Compositional Performability Evaluation for STATEMATE

Eckard Böde[3] Marc Herbstritt[2] Holger Hermanns[1] Sven Johr[1]
Thomas Peikenkamp[3] Reza Pulungan[1] Ralf Wimmer[2] Bernd Becker[2]

[1]Saarland University, Saarbrücken, Germany
[2]Albert-Ludwigs-University Freiburg im Breisgau, Germany
[3]Kuratorium OFFIS e.V., Oldenburg, Germany

***Abstract*— This paper reports on our efforts to link an industrial state-of-the-art modelling tool to academic state-of-the-art analysis algorithms. In a nutshell, we enable timed reachability analysis of uniform continuous-time Markov decision processes, which are generated from STATEMATE models. We give a detailed explanation of several construction, transformation, reduction, and analysis steps required to make this possible. The entire tool flow has been implemented, and it is applied to a nontrivial example.**

I. MOTIVATION

This paper reports a success story. It is the story of how we managed 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 toolset 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 [1] 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 stochastic 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) [2]. 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 representations, the earlier steps are symbolic, and cumulate in a novel symbolic branching bisimulation minimization algorithm.

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.

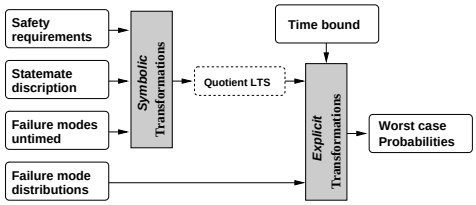

Fig. 1. Tool chain overview.

An overview of the tool chain is depicted in Fig. 1. Apart from a STATEMATE design, a collection of *failure modes* and *safety requirements* constitute the input of our symbolic transformations. *Failure distributions*, describing the likelihood of failure occurrences as time elapses, are another ingredient of the problem specification, but are only considered in the later, explicit stage of the tool chain, which finally yields the worst-case probability to violate the safety requirements at a given *time bound*.

We apply the entire tool chain to a parametric case study from the train control domain. For this example, we manage to avoid state spaces in the order of 10^{23}, and instead only need to handle models of up to 10^5 states and 10^6 transitions.

In summary, the paper makes the following contributions. It reports on (1) the first implementation of a time bounded reachability algorithm for uCTMDPs, (2) the first – to our knowledge – entirely BDD-based algorithm for computing branching bisimulation quotients, (3) a compositional method to construct uniform CTMDPs, (4) the integration of these pieces in a useable tool chain, and (5) the application of this tool chain to a nontrivial example.

Organization of the paper. The paper is organized as follows. Section II introduces some basic definitions. Section III explains the symbolic steps of our tool chain, while Section IV covers the subsequent explicit transformations. Section V describes how we implemented the entire tool flow, and Section VI demonstrates its practical feasibility for an example from the train control domain. Finally, Section VII concludes the paper.

II. MODEL BASICS

This section briefly reviews the most important concepts needed in this paper. The construction process revolves around

different flavors of *interactive Markov chains* [2], an orthogonal combination of labelled transition systems and continuous-time Markov chains. We consider a basic set of actions A which contains the distinguished action τ. This action is deemed unobservable and plays a crucial role in our approach, since it is used for abstracting behaviors 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, A, T, R) where S is a non-empty set of states, A is the above set of actions, $T \subseteq S \times A \times S$ is a set of interactive transitions, and $R \subseteq S \times \mathbb{R}^+ \times S$ is a set of Markov transitions.

By $R(s, s')$ we denote the transition rate from s to s', i. e., $R(s, s') = \lambda$ iff $(s, \lambda, s') \in R$ and 0 otherwise. A labelled transition system (LTS) is a triple (S, A, T) whenever $(S, A, T, \emptyset)$ is an IMC. A continuous-time Markov chain is a triple (S, A, R) whenever $(S, A, \emptyset, R)$ is an IMC.
Notation: For IMC $\mathcal{I} = (S, A, T, R)$, 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 *instable*. For stable state s and $C \subseteq S$ we define $\mathbf{r}(s, C) = \sum_{s' \in C} R(s, s')$. For instable s, $\mathbf{r}(s, C) = 0$. The distinction between stable and instable states is justified by the notion of maximal progress, see [2] 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 $\xrightarrow{a^*}$ for the reflexive transitive closure of $\xrightarrow{a}$. If $R(s, s') = \lambda > 0$ we will sometimes depict this as $s \overset{\lambda}{\dashrightarrow} s'$. For an equivalence relation B on S, we let S/B denote the set of equivalence classes of B.

Definition 2 (Stochastic Branching Bisimulation)
For a given IMC $\mathcal{I} = (S, A, T, R)$, 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 $(s_1, t_1) \in B$ then

1) $s_1 \xrightarrow{a} s_2$ implies
   either $a = \tau$ and $(s_2, t_1) \in B$,
   or $\exists t_1', t_2 \in S : t_1 \xrightarrow{\tau^*} t_1' \xrightarrow{a} t_2 \wedge (s_1, t_1') \in B \wedge (s_2, t_2) \in B$,
   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 [3] and stochastic weak bisimulation [2]. 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 $M = (S, A, T)$, an equivalence relation $B \subseteq S \times S$ is a *branching bisimulation* iff it is a stochastic branching bisimulation on $(S, A, T, \emptyset)$. Two states are *branching bisimilar* iff they are contained in some branching bisimulation.

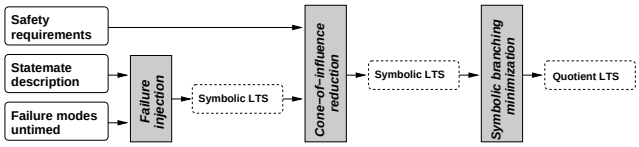

Fig. 2. Symbolic tool flow for shrink-fitting STATEMATE designs.

