Compositional Analysis of Probabilistic Timed Graph Transformation Systems

MARIAMAXIMOVA, SVEN SCHNEIDER, and HOLGER GIESE, University of Potsdam, Hasso Plattner Institute, Germany

The analysis of behavioral models is of high importance for cyber-physical systems, as the systems often encompass complex behavior based on, e.g., concurrent components with mutual exclusion or probabilistic failures on demand. The rule-based formalism of Probabilistic Timed Graph Transformation Systems (PTGTSs) is a suitable choice when the models representing states of the system can be understood as graphs and timed and probabilistic behavior is important. However, model checking PTGTSs is limited to systems with rather small state spaces.

We present an approach for the analysis of large-scale systems modeled as PTGTSs by systematically decomposing their state spaces into manageable fragments. To obtain qualitative and quantitative analysis results for a large-scale system, we verify that results obtained for its fragments serve as overapproximations for the corresponding results of the large-scale system. Hence, our approach allows for the detection of violations of qualitative and quantitative safety properties for the large-scale system under analysis. We consider a running example in which shuttles drive on tracks of a large-scale topology and autonomously coordinate their local behavior with other shuttles nearby. For this running example, we verify that (a) shuttles can always make the expected forward progress using several properties, (b) shuttles never collide, and (c) shuttles are unlikely to execute emergency brakes in two scenarios. In our evaluation, we apply an implementation of our approach in the tool AutoGraph to our running example.

CCS Concepts: • Theory of computation → Timed and hybrid models; Probabilistic computation; Distributed computing models; Parallel computing models; Graph algorithms analysis; Logic and verification; Modal and temporal logics; Automated reasoning;

Additional Key Words and Phrases: Cyber-physical systems, graph transformation systems, qualitative analysis, quantitative analysis, probabilistic timed systems, compositional analysis, model checking

ACM Reference format:
Maria Maximova, Sven Schneider, and Holger Giese. 2023. Compositional Analysis of Probabilistic Timed Graph Transformation Systems. Form. Asp. Comput. 35, 3, Article 16 (September 2023), 79 pages.
https://doi.org/10.1145/3572782

1 INTRODUCTION

Real-time cyber-physical systems often emit a complex behavior based on, e.g., concurrent communicating components with mutual exclusion or probabilistic failures on demand. Consequently,
modeling formalisms for capturing such systems must suitably support the modeling of their complex behaviors. In such a model-driven approach, the analysis of behavioral models w.r.t. a provided specification is vital to ensure overall soundness of the resulting system.

The rule-based transformation of graphs is a suitable choice when the states of the system can be understood as graphs. In particular, the formalism of Probabilistic Timed Graph Transformation Systems (PTGTSs) \[36\] extends the standard rule-based transformation of graphs such that probabilistic and timed behavior is covered by supporting (a) non-deterministic choice among steps, (b) probabilistic choice among step results, and (c) steps representing the non-deterministic passage of time. The use of non-determinism among steps in models such as PTGTSs allows for concrete instantiations of the model to resolve such non-determinism based on aspects outside of the model’s scope (such as additional requirements, randomization, or user interactions). The resolution of non-determinism is thereby permitted to vary each time the system is in a state where the model offers a non-deterministic choice among steps. The probabilistic choice among step results allows to model a probabilistic failure on demand as a possible step result (alternative to the successful step result) with usually a low probability. The use of non-determinism to represent the passage of time allows for a broad range of actual behavior where durations of actions cannot be precisely predicted at modeling time but are rather specified using lower and upper bounds.

As a running example, discussed in detail in Section 3, we model a cyber-physical system in which shuttles drive autonomously on tracks of a large-scale topology and coordinate their driving behavior locally using shuttle-to-shuttle wireless communication as a PTGTS and additionally integrate the coordination behavior among shuttles modeled in \[36\]. In this PTGTS, non-determinism originates, e.g., due to multiple independent shuttles and an unrestricted choice of successor tracks, when shuttles move across a branch in the track topology. Probabilistic choice is used in this PTGTS to model that shuttles do not always observe every traffic light and that inter-shuttle wireless communication may not succeed due to perturbations. Lastly, timing constraints are used to, e.g., restrict the speed of shuttles based on their current driving mode and to ensure that certain steps are performed urgently.

