Worst-case Performance Analysis of SDF-based Parameterized Dataflow

Mladen Skelin, Marc Geilen, Francky Catthoor and Sverre Hendseth

***Abstract*—Dynamic dataflow models of computation (MoCs) have been introduced to provide designers with sufficient expressive power to capture increasing levels of dynamism in present-day streaming applications. Among dynamic dataflow MoCs, parameterized dataflow MoCs hold an important place. This is due to the fact that they allow for a compact representation of fine-grained data-dependent dynamics inherent to many present-day streaming applications.**

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

In this work, we (in a parametric fashion) analyze worst-case performance metrics (throughput and latency) of an important class of parameterized dataflow MoCs based on synchronous dataflow (SDF). We refer to such models as SDF-based parameterized dataflow (SDF-PDF). We show that parametric analysis in many cases allows to derive tighter conservative worst-case throughput and latency guarantees than the existing (nonparametric) techniques that rely on the creation of "worst-case SDF abstractions" of original parameterized specifications. Furthermore, we discuss how by using parametric analysis we can help address the scalability issues of enumerative analysis techniques.

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. 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.

***Index Terms*—parameterized dataflow, synchronous dataflow, SDF-based parameterized dataflow, Max-plus algebra, worst-case performance.**

I. Introduction

DATAFLOW models of computation (MoCs) have proven their value in modeling of streaming applications. 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 [1]. These token production and consumption numbers are called actor port *rates*. Actors fire according to a set of firing rules which specify what and how many tokens must be available at input ports for the firing to be enabled. In presence of feedback loops, actors in the loop would never become enabled because they depend on each other for tokens. This would lead the graph to a deadlock. Therefore, *initial tokens* must be placed on feedback channels. In timed dataflow under consideration in this paper, actor firing takes a finite amount of time called the *actor firing delay*. Furthermore, port rates are viewed as part of the actor *type signature* along with the type of the tokens [2][3]. Port rates can be used to define a graph *iteration*, or a set of actor firings that has no net-effect on the token distribution of the graph.

Dataflow MoCs have been traditionally divided into two classes: static dataflow MoCs [4] and dynamic dataflow MoCs [5].

Static dataflow MoCs are in wide use due to their predictability, strong formal properties and amenability to powerful optimization techniques [5].

Most well-known representatives of static dataflow are synchronous dataflow (SDF) [6] and cyclo-static dataflow (CSDF) [7]. In SDF, actor type signatures are fixed and known at compile-time. 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 tool choice for capturing the dynamic behavior inherent to present-day streaming applications. The need for expressive power beyond that offered by static dataflow MoCs brings us to the class of dataflow MoCs we call dynamic dataflow MoCs. Dynamic dataflow MoCs can be viewed as dataflow formalisms in which actor type signatures (port rates) and actor firing delays for timed models vary in ways not entirely predictable at compile-time [5].

In relation to the concept used to represent the dataflow dynamics, the work of [5] defines two subclasses of dynamic dataflow MoCs. First subclass refers to dataflow MoCs that are developed around an interacting combination of finite-state machines (FSM) and dataflow graphs. Models such as FSM-based scenario-aware dataflow (FSM-SADF) [8] and heterochronous dataflow (HDF) [2] are well-known examples

Manuscript received December 26, 2015; revised October 26, 2016; accepted December 7, 2016. This work was partly supported by ITEA 3 project 14014 ASSUME.

M. Skelin was with the Norwegian University of Science and Technology, 7491 Trondheim, Norway. He is now with the Eindhoven University of Technology, 5600 MB Eindhoven, The Netherlands (e-mail: m.skelin@tue.nl).

M. Geilen is with the Eindhoven University of Technology, 5600 MB Eindhoven, The Netherlands.

F. Catthoor is with IMEC vzw, 3001 Leuven, Belgium.

S. Hendseth is with the the Norwegian University of Science and Technology, 7491 Trondheim, Norway.

of such FSM/dataflow hybrids where an FSM is used to decouple control from concurrency.

In HDF, each FSM state is mode-refined by a submodel, where each refinement has different actor port rates. In FSM-SADF, each FSM state is associated with an SDF model of a scenario the state corresponds to. This has the effect that across FSM-SADF iterations actors operate in different *modes* or *scenarios*. In different scenarios, actors have different firing delays and port rates.

In the second subclass, a member of which we focus in this paper, dataflow dynamics are represented by alternative means. This is advantageous for the users of design tools that are accustomed to working in the dataflow domain and for which the FSM integration may represent an experimental concept [9].

Examples of such models are Boolean dataflow (BDF) [10], dynamic dataflow (DDF) [10] and parameterized dataflow [5].

In this paper, we are interested in 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* [5]. This way, using parameters, one is able to express fine-grained data-dependent dynamics of present-day streaming applications in a compact way. In particular, 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 are parameterized SDF (PSDF) [9], schedulable parametric dataflow (SPDF) [11], Boolean parametric dataflow (BPDF) [12] and variable rate dataflow (VRDF) [13].

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). Although such models have been parametrically analyzed for functional behavior and correctness, the parametric analysis of their temporal behavior, in particular analysis of their performance metrics such as throughput and latency, has received far less attention. However, there are remedies to this. In certain cases it is possible to create a "worst-case SDF abstraction" of the original parameterized specification that can be subjected to standard SDF performance analysis techniques [14][15]. The information needed to construct such "worst-case SDF abstraction" would include the upper endpoints of parameterized actor firing delays assuming that these are initially box constrained. The validity of such an abstraction follows from the monotonicity property of SDF [16] that SDF-PDF inherits.

However, using upper endpoints of firing delay parameters 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 [17] 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. In particular, a increase in rate value can lead to a decrease in the duration of graph iteration. Things get even worse if these rates show functional dependence on characteristics of the input signal.

A solution to this problem is enumeration, where one would consider all possible parameter value combinations. However, the run-time of enumerative analysis will in many cases in practice be prohibitive due to large spans of values the parameters can attain (compactness issues).

The aforementioned justifies the need for novel worst-case parametric performance analysis techniques that by operating directly on graph parameters remove the need for the touchy construction process of "worst-case SDF abstractions" of original parameterized specifications. Furthermore, we require that the parametric analysis can account for complex parameter inter-dependencies, and so avoid the pessimism the analysis based on "worst-case SDF abstraction" suffers from because it disregards these inter-dependencies and considers only the upper parameter interval endpoints. Finally, by working directly with parameters we remove the need for successive analysis of all parameter value combinations and so we help address the scalability problems the enumerative analysis is prone to.

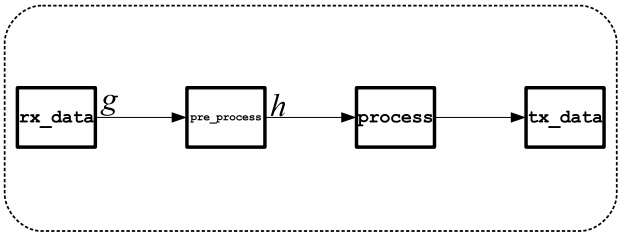
In this work we present such a worst-case performance analysis framework for SDF-PDF specifications in consideration of certain technical constraints we impose on the input graph structures. Within the framework, we consider self-timed execution of SDF-PDF structures. Self-timed execution is a schedule 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 the system captured by an SDF/SDF-PDF model [16]. We base our approach on the Max-plus algebraic [18] semantics of self-timed execution of SDF that SDF-PDF inherits. We model parameter reconfigurations using the theory of Max-plus automata [19] 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.

The remainder of this paper is structured as follows. In Section II we illustrate the importance of parameterized dataflow MoCs for modeling applications exposing fine-grained data-dependent dynamics and we outline the performance analysis challenges for such specifications. Section III discusses the related work. Section IV presents preliminary concepts. Section V formally defines SDF-PDF followed by Section VI that develops its Max-plus semantics. Section VII formally defines the performance metrics of interest and Section VIII presents techniques for computation of those metrics. Section IX demonstrates the application of our techniques on a realistic-case study from the multimedia domain. Finally, Section X concludes.

```
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. 1. Motivational example.

II. Motivational example

Parameterized dataflow MoCs are important because they by means of dynamic parameters allow for a compact representation of streaming applications exposing fine-grained data-dependent dynamics. Furthermore, they define precise semantics for parameter reconfiguration across application activations.

A motivational example of an application that illustrates the importance of parameterized dataflow as a modeling and analysis concept is shown in Fig. 1. The C specification of the example application is shown in Fig. 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. 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. 1a abstracts $2,252,126$ application behaviors (to see how this number was obtained we refer the interested reader to [20]). Therefore, we can say that the application exposes fine-grained data-dependent dynamics. The data-dependent behavior of the application can be compactly expressed (as a single entity) using the parameterized dataflow structure of Fig. 1b where loop bounds are captured by parameterized graph rates (actor firing delays are implied). This way we avoid the need for enumeration of $P_{m,n}$ that would result in $2,252,126$ nonparameterized dataflow structures accounting for all (g, h) combinations.

Furthermore, unlike the C specification that favors a sequential implementation, the (parameterized) dataflow specification reveals the application parallelism and encourages a parallel implementation. 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.

III. Related work

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

We start with PSDF [9]. PSDF introduces parameters in the definition of an SDF actor that control its functionality and/or its 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 [11] is a MoC closely related to PSDF. SPDF explicitly defines requirements that a parameterized dataflow specification must satisfy so that questions about 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 [13] facilities for frequent changes of actor port rates by means of parameterization. In particular, actor rates

may vary arbitrarily and are not necessarily constant over a complete iteration.

As originally proposed, all these models, except VRDF are untimed and consequently not accompanied by any kind of temporal analysis. Paper [13] discusses the temporal aspects of VRDF. However, it is of limited scope as the analysis methods are restricted to (conservative) buffer sizing under throughput constraints.

Some results exist, though, on parametric throughput analysis of SDF. In particular, the authors of [21] 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 authors of [22] take the story of [21] further by proposing algorithms to derive buffer sizes needed by acyclic SDF graphs with parametric rates to attain their maximal throughput. Here, throughput of an SDFG graph is also given as a function of graph parameters. However, the main difference between the work of [21] and [22] and ours is in the semantics of parameterization. In particular, the purport of parameterization in [21] and [22] 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 aforementioned works. In our work, parameterization of SDF implies dynamic parameters, i.e. parameters whose values change at run-time. From worst-case performance analysis point of view this means that the worst-case behavior may be defined over a cyclic pattern of parameter changes and not by a single parameter setting.

The work of [23] applies the technique of [21] to FSM-SADF which is a dynamic dataflow MoC by introducing parameterized actor firing delays to FSM-SADF. However, rates are left constant within scenarios and firing delay parameters are again fixed across scenarios, i.e. they are static. Furthermore, FSM-SADF implies a reasonably sized set of application scenarios/modes/behaviors and cannot compactly capture the fine-grained data-dependent application dynamics.

The most related work to ours is that of [24]. It proposes a fully parametric worst-case throughput analysis scheme for SPDF based on Max-plus linear system theory. However, the Max-plus characterization of SPDF graphs incurs too much pessimism because in that work each actor needs to wait for all its dependencies to complete all their firings within an iteration before it commences firing. This is unnecessarily restricting and the presentation of [24] can be made significantly tighter (cf. Section VI) yielding a tighter throughput estimate.

IV. Preliminaries

This section recaps Max-plus algebra and elaborates the concepts of SDF and FSM-SADF to the very detail. Although aware that the level of detail involved increases the risk of boring the reader, we feel a detailed treatment of preliminary concepts is needed to show the semantic link between SDF-PDF and FSM-SADF later in Section V. This link enables us to reformulate the performance analysis results of FSM-SADF and apply them to SDF-PDF in Section VIII which is the main result of this paper.

A. Max-plus algebra

We briefly introduce basic Max-plus algebra notation. Define $\mathbb{R}_{\max} = \mathbb{R} \cup \{-\infty\}$, where $\mathbb{R}$ is the set of real numbers. Let $a \oplus b = \max(a, b)$ and $a \otimes b = a + b$ for $a, b \in \mathbb{R}_{\max}$. For $a \in \mathbb{R}_{\max}$, $-\infty \oplus a = a \oplus -\infty = a$ and $a \otimes -\infty = -\infty \otimes a = -\infty$, i.e. $-\infty$ is the zero element of the $\oplus$ operation. By Max-plus algebra we understand the analogue of linear algebra developed for the pair of operations $(\oplus, \otimes)$ (pronounced "oplus" and "otimes", respectively) extended to matrices and vectors and denoted by $\mathcal{R}_{\max} = \{\mathbb{R}_{\max}, \oplus, \otimes\}$. 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 (sup-) sum of matrices $A, B \in \mathbb{R}^{n\times n}_{\max}$, denoted by $A \oplus B$ is defined by $[A \oplus B]_{i,j} = [A]_{i,j} \oplus [B]_{i,j}$ where $[A]_{i,j}$ and $[B]_{i,j}$ are entries of matrices A and B with indices i and j. The matrix product $A \otimes B$ is defined by $[A \otimes B]_{i,j} = \bigoplus_{k=1}^{n} [A]_{i,k} \otimes [B]_{k,j}$. For a vector $\mathbf{a} = [a_1, \ldots, a_n]^T \in \mathbb{R}^n_{\max}$, $||\mathbf{a}||$ denotes the vector norm, defined as $||\mathbf{a}|| = \bigoplus_{i=1}^{n} a_i$. With $A \in \mathbb{R}^{n\times n}_{\max}$ and $c \in \mathbb{R}$, we use denotations $A \otimes c$ or $c \otimes A$ for 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 $A \in \mathbb{R}^{n\times n}_{\max}$, $A^{\otimes k} = \bigotimes_{k} A$ where $k \in \mathbb{N}_{>0}$. For scalars $c \in \mathbb{R}$ and $\alpha \in \mathbb{R}$, $c^{\otimes\alpha} = \alpha \cdot c$ where $\cdot$ stands for multiplication in "regular algebra". Furthermore, it is easy to verify that Max-plus matrix multiplication is linear, i.e. $M \otimes (\mathbf{a} \oplus \mathbf{b}) = M \otimes \mathbf{a} \oplus M \otimes \mathbf{b}$ and $M \otimes (c \otimes \mathbf{a}) = c \otimes M \otimes \mathbf{a}$ for all $M \in \mathbb{R}^{n\times n}_{\max}$, $\mathbf{a}, \mathbf{b} \in \mathbb{R}^n_{\max}$ and $c \in \mathbb{R}_{\max}$. Now, let $M, N \in \mathbb{R}^{n\times n}_{\max}$. We write $M \preceq N$ if $[M]_{i,j} \leq [N]_{i,j}$ for all $i \in \{1, \ldots, n\}$ and $j \in \{1, \ldots, n\}$. In addition, the matrix multiplication is monotone, which means that if $\mathbf{a} \preceq \mathbf{b}$, then $M \otimes \mathbf{a} \preceq M \otimes \mathbf{b}$. Similarly, if $M \preceq N$, then $M \otimes \mathbf{a} \preceq N \otimes \mathbf{a}$.

B. Dataflow basics

In all dataflow MoCs, actors communicate by sending tokens along unidirectional channels with one producer and one consumer. On a channel, these tokens form *token sequences* that we define similarly as the concept of signals is defined in [25].

Definition 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 brackets 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 parenthesis, 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 defined as 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. A firing of an actor is enabled once its *firing rule* is satisfied. 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 (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. In this work we focus on compositions of SDF-PDF actors derived by parameterization of SDF actors to be defined next.

C. Synchronous dataflow

SDF is the most widely used, stable and mature dataflow MoC. In timed SDF both rates and actor firing delays are fixed and known at compile-time. SDF is an uninterpreted dataflow MoC, which means that the actual meaning of the computations and semantics of data tokens are not relevant [26]. 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, of course, 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 for A takes the form

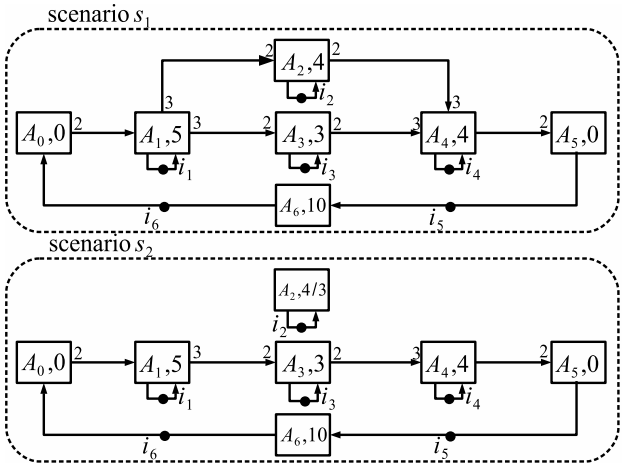
$$R = \{([\sigma_{p_1}(n)]_{n=1}^{r_A(p_1)}, \ldots, [\sigma_{p_U}(n)]_{n=1}^{r_A(p_U)})\} \tag{1}$$

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

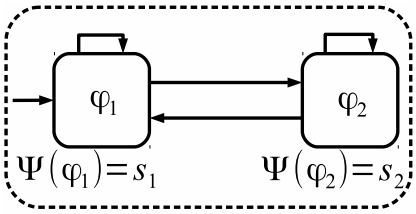
Firing rule of (1) says that in every firing A consumes $r_A(p_1)$ input tokens from port p_1 and so on until the last input port p_U from which it consumes $r_A(p_U)$ tokens regardless of their value (notation $*$). The firing rules do not depend on the availability times of input tokens, and therefore the notation $\perp$ is used.

Values $r_A(p_i)$ are fixed and known at compile-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. For $\tau_{q_i}(n) = t$, from now on, we will use the notation $\tau(q_i)(n) = t$.



(a) Scenario SDFGs.



(b) Scenario FSM.

Fig. 2. Example FSM-SADFG.

For a particular output port $q_i \in Q$ of actor A with firing delay d the following equation holds under self-timed execution [27]

$$\begin{aligned} \tau(q_i)(n) &= d + \max_{p_i \in P} \tau(p_i) \left(\left\lfloor \frac{n}{r_A(q_i)} \right\rfloor \cdot r_A(p_i) \right) \\ &= d \otimes \bigoplus_{p_i \in P} \tau(p_i) \left(\left\lfloor \frac{n}{r_A(q_i)} \right\rfloor \cdot r_A(p_i) \right). \end{aligned} \tag{2}$$

Equation (2) defines the Max-plus semantics of SDF and transitively of SDF-PDF that we elaborate later on. In relation to Max-plus algebra, the 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 3.

Definition 3 (SDFG)**.** *An SDFG $G = (\mathcal{A}, C, d, r, i)$ is a tuple where $\mathcal{A}$ is the set of actors, $C \subseteq \mathcal{A} \times \mathcal{A}$ is the set 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.*

Scenario s_1 graph of Fig. 2a is 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. The schedule for the running example can be denoted using the term $A_0^1A_1^2A_2^3A_3^3A_4^2A_5^1A_6^1$ where exponents represent actor repetition counts. We consider SDFGs that are consistent and deadlock-free. The graph that is inconsistent may deadlock or be unbounded. The existence of a repetition vector implies consistency. The repetition vector of an SDFG says how many times a particular graph actor needs to be fired in an valid schedule/iteration. It is computed using the set of so-called balance equations [6] . 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 [6].

D. Max-plus algebra for SDF

Max-plus algebra [18] is used to capture the semantics of self-timed execution od SDF.

Equation (2) 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 is fully determined by the availability times of initial tokens. In the SDF domain, initial tokens represent initial conditions for execution [28]. If the production timestamps of initial tokens after the kth graph iteration are collected in the vector $\boldsymbol{\gamma}(k) \in \mathbb{R}_{\max}^{|I|}$ the evolution of an SDFG G is given by the following recursive Max-plus linear equation

$$\boldsymbol{\gamma}(k+1) = M_G \otimes \boldsymbol{\gamma}(k). \tag{3}$$

In (3), $M_G \in \mathbb{R}_{\max}^{|I|\times|I|}$ is the SDFG Max-plus matrix, I is the set of initial tokens of the SDFG and $\boldsymbol{\gamma}(k)$ 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 $\boldsymbol{\gamma}(k+1)$ and $\boldsymbol{\gamma}(k)$.