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

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 [1], 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 $\sum_{s' \in S} R(s') = e$.

III. FROM STATEMATE TO QUOTIENT LTS: SYMBOLIC TRANSFORMATIONS

In this section we describe the compilation of STATEMATE designs together with a specification of failure occurrences into a labelled transition system (LTS). Additionally, we describe how this LTS is built and manipulated by symbolic representations and manipulations, and we describe how the resulting symbolic representation, a BDD, is minimized with our novel branching bisimulation minimization algorithm.

A. Model Generation

The left part of Fig. 2 depicts the principal translation and analysis steps necessary to successfully perform the reduction. The main ingredients of the problem specification are: first, a STATEMATE model specifying the *nominal behavior* of the system; second, a number of *failures* that are injected, leading to a model encompassing also the dysfunctional behavior; and third, a specification of *safety-critical states*. The rationale for keeping the failures separated from the model is described in [4]. It can be considered as a compositional approach in an otherwise non-compositional formalism.
1) Compilation: In the first step of the compilation the STATEMATE model is translated into a simple intermediate language by replacing graphical items of the specification language by state variables, replacing structured data types by simple ones, and discretizing continuous variables (see [5] for details). The generated intermediate model represents the nominal behavior, and can be viewed as an LTS $M = (S, A, T)$.
2) Failure injection: In the second step, we inject failure behavior to our model by an instantiation of elements of a failure library. The behavior associated with these elements is parametrized so that, e. g., *stuck-at failures* have a parameter indicating the value to which a certain variable is stuck, once the failure occurs. The concrete values of these parameters are specified during instantiation. For the purpose of the considerations that follow, it is sufficient to know that the system extended with a set of failures F "includes" the original

system in the following sense: A labelled transition system $M' = (S, A', T')$ is called an F-extension of $M = (S, A, T)$ iff A' is the disjoint union of A and F, $T' \supset T$, and $\forall s \in S : \forall f \in F : \exists s' \in S : (s, f, s') \in T'$. (The compilation gurantees that the system deviates from the nominal behaviour if and only if a transition in $T' \setminus T$ is taken.) According to this definition, any failure can occur at any time; the rationale being that we may not exclude some occurrences of failures *a priori*.

3) Safety-critical states: After introducing the failure behavior, we need to identify safety-critical states, i. e., states where safety requirements are violated. In a nutshell, these states are characterized by a predicate over the basic states and the variable valuations of the design.

4) Cone-of-influence reduction and introduction of τ-actions: The predicate encoding safety-critical states is first used to reduce the size of the model by performing a cone-of-influence reduction (COI) with respect to these states [6]. This reduction is achieved by eliminating 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 associate small, property-specific models to large designs. It is implemented symbolically. To prepare for a further reduction step on the symbolic representation, we replace all nominal actions, i. e., labels not in F, by the unique τ action. In this way, the concrete nominal behavior is abstracted away, but its effect on the reachability of safety-critical states remains. In other words, the vulnerability of the nominal system with respect to failures is still present. We are then left with a model where only failures and τ-transitions are visible.

5) BDD generation of LTS: The previous reduction steps are carried out on a symbolic, but non-canonical representation of the system model. After the reduction steps the reduced model is translated into a canonical model with binary values only. The result is a BLIF-MV file suitable for the VIS model checker [7]. The latter is used to flatten the hierarchy of the design and to transform the net-list-like input format into a relational representation of the transition relation. Using VIS, we get a symbolic BDD-representation of our LTS. Further, the predicate representing safety-critical states is encoded into a BDD.

These two BDDs are then passed on 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-critical states) will be used as a starting point for this algorithm.

B. Symbolic Minimization

In this section we describe the *fully symbolic* implementation of a branching bisimulation algorithm that works entirely on BDDs. We assume that the reader is familiar with standard BDDs and the corresponding algorithms. For a comprehensive treatment see [8]. To shorten writing, we first introduce some notations. For LTS $M = (S, A, T)$ and partition P of S we write $\xrightarrow[P]{a}$ for a transition that is *inert* w.r.t. P, and $\xrightarrow[P]{a^*}$ for the reflexive transitive closure of $\xrightarrow[P]{a}$. Inert means that the source and target state of a transition are contained in the same block. By $P(s)$ we denote the block of P that contains the state s, i. e., $P(s) = \{t \in S \mid \exists B \in P : s \in B \wedge t \in B\}$.

Algorithm 1 Computation of the coarsest branching bisimulation

```
1: procedure BranchingBisimulation
2:     P_old ← ∅, P ← initial partition
3:     while P_old ≠ P do
4:         P_old ← P
5:         for all blocks B of P_old do
6:             P ← P \ {B} ∪ sigref_P(B)
7:     return P
```

In [9], Blom and Orzan present a novel approach for the distributed computation of branching bisimulation. Their algorithm is based on analyzing the *signatures* of states w.r.t. the current partition. The signature of a state is like a fingerprint identifying possible actions that can be executed in the state. To preserve branching bisimilarity, the unobservable action τ is taken into account by ignoring sequences of inert τ-actions. Let $P = \{B_0, \ldots, B_{p-1}\}$ be a partition of the state space S. The signature $\mathrm{sig}_P(s)$ of a state s w.r.t. partition P is formally defined as

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

Then, a refinement of a partition can be computed by grouping the set of states having the same signature:

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

The fixpoint algorithm of [9] is sketched in Algorithm 1. We extend the algorithm such that we can start with an initial partition provided by the BDD representation of the above-mentioned predicate separating safety-critical and non-critical states. Additionally, we integrate a simple, but efficient optimization technique, not applicable in the distributed algorithm of [9], that handles not all blocks altogether but takes one block at a time. For this block the signature refinement is computed and the corresponding result is updated *in situ* in the current partition. This *block forwarding* technique results in impressive speedups within our experiments due to the reduced number of iterations of the fixpoint algorithm (see [10] for details).

