Compositional Dependability Evaluation for STATEMATE

Eckard Böde, Marc Herbstritt, Holger Hermanns, Sven Johr, Thomas Peikenkamp, Reza Pulungan, Jan Rakow, Ralf Wimmer, *Student Member, IEEE,* Bernd Becker, *Fellow, IEEE,*

***Abstract*— Software and system dependability is getting ever more important in embedded system design. Current industrial practice of model-based analysis is supported by state-transition diagrammatic notations such as Statecharts. State-of-the-art modelling tools like STATEMATE support safety and failure-effect analysis at design time, but restricted to qualitative properties. This paper reports on a (plug-in) extension of STATEMATE enabling the evaluation of quantitative dependability properties at design time. The extension is compositional in the way the model is augmented with probabilistic timing information. This fact is exploited in the construction of the underlying mathematical model, a uniform continuous-time Markov decision process, on which we are able to check requirements of the form: *"The probability to hit a safety-critical system configuration within a mission time of 3 hours is at most 0.01."* We give a detailed explanation of the construction and evaluation steps making this possible, and report on a nontrivial case study of a high-speed train signalling system where the tool has been applied successfully.**

***Index Terms*— Real-time and embedded systems, Fault tolerance, Modelling techniques, Reliability, availability, and serviceability, Model checking, Reliability, Design notations and documentation, State diagrams.**

I. MOTIVATION

ENGINEERS of safety-critical embedded software are facing great challenges. To ensure safe and dependable behaviour of the final system requires careful design-time modelling and analysis. Often behavioural models are developed in the form of (huge) state-transition diagrams of various kinds, which are evaluated using verification and validation tools, for instance model checkers. When it comes to studying performance and dependability of such systems, the industrial practice uses stochastic models such as Markov chains, simulation models, or probabilistic interpretations of fault trees, to estimate system performance and especially failure risks. These latter models are often developed separately from the state-transition models used for studying functional correctness. This is especially problematic if the functional behaviour itself is affected by failures, or is specified to compensate for component failures, like in repairable or fault-resilient system designs.

Manuscript received January 27, 2008; revised September 04, 2008.

This work is supported by the German Research Council (DFG) as part of the Transregional Collaborative Research Center "Automatic Verification and Analysis of Complex Systems" (SFB/TR 14 AVACS). See `www.avacs.org` for more information.

E. Böde and T. Peikenkamp are with OFFIS Institute for Information Technology, Oldenburg, Germany.

M. Herbstritt, R. Wimmer and B. Becker are with Albert-Ludwigs-University Freiburg im Breisgau, Germany.

H. Hermanns, S. Johr and R. Pulungan are with Saarland University, Saarbrücken, Germany.

J. Rakow is with Carl von Ossietzky University, Oldenburg, Germany.

This disturbing gap is amplified by a considerable distance of theoretical advances to the industrial daily practise. Especially the area of probabilistic verification and stochastic model checking has seen great advances in the past years [1]–[4], which seem ready for industrial practise.

Motivated by this observation, we have undertaken efforts to integrate very recent advances in stochastic model checking into a modelling environment with a stable industrial user group. The modelling environment is STATEMATE, a Statechart-based tool-set used in several avionic and automotive companies like AIRBUS or BMW. The model checking is based on computing time bounded reachability probabilities, and allows us to verify properties like: *"The probability to hit a safety-critical system configuration within a mission time of 3 hours is at most 0.01."* The algorithmic workhorse to validate (or refute) such properties is *the first* implementation of an algorithm [4] which computes the worst-case (or best-case) time bounded reachability probability in a *uniform continuous-time Markov decision process* (uCTMDP). This combination of Statechart-modelling and uCTMDP analysis raises theoretical and practical questions, both of which are answered in this paper. On the theoretical side, we describe how the STATEMATE-model can be enriched with real-time probabilistic time aspects, and then transformed into a CTMDP which is uniform by construction. One key feature of this approach is that the model construction steps rely heavily on compositional properties of the intermediate model, which is the model of *interactive Markov chains* (IMCs) [5]. On the practical side, we report how symbolic (i. e. BDD-based) representations and compositional methods can be exploited to keep the model sizes manageable. While the later steps in our analysis trajectory use explicit state space representations, the earlier steps are symbolic, and use a novel and very effective symbolic branching bisimulation minimisation algorithm.

From an engineer's perspective, a typical analysis scenario is shown in Fig. 1, which will serve as a running example: The STATEMATE *design* represents the functional behaviour of a heating system. Owed to its safety-critical nature, the model contains distinguished *safety-critical states* (here `TLE`, top level event). To identify them, techniques like Functional Hazard Analysis (FHA) [6] are often employed, but this is beyond the scope of this paper. The design also comprises distinguished *delay transitions*, which correspond to the bold arrows. These transitions have an effect or are effected by the advance of time. Mostly, delay transitions indicate component failures (`FM`, failure of monitor and `FS`, failure of sensor), but the concept is more flexible, as we will explain later. A typical dependability requirement for such a system demands that the risk of hitting any safety-critical state within a given mission *time bound* is below some threshold. Such requirements refer to a real-time probabilistic interpretation of the model. In our approach this interpretation is based on *delay*

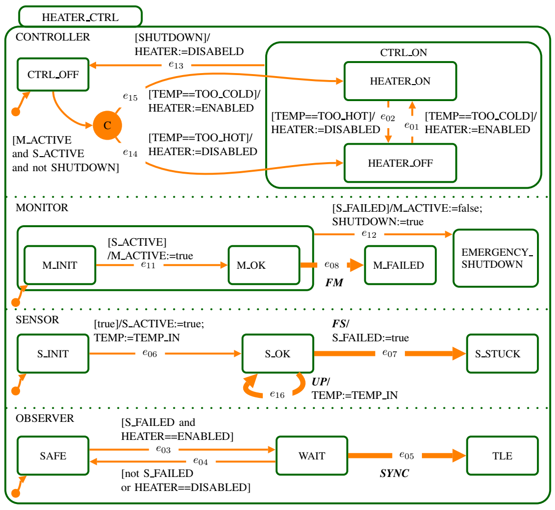

HEATER_CTRL
CONTROLLER
[SHUTDOWN]/ HEATER:=DISABELD
e_{13}
CTRL_ON
CTRL_OFF
HEATER_ON
e_{15}
[TEMP==TOO_COLD]/ HEATER:=ENABLED
C
[TEMP==TOO_HOT]/ HEATER:=DISABLED
e_{02}
e_{01}
[TEMP==TOO_COLD]/ HEATER:=ENABLED
[M_ACTIVE and S_ACTIVE and not SHUTDOWN]
e_{14}
[TEMP==TOO_HOT]/ HEATER:=DISABLED
HEATER_OFF
MONITOR
[S_FAILED]/M_ACTIVE:=false; SHUTDOWN:=true
e_{12}
[S_ACTIVE] /M_ACTIVE:=true
M_INIT
e_{11}
M_OK
e_{08}
FM
M_FAILED
EMERGENCY_ SHUTDOWN
SENSOR
[true]/S_ACTIVE:=true; TEMP:=TEMP_IN
FS/ S_FAILED:=true
S_INIT
e_{06}
S_OK
e_{07}
S_STUCK
e_{16}
UP/ TEMP:=TEMP_IN
OBSERVER
[S_FAILED and HEATER==ENABLED]
SAFE
e_{03}
e_{04}
WAIT
e_{05}
TLE
SYNC
[not S_FAILED or HEATER==DISABLED]

Fig. 1. An extended Statechart: the heater example.

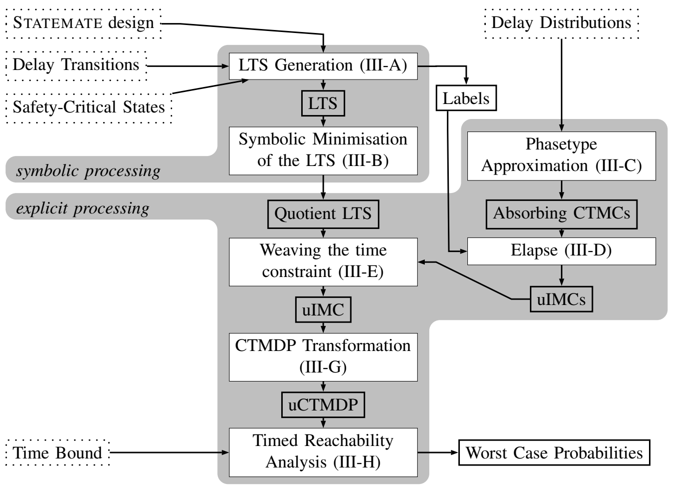


Fig. 2. Overview: processing steps and basic models.

distributions, in the form of continuous probability distributions affecting the occurrence of the delay transitions. In the heater example, this means that the engineer is asked to provide the distribution of the time to failure for the monitor (`FM`) and the sensor (`FS`).

All in all, the STATEMATE *design*, *safety-critical states*, *delay transitions*, *delay distributions*, and a *time bound* make up the ensemble of information needed to verify the above dependability requirement. They are indicated as dashed boxes in Fig. 2. Based on these inputs, our tool computes the *worst-case probability* to reach a safety-critical state within the time bound, thus providing the engineer with the required model-based prediction.

The heater shown in Fig. 1 demonstrates some of the intricacies of typical behavioural models that make them hard to analyse: large state-based systems with parallel activities (separated by dashed lines) that communicate by means of shared variables (e. g. `TEMP`), non-determinism resulting from specification freedom or unknown variables (e. g. `TEMP_IN`), and prioritised transitions are typical features that are present in today's systems designs.

In summary, this journal paper makes the following contributions, based on the conference publications [7]–[10]. We (1) devise a sound and effective methodology to assess dependability of industrial-size safety-critical designs. This is based on (2) the first implementation of a time bounded reachability algorithm for uCTMDPs [7], (3) the first—to our knowledge—entirely BDD-based algorithm for computing branching bisimulation quotients [9], (4) a provably sound compositional method to construct uniform CTMDPs [8], [10], and (5) the integration of these pieces into an effective tool chain [7]. We report (6) on the results of applying this tool chain to a nontrivial case study from the train control domain. For this case study, we manage to avoid state spaces in the order of 10^{40}, and instead only need to handle models of up to 10^5 states and 10^6 transitions.

Organisation of the paper. The paper is organised as follows. Section II introduces the basic mathematical concepts and models that are combined in our tool. The interplay of the various concepts is described in detail in Section III. Section IV demonstrates the practical feasibility of our approach by an example from the train control domain. Section V concludes the paper.

II. Basic Models

This section introduces the behavioural models used in this paper. First, we address foundations related to the stochastic aspects of our compositional approach. We then introduce a variation of the *Statechart* formalism, which we use as the main vehicle to describe functional behavioural models and explain its particularities.

A. Stochastic Models

The construction process revolves around different flavours of *interactive Markov chains* [5], an orthogonal combination of labelled transition systems and continuous-time Markov chains. We consider a basic set of actions A and let $Act = A \,\dot{\cup}\, \{\tau\}$ where τ is a distinguished action not in A. This action is deemed unobservable and plays a crucial role in our approach, since it is used for abstracting behaviours of the system, which at certain stages are irrelevant for the transformation steps that follow.

Definition 1 (IMC)
An *interactive Markov chain* (IMC) is a tuple (S, Act, T, R, s_0) where S is a non-empty set of states, Act is the above set of actions, $T \subseteq S \times Act \times S$ is a set of interactive transitions, $R \subseteq S \times \mathbb{R}^+ \times S$ associates a set of Markov transitions with each state, and $s_0 \in S$ is the initial state.

$R(s, s')$ denotes the transition rate from states s to s', i. e., $R(s, s') = \sum_{\lambda \in \Lambda} \lambda$ where $\Lambda = \{\lambda \mid (s, \lambda, s') \in R\}$. A labelled transition system (LTS) is a tuple (S, Act, T, s_0) whenever $(S, Act, T, \emptyset, s_0)$ is an IMC. A continuous-time Markov chain (CTMC) is a tuple (S, Act, R, s_0) whenever $(S, Act, \emptyset, R, s_0)$ is an IMC.
Notation: For IMC $\mathcal{I} = (S, Act, T, R, s_0)$, a state $s \in S$ is *stable*, written $s \not\xrightarrow{\tau}$, if $\forall s' \in S$: $(s, \tau, s') \notin T$. Otherwise s is called *unstable*. For stable state s and $C \subseteq S$ we define $\mathbf{r}(s, C) = \sum_{s' \in C} R(s, s')$. For unstable s, $\mathbf{r}(s, C) = 0$. The distinction between stable and unstable states is justified by the notion of maximal progress, see [5] for details. IMC $\mathcal{I}$ is called *uniform*, iff $\exists e \in \mathbb{R}^+$ such that $\forall s \in S : s \not\xrightarrow{\tau}$ implies $\mathbf{r}(s, S) = e$. We write $s \xrightarrow{a} t$ for $(s, a, t) \in T$, and $\xrightarrow{\tau^*}$ for the reflexive transitive closure of $\xrightarrow{\tau}$. If $R(s, s') = \lambda > 0$ we will sometimes depict this as $s \overset{\lambda}{\dashrightarrow} s'$.

For an equivalence relation $B \subseteq S \times S$, we let S/B denote the set of equivalence classes of B and $B(s)$ the equivalence class of s. If s and t are contained in the same equivalence class of B, we write $s \equiv_B t$. Furthermore, we write $s \xrightarrow[B]{\tau^*} t$ if $s \xrightarrow{\tau^*} t$ and $s \equiv_B t$. Such transition sequences whose source and target state are contained in the same equivalence class of B are called *inert*. An equivalence relation B is a refinement of an equivalence relation B' (denoted by $B \sqsubseteq B'$) iff $\forall s, t : s \equiv_B t \Rightarrow s \equiv_{B'} t$.

Definition 2 (Stochastic Branching Bisimulation)
For a given IMC $\mathcal{I} = (S, Act, T, R, s_0)$, an equivalence relation $B \subseteq S \times S$ is a *stochastic branching bisimulation* iff for all $s_1, s_2, t_1 \in S$ the following holds: If $s_1 \equiv_B t_1$ then

1) $s_1 \xrightarrow{a} s_2$ implies
either $a = \tau$ and $s_2 \equiv_B t_1$,
or $\exists t_1', t_2 \in S : t_1 \xrightarrow[B]{\tau^*} t_1' \xrightarrow{a} t_2 \wedge s_2 \equiv_B t_2$,
and
2) $s_1 \not\xrightarrow{\tau}$ implies
$\exists t_1' : t_1 \xrightarrow{\tau^*} t_1' \not\xrightarrow{\tau}: \forall C \in S/B : \mathbf{r}(s_1, C) = \mathbf{r}(t_1', C)$.

Two states are *stochastic branching bisimilar*, iff they are contained in some stochastic branching bisimulation B.

This notion is a variant of branching bisimulation [11] and stochastic weak bisimulation [5]. For LTSs, the definition coincides with that of the original branching bisimulation, which we can hence define as follows.

Definition 3
For a given LTS (S, Act, T, s_0), an equivalence relation $B \subseteq S \times S$ is a *branching bisimulation* iff it is a stochastic branching bisimulation on $(S, Act, T, \emptyset, s_0)$. Two states are *branching bisimilar* iff they are contained in some branching bisimulation.

Definition 4 (CTMDP)
A *continuous-time Markov decision process* (CTMDP) is a tuple $(S, L, \mathbf{R}, s_0)$ where S is a non-empty set of states, L is a set of transition labels, $\mathbf{R} \subseteq S \times L \times (S \mapsto \mathbb{R}^+)$ is the set of transitions, and s_0 is the initial state.

