Probabilistic Model Checking for Uncertain Scenario-Aware Data Flow

Article *in* ACM Transactions on Embedded Computing Systems · February 2016

DOI: 10.1145/2914788

2 authors:

Joost-Pieter Katoen
RWTH Aachen University

Hao Wu
RWTH Aachen University

Probabilistic Model Checking for Uncertain Scenario-Aware Data Flow

JOOST-PIETER KATOEN, Software Modelling and Verification Group, RWTH Aachen University
HAO WU, Software Modelling and Verification Group, RWTH Aachen University

The Scenario-Aware Dataflow (SADF) model is based on concurrent actors that interact via channels. It combines streaming data and control to capture scenarios while incorporating hard and soft real-time aspects. To model data-flow computations that are subject to uncertainty, SADF models are equipped with random primitives. We propose to use probabilistic model checking to analyse uncertain SADF models. We show how measures such as expected time, long-run objectives like throughput, as well as timed reachability—can a given system configuration be reached within a deadline with high probability?—can be automatically determined. The crux of our method is a *compositional* semantics of SADF with exponential agent execution times combined with *automated abstraction* techniques akin to partial-order reduction. We present the semantics in detail, and show how it accommodates the incorporation of execution platforms enabling the analysis of energy consumption. The feasibility of our approach is illustrated by analysing several quantitative measures of an MPEG-4 decoder and an industrial face recognition application.

Additional Key Words and Phrases: Data-flow computation, Digital signal processing, Embedded systems, Energy consumption, Markov (Reward) automata, Model checking

ACM Reference Format:
Joost-Pieter Katoen, Hao Wu, 2016. Probabilistic Model Checking for Uncertain Scenario-Aware Data Flow *ACM Trans. Embedd. Comput. Syst.* XX, YY, Article (January 2016), 28 pages.
DOI: 0000001.0000001

1. INTRODUCTION

Synchronous Data Flow. Embedded systems such as smart phones, TV sets and modern printing devices typically involve intensive multimedia processing. Current applications require massive data signal processing facilities like photo editing, face recognition, audio and video streaming, and need to adhere to demanding QoS requirements. Such data processing facilities are adequately described in data-flow languages such as Synchronous Dataflow (SDF, for short) [Lee and Messerschmitt 1987], a language which is equally expressive as weighted marked graphs. SDF has been used extensively [Bhattacharyya et al. 2013; Eker et al. 2003; Sriram and Bhattacharyya 2009] and originates from the field of digital signal processing where a coherent set of interacting tasks with different execution frequencies are to be performed in a distributed and pipelined fashion by a number of parallel computing resources as provided, e.g., by Multi-Processor Systems-on-Chips (MPSoC). Modern embedded applications are very dynamic in the sense that their execution costs such as memory usage,

This work is supported by the European Union's Seventh Framework Programme for Research (FP7), under grant no. 318490 (FP7-ICT-2011-8, the SENSATION project). This work is partially based on the authors' previous work "Exponentially timed SADF: Compositional semantics, reductions, and analysis." published at EMSOFT 2014 in New Delhi, India.
Author's addresses: Joost-Pieter Katoen/Hao Wu, Software Modelling and Verification Group, Computer Science Department, RWTH Aachen University, D-52066, Aachen, Germany, Email: {katoen, hao.wu}@cs.rwth-aachen.de.

DOI: 0000001.0000001

energy consumption vary substantially depending on their context (like input data and quality level).

Uncertainty in Scenario-Aware Data Flow. Data-flow languages [Buck and Lee 1993; Geilen and Basten 2004; Kahn 1974] are intended as an implementation framework for such dynamic applications. However, these languages lack facilities for predicting performance during embedded system design—how to deal with uncertain fluctuations in data availability, data quality etc.—and have rather limited means to describe dynamic dataflow behaviour. The *Scenario-Aware Dataflow (SADF)* language [Theelen et al. 2008; Theelen et al. 2006; Bhattacharyya et al. 2013] extends SDF so as to develop adequate scenario-aware performance models of specifications expressed in other data-flow formalisms. Like most data-flow languages, SADF is based on (asynchronously concurrent) actors that interact via (unbounded) FIFO (first-in, first-out) channels. The novelty of SADF is its combination of streaming data and control to capture scenarios, as well as combining both hard and soft real-time aspects. A recent survey of data-flow formalisms such as SDF and SADF as well as their relationship is given in [Bhattacharyya et al. 2013]. This paper focuses in particular on the possibility in SADF to model *uncertainty*, both in the execution times of processes (called actors) as well as in the generation of scenarios.

Exponentially timed SADF. This paper considers *exponentially timed* SADF (called eSADF), a version of SADF in which the duration of all firings of actors are governed by negative exponential probability distributions. eSADF can be considered as an extension of exponentially timed SDF as originally proposed in [Sriram and Bhattacharyya 2009].) Although the restriction to exponential distributions seems a severe restriction at first sight, there are (at least) three good reasons to consider it. First, this assumption is a rather adequate abstraction when considering that actor firings are typically subject to random fluctuations (e.g., in hardware) and only mean durations are known. Technically speaking, the exponential distribution maximises the entropy under these assumptions. That is to say, if only mean values are known, the statistically most neutral assumption is to have these phenomena exponentially distributed. Secondly, series-parallel combinations of exponential distributions (so-called phase-type distributions) can approximate any arbitrary continuous probability distribution with arbitrary precision. Our semantic model and analysis algorithms support the analysis of these phase-type distributions. Finally, the use of exponential distributions enables the usage of modern probabilistic model-checking tools for the quantitative analysis of SADF models.

Compositional semantics. As eSADF is based on asynchronously communicating actors, firings have exponential durations, and sub-scenario selection is based on discrete-time Markov chains, *Markov automata* (MA) [Deng and Hennessy 2013; Eisentraut et al. 2010] are a natural choice for capturing the semantics of eSADF. These automata are transition systems in which the target of an interactive transition is a distribution over states (rather than a single state), and that incorporates random delay transitions to model firing durations. Non-determinism occurs if several interactive transitions emanate from a given state. This paper provides a *compositional* semantics of eSADF using Markov automata. The compositional aspect naturally reflects the logical structure of the eSADF graph, is easily amenable to single actor replacements—as just the semantics of that actor is to be adapted while the remaining automata remain the same—and finally enables component-wise reduction. The latter technique is important in case the state space of the eSADF graph is too large to be handled. Our compositional semantics allows for replacing the automaton for a few actors by an equivalent, but much smaller, automaton. This technique has e.g., been exploited in [Theelen et al. 2012]. Compositionality in SDF has recently also been exploited for modular code generation [Tripakis et al. 2013]. Our semantics is defined using a succinct process algebra for describing MA [Timmer et al. 2012] in a textual way. As a result, the semantics is relatively easy to comprehend and modular.

Automated abstraction. Markov automata of realistic, industrially-sized eSADF graphs can be huge and too large to be handled. One of the main causes for this state space exposition is the highly concurrent character of typical data-flow computations in which many agents run in parallel. To diminish this effect on the state space growth, we use *confluence reduction* [Timmer et al. 2013]—a technique akin to partial-order reduction—that allows for an on-the-fly reduction of the state space. The key of confluence reduction is to detect independent concurrent transitions such that for the analysis only one ordering of concurrent transitions needs to be considered (instead of all possible orderings). We show that all non-determinism in the MA-semantics of eSADF arises from the independent execution of actors, and can (in theory) be eliminated using confluence reduction. As confluence reduction is performed at the language level (i.e., the process algebra) using conservative heuristics, non-determinism may persist after reduction, but if so, it does not influence quantitative measures.

Quantitative analysis of eSADF graphs. To analyse eSADF graphs we exploit recently developed algorithms and software tool-support for verifying Markov automata. The main challenge in MA analysis is the intricate interplay between exponential durations and non-determinism. The latter arises naturally by the concurrent execution of the different actors. It was recently shown that several measures-of-interest such as expected time and long-run average objectives can be reduced to efficient analysis techniques for Markov decision processes [Guck et al. 2014a]. In addition, timed reachability objectives—can a certain system configuration be reached within a deadline with a high probability?—were shown to be computable by appropriate discretisation techniques [Guck et al. 2014a], and extensions towards the treatment of costs (modelling energy consumption) in Markov automata have become available [Guck et al. 2014b]. Our MA semantics facilitates the quantitative analysis of eSADF graphs using these novel and efficient analysis techniques. To sum up, our semantics is conceptually simple, compositional, and yields a rigorous framework for quantitative analysis of eSADF.

Case studies. We have developed a prototypical implementation of our approach that maps eSADF graphs (expressed in the XML-format as supported by the SDF3 tool [1]) onto Markov automata. These MA can then be analysed by the analysis tool presented in [Guck et al. 2014a]. An extension with confluence reduction enables the minimisation of MA. This paper shows the feasibility of our approach by presenting two case studies. The MPEG-4 decoder is a benchmark SADF case from the literature [Theelen et al. 2006]. We show the effect of confluence reduction and analyse several measures of interest of the MPEG-4 decoder such as buffer occupancy, throughput, and probability to reach certain buffer occupancies within a given deadline. We compare the results to analysis results using the SDF3 tool for SADF and to [Theelen et al. 2012]. As a second case study we present the analysis of an industrial face recognition application. This model is substantially larger than the MPEG-4 decoder as it exhibits a high degree of parallelism. We study the quantitative effect of including auto-concurrency, and analyse several metrics. We conclude by presenting an extension of our framework by incorporating execution platforms on which the eSADF agents are supposed to be executed. This allows for studying the quantitative effect of exploiting multi-core

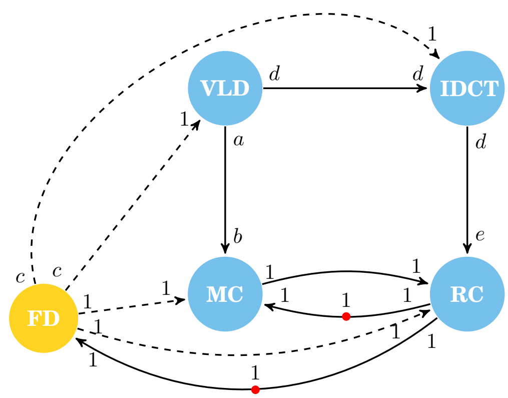
[1] See http://www.es.ele.tue.nl/sadf/xml.php.

platforms, as well as the quantitative impact of dynamic voltage and frequency scaling (DVFS) and dynamic power management (DPM). We present an energy analysis of the MPEG-4 decoder for a simple execution platform.

Organisation of the paper.. Section 2 and 3 introduce eSADF and MA, the operational model used for our semantics, respectively. Section 4 presents the compositional eSADF semantics in detail. Section 5 discusses non-determinism, and the quantitative evaluation of eSADF using MA. Section 6 presents the MPEG-4 decoder and the face recognition case studies. Section 7 presents our approach to incorporate execution platforms. Section 8 discussed related work and Section 9 concludes. This paper is based on the conference paper [Katoen and Wu 2014] and extends it with the industrial face recognition case study, the incorporation of the executing platform, and an energy analysis of the MPEG-4 decoder benchmark.

2. SCENARIO-AWARE DATAFLOW

In this section, we will give a formal definition of an exponentially-timed SADF (eSADF) graph which is based on [Theelen 2007]. Figure 1 illustrates an SADF graph of an MPEG4-decoder. We will first define the channels in an eSADF graph, which can transfer information between the agents (called processes) in eSADF. Based on the types of the elements that the channels can transfer, the scenarios are defined. Afterwards, we introduce two distinct kinds of processes, i.e., kernels and detectors. In order to build an eSADF graph correctly, the processes are connected by channels which are assumed to be "type-consistent" with the processes' ports. In the end, we give the formal definition of an eSADF graph. Let $\mathcal{C}$ denote the set of channels.