A model checking approach for PTGTSs w.r.t. probabilistic metric temporal properties was introduced in \[36\]. However, the potentially high degree of concurrency among a possibly large number of active components (also called actors) in a system modeled as a PTGTS may result, due to the state space explosion problem, in intractably large state spaces limiting applicability of model checking. As a workaround, a selected set of small PTGTS examples may be considered in a naive informal decomposition approach, hopefully capturing all system-specific challenges to establish trust that the model exhibits the required safe behavior and that unwanted behavior is sufficiently unlikely. However, it cannot be excluded that the considered small PTGTS examples do not reveal all the threatening behavior when they are model checked independently, as only few start conditions are considered and the system behavior at the intersections of such small PTGTS examples is not explicitly modeled and therefore not analyzed. To demonstrate this problem, we integrate the behavioral aspects modeled in \[36\] into our running example and discuss the workflow of repairing the PTGTS based on the modeling problems detected using our approach.

To mitigate the state space explosion problem, we present a decomposition-based approach (see Figure 1 for a visualization) for the analysis of large-scale systems modeled as PTGTSs to rule out violations of qualitative and quantitative safety properties as well as certain violations of liveness properties.

As a first step of this decomposition-based approach, we capture the underlying static large-scale topology contained in a large-scale state of a large-scale system as a subgraph that is not changed by graph transformation (see Figure 1(b)). For cyber-physical systems, the existence of such an
underlying static large-scale topology can be justified, e.g., by a logical or spatial distribution. Given a set of so-called fragment topologies, which capture different patterns occurring in the static topology, we then describe how the static topology is covered by overlapping instances of these (ideally very small) fragment topologies (see also Figure 1(a)). The dynamic (i.e., non-static) components are thereby distributed to the fragment topologies leading to a unique fragment state for each embedding of a fragment topology. Moreover, to rule out inadequate embeddings, we specify how multiple embeddings of fragment topologies can overlap in their borders (see the right part of Figure 1(a)).

As a second step of this decomposition-based approach, we construct for each fragment topology a fragment system given by an adapted PTGTS. Such an adapted PTGTS is then ensured to (a) exhibit the same behavior on the non-overlapped part of the fragment topology (named core) and to (b) simulate all possible behaviors that can happen for any occurrence of the fragment topology in a large-scale topology. Using our approach, we decompose a single overall system into \( n \) fragment systems, where \( n \) is given by the number of different fragment topologies. Thereby, each fragment system symbolically captures the behavior on all embeddings of its underlying fragment topology in the large-scale system. Consequently, the state space explosion problem is mitigated by (i) reducing the concurrency between steps by splitting the system into smaller subsystems and (ii) only considering during analysis a single instance fragment system for each embedded fragment topology.

To obtain the adapted PTGTS representing a fragment system for a given fragment topology, we include modifications of the rules of the original PTGTS operating on the border of a fragment topology into the adapted PTGTS. Such a modified rule is then applicable whenever the original
rule was applicable in the original PTGTS. With this direct relationship between behaviors on the fragment topology and the overall topology, we obtain i.a. that the likelihood of an unwanted or forbidden graph pattern in one of the adapted PTGTSs is an upper bound for its likelihood in its embedding in the large-scale PTGTS.

As a last step of this decomposition-based approach, exploiting our decomposition to counter the state space explosion problem, we apply a novel approach based on model checking to the PTGTSs constructed for the fragment topologies instead of performing model checking directly on the PTGTS modeling the large-scale system.

The novel approach based on model checking improves our earlier work in [37] and consists of a novel state space generation technique then supporting on the foundation of the generated state space the analysis of further qualitative properties and the analysis of quantitative probabilistic properties to result in more precise results. The novel state space generation technique for PTGTSs improves upon earlier state space generation techniques discussed in, e.g., [36, 37] by avoiding the generation and integration of states in the state space that are, due to temporal constraints, actually unreachable. This improvement allows for a terminating state space generation when possible (as opposed to [36, 37] where state space generation possibly did not terminate due to an unbounded number of unreachable states being traversed). Besides allowing for a terminating state space generation resulting in a finite state space, avoiding the generation and integration of unreachable states can reduce the time and space requirements for PTGTSs also for cases where the earlier model checking techniques terminated, since the additional overhead of checking temporal constraints can often be smaller compared to the cost for falsely generating and integrating unreachable states. Moreover, this state space generation technique generating state spaces only consisting of reachable states then allows for the development of a novel monitoring technique to track active components across steps, which are the shuttles in our running example. This monitoring technique is then exploited to extend analysis of a large-scale PTGTS for qualitative and quantitative reachability from before to the further important properties of timelock freedom, actor-based deadlock freedom, actor-based livelock freedom, and actor-based fairness as well as enhancing the precision for the analysis of quantitative safety. The novel state space generation technique as well as the monitoring technique for active components supporting together the analysis of PTGTSs w.r.t. the mentioned properties has been implemented in the tool AutoGraph. In our empirical evaluation, we apply this implementation to our running example of shuttles driving on a track topology extended from [37] and verify that (a) shuttles can always make the expected forward progress using several properties, (b) shuttles never collide, and (c) shuttles are unlikely to execute emergency brakes (which may be executed by shuttles due to the prior occurrence of failures on demand such as overlooked traffic lights or multiple failed coordination attempts among shuttles).

