = (x_1 = 3, X_{11} = 10)$. First, we compute the linearized parameterized scenario s_1^{P} and s_1^{P} matrices using Algorithm 5.5.

These matrices take the form

$$\tilde{\mathcal{M}}^{\mathrm{par}\prime}_{s_1^{\mathrm{P}}}(\tilde{\mathbf{x}}_r) = \begin{bmatrix} X_{11} & -\infty & -\infty & X_{11} \\ -\infty & x_1+10 & -\infty & x_1+10 \\ X_{11} & x_1+10 & -\infty & x_1+10 \\ -\infty & -\infty & 0 & -\infty \end{bmatrix} \tag{5.62}$$

and

$$\tilde{\mathcal{M}}^{\mathrm{par}\prime}_{s_2^{\mathrm{P}}}(\tilde{\mathbf{x}}_r) = \begin{bmatrix} 0 & -\infty & -\infty & -\infty \\ -\infty & X_{11}+4 & -\infty & X_{11}+4 \\ -\infty & X_{11}+4 & -\infty & X_{11}+4 \\ -\infty & -\infty & 0 & -\infty \end{bmatrix}. \tag{5.63}$$

and represent conservatives estimates of matrices $\mathcal{M}^{\mathrm{par}}_{s_1^{\mathrm{P}}}(\tilde{\mathbf{x}}_r)$ and $\mathcal{M}^{\mathrm{par}}_{s_2^{\mathrm{P}}}(\tilde{\mathbf{x}}_r)$, respectively. This is because Algorithm 5.5 uses Propositions 3.1 and 3.2 when deriving them.

The matrices are now used to construct the linearized version of the parameterized throughput graph. The structure is shown in Fig. 5.9 where the critical cycle is depicted using bold arrows. From the throughput graph we can determine (bound) the linearized MCME for $\tilde{\mathbf{x}}_r$. In particular, $\tilde{\mathcal{M}}_{\mathbf{x}_r} \leq \frac{1}{2}\cdot(X_{11}+x_1+14)$. In the next step of the divide & conquer one needs to determine the throughput of the corner points $\tilde{\mathbf{x}}_1 = (x_1 = 1, X_{11} = 1)$,

$\tilde{\mathbf{x}}_2 = (x_1 = 3, X_{11} = 5)$ and $\tilde{\mathbf{x}}_3 = (x_1 = 5, X_{11} = 25)$ of $\tilde{X}_{\mathsf{FP}}$ and check whether its inverse is equal to the $\tilde{\mathcal{M}}_{\mathbf{x}_r}$ evaluated for each particular corner point, i.e. $\tilde{\mathcal{M}}_{\mathbf{x}_r}(\mathbf{x}_i)$, where $i = 1, \ldots, 3$.

Continuing the divide & conquer we obtain the following three MCMEs valid in three different throughput regions $r_1, r_2, r_3 \subseteq \tilde{X}_{\mathsf{FP}}$

$$\begin{aligned}
&\mathbf{M}_{\tilde{\mathbf{x}} \in r_1} \leq x_1 + 10 \quad \text{for} \quad \tilde{\mathbf{x}} \in r_1, \\
&\mathbf{M}_{\tilde{\mathbf{x}} \in r_2} \leq \frac{1}{2} \cdot (X_{11} + x_1 + 14) \quad \text{for} \quad \tilde{\mathbf{x}} \in r_2 \quad \text{and} \\
&\mathbf{M}_{\tilde{\mathbf{x}} \in r_3} \leq X_{11} \quad \text{for} \quad \tilde{\mathbf{x}} \in r_3.
\end{aligned} \tag{5.64}$$

Recall that (5.64) defines conservative upper bounds on MCMEs as the scenario matrices we used to obtain them are conservative estimates themselves. The larger the repetition vector entries of the scenario graphs are, the smaller the relative approximation error is. Throughput regions r_1, r_2 and r_3 are depicted in Fig. 5.8 and are defined by the following two hyperplanes that divide the initial linearized domain: $\mathbf{M}_{\tilde{\mathbf{x}} \in r_1} - \mathbf{M}_{\tilde{\mathbf{x}} \in r_2} = 0$ and $\mathbf{M}_{\tilde{\mathbf{x}} \in r_3} - \mathbf{M}_{\tilde{\mathbf{x}} \in r_2} = 0$. Throughput expressions are merely the inverses of MCMEs. Note that both are expressed as functions of $\tilde{\mathbf{x}} \in \tilde{X}_{\mathsf{FP}}$. However, the designer is interested only in feasible configurations, that is the configurations $\mathbf{x} \in X_{\mathsf{FP}}$. Therefore, for a given configuration $\mathbf{x} \in X_{\mathsf{FP}}$, we need to compute $\tilde{\mathbf{x}} \in \tilde{X}_{\mathsf{FP}}$ using the $\tilde{X}_{\mathsf{FP}}$ metadata that maps $\mathbf{x}$ to $\tilde{\mathbf{x}}$ using the relationship between the original parameters and the lifting parameters (cf. (5.52)). E.g. for $\mathbf{x} = \{x_1 = 4\}$, via (5.57), $\tilde{\mathbf{x}} = \{x_1 = 4, X_{11} = 16\}$ which is a feasible linearized configuration. Configuration $\tilde{\mathbf{x}}$ belongs to r_2. Using (5.64) we obtain the conservative throughput estimate of $\mathbf{Th}_{\tilde{\mathbf{x}} \in r_2}(\tilde{\mathbf{x}}) \geq \frac{1}{17} = 0.059$ iterations per time unit. If we would apply the analysis of Chapter 4 to the graph instantiated from the configuration $\mathbf{x} = \{x_1 = 4\}$ we would, of course, obtain the same result.

5.6.2 Tightness of performance bound and technical aspects of the analysis

The parametric throughput analysis presented in this chapter is conservative and incurs an over-approximation to the degree incurred by the Max-plus algebraic characterization of SDF-PDFGs of Chapter 3. This is because Algorithm 5.5 is nothing but a specialized version of Algorithm 3.1 that entails conservativeness by employing Propositions (3.1) and 3.2 to derive parameterized scenario SDF-PDFG matrices. Recall that these matrices we later on compose into the parameterized throughput graph from which we derive the associated throughput expression(s). Still, the relative error shrinks with the growing repetition vectors of the scenario graphs (cf. Subsection 3.5.4).

The analysis was performed manually on the synthetic case study application as presented above. Assuming that the analysis of Chapter 3 is automated, the automation of the analysis of this chapter would consist of the implementation of the polynomization (cf. Subsection 5.5.3) and linearization (cf. Subsection 5.5.4) techniques presented on top of Algorithm 3.1, i.e. Algorithm 5.5. Furthermore, polyhedra manipulation techniques must be foreseen to implement the functionality of the divide & conquer technique (cf. Algorithm 5.3). In the mere technical sense, this would entail the integration of a symbolic computation framework as GiNaC [10][11] and a polyhedra manipulation library such as `cddlib` [39] with the SDF^3 tool. A conservative estimate of the development time needed to accomplish this would amount up to a few months.

5.7 Summary

In this chapter we have proposed a parametric throughput analysis technique for SDF-PFSM-SADFGs with static parameters that generalizes the results of [33] in a twofold manner. First, the analysis of [33] does not allow for parameterized rates. Our analysis supports parameterized rates. That is our first contribution. Second, the analysis of [33] requires that firing delays are expressed as linear combinations of parameters. In our work we allow a more general polynomial representation. That is our second contribution. We base our approach on the theory of Max-plus automata applied after problem linearization via RLT that creates a linear relaxation of the original problem's feasible region.

Chapter 6

Model checking of FSM-SADF using timed automata

In earlier chapters, among others, we presented (worst-case) performance analysis techniques for a parameterized generalization of FSM-SADF, i.e. SDF-PFSM-SADF. The analysis is based on the Max-plus algebraic semantics of SDF-PFSM-SADF that it inherits from FSM-SADF. However, this this type of analysis only treats temporal properties of FSM-SADF that can be defined at the graph iteration level and provides no insight into intra-iteration behavior of the graph.

Nevertheless, analysis exists for FSM-SADF that can consider other properties than available via Max-plus-based analysis. In particular, this analysis is based on specialized extensions of techniques used in conventional model checking and is implemented in the SDF^3 tool. However, this analysis is limited to only a predefined set of properties that the user cannot influence. Therefore, with the goal of providing means for enabling the user to easily expand the analysis portfolio of FSM-SADF, in this chapter we report on the translation of the FSM-SADF formalism to UPPAAL timed automata that enables a more general verification than currently supported by existing tools. We base our translation on a compositional approach where the input FSM-SADF model is represented as a parallel composition of its integral components modeled as automata. Thereafter, we show how to model check quantitative and qualitative properties both supported and not supported by the existing tools. We demonstrate our approach on a realistic case study from the multimedia domain.

Different parts of this chapter have been published in [104][103].

6.1 Introduction

The SADF formalism [113] captures application dynamism using *scenarios*. Scenarios represent distinct application operating modes that occur during its lifetime. Rates and actor firing delays differ from one scenario to the other. The data dependent conditions that determine scenario occurrence patterns are abstracted into Markov chains. SADF to a large extent preserves SDF's compile-time analyzability [110]. State-of-the-art SADF techniques are implemented in the SDF^3 tool [112] as proposed by [116] and are based on specialized extensions of techniques used in conventional model checking.

FSM-SADF [111][46] is a restricted form of SADF introduced to speed-up the analysis of the original formalism. FSM-SADF has been used in several important modeling [97][96][32] and optimization contexts [32]. It is a restricted form because unlike SADF, it does not support hierarchical control through the use of nested Markov chains. This means that FSM-SADF cannot support sub-scenarios, i.e. in FSM-SADF actor rates and firing delays can change only at scenario boundaries while in SADF, they can change even within a scenario. In addition, FSM-SADF uses fixed actor firing delays per scenario, while SADF uses discrete distributions per scenario. Furthermore, unlike the probabilistic abstraction approach of SADF, FSM-SADF uses a non-deterministic abstraction where scenario sequences are specified by a non-deterministic FSM. These restrictions render the FSM-SADF analysis faster than the analysis of SADF. In addition, FSM-SADF outcompetes SADF in terms of implementation efficiency [110]. However, the analysis might be less precise due to the abstractions made. On the other hand, FSM-SADF extends SADF by allowing actor-level auto-concurrency (simultaneous executions of a particular actor) which is explicitly prohibited in SADF as it may violate the determinacy of the model [113]. Thus, the parallelism embedded in an FSM-SADF specification is implicitly greater than the one embedded in an SADF specification. Note that, in essence, auto-concurrency in FSM-SADF can cause determinacy related problems as in SADF, but unlike SADF, FSM-SADF specifies polices to reinstate it. In particular, we mean the FIFO policy closely related to its Max-plus semantics introduced in [46] that we will elaborate soon enough. State-of-the art FSM-SADF techniques are implemented in the SDF^3 tool [112]. In particular, me mean Max-plus algebra-based techniques addressing the worst-case throughput and latency computation. With these techniques, as shown in previous chapters, the analysis is done compositionally per scenario. By using compositional analysis, we avoid the state-space size induced problems that occasionally occur with the model-checking techniques of SADF. However, all comes with a price. In particular, the Max-plus algebraic apparatus of FSM-SADF proposed by [46] is able to address only temporal properties of FSM-SADF defined at iteration granularity (cf. Definitions 4.4 and 4.5). If we wish to analyze other temporal properties concerning intra-iteration behavior of actors or functional properties in general, we must use the model-checking engine of SADF. However, the model checker of SADF explicitly excludes auto-concurrency and therefore we are limited to consideration only of FSM-SADF models that exclude auto-concurrency too.

In addition, tools such as SDF^3 can be too specialized in the sense that they can only handle predefined properties, thus lacking support for analyzing user-defined properties. To circumvent this limitation, in this chapter we propose a translation of the FSM-SADF formalism to timed automata (TA) [6] as the first step to enable more general verification. This is our first contribution. Using TA has a number of advantages, in that very efficient abstractions exist. For example, temporal logics can express many of the properties common in reasoning about timed systems with concurrency. Furthermore, TA models of dataflow specifications can be easily extended to add costs such as energy, and include the behavior of the underlying implementation platform. This would in the future give us the possibility of using FSM-SADF for reachability analysis of embedded dynamic streaming applications through an optimal control formulation using model-checking techniques. Although other members of the SADF MoC family have been translated to model checkers before [114, 115, 66] our translation is (to the best of our knowledge) the first one that allows auto-concurrency in the model and is able to ensure determinacy in its presence, by using the scenario-level FIFO policy of [46] in a model checking context. This is our second contribution. Auto-concurrency is important as it allows the designer to embed more parallelism in the specification. Still, auto-concurrency can cause determinacy issues and that is why existing translations [114, 115, 66] pragmatically decide to prohibit it. Unfortunately, as discussed in Section 6.7, auto-concurrency has a negative effect on the scalability of the analysis.

We demonstrate our approach using a multimedia case study modeled as an FSM-SADFG for which we compute important quantitative and qualitative properties, some of which are not supported by the SDF^3 tool. We use the UPPAAL [14] state-of-the-art TA model checker.

6.2 Related work

The SADF model has already been subjected to model checking in [114], which discusses a performance model-checking approach for SADF where its semantics is based on a timed probabilistic (labeled transition) system (TPS). In the TPS of SADF nondeterminism can be arbitrarily resolved as it originates from concurrency of the actors, i.e. the ordering of timeless actions will not affect the net behavior of the system. We call this property of SADF *action determinacy* (or the diamond-property in a non-probabilistic setting). This property is the key to efficient analysis implemented in SDF^3 [112]. However, SDF^3 is not a general model-checker and it supports only a set of predefined properties such as deadlock freedom, maximum buffer occupancy, inter-firing latency, etc.

Theelen et al. [115] report on the use of the construction and analysis of distributed processes (CADP) tool suite in model checking of SADF using interactive Markov chains (IMC). The inability of IMC in supporting probabilistic choices is compensated by CADP. Also, IMC relies on exponentially distributed time, and therefore the original discrete distribution of SADF needed to be replaced by a single exponential distribution. Therefore, CADP may not always deliver the same result as SDF^3. Moreover, CADP is unable to evaluate reward-based properties and therefore cannot be used for the computation of throughput and buffer occupancy.

The work of Katoen et al. [66] extends the framework of [115] by introducing Markov automata (MA) based semantics of SADF. MA is a combination of probabilistic automata and Markov decision processes. The firing delays of actors are specified by negative exponential distributions. State-space reductions are based on confluence reduction which utilizes action determinacy for SADF actors. The approach can be used to obtain quantitative properties such as buffer occupancy, latency and throughput.

What is common to the aforementioned approaches is that they all consider SADF. In FSM-SADF on the other hand, non-determinism is an explicit property of the model, and not a side-effect of concurrency. In addition, FSM-SADF allows actor-level auto-concurrency which is explicitly prohibited in SADF. Geilen et al. [46] introduce the Max-plus algebraic semantics of FSM-SADF that can be used to obtain worst-case throughput and latency values in the presence of auto-concurrency. However, due to the nature of the Max-plus representation of a scenario, the approach of [46] can only give insight into the model's temporal behavior at scenario (iteration) boundaries. Therefore, the analysis of [46] is limited to throughput and latency computations only.

Fakih et al. [37] and Ahmad et al. [3] have previously used TA to model SDF, the original synchronous dataflow formalism. However, the TA model of SDF cannot be used to capture FSM-SADF as FSM-SADF unlike SDF includes a combination of streaming data and control. Moreover, the works of [37][3] are more concerned with modelling lower-level details of the scheduling on a given execution platform.

6.3 Definition of FSM-SADF

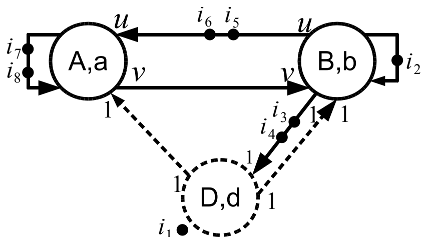
We first redefine the concept of FSM-SADF of Definition 2.6 of Chapter 2. The presentation of FSM-SADF (of that definition) as a collection of SDFG-modeled scenarios served very well as the cornerstone of the Max-plus-based compositional analysis defined by [46] (cf. Section 2.4.2). However, in the model-checking context of this chapter it is convenient to establish an equivalent but a more (traditional) operational semantics centered definition of FSM-SADF as initially done in [111].

Therefore, we proceed by defining our dialect of the FSM-SADF formalism in terms of a network of processes of certain properties. Thereafter, we define its operational semantics that forms the basis for the UPPAAL translation by building on the work of [111] and [113]. But before getting all too formal, we gently introduce the concept using an example.