Rate	Scenario I	P_0	P_x
a	0	0	1
b	0	0	x
c	99	0	x
d	1	0	1
e	99	0	x

$x \in \{30, 40, 50, 60, 70, 80, 99\}$

Fig. 1. An SADF model for an MPEG-4 decoder [Theelen et al. 2006]

DEFINITION 1. **(Channel).** *A* channel $c \in \mathcal{C}$ *is a (possibly unbounded) FIFO queue to carry information modeled by tokens of a certain type (e.g. none, integers, Boolean, symbols, etc.). Based on these types we distinguish channels by:*

— data *channels whose type is none, i.e., its tokens are only placeholders and not valued, and*
— control *channels whose type is not none.*

The justification of having two types of channels is that data channels are like the channels in traditional SDF [Lee and Messerschmitt 1987], whereas control channels are key features in (e)SADF to carry the scenario (control) information between processes. We therefore have $\mathcal{C} = \mathcal{DC} \cup \mathcal{CC}$ with $\mathcal{DC}$ the set of data channels, $\mathcal{CC}$ the set of control channels, and $\mathcal{DC} \cap \mathcal{CC} = \emptyset$. For a control channel $cc \in \mathcal{CC}$, we denote by Σ_{cc} the type of cc, which is the set of elements (values of tokens) that can be carried in cc, e.g. $\Sigma_{cc} = \{a, \ldots, z\}$ or $\Sigma_{cc} = \mathbb{B} = \{0, 1\}$. Based on these types of channels, we can now define the scenario for an ordered set C of control channels as follows.

DEFINITION 2. **(Scenario)** *For an ordered set $C = (cc_1, \ldots, cc_k)$ of control channels, a* scenario *is an ordered k-tuple $(\sigma_1, \ldots, \sigma_k)$ where σ_i is a value of type Σ_{cc_i} for channel $cc_i \in C$. The set of possible scenarios for C is denoted by $\Sigma_C = \prod_{cc \in C} \Sigma_{cc}$.*

If we consider $C = (cc_1, cc_2)$ with $k = 2$, $\Sigma_{cc_1} = \{s, t\}$ and $\Sigma_{cc_2} = \{1, 2\}$ for example, the possible scenario set $\Sigma_C = \{(s, 1), (s, 2), (t, 1), (t, 2)\}$. The set of scenarios can be used to transfer the control information.

Now we define kernels and detectors, which can be treated as the "executable" processes in eSADF. Processes are connected with each other by channels through their ports. We denote the set of kernels as $\mathcal{K}$, the set of detectors as $\mathcal{D}$ with $\mathcal{K} \cap \mathcal{D} = \emptyset$. The set of processes $\mathcal{P} = \mathcal{K} \cup \mathcal{D}$.

DEFINITION 3. **(Kernel).** *A* kernel *K is a pair $(P, \mathcal{S})$ with*

— *P is a* set of ports *partitioned into the set P_I, P_O and P_C of input, output and control ports, respectively;*

— *$\mathcal{S}$ is the* scenario setting *(Σ, R, E) (a triple) with:*

$\triangleright$ *$\Sigma = \prod_{p_c \in P_C} \Sigma_{p_c}$ is the* set of scenarios *in which K can operate, where Σ_{p_c} is the type (set of values) that is allowed to pass through port p_c,*

$\triangleright$ *$R : \Sigma \times P \to \mathbb{N}$, the* consumption/production rate function*, such that $R(\sigma, p_c) = 1$ for any control port p_c and scenario σ,*

$\triangleright$ *$E : \Sigma \to \mathbb{R}_{>0}$, the* execution rate function*, i.e., $E(\sigma) = r$ means that the execution time of scenario σ by K is exponentially distributed with mean $1/r$.*

Intuitively, for a kernel K, the possible scenarios in which it can operate are given by the values of ordered tuples from its control ports. The consumption/production rate function gives the number of tokens to be consumed or produced after K's execution in such scenarios into K's ports accordingly. Since at each time we only need one token from each control port to select the scenario, $R(\sigma, p_c)$ equals 1 for every control channel. For data channels the rate function may be arbitrarily large (but finite).

Detectors are not only an "executable" component similar to a kernel, but are equipped with more features such as capturing the (sub-)scenario occurrence and sending control information by the generation of control tokens. Detectors do not operate in scenarios but in sub-scenarios which are determined by scenarios received via the control ports.

DEFINITION 4. **(Detector).** *A* detector *D is pair $(P, \mathcal{S})$ with*

— *P is a* set of ports *partitioned into P_I, P_O and P_C (as for kernels). P_O is partitioned into P_{O_d} and P_{O_c}, the data and control output ports;*

— *$\mathcal{S}$ is the* sub-scenario setting *$(\Sigma, \Omega, F, R, E, t)$ with*

$\triangleright$ *$\Sigma = \prod_{p_c \in P_C} \Sigma_{p_c}$ is the set of scenarios of D,*

$\triangleright$ *Ω, a non-empty finite* set of sub-scenarios values*,*

$\triangleright$ *$F : \Sigma \to M$, the* random decision function*. M is the set of DTMCs $(\mathbb{S}, \iota, \mathbb{P}, \Phi)$* [2]*. Function F associates a DTMC to each scenario $\sigma \in \Sigma$. Here, $\mathbb{S}, \iota$ and $\mathbb{P}$ are the finite*

[2]DTMC = Discrete-Time Markov Chain. This is a finite-state automaton in which all transitions are equipped with a discrete probability.

state space, initial state, and one-step transition probability function, respectively, and $\Phi : \mathbb{S} \to \Omega$ *the sub-scenario decision function,*

$\triangleright$ $R : \Omega \times P \to \mathbb{N}$*, the* consumption/production rate *function, such that* $R(\omega, p_c) = 1$ *for any sub-scenario* ω *and control port* p_c*,*

$\triangleright$ $E : \Omega \to \mathbb{R}_{>0}$*, the* execution rate time *function,*

$\triangleright$ $t : \Omega \times P_{O_c} \to \Sigma_{p_{o_c}}$*, the* sub-scenario token production *function.* $t(\omega, p_{o_c})$ *determines the value of a token to be produced into an output port* $p_{o_c} \in P_{O_c}$ *after* D*'s execution in sub-scenario* ω*. Note that* $R(\omega, p_{o_c})$ *is the number of valued tokens to be produced.*

Intuitively, a detector works in two phases. In the first phase, scenarios of the detector are determined in the same way as for kernels. After the scenario σ is determined, i.e. there is at least one scenario token in each control channel, the DTMC of this scenario is entered, which is defined by $F(\sigma) \in M$. The entry point of such DTMC is its last occupied state. After moving from this state to one of its successor states, the current sub-scenarios and probabilities of new sub-scenarios can be computed by the functions Φ and $\mathbb{P}$. This is an important difference with SDF: the next sub-scenario in SADF is determined probabilistically, so as to model uncertainty. (This should not be confused with the execution times of actors being randomly distributed.) In the second phase, D will execute in these sub-scenarios. The rate function R and expected execution time function E are then defined for such execution. After the execution of the sub-scenario, the detector D consumes/produces the tokens of its ports. In order to send control information, the valued tokens defined by function t are produced into control output ports and the number of such tokens to be produced is defined by R.

In order to define the exponential-time SADF (eSADF) graph, we need a consistent way to connect the processes by using channels. We define the consistency between the channels and a process's ports, which determines whether a channel and two ports are compatible.

DEFINITION 5. **(Type consistency).** *We call a channel "type consistent" with two ports in a (two) process(es), if*

— *the channel is a data channel and it connects a (data) output port of a kernel (detector) with an input port of same / another process;*

— *the channel is a control channel and it connects a control output port of a detector with a control port of same / another process, and in addition the types of the channel and both ports coincide.*

An eSADF graph is *type consistent* if all its channels are type consistent.

DEFINITION 6. **(eSADF graph).** *An eSADF graph* $\mathcal{G} = (\mathcal{P}, \mathcal{C}^*)$*, where* $\mathcal{P}$ *is a finite set of processes (vertices),* $\mathcal{C}^*$ *is the set of "type consistent" channels (edges) of the form* $\big((src(c), tgt(c)\big)$ *where* $src(c), tgt(c)$ *are the source and target ports of channel* $c \in \mathcal{C}$*, with proper initialization.* [3]

Example 2.1. Figure 1 illustrates an eSADF graph of an MPEG4-decoder (for the simple profile [Theelen et al. 2006]). It consists of the kernels VLD, IDCT, MC and RC and the detector FD. The decoder can process I and P frames in video streams, which consist of different numbers of macro blocks. This involves operations like *Variable Length Decoding* (VLD), *Inverse Discrete Cosine Transformation* (IDCT), *Motion Compensation* (MC), and *Reconstruction* (RC). The kernels VLD and IDCT fire once

[3] Note that "rate" consistent eSADF can be defined as in [Theelen et al. 2006].

per macro block in a decoded frame, while MC and RC fire once per frame. The detector FD detects the frame type occurrences in video streams. If it detects an I frame, all macro blocks are decoded by VLD and IDCT, and RC will reconstruct the image (without MC). We assume an image size of 176×144 pixels (QCIF), i.e, an I frame consists of 99 macro blocks (cf. Figure 1 right table). If FD detects a P frame, then MC computes the correct position of macro blocks from the previous frame out of motion vectors. We assume that the number of processing motion vectors equals the number of decoding macro blocks and is 0 or $x \in \{30, 40, 50, 60, 70, 80, 99\}$ representing conservative approximations for a range of real motion vectors [Theelen et al. 2006]. This is captured by the scenarios P_0 and P_x as indicated in Figure 1 (right). All parameters are taken from [Theelen et al. 2006], which are obtained by using a specific profiling tool.

3. MARKOV AUTOMATA

In this section, we introduce the semantic model, Markov automata (MA), for eSADF. Briefly speaking, an MA is an extended labeled transition system (LTS) equipped with both continuous time stochastic and nondeterministic transitions, and hence able to expresses the complete semantics [Eisentraut et al. 2013] of modelling languages such as dynamic fault trees [Boudali et al. 2009], domain-specific language AADL [Bozzano et al. 2011], and generalised stochastic Petri nets (GSPNs) [Marsan et al. 1984]. The treatment of MA is kept brief here; for a full description we refer to [Eisentraut et al. 2010; Deng and Hennessy 2013]. The crucial observation is that MA are a very natural semantic model of eSADF: concurrency can be modelled by non-determinism (interleaving), exponential delays by Markovian transitions, the probabilistic selection of sub-scenarios by discrete probabilistic branching, and MA are compositional. To the best of our knowledge there is no other model that has these characteristics.

The syntax of MA. A distribution μ over a countable set S is a function $\mu : S \to [0, 1]$ such that $\sum_{s \in S} \mu(s) = 1$. Let $\text{Distr}(S)$ be the set of all distributions over set S.

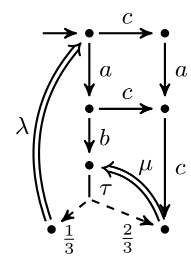
DEFINITION 7. **(Markov automata)** *A* Markov automaton (MA) *is a tuple* $\mathcal{M} = (S, s_0, Act, \hookrightarrow, \Rightarrow)$*, where*

— S *is a countable set of* states*,*
— $s_0 \in S$ *is the* initial state*,*
— Act *is a countable set of* actions*,*
— $\hookrightarrow \subseteq S \times Act \times \text{Distr}(S)$ *is the* interactive probabilistic transition relation*,*
— $\Rightarrow \subseteq S \times \mathbb{R}_{>0} \times S$ *is the* Markovian transition relation*.*

The semantics of MA. We let $\tau \in Act$ denote the invisible internal action and abbreviate $(s, a, \mu) \in \hookrightarrow$ as $s \overset{a}{\hookrightarrow} \mu$, which means if we are at the state s, the probability of reaching state s' by taking action a is $\mu(s')$.Similarly, we abbreviate $(s, \lambda, s') \in \Rightarrow$ as $s \overset{\lambda}{\Rightarrow} s'$, which means the probability of moving from state s to s' within time t is exponentially distributed and equals $1 - e^{-\lambda t}$. We call λ the *rate* of transition $s \overset{\lambda}{\Rightarrow} s'$. Furthermore, we say a state s is *Markovian* iff s has only outgoing Markovian transitions; it is *interactive* iff it has only outgoing interactive probabilistic transitions; it is a *hybrid* state, otherwise. For states s, s', we let $\mathbf{R}(s, s') = \sum\{\lambda \mid s \overset{\lambda}{\Rightarrow} s'\}$ be the *rate* between states s and s', and let $\mathbf{E}(s) = \sum_{s' \in S} \mathbf{R}(s, s')$ be the *exit rate* of s. The probability of leaving the state s within time t is given by $1 - e^{-\mathbf{E}(s) \cdot t}$. If $\mathbf{R}(s, s') > 0$ for more than one state s', a *race* exists between such states after leaving s. For a particular state s', the probability of winning the race is given by the *branching probability distribution* $\mathbf{P}(s, s') = \frac{\mathbf{R}(s,s')}{\mathbf{E}(s)}$.

Maximal progress.. If in a state both Markovian transitions and internal transitions τ are enabled, the maximal progress assumption asserts that internal transitions τ take precedence over Markovian transitions. This is justified by the fact that the probability of taking a Markovian transition immediately is zero (as $Pr(delay \leqslant 0) = 1 - e^{-\lambda \cdot 0} = 0$), whereas the internal transition τ happens immediately (since they cannot be delayed).

Closed MA. A *closed MA* is an MA which is self-contained and has no further synchronization with other components. For a closed MA, since all interactive probabilistic transitions are not subject to any other transitions for synchronization purpose, they cannot be delayed. Therefore, we can safely hide them and turn them into internal transitions τ. A closed MA has no *hybrid* state due to maximal progress. All outgoing transitions of a state in a closed MA are either interactive probabilistic transitions labelled by τ or Markovian transitions (cf. Figure 3 (c)). Note that non-determinism exists when there are multiple internal probabilistic transitions emanating from one state.



A sample MA

MA Process Algebra (MAPA). To generate an MA, a language based on μCRL [Groote and Ponse 1995] called MA Process Algebra (MAPA) was introduced in [Timmer et al. 2012]. We use MAPA to define our MA semantics for eSADF.

DEFINITION 8. **(Process terms).** *A* process term *in MAPA adheres to the syntax:*

$$p ::= Y(\mathbf{t}) \mid c \Rightarrow p \mid p + p \mid \sum_{\mathbf{x}:\mathbf{D}} p \mid a(\mathbf{t}) \sum_{\mathbf{x}:\mathbf{D}}\!\!\!\!\bullet\;\, f : p \mid (\lambda) \cdot p$$

Let Prc denote the set of process names, Act denote a countable universe of actions, and $\mathbf{x}$ denote a vector of variables ranging over a (possibly infinite) type domain $\mathbf{D}$. If the cardinality of $\mathbf{x}$ exceeds one, $\mathbf{D}$ is a Cartesian product. Observe that this matches the scenario definition based on types in eSADF graphs and simplifies our MA definition for eSADF graphs. In the process term $Y(\mathbf{t})$, $Y \in Prc$ is a process name, $\mathbf{t}$ is a vector of data expressions, and $Y(\mathbf{t})$ expresses a process Y initialized by setting its variables to $\mathbf{t}$. $c \Rightarrow p$ is a guarded expression, which asserts if the condition c (a boolean expression) is satisfied, then the process will behave like the process p, otherwise it will do nothing. The operator $p_1 + p_2$ expresses a non-deterministic choice between the left process p_1 and the right process p_2. Further if there is a (possibly infinite) nondeterministic choice over a data type $\mathbf{D}$, this is denoted by the term $\Sigma_{x:\mathbf{D}}\, p$. The term $a(\mathbf{t}) \sum\!\!\!\!\bullet_{\mathbf{x}:\mathbf{D}}\, f : p$ states that the process can perform an $a(\mathbf{t})$ action (an action based on the vector $\mathbf{t}$) and then resolves a probabilistic choice over $\mathbf{D}$ determined by a predefined function f. The function application $f[\mathbf{x} := \mathbf{d}]$ returns the probability when the variables are evaluated as $\mathbf{d} \in \mathbf{D}$. The term $(\lambda) \cdot p$ expresses that after an exponentially distributed delay with rate $\lambda \in \mathbb{R}_{\geqslant 0}$, the process will behave like p. We will see later that the MAPA language is concise and expressive enough to handle our eSADF semantics, since it allows processes with different data types and is equipped with both interactive probabilistic transitions and Markovian transitions. MA can now be obtained by the modular construction using MAPA processes through parallel composition, encapsulation, hiding and renaming operators [Timmer et al. 2012].

DEFINITION 9. **(Parallel process terms)** *An initial process in parallel MAPA is a term obeying the syntax:*

$$q ::= Y(\mathbf{t}) \mid q \parallel_I q \mid \partial_E(q) \mid \tau_H(q) \mid \rho_R(q)$$

Here, $Y \in Prc$ is a process name, $\mathbf{t}$ is a vector of data expressions, $I, E, H \subseteq Act$ are sets of actions, and the function $R : Act\backslash\{\tau\} \to Act\backslash\{\tau\}$. A parallel MAPA specification *is a set of MAPA process equations with an initial process q defined by the above rules.*

In an initial MAPA process generated by the rules above, $q_1 \parallel_I q_2$ is *parallel composition* w.r.t the action set I (defined in [Timmer et al. 2012] with $\gamma(a,a) = a, a \in I$) and $\tau_H(q)$ *hides* the actions in H, i.e., turns all the actions in H into τ and removes their parameters. Since we only use these two operators later in our definition, we refer for further information about MA and the MAPA language to [Timmer et al. 2012].

4. COMPOSITIONAL SEMANTICS OF ESADF

In this section, we define an MA semantics for eSADF. The first consideration is to model the channels and the processes in eSADF separately. This approach is easy to understand and the MAs for processes are finite whereas the MAs for channels are countably infinite as channels are unbounded. However, the drawback is the intermediate state space caused by the communication between the processes and the channels. For example, a data channel must either know the current operating scenario of the process which connects with it through its output port and then notify the process whether the tokens are available in such operating scenario or constantly send its channel status to the process. Hence we model the control channels as FIFO buffers and the data channel as natural numbers into the process's definition. As the process's behavior merely depends on the token availabilities and/or the contents of these channels, no further status synchronization between other components is needed. After all channels are integrated into the corresponding processes, the MAs for processes take care of the channel's status update. This is done by using the action synchronization between processes.

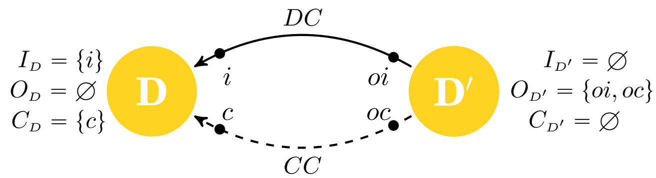
REMARK 1. *A kernel is a special kind of detector, in which*

(*1*) *no output channel is of type control channel,*

(*2*) *its subscenario set is identical with its scenario set, and*

(*3*) *for each scenario σ, $F(\sigma)$ is a Markov chain with only one state $s \in \mathbb{S}$, $\mathbb{P}(s,s) = 1$ and $\Phi(s) = \sigma$.*

From now on we will only consider the MA semantics of a detector, since the MA semantics of a kernel can be easily derived from the detector's semantics (due to Remark 1). Recall that a detector can have more than one data channel (data channel and/or control channels) connected through its input ports (input ports and/or control ports) of a single kernel (detector). Moreover, since we integrate the input channels as variables into their process's definition, we only consider these channels and the corresponding ports. For simplicity's sake we assume that the detector, say D, has only one such data channel and control channel from one detector, say D' (see following figure). This is easily generalized to several channels. Detector D has an input data channel $DC_{(D',D)}$ from D' which is connected through D''s output port $oi_{D',D}$ and D's input port $i_{D',D}$ and a control channel $CC_{(D',D)}$ from D' which is connected through D''s output port $oc_{D',D}$ and D's control port $c_{D',D}$. As mentioned earlier, the channels in eSADF are modeled as variables. Let the variable $dc_{(D',D)} \in \mathbb{N}$ represent the current number of tokens in data channel $DC_{(D',D)}$. For control channels, we let the variable $cc_{(D',D)}$ represent the current status of control channel $CC_{(D',D)}$ where $\Sigma_{(D',D)}$ is the set of values of the scenario token which can be stored in $CC_{(D',D)}$. Hence we have $cc_{(D',D)} \in \Sigma^*_{cc_{(D',D)}}$. Various operators on these variables are defined, for instance, $|\ cc_{(D',D)}\ |$ returns the length of the sequence; $\texttt{head}(cc_{(D',D)})$ returns the first element (head) of the sequence; $\texttt{tail}(cc_{(D',D)})$ is only defined for non-empty channels and returns the remaining string by removing $\texttt{head}(cc_{(D',D)})$, and $cc_{(D',D)} +\!\!+\ \omega$ for $\omega \in \Sigma^*$ denotes concatenation. If clear from the context, we omit the subscript (D',D). Note that the general definition for detectors with several data and control channels can be easily derived from this simplified definition, since we can just replace the single variable by a vector of variables.



The MA semantics of a detector D consists of two modules: the *(sub)scenario module* SM and the *function module* FM. First, we give the MA definition of scenario module SM. Since SM only communicates (synchronizes) with FM via requesting and waiting for sub-scenario decisions, no variable is used in SM. Recall that for each scenario $\sigma \in \Sigma$, $(\mathbb{S}, \iota, \mathbb{P}, \Phi)$ is a non-empty finite Markov chain $(\mathbb{S}, \iota, \mathbb{P})$ associated with a function $\Phi : \mathbb{S} \to \Omega$. Now we define an MA for $(\mathbb{S}, \iota, \mathbb{P}, \Phi)$ for each scenario $\sigma \in \Sigma$, and then compose them in parallel. For scenario σ, we assume that $\mathbb{S} = \{S_0, \ldots, S_n\}, n \in \mathbb{N}$ and $\iota = S_0$. We also let the set $Post(S) = \{\, T \mid \mathbb{P}(S,T) > 0 \,\}$ for $S \in \mathbb{S}$.

Semantics of detectors.. For a better understanding, we distinguish input and output actions: if the synchronization actions are initiated by an MA, these actions are overlined by "$\overline{}$", and the synchronization actions waiting for the synchronization are denoted as usual. Note that this notation will not affect the original MA semantics.

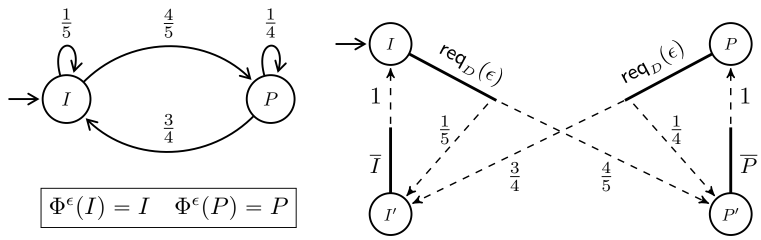
DEFINITION 10. **(Semantics of a scenario)** *The semantics of the* scenario module SM^σ *of detector* D *in scenario* σ *is defined by:*

$$SM(S:\mathbb{S}) := \mathsf{req}(\sigma) \sum_{T:Post(S)} \mathbb{P}(S,T) : SM'(T)$$

$$SM'(S:\mathbb{S}) := \overline{\mathsf{subsc}(\omega)}.SM(S) \ \ \textit{where } \omega = \Phi(S) \in \Omega$$

The task of SM is to simulate the DTMC embedded in scenario σ to return the sub-scenario decision which the detector D is going to execute. Since in the original SADF semantics the sub-scenario [Theelen 2007] is selected by both making a one-step transition in the DTMC and checking the function Φ, we use an intermediate state for every original state in the DTMC. To this end, we let $SM(S)$ represent the behavior of the original state and $SM'(S)$ represent the behavior of the intermediate state of S. First, if SM receives the sub-scenario decision request from function module FM in σ (i.e. action $\mathsf{req}(\sigma)$), and the last left state in the DTMC for σ is S, we make a one-step to the intermediate state of S's successor states (i.e. the intermediate states of $Post(S)$) with probabilities determined by $\mathbb{P}$. The behavior of the intermediate state just returns the sub-scenario decision using $\Phi(S)$, and then behaves like state $SM(S)$.