This article is structured as follows: In Section 2, we further discuss related work. In Section 3, we introduce our (extended) running example from the domain of cyber-physical systems. In Section 4, we recapitulate the necessary preliminaries on probabilistic timed automata and their model checking. In Section 5, we recall the formalism of PTGTSs, discuss the modeling details for our running example, and present a novel state space generation procedure for PTGTSs to improve the analysis capabilities of our approach later on. In Section 6, we discuss how fragment topologies can be employed to cover the large-scale topologies, which are used as static substructures in a large-scale system and how this coverage can be extended to the states of that system. In Section 7, we present our decomposition-based approach allowing to split a system into more manageable fragment systems to ease subsequent analysis. In Section 8, we discuss analysis procedures that can be applied to the obtained fragment systems and how the obtained analysis results carry over to the large-scale system. In Section 9, we present an evaluation of the conceptual results for our
running example and report on the obtained analysis results. Finally, in Section 10, we close the article with a conclusion and an outlook on planned future work.

2 RELATED WORK

In this article, we focus on PTGTSs [36] based on Probabilistic Timed Automata (PTA) [34] with their modeling characteristics already discussed in the introduction. Other extensions of Graph Transformation Systems (GTSs) have been introduced before such as Probabilistic Graph Transformation Systems (PGTSSs) [30] based on Markov Decision Processes (MDPs) or Probabilistic Automata (PA) [46, 53], Timed Graph Transformation Systems (TGTSS) [6, 19, 42] based on Timed Automata (TA) [2], and stochastic GTSs [9, 24, 25] based on Continuous Time Markov Chains (CTMCs) [43]. These approaches have in common that they can be translated into the mentioned automata-based formalisms where existing analysis algorithms (e.g., model checking such as in [33, 34] for PTA and [28] for CTMCs) can be applied with varying efficiency. Tools used for analysis for such automata are, e.g., the tools Prism [31] supporting the model checking of PA, TA, PTA, and CTMCs and UPPAAL [1] supporting the simulation and model checking of TA with varying feature sets. Furthermore, variations of TGTSSs have been also discussed in [9, 51] where time and timing constraints are encoded not based on underlying models such as TA directly but where analysis may then be limited to the analysis of singular generated traces. Also note that PTGTSs have been extended recently to Interval Probabilistic Timed Graph Transformation Systems (IPTGTSs) in [38] based on Interval Probabilistic Timed Automata (IPTA) [29, 55] also supporting a non-deterministic specification of probabilities for cases where precise probabilities are not known or unreasonable due to, e.g., physical perturbation. For comparison, CTMCs and therefore stochastic GTSs are deterministic and therefore deviate from the non-deterministic PGTSSs, TGTSS, PTGTS, and IPTGTSs, for which non-determinism makes analysis a harder problem with less-efficient analysis procedures. Hence, when using CTMCs or stochastic GTSs, special care is required to ensure that analysis results effectively translate to the actual system to be analyzed, since (a) rate parameters can often only be estimated and (b) state residence times are often not perfectly exponential. In contrast, analysis results for the automata models of PA, TA, PTA, and IPTA are subject to different resolutions of non-determinism resulting in intervals of, e.g., probabilities based on probability minimizing and probability maximizing resolutions of non-determinism.

The idea pursued in this article to decompose a system into subsystems for the analysis or to compose it from analyzed subsystems has been studied intensively [10] but our suggested compositional approach has distinguishing characteristics.

