Flexible Support for Time and Costs in Scenario-Aware Dataflow

Arnd Hartmanns
University of Twente
Enschede, The Netherlands

Holger Hermanns
Saarland University
Saarbrücken, Germany

Michael Bungert
Saarland University
Saarbrücken, Germany

ABSTRACT

Scenario-aware dataflow is a formalism to model modern dynamic embedded applications whose behaviour is heavily dependent on input data or the operational environment. Key behavioural aspects are the execution times and energy consumption of a system's components. In this paper, we introduce *flexible scenario-aware dataflow*: a proper generalisation of previous definitions that allows any execution time to be specified as discretely or continuously random or nondeterministic. Additionally, it supports the modelling of abstract costs like the energy usage of components. We give a formal compositional semantics in terms of networks of stochastic timed automata. We have implemented support for analysing performance properties of flexible scenario-aware dataflow graphs via simulation and model checking. A number of reduction techniques are applied to make the underlying state spaces tractable for model checking. We evaluate the scalability and performance of our new model and implementation on standard benchmarks.

1. INTRODUCTION

Scenario-aware dataflow (SADF [15]) is an extension of synchronous dataflow (SDF [10]) supporting the description of variations in data processing (e.g. changes in speed, or disabling of components) according to predefined scenarios. It also allows data- and scenario-dependent switching between scenarios. As such, SADF extends the applicability of dataflow formalisms, which have traditionally seen heavy use in digital signal processing applications, to the setting of modern dynamic embedded applications where behaviour may change drastically depending on changes in input data or the operational environment. For example, the processing tasks and times in MPEG video decoding differ significantly between I-, P- and B-frames. Mapping frame types to scenarios would allow this to be encoded naturally in SADF.

Energy usages and execution times of the actors appearing in an SADF graph are key features of the embedded applications considered, with direct impacts on characteristics such as throughput, buffer occupancy, or battery lifetime. So far, SADF has been considered with execution times (or: processing delays) either chosen from discrete, finite-support probability distributions [16], or sampled from the exponential distribution with certain (scenario-dependent) rates [9, 13]. The latter has been called exponentially-timed SADF, or eSADF for short.

In this paper, we generalise both notions to *flexible SADF*, or xSADF for short. This allows not only execution times following arbitrary probability distributions (including discrete ones as used in [16] and continuous ones such as the continuous uniform, normal, or exponential distribution), but also the nondeterministic choice of delays over given (sets of) time intervals with hard bounds, and any combination thereof. We see nondeterminism as a desirable modelling feature to enable abstraction and deliberate underspecification. As such, we enable both probabilistic as well as nondeterministic switching between scenarios. Additionally, we include in xSADF a way to specify abstract costs such as the energy consumed by an actor during idle times, during different processing modes, or for communication as a core feature. This is inspired by [17] and enables new and important kinds of properties to be studied, including expected energy usages or battery lifetime estimations.

To equip xSADF with a formal and hence unambiguous semantics, we need a formalism that supports the necessary combination of quantitative aspects: soft and hard real-time behaviour, discrete and continuous probabilistic decisions, nondeterministic choices, and costs. We find support for all of these aspects in the model of stochastic timed automata (STA [7]). Despite their expressiveness, STA can be seen as a straightforward generalisation of existing, well-known formalisms such as timed automata or Markov decision processes. Our semantics takes advantage of the fact that STA support compositional modelling through a standard parallel composition operator. The analysis of STA by means of model checking is challenging, but first techniques and implementations are available [7].

We have added support for xSADF models to the Modest Toolset [8]: we implemented the STA semantics of xSADF as a model transformation from (an extension of) the standard Sdf 3 [14] format to the toolset's internal metamodel for networks of STA. As a result, it is possible today to create xSADF models in an established format and directly use the Modest Toolset's model-checking and simulation tools for a fully automated analysis.

In this paper, after reviewing the formalisms of STA, SADF and eSADF (Section 2), we formally define flexible SADF

EMSOFT'16, October 01-07 2016, Pittsburgh, PA, USA

ISBN 978-1-4503-4485-2/16/10…$15.00

DOI: http://dx.doi.org/10.1145/2968478.2968496

and its semantics (Section 3). We report on our implementation of analysis support for xSADF (Section 4), including an overview of the optimisations and reductions that we apply in order to make xSADF models tractable for model checking. Using variations of benchmark case studies from the literature, we evaluate the performance and scalability of this implementation (Section 5).

2. PRELIMINARIES

$\mathbb{N}$ is $\{0, 1, \dots\}$, the set of natural numbers. $\mathbb{R}^+$ is the set of positive, $\mathbb{R}_0^+$ the set of non-negative real numbers. Given some set S, we write 2^S to denote its powerset. A discrete *probability distribution* over a set S is a function $\mu \in S \to [0,1]$ such that $\text{support}(\mu) \stackrel{\text{def}}{=} \{s \in S \mid \mu(s) > 0\}$ is countable and $\sum_{s \in \text{support}(\mu)} \mu(s) = 1$. $\text{Dist}(S)$ is the set of all discrete probability distributions over S. We write $\mathcal{D}(s)$ for the *Dirac distribution* for s, defined by $\mathcal{D}(s)(s) = 1$. The basic probabilistic formalisms that we consider are Markov chains and Markov decision processes [12]:

Definition 1. A *Markov decision process* (MDP) is a triple $\langle S, T, s_{init} \rangle$ where S is a finite set of *states*, $T \in S \to 2^{\text{Dist}(S)}$ is the transition function with $|T(s)| > 0$ for all $s \in S$, and $s_{init} \in S$ is the initial state. If $|T(s)| = 1$ for all $s \in S$, then the MDP is a *discrete-time Markov chain* (DTMC).

The transition function maps each state to a set of distributions: it thereby combines the *nondeterministic* selection of a distribution with the *probabilistic* selection of a successor state according to the chosen distribution. We write MDP_S for the set of all MDP with state set S. A DTMC is a "deterministic" MDP and thus a fully stochastic process.

Given a set *Var* of *variables* where each variable x has an associated domain (or type) $\text{Dom}(x)$ and initial value in $\text{Dom}(x)$, we let *Val* denote the set of variable *valuations*, i.e. of functions $Var \to \bigcup_{x \in Var} \text{Dom}(x)$ where $v \in Val \Rightarrow \forall\, x \in Var\colon v(x) \in \text{Dom}(x)$. We consider three classes of abstract *expressions* over variables: arithmetic expressions *Axp* (e.g. $x{+}2$), Boolean expressions *Bxp* (e.g. $x{+}2 \leq 3$), and *sampling expressions* $Sxp \supsetneq Axp$ that may include continuous and discrete probability distributions (e.g. $3 \cdot \textsc{Normal}(x, 1)$) as well as nondeterministic choices (e.g. $x + \textsc{Nondet}(2, 4)$). The set of *updates*, i.e. of sets of assignments to be executed atomically, is $Upd = 2^{Var \times Sxp}$. We write $v := e$ for the assignment $\langle v, e \rangle$. For example, the update

$$\{x := \textsc{Uniform}(1,3), y := x, z := \textsc{Nondet}([1, y])\}$$

assigns to variable x a new value sampled uniformly from the interval $[1, 3]$, to variable y the previous value of x, and to variable z a nondeterministically selected value that is at least 1 and at most the previous value of y. For $x \in Var$ and $\mu \in \text{Dist}(\text{Dom}(x))$, we write $x := \textsc{Sample}(\mu)$ for a probabilistic choice of new value for x according to μ.

2.1 Scenario-Aware Dataflow

We informally recall the formalism of SADF and its semantics using an example graph (based on the example of [16], but using notation similar to [9]) here. We formally define it as a special case of xSADF in Section 3.

The left-hand side of Figure 1 shows our example SADF graph. It consists of the *processes* $\mathbf{A}$, $\mathbf{B}$, $\mathbf{D}$ and $\mathbf{D}'$. Of these, $\mathbf{A}$ and $\mathbf{B}$ are *kernels* while $\mathbf{D}$ and $\mathbf{D}'$ are *detectors*. Processes are connected by *channels*: data channels, drawn with solid lines, carry untyped data tokens from one process to another, while control channels, drawn with dashed lines, carry typed scenario tokens from a detector to another process. We refer to the channel that carries tokens from X to Y as ch_Y^X. For data channels, we specify the initial number of tokens on the channel; in our example, we initially have 2 tokens on $ch_{\mathbf{A}}^{\mathbf{B}}$ and 1 token on $ch_{\mathbf{D}}^{\mathbf{B}}$. Control channels are initially empty. The types of tokens that a control channel ch can carry are given by $\Sigma(ch)$; here, let us use $\Sigma(ch_{\mathbf{D}}^{\mathbf{D}'}) = \{u\}$ and $\Sigma(ch_{\mathbf{A}}^{\mathbf{D}}) = \Sigma(ch_{\mathbf{B}}^{\mathbf{D}}) = \{v, w\}$.

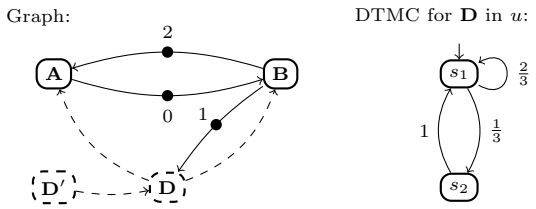

Figure 1: An SADF graph and subscenario DTMC

A kernel processes information: it repeatedly (1) takes one scenario token from each of its incoming control channels, (2) takes a specified number of data tokens from each of its incoming data channels, (3) processes the data for some time, and then (4) adds a number of data tokens to its outgoing data channels. The combination of the types of the scenario tokens consumed in step 1 is the *scenario* that the kernel operates in. The scenario in turn determines the number of data tokens consumed and produced in steps 2 and 4 as well as the execution time incurred in step 3.

A detector is a more powerful kind of kernel in that it also supports output of scenario tokens. Its behaviour is governed by the scenario (which is *externally* determined just as in a kernel) in combination with an *internally* selected *subscenario*: after step 1, one transition is performed in a scenario-specific state machine, with its new state being the current subscenario. SADF uses DTMC for this purpose. The subscenario selection DTMC that $\mathbf{D}$ uses when it is in scenario u is shown on the right-hand side of Figure 1. The set of subscenarios of $\mathbf{D}$ is thus $\{s_1, s_2\}$.

To complete the definition of SADF and our example, we need to specify the numbers of data tokens consumed and produced, the types of scenario tokens produced, and the execution times. For each of these pieces of information and each kernel and detector, we define a function that depends on the current (sub)scenario. For our example, let the production and consumption rates of data tokens be given by

$$\begin{aligned} R_{\mathbf{A}} = R_{\mathbf{B}} = \{&\langle v,\ ch_{\mathbf{B}}^{\mathbf{A}} \rangle \mapsto 1, \langle v,\ ch_{\mathbf{A}}^{\mathbf{B}} \rangle \mapsto 2, \\ &\langle w, ch_{\mathbf{B}}^{\mathbf{A}} \rangle \mapsto 0, \langle w, ch_{\mathbf{A}}^{\mathbf{B}} \rangle \mapsto 0\}, \\ R_{\mathbf{D}} = \{&\langle s_1, ch_{\mathbf{D}}^{\mathbf{B}} \rangle \mapsto 1, \langle s_2, ch_{\mathbf{D}}^{\mathbf{B}} \rangle \mapsto 1\} \end{aligned}$$

while the type of scenario tokens produced by $\mathbf{D}$ is given by

$$\begin{aligned} P_{\mathbf{D}} = \{&\langle s_1, ch_{\mathbf{A}}^{\mathbf{D}} \rangle \mapsto v,\ \langle s_1, ch_{\mathbf{B}}^{\mathbf{D}} \rangle \mapsto v, \\ &\langle s_2, ch_{\mathbf{A}}^{\mathbf{D}} \rangle \mapsto w, \langle s_2, ch_{\mathbf{B}}^{\mathbf{D}} \rangle \mapsto w\}. \end{aligned}$$

Observe how the internal selection of subscenarios in detectors is what allows switching between different scenarios in the first place. To specify the execution times of each kernel or detector, we associate to their possible (sub)scenarios a probability distribution that assigns a probability to a finite

set of possible delays:

$$T^s_{\mathbf{A}} = T^s_{\mathbf{B}} = \{v \mapsto \{1 \mapsto \tfrac{1}{3}, 2 \mapsto \tfrac{1}{3}, 3 \mapsto \tfrac{1}{3}\},\ w \mapsto \mathcal{D}(1)\},$$
$$T^s_{\mathbf{D}} = \{s_1 \mapsto \{0 \mapsto \tfrac{1}{2}, 1 \mapsto \tfrac{1}{4}, 2 \mapsto \tfrac{1}{4}\},\ s_2 \mapsto \mathcal{D}(1)\}$$

In this model, every node thus waits for exactly one time unit (e.g. 1 second) in the "switched-off" scenario w resp. s_2. The execution time in the "active" scenario v follows the discrete uniform distribution over $\{1, 2, 3\}$ time units for **A** and **B**. **D** on the other hand is twice as likely to process its input immediately versus taking either 1 or 2 time units with equal probability.

We have not described the detector $\mathbf{D}'$ so far: its only purpose is to illustrate that detectors can receive scenario tokens that determine the subscenario DTMC, and how this is achieved in the STA semantics of xSADF explained later on.

2.1.1 Measures of interest

Given an SADF model of a system (e.g. of a streaming data processing application supporting video or audio coding), we would like to compute a number of values describing different aspects of the system's performance. Examples that are supported by the SDF 3 [14] tool include, for specified channels or processes,

- buffer occupancy: the number of tokens in a channel,
- response delay: the time until the first processing of data completes in a kernel or detector,
- inter-firing latency: the time between two subsequent completions of data processing, and
- throughput: the number of completions of data processing per time unit.

Depending on the value, we may ask for maximum/minimum values (e.g. the maximum number of tokens ever in a particular channel), probabilities (e.g. the probability of the response delay being lower than t time units), or expected values (e.g. the expected throughput). Using SDF 3 and depending on the type of value, simulation or model checking can be used to compute these values. In contrast to simulation, model checking gives exact results and works well for rare events, but its applicability to large SADF graphs is limited by the state space explosion problem.

2.1.2 Exponentially-timed SADF

If we consider execution times to be sampled from the exponential distribution instead of from finite-support probability distributions, we obtain an eSADF graph. The exponential distribution is parameterised by a rate $\lambda \in \mathbb{R}^+$ resulting in a distribution with mean $\frac{1}{\lambda}$. In eSADF, the rate parameter depends on the current (sub)scenario. If we thus use mappings from (sub)scenarios to rates as execution time functions in our example graph, e.g.

$$T^e_{\mathbf{A}} = T^e_{\mathbf{B}} = \{v \mapsto \tfrac{1}{2},\ w \mapsto 1\},\ T^e_{\mathbf{D}} = \{s_1 \mapsto \tfrac{4}{3},\ s_2 \mapsto 1\}$$

instead of $T^s_{\mathbf{A}}$, $T^s_{\mathbf{B}}$ and $T^s_{\mathbf{D}}$, then our example becomes an eSADF graph. Observe that with this particular choice of rates we have the same *mean* execution times as before, but the actual execution time *distributions* are very different.

The exponential distribution is memoryless. This has enabled the development of efficient analysis methods for continuous-time stochastic models that use but exponentially-distributed delays [2]. The semantics of eSADF is defined in terms of such a model, and as a result eSADF graphs can be model-checked efficiently [9].

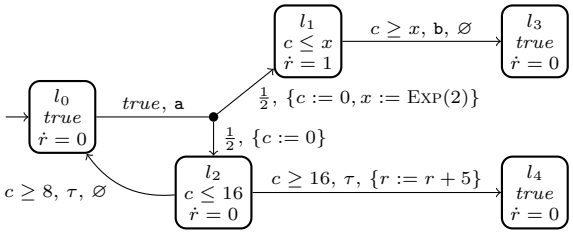

Figure 2: A stochastic timed automaton [7]

2.2 Stochastic Timed Automata

Stochastic timed automata [7] support nondeterministic decisions, real-time behaviour, continuous and discrete probabilistic selections, and any combination thereof. Being a generalisation of timed automata, they deal with time via *clock variables* (or *clocks*). Clocks take values in $\mathbb{R}^+_0$ and advance synchronously over time with rate 1. We restrict expressions in *Bxp* to be *clock constraints*: they must be of the form $e \mid b_1 \wedge b_2 \mid b_1 \vee b_2 \mid c \geq n \mid c \leq n$ where the c are clocks, $b_1, b_2 \in Bxp$ are clock constraints, and $n \in Axp$ as well as $e \in Bxp$ are clock-free expressions without restrictions. We do not allow clocks in expressions in *Axp* or *Sxp*.