In the following we describe briefly how the algorithm of [9] can be lifted to BDDs. 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 action, and the target state, respectively. Additionally, we have a BDD $\mathcal{S}$ for the signatures with $\mathcal{S}(s, a, k) = 1$ iff $(a, B_k) \in \mathrm{sig}(s)$.

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, $\text{level}(s_i) < \text{level}(a_j)$ and $\text{level}(s_i) < \text{level}(k_l)$ hold for all i, j and l. This enables us to exploit the following observation. 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. Then, the sub-BDD at v is a representation of the signature of s. The point is that for all states having the same signature as s, the corresponding paths lead also to this BDD-node v. Therefore, to get 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.

Finally, after we have reached the fixpoint in Algorithm 1, we have to extract the quotient LTS from the final partition. This can be performed by mapping all states of a block onto one quotient state. The resulting model is 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 [10], [11] for more details and for experimental evaluations of this minimization algorithm.

IV. From Quotient LTS to CTMDP Analysis: Explicit Transformations

This section describes the timed behavior of failures modes, their approximation by phase-type distributions, and how we weave them into the minimized LTS obtained from the symbolic minimization phase (cf. Section III-B). We will first lay out the principal philosophy and its technical constituents, needed to integrate failure distributions in a compositional manner. Then we turn our attention to a specific fine point, namely that we intend to ensure that the result of our construction is a *uniform* IMC, and how we indeed ensure this. As a final step, we describe how we transform the uniform IMC into a uniform CTMDP, from which we extract the performability measure of interest, namely the worst-case probability to reach a safety-critical state within a given time bound.

A. Time constraints and composition

As noted in Section III-A, each failure mode is governed by a continuous probability distribution, describing the time-to-failure, a random variable corresponding to time up to which the occurrence of the failure is to be delayed, since the initialization of the system. Typical distributions in this context are Weibull, deterministic or exponential distributions, or distributions resulting from measurements of real equipment. In the current state of the implementation we consider non-repairable systems only, but this restriction is not a methodological one.

1) Phase-type Approximation: The approach we follow renders the model under study into a Markov model. To achieve this, we must represent the failure distributions provided to us into a Markov model. This is achieved by a nowadays widespread approach, which is based on *phase-type* approximation. A phase-type distribution is the distribution of the time until absorption in a finite and absorbing Markov chain [12]. The class of phase-type distributions thus consists of all serial, parallel and cyclic arrangements of exponential distributions. This means that the tractability and possibilities of explicit solutions that are encountered when dealing with exponential distributions are retained when phase-type distributions are substituted in their stead. Furthermore, the class of phase-type distributions is topologically dense [13]. 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 [14], [15], [16], [17].

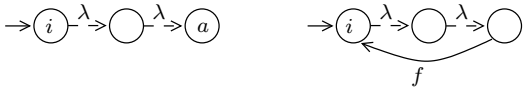

Fig. 4. A simple phase-type distribution (left) and the effect of elapse (right).

We now assume that the delay of each failure $f \in F$ in the system model is given by a phase-type distribution PH_f determining the time-to-failure. This phase-type distribution can be obtained by approximating the known failure mode distribution with one of the available algorithms. We implemented a variant thereof, based on orthogonal distance fitting [18]. The implementation ensures that the initial distribution is assigned to a single state.

2) Elapse: Structurally, PH_f is a continuous-time Markov chain (S, A, R) with a distinguished initial state i and absorbing state a. Operationally, the distribution PH_f can be viewed as describing the time up to which the occurrence of failure f has to be delayed, since the start of the system. This interpretation is a special case of what is called a *time constraint* in [19], where an *elapse* operator is introduced. This operator enriches PH_f with "synchronization potential" needed to effectively *weave* the Markov chain of PH_f into the behavior described by some LTS or IMC.

To provide some intuition of the semantics of this operator, we here discuss the effect of the *elapse* operator, which we denote by $\mathsf{elapse}(PH_f, f)$, on PH_f. Recall that PH_f is given by a MC (S, A, R) with distinguished initial state i and absorbing state a. Then, $\mathsf{elapse}(PH_f, f)$ will generate IMC $(S, A, \{a \xrightarrow{f} i\}, R)$, i. e., a transition labelled f now connects the absorbing and the initial state, see Fig. 4.

3) Weaving the time constraint: In $\mathsf{elapse}(PH_f, f)$, between any two occurrences of failure f, there must be a delay which is given by PH_f. To enforce this also for the LTS of our system under study, we weave this IMC with the LTS, where weaving is just another word for interleaving, with proper synchronization.

To this end, we use the process algebraic parallel composition operator. Intuitively, given IMCs $\mathcal{I}$ and $\mathcal{J}$, in $\mathcal{I}|[f]|\mathcal{J}$ both IMCs have to synchronize on f, while they interleave all other transitions. For $\mathcal{I} = \mathsf{elapse}(PH_f, f)$, this has the expected effect, namely that between any two f-transitions in $\mathcal{J}$, the MC associated with PH_f is weaved. The semantic rules of parallel composition of IMCs are as follows.

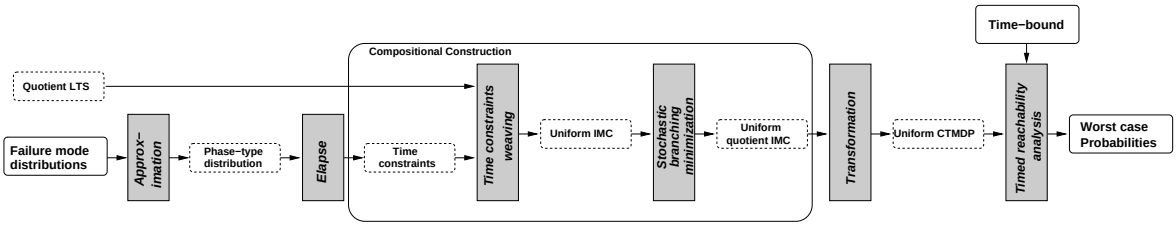

Fig. 3. Explicit tool flow for failure-distribution weaving and CTMDP analysis.

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

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

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

With this operator, and the elapse-operator, we can weave the failure distributions one-by-one with the original system. During this phase the IMC is explicitly represented and grows in size. One way of counteracting this is to minimize according to stochastic branching bisimulation. For this we use a stochastic branching bisimulation minimization algorithm [20], 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).

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

$$\frac{s \xrightarrow{\lambda} s'}{\text{hide } a \text{ in } (s) \xrightarrow{\lambda} \text{hide } a \text{ in } (s')}$$

In other words, we start from the initial explicit LTS as Sys_0, and incrementally build an IMC where the failure distributions are weaved, which is achieved by constructing

$$\text{hide } f_i \text{ in } (\text{elapse}(PH_{f_i}, f_i)|[f_i]|Sys_{i-1})$$

and minimizing the result with respect to stochastic branching bisimulation to form Sys_i. If we are dealing with n different failures, then the resulting IMC Sys_n does not contain any failure transitions anymore, but the failure distributions now interleave in the correct way governing the time to reach a safety-critical state.

The above approach alternates construction and minimization steps, and as such it deviates from the sequential procedure indicated in Fig. 3: it replaces the trajectory from *quotient LTS* and *time constraints* to *quotient IMC* by the one depicted in Fig. 5. 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.

B. Uniformity

So far we have ignored the word '*uniform*' which is attached to the IMC models appearing in Fig. 3. We recall that 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 will later understand why we are aiming at a uniform quotient IMC. To arrive there, we will ensure that our entire construction process preserves uniformity. Without going into the technical details, this is ensured due to the following properties [21]:

- The parallel composition of two uniform IMCs is a uniform IMC.
- Hiding of any action in a uniform IMC preserves uniformity.
- The stochastic branching bisimulation quotient of a uniform IMC is a uniform IMC.

Thus, to preserve uniformity by construction, we are left with the requirement that all our input models must be uniform. 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, A, R) be the CTMC of some phase-type distribution PH_f with initial state i and absorbing state a, and let $e = \max_{s \in S} \mathbf{r}(s, S)$. We associated a uniform CTMC by (S, A, R'), where $R'(s, s') = e - \mathbf{r}(s, S \backslash \{s\})$ if $s = s'$ and $R(s, s')$ 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 $1/e$ time units, regardless of the state considered[1].

For the example from Fig. 4, the uniform variants are obtained by equipping the rightmost states each 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.

C. CTMDP Transformation and Analysis

In Section IV-A timed behavior was incorporated in the system description turning the LTS into an IMC, and Section IV-B

[1]The uniform CTMC is – strictly speaking – not an absorbing one, since the a-state is now equipped with an e-loop. Still, the time to hit this state a is distributed according to PH_f.

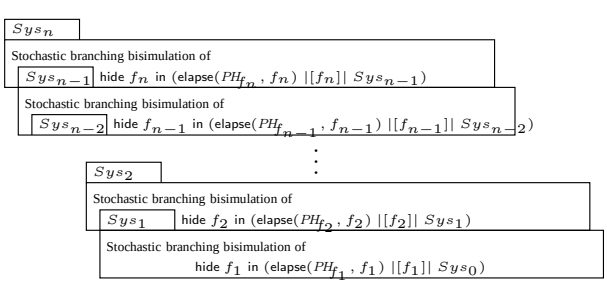

Fig. 5. Compositional weaving of phase-type distributions.

discussed how to ensure uniformity of the result. This section describes a transformation from uniform IMCs to uniform CTMDPs. We also detail how we analyze the resulting model, to distill worst-case probabilities of hitting a safety-critical state within a given time bound.

1) Transformation: The model we are dealing with is the complete description of the system under consideration, and can hence be viewed as a *closed system*. This means that the transformation to be carried out now is no longer compositional, which is justified by the fact that 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 [2]). Given an IMC $\mathcal{I} = (S, A, T, R)$, we can partition the set S into three disjoint sets of states. These are the sets of (1) *interactive states*, where no Markov transitions are possible, (2) *Markov states*, where only Markov transitions are possible, (3) *hybrid states* where at least one Markov and at least one interactive transition is possible.

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 state, and that they alternate strictly. We call this class of IMC *strictly alternating*.

We now sketch a transformation which turns any IMC into a strictly alternating one [22], while preserving the probabilistic behavior. The transformation involves (1) identifying Markov and interactive states, (2) breaking sequences of Markov states, (3) merging sequences of interactive states – 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 first step is achieved by implementing the urgency hypothesis: This means that we cut off all emanating Markov transitions from hybrid states, turning them into interactive states. Step (2) is straightforward: As interactive transitions are deemed to consume no time, an interactive transition can be inserted in-between any two consecutive Markov transitions. Step (3) is more involved, it is based on the transitive closure of the interactive transition relation. For each interactive state s that has at least one Markov state as direct predecessor and at least one interactive state as direct successor, we determine the Markov states which terminate all these sequences of interactive transitions. This means that the transitive closure on the interactive transition relation is calculated in a fashion that returns all of these Markov states. These states are used to define a strictly alternating IMC where interactive transitions are labelled by words of $(A \backslash \{\tau\})^+ \dot{\cup} \{\tau\}$ and always end in a Markov state. Interactive states whose direct predecessors are interactive states only (except the initial state) are removed from the resulting state space.