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.

Matrix M_G of (3) can be derived by symbolically executing one iteration of the corresponding SDFG with the intention of relating the entries of $\boldsymbol{\gamma}(k+1) = [t'_{i_1}, \ldots, t'_{i_{|I|}}]$ and $\boldsymbol{\gamma}(k) = [t_{i_1}, \ldots, t_{i_{|I|}}]$ 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 [16], that the production timestamp t of any graph token can be represented as a Max-plus scalar product

$$t = \bigoplus_{i_j \in I} m_j \otimes t_{i_j} = [m_1, \ldots, m_{|I|}] \otimes \boldsymbol{\gamma}(k). \tag{4}$$

between a vector of suitable constants called the *initial token dependency vector* or shortly the dependency vector and the timestamp vector of the kth iteration. Then, also the entries of $\boldsymbol{\gamma}(k+1)$ can be written as linear combinations of entries of $\boldsymbol{\gamma}(k)$ as follows

$$t'_{i_l} = \bigoplus_{i_j \in I} m_{l,j} \otimes t_{i_j} = [m_{l,1}, \ldots m_{l,|I|}] \otimes \boldsymbol{\gamma}(k). \tag{5}$$

It straightforwardly follows from (3) and (5) 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 [16].

The Max-plus matrix of the scenario s_1 SDFG of Fig. 2a is given in (6).

$$M_G = \begin{bmatrix} 10 & -\infty & -\infty & -\infty & -\infty & 10 \\ 18 & 12 & -\infty & -\infty & -\infty & 18 \\ 16 & -\infty & 9 & -\infty & -\infty & 16 \\ 22 & 16 & 14 & 8 & -\infty & 22 \\ 22 & 16 & 14 & 8 & -\infty & 22 \\ -\infty & -\infty & -\infty & -\infty & 10 & -\infty \end{bmatrix} \tag{6}$$

E. FSM-based scenario-aware dataflow

The concept of synchronous dataflow scenarios [16] extends the expressive power of SDF by combining streaming data and finite control into a MoC called FSM-SADF [8]. 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 $S_A = \{s_{A1}, \ldots, s_{AZ}\}$ be the set of scenarios of A, where $Z = |S_A|$. Let function $r_A : (P \cup Q) \times S_A \to \mathbb{N}_{>0}$ give the rate value for a given actor port and a given scenario. Let function $d_A : S_A \to \mathbb{R}_{\geq 0}$ return the firing delay of an actor for a given scenario. Then, the general firing rule of an FSM-SADF actor is as follows

$$R = \{([\sigma_{p_1}(n)]_{n=1}^{r_A(p_1,s_{Ai})}, \ldots, [\sigma_{p_U}(n)]_{n=1}^{r_A(p_U,s_{Ai})})_{i=1}^{Z}\}, \tag{7}$$

where $\sigma_{p_i}(n) = (*, \perp)$ for all $i = 1, \ldots, U$. Unlike an SDF actor that has only one firing rule (cf. (1)), an FSM-SADF actor has as many firing rules as there are scenarios (notice that R of (7) 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

$$\tau(q_i)(n) = d_A(\overline{s}_Q(q_i, L)) \otimes \bigoplus_{p_i \in P} \tau(p_i)(\sum_{j=1}^{L} r_A(p_i, \overline{s}_Q(q_i, j))). \quad (8)$$

In (8), $\overline{s}_Q : Q \times [1, \ldots, L] \to S_A$ 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. (1), (2)), FSM-SADF actors (cf. (7), (8)) across different firings in different scenarios consume and produce different numbers of tokens while their firings take different amounts of time across scenarios. The ways in which rates and actor firing delays change is not entirely predictable at compile-time. In particular, in FSM-SADF scenario variations are nondeterministic. Thus is FSM-SADF a dynamic dataflow MoC. Note that different scenario sequences will result in different evaluations of (8) 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. But to simplify the notation and be in correspondence with the existing dataflow literature [29] we abuse the notion of a function in this context.

Naturally, FSM-SADF actors are composed into graphs. An example of an FSM-SADFG is shown in Fig. 2. The graph has two scenarios: s_1 and s_2 modeled by two SDFGs. The difference between the scenarios[1] is that scenario s_2 SDFG compared to scenario s_1 SDFG misses channels (A_1, A_2) and (A_2, A_4), while the firing delays of actor A_2 are different in different scenarios. The scenario FSM has two states where each of the states corresponds to one scenario. In the figure, state ϕ_1 corresponds to s_1, while ϕ_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 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, (7) and (8) reduce to (1) and (2), 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 $|S_A| = 1$ is an SDF actor. Therefore, FSM-SADF generalizes SDF. We give the formal definition of FSM-SADF as adopted from [30]. First we define the scenario FSM as follows.

Definition 4 (Scenario FSM)**.** *Given a set S of scenarios, a scenario FSM F on S is a tuple $F = (\Phi, \phi_0, \delta, \Psi)$, where Φ is the set of states, ϕ_0 is the initial state, $\delta \subseteq \Phi \times \Phi$ is the transition relation and $\Psi : \Phi \to S$ is the scenario labeling.*

Thereafter, we define FSM-SADF in Definition 5.

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

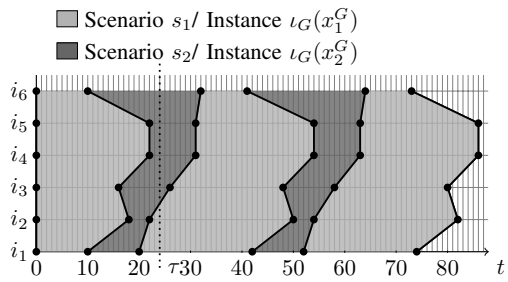


Fig. 3. Execution of FSM-SADFG of Fig. 2 and the execution of SDF-PDFG of Fig. 4 with $X_G = \{x_1^G, x_2^G\}$.

Definition 5 (FSM-SADF)**.** *FSM-SADF F is a tuple $\mathsf{F} = (S, F)$ where S is the set of scenarios and F is an FSM on S.*

F. 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 $\boldsymbol{\gamma}(0), \boldsymbol{\gamma}(1), \ldots$ where

$$\boldsymbol{\gamma}(k+1) = \mathcal{M}_{\mathsf{F}}(\zeta_{\mathsf{F}}(k+1)) \otimes \boldsymbol{\gamma}(k). \quad (9)$$

In (9), $\mathcal{M}_{\mathsf{F}} : S \to \mathbb{R}_{\max}^{|I|\times|I|}$, returns the Max-plus matrix of the scenario SDFG, $\zeta_{\mathsf{F}} : \mathbb{N}_{>0} \to S$ returns the scenario of the $(k+1)$st FSM-SADFG iteration. Fig. 3 shows the execution of the running example FSM-SADFG for scenario sequence s_1, s_2, s_1, s_2, s_1. Time is depicted horizontally and the six tokens of the timestamp vector are depicted vertically so that lines visualize the timestamp vectors of (9) as they evolve across scenarios depicted in different colors. Let

$$\overline{s} = s_1, \ldots, s_k \in L \quad (10)$$

denote a sequence of scenarios where $L \subseteq S^*$ defines a restriction of S^* determined by the scenario FSM. It had been shown in [8] that the completion time of (10) can be defined as follows

$$\mathcal{A} = \boldsymbol{\alpha}^T \otimes \mu(\overline{s}) \otimes \boldsymbol{\beta}, \quad (11)$$

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

$$\mu(\overline{s}) = \mathcal{M}_{\mathsf{F}}(s_k) \otimes \ldots \otimes \mathcal{M}_{\mathsf{F}}(s_1) \quad (12)$$

and $\boldsymbol{\beta}$ is the initial delay. The initial delay $\boldsymbol{\beta}$ specifies the initial enabling time of initial tokens and typically $\boldsymbol{\beta} = \mathbf{0}$, while the final delay $\boldsymbol{\alpha}$ 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 $\boldsymbol{\alpha} = \mathbf{0}$. The triple $\mathcal{A} = (\boldsymbol{\alpha}, \mu, \boldsymbol{\beta})$ defines a Max-plus automaton [19]. The theory of Max-plus automata had been used in [8] to analyze worst-case performance of FSM-SADFG. We leave this matter aside now and explain it in detail in Section VIII when we reformulate the FSM-SADF results and apply them to SDF-PDF.

V. 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 VIII under certain restriction concerning the input graph structure. We first focus on SDF-PDF actors in Section V-A, that we compose into SDF-PDF graphs (SDF-PDFGs) in Section V-B. Thereafter in Section V-C we define the subset of SDF-PDF we shall use for performance analysis of systems. Finally, in Section V-D we discuss the semantic link between our analysis model and FSM-SADF that is a crucial argument in the definition of the Max-plus algebraic semantics of SDF-PDF in Section VI.

A. SDF-PDF actors

We start by defining our parameterization scope. With SDF in mind 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.

We proceed by defining 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 expression production 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. 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 parametric expressions 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.

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 $\mathcal{P}_{Ad}$. 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 expression of $\mathcal{R}_A$ valid for that port at a specific $x^A \in X_A$. Similarly, let $d_A : X_A \to \mathbb{R}_{\geq 0}$ given a configuration return the firing delay of an actor. It does so by evaluating the expression of $\mathcal{D}_A$ that gives the parameterized firing delay of A for $x^A \in X_A$.

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

$$R = \{([\sigma_{p_1}(n)]_{n=1}^{r_A(p_1, x_i^A)}, \ldots, [\sigma_{p_U}(n)]_{n=1}^{r_A(p_U, x_i^A)})_{i=1}^{Z}\}, \quad (13)$$

where $\sigma_{p_i}(n) = (*, \perp)$ for all $i = 1, \ldots, U$. From (13) 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

$$\begin{aligned} \tau(q_i)(n) =& d_A(\overline{x}_Q(q_i, L)) \\ & \otimes \bigoplus_{p_i \in P} \tau(p_i)(\sum_{j=1}^{L} r_A(p_i, \overline{x}_Q(q_i, j))). \end{aligned} \quad (14)$$

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 (14) 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 (14) can take many different values at the same 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 (14) 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 a "normal" SDF actor, i.e. for an X_A with a single configuration, it is trivial to check that (13) and (14) reduce to (1) and (2). This shows that SDF-PDF generalizes SDF.

B. SDF-PDF graphs

SDF-PDF actors are composed into SDF-PDF graphs (SDF-PDFGs). From the considerations of the previous section on SDF-PDF actors and the definition of SDFG (cf. Definition 3), with a minimal change in notation, it is rather straightforward to induce a definition for the composition of SDF-PDF actors, i.e. SDF-PDFG.

Let graph rates be expressed as parametric expressions. Let the production of these expressions 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 6 (SDF-PDFG)**.** *An SDF-PDFG is 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 set 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 6 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.

The SDF-PDF model of Definition 6 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 [9] allows to specify arbitrary inter-dependencies between parameters involved in the construction of rate and firing delay expressions. However, do note at this point that we will not be able to analyze statically SDF-PDFGs in their full generality. This is explained in detail in sections to follow.

C. 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 model, 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 the 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 [31].

Furthermore, we restrict our attention to 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 6 it is not decidable (at compile-time) whether they are consistent/bounded, deadlock-free and quasi-static schedulable.

To assure consistency/boundedness, deadlock-freedom and quasi-static schedulability we need to restrict parameterization patterns. In particular we need to restrict $\mathcal{R}$ so that compile-time guarantees on 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 [11] 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 [32]. Therefore, we take the definition of $\mathcal{R}$ from [11] for our analysis model as follows

$$\mathcal{R} := k \mid p \mid \mathcal{R}_1 \cdot \mathcal{R}_2. \tag{15}$$

In (15), $k \in \mathbb{N}_{>0}$ 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 6 allows it. This is because a rate of 0 in the context of a channel balance equation of an SDF-PDFG has an ambiguous meaning. 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 entries 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

$$\mathcal{B} := \texttt{tt} \mid \texttt{ff} \mid b \mid \neg\mathcal{B} \mid \mathcal{B}_1 \wedge \mathcal{B}_2 \mid \mathcal{B}_1 \vee \mathcal{B}_2, \tag{16}$$

where $b \in \mathcal{P}_b$ be a grammar defined over $\mathcal{P}_b$. Let $\lambda : C \to \mathcal{B}$ return for a channel its condition. As mentioned, if $\lambda(c)$ evaluates to `ff`, channel c 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 [10] as they serve as syntactical sugar to account for the possibility of a rate being equal to 0.

To ensure that all graph rates and channel conditions 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. Note that this differs slightly from [11] where integer parameters can change inside iterations, and from [12] where Boolean parameters can change inside iterations (but not integer parameters). Across iterations, configurations change arbitrary through reconfiguration, which can be modeled as an nondeterministic choice. Therefore, SDF-PDF supports nondeterminism unlike (to the best of our knowledge) any other parameterized dataflow MoC based on SDF [33].

Fig. 4 shows an example SDF-PDFG.