Firstly, the vast majority of approaches (such as process algebras or similar models) assume that the modeling formalism supports the composition/decomposition as a first-class concept such that compositional analysis techniques are directly applicable, as the subsystem models cover all possible behaviors in all contexts. This approach was, e.g., used in [54], where the process algebra of stochastic hybrid CSP is considered with its built-in parallel composition operator. Similarly, a given composition operation has also been employed in [32], where the large-scale PA is given as the explicit composition of multiple smaller PA using the standard PA composition operation based on actions. Another approach is given in [27], where the standard composition operation of PA is used again but where sub-components may not need to be reverified/reanalyzed using model checking upon system alterations when these can be shown to not affect the given sub-component. Such approaches then typically rely on a user-provided decomposition and attempt to verify/analyze the system as a whole by composing analysis results obtained for its components relying on the fixed notion of composition. In contrast, we do not rely on a built-in decomposition operator but rather allow for a flexible derivation of a
decomposition of a large-scale topology in terms of fragment topologies, overlappings, and a suitable overapproximation on the border, which are not predefined by the modeling formalism.

Secondly, several approaches rely on a protocol-like specification of how the decomposed subsystems interact, while in our approach the overapproximation is derived systematically from the PTGTS model that does not necessarily provide such a protocol-like specification already. The compositional analysis approach for GTSs from [18, 48] defines explicit interfaces, which are used to consider whether the behavior of two independent graphs glued via these interfaces (requiring that local transitions are compatible) cover jointly all global transitions. Moreover, in further approaches, protocols for the roles of collaborations and ports of components have been assumed. For example, in [22], the idea to overapproximate the environment and border is explored for timed automata with explicit models of the roles in form of protocol automata. This idea has been combined with dynamic collaborations in [19, 21] captured by TGTSs and their analysis via inductive invariant checking [5, 6]. Later on, this approach has been extended to role, component, and collaboration behavior, which is captured by TGTSs and hybrid GTSs in [7] and [4], respectively. However, as opposed to the presented approach, in all these cases an explicit concept of interface is assumed to separate parts that are analyzed in isolation.

3 RUNNING EXAMPLE

We now informally introduce a scenario (based on the RailCab project [47]) of autonomous shuttles driving on a large-scale topology, which serves as a running example in the remainder of this article. Based on this introduction, we will discuss how we model this shuttle scenario as a PTGTS in Section 5.

In the considered shuttle scenario, a track topology containing a possibly very large number of tracks of approximately equal length is given. See Figure 2(a) for such a track topology (with a rather small number of tracks for demonstration purposes). In the following, we focus on those large-scale track topologies that are constructed using the set of fragment topologies given in Figure 2(b). Note that we discuss the composition of fragment topologies leading to large-scale topologies and, vice versa, the decomposition of large-scale topologies into fragment topologies in more detail in Section 6 and Appendix A. The fragment topologies define basic patterns admissible to occur in large-scale track topologies. As depicted in Figure 2(a), tracks are connected to predecessor tracks and successor tracks via directed connections. In the simplest case, such connections between adjacent tracks result in straight track sequences (cf. FT3 in Figure 2(b)). Two track sequences can be joined together at a join-track (i.e., a track with two predecessors) leading to a join fragment topology (cf. FT9 in Figure 2(b)). In such a join fragment topology, shuttles may locally communicate when being on opposing sides of a conflict to coordinate their further movement to agree on their sequentialization at the join-point to avoid emergency braking. Also, two track sequences can split up at a fork-track (i.e., a track with two successor tracks) leading to a fork fragment topology (cf. FT7 in Figure 2(b)). Moreover, depots may have a directed connection to a track allowing shuttles to enter or exit the track topology (cf. FT1 and FT2 in Figure 2(b)). In our shuttle scenario, we allow for depots from which arbitrarily many shuttles can enter the track topology: Even if this would not be true for the modeled system, this just means that there are some additional (non-)deterministic steps in the model that are not realizable in the system. Also, for the sake of simplicity, we always allow shuttles to enter depots, which means that we assume the existence of some coordination mechanism ensuring that a shuttle only drives towards a depot that has at least enough capacity for that shuttle. From a practical point of view, such depots can also be understood/employed as connections to further areas of an even larger track topology outside of the scope of the current approach.
Shuttles are always located on a single track and drive forward following the connections between tracks. Shuttles may be in one of the three modes DRIVE, STOP, or BRAKE. Being in mode DRIVE, shuttles drive to the next track (respecting the direction of the connection between the tracks) with a certain velocity, which may be slow ([3, 4] time units per track) or fast ([2, 3] time units per track). In both cases, the shuttle controls its acceleration such that it moves to the next track within the given interval. Regularly, shuttles change into mode STOP and decrease their velocity to 0 tracks per time unit, which allows them to avoid coming too close to other shuttles. The central behavioral requirement for shuttles is the absence of collisions (given by two distinct shuttles on a common track). To avoid collisions, shuttles may need to execute emergency brakes changing into mode BRