Performance Model Checking Scenario-Aware Dataflow

Bart Theelen[1], Marc Geilen[2] and Jeroen Voeten[1,2]

[1] Embedded Systems Institute
[2] Eindhoven University of Technology, Department of Electrical Engineering
P.O. Box 513, 5600 MB Eindhoven, The Netherlands

Abstract. Dataflow formalisms are useful for specifying signal processing and streaming applications. To adequately capture the dynamic aspects of modern applications, the formalism of Scenario-Aware Dataflow (SADF) was recently introduced, which allows analysis of worst/best-case and average-case performance across different modes of operation (scenarios). The semantic model of SADF integrates non-deterministic and discrete probabilistic behaviour with generic discrete time distributions. This combination is different from the semantic models underlying contemporary quantitative model checking approaches, which often assume exponentially distributed or continuous time or they lack support for expressing discrete probabilistic behaviour. This paper discusses a model-checking approach for computing quantitative properties of SADF models such as throughput, time-weighted average buffer occupancy and maximum response time. A compositional state-space reduction technique is introduced as well as an efficient implementation of this method that combines model construction with on-the-fly state-space reductions. Strong reductions are possible because of special semantic properties of SADF, which are common to dataflow models. We illustrate this efficiency with several case studies from the multi-media domain.

1 Introduction

Signal processing and streaming applications are often described as a set of tasks, actors or processes with data and control dependencies to exploit the parallel and pipelined execution capabilities of hardware platforms. Modern streaming applications are increasingly dynamic, with large variations in the required resources. Neglecting these variations when evaluating key properties like throughput and buffer occupancy can result in overly pessimistic performance bounds [10], while the average-case behaviour can often not be studied adequately based on the same model. The recently introduced formalism of *Scenario-Aware Dataflow* (SADF) [33] adequately captures dynamism in modern streaming applications using *scenarios*. Such scenarios denote distinct modes of operation (like processing I, P or B frames in MPEG-4 video processing), in which resource requirements can differ substantially [11, 25].

This paper presents the techniques underlying the computation of exact worst/best-case and average-case performance numbers for SADF models as implemented in the SDF^3 toolkit [32, 29]. Although these techniques are strongly inspired by existing model checking techniques, they are not based on existing model checkers that support quantitative analysis. This is because of semantic differences between the model of SADF and

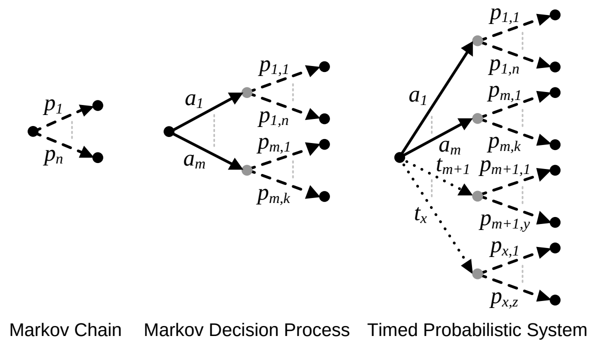


Fig. 1. Transitions in Probabilistic Automata

the models underlying common quantitative model checkers combined with the diversity of metrics that we want to analyse. The semantics of SADF [34] uses the formalism of *Timed Probabilistic (Labelled Transition) Systems* (TPS) [1]. Like other automata, a TPS describes behaviour in terms of states and transitions. TPS is a variant of probabilistic timed automata in [26] (called Simple Segala Model in [28]), which extend Markov decision processes (MDPs) [5, 24] by distinguishing time-less action transitions from transitions for advancing time. Figure 1 shows how MDP alternates non-determinism between actions $a_1, \ldots, a_m$ with probabilistic choices. Actions in TPS are time-less. Time is modelled explicitly using separate transitions, where the labels $t_{m+1} \ldots t_x$ in Figure 1 refer to an *exact* (positive) amount of time (e.g., they do not refer to parameters of exponential distributions). Modelling discrete time distributions can be accomplished by using the two-step approach depicted in Figure 2. It consists of some (internal) action capturing the probabilistic choice of alternative time durations (think of drawing a sample from a discrete distribution), followed by a time transition for each of the possible time durations. In Figure 2, time advances t_i time units with probability p_i for $i = 1, \ldots, n$. This approach is suitable for capturing any discrete time distribution and matches well with the way discrete-event models are commonly implemented. The semantics of SADF in [34] adopts this approach to ease calibration of SADF models with profiling data obtained through static and statistic code analysis.

Several quantitative model checkers exist but using them for SADF is problematic. CADP [12] is a model checking toolbox that supports (amongst others) Interactive Markov Chains (IMC) [13] for quantitative analysis. Although it supports non-deterministic choice between alternative behaviours, IMC itself does not support probabilistic choices and it relies on exponentially distributed time. The probabilistic model checker MRMC [14], which operates on more elementary automata, supports both probabilistic and non-deterministic choices but only in combination with exponentially distributed time. UPPAAL [17] is well-known for its ability to verify qualitative properties of timed systems and the recent extension UPPAAL-PRO adds support for probabilistic choices. Its continuous-time model is again different from the time model of TPS. Nevertheless, TPS can be captured reasonably straightforwardly in UPPAAL-PRO. However, only analysis of maximum probabilistic reachability properties is supported, which is insufficient to compute metrics like throughput and time-weighted av-

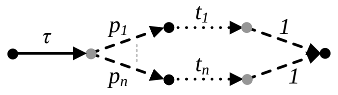


Fig. 2. Generic Discrete Time Distribution

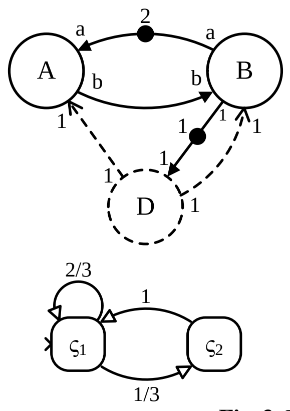
erage buffer occupancy. PRISM [15] can verify a wider range of quantitative properties for various probabilistic models including (Priced) Probabilistic Timed Automata. Analysis of reward-based metrics like throughput and buffer occupancy is however not supported for such automata. Furthermore, PRISM lacks features like the concept of urgency in UPPAAL to ease controlling resolution of non-determinism, which is very useful in the context of dataflow formalisms [8, 9]. We discuss this aspect in Section 3.

Given the mismatch with existing model checkers, we propose a novel, two-phase approach. In the first phase we construct a reduced, but adequate Markov reward model on which elementary techniques for computing performance numbers can be applied in the second phase, possibly using existing tools. From the basic analysis results, we construct results for more complex performance metrics in a compositional way. This paper focuses on the first phase where we apply several model checking techniques optimized for SADF. These techniques restrain state-space size by exploiting key semantic properties of SADF as well as neglecting events that do not affect a performance result directly. We present how this approach allows for an *on-the-fly* construction of the Markov reward model, which is the key contribution of this paper. A number of case studies from the multi-media domain illustrate the achievable state-space reductions by taking semantic properties and the relevance of events into account.

The remainder of this paper is organised as follows. The next section discusses relevant properties of TPSs defined by SADF semantics. Section 3 presents our performance model checking approach and how it can be implemented efficiently. In Section 4, we present algorithms for computing throughput of SADF models. Section 5 illustrates the efficiency of our approach based on several case studies from the multi-media domain. Conclusions and directions for future research are summarised in Section 6.

2 Scenario-Aware Dataflow

Scenario-Aware Dataflow is an extension of Synchronous Dataflow (SDF) [19] (also known as weighted marked graphs in Petri Net theory), which allows analysis of many correctness and performance properties like absence of deadlock and throughput [27, 8]. SADF combines the traditional data-driven behaviour of SDF with state-machine based control behaviour to capture dynamism. Figure 3 shows an illustrative SADF model in the top-left corner. The vertices denote *processes* while the edges are *channels* reflecting (potential) dependencies between those processes. Two types of processes are distinguished. *Kernels* (solid vertices) reflect the data processing part of an application (such as variable length decoding for MPEG-4), whereas *detectors* (dashed vertices) model the control part, responsible for dynamically determining the scenario in which processes operate. The possible orders in which scenarios (ς_1 and ς_2 in Figure 3) occur is captured by state machines. In reality, these state machines coordinate the operation mode based on data-dependent conditions like the type of frame to decode



	Scenario	
Rate	ς_1	ς_2
a	2	0
b	1	0

Process	*Scenario*	E	$\mathbb{P}(E)$
A	ς_1	19	8/11
		51	1/11
		57	2/11
B	ς_1	5	5/19
		17	12/19
		47	2/19
	ς_2	13	1/15
		31	13/15
		63	1/15
D	ς_1	3	4/14
		13	1/14
		29	9/14
	ς_2	7	3/4
		19	1/4

Fig. 3. Example SADF Model

in MPEG-4. SADF abstracts from the actual conditions, taking either a probabilistic or non-deterministic abstraction approach. In case probabilistic information is available on the scenario occurrences, the state machines are (discrete) Markov chains in which case both worst/best-case and average-case analysis becomes possible. Otherwise, the state machines reduce to non-deterministic state machines, which still allows for analysing worst/best-case performance as discussed in [6] for a restricted form of SADF. In this paper, the state machines are considered to be Markov chains in line with [33, 34].

A *token* is a unit of information communicated between processes. Such a token can for instance model a frame, line or pixel. The availability of tokens in the (conceptually unbounded) FIFO buffer corresponding to each channel is shown with a dot, labelled with the number of available tokens. Detectors inform other processes about the scenario to operate in by sending them scenario-valued tokens via *control channels*. Control channels are shown as dashed arrows, while solid arrows denote *data channels* (in which data values of tokens have been abstracted). A *production/consumption rate* refers to the number of tokens produced/consumed by a process via a channel each time it *fires*. Rates of 0 are supported to specify absence of data dependencies. The left-hand table in Figure 3 shows that parameterised rates a and b are 0 for scenario ς_2.

The firing of a kernel k starts when one token has become available on (each of) its control input(s). These token(s) determine the scenario, which in turn fixes the parameterised rates. Subsequently, k waits until a number of tokens equal to the consumption rate becomes available on each data input. Then k starts its data processing behaviour, which takes an amount of time given by a sample drawn from the execution time distribution for the active scenario. The right-hand table in Figure 3 shows the possible execution times E and their occurrence probabilities $\mathbb{P}(E)$ for all processes. The firing of k ends by removing a number of tokens, equal to the consumption rate, from each input and producing a number (equal to the production rate) of tokens to each output.

In case a detector d has no control inputs[1], its firing starts with determining its subscenario by making a transition in the associated Markov chain. For detector D, this Markov chain is depicted in the bottom-left corner of Figure 3, where the state names indicate the subscenario to detect. After establishing the subscenario, firing of

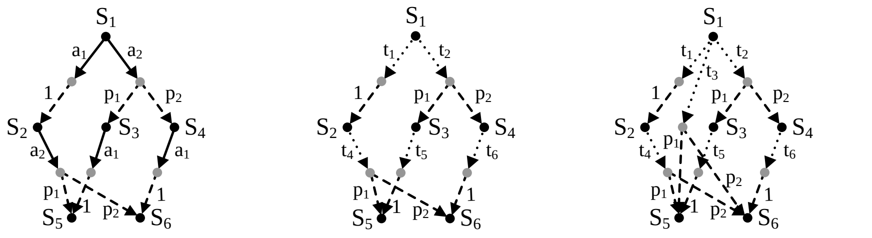
[1] We simplify our explanation in line with [33]. The complete explanation can be found in [34].

d continues similarly as a kernel by fixing the parameterised rates. After sufficient tokens have become available on all data inputs, d performs its behaviour which takes an amount of time drawn from the appropriate execution time distribution. Firing of d ends with removing a number, equal to the consumption rate, of tokens from each input and producing a number (equal to the production rate) of tokens to each output, where the tokens written to control channels are valued with the subscenario. Notice that these valued tokens coherently affect the behaviour of kernels A and B in Figure 3. Succinctly capturing such correlations that often exist between dynamic changes in resource requirements for different processes is a key feature of SADF.

2.1 TPS Semantics

Before we discuss the semantics of SADF, we introduce some notation for TPS. Consider a finite set $\mathbb{S}$ of states and let $\mathcal{D}(\mathbb{S})$ denote the set of probability distributions over $\mathbb{S}$, $\mathcal{D}(\mathbb{S}) = \{\boldsymbol{\pi} : \mathbb{S} \to [0,1] \mid \sum_{S\in\mathbb{S}} \boldsymbol{\pi}(S) = 1\}$. A *Timed Probabilistic Systems* (TPS) with initial state $S^* \in \mathbb{S}$ is a transition system $(\mathbb{S}, S^*, \mathcal{A}, \mathbb{A}, \mathcal{T}, \mathbb{T})$ where $\mathcal{A}$ is a finite set of actions, $\mathcal{T}$ denotes the time domain (e.g., the positive reals or integers), while $\mathbb{A}$ and $\mathbb{T}$ are two sets of labelled transition relations. Set $\mathbb{A}$ is a subset of $\mathbb{S} \times \mathcal{A} \times \mathcal{D}(\mathbb{S})$ and defines the action transitions. Relation $S \xrightarrow{a} \boldsymbol{\pi}$ with $\boldsymbol{\pi} \in \mathcal{D}(\mathbb{S})$ holds if action a can be performed from state S, after which the system transits to state T with probability $\boldsymbol{\pi}(T)$. The set $\mathbb{T}$ is a finite subset of $\mathbb{S} \times \mathcal{T} \times \mathcal{D}(\mathbb{S})$ and denotes the time transitions. Relation $S \xrightarrow{t} \boldsymbol{\pi}$ with $\boldsymbol{\pi} \in \mathcal{D}(\mathbb{S})$ holds if from state S the time can advance for a (positive) amount t, after which the system transits to state T with probability $\boldsymbol{\pi}(T)$. If several action and/or time transition relations hold, the choice of which transition is performed is made non-deterministically. Subsequently, a probabilistic choice determines the target state. The visualisation of TPSs in Figures 1 and 2 shows states in $\mathbb{S}$ as black dots. Action transitions are drawn as solid arrows labelled with an action in $\mathcal{A}$, immediately followed (through a grey dot) by a fan-out of dashed arrows representing a distribution in $\mathcal{D}(\mathbb{S})$. Time transitions are depicted as dotted arrows labelled with a time duration in $\mathcal{T}$ and are similarly followed by a probabilistic fan-out.

TPS can be used for defining the semantics of systems in a compositional way. That is, one defines the semantics of each component as TPS, possibly together with condi-



(a) **Action Determinacy** The probabilities on all of the paths from S_1 to each of the states S_5 and S_6 are equal and both actions a_1 and a_2 are performed

(b) **Time Determinacy** The probabilities on all of the paths from state S_1 to each of the states S_5 and S_6 are equal and also $t_1+t_4 = t_2+t_5 = t_2+t_6$

(c) **Time Additivity** The probabilities on all paths from state S_1 to each of the states S_5 and S_6 are equal and $t_3 = t_1 + t_4 = t_2 + t_5 = t_2 + t_6$

Fig. 4. Some Properties of Timed Probabilistic Systems

tions on actions or advancing time that depend on other components. Parallel composition of the component TPSs resolves such conditions and yields a TPS for the complete system. Figure 4 illustrates a few useful properties [18, 22, 36] that a TPS may satisfy in general. *Action determinacy* (in a non-probabilistic setting also known as the diamond property) is shown in Figure 4(a). It defines that the net behavioural effect of alternative paths of successive action transitions to a common target state is invariant to how non-deterministic choices between those alternative paths are resolved. Hence, only one (arbitrary) path needs to be explored. Given the discrete time model of TPS, one can consider a similar property for time transitions. *Time determinacy* defines that the net effect in advancing time is invariant to non-deterministic choices, see Figure 4(b). A slightly stronger property is *time additivity* or time continuity [36], which specifies that time transitions can be arbitrarily split into smaller transitions or combined into larger ones. Compared to time determinacy, this requires also the existence of a direct time transition to the target state, see Figure 4(c). Another relevant property that a TPS may satisfy in general is *action persistency*, which indicates that advancing time does not disable any actions that could have been performed.

The semantics of SADF in [34] defines six types of action transitions and one type of time transition. We briefly discuss all these transition types. As described above, each process behaves repetitively according to a fixed pattern. Kernels perform *control*, *start* and *end* actions, whereas detectors follow a pattern of *detect*, *start* and *end* actions. The *control* and *detect* actions determine the (sub)scenario in which a kernel respectively detector is going to operate. They fix the corresponding parameterised rates and execution time distribution when sufficient control tokens are available. The *start* actions happen when sufficient data tokens are available, and *end* actions finalise the firing with consuming and producing the appropriate amounts of tokens from inputs respectively onto outputs. Only *detect* and *start* actions may have a probabilistic fan-out with multiple target states – all other transition types, including time transitions, have a single target state. Together, the probabilistic fan-out of *start* transitions and time transitions follow the pattern of Figure 2 to model discrete time distributions like those exemplified in Figure 3. The probabilistic fan-out of *detect* transitions follow the probabilistic choices of the detector's Markov chain(s). Notice that part of the *control*, *detect* and *start* transitions are conditions on the availability of tokens in the FIFO buffers connected to the inputs of processes. Turning our attention to time transitions, we first remark that there is one global notion of time. Time can advance whenever some process has performed its *start* action, with the additional condition that no *end* actions are enabled. Such *end* action becomes enabled when the remaining execution time for a process has reduced to 0 as a result of advancing time. Time advances (at most) with the amount to enable new *end* actions, unless no actions can become enabled anymore (deadlock).

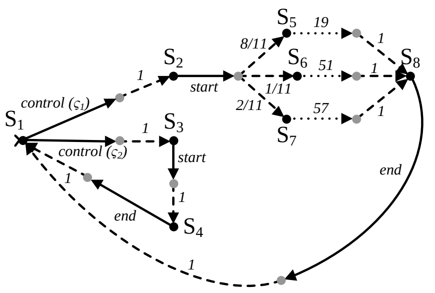
2.2 Semantic Properties of SADF

We can now deduce several important properties of a TPS defined by an SADF model. Our performance model checking approach exploits these to restrain state-space size. To simplify our explanation, we ignore the possibility of deadlocks.

We first discuss time transitions in more detail. For an individual process p, time transitions can only exist after a start action and before the end action. The amount E of advanced time is drawn from an execution time distribution. Observe that the definition of time transitions allows to globally synchronise the advance of time with other processes in a compositional way by being prepared to advance time for p with an arbitrary amount less than E. This requires time additivity, which basically allows splitting time transitions in arbitrarily many smaller time transitions. The composed TPS semantics only explicitly represents the maximum amount of time to pass before an end action is enabled. As a result, at most one time transition can exist from a state and time transitions are always preceded and succeeded by actions. Since time transitions can only be enabled when no process is about to finalise its firing (with an end action), they are succeeded by end actions. Observe that non-determinism between advancing time and performing actions other than end actions is still possible. Consider a state S that is entered after p has performed a start action. A time transition is now enabled for p from all states reachable from S up to states in which some end action is enabled. Although the TPS semantics covers all possible scheduling policies, it is often convenient to assume a specific class of policies that prescribes when actions are to be scheduled for execution. An important class are those policies where actions occur without delays (i.e., all actions are performed before time advances, after which new actions may become enabled again). Such *action urgency* [22] matches with what is known as *self-timed execution* for SDF models [27]. Self-timed execution of SDF models ensures that throughput is maximised [7]. Assuming action urgency may however be disadvantageous for optimising other metrics like latency [9]. Following [33, 34, 6], we adopt the concept of self-timed execution for SADF. Consistently prioritising actions over advancing time is equivalent to extending the original condition of time transitions in Section 2.1 to one where no action transitions are enabled (i.e., instead of just end actions). This excludes the possibility that both action and time transitions can be enabled from a state.

We now turn our attention to the action transitions. As described above, individual processes exhibit only deterministic behaviour[2] as patterns of control/detect, start, time and end transitions. The actions a process p performs only depend on its own state and the buffer status of channels connected directly to p. Advancing time does not disable any enabled action of p. In fact, actions performed by any process other than p do not disable any enabled action of p, which is stronger than action persistency. This further implies the impossibility of having cycles in the TPS part between any two time transitions in case the system is finite. Deterministic choices may exist between control and detect actions due to mutual exclusive conditions on the values of control tokens, while probabilistic choices may arise from execution time distributions and the Markov chains associated with detectors. This is illustrated in Figure 5 for kernel A and detector D from Figure 3. A can perform two control actions from its initial state S_1 depending on the value of the received control token (ς_1 or ς_2). D determines the subscenario (ς_1 or ς_2) based on the Markov chain in Figure 3 which causes the probabilistic fan-out for the detect actions in Figure 5(b). The parallel composition may yield non-determinism in the composed TPS due to different processes interleaving their independent actions. In other words, any non-determinism in the composed TPS originates from concurrency, and hence it is action determinate [33]. This means that non-determinism between actions can be arbitrarily resolved without affecting the net behaviour of the system.

[2] This is not true if the Markov chains reduce to non-deterministic state-machines as in [6].


(a) TPS for Kernel A

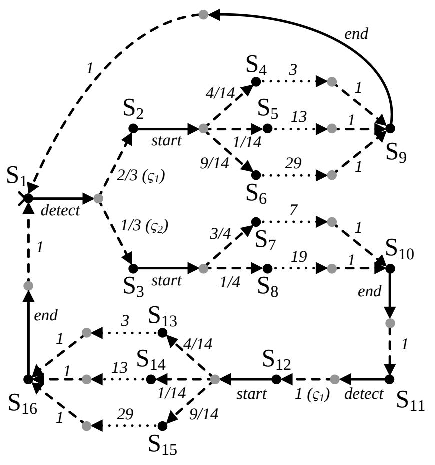

(b) TPS for Detector D

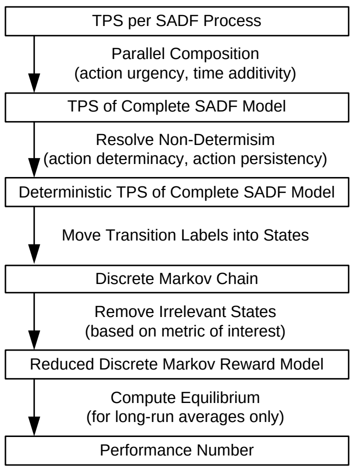
Fig. 5. Semantic Model of Individual Processes (Conditions on Transitions are Omitted)

3 Performance Model Checking

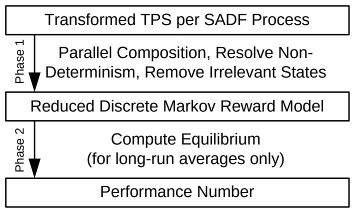
Model checking often exploits properties like action determinacy (diamond property) by means of bisimulation reductions to restrain state-space size. The crux is that redundantly captured behavioural details are removed from the state space. However, such details may be relevant for certain performance metrics [16]. The instantaneous maximum buffer occupancy of SADF channels, for instance, does depend on the order of writing and reading tokens to/from channels. On the other hand, assuming action urgency refers to a class of scheduling policies that (partially) resolves non-determinism. A model checking approach that supports prioritising specific non-deterministic options may avoid constructing the (much larger) state space in which all non-determinism is still available. UPPAAL is an example model checker providing some support for such approach. Finally, certain behaviour may not directly affect a metric, which therefore suggests to consider state spaces that only capture the relevant behaviour. This section presents how our approach exploits these properties to restrain state-space size.

3.1 Strategy for Computing Performance

The proposed strategy, which relies on generic techniques from [35, 31], is visualised in Figure 6(a). The idea is to derive a small, but adequate Markov reward model on which elementary techniques for computing concrete performance numbers can be applied. The first step towards a Markov reward model is to construct a TPS of the complete SADF model by parallel composition of the TPSs of individual processes (such as those in Figure 5). This step takes the guards on transitions of the component TPSs into account and uses the property of time additivity and the assumption of action urgency. The resulting composed TPS may include non-deterministic choices between (concurrent) actions and probabilistic choices (from detect and start actions).


TPS per SADF Process
Parallel Composition
(action urgency, time additivity)
TPS of Complete SADF Model
Resolve Non-Determisim
(action determinacy, action persistency)
Deterministic TPS of Complete SADF Model
Move Transition Labels into States
Discrete Markov Chain
Remove Irrelevant States
(based on metric of interest)
Reduced Discrete Markov Reward Model
Compute Equilibrium
(for long-run averages only)
Performance Number

(a) Conceptual Analysis Process


Transformed TPS per SADF Process
Phase 1
Parallel Composition, Resolve Non-Determinism, Remove Irrelevant States
Reduced Discrete Markov Reward Model
Phase 2
Compute Equilibrium
(for long-run averages only)
Performance Number

(b) Practical Implementation

Fig. 6. Performance Model Checking Strategy

The second step is to resolve any remaining non-determinism. Action persistency and action determinacy ensure that time-dependent performance metrics are not affected by the policy for resolving non-determinism (any of the enabled actions may be selected). In [33], this is demonstrated for time-dependent long-run averages like throughput. To compute the instantaneous maximum buffer occupancy, the reservation of buffer space at the start of the producer must be prioritised over the consumption of tokens at the completion of the consumer [34] (i.e., prioritise all start actions over end actions). The resulting TPS can only have probabilistic choices. We denote this TPS by $(\mathbb{S}', S^*, \mathcal{A}, \mathbb{A}', \mathcal{T}, \mathbb{T}')$, where $\mathbb{S}' \subseteq \mathbb{S}$ is the remaining state space and $\mathbb{A}' \subseteq \mathbb{A}$, $\mathbb{T}' \subseteq \mathbb{T}$ are the remaining action and time transitions *after* resolving non-determinism.

The third step is to construct a discrete[3] Markov chain, which is now possible because at most one action or time transition leaves from any state in $\mathbb{S}'$. Performance metrics can be expressed as some combination of reward functions [30] that are evaluated on the obtained Markov chain. We follow the approach of [35] to move the action/time labels of transitions into their (destination) states similarly as in [21, 23, 3]. The idea is that information on the occurrence of actions and passage of time can be retrieved by reward functions on the states only (i.e., not also on transitions). The state space $\mathcal{S}$ of the Markov chain obtained after the transformation is a subset of $\mathbb{S}' \times (\mathcal{A} \cup \mathcal{T} \cup \{-\})$. The size of $\mathcal{S}$ equals 1 plus the number of transitions with different label/target-state combinations (if $\mathbb{S}'$ is finite) [35]. The interpretation of a state (S, a) in $\mathcal{S}$ is that $S \in \mathbb{S}'$ is entered after having performed action $a \in \mathcal{A}$. State $(S, t) \in \mathcal{S}$ denotes that S is entered after time has advanced with $t \in \mathcal{T}$ time units, while $(S, -) \in \mathcal{S}$ denotes the entrance of S without performing any transition. Observe that only initial state S^* is entered without performing a transition. The one-step transition probabilities of the obtained Markov chain follow straightforwardly from the transitions in $\mathbb{A}'$ and $\mathbb{T}'$.

[3] We deliberately avoid the term discrete-time since there is no relation between the concept of time in TPS and the concept of time in the traditional meaning of discrete-time Markov chains.

The final two steps take the metric of interest into account to compute an exact performance number. We adopt the formalism of *temporal rewards* from [35] to specify metrics. The crux of temporal rewards is that they may not only depend on the current state, like traditional rewards, but also on states visited in the past. We use this ability to express the total amount of time elapsed for a sequence of states, which is an essential component of many time-dependent properties. However, not all states may (directly) contribute to the performance result. We use the state-space reduction technique from [31] to construct a Markov reward model that only includes states in which actions have occurred that *changed* relevant rewards. The actual performance result is then computed from this reduced Markov reward model, after calculating its steady-state or equilibrium distribution in case of long-run average metrics. The next two subsections illustrate how performance metrics can be expressed with temporal rewards and how the reduced Markov reward model is obtained. Section 4 brings the strategy from Figure 6(a) into practice by presenting algorithms performing all steps including constructing the reduced Markov reward model at once, i.e., in an efficient on-the-fly manner.

3.2 Example Performance Metrics

Performance metrics often express properties related to specific actions and/or the advance of time. We illustrate our approach with the more difficult case of long-run averages, since they need to take the equilibrium distribution into account. We aim to exemplify that many long-run averages can be expressed as algebraic combinations of the elementary long-run average [31], which can be evaluated using basic techniques.

Consider the throughput of a process p in an SADF model, which is defined as the long-run average number of firing completions of p per time unit. We use a reward $c : \mathcal{S} \to \{0, 1\}$ to indicate states in which an action has occurred that directly affects the metric of interest. For the throughput example, $c(U) = 1$ in case p has performed an end action in state $U \in \mathcal{S}$, and $c(U) = 0$ otherwise. States for which c evaluates to 1 are called *relevant*, the others are irrelevant. We further define a temporal reward $\varDelta$ that gives the sum of time transition labels $t \in \mathcal{T}$ encountered for a realised sequence of states, which restarts the addition each time a relevant state is visited. In other words, $\varDelta$ denotes the total amount of time elapsed up to visiting the next relevant state.

We now let $\{X_i\}$ denote the stochastic process corresponding to the Markov chain with state space $\mathcal{S}$, where i identifies the i^{th} state visited. The throughput of p can now be expressed as an *event-rate*. For completeness, we give the exact specification in Figure 7, where the limit notation refers to almost sure convergence [2]. It basically divides the number of events (i.e., firing completions of process p) by the average amount of time elapsed between such events. More important is to observe that an event rate resembles the reciprocal of the elementary long-run *sample average* of some reward r, see Figure 7. Many other long-run averages can be expressed as algebraic combinations of sample averages for appropriate definitions of rewards r and c [31]. Examples include the variance in time elapsed between two firing completions of p (a substraction of two sample averages) and the time-weighted average buffer occupancy (a quotient of two sample averages). Hence, given an approach to compute sample averages, many other long-run averages can be computed in a component-wise manner. The next section gives an exact approach to compute sample averages without considering irrelevant states.

$$\lim_{n\to\infty} \frac{\sum_{i=1}^{n} r(X_i)\cdot c(X_i)}{\sum_{i=1}^{n} c(X_i)} \qquad \lim_{n\to\infty} \frac{\sum_{i=2}^{n} c(X_i)}{\sum_{i=2}^{n} \Delta(X_{i-1})\cdot c(X_i)}$$

sample-average event rate

$$\lim_{n\to\infty} \frac{\sum_{i=2}^{n} r(X_{i-1})\cdot \Delta(X_{i-1})\cdot c(X_i)}{\sum_{i=2}^{n} \Delta(X_{i-1})\cdot c(X_i)}$$

time-weighted average

Fig. 7. Example Generic Forms of Long-Run Averages Comprising Similar Terms

3.3 Metric Dependent State-Space Reduction

The relevance of states indicated by c can be exploited to reduce the state space considerably. Consider an ergodic [4, 30] Markov chain $\{X_i\}$ with state space $\mathcal{S}$ for which we need to compute the sample average in Figure 7. By the Ergodic theorem [4, 30], it is not difficult to prove [31] that if $\{X_i\}$ has a relevant positive recurrent state, then

$$\lim_{n\to\infty} \frac{\sum_{i=1}^{n} r(X_i)\cdot c(X_i)}{\sum_{i=1}^{n} c(X_i)} = \frac{\sum_{U\in\mathcal{S}} \pi_U \cdot r(U)\cdot c(U)}{\sum_{U\in\mathcal{S}} \pi_U \cdot c(U)}$$

where π_U is the equilibrium probability of U. Observe that this equation merely depends on rewards earned in relevant states. Hence, it makes sense to investigate whether metrics can be evaluated without considering the irrelevant states at all. Let $\mathcal{S}^c = \{U \in \mathcal{S} \mid c(U) = 1\}$ denote the set of relevant states and define random variable X_i^c as the i^{th} relevant state that is visited (which, for simplicity, we conveniently assume to exist). Now, we can derive some useful theorems, for all of which proofs can be found in [31].

Theorem 1 (Reduction Theorem). *If an ergodic Markov chain $\{X_i\}$ has a relevant positive recurrent state, then $\{X_i^c\}$ is also an ergodic Markov chain.*

Theorem 2 (Preservation of Long-Run Averages). *If an ergodic Markov chain $\{X_i\}$ has a relevant positive recurrent state, then*

$$\frac{\sum_{U\in\mathcal{S}} \pi_U \cdot r(U)\cdot c(U)}{\sum_{U\in\mathcal{S}} \pi_U \cdot c(U)} = \sum_{U\in\mathcal{S}^c} \pi_U^c \cdot r(U)$$

where π^c is the equilibrium distribution of $\{X_i^c\}$.

Hence, the sample average in Figure 7 indeed depends only on rewards r earned in relevant states. Computing it, however, requires equilibrium distribution π^c which relates to the equilibrium distribution π of $\{X_i\}$ as follows.

Theorem 3. *Equilibrium distribution π^c is given by $\pi_U^c = \frac{\pi_U}{\sum_{V\in\mathcal{S}^c} \pi_V}$ for all $U \in \mathcal{S}^c$.*

To avoid storing the complete state space $\mathcal{S}$, we seek a method to determine π^c without first computing π. To present our approach, we consider the matrices of one-step transition probabilities $\mathcal{P}$ and $\mathcal{P}^c$ for the Markov chains $\{X_i\}$ and $\{X_i^c\}$ respectively.

To relate $\mathcal{P}^c$ with $\mathcal{P}$, we introduce matrix M. For any $U \in \mathcal{S}$ and $V \in \mathcal{S}^c$, define $M_{U,V}$ as the probability that $\{X_i\}$ makes a sequence of transitions leading to V when departing from U such that any intermediately visited state is irrelevant. Observe that for $U \in \mathcal{S}^c$, $M_{U,V}$ equals the probability $\mathcal{P}^c_{U,V}$ that $\{X^c_i\}$ transfers from U to V.

Theorem 4. *For all $U \in \mathcal{S}$ and $V \in \mathcal{S}^c$, the elements $M_{U,V}$ of matrix M satisfy the system of linear equations given by $M_{U,V} = \mathcal{P}_{U,V} + \sum_{Q \in \mathcal{S} \setminus \mathcal{S}^c} \mathcal{P}_{U,Q} \cdot M_{Q,V}$ which has a unique solution in case the conditions of the reduction theorem are satisfied.*

Consider the computation of $\mathcal{P}^c_{U,V}$ from a given $U \in \mathcal{S}^c$ to a state V in the set $\mathcal{R} \subseteq \mathcal{S}^c$ of all relevant states reachable from U by intermediately visiting irrelevant states only. By constructing the subgraph of $\{X_i\}$ that departs from U and ends in all states of $\mathcal{R}$, the transition probabilities $\mathcal{P}^c_{U,V}$ for each $V \in \mathcal{R}$ can be computed by solving that part of the equations defining M related to this subgraph. Solving the equations is easy, because, as observed in Section 2.2, there are no cycles between states of an SADF model where time has advanced. Computing $\mathcal{P}^c_{U,V}$ for each V therefore boils down to adding the probabilities on the *finite* number of paths between U and V. The subgraph can now be replaced with transitions for $\{X^c_i\}$ between U and each $V \in \mathcal{R}$ with their corresponding probabilities $\mathcal{P}^c_{U,V}$. In this way, only a part of $\{X_i\}$ is constructed at any moment in time while only temporarily storing irrelevant states.