EXAMPLE 1. *A simple example shows how to translate the DTMC and sub-scenario decision function (left figure below) in eSADF to an MA (right).*



We now initialize the scenario module of D in scenario σ by setting $SM^\sigma = SM(S := S_0)$, where $S_0 = \iota^\sigma$ is the initial state of the DMTC of σ.

DEFINITION 11. **(Scenario module)** *The* scenario module *of detector D is:*

$$SM_D = SM^{\sigma_1} \parallel\!| \, SM^{\sigma_2} \parallel\!| \cdots \parallel\!| \, SM^{\sigma_n}$$

where $\parallel\!|$ equals $\|_\varnothing$ (as there is no synchronization between the MAs for the scenarios).

DEFINITION 12. **(Functional module)** *The* functional module FM *of a detector D is defined by:*

$$\begin{aligned}
&FM\,(dc : \mathbb{N},\, cc : \Sigma^*,\, subsc : \Omega_D) := \\
&\qquad (|cc| \geqslant 1 \wedge \mathtt{head}(cc) = \sigma \wedge subsc = \bot) \Rightarrow \overline{\mathsf{req}(\sigma)}.\sum_{\omega\in\Omega} \mathsf{subsc}(\omega).FM\,(dc,\, cc,\, subsc := \omega) \\
&+ \sum_{\omega\in\Omega_D} (subsc = \omega \wedge dc \geqslant R_D(\omega, i)) \\
&\qquad\qquad \Rightarrow (\lambda_D^\omega).\overline{\mathsf{exe}_D(\omega)}.FM(dc - R_D(\omega, i),\, \mathtt{tail}(cc),\, subsc := \bot) \\
&+ \sum_{\omega'\in\Omega_{D'}} \mathsf{exe}_{D'}(\omega').FM(dc + R_{D'}(\omega', oi),\, cc \mathbin{+\!\!+} t_{D'}(\omega', oc),\, subsc)
\end{aligned}$$

The tasks of FM are manifold. FM has three parameters: the number of tokens in data channel $dc_{(D',D)}$, the content of control channel $cc_{(D',D)}$, and the current operating sub-scenario $subsc$. One task of FM is to infer both the current scenario of D from the content of the first scenario token in each control channel and whether the subscenario is already available (i.e., equal to ω) or undefined ($\bot$). If the subscenario is undefined and the current scenario can be determined ($\mathtt{head}(cc) = \sigma$), FM will synchronize with SM to determine the operating sub-scenario in σ (by action $\overline{\mathsf{req}(\sigma)}$). After SM returns the sub-scenario, say ω, FM writes ω into variable $subsc$. After the sub-scenario is available (i.e. $subsc \neq \bot$), FM can execute as soon as there are enough data tokens in dc, which is checked by inspecting the rate function $R_D(\omega, i)$ for port i. If there are enough tokens, FM can execute, and the execution time is exponentially distributed with a mean duration of $1/\lambda^\omega$. After the execution, FM will synchronize with another process to update the corresponding channel status (i.e. process the tokens to its output channels by action $\overline{\mathsf{exe}_D(\omega)}$ action) and consumes the tokens from the input channels (i.e. updates its own variables' values). The last task of FM is to let other processes update their input channel status (i.e. to produce tokens onto the channels which are the input channels of D) after their executions.

We are now in a position to define the MA semantics of a detector D as the composition of its functional and scenario module.

DEFINITION 13. **(Detector semantics)** *The semantics of detector D is given by:*

$$\mathcal{M}_D := \tau_{I_D}(FM_D \parallel_{I_D} SM_D)$$

where $I_D = \{ \mathsf{req}(\sigma) \mid \sigma \in \Sigma \} \cup \{ \omega \mid \omega \in \Omega \}$ *and*

$$FM_D = FM(dc := \phi^*(DC),\, cc := \psi^*(CC),\, subsc := \bot)$$

where ϕ^*, ψ^* are the initial content of data channels and control channels in $\mathcal{C}^*$, respectively.

The action set I_D includes the sub-scenario request action req with all scenario values and the sub-scenario return action with all sub-scenario values. As the actions in I_D are only used for synchronizing FM_D and SM_D, all these actions are made unobservable by applying $\tau_{I_D}(\cdot)$.

Semantics of kernels.. The semantics of kernel K is obtained by simplifying the semantics of a detector D_K. First, the set of scenarios Σ_{D_k} and the set of sub-scenarios Ω_{D_K} of D_K are identical to K's set of scenarios Σ_K (i.e., $\Sigma_{D_K} = \Omega_{D_K} = \Sigma_K$). Similarly the consumption/production rate function R of D_K equals R of K. AsK has no control outports (so does D_K), the function t and rate function R are not defined in D_K. The scenario module SM of D_K in scenario ω is defined as described in Remark 1 by:

$$\begin{aligned} SM(S_0) &:= \ \mathsf{req}(\omega).SM'(S_0) \\ SM'(S_0) &:= \overline{\mathsf{subsc}(\omega)}.SM(S_0) \end{aligned}$$

Semantics of eSADF graphs.. Finally, we define the MA semantics for an eSADF graph. We assume that an eSADF graph consists of a set of detectors $\{D_1, \ldots, D_n\}, n \in \mathbb{N}$. For each detector D_i ($1 \leqslant i \leqslant n$) let MA $\mathcal{M}_{D_i}$ be its semantics and Act_{D_i} the set of interactive actions in D_i.

DEFINITION 14. **(eSADF semantics)** *The MA for eSADF graph $\mathcal{G} = (\mathcal{P}, \mathcal{C}^*)$ is:*

$$\mathcal{M}_{\mathcal{G}} := \tau_H\big(\mathcal{M}_{D_1} \parallel_{I_1} \cdots \parallel_{I_{n-1}} \mathcal{M}_{D_n}\big)$$

where $\parallel$ is left-associative, $I_i = Act_{D_{i+1}} \cap (Act_{D_1} \cup \cdots \cup Act_{D_i})$ for $1 \leqslant i \leqslant n$, and $H = Act_{D_1} \cup \cdots \cup Act_{D_n}$.

EXAMPLE 2. *The eSADF graph in Figure 2 (left) consists of detector A and kernel B, control channel $CC_{(A,B)}$ and data channels $DC_{(B,B)}$ and $DC_{(B,A)}$. Production and consumption rates equal to 1 are omitted, and the red numbered points indicate the number of initial tokens in these channels (control channels are initially empty). Kernel B can execute in scenarios I and P. The execution time of I is exponentially distributed with mean duration one; P has mean duration $\frac{1}{2}$. The scenario occurrence is decided by A based on the embedded* **DTMC** *(cf. Example 1) and sent to B via the scenario tokens valued with I and P through channel $CC_{(A,B)}$. Since there is no input control channel for A, A always executes in a default scenario ϵ. Here we assume the sub-scenario decision procedure in A will be done immediately.*

5. QUANTITATIVE ANALYSIS

Now that we have seen that the behaviours of an eSADF graph and its actors can be adequately described by means of a Markov automaton (MA), we address how quantitative measures on eSADF graphs, e.g., expected time and long-run objectives, as well as probabilities of certain events happening unto a certain deadline, can be obtained. A detailed treatment of the algorithms is outside the scope of this paper; a full explanation can be found in [Guck et al. 2014a].

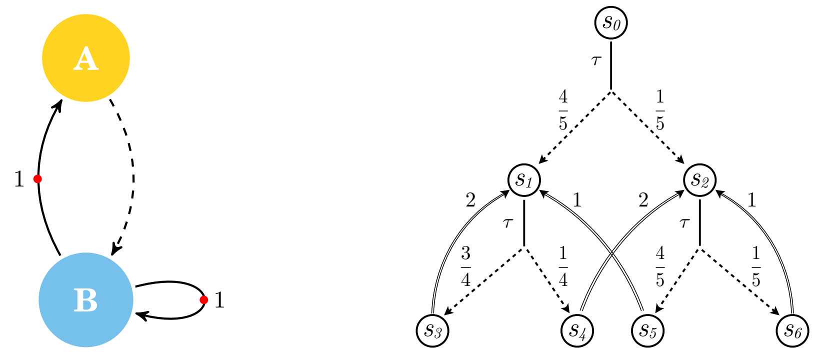


Fig. 2. A sample eSADF graph and its (somewhat simplified) MA

5.1. Analysis of Markov Automata

Given a finite-state MA, we consider three quantitative objectives: expected time, long-run average, and timed (interval) reachability. *Expected time* objectives focus on determining the minimal (or maximal) expected time to reach a given set of states. *Long-run* objectives determine the fraction of time to be in a set of states when considering an infinite time horizon. *Timed reachability* objectives are focused on the probability to reach a set of states within a given time interval. As MA exhibit non-determinism, we focus on maximal and minimal values for all three objectives. These correspond to the best and worst possible resolution of the non-determinism present in the MA. Expected-time and long-run average objectives can be efficiently reduced to well-known problems on Markov decision processes such as stochastic shortest path, maximal end-component decomposition, and long-run ratio objectives. As shown in [Guck et al. 2014a], the reduction to these well-investigated problems enables the usage of efficient analysis techniques such as linear programming, value iteration, and maximal end-component decomposition, which all have a polynomial time-complexity in the size of the MA. This all relies on the fact that for optimal expected time or long-run objectives, the resolution of the non-determinism by extremely simple (so-called memoryless) policies suffices. Timed reachability objectives however are harder to obtain. The main technical complication here is that the optimal way of resolving the non-determinism requires policies with infinite—even uncountably large—memory. Intuitively this can be seen as follows. Assume that in an MA there is a non-deterministic choice between two options: either reaching a target state slowly, but almost surely (i.e., with probability one), or reaching a target soon, but with the risk that the target is not reached at all (so the target is reach with a probability strictly smaller than one). Then it makes a difference whether ample time remains to reach the target, or whether there is almost no time left. In the latter case, the fast but unsafe option is optimal, whereas in the first case the slow and safe option is optimal. The way around this is to resort to *discretisation*. Here, the time interval is split into equally-sized discretisation steps, each of length δ. The discretisation step is assumed to be small enough such that with high probability it carries at most one Markovian transition. This yields a discretised MA, a variant of a semi-MDP, obtained by summarising the behaviour of the MA at equidistant time points. The analysis of the dMA yields an approximation of the true timed reachability probability in the MA where the error is bounded (depending on the discretisation step δ and the largest rate occurring in the MA). As we will see in the case studies (Section 6), the analysis of timed reachability objectives therefore is time-consuming.

5.2. Confluence Reduction and Non-Determinism

Prior to analysing an MA, we employ a state-space reduction. This is an on-the-fly technique, meaning that it can be used while generating the MA from a given eSADF graph. In addition, it is a symbolic technique, that is, it is directly applicable on the MAPA terms that describe the behaviour of an eSADF graph. Besides being an effective and efficient state-space reduction technique, confluence reduction—in theory—can reduce *all* non-determinism in the MA semantics of an eSADF graph.

Confluence reduction. Confluence reduction [Timmer et al. 2012] reduces the state space based on commutativity of transitions, removing nondeterministic transitions caused by parallel composition of independent components. It is similar in spirit as partial-order reduction. The reduction preserves the three quantitative metrics of interest described above. Based on heuristics to detect confluence in MAPA terms, the state space is reduced in an *on-the-fly* manner. Its effect is illustrated in Figure 3, which gives the MA semantics (b) for a simple stochastic Petri net with immediate transitions (solid bars) and timed transitions (open bars) (a) and afterwards reduces the state space (c) by applying confluence reduction. The key observation is that the commutativity of the immediate transitions t_1 and t_4, and t_2 and t_4 is exploited in the reduction. Confluence reduction yields a reduction of 7 to 4 states.