Any CTMDP can be viewed as a special IMC in which interactive transitions and Markov transitions occur in a strictly alternating manner. This will be used in the final step of our construction. As in [4], a CTMDP is called *uniform* iff $\exists e \in \mathbb{R}^+$ such that $\forall s \in S$ and $\forall l \in L : (s, l, R) \in \mathbf{R}$ implies $\sum_{s' \in S} R(s') = e$.
Both IMC and CTMDP have inherent non-determinism that has to be resolved by an entity called *scheduler*. A scheduler is a function that determines how to proceed next for a given state s. In state s, it resolves non-determinism by picking a particular enabled action. It does so on the basis of information about the current state and the history of the system evolution. In full generality, schedulers may decide on the basis of the entire history of the system, and may decide using randomisation (i. e., probability distributions over enabled actions). In a timed model, the history of the system may even be a timed one.
Scheduler theory is not in the core scope of this article, but we like to point out that measurability issues quickly arise for schedulers that base their decision on time. We have introduced the class of *measurable* schedulers for which probability measures on paths are guaranteed to exist [12], [13]. For a given IMC $\mathcal{I}$ (respectively CTMDP $\mathcal{C}$) and measurable scheduler D (D') over $\mathcal{I}$ ($\mathcal{C}$) the probability measure on paths is denoted by $Pr_{\mathcal{I},D}$ ($Pr_{\mathcal{C},D'}$).

B. Extended Statecharts

In this section, we introduce the user-visible formalism to specify behavioural models, in the form of *extended Statecharts*, which are based on STATEMATE Statecharts. The language definition presented here focuses on the relevant core needed to clearly expose the syntactical and semantic extensions to the conventional STATEMATE formalism. Before providing a more formal definition, we have a closer look at the behaviour of the heater introduced in Section I.

Example 1 The main parts of the heater are a `CONTROLLER`, a `MONITOR`, a `SENSOR`, and an `OBSERVER`. The `SENSOR` is responsible for measuring the temperature at periodic intervals and makes the data available to the `CONTROLLER`: After initialisation the sensor becomes active and, in regular intervals (triggered by the `UP` action), reads a new temperature from the environment `TEMP_IN`. New temperature values—either `TOO_COLD` or `TOO_HOT`—are chosen non-deterministically from this environment and stored in the local variable `TEMP` (from where they will be read by the `CONTROLLER`). A failure (triggered by the action `FS`) can cause the sensor to become inoperational and therefore prevents further updates of the temperature value. The bold arrow indicates here that the failure `FS` occurs after some unknown time (of which we will later provide its distribution function). The `CONTROLLER` initially waits for both the sensor and the monitor to become active. After that, depending on the current temperature stored in `TEMP`, the heater is switched on or off by setting the control signal `HEATER` to `ENABLED` or `DISABLED`. In case the monitor detects a sensor failure, the system shuts down, in order to prevent critical situations in which the heater may overheat. This might happen if it continues to operate even though the current actual temperature `TEMP_IN` is already `TOO_HOT`, but this is not sensed by the failed sensor. The purpose of the `MONITOR` is to check for failures in the sensor and to send a shutdown signal to the controller if one is detected. After activation of both the sensor and the monitor any sensor failure will be detected and lead to a safe shutdown of the system. Note that, if a failure in the monitor occurs prior to a failure in the sensor (by transition to `M_FAILED`), there is still a possibility of reaching a safety-critical situation. The `OBSERVER` is not an actual part of the system but rather a means of specifying requirements in the analysis framework[1]. It is used here to observe whether a safety-critical system state (we also refer to this as top-level event, `TLE`) has been reached. This is the case if the sensor has failed (thus preventing the system to update the temperature) while the heater is still on. Internal switching times of the heater are very small compared to occurrence times of other events (e. g. occurrence of failures `FS` or `FM`). This means that in case of a `SENSOR` failure the propagation time for passing the failure from the `SENSOR` via the `MONITOR` to the `CONTROLLER` are negligible. It is thus unrealistic to move from the `WAIT` state to the unsafe `TLE` state in this situation. In our modelling, this is achieved by making the respective transition `SYNC` a *delay transition*, indicated by the bold arrow.

Conventional Statecharts [14], [15] support non-determinism, but no stochastic time aspects. In order to enable the integration of stochastic durations in this modelling formalism, extended Statecharts allow one to refer to particular Statechart transitions by a distinguished set of action labels A. Such labelled transitions, called *delay transitions*, are later used as reference anchors to start, stop, or interrupt the advance of time, in combination with a set of stochastic time-constraints, and as a whole allow us to model stochastic, non-deterministic systems. Statecharts have an intuitive graphical syntax with a corresponding textual syntax, which we vary as follows.

[1]In a production environment one would expect to keep such requirements separately from the model.

Definition 5 (Extended Statecharts)
An *extended Statechart* $SC = (N, A, V, G, S, E, m, r, d, c)$ is a 10-tuple, with

- N is a finite set of nodes,
- A a finite set of action labels,
- V a finite set of variables with a (possibly empty) subset I of input variables,
- G a finite set of boolean expressions on V,
- S a finite set of variable assignment statements,
- $E \subset N \times A \,\dot\cup\, \{\tau\} \times G \times 2^S \times N$ is a finite set of edges,
- $m : N \to \{Basic,\ Or,\ And\}$ is a type function, identifying nodes as $Basic$ nodes, Or nodes, or And nodes.
- $r \in N$, $m(r) = Or$ is the root node of SC,
- $d : \{n : n \in N \wedge m(n) = Or\} \longrightarrow N$ assigns a default node to each node of type Or,
- $c : N \to 2^N$ a child relation introducing hierarchy on N.

The main extension to [14], [15] is the labelling of edges by elements of $A \,\dot\cup\, \{\tau\}$. While action labels in A will remain visible for the later transformation steps, and especially be used for synchronisation with start and stop events of stochastic time-constraints (see Section III-D), the label τ is used for ordinary Statechart edges. For the sake of brevity, the above definition omits some well-formedness conditions (cf. [14], [16]) that are unchanged with respect to STATEMATE Statecharts.

Example 2 The Statechart of Fig. 1 provides some intuition of the drawing conventions for extended Statecharts. The And node HEATER_CTRL is the only child of the root node r. The hierarchy determined by c is shown by nesting of states. Default nodes are indicated by arrows without source node. The edge $e_{02} = (\texttt{HEATER\_ON}, \tau, \texttt{TEMP==TOO\_HOT}, \{\texttt{HEATER:=DISABLED}\}, \texttt{HEATER\_OFF})$ is a τ-labelled Statechart edge which we draw as a thin line by convention, while edge $e_{16} = (\texttt{S\_OK}, \texttt{UP}, true, \{\texttt{TEMP:=TEMP\_IN}\}, \texttt{S\_OK})$ and edge $e_{05} = (\texttt{WAIT}, \texttt{SYNC}, true, \emptyset, \texttt{TLE})$ are labelled by elements of A respectively, and hence drawn bold. We implicitly define the guard g of such bold edges to be $true$. In this example $A = \{\texttt{UP}, \texttt{FM}, \texttt{FS}, \texttt{SYNC}\}$. In the above explanation, we refer to an additional edge labelling (e. g. e_{02}). These labels are not part of the Statechart itself, but used to keep the explanations uncluttered. It allows us to refer to 'edge e_{02}' in place of 'edge $(\texttt{HEATER\_ON}, \tau, \texttt{TEMP==TOO\_HOT}, \{\texttt{HEATER:=DISABLED}\}, \texttt{HEATER\_OFF})$'.

Essentially, the behaviour of an extended Statechart is in line with that of conventional STATEMATE Statecharts [14], [15], except that extended Statecharts allow for a more refined control over which sets of edges are allowed to be fired in orthogonal components within one step. We introduce the following usual notions to determine this semantics. The *scope* $sc(e)$ of an edge $e \in E$ is the most nested Or state that contains the edges nodes. We use $de(n)$ to denote the depth of node $n \in N$ in the node hierarchy c and define $de(SC) = max(\{de(n) : n \in N\})$. The *priority* of an edge e is given by its scope distance from the root r. We define the priority relation $e \leq_p e'$, s. t. $e \leq_p e'$ iff $de(SC) - de(sc(e)) \leq de(SC) - de(sc(e'))$. Two edges $e, e' \in E$ are orthogonal, denoted $e \perp e'$, iff either $e = e'$ or their scopes are different children of some And node or their descendants. In the heater Statechart, it holds $e_{12} \perp e_{03}$, for example. The labels in A affect the semantics of the edges they label as follows.

Definition 6 (Configurations)
Let $\mathcal{D}$ be the data domain of the variables V. A *configuration* of an extended Statechart SC is a pair $\mathfrak{c} = (M, \sigma) \in C \subset 2^N \times \Sigma$, where Σ is the set of all variable valuations $\sigma : V \setminus I \to \mathcal{D}$ and M is a set satisfying

1) $r \in M$,
2) $n \in M,\ m(n) = Or$ implies $\nexists n' \in c(n) :\ n' \in M$,
3) $n \in M,\ m(n) = And$ implies $\forall n' \in c(n) :\ n' \in M$.

Such a node set M is called a *valid node configuration*. We denote $\mathfrak{c}_0$ for the unique initial configuration of Statechart SC, given by an initial valuation of the variables σ_0 and the node configuration determined by d.
The set of all configurations of SC is denoted by $Conf$.

With $dc(M)$ for some $M \subset N$ we refer to the *default completion*, as the smallest superset of node set M, s. t. $dc(M)$ is a valid node configuration. In particular $dc(M)$ comprises the default node $d(n)$, for all those Or nodes $n \in dc(M)$, that are not already represented by a child node in M. The scope completion $scc(e)$ of edge e is the maximal set of child nodes derived by recursive application of c to the edges scope node $sc(e)$. Intuitively, configurations comprise all current $Basic$ nodes and their parent nodes (given by inverse of c) and a valuation of the variables V. In the following examples we will denote valuations σ by equation tuples and the node sets M will, for the sake of brevity, contain $Basic$ nodes only.

Example 3 Five possible configurations in the example chart in Fig. 1 are given by

$$\begin{aligned}
\mathfrak{c}_{01} &= (\{\texttt{CTRL\_OFF,M\_INIT,S\_INIT,SAFE}\},\\
&\quad \texttt{(TEMP=TOO\_COLD,HEATER=DISABLED,M\_ACTIVE=false,}\\
&\quad\ \texttt{S\_ACTIVE=false,S\_FAILED=false,SHUTDOWN=false)})\\
\mathfrak{c}_{04} &= (\{\texttt{HEATER\_ON,M\_OK,S\_OK,SAFE}\},\\
&\quad \texttt{(TEMP=TOO\_COLD,HEATER=ENABLED,M\_ACTIVE=true,}\\
&\quad\ \texttt{S\_ACTIVE=true,S\_FAILED=false,SHUTDOWN=false)})\\
\mathfrak{c}_{09} &= (\{\texttt{HEATER\_ON,M\_OK,S\_OK,SAFE}\},\\
&\quad \texttt{(TEMP=TOO\_HOT,HEATER=ENABLED,M\_ACTIVE=true,}\\
&\quad\ \texttt{S\_ACTIVE=true,S\_FAILED=false,SHUTDOWN=false)})\\
\mathfrak{c}_{08} &= (\{\texttt{HEATER\_OFF,M\_OK,S\_OK,SAFE}\},\\
&\quad \texttt{(TEMP=TOO\_HOT,HEATER=DISABLED,M\_ACTIVE=true,}\\
&\quad\ \texttt{S\_ACTIVE=true,S\_FAILED=false,SHUTDOWN=false)})\\
\mathfrak{c}_{16} &= (\{\texttt{HEATER\_ON,EMERGENCY\_SHUTDOWN,S\_STUCK,WAIT}\},\\
&\quad \texttt{(TEMP=TOO\_COLD,HEATER=ENABLED,M\_ACTIVE=false,}\\
&\quad\ \texttt{S\_ACTIVE=true,S\_FAILED=true,SHUTDOWN=true)}).
\end{aligned}$$

The semantics of extended Statecharts is given by sequences of such configurations:

Definition 7 (Configuration Paths)
For extended Statechart SC, the transition relation $\longrightarrow\ \subseteq C \times A \,\dot\cup\, \tau \times C$ is composed of two types of transitions:
Internal Step. $\mathfrak{c} = (M, \sigma) \xrightarrow{\tau} \mathfrak{c}' = (M', \sigma')$, iff there exists a maximal set of edges $\mathcal{E} = \{e_i : e_{1 \leq i \leq k} = (n_i, a_i, g_i, s_i, n'_i) \in E\}$ so that

1) $\mathcal{E} \subseteq \mathcal{E}_{en} = \{e = (n, a, g, s, n') \in E : n \in M$ and g evaluates to true in $\sigma\}$,
2) $\forall e_i \in \mathcal{E} : a_i = \tau$ and $\forall e_i, e_j \in \mathcal{E} : e_i \perp e_j$,
3) $\forall e \in \mathcal{E}_{en} \setminus \mathcal{E}\ \exists e' \in \mathcal{E}$, s. t. $e \not\perp e'$ and $e \leq_p e'$,

and σ' is obtained from σ by applying the statement sets $s_{1 \leq i \leq k}$ in some permutation on σ and $M' = dc((M \setminus \bigcup_{i=1}^{k} scc(e_i)) \cup \{n'_i\}_{1 \leq i \leq k})$.
External Step. $\mathfrak{c} = (M, \sigma) \xrightarrow{a} \mathfrak{c}' = (M', \sigma')$, iff

1) $\nexists e = (n, \tau, g, s, n') \in E : n \in M$ and g evaluates to true in σ,
2) $\exists e = (n, a, g, s, n') \in E : a \in A$ and $n \in M$,

and σ' is obtained from σ by applying the statement set s on σ and $M' = dc((M \setminus scc(e)) \cup \{n'_i\}_{1\leq i\leq n})$.
A sequence of alternating configurations and actions

$$\mathfrak{c}_0 \xrightarrow{l_0} \mathfrak{c}_1 \xrightarrow{l_1} \mathfrak{c}_2 \xrightarrow{l_2} \dots$$

is called *configuration path*.

In a nutshell, we embed the conventional Statechart configuration transitions in the *Internal Step* rule. Such a step comprises firing of a maximal set of τ-labelled edges (thin arrows) in orthogonal components, and thus implements truly concurrent executions. Instead a *delay transition* between Statechart configurations is defined by the *External Step* rule. It restricts the (bold) labelled edges to be fired *in mutual isolation*, and only if no τ-labelled edge can be taken. Since bold edges relate to time-relevant events, this mutual isolation allows us to recover particular configuration transitions, relative to other transitions. The semantics also embodies that internal steps do not take time, by giving them precedence over external steps. This idea of timeless computation is typical for the super-step semantics of STATEMATE Statecharts [14].