4 Practical Implementation

With the results of Section 3, we propose to construct the reduced Markov reward model in an on-the-fly way. This is visualised as the first phase of our implemented strategy in Figure 6(b), where the second phase computes π^c and the final performance number. We illustrate our approach with the throughput example of Section 3.2, i.e., the event rate in Figure 7. To this end, we introduce some additional notation. Let $\varOmega_{U,V}$ denote the set of all paths between two states U and V of $\{X_i\}$ without intermediate visits to relevant states. Such a path refers to a realised sequence of states for $\{X_i\}$ from U to V. We use $\mathbb{P}_\rho$ to denote the probability on a specific path $\rho \in \varOmega_{U,V}$, which equals the product of all one-step transition probabilities of $\{X_i\}$ encountered on this path. Observe that if path $\rho \in \varOmega_{U,V}$ consists of a single transition, i.e., V is a direct successor of U, then $\mathbb{P}_\rho = \mathcal{P}_{U,V}$. Moreover, for $V \in \mathcal{S}^c$, we have $\sum_{\rho \in \varOmega_{U,V}} \mathbb{P}_\rho = M_{U,V}$. We also annotate previously defined temporal reward $\varDelta$ as $\varDelta_\rho$ to denote the total amount of time elapsed for a specific path $\rho \in \varOmega_{U,V}$. We can now express the expected total amount of time $\overline{\varDelta}_U$[4] that elapses until visiting the next relevant state when departing from U as follows

$$\overline{\varDelta}_U = \sum_{V \in \mathcal{S}^c} \sum_{\rho \in \varOmega_{U,V}} \mathbb{P}_\rho \cdot \varDelta_\rho$$

Following from Theorem 2 on preservation of long-run averages, we can now derive for the throughput metric in Section 3.2, i.e., the event rate in Figure 7, that

$$\lim_{n \to \infty} \frac{\sum_{i=2}^{n} c(X_i)}{\sum_{i=2}^{n} \varDelta(X_{i-1}) \cdot c(X_i)} = \frac{1}{\sum_{U \in \mathcal{S}^c} \pi^c_U \cdot \overline{\varDelta}_U}$$

[4] As a bonus, we like to mention that $\overline{\varDelta}_U$ equals the expected response time of a process p in case U is initial state $(S^*, -)$ and c identifies states in which p has performed an end action.

The first phase of our practical implementation in Figure 6(b) constructs $\mathcal{S}^c$ and $\mathcal{P}^c$ to enable computing π^c in the second phase. The first phase also computes $\overline{\varDelta}_U$ for all $U \in \mathcal{S}^c$, while the second phase combines the results of all these components into the final performance number.

We present in detail the algorithms for the first phase. It starts from individual TPSs for each process in the SADF model, for which the action/time labels of the transitions have already been shifted into their (destination) states. The initial state $(S^*, -)$ of Markov chain $\{X_i\}$ follows straightforwardly from the initial states of the component TPSs. From $(S^*, -)$, we construct the reduced Markov reward model based on the algorithms in Figure 8. For convenience, we assume that each path $\rho \in \varOmega_{U,V}$ from U to some V is stored as a pair $(\mathbb{P}_\rho, \varDelta_\rho)$ together with the source state U.

Algorithm *PROGRESS* in Figure 8 constructs all paths from some state $U \in \mathcal{S}$ to relevant states $V \in \mathcal{S}^c$ without intermediate visits to relevant states with a depth-first search. The probabilities $\mathbb{P}_\rho$ and durations $\varDelta_\rho$ for each of the possible paths ρ are computed while backtracking. After initialising $\mathcal{I}$ and $\mathcal{R}$ to store the newly discovered irrelevant and relevant states respectively, lines 2 through 9 perform a single enabled transition. Line 2 prioritises actions over advancing time conform to the assumption of action urgency. In line 3, an enabled action $a \in \mathcal{A}$ for any process is selected conform to the policy for resolving non-determinism. In case action a is relevant for the performance metric of interest, then *RSTEP* in line 5 determines the immediate next relevant states R, which are added to $\mathcal{R}$. In case any such R is not yet in $\mathcal{S}^c$, it is also added to $\mathcal{S}^c$. In addition, state U is updated to store the paths $\varOmega_{U,R}$ for all $R \in \mathcal{R}$ as pairs $(\mathcal{P}_{U,R}, 0)$. On the other hand, if a is irrelevant, then *ISTEP* determines in line 6 the next irrelevant states I, which are added to $\mathcal{I}$. Moreover, U is updated with paths $\varOmega_{U,I}$ for all $I \in \mathcal{I}$ as pairs $(\mathcal{P}_{U,I}, 0)$. If in line 3 no actions are enabled but time can advance in line 7 with t units, then *ISTEP* in line 8 adds irrelevant next state I entered after performing the time transition to $\mathcal{I}$ while adding the path to I as the pair $(1, t)$ to U. Line 9 identifies deadlock in case no actions are enabled nor time can advance. Lines 10 through 12 of *PROGRESS* construct the subgraph of $\{X_i\}$ from U until reaching relevant states (which are stored in $\mathcal{R}$). In case a relevant action has been performed, the depth-first search ends. The backtracking procedure in lines 13 through 18 computes the probabilities $\mathbb{P}_\rho$ and durations $\varDelta_\rho$ for all paths $\rho \in \varOmega_{U,V}$ from U to relevant states V, where $\top$ in line 16 denotes a reward that returns the amount of time that has elapsed in each state (which equals 0 if an action was performed). Moreover, the paths of U are properly updated to include only direct paths to the relevant end states V (and thereby discarding all intermediately visited irrelevant states).

Algorithm *CONSTRUCT* in Figure 8 relies on *PROGRESS* to construct the reduced Markov chain $\{X_i^c\}$. After determining the relevant states reachable from a state U, the expected time $\overline{\varDelta}_U$ is computed in line 2. For $U \in \mathcal{S}^c$, the probabilities $\mathcal{P}^c_{U,V}$ for all $V \in \mathcal{R}$ are computed in line 5. The recursive construction in line 6 and 7 completes construction of $\{X_i^c\}$ from all relevant states, while the initial call of *CONSTRUCT*$((S^*, -), \varnothing)$ implements the first phase depicted in Figure 6(b).

The algorithms in Figure 8 can easily be adapted to compute performance metrics expressible as a worst/best-case or as (expected) reachability property. The current implementation in SDF^3 [32] for computing various metrics of SADF models uses par-

```
**Algorithm** *PROGRESS*$(U, \mathcal{S}^c)$
**Input:** A source state $U \in \mathcal{S}$ and the current $\mathcal{S}^c$
**Output:** Set $\mathcal{R} \subseteq \mathcal{S}^c$ of newly discovered relevant states reachable
    from $U$, which are also added to $\mathcal{S}^c$
1.  $\mathcal{I} \leftarrow \varnothing$ and $\mathcal{R} \leftarrow \varnothing$
2.  if actions are enabled
3.    then select an enabled action $a \in \mathcal{A}$
4.        if action $a$ is relevant
5.          then $(\mathcal{R}, U, \mathcal{S}^c) \leftarrow$ RSTEP$(U, a, \mathcal{S}^c)$
6.          else $(\mathcal{I}, U) \leftarrow$ ISTEP$(U, a)$
7.    else if time can advance for $t \in \mathcal{T}$ units
8.          then $(\mathcal{I}, U) \leftarrow$ ISTEP$(U, t)$
9.    else deadlock detected
10. for each irrelevant state $I \in \mathcal{I}$
11.   do $(\mathcal{R}', I, \mathcal{S}^c) \leftarrow$ PROGRESS$(I, \mathcal{S}^c)$
12.      $\mathcal{R} \leftarrow \mathcal{R} \cup \mathcal{R}'$
13. for each irrelevant state $I \in \mathcal{I}$
14.       for each path $\rho \in \Omega_{U,I}$
15.         do for each direct successor state $V$ of $I$
16.                 do add path $(\mathbb{P}_\rho \cdot \mathcal{P}_{I,V}, \Delta_\rho + \top(I))$ to $U$
17. for each irrelevant state $I \in \mathcal{I}$
18.   do remove all paths $\rho \in \Omega_{U,I}$ from $U$
19. return $(\mathcal{R}, U, \mathcal{S}^c)$
```

```
**Algorithm** *CONSTRUCT*$(U, \mathcal{S}^c)$
**Input:** A source state $U \in \mathcal{S}$ and the current $\mathcal{S}^c$
**Output:** Updated $\mathcal{S}^c$, $\mathcal{P}^c$ and $\overline{\Delta}$
1. $(\mathcal{R}, U, \mathcal{S}^c) \leftarrow$ PROGRESS$(U, \mathcal{S}^c)$
2. $\overline{\Delta}_U \leftarrow \sum_{V \in \mathcal{R}} \sum_{\rho \in \Omega_{U,V}} \mathbb{P}_\rho \cdot \Delta_\rho$
3. if $U \in \mathcal{S}^c$
4.   then for each $V \in \mathcal{R}$
5.          do $\mathcal{P}^c_{U,V} \leftarrow \sum_{\rho \in \Omega_{U,V}} \mathbb{P}_\rho$
6. for each $V \in \mathcal{R}$
7.     do $(\mathcal{S}^c, \mathcal{P}^c, \overline{\Delta}) \leftarrow$ CONSTRUCT$(V, \mathcal{S}^c)$
8. return $(\mathcal{S}^c, \mathcal{P}^c, \overline{\Delta})$
```

Fig. 8. On-the-fly Construction of the Reduced Markov Reward Model

tially dedicated variants of the algorithms in Figure 8 for doing so, but it also relies on reusing large parts for common terms in different metric types like those in Figure 7. The on-the-fly construction of the reduced Markov reward model has enabled computing the performance of much larger SADF models compared to directly implementing the strategy of Figure 6(a) as we illustrate for several experiments in the next section.

5 Experimental Results

We demonstrate the applicability of our performance model checking approach by computing the throughput for the dynamic applications in the literature listed in Table 1. The examples are ordered in size of their state spaces. The MPEG-4 SP and MP3 examples show increased state spaces when the amount of concurrency by pipelining degree parameter PD increases. All results in Table 1 are obtained using an Intel Centrino 2 based machine at 2.5Ghz running SDF^3 in a virtual machine with 1.5GB of memory. The entries marked – and † in Table 1 denote that it was infeasible to determine the considered aspect either within the available memory or 6 hours of run-time respectively. The $|\mathbb{S}''|$ column presents the size of the composed TPS $(\mathbb{S}, S^*, \mathcal{A}, \mathbb{A}, \mathcal{T}, \mathbb{T})$, where the transition labels have already been shifted into their destination states. In other words, action urgency and time additivity have been applied for parallel composition of the individual TPSs for each process, but non-determinism as a consequence of concurrency has not yet been resolved. Hence, $|\mathbb{S}''|$ is the size of the primary transition system of our approach without taking specific semantic properties of SADF into account. It is clear that concurrency and dynamism easily make construction of this transition system infeasible. The results show that without the possibility of applying any reductions on-the-fly, computing performance of the MPEG-4 SP and MP3 examples would be infeasible.

The $|\mathcal{S}|$ column indicates the size of the Markov chain $\{X_i\}$ obtained after resolution of non-determinism. Resolving non-determinism clearly reduces the state space

	Reference	*Remark*	$\lvert\mathbb{S}''\rvert$	$\lvert\mathcal{S}\rvert$	*Process*	$\lvert\mathcal{S}^c\rvert$	*Reduction* [%]	*Run-Time* [s]	*Memory* [MB]
MPEG-4 AVC	[25]		185	183	v_4	18	*90.2*	≤ 0.001	0.272
Running Example	Figure 3		661	375	B	11	*97.1*	0.012	0.384
Channel Equalizer	[20]		2185	296	*cf*	8	*97.3*	0.012	0.672
MPEG-4 SP	[33, 32]	PD = 1	–	38440	RC	9	*99.9*	0.8	7.9
MPEG-4 SP	[33, 32]	PD = 2	–	483400	RC	576	*99.9*	40.7	16.3
MPEG-4 SP	[33, 32]	PD = 3	–	–	RC	8253	–	906.9	94
MP3	[34]	PD = 1	–	–	Write	5	–	26.8	64.6
MP3	[34]	PD = 2	–	–	Write	15	–	624.6	165
MP3	[34]	PD = 3	–	–	Write	15	–	20356	275.5
MP3	[34]	$4 \leq$ PD ≤ 9	–	–	Write	†	–	> 21600	†

State-space sizes: $\lvert\mathbb{S}''\rvert$ is the unreduced size, $\lvert\mathcal{S}\rvert$ is after resolving non-determinism and $\lvert\mathcal{S}^c\rvert$ is after complete reduction

Table 1. State-Space Reductions for Throughput Analyses

and even allows storing $\{X_i\}$ for some of the MPEG-4 SP examples. The next three columns present the effect of taking the relevance of actions into account for computing the throughput of the listed processes, which are those processes that determine the application's final output. Column $|\mathcal{S}^c|$ gives the size of the reduced Markov reward model $\{X_i^c\}$, which is the number of states that is finally stored at completing the first phase in Figure 6(b). Observe the considerable relative reductions $\frac{|\mathcal{S}|-|\mathcal{S}^c|}{|\mathcal{S}|} \cdot 100\%$ in the eight column that can still be achieved after resolving non-determinism. As we use an on-the-fly implementation to obtain $\{X_i^c\}$, much bigger examples can be analysed without the need to first completely store the primary transition system or $\{X_i\}$, which is the essence of our approach. The one but last column lists the run-times for both phases in Figure 6(b) together, while the last column shows the peak memory usage. Computing the throughput for the full MP3 example (PD = 9) turns out to be infeasible as the case of PD = 4 for this example already requires more than 6 hours of run-time.