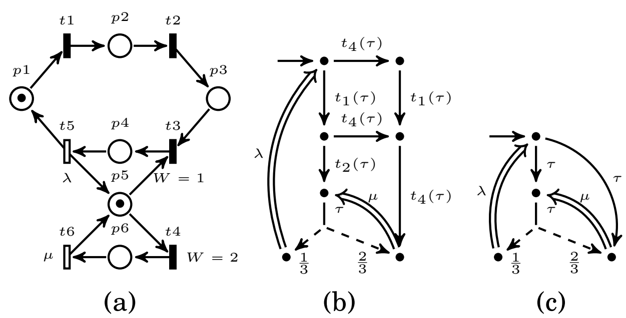


Fig. 3. (a) a sample GSPN, (b) its MA semantics, and (c) its reduced MA

Confluence reduction in a nutshell. The basic idea of confluence reduction is to determine the confluent sets of transitions [Timmer et al. 2012]. To obtain these, groups consisting of only confluent interactive probabilistic transitions should satisfy the following conditions: 1) all transitions are τ-transitions with Dirac distribution, 2) all transitions enabled prior to a transition in this group are still enabled after taking such transition. The diagram left above illustrates the latter constraint.

$$\begin{tikzcd} s \arrow[r, "\tau", "T"'] \arrow[d, "a"', "C"] & t \arrow[d, "a"', "C"] \\ \mu \arrow[r, phantom, "\equiv_R"] & \nu \end{tikzcd}$$

If transition $s \xrightarrow{\tau} t$ is in a group, say T, and if $s \xrightarrow{a} \mu$, then $t \xrightarrow{a} \nu$ must exist such that μ and ν are related, i.e., all states in the support of μ and ν are connected by transitions from T. Timmer *et al.* proved that the transitions satisfying the conditions above connect *divergence-sensitive branching bisimilar* states [Timmer et al. 2013]. Hence it is safe to prioritise confluent transitions. As the intermediate states on a confluent path are bisimilar, they can be aggregated. Confluence reduction is applied on syntactic MAPA terms in an on-the-fly manner thus avoiding a full state space generation prior to the reduction (as opposed to bisimulation reduction [Theelen et al. 2012] which requires the construction of a full state space prior to reduction). Case studies show a state space reduction from 26% to 83% [Timmer et al. 2012]; for the MPEG-4 decoder and the face recognition case study this is about 66% (cf. Table I).

Non-determinism in eSADF. Non-determinism in our MA semantics only arises from the execution of independent concurrent actors in eSADF graphs:

THEOREM 1. *Non-determinism only occurs between sub-scenario decision actions from different, independent processes. All these transitions are confluent, i.e., they yield the same (Markovian) state. The probability distribution to reach such states is independent from the resolution of the non-determinism.*

The proof sketch of this result is provided in [Katoen and Wu 2014] and confirms a similar result in [Theelen et al. 2006] for an alternative SADF semantics. The key point is that, since the enabled probabilistic choice among sub-scenarios is instantaneous, the transitions representing the time progress can not be enabled before all such enabled probabilistic transitions[4]. Since these enabled probabilistic transitions are independent, they are confluent to the state where the Markovian transitions are enabled. Whereas in SADF [Theelen et al. 2006] timed transitions are deterministic (since it always takes the earliest finished execution time of a process), in eSADF they are probabilistic and resolved by the race condition. Thanks to the above result, confluence reduction can potentially reduce all non-determinism from the MA semantics of an eSADF graph. As heuristics are used, this is not always established in practice, but then the above result guarantees that worst and best case quantities coincide.

6. CASE STUDIES

In this section, we provide two case studies: an eSADF of the MPEG-4 decoder benchmark and an eSDF[5] of a real-life face recognition application (that we obtained from an industrial partner). Different system metrics, such as throughput of different kernels (actors), channel buffer occupancies and response delay are obtained from the quantitative analysis of the resulting MA.

6.1. The MPEG-4 Decoder Benchmark

We consider the eSADF graph of an MPEG4-decoder as given in Figure 1 (page 4). Applying confluence reduction to the MA of the MPEG-4 decoder reduces the state space by about a factor 3:

Table I. MA size for sample eSADF graphs

		before red.	with conf. red
Example 2	#states	19	7
	#transitions	19	7
MPEG-4	#states	61918	20992
	#transitions	81847	40910
	#non-det. state	4	1
Face recognition	#states	106784	29440
	#transitions	154688	77344

Note that non-determinism for the MPEG-4 decoder is not completely eliminated by confluence reduction, as heuristics are used to detect confluent transitions.

Our MA semantics allows for determining several performance metrics for the MPEG-4 decoder in a fully automated manner. We illustrate this for various quantitative measures of the MPEG-4 decoder example.

[4]In SADF this is ensured by action-urgency; here by maximal progress.
[5]Since SDF is a subclass of SADF, our eSADF semantics is readily applicable to eSDF.

Buffer occupancy. We consider two long-run properties: *the average number of tokens and the probability distribution of tokens in a channel.* The average buffer occupancy of

Table II. Average buffer occupancy in each MPEG-4 channel

	p1	p2	p3	p4	p5
av. #tokens	idct-rc	vld-mc	vld-idct	mc-rc	rc-mc
MaMa	38.2584	27.7168	1.3118	0.2071	0.7929
SDF^3	42.2215	24.2215	1.18198	0.4412	0.8574

each channel of the MPEG-4 decoder is shown in Table II. (This includes a comparison with the results obtained for the MPEG-4 decoder with the SDF^3 tool, using deterministic execution times.) From the results, we observe that the channels of IDCT-RC and VLD-MC have a much higher average occupancy than the other three. This is due to the fact that in case of the I frame and P_x frames, IDCT and VLD need to execute only one time, whereas RC and MC need to compute 99 and x times, respectively. Hence the tokens can accumulate in both channels waiting for processing. The average number of tokens in channels MC-RC and RC-MC together is one. This is due to the cyclic channels between MC and RC, which guarantees that MC and RC execute at the same rate. [6]

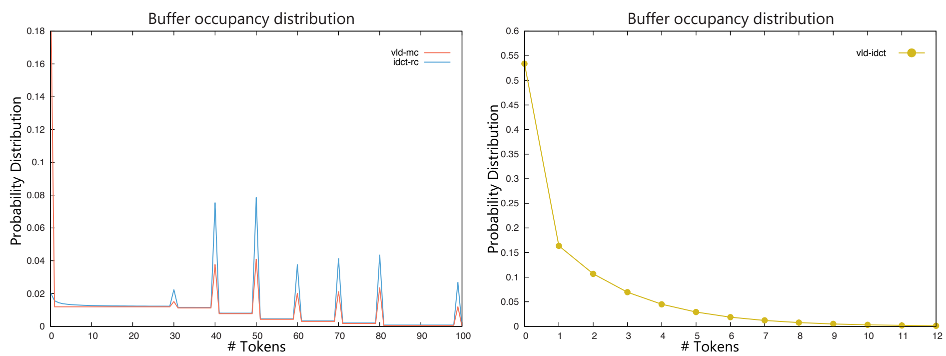


Fig. 4. Token distribution in three MPEG-4 channels

The token distributions of three data channels are shown in Figure 4. The peak values for channels VLD-MC and IDCT-RC are caused by conservatively approximating the motion vectors (cf. Section 6.1) by fixed numbers (i.e, $x \in \{30, 40, 50, 60, 70, 80, 99\}$). Further, the average number of tokens in channel VLD-IDCT is 0.57, which is possibly due to that VLD will produce at most one token to the channel VLD-IDCT at each time. Thus, the probability of having more than three tokens in channel VLD-IDCT is very low, whereas the probability of VLD-IDCT being empty is quite high ($\geqslant 0.65$).

Expected time. The second property considered is *the expected time and time-bounded reachability probability for a channel to reach its 50% (p6) and 90% capacity*, respectively. In our evaluation, we let the maximal number of tokens in the channel to be the capacity of that channel. Afterwards, we mark such states as target states where the current number of tokens in the channel is more than 50% and 90% of its capacity, respectively. Here, we only the show the result in 50% case of channel VLD-MC in Figure 5.

[6]Note that this does not hold for SDF^3 which hints at a possible flaw in this tool.

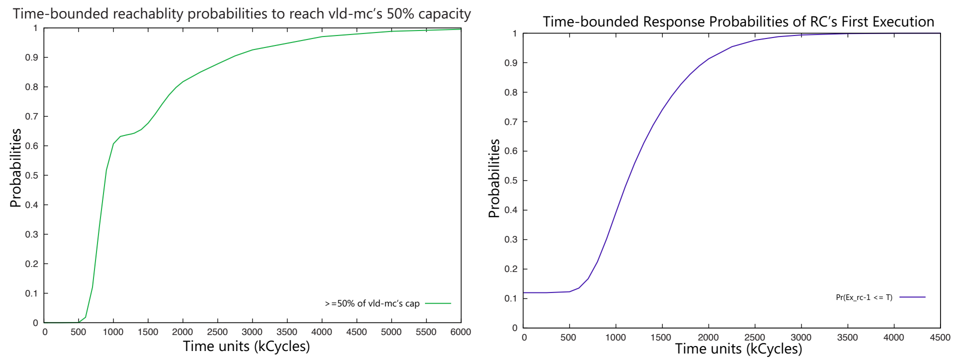


Fig. 5. Time-bounded reachability probabilities to reach VLD-MC's 50% capacity (left) / First response probabilities of RC within time t (right)

Response delay. We exploit time-bounded reachability objectives for *response time estimation*. This allows for answering questions such as "what is the expected time until a process finishes its first execution?" or "what is the probability of a process responses for the first time within time t?". It is equal to compute the expected time or time-bounded reachability probabilities from the initial state to those states where the process has just finished its first execution. Taking RC for example, we get 1152.617 (kCycle) as the answer to the first question and Figure 5 to the second (p7). Since there is probability 0.12 to have a P_0 frame which means a still video frame, RC just copies it from MC, the probability of RC finishes its first execution within time 0 is 0.12.

Throughput and inter-firing latency. We compute *the throughput* of a kernel by the following approach. First, we compute the long-run average probability (P_σ) of a kernel executing in scenario $\sigma \in \Sigma$. This can be done by adding a Boolean variable to the kernel's MAPA definition and set the Boolean to true when the execution condition is satisfied and set it to false when the execution finishes. Since the expected execution time (E_σ) of a kernel in scenario σ is known, the throughput of this kernel is computed as sum of the long-run average probability P_σ divided by the expected time E_σ for each scenario σ:

$$Tr = \sum_{\sigma\in\Sigma} \big(\lim_{t\to\infty} \frac{P_\sigma \cdot t}{E_\sigma} \cdot \frac{1}{t} \big) = \sum_{\sigma\in\Sigma} \frac{P_\sigma}{E_\sigma} = \sum_{\sigma\in\Sigma} \lambda_\sigma P_\sigma \ .$$

The results for the MPEG-4 decoder are shown in Table III.

Table III. Throughput of each kernel in MPEG-4 decoder

	p8	p9	p10	p11
Throughput	**IDCT**	**VLD**	**MC**	**RC**
MaMa	0.0423732	0.0423732	0.000746128	0.000746128
SDF3	0.0437919	0.0437919	0.000745268	0.000745268

Analogously, the average inter-firing delay (In) can be computed as:

$$In = \sum_{\sigma\in\Sigma} \big(\frac{(1-P_\sigma)}{\sum_{\sigma\in\Sigma} P_\sigma \cdot \lambda_\sigma} \cdot \frac{P_\sigma}{\sum_{\sigma\in\Sigma} P_\sigma} \big) \ .$$

We take MC as an example and compute the In of MC as 1460.5 (1341.8 in SDF^3) kCycles. The verification times to check the various properties is indicated in Table IV. The numbers in brackets are the run times of SDF^3 (where n.a. stands for not applicable).

All experiments have been conducted on a machine with an 48-core CPU at 2.1GHz and 192GB memory.