Example 4 Starting in the initial configuration $\mathfrak{c}_\circ = \mathfrak{c}_{\circ 1}$ of the Statechart in Fig. 1, by firing the (τ labelled edges) e_{06}, e_{11}, e_{15} one after the other, finally configuration $\mathfrak{c}_{\circ 4}$ is reached. Note that the *non-determinism* introduced by the input variable TEMP_IN $\in I$ allows a second sequence, which comprises e_{14} instead of e_{15} and thus leads to configuration $\mathfrak{c}_{\circ 8}$ instead of $\mathfrak{c}_{\circ 4}$. In $\mathfrak{c}_{\circ 4}$ a *delayed transition*, comprising edge e_{08}, e_{07} or e_{16} may be fired. In the overall composition context, firing these transitions will be constrained by (stochastic) delays: While e_{16} could be constrained to be fired every n minutes, e_{08} and e_{16} are constrained to be fired after 10^5 hours in the mean, for example.
The operational behaviour updates (e_{16}) the temperature TEMP in regular intervals and thus yields firing of e_{02} and e_{01}, respectively, again depending on the value of the non-deterministic input TEMP_IN. After some time the sensor fails (e_{07}) and the temperature value is stuck at the current value of TEMP. Consequently, a configuration is reached, where S_FAILED==true. Provided that the temperature currently is stuck at the value TEMP==TOO_COLD and thus it holds HEATER==ENABLED, immediately an *internal step* comprising *concurrent firing* of the edges e_{03}, e_{12} takes place.
Now the heater is in configuration $\mathfrak{c}_{16}$. The *external step* e_{05} labelled by SYNC may *not* be fired in this configuration, as further τ labelled progress is possible: e_{13} is fired. Intuitively, the monitor has observed the sensor failure and initiated an emergency shutdown. As this behaviour does not yield a safety-critical situation, the node $\{$TLE$\}$ is not entered.

We allow *hiding* of action labels once the semantics is generated. This is a simple transformation that replaces the respective transition labels by τ. This allows us to keep the effect of external steps without keeping their labels.

Example 5 In the running example, we hide the action SYNC.

III. FROM STATEMATE TO NUMBERS

This section presents a detailed description of our construction and transformation process. This process starts with the input parameters mentioned in Section I and returns the worst-case probability to reach a safety-critical state within the provided time bound. An overview of the steps is given in Fig. 2. The figure indicates that the first two steps are applied to a symbolic (i. e. BDD-based) representation of the system: The first, detailed in Section III-A, transforms the STATEMATE design plus the delay transitions and the safety-critical states into an LTS. The second step (Section III-B) minimises the system and produces an explicit state representation of the system by computing the quotient of the LTS w. r. t. a branching bisimulation induced by the labels appearing in the LTS.
Up to this point, the delay transitions are contained in the resulting model, but not their associated delay distributions. We incorporate them as follows. First, the given delay distributions—typical distributions in this context are Weibull, deterministic, exponential distributions, or distributions obtained from empirical measurements—are approximated by phase-type distributions, which can be represented by absorbing Markov chains (Section III-C). These chains are combined with the delay transitions via the action labels associated with them. This is performed by a dedicated *elapse* operator, that transforms the delay distributions into so-called time-constraint IMCs, governing the timely occurrences of the delay transitions in the system (Section III-D). These time-constraints are then weaved into the quotient LTS (Section III-E) resulting in an IMC that encodes all user-specified inputs (STATEMATE design, delay transitions plus their associated delay distributions, and the safety critical states.). This IMC represents thus the entire system under study and captures the behavioural essence of both the functional and the dependability aspects.
This monolithic system model is subjected to a transformation procedure (Section III-G) that preserves timed reachability properties and computes the *underlying* CTMDP of the IMC. From this CTMDP we can compute the worst-case probabilities to reach the set of safety-critical states within a given time bound (Section III-H) by using the probabilistic model checker MRMC [17]. Comparing this to the given threshold of the safety requirement allows us to determine whether the requirement is satisfied by the given system. As highlighted in the motivation, this algorithm works only for *uniform* CTMDPs. In uniform CTMDPs, state changes occur according to a Poisson process in time. However, we ensure uniformity (Section III-F) all along our tool chain and ensure that the resulting CTMDP is indeed uniform.

A. LTS Generation

An *extended Statechart* $SC = (N, A, V, G, S, E, m, r, d, c)$, with initial configuration $\mathfrak{c}_0$, can be considered as an LTS $M = (S^M, Act^M, T^M, s_0^M)$ by setting

- $S^M = Conf \,\dot{\cup}\, \{\mathfrak{c}_{\text{init}}\}$, the set of all valid configurations in SC plus a unique pre-initial state $\mathfrak{c}_{\text{init}}$.
- $Act^M = A \,\dot{\cup}\, \{\tau\} \,\dot{\cup}\, \{\texttt{INIT}\}$, the set of labels occurring in the Statechart steps plus a unique label INIT.
- $T^M = \{(\mathfrak{c}_{\text{init}}, \texttt{INIT}, \mathfrak{c}_0)\} \cup \{(\mathfrak{c}, a, \mathfrak{c}') : \mathfrak{c} \xrightarrow{a} \mathfrak{c}'\} \subseteq S^M \times A^M \times S^M$, the set of transitions possible between the Statechart configurations plus an additional transition $\mathfrak{c}_{\text{init}} \xrightarrow{\texttt{INIT}} \mathfrak{c}_0$, introduced to represent the system start.
- $s_0^M = \mathfrak{c}_{\text{init}}$, a pre-initial configuration of Statechart SC.

Note that we added edge $(\mathfrak{c}_{\text{init}}, \texttt{INIT}, \mathfrak{c}_0)$ as a unique reference anchor for the system start. Apart from this LTS, a set of safety critical states $S_{cr}^M \subset S^M$ is distilled from the Statechart. This set is induced by a user-specified characterisation of critical Statechart nodes $N_{cr} \subset N$, by setting $S_{cr}^M = \{\mathfrak{c} = (M_\mathfrak{c}, \sigma_\mathfrak{c}) : M_\mathfrak{c} \cap N_{cr} \neq \emptyset\}$.

The first step of the translation extracts such an LTS M from the *extended Statechart* SC, together with the set S_{cr}^{M}.

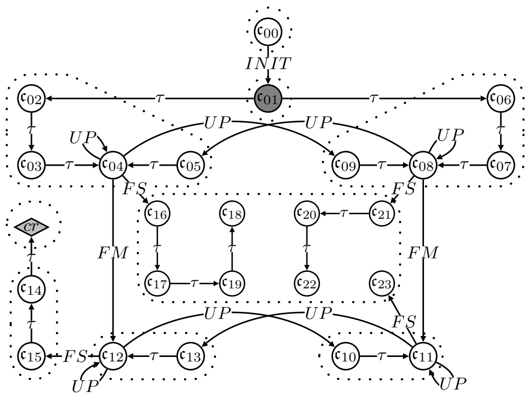


Fig. 3. LTS of the heater example.

Example 6 Given the extended Statechart of Fig. 1 and the specification of safety-critical states as the node set $\{\texttt{TLE}\}$, the LTS in Fig. 3 is derived. The states of this LTS in particular refer to the configurations of the examples in Section II-B. Moreover it shows the configuration path to the safety-critical state we are interested in during the *timed reachability analysis*. Recall, that in $\mathfrak{c}_{04}$ the sensor failure (`FS`) yields a safe shutdown of the system. If instead `MONITOR` (`FM`) fails prior to the `SENSOR` (`FS`), then $\mathfrak{c}_{12}$ is reached. Here the shutdown (namely Statechart edge e_{12}) can no longer be taken. Hence, after the `Sensor` has failed (`FS`), in $\mathfrak{c}_{14}$ no further τ progress is possible and thus the `SYNC` edge e_{05} is fired.

The translation comprises several stages that finally result in an efficient BDD-based encoding of the LTS.
In particular, some standard reduction techniques (e. g. cone-of-influence reduction (COI) as presented in [18]) are used to eliminate variables whose values do not contribute to the reachability of the safety-critical states. This elimination reduces the number of possible variable valuations and therefore also the state space of the resulting model. This COI reduction is property specific, and as such, a core step allowing us to obtain small, property-specific models from large designs.
The result of this stage is passed to an extended version of the VIS model checker [19], that we primarily use to restrict the transition relation to the reachable transitions only and as a framework to implement the final semantics described above. Using VIS, we get an LTS symbolically encoded as a BDD.
Three BDDs, coding the LTS, the set of safety-critical states and the initial state, are then passed to the symbolic branching bisimulation algorithm, which will be described in the next section. The partition of the state space induced by the predicate (into safety-critical and non-safety-critical states) will be used as a starting point for this algorithm.

B. Symbolic Minimisation of the LTS

To further reduce the models (beyond the COI reduction) to a size which can be handled by the explicit part of the tool chain, the development of an *entirely symbolic* branching minimisation algorithm was necessary. We assume that the reader is familiar with BDDs and the corresponding algorithms. For a comprehensive treatment see e. g. [20].
In [21], Blom and Orzan presented a novel approach for the distributed computation of branching bisimulation. Their algorithm is based on analysing the *signatures* of states w. r. t. the current partition. The signature of a state is like a fingerprint identifying possible actions which can be executed in that state. To preserve branching bisimilarity, the unobservable action τ is taken into account by ignoring inert sequences of τ-transitions.
Let $P = \{B_0, \dots, B_{p-1}\}$ be a partition of the state space S. The signature $\mathrm{sig}(P, s)$ of a state s w. r. t. P is formally defined as

$$\mathrm{sig}(P,s) = \big\{(a,B) \in Act \times P \mid \exists s' \in S, s'' \in B : s \xrightarrow[P]{\tau^*} s' \xrightarrow{a} s'' \wedge (a \neq \tau \vee s \notin B)\big\}.$$

Then, a refinement of the partition can be computed by splitting the blocks of the current partition according to the signatures of their states:

$$\mathrm{sigref}(P,B) = \big\{\{s' \in B \mid \mathrm{sig}(P,s) = \mathrm{sig}(P,s')\} \,\big|\, s \in B\big\}.$$
$$\mathrm{sigref}(P) = \bigcup_{B \in P} \mathrm{sigref}(P,B).$$

Starting with the initial partition, which is provided by the predicate separating safety-critical and non-critical states, we iteratively apply the sigref-operator, until a fix-point is reached.

Theorem 1 (adapted from Blom/Orzan, [21]) *Let $P^{(0)}, P^{(1)}, \dots$ be a sequence of partitions with $P^{(i)} = \mathrm{sigref}\big(P^{(i-1)}\big)$. Then there is $n > 0$ such that $P^{(n)} = P^{(n-1)}$. $P^{(n)}$ is the coarsest branching bisimulation which refines $P^{(0)}$.*

We will illustrate the function of the algorithm in the following example.

Example 7 For the heater example (cf. Fig. 3), we set the initial partition $P^{(0)} = \big(\{\mathfrak{c}_{00}, \dots, \mathfrak{c}_{23}\}, \{cr\}\big) = (B_0^0, B_1^0)$, as induced by the characterisation of configurations to be safety critical. It is depicted in Fig. 3 by diamond-shaped and round circles, respectively. Computing the signatures w. r. t. $P^{(0)}$, we obtain the following sets

$$\begin{aligned}
\mathrm{sig}(P^{(0)}, \mathfrak{c}_{00}) &= \big\{(\texttt{INIT}, B_0^0)\big\},\\
\mathrm{sig}(P^{(0)}, \mathfrak{c}_{01}) &= \mathrm{sig}(P^{(0)}, \mathfrak{c}_{02}) = \dots = \mathrm{sig}(P^{(0)}, \mathfrak{c}_{09}),\\
&= \big\{(\texttt{UP}, B_0^0), (\texttt{FM}, B_0^0), (\texttt{FS}, B_0^0)\big\},\\
\mathrm{sig}(P^{(0)}, \mathfrak{c}_{16}) &= \mathrm{sig}(P^{(0)}, \mathfrak{c}_{17}) = \dots = \mathrm{sig}(P^{(0)}, \mathfrak{c}_{19}),\\
&= \mathrm{sig}(P^{(0)}, \mathfrak{c}_{20}) = \dots = \mathrm{sig}(P^{(0)}, \mathfrak{c}_{23}),\\
&= \emptyset,\\
\mathrm{sig}(P^{(0)}, \mathfrak{c}_{10}) &= \mathrm{sig}(P^{(0)}, \mathfrak{c}_{11}) = \dots = \mathrm{sig}(P^{(0)}, \mathfrak{c}_{13}),\\
&= \big\{(\texttt{UP}, B_0^0), (\texttt{FS}, B_0^0)\big\},\\
\mathrm{sig}(P^{(0)}, \mathfrak{c}_{14}) &= \mathrm{sig}(P^{(0)}, \mathfrak{c}_{15}) = \big\{(\tau, B_1^0)\big\},\\
\mathrm{sig}(P^{(0)}, cr) &= \emptyset.
\end{aligned}$$

Hence, the application of the sigref-operator splits the two blocks of the initial partition into the following six parts: $P^{(1)} = \big(\{\mathfrak{c}_{00}\}, \{\mathfrak{c}_{01}, \dots, \mathfrak{c}_{09}\}, \{\mathfrak{c}_{16}, \dots, \mathfrak{c}_{19}, \mathfrak{c}_{20}, \dots, \mathfrak{c}_{23}\}, \{\mathfrak{c}_{10}, \dots, \mathfrak{c}_{13}\}, \{\mathfrak{c}_{14}, \mathfrak{c}_{15}\}, \{cr\}\big)$. Please note that cr and, for example, $\mathfrak{c}_{16}$ are placed in different blocks, although they have the same signature w. r. t. $P^{(0)}$, since they are not equivalent in $P^{(0)}$.
The iteration of this refinement step, until a fix-point is reached, leads to the following final branching bisimulation, which comprises nine blocks: $P^{(n)} = \big(\{\mathfrak{c}_{00}\}, \{\mathfrak{c}_{01}\}, \{\mathfrak{c}_{10}, \mathfrak{c}_{11}\}, \{\mathfrak{c}_{12}, \mathfrak{c}_{13}\}, \{\mathfrak{c}_{14}, \mathfrak{c}_{15}\}, \{\mathfrak{c}_{02} \dots \mathfrak{c}_{05}\}, \{\mathfrak{c}_{06} \dots \mathfrak{c}_{09}\}, \{\mathfrak{c}_{16} \dots \mathfrak{c}_{23}\}, \{cr\}\big)$. This partition is depicted in Fig. 3 using dashed frames around equivalent states.

In the following we describe briefly how this signature-based refinement can be turned into a BDD-based algorithm.

TABLE I
BASIC OPERATIONS FOR SIGNATURE COMPUTATION

Operation	*BDD expression*
τ-transitions	$\mathcal{T}_{\vert a=\tau}(s,a,t)$
Pairs of equivalent states	$\mathrm{inert}(s,t) = \exists k : \big(\mathcal{P}(s,k) \wedge \mathcal{P}(t,k)\big)$
Non-τ- or non-inert ("observable") transitions	$\mathrm{obs}(s,a,t) = \mathcal{T}(s,a,t) \wedge \neg(\mathrm{inert}(s,t) \wedge a \equiv \tau)$
Reflexive transitive closure of $\mathcal{R}(s,t)$	$\mathrm{Closure}(\mathcal{R})$
Concatenation of $\mathcal{R}_1(s,t)$ and $\mathcal{R}_2(s,t)$	$\exists x : \big(\mathcal{R}_1(s,x) \wedge \mathcal{R}_2(x,t)\big)$
Substitution of t in $\mathcal{R}(s,t)$ by its block number	$\exists t : \big(\mathcal{R}(s,t) \wedge \mathcal{P}(t,k)\big)$

Algorithm 1 Signature for Branching Bisimulation

```
1: procedure SIGBRANCHING
2:     $\mathrm{tauSteps}(s,t) \leftarrow \mathrm{Closure}(\mathcal{T}_{|a=\tau}(s,a,t)) \wedge \mathrm{inert}(s,t)$
3:     $\mathrm{seq}(s,a,t) = \exists x : \big(\mathrm{tauSteps}(s,x) \wedge \mathrm{obs}(x,a,t)\big)$
4:     return $\exists t : \big(\mathrm{seq}(s,a,t) \wedge \mathcal{P}(t,k)\big)$
```

a) Data representation: The starting point is a BDD $\mathcal{T}$ for the transition relation with $\mathcal{T}(s,a,t) = 1$ iff $s \xrightarrow{a} t$. Note that the state space is implicitly encoded by $\mathcal{T}$. The BDD relies on a vector of variables s, a, and t to encode the current state, the transition label, and the target state, respectively.

