Exponentially Timed SADF: Compositional Semantics, Reductions, and Analysis

Joost-Pieter Katoen
Software Modelling and Verification Group
RWTH Aachen University
katoen@cs.rwth-aachen.de

Hao Wu
Software Modelling and Verification Group
RWTH Aachen University
hao.wu@cs.rwth-aachen.de

ABSTRACT

This paper presents a rigorous compositional semantics for SADF (Scenario-Aware Data Flow), an extension of SDF for scenario-based embedded system design which has its roots in digital signal processing. We show that Markov automata (MA), a novel combination of probabilistic automata and continuous-time Markov decision processes, provides a natural semantics when all execution times are exponential. The semantics is fully compositional, i.e., each SADF agent is modeled by a single automaton which are all put in parallel. We show how stochastic model checking can be used to analyse the MA, yielding measures such as expected time, long-run objectives, throughput, and timed reachability. Using aggressive reduction techniques for Markov automata that are akin to partial-order reduction, scalability of analysis is achieved, and all non-determinism can be eliminated.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: Modeling techniques, Performance attributes

General Terms

Design, Performance, Verification

Keywords

SADF, Compositional semantics, Stochastic model checking, State space reduction

1. INTRODUCTION

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, 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) [13], a language which is equally expressive as weighted marked graphs. SDF has been used extensively [1, 8, 16] 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 (e.g., memory usage, energy) vary substantially depending on their context (e.g., input data and quality level). Dataflow languages [4, 9, 12] are intended as an implementation framework for such dynamic applications. However, these languages lack facilities for predicting performance during embedded system design and have rather limited means to describe dynamic dataflow behaviour. The *Scenario-Aware Dataflow (SADF)* language [18, 19, 1] 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 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 is given in [1].

In this paper, we consider *exponentially timed* SADF (called eSADF), i.e., a version of SADF in which the duration of all firings of actors are governed by negative exponential distributions. (Exponentially timed SDF has been considered in [16, Ch. 8].) 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.) Besides, using exponential distributions enables the usage of modern probabilistic model-checking tools for the quantitative analysis. 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) [5, 7] are a natural choice for capturing the semantics of eSADF. MA 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 formal *compositional* semantics of eSADF using MA. The compositional aspect naturally reflects the logical structure

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions@acm.org.
ESWEEK'14, October 12 - 17, 2014, New Delhi, India
Copyright 2014 ACM 978-1-4503-3052-7/14/10 ...$15.00.
http://dx.doi.org/10.1145/2656045.2656058

of the eSADF graph enabling component-wise reduction. Compositionality in SDF has recently been exploited for modular code generation [23]. Our semantics is defined using a succinct process algebra for describing MA [21]. As a result, the semantics is relatively easy to comprehend and modular. Confluence reduction [22] – a technique akin to partial-order reduction – allows for an on-the-fly reduction of the state space. 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. The MA-semantics enables the automated quantitative evaluation of several performance measures of interest such as expected time, long-run average and timed reachability objectives [11]. Using an MPEG-4 decoder, we show the effect of confluence reduction and analyse several measures of interest such as buffer occupancy and throughput. We compare the results to analysis results using the SDF3 tool for SADF [18] and to [20]. To sum up, our semantics is simple and yields a rigorous framework for analysing eSADF graphs. It extends [20] in several ways, most notably by a simpler semantic model (resulting in a more succinct state space), a fully detailed semantics, an on-the-fly reduction technique (rather than bisimulation), and various analysis facilities (rather than just transient and steady-state analysis).

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. Section 5 discusses non-determinism, and the quantitative evaluation of eSADF using MA. Section 6 discusses related work and Section 7 concludes.

2. SCENARIO-AWARE DATAFLOW

In this section, we will give a formal definition of an exponentially-timed SADF (eSADF) graph which is based on [17]. Figure 4 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.

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 [13], 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 $\mathcal{DC} \cap \mathcal{CC} = \varnothing$. 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 by 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 $C = (cc_1, cc_2)$, $\Sigma_{cc_1} = \{s, t\}$ 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 which captures the dynamics in systems.

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 $\mathcal{D}$. The set of processes $\mathcal{P} = \mathcal{K} \cup \mathcal{D}$ with $\mathcal{K} \cap \mathcal{D} = \varnothing$.

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

- P *is a* set of ports *partitioned into the sets* 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.

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 execute in scenarios but in sub-scenarios which are determined by scenarios it receives via its 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*,*
  $\triangleright\, \Omega$*, a non-empty finite* set of sub-scenarios values*,*
  $\triangleright\, F : \Sigma \to M$*, the* random decision function*.* M *is the set of DTMCs* $(\mathbb{S}, \iota, \mathbb{P}, \Phi)$ [1]*. Function* F *associates a*

[1]DTMC = Discrete-Time Markov Chain.

DTMC to each scenario $\sigma \in \Sigma$. Here, $\mathbb{S}, \iota$ and $\mathbb{P}$ are the finite 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 $o_c \in P_{O_c}$ 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, say 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 $\mathbb{P}$. In the second phase, D will execute these sub-scenarios. The production rate function R and execution rate 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 exponential-time SADF (eSADF) graphs, we need a consistent way to connect the processes (i.e. kernels or detectors) 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.*

We call an eSADF *type consistent*, if all the channels in the eSADF graph 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 of content.* [2]

3. MARKOV AUTOMATA

In this section, we introduce the semantic model, Markov automata (MA) [7, 5], 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 [6] of modelling languages such as dynamic fault trees [2], domain-specific language AADL [3], and generalised stochastic Petri nets (GSPNs) [14].

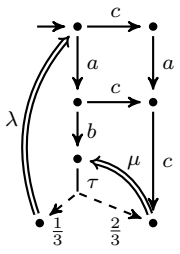
The syntax of MA. A distribution μ over a countable set S is a function $\mu : S \to [0, 1]$ such that $\sum_{s \in S} \mu(s) = 1$. Let $\mathrm{Distr}(S)$ be the set of all distributions over set S.

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,
- *$\hookrightarrow \subseteq S \times Act \times \mathrm{Distr}(S)$ 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* $\mathbf{P}(s, s') = \frac{\mathbf{R}(s, s')}{\mathbf{E}(s)}$.



A sample MA

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 synchronization, 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 2 (c)). Note that non-determinism exists when there are multiple

[2]Note that "rate" consistent eSADF can be defined as in [19].

internal probabilistic transitions emanating from one state.

MA Process Algebra (MAPA). To generate an MA, a language based on μCRL [10] called MA Process Algebra (MAPA) was introduced in [21]. We use MAPA to define our MA semantics for eSADF.

Definition 8. **[Process terms]** *A process term in MAPA can be generated by the following rules:*

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

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. 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 $a(\mathbf{t}) \sum_{\mathbf{x}:\mathbf{D}} f : p$ 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 $\lambda \in \mathbb{R}_{>0}$, 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 [21].

Definition 9. *An initial process in parallel MAPA is any term that can be generated by the following rules:*

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

Here, $Y \in Prc$ is a process name, $\mathbf{t}$ is a vector of data expressions, $I, H \subseteq Act\backslash\{\tau\}$ are sets of actions. 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 the *parallel composition* of q_1 and q_2 w.r.t the action set I (defined in [21] 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 MAPA to [21].

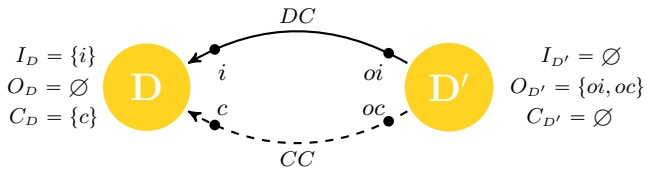
4. SEMANTICS OF eSADF

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 unbounded 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 as part of 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 gen-



eralized 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, $\mid cc_{(D',D)} \mid$ 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 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 "—", and the synchronization actions waiting for the synchronization are denoted as usual. Note that this notation will not affect the original MA semantics.

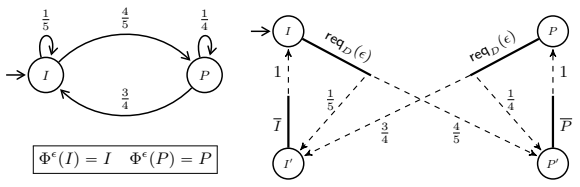
Definition 10. *The MA semantics of the scenario module SM^σ of detector D in scenario σ is defined by:*

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

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

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 [17] 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 through synchronization subsc 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) of a scenario in eSADF to an MA (the corresponding SM) (right figure).*



We initialize now 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. *The scenario module of detector D is:*

$$SM_D = SM^{\sigma_1} \;|||\; SM^{\sigma_2} \;|||\; \cdots \;|||\; SM^{\sigma_n}$$

where $|||$ equals $\|_\varnothing$ (as there is no synchronization between the MAs for scenarios), and $\Sigma = \{\sigma_1, \ldots, \sigma_n\}, n \in \mathbb{N}$.

Definition 12. *The MA semantics of the functional module FM of a detector D is defined by the MAPA term:*

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

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 ($\texttt{head}(cc) = \sigma$), 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.

Definition 13. *The FM of detector D is defined by:*

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

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

Definition 14. *The MA semantics of detector D is:*

$$\mathcal{M}_D := \tau_{I_D}(FM_D \;\|_{I_D}\; SM_D)$$

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

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. As K has no control outports (so does D_K), the function t and rate function R are not defined. The scenario module SM of D_K in scenario ω is defined as described in Remark 1 by:

$$SM(S_0) := \quad \mathsf{req}(\omega).SM'(S_0)$$
$$SM'(S_0) := \quad \overline{\mathsf{subsc}(\omega)}.SM(S_0)$$

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 15. *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 1 (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.*

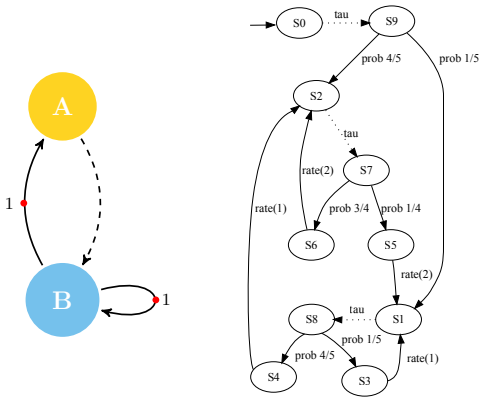


Figure 1. A sample eSADF graph and its optimized MA

5. ANALYSIS

This section presents an on-the-fly reduction technique for MA. This reduction is used for two purposes. First, we show that *all* non-determinism in the MA semantics of an eSADF graph can be safely removed. Then we show that reduction scales the quantitative evaluation of eSADF graphs.

5.1 State Space Reduction & Non-Determinism

Confluence reduction. Confluence reduction [21] reduces the state space based on commutativity of transitions, removing nondeterministic transitions caused by parallel composition of independent components. It is similar in spirit to partial-order reduction. The reduction preserves the quantitative metrics of interest. 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 2, which gives the MA semantics (b) for a GSPN (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.

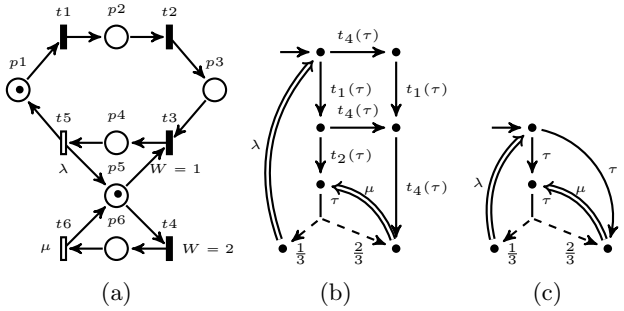
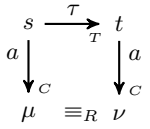


Figure 2: (a) a sample GSPN, (b) its MA semantics, and (c) its reduced MA

Confluence reduction in a nutshell. The basic idea of confluence reduction is to determine the confluent sets of transitions [21]. 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 right diagram illustrates the latter constraint. 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 [22]. 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 [20]). Case studies show a state space reduction from 26% to 83% [21]; for the MPEG-4 decoder this is about 66% (cf. Table 1).



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.*

Proof (sketch) Now, we will show that possible non-determinism in the resulting MA of an eSADF graph is due to execution of independent actors only. Later on, we will argue that allo this non-determinism can be eliminated—while preserving quantitative properties—by confluence reduction. Recall that hybrid states do not occur in the MA-semantics $\mathcal{M}_\mathcal{G}$ of an eSADF graph $\mathcal{G}$, as all interactions have been turned into τ-transitions by hiding. We will show that the non-determinism between these τ-transitions results from independent sub-scenario decision transitions in distinct actors. We start off by making the following observations. The interactive transitions in MA $\mathcal{M}_\mathcal{G}$ are either the channel status synchronization between different processes (action exe), or sub-scenario decision transitions within a process (actions req and subsc). Sub-scenario decision transitions are fully autonomous and are not subject to synchronisation.

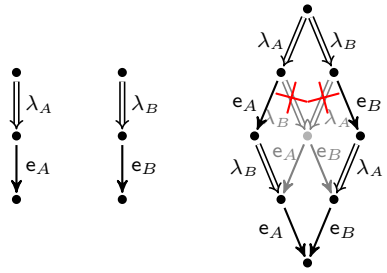


Figure 3: The state where both synchronization transitions are enabled is unreachable due to maximal progress assumption

FACT 1. *There is no non-determinism between any channel status synchronization actions* exe.

This can be seen as follows. Observe that a channel status update synchronization action (i.e., exe_p) can only be enabled after a preceding Markovian transition and, as it is autonomous, occurs immediately. Two of these actions are thus simultaneously enabled with probability zero. As a result, the MA $\mathcal{M}_\mathcal{G}$ does not contain a state in which two synchronization actions are enabled. This situation is illustrated in Figure 3.

FACT 2. *There is no non-determinism between any sub-scenario decision (action* req *or* subsc*) and channel status synchronization (action* exe*).*

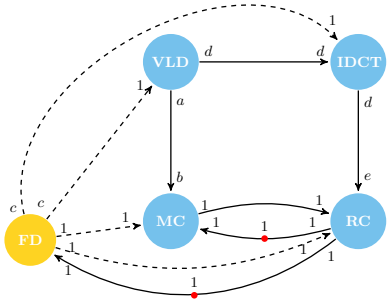
We argue by contraposition. Assume there is a non-deterministic choice between a channel status synchronization transition and a sub-scenario decision transition. As argued above, channel status synchronization is immediately preceded by a Markovian transition, t say. Since Markovian transitions do not change any channel status (as it does not update any variables), the sub scenario decision transition is also enabled before t. Due to the maximal progress assumption, the sub-scenario decision transition must happen before t and therefore before the channel status synchronization transition. Contradiction.

FACT 3. *If non-determinism occurs between two* req*-actions or two* subsc*-actions, then these actions are independent.*

(Here, actions a and b are independent if in any state in which both actions are enabled, the occurrence of a does not disable b, and vice versa. Moreover, the resulting state after firing these transitions in any order, is the same.) Suppose there are two (or more) sub-scenario decisions (action req) are enabled. Evidently, these transitions originate from different actors. Since there are no shared variables between actors, and the execution of a sub-scenario decision transition in one actor does not disable such transition in another actor, these actions do not disable each other and result in the same state. Gathering the above results yields, as pointed out in [17] using a more informal and different semantic approach, the Theorem 1.

The proof confirms a similar result in [19] for an alternative SADF semantics. The key point is that, since the enabled probabilistic choice among sub-scenarios is instanteneous, the transitions representing the time progress can not be enabled before all such enabled probabilistic transitions[3] have occurred. Since these enabled probabilistic transitions are independent, they are confluent to the state where the Markovian transitions are enabled. Whereas in SADF [19] 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.

5.2 Example: an MPEG-4 Decoder



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

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

Figure 4. An SADF model for an MPEG-4 decoder [19]

We consider an eSADF graph (cf. Figure 4) of an MPEG-4 decoder for the simple profile [19]. This 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 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

[3] In SADF this is ensured by action-urgency; here by maximal progress.

size of 176 × 144 pixels (QCIF), i.e, an I frame consists of 99 macro blocks. (cf. Figure 4 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 [19]. This is captured by the scenarios P_0 and P_x as indicated in Figure 4 (right). All parameters in our case study are taken from [19], which are obtained by using a specific profiling tool. We have developed a prototypical tool for obtaining MAPA terms of eSADF graphs (expressed in the XML variant for SADF [4]). The generated MAPA term for the eSADF graph of Example 2 is given in Figure 8 (cf. Appendix). Applying confluence reduction reduces the state space by about a factor 3:

		no red.	with conf. red.
Ex. 2	#states	19	7
	#transitions	19	7
MPEG-4	#states	61918	20992
	#transitions	81847	40910
	#non-det. states	4	1

Table 1. MA size for sample eSADF graphs

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

5.3 Automated Quantitative Evaluation

Our MA semantics allows for determining several performance metrics for the MPEG-4 decoder in a fully automated manner. These algorithms are described in [11]. We illustrate this for various quantitative measures of the MPEG-4 decoder example.

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

	p1	p2	p3	p4	p5
av. # tok.	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	24.3675	1.18198	0.4412	0.8574

Table 2: Average buffer occupancy in each MPEG-4 channel

each channel of the MPEG-4 decoder is shown in Table 2. 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. [5]

The token distributions of three data channels are shown in Figure 5. The peak values for channels VLD-MC and IDCT-RC are caused by conservatively approximating the

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

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

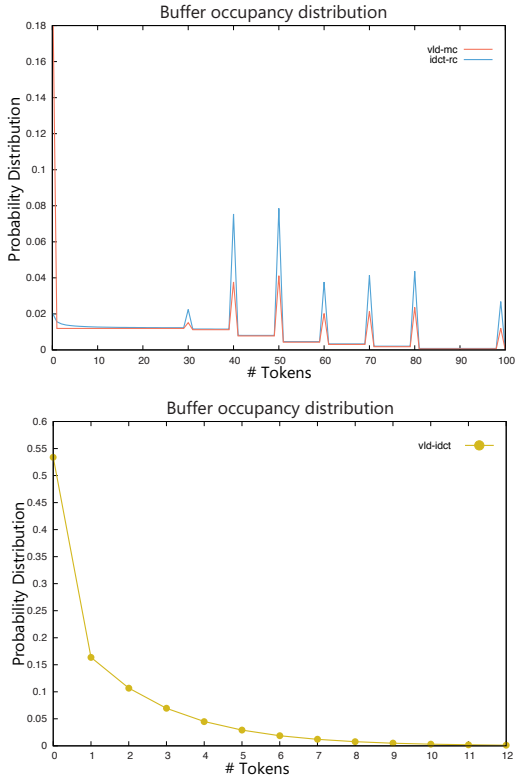


Figure 5. Token distribution in two MPEG-4 channels

motion vectors (cf. Section 5.2) 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 of channels 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.

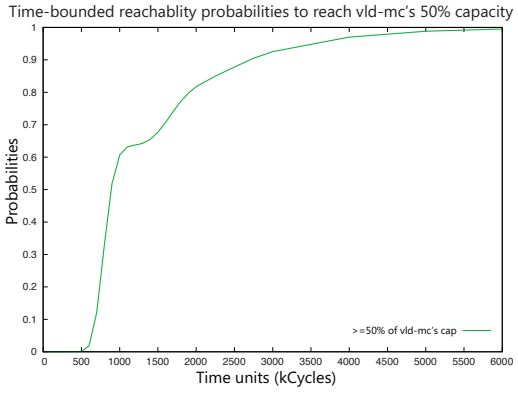


Figure 6: Time-bounded reachability probabilities to reach VLD-MC's 50% capacity

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. For space sake, we only show the result in 50% case of channel VLD-MC in Figure 6.

Response delay. The time-bounded reachability evaluation can also be utilized for *response time estimation*. Here we can 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 the 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 7 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.

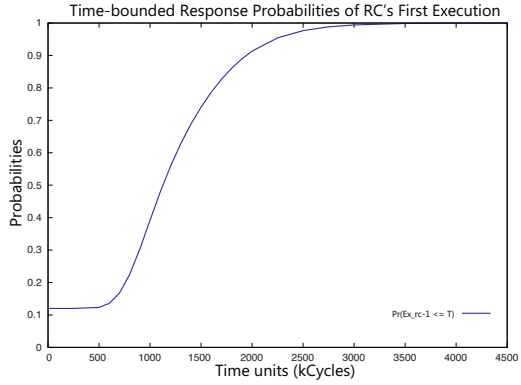


Figure 7. First response probabilities of RC within time t

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 again 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 are shown in Table 3.

	p8	p19	p10	p11
through.	IDCT	VLD	MC	RC
MaMa	0.0423732	0.0423732	0.000746128	0.000746128
SDF^3	0.0437919	0.0437919	0.000745268	0.000745268

Table 3. Throughput of each kernel in MPEG-4 decoder

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 here MC for an example and compute the In of MC as 1460.5 (1341.8 in SDF^3) kCycles. For an overview, the table of verification time (compare with SDF^3) of properties (on a machine with an 48-core CPU at 2.1GHz and 192GB memory) is shown in Table 4.

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

Table 4. Verification time of properties [6]

6. RELATED WORK

Whereas we consider exponentially timed SADF – akin to exponentially timed SDF [16] – and use Markov automata as semantic model, the original works on SADF focus on a *timed* semantics using Timed Probabilistic Systems (TPS) [18, 17, 19]. 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. 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 [19] for TPS semantics). Finally, confluence reduction allows for an on-the-fly state space reduction which (to the best of our knowledge) does seem to exist for the TPS semantics.

Earlier work [20] exploited the CADP tool-set for model checking eSADF. There are various benefits and differences with our current approach. First, we provide a *full formal* definition of the eSADF semantics. Secondly, the operational model in [20] 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 [15] to delete probabilistic branching as applied in [20]. 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 [20]) is not. As shown in the following table

	no red.	with red.	red. factor
[20]	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 [20] while preserving basically the same quantitative measures.

7. CONCLUSION AND FUTURE WORK

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

We presented a compositional semantics of eSADF, SADF in which all executions take exponential time. We showed that non-determinism only results from executing independent processes, and that this can be eliminated using confluence reduction. Our semantics enables quantitative evaluation using modern stochastic model checking. Given the close relationship between GSPN and MA [6], the MA semantics of SADF is a good starting point to address the open question to which class of Petri nets SADF corresponds.

Acknowledgement This work is funded by the EU FP7 project SENSATION[7] and the EU exchange project MEALS[8] under grant agreement no. 318490 and 295261, respectively.

8. REFERENCES

[1] S. S. Bhattacharyya, E. F. Deprettere, and B. D. Theelen. Dynamic dataflow graphs. In *Handbook of Signal Processing Systems*, pages 905–944. Springer, 2013.

[2] H. Boudali, A. P. Nijmeijer, and M. Stoelinga. Dftsim: a simulation tool for extended dynamic fault trees. In *SpringSim*, page 31. SCS/ACM, 2009.

[3] M. Bozzano, A. Cimatti, J.-P. Katoen, V. Nguyen, T. Noll, and M. Roveri. Safety, dependability and performance analysis of extended AADL models. *Comp. J.*, 54(5):754–775, 2011.

[4] J. Buck and E. Lee. Scheduling dynamic dataflow graphs with bounded memory using the token flow model. In *ICASSP*, pages 429–432, 1993.

[5] Y. Deng and M. Hennessy. On the semantics of Markov automata. *Inf. and Comp.*, 222:139–168, 2013.

[6] C. Eisentraut, H. Hermanns, J.-P. Katoen, and L. Zhang. A semantics for every GSPN. In *Petri Nets*, volume 7927 of *LNCS*, pages 90–109. Springer, 2013.

[7] C. Eisentraut, H. Hermanns, and L. Zhang. On probabilistic automata in continuous time. In *LICS*, pages 342–351, 2010.

[8] J. Eker, J. Janneck, E. Lee, J. Liu, X. Liu, J. Ludvig, S. Neuendorffer, S. Sachs, and Y. Xiong. Taming heterogeneity - the Ptolemy approach. *Proc. IEEE*, 91(1):127–144, 2003.

[9] M. Geilen and T. Basten. Reactive process networks. In *EMSOFT*, pages 137–146. ACM, 2004.

[10] J. Groote and A. Ponse. The syntax and semantics of μCRL. In *Algebra of Communicating Processes*, Workshops in Computing, pages 26–62. Springer, 1995.

[11] D. Guck, H. Hatefi, H. Hermanns, J.-P. Katoen, and M. Timmer. Modelling, reduction and analysis of Markov automata. In *QEST*, volume 8054 of *LNCS*, pages 55–71, 2013.

[12] G. Kahn. The semantics of simple language for parallel programming. In *IFIP Congress*, pages 471–475, 1974.

[13] E. A. Lee and D. G. Messerschmitt. Synchronous data flow: Describing signal processing algorithm for parallel computation. In *COMPCON*, pages 310–315. IEEE, 1987.

[14] M. A. Marsan, G. Conte, and G. Balbo. A class of generalized stochastic Petri nets for the performance evaluation of multiprocessor systems. *ACM TOCS*, 2(2):93–122, 1984.

[7] http://www.sensation-project.eu/

[8] http://www.meals-project.eu/

[15] M. Rettelbach. Probabilistic branching in Markovian process algebras. *Comput. J.*, 38(7):590–599, 1995.

[16] S. Sriram and S. S. Bhattacharyya. *Embedded Multiprocessors: Scheduling and Synchronization*. CRC Press, 2009.

[17] B. Theelen. A performance analysis tool for scenario-aware streaming applications. In *QEST*, pages 269–270, 2007.

[18] B. Theelen, M. Geilen, S. Stuijk, S. Gheorghita, T. Basten, J. Voeten, and A. Ghamarian. Scenario-aware dataflow. Technical Report ESR-2008-08, TU Eindhoven, 2008.

[19] B. D. Theelen, M. Geilen, T. Basten, J. Voeten, S. V. Gheorghita, and S. Stuijk. A scenario-aware data flow model for combined long-run average and worst-case performance analysis. In *MEMOCODE*, pages 185–194. IEEE, 2006.

[20] B. D. Theelen, J.-P. Katoen, and H. Wu. Model checking of scenario-aware dataflow with CADP. In *DATE*, pages 653–658. IEEE, 2012.

[21] M. Timmer, J.-P. Katoen, J. van de Pol, and M. Stoelinga. Efficient modelling and generation of Markov automata. In *CONCUR*, volume 7454 of *LNCS*, pages 364–379, 2012.

[22] M. Timmer, J. van de Pol, and M. Stoelinga. Confluence reduction for Markov automata. In *FORMATS*, volume 8053 of *LNCS*, pages 243–257. Springer, 2013.

[23] S. Tripakis, D. N. Bui, M. Geilen, B. Rodiers, and E. A. Lee. Compositionality in synchronous data flow: Modular code generation from hierarchical SDF graphs. *ACM Trans. Embedded Comput. Syst.*, 12(3):83:1–83:26, 2013.