Table IV. Verification time of properties [8]

p1	p2	p3	p4	p5	p6
489m (6s)	592m (3.4s)	141m (11.4s)	11m (2.7s)	10.4m (12.7s)	14.7h (n.a)
p7	p8	p9	p10	p11	
456m (n.a)	24.87m (4.76s)	18.48m (6.56s)	4.45m (13.02s)	5.1m (3.2s)	

6.2. Face Recognition

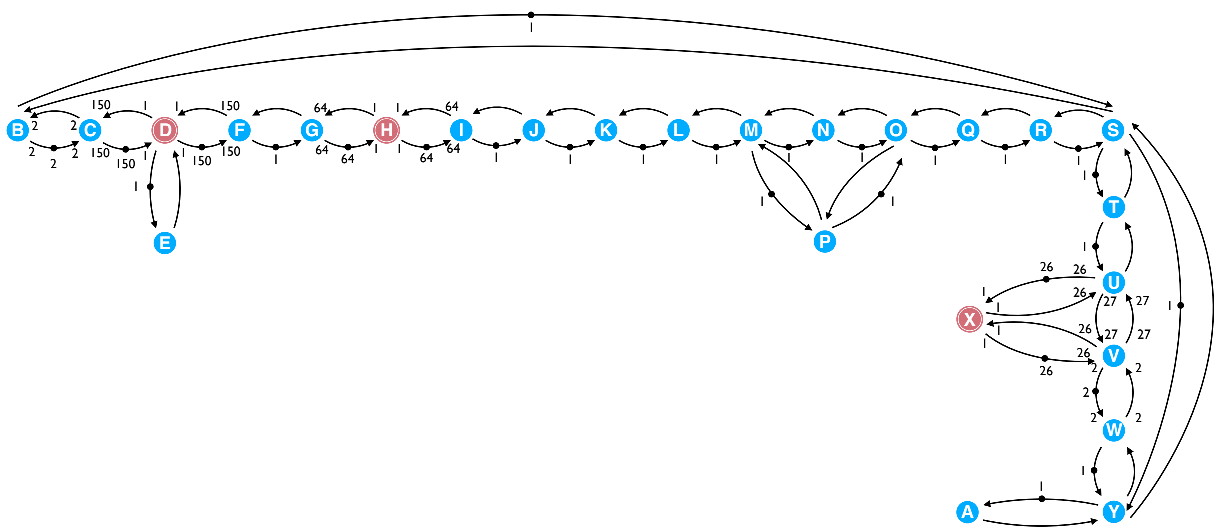


Fig. 6. The SDF model for face recognition application

The face recognition example. In this case study, we apply our approach to an SDF model (see Figure 6) of a real-life application of face recognition. The SDF models the processing of a single frame in a two-dimensional picture. Due to confidentiality issues, we have anonymised all actor names; the rest of the graph is as provided to us by an industrial partner. The SDF consists of 25 kernels. For the sake of readability, production/consumption rates equal to 1 are omitted; furthermore, the initial number of tokens in a channel is marked by a black dot attached a number. Note that an SDF graph can be considered as an SADF graph without detector and every kernel executes in a default scenario (no sub-scenarios and no probabilistic selection of such sub-scenarios). We can thus apply our approach without any problems to SDF graphs. There is however one difference: SADF does not allow auto-concurrency whereas SDF does. In Figure 6, the actors in which auto-concurrency is disallowed (hence we omitted the self-loop with rate 1 in such actors for simplicity) are marked as blue, while the

[8]These verification times seem prohibitive, but exploit algorithms that allow for analysing MA with non-determinism.

actors (i.e., actor D, H, and X) in which auto-concurrency is allowed are marked red. Auto-concurrency amounts to the simultaneous firing of an actor (provided the firing condition is multiply enabled). Auto-concurrency can be expressed in our MA semantics as the parallel composition of multiple *enabled* copies of the actor process. Due to the fact that a process with multiple (say n) Markovian delays λ (representing the firings) enabled in parallel is equivalent—in the sense of branching bisimulation—to a process which evolves with rate $n{\cdot}\lambda$ to its successor, and then evolves subsequently with rate $(n{-}1){\cdot}\lambda$ and so forth, see Figure 7. Thus, the execution rate in the original MA definition of such actor with auto-concurrency becomes multiple times (= number of enabled copies of the actor) the original rate. This allows us to quantitatively assess the impact of auto-concurrency on the face recognition example in Figure 6 In our ex-

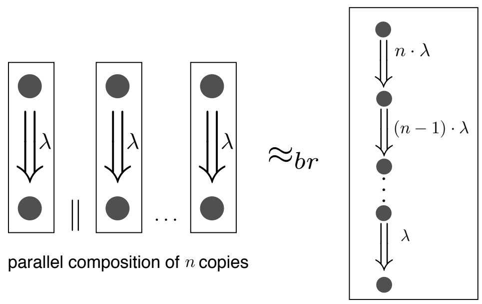


Fig. 7. Treating auto-concurrency in the MA semantics of SDF

periments, we evaluate the SDF without auto-concurrency for actors D, H, and X, and provide a quantitative comparison between the two versions. The state space of the MA is about 1,000,000 states (see Table I) and can be reduced by confluence reduction by a factor 66%. The computation time of the throughput of each actor varies from 3 to 10 minutes, and the computation of the token distribution for each case takes about 1 to 2 minutes. As indicated before, the most time-consuming property to compute is the probability of reaching a (critical) situation within a given time-bound, which varies from minutes to a few hours depending on the given time-bounds.

Throughput. We first compute the throughput of each actor in the SDF graph. We observe that all the actors except the "auto-concurrent" actors D, H, X and E have the same throughput. This throughput is the base throughput of the SDF graph. It follows that the throughput of D, E is 150 times, H 64 times, and X 26 times the base throughput, respectively. On the other hand, the SDF's throughput is increased by a factor 1.5 when auto-concurrency is exploited. Thus, auto-concurrency increases the throughput by about 50%. The results are listed in Table. V. Note that since in SDF the actors have only one default scenario, the inter-firing latency of an actor is simply the reciprocal of its throughput.

Table V. Throughput of each actor in the face recognition application

Throughput	A–C/F/G/I–W/Y	D & E	H	X
Auto–Concurrency	0.1226	18.390	7.8469	3.187
No–Auto–Concurrency	0.08805	13.20900	5.6361	2.289

Table VI. Average tokens and token distribution in different channels

Average Buffer Occupancy	c-d	d-f	d-e	g-h	h-i	v-x	u-x
Auto-Concurrency	82.16925	82.05888	0.09196	62.35874	63.57629	0.06694	25.57622
No-Auto-Concurrency	101.28171	101.20245	0.06605	54.37433	54.10945	0.64911	25.23185

c-d/d-f

Token distribution	0	1-149	150
Auto-Concurrency	0.00466	0.00601	0.10025
No-Auto-Concurrency	0.00335	0.00431	0.35377

d-e

Token distribution	0	1
Auto-Concurrency	0.908041	0.09196
No-Auto-Concurrency	0.93395	0.066048

Buffer occupancy. As for the MPEG-4 decoder, we determine the average number of tokens and the distribution of tokens in different representative channels. The outcomes are shown in Table VI and VII. Observe that channel `c-d` and `d-f` have almost the same average number of tokens, and the probability of having one token in channel `d-e` is much higher than for `e-d` (one token is either in `e-d` or `d-e`). This is due to the heavy workload at actor `D`. Moreover, we observe that without auto-concurrency, more tokens will accumulate in some channels in average, but this does not always occur. For some channels, such as `u-x`, there is almost no impact on the average number of tokens. The distribution of different number of tokens in channel `c-d`, `d-f`, `d-e`, `v-x` and `d-e` is also given. We see that the probability distribution is monotonically decreasing or increasing, and the large probability (such as 0.99 in `v-x` and 0.969 in `h-i`) is at marginal values (0 or 64).

Table VII. Token distribution in channels v-x & h-i when using auto-concurrency

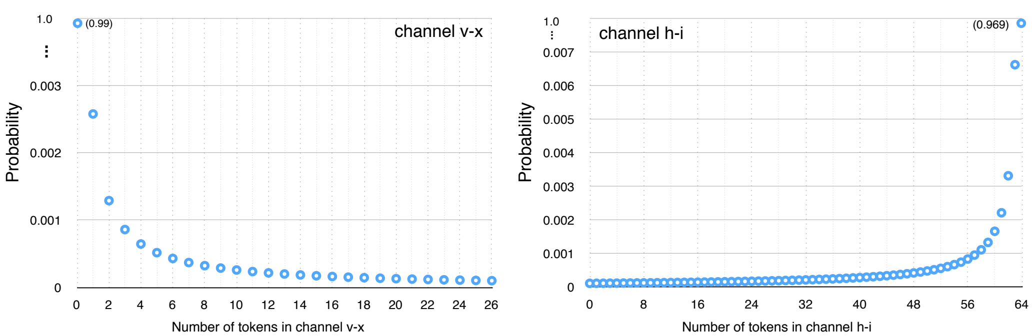


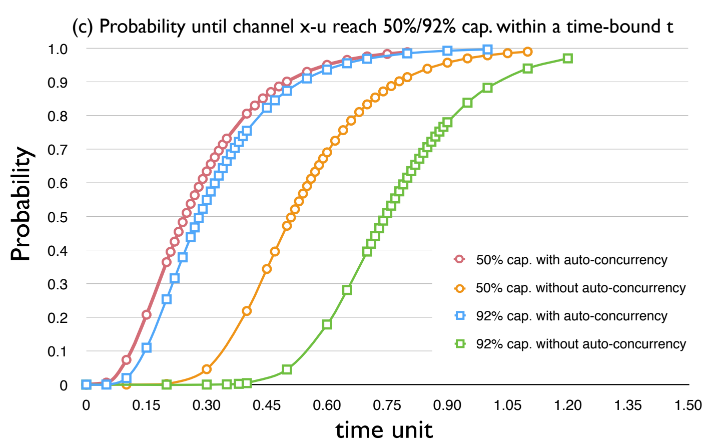
Table VIII. The expected time (a), time-bounded reachability to critical situation (b), and response delay of actor `B` (c)

(a)

Expected Time (x-u)	50% cap.	92% cap.
Auto-Concurrency	0.54090	0.77189
No-Auto-Concurrency	0.28206	0.31735

(b)

Response Delay (actor B)	
Auto-Concurrency	8.5836532
No-Auto-Concurrency	12.248275



Expected time. In this part, we consider the expected time and time-bounded reachability to certain critical situations, such as 50% or 90% channel's capacity is reached. We can see from the Figure VIII (c) that in absence of auto-concurrency a large difference between the probability of reaching 50% and 90% within a given time-bound is obtained. If auto-concurrency is used, the curves of reaching 50% and 92% channel capacity are tight together, which means that if a channel reaches 50% capacity, it will reach 92% of its capacity soon thereafter. If no auto-concurrency used, the period between these two situations becomes significantly larger.

Response delay. In the property of response delay, we consider the first response until the actor B, which is the most significant actor in the SDF, since it is the "last" actor in the SDF to finish one cycle of all the tasks of actors. Again, from the Table. VIII (b) we can conclude that auto-concurrency improves the response delay by about 30%.

7. EXTENSION OF ESADF WITH HARDWARE PLATFORM

In this section, we will introduce our extended model of eSADF by incorporating the hardware platform. Adopting eSADF as model of computation (MoC) as before, we extend our system to additional models: the model of architecture (MoA), the model of computation and communication (MoCC), and the model of performance (MoP). Our approach is illustrated in Figure 8, and details are provided below.