Beside the system itself we have to represent partitions and signatures symbolically. For the partition representation several possibilities have been used in the literature: Bouali and de Simone [22] represent the corresponding equivalence relation $\equiv_P$; another possibility is to use one BDD per block or a compact logarithmic encoding thereof [23]. We have decided to use the following novel technique, since it supports partition refinement and quotient computation very efficiently. To represent the current partition, we assign a unique number to each block, i. e. $P = \{B_0, \dots, B_{n-1}\}$, and encode it using additional BDD variables. The representation is thus a BDD $\mathcal{P}(s,k)$ such that $\mathcal{P}(s,k) = 1$ iff $s \in B_k$. Additionally, we have a BDD $\mathcal{S}$ for the signatures with $\mathcal{S}(s,a,k) = 1$ iff $(a, B_k) \in \mathrm{sig}(s)$.

b) Signature computation: Given a partition $\mathcal{P}(s,k)$ in our BDD-based representation, we have to compute the signatures of the states. To facilitate this, we provide several core operations which are listed in Table I. The table contains a description of each operation and an expression for the BDD-based implementation. The notation $\mathcal{T}_{|a=\tau}$ denotes the co-factor operation on BDDs (setting the a variables to the value τ). For the computation of the reflexive transitive closure $\mathrm{Closure}(\mathcal{R})$ of a relation $\mathcal{R}$, there exist several symbolic algorithms, see e. g. [24], [25]. We apply the (at least in our experiments) more efficient method of [24] to compute $\xrightarrow[(P)]{\tau^*}$.

Algorithm 1 shows the pseudo-code for computing the signatures for branching bisimulation.

At first, all pairs of states that are connected by an inert sequence of τ-transitions are computed. In line 3 we extract all transitions that are either not inert or not labelled with τ and concatenate them with the inert τ-sequence. In the third step we replace the target state of the transition sequence by its block number. The signatures for other kinds of bisimulation can be computed in a similar way. Please note that everything that does not depend on the current partition, like the closure of the τ-steps, can be computed once as a preprocessing step.

c) Partition refinement: The novelty of our approach is a dedicated BDD-operator for identifying states that have the same signature, thus enabling a full BDD-based methodology. To do so, we place the s_i-variables at the beginning of the variable order of the BDDs. Then, $\mathrm{level}(s_i) < \mathrm{level}(a_j)$ and $\mathrm{level}(s_i) < \mathrm{level}(k_l)$ hold for all i, j, and l. This enables us to exploit the following observation (see Fig. 4): Let s be the encoding of a state and v the BDD node that is reached when following the path from the BDD root according to s. The sub-BDD at v is a representation of the signature of s. Since the BDD is reduced, all states which have the same signature as s lead to the same BDD-node v. Therefore, to get the representation of the new block that contains s and *all* other states having the same signature as s, we simply have to replace the sub-BDD at v by the BDD for the encoding of the new block number k. This can be achieved by traversing the BDD recursively in linear time in the size of the BDD.

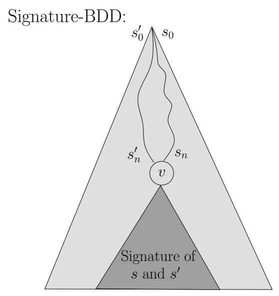


Fig. 4. Symbolic partition refinement.

d) Optimisation Techniques: Additionally, we integrate some simple, but efficient optimisation techniques [26]:

The first observation is that the computation of the expression $\exists k : \mathcal{P}(s,k) \wedge \mathcal{P}(t,k)$ is computationally very expensive, since the state (s,t) and block number variables (k) are placed at different ends of the variable order. To avoid the computation of this expression we only refine one block $\mathcal{B}(s)$ at a time. Then we can replace the expensive expression by $\mathcal{B}(s) \wedge \mathcal{B}(t)$. This enables us to update the partition after the refinement of each block. So the finer partition is already used for the subsequent blocks of the same iteration. This reduces the number of iterations required to reach the fixpoint. Furthermore, by applying block-wise refinement we are able to handle arbitrary initial partitions.

The second observation is that in each iteration (except in the first few iterations) only a very small number of blocks is indeed split. We can exploit this in the following way: Splitting a block only influences the stability of blocks which are directly connected with the split block by an observable transition. We capture this in the backward signature:

$$\mathrm{bwsig}(P,B) = \big\{B' \in P \mid \exists s \in B \exists a \in Act : (a, B') \in \mathrm{sig}(P,s)\big\}.$$

Since the same problem as above—the dependence on the inert τ-sequences—prevents an efficient implementation, we only

Algorithm 2 Computation of the coarsest branching bisimulation

```
1: procedure BranchingBisimulation(LTS $M$, Partition $P^{(0)}$)
2:     $U, P \leftarrow P^{(0)}$
3:     while $U \neq \emptyset$ do
4:         $U_{\text{new}} \leftarrow \emptyset$
5:         for all blocks $B \in U$ do
6:             $P \leftarrow \big(P \setminus \{B\}\big) \cup \mathrm{sigref}(P, B)$
7:             $U_{\text{new}} \leftarrow U_{\text{new}} \setminus \{B\}$
8:             if $B$ was split then
9:                 $U_{\text{new}} \leftarrow U_{\text{new}} \cup \mathrm{bwsig}^{oa}(P, B)$
10:        $U \leftarrow U_{\text{new}}$
11:    return $P$
```

compute an over-approximation by ignoring that τ-steps only influence the stability of blocks if they are not inert. This leads to $\mathrm{bwsig}^{oa}(P, B)$.

The algorithm which applies all these optimisations is shown in Algorithm 2.

P contains the current partition; U denotes the potentially unstable blocks which have to be refined in the current iteration. U_{new} stores all blocks which became potentially unstable during the current iteration. We iterate the following step until no unstable blocks are left: We replace the current block in the current partition by the result of its refinement and declare it as stable. If it has been split, we add the blocks of its backward signature to the potentially unstable blocks.

e) Quotient extraction: Finally, after we have reached the fixpoint, we have to extract the quotient LTS from the final partition. This can be done by mapping all states of a block onto one quotient state, which is given by the block number. Each quotient state which is safety critical is decorated with a self-transition labelled cr during the extraction. The resulting model is now represented in an explicit form, i. e., all states are explicitly enumerated, and passed on to the next phase of the tool chain. We refer to [27], [28] for more details and for experimental evaluations of this minimisation algorithm.

C. Phase-Type Approximation

The approach we follow renders the model under study into a Markov model. To achieve this, we must represent the delay distributions provided to us in a Markov chain. This is possible by applying a widespread approach based on phase-type approximation [29], [30].

A phase-type distribution is the distribution of the time until absorption in a finite and absorbing Markov chain [31]. Let (S, Act, R, s_1) be CTMC with $S = \{s_1, s_2, \cdots, s_n, s_a\}$. Further, let s_a be an absorbing state (i. e., $\mathbf{r}(s_a, S) = 0$) and let all other states s_i, for $1 \leq i \leq n$, be transient (i. e., there is a non-zero probability that the state will never be visited once it is visited). The set of Markovian transitions R is related to the corresponding infinitesimal generator matrix $\mathbf{Q}$ by: for all $s, s' \in S$, $\mathbf{Q}(s, s') = R(s, s')$ if $s \neq s'$ else $\mathbf{Q}(s, s) = -\mathbf{r}(s, S)$. The generator matrix of the Markov chain can be written as

$$\mathbf{Q} = \begin{bmatrix} \mathbf{B} & \vec{B} \\ \mathbf{0} & 0 \end{bmatrix}.$$

Matrix $\mathbf{B}$ is non-singular because the first n states in the Markov chain are transient. Vector $\vec{B}$ is a column vector where its component $\vec{B}_i$, for $1 \leq i \leq n$, represents the transition rate from state s_i to the absorbing state. Then the probability distribution of the time to absorption in the CTMC is called a *phase-type* distribution. Note that in our definition a phase-type distribution starts from a single state with probability 1. This does not restrict the generality of the definition, because any phase-type distribution with arbitrary initial distribution can be transformed into our form by using the procedure described in [32].

The class of phase-type distributions is topologically dense [33]. In principle, any probability distribution on $[0, \infty)$ can be approximated arbitrarily closely by a phase-type distribution given enough phases, i. e., states. Efficient approximation algorithms are available, such as those based on expectation-minimisation [29], [30], [34] and that based on the least-square method [35]. We implemented a variant thereof, based on orthogonal distance fitting [36], which is a type of least-square fitting method where the errors to be minimised are measured according to geometric distance.

The approximation algorithms mentioned above are general purpose. Two specific delay distributions are approximated in a more straightforward manner. Delay distributions with an underlying Poisson process are best represented by exponential distributions. Erlang distributions are suitable for approximating deterministic delay distributions or fixed delays [37], [38]. For instance, according to [38], an Erlang distribution with shape parameter 5 and rate 5 is sufficient to approximate a fixed delay at 1 time unit. Better approximations may be achieved by increasing the shape and rate parameters.

D. Elapse

We now assume that the delay of each delay transition d in the system model is given by a phase-type distribution PH_d determining the time until the transition occurs. Structurally, PH_d is a CTMC (S, Act, R, s_1) with a distinguished initial state s_1 and an absorbing state s_a. Operationally, the distribution PH_d can be viewed as describing the time up to which the occurrence of transition d has to be delayed, since it is triggered by the occurrence of some delay transition (called starting) unless another delay transition (called breaking) occurs in the mean time. This interpretation is called a *time-constraint* in [39], where an *elapse* operator is introduced. This operator enriches PH_d with 'synchronisation potentials' needed to effectively *weave* the underlying CTMC of PH_d into the behaviour described by some LTS or IMC.

Definition 8 (Elapse Operator)
Let $\mathfrak{s}$, $\mathfrak{d}$ and $\mathfrak{b}$ be the sets of starting, delay and breaking transitions, respectively. $\mathsf{elapse}(PH_d, \mathfrak{s}, \mathfrak{d}, \mathfrak{b})$ is an IMC (S', Act', T, R, s_s) where the following holds

1) if $\mathfrak{s} - \mathfrak{d} - \mathfrak{b} \neq \emptyset$ then $S' = S \cup \{s_s\}$ else $S' = S$, $s_s = s_1$,
2) $Act' = Act \ \cup \ \mathfrak{s} \ \cup \ \mathfrak{d} \ \cup \ \mathfrak{b}$,
3) $\forall s \in \mathfrak{s} - \mathfrak{d} - \mathfrak{b} : s_s \xrightarrow{s} s_1 \in T$,
4) $\forall d \in \mathfrak{d} \cap \mathfrak{s} : s_a \xrightarrow{d} s_1 \in T$,
5) $\forall d \in \mathfrak{d} - \mathfrak{s} : s_s \xrightarrow{d} s_s, \ s_a \xrightarrow{d} s_s \in T$,
6) $\forall b \in \mathfrak{b} \cap \mathfrak{s} : \forall t \in S : t \xrightarrow{b} s_1 \in T$,
7) $\forall b \in \mathfrak{b} - \mathfrak{s} : \forall t \in S : t \xrightarrow{b} s_s \in T$.

Example 8 Three time-constraints are required in the running heater example. The delays before the occurrences of sensor and monitor failures (`FM` and `FS`) are initialised when the system starts, namely when `INIT` occurs. The delay before the occurrences of temperature update (`UP`), on the other hand, is initialised by `INIT`, and afterwards

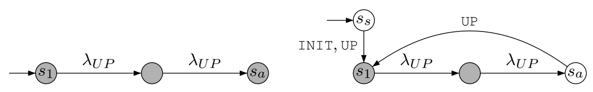


Fig. 5. A simple absorbing CTMC (left) and the elapse operator (right).

restarted continuously after each delay elapses. In this example, we use simple exponential distributions with rates λ_{FM} and λ_{FS} as the phase-type distributions for the first two time-constraints. They are denoted by $Exp(\lambda_{FM})$ and $Exp(\lambda_{FS})$, respectively. For the third, an Erlang distribution with rate parameter λ_{UP} and shape parameter 2—denoted by $Erl(\lambda_{UP}, 2)$—is used. The three time-constraints are obtained from the following elapse operations

1) $\mathsf{elapse}(Exp(\lambda_{FS}), \{\mathtt{INIT}\}, \{\mathtt{FS}\}, \emptyset)$,
2) $\mathsf{elapse}(Exp(\lambda_{FM}), \{\mathtt{INIT}\}, \{\mathtt{FM}\}, \emptyset)$,
3) $\mathsf{elapse}(Erl(\lambda_{UP}, 2), \{\mathtt{INIT}, \mathtt{UP}\}, \{\mathtt{UP}\}, \emptyset)$.

Fig. 5–left shows $Erl(\lambda_{UP}, 2)$, while Fig. 5–right depicts the IMC of the third time-constraint.

E. Weaving the Time-Constraints

In $\mathsf{elapse}(PH_d, \mathfrak{s}, \{d\}, \mathfrak{b})$, between any two occurrences of a delay transition d, there must be a delay which is given by the CTMC PH_d. To enforce this also for the LTS of our system under study, we weave this uniform IMC with the LTS, where weaving is just another word for interleaving, with proper synchronisation.

To this end, we use the process algebraic parallel composition operator. Intuitively, given IMCs $\mathcal{I}$ and $\mathcal{J}$ and their set of common transitions $\mathfrak{c}$, in $\mathcal{I}|[\mathfrak{c}]|\mathcal{J}$ both IMCs have to synchronise on $\mathfrak{c}$-transitions, while they interleave all other transitions. For $\mathcal{I} = \mathsf{elapse}(PH_d, \mathfrak{s}, \{d\}, \mathfrak{b})$, this has the expected effect, namely that between any two d-transitions in $\mathcal{J}$, the Markov chain associated with PH_d is weaved. The semantic rules of parallel composition of IMCs are as follows (where it is understood that whenever multiple distinct proof trees for the same Markov transition exist, the respective rate is weighted by that multiplicity.)

$$\frac{s\xrightarrow{a}s' \qquad a\notin\{a_1\ldots a_n\}}{s|[a_1\ldots a_n]|v\xrightarrow{a}s'|[a_1\ldots a_n]|v} \qquad \frac{v\xrightarrow{a}v' \qquad a\notin\{a_1\ldots a_n\}}{s|[a_1\ldots a_n]|v\xrightarrow{a}s|[a_1\ldots a_n]|v'}$$

$$\frac{s\xrightarrow{a}s' \qquad v\xrightarrow{a}v' \qquad a\in\{a_1\ldots a_n\}}{s|[a_1\ldots a_n]|v\xrightarrow{a}s'|[a_1\ldots a_n]|v'}$$

$$\frac{s\overset{\lambda}{\dashrightarrow}s'}{s|[a_1\ldots a_n]|v\overset{\lambda}{\dashrightarrow}s'|[a_1\ldots a_n]|v} \qquad \frac{v\overset{\lambda}{\dashrightarrow}v'}{s|[a_1\ldots a_n]|v\overset{\lambda}{\dashrightarrow}s|[a_1\ldots a_n]|v'}$$

With this operator, and the elapse operator, we can weave the delay distributions one by one into the original system.