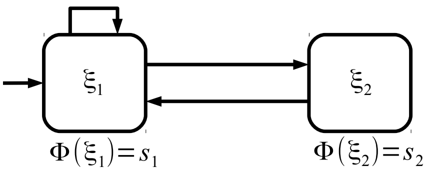
6.3.1 Example FSM-SADFG

Consider the FSM-SADFG of Fig. 6.1. In FSM-SADF, two types of processes[1] can be distinguished: *kernels* and *detectors* [111][113]. Kernels specify the data processing part of the application, while detectors model the control part of the application. The scenario graph of Fig. 6.1a consists of three vertices representing processes. Processes A and B are kernels (continuous lines) while process D (dashed line) serves as a detector. In FSM-SADF there can only exist one detector, i.e. the control is global. D determines in which scenario kernels A and B operate by sending them *control tokens* via control channels (dashed lines). Control tokens are valued. Tokens exchanged over data channels (continuous lines) are called *data tokens* and we abstract from their values. Channels are considered as FIFO buffers of infinite capacity. We use the terms channel and buffer interchangeably. The graph has 8 initial tokens labeled $i_1, \ldots, i_8$. Token i_1 is the initial token of the detector's implicit self-edge (channel with the

[1]Process is just another name for an actor and we use the terms interchangeably.



(a) Scenario graph



(b) Scenario FSM

	u	v
Scenario s_1	1	1
Scenario s_2	1	2

(c) Rates in scenarios s_1 and s_2

	a	b	d
Scenario s_1	4	1	1
Scenario s_2	2	5	1

(d) Actor firing delays in scenarios s_1 and s_2

Fig. 6.1: Example FSM-SADF graph

same source and destination actor). This edge is usually not drawn as FSM-SADF prohibits auto-concurrency for the detector. The running example defines two scenarios: s_1 and s_2. Depending on the operating scenario, the graph will change its properties. Fig. 6.1c specifies changes for graph rates over scenarios, while Fig. 6.1d specifies how process firing delays change over scenarios. The running example defines the FSM of Fig. 6.1b that determines the possible scenario occurrences where every FSM state is labeled with a scenario: state ξ_1 is labeled with s_1 and ξ_2 is labeled with s_2.

6.3.2 FSM-SADF redefined

Our full definition of FSM-SADF in the context of its traditional operational semantics follows next. Compared with the one that can be found in [111], it is more concise because it is not necessary to represent sets of detectors and ports explicitly.

Definition 6.1 (FSM-SADF graph). *An FSM-SADF graph is a tuple* $\mathsf{F} = (\mathcal{S}, \mathcal{K}, \mathsf{B}, E, R_p, R_c, \Xi, \mathbb{T}, \xi_0, \Phi, t, \phi_\iota, \psi_\iota)$*, where*

1. *$\mathcal{S}$ is the nonempty finite set of scenarios,*
2. *$\mathcal{K}$ is the nonempty finite set of kernels,*
   - $\mathsf{P} = \mathcal{K} \cup \{d\}$*, where $d \notin \mathcal{K}$ denotes the unique detector, is the set of processes,*
3. $\mathsf{B} \subseteq \mathcal{K} \times \mathsf{P}$ *is the set of buffers,*

4. *$E : \mathsf{P} \times \mathcal{S} \to \mathbb{N}_0$ is the firing delay for each process in each scenario,*
5. *$R_p, R_c : \mathsf{B} \times \mathcal{S} \to \mathbb{N}_0$ is the production (consumption) rate of the kernel producing to (process consuming from) each buffer in each scenario,*
6. *$(\Xi, \xi_0, \mathbb{T}, \Phi)$ is the FSM of the detector, where Ξ is the nonempty set of states, $\xi_0 \in \Xi$ is the initial state, $\mathbb{T} : \Xi \to 2^{\Xi}$ is the transition function, and $\Phi : \Xi \to \mathcal{S}$ associates each state with a scenario,*
7. *$t : \mathcal{K} \times \mathcal{S} \to \mathcal{S}^+$ is the string of scenarios sent to the FIFO of each kernel in each scenario of the detector,*
8. *$\phi_\iota : \mathsf{B} \to \mathbb{N}_0$ is the initial buffer status,*
9. *$\psi_\iota : \mathcal{K} \to \mathcal{S}^*$ is the initial control status.*

The detector is connected to every kernel by an explicitly ordered (FIFO) control channel. We further define $In(p) = \{b \in \mathsf{B} \mid \pi_r(b) = p\}$, where π_r is the right projection function, to be the set of buffers that process p consumes from (that input into p). Similarly, $Out(k) = \{b \in \mathsf{B} \mid \pi_l(b) = k\}$.

In anticipation of the next section we define $\varnothing$ to be the empty multiset, $\mathbb{P}$ to be the set of all submultisets of its input set, $\uplus$ to be the multiset sum, and $\setminus$ to be the zero-truncated asymmetric multiset difference. For example let $A = \{1, 1\}$ and $B = \{1, 2\}$. Then $A \cup B = \{1, 1, 2\}$ (maxima of multiplicities), $A \uplus B = \{1, 1, 1, 2\}$ (sums of multiplicities), $A \setminus B = \{1\}$, and $B \setminus A = \{2\}$. For strings $\sigma, \tau, \nu \in \mathcal{S}^*$ we define σ_i to be the ith element of σ, $\sigma + \tau$ to be the concatenation of σ and τ, and, if $\nu = \sigma + \tau$, then $\nu - \sigma = \tau$.

6.3.3 Operational semantics

The behavior of an FSM-SADF graph is defined as a transition system where states are configurations.

Definition 6.2 (Configuration). *A configuration of an FSM-SADF graph $\mathsf{F} = (\mathcal{S}, \mathcal{K}, \mathsf{B}, E, R_p, R_c, \Xi, \mathbb{T}, \xi_0, \Phi, t, \phi_\iota, \psi_\iota)$ is a tuple $(\phi, \psi, \kappa, \delta)$, where ϕ is a buffer status, ψ a control status, κ a kernel status, and δ a detector status:*

- *A buffer status is a function $\phi : \mathsf{B} \to \mathbb{N}_0$ from each buffer to the number of tokens it stores,*
- *A control status is a function $\psi : \mathcal{K} \to \mathcal{S}^*$ from each kernel to the string of scenarios (control tokens) its FIFO stores,*

- *A kernel status is a function $\kappa : \mathcal{K} \to \mathbb{P}(\mathcal{S} \times \mathbb{N}_0)$ that to each kernel assigns a multiset of ongoing firings and their remaining execution times,*
- *A detector status is a pair $\delta \in \Xi \times (\mathbb{N}_0 \cup \{-\})$ that represents the state of the FSM and the remaining execution time of the ongoing firing, or, if there is no ongoing firing, the value $-$.*

The initial configuration of F is $(\phi_\iota, \psi_\iota, \kappa_\iota, \delta_\iota)$, where ϕ_ι and ψ_ι are defined in F, $\kappa_\iota = \mathcal{K} \times \{\varnothing\}$ and $\delta_\iota = (\iota, -)$.

Five types of configuration transitions are distinguished.

Definition 6.3 (Kernel Start Action)**.** *A kernel start action transition $(\phi, \psi, \kappa, \delta) \xrightarrow{\text{start}(k)} (\phi', \psi', \kappa', \delta)$ represents the start of a firing of kernel k. Let $s = \psi(k)_1$ denote the scenario of the firing (if it is defined). The transition is enabled if $|\psi(k)| \geq 1$ and $\forall b \in In(k) : \phi(b) \geq R_c(b, s)$. The resulting statuses are defined as*

$$\begin{aligned}
\phi' &= \phi[b \mapsto \phi(b) - R_c(b, s)] \quad \textit{for all } b \in In(k) \\
\psi' &= \psi[k \mapsto \psi(k) - s] \\
\kappa' &= \kappa[k \mapsto \kappa(k) \uplus \{(s, E(k, s))\}]
\end{aligned}$$

Definition 6.4 (Kernel End Action)**.** *A kernel end action transition $(\phi, \psi, \kappa, \delta) \xrightarrow{\text{end}(k)} (\phi', \psi, \kappa', \delta)$ is the end of a firing of kernel k. It is enabled if $\exists s \in \mathcal{S} : (s, 0) \in \kappa(k)$. The resulting buffer and kernel statuses are*

$$\begin{aligned}
\phi' &= \phi[b \mapsto \phi(b) + R_p(b, s)] \quad \textit{for all } b \in Out(k) \\
\kappa' &= \kappa[k \mapsto \kappa(k) \setminus \{(s, 0)\}]
\end{aligned}$$

Definition 6.5 (Detector Start Action)**.** *A detector start action transition $(\phi, \psi, \kappa, \delta) \xrightarrow{\text{start}(d)} (\phi', \psi, \kappa, \delta')$ represents the start of a firing of the detector, d. It is enabled if there is no ongoing firing $\exists s \in \Xi : \delta = (s, -)$ and all inputs are available $\forall b \in In(d) : \phi(b) \geq R_c(b, \Phi(s))$. The resulting statuses are*

$$\begin{aligned}
\phi' &= \phi[b \mapsto \phi(b) - R_c(b, \Phi(s))] \quad \textit{for all } b \in In(d) \\
\delta' &= (s, E(d, \Phi(s)))
\end{aligned}$$

Definition 6.6 (Detector End Action)**.** *A detector end action transition $(\phi, \psi, \kappa, \delta) \xrightarrow{\text{end}(d)} (\phi, \psi', \kappa, \delta')$ is enabled if $\exists s \in \Xi : \delta = (s, 0)$, and the resulting statuses are*

$$\begin{aligned}
\psi' &= \psi[k \mapsto \psi(k) + t(k, \Phi(s))] \quad \textit{for all } k \in \mathcal{K} \\
\delta' &= (s', -) \quad \textit{for some } s' \in \mathbb{T}(s)
\end{aligned}$$

In [111] time transitions are defined very generally, such that to account for given scheduling/resource constraints one needs to instantiate the time transitions needed. In the following we will assume a unconstrained execution, namely that all ongoing firings advance at the same pace.

Definition 6.7 (Time Transition). *A time transition* $(\phi, \psi, \kappa, \delta) \xrightarrow{\text{time}(t)} (\phi, \psi, \kappa', \delta')$ *represents time progressing* t *time units. It is enabled if no kernel end or detector end transition is enabled, and* t *is the smallest remaining execution time of any ongoing firing. The resulting kernel status is*

$$\kappa' = \kappa[k \mapsto \{(s, n-t) | (s,n) \in \kappa(k)\}] \quad \textit{for all } k \in \mathcal{K}$$

using multiset comprehension. The detector status $\delta = (s, n)$ *is updated as* $\delta' = (s, n-t)$*, unless* $n = -$ *in which case it is unchanged,* $\delta' = \delta$.

6.3.4 Overtaking problem and determinacy

The operational semantics of Section 6.3.3 allows simultaneous executions of a particular kernel, i.e. auto-concurrency. Note that the kernel status of Definition 6.2 entails a multiset of ongoing firings. In the case of the detector, auto-concurrency is prohibited as its status entails only one possible ongoing firing. With auto-concurrency and due to the potential difference in kernel execution times over different scenarios, tokens may "overtake" each other which makes it hard to ensure determinacy [113].

6.3.4.1 Overtaking by example

Let us illustrate this using the example FSM-SADFG of Fig. 6.1 and the concept of token sequence of Definition 2.1. For data token sequences we specify the scenario in which they were produced by defining V of Definition 2.1 as $V = \{*\} \times \mathcal{S}$. For control token sequences, $V = \mathcal{S}$.

E.g., $[((*, s), t)]$ denotes a sequence containing one data token of arbitrary value $(*)$ produced in scenario s at t time-units. We consider the execution of the FSM-SADFG of Fig. 6.1 from $t = 0$. In this case $V = \{*\} \times \{s_1, s_2\}$ for data token sequences and $V = \{s_1, s_2\}$ for control token sequences. At $t = 0$, the only enabled process is the detector D. Assume that, by firing D, the FSM makes a transition from the initial state ξ_1 to state ξ_2. State ξ_1 corresponds to scenario s_1, and ξ_2 corresponds to s_2. After 1 time-unit, the control channels (D, A) and (D, B) host the token sequence $[(s_1, 1)]$. Channel (B, D) now hosts $[((*, \perp), 0)]$ which refers to the initial token i_3 as i_4 was just consumed by D firing. We use the $\perp$ notation to leave the scenario value unspecified as we do not know in which scenario initial tokens were produced. Now A can start firing by consuming the control token from channel (D, A), initial token i_8 from its self-edge and initial token i_6 from channel (B, A). The new status of (D, A) becomes $[]$ (empty token sequence) and the new status of (B, A) becomes $[(*, \perp), 0)]$ that refers to initial token i_5. As there is still one initial token left on channel (B, D), D can perform its second firing. This results in channel (D, A) hosting $[(s_2, 2)]$, channel (D, B) hosting $[(s_1, 1), (s_2, 2)]$ and channel (B, D) hosting empty token sequence $[]$ with the assumption that the FSM transition (ξ_2, ξ_1) was taken. As A is auto-concurrent it can commence its second firing, but this time in scenario s_2 while still being busy with the first one in scenario s_1. Due to the difference in firing delays of A in scenarios s_1 and s_2, the firing of scenario s_2 started at $t = 2$ will finish earlier than the earlier firing of scenario s_1 started at $t = 1$. Therefore, at $t = 4$ the channel (A, B) will host the sequence $[((*, s_2), 4), ((*, s_2), 4)]$ as the firing delay of A in scenario s_2 equals to 2 time-units. On the other hand channel (D, B) hosts the sequence $[(s_1, 1), (s_2, 2)]$. Kernel B will therefore commence firing in scenario s_1 but by consuming the token $((*, s_2), 4)$ produced by A in scenario s_2 and will not wait for the "right" scenario s_1 token that will be produced at $t = 5$. After B had completed this firing, the status of (B, B) becomes $[((*, s_1), 5)]$.

6.3.4.2 Reinstating determinacy

The "overtaking" phenomenon just discussed results in tokens being consumed in another scenario than the one they were produced in. This makes it hard to ensure determinacy [113]. It is generally the modeler's responsibility to ensure that the model does not exhibit overtaking or that overtaking can be properly interpreted, i.e. that it does not invalidate the functional correctness of the system. It could however be handled with a policy in the semantics that ensures that tokens are only consumed by a kernel in the scenario they belong to.

In our work, we adopt the scenario-level FIFO policy of [46] inherent to the Max-plus semantics of FSM-SADF introduced in the same paper and used throughout this thesis in earlier chapters. Recall, that one can view the execution of the FSM-SADFG of Fig. 6.1 as the execution of a sequence of SDF graphs obtained by applying the configurations of Fig. 6.1c and Fig. 6.1d to the scenario graph of Fig. 6.1a. By doing so, one avoids the overtaking problem as in a single scenario there can be no overtaking thanks to the static nature of SDF. At a later stage, scenarios are "glued" together and synchronized by the set of initial tokens. This can be done as initial tokens produced at the end of a scenario contain enough information

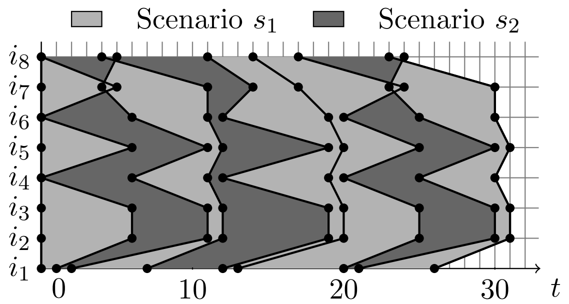


Fig. 6.2: Execution of the FSM-SADFG of Fig. 6.1

to determine the timing of the next scenario [46]. We assume self-timed execution. Fig. 6.2 shows the pipelined (self-timed) execution of the FSM-SADFG of Fig. 6.1. The start time of a scenario is determined by the reproduction times of initial tokens in the previous scenario. In Fig. 6.2, we see overtaking along the axes of availability times of tokens i_7 and i_8. E.g., we see that i_7 of scenario s_2 is actually produced (at $t = 4$) before i_7 of scenario s_1 (at $t = 5$), even though scenario s_1 was started first. In spite of this, the firing of B in scenario s_1 will be initialized by the availability of i_7 belonging to s_1 and not i_7 belonging to s_2 although it is produced first. Therefore, i_2 will be produced at $t = 6$ and not at $t = 5$ as discussed earlier. This way, determinacy is ensured.

To introduce the Max-plus scenario-level FIFO policy to the semantics of Section 6.3.3 one has to find a way to decouple scenarios. This could be modeled by data channels having a separate buffer for each scenario in the system and kernels only writing to and consuming from buffers belonging to the scenario they are currently operating in. For control channels such replication is not necessary as the detector is by definition "sequential", i.e. non auto-concurrent. With this concept of "scenario buffers", no overtaking between different scenarios is possible.

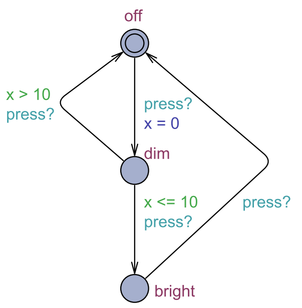
However, an interesting question emerges. In which scenario were the initial tokens produced? If we treat initial tokens as a special class (no scenario buffer affiliation) we will easily violate the functional correctness of the system, e.g. introduce a deadlock. In the scenario graph of Fig. 6.1 imagine that the detector has fired twice by consuming initial tokens i_3 and i_4 following the path $\xi_1 \to \xi_1 \to \xi_1$ of the FSM. This means that the graph has executed the scenario sequence $s_1 s_1$. To complete the sequence, actor B has fired twice so channel (B, D) hosts the sequence $[((*, s_1), t_1), ((*, s_1), t_2)]$. Now, if the transition $\xi_1 \to \xi_2$ of the FSM is to be taken, the resulting (B, D)

channel state after the completion of the scenario will be $[((*, s_1), t_2), ((*, s_1), t_3)]$. Being in state ξ_2, the FSM can only perform the transition $\xi_2 \rightarrow \xi_1$. By the FIFO policy, to do that tokens belonging to s_2 need to be consumed. However, channel (B, D) only hosts tokens belonging to s_1. Therefore, a deadlock occurs.

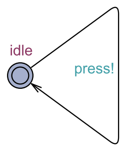
Another approach might be to force initial token scenario affiliation. However, this approach would also violate the functional correctness of the model by introducing a deadlock or by restricting the language the scenario FSM accepts.

As it is not clear how to deal with initial tokens belonging to channels on which overtaking can take place, we only allow overtaking on channels with no initial tokens, i.e. auto-concurrent actors can only produce in data buffers that are initially empty. This excludes actor self-edges as these are used to limit actor's auto-concurrency by assigning them an appropriate number of initial tokens. Actually, in the UPPAAL model of FSM-SADF of Section 6.4 they will be modeled as the number of instances of a particular actor in the system. Consequentially, we only replicate data buffers that are filled by auto-concurrent actors and are initially empty. Under this restriction on the structure of the input FSM-SADF specification, the FIFO policy can be straightforwardly encoded into the semantics of Section 6.3.3 by changing the definition of the set of buffers of Definition 6.1 to $\mathsf{B} \subseteq \mathcal{K} \times \mathsf{P} \times (\mathcal{S} \cup \mathbf{s})$ where $\mathbf{s}$ is the "default scenario", which marks the buffer used when there is no overtaking on the channel, i.e. the default buffer. Also, we redefine $In(p, s) = \{b \in \mathsf{B} \mid \pi_2(b) = p \wedge (\pi_3(b) = s \text{ when } \omega(b) = 1\,;\, \mathbf{s} \text{ otherwise})\}$, where π_2 and π_3 are the 2nd and 3rd projection function, respectively and $\omega : \mathsf{B} \rightarrow \{0, 1\}$ is the function returning the information whether overtaking is possible on the channel implemented by buffer b. Similarly, $Out(k, s) = \{b \in \mathsf{B} \mid \pi_1(b) = p \wedge (\pi_3(b) = s \text{ when } \omega(b) = 1\,;\, \mathbf{s} \text{ otherwise})\}$, where π_1 is the 1st projection function. Last, kernel start and end actions must be altered to fit the new definitions. E.g. the kernel start action transition is enabled if $|\psi(k)| \geq 1$ and $\forall b \in In(k, s) : \phi(b) \geq R_c(b, s)$.

Another option is to simply disallow auto-concurrency as it was done in works on model checking SADF [113][114][115][66]. In this case, the notion of a multiset of ongoing kernel firings in the kernel status of Definition 6.2 has to be changed so each kernel can have zero or one ongoing firings: $\kappa : \mathcal{K} \rightarrow (\mathcal{S} \times \mathbb{N}_0) \cup \{-\}$. To reflect this re-definition, the kernel start and end actions have to be re-adjusted in the spirit of detector start and end actions (recall that the detector is "sequential" by definition).



(a) Light



(b) User

Fig. 6.3: Network of timed automata in UPPAAL

6.4 Translation of FSM-SADF to Timed Automata

To be able to model check an FSM-SADF specification, we encode the operational semantics of FSM-SADF in the UPPAAL model checker. The correctness of the translation follows from the construction itself as explained in the remainder of this section. We limit our attention to self-timed bounded FSM-SADF specifications [110].

In UPPAAL, a system is modeled as a network of TA that is extended with bounded discrete variables that are part of the state. These variables can be read, written and are subject to common arithmetic operations.

We recall the definition of TA where we use $\mathcal{B}(\mathcal{C})$ to denote the set of constraints defined over a finite set of real-valued variables $\mathcal{C}$ called *clocks* and where $Act = \{a!, a?, \ldots\}$ is a finite alphabet of synchronization actions.

Definition 6.8 (Timed automaton (TA)). *A timed automaton $\mathbb{A}$ is a tuple (L, l^0, E, I), where L is a finite set of locations (nodes), l^0 is the initial location, $E \subseteq L \times \mathcal{B}(\mathcal{C}) \times Act \times 2^{\mathcal{C}} \times L$ is the set of edges between locations with a guard, an action and a set of clocks to be reset, and $I : L \to \mathcal{B}(\mathcal{C})$ assigns invariants to locations. We shall write $l \xrightarrow{g,a,r} l'$ when $(l, g, a, r, l') \in E$.*

A state of the system modeled in UPPAAL is defined by the locations of all automata, the clock values, and the values of the discrete variables. Every automaton may fire an edge (sometimes misleadingly called a transition) separately or synchronize with another automaton, which leads to a new state [14]. An example of a network of timed automata is shown in Fig. 6.3.

In the graphical representation of the TA, edges are decorated with labels. The green labels are used to specify guards, update labels are violet, while labels concerning channel synchronization are gray. Guards and update labels operate on clocks as well as on discrete variables. Locations can entail invariants that are denoted in pink color.

The network of Fig. 6.3 models a time-dependent light-switch (Fig. 6.3a) and its user (Fig. 6.3b). The switch and the user communicate using the `press` labels (channel). The user can press the switch (`press!`) randomly at any time or even not press the switch at all. The switch waits to be pressed (`press?`). If the user presses the switch, the light is on, but dimmed (location **dim**). If the user presses the switch again, but after more than 10 time-units (guard `x > 10`), the light is off (location **off**). If the user presses the switch within 10 time-units (guard `x <= 10`) the light is brightened (location **bright**). At this point, whenever the user presses the switch, the light will turn off (location **off**).

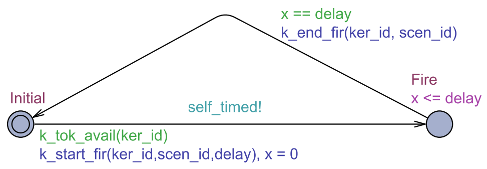
The FSM-SADF configuration of Definition 6.2 is modeled so that the kernel and detector statuses are encoded in the states of the TA, while the buffer and control statuses are modeled explicitly using discrete variables. These discrete variables are read and written during kernel/detector start/end actions. Operations performed on discrete variables correspond to checking the availability of input tokens, token consumption and token production. Discrete variables do not add to the expressive power of the formalism, and for presentation purposes, we do not encode their use in the TA edge firings.

Given an FSM-SADF F, we generate a parallel composition of TA

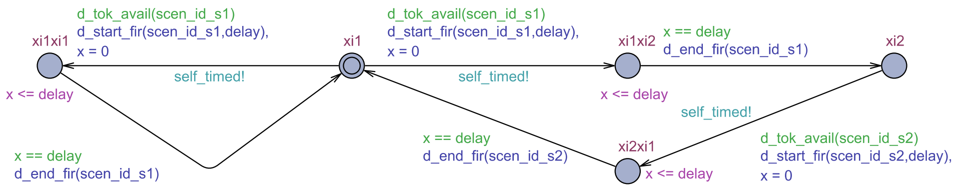
$$\mathit{System} = \mathbb{A}_{k_1}^{\|\upsilon(k_1)} \parallel \ldots \parallel \mathbb{A}_{k_n}^{\|\upsilon(k_n)} \parallel \mathbb{A}_d, \tag{6.1}$$

where $k_i \in \mathcal{K}$ and $n = |\mathcal{K}|$. By $\mathbb{A}_{k_i}^{\|\upsilon(k_i)}$ we denote the fact that $\upsilon(k_i)$ TA in parallel are used to model kernel k_i. Function $\upsilon : \mathcal{K} \to \mathbb{N}_{>0}$ gives the realized auto-concurrency of a kernel. If the kernel k_i has a self-edge, i.e. $(k_i, k_i) \in \mathsf{B}$, then $\upsilon(k_i) = \phi_\iota((k_i, k_i))$. If kernel k_i has no self-edge, $\upsilon(k_i)$ can be found experimentally. We assume some N_{k_i}, then we need to verify that the actual $\upsilon(k_i)$ is strictly smaller, i.e., $\upsilon(k_i) < N_{k_i}$. This is discussed in Section 6.5.

Fig. 6.4 shows the UPPAAL model of the FSM-SADFG of Fig. 6.1. In the description language of UPPAAL, processes are obtained as instances of parametrized process templates. In our translation we define two templates: The kernel template of Fig. 6.4a and the detector template of Fig. 6.4b. The kernel template is generic, while the detector template is customized to correspond to the FSM of the input FSM-SADF specification. Note that control buffers are implemented as FIFOs where the values of FIFO elements


x == delay
k_end_fir(ker_id, scen_id)
Initial
Fire
x <= delay
self_timed!
k_tok_avail(ker_id)
k_start_fir(ker_id,scen_id,delay), x = 0

(a) Kernel in UPPAAL



(b) Detector in UPPAAL

Fig. 6.4: UPPAAL model of the FSM-SADFG of Fig. 6.1

are the scenario IDs, while data buffers are abstracted into integers as only the amount of data buffer tokens matters, not their value.

Every kernel $k_i \in \mathcal{K}$ is translated to the TA $\mathbb{A}_{k_i} = (L_i, l_i^0, E_i, I_i)$ where $L_i = \{\texttt{Initial}, \texttt{Fire}\}$, $l_i^0 = \texttt{Initial}$, and E_i and I_i are given as follows. The edge

$$\texttt{Initial} \xrightarrow{|\psi(k_i)|\geq 1 \wedge \forall b\in In(k_i):\phi(b)\geq R_c(b,s),\emptyset,\{x_i\}} \texttt{Fire}$$

corresponds to the kernel start action. To start firing, the kernel must first gain knowledge in which scenario is it operating in. This information is stored in the kernel's control buffer. Therefore, the kernel *peeks* into its control buffer if it is not empty, finds out the operating scenario s and waits for the availability of the required number of tokens in its data buffers. This behavior is encoded using the guard `bool k_tok_available(int ker_id)` where `ker_id` is the ID of the kernel. Once the guard evaluates to `true`, the kernel can actually perform the start action, by consuming input tokens both from its control buffer and its data buffers. Consumption corresponds to statuses being decremented. This behavior is encoded by the function `void k_start_fir(int ker_id, int& scen_id, int& delay)`, where `scen_id` is the ID of the operating scenario, and `delay` is the kernel's firing delay in the operating scenario. Aforementioned variables get their values within the update label although these are known during the evaluation of the `k_tok_avail` guard. This is because guards in UPPAAL must be side-effect free. These variables are needed by the kernel end action that corresponds to the edge

$$\texttt{Fire} \xrightarrow{x_i = E(k_i, s), \emptyset, \emptyset} \texttt{Initial}$$

and the invariant $I(\texttt{Fire}) = x_i \leq E(k_i, s)$. These two ensure that the system stays in the location `Fire` for exactly the execution time $E(k_i, s)$ of the kernel k_i in scenario s. Thus, time transitions are encoded implicitly in the operation of the network of TA, for which time progresses in unison. The resulting data token production is coded in the function `void k_end_fir(int ker_id, int scen_id)`.

The detector TA uses the structure of its FSM (e.g. locations `xi1` and `xi2` of Fig. 6.4b correspond to states ξ_1 and ξ_2 of the FSM of Fig. 6.1b), but embeds in each transition a firing location wherein time can pass between the events of consuming the input tokens and producing the output tokens. We encode it as $\mathbb{A}_d = (L_d, l_d^0, E_d, I_d)$, where $L_d = \Xi \cup \{(\xi, \xi') \mid \xi, \xi' \in \Xi \wedge \xi' \in \mathbb{T}(\xi)\}$ and $l_d^0 = \xi_0$. The edge set E_d is defined such that each transition $\xi_i \to \xi_j$ described by $\mathbb{T}$ is translated into a detector start edge followed by a detector end edge:

$$\xi_i \xrightarrow{\forall b \in In(d): \phi(b) \geq R_c(b, \Phi(\xi_i)), \emptyset, \{x_d\}} (\xi_i, \xi_j) \xrightarrow{x_d = E(d, \Phi(\xi_i)), \emptyset, \emptyset} \xi_j$$

The invariant function I_d is defined such that each firing location (ξ_i, ξ_j) maps to the invariant $x_d \leq E(d, \Phi(\xi_i))$. The guard `bool d_tok_avail(int scen_id)` of edge $\xi_i \to (\xi_i, \xi_j)$ ensures that there are enough tokens present in detector's input data buffers before it commences firing. The operating scenario of the detector depends on the current state of the scenario FSM. The function `void d_start_fir(int scen_id, int& delay)` updates data buffer statuses as the result of the detector start action being performed. The `delay` variable receives its value within the function. The update function `void d_end_firing(int scen_id)` of edge $(\xi_i, \xi_j) \to \xi_j$ encodes the effects of the detector end action to control statuses, i.e. the production of control tokens.

To ensure determinacy in the presence of auto-concurrency we revert to the considerations of Section 6.3.4. In UPPAAL, this means that we replicate data buffers which are being filled by auto-concurrent kernels over scenarios by simply declaring them as arrays of integers, where the array index corresponds to the scenario ID. The aforementioned guard and update functions of Fig. 6.4 all use `scen_id` as the input parameter and will therefore operate on the correct buffer replicas. As mentioned in Section 6.3.4, auto-concurrency is only allowed for kernels that produce to buffers with no initial tokens. Buffers in which overtaking is not possible, are not replicated and the "default container" is used to store the number of tokens present in the buffer. Whether overtaking is possible or not in a buffer is encoded with a configuration constant that is checked in guard and update functions.

When it comes to scheduling policies, the operational semantics of Section 6.3.3 does not prescribe a particular one and consequentially neither does the previously discussed TA translation. However, it is often convenient to assume a certain class of scheduling policies. An important class are those policies where actions take place without delay, i.e. processes fire as soon as they are enabled. Recall from earlier chapters that we refer to executions under such policies as *self-timed executions*. In UPPAAL, the concept of urgency can be exploited to impose such a policy. Specifically, an urgent broadcast channel [14] can be used to force kernel and detector start actions without delay (channel `self_timed` in Fig. 6.4). Following the work of [114][46] [66][115] we adopt the concept of self-timed execution in the remainder of this chapter.

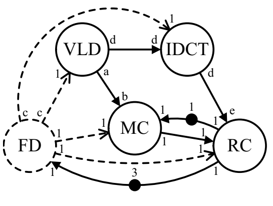
To conclude, the translation of this section enables the user to represent an FSM-SADF specification as a network of TA. The user can chose between allowing and prohibiting auto-concurrency, by the use of system-level declarations. In case auto-concurrency is enabled, the user can chose to use or not to use the scenario-level FIFO policy of [46] to ensure determinacy. Furthermore, the user can choose to consider self-timed execution or not by the use of the urgency concept in UPPAAL.

6.5 Model checking of TA model

In this section we demonstrate examples of qualitative and quantitative analysis of FSM-SADF specifications using the UPPAAL model checker and its query language. UPPAAL's query language is used to specify the properties to be checked and is a subset of TCTL (timed computation tree logic) [5]. We use the MPEG-4 decoder of Fig. 6.5 as our case-study [110][111]. All parameters in our case study are taken from [111].

The decoder functionality is given by the scenario graph of Fig. 6.5a. The decoder processes streams consisting of I and P frames. These frames consist of a variable number of macroblocks (0 to 99 for QCIF). The detector (FD) detects the frame type. If the frame type is I, all frame macro-blocks are decoded by the VLD and IDCT kernels, while the image is reconstructed by the RC kernel. When the frame type is P, motion compensation is required in the decoding process. Its functionality is implemented by the MC kernel. I frame processing defines the I_{99} scenario, while P frame processing defines the P_x scenarios where x stands for the number of macro-blocks in the P frame and $x \in \{0, 30, 40, 50, 60, 70, 80, 99\}$. The scenario occurrence pattern is defined by the scenario FSM of Fig. 6.5b.

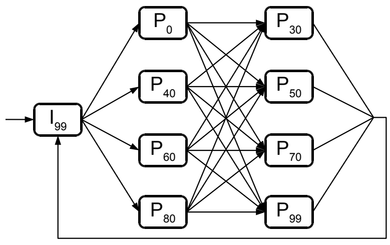
FSM-SADF is a non-deterministic model. With non-deterministic mod-



Rate	Scenario I_{99}	P_0	P_x
a	0	0	1
b	0	0	x
c	99	1	x
d	1	0	1
e	99	0	x

$x \in \{30, 40, 50, 60, 70, 80, 99\}$

(a) Scenario graph



(b) Scenario FSM

Fig. 6.5: FSM-SADF model of an MPEG-4 decoder [110][111]

els we are interested in worst-case analysis. The particular metrics analyzed, the obtained results and the associated time and memory usage are shown in Table 6.1. We compare our results and resource requirements to those of the SDF^3 tool. In SDF^3, two sets of algorithms can be used to analyze FSM-SADF: 1) The algorithms of the customized model checker [116, 114] of SADF by considering states for which the involved reward is maximal. However, these algorithms can be used only if there is no auto-concurrency in the FSM-SADFG (e.g. all kernels have self-edges with one initial token). In that case, the graph can be analyzed for deadlock freedom, maximum buffer occupancy, maximum inter-firing latency, maximum response delay and throughput. 2) The Max-plus based algorithms of [46] specifically designed for FSM-SADF that can only analyze throughput and latency. The Max-plus techniques for what they offer (throughput and latency) will outperform their model-checking counterparts. However, the model-checking based SADF techniques offer a wider palette of metrics amenable to analysis. Therefore, we base our comparison on the techniques of SADF. To be able to compare our results to those of SDF^3, we limit the auto-concurrency of all kernels in Fig. 6.5a to one (imagine that all kernels have a self-edge with one initial token). Before proceeding, we point out a subtle difference between the operational semantics of SADF and FSM-SADF. In contrast to the SADF model, in FSM-SADF the value of the control tokens produced by the detector end action depends on the current state of the FSM and not on

Table 6.1: MPEG-4 verification time and virtual memory usage

	SDF^3			**UPPAAL**		
Analysis	**Result**	**Time [s]**	**Mem [MB]**	**Result**	**Time [s]**	**Mem [MB]**
Deadlock freedom	Deadlock free	n/a	n/a	Deadlock free	17.07	564
Maximum buffer occupancy, all buffers	278, 278, 3, 3, 2, 99, 1, 1, 116, 3	17.27	169	278, 278, 3, 3, 1, 99, 1, 1, 108, 3	7.31	551
Maximum inter-firing latency, all processes	4327, 7470, 40, 4303, 40	6.44	171	4327, 7470, 40, 4327, 57	11.45	557
Maximum response delay, all processes	0, 0, 40, 4327, 57	0.08	85	0, 0, 40, 4327, 57	0.94	312
Throughput	0.000363636	1.10	86	n/a	n/a	n/a
Process interleavings, MC & RC	n/a			1	8.34	556
Inter-process delay, MC & RC	n/a			≤ 5000	13.61	561
Realized kernel auto-concurrency, VLD	n/a			2	31.84	1072
Pipeline depth	n/a			3	7.30	551
Parallelism between actors, MC & RC	n/a			None	7.29	551

the next state. Therefore, to obtain the same behavior and corresponding analysis results, the FSM of Fig. 6.5b needs to be augmented with the one additional state I' before being subjected to SDF^3 analysis. State I' must be declared initial with one outgoing transition leading to the "original" initial state I.

The experiments were performed on an Intel Core i5-750 CPU with 8 GB of main memory running UPPAAL 4.1.19 64-bit on Linux. The default settings were used and UPPAAL was restarted between each query.

6.5.1 Deadlock

Analyzing for deadlock freedom is achieved with the UPPAAL query (`A[] not deadlock`) where "`A`" is to be read as "always (invariantly)". In SDF^3, checking whether a FSM-SADFG is deadlock free is performed during the computation of any metric, i.e. there is no special option for this type of analysis and the respective time and memory requirements cannot be given (n/a in Table 6.1).

6.5.2 Maximum buffer occupancy

Maximum buffer occupancy of a particular buffer under all possible self-timed schedules is obtained using the UPPAAL supremum operator, e.g. (`sup:` b_i). In SDF^3, we use the input argument `--compute buffer _occupancy"(maximum)"` during application invocation. The results are ordered by the sequence: FD2VLD, FD2IDCT, FD2MC, FD2RC, VLD2IDCT, VLD2MC, RC2MC, MC2RC, IDCT2RC, RC2FD. The difference in results between SDF^3 and UPPAAL is due to another subtle difference in the operational semantics of SADF and FSM-SADF. In FSM-SADF, token consumption takes place during detector/kernel start actions, while in SADF consumptions take place at a later point - during the detector/kernel end actions. Therefore, SDF^3 will often deliver higher values because in SADF tokens then get to spend more time in buffers than in FSM-SADF where they during the firing may, conditionally speaking, be considered to be locally stored in the kernel itself and not in the input buffer.

6.5.3 Maximum inter-firing latencies

Maximum inter-firing latency of a process is defined as the maximum elapsed time between two successive firing completions of the process. In UPPAAL, process latencies can be obtained as suprema of the clock `y` that is reset every time process p completes its firing, i.e. `sup:` p`.y` while p`.y = 0` every time edge $p.$`Fire` $\rightarrow$ $p.$`Initial` of Fig. 6.4a is fired for the kernels or every time edge $d.(\xi_i, \xi_j) \rightarrow d.\xi_j$ of Fig. 6.4b is fired for the detector. In SDF^3, we use the input argument `--compute inter_firing_latency"(maximum)"`. The results are sequenced by FD, MC, VLD, RC, IDCT. In our experiments we assume that all processes have just competed firing at $t = 0$, and therefore in the latency measurement we include also the difference in time between the completion of the first firing of the process and $t = 0$, while SDF^3 starts measuring the time difference only after the first firing of the process had completed. Therefore, we observe the difference in values delivered by SDF^3 and UPPAAL. The difference is entirely due to differing definitions of inter-firing latencies used in a particular setting.

6.5.4 Maximum Response Delays

Maximum response delay of a process denotes the maximum time until the first firing completion of that process. In UPPAAL, we determine it by checking the relationship between the maximum response delay of a process p and a constraint r using the query (`E<> !`p`.bFirstFirCompleted and `p`.y >=` r), where p`.y` is a clock that is never reset, and p.`bFirstFirCompleted` is a variable set to `true` when p completes its first firing. In SDF^3, we use the input argument `--compute response_delay"(maximum)"`.

6.5.5 Throughput

Throughput of an FSM-SADFG is defined as the long-run average of completed iterations per time unit. However, the analysis of this chapter, unlike the Max-plus-based analysis of earlier ones, works at the process firing granularity. Therefore, we must find a way how to define throughput in terms of process firings.

Throughput of a process is defined as the long-run average number of firing completions of a process per-time unit. In SDF, the throughput of the entire graph is defined as the throughput of a process normalized (divided) by the number of firings of that process within the graph iteration [50]. The same definition can be applied to FSM-SADF. Furthermore, as in FSM-SADF the repetition vector entry of the detector equals to one for all scenarios (detector fires once per scenario), so is the throughput of the entire graph equal to the throughput of the detector process. The TCTL [5] based query language of UPPAAL cannot be used to evaluate such long-run averages. In SDF^3, we use the input argument `--compute throughput"(FD)"`.

6.5.6 Process interleaving

We continue with a set of simple reachability properties not supported by SDF^3, but that can easily be verified in UPPAAL.

We check the interleaving of different process firings, e.g. "between two consecutive firing completions of the process p, process q completes at least n firings", etc. For this we use a *leads to* query (whenever a eventually b) and a counter variable: (p`.Fire --> `q`.FireCount >= `n). Variable q`.FireCount` is reset every time p takes the edge `Fire` $\rightarrow$ `Initial` and incremented every time q takes the same edge. In the experiment, $p =$ MC, $q =$ RC, and $n = 1$.

6.5.7 Inter-process delay

We can also check whether the maximum delay between the completion times of firings of two processes within a scenario is greater than, less than or equal to a predefined value by constructing a query monitor TA that synchronizes with the events of firing completions of the processes it monitors. In the case of a kernel p, this synchronization takes place when the edge p`.Fire` $\rightarrow$ p`.Initial` is taken. In the experiment we verify that the MC-RC delay is always smaller than 5000.

6.5.8 Realized kernel auto-concurrency

If we bound the auto-concurrency of a particular kernel p to N_p, we can check whether it has been fully utilized or not. Allowing N_p concurrent executions of p means that we have assigned N_p processing elements to p. If all are not used, the idle ones can be assigned to kernels of another application. This design decision might improve the overall performance of the system hosting multiple applications. To determine the realized auto-concurrency of a kernel p we use the query `sup:` *p.count* where *p.count* is a variable that is incremented every time edge `Initial` $\rightarrow$ `Fire` is taken and decremented whenever the edge `Fire` $\rightarrow$ `Initial` is taken. In this experiment, $p = VLD$, $N_{VLD} = 2$ and `sup` : $VLD.count = 2$.

6.5.9 Pipeline depth

Pipeline depth denotes the maximum number of scenario executions active at the same time. In case of our MPEG-4 case study, the beginning of a scenario is marked by the firing completion of the detector FD, while its end is marked by the firing completion of the RC process. To compute the pipeline depth *pdepth*, use the query `sup` : *pdepth* where *pdepth* is incremented every time FD takes the edge $(\xi_i, \xi_j) \to \xi_j$ and decremented every time RC takes the `Fire` $\to$ `Initial` edge. In this case the pipeline depth is three as there are three initial tokens in the data buffer (RC, FD). This was immediately visible in this example, but one can easily imagine a more complicated initial token distribution where mere visual inspection would not suffice.

6.5.10 Parallelism between actors

We can check whether processes in an arbitrary subset of `P` can fire in parallel. E.g., processes p and q can fire in parallel if the query `E<>` p`.Fire and` q`.Fire` evaluates to `true`. In our experiment $p = MC$ and $q = RC$. These two cannot fire in parallel.

6.6 Discussion

Another look at Table 6.1 reveals that UPPAAL allows us to check the model against various properties, many of which are not supported by the SDF^3 tool-set.

In our approach, an FSM-SADF model is encoded in the UPPAAL model checker as follows. First, the structure of the scenario graph including the description of scenarios (rates and firing delays across scenarios) is fed to our modeling framework as a simple configuration table. Second, the scenario FSM is specified using the UPPAAL graphical user interface. Therefore, no coding is required from the user side. The only effort the user has to undertake (as with any other tool) is to describe the model to the tool. The analysis is by definition automatic and exact in terms of the input model, meaning that if the input model is an exact representation of reality, so is the analysis. Particular analyses are performed by running the UPPAAL verifier module as discussed in Section 6.5 while their run-times are specified in Table 6.1.

On average, UPPAAL analysis will take the same time as that of SDF^3, but with higher memory demands. This observation justifies the use of a general verification tool such as UPPAAL as a complement to specialized tools. The flexibility of the UPPAAL's TCTL based query language and the possibility of construction of various query monitor automata allows the user to easily compute various qualitative and quantitative properties of the model.

The only metric not supported by UPPAAL that is available in SDF^3 is

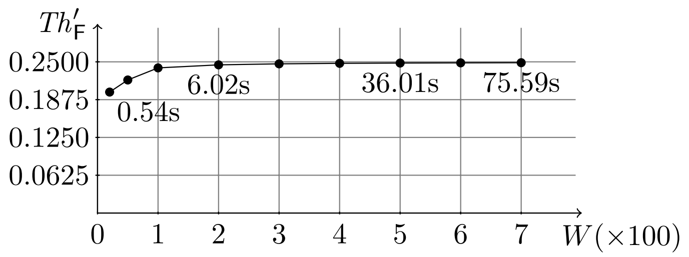


Fig. 6.6: Convergence of the horizon method for throughput estimation for the FSM-SADFG of Fig. 6.1

throughput. Here we take the opportunity to shortly discuss the applicability of UPPAAL and TCTL in a conservative estimation of the throughput value.

Let $\mathcal{W}$ be a window of time of finite duration W. The throughput equals to the long run average number of firing completions of the detector process per time-unit. Therefore, let c be a variable that is incremented every time the detector completes a firing, i.e. takes the $(\xi_i, \xi_j) \to \xi_j$ edge and reset every time when clock T equals to W $(T == W)$ along with the clock T itself. By defining the property `A[]` $(T$ `==` $W)$ `imply` c `>=` H, we can verify that the value of c within **all** time windows $\mathcal{W}$ will be greater or equal than the value of the *horizon* H. If the property is satisfied, the conservative (property holds along the entire time axis divided into windows $\mathcal{W}$ of duration W) throughput estimate Th'_F is given by $Th'_F = H/W$. By using larger values of W and by finding the maximum H for which the property holds, we tighten the estimate at the price of increased analysis time. We call this method of conservative throughput estimation for FSM-SADF the *horizon method*. Unfortunately, the horizon method scales poorly, and we could not apply it successfully to our MPEG-4 case study. In practice, this is not a problem because the Max-plus-based engine of SDF^3 can be used to effectively compute the throughput of large FSM-SADFGs. However, for completeness we show how to use UPPAAL to compute the worst-case throughput using the "small" example FSM-SADF graph of Fig. 6.1. Fig. 6.6 shows how Th'_F converges to the exact value of 0.25 for growing W. Some points are decorated with the time required by UPPAAL to deliver the estimate value.

Table 6.2: UPPAAL vs. SDF^3 scalability via maximum buffer occupancy

	SDF^3		UPPAAL	
Pipeline depth	**Time [s]**	**Mem [MB]**	**Time [s]**	**Mem [MB]**
1	0.80	90	1.14	121
2	5.5	113	2.96	196
3	17.96	169	7.42	567
4	64.66	319	20.44	1144
5	392.02	662	81.17	2551
6	$>$ 1800	$>$ 1708	$>$ 133	$>$ 5752

6.7 Scalability issues

We investigate the scalability of the TA FSM-SADF analysis from two angles.

First, we consider time and memory requirements for the maximum buffer occupancy calculation for the MPEG-4 case study while increasing the pipeline depth, i.e. the number of initial tokens on (RC, FD) channel. The results and the comparison with the SDF^3 tool are shown in Table 6.2. With the pipeline depth of six, both tools experience state-space induced complexity problems. Among the metrics supported by SDF^3, the maximum buffer occupancy is the most demanding computation as the diamond property of the underlying TPS [114] cannot be exploited, i.e. all interleavings of timeless actions need to be considered. UPPAAL will by default considers all possible interleavings of timeless actions and will time-wise perform better that SDF^3 for this type of analysis.

Second, for UPPAAL, we check how it copes with auto-concurrency when performing the maximum buffer occupancy analysis. We perform the experiments by increasing the number of *VLD* kernel instances in the model. The results of the experiments are shown in Table 6.3. Already with three instances of *VLD* we experience state-space induced complexity problems. This indicates poorer scalability of the translation in the presence of auto-concurrency when performing maximum buffer occupancy analysis when compared to the case with no auto-concurrency at all.

It is not clear how to improve the scalability of the model for maximum buffer occupancy analysis and other analysis defined by queries over discrete variables. This is because in these type of analysis all action interleaving

Table 6.3: UPPAAL scalability via maximum buffer occupancy for increased auto-concurrency

	UPPAAL	
Number of VLD instances	**Time [s]**	**Mem [MB]**
1	7.48	559
2	31.44	1088
3	> 175.00	> 6249

need to be considered. However, for the analysis of temporal properties like inter-firing latency and process response times (model-checking against clocks), the SADF/FSM-SADF diamond property [114] could be exploited. Unfortunately, traditional partial order reduction techniques embedded in UPPAAL are insufficient to take advantage of this. However, process priorities of UPPAAL could be used to do exactly that by arbitrarily prioritizing process timeless actions which then can be considered as a simple form of partial order reduction. This way significant state space reductions could be achieved. We leave these considerations to future work.

6.8 Summary

FSM-SADF is a powerful dataflow formalism that is able to capture the dynamic behavior of modern streaming applications while offering a good trade-off between expressiveness, analyzability and implementation efficiency. However, the formalism is currently only supported by the SDF^3 tool-set which implements a predefined set of properties that can be analyzed/verified. In this chapter we proposed a translation of (nonparametric) FSM-SADF to TA, thereby enabling the use of the UPPAAL model checker for analyzing and verifying user-defined properties in a straightforward manner. Our translation of FSM-SADF model to TA is also the first translation of a member of the SADF MoC family to a model-checker that supports auto-concurrency. We also report on the scalability issues experienced.

For SDF-PFSM-SADF of Chapter 4 such a translation is not possible because UPPAAL or (to the best of our knowledge) any other widely-used model checker do not support parametric discrete variables that are part of the state and that could be used to model parameterized rates of SDF-PFSM-SADFG. Support exists however for capturing of parameterized firing delays within the concept of parametric timed automata (PTA) [62]. In particular, parametric timed automata allow within clock constraints the use of parameters in place of constants. Nevertheless, PTA does not allow to specify complex parameter inter-dependencies and the analysis would in many cases be overly pessimistic.

6.9 A final remark

In FSM-SADF, scenario releases performed by the detector are data-driven. However, many (dynamic) streaming systems make use of event driven mechanisms to control the data-intensive pars of the system. With that in mind, we refer the reader to Appendix A that briefly discusses a flavor of FSM-SADF with event-driven control, i.e. where scenario releases are made dependent on external events.