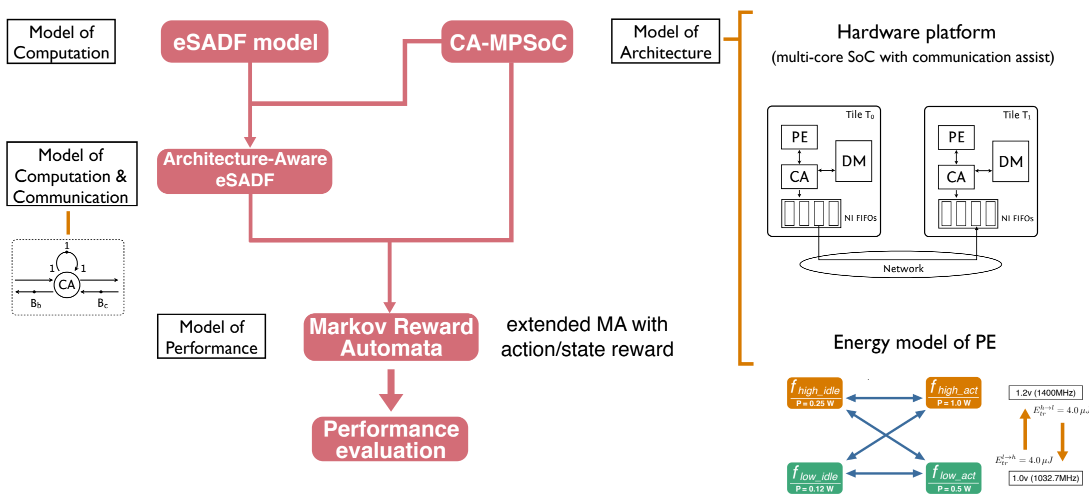


Fig. 8. An overview of our approach extended to hardware platform

7.1. The Hardware Platform (MoA)

Since eSADF abstracts the software applications as MoC, we need a predicable model of architecture (MoA) which is suit able for modeling the hardware platform. There are various approaches [Castrillon and Leupers 2013]; we adopt the approach of CA-MPSoC introduced in [Shabbir et al. 2010a; Shabbir et al. 2010b] due to:

- The concept of communication assist (CA) clearly and concisely decouples the *computation* and *communication* tasks, and analysis can be performed easily and more predictable;

- The CA for a tile-based MPSoC has been implemented (hardware coded) and shown to be up-to 44% smaller compared to existing DMA controllers [Shabbir et al. 2010a; Shabbir et al. 2010b];
- An eSADF model for the CA can be easily integrated/extended into the SADF model for the application, and hence an intermediate *architecture aware* CA-based SADF can be derived for further use.

The CA-MPSoC proposed in [Shabbir et al. 2010a; Shabbir et al. 2010b] is illustrated in Figure 9. We briefly explain the CA-MPSoC; for implementation details of such platform please refer to [Shabbir et al. 2010a; Shabbir et al. 2010b]. The CA-MPSoC

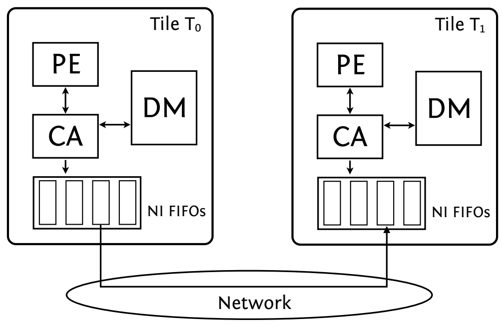


Fig. 9. The CA-based hardware platform

consists of tiles which are connected by a network. Each tile (e.g tile T_0) contains a processing element (PE), a communication assist (CA), data memory (DM), and network interface (NI) FIFOs. These NI FIFOs are connected by the partial *point-to-point* network, which does not require any storage and provides a directly data transfer from the DM of one tile to anther.

Processing element and data memory. The PE executes the tasks (i.e., the actors) which are deployed to it. Following the ideas of [Shabbir et al. 2010a; Shabbir et al. 2010b], we assume PEs to have no caches and being non-preemptive. Furthermore, in order to predicate the energy consumption of the synthesized system, the PE is assumed to provide energy optimisation strategies, such as dynamic voltage frequency scaling (DVFS) and/or dynamic power management (DPM). The DM is dual-ported, and directly connected to CA to avoid unpredictable arbiter.

Communication assist. The CA is the interface to transfer data between tiles. The benefit of CA is that it decouples the communication from computation which eases our performance analysis. Let us explain a typical scenario to show how the CA works. We assume that tile T_0 runs the producer actor and tile T_1 runs the consuming actor. First the PE of T_0 asks the CA for space in DM and CA will allocate the space for PE. After the execution, T_0 releases the space and CA will transfer the produced data via NI FIFO point-to-point through the network. The CA of T_1 receives the data and places it in the memory (DM) of T_1. The PE of T_1 which is executing as the consumer actor asks the CA for the availability of data and processes the data which is returned by the CA. Since the real hardware implementation is out of the scope of our work, we omit this part here and refer to the original work.

The energy model of MoA. In order to predicate the energy consumption of hardware platforms on which the eSADF graph is deployed, we define the *energy model* of our MoA. Due to the separation of computation and communication tasks, we can naturally divide the energy model into two parts: the computation energy and communication energy consumption. Obviously, we consider the computation energy consumption happens mainly in the PE, while the communication energy consumption takes mainly place in the CA.

The energy model of PE. In our model, we consider a modern PE equipped with DVFS and/or DPM to reduce both dynamic and static power consumption. DVFS reduces the energy consumption of PE by dynamic scaling the voltage and the frequency of PE. We consider in our model the inter-task DVFS, which allows the frequency to be changed only after a task is finished on this PE. We does not take the intra-task DVFS into account due to its complexity. DPM reduces the energy consumption by allowing the processor to get into sleep mode after a certain idle period. The sleep mode has a much lower static power consumption than the idle mode. Note that the *break-even* time is the key factor in DPM, since only the idle period is longer than such break-even time, the system will get energy benefit.

7.2. Architecture-aware eSADF - MoCC

In this section, we add the CA to our MoC which results an *architecture-aware* eSADF model. This model does not only model the computation but also the communication of applications (hence, we also call it the model of computation and communication (MoCC)). This intermediate model will be translated together with MoA into our model of performance (MoP) for further analysis.

The eSADF model of CA. As in [Shabbir et al. 2010a], a CA[9] can be represented by an (partial) actor in an eSADF shown in Figure 10. The self-loop channel with one initial token guarantees the firing ordering of the actor. Note that, since CA lies between the DM and NI FIFOs, each channel between two actors in the eSADF of MoC requires a CA actor. The rate of CA (except for the self-loop channel) is always one word, since each firing of CA will transfer one word[10] from DM to NI-FIFOs, if it is a sending CA, or one word from NI-FIFO to DM, if it is a receiving CA. The left-hand side of the CA represents the data transfer process between the DM and CA, where the initial token number B_b is the buffer size requested in DM by the PE. The right-hand side represents the data transfer process between the CA and NI-FIFOs, where the B_c is the depth of the NI-FIFOs.

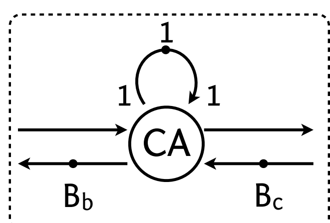


Fig. 10. The eSADF actor of CA

[9]This CA represents a sending CA, where the direction of channels will reverse, if it is a receiving CA.
[10]E.g., one word equals 4 pixels in an encoder application.

The execution time of CA. For each channel, the CA needs 2 cycles to finish the following two tasks, respectively. First, CA checks whether there is data to be transferred and then performs the transfer. Since the CA asks the channels in a round robin fashion, the execution time of CA is computed by:

$$t_{ca} = 2 \times \#CH\,.$$

Where the $\#CH$ denotes the number of channels.

7.3. Markov reward automata - MoP

In this section, we introduce the semantic model, Markov reward automata (MRA) [Guck et al. 2014b], which extends Markov automata (MA) [Eisentraut et al. 2010; Deng and Hennessy 2013] with state rewards and transition rewards, as the model of performance (MoP) in our work. Briefly speaking, an MRA is an extended labeled transition system (LTS):

— equipped with both continuous time stochastic and nondeterministic transitions,
— for each state a cost/gain reward is assigned, which indicates the cost/gain per time unit, if the system stay in that sate,
— for each transition a transition reward is assigned in order to indicate the cost/gain if this transition is taken.

DEFINITION 15. **(Markov reward automata)** *A* Markov reward automaton (MRA) *is a tuple* $\mathcal{M} = (S, s_0, Act, \hookrightarrow, \Rightarrow, \rho, \eta)$*, where* $(S, s_0, Act, \hookrightarrow, \Rightarrow)$ *is an MA, and*

— $\rho : S \to \mathbb{R}_{\geqslant 0}$ *is the* state-reward *function,*
— $\eta : S \times Act \cup \{\chi\} \times \mathrm{Distr}(S) \to \mathbb{R}_{\geqslant 0}$ *is the* transition-reward *function.*

Note that, since we also need to define the transition reward on outgoing Markovian transitions, a "equivalent" rewriting function γ is defined in order to translate the outgoing Markovian rates of a state to be an action followed by a distribution (similar to interactive probabilistic transition). The γ function takes a Markovian state and attaches it with an action $\chi(E(s))$ together with the exit rate $E(s)$ in it and then followed by a distribution which is identical to $\mathbb{P}_s$. An example of rewriting function γ over the outgoing Markovian transitions of a state is shown in Figure 11.

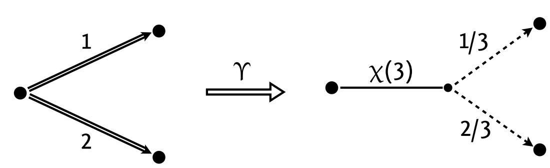


Fig. 11. The γ function in MRA

7.4. Case study - MPEG-4 decoder with hardware platform

To exhibit our approach proposed above, we conduct a case study on the MPEG-4 decoder introduced earlier together with a hardware model of MPSoC based on Samsung Exynos 4210. Since we are mostly interested in the energy consumption, we make some simplifications in our model: 1) comparing with the (average) execution time of actors in MPEG-4 which is about 10 to 400 kCycle and CA takes only 2 to 10 cycles, we ignore the communication part (i.e. the CA actors) in our experiment, 2) when modelling of DVFS, we assume only two voltage-frequency levels, i.e., the lowest voltage of 1V with 1032.7 MHz and the highest voltage of 1.2V with 1400 MHz, 3) the DPM mechanism is omitted, since Samsung Exynos 4210 does not support this feature. Note that, if the communication and DPM need to be considered, we are able to add the CA/DPM into the system but with more effort as mentioned previously. In our platform, we assume there are only 2 processors, and each processor can run in different frequency independently. The energy model is shown in Figure 12, and the energy consumption of the hardware platform of Samsung Exynos 4210 is computed by utilising the following formula given in [Park et al. 2013] as:

$$P_{\text{cpu}} = 0.446\,\nu_{\text{proc}} V_{\text{cpu}}^2 f_{\text{cpu}} + 0.1793\,V_{\text{cpu}} - 0.1527$$

where ν_{proc} is the total utilisation and we set in idle state $\nu_{\text{proc}} = 0.2$ and in running state $\nu_{\text{proc}} = 1$.

Each processor's behaviour is modelled by an MRA, in which the energy consumption is modelled by the state reward, and the energy consumed by downscaling/upscaling in DVFS is modelled by the transition reward. Each actor in the eSADF of MPEG-4 decoder can be deployed to the processors by using synchronisation. Note that, we assume a fully dynamic scheduling rather than a static one in our model. We also restrict the scenarios in the MPEG-4 decoder to only I, P_0 and P_{50}-frames.

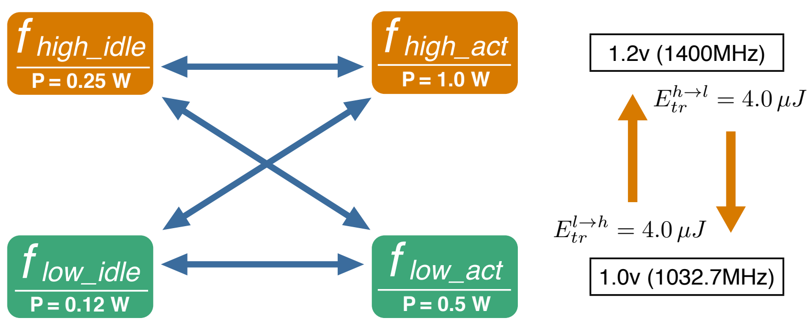


Fig. 12. The energy model of PE based on Samsung Exyons 4210

The experimental results are summarized in Table IX. As before, we apply confluence reduction. First, we evaluate the maximal and minimal power consumption by computing the max/min long-run reward in the resulting MA. Then, we compute the maximal and minimal throughput of actors IDCT and RC (note that, VLD has the same throughput as IDCT, and MC has the same throughput as RC). The throughput here is the number of firing times in one *ms* of such actor. From the result that the difference between the minimal and maximal throughputs of RC is much smaller than the difference between the maximal and minimal energy consumption, there should exist a balanced scheduler which can keep the energy consumption reasonably low, while the throughput is still acceptable. The last property we compute is the energy consumed when the MPEG-4 decoder finishes its first iteration. This is computed by the expected cumulative reward from the initial state to the goal states.