For the example graph, $\mathcal{P}_i = \{p, q\}$, $\mathcal{P}_d = \{a_1, a_2, a_3, a_4\}$ and $\mathcal{P}_b = \{b\}$. Channels (A_1, A_2) and (A_2, A_4) are made conditional using Boolean parameter b. The graph's quasi-static schedule takes the form $A_0^1 A_1^q A_2^p A_3^p A_4^q A_5^1 A_6^1$ and its the

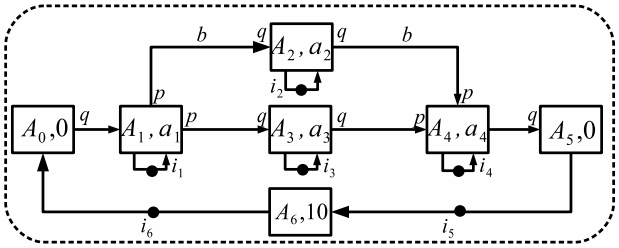


Fig. 4. Example SDF-PDFG.

repetition vector equals to $\Gamma(A_0, A_1, A_2, A_3, A_4, A_5, A_6) = (1, q, p, p, q, 1, 1)$.

D. 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 with the intention of relating the concepts of reconfiguration in SDF-PDF and scenario transition in FSM-SADF from the temporal angle.

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 (7) and (8) to that of SDF-PDF actors of (13) and (14) 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 an 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 $\boldsymbol{\gamma}(0), \boldsymbol{\gamma}(1), \boldsymbol{\gamma}(2), \ldots$ 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. 4. 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 $x_2^G = \{b = \texttt{ff}, p = 3, q = 2, a_1 = 5, a_2 = 4, a_3 = 4, a_4 = 4\}$. 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. 2a. As the FSM-SADFG of Fig. 2a has a fully connected FSM, with the correspondence $x_1^G \equiv s_1$ and $x_2^G \equiv s_2$, any configuration/instance sequence of the SDF-PDFG has an "equivalent" FSM-SADF scenario sequence. The completion times of these sequences expressed by means of $\boldsymbol{\gamma}(k)$ are captured by contours of Fig. 3 for the instance/scenario sequence $x_1^G \equiv s_1, x_2^G \equiv s_2, x_1^G \equiv s_1, x_2^G \equiv s_2, x_1^G \equiv s_1$.

VI. 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.

A. Max-plus algebraic semantics of SDF-PDF

With regard to the Max-plus algebraic semantics of FSM-SADF of (9) and the semantic link of SDF-PDF and FSM-SADF explained in the previous section, the evolution of an SDF-PDFG G can be given as a recursive Max-plus linear equation relating the timestamps vectors $\boldsymbol{\gamma}(k+1)$ and $\boldsymbol{\gamma}(k)$ of initial tokens after the $(k+1)$st and the kth SDF-PDFG iteration, respectively, as follows

$$\boldsymbol{\gamma}(k+1) = \mathcal{M}_G(\zeta_G(k+1)) \otimes \boldsymbol{\gamma}(k). \tag{17}$$

In (17), $\mathcal{M}_G : X_G \to \mathbb{R}_{\max}^{|I|\times|I|}$ 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 infeasible 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

$$\boldsymbol{\gamma}(k+1) = \underbrace{\left(\mathcal{M}_G^{\mathrm{par}}(\zeta_G(k+1))\right)\left(\zeta_G(k+1)\right)}_{\mathcal{M}_G(\zeta_G(k+1))} \otimes \boldsymbol{\gamma}(k). \quad (18)$$

In (18), $\mathcal{M}_G^{\mathrm{par}}(x^G)$ denotes a mapping that for each $x^G \in X_G$ returns the associated parameterized Max-plus matrix 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 $\zeta_G(k+1) = x^G$. By using parameterized matrices, one does not need to perform an enumeration of X_G. The difficulty is moved, however, to determining the mapping $\mathcal{M}_G^{\mathrm{par}}$ 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.

Because 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

$$t = \bigoplus_{i_j \in I} m_j^{\mathrm{par}} \otimes t_{i_j} = [m_1^{\mathrm{par}}, \ldots, m_{|I|}^{\mathrm{par}}] \otimes \boldsymbol{\gamma}(k), \quad (19)$$

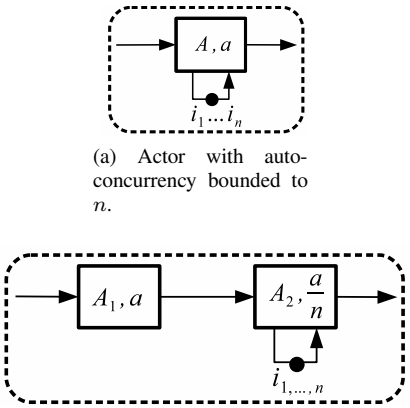
where t_{i_j} are the timestamps of initial tokens after the kth graph iteration and m_j^{par} are now parametric expressions. Therefore, the timestamps of initial tokens at the end of the $(k+1)$st iteration embedded in $\boldsymbol{\gamma}(k+1)$ can be computed as follows

$$t'_{i_l} = \bigoplus_{i_j \in I} m_{l,j}^{\mathrm{par}} \otimes t_{i_j} = [m_{l,1}^{\mathrm{par}}, \ldots, m_{l,|I|}^{\mathrm{par}}] \otimes \boldsymbol{\gamma}(k). \quad (20)$$

In this case, dependency vectors $[m_{l,1}^{\mathrm{par}}, \ldots, m_{l,|I|}^{\mathrm{par}}]$ 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 mapping $\mathcal{M}_G^{\mathrm{par}}$. Thus, the challenge lies in determining expressions of type (20). 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 1. *For all SDF-PDFG channels $c \in C$ such that $src(c) \neq dst(c)$ and $i(c) > 0$, $i(c) > r(dst(c), c) \cdot \Gamma(dst(c))$ 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 [34][11][9]. In the context of our Max-plus analysis we impose this requirement as it is not clear how to deal with schedule loops of length greater than one [32] with parametric repetition counts. Note that this requirement limits our ability to model certain types of resource constraints that incur circular dependencies in the graph. A good example are channel buffer sizes that we encode by means of back-pressure. In that case, the assumption we must work with is that of buffer-space overallocation. This means that we assume that enough buffer space for the application is allocated so that the performance is not affected.



(a) Actor with auto-concurrency bounded to n.

(b) Latency-rate abstraction of the actor above.

Fig. 5. Latency-rate abstraction

Requirement 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 2 we allow either full auto-concurrency for an actor or no auto-concurrency at all. This is because, during the process of determining $\mathcal{M}_G^{\mathrm{par}}$ with regard to (19) 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 [35] that conservatively captures its temporal behavior. Fig. 5b shows an abstraction of an actor with auto-concurrency bounded to n displayed in Fig. 5a. Note that the collection $i_1, \ldots, i_n$ of Fig. 5a is collapsed into a single token $i_{1,\ldots,n}$ of Fig. 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 1 using the notion of local iterations [11]. 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.

B. Max-plus model of SDF-PDF execution

1) The basics: To determine $\mathcal{M}_G^{\mathrm{par}}$, as with SDF [16], 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 (20). 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. (4)). 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 production times and ordering of tokens using the mapping $\tau(A_j, n)$ that returns the timestamp vector ($\in \mathbb{R}_{\max}^{|I|}$) of the token produced by the nth firing of actor $A_j \in \mathcal{A}$. Note that $n \in \mathbb{Z}$, where 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)) that SDF-PDF inherits (cf. (14)):

$$\tau(A_j, n) = \bigoplus_{A_i|(A_i,A_j)\in C} \varepsilon((A_i, A_j)) \tag{21a}$$

$$\otimes\, \tau\left(A_i, \left\lceil \frac{n \cdot r(A_j,(A_i,A_j)) - i((A_i,A_j))}{r(A_i,(A_i,A_j))} \right\rceil\right) \tag{21b}$$

$$\otimes\, d(A_j). \tag{21c}$$

Equation (21) encodes the Max-plus semantics of self-timed execution of an SDF-PDF actor within an iteration of the superordinate SDF-PDFG. Synchronization is expressed via (21b) 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 producing 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 (21c). However, in the presence of conditional 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 (21a), where $\varepsilon : C \to \{0, -\infty\}$ is defined as follows

$$\varepsilon(c) = \begin{cases} 0 & \text{if } \lambda(c) = \texttt{tt} \\ -\infty & \text{if } \lambda(c) = \texttt{ff} \end{cases} \tag{22}$$

for all $c \in C$.

2) Computation of the actor response: We show now how to compute $\tau(\cdot, n)$ for an actor within an SDF-PDFG iteration. Equation (21) reveals that in computing the response of an actor, different inputs (channels) can be treated in isolation. Ultimately, particular contributions need to be superposed.

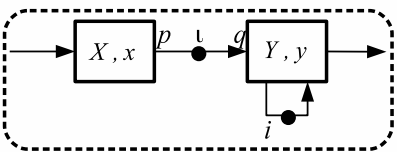


Fig. 6. SDF-PDFG channel.

This corresponds to the Max-plus superposition principle [18]. We first show how to apply (21) to one input channel.

Consider a general SDF-PDF channel structure of Fig. 6. 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 ι. By default, it is enabled, i.e. the implied channel condition always evaluates to $\texttt{tt}$. We compute the output of actor Y using (21). We only treat the case when in the figure, $\iota = 0$. The case where $\iota > 0$ is trivial due to Requirement 1. More precisely, within one iteration of the graph, actor's demand for input tokens on that channel will always refer to one of those ι initial tokens, i.e. no firings of X within the iteration need to be considered. For actor Y with $\iota = 0$, (21) transforms into

$$\tau(Y, n) = \left(\tau\left(Y, n-1\right) \oplus \tau\left(X, \left\lceil \frac{n \cdot q}{p} \right\rceil\right)\right) \otimes y. \tag{23}$$

We treat (23) 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 (23) for k times and substitute it back, we obtain

$$\tau(Y, n) = \tau(Y, n-k) \otimes y^{\otimes k} \oplus \bigoplus_{i=1}^{k} \tau\left(X, \left\lceil \frac{(n-i+1)\cdot q}{p} \right\rceil\right) \otimes y^{\otimes i}. \tag{24}$$

We obtain the base case when $k = n$ from (24) as follows

$$\tau(Y, n) = \tau(Y, 0) \otimes y^{\otimes n} \oplus \underbrace{\bigoplus_{i=1}^{n} \tau\left(X, \left\lceil \frac{(n-i+1)\cdot q}{p} \right\rceil\right) \otimes y^{\otimes i}}_{conv(\tau(X, \lceil \frac{n\cdot q}{p} \rceil), h(Y,n))}. \tag{25}$$

In the second term of the Max-plus summation of (25) 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

$$u(n) = \begin{cases} 0 & \text{if } n = 1 \\ -\infty & \text{otherwise} \end{cases}, \text{for all } n \in \mathbb{N}_{>0}. \tag{26}$$

For a complete presentation we refer to [36]. When the actor has a self-edge with one initial token, its impulse response takes the form

$$h(Y, n) = y^{\otimes n}\,, \text{for all } n \in \mathbb{N}_{>0}. \tag{27}$$

An actor without a self-edge, on the other hand, can be interpreted as an actor with a self-edge with an infinite stock of initial tokens all available at $t = -\infty$.

Therefore, using (21) for the structure of Fig. 6 with $\iota = 0$ and while assuming that there is an infinite stock of initial tokens hosted by self-edge (Y, Y) the following equation holds:

$$\tau(Y,n) = \left(\tau(Y,n-m) \oplus \tau\left(X, \left\lceil \frac{n \cdot q}{p} \right\rceil\right)\right) \otimes y, \quad (28)$$

where $m \to \infty$ (to account for an infinite stock of initial tokens). In this case, with $m \to \infty$, for all $n \in \mathbb{N}_{>0}$, $\tau(Y, n-m) = -\infty$. Therefore, (28) reduces to

$$\tau(Y,n) = \tau\left(X, \left\lceil \frac{n \cdot q}{p} \right\rceil\right) \otimes y. \quad (29)$$

We now take a breath to recap. In particular, equations (25) and (29) reveal how to compute the response of an actor contributed by one input dependency (channel). When an actor does not have a self-edge (stateless actor) it only delays the input tokens as follows straightforwardly from (29). In case an actor has a self-edge (state), a convolution of its impulse response and the input token sequence needs to be considered as given by (25). For completeness, we formally define the concept of Max-plus convolution.

Definition 7. *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*

$$conv(\varsigma_1, \varsigma_2)(n) = \bigoplus_{i=1}^{n} \varsigma_1(n-i+1) \otimes \varsigma_2(i). \quad (30)$$

The convolutional form of (25) due to its recursiveness stands in our way of determining the response of an actor as an explicit function of n. Therefore, we must express the convolutional part as an explicit function of n.

This is a difficult task in a parametric setting. This is due to the fact that any system defined in (3) is eventually periodic [37]. In SDF and transitively SDF-PDF this means that the responses of actors will be eventually periodic sequences [37] as long as the inputs are eventually periodic sequences themselves.

We say that a sequence $\varsigma : \mathbb{N}_{>0} \to \mathbb{R}_{\max}$ is eventually periodic with period c and ratio π if there exists an integer t such that

$$\forall n \geq t : \varsigma(n+c) = \varsigma(n) \otimes \pi^{\otimes c}, \quad (31)$$

where the concept of eventual periodicity naturally extends to vectors.

Therefore, the convolution of (25) will itself be eventually periodic. To have the convolutional term in (25) expressed as explicit function of n we now simplify the analysis into an entirely linear pattern.

In particular, in the following proposition, we show how to conservatively bound the convolution of (25) using a linear sequence (and eventually periodic sequence where $c = 1$) that attains the ratio of the original eventually periodic sequence.

Proposition 1. *Let $\varsigma_1(n)$ and $\varsigma_2(n)$ be two sequences in $\mathbb{R}_{\max}$ such that $\varsigma_1(n) = \delta_1 \otimes {\pi_1}^{\otimes \lceil r \cdot n \rceil}$ 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:*

$$conv(\varsigma_1, \varsigma_2)(n) < \begin{cases} \delta_1 \otimes {\pi_1}^{\otimes(1+r)} \otimes {\pi_2}^{\otimes n} & \text{if } \pi_2 \geq r \cdot \pi_1 \\ \delta_1 \otimes \pi_2 \otimes {\pi_1}^{\otimes(1+r\cdot n)} & \text{if } \pi_2 \leq r \cdot \pi_1 \end{cases} \quad (32)$$

The estimate is exact in the ratio part. Therefore, for $n \to \infty$, the relative estimation error will move towards 0.

Because impulse responses of actors with self-edges are linear themselves, the result of Proposition 1 allows us to conservatively bound the response of an arbitrary actor A_k of an SDF-PDFG compliant to Requirements 1 and 2 within an iteration using a delay-ratio (δ, π) abstraction as follows

$$\begin{aligned} \hat{\tau}(A_k, n) &= \bigoplus_{i_j} (\delta_{k,j} \otimes {\pi_{k,j}}^{\otimes n}) \otimes t_{i_j} \\ &= [(\delta_{k,1} \otimes {\pi_{k,1}}^{\otimes n}), \ldots, (\delta_{k,|I|} \otimes {\pi_{k,|I|}}^{\otimes n})] \otimes \boldsymbol{\gamma}(k). \end{aligned} \quad (33)$$

A delay-ratio (δ, π) abstraction defines the dependency vector entries as linear functions of n, i.e. the actor firing index. 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 ratios) firing delay of the slowest actor in the path defined by the producing actors of the two tokens.

In case of an actor with multiple input channels, ultimately, the contributions of different input channels need to be superposed. Across corresponding dependency vector entries of different contribution, the following proposition defines a conservative bound for the corresponding output dependency vector entry.

Proposition 2. *Let $\Omega = \{\varsigma_1(1), \ldots, \varsigma_N(n)\}$ be a set of 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 $\Sigma(n) = \bigoplus_{i=1}^{N} \varsigma_i(n)$. Then, $\Sigma(n)$ attains the following conservative bound for all $n \in \mathbb{N}_{>0}$*

$$\hat{\Sigma}(n) = \delta \otimes \pi^{\otimes n}. \quad (34)$$

The result of (34) defines a linear sequence that attains the ratio of the slowest input sequence and the delay of the most delayed sequence. We use a conservative estimate to avoid the complexity of the computation of n at which the sequence with the maximum ratio starts to dominate. Such considerations would significantly degrade the performance of the parameterized matrix generation algorithm we present in the sections to come. Instead, by using (34) we have a conservative bound for all n. In this case too, when $n \to \infty$, the relative estimation error moves towards 0 because the estimate is exact in the ratio part.

By using (25) and (29) in conjunction with Proposition 1 and Proposition 2 we can compute the response of any graph actor within an iteration expressed in the form of (33) that we use to render $\mathcal{M}_G^{\text{par}}$. We show how to do this in on an example in the section to come.

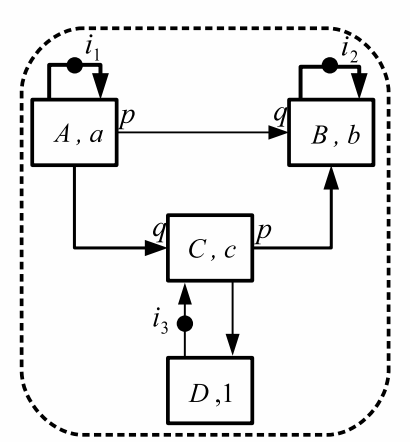


Fig. 7. Example SDF-PDFG.

C. Example

We exemplify the parameterized matrix generation process using the SDF-PDF specification of Fig. 7. This is an artificial but an illustrative example that covers the relevant cases in actor response computation elaborated prior to this section. Note that we discuss the application of our techniques to a realistic case study later on in Section IX.

Here, for the running example, we will compute the responses of graph actors within a single iteration using its quasi-static schedule that is given as A^qCDB^p. The timestamp vector of the kth SDF-PDFG iteration is specified as

$$\boldsymbol{\gamma}(k) = [t_{i_1}, t_{i_2}, t_{i_3}]^T. \tag{35}$$

Furthermore, every entry of $\boldsymbol{\gamma}(k)$ can be written in terms of (21) (by associating it with the corresponding producing actor) and onward as the Max-plus scalar product of (19).

$$\begin{aligned} t_{i_1} &= \tau(A,0) = [0, -\infty. - \infty] \otimes \boldsymbol{\gamma}(k), \\ t_{i_2} &= \tau(B,0) = [-\infty, 0, -\infty] \otimes \boldsymbol{\gamma}(k), \\ t_{i_3} &= \tau(D,0) = [-\infty, -\infty, 0] \otimes \boldsymbol{\gamma}(k). \end{aligned} \tag{36}$$

According to the quasi-static schedule of the specification, actor A will start firing first. In has only one dependency, namely its self-edge. To compute its response we use (25) where we disregard the right-hand term in the sum because the actor has no input dependencies other than the self-edge. By doing so, we obtain:

$$\begin{aligned} \tau(A,n) &= \tau(A,0) \otimes a^{\otimes n} \\ &= [a^{\otimes n}, -\infty, -\infty] \otimes \boldsymbol{\gamma}(k). \end{aligned} \tag{37}$$

The successful completion of q firings of actor A is followed by a singe firing of actor C (within an iteration). Notice that actor C has an input channel hosting one initial token. Due to Requirement 1, this channel must host a sufficient number of initial tokens to fire actor C as many times it is scheduled to fire within an iteration. We use (21) to calculate the time at which actor C completes its first (and single) firing within the iteration by accounting for all its input dependencies as follows:

$$\tau(C,1) = (\tau(A,q) \oplus \tau(D,0)) \otimes c. \tag{38}$$

Using (37) and (36), the equation above expands to:

$$\begin{aligned} \tau(C,1) &= \left([a^{\otimes q}, -\infty, -\infty] \oplus [-\infty, -\infty, 0]\right) \otimes \boldsymbol{\gamma}(k) \otimes c \\ &= [c \otimes a^{\otimes q}, -\infty, c] \otimes \boldsymbol{\gamma}(k). \end{aligned} \tag{39}$$

The firing of actor C enables one firing of actor D that completes exactly at:

$$\begin{aligned} \tau(D,1) &= \tau(C,1) \otimes 1 = \\ &= [1 \otimes c \otimes a^{\otimes q}, -\infty, 1 \otimes c] \otimes \boldsymbol{\gamma}(k). \end{aligned} \tag{40}$$

Finally, actor B can perform its firings within the iteration. It has input dependencies, realized in channels $c_1 = (A, B)$, $c_2 = (C, B)$ and in self-edge $c_3 = (C, C)$. To compute its response we apply the Max-plus superposition principle that allows us to consider one external input dependency at a time. The self-edge dependency is considered internal to the actor as it defines the actor state. Therefore, it is always included in the computation of the actor response for a particular external input dependency. In particular, we compute the response of actor B as follows

$$\tau(B,n) = \tau_{c_1}(B,n) \oplus \tau_{c_2}(B,n), \tag{41}$$

where $\tau_{c_1}(B,n)$ and $\tau_{c_2}(B,n)$ stand for responses of B contributed by dependencies realized in channels c_1 and c_2, respectively.

First we calculate $\tau_{c_1}(B,n)$ as follows:

$$\tau_{c_1}(B,n) = \left(\tau(B,n-1) \oplus \tau\left(A, \left\lceil \frac{n \cdot q}{p} \right\rceil\right)\right) \otimes b. \tag{42}$$

The equation above has the same structure as that of (23) and via (25) can be formulated as follows

$$\tau_{c_1}(B,n) = [conv(a^{\otimes \lceil \frac{q}{p} \cdot n \rceil}, b^{\otimes n}), b^{\otimes n}, -\infty] \otimes \boldsymbol{\gamma}(k). \tag{43}$$

Every entry of the dependency vector of the equation above involving a convolution needs to be treated by Proposition 1 that gives rise to two cases that split the original parameter space (the graph domain) into two exclusive parts. In (43) there is only one entry involving a convolution. According to Proposition 1, the calculation continues in the two parameter subspaces defined respectively by the inequalities $p \cdot b \geq q \cdot a$ and $p \cdot b \leq q \cdot a$. In our exercise we proceed by considering the split of the parameter space defined by

$$C_1 \equiv p \cdot b \geq q \cdot a. \tag{44}$$

With (32) and (44), (43) transforms into

$$\hat{\tau}_{c_1}(B,n) = [a^{\otimes(1+\frac{q}{p})} \otimes b^{\otimes n}, b^{\otimes n}, -\infty] \otimes \boldsymbol{\gamma}(k), \tag{45}$$

where $\hat{\tau}_{c_1}(B,n)$ denotes a conservative estimate of $\tau_{c_1}(B,n)$.

We can now proceed with the calculation of the response of B as contributed by channel $c_2 = (C, B)$. According to (21), the following equation holds:

$$\tau_{c_2}(B,n) = \left(\tau(B,n-1) \oplus \tau\left(C, \left\lceil \frac{n}{p} \right\rceil\right)\right) \otimes b. \tag{46}$$

Within an iteration B fires p times, i.e. $n \in [1, \ldots, p]$. As we are only interested in responses of actors within a single iteration, (46) transforms into:

$$\tau_{c_2}(B,n) = (\tau(B,n-1) \oplus \tau(C,1)) \otimes b. \tag{47}$$

If we now substitute (39) into (47), and apply backward substitution to get rid of the recursive term, we obtain:

$$\tau_{c_2}(B,n) = [c \otimes a^{\otimes q} \otimes b^{\otimes n}, b^{\otimes n}, c \otimes b^{\otimes n}] \otimes \boldsymbol{\gamma}(k). \quad (48)$$

We now need to take a look back at (41). If we substitute (45) and (48) into (41), we obtain:

$$\hat{\tau}(B,n) = [a^{\otimes(1+\frac{q}{p})} \otimes b^{\otimes n} \oplus c \otimes a^{\otimes q} \otimes b^{\otimes n}, b^{\otimes n}, c \otimes b^{\otimes n}] \otimes \boldsymbol{\gamma}(k). \quad (49)$$

To get rid of the $\oplus$ in the first entry of the dependency vector of (49), we apply Proposition 2. More precisely, we define the following delay-ratio abstractions as follows: $\delta_1 = a^{\otimes(1+\frac{q}{p})}$, $\pi_1 = b$, $\delta_2 = c \otimes a^{\otimes q}$ and $\pi_2 = b$. As the ratio pairs are equal, we only need to consider the delay pairs. By doing so, Proposition 2 enforces a split of the parameter space based on two inequalities: $\delta_1 \geq \delta_2$ and $\delta_1 \leq \delta_2$. In the exercise, we proceed with the first option defined by

$$C_2 \equiv (1 + \frac{q}{p}) \cdot a \geq c + q \cdot a. \quad (50)$$

Thanks to the inequality (50),the $\oplus$ in the first entry of (49) disappears, so (49) transforms into

$$\hat{\tau}(B,n) = [a^{\otimes(1+\frac{q}{p})} \otimes b^{\otimes n}, b^{\otimes n}, c \otimes b^{\otimes n}] \otimes \boldsymbol{\gamma}(k). \quad (51)$$

Note that, in general, Proposition 2 involves a procedure where corresponding entries of dependency vectors of responses per contributing channels are all represented as delay-ratio pairs. At that point, Proposition 2, per each dependency vector entry, defines maximally four splits in the parameter space. The four splits in the current dependency vector entry are carried over to the consideration of the next dependency vector entry.

The calculation of (51) completes the iteration for our exercise. What remains to be done is to determine the entries of $\boldsymbol{\gamma}(k+1) = [t'_{i_1}, t'_{i_2}, t'_{i_3}]^T$ from the calculated 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

$$t'_{i_1} = \tau(A,q) \quad t'_{i_2} = \hat{\tau}(B,p) \quad t'_{i_3} = \tau(D,1) \quad (52)$$

we obtain the parameterized matrix of (53)

$$\hat{\mathcal{M}}_G^{\text{par}}(x^G) = \begin{bmatrix} {a_1}^{\otimes q} & -\infty & -\infty \\ a^{\otimes(1+\frac{q}{p})} \otimes b^{\otimes p} & b^{\otimes p} & c \otimes b^{\otimes p} \\ 1 \otimes c \otimes a^{\otimes q} & -\infty & 1 \otimes c \end{bmatrix} \quad (53)$$

where $x^G \in X_G \cap (p \cdot b \geq q \cdot a) \cap ((1+\frac{q}{p}) \cdot a \geq c + q \cdot a)$, i.e. x^G belongs to the part of the original graph domain refined by the set of constraints of (44) and (50). 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 $\mathcal{M}_G^{\text{par}}$. Fig. 8 illustrates the partitioning of the parameter space (domain) X_G for the running example. Note that the example graph of Fig. 7 did not include any Boolean parameters to simplify presentation. They are easily taken into account in the presented analysis in a way that evaluations of

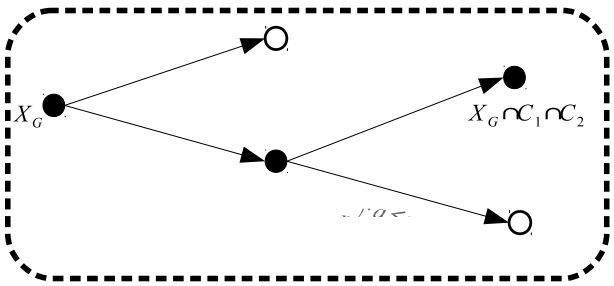


Fig. 8. Exploration tree.

associated channel conditions to either `tt` or `ff` represents nothing but splits in the parameter space.

The matrix of (53) is defined by the path determined by the black nodes of the exploration tree. Note that the matrix of (53) is a conservative estimate of $\mathcal{M}_G^{\text{par}}(x^G)$, thus denoted $\hat{\mathcal{M}}_G^{\text{par}}(x^G)$.

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 an Max-plus SDFG matrix. In particular, $[\mathcal{M}_G^{\text{par}}(x^G)]_{m,n}$ 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, $[\mathcal{M}_G^{\text{par}}(x^G)]_{m,n}$ defines the latency of the slowest path in the graph connecting two initial tokens. 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 $[\hat{\mathcal{M}}_G^{\text{par}}(x^G)]_{2,1}$ as obtained from (51), we see that actor B is the bottleneck of the path from i_1 to i_2, i.e. it has the maximum 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.

D. Algorithm for deriving $\mathcal{M}_G^{\text{par}}$

Algorithm 1 specifies the previously described procedure for deriving the mapping $\mathcal{M}_G^{\text{par}}$.

It is defined by a recursive function `ExploreGraph` that explores the tree-like structures like that of Fig. 8 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 maximum delay among all the delays observed for the currently processed dependency vector entry, `curr_init_tok_dep_ratio_ndx` the index

ALGORITHM 1: Calculate $\mathcal{M}_G^{\text{par}}$