Chapter 7

Conclusion and future work

7.1 Conclusion

Dataflow MoCs have been traditionally used in the modeling, analysis and synthesis of (real-time) streaming applications. In the world of dataflow MoCs, SDF is regarded as the most mature and stable one. Thanks to its static nature where the numbers of tokens to be produced and consumed are constant and known at design-time, it is predictable, statically schedulable and amenable to powerful optimization techniques but at the cost of limited expressive power.

However, modern streaming applications exhibit increasing levels of dynamic behavior and SDF is not expressive enough to capture their dynamic behavior. Nevertheless, for many dynamic applications it is possible to construct conservative SDF models. However, using those models in the design process results in over-allocation of already scarce embedded system resources. In a present-day design setting where the market is pushing for more and more functionality to be added on the same device, prolonged battery life and decrease in end product price, such an over-allocation is not acceptable. Therefore, designers of embedded (real-time) streaming systems must must be made familiar with other dataflow MoCs than SDF. Many of those models are indeed sufficiently expressive to capture complex dynamic behavioral patterns, examples being KPN and DDF models. However, the increase in expressiveness with these models must be paid in terms of significantly reduced analyzability.

In particular, in a real-time setting that we focused on in this work, where the correctness of the system depends not only on the logical result of the computation, but also on the time at which results are produced, rigorous temporal analysis needs to be performed on the system model to derive associated real-time guarantees. Unfortunately, for many dynamic dataflow MoCs the problem of determining these real-time properties is undecidable.

However, there exist models that are sufficiently expressive to capture dynamic aspects of modern streaming application while retaining design-time analyzability. One of those models is FSM-SADF. FSM-SADF considers applications whose dynamic behavior can be seen as a progression of a collection of fairly static patterns called scenarios. Each scenario is then modeled by an SDF model while a nondeterministic FSM abstracts application control requirements by constraining scenario occurrence patterns.

Although well-suited for capturing applications exposing coarse-grained dynamism, once the number of scenarios starts to grow, FSM-SADF will experience compactness problems that hamper its use in the modeling and analysis of applications exposing fine-grained data-dependent dynamism. Simply put, the size of the model will become unmanageable.

To alleviate the problems that hamper the use of FSM-SADF in analyzing such applications, in Chapter 3 we introduce the concept of SDF-PDF that integrates dynamic parameters with SDF. By using parameters we can compactly represent applications exposing fine-grained data-dependent dynamism. By showing that the concept of configuration in SDF-PDF is correspondent to the concept of a scenario in FSM-SADF, we were able to model the phenomena of parameter reconfiguration of SDF-PDF using the theory of Max-plus automata and show how to derive worst-case performance guarantees for SDF-PDF. We did this by recasting and extending the FSM-SADF analysis techniques to a parametric context.

SDF-PDF as defined in Chapter 3 and parameterized dataflow (PDF) in general are specialized for modeling of data-intensive applications that exhibit fine-grained data-dependent dynamism. However, there exists many applications that combine the former with intricate control requirements. To be able to compactly capture such applications, we add support for control to PDF. In particular, in Chapter 4 we show how to combine the concept of parameter reconfiguration in PDF with finite-state control yielding a novel FSM/dataflow hybrid we refer to as PFSM-SADF. In PFSM-SADF, every parameterized scenario is modeled by an PDFG, while an FSM constrains parameterized scenario occurrence patterns. For an SDF-based specialization of PFSM-SADF, called SDF-PFSM-SADF we show how to derive worst-case performance guarantees using the theory of Max-plus automata.

Following Chapter 4, in Chapter 5 we address the throughput computation problem for an important setting of SDF-PFSM-SADF, in particular one in which SDF-PFSM-SADFG parameters are deemed static, i.e. once set they do not change or change infrequently. We show how combining the existing results on parametric throughput analysis of FSM-SADF, our own Max-plus characterization of parameterized scenarios developed in Chapters 3 and 4 and the RLT technique we can express throughput of the graph as a simple function of graph and design environment parameters. Such expressions can then be used both at design-time and run-time to perform various types of optimizations.

The main lesson learned from Chapters 3, 4 and 5 is that parameterization can be used as an effective syntactic construct that allows for a compact representation (keeps the model size manageable) of scenario-aware streaming applications exposing fine-grained data-dependent dynamism. Furthermore, we have shown that by using parameterization we are able to replace enumerative analysis the run-time of which may oftentimes be prohibitive with a single parametric analysis. However, the price to pay for that is the tightness of the analysis.

The temporal analysis techniques presented throughout Chapters 3, 4 and 5 are, in essence, based on the Max-plus algebraic semantics of FSM-SADF. For FSM-SADF (and transitively for SDF-PDF and PFSM-SADF), the analysis works at an iteration granularity and provides no insight into the behavior of particular actors within iterations. Furthermore, the scope of the analysis is limited to non-functional properties, i.e. to worst-case throughput and latency. Nevertheless, analyses exists for FSM-SADF that can consider other properties than available via Max-plus-based analysis and are available in the SDF^3 tool. Examples are deadlock freedom, buffer occupancies, etc. However, the analysis of SDF^3 explicitly excludes auto-concurrency and is limited to only a predefined set of properties that the user cannot influence. To extend the analysis by taking auto-concurrency into account, in Chapter 6 we report on the translation of the FSM-SADF formalism to UPPAAL timed automata that enables a more general verification than currently supported by existing tools.

The lessons learned from Chapter 6 are interesting too. On one hand, applying general purpose model-checkers to FSM-SADF does enable quick definition and verification of user-defined properties which SDF^3 does not support. On the other hand, a general purpose model checker cannot in general make use of intrinsic properties of FSM-SADF like action determinacy (as a custom model-checker can) to reduce state-space of the model and so improve the scalability of the analysis. To achieve such reductions, the user has to be inventive and use the available modeling constructs of the model-checker (like priorities in UPPAAL).

Related to Chapter 6 but outside its scope, in Appendix A we briefly consider combining FSM-SADF and even-driven control to enable modeling and analysis of (interactive) streaming applications which make use of event-driven mechanisms to control the operation of data-intensive parts. We showed that under non-preemptive FIFO scheduling policy it is possible to formulate and solve the schedulability problem for such applications as a reachability problem for ordinary timed automata.

The modeling concept of Chapter 6 can be regarded as a special case of more general RPN, while its further development in the notation and semantics of RPN is left for future work.

7.2 Proposals for future work

This thesis investigated real-time (and some functional) aspects of modeling and analysis of scenario-aware real-time streaming applications by exploiting the concept of scenario-aware dataflow. With regard to the contributions of the thesis briefly listed in Section 7.1 there are still some open questions that deserve future research.

- **Full automation.**
  Currently some parts of the parametric Max-plus analysis engine of this thesis are not implemented in software but are performed manually. In particular, we mean the Max-plus characterization of SDF-PDFGs of Algorithm 3.1. Therefore, it would be advantageous to fully automate the analysis framework potentially on top of the TU/e SDF^3 tool and in detail analyze its scalability.

- **Alleviation of limitations imposed to the SDF-PDF input graph structures.**
  The Max-plus characterization procedure of SDF-PDF structures specified in Algorithm 3.1 requires that the input graph structures satisfy Requirements 3.1 and 3.2. In particular, Requirement 3.1 limits the applicability of our techniques in the presence of cyclic dependencies not defined by saturated channels. A starting point to eliminate that requirement would be to apply a compositional analysis where input graph subgraphs that do satisfy Requirement 3.1 would first be analyzed in isolation, yielding the Max-plus characterization that would be used as a building block in the Max-plus characterization of the graph as a whole.

- **Parameterized subscenarios.**

In PFSM-SADF, scenarios are allowed to change in-between iterations. However, it would be advantageous to allow parts of the graph to be reconfigured within the iteration of the graph as a whole. This leads us to the concept of subscenarios that involves a hierarchical FSM discipline. Therefore, in the context of PFSM-SADF it would be interesting to consider (parameter) reconfigurations at the boundaries of local scenario graph iterations that are dictated by a local control mechanism in the form of an FSM.

- **Latency as a maximal delay between corresponding firings of SDF-PDFG and SDF-PFSM-SADFG actors.**
  In the context of SDF-PDF an SDF-PFSM-SADF latency was defined as a linear bound on actor firings with regard to a given period. Other definitions of latency exist. In particular, an interesting one may be the one defined as the maximal delay between the firings of actors in the graph representing the input and the output of the system [51]. With such a definition, different methods for latency analysis than the ones presented in Chapters 3 and 4 would need to be developed. These would potentially, in addition to worst-case evaluation Max-plus matrices, need to use the best-case evaluation matrices that specify the earliest production times of tokens. Using the two, the latency can be derived as the difference of the latest and earliest firing times of actors representing the output and the input to the system, respectively.

- **Throughput-constrained DVFS.**
  Section 5.2 briefly motivates the main result of Chapter 5 by demonstrating how, if one had throughput of a throughput-constrained application expressed as a simple function of parameters, this information could be used at design-time to optimize energy consumption. An interesting next step would be to consider how the same could be done at run-time (in an environment where parameters infrequently change) by employing a heuristic that would give a suboptimal solution to the optimization problem of (5.7).

- **Cost-optimal reachability analysis.**
  Chapter 6 proposes a translation of FSM-SADF to UPPAAL timed automata. However, some scalability issues have been reported. With regard to those, we believe it is possible to improve the scalability of the analysis (at least in the analysis of temporal properties) by using priorities in UPPAAL which would serve as means for achieving a simple form of partial order reduction. Furthermore, as our translation also

sets the first milestone towards enabling the use of FSM-SADF in a wider context, e.g. cost-optimal analysis, it would be worthwhile to investigate reachability analysis of applications modeled by FSM-SADF through an optimal control formulation using the UPPAAL family of model checkers. E.g., by exploiting the priced timed automata formalism using the UPPAAL CORA tool, where energy associated with an actor firing could be formulated as a cost, one could compute scenario schedules/sequences that minimize energy consumption.

- **Event-driven scenario-aware dataflow.**
  In Appendix A we rather informally hinted how one can combine FSM and event-driven control and establish schedulability analysis for the combination. A significant amount of additional research effort is needed to fully formalize the concept and extend the analysis to different scheduling policies including preemptive ones by using stopwatch automata in UPPAAL.

Appendix

A Event-driven control and FSM-based scenario-aware dataflow

FSM-SADF combines streaming data and finite control. Control is embodied in the detector process that is operationally nothing but an FSM. The reactions of the FSM are triggered by availability of tokens at detector's dataflow input ports that are produced by kernels. These productions may be considered synchronous events.

However, many (real-time) streaming systems depend on asynchronous event-driven mechanisms that control the operation of dataflow processes. These mechanism are often based on FSMs fueled by external inputs.

Therefore, we briefly consider modeling and analysis possibilities for such systems using a flavor of FSM-SADF the FSM of which is in addition to being fueled by feedback from the kernels, also fueled by external inputs. Furthermore, we show how the construct can be translated to UPPAAL timed automata and subjected to schedulability analysis.

The contribution of this appendix has not been (yet) published as fully formalizing and completing the result requires a significant amount of additional (and future) research effort.

A.1 Introduction

FSM-SADF is an FSM/dataflow hybrid. It is composed out of a number of processes we call kernels and a single detector. The detector is defined by an FSM that captures application control requirements.

Externally, the detector FSM obeys the SDF semantics which means that it consumes a number of tokens on every input. The production and consumption numbers are dependent on the states of the FSM, i.e. scenarios. Every firing of the detector entails a reaction of the FSM which then means that these reactions are data-driven, i.e. triggered by the availability of input tokens that enable the detector firing.

In particular, if we adopt the definition of an event from [68] that says that an event is any occurrence that causes a change of control flow, we may consider reactions of the FSM-SADF FSM to be triggered by synchronous events. Synchronous, because in the parlance of [68], these events are those that occur at predictable times in the flow-control. Examples of such events are conditional branch instruction or the occurrence of an internal trap interrupt. All these can be anticipated.

However, there exist events that cannot be anticipated such as a hardware interrupts. These (external) events occur at unpredictable times in the flow-control [68] and consequently change it. We refer to such events as asynchronous events that we will from now on simply refer to as events.

FSM-SADF has no provision for capturing systems that make use of event-driven mechanisms.

Therefore, we use the opportunity to discuss an extension of FSM-SADF able to capture systems where event-driven mechanisms based on FSMs are used to control the operation of data-intensive system parts, i.e. the kernels. In particular, we will modify the detector's FSM in a way that in addition to reacting to feedback from kernels it will now also react to external inputs. Thereafter, we show how to perform the translation of the novel construct to UPPAAL timed automata for verification purposes where we will focus on the interesting problem of schedulability analysis.

A.2 Our analysis model

A.2.1 The basics

Generally speaking, we consider systems where different interacting components by making use of events control the operation of data-intensive components of the systems, i.e. the dataflow components.

Therefore, it is only natural to differ between two types of processes in such systems as done in [56]: *data*- and *event*-driven processes where event-driven processes control the operation of data-driven processes. In addition, we assume that event-driven processes are based on FSMs that are fueled by external inputs as well as by the feedback from the data-driven parts of the system.

Inspired by [56], we now depict these two types of processes in Fig. 1. Event-driven processes must be activated immediately after an event connected to one of the inputs has occurred. Data-driven processes are activated after there is enough data tokens at their inputs. The required token quantities are specified by configurations the even-driven processes deliver

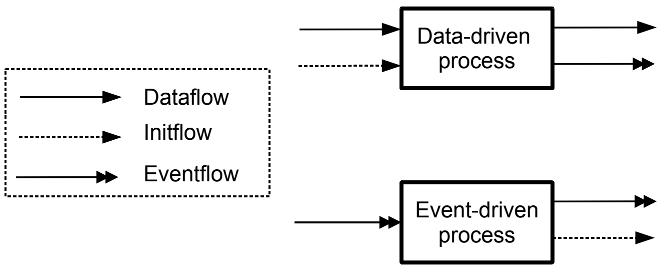


Fig. 1: Event- and data-driven processes.

to data-driven processes.

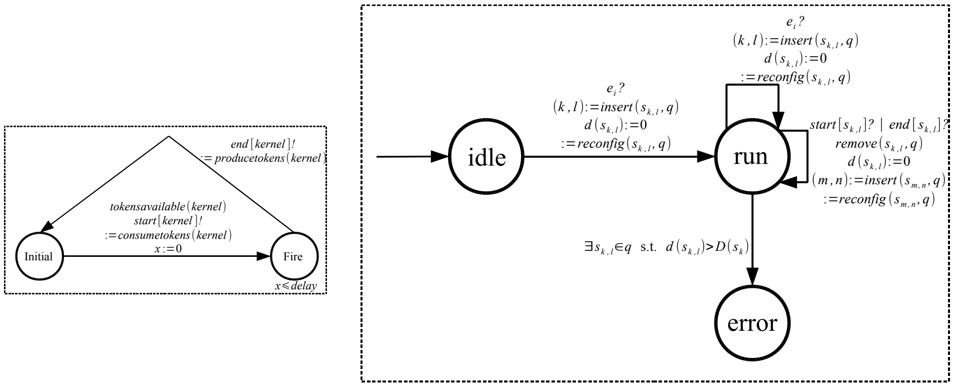
Events are communicated using the *eventflow* queues, configurations are delivered to the data-driven processes using the *initflow* streams, while data tokens are communicated using the *dataflow* streams.

Our objective is to combine event-driven control with FSM-SADF. Now, FSM-SADF originally recognizes two types of processes: kernels and detectors. Kernels abstract the data-intensive parts of the system, while detectors abstract control-oriented parts of the system that control the data-dominated parts of the system, i.e. the kernels.

This means that in the context of Fig. 1, kernels are data-driven processes. They are configured using *initflow* streams in the usual way as described in Chapter 6 (the *initflow* stream in Fig. 1 is a control channel in the parlance of FSM-SADF). Detectors produce *initflow* tokens (control tokens). An FSM-SADF detector encloses an FSM that externally obeys the SDF semantics. Now, to incorporate even-driven control, detectors are added features enabling them to react to externally generated events and control the operation of kernels based on those events and, as usual, based on the feedback from kernels. *Eventflow* is used to communicate events between the environment and the detector and the kernels and the detector (note that kernels can have *eventflow* outputs too). Based on external events and feedback from the kernels, detectors configure kernels using *initflow* in the usual way, i.e. by scheduling a scenario execution.

Now, TA have proven expressive enough to capture many real-time systems in particular event-driven systems [83]. In Chapter 6 we have shown how to generate models of systems expressed in FSM-SADF in UPPAAL TA. Therefore, it is only natural to combine the two to produce a UPPAAL timed automata analysis model for the FSM-SADF with event-driven control.

As a digression from the discussion above (aside the TA part) it follows that our analysis model can actually be regarded as a special case of RPN



(a) Kernel. (b) Detector.

Fig. 2: UPPAAL TA models of FSM-SADF processes with event-driven control.