Table IX. The experimental results of MPEG-4 decoder with hardware platform

#states	#transitions		Power cons.	Thr. IDCT	Thr. RC	Exp. energy 1 iter.
215851	314609	Min	0.746337 (W)	21.975093	0.415335	1.581342 (*mJ*)
		Max	1.528917 (W)	30.279699	0.563065	2.423743 (*mJ*)

8. RELATED WORK

SADF semantics. Whereas we consider exponentially timed SADF—akin to exponentially timed SDF [Sriram and Bhattacharyya 2009]—and use Markov automata as semantic model, the original works on SADF focus on a *real-time* semantics using so-called Timed Probabilistic Systems (TPS) [Theelen et al. 2008; Theelen 2007; Theelen et al. 2006]. TPS have deterministic delays, and discrete probabilistic branching. A direct comparison of analysis results is thus not possible. The compositional nature of our semantics together with the memoryless property of exponential distributions yields a simple and lean semantics. In contrast, the TPS semantics has to account for actors that are enabled at the same time; this occurs in our framework with probability zero. In addition, the probability measure of Zeno paths for MA obtained from eSADF graphs is zero. This avoids a special (an typically intricate) treatment so as to exclude Zeno paths from the analysis. The simplicity of our semantics allows for considering kernels as simplified detectors, and providing a relatively straightforward formal proof (sketch) of the absence of non-determinism—confirming the result in [Theelen et al. 2006] for their TPS semantics. Finally, confluence reduction allows for an on-the-fly state space reduction which (to the best of our knowledge) does not exist for the TPS semantics. As time in deterministically timed systems has a global synchronising character, efforts to apply partial-order reduction to timed systems have not been very successful.

Model checking SADF. Earlier work [Theelen et al. 2012] exploited the CADP toolset for model checking eSADF. There are various benefits and differences with the approach in this paper. First, we provide a *full formal* definition of the eSADF semantics. Secondly, the operational model in [Theelen et al. 2012] is better suited for SDF than for SADF. In particular, it does not natively support probabilistic choices (as needed for random sub-scenario selection in SADF). Using MA, there is no need for awkward—and incomplete—transformations [Rettelbach 1995] to delete probabilistic branching as applied in [Theelen et al. 2012]. This results in smaller models. In addition, using MA a much richer palette of quantitative measures can be supported whereas CADP only supports transient and steady-state measures. In fact, the absence of non-determinism allows for a full-fledged model checking of stochastic versions of CTL. Finally, confluence reduction is an *on-the-fly* technique whereas bisimulation reduction (as applied in [Theelen et al. 2012]) is not. As shown in the following table

	no red.	with red.	red. factor
[Theelen et al. 2012]	121430	20664	5.88
Our work	47266	16042	2.95

the use of MA yields smaller models (without reduction), whereas confluence reduction outperforms branching bisimulation used in [Theelen et al. 2012] while preserving three quantitative measures addressed in this paper.

Energy analysis by model checking. Several models and model-checking approaches have been extended towards the treatment of costs, or dually: rewards. Prominent examples are priced timed automata and Markov reward chains. Like in MA, states are equipped with a reward that grows linearly depending on the state residence time. Markov reward chains are extensions of DTMCs in which a reward is earned on visiting a state (no dependency on state residence times). [Norman et al. 2005] applied this model class so as to quantify the impact of various DPM schemes. This has been done using the PRISM model checker. This paper applies a similar analysis on continuous-time probabilistic models that include non-determinism. Our semantics and analysis algorithms allows similar analyses for all eSADF models. Recently, [Ahmad et al. 2015] provided a mapping from SADF (together with an execution platform) onto timed automata, and using the timed automata model checker UPPAAL, the authors showed how to determine the schedulability of SADF graphs. An extension of this approach with prices enables the verification of timed energy usage.

9. CONCLUSION AND FUTURE WORK

We presented a compositional semantics of eSADF, SADF in which all executions take exponential time. The semantics is provided in terms of Markov automata, a formal model that naturally fits all the ingredients of eSADF. Two case studies have been provided that illustrate achievable state space reductions using confluence reduction and obtaining quantitative assertions about eSADF graphs in a fully automated manner. The incorporation of the execution platform into our framework is shown to enable energy analysis.

Future work consists of considering more realistic execution platforms, the comparison of different deployment strategies of SADF actors, and the use of parametric verification to synthesise maximal (or minimal) execution times from high-level specifications.

ACKNOWLEDGMENTS

References

Waheed Ahmad, Philip K. F. Hölzenspies, Mariëlle Stoelinga, and Jaco van de Pol. 2015. Green Computing: Power Optimisation of VFI-Based Real-Time Multiprocessor Dataflow Applications. In *Euromicro Conf. on Digital System Design (DSD)*. IEEE Computer Society, 271–275.

Shuvra S. Bhattacharyya, Ed F. Deprettere, and Bart D. Theelen. 2013. Dynamic Dataflow Graphs. In *Handbook of Signal Processing Systems*. Springer, 905–944.

Hichem Boudali, A. P. Nijmeijer, and Mariëlle Stoelinga. 2009. DFTSim: A Simulation Tool for Extended Dynamic Fault Trees. In *SpringSim*. SCS/ACM.

Marco Bozzano, Alessandro Cimatti, Joost-Pieter Katoen, Viet Yen Nguyen, Thomas Noll, and Marco Roveri. 2011. Safety, Dependability and Performance Analysis of Extended AADL Models. *Computer Journal* 54, 5 (2011), 754–775.

J.T. Buck and Edward A. Lee. 1993. Scheduling dynamic dataflow graphs with bounded memory using the token flow model. In *IEEE Int. Conf. on Acoustics, Speech, and Signal Processing (ICASSP)*, Vol. 1. 429–432.

Jeronimo Castrillon and Rainer Leupers. 2013. *Programming Heterogeneous MPSoCs: Tool Flows to Close the Software Productivity*. Springer.

Yuxin Deng and Matthew Hennessy. 2013. On the Semantics of Markov Automata. *Information and Computation* 222 (2013), 139–168.

Christian Eisentraut, Holger Hermanns, Joost-Pieter Katoen, and Lijun Zhang. 2013. A Semantics for Every GSPN. In *Petri Nets (LNCS)*, Vol. 7927. Springer, 90–109.

Christian Eisentraut, Holger Hermanns, and Lijun Zhang. 2010. On Probabilistic Automata in Continuous Time. In *IEEE Symp. on Logic in Computer Science (LICS)*. IEEE, 342–351.

Johan Eker, Jörn Janneck, Edward Lee, Jie Liu, Xiaojun Liu, Jozsef Ludvig, Stephen Neuendorffer, Sonia Sachs, and Yuhong Xiong. 2003. Taming Heterogeneity - The Ptolemy Approach. *Proc. IEEE* 91, 1 (2003), 127–144.

Marc Geilen and Twan Basten. 2004. Reactive Process Networks. In *EMSOFT*. ACM, 137–146.

Jan Friso Groote and Alban Ponse. 1995. The Syntax and Semantics of μCRL. In *Algebra of Communicating Processes*. Springer, 26–62. http://dx.doi.org/10.1007/978-1-4471-2120-6_2

Dennis Guck, Hassan Hatefi, Holger Hermanns, Joost-Pieter Katoen, and Mark Timmer. 2014a. Analysis of Timed and Long-run Objectives for Markov Automata. *Logical Methods in Computer Science* 10 (2014), 1–29.

Dennis Guck, Mark Timmer, Hassan Hatefi, Enno Ruijters, and Mariëlle Stoelinga. 2014b. Modelling and Analysis of Markov Reward Automata. In *Int. Symp. on Automated Technology for Verification and Analysis (ATVA) (LNCS)*, Vol. 8837. Springer, 168–184.

Gilles Kahn. 1974. The Semantics of Simple Language for Parallel Programming. In *IFIP Congress*. 471–475.

Joost-Pieter Katoen and Hao Wu. 2014. Exponentially Timed SADF: Compositional Semantics, Reductions, and Analysis. In *Int. Conf. on Embedded Software (EMSOFT)*. ACM.

Edward A. Lee and David G. Messerschmitt. 1987. Synchronous Data Flow: Describing Signal Processing Algorithm for Parallel Computation. In *COMPCON*. IEEE, 310–315.

Marco Ajmone Marsan, Gianni Conte, and Gianfranco Balbo. 1984. A Class of Generalized Stochastic Petri Nets for the Performance Evaluation of Multiprocessor Systems. *ACM TOCS* 2, 2 (1984), 93–122.

Gethin Norman, David Parker, Marta Z. Kwiatkowska, Sandeep K. Shukla, and Rajesh Gupta. 2005. Using probabilistic model checking for dynamic power management. *Formal Asp. Comput.* 17, 2 (2005), 160–176. http://dx.doi.org/10.1007/s00165-005-0062-0

Sangyoung Park, Jaehyun Park, Donghwa Shin, Yanzhi Wang, Qing Xie, Massoud Pedram, and Naehyuck Chang. 2013. Accurate Modeling of the Delay and Energy Overhead of Dynamic Voltage and Frequency Scaling in Modern Microprocessors. *IEEE Trans. on CAD of Integrated Circuits and Systems* 32, 5 (2013), 695–708.

Michael Rettelbach. 1995. Probabilistic Branching in Markovian Process Algebras. *Computer Journal* 38, 7 (1995), 590–599.

Ahsan Shabbir, Akash Kumar, Sander Stuijk, Bart Mesman, and Henk Corporaal. 2010a. CA-MPSoC: An Automated Design Flow for Predictable Multi-processor Architectures for Multiple Applications. *Journal of Systems Architecture - Embedded Systems Design* 56, 7 (2010), 265–277.

Ahsan Shabbir, Sander Stuijk, Akash Kumar, Bart D. Theelen, Bart Mesman, and Henk Corporaal. 2010b. A predictable communication assist. In *7th Conf. on Computing Frontiers*. ACM, 97–98.

Sundararajan Sriram and Shuvra S. Bhattacharyya. 2009. *Embedded Multiprocessors: Scheduling and Synchronization*. CRC Press.

Bart D. Theelen. 2007. A Performance Analysis Tool for Scenario-Aware Streaming Applications. In *QEST*. 269–270.

Bart D. Theelen, Marc Geilen, Twan Basten, Jeroen Voeten, Stefan Valentin Gheorghita, and Sander Stuijk. 2006. A scenario-aware data flow model for combined long-run average and worst-case performance analysis. In *MEMOCODE*. IEEE, 185–194.

Bart D. Theelen, Marc Geilen, Sander Stuijk, Stefan Valentin Gheorghita, Twan Basten, Jeroen Voeten, and A. Ghamarian. 2008. *Scenario-aware dataflow*. Technical Report ESR-2008-08. TU Eindhoven.

Bart D. Theelen, Joost-Pieter Katoen, and Hao Wu. 2012. Model checking of Scenario-Aware Dataflow with CADP. In *DATE*. IEEE, 653–658.

Mark Timmer, Joost-Pieter Katoen, Jaco van de Pol, and Mariëlle Stoelinga. 2012. Efficient Modelling and Generation of Markov Automata. In *Int. Conf. on Concurrency Theory (CONCUR) (LNCS)*, Vol. 7454. Springer, 364–379.

Mark Timmer, Jaco van de Pol, and Mariëlle Stoelinga. 2013. Confluence Reduction for Markov Automata. In *FORMATS (LNCS)*, Vol. 8053. Springer, 243–257.

Stavros Tripakis, Dai N. Bui, Marc Geilen, Bert Rodiers, and Edward A. Lee. 2013. Compositionality in synchronous data flow: Modular code generation from hierarchical SDF graphs. *ACM Trans. Embedded Comput. Syst.* 12, 3 (2013), 83:1–83:26.

Received ; revised ; accepted

Online Appendix to: Probabilistic Model Checking for Uncertain Scenario-Aware Data Flow

JOOST-PIETER KATOEN, Software Modelling and Verification Group, RWTH Aachen University
HAO WU, Software Modelling and Verification Group, RWTH Aachen University

 1539-9087/2016/01-ART $15.00
DOI: 0000001.0000001