During this composition phase the IMC is explicitly represented and grows in size. One way of counteracting this is to minimise according to stochastic branching bisimulation. For this we use a stochastic branching bisimulation minimisation algorithm [40], together with the *abstraction* (or *hiding*) operator. The semantics of the abstraction operator is as follows (we give it for single actions only here for the sake of brevity.)

$$\frac{s\xrightarrow{b}s' \qquad a\neq b}{\mathsf{hide}\ a\ \mathsf{in}\ (s)\xrightarrow{b}\mathsf{hide}\ a\ \mathsf{in}\ (s')} \qquad \frac{s\xrightarrow{a}s'}{\mathsf{hide}\ a\ \mathsf{in}\ (s)\xrightarrow{\tau}\mathsf{hide}\ a\ \mathsf{in}\ (s')}$$

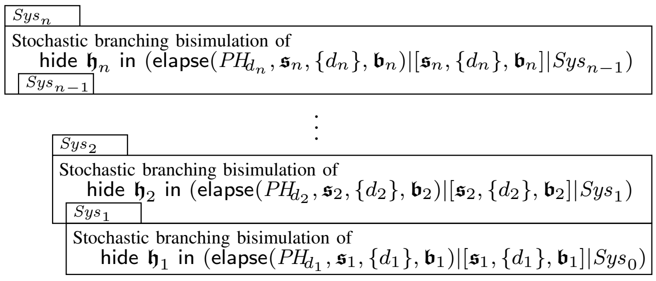
$$\frac{s\overset{\lambda}{\dashrightarrow}s'}{\mathsf{hide}\ a\ \mathsf{in}\ (s)\overset{\lambda}{\dashrightarrow}\mathsf{hide}\ a\ \mathsf{in}\ (s')}$$

In our composition scheme, we start from the initial explicit LTS as Sys_0, and incrementally build an IMC where the delay distributions are weaved, which is achieved by constructing

$$\mathsf{hide}\ \mathfrak{h}_i\ \mathsf{in}\ (\mathsf{elapse}(PH_{d_i}, \mathfrak{s}_i, \{d_i\}, \mathfrak{b}_i)\ |[\mathfrak{s}_i, \{d_i\}, \mathfrak{b}_i]|\ Sys_{i-1})$$

where $\mathfrak{h}_i$ is the set of transitions that can be hidden at this stage, namely those that will not be used in further synchronisations. The result of the construction is then minimised with respect to stochastic branching bisimulation to form Sys_i. If we are dealing with n different delay transitions, then the resulting IMC Sys_n does not contain any delay transitions anymore, but the delay distributions now interleave in the correct way governing the time to reach a safety-critical state.

The above approach alternates construction and minimisation steps, and as such it deviates from the sequential procedure indicated in Fig. 2: it replaces the trajectory from *quotient LTS* and *uIMCs* to *uIMC* by the one depicted in Fig. 6. This *compositional approach* is justified, because stochastic branching bisimulation is compatible with the two operators we have introduced: it is a congruence for parallel composition and hiding.



Sys_n
Stochastic branching bisimulation of
$\mathsf{hide}\ \mathfrak{h}_n\ \mathsf{in}\ (\mathsf{elapse}(PH_{d_n}, \mathfrak{s}_n, \{d_n\}, \mathfrak{b}_n)|[\mathfrak{s}_n, \{d_n\}, \mathfrak{b}_n]|Sys_{n-1})$
Sys_{n-1}
Sys_2
Stochastic branching bisimulation of
$\mathsf{hide}\ \mathfrak{h}_2\ \mathsf{in}\ (\mathsf{elapse}(PH_{d_2}, \mathfrak{s}_2, \{d_2\}, \mathfrak{b}_2)|[\mathfrak{s}_2, \{d_2\}, \mathfrak{b}_2]|Sys_1)$
Sys_1
Stochastic branching bisimulation of
$\mathsf{hide}\ \mathfrak{h}_1\ \mathsf{in}\ (\mathsf{elapse}(PH_{d_1}, \mathfrak{s}_1, \{d_1\}, \mathfrak{b}_1)|[\mathfrak{s}_1, \{d_1\}, \mathfrak{b}_1]|Sys_0)$

Fig. 6. Compositional weaving of phase-type distributions.

F. Uniformity

So far we have ignored the word *'uniform'* (abbreviated 'u') which is attached to the IMC models appearing in Fig. 2. We recall that IMC $\mathcal{I}$ is called *uniform*, iff $\exists e \in \mathbb{R}^+$ such that $\forall s \in S : s \overset{\tau}{\not\rightarrow}$ implies $\mathbf{r}(s, S) = e$.

In the following, we establish that parallel composition, hiding and (stochastic) branching bisimulation preserve uniformity.

Theorem 2 *Let $\mathcal{I}$ and $\mathcal{J}$ be two uniform IMCs with uniform rate $E_{\mathcal{I}}$ and $E_{\mathcal{J}}$, respectively.*

1) *For a given set of actions A it holds that IMC $\mathcal{I}|[A]|\mathcal{J}$ is uniform with rate $E_{\mathcal{I}} + E_{\mathcal{J}}$.*
2) *For a given action a it holds that IMC* $\mathsf{hide}\ a\ \mathsf{in}\ (\mathcal{I})$ *is uniform with rate $E_{\mathcal{I}}$.*
3) *Suppose that $\mathcal{I}$ and $\mathcal{J}$ are stochastic branching bisimilar according to Definition 2. Then it holds that $E_{\mathcal{I}} = E_{\mathcal{J}}$.*

For a detailed proof we refer to [13].

As a result of Theorem 2, we are left with the requirement that all our input models must be uniform in order to ensure uniformity by construction. Since any LTS is a uniform IMC by definition, we only need to ensure that the time-constraints, which are used for composition, are uniform, too.

Technically, this can be achieved as follows. Let (S, Act, R, s_1) be the CTMC of some phase-type distribution PH_d with initial state s_1 and absorbing state s_a, and let $e = \max_{s\in S} \mathbf{r}(s, S)$. To this CTMC we associated a uniform CTMC (S, Act, R', s_1), where $R'(s, s') = R(s, s')$ if $s \neq s'$ and $R'(s, s) = e - \mathbf{r}(s, S\backslash\{s\})$ otherwise. Under the usual interpretation of CTMCs, there is no difference between the two CTMCs (since the induced generator matrices are identical). For our purposes, however, we note a seemingly minor difference, namely that in the uniform CTMC, jumps occur on average after $1/\mathrm{e}$ time units, regardless of the state considered[2].

For the example in Fig. 5, the uniform variant is obtained by equipping state s_a with a looping λ-transition. Here, and in general, the result can be easily ensured to be a uniform IMC. All in all, the time-constraints, and the input LTS are uniform, and thus our construction preserves uniformity all along.

G. CTMDP Transformation

In Section III-D time behaviour has been incorporated into the system description, turning the LTS into an IMC. This section describes a transformation from IMCs to CTMDPs that preserves timed reachability properties and uniformity.

The model we are dealing with is the complete description of the system under consideration, and therefore can be viewed as a *closed system*. This means that the transformation to be carried out now no longer needs to be compositional, because all necessary composition operations have been performed in earlier steps of the tool chain. As a consequence we will now employ an *urgency assumption*, i. e., we assume that interactive transitions take zero time (which is a non-compositional hypothesis [5]).

Given an IMC $\mathcal{I} = (S, Act, T, R, s_0)$, we can partition the set S into three disjoint sets of states. These are the sets of (1) *interactive states*, where a state has no Markov transitions (denoted S_I); (2) *Markov states*, where a state has only Markov transitions (denoted S_M); and (3) *hybrid states* where a state has at least one Markov and at least one interactive transitions (denoted S_H).

Recall that any CTMDP can be viewed as a special IMC in which interactive states and Markov states occur in a strictly alternating manner. Thus, in order to turn an IMC $\mathcal{I}$ into a CTMDP $\mathcal{C}$ we have to ensure that all states are either Markov or interactive states, and that they strictly alternate. We call this class of IMC *strictly alternating*.

We now discuss a transformation which turns any IMC into a strictly alternating one [13], while preserving the probabilistic behaviour. The transformation involves: (1) ensuring that the state space contains interactive and Markov states only (called *alternating* IMC); (2) making the target state of each Markov transition an interactive state (called *Markov alternating* IMC); and (3) making the target state of each interactive transition a Markov state (called *interactive alternating* IMC)—where the order of steps 2 and 3 can be swapped. As a result we end up in a strictly alternating IMC which directly corresponds to a CTMDP. The terminology used here to name the various intermediate models is inspired by Hansson [41].

We use the IMC depicted in Fig. 7(a) as an example to show the effect of the singular transformation steps. This example is taken from [10]. Here, state s_1 (light grey) is a hybrid state, s_6, s_7 and s_8 are Markov states (grey), and all other states are interactive states (white).

Step (1): Alternating IMC: An alternating IMC does not possess any hybrid states anymore. Whenever s was a hybrid state in the closed IMC $\mathcal{I}$, s is interpreted as interactive state in the alternating counterpart of $\mathcal{I}$, i. e., all of its emanating Markov transitions are cut off, and the Markov transition relation is changed to $(S_M \times \mathbb{R}^+ \times S) \cap R$. This is justified by the *urgency assumption* that is imposed on the closed IMC $\mathcal{I}$ that is subject to the transformation. The alternating IMC of the example IMC is depicted in Fig. 7(b). In general, certain parts of the LTS may become unreachable as a result of this step, which in practice often shrinks the state space to be considered in subsequent steps drastically.

Step (2): Markov alternating IMC: Turning an alternating IMC into a Markov alternating IMC requires splitting the sequences of Markov transitions, as follows. Suppose $s, s' \in S_M$ and $s \overset{\lambda}{\dashrightarrow} s'$. In order to break this sequence of Markov states, we introduce a fresh interactive state (s, s') which is connected to s via $s \overset{\lambda}{\dashrightarrow} (s, s')$. State (s, s') in turn is connected via $(s, s') \xrightarrow{\tau} s'$ to s'. This yields the following transformation step. For a given alternating IMC $\mathcal{I} = (S = S_M \,\dot\cup\, S_I, Act, T, R, s_0)$, we define its *Markov alternating* counterpart as IMC (S', Act, T', R', s_0) with

- $S' = S \,\dot\cup\, \{(s, s') \in S_M \times S_M \mid \exists \lambda \in \mathbb{R}^+ : (s, s') \in R\}$,
- $T' = T \,\dot\cup\, \{((s, s'), \tau, s') \in S' \times \{\tau\} \times S_M \mid \exists \lambda \in \mathbb{R}^+ : (s, s') \in R\}$,
- $R' = R \cap (S_M \times \mathbb{R}^+ \times S_I) \,\dot\cup\, \{(s, \lambda, (s, s')) \in S_M \times \mathbb{R}^+ \times S' \mid (s, s') \in R\}$.

We illustrate this transformation step in Fig. 7(c) where $s'_6 = (s_6, s_7)$ and $s'_7 = (s_7, s_8)$ are the freshly inserted states.

Step (3): Interactive alternating IMC: We now handle sequences of interactive transitions ending in a Markov state. To compress these sequences, we calculate the transitive closure of interactive transitions for each interactive state s that (is either the initial state of the IMC or) has at least one Markov predecessor. The computation is carried out in a way such that we get all Markov successors of s that terminate these sequences. We label the resulting compressed transitions with words from the alphabet $A^+ \,\dot\cup\, \{\tau\}$ (also denoted $Words$). Interactive states that do not have any Markov state as a predecessor will not be contained in the resulting interactive (or strictly) alternating IMC any more. These states violate the strict alternation of interactive and Markov states and therefore will not be contained in the CTMDP.

For Markov alternating IMC $\mathcal{I} = (S, Act, T, R, s_0)$ we define its *strictly alternating* counterpart as IMC $(S', Words, T', R, s_0)$, with

- $S' = S_M \,\dot\cup\, S'_I$ where $S'_I = \{s \in S_I \mid \exists t \in S_M : t \overset{\lambda}{\dashrightarrow} s$, for some $\lambda \in \mathbb{R}^+\} \cup \{s_0\}$,
- $T' := \{(s, W, t) \in S'_I \times Words \times S_M \mid s \overset{W}{\Longrightarrow} t\}$.

This yields a strictly alternating IMC. So, after applying steps (1)–(3) to IMC $\mathcal{M}$ we obtain an IMC $\mathcal{M}'$ which is strictly alternating and where each interactive transition is labelled by a word W from the alphabet $A^+ \,\dot\cup\, \{\tau\}$. The strictly alternating IMC $\mathcal{M}' = (S = S_I \,\dot\cup\, S_M, Words, \longrightarrow, \dashrightarrow, s_0)$ can now be interpreted as a CTMDP $\mathcal{C}_{\mathcal{M}} = (S_I, Words, \mathbf{R}, s_0)$ where $\mathbf{R} := \{(s, W, R) | R(s, s') = \sum_{i=1}^{n} \lambda_i$ iff $\exists u \in S_M, \lambda_i \in \mathbb{R}_{\geq 0}$ such that $s \xrightarrow{W} u \wedge u \overset{\lambda_i}{\dashrightarrow} s', i = 1, 2 \ldots, n\}$.

Interpretation: A strictly alternating IMC can directly be interpreted as a CTMDP. Fig. 7(e) depicts the corresponding CTMDP of the strictly alternating IMC in Fig. 7(d).

Example 9 Once the time-constraints described in the previous example are woven into the quotient LTS of the heater example, we obtain a monolithic uIMC. Using the CTMDP transformation, the uIMC can be converted into a uCTMDP. The resulting uCTMDP

[2]The uniform CTMC is—strictly speaking—not an absorbing one, since the state s_a is now equipped with an e-loop. Nevertheless, the time to hit this state s_a is still distributed according to PH_d.

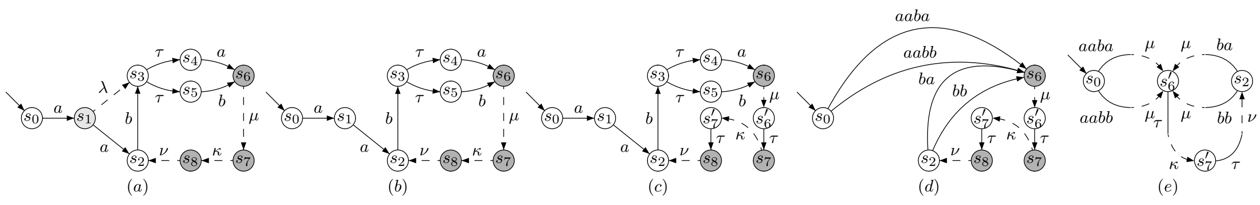


Fig. 7. Transformation: (a) example IMC, (b) alternating IMC, (c) Markov alternating IMC, (d) strictly alternating IMC, and (e) the corresponding CTMDP.

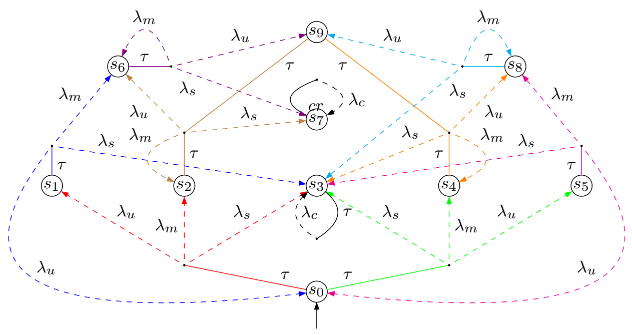


Fig. 8. uCTMDP of the heater example.

is depicted in Fig. 8, where $\lambda_s = \lambda_{FS}$, $\lambda_m = \lambda_{FM}$, $\lambda_u = \lambda_{UP}$ and $\lambda_c = \lambda_{FS} + \lambda_{UP} + \lambda_{FM}$. The safety-critical state s_7 is marked by a self-loop labelled *cr*.[3]

The *non-determinism*, modelling the unpredictable environmental temperatures (cf. input variable TEMP_IN in the Statechart), is represented in the states s_0 and s_9. Intuitively these non-deterministic choices embody the temperature update to either TOO_HOT or TOO_COLD. Sink state s_3 can be interpreted as a safe state: either the heater has performed a safe shutdown, or simply does not heat, as the SENSOR is stuck already at temperature TOO_HOT.

As indicated in the introduction to this section, the transformation preserves uniformity. To establish this formally and to clarify more detailed correspondences of the two models, we fix IMC $\mathcal{I} = (S^{\mathcal{I}}, Act, \longrightarrow, \dashrightarrow, s_0)$ and let CTMDP $\mathcal{C} = (S^{\mathcal{C}}, Words, \mathbf{R}, s_0)$ denote the result of the transformation.

Theorem 3 *If $\mathcal{I}$ is uniform then $\mathcal{C}$ is uniform with the same rate.*

This theorem is shown in [13]. Beside uniformity, the transformation preserves the timed probabilistic behaviour in a very strict sense. In particular, for each scheduler over $\mathcal{I}$ there exists a scheduler over the corresponding $\mathcal{C}$ such that probabilities on measurable sets of paths agree. Path correspondence is defined via a mapping Ψ, which for a given path σ in $\mathcal{I}$ identifies the unique corresponding path $\Psi(\sigma)$ in CTMDP $\mathcal{C}$.

Theorem 4 *(1) For each scheduler D over $\mathcal{I}$ there exists a scheduler D' over $\mathcal{C}$ such that for all measurable sets P of paths (in $\mathcal{I}$)*

$$Pr_{\mathcal{I},D}(P) = Pr_{\mathcal{C},D'}(\Psi(P)) \ .$$

(2) For each scheduler D over $\mathcal{C}$ there exists a scheduler D' over $\mathcal{I}$ such that for all measurable sets P of paths (in $\mathcal{C}$)

$$Pr_{\mathcal{C},D}(P) = Pr_{\mathcal{I},D'}(\Phi(P)) \ .$$

We refer to [13] for a detailed proof.

H. Timed Reachability Analysis

The model obtained after performing the transformation described above is a uniform CTMDP, since the input IMC is uniform. Our aim is to calculate the worst-case probability of reaching any of the safety-critical states within a given time bound.

For CTMCs, the corresponding question can be reduced to an instance of transient analysis [42], for which efficient and numerically stable iterative algorithms are known, based on *uniformisation*. Timed reachability analysis of stochastic systems with non-determinism is not that straightforward. For uniform CTMDPs this problem was tackled in [4]. For a uniform CTMDP $\mathcal{C}$ with uniform rate E we aim at calculating the maximal probability to reach a given set of states B within t time units from a particular state s in $\mathcal{C}$ w. r. t. all schedulers $D \in Sched$. We denote this by

$$\sup_{D \in Sched} Pr_{\mathcal{C},D}(s, \overset{\leq t}{\rightsquigarrow} B),$$

where $s, \overset{\leq t}{\rightsquigarrow} B$ denotes the measurable set of paths starting in s and hitting a state $s' \in B$ within t time units. [4] studies the problem of approximating this probability for $Sched$ being the class of all *untimed history-dependent schedulers* that may use *randomisation*. The algorithm is based on three observations: (1) randomisation does not add to the power of the schedulers, (2) history-dependence only adds in the form of step-dependence. A step-dependent scheduler only counts state changes instead of recording the entire history. Further, observation (3) is that the step-dependence is only decisive up to a specific depth k which can be precomputed on the basis of E, t and the accuracy ε of the approximation.

Thus, it is sufficient to consider non-randomised k-truncated step-dependent scheduler $D : S \times \{0, \ldots, k\} \mapsto L$. Unfortunately, the number of such schedulers can be exponential in the value of k. However, in order to derive the maximal value of $Pr_{\mathcal{C},D}(s, \overset{\leq t}{\rightsquigarrow} B)$, the actions to be selected by a (worst-case) scheduler D can be computed by a greedy backward strategy. Due to space constraints we refer to [4] for an elaborate discussion of this greedy algorithm, which is linear in k and linear in the size of L. The algorithm returns for each state the worst-case probability to reach a state $s \in B$ within time t.

We apply this algorithm to the uniform CTMDP $\mathcal{C} = (S, L, \mathbf{R}, s_0)$, where the set of goal states $B \subseteq S$ corresponds to the set of safety-critical states. By looking up the probability returned for the initial state s_0, we finally arrive at the worst-case probability to reach a safety-critical state within time t for the system in question.

[3]Recall that all safety-critical states in the symbolic part are, in the explicit part, decorated with a self-loop labelled *cr*. This encoding preserves the relevant information, and is needed because the latter format is strictly transition-oriented, and does not allow information to be directly attached to states. We remark that this strictness is what enables our compositional approach, because state identities can be considered entirely irrelevant, the entire information is in the transition structure. A more elaborate discussion of the issue of state vs. transition labelling in the context considered here can be found in [10].

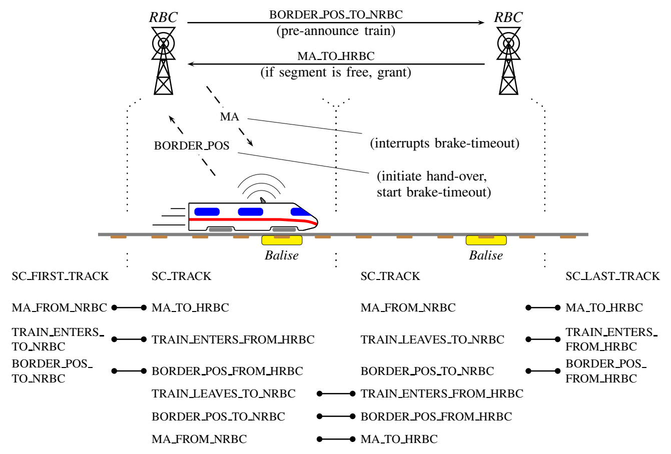


Fig. 9. ETCS: RBC hand-over scenario and model architecture.

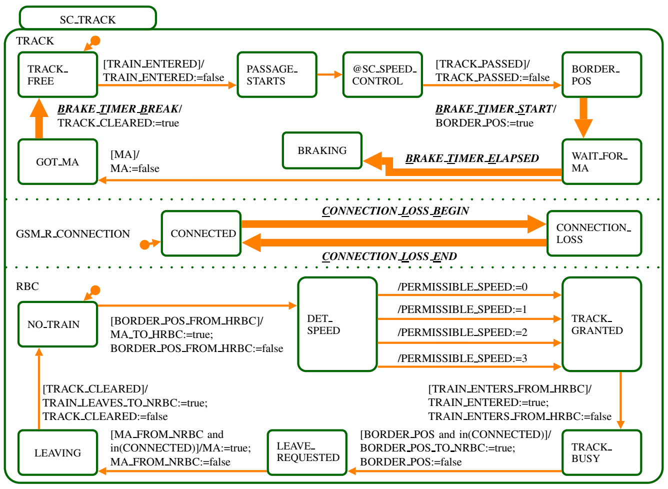

SC_TRACK
TRACK
TRACK_FREE
[TRAIN_ENTERED]/ TRAIN_ENTERED:=false
PASSAGE_STARTS
@SC_SPEED_CONTROL
[TRACK_PASSED]/ TRACK_PASSED:=false
BORDER_POS
BRAKE_TIMER_BREAK/ TRACK_CLEARED:=true
BRAKE_TIMER_START/ BORDER_POS:=true
GOT_MA
[MA]/ MA:=false
BRAKING
BRAKE_TIMER_ELAPSED
WAIT_FOR_MA
GSM_R_CONNECTION
CONNECTED
CONNECTION_LOSS_BEGIN
CONNECTION_LOSS_END
CONNECTION_LOSS
RBC
NO_TRAIN
[BORDER_POS_FROM_HRBC]/ MA_TO_HRBC:=true; BORDER_POS_FROM_HRBC:=false
DET_SPEED
/PERMISSIBLE_SPEED:=0
/PERMISSIBLE_SPEED:=1
/PERMISSIBLE_SPEED:=2
/PERMISSIBLE_SPEED:=3
TRACK_GRANTED
[TRACK_CLEARED]/ TRAIN_LEAVES_TO_NRBC:=true; TRACK_CLEARED:=false
[TRAIN_ENTERS_FROM_HRBC]/ TRAIN_ENTERED:=true; TRAIN_ENTERS_FROM_HRBC:=false
LEAVING
[MA_FROM_NRBC and in(CONNECTED)]/MA:=true; MA_FROM_NRBC:=false
LEAVE_REQUESTED
[BORDER_POS and in(CONNECTED)]/ BORDER_POS_TO_NRBC:=true; BORDER_POS:=false
TRACK_BUSY

Fig. 10. Model of a single track segment.

Example 10 We continue the heater example and set $\lambda_{FS} = 0.005$, $\lambda_{FM} = 0.0005$ and $\lambda_{UP} = 2$. Thus, on average the sensor fails once in 200 time units, the monitor fails once in 2000 time units, while the updates commence once a time unit. In this case, the worst-case probability for the heater to reach the safety-critical state s_7 (in which the heater heats forever) within 1000 time units $(\sup_{D\in Sched} Pr_{Heater,D}(s_0, \overset{\leq 1000}{\rightsquigarrow} s_7))$ is calculated to be 0.0878864.

It is worth noting that this algorithm requires the CTMDP to be uniform. Intuitively, the reason is that in uniform CTMDPs with uniform rate E, jumps occur on average after $^1/_E$ time units, regardless of the state considered, while in non-uniform CTMDPs the average time between two jumps varies from state to state, and thus the precise history of visited states provides more information about the estimated time that has elapsed, than just counting the number of steps. We refer to [4] for a non-uniform CTMDP example where this fact is exploited to construct a history-dependent scheduler which is—with respect to timed reachability—strictly more powerful than any step-dependent one.

IV. Case Study

This section demonstrates the application of our tool chain to an example taken from the context of the upcoming European train control system (ETCS) standard. Our purpose is to study and demonstrate the strength and limitations of the tool chain, therefore we deviate in some aspects from the standard [43] and set the focus on the STATEMATE design's scalability.

A. Description

ETCS and GSM-R (Global System for Mobile communications–Railway, an adaptation of the GSM wireless protocol) are designed to replace the multitude of incompatible safety systems used by European railways and to enable safe and fast transnational railway service. In application level 2 of the upcoming ETCS standard, trains report their exact position and direction of travel via GSM-R to the so-called radio block centres (RBC). These RBCs monitor the train movements, grant (or deny) 'Movement Authorities' (MA) for a particular track segment and provide additional route-related information, such as the permissible speed to the trains. An RBC has two roles: as a neighbour RBC (NRBC) to the train in the track segment prior to its scope and as a hand-over RBC (HRBC) to the train in the track segment inside its scope. Balises, mounted in the track bed, are used as electronic milestones to indicate special track regions, such as border positions that mark the end of an RBC's scope. Here NRBC takes over the responsibility of an approaching train from the current HRBC. In this hand-over situation, the HRBC acts as an agent between the NRBC and the train. In particular, it has to request a valid MA for the subsequent track segment from the NRBC and then to forward the MA, once it is granted, to the train. A snapshot of this hand-over is depicted in Fig. 9.

In this case study, we examine the braking probabilities of trains in such hand-over situations: As soon as a train reaches a border position balise, it requests an MA for the track segment controlled by the neighbour RBC via GSM-R. The train now waits for a new MA for the subsequent track segment within a certain time interval. If during this interval, the MA is not received, the train must start a braking manoeuvre. The braking probabilities are influenced by the interplay between (i) the connection losses in the GSM-R-based MA communication and (ii) the variation of permissible speeds of the trains.

B. Modelling

The STATEMATE model consists of the Statecharts depicted in Fig. 10 and 11. The lower part of Fig. 9 shows the mappings applied to combine these Statecharts to the overall model. For example, when connecting an instance of the `SC_FIRST_TRACK` with a `SC_TRACK` chart, leaving the scope of the former means entering the scope of the latter. Hence the mapping from `TRAIN_ENTERS_TO_NRBC` to `TRAIN_ENTERS_FROM_HRBC`. The delay distributions to be incorporated into the model are listed in Table II.

The Statechart `SC_TRACK`, divided into three concurrent components, captures the model's core behaviour. Initially the track segment is free and the RBC is idle (`NO_TRAIN`). In this state an MA request for an approaching train (`BORDER_POS_FROM_HRBC`) is granted (`MA_TO_HRBC:=true`) immediately. Afterwards the permissible train speed is chosen (`/PERMISSIBLE_SPEED:=0,1,2,3`) non-deterministically. As soon as the approaching train leaves the subsequent track segment (`TRAIN_LEAVES`), the RBC is busy[4]

[4]Note the mapping from `TRAIN_LEAVES_TO_NRBC` to `TRAIN_ENTERS_FROM_HRBC` (cf. Fig. 9.)

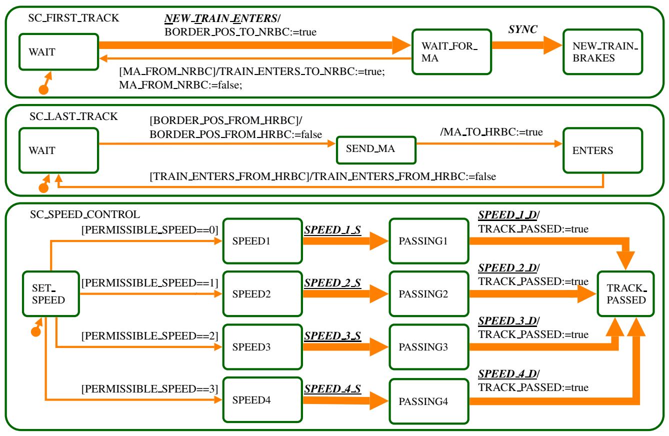


Fig. 11. From top to bottom: model of (a) the first, (b) the last track segment, and (c) the speed control unit.

(TRACK_BUSY). The track is now passed by the train in accordance to the chosen permissible speed (SC_SPEED_CONTROL, delays B1–B4 in Table II), until the border position is reached (BORDER_POS). The train then informs the RBC (BORDER_POS) that it is in the border position; it starts a brake timer (BRAKE_TIMER_START, delay C1) and waits for a new MA (MA). If the timer elapsed (BRAKE_TIMER_ELAPSED) before the MA is received, the critical state (BRAKING) is reached, otherwise the timer is reset (BRAKE_TIMER_BREAK).

The reception of the MA request by an RBC (BORDER_POS) depends on the state of the GSM-R connection (BORDER_POS and in(CONNECTED)). This connection is modelled in the GSM_R_CONNECTION component. From time to time (delay A1) the connection may be lost (CONNECTION_LOSS) for a certain amount of time (delay A2). Furthermore, in order for an MA to be granted before the train starts braking, the next track segment has to be cleared by the preceding train. Therefore the speed of the train in that track segment is the second parameter influencing the overall probability for observing a train braking.

Table II shows the parameters of the elapse operator. There are eight types of time-constraints used in this case study, named according to the first column. The starting, delay and breaking transitions associated with these time-constraints are listed in columns 2–4, respectively. The names of these transitions are shorthands of those in Fig. 10 and 11, given by the underlined characters.

We use two sets of phase-type distributions for the delays of the time-constraints, denoted by “Expr. 1” (shorthand for Experiment 1) and “Expr. 2” in columns 5–6 of Table II. Some of the delays are distributed according to exponential distributions $Exp(\lambda)$, while others are given by deterministic distributions or fixed delays. The latter are approximated by Erlang distributions $Erl(\lambda, k)$. Note that the phase-type distributions used for each time-constraint in both experiments have the same mean value. The Erlang distributions, however, have less variance.

C. Statistics

In this section, we provide some statistics that were obtained from experiments on the ETCS case study where we vary the number of track segments (Tracks) and speed profile choices (Choices). Experiments related to the STATEMATE-plugin were carried out on a PC with P4 2.66 GHz processor with 1 GB RAM running

TABLE II

PHASE-TYPES AND OTHER PARAMETERS OF ELAPSE OPERATION

Name	Starting	Delay	Breaking	PH of Expr. 1	PH of Expr. 2
A1	INIT, C_L_E	C_L_B	-	$Exp(0.001)$	$Exp(0.001)$
A2	C_L_B	C_L_E	-	$Exp(10.0)$	$Exp(10.0)$
B1	SPEED_1_S	SPEED_1_D	-	$Exp(1.0)$	$Erl(5.0, 5)$
B2	SPEED_2_S	SPEED_2_D	-	$Exp(1.5)$	$Erl(7.5, 5)$
B3	SPEED_3_S	SPEED_3_D	-	$Exp(2.0)$	$Erl(10.0, 5)$
B4	SPEED_4_S	SPEED_4_D	-	$Exp(2.5)$	$Erl(12.5, 5)$
C1	B_T_S	B_T_E	B_T_B	$Exp(3.0)$	$Exp(3.0)$
D1	INIT, N_T_E	N_T_E	-	$Exp(0.4)$	$Exp(0.4)$

Windows XP SP2. All other experiments were run on PCs with P4 2.66 GHz processor with 2 GB RAM running Linux 2.6.15-1-k7.

1) Symbolic Transformation: In Table III we show the results for the symbolic translation and minimisation steps.[5] The table lists both the number of bits necessary to encode the state space and transitions (columns Potential *s bits* and *t bits*) as well as the actual size of the generated LTS (reachable *states* and *transitions*). The number of bits in the *Potential* columns corresponds to a state space size of 2^s states and 2^{s+t} transitions.

In order to emphasise the importance of the cone-of-influence (COI) reduction prior to the translation into an LTS we also performed additional experiments where the COI reduction was disabled. From this column it is evident that the LTS size is only marginally impacted but nevertheless it is an essential step in the overall translation process. As can be seen from the *Time* column, even relatively small models cannot be handled without COI reduction since the intermediate models used during the LTS computation become too large.

The columns in the middle of the table show the size of the actual models that were used in subsequent experiments and those to the right show the size of the LTS after the symbolic branching bisimulation minimisation and the time needed to compute it.

2) IMC Construction: In Table IV, we report the result related to the compositional construction and minimisation of the IMCs. For each model, we provide the size of the *largest intermediate state space* we have to handle when weaving the time-constraints to the minimised LTS model and the computation time (in seconds) required to generate and minimise all intermediate models until the final model is obtained. The sizes of the state spaces of the final models are also provided. The table is divided into two parts, which correspond to the two types of experiments specified in Table II.

For both types of experiments, the state space and the computation time increase with the number of track segments and speed choices. The largest model we handle in the case study is the 4 tracks and 2 choices case in the second experiment. The final IMC has 416274 states and it takes around 68 hours to compose.

3) CTMDP Transformation: In Table V, we present the result related to the transformation from IMC to CTMDP. We provide the number of states and transitions for the resulting CTMDPs, together with the computation time (in seconds) required for the corresponding transformations. The sizes of the quotient IMCs input to these transformations are shown in the last two columns of Table IV. The column depicting the number of CTMDP transitions deserves a special comment. Since each transition in a CTMDP is a tuple (s, l, R) with the function R assigning rates to the successor states, representing one transition may, in the worst

[5]Each entry marked by “$*$)” in the table indicates that the corresponding experiment cannot be completed due to the length of time required for its computation.

TABLE III

SYMBOLIC STEPS: STATEMATE SAFETY ANALYSIS AND MINIMISATION STATISTICS

Tracks –	Model Size Without COI					Model Size With COI					Branching Bisimulation		
	Potential		Reachable		Time	Potential		Reachable		Time	Min. Result		Time
Choices	s bits	t bits	States	Trans.	(sec.)	s bits	t bits	States	Trans.	(sec.)	States	Trans.	(sec.)
1 – 1	37	18	102	397	24.7	37	16	102	157	2.3	36	87	0.05
1 – 2	45	20	203	650	2670.5	41	16	203	325	2.9	50	123	0.07
1 – 3	45	22	304	1010	5322.4	45	22	304	505	3.0	62	157	0.09
1 – 4	45	23	465	802	6945.7	45	23	405	697	9.0	74	191	0.10
2 – 1	64	33	1693	10315	867.8	64	29	1693	3451	16.3	411	1709	1.05
2 – 2	80	37	6769	114440	44970.8	72	29	6769	14305	413.1	852	3543	3.11
2 – 3	80	41	15229	533104	364276.6	80	41	15229	33319	994.8	1318	5655	5.37
2 – 4	*)	*)	*)	*)	*)	80	43	27073	61249	357.4	1880	8231	6.63
3 – 1	*)	*)	*)	*)	*)	91	42	28085	66857	164.6	4851	28455	42.59
3 – 2	*)	*)	*)	*)	*)	103	42	224673	553569	37.4	14616	84947	278.47
3 – 3	*)	*)	*)	*)	*)	115	60	758269	1931473	49.6	28208	169579	639.48
4 – 1	*)	*)	*)	*)	*)	118	55	457097	1205873	68.1	57381	434213	3555.73
4 – 2	*)	*)	*)	*)	*)	134	55	7313537	19953537	6960.6	250444	1861767	143334.70
4 – 3	*)	*)	*)	*)	*)	150	79	37024777	104353921	2923.8	*)	*)	*)

TABLE IV

EXPLICIT STEPS: COMPOSITION AND MINIMISATION STATISTICS

Expr.	Tracks –	Compositional Construction			Final Quotient IMC	
	Choices	States	Transitions	Time (sec.)	States	Transitions
1	1 – 1	71	261	16.95	8	23
	1 – 2	59	225	20.40	13	37
	1 – 3	71	275	23.78	15	47
	1 – 4	83	325	27.11	17	57
1	2 – 1	475	2640	27.62	35	150
	2 – 2	999	5577	34.65	105	402
	2 – 3	1440	8130	41.61	153	655
	2 – 4	2099	12109	49.01	209	964
1	3 – 1	5683	41632	39.91	189	1060
	3 – 2	17151	125035	61.69	922	4427
	3 – 3	33007	242643	100.24	1680	9056
1	4 – 1	67237	610938	103.21	1061	7254
	4 – 2	306699	2855455	3372.16	8450	47734
2	1 – 1	71	261	18.70	16	55
	1 – 2	79	325	22.58	29	101
	1 – 3	99	419	26.87	39	143
	1 – 4	119	513	30.43	49	185
2	2 – 1	731	3888	31.72	187	1006
	2 – 2	1755	11059	39.55	665	3442
	2 – 3	3127	20737	47.71	1281	6991
	2 – 4	4899	33495	57.83	2097	11780
2	3 – 1	10075	75377	50.01	2573	18260
	3 – 2	53858	387501	293.53	16602	112011
	3 – 3	134555	1061958	1114.82	44880	320504
2	4 – 1	143641	1343747	785.30	35637	313270
	4 – 2	1350908	11619969	243687.33	416274	3452502

TABLE V

EXPLICIT STEPS: CTMDP TRANSFORMATION AND ANALYSIS STATISTICS

Expr.	Tracks –	Uniform CTMDP		Time	Analysis ($\sup_D \Pr_D(s, \overset{\leq t}{\rightsquigarrow} B)$ for different t) and Execution Time in μsec.					
	Choices	States	Transitions	(sec.)	$t = 1$	Time	$t = 5$	Time	$t = 10$	Time
1	1 – 1	13	25 (1.92)	0.36	0.0471218	192	0.3379318	156	0.5857440	193
	1 – 2	19	39 (2.05)	0.36	0.0471218	277	0.3379318	283	0.5857440	364
	1 – 3	23	51 (2.22)	0.36	0.0471219	349	0.3379318	381	0.5857440	500
	1 – 4	27	63 (2.33)	0.36	0.0471219	411	0.3379318	547	0.5857440	647
1	2 – 1	59	167 (2.83)	0.36	0.0549173	633	0.3835667	878	0.6575970	1257
	2 – 2	146	418 (2.86)	0.37	0.0483948	1980	0.3883743	3151	0.6621517	4539
	2 – 3	226	693 (3.07)	0.37	0.0486557	3529	0.3924282	5622	0.6665755	8422
	2 – 4	322	1032 (3.20)	0.38	0.0488844	5157	0.3957582	8971	0.6703472	14030
1	3 – 1	319	1159 (3.63)	0.37	0.0481474	3891	0.4036817	6821	0.6949978	10636
	3 – 2	1308	4637 (3.55)	0.46	0.0484857	23783	0.4139489	44175	0.7053435	70686
	3 – 3	2520	9590 (3.81)	0.55	0.0487916	53024	0.4224391	104475	0.7137807	174939
1	4 – 1	1787	7811 (4.37)	0.47	0.0481494	27827	0.4122475	56871	0.7166049	98012
	4 – 2	11902	49676 (4.17)	1.26	0.0484907	304608	0.4271817	632008	0.7321903	1088549
2	1 – 1	29	65 (2.24)	0.50	0.0574318	321	0.3702204	358	0.6196606	442
	1 – 2	51	119 (2.33)	0.48	0.0574318	545	0.3702204	704	0.6196606	991
	1 – 3	71	171 (2.40)	0.48	0.0574318	774	0.3702204	1196	0.6196606	1733
	1 – 4	91	223 (2.45)	0.48	0.0574318	1013	0.3702204	1747	0.6196606	2793
2	2 – 1	363	1175 (3.23)	0.50	0.0574456	3886	0.3885393	6396	0.6483349	9728
	2 – 2	1202	3938 (3.27)	0.54	0.0574755	15412	0.3962310	29214	0.6583843	47731
	2 – 3	2386	8037 (3.36)	0.57	0.0575228	35035	0.4037369	74749	0.6679403	132162
	2 – 4	3970	13576 (3.41)	0.65	0.0575811	65762	0.4101266	161831	0.6760732	293089
2	3 – 1	4991	20599 (4.12)	0.69	0.0574456	74233	0.3976218	146828	0.6669833	241163
	3 – 2	30028	124493 (4.14)	1.83	0.0574755	600453	0.4149666	1350805	0.6906316	2432469
	3 – 3	83616	357302 (4.27)	4.54	0.0575229	1984091	0.4311642	5261678	0.7111358	9628030
2	4 – 1	69115	345667 (5.00)	3.93	0.0574456	1403805	0.4010432	3128218	0.6794333	5612398
	4 – 2	751870	3764716 (5.00)	42.84	0.0574755	22297183	0.4251593	57290732	0.7149502	104880604

case, already require space in the order of the number of states. Of course, this is not the case: the functions are very sparse. The numbers denoted in brackets are the average number of non-zero entries per transition.

Overall, the transformation requires computation times in the order of seconds. For the biggest model, namely the 4 tracks with 2 choices case in the second experiment, the transformation runs for less than 42 seconds converting an IMC having 416274 states to a CTMDP having 751870 states.

4) CTMDP Analysis: In this section, we give the result of the reachability analysis applied to the case study. The reachability analysis basically provides us with the maximum probabilities that any train is forced to brake in any track segment within some duration of time. We supply several different settings for the analysis by varying the numbers of tracks and speed choices. A series of experiments is devoted to each of the set of time-constraints specified in Table II. For both of these series, the arrivals of a new train at the first track segment is governed by an exponential distribution with rate 0.4, which means on average a train arrives once every 2.5 time units. A time unit is the average time for a train to pass through one track segment, when it travels with the slowest speed (`SPEED_1`), which corresponds to the mean values of $Exp(1.0)$ and $Erl(5.0, 5)$.

In Table V columns 6–11, we summarise the obtained result. We report the braking probabilities for different time bounds t, i. e., for 1, 5, 10 time units. The runtime of the extended MRMC model checker in computing the probabilities (given in microseconds) is shown for each time bound. The table shows that the runtime grows according to the sizes of the CTMDPs and the time bounds. The reason why the time bounds affect the computation time is that in the uniformisation method used by the algorithm: larger time bound requires more iterations for the same error bound. For the biggest model, namely the 4 tracks with 2 choices case in the second experiment, the tool runs for around 104 seconds to obtain the probability for time bound $t = 10$.

Observing the obtained braking probabilities, we can conclude that, in general, the probabilities increase when either the track number, or the speed choice number, or the time bound is increased. The experiments also indicate that tuning the speed of the trains more precisely results in increased chances of braking, as shown by the uniformly higher braking probabilities in the second experiment than those of the first. Furthermore, the braking probabilities can be deemed large, since within 10 time units, which means that in average 4 trains have entered the first track, the probability that any one of them has to brake is more than one half in all experimental cases.

V. Conclusion

In this paper we demonstrated how to address dependability properties in a typical, functional behaviour-oriented industrial modelling environment. We defined extended Statecharts to enable the integration of real-time probabilistic phenomena, like failure occurrences, with non-determinism that typically arises in the specification of the functional behaviour. Thereby we avoided a coherency gap between the dependability oriented failure models and models explicating the functional behaviour of the system, at the same time extending an existing industrial modelling environment with quantitative (i. e. real-time probabilistic) behaviour.

We demonstrated how to analyse the specifications, taking advantage of recent advances in the area of both stochastic modelling and stochastic model checking. By combining compositional modelling and novel algorithmic analysis techniques, we arrive at an overall methodology to compute the worst-case dependability risk. The complexity challenges posed by the probabilistic verification problem could only be addressed by (1) performing a state space reduction on the non-deterministic part of the model by means of a *symbolic* minimisation capable of handling huge state spaces and (2) weaving stochastic time-constraints *after* this reduction into the model. A key enabler for this compositional approach was the selection of an adequate intermediate model that generalises both labelled transition systems (LTS) and continuous-time Markov chains: interactive Markov chains, which provides the necessary compositional operations while preserving uniformity which, in turn, allows for the use of existing stochastic model checkers. The developed technology was applied to a non-trivial case study from the train control domain with an explication of the improvements contributed by each of the relevant translation steps.

One may wonder why we did not compare our experimental results with simulation studies of the same system. This is an important question with a short answer: it is impossible to use simulation on the models we needed to consider. Simulation is only possible if the model is a stochastic process, which is not the case here. By fixing a particular scheduler it is possible to arrive at a specific instance of the final CTMDP which is a stochastic process (a CTMC), but we did not explore this, since there is no estimate how far the values obtained from simulating that instance differ from the worst-case result.