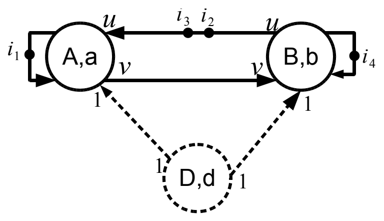
where the underlying process network(s) are SDFG(s) where one iteration of these SDFGs constitutes a single streaming transaction. For more details we refer to [45].

A.2.2 UPPAAL timed automata model of FSM-SADF with event-driven control

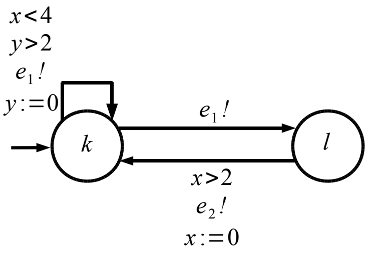
We build our TA analysis model using the translation of FSM-SADF to UPPAAL TA presented in Chapter 6. In particular, the modeling of *dataflow* and *initflow* (control channels) remains entirely the same. Event-based interactions are modeled using UPPAAL channels. A basic model of a kernel is shown in Fig. 2a.

It is of the same structure as the model of the kernel in Fig. 6.4a. Added channels `start` and `end` are used to notify the detector that the firing of the kernel had just started or completed, respectively. Note that events announcing the start or an end of actor firing in the parlance of [68] are synchronous events as they can be anticipated.

A basic model of the detector is shown in Fig. 2b. What a detector basically does, is that it reacts to external events $\{e_i\}$. Once an event is raised, the detector configures the kernels to execute some selected scenario $s_k \in \mathcal{S}$ ($\texttt{reconfig}(\texttt{s}_{\texttt{k,l}}, \cdot)$). Detector also receives feedback from kernels via `start` and `end` channels and performs, if necessary, any further kernel reconfiguration, i.e. release of a new scenario. As mentioned, this is a basic model and can be extended with a richer state structure. Further details



(a) Scenario graph.



(b) TA model of the external environment

	u	v
Scenario s_1	1	1
Scenario s_2	1	2

(c) Rates in scenarios s_1 and s_2

	a	b	d
Scenario s_1	4	1	1
Scenario s_2	2	5	1

(d) Actor firing delays in scenarios s_1 and s_2

Fig. 3: Example FSM-SADF with event-driven control.

regarding Fig. 2b are explained in the section to come.

A.3 Schedulability analysis

We now demonstrate on an artificial case study how we can use the UPPAAL TA model of FSM-SADF with even-driven control for schedulability analysis of dynamic even-driven systems. We build on the approach of [83] that showed how the schedulability problem of a real-time task set can be transformed to a reachability problem for ordinary timed automata and thus it is decidable. We show that the same can be done in a more general context where the simple task model of [83] is refined into a dynamically reconfigurable dataflow structure where reconfiguration patterns depend on the current state of the system and external inputs.

Consider the application shown in Fig. 3.

The data-intensive part of the application is abstracted into the scenario graph of Fig. 3a that defines two scenarios: s_1 and s_2 in detail specified by Fig. 3d and Fig. 3c. The system is to react to two events: e_1 and e_2. The timing constraints on these events are specified using the TA of Fig. 3b with two clocks x and y. Actually, the advantage with timed automata is that one can specify very relaxed timing constraints on events whereas with the traditional approach events are often considered to be periodic [83].

Now, the semantics of the events is as follows. The occurrence of event

e_1 requires the execution of scenario s_1 while the occurrence of e_2 requires the execution of scenario s_2. The hard-real time requirement imposed to the system is that the scenario must complete within a given time interval after the corresponding event had been raised. This interval defines the scenario deadline. Our task is to verify that that these deadlines are always met. We consider FIFO scheduling policy where a scenario cannot be preempted.

In this case, the detector can be easily constructed from the basic model of Fig. 2b. In particular, it retains the same state structure. Once an event had occurred (e_k?), scenario s_k is inserted into the list q of active scenarios ($\texttt{insert}(s_{k,l}, q)$) and a clock that measures its time of presence in the list is reset ($d(s_{k,l}) := 0$). We use index l to address a particular scenario instance as there may be more active instances of the same scenario. Finally, the detector emits control tokens into kernel control buffers (*initflow*) as an actual act of scenario scheduling ($\texttt{reconfig}(s_{k,l}, q)$). Once the scenario $s_{k,l}$ completes ($\texttt{end}[\texttt{s}_{\texttt{k,l}}]$), the scenario is removed from the active scenario list.

The predicate $\exists\, s_{k,l} \in q$ s.t. $d(k,l) > D(s_k)$ defining the guard on the transition to `error` state is used to denote the situation when the active scenario list becomes nonschedulable. To show that the system is schedulable we must show that the error state is never reached. This can be verified using the TCTL UPPAAL query `A[] not detector.error`.

In our experiment with $D(s_1) = 30$ and $D(s_2) = 20$ the system can be proven schedulable in 0.38s while using 36MB of static memory on an Intel Core i5-750 machine running at 2.66GHz. If we set $D(s_2) = 10$, the system is no longer schedulable.

Although we only consider FIFO scheduling, note that the detector model allows for modeling of more elaborate scheduling schemes. To do so, the user needs only to accordingly "implement" the UPPAAL update label `reconfig` of Fig. 2b.

A.4 Summary

In this section we have informally shown how event-driven control mechanisms can be combined with the existing reconfiguration facilities of FSM-SADF. Using the combination of the two we can capture systems than make use of even-driven mechanisms based on FSMs to control the operation of their data-intensive parts. Furthermore, we performed a proof-of-concept translation of the combination to UPPAAL timed automata. Finally, we have shown how to perform the schedulability analysis of the model under a FIFO non-preemptive scheduling scheme. Note that a significant amount of research work is needed to fully formalize the concept (most likely in the parlance of RPN [45]) and evaluate it on a set of realistic case studies.

Bibliography

[1] The Open SystemC initiative. `http://www.systemc.org/home/`. Accessed: 2016-01-04.

[2] U. Ahmad, Min Li, S. Pollin, R. Fasthuber, L. Van der Perre, and F. Catthoor. Bounded block parallel lattice reduction algorithm for mimo-ofdm and its application in lte mimo receiver. In *Signal Processing Systems (SIPS), 2010 IEEE Workshop on*, pages 168–173, Oct 2010.

[3] W. Ahmad, R. de Groote, P.K.F. Hölzenspies, M. Stoelinga, and J. van de Pol. Resource-constrained optimal scheduling of synchronous dataflow graphs via timed automata. In *Application of Concurrency to System Design (ACSD), 2014 14th International Conference on*, pages 72–81, June 2014.

[4] S. Altmeyer, C. Humbert, B. Lisper, and R. Wilhelm. Parametric timing analysis for complex architectures. In *Embedded and Real-Time Computing Systems and Applications, 2008. RTCSA '08. 14th IEEE International Conference on*, pages 367–376, Aug 2008.

[5] R. Alur, C. Courcoubetis, and D. Dill. Model-checking for real-time systems. In *Logic in Computer Science, 1990. LICS '90, Proceedings., Fifth Annual IEEE Symposium on e*, pages 414–425, Jun 1990.

[6] Rajeev Alur and David L. Dill. A theory of timed automata. *Theor. Comput. Sci.*, 126(2):183–235, April 1994.

[7] François Baccelli, Guy Cohen, Geert Jan Olsder, and Jean-Pierre Quadrat. Synchronization and linearity: an algebra for discrete event systems, 2001.

[8] Brian Bailey, Grant Martin, and Thomas Anderson. *Taxonomies for the Development and Verification of digital systems.* Springer US, 2005.

[9] Shuvra S. Battacharyya, Edward A. Lee, and Praveen K. Murthy. *Software Synthesis from Dataflow Graphs.* Kluwer Academic Publishers, Norwell, MA, USA, 1996.

[10] Christian Bauer, Alexander Frink, and Richard Kreckel. Symbolic computation and automated reasoning. chapter The GiNaC Framework for Symbolic Computation... (Poster Session), pages 241–242. A. K. Peters, Ltd., Natick, MA, USA, 2001.

[11] Christian Bauer, Alexander Frink, and Richard Kreckel. Introduction to the ginac framework for symbolic computation within the C++ programming language. *J. Symb. Comput.*, 33(1):1–12, January 2002.

[12] V. Bebelis, P. Fradet, A. Girault, and B. Lavigueur. BPDF: A statically analyzable dataflow model with integer and boolean parameters. In *Embedded Software (EMSOFT), 2013 Proceedings of the International Conference on*, pages 1–10, Sept 2013.

[13] Vagelis Bebelis, Pascal Fradet, and Alain Girault. A framework to schedule parametric dataflow applications on many-core platforms. In *Proceedings of the 2014 SIGPLAN/SIGBED Conference on Languages, Compilers and Tools for Embedded Systems*, LCTES '14, pages 125–134, New York, NY, USA, 2014. ACM.

[14] Gerd Behrmann, Alexandre David, and Kim Guldstrand Larsen. A tutorial on Uppaal. In Marco Bernardo and Flavio Corradini, editors, *Formal Methods for the Design of Real-Time Systems, International School on Formal Methods for the Design of Computer, Communication and Software Systems, SFM-RT 2004, Bertinoro, Italy, September 13-18, 2004, Revised Lectures*, volume 3185 of *Lecture Notes in Computer Science*. Springer, 2004.

[15] Evangelos Bempelis. *Boolean Parametric Data Flow Modeling - Analyses - Implementation.* Theses, Université Grenoble Alpes, February 2015.

[16] B. Bhattacharya and S.S. Bhattacharyya. Parameterized dataflow modeling for DSP systems. *Signal Processing, IEEE Transactions on*, 49(10):2408–2421, Oct 2001.

[17] Bishnupriya Bhattacharya and Shuvra S. Bhattacharyya. Quasi-static scheduling of reconfigurable dataflow graphs for DSP systems. In *Proceedings of the 11th IEEE International Workshop on Rapid System*

Prototyping (RSP 2000), Paris, France, June 21-23, 2000, pages 84–89, 2000.

[18] Shuvra S. Bhattacharyya, Ed F. Deprettere, and Bart D. Theelen. Dynamic dataflow graphs. In Shuvra S. Bhattacharyya, Ed F. Deprettere, Rainer Leupers, and Jarmo Takala, editors, *Handbook of Signal Processing Systems*, pages 905–944. Springer New York, 2013.

[19] G. Bilsen, M. Engels, R. Lauwereins, and J. Peperstraete. Cycle-static dataflow. *Signal Processing, IEEE Transactions on*, 44(2):397–408, Feb 1996.

[20] J. Buck and R. Vaidyanathan. Heterogeneous modeling and simulation of embedded systems in El Greco. In *Hardware/Software Codesign, 2000. CODES 2000. Proceedings of the Eighth International Workshop on*, pages 142–146, May 2000.

[21] Joseph T. Buck. *Scheduling dynamic dataflow graphs with bounded memory using the token flow model.* PhD thesis, EECS Department, University of California, Berkeley, 1993.

[22] J.T. Buck and E.A. Lee. Scheduling dynamic dataflow graphs with bounded memory using the token flow model. In *Acoustics, Speech, and Signal Processing, 1993. ICASSP-93., 1993 IEEE International Conference on*, volume 1, pages 429–432 vol.1, April 1993.

[23] Florian Bugarin, Didier Henrion, and Jean-Bernard Lasserre. Minimizing the sum of many rational functions. *arXiv preprint arXiv:1102.4954*, 2011.

[24] T.D. Burd and R.W. Brodersen. Energy efficient cmos microprocessor design. In *System Sciences, 1995. Proceedings of the Twenty-Eighth Hawaii International Conference on*, volume 1, pages 288–297 vol.1, Jan 1995.

[25] Anantha P Chandrakasan, Samuel Sheng, and Robert W Brodersen. Low-power CMOS digital design. *IEICE Transactions on Electronics*, 75(4):371–382, 1992.

[26] Bernadette Charron-Bost, Matthias Fgger, and Thomas Nowak. Transience bounds for distributed algorithms. In Victor Braberman and Laurent Fribourg, editors, *Formal Modeling and Analysis of Timed Systems*, volume 8053 of *Lecture Notes in Computer Science*, pages 77–90. Springer Berlin Heidelberg, 2013.

[27] Philippe Clauss and Vincent Loechner. Parametric analysis of polyhedral iteration spaces. *Journal of VLSI signal processing systems for signal, image and video technology*, 19(2):179–194, 1998.

[28] Jean Cochet-Terrasson, Guy Cohen, Stéphane Gaubert, Michael McGettrick, and Jean-Pierre Quadrat. Numerical computation of spectral elements in max-plus algebra. In *Proc. IFAC Conf. on Syst. Structure and Control*, 1998.

[29] Gary Cohen, Didier Dubois, Jean-Pierre Quadrat, and Michel Viot. Analyse du comportement périodique de systèmes de production par la théorie des dioïdes. Technical Report 191, INRIA, 1983.

[30] Guy Cohen, Stéphane Gaubert, and Jean-Pierre Quadrat. Max-plus algebra and system theory: Where we are and where to go now. *Annual Reviews in Control*, 23:207 – 219, 1999.

[31] Nathalie Cossement, Rudy Lauwereins, and Francky Catthoor. DF*: An extension of synchronous dataflow with data dependency and non-determinism. In *Forum on Design Languages, 2000. FDL 2000.*, 2000.

[32] M. Damavandpeyma, S. Stuijk, T. Basten, M. Geilen, and H. Corporaal. Throughput-constrained DVFS for scenario-aware dataflow graphs. In *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2013 IEEE 19th*, pages 175–184, April 2013.

[33] M. Damavandpeyma, S. Stuijk, M. Geilen, T. Basten, and H. Corporaal. Parametric throughput analysis of scenario-aware dataflow graphs. In *Computer Design (ICCD), 2012 IEEE 30th International Conference on*, pages 219–226, Sept 2012.

[34] A. Dasdan and R.K. Gupta. Faster maximum and minimum mean cycle algorithms for system-performance analysis. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 17(10):889–899, Oct 1998.

[35] K. Desnos, M. Pelcat, J.-F. Nezan, S.S. Bhattacharyya, and S. Aridhi. PiMM: Parameterized and interfaced dataflow meta-model for mpsocs runtime reconfiguration. In *Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS XIII), 2013 International Conference on*, pages 41–48, July 2013.

[36] C. Ebert and C. Jones. Embedded software: Facts, figures, and future. *Computer*, 42(4):42–52, April 2009.

[37] M. Fakih, K. Gruttner, M. Franzle, and A. Rettberg. Towards performance analysis of SDFGs mapped to shared-bus architectures using model-checking. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2013*, pages 1167–1172, March 2013.

[38] Pascal Fradet, Alain Girault, and Peter Poplavko. SPDF: A schedulable parametric data-flow moc. In *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '12, pages 769–774, San Jose, CA, USA, 2012. EDA Consortium.

[39] Komei Fukuda. Cddlib reference manual. *Report version 093a, McGill University, Montréal, Quebec, Canada*, 2003.

[40] S. Gaubert. Performance evaluation of (max,+) automata. *Automatic Control, IEEE Transactions on*, 40(12):2014–2025, Dec 1995.

[41] S. Gaubert and J. Mairesse. Modeling and analysis of timed petri nets using heaps of pieces. *Automatic Control, IEEE Transactions on*, 44(4):683–697, Apr 1999.

[42] Stéphane Gaubert, Peter Butkovic, and Raymond Cuninghame-Green. Minimal (max,+) realization of convex sequences. *SIAM Journal on Control and Optimization*, 36(1):137–147, 1998.

[43] Marc Geilen. Reduction techniques for synchronous dataflow graphs. In *Proceedings of the 46th Annual Design Automation Conference*, DAC '09, pages 911–916, New York, NY, USA, 2009. ACM.

[44] Marc Geilen. Synchronous dataflow scenarios. *ACM Trans. Embed. Comput. Syst.*, 10(2):16:1–16:31, January 2011.

[45] Marc Geilen and Twan Basten. Reactive process networks. In *Proceedings of the 4th ACM International Conference on Embedded Software*, EMSOFT '04, pages 137–146, New York, NY, USA, 2004. ACM.

[46] Marc Geilen and Sander Stuijk. Worst-case performance analysis of synchronous dataflow scenarios. In *Proceedings of the Eighth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, CODES/ISSS '10, pages 125–134, New York, NY, USA, 2010. ACM.

[47] Marc Geilen, Stavros Tripakis, and Maarten Wiggers. The earlier the better: A theory of timed actor interfaces. Technical Report UCB/EECS-2010-130, EECS Department, University of California, Berkeley, Oct 2010.

[48] A. H. Ghamarian, M. C. W. Geilen, T. Basten, and S. Stuijk. Parametric throughput analysis of synchronous data flow graphs. In *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '08, pages 116–121, New York, NY, USA, 2008. ACM.