6 Conclusions and Outlook

Using state-of-the-art quantitative model checkers for computing exact performance numbers of SADF models is infeasible due to the underlying time model of generic discrete execution time distributions combined with the diversity of the performance metrics of interest. Inspired by generic model checking techniques, this paper proposes a novel approach that exploits various semantic properties of dataflow models, in this case SADF, to counter state-space explosion. The idea to take the relevance of actions into account that directly affect the metric of interest shows the possibility of substantial state-space reductions after resolving non-determinism originating from concurrency, without affecting the final performance number. We proposed an efficient on-the-fly implementation of our approach that does not require storing the complete state space before applying any of the considered reductions. Despite the effectiveness of the proposed approach, several improvements may still be possible when considering state-of-the-art techniques for other semantic models. Concepts like symbolic state space representations and a more component-wise construction of the relevant state space are just a few aspects to investigate. Other directions for future research are to investigate how the metric-dependent Markov chain based reduction technique can be lifted to a proper bisimulation reduction at TPS level to develop a generic TPS-based performance model checker where users can specify any metric of interest in a temporal reward formula.

References

1. L. de Alfaro. *Formal Verification of Probabilistic Systems*. PhD Thesis. Stanford University, 1997
2. P. Billingsley. *Probability and Measure*. Second edition. Wiley, 1995
3. S. Chaki, E.M. Clarke, J.Ouaknine, N. Sharygina and N. Sinha. State/Event-Based Software Model Checking. *Proceedings of IFM '04*, pp 128–147, 2004
4. K.L. Chung. *Markov Chains with Stationary Transition Probabilities*. Springer-Verlag, 1967.
5. C. Derman. *Finite State Markovian Decision Processes*. Academic Press, 1970.
6. M.C.W. Geilen and S. Stuijk. Worst-case Performance Analysis of Synchronous Dataflow Scenarios. *Proceedings of CODES+ISSS'10*, pp 125–134, 2010
7. R. Govindarajan and G.R. Gao. Rate-Optimal Schedule for Multi-Rate DSP Computations. *Journal of VLSI Signal Processing*, vol 9, pp 211235, 1995
8. A.H. Ghamarian, M.C.W. Geilen, S. Stuijk, T. Basten, A.J.M. Moonen, M.J.G. Bekooij, B.D. Theelen and M.R. Mousavi. Throughput Analysis of Synchronous Data Flow Graphs. *Proceedings of ACSD'06*, pp. 25–34, 2006
9. A.H. Ghamarian, S. Stuijk, T. Basten, M.C.W. Geilen and B.D. Theelen. Latency Minimization for Synchronous Data Flow Graphs. *Proceedings of DSD'07*, pp 189–196, 2007
10. S.V. Gheorghita, S. Stuijk, T. Basten and H. Corporaal. Automatic Scenario Detection for Improved WCET Estimation. *Proceedings of DAC'05*, pp 101–104, 2005
11. S.V. Gheorghita, T. Basten and H. Corporaal. Application Scenarios in Streaming-Oriented Embedded System Design. *Proceeedings of SoC'06*, pp 175–178, 2006
12. H. Garavel, F. Lang, R. Mateescu and W. Serwe. CADP 2010: A Toolbox for the Construction and Analysis of Distributed Processes. *Proceedings of TACAS'11*, LNCS 6605, pp 372–387, 2011
13. H. Hermanns. Interactive Markov Chains and the Quest for Quantified Quality. LNCS 2428, 2002
14. J.-P. Katoen, E.M. Hahn, H. Hermanns, D. Jansen and I. Zapreev. The Ins and Outs of the Probabilistic Model Checker MRMC. *Proceedings of QEST'09*, pp 167–176, 2009
15. M. Kwiatkowska, G. Norman and D. Parker. PRISM 4.0: Verification of Probabilistic Real-time Systems. *Accepted for CAV'11*, 2011
16. M. Kwiatkowska, G. Norman, D. Parker, and J. Sproston. Performance Analysis of Probabilistic Timed Automata using Digital Clocks. *Formal Methods in System Design*, vol 29, pp 33-78, 2006
17. K. Larsen, P. Pettersson and W. Yi. UPPAAL in a Nutshell. *International Journal on Software Tools for Technology Transfer*, vol 1, no 1-2, pp 134–152, 1997
18. K.G. Larsen and W. Yi. Time abstracted bisimulation: Implicit specifications and decidability. *Information and Communication*, vol 134, pp 75-103, 1997
19. E. Lee and D. Messerschmitt. Synchronous Data Flow. *IEEE Proceedings*, vol 75, no 9, pp 1235–1245, 1987
20. A. Moonen, M. Bekooij, R. van den Berg and J. van Meerbergen. Practical and Accurate Throughput Analysis with the Cyclo Static Dataflow Model. *Proceedings of MASCOTS'07*, IEEE, pp 238–245, 2007
21. R. de Nicola and F.W. Vaandrager. Action versus state based logics for transition systems. *Proceedings of LITP'90*, LNCS 469, pp 407–419, 1990
22. X. Nicollin and J. Sifakis. An Overview of Synthesis on Timed Process Algebras. *Proceedings of CAV'91*, pp 376–398, 1991
23. W.D. Obal and W.H. Sanders. State-Space Support for Path-Based Reward Variables. *Performance Evaluation*, vol 35, no 3/4, pp 233–251, 1999
24. M.L. Puterman. *Markov Decesion Processes*. Wiley, 1995

25. P. Poplavko, T. Basten and J. van Meerbergen. Execution-Time Prediction for Dynamic Streaming Applications with Task-Level Parallelism. *Proceedings of DSD'07*, IEEE, pp 228–235, 2007
26. R. Segala. *Modeling and Verification of Randomized Distributed Real-Time Systems*. PhD Thesis. Massachusetts Institute of Technology, 1995
27. S. Siram and S.S. Bhattacharyya. *Embedded Multiprocessors; Scheduling and Synchronization*. Second Edition, CRC Press, 2009
28. A. Sokolova. *Coalgebraic Analysis of Probabilistic Systems*. PhD Thesis. Eindhoven University of Technology, 2005
29. S. Stuijk, M.C.W. Geilen and T. Basten. SDF^3: SDF For Free. *Proceedings of ACSD'06*, pp 276–278, 2006
30. H.C. Tijms. *Stochastic Models; An Algorithmic Approach*. John Wiley & Sons, 1994
31. B.D. Theelen. *Performance Modelling for System-Level Design*. Ph.D. Thesis. Eindhoven University of Technology, 2004
32. B.D. Theelen. A Performance Analysis Tool for Scenario-Aware Streaming Applications. *Proceedings of QEST'07*, 2007
33. B.D. Theelen, M.C.W. Geilen, T. Basten, J.P.M. Voeten, S.V. Gheorghita and S. Stuijk. A Scenario-Aware Data Flow Model for Combined Long-Run Average and Worst-Case Performance Analysis. *Proceedings of MEMOCODE'06*, pp 185–194, 2006
34. B.D. Theelen, M.C.W. Geilen, S. Stuijk, S.V. Gheorghita, T. Basten, J.P.M. Voeten and A.H. Ghamarian. *Scenario-Aware Dataflow*. Technical Report ESR-2008-08, Eindhoven University of Technology, July 2008
35. J.P.M. Voeten. Performance Evaluation with Temporal Rewards. *Performance Evaluation*, vol 50, no 2/3, pp 189–218, 2002
36. W. Yi. Real-time behaviour of asynchronous agents. *Proceedings of CONCUR'90*, LNCS 458, pp 502–520, 1990