From a fault analysis perspective, our model-based approach is exact in the sense that our quantification takes into account the precise ordering of events on the critical paths leading to the safety-critical situation. This is superior to more naive analyses methods (like fault trees) that often are too pessimistic, since they cannot incorporate the ordering.

CTMDPs constitute a model class that is well studied from the operations research and artificial intelligence perspective. In this paper we focused on timed reachability properties for CTMDP, owed to its importance for dependability questions. However, the CTMDP models generated by our construction process are amenable to a variety of other analyses, for which algorithms are available. We can analyse steady state properties such as the long-run average (availability), the long-run expected reward rate, the long-run instantaneous reward or the long-run expected accumulated reward. Transient analyses that could be performed include expected reward rate per time unit, instantaneous reward, and expected accumulated reward. Most of these quantities require the model to be decorated with a reward structure, specifying the costs (or bonuses) for states and events. To derive such a decoration from an extended Statechart model is not difficult. Thus, our construction has paved the way for integrating also these analyses coherently in a functional behaviour-oriented, model-based development process.

Acknowledgements. We would like to thank Samuel Wischmeyer (Carl von Ossietzky University, Oldenburg) and Hubert Garavel (INRIA Rhônes-Alpes) for their valuable support during the preparation of this paper. The comments received during the reviewing process were a great help in improving the paper.

REFERENCES

[1] J. Hillston, *A compositional approach to performance modelling*. Cambridge University Press, 1996.

[2] D. Kartson, G. Balbo, S. Donatelli, G. Franceschinis, and G. Conte, *Modelling with Generalized Stochastic Petri Nets*. John Wiley & Sons, Inc., 1994.

[3] A. Hinton, M. Kwiatkowska, G. Norman, and D. Parker, "PRISM: A tool for automatic verification of probabilistic systems," in *Proc. of the 12th Int'l Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, ser. LNCS, vol. 3920. Springer, 2006, pp. 441–444.

[4] C. Baier, H. Hermanns, J.-P. Katoen, and B. R. Haverkort, "Efficient computation of time-bounded reachability probabilities in uniform continuous-time Markov decision processes." *Theoretical Computer Science*, vol. 345, no. 1, pp. 2–26, 2005.

[5] H. Hermanns, *Interactive Markov Chains and the Quest for Quantified Quality*, ser. LNCS vol. 2428. Springer, 2002.

[6] ARP4761, *Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment*. Society of Automotive Engineers: Aerospace Recommended Practice, 1996.

[7] E. Böde, M. Herbstritt, H. Hermanns, S. Johr, T. Peikenkamp, R. Pulungan, R. Wimmer, and B. Becker, "Compositional performability evaluation for STATEMATE," in *Proc. of the 3rd Int'l Conf. on the Quantitative Evaluation of Systems (QEST)*. IEEE Computer Society, 2006, pp. 167–178.

[8] H. Hermanns and S. Johr, "Uniformity by construction in the analysis of nondeterministic stochastic systems," in *Proc. of the 37th Annual IEEE/IFIP Int'l Conf. on Dependable Systems and Networks (DSN)*. IEEE Computer Society, 2007, pp. 718–728.

[9] R. Wimmer, M. Herbstritt, H. Hermanns, K. Strampp, and B. Becker, "Sigref – A symbolic bisimulation tool box," in *Proc. of the 4th Int'l Symp. on Automated Technology for Verification and Analysis (ATVA)*, ser. LNCS, vol. 4218. Springer, 2006, pp. 477–492.

[10] H. Hermanns and S. Johr, "May we reach it? or must we? in what time? with what probability?" in *Proc. of the 14th GI/ITG Conf. on Measurement, Modelling and Evaluation of Computer and Communication Systems (MMB)*. VDE Verlag, 2008, pp. 125–140.

[11] R. J. van Glabbeek and W. P. Weijland, "Branching Time and Abstraction in Bisimulation Semantics," *Journal of the ACM*, vol. 43, no. 3, pp. 555–600, 1996.

[12] N. Wolovick and S. Johr, "A characterization of meaningful schedulers for continuous-time Markov decision processes," in *Proc. of the 4th Int'l Conf. on Formal Modeling and Analysis of Timed Systems (FORMATS)*, ser. LNCS, vol. 4202. Springer-Verlag, 2006, pp. 352–367.

[13] S. Johr, "Model Checking Compositional Markov Systems," Ph.D. thesis, Saarland University, Germany, 2007.

[14] D. Harel and A. Naamad, "The STATEMATE semantics of statecharts," *ACM Transactions on Software Engineering and Methodology*, vol. 5, no. 4, pp. 293–333, 1996.

[15] D. Harel and M. Politi, *Modelling Reactive Systems with Statecharts: The STATEMATE Approach*. McGraw-Hill, 1998.

[16] W. Damm, B. Josko, H. Hungar, and A. Pnueli, "A compositional real-time semantics of STATEMATE designs," in *Revised Lectures from the Int'l Symp. on Compositionality: The Significant Difference (COMPOS)*, ser. LNCS, vol. 1536, 1998, pp. 186–238.

[17] J.-P. Katoen, M. Khattri, and I. S. Zapreev, "A Markov reward model checker," in *Proc. of the 2nd Int'l Conf. on Quantitative Evaluation of Systems (QEST)*. IEEE Computer Society, 2005, pp. 243–244.

[18] E. Clarke, O. Grumberg, and D. Peled, *Model Checking*. MIT Press, 1999.

[19] R. K. Brayton, G. D. Hachtel, A. L. Sangiovanni-Vincentelli, F. Somenzi, A. Aziz, S.-T. Cheng, S. A. Edwards, S. P. Khatri, Y. Kukimoto, A. Pardo, S. Qadeer, R. K. Ranjan, S. Sarwary, T. R. Shiple, G. Swamy, and T. Villa, "VIS: A system for verification and synthesis," in *Proc. of the 8th Int'l Conf. on Computer Aided Verification (CAV)*, ser. LNCS, vol. 1102, 1996.

[20] R. Drechsler and B. Becker, *Binary Decision Diagrams – Theory and Implementation*. Kluwer Academic Publishers, 1998.

[21] S. Blom and S. Orzan, "Distributed branching bisimulation reduction of state spaces," *Electronic Notes in Theoretical Computer Science*, vol. 89, no. 1, pp. 99–113, 2003.

[22] A. Bouali and R. de Simone, "Symbolic bisimulation minimisation," in *Proc. of the 4th Int'l Work. on Computer Aided Verification (CAV)*, ser. LNCS, vol. 663, 1992, pp. 96–108.

[23] S. Derisavi, "A symbolic algorithm for optimal Markov chain lumping," in *Proc. of the 13th Int'l Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, ser. LNCS, vol. 4424. Springer, 2007, pp. 139–154.

[24] J. R. Burch, E. M. Clarke, K. L. McMillan, and D. L. Dill, "Sequential circuit verification using symbolic model checking," in *Proc. of the 27th ACM/IEEE Conf. on Design Automation (DAC)*. ACM Press, 1990, pp. 46–51.

[25] Y. Matsunaga, P. C. McGeer, and R. K. Brayton, "On Computing the Transitive Closure of a State Transition Relation," in *Proc. of the 30th Int'l Conf. on Design Automation (DAC)*. ACM Press, 1993, pp. 260–265.

[26] R. Wimmer, M. Herbstritt, and B. Becker, "Optimization techniques for BDD-based bisimulation minimization," in *Proc. of the 17th ACM Great Lakes Symposium on VLSI*. ACM Press, 2007, pp. 405–410.

[27] M. Herbstritt, R. Wimmer, T. Peikenkamp, E. Böde, H. Hermanns, S. Johr, M. Adelaide, and B. Becker, "Analysis of large safety-critical systems: A quantitative approach," SFB/TR 14 AVACS, Reports of SFB/TR 14 AVACS 8, 2006, ISSN: 1860-9821, http://www.avacs.org.

[28] R. Wimmer, M. Herbstritt, and B. Becker, "Minimization of Large State Spaces using Symbolic Branching Bisimulation," in *Proc. of IEEE Work. on Design and Diagnostics of Electronic Circuits and Systems (DDECS)*, 2006, pp. 7–12.

[29] R. E. A. Khayari, R. Sadre, and B. R. Haverkort, "Fitting world-wide web request traces with the EM-algorithm." *Performance Evaluation*, vol. 52, no. 2–3, pp. 175–191, 2003.

[30] A. Thümmler, P. Buchholz, and M. Telek, "A novel approach for fitting probability distributions to real trace data with the EM algorithm." in *Proc. of the Int'l Conf. on Dependable Systems and Networks (DSN)*, 2005, pp. 712–721.

[31] M. F. Neuts, *Matrix-Geometric Solutions in Stochastic Models: An Algorithmic Approach*. Dover, 1981.

[32] C. Commault and J. P. Chemla, "On dual and minimal phase-type representations," *Communication on Statistics: Stochastic Models*, vol. 9, no. 3, pp. 421–434, 1993.

[33] M. A. Johnson and M. R. Taaffe, "The denseness of phase distributions," Purdue University, Purdue School of Industrial Engineering Research Memoranda 88-20, 1988.

[34] S. Asmussen, O. Nerman, and M. Olsson, "Fitting phase-type distributions via the EM algorithm," *Scandinavian Journal of Statistics*, vol. 23, no. 4, pp. 419–441, 1996.

[35] A. Horváth and M. Telek, "Phfit: A general phase-type fitting tool." in *Proc. of the 12th Int'l Conf. on Computer Performance Evaluation, Modelling Techniques and Tools (TOOLS)*, ser. LNCS, vol. 2324, 2002, pp. 82–91.

[36] R. Pulungan and H. Hermanns, "Orthogonal distance fitting for phase-type distributions," SFB/TR 14 AVACS, Reports of SFB/TR 14 AVACS 10, 2006, ISSN: 1860-9821, http://www.avacs.org.

[37] J. Abate, G. Choudhury, and W. Whitt, "Calculation of the GI/G/1 waiting time distribution and its cumulants from Pollaczek's formulas," *Archiv für Elektronik und Übertragungstechnik*, vol. 47, no. 5/6, pp. 311–321, 1993.

[38] K. Mitchell, J. Place, and A. van de Liefvoort, "Analytic modeling with matrix exponential distributions," *Simulation Councils Proceedings Series*, vol. 28, no. 1, pp. 201–204, 1996.

[39] H. Hermanns and J.-P. Katoen, "Automated compositional Markov chain generation for a plain-old telephone system," *Science of Computer Programming*, vol. 36, pp. 97–127, 2000.

[40] BCG_MIN, "Project Website," January 2008, http://www.inrialpes.fr/vasy/cadp/man/bcg_min.html.

[41] H. A. Hansson, *Time and Probability in Formal Design of Distributed Systems*. Elsevier Science Inc., 1994.

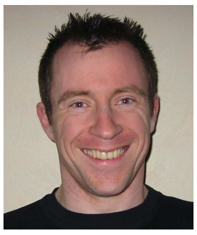
[42] C. Baier, B. R. Haverkort, H. Hermanns, and J.-P. Katoen, "Model-checking algorithms for continuous-time Markov chains." *IEEE Transaction on Software Engineering*, vol. 29, no. 6, pp. 524–541, 2003.

[43] "System Requirements Specification – Chapter 2 – Basic System Description," ALCATEL, ALSTOM, ANSALDO SIGNAL, BOMBARDIER, INVENSYS RAIL, SIEMENS, Tech. Rep., 2002.

Eckard Böde received his diploma in computer science from the Carl von Ossietzky University Oldenburg (Germany) in 2001. Since 2001 he has been working at the OFFIS research institute in Oldenburg. His research interests include the development and application of formal methods for the analysis of safety critical systems.

Marc Herbstritt received the diploma and the Ph.D. degree in computer science from Albert-Ludwigs-University, Freiburg (Germany), in 2000 and 2008, respectively. From 2004-2007, he worked at the German Transregional Collaborative Reasearch Center AVACS. His research interests focus on satisfiability-based verification methodologies, especially in the context of partial system designs. Since 2008, Dr. Herbstritt is editor for Applied Sciences and Computer Science at Birkhäuser Verlag AG, Basel (Switzerland).

Holger Hermanns studied at the University of Bordeaux, France, and the University of Erlangen/Nürnberg, Germany, where he received a diploma degree in computer science in 1993 (with honors) and a PhD degree from the Department of Computer Science in 1998 (with honors). From 1998 to 2006 he has been with the University of Twente, the Netherlands, holding an associate professor position since October 2001. Since 2003 he heads the Dependable Systems and Software Group at Saarland University, Germany. He has published more than 100 scientific papers, holds various research grants, and has co-chaired several international conferences including CAV, CONCUR and TACAS. His research interests include modeling and verification of concurrent systems, resource-aware embedded systems, and compositional performance and dependability evaluation.



Sven Johr received his diploma degree in computer science in 2002 from the RWTH Aachen, Germany. He received his Ph.D. degree (2007) from the Saarland University, Germany. Since the beginning of 2008 he is working in the research and development department of the company TWT Science & Innovation. He is currently involved in research projects on 3D search engines, micro-fabrication technologies and the 3D reconstruction of manufactured parts. Besides that, his research interests encompass stochastic systems with inherent non-determinism and stochastic learning techniques.

Thomas Peikenkamp has been working for more than 15 years on methods and tools for the development of safety-critical systems and about 8 years on model-based safety assessment methods. Currently he is leading the group for Safety Analysis & Verification in the R&D Division “Transportation” at OFFIS research institute.

Reza Pulungan received his bachelor’s degree from Universitas Gadjah Mada (Indonesia) and his master’s degree from University of Twente (the Netherlands). Since 2004, he has been a Ph.D. student in Dependable Systems and Software, Saarland University (Germany). His research interests lie in the field of stochastic processes, especially Markov processes and phase-type distributions. He is also interested in modelling and analysis of networked systems.

Jan Rakow received his diploma degree in computer science from the Carl von Ossietzky University, Oldenburg (Germany). Since 2006 he is working at the German Transregional Collaborative Research Center AVACS. His research interests include the development of verification techniques for and the modelling of safety critical systems, especially networked transportation systems.

Ralf Wimmer (IEEE student member) received his diploma degree in computer science from the Albert-Ludwigs-University, Freiburg (Germany) in 2004. Since 2005 he is working as a Ph.D. student at the German Transregional Collaborative Research Center AVACS in Freiburg. His research interests are applications of symbolic methods for stochastic verification.

Bernd Becker received a Diploma degree in Mathematics (1979), a Doctoral degree (1982) and a Habilitation degree in Computer Science (1988), all from Saarland University. Between 1979 and 1988, he was with the collaborative research center “Electronic Speech Recognition” (79-81), with the Chair for Computer Science and Applied Mathematics (81-83) and the collaborative research center “VLSI Design Methods and Parallelism” (84-88) at Saarland University. From 1989 to 1995, he was an Associate Professsor for “Complexity Theory and Efficient Algorithms” at J.W.Goethe-University Frankfurt. Since 1995, he has been with the Faculty of Applied Sciences at Albert-Ludwigs-University Freiburg as a Full Professor (Chair of Computer Architecture).

The research activities of Bernd Becker have been primarily in the area of computer-aided design, test and verification of (digital) circuits and systems (VLSI CAD). More recently, he has been working on verification methods for embedded systems and test techniques for nanoelectronic circuitry. He has published more than 200 papers in peer-reviewed conferences and journals and has been on the programme committees of numerous major international conferences. Bernd Becker was the General Chair of the IEEE European Test Symposium 2007. He is a fellow of IEEE.