We note that step 3 destroys the branching structure of the IMC, and is as such not compositional. However, we can show that the entire transformation does not alter the probabilistic behavior of the output CTMDP relative to the input IMC. Formalizing this property requires the introduction of probability measures, trace distributions and schedulers for IMCs (and CTMDPs). Then, we can show that given an input IMC $\mathcal{I}$ together with scheduler D and its associated output CTMDP $\mathcal{C}$ with scheduler D' *corresponding* to D, it holds that the scheduler dependent probabilities of reaching a particular set B of states within t time units coincide. A formal proof of this property is not in the scope of this paper, but is presented in [21]. In addition, the transformation can be shown to preserve uniformity: If IMC $\mathcal{I}$ is uniform, then the associated CTMDP $\mathcal{C}$ is as well.

2) Timed Reachability Analysis: The model obtained after performing the transformation described above is a uniform CTMDP. 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 [23], for which efficient and numerically stable iterative algorithms are known, based on *uniformization*. Timed reachability analysis of stochastic systems with nondeterminism is not that straightforward. For uniform CTMDPs this problem was tackled in [1]. The canonical approach to associate a stochastic process to a stochastic system with nondeterminism uses a sufficiently general class of *scheduler*. A scheduler is a function that determines how to proceed next for a given state s. For a given state s, it resolves nondeterminism 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 randomization (i. e., probability distributions over enabled actions). In a timed model, the history of the system may even be a timed one. Intuitively, the more power (in terms of knowledge and randomness) a scheduler class $Sched$ provides, the more widely the resulting probabilities vary when ranging over all possible schedulers in $Sched$.

For a uCTMDP $\mathcal{C}$ with uniform rate E we aim to calculate 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

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

(and must refer to [1] for a precise definition of the uCTMC induced by D and the probability measure $\Pr_D$). [1] studies the problem of approximating this probability for $Sched$ being the class of all *untimed history-dependent schedulers* that may use *randomization*. In short, their algorithm is based on three observations: (1) randomization does not add to the power of the schedulers, (2) history-dependence only adds in the form of step-dependence, (3) 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-randomized 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, the authors show that in order to derive the maximal value of $\Pr_D(s, \overset{\leq t}{\rightsquigarrow} B)$, the actions to be selected by a (worst-case) scheduler D can be computed by a greedy backward strategy. For instance, $D(s, k)$ for $s \in S$ needs to

choose an action such that the probability to reach a B-state in one step is maximal. Due to space constraints we refer to [1] 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. By looking up the computed value for the initial state, we finally obtain the result we are looking for.

Given the uniform CTMDP $\mathcal{C} = (S, L, \mathbf{R})$, a set of goal states $B \subseteq S$, and a time point t, the algorithm approximates the vector $\Pr_D(\overset{\leq t}{\rightsquigarrow} B)$ containing state-wise maximal probabilities to reach B within time t. Thus, assuming that the set B of states corresponds to safety-critical states, the algorithm returns for each state the worst-case probability to reach a safety-critical $s \in B$ within time t. By looking up the probability for the initial state, we finally answer that question for the system studied.

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 [1] 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.

V. Implementation

This section explains how the tool flow (cf. Fig. 2) described in detail in the preceding two sections is put into practice by an interoperation of different toolkits: STATEMATE, CADP, and ETMCC.

The aforementioned symbolic manipulations have been implemented as a plugin for STATEMATE. After starting the tool on the model to be investigated the compilation (III-A.1) is performed. Then the user is allowed to enter failure modes (III-A.2) and safety-critical states (III-A.3) based on the given model. After having completed the problem specification the process continues with the symbolic cone-of-influence reduction and τ-relabelling (III-A.4) followed by the BDD generation (III-A.5). The subsequent symbolic minimization described in Section III-B constitutes the final step carried out by the STATEMATE-plugin. It generates an XML-representation of the quotient LTS, which can be either in symbolic form, or in explicit form. The latter enumerates all states and transitions, together with the initial state of the model. Each state in this explicit representation is decorated with a flag indicating whether the state is safety-critical or not.

The explicit part of the tool flow first transforms this XML-file into a file in the BCG-format[2]. The latter is a compact file-format for explicit representations of LTSs, and is the core format of the CADP toolkit. CADP is a construction and verification toolkit developed by the VASY-team of Hubert Garavel at INRIA Rhône-Alpes, and is strictly based on process algebraic principles [24]. It has been extended to enable compositional performance evaluation with IMC [25]. In particular, it provides genuine support for parallel composition and hiding on IMC, and it provides an efficient implementation of stochastic branching bisimulation, in the form of the tool-component BCGMIN [20].

Therefore, the compositional construction steps illustrated in Fig. 5 are performed by interaction with CADP: *time-constraint weaving* and *stochastic branching minimization*. To enable mechanized interaction, CADP provides a scripting language, SVL, which is particularly convenient to experiment with different strategies to alternate construction and minimization steps. Note that due to the considerations in Section IV-A.3, we can perform minimization after every construction or after some construction steps, which gives an interesting time-space tradeoff, further discussed in Section VI-C.

The phase-type approximation algorithm and the elapse-operator are implemented as stand-alone tools, which take the input failure distribution, the failure mode, and some further parameters (such as the number of phases used for approximation). They produce a uniform IMC, stored in BCG-format.

The final transformation to CTMDPs in turn takes a BCG-file of the final uniform IMC, and generates a uCTMDP in a format readable by the ETMCC model checker[3]. ETMCC is a CSL model checker for Markov chains [26]. We adapted the data structures and extend it with a sparse-matrix implementation of the timed reachability algorithm. This algorithm finally calculates the worst-case probability to reach the set of safety critical states within a user-specified time bound.

The entire tool flow is running in a prototypical form, and we are currently performing numerous experiments to identify bottlenecks and to improve interoperability. One of our latest experiments is reported in the next section.

VI. Case Study

This section applies our tool chain to an example taken from the context of the upcoming European train control system standard ETCS. At the current stage, the purpose of this example is to study and demonstrate the strength and limitations of the tool chain rather than providing new insight into the case which has been studied in [27], [28], [29]. Therefore, we deviate in some aspects from the true characteristics as laid out in the standard. Experiments related to the STATEMATE-plugin were carried out on a PC with P4 2.66 GHz processor with 1GB RAM running 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.

[2]All states in the XML-file flagged as safety-critical are in the BCG-file decorated with a self-loop labelled "unsafe". 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.

[3]In this transformation, all self-loops labelled "unsafe" which identify the safety-critical system states are collected in an explicit enumeration of the safety-critical states, thus re-assembling the predicate to be used in the final model.

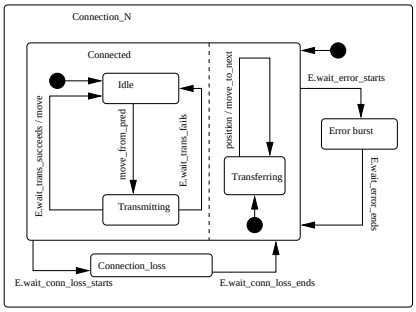

Fig. 6. Model of the Connection between the Train and the RBC.

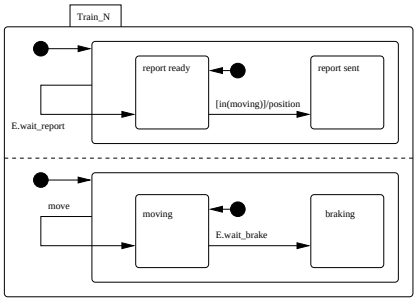

Fig. 7. Model of the Train Internals.

A. Description

In level 3 of the upcoming ETCS standard, high-speed trains will be allowed to follow each other at close distances. To assure safety in this mode of operation, the trains communicate with trackside "radio block centers" (RBCs), reporting their positions at periodic time intervals, and receiving the right to move on (so-called "movement authorities") from them at similar intervals. Communication between train and RBC is based on GSM-R, an adaptation of GSM wireless protocol. This communication infrastructure is error-prone and subject to failures which may cause delays in sending and receiving of critical messages.

Here we study the effects of these failures on the proper functioning and safety of the system. When constructing the model, one of the main concerns was scalability. The current implementation can handle arbitrary many consecutive trains on a single track. Here we report on instances of the model with up to four trains.

For our case, we assumed the RBC to operate as follows: It receives the current position of each moving train. To authorize a train to move on, it sends an authorization message. The idea is that the RBC only sends a moving authorization once it has received the position from the preceding train. Since a train is only allowed to send its new position if it is moving, each train can only move if the previous trains did already send a "move" before. A special case has to be observed for the first train, since there is no predecessor train the moving authorization for this train is always valid.

Several failures have been taken into account that can lead to faulty and unsafe behavior. For example, the communication between the RBC and the trains can be lost.

B. Modelling

1) STATEMATE *Description and Failure Modes:* In Fig. 6 and 7, some actions are prefixed with `E.wait` and some are not. All prefixed actions denote delayed actions. They are preserved during minimization and will later be associated with phase-type distributions. In the terminology of Section III, they serve as our *failure modes*.

Initially, the RBC is idle (state `IDLE`). Upon receiving a position information from the train in front, i.e., event `MOVE_FROM_PRED`, it tries to transmit a moving authorization. Depending on the environmental circumstances, this either fails or succeeds (conditions `TRANS_FAILS` or `TRANS_SUCCEEDS`). The moving authorization will be submitted as an event (`MOVE`) to the parallel state which represents the train. If a train successfully transmits its position report to the RBC, an affirmative signal (`MOVE_TO_NEXT`) is sent to the next train.

Two types of errors can disturb the communication between the RBC and the train. The occurrence of `ERROR_STARTS` indicates errors in the communication. The condition `CONN_LOSS_STARTS`, on the other hand, signals a connection loss. At the end of error and connection lost, the conditions `ERROR_ENDS` and `CONN_LOSS_ENDS`, respectively, are set.

The train consists of two parallel activities, which are modelled in STATEMATE by an AND-node (see Fig. 7). The lower node controls the movement of the train. Upon getting a `MOVE` event from the RBC, the train is in the `MOVING` state until the `BRAKE` condition is set. The train then waits in the `BRAKING` state until a new moving authorization arrives. The upper node controls the position reports. If the lower node is in state `MOVING`, a new position is reported (via the `POSITION` event). Afterwards, the train has to wait in the state `REPORT_SENT` for a new `REPORT` event, which indicates, that all necessary information for a new report has been collected. It then changes to the `REPORT_READY` state, from which it can send a new position report (provided that it is in the `MOVING` state).

2) Safety requirements: We consider all system states as *unsafe*, where the system occupies the node `BRAKING`.

3) Failure mode distributions: The failure mode distributions used are taken from [29], now interpreted for multiple trains. Some of the delays associated with the failure modes are distributed according to exponential distributions, others are given by deterministic distributions. The latter are approximated directly by Erlang distributions with n phases [30]. We made some experiments to understand the sensitivity of the numerical results and of the state space sizes on different values of n.

The delay of `TRANS_SUCCEEDS`, indicating the delay to establish a GSM-R connection, is at most 5 seconds with 95 % and at most 7.5 seconds with 99.9 % probability. We approximated this delay by our prototype tool. Fig. 8 depicts the resulting CTMC obtained. To simplify the figure, the chain is not

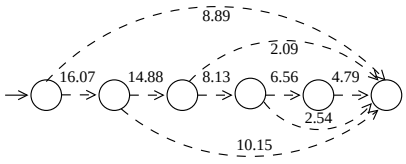

Fig. 8. Phase-type approximation of the delay of `TRANS_SUCCEEDS`.

uniform, i.e., self-loops are omitted.

C. Statistics

In this section, we give some statistics we obtained from experiments on the ETCS case study where we vary the number of consequtive trains. The delays of events BRAKE and REPORT are distributed by deterministic distribution 25 and 5 seconds, respectively, and they are approximated by Erlang distributions. The different settings we use are determined by the number of phases (namely 1, 5 and 10) in the approximating Erlang distributions.

1) Symbolic Transformation: Table I gives an overview of the computation time and the model sizes for the symbolic part of our tool chain (cf. Section III), as generated by our STATEMATE-plugin. We display the bitvector sizes for states and transitions of the generated LTS, with and without cone-of-influence reduction. The bitvector size corresponds to a potential state space of the model, where a bitvector size of x gives a potential of 2^x. We also show the actual reachable state space, and the result of symbolic branching minimization, as well as the overall computation time (in seconds) in the table.

2) IMC Construction: In Table II and III, we report results concerning the construction and minimization using CADP (Section IV-A). Experimental results are displayed for monolithic (Table II) and compositional (Table III) construction (cf. Fig. 5). For each type of construction, we report the size of the *largest intermediate state space* we needed to handle, the construction time (Generation) and the Minimization time in seconds. The state spaces of the final results are also provided. For the compositional approach, we report the accumulated time (G+M) over all steps.

The advantage of using compositional construction in terms of space and time is apparent. Stepwise minimization keeps the size of state spaces low. This, in turns, reduces the duration of the minimization time in the next step, and so on, thus saving significant amount of time.

3) CTMDP Transformation: Statistical results for the transformation from IMC to CTMDP (Section IV-C) are displayed in Table IV. We give the number of states and transitions for the quotient IMC and the resulting CTMDP, together with the computation time required for this transformation. The column depicting the number of CTMDP transitions deserves a special comment. Since transitions in CTMDPs are triples (s, l, R) with a function R assigning rates to successor states, representing one transition may in the worst 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 nonzero entries per transition.

4) CTMDP Analysis: The runtime of the extended ETMCC model checker is shown in the last two columns of Table IV. The computation time needed to compute the worst case probability to reach the set of safety critical states has been computed for time bounds of 10 and 180 seconds, respectively. Since the timed reachability algorithm is implemented prototypically so far, we are actually quite satisfied with its performance.

VII. CONCLUSION

This paper has made the following contributions. It reported on (1) the first implementation of a time bounded reachability algorithm for CTMDPs, (2) the first – to our knowledge – entirely BDD-based algorithm for computing branching bisimulation quotients, (3) a compositional method to construct uniform CTMDPs, (4) the integration of these pieces in a useable tool chain, and (5) the application of this tool chain to a nontrivial example.

We are currently experimenting with the tool chain to identify bottlenecks and to improve interoperability. We feel that the tool chain as such is long and not easy to debug. In the future, we plan to make more phases of the tool flow work with purely symbolic data structures. Further, we are working on alleviating some of the modelling restrictions, which are currently dictated by the way failure modes are handled by STATEMATE. Concretely, we are going to open the approach towards repairable systems and other types of failures, and to allow time constraints to be attached to non-failure events in the system.

Acknowledgements. We would like to thank Michael Adelaide (OFFIS Oldenburg) and Hubert Garavel (INRIA Rhônes-Alpes) for their valuable support during the preparation of this paper.

REFERENCES

[1] 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." *Theor. Comput. Sci.*, vol. 345, no. 1, pp. 2–26, 2005.

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

[3] 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.

[4] T. Peikenkamp, E. Böde, I. Brückner, H. Spenke, M. Bretschneider, and H. Holberg, "Model-based Safety Analysis of a Flap Control System," in *Proc. of INCOSE*, Toulouse, 2004.

[5] T. Bienmüller, U. Brockmeyer, W. Damm, G. Döhmen, C. Eßmann, H.-J. Holberg, H. Hungar, B. Josko, R. Schlör, G. Wittich, H. Wittke, G. Clements, J. Rowlands, and E. Sefton, "Formal Verification of an Avionics Application using Abstraction and Symbolic Model Checking," in *Proc. of Safety-critical Systems Symposium*. Safety-Critical Systems Club, 1999, pp. 150–173.

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

[7] The VIS Group, "VIS: A system for verification and synthesis," in *Proc. of CAV*, ser. LNCS, vol. 1102, July 1996.

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

[9] S. Blom and S. Orzan, "Distributed branching bisimulation reduction of state spaces," in *Proc. of Int'l Work. on Parallel and Distributed Model Checking*, 2003.

[10] 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," Reports of SFB/TR 14 AVACS 8, Jan 2006, ISSN: 1860-9821, http://www.avacs.org.

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

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

[13] M. A. Johnson and M. R. Taaffe, "The denseness of phase distributions," Purdue University, 1988.

[14] S. Asmussen, O. Nerman, and M. Olsson, "Fitting phase-type distributions via the em algorithm," *Scand. J. Stat.*, vol. 23, no. 4, pp. 419–441, 1996.

TABLE I

SYMBOLIC STEPS: STATEMATE SAFETY ANALYSIS AND MINIMIZATION STATISTICS

Trains	Without COI					With COI					Branching Bisimulation		
	Potential		Reachable		Time	Potential		Reachable		Time	Min. Result		Time
	s bits	t bits	s	t	(sec.)	s bits	t bits	s	t	(sec.)	s	t	(sec.)
2	18	12	253	11132	6.9	16	12	121	5324	0.3	25	359	0.07
3	30	22	10585	3217840	30.2	28	22	5041	1532464	1.9	79	2065	1.16
4	42	32	444529	768146112	897.5	40	32	211681	365784768	6	79	2969	43.19
5	54	42	18670200	167284992000	18677	52	42	8890560	79659417600	6.1	79	4341	1150.94

TABLE II

MONOLITHIC CONSTRUCTION FOR ETCS WITH 2 TRAINS

Phases	Monolithic Construction			
	States	Transitions	G Time (sec.)	M Time (sec.)
1	33600	518464	12	3
5	302400	4142016	22	402
10	1016400	13521376	46	5154

TABLE III

EXPLICIT STEPS: COMPOSITION AND MINIMIZATION STATISTICS

Trains	Phases	Compositional Construction			Final Quotient IMC	
		States	Transitions	G + M Time (sec.)	States	Transitions
2	1	600	2505	42	355	1590
	5	10000	53625	61	5875	39500
	10	37500	207500	511	20000	154750
3	1	3240	16064	58	1375	5225
	5	64440	354100	813	36070	159119
	10	249480	1382900	10666	113650	533500
4	1	2870	11260	53	1435	5475
	5	57950	260350	420	30575	141000
	10	224900	1022700	7391	119650	558500

TABLE IV

EXPLICIT STEPS: CTMDP TRANSFORMATION AND ANALYSIS STATISTICS

Trains	Phases	Quotient IMC		Uniform CTMDP		Time	Time for Analysis of Formula (sec.)	
		States	Transitions	States	Transitions	(sec.)	$\sup_D \Pr_D(s, \overset{\leq 10}{\rightsquigarrow} B)$	$\sup_D \Pr_D(s, \overset{\leq 180}{\rightsquigarrow} B)$
2	1	358	1593	227	352 (1.75)	3.39	0.06	0.44
	5	5878	39503	3127	3752 (4.60)	3.67	0.54	7.00
	10	22003	154753	11252	12502 (5.52)	4.70	2.23	31.15
3	1	1378	5228	787	1347 (1.10)	3.61	0.14	2.01
	5	36073	159113	21722	35942 (1.55)	4.99	6.24	89.39
	10	113653	533503	56452	90402 (1.84)	8.46	17.95	254.29
4	1	1438	5478	817	1457 (1.01)	3.53	0.16	2.28
	5	30578	141003	15477	26577 (1.57)	4.86	4.43	62.83
	10	119653	558453	59452	101402 (1.64)	8.40	19.94	280.88

[15] A. Horváth and M. Telek, "Phfit: A general phase-type fitting tool." in *Computer Performance Evaluation / TOOLS*, 2002, pp. 82–91.

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

[17] A. Thümmler, P. Buchholz, and M. Telek, "A novel approach for fitting probability distributions to real trace data with the em algorithm." in *DSN*, 2005, pp. 712–721.

[18] R. Pulungan, "Orthogonal distance fitting for phase-type distributions," Tech. Rep., Forthcoming.

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

[20] BCG_MIN, "Project Website," March 2006, `http://www.inrialpes.fr/vasy/cadp/man/bcg_min.html`.

[21] H. Hermanns and S. Johr, "Compositional construction and analysis of uniform interactive markov chains," *In preparation for TACAS submission*.

[22] S. Johr, "Transforming Stochastic Activity Networks to continuous-time Markov decision processes," in *Proceedings of the PMCCS-7*, 2005.

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

[24] H. Garavel, F. Lang, and R. Mateescu, "An overview of CADP 2001," *European Assoc. for Software Science and Technology (EASST) Newsletter*, vol. 4, pp. 13–24, 2002.

[25] H. Garavel and H. Hermanns, "On Combining Functional Verification and Performance Evaluation Using CADP," in *Prof. of FME*, ser. LNCS, vol. 2391, 2002.

[26] H. Hermanns, J.-P. Katoen, J. Meyer-Kayser, and M. Siegle, "A Tool for Model-Checking Markov Chains," *J. STTT*, vol. 4, no. 2, pp. 153–172, 2003.

[27] ERTMS, "Project Website," Jan 20 2006, `http://ertms.uic.asso.fr/etcs.html`.

[28] A. Zimmermann and G. Hommel, "A train control system case study in model-based real time system design." in *IPDPS*, 2003, p. 118.

[29] H. Hermanns, D. N. Jansen, and Y. S. Usenko, "From stocharts to modest: a comparative reliability analysis of train radio communications." in *WOSP*, 2005, pp. 13–23.

[30] J. Abate, G. L. Choudhury, and W. Whitt, "Calculation of the GI/G/1 waiting-time distribution and its cumulants from pollaczek's formulas," *AEÜ Hirzel Verlag*, vol. 47, 5/6, pp. 311–321, 1993.