```
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
```

of the currently set maximum ratio among all the ratios observed for the currently processed dependency vector entry, `in_contr_comput_completed` the 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 has 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). Recall here how the matrix of (53) was composed using the actor responses of (52).

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

If the satisfiability check does not fail and the channels 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). In the example of Section VI-C we considered a convolution while computing the response of actors B.

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 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 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 maximum (cf. Line 37). Once all ratios 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). In the context of Section VI-C, the procedure described was applied to actor B (cf. (41)). 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 in terms of number of actors, dependencies among them (channels), parameters and the number of initial tokens. It is also computationally intense in the sense that checking feasibility and satisfiability in Line 2 involves the use of constraint satisfaction and SAT solvers, respectively. However, practical applications 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. Even if there are dependencies, many feasibility/satisfiability problems in Line 2 will typically be re-encountered and need not to be solved again. Furthermore, the definitions of the domains may be such that many exploration paths will be pruned out due to infeasibility. Therefore, it is reasonable to assume that the worst-case in terms of complexity will rarely be realized. All this could make the computational effort of Algorithm 1 reasonable and render it applicable to graphs counting up to a few dozen actors and initial tokens while decorated by a few critical parameters.

The set of matrices obtained via Algorithm 1 when evaluated for 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 1 and 2. Mathematically speaking,

$$\left(\hat{\mathcal{M}}_G^{\mathrm{par}}(x^G)\right)(x^G) \succeq \left(\mathcal{M}_G^{\mathrm{par}}(x^G)\right)(x^G) \tag{54}$$

for all $x^G \in X_G$, where $\left(\mathcal{M}_G^{\mathrm{par}}(x^G)\right)(x^G) = M_{\iota_G(x^G)}$. The approximation of (54) 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 1 and 2). In relation to that, it is interesting to notice that the relative estimation error (per matrix entry) moves towards 0 with growing entry values of the graph repetition vector. This is due to the delay-ratio abstraction we use. In particular, the larger the number of firings of an actor is within an iteration, the "ratio" part of the abstraction will quantitatively contribute more to the actor firing completion times. In practice, in many cases, repetition vector entries will typically be strictly increasing functions of parameters. Therefore, a growth in values of parameters will manifest itself as a growth in values of repetition vector entries.

We use the following example to illustrate this. In particular, we consider an arbitrary entry of the left and right-hand side matrices of (54) for the graph of Fig. 7 with the repetition vector $\Gamma(A,B,C,D) = [q,p,1,1,]$. Let $x^G = \{p = 1, q = 20, a = 5, b = 100, c = 1\}$. In this case, $\left[\left(\mathcal{M}_G^{\mathrm{par}}(x^G)\right)(x^G)\right]_{2,1} = 205$, while $[M_{\iota_G(x^G)}]_{2,1} = 201$. For growing values of repetition vector entries, i.e. for growing values of p and q, the ratio component becomes more dominant and the relative error shrinks and for $\{p,q\} \to \infty$ it reaches 0. E.g, with $x^G = \{p = 301, q = 199, a = 5, b = 100, c = 1\}$, $\left[\left(\mathcal{M}_G^{\mathrm{par}}(x^G)\right)(x^G)\right]_{2,1} = 31096$, while $[M_{\iota_G(x^G)}]_{4,1} = 31096$. Actually, in the later case we have the exact result because for this configuration, (50) no longer holds which takes us to a subdomain where the parameterized matrix does not introduce an estimation error at all. However, this is only the case for this example, and not for the general case where we only may expect a relative error shrinkage and not an exact result.

VII. Problem definition

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

A. 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 throughput used for SDF [14] and FSM-SADF [8] 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 $\boldsymbol{\gamma}(k) = \boldsymbol{\gamma}(0), \boldsymbol{\gamma}(1), \boldsymbol{\gamma}(2), \ldots$ The completion time of the kth SDF-PDFG iteration is given by the norm of $\boldsymbol{\gamma}(k)$, i.e. $||\boldsymbol{\gamma}(k)||$. Therefore, inspired by [8] we can define the worst-case throughput of an SDF-PDFG as follows.

Definition 8 (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 $\boldsymbol{\gamma}(k)$, for every $\epsilon \in \mathbb{R}$ such that $\epsilon > 0$, there is some $K \in \mathbb{N}_{>0}$ s.t. for all $L \in \mathbb{N}_{>0}$, $L > K$, $\frac{L}{||\boldsymbol{\gamma}(L)||} > Th_G - \epsilon$.*

Simply put, Definition 8 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 [16][30] and therefore the need for a somewhat cumbersome formulation of Definition 8.

B. Worst-case latency

Similarly as proposed by [8][38] for FSM-SADF, we define the worst-case latency of an SDF-PDFG G relative to a period that equals to its worst-case throughput Th_G as follows.

Definition 9 (Worst-case latency)**.** *Worst-case latency of an SDF-PDFG G relative to its worst-case throughput Th_G is defined as the smallest vector $\mathbf{L}_G$ such that for every possible instance sequence and its associated Max-plus timestamp vector sequence $\boldsymbol{\gamma}(k)$, for every $k \geq 0$, $\boldsymbol{\gamma}(k) \leq \frac{k}{Th_G} + \mathbf{L}_G$.*

Simply put, $\mathbf{L}_G$ is a vector that shows when at the latest a particular SDF-PDFG iteration with index k can possibly finish with respect to the reference point of time $\frac{k}{Th_G}$.

VIII. Performance analysis for SDF-PDF

A. 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, 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).

The definitions for worst-case throughput and latency for an SDF-PDFG (cf. Definition 8 and 9) 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 V-D and the definition of the completion time (11) of a sequence of FSM-SADF scenarios of (10) we can define the completion time of a sequence of configurations

$$\overline{x}^G = x_1^G, \ldots, x_k^G \in X_G^* \tag{55}$$

of an SDF-PDFG as follows

$$\mathcal{A} = \boldsymbol{\alpha}^T \otimes \mu(\overline{x}^G) \otimes \boldsymbol{\beta}, \tag{56}$$

where $\boldsymbol{\alpha}$ 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

$$\mu(\overline{x}^G) = \mathcal{M}_G(x_k^G) \otimes \ldots \otimes \mathcal{M}_G(x_1^G) \tag{57}$$

and $\boldsymbol{\beta}$ is the initial delay. The structure $\mathcal{A} = (\boldsymbol{\alpha}, \mu, \boldsymbol{\beta})$ defines the Max-plus automaton tuple of an SDF-PDFG G. Note that in (55), $\overline{x}^G \in X_G^*$ which means that configurations are sequenced nondeterministically/arbitrarily.

B. Worst-case throughput

The Max-plus automaton structure of (56) with the morphism μ of (57) 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 has been used to study the performance of FSM-SADF [8][30][39]. In particular, we focus on the results obtained for an FSM-SADFG with a fully connected FSM. This is because, as discussed in Section V-D, in the worst-case, the temporal behavior of an SDF-PDFG in terms of vectors $\boldsymbol{\gamma}(k)$ 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 $M_G^{\mathrm{w-c}}$ of SDF-PDFG G as follows

$$M_G^{\mathrm{w-c}} = \bigoplus_{x_i^G \in X_G} \mathcal{M}_G(x_i^G). \tag{58}$$

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 FSM-SADFGs with fully connected FSMs (cf. Proposition 5.2 of [8] that directly follows from Theorem 2 of [19]), the worst-case throughput of SDF-PDFG G corresponds to the inverse of the maximum cycle mean (MCM) [40] of the communication graph of $M_G^{\mathrm{w-c}}$. Formally,

$$\frac{1}{Th_G} = \max_c \frac{\sum_{e\in c} w(e)}{\sum_{e\in c} 1}, \tag{59}$$

where $\max$ is taken over all circuits c of $\mathcal{G}(M_G^{\mathrm{w-c}})$ and the sums are taken over all edges e of c. Map $w : \mathcal{E} \to \mathbb{R}$ returns the edge weight.

The intuitive explanation is as follows. The weights of the edges correspond to the entries of $M_G^{\mathrm{w-c}}$ and represent the 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 (58). Enumeration of X_G imposes itself as a solution to the problem. However, as we already stated in Section VI-A enumeration of X_G may not be feasible in practice for large X_G. Even with relatively small domains, if the repetition vector entries 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 [8] show that the time needed to produce $M_{\iota_G(x^G)}$ scales "more than linearly" with growing repetition vector entries.

However, the enumeration problem can be avoided by using the set of parameterized SDF-PDFG matrices as obtained by Algorithm 1. In that case, the problem of (58) transforms into

$$M_G^{\mathrm{w-c}} \preceq \bigoplus_{M^{\mathrm{par}} \in cod(\hat{\mathcal{M}}_G^{\mathrm{par}})} \underbrace{\bigoplus_{x^G \in X_G \text{ s.t. } \hat{\mathcal{M}}^{\mathrm{par}}(x^G)=M^{\mathrm{par}}} (M^{\mathrm{par}})(x^G)}_{M^{\mathrm{opt}}} . \tag{60}$$

In (60), $cod(\hat{\mathcal{M}}_G^{\mathrm{par}})$ denotes the codomain of the mapping $\hat{\mathcal{M}}_G^{\mathrm{par}}$. Simply put, $cod(\hat{\mathcal{M}}_G^{\mathrm{par}})$ is the set of parameterized matrices obtained by executing Algorithm 1. Notation $\{x^G \in dom(\hat{\mathcal{M}}_G^{\mathrm{par}}) \text{ s.t. } \hat{\mathcal{M}}_G^{\mathrm{par}}(x^G) = M^{\mathrm{par}}\}$ captures the set of configurations the for which $\hat{\mathcal{M}}_G^{\mathrm{par}}(x^G)$ is valid. Note that the right hand side of (60) defines a conservative estimate of $M_G^{\mathrm{w-c}}$, denoted $\hat{M}_G^{\mathrm{w-c}}$, due to (54) and so (but not necessarily) the Th_G obtained using (60) may be a conservative estimate of the actual value, denoted $\check{Th}_G$. Nevertheless, for growing repetition vector entries of the graph, for reasons disclosed in Section VI-A, the relative estimation error moves towards 0.

In (60), matrix M^{opt} as a concrete matrix can be obtained by solving a series of optimization problems as follows:

$$\begin{aligned} &\textbf{foreach } (i,j) \text{ s.t. } [M^{\mathrm{par}}]_{ij} \neq -\infty \textbf{ do} \\ &\quad \underset{x^G}{\text{maximize}} \quad [M^{\mathrm{par}}(x^G)]_{i,j} \\ &\quad \text{subject to} \quad x^G \in X_G \text{ s.t. } \hat{\mathcal{M}}_G^{\mathrm{par}}(x^G) = M^{\mathrm{par}}. \end{aligned} \tag{61}$$

The type of optimization problems encountered in (61) 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 notions of boundedness, deadlock-freedom and schedulability. If only rates were subject to parameterization, in the context of (61) we would be facing rational function of polynomials. These problems can be converted to polynomial programming problems [41] and efficiently solved using the techniques of [42]. 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 (61). To stay in the scope of polynomial programming, we limit $\mathcal{D}$ to polynomial functions of parameters or more formally

$$\mathcal{D} := k \mid d \mid \mathcal{D}_1 \cdot \mathcal{D}_2 \mid \mathcal{D}_1 + \mathcal{D}_2 \tag{62}$$

where $d \in \mathcal{P}_d$ and $k \in \mathbb{R}_{\geq 0}$. 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 [43]. 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 (61) exists.

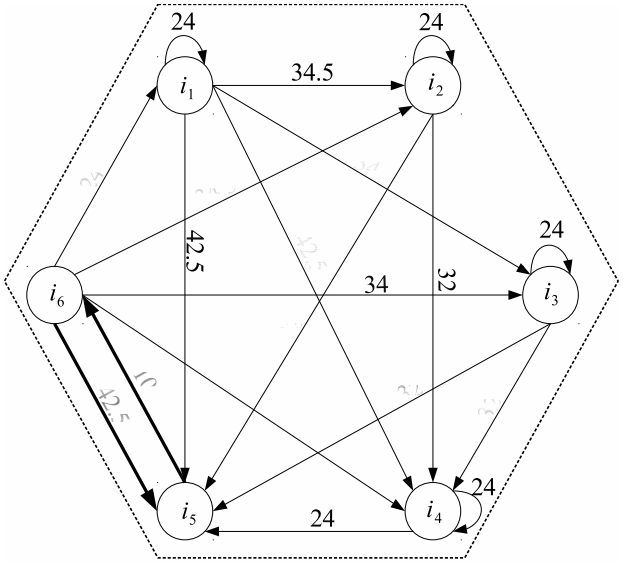


Fig. 9. Communication graph of Max-plus matrix of (64).

We exemplify using the SDF-PDFG of Fig. 4. Assume that its domain is given by

$$\begin{aligned} X_G = &(b = \mathtt{tt}) \cap (p \cdot a_1 \geq q \cdot a_1) \cap (p \cdot a_3 \leq q \cdot a_1) \\ &\cap (q \cdot a_4 \leq p \cdot a_2) \cap (a_4 \leq a_1) \cap (q \cdot a_4 \leq p \cdot a_3) \\ &\cap \big\{p = w_1 \cdot w_2, w_1 + w_2 = 2 \cdot x_1 - x_2, \\ &p \in [1,10], q \in [1,10], w_1 \in [1,3], w_2 \in [1,4], \\ &x_1 \in [1,3], x_2 \in [1,5], a_1 \in [1,7], a_2 = 4, \\ &a_3 \in [1,5], a_4 = 4\big\} \end{aligned} \tag{63}$$

Equation (63) is a very illustrative example of a domain specification because it shows how graph parameters (rates and actor firing delays) may exhibit arbitrary nonlinear interdependencies and dependencies 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 (63) in addition defines a default parameter interval for each parameter, e.g $p \in [1,10]$. After running (61), we obtain the worst-case evaluation matrix specified by (64).

$$\hat{M}_G^{\mathrm{w-c}} = \begin{bmatrix} 24 & -\infty & -\infty & -\infty & -\infty & 24 \\ 34.5 & 24 & -\infty & -\infty & -\infty & 34.5 \\ 34 & -\infty & 24 & -\infty & -\infty & 34 \\ 42.5 & 32 & 32 & 24 & -\infty & 42.5 \\ 42.5 & 32 & 32 & 24 & -\infty & 42.5 \\ -\infty & -\infty & -\infty & -\infty & 0 & -\infty \end{bmatrix} \tag{64}$$

The communication graph of $\hat{M}_G^{\mathrm{w-c}}$ is shown in Fig. 9. In the graphical representation of $\mathcal{G}(\hat{M}_G^{\mathrm{w-c}})$, rather than using numerical values $\{1, \ldots, n, \ldots, |I|\}$ 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 $\frac{1}{2} \cdot (42.5 + 10) = 26.25$, and therefore $\check{Th}_G = \frac{1}{26.25}$ iterations per time-unit, i.e. $\frac{1}{26.25}$ is a conservative estimate or an exact

value if the critical cycle is defined by a self-edge of the communication graph. As mentioned, if it is a conservative estimate, the relative error will shrink with growing values of graph's repetition vector entries.