Definition 2. A *stochastic timed automaton* (STA) is a 7-tuple $\langle Loc, Var, A, E, l_{init}, Inv, Rew\rangle$ where *Loc* is a finite set of locations, $Var \supseteq CV \uplus RV$ is a finite set of variables with subsets CV of clocks and RV of *rewards*, $A \supseteq \{\tau\}$ is the finite alphabet, $E \in Loc \to 2^{Bxp \times A \times \text{Dist}(Upd \times Loc)}$ is the edge function, $l_{init} \in Loc$ is the initial location, $Inv \in Loc \to Bxp$ is the invariant function, and $Rew \in Loc \times RV \to Axp$ is the rate reward function. The edge function maps locations to finite sets of edges, which in turn consist of a guard, a label and a finite-support probability distribution over updates and target locations, with the restriction that assignments involving a clock (a reward) must be of the form $c := n$ ($r := r + n$) with $n \in Axp$ reward-free. If $\langle g, a, \mu\rangle \in E(l)$, we write $l \xrightarrow{g,a} \mu$. A location's invariant is a clock constraint that allows time to pass as long as it evaluates to *true*.

MDP are untimed STA without continuous distributions. Given an MDP $M = \langle S, T, s_{init}\rangle$, we define $STA(M, act, v)$ as the STA $\langle S, \varnothing, \{\tau, act\}, E^v_T, \{s \mapsto true \mid s \in S\}, \varnothing\rangle$ where $\mu \in T(s) \Leftrightarrow \langle true, act, \mu^v_E\rangle \in E^v_T(s)$ and μ^v_E is defined by $\mu(s) = \mu^v_E(\langle\{v := s\}, s\rangle)$. An example is shown in Figure 5. An STA in which no update contains a continuous probability distribution is a probabilistic timed automaton (PTA [11]). STA can also be seen as generalised semi-Markov processes (GSMP) extended with discrete and continuous nondeterminism.

We use the example STA shown in Figure 2 to convey an intuition of the semantics of STA and refer the interested reader to [7] for a formal definition. This STA has locations l_0 through l_4 and three variables c, x and r, where c is a *clock* and r a reward. The invariant is given on the second line, the rate reward on the last line of each location. The invariants specify when time is allowed to advance; so in location l_2, time can progress until clock c reaches the value 16, at which point l_2 has to be left via an edge before time can pass again. If that were impossible, a timelock would occur. When time is spent in some location, the values of all rewards increase at the given rate. Guards specify when an edge is enabled, while the action labels are used in parallel composition (see

below). The only edge out of l_0 has guard *true* and is labelled with action `a`. It leads to l_1 or l_2 with probability $\frac{1}{2}$ each. In both cases, clock c is reset to zero, and when we go to l_1, the real-valued variable x is updated with a randomly selected value according to the exponential distribution with rate 2. When an edge has a single target location, we omit the branching in the graphical representation of the STA; likewise, we may omit *true* guards.

In our example STA, we have several types of delays and decisions: The choice of target location when leaving l_0 is *probabilistic*. When we arrive in l_2, the edge back to l_0 can be taken after a delay *nondeterministically* chosen between 8 and 16 time units. If we choose to wait for the full 16 time units, there is an additional (discrete) *nondeterministic* choice of going to l_4 instead. When we go from l_0 to l_1, the update of x combined with the way x is subsequently used in the invariant of l_1 and the guard of the edge to l_2 implies that the amount of time spent in l_1 follows the exponential distribution with rate 2, i.e. it is a *stochastic* delay.

Our example contains both rate rewards (of rate 1 in l_1) and edge rewards (of 5 from l_2 to l_4). Rewards are used to observe the accumulated effect of staying in a certain state (accumulating reward over time) or of performing a certain action (incurring an immediate reward). An orthogonal feature that can be added to any automata-based model, they are also referred to as *costs* or *prices*. They see prominent use in verification with priced timed automata [1, 3] as well as in AI and planning in discounted form with MDP [12].

2.2.1 Parallel composition

Given two STA, we can define their parallel composition using a standard interleaving semantics. We use a CSP-like parallel composition operator $\|$ that forces edges with the same action label to synchronise. In particular, in the parallel composition $M_1 \| M_2$, if STA M_1 wants to take an edge labelled `a` and M_2 also has an edge with the same label at some point, then M_1 is forced to either wait until M_2 is also ready to take an edge labelled `a`, resulting in synchronisation, or to take an edge with a different label if possible. We also allow variables to be shared between STA that run in parallel. We call a given set of STA $\{M_1, \ldots, M_n\}$ a *network of STA* and identify it with the parallel composition $M_1 \| \ldots \| M_n$. Again, for a formal definition of STA parallel composition, we refer the interested reader to [7].

2.2.2 Model checking for STA

STA are very expressive, and a model checking technique for STA has been developed only recently [7]. It works by first replacing sampling from continuous probability distributions by a discrete probabilistic selection of an interval from the distribution's support according to the intervals' probability masses. Then, a concrete value from the chosen interval is selected nondeterministically. We could e.g. replace the assignment $x := \textsc{Exp}(\lambda)$ by

$$[x_l, x_u] := \textsc{Sample}(\{[0,1] \mapsto 1 - \mathrm{e}^{-\lambda}, [1,\infty) \mapsto \mathrm{e}^{-\lambda}\})$$

followed by $x := \textsc{Nondet}([x_l, x_u])$. The result is a PTA, for which several model checking techniques are available [11]. The Modest Toolset [8] uses the digital clocks technique, where clocks are replaced by bounded integer variables, resulting in an MDP that is analysed using e.g. value iteration.

While the PTA model checking step is exact, the replacement of the continuous distributions is an abstraction, which by construction is safe. The overall STA model checking approach thus delivers upper/lower bounds on maximum/minimum reachability probabilities (i.e. answers for queries of the type "what is the max./min. probability to eventually reach a certain set of states") and on expected accumulated rewards (i.e. answers for queries such as "what is the max. expected amount of energy consumed within the first t time units"). The maximisation/minimisation is due to the need to range over all resolutions of nondeterminism.

3. FLEXIBLE SADF

We generalise SADF to support costs (like energy usage) plus execution times selected nondeterministically or based on arbitrary discrete or continuous probability distributions. We call the result *flexible SADF*, or xSADF for short.

3.1 Syntax of xSADF

Building on the intuitive explanation of SADF from the previous section, we now formally define xSADF graphs:

Definition 3. An *xSADF graph* is a 5-tuple $\langle \mathcal{P}, \mathcal{C}, I, Sct, \Sigma \rangle$ where $\mathcal{P} = \mathcal{K} \uplus \mathcal{D}$ is a finite set of *processes*, partitioned into kernels $\mathcal{K}$ and detectors $\mathcal{D}$, $\mathcal{C} = \mathcal{DC} \uplus \mathcal{CC}$ is a finite set of *channels*, partitioned into *data channels* $\mathcal{DC}$ and *control channels* $\mathcal{CC}$, $I \in \mathcal{DC} \to \mathbb{N}$ specifies the initial number of tokens in data channels, $\Sigma \in \mathcal{CC} \to 2^{ST}$ maps control channels to a subset of the finite set Sct of *scenario tokens*, and the association between processes and channels is well-defined.

We assume some total order on control channels, and that all sets of control channels appearing are ordered. This allows us to properly define scenarios as the combinations of the scenarios tokens taken from a set of control channels:

Definition 4. The set of *scenarios* for an (ordered) set $C = \{cc_1, \ldots, cc_k\} \subseteq \mathcal{CC}$ is $\Sigma_C \stackrel{\text{def}}{=} \prod_{i=1}^{k} \Sigma(cc_i)$.

A kernel consists of incoming control channels, data channels, and its behaviour is determined by its scenarios:

Definition 5. A *kernel* $K \in \mathcal{K}$ is a 6-tuple

$$K = \langle \mathcal{CC}_K, \mathcal{DC}_K, R_K, U^{\mathrm{i}}_K, T_K, U^{\mathrm{r}}_K \rangle$$

where $\mathcal{CC}_K = \mathcal{CC}^{in}_K \subseteq \mathcal{CC}$ are the incoming control channels, $\mathcal{DC}_K = \mathcal{DC}^{in}_K \cup \mathcal{DC}^{out}_K \subseteq \mathcal{DC}$ are the data channels, with incoming channels in $\mathcal{DC}^{in}_K$ and outgoing channels in $\mathcal{DC}^{out}_K$, $R_K \in \Sigma_{\mathcal{CC}_K} \times \mathcal{DC}_K \to \mathbb{N}$ is the token rate function that maps each data channel to the number of tokens produced or consumed at once in the given scenario, $U^{\mathrm{i}}_K \in \Sigma_{\mathcal{CC}_K} \times \mathcal{DC}_K \cup \mathcal{CC}_K \to \mathbb{R}^+_0$ is the immediate cost of producing or consuming one token, $T_K \in \Sigma_{\mathcal{CC}_K} \to Sxp$ is the execution time function, and $U^{\mathrm{r}}_K \in \Sigma_{\mathcal{CC}_K} \cup \{\bot\} \to \mathbb{R}^+_0$ is the cost rate function that determines the cost per time unit of processing data in a scenario or of being in the idle state $\bot$.

We see that the differences between SADF (or eSADF) and xSADF are that (a) the execution time functions T now map to expressions in Sxp, which may include continuous stochastic or nondeterministic choices, and (b) that immediate and rate costs are tracked via U^{i} and U^{r}.

A detector is a kernel that may also have outgoing control channels. Its behaviour is determined by subscenarios, which are selected by scenario-dependent MDP:

Definition 6. A *detector* $D \in \mathcal{D}$ is a 9-tuple

$$D = \langle \mathcal{CC}_D, \mathcal{DC}_D, \Omega_D, F_D, R_D, U^{\mathrm{i}}_D, T_D, U^{\mathrm{r}}_D, P_D \rangle$$

where $\mathcal{CC}_D = \mathcal{CC}_D^{in} \cup \mathcal{CC}_D^{out} \subseteq \mathcal{CC}$ are the incoming and outgoing control channels, $\mathcal{DC}_D$ is as for kernels, Ω_D is a finite set of *subscenarios*, $F_D \in \Sigma_{\mathcal{CC}_D^{in}} \to MDP_{\Omega_D}$ is the subscenario decision function that maps each scenario to an MDP, $R_D \in \Omega_D \times \mathcal{DC}_D \cup \mathcal{CC}_D^{out} \to \mathbb{N}$ is the token rate function that maps each data channel or outgoing control channel to the *number* of tokens produced or consumed at once in a subscenario, $U_K^{\mathrm{i}} \in (\Sigma_{\mathcal{CC}_D^{in}} \times \mathcal{CC}_D^{in}) \cup (\Omega_D \times \mathcal{DC}_D \cup \mathcal{CC}_D^{out}) \to \mathbb{R}_0^+$ is the immediate cost of producing or consuming one token, $T_D \in \Omega_D \to Sxp$ is the execution time function, $U_D^{\mathrm{r}} \in \Omega_D \cup \{\bot\} \to \mathbb{R}_0^+$ is the cost rate function, and $P_D \in \Omega_D \times \mathcal{CC}_D^{out} \to Sct$ is the scenario token production function with $P_D(\langle s, cc\rangle) \in \Sigma(cc)$ that determines, for every subscenario, the *type* of scenario tokens to produce for each outgoing control channel.

The third difference to SADF (or eSADF) is consequently that (c) the next subscenario is selected by an MDP instead of a DTMC, allowing nondeterministic subscenario selection.

Channels are associated to processes as either incoming or outgoing channels. We require that every data channel connects exactly two processes and that every control channel connects exactly one detector to one process:

Definition 7. In an xSADF graph, the association between processes and channels is *well-defined* if for every $ch \in \mathcal{C}$, there is exactly one $P_{in} \in \mathcal{P}$ and one $P_{out} \in \mathcal{P}$ s.t. either $ch \in \mathcal{CC}_{P_{in}}^{in}$ and $ch \in \mathcal{CC}_{P_{out}}^{out}$ or $ch \in \mathcal{DC}_{P_{in}}^{in}$ and $ch \in \mathcal{DC}_{P_{out}}^{out}$.

With these definitions, it is now easy to see that SADF and eSADF are special cases of xSADF: If the energy rate functions always return zero and (i) the execution time functions always return expressions corresponding to sampling from finite-support distributions, then the graph is an SADF graph. If alternatively (ii) the execution time functions always return expressions of the form $\textsc{Exp}(\lambda)$ with λ depending on the (sub)scenario, then it is an eSADF graph.

3.2 Semantics of xSADF

We define a formal semantics of xSADF graphs in terms of STA, which are a natural match for the continuous stochastic-nondeterministic features that we added. Like the eSADF semantics given in terms of Markov automata [9], our semantics is compositional: every process is mapped to one STA representing its higher-level behaviour, plus one additional STA for each scenario of each detector to implement the corresponding subscenario selection MDP. The overall semantics of the xSADF graph is the parallel composition of the semantics of its components. Communication between the process STA happens via shared variables that represent the channels, while detectors additionally synchronise on shared actions with their subscenario MDP.

3.2.1 Channel semantics

Let $\mathcal{C} = \mathcal{DC} \uplus \mathcal{CC}$ be the set of channels of an xSADF graph as defined above. For each data channel $dc \in \mathcal{DC}$, we include a shared variable dc in the semantics with $\mathrm{Dom}(dc) = \mathbb{N}$ and initial value $I(dc)$. For each control channel $cc \in \mathcal{CC}$, we use a shared variable cc whose type is a queue of scenario tokens that is initially empty. In implementations, we assume a bijection between scenario tokens and integers, and use queues over $\mathbb{N}$ for the control channels. We define three functions on queues q: $\mathsf{hd}(q)$ returns the first element of q, $\mathsf{tl}(q)$ returns a new queue obtained by removing the first element from q, and $\mathsf{enq}(q, t, n)$ returns a new queue obtained by appending $n \in \mathbb{N}$ tokens $t \in Sct$ to the end of q.

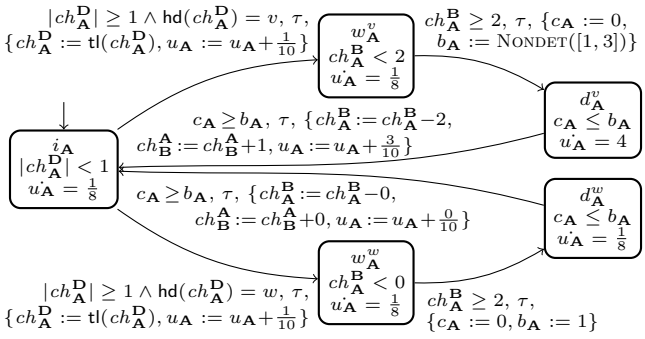

Figure 3: STA semantics of the kernel A

3.2.2 Process semantics

The STA for a kernel K has three different classes of locations: an initial location i_K, and for every scenario σ, one location w_K^σ to wait for data tokens and one "delaying" location d_K^σ. In i_K, the automaton waits for a scenario token to be available on every incoming control channel. As soon as this is the case, it moves (implementing step 1 of Section 2.1) to w_K^σ for the resulting scenario σ. It waits in w_K^σ until at least $R(\langle\sigma, dc\rangle)$ data tokens are available on each data channel $dc \in \mathcal{DC}_K^{in}$, then moves to d_K^σ (step 2). On this edge, clock c_K is reset to zero and real-valued variable b_K is set to an execution time selected via expression $T_K(\sigma)$. The invariant of d_K^σ is $c_K \leq b_K$; combined with the guard $c_K \geq b_K$ of the edge back to i_K from d_K^σ, this implements the execution time delay of step 3. The token production of step 4 is also performed on that edge back to i_K. Formally:

Definition 8. Given a kernel $K \in \mathcal{K}$ as defined above with control channels $\mathcal{CC}_K^{in} = \{cc_1, \ldots, cc_k\}$, its STA semantics is

$$M_K = \langle Loc_K, Var_K, \{\tau\}, E_K, i_K, Inv_K, Rew_K\rangle$$

where $Loc_K = \{i_K\} \cup \{w_K^\sigma, d_K^\sigma \mid \sigma \in \Sigma_{\mathcal{CC}_K^{in}}\}$, $Var_K = \mathcal{CC}_K \cup \mathcal{DC}_K \cup \{c_K, b_K, u_K\}$ with c_K a clock, b_K of type $\mathbb{R}$ and u_K a reward, E_K defines the following edges, for each $\sigma \in \Sigma_{\mathcal{CC}_K^{in}}$:

$$i_K \xrightarrow{gc_\sigma^K, \tau} \mathcal{D}(\langle updi_\sigma^K, w_K^\sigma\rangle)$$

$$w_K^\sigma \xrightarrow{gd_\sigma^K, \tau} \mathcal{D}(\{c_K := 0, b_K := T_K(\sigma)\}, d_K^\sigma)$$

$$d_K^\sigma \xrightarrow{c_K \geq b_K, \tau} \mathcal{D}(updk_\sigma^K, i_K)$$

with $gc_{\langle\sigma_1,\ldots,\sigma_k\rangle}^K \stackrel{\text{def}}{=} \bigwedge_{cc \in \mathcal{CC}_K^{in}} |cc| \geq 1 \wedge \bigwedge_{i=1}^k \mathsf{hd}(cc_i) = \sigma_i$,

$gd_\sigma^K \stackrel{\text{def}}{=} \bigwedge_{dc \in \mathcal{DC}_K^{in}} dc \geq R(\langle\sigma, dc\rangle)$,

$updi_\sigma^K \stackrel{\text{def}}{=} \{cc := \mathsf{tl}(cc) \mid cc \in \mathcal{CC}_K^{in}\}$
$\cup \{u_K := u_K + \sum_{cc \in \mathcal{CC}_K^{in}} U_K^{\mathrm{i}}(\langle\sigma, cc\rangle)\}$, and

$updk_\sigma^K \stackrel{\text{def}}{=} \{dc := dc - R_K(\langle\sigma, dc\rangle) \mid dc \in \mathcal{DC}_K^{in}\}$
$\cup \{dc := dc + R_K(\langle\sigma, dc\rangle) \mid dc \in \mathcal{DC}_K^{out}\}$
$\cup \{u_K := u_K + \sum_{dc \in \mathcal{DC}_K} R_K(\langle\sigma, dc\rangle) \cdot U_K^{\mathrm{i}}(\langle\sigma, dc\rangle)\}$,

the invariant function is defined by

$$Inv_K(l) = \begin{cases} \bigvee_{cc \in \mathcal{CC}_K^{in}} |cc| < 1 & \text{if } l = i_K \\ \neg\, gd_\sigma^K & \text{if } l = w_K^\sigma \\ c_K \leq b_K & \text{if } l = d_K^\sigma, \end{cases}$$

and the rate rewards by $Rew_K(\langle l, u_K\rangle) = U^{\mathrm{r}}(\sigma)$ if $l = d_K^\sigma$ and $Rew_K(\langle l, u_K\rangle) = U^{\mathrm{r}}(\bot)$ otherwise.

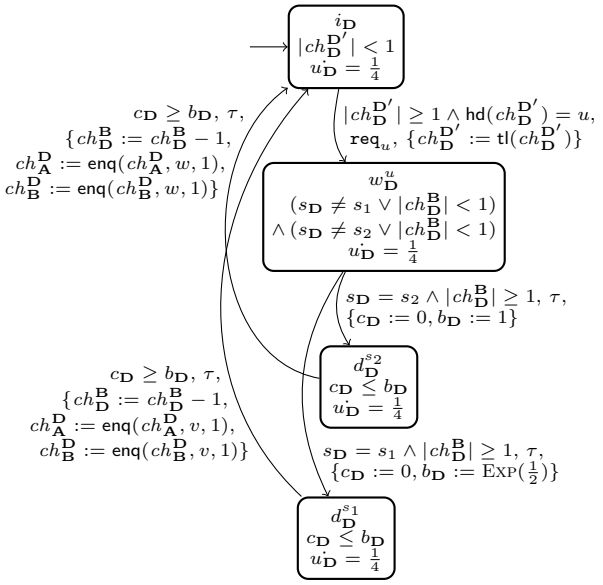

Figure 4: STA for detector D (high-level part)

The reward u_K keeps track of the kernel's immediate and rate costs. We show in Figure 3 the STA for kernel **A** of our example from Section 2.1. For the execution times, we use $T_{\mathbf{A}} = \{v \mapsto \textsc{Nondet}([1,3]), w \mapsto 1\}$, for the cost rates $U^{\mathrm{r}}_{\mathbf{A}} = \{v \mapsto 4, w \mapsto \frac{1}{8}, \bot \mapsto \frac{1}{8}\}$, and U^{i} assigns cost $\frac{1}{10}$ to every token production and consumption in every scenario.

The STA semantics for the high-level behaviour of a detector is very similar. The main difference is that most behaviour is determined by the current subscenario $s_D \in \Omega_D$. The shared variable s_D is updated by the subscenario MDP on synchronisation on action req_σ in scenario σ. Formally:

Definition 9. Given a detector $D \in \mathcal{D}$ as usual with control channels $\mathcal{CC}^{in}_K = \{cc_1, \ldots, cc_n\}$, its STA semantics is

$$M_D = \langle Loc_D, Var_D, A_D, E_D, i_D, Inv_D, Rew_D \rangle$$

where $Loc_D = \{i_D\} \cup \{w^\sigma_D \mid \sigma \in \Sigma_{\mathcal{CC}^{in}_K}\} \cup \{d_\omega \mid \omega \in \Omega_D\}$, $Var_D = \mathcal{CC}_D \cup \mathcal{DC}_D \cup \{c_D, b_D, u_D, s_D\}$ with c_D a clock, b_D of type $\mathbb{R}$, u_D a reward and s_D of type Ω_D, $A_D = \{\tau\} \cup \{\mathtt{req}^D_\sigma \mid \sigma \in \Sigma_{\mathcal{CC}^{in}_K}\}$, E_D defines the following edges, for each $\sigma \in \Sigma_{\mathcal{CC}^{in}_D}$ and $\omega \in \Omega_D$:

$$i_D \xrightarrow{gc^D_\sigma, \mathtt{req}^D_\sigma} \mathcal{D}(\langle updi^D_\sigma, w^\sigma_D \rangle)$$
$$w^\sigma_D \xrightarrow{s_D = \omega \wedge gd^D_\omega, \tau} \mathcal{D}(\{c_D := 0, b_D := T_D(\omega)\}, d^\omega_D)$$
$$d^\omega_D \xrightarrow{c_D \geq b_D, \tau} \mathcal{D}(updd^D_\omega, i_D)$$

with $updd^D_\omega \stackrel{\text{def}}{=} updk^K_\omega$
$\cup \{u_D := u_D + \sum_{cc \in \mathcal{CC}^{out}_D} R_D(\langle \omega, cc \rangle) \cdot U^{\mathrm{i}}_D(\langle \omega, cc \rangle)\}$
$\cup \{cc := \mathsf{enq}(cc, P_D(\langle \omega, cc \rangle), R_D(\langle \omega, cc \rangle)) \mid cc \in \mathcal{CC}^{out}_D\}$,
the invariant function is defined by

$$Inv_D(l) = \begin{cases} \bigvee_{cc \in \mathcal{CC}^{in}_K} |cc| < 1 & \text{if } l = i_K \\ \bigwedge_{\omega \in \Omega_D} \neg (s_D = \omega \wedge gd^D_\omega) & \text{if } l = w^\sigma_D \\ c_D \leq b_D & \text{if } l = d^\omega_D, \end{cases}$$

and the rewards by $Rew_D(\langle l, u_D \rangle) = \begin{cases} U^{\mathrm{r}}(\omega) & \text{if } l = d^\omega_D \\ U^{\mathrm{r}}(\bot) & \text{otherwise.} \end{cases}$

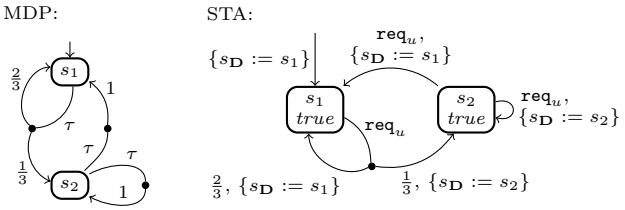

Figure 5: Subscenario MDP and STA semantics

The STA for the high-level behaviour of our example detector **D** is shown in Figure 4. For execution times we use $T_{\mathbf{D}} = \{s_1 \mapsto \textsc{Exp}(\frac{1}{2}), s_2 \mapsto 1\}$, and for cost rates $U^{\mathrm{r}}_{\mathbf{D}} = \{s_1 \mapsto \frac{1}{4}, s_2 \mapsto \frac{1}{4}, \bot \mapsto \frac{1}{4}\}$. In this example, we omit immediate costs for clarity.

3.2.3 *Subscenario MDP semantics*

For each scenario σ, a subscenario selection MDP $F_D(\sigma)$ is associated to a detector D. We include each of these as a separate STA in the semantics. The STA semantics of $F_D(\sigma)$ is simply $M^\sigma_{F_D} = STA(F_D(\sigma), \mathtt{req}^D_\sigma, s_D)$. Let us use the MDP shown on the left-hand side of Figure 5 for **D** in its only scenario u (in place of the DTMC shown in Figure 1). Then its STA semantics is shown on the right-hand side of that same figure. The coordination between the high-level behaviour STA M_K and the subscenario MDP $M^\sigma_{F_D}$ works as follows: we have seen in Definition 9 that, when in location i_D and the first values of the incoming control channels correspond to scenario σ, M_K moves to w^σ_K with an edge labelled with action $\mathtt{req}_\sigma$. Because all edges in $M^\sigma_{F_D}$ are labelled with action $\mathtt{req}_\sigma$, the edge in M_K has to synchronise with one of the edges in $M^\sigma_{F_D}$, corresponding to the transition from one state to another in $M^\sigma_{F_D}$. These states already represent subscenarios, but M_K cannot "read" the state of another STA. The edges of $M^\sigma_{F_D}$ therefore also assign to shared variable s_D the new state, i.e. the new subscenario. As we have seen in Definition 9, the subsequent behaviour of M_K then depends on the value of variable s_D.

3.3 Properties

As described in Section 2.2.2, we can compute (safe bounds on) reachability probabilities and expected accumulated rewards via model checking or simulation, applied to the STA semantics of an xSADF graph. This enables the computation of both the probability distribution over time as well as the expected values at points in time for the measures of interest listed in Section 2.1.1. For example, we can derive

- the maximum and minimum probability that a given channel contains n or more tokens at some point within t time units for $t \in \{0, \ldots, t_{\max}\}$,
- the maximum and minimum expected response delay, or
- the maximum and minimum expected average throughput over the time interval $[0, t]$ for $t \in \{0, \ldots, t_{\max}\}$

where $t_{\max}$ is the time horizon that we are interested in. Queries for max./min. values (not probabilities) as in SDF 3 are not implemented in STA model checkers, albeit being straightforward in theory. On the other hand, the question of what a "steady-state" value in an STA is and how to compute it is an open problem at this time, so steady-state properties of true xSADF graphs cannot yet be computed.

We use rewards u_P to keep track of the cost accumulated by process P. This allows us to additionally compute e.g. the

expected energy consumption of P over time, or the probability that the amount of data generated by all processes together exceeds a threshold before a given point in time.

4. IMPLEMENTATION

We have added support for the analysis of xSADF models to the MODEST TOOLSET [8], a modular set of tools for stochastic timed models. The MODEST TOOLSET is centered around a metamodel for networks of stochastic timed automata. It supports a number of input languages that map to this metamodel as well as several analysis tools, including the model checker MCSTA [7] and the simulator/statistical model checker MODES [4]. Due to its modularity, we were able to add support for xSADF models given in the standard SDF 3 format as an input language in a separate binary module without having to change any other aspect of the toolset. Doing so immediately resulted in xSADF support in all included analysis tools, in particular MCSTA and MODES.

We have extended the XML-based SDF 3 format for SADF to support costs as well as continuous-stochastic and nondeterministic delays. So far, the format allowed the specification of finite-support execution time distributions via sets of weighted "profiles" for each process, for example

```
<profile execution_time="0" weight="2" />
<profile execution_time="1" weight="1" />
<profile execution_time="2" weight="1" />
```

for $T^s_{\mathbf{D}}(s_1) = \{0 \mapsto \frac{1}{2}, 1 \mapsto \frac{1}{4}, 2 \mapsto \frac{1}{4}\}$ of our example from Section 2.1. Our implementation adds support for a concise set of *expressions* as execution times to specify continuous distributions or nondeterministic selections, for example

```
<profile execution_time="NONDET(1, 3) "/>
```

or `<profile execution_time="SAMPLE(EXP(0.5)) "/>`

for $T_{\mathbf{A}}(v) = \text{NONDET}([1,3])$ or $T_{\mathbf{D}}(s_1) = \text{EXP}(\frac{1}{2})$ as in our examples from Section 3.2.2.

As long as a given xSADF graph does not contain non-spurious nondeterminism [4], a simulation-based analysis with MODES is possible. Unaffected by the size of a model's state space, this kind of analysis scales to very large xSADF graphs. However, simulation does not work as soon as true nondeterminism is present, and needs a very large number of runs to obtain precise results, especially in the presence of rare events. We thus focus on the analysis of xSADF graphs using STA model checking in the remainder of this paper.

4.1 Reductions and Optimisations

The key challenge in model checking xSADF graphs is that the state spaces of the underlying MDP semantics (cf. Section 2.2.2) grow very quickly with the number of processes and scenarios, the sizes of the channels, and the complexity of the execution times. In order to be able to analyse realistic-size xSADF graphs within reasonable time and memory limits, we thus implemented a number of optimisations and reductions both on the level of the STA encoding of the graph as well as within MCSTA's MDP analysis engine.

4.1.1 *Efficient queues*

The exploration of the reachable state space, which is a necessity prior to performing value iteration in MDP model checking, was very slow in our first implementation. Profiling showed that this was due to the queue data structure for control channels generated by our xSADF input module. The MODEST TOOLSET's STA metamodel allows the definition of recursive datatypes with functions that operate on them. We first generated a naive linked-list implementation

```
type queue = { int hd, queue option tl }
```

that stored one scenario token `hd` per queue element, leading to deeply recursive representations of long queues and high runtimes for the functions operating on them. We replaced this by a data structure that groups runs of identical tokens

```
type queue = { int token, int count, queue option tl }
```

with `tl` and `enq` functions as described in Section 3.2.1 that maintain a minimal representation. This resulted in a speedup from a few dozen states per second to several tens of thousands of states per second.

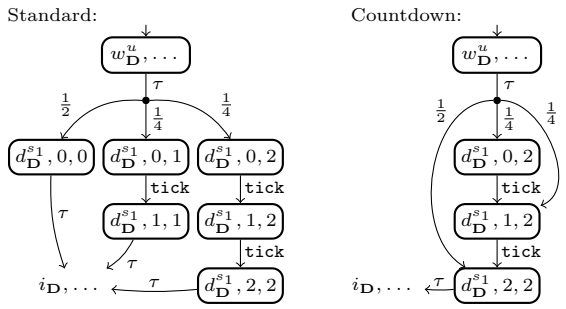

Figure 6: Standard vs. countdown delays

4.1.2 *Active clocks and countdown timers*

As described in Section 2.2.2, MCSTA uses a digital clocks semantics to transform timed models into MDP. This induces a worst-case blowup exponential in the number of clocks and the maximal values that the clocks are compared with. We use two techniques to mitigate this blowup: active clocks reduction and countdown timers.

The former, well known from timed automata verification, resets a clock c to zero and stops incrementing it whenever the system is in a location from which c will not, on any path through the model, be used in a guard or invariant before it is assigned a new value. Since the clock variables in kernels and detectors are only used in the d^{σ}_{P} locations, this leads to noticeably smaller state spaces (cf. Section 5.1). We added this reduction to MCSTA's digital clocks transformation.

The latter is applicable when execution times are selected from a distribution with bounded support (e.g. from the continuous uniform distribution or from finite-support distributions as in SADF). It works by replacing the updates of the form $\{c := 0, b := e\}$ for $e \in \mathit{Sxp}$ with maximum possible value $e_{\max} \in \mathbb{R}^+$ by $\{c := e_{\max} - e, b := e_{\max}\}$ in the semantics of kernels and detectors. Let us illustrate the effect using the execution time distribution $T^s_{\mathbf{D}}(s_1) = \{0 \mapsto \frac{1}{2}, 1 \mapsto \frac{1}{4}, 2 \mapsto \frac{1}{4}\}$ from our example of Section 2.1. The left-hand side of Figure 6 shows the relevant fragment of the digital clocks semantics of the corresponding delay when using the standard semantics. The right-hand side of that figure shows the same state-space fragment when using the countdown timer optimisation. States are labelled with the location that the STA is in, plus the values of variables $c_{\mathbf{D}}$ (first number) and $b_{\mathbf{D}}$ (second number). In particular for models with many finite-support probability distributions for execution times (like SADF graphs), this optimisation induces a dramatic reduction in state-space size. We perform this optimisation inside our xSADF input module.

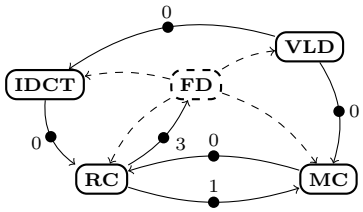

Figure 7: Structure of the `mpeg4_sp` model

red.	states	exploration mem	exploration time	exploration rate	analysis mem	analysis time
–	158.3 M	7.7 GB	718 s	220 k/s	9.4 GB	397 s
ac	48.1 M	2.3 GB	171 s	281 k/s	2.6 GB	64 s
ac ct	7.5 M	1.0 GB	26 s	289 k/s	0.4 GB	15 s
ac ec	6.4 M	0.9 GB	119 s	54 k/s	0.4 GB	7 s
ac ct ec	2.4 M	0.5 GB	47 s	52 k/s	0.2 GB	2 s

Table 1: Impact of reductions (`mpeg4_r`, MC)

4.1.3 *Essential state chain reduction*

The final reduction that we make use of is inspired by the essential states reduction of [5], which collapses those states in an MDP that eventually lead to the same "essential" state with probability 1. We implemented a simpler version inside MCSTA's state-space exploration procedure: if a state s has a single outgoing transition that leads to one successor state s' with probability 1, we replace s by s' and reduce recursively for s', effectively collapsing the chain of two or more states to a single one. For this to be sound, we also require that the transition from s to s' does not change values that are observed in the properties being checked. Although the conditions appear restrictive, the chains of states that they apply to are common in digital clocks MDP. If we were to apply this reduction to the fragments shown in Figure 6, *all* the $d^v_{\mathbf{A}}$-states would be merged with the subsequent $i_{\mathbf{A}}$ states and thus removed. We also implemented the original essential states reduction for comparison. On xSADF models, however, it only removes a small number of additional states compared to the essential state chain reduction as described above, yet incurs a significant runtime overhead.

5. EVALUATION

We applied our implementation of xSADF model checking to variations of the MPEG-4 SP decoder model provided on the SDF 3 website.[1] We first study the impact of the reduction techniques described in the previous section. We then look at how the different types of execution times influence the state-space sizes and analysis performance. Finally, we apply xSADF's new ability to evaluate energy usage as costs.

All experiments were performed on an Intel Core i5 6600T system (2.7 GHz) with 16 GB of memory running 64-bit Windows 10. Where we report the number of states of a model, this refers to the state space of the MDP of the digital clocks semantics of the PTA obtained from the STA semantics (cf. Section 2.2.2). It thus includes four kinds of blowup compared to the size of the compositional semantics itself: the parallel composition is flattened to a single product automaton, each continuous sampling is replaced by a choice over a number of intervals, the possible values of the variables (including the queues for control channels) become part of the states, and finally the digital clocks semantics makes the integer values of all clocks another component of the states. MDP model checking proceeds in two phases: first the *exploration* of the reachable state space, then a numerical *analysis* to compute the value of the property being checked (using value iteration in our implementation). We report performance data for these two phases separately. The size of the reachable state space that needs to be explored, and which we report, depends on the property: goal states can be made absorbing, so that the part of the state space that is entirely "behind" goal states is ignored.

[1] www.es.ele.tue.nl/sadf/examples.php

5.1 Impact of the Reductions

We first study the impact of the various reduction techniques. The full `mpeg4_sp` model consists of four kernels and one detector operating in nine scenarios with a pipelining degree of 3. Its graph structure is shown in Figure 7. The execution times of the kernels are given by finite-support distributions with four different possible times in most cases. For our evaluations, we removed six of the scenarios (the remaining ones being `I`, `P50` and `P90`) and limited the pipelining degree to 1, resulting in the model we call `mpeg4_r`.

We compute the maximum expected response delay of kernel **MC**. The performance results are shown in Table 1. The first column indicates the enabled reductions: active clocks (ac), countdown timers (ct), and essential state chains (ec). We then list the number of reachable states in the underlying MDP (in millions of states), the peak memory usage during state space exploration (column "mem") as well as the total time and rate (in thousands of states per second). For the analysis phase, we list memory usage and total time.

We see that, if we disable all reductions, we need to explore 66 times as many states as when all of them are enabled. The factors are 15 for memory usage and 23 for runtime. Enabling the essential state chains reduction lowers the performance of the exploration phase. However, it yields significant memory savings, and memory is usually the limiting factor in model checking. Even so, it almost makes up for the slower exploration by speeding up the analysis phase.

5.2 Varying the Execution Times

The next point that we are interested in is how the state space sizes depend on the kinds of execution times used. For this purpose, we created five new versions of the `mpeg4_r` model by modifying the execution times of all three scenarios of kernel **VLD**. In the original model, they are determined by a finite-support distribution with mean 16:

$T_{\mathtt{VLD}}(*) = \textsc{Sample}(\{10 \mapsto 0.6, 20 \mapsto 0.25, 30 \mapsto 0.1, 40 \mapsto 0.05\})$.

We study the following variants:

- deterministic: the execution time is always 16,
- nondet.: it is nondeterministic in the interval $[10, 40]$,
- uniform: it is selected from the continuous uniform distribution over $[10, 40]$ (which has mean 25),
- normal: it is sampled from the normal distribution with mean 16 and standard deviation 8, and
- exponential: the execution time is sampled from the exponential distribution with rate $\frac{1}{16}$, i.e. with mean 16.

For each of these models and the original, we compute the *minimum* expected response delay of kernel **IDCT**, reported in column "result" of Table 2. Due to the safe abstraction within the STA model checking algorithm, the computed

type	states	exploration mem	exploration time	analysis time	analysis result
deterministic	559	153 MB	0 s	0 s	27.35
nondet.	1234	153 MB	0 s	0 s	21.35
finite-support	1447	153 MB	0 s	0 s	27.35
uniform	13.5 k	155 MB	0 s	0 s	35.85
normal	2.9 M	957 MB	26 s	2 s	26.81
exponential	4.5 M	1557 MB	42 s	3 s	26.08

Table 2: Impact of execution times (`mpeg4_r`, IDCT)

delays are *lower* bounds on the actual value for the uniform, normal and exponential models. In the nondeterministic model, the analysis selects those execution times within the prescribed intervals that *minimise* the overall result.

Using fixed deterministic times or overapproximating the original behaviour with nondeterminism leads to smaller state spaces, as expected. The uniform distribution causes a moderate blowup, while the two continuous distributions with unbounded support result in significantly larger state spaces. This is due to the need to account for all the intervals generated by the safe abstraction in the analysis. Furthermore, because the distributions' supports are unbounded, we cannot use the countdown timers reduction (which we thus disabled for all models in this comparison).

5.3 Flexible Support for Costs

Aside from full support for nondeterminism and extended expressivity in specifying execution times, xSADF adds facilities to specify (sub)scenario-dependent immediate costs for token production/consumption as well as rate costs accumulated over time when processes are busy or idle. A natural use of this feature annotates the components of an xSADF graph with energy usage information: the cost rates then represent the expectable power needed for processing data or when idling; the immediate costs on token production/consumption can serve as an abstraction of the energy consumed by infrastructure like buses and buffers when transmitting data between components. Many other interpretations of costs are possible, such as counting the number of tokens produced, weighted by token type or channel, etc.

A standard analysis task supported by our implementation is to compute the expected cost at time t. This is solved for STA by adding a fresh clock c that is never reset and computing the expected accumulated value of interest for the goal states identified by $c \geq t$. However, this procedure effectively "unrolls" the state space over time, leading to drastic state-space explosion. To combat this, we reduced our `mpeg4_r` model again by *scaling time*: all execution times were divided by a factor of 5, resulting in the model we call `mpeg4_s`. Thus, to obtain results valid for `mpeg4_r`, the time values of an analysis of `mpeg4_s` need to be multiplied by 5. However, some of the execution times in kernel **IDCT** are not evenly divisible by 5, yet STA model checking requires integer time bounds. We thus replace each execution time x in **IDCT** by $\textsc{Nondet}([\lfloor \frac{x}{5} \rfloor, \lceil \frac{x}{5} \rceil])$. In this way, the minimum (maximum) expected cost computed on `mpeg4_s` times 5 is a lower (upper) bound on the minimum (maximum) expected cost in `mpeg4_r`. In a formalism without nondeterminism (such as the original SADF definition), the only alternative would be simple rounding, which would not guarantee any relationship between the computed costs of the two models.

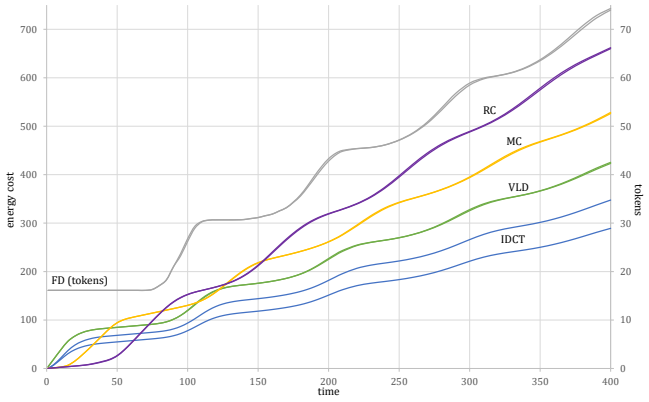

Figure 8: Expected costs over time (`mpeg4_s`)

For our study, we have set the energy usage rates (in e.g. watts) for all kernels to $\frac{1}{4}$ when idle and 2 (4) when processing an I-frame (P-frame). Producing a data token incurs an energy cost of 1 for communication overhead. In practice, the power specification for the hardware at hand should of course be used. Detector **FD** may be seen as a modelling artifact that represents the stream of incoming data but does not correspond to a piece of hard- or software in itself. We thus set its energy usage to 0, but still associate a cost of 1 to every (scenario) token it produces. This allows us to also study the amount of data being processed into the decoder.

Figure 8 displays the accumulated energy consumption of the kernels over time together with the expected number of tokens generated by **FD**. We plot the max. and min. values (shown as pairs of lines of the same color) at time points $t \in \{5, 10, \ldots, 195, 200, 210, \ldots, 400\}$. The differences between max. and min. are due to the nondeterminism introduced. This has a direct effect on the behaviour of **IDCT**, but takes long to propagate visibly into the behaviour of the other components. Furthermore, we clearly see the effect of cyclic system behaviour as data is processed frame by frame, and we see an order in the kernels' active periods. Notably, we compute and display expected values (means) instead of displaying single concrete runs of the system. Therefore the results average out over time as the processes probabilistically move into different scenarios of different durations.

6. CONCLUSION

We have presented *flexible SADF*, called xSADF, supporting scenario-dependent cost annotations and allowing general stochastic and nondeterministic processing delays. It is a proper extension of both the discrete probabilistically-timed SADF of Theelen et al. [16] and the exponentially-timed eSADF of Katoen and Wu [9]. We have presented a compositional semantics of xSADF in terms of stochastic timed automata (STA), which in turn can be analysed using recently developed model checking techniques for STA.

Table 3 gives an overview of the expressivity of our contribution relative to its predecessors. The great flexibility of xSADF enables a more accurate and adequate modelling of scenarios and subscenarios, especially with respect to their impact on costs. This makes it possible to analyse new quantities, such as expected accumulated energy consumption. But of course, the added expressiveness comes at the price of a more involved analysis. We have focussed on the

feature		SADF	eSADF	xSADF
exec. times	deterministic	✓	–	✓
	finite-support	✓	–	✓
	exponential	–	✓	✓
	generally distr.	–	–	✓
	nondeterministic	–	–	✓
subscen. selection	probabilistic	✓	✓	✓
	nondeterministic	–	–	✓
costs (e.g. energy usage)		–	–	✓

Table 3: Expressiveness of xSADF

STA model checking approach, and have described several optimisations that enable effective analysis. We have not discussed the simulation-based (or statistical model checking) analysis of xSADF, albeit this being another avenue for models with only spurious nondeterminism.

We have provided experimental evidence that the different optimisations can reduce model size and solution time by an order of several magnitudes. We have also demonstrated the possibility to study the impact of different assumptions on the execution time in an otherwise unchanged model, something that was not possible in the predecessor works, which are tailored to a single assumption (either finite support or exponential). Finally, we have used the MODEST TOOLSET implementation as a vehicle to derive the accumulated energy consumption as a function of advancing time. Notably, it is straightforward to associate other interpretations (such as costs in $) to the cost modelling mechanisms we support.

PTA and MDP model checking are active fields of research. Improvements in these fields directly lead to improvements in (STA and thus in) xSADF model checking. This is the benefit of using an STA semantics.

Future work.

The modelling of costs in xSADF can be further refined; we could e.g. attach per-token rate costs to channels to keep track of the energy used while data is buffered. Costs can currently be observed in properties during the analysis, but this information is not available to the processes during their execution within the model itself. Allowing them to observe and react to cost information, such as the current energy consumption rate or the available energy supply, could make it possible to naturally represent power management schemes like dynamic voltage and frequency scaling (DVFS) inside an xSADF model and analyse their effects. Of particular interest would be the inclusion of battery models, which would allow a detector to, for example, switch off components when the battery runs low. Adding features like these means that the semantics becomes a network of stochastic hybrid automata (SHA [6]). SHA are well-understood in theory but model-checking tools are still in a prototypical stage and limited to very small state spaces.

Acknowledgments.

This work is supported by the 3TU project "Big Software on the Run", the ERC Advanced Grant 695614 (POWVER), the EU 7th Framework Programme under grant agreement no. 318490 (SENSATION), and the CDZ project 1023 (CAP).

7. REFERENCES

[1] R. Alur, S. La Torre, and G. J. Pappas. Optimal paths in weighted timed automata. In *HSCC*, volume 2034 of *LNCS*, pages 49–62. Springer, 2001.

[2] C. Baier, B. R. Haverkort, H. Hermanns, and J.-P. Katoen. Model-checking algorithms for continuous-time Markov chains. *IEEE Transactions on Software Engineering*, 29(6):524–541, 2003.

[3] G. Behrmann, A. Fehnker, T. Hune, K. G. Larsen, P. Pettersson, J. Romijn, and F. W. Vaandrager. Minimum-cost reachability for priced timed automata. In *HSCC*, volume 2034 of *LNCS*, pages 147–161. Springer, 2001.

[4] J. Bogdoll, A. Hartmanns, and H. Hermanns. Simulation and statistical model checking for Modestly nondeterministic models. In *MMB & DFT*, volume 7201 of *LNCS*, pages 249–252. Springer, 2012.

[5] P. R. D'Argenio, B. Jeannet, H. E. Jensen, and K. G. Larsen. Reduction and refinement strategies for probabilistic analysis. In *PAPM-PROBMIV*, volume 2399 of *LNCS*, pages 57–76. Springer, 2002.

[6] M. Fränzle, E. M. Hahn, H. Hermanns, N. Wolovick, and L. Zhang. Measurability and safety verification for stochastic hybrid systems. In *HSCC*, pages 43–52. ACM, 2011.

[7] E. M. Hahn, A. Hartmanns, and H. Hermanns. Reachability and reward checking for stochastic timed automata. *ECEASST*, 70, November 2014.

[8] A. Hartmanns and H. Hermanns. The Modest Toolset: An integrated environment for quantitative modelling and verification. In *TACAS*, volume 8413 of *LNCS*, pages 593–598. Springer, 2014.

[9] J.-P. Katoen and H. Wu. Exponentially timed SADF: compositional semantics, reductions, and analysis. In *EMSOFT*, pages 1:1–1:10. ACM, 2014.

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

[11] G. Norman, D. Parker, and J. Sproston. Model checking for probabilistic timed automata. *Formal Methods in System Design*, 43(2):164–190, 2013.

[12] M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons Inc., New York, 1994.

[13] S. Sriram and S. S. Bhattacharyya. *Embedded Multiprocessors: Scheduling and Synchronization*. CRC Press, 2nd edition, 2009.

[14] S. Stuijk, M. Geilen, and T. Basten. Sdf^3: SDF for free. In *ACSD*, pages 276–278. IEEE Comp. S., 2006.

[15] 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.

[16] B. D. Theelen, M. Geilen, and J. Voeten. Performance model checking scenario-aware dataflow. In *FORMATS*, volume 6919 of *LNCS*, pages 43–59. Springer, 2011.

[17] J. Zimmermann, O. Bringmann, and W. Rosenstiel. Analysis of multi-domain scenarios for optimized dynamic power management strategies. In *DATE*, pages 862–865. IEEE, 2012.