[49] A. H. Ghamarian, M. C. W. Geilen, S. Stuijk, T. Basten, B. D. Theelen, M. R. Mousavi, A. J. M. Moonen, and M. J. G. Bekooij. Throughput analysis of synchronous data flow graphs. In *Proceedings of the Sixth International Conference on Application of Concurrency to System Design*, ACSD '06, pages 25–36, Washington, DC, USA, 2006. IEEE Computer Society.

[50] A.H. Ghamarian, M.C.W. Geilen, S. Stuijk, T. Basten, A.J.M. Moonen, M.J.G. Bekooij, B.D. Theelen, and M.R. Mousavi. Throughput analysis of synchronous data flow graphs. In *Application of Concurrency to System Design, 2006. ACSD 2006. Sixth International Conference on*, pages 25–36, June 2006.

[51] A.H. Ghamarian, S. Stuijk, T. Basten, M.C.W. Geilen, and B.D. Theelen. Latency minimization for synchronous data flow graphs. In *Digital System Design Architectures, Methods and Tools, 2007. DSD 2007. 10th Euromicro Conference on*, pages 189–196, Aug 2007.

[52] Stefan Valentin Gheorghita, Martin Palkovic, Juan Hamers, Arnout Vandecappelle, Stelios Mamagkakis, Twan Basten, Lieven Eeckhout, Henk Corporaal, Francky Catthoor, Frederik Vandeputte, and Koen De Bosschere. System-scenario-based design of dynamic embedded systems. *ACM Trans. Des. Autom. Electron. Syst.*, 14(1):3:1–3:45, January 2009.

[53] A. Girault, Bilung Lee, and E.A. Lee. Hierarchical finite state machines with multiple concurrency models. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 18(6):742–760, Jun 1999.

[54] Laurent Granvilliers and Frédéric Benhamou. Algorithm 852: RealPaver: An interval solver using constraint satisfaction techniques. *ACM Trans. Math. Softw.*, 32(1):138–156, March 2006.

[55] D. Grois and O. Hadar. Recent advances in region-of-interest coding. *Recent Advances on Video Coding*, pages 49–76, 2011.

[56] T. Grotker, R. Schoenen, and H. Meyr. PCC: A modeling technique for mixed control/data flow systems. In *Proceedings of the 1997 European Conference on Design and Test*, EDTC '97, pages 482–, Washington, DC, USA, 1997. IEEE Computer Society.

[57] Ruirui Gu, J.W. Janneck, S.S. Bhattacharyya, M. Raulet, M. Wipliez, and W. Plishker. Exploring the concurrency of an MPEG RVC decoder based on dataflow program analysis. *Circuits and Systems for Video Technology, IEEE Transactions on*, 19(11):1646–1657, Nov 2009.

[58] Soonhoi Ha and Hyunok Oh. Decidable dataflow models for signal processing: Synchronous dataflow and its extensions. In Shuvra S. Bhattacharyya, Ed F. Deprettere, Rainer Leupers, and Jarmo Takala, editors, *Handbook of Signal Processing Systems*, pages 1083–1109. Springer New York, 2013.

[59] E. Hammari, F. Catthoor, J. Huisken, and P. G. Kjeldsberg. Application of medium-grain multiprocessor mapping methodology to epileptic seizure predictor. In *NORCHIP, 2010*, pages 1–6, Nov 2010.

[60] Elena Hammari, Francky Catthoor, Per Gunnar Kjeldsberg, Jos Huisken, K Tsakalis, and L Iasemidis. Identifying data-dependent system scenarios in a dynamic embedded system. In *Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA)*, page 1. The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), 2012.

[61] Bernd Heidergott, Geert Jan Olsder, and Jacob Van Der Woude. *Max Plus at Work: Modeling and Analysis of Synchronized Systems: A Course on Max-Plus Algebra and Its Applications.* Princeton University Press, 2006.

[62] Thomas Hune, Judi Romijn, Mariëelle Stoelinga, and Frits Vaandrager. *Tools and Algorithms for the Construction and Analysis of Systems: 7th International Conference, TACAS 2001 Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2001 Genova, Italy, April 2–6, 2001 Proceedings*, chapter Linear Parametric Model Checking of Timed Automata, pages 189–203. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001.

[63] Axel Jantsch. *Modeling embedded systems and SoCs: concurrency and time in models of computation.* Morgan Kaufmann, 2004.

[64] N.K. Jha. Low power system scheduling and synthesis. In *Computer Aided Design, 2001. ICCAD 2001. IEEE/ACM International Conference on*, pages 259–263, Nov 2001.

[65] G. Kahn. The semantics of a simple language for parallel programming. In J. L. Rosenfeld, editor, *Information processing*, pages 471–475, Stockholm, Sweden, Aug 1974. North Holland, Amsterdam.

[66] J.-P. Katoen and Hao Wu. Exponentially timed SADF: Compositional semantics, reductions, and analysis. In *Embedded Software (EMSOFT), 2014 International Conference on*, pages 1–10, Oct 2014.

[67] K.M. Kavi, B.P. Buckles, and U. Narayan Bhat. A formal definition of data flow graph models. *Computers, IEEE Transactions on*, C-35(11):940–948, Nov 1986.

[68] Phillip A. Laplante and Seppo J. Ovaska. *Real-time systems design and analysis: tools for the practitioner.* John Wiley and Sons, 2011.

[69] E. Lee and D.G. Messerschmitt. Static scheduling of synchronous data flow programs for digital signal processing. *Computers, IEEE Transactions on*, C-36(1):24–35, Jan 1987.

[70] E.A. Lee and D.G. Messerschmitt. Synchronous data flow. *Proceedings of the IEEE*, 75(9):1235–1245, Sept 1987.

[71] E.A. Lee and A. Sangiovanni-Vincentelli. A framework for comparing models of computation. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 17(12):1217–1229, Dec 1998.

[72] Edward A. Lee. A denotational semantics for dataflow with firing. In *Memorandum UCB/ERL M97/3, Electronics Research Labaratory, Berkeley, CA 94720*, 1997.

[73] Björn Lisper. Fully automatic, parametric worst-case execution time analysis. In *In Workshop on Worst-Case Execution Time (WCET) Analysis*, pages 99–102, 2003.

[74] Marco Locatelli and Fabio Schoen. *Global optimization: theory, algorithms, and applications*, volume 15. SIAM, 2013.

[75] Zhe Ma, Pol Marchal, Daniele Paolo Scarpazza, Peng Yang, Chun Wong, Jos Ignacio Gmez, Stefaan Himpe, Chantal Ykman Couvreur, and Francky Catthoor. *Systematic Methodology for Real-Time Cost-Effective Mapping of Dynamic Concurrent Task-Based Systems on Heterogeneous Platforms.* Springer Netherlands, 2007.

[76] Stylianos Mamagkakis, Dimitrios Soudris, and Francky Catthoor. Middleware design optimization of wireless protocols based on the exploitation of dynamic input patterns. In *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '07, pages 1036–1041, San Jose, CA, USA, 2007. EDA Consortium.

[77] Jianfeng Mao, Qianchuan Zhao, and C.G. Cassandras. Optimal dynamic voltage scaling in power-limited systems with real-time constraints. In *Decision and Control, 2004. CDC. 43rd IEEE Conference on*, volume 2, pages 1472–1477 Vol.2, Dec 2004.

[78] Lei Miao and C.G. Cassandras. Optimality of static control policies in some discrete-event systems. *Automatic Control, IEEE Transactions on*, 50(9):1427–1431, Sept 2005.

[79] Narasinga Rao Miniskar, Elena Hammari, Satyakiran Munaga, Stylianos Mamagkakis, Per Gunnar Kjeldsberg, and Francky Catthoor. *Embedded Computer Systems: Architectures, Modeling, and Simulation: 9th International Workshop, SAMOS 2009, Samos, Greece, July 20-23, 2009. Proceedings*, chapter Scenario Based Mapping of Dynamic Applications on MPSoC: A 3D Graphics Case Study, pages 48–57. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[80] T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, Apr 1989.

[81] S. Neuendorffer and E. Lee. Hierarchical reconfiguration of dataflow models. In *Formal Methods and Models for Co-Design, 2004. MEMOCODE '04. Proceedings. Second ACM and IEEE International Conference on*, pages 179–188, June 2004.

[82] Gabriela Nicolescu and Pieter J. Mosterman. *Model-Based Design for Embedded Systems.* CRC Press, Inc., Boca Raton, FL, USA, 1st edition, 2009.

[83] C. Norstrøm, A. Wall, and Wang Yi. Timed automata as task models for event-driven systems. In *Real-Time Computing Systems and*

Applications, 1999. RTCSA '99. Sixth International Conference on, pages 182–189, 1999.

[84] J. Ostermann, J. Bormans, P. List, D. Marpe, M. Narroschke, F. Pereira, T. Stockhammer, and T. Wedi. Video coding with H.264/AVC: tools, performance, and complexity. *Circuits and Systems Magazine, IEEE*, 4(1):7–28, First 2004.

[85] M. Pankert, O. Mauss, S. Ritz, and H. Meyr. Dynamic data flow and control flow in high level DSP code synthesis. In *Acoustics, Speech, and Signal Processing, 1994. ICASSP-94., 1994 IEEE International Conference on*, volume ii, pages II/449–II/452 vol.2, Apr 1994.

[86] J. Piat, S.S. Bhattacharyya, and M. Raulet. Interface-based hierarchy for synchronous data-flow graphs. In *Signal Processing Systems, 2009. SiPS 2009. IEEE Workshop on*, pages 145–150, Oct 2009.

[87] P. Poplavko, T. Basten, M. Bekooij, J. van Meerbergen, and B. Mesman. Task-level timing models for guaranteed performance in multiprocessor networks-on-chip. In *Proceedings of the 2003 International Conference on Compilers, Architecture and Synthesis for Embedded Systems*, CASES '03, pages 63–72, New York, NY, USA, 2003. ACM.

[88] P. Poplavko, T. Basten, and J. van Meerbergen. Execution-time prediction for dynamic streaming applications with task-level parallelism. In *Digital System Design Architectures, Methods and Tools, 2007. DSD 2007. 10th Euromicro Conference on*, pages 228–235, Aug 2007.

[89] Michael James David Powell. *Approximation theory and methods.* Cambridge university press, 1981.

[90] Claudius Ptolemaeus, editor. *System Design, Modeling, and Simulation using Ptolemy II.* Ptolemy.org, 2014.

[91] Alberto Sangiovanni-Vincentelli and Grant Martin. Platform-based design and software design methodology for embedded systems. *IEEE Des. Test*, 18(6):23–33, November 2001.

[92] Alberto Sangiovanni-Vincentelli, Haibo Zeng, Marco Di Natale, and Peter Marwedel, editors. *Embedded Systems Development: From Functional Models to Implementations*, volume 20 of *Embedded Systems.* Springer New York, 2014.

[93] M. Sgroi, L. Lavagno, Y. Watanabe, and A. Sangiovanni-Vincentelli. Synthesis of embedded software using free-choice petri nets. In *Design Automation Conference, 1999. Proceedings. 36th*, pages 805–810, 1999.

[94] Hanif D. Sherali and Cihan H. Tuncbilek. A global optimization algorithm for polynomial programming problems using a reformulation-linearization technique. *Journal of Global Optimization*, 2(1):101–112, 1992.

[95] F. Siyoum, M. Geilen, and H. Corporaal. End-to-end latency analysis of dataflow scenarios mapped onto shared heterogeneous resources. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 35(4):535–548, April 2016.

[96] F. Siyoum, M. Geilen, J. Eker, C. von Platen, and H. Corporaal. Automated extraction of scenario sequences from disciplined dataflow networks. In *Formal Methods and Models for Codesign (MEMOCODE), 2013 Eleventh IEEE/ACM International Conference on*, pages 47–56, Oct 2013.

[97] F. Siyoum, M. Geilen, O. Moreira, R. Nas, and H. Corporaal. Analyzing synchronous dataflow scenarios for dynamic software-defined radio applications. In *System on Chip (SoC), 2011 International Symposium on*, pages 14–21, Oct 2011.

[98] Firew Siyoum, Marc Geilen, Orlando Moreira, and Henk Corporaal. Worst-case throughput analysis of real-time dynamic streaming applications. In *Proceedings of the Eighth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, CODES+ISSS '12, pages 463–472, New York, NY, USA, 2012. ACM.

[99] M. Skelin, M. Geilen, F. Catthoor, and S. Hendseth. Worst-case latency analysis of sdf-based parametrized dataflow mocs. In *Design and Architectures for Signal and Image Processing (DASIP), 2015 Conference on*, pages 1–6, Sept 2015.

[100] M. Skelin, M. Geilen, F. Catthoor, and S. Hendseth. Worst-case performance analysis for SDF-based parameterize dataflow. 2015. Under review as an invited paper for publication in special issue of Elsevier's Microprocessors and Microsystems journal (MICPRO) on DSD 2015.

[101] M. Skelin, M. Geilen, F. Catthoor, and S. Hendseth. Worst-case throughput analysis of SDF-based parametrized dataflow. In *Digital System Design (DSD), 2015 Euromicro Conference on*, pages 17–24, Aug 2015.

[102] M. Skelin, M. Geilen, F. Catthoor, and S. Hendseth. Parameterized dataflow scenarios. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, PP(99):1–1, 2016.

[103] M. Skelin, E. R. Wognsen, M. C. Olesen, R. R. Hansen, and K. G. Larsen. Model checking of finite-state machine-based scenario-aware dataflow using timed automata. In *Industrial Embedded Systems (SIES), 2015 10th IEEE International Symposium on*, pages 1–10, June 2015.

[104] M. Skelin, E.R. Wognsen, M.C. Olesen, R.R. Hansen, and K.G. Larsen. Towards translating FSM-SADF to timed automata. In *Proceedings of the first international workshop on Investigating dataflow in embedded computing architectures, IDEA 2015, Amsterdam, The Netherlands, January 19 - 21, 2015*, pages 13–16, 2015.

[105] Mladen Skelin, Marc Geilen, Francky Catthoor, and Sverre Hendseth. Worst-case throughput analysis for parametric rate and parametric actor execution time scenario-aware dataflow graphs. In *Proceedings 1st International Workshop on Synthesis of Continuous Parameters, SynCoP 2014, Grenoble, France, 6th April 2014.*, pages 65–79, 2014.

[106] Mladen Skelin, Marc Geilen, Francky Catthoor, and Sverre Hendseth. Parametrized dataflow scenarios. In *Proceedings of the 12th International Conference on Embedded Software*, EMSOFT '15, pages 95–104, Piscataway, NJ, USA, 2015. IEEE Press.

[107] Sundararajan Sriram and Shuvra S. Bhattacharyya. *Embedded Multiprocessors: Scheduling and Synchronization.* Marcel Dekker, Inc., New York, NY, USA, 1st edition, 2000.

[108] S. Stuijk, M. Geilen, and T. Basten. Throughput-buffering trade-off exploration for cyclo-static and synchronous dataflow graphs. *Computers, IEEE Transactions on*, 57(10):1331–1345, Oct 2008.

[109] S. Stuijk, M. Geilen, and T. Basten. A predictable multiprocessor design flow for streaming applications with dynamic behaviour. In *Digital System Design: Architectures, Methods and Tools (DSD), 2010 13th Euromicro Conference on*, pages 548–555, Sept 2010.

[110] S. Stuijk, M. Geilen, B. Theelen, and T. Basten. Scenario-aware dataflow: Modeling, analysis and implementation of dynamic applications. In *Embedded Computer Systems (SAMOS), 2011 International Conference on*, pages 404–411, July 2011.

[111] S Stuijk, AH Ghamarian, BD Theelen, MCW Geilen, and T Basten. FSM-based SADF. Technical report, Eindhoven University of Technology, 2008. MNEMEE internal report.

[112] Sander Stuijk, Marc Geilen, and Twan Basten. SDF^3: SDF for free. *2010 10th International Conference on Application of Concurrency to System Design*, 0:276–278, 2006.

[113] B. D. Theelen, M. C. W. Geilen, T. Basten, J. P. M. Voeten, S. V. Gheorghita, and S. Stuijk. 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. Proceedings.*, MEMOCODE '06, pages 185–194, Washington, DC, USA, 2006. IEEE Computer Society.

[114] Bart D. Theelen, Marc Geilen, and Jeroen Voeten. Performance Model Checking Scenario-Aware Dataflow. In *Proceedings of the 9th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS)*, volume 6919 of *Lecture Notes in Computer Science*, Aalborg, Denmark, 2011. Springer.

[115] Bart D. Theelen, Joost-Pieter Katoen, and Hao Wu. Model checking of Scenario-Aware Dataflow with CADP. In Wolfgang Rosenstiel and Lothar Thiele, editors, *Proceedings of the 2012 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Dresden, Germany, 2012. IEEE.

[116] B.D. Theelen. A performance analysis tool for scenario-aware streaming applications. In *Quantitative Evaluation of Systems, 2007. QEST 2007. Fourth International Conference on the*, pages 269–270, Sept 2007.

[117] L. Thiele, K. Strehl, D. Ziegengein, R. Ernst, and J. Teich. Funstate-an internal design representation for codesign. In *Computer-Aided Design, 1999. Digest of Technical Papers. 1999 IEEE/ACM International Conference on*, pages 558–565, Nov 1999.

[118] Filip Thoen and Francky Catthoor. *Modeling, Verification and Exploration of Task-Level Concurrency in Real-Time Embedded Systems.* Springer US, 2000.

[119] Maarten H. Wiggers, Marco J. G. Bekooij, and Gerard J. M. Smit. Modelling run-time arbitration by latency-rate servers in dataflow graphs. In *Proceedingsof the 10th International Workshop on Software &Amp; Compilers for Embedded Systems*, SCOPES '07, pages 11–22, New York, NY, USA, 2007. ACM.

[120] Maarten H. Wiggers, Marco J. G. Bekooij, and Gerard J. M. Smit. Buffer capacity computation for throughput-constrained modal task graphs. *ACM Trans. Embed. Comput. Syst.*, 10(2):17:1–17:59, January 2011.

[121] M.H. Wiggers, M.J.G. Bekooij, and G.J.M. Smit. Buffer capacity computation for throughput constrained streaming applications with data-dependent inter-task communication. In *Real-Time and Embedded Technology and Applications Symposium, 2008. RTAS '08. IEEE*, pages 183–194, April 2008.

[122] Peng Yang, Paul Marchal, Chun Wong, Stefaan Himpe, Francky Catthoor, Patrick David, Johan Vounckx, and Rudy Lauwereins. Managing dynamic concurrent tasks in embedded real-time multimedia systems. In *Proceedings of the 15th International Symposium on System Synthesis*, ISSS '02, pages 112–119, New York, NY, USA, 2002. ACM.

[123] Yang Yang, M. Geilen, T. Basten, S. Stuijk, and H. Corporaal. Exploring trade-offs between performance and resource requirements for synchronous dataflow graphs. In *Embedded Systems for Real-Time Multimedia, 2009. ESTIMedia 2009. IEEE/ACM/IFIP 7th Workshop on*, pages 96–105, Oct 2009.

[124] Richard Zurawski. *Embedded systems handbook.* CRC Press, 2005.

Abbreviations

BDF	boolean dataflow
BPDF	boolean parametric dataflow
CSDF	cyclo-static dataflow
CSDF-PDF	CSDF-based PDF
DDF	dynamic dataflow
FSM	finite-state machine
FSM-SADFG	FSM-SADF graph
FSM-SADF	FSM-based scenario-aware dataflow
HDF	heterochronous dataflow
KPN	Kahn process network
MCME	MCM expression
MoC	model of computation
PDFG	PDF graph
PDF	parameterized dataflow
PFSM-SADFG	PFSM-SADF graph
PFSM-SADF	FSM-based parameterized scenario-aware dataflow
PiMM	parameterized and interfaced dataflow meta-model
PSDF	parameterized SDF
PTA	parametric timed automata
RLT	reformulation-linearization technique
RPN	reactive process network
SADF	scenario-aware dataflow
SDF^3	SDF for free
SDF	synchronous dataflow
SDF-PDFG	SDF-PDF graph
SDF-PDF	SDF-based parameterized dataflow
SDF-PFSM-SADFG	SDF-PFSM-SADF graph
SDF-PFSM-SADF	SDF-based PFSM-SADF

SPDF	schadulable parametric dataflow
SSDF	scalable SDF
VPDF	variable-rate phased dataflow
VRDF	variable-rate dataflow
CMOS	complementary metal oxide semiconductor
DSE	design space exploration
DSP	digital signal processor
DVFS	dynamic voltage and frequency scaling
FIFO	first in first out
MCM	maximum cycle mean
QoS	quality of service
Qss	quasi-static schedule
SDFG	SDF graph
SoC	system on chip
TA	timed automata
WCET	worst-case execution time

Symbols

I	set of initial tokens of a dataflow graph
B	buffers of an FSM-SADFG
$\phi(b)$	status of FSM-SADFG buffer b
$\mathbf{x}$	configuration of an SDF-PFSM-SADFG
$\overline{x}$	configuration sequence of an SDF-PDFG
$R_c(b, s)$	consumption rate of FSM-SADFG kernel consuming from buffer b in scenario s
$\psi(k)$	status of the control buffer of FSM-SADFG kernel k
d	detector of an FSM-SADFG
δ	status of an FSM-SADFG detector
$E(p)$	firing delay of FSM-SADFG process p
F	FSM-SADF scenario FSM
F	FSM-SADF
$\mathcal{S}_A$	scenario set of FSM-SADF actor A
$\mathcal{S}$	set of FSM-SADF scenarios
$mcm(\mathcal{G})$	maximum cycle mean of weighted directed graph $\mathcal{G}$
$In(p)$	input buffers of FSM-SADFG process p
i_l	initial token of a dataflow graph with index l
$\mathcal{K}$	kernels of an FSM-SADFG
$\kappa(k)$	status of FSM-SADFG kernel k
$L_{\mathsf{F}^\mathrm{P}}$	worst-case latency of SDF-PFSM-SADF F^P
L_G	worst-case latency of SDF-PDFG G
$L'_{\mathsf{F}^\mathrm{P}}$	conservative bound to $L_{\mathsf{F}^\mathrm{P}}$
L'_G	conservative bound to L_G
$\tilde{\mathbf{x}}$	linearized configuration of an SDF-PFSM-SADFG
$\mathcal{P}$	set of arbitrary parameters
$cod(f)$	codomain of mapping f

$dom(f)$	domain of mapping f
$conv(\varsigma_1(n), \varsigma_2(n))$	Max-plus convolution of Max-plus sequences $\varsigma_1(n)$ and $\varsigma_2(n)$
t_{i_l}	production time of i_l after the kth graph iteration
t'_{i_l}	production time of i_l after the $(k+1)$st graph iteration
$\gamma(k)$	vector of production times of initial tokens after the kth graph iteration or shortly the Max-plus timestamp vector
$\mathbf{M}_{\mathbf{x}_r}$	MCME for SDF-PFSM-SADFG configuration $\mathbf{x}_r$
$Out(k)$	output buffers of FSM-SADFG kernel k
$\overline{\gamma}$	sequence of Max-plus timestamp vectors
x^A	configuration of PDF/SDF-PDF actor A
$\mathcal{D}_A$	grammar used to define the firing delay of SDF-PDF actor A
X_A	domain of PDF/SDF-PDF actor A
$X_{s_j^{\mathrm{P}}}$	domain of parameterized scenario s_j^{P}
$\mathcal{P}_d$	firing delay parameters of an SDF-PDFG
$d(A)$	firing delay of SDF-PDFG/SDFG actor A
$\mathcal{R}_A$	grammar used to define rates of SDF-PDF actor A
$\mathcal{P}_i$	rate parameters of an SDF-PDFG
$r(A, c)$	rate of SDF-PDFG/SDFG port (A, c)
F^{P}	PFSM-SADF scenario FSM
$\zeta_{\mathsf{F}^{\mathrm{P}}}(k)$	the scenario and the configuration of the kth SDF-PFSM-SADFG iteration
ξ_0^{P}	initial state of F^{P}
$\mathbb{M}_{\mathsf{F}^{\mathrm{P}}}(s_j^{\mathrm{P}})$	(returns) mapping $\mathcal{M}^{\mathrm{par}}_{s_j^{\mathrm{P}}}$
F^{P}	PFSM-SADF
s_j^{P}	PFSM-SADF scenario
Ψ^{P}	scenario labeling of F^{P}
$\mathcal{S}^{\mathrm{P}}$	set of PFSM-SADF scenarios
$\mathsf{M}_{\mathsf{F}^{\mathrm{P}}}$	set of mappings $\{\mathcal{M}^{\mathrm{par}}_{s_j^{\mathrm{P}}} : s_j^{\mathrm{P}} \in \mathcal{S}^{\mathrm{P}}\}$
ξ_j^{P}	state of F^{P}
Ξ^{P}	states of F^{P}
$\mathbb{T}^{\mathrm{P}}$	transition function of F^{P}

$\hat{\mathbf{x}}$	polynomized configuration of an SDF-PFSM-SADFG
P	process of an FSM-SADFG
$R_p(b, s)$	production rate of FSM-SADFG kernel producing to buffer b in scenario s
$\tau(A, n)$	production timestamp of tokens produced by the nth firing of actor A
$\upsilon(p)$	realized auto-concurrency of FSM-SADFG kernel p
M_G	Max-plus matrix of SDFG G
$\zeta_G(k)$	configuration determining the instance running as kth SDF-PDFG iteration
$\mathcal{P}_b$	boolean parameters of an SDF-PDFG
$\beta(c)$	condition of SDF-PDFG channel c
$\iota_G(x^G)$	instance of PDFG/SDF-PDFG G determined by configuration x^G
$\mathcal{M}_G(x^G)$	Max-plus matrix of SDF-PDFG G instance $\iota_G(x^G)$
$\mathcal{M}^{\mathrm{par}\prime}_{s^{\mathrm{P}}_k}(\tilde{\mathbf{x}}_r)$	conservative estimate of $\tilde{\mathcal{M}}^{\mathrm{par}}_{s^{\mathrm{P}}_k}(\tilde{\mathbf{x}}_r)$
$\tilde{\mathcal{M}}^{\mathrm{par}}_{s^{\mathrm{P}}_k}(\tilde{\mathbf{x}}_r)$	linearized parameterized matrix of scenario s^{P}_k valid for linearized SDF-PFSM-SADFG configuration $\tilde{\mathbf{x}}_r$
$X_{\mathsf{F}^{\mathrm{P}}}$	domain of an PFSM-SADFG
$\hat{X}_{\mathsf{F}^{\mathrm{P}}}$	polynomized domain of an SDF-PFSM-SADFG
$\overline{s}^{\mathrm{P}}$	sequence of parameterized scenarios
E	set of all arithmetic expressions defined on $(\mathcal{P}_i \cup \mathcal{P}_d \cup \mathbb{R}_{\max})$
$Th_{\mathsf{F}^{\mathrm{P}}}$	worst-case throughput of SDF-PFSM-SADF F^{P}
Th_G	worst-case throughput of SDF-PDFG G
$Th'_{\mathsf{F}^{\mathrm{P}}}$	conservative bound to $Th_{\mathsf{F}^{\mathrm{P}}}$
Th'_G	conservative bound to Th_G
$\mathbf{Th}_{\mathbf{x}_r}$	throughput expression for SDF-PFSM-SADFG configuration $\mathbf{x}_r$
$Th_{\mathsf{F}^{\mathrm{P}}}(\mathbf{x}_r)$	throughput value for SDF-PFSM-SADFG configuration $\mathbf{x}_r$
$\mathbb{A}$	timed automaton
$M^{\mathrm{w-c}}_{s^{\mathrm{P}}_j}$	worst-case evaluation Max-plus matrix of scenario s^{P}_j SDF-PDFG

$M_G^{\mathrm{w-c}}$	worst-case evaluation matrix of SDF-PDFG G
$M_{s_j^{\mathrm{P}}}^{\mathrm{w-c}\prime}$	conservative estimate of $M_{s_j^{\mathrm{P}}}^{\mathrm{w-c}}$
$M_G^{\mathrm{w-c}\prime}$	conservative estimate of $M_G^{\mathrm{w-c}}$
$d_A(x^A)$	firing delay of SDF-PDF actor A with configuration x^A
$d_A(s_A)$	firing delay of FSM-SADF actor A in scenario s_A
$r_A(p, x^A)$	rate of SDF-PDF actor A's port p with configuration x^A
$r_A(p, s_A)$	rate of FSM-SADF actor A's port p in scenario s_A
$\mathcal{P}_{Ai}$	rate parameters of SDF-PDF actor A
f^T	timed actor firing function
f	actor firing function
R	actor firing rules
P	set of actor input ports
Q	set of actor output ports
G	dataflow graph
$\mathcal{A}$	set of PDF/SDF-PDFG/SDFG graph actors
C	set of PDF/SDF-PDFG/SDFG graph channels
$dst(c)$	destination actor of channel c
$\zeta_{\mathsf{F}}(k)$	scenario of the kth FSM-SADFG iteration
$\mathcal{M}_{\mathsf{F}}(s)$	Max-plus matrix of FSM-SADF scenario s SDFG
ξ_0	initial state of F
$\mathcal{A}$	Max-plus automaton
α	final delay of a Max-plus automaton
β	initial delay of a Max-plus automaton
μ	Max-plus automaton morphism
Φ	scenario labeling of F
Ξ	states of F
$\mathbb{T}$	transition function of F
π_l	left projection function
a^{norm}	normalized Max-plus vector a
x^G	configuration of SDF-PDFG G
X_G	domain of SDF-PDFG G
$i(c)$	number of initial tokens on SDF-PDFG/SDFG channel c

$\mathcal{A}^{\mathrm{P}}$	Max-plus automaton
α^{P}	final delay of a Max-plus automaton
β^{P}	initial delay of a Max-plus automaton
μ^{P}	Max-plus automaton morphism
π_r	right projection function
Γ	dataflow graph repetition vector
$\mathcal{B}$	grammar generating channel conditions in an SDF-PDFG
$\mathcal{D}$	grammar defining firing delays in an SDF-PDFG
$\mathcal{R}$	grammar defining rates in an SDF-PDFG
$\mathcal{M}_G^{\mathrm{par}}(x^G)$	parameterized SDF-PDFG matrix valid for configuration x^G
$\mathcal{M}_G^{\mathrm{par}\prime}(x^G)$	conservative approximation of $\mathcal{M}_G^{\mathrm{par}}(x^G)$
$\tilde{X}_{\mathsf{F}^{\mathrm{P}}}$	linearized domain of an SDF-PFSM-SADFG
$\overline{s}$	sequence of FSM-SADF scenarios
Σ	set of all token sequences
Σ^N	set of all tuples of N token sequences
$\mathbb{R}_{\max}^{n\times n}$	set of $n \times n$ Max-plus matrices
$\mathbb{R}_{\max}^{n}$	set of n dimensional Max-plus vectors
$src(c)$	source actor of channel c
σ	token sequence
$\tau(q)(n)$	production time of the nth token produced by an actor at its port q
$\boldsymbol{\sigma}$	tuple of N token sequences
$\|\|a\|\|$	norm of Max-plus vector a
$r_A(p)$	rate of SDFG actor A's port p
$\mathcal{P}_{Ad}$	firing delay parameters of SDF-PDF actor A

List of publications

Journals

- M. Skelin; M. Geilen; F. Catthoor; S. Hendseth, "Parameterized Dataflow Scenarios," in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol.PP, no.99, pp.1-1
doi: 10.1109/TCAD.2016.2597223

- Skelin, M.; Geilen, M.; Catthoor, F.; Hendseth, S., "Worst-case performance analysis for SDF-based parameterized dataflow," under review for publication in Elsevier's Microprocessors and Microsystems journal (MICPRO)

Conferences

- Skelin, M.; Wognsen, E.R.; Olesen, M.C.; Hansen, R.R.; Larsen, K.G., "Model checking of finite-state machine-based scenario-aware dataflow using timed automata," in Industrial Embedded Systems (SIES), 2015 10th IEEE International Symposium on, pp.1-10, 8-10 June 2015
doi: 10.1109/SIES.2015.7185065

- Skelin, M.; Geilen, M.; Catthoor, F.; Hendseth, S., "Worst-Case Throughput Analysis of SDF-Based Parametrized Dataflow," in Digital System Design (DSD), 2015 Euromicro Conference on, pp.17-24, 26-28 Aug. 2015
doi: 10.1109/DSD.2015.25

- Skelin, Mladen; Geilen, Marc; Catthoor, Francky; Hendseth, Sverre, "Worst-case latency analysis of SDF-based parametrized dataflow MoCs," in Design and Architectures for Signal and Image Processing (DASIP), 2015 Conference on, pp.1-6, 23-25 Sept. 2015
doi: 10.1109/DASIP.2015.7367259

- Skelin, M.; Geilen, M.; Catthoor, F.; Hendseth, S., "Parametrized dataflow scenarios," in Embedded Software (EMSOFT), 2015 International Conference on, pp.95-104, 4-9 Oct. 2015
  doi: 10.1109/EMSOFT.2015.7318264

Workshops

- Skelin, M.; Geilen M., Catthoor, F.; Hendseth, S., "Worst-case throughput analysis for parametric rate and parametric actor execution time scenario- aware data ow graphs," In Proceedings 1st International Workshop on Synthesis of Continuous Parameters, SynCoP 2014, Grenoble, France, 6th April 2014., pp. 65-79, 2014
  doi: 10.4204/EPTCS.145.7

- Skelin, M.; Wognsen, E.R.; Olesen, M.C.; Hansen, R.R.; Larsen, K.G., "Towards translating FSM-SADF to timed automata," in Proceedings of the first international workshop on Investigating data ow in embedded computing architectures, IDEA 2015, Amsterdam, The Netherlands, January 19 - 21, 2015, pp. 13-16, 2015.
  available: http://repository.tue.nl/795650