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. This is possible in this example (but not in the general case) because the parametric repetition vector entries of the specification are strictly increasing functions of parameters. It is worthwhile mentioning that in many practical cases where repetition vector entries are fractional functions of parameters, they can be normalized [12] and so be rendered strictly increasing functions of parameters.

By using upper bound of parameters in the running example we obtain the throughput value of $\check{Th}_G = \frac{1}{70}$ time-units per iteration which is clearly a significant overapproximation 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 (63).

C. Worst-case latency

From Definition 9 it follows that determining the worst-case latency equals to finding the smallest $\mathbf{L}_G$ such that $\boldsymbol{\gamma}(k) \leq \mathbf{L}_G + \frac{k}{Th_G}$. Therefore,

$$\mathbf{L}_G = \bigoplus_k (\boldsymbol{\gamma}(k) - \frac{k}{Th_G}). \tag{65}$$

It follows from (65) that given Th_G we are interested in the maximum value $\boldsymbol{\gamma}(k)$ can attain for all possible instance sequences. As follows from the Max-plus algebraic semantics of SDF-PDFG defined by (56) and (57) and the monotonicity of SDF-PDFG, $\boldsymbol{\gamma}(k)$ can be conservatively bounded using the worst-case evaluation vector $\boldsymbol{\gamma}^{\mathrm{w-c}}(k)$ as follows

$$\boldsymbol{\gamma}(k) \preceq \boldsymbol{\gamma}^{\mathrm{w-c}}(k). \tag{66}$$

In (66),

$$\boldsymbol{\gamma}^{\mathrm{w-c}}(k) = M_G^{\mathrm{w-c}} \otimes \boldsymbol{\gamma}^{\mathrm{w-c}}(k-1) \tag{67}$$

for all $k > 0$ where $\boldsymbol{\gamma}^{\mathrm{w-c}}(0)$ stores the initial availability timestamps of the graph's initial tokens. Equation (67) can be re-written to an explicit form as follows

$$\boldsymbol{\gamma}^{\mathrm{w-c}}(k) = M_G^{\mathrm{w-c}^{\otimes k}} \otimes \boldsymbol{\gamma}^{\mathrm{w-c}}(0). \tag{68}$$

Therefore, the worst-case latency computation problem for SDF-PDF of (65) transforms into

$$\mathbf{L}_G \preceq \bigoplus_k (\boldsymbol{\gamma}^{\mathrm{w-c}}(k) - \frac{k}{Th_G}). \tag{69}$$

for the given enabling vector $\boldsymbol{\gamma}^{\mathrm{w-c}}(0)$. Typically, $\boldsymbol{\gamma}^{\mathrm{w-c}}(0) = \mathbf{0}$. The right hand side expression of (69) now represents a conservative bound on $\mathbf{L}_G$ that we want to determine, denoted $\hat{\mathbf{L}}_G$.

At first glance, this bound seems hard to compute because we need to consider all $\boldsymbol{\gamma}^{\mathrm{w-c}}(k)$ vectors up to an arbitrary large k as we consider finite SDF-PDFG executions of arbitrary length (as in streaming applications). However, the sequence $\boldsymbol{\gamma}^{\mathrm{w-c}}(k)$ as defined by (67) and (68) has a very nice property. In particular, it follows from the Max-plus spectral theory [44][18] that the sequence of vectors given by $\boldsymbol{\gamma}(k+1) = M \otimes \boldsymbol{\gamma}(k)$ 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

$$\boldsymbol{\gamma}(k+c) = \boldsymbol{\gamma}(k) \otimes c \otimes \boldsymbol{\eta}. \tag{70}$$

In (70), $\boldsymbol{\eta} \in \mathbb{R}_{\max}^m$ 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 maximum among cycle-time vector entries. For more details we refer to [45] and [44].

Therefore, the periodicity property of (70) allows us to solve the problem of (69) by only determining the first $t+c$ timestamp vectors of (67), i.e. for $k = 1, \ldots, t+c$. This is because for the values of k beyond $t+c$, the growth rate of $\boldsymbol{\gamma}^{\mathrm{w-c}}(k)$ cannot be faster than determined by the cycle-time vector and consequently the inverse of $\check{Th}_G$ that is the maximum among all cycle time vector entries and will not lead to a larger $\hat{\mathbf{L}}_G$.

We compute $\hat{\mathbf{L}}_G$ for the running example SDF-PDFG with $\hat{M}_G^{\mathrm{w-c}}$ of (64), $\boldsymbol{\gamma}^{\mathrm{w-c}}(0) = [0,0,0,0,0,0]^T$ and $\frac{1}{\check{Th}_G} = 26.25$ time-units per iteration. For $\hat{M}_G^{\mathrm{w-c}}$, $\boldsymbol{\eta} = [26.25]^T$, $c = 2$ and $t = 2$ and therefore

$$\begin{aligned}\hat{\mathbf{L}}_G \leq \bigoplus \big\{ &[0,0,0,0]^T, [24, 34.5, 34, 42.5, 42.5, 10]^T - 26.25, \\ &[48, 58.5, 58, 66.5, 66.5, 52.5]^T - 52.5, \\ &[76.5, 87, 86.5, 95, 95, 76.5]^T - 78.75 \big\} \\ &= [0, 8.25, 7.75, 16.25, 16.25, 0]^T.\end{aligned}$$

IX. Case study

In this section, we demonstrate the application of our analysis techniques to 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 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. We also discuss how using our technique one can in many situations address the scalability issues of enumerative analysis.

ROI coding [46] 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. 10b. 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. 10a adopted from [47] 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. 10a corresponds to the 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 types of blocks subject to processing. In particular, we differentiate between three types of blocks: 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 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

$$
\begin{aligned}
X_G = \big\{ & p = (2 \cdot \xi_M \cdot 2 \cdot \xi_m)/(16 \cdot 16), p \in [1, P], && \text{(71a)}\\
& q \in [1, 16] && \text{(71b)}\\
& p' \geq \mu \cdot P, p + p' \leq P && \text{(71c)}\\
& o^2 = 4 \cdot \pi^2 (\xi_M^2 + \xi_m^2), o \geq O && \text{(71d)}\\
& \epsilon^2 \cdot \xi_M^2 = \xi_M^2 - \xi_m^2, \epsilon = E, 2 \cdot \xi_M \leq w, && \text{(71e)}\\
& 2 \cdot \xi_m \leq h && \text{(71f)}\\
& a = a_{\text{ref}}, b = b_{\text{ref}}, c = c_{\text{ref}}, d = d_{\text{ref}}, && \text{(71g)}\\
& e = e_{\text{ref}}, f = f_{\text{ref}}, g = q_{\text{ref}}, h = h_{\text{ref}} \big\}. && \text{(71h)}
\end{aligned}
$$

The number of macroblocks p within the slice is given by the area of the ellipse's bounding rectangle (cf. (71a)). Note that the size of a macroblock is 16×16 pixels. Depending on resolution, the picture/frame consists of maximally P macroblocks (cf. (71a)). The number of blocks within a macroblock q is constrained by (71b). It is known that o is always greater than a certain predefined constant O (cf. (71d)), i.e. O defines the maximum 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. (71e) and (71f)). Within a picture, it is assumed that the background always occupies the portion μ of the picture/frame comprising p' macroblocks (cf. (71c)). Referent actor execution times (cf. (71g) and (71h)) were taken from [47] and are expressed in cycles of the STMicroelectronics STxP70 processor.

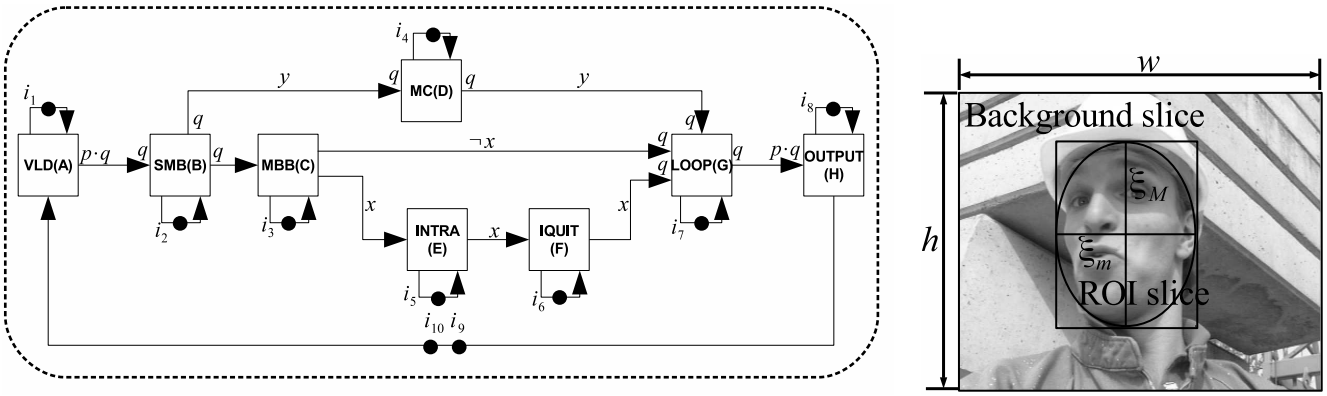
From the case study we see the modeling flexibility the SDF-PDF offers. In particular, it allows to express fine-grained data dependent behavior using parameters. The values parameters may attain at run-time can depend on the characteristics of the input data (the input signal). In the case study, this is 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 VIII 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 maximum default values for all parameters) we obtain the guaranteed throughput value of $1.05699 \cdot 10^{-7}$ slices per cycle. This is possible because the parametric repetition vector entries of the specification are strictly increasing functions of parameters [47]. Both values are conservative approximations of the actual value but our technique tightens the SDF result by 39.65 percent. This is because the latter analysis cannot take into account complex parameter inter-dependencies.

In cases where there exist no such inter-dependencies and the worst-case behavior of the graph can indeed be defined by using default upper bound values of parameters, our technique will most likely introduce unnecessary overhead both in time and the result and it is wiser to use the "worst-case SDF" type of analysis. Hence, it is up to the designer to choose the technique most suitable for the problem at hand.

The case study also indicates that our technique helps addressing the scalability issues of enumerative analysis for applications with vast domains. In particular, imagine the case where one cannot use the "worst-case SDF" type of analysis because it is not clear which values of parameters define the worst-case behavior of the graph. Then, one can resort to exact enumerative analysis. However, for types of applications resembling to our case study, the exact enumerative analysis is hardly practicable (it will take hours) because the analysis time of only a single configuration of the VC-1 decoder SDF-PDFG with a repetition vector whose entries attain large values can take up to 80s on an Intel Core i5-750 CPU running at 2.67GHz with 8GB main memory. On the other hand, our analysis was manually completed (with the help of an industrial optimization tool) within few hours. This shows the benefits of parametric analysis that we expect to fully utilize when our technique is fully automatized. In particular, by using symbolic (parametric) analysis we avoid the need for successive analysis of all configurations the run-time of which may be prohibitive, but the price we pay is in accuracy of the analysis. If we need the tightest performance estimates, we will of course have to resort to enumerative analysis.

All aforementioned speaks in favor of our technique as a valuable method that complements the exiting ones and can be very helpful in providing early performance estimates to the designer of dynamic systems exposing fine-grained dynamic behavior. Still, given the computational intensiveness of the technique both in terms of Algorithm 1 and (61), our approach



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

(b) ROI coding.

Fig. 10. Case study.

is primarily intended for deriving early performance estimates for graphs counting up to a few dozen actors and initial tokens while involving a few critical parameters.

X. Discussion and conclusion

In this paper we considered the worst-case performance analysis problem for dynamic streaming applications that can be captured using SDF-PDF where application/design-space parameters typically expose complex interdependencies and can attain values from large or even infinite domains. We believe that our technique is a valuable addition to the existing techniques for performance analysis of dynamic streaming applications of this type for several reasons.

First, so far, the problem was coarsely treated using the existing SDF techniques that typically incur too pessimistic over-approximations when parameters expose arbitrary inter-dependencies. Using the VC-1 decoder case study, we have shown that in cases where parameters expose inter-dependencies, our techniques is able to produce significantly tighter but still conservative performance estimates.

Second, compared to enumerative analysis techniques of application graph's with huge domains, the VC-1 decoder case study indicates that our technique will for applications of similar characteristics typically perform better at the cost of analysis accuracy. Still, given the computational intensiveness of our technique, to perform a comprehensive scalability analysis, it needs to be fully automated which is a subject of future work.

Furthermore, for our technique to be applicable, input specifications must satisfy certain requirements that restrict modeling of some types of resource constraints. Finding ways to alleviate these restrictions is a subject of future work too.

XI. Acknowledgment

This work was partly supported by ITEA 3 project 14014 ASSUME.

References

[1] R. Zurawski, *Embedded systems handbook*. CRC Press, 2005.

[2] A. Girault, B. Lee, and E. Lee, "Hierarchical finite state machines with multiple concurrency models," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 18, no. 6, pp. 742–760, Jun 1999.

[3] A. Jantsch, *Modeling embedded systems and SoCs: concurrency and time in models of computation*. Morgan Kaufmann, 2004.

[4] S. Ha and H. Oh, "Decidable dataflow models for signal processing: Synchronous dataflow and its extensions," in *Handbook of Signal Processing Systems*, S. S. Bhattacharyya, E. F. Deprettere, R. Leupers, and J. Takala, Eds. Springer New York, 2013, pp. 1083–1109.

[5] S. S. Bhattacharyya, E. F. Deprettere, and B. D. Theelen, "Dynamic dataflow graphs," in *Handbook of Signal Processing Systems*, S. S. Bhattacharyya, E. F. Deprettere, R. Leupers, and J. Takala, Eds. Springer New York, 2013, pp. 905–944.

[6] E. Lee and D. Messerschmitt, "Synchronous data flow," *Proceedings of the IEEE*, vol. 75, no. 9, pp. 1235–1245, Sept 1987.

[7] G. Bilsen, M. Engels, R. Lauwereins, and J. Peperstraete, "Cycle-static dataflow," *Signal Processing, IEEE Transactions on*, vol. 44, no. 2, pp. 397–408, Feb 1996.

[8] M. Geilen and S. 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*, ser. CODES/ISSS '10. New York, NY, USA: ACM, 2010, pp. 125–134.

[9] B. Bhattacharya and S. Bhattacharyya, "Parameterized dataflow modeling for DSP systems," *Signal Processing, IEEE Transactions on*, vol. 49, no. 10, pp. 2408–2421, Oct 2001.

[10] J. Buck and E. 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*, vol. 1, April 1993, pp. 429–432 vol.1.

[11] P. Fradet, A. Girault, and P. Poplavko, "SPDF: A schedulable parametric data-flow MoC," in *Proceedings of the Conference on Design, Automation and Test in Europe*, ser. DATE '12. San Jose, CA, USA: EDA Consortium, 2012, pp. 769–774.

[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*, Sept 2013, pp. 1–10.

[13] M. Wiggers, M. Bekooij, and G. 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*, April 2008, pp. 183–194.

[14] A. Ghamarian, M. Geilen, S. Stuijk, T. Basten, A. Moonen, M. Bekooij, B. Theelen, and M. Mousavi, "Throughput analysis of synchronous data flow graphs," in *Application of Concurrency to System Design, 2006. ACSD 2006. Sixth International Conference on*, June 2006, pp. 25–36.

[15] A. Ghamarian, S. Stuijk, T. Basten, M. Geilen, and B. Theelen, "Latency minimization for synchronous data flow graphs," in *Digital System Design Architectures, Methods and Tools, 2007. DSD 2007. 10th Euromicro Conference on*, Aug 2007, pp. 189–196.

[16] M. Geilen, "Synchronous dataflow scenarios," *ACM Trans. Embed. Comput. Syst.*, vol. 10, no. 2, pp. 16:1–16:31, Jan. 2011.

[17] 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*, Aug 2008, pp. 367–376.

[18] F. Baccelli, G. Cohen, G. J. Olsder, and J.-P. Quadrat, "Synchronization and linearity: an algebra for discrete event systems," 2001.

[19] S. Gaubert, "Performance evaluation of (max,+) automata," *Automatic Control, IEEE Transactions on*, vol. 40, no. 12, pp. 2014–2025, Dec 1995.

[20] P. Clauss and V. Loechner, "Parametric analysis of polyhedral iteration spaces," *Journal of VLSI signal processing systems for signal, image and video technology*, vol. 19, no. 2, pp. 179–194, 1998.

[21] 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*, ser. DATE '08. New York, NY, USA: ACM, 2008, pp. 116–121.

[22] A. Bouakaz, P. Fradet, and A. Girault, "Symbolic buffer sizing for throughput-optimal scheduling of dataflow graphs," in *2016 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, April 2016, pp. 1–10.

[23] 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*, Sept 2012, pp. 219–226.

[24] M. Skelin, M. Geilen, F. Catthoor, and S. 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.*, 2014, pp. 65–79.

[25] E. Lee and A. Sangiovanni-Vincentelli, "A framework for comparing models of computation," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 17, no. 12, pp. 1217–1229, Dec 1998.

[26] K. Kavi, B. Buckles, and U. N. Bhat, "A formal definition of data flow graph models," *Computers, IEEE Transactions on*, vol. C-35, no. 11, pp. 940–948, Nov 1986.

[27] M. Geilen, S. Tripakis, and M. Wiggers, "The earlier the better: A theory of timed actor interfaces," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2010-130, Oct 2010.

[28] C. Ptolemaeus, Ed., *System Design, Modeling, and Simulation using Ptolemy II*. Ptolemy.org, 2014.

[29] E. A. Lee, "A denotational semantics for dataflow with firing," in *Memorandum UCB/ERL M97/3, Electronics Research Labaratory, Berkeley, CA 94720*, 1997.

[30] F. Siyoum, M. Geilen, O. Moreira, and H. 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*, ser. CODES+ISSS '12. New York, NY, USA: ACM, 2012, pp. 463–472.

[31] 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*, 1999, pp. 805–810.

[32] S. S. Battacharyya, E. A. Lee, and P. K. Murthy, *Software Synthesis from Dataflow Graphs*. Norwell, MA, USA: Kluwer Academic Publishers, 1996.

[33] K. Desnos, M. Pelcat, J.-F. Nezan, 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*, July 2013, pp. 41–48.

[34] B. Bhattacharya and S. Bhattacharyya, "Quasi-static scheduling of reconfigurable dataflow graphs for DSP systems." in *IEEE International Workshop on Rapid System Prototyping*. IEEE Computer Society, 2000, pp. 84–89.

[35] M. H. Wiggers, M. J. G. Bekooij, and G. 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*, ser. SCOPES '07. New York, NY, USA: ACM, 2007, pp. 11–22.

[36] S. Gaubert, P. Butkovic, and R. Cuninghame-Green, "Minimal (max,+) realization of convex sequences," *SIAM Journal on Control and Optimization*, vol. 36, no. 1, pp. 137–147, 1998.

[37] B. Charron-Bost, M. Függer, and T. Nowak, *Transience Bounds for Distributed Algorithms*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 77–90.

[38] 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*, July 2011, pp. 404–411.

[39] 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*, vol. 35, no. 4, pp. 535–548, April 2016.

[40] A. Dasdan and R. Gupta, "Faster maximum and minimum mean cycle algorithms for system-performance analysis," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 17, no. 10, pp. 889–899, Oct 1998.

[41] F. Bugarin, D. Henrion, and J.-B. Lasserre, "Minimizing the sum of many rational functions," *arXiv preprint arXiv:1102.4954*, 2011.

[42] H. Sherali and C. Tuncbilek, "A global optimization algorithm for polynomial programming problems using a reformulation-linearization technique," *Journal of Global Optimization*, vol. 2, no. 1, pp. 101–112, 1992.

[43] M. J. D. Powell, *Approximation theory and methods*. Cambridge university press, 1981.

[44] B. Heidergott, G. J. Olsder, and J. 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.

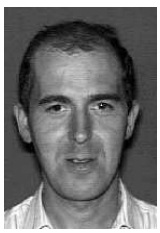
[45] J. Cochet-Terrasson, G. Cohen, S. Gaubert, M. McGettrick, and J.-P. Quadrat, "Numerical computation of spectral elements in max-plus algebra," in *Proc. IFAC Conf. on Syst. Structure and Control*, 1998.

[46] D. Grois and O. Hadar, "Recent advances in region-of-interest coding," *Recent Advances on Video Coding*, pp. 49–76, 2011.

[47] V. Bebelis, P. Fradet, and A. 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*, ser. LCTES '14. New York, NY, USA: ACM, 2014, pp. 125–134.

Mladen Skelin received the M.Eng. degree from the Faculty of Electrical Engineering and Computing, University of Zagreb, Zagreb, Croatia, and the Ph.D. degree from the Norwegian University of Science and Technology, Trondheim, Norway and KU Leuven, Leuven, Belgium. He is a post-doctoral researcher with the Department of Electrical Engineering, Eindhoven University of Technology, Eindhoven, The Netherlands. His current research interests include model-based design and analysis of embedded systems with a special focus on dataflow models of computation.

Marc Geilen is an assistant professor in the Department of Electrical Engineering at Eindhoven University of Technology. He holds an MSc and a PhD from Eindhoven University of Technology. His research interests include formal models of computation, multiprocessor systems-on-chip, networked embedded systems and cyber-physical systems, and multi-objective optimization and trade-off analysis. He is a member of IEEE. He has served on various TPCs and on organizing committees for several conferences including DATE as a topic chair. He co-supervised 6 PhD students.



Francky Catthoor received the engineering degree and a PhD in electrical engineering from the Katholieke Universiteit Leuven, Belgium in 1982 and 1987 respectively. Between 1987 and 2000, he has headed several research domains in the area of high-level and system synthesis techniques and architectural methodologies, including related application and deep submicron technology aspects, and smart photo-voltaic modules, all at IMEC, Heverlee, Belgium. Currently he is an imec fellow. He is part-time full professor at the EE department of the K.U.Leuven. In 1986 he received the Young Scientist Award from the Marconi International Fellowship Council. He has been associate editor for several IEEE and ACM journals, like Trans. on VLSI Systems, Trans. on Multimedia, and ACM TODAES. He was the program chair of several conferences including ISSS97 and SIPS01. He has been elected an IEEE fellow in 2005.

Sverre Hendseth serves as an associate professor in the Department of Engineering Cybernetics at the Norwegian University of Science and Technology. He received his Siv.Ing. (MSc) and Dr.Ing. (PhD) degree in engineering cybernetics from the Norwegian Institute of Technology (NTH) in 1987 and 1995, respectively. His research interests include real-time systems and embedded software.

APPENDIX A
DERIVING THE SDFG MAX-PLUS MATRIX OF (6)

The Max-plus matrix of the scenario s_1 SDFG of Fig. 2a is given in (6). We show how to derive it via symbolic execution of one iteration of the SDFG.

The example SDFG has six initial tokens. We represent the timestamp vector of the kth graph iteration as

$$\boldsymbol{\gamma}(k) = [t_{i_1}, t_{i_2}, t_{i_3}, t_{i_4}, t_{i_5}, t_{i_6}]^T. \tag{72}$$

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

$$\boldsymbol{\gamma}(k+1) = [t'_{i_1}, t'_{i_2}, t'_{i_3}, t'_{i_4}, t'_{i_5}, t'_{i_6}]^T. \tag{73}$$

The iteration schedule of the graph is given as follows: $A_1^1 A_2^2 A_3^3 A_4^3 A_5^2 A_6^1$.

According to the iteration 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

$$t_{i_6} = [-\infty, -\infty, -\infty, -\infty, -\infty, 0] \otimes \boldsymbol{\gamma}(k). \tag{74}$$

Therefore, according to the Max-plus semantics of SDF of (2), the tokens produced by its firing are determined by the timestamp vector

$$t_{i_6} \otimes 0 = [-\infty, -\infty, -\infty, -\infty, -\infty, 0] \otimes \boldsymbol{\gamma}(k). \tag{75}$$

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

$$t_{i_1} = [0, -\infty, -\infty, -\infty, -\infty, -\infty] \otimes \boldsymbol{\gamma}(k). \tag{76}$$

The timestamps of tokens of channel (A_0, A_1) are given by (75). 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

$$\begin{aligned} &([0, -\infty, -\infty, -\infty, -\infty, -\infty] \otimes \boldsymbol{\gamma}(k) \\ &\oplus [-\infty, -\infty, -\infty, -\infty, -\infty, 0] \otimes \boldsymbol{\gamma}(k)) \otimes 5 \\ &= [5, -\infty, -\infty, -\infty, -\infty, 5] \otimes \boldsymbol{\gamma}(k). \end{aligned} \tag{77}$$

Similarly, the second firing consumes the self-edge token and the remaining token from (A_0, A_1) whose timestamp is given by (75). However, now the self-edge token carries the timestamp of (77) as it was produced in the first firing of A_1. Therefore, the three tokens produced by A_1 carry the timestamp

$$\begin{aligned} &([5, -\infty, -\infty, -\infty, -\infty, 5] \otimes \boldsymbol{\gamma}(k) \\ &\oplus [-\infty, -\infty, -\infty, -\infty, -\infty, 0]) \otimes 5 \\ &= [10, -\infty, -\infty, -\infty, -\infty, 10] \otimes \boldsymbol{\gamma}(k) \end{aligned} \tag{78}$$

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

$$t'_{i_1} = [10, -\infty, -\infty, -\infty, -\infty, 10] \otimes \boldsymbol{\gamma}(k). \tag{79}$$

By continuing the symbolic execution until the completion of the iteration we will obtain the new timestamps of remaining initial tokens. By collecting the corresponding dependency vectors, we obtain M_G of (6).

APPENDIX B
PROOF OF PROPOSITION 1

Proof. We prove this by induction using the argument

$$x \leq \lceil x \rceil < x + 1. \tag{80}$$

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 (30), we obtain

$$conv(\varsigma_1, \varsigma_2)(n) = \delta_1 \otimes \bigoplus_{i=1}^{n} {\pi_1}^{\otimes \lceil r \cdot (n-i+1) \rceil} \otimes {\pi_2}^{\otimes i}. \tag{81}$$

For $n = 1$, (81) reduces to

$$conv(\varsigma_1, \varsigma_2)(1) = \delta_1 \otimes {\pi_1}^{\otimes \lceil r \rceil} \otimes \pi_2. \tag{82}$$

By combining (80) and (82) we obtain the following inequality

$$conv(\varsigma_1, \varsigma_2)(1) = \delta_1 \otimes {\pi_1}^{\otimes \lceil r \rceil} \otimes \pi_2 < \delta_1 \otimes {\pi_1}^{\otimes (1+r)} \otimes \pi_2 \tag{83}$$

that proves the base case. We continue with the induction step, i.e. evaluate (81) for $(n+1)$ with the induction hypothesis of (32) where $\pi_2 \geq r \cdot \pi_1$. We obtain

$$\begin{aligned} conv(\varsigma_1, \varsigma_2)(n+1) &= \delta_1 \otimes \bigoplus_{i=1}^{n+1} {\pi_1}^{\otimes \lceil r \cdot (n-i+2)) \rceil} \otimes {\pi_2}^{\otimes i} \\ &= \delta_1 \otimes \bigoplus_{i=1}^{n} {\pi_1}^{\otimes \lceil r \cdot (n-i+1) \rceil} \otimes {\pi_2}^{\otimes i} \\ &\oplus \delta_1 \otimes {\pi_1}^{\otimes \lceil r \rceil} \otimes {\pi_2}^{\otimes (n+1)} \\ &= conv(\varsigma_1, \varsigma_2)(n) \\ &\oplus \delta_1 \otimes {\pi_1}^{\otimes \lceil r \rceil} \otimes {\pi_2}^{\otimes (n+1)}. \end{aligned} \tag{84}$$

By substituting the induction hypothesis into (84) we obtain the following inequality

$$\begin{aligned} conv(\varsigma_1, \varsigma_2)(n+1) < \delta_1 \otimes {\pi_1}^{\otimes (1+r)} \otimes {\pi_2}^{\otimes n} \\ \oplus \delta_1 \otimes {\pi_1}^{\otimes \lceil r \rceil} \otimes {\pi_2}^{\otimes (n+1)}. \end{aligned} \tag{85}$$

If we use (80) to get rid of the ceiling in (85), we obtain

$$\begin{aligned} conv(\varsigma_1, \varsigma_2)(n+1) &< \delta_1 \otimes {\pi_1}^{\otimes (1+r)} \otimes {\pi_2}^{\otimes n} \\ &\oplus \delta_1 \otimes {\pi_1}^{\otimes (1+r)} \otimes {\pi_2}^{\otimes (n+1)} \\ conv(\varsigma_1, \varsigma_2)(n+1) &< \delta_1 \otimes {\pi_1}^{\otimes (1+r)} \otimes {\pi_2}^{\otimes (n+1)}. \end{aligned} \tag{86}$$

Inequality (86) 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 (82). We proceed with

the induction step. By substituting the induction hypothesis into (84) we obtain the following inequality

$$\begin{aligned} conv(\varsigma_1,\varsigma_2)(n+1) < \delta_1 \otimes \pi_2 \otimes {\pi_1}^{\otimes(1+r\cdot n)} \\ \oplus \delta_1 \otimes {\pi_1}^{\otimes\lceil r\rceil} \otimes {\pi_2}^{\otimes(n+1)}. \end{aligned} \tag{87}$$

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

$$\begin{aligned} conv(\varsigma_1,\varsigma_2)(n+1) < \delta_1 \otimes \pi_2 \otimes ({\pi_1}^{\otimes(1+r\cdot n)} \\ \oplus {\pi_1}^{\otimes(1+r)} \otimes \underbrace{{\pi_2}^{\otimes n}}_{{\pi_1}^{\otimes(r\cdot n)}}). \end{aligned} \tag{88}$$

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 (88)) so that the (88) still holds. We obtain

$$\begin{aligned} conv(\varsigma_1,\varsigma_2)(n+1) < \delta_1 \otimes \pi_2 \otimes ({\pi_1}^{\otimes(1+r\cdot n)} \\ \oplus {\pi_1}^{\otimes(1+r)} \otimes {\pi_1}^{\otimes(r\cdot n)}) \\ conv(\varsigma_1,\varsigma_2)(n+1) < \delta_1 \otimes \pi_2 \otimes ({\pi_1}^{\otimes(1+r\cdot n)} \\ \oplus {\pi_1}^{\otimes(1+r\cdot(n+1))}). \end{aligned} \tag{89}$$

If follows from (89) that

$$conv(\varsigma_1,\varsigma_2)(n+1) < \delta_1 \otimes \pi_2 \otimes ({\pi_1}^{\otimes(1+r\cdot(n+1))}), \tag{90}$$

which completes the proof. □

APPENDIX C
PROOF OF PROPOSITION 2

Proof. By taking π for the growth rate (ratio) of the conservative estimate and 0 for the initial delay, the conservative estimate 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 maximum ratio will dominate. By taking δ for the delay of the estimate, the estimate will dominate for all $n \in \mathbb{N}_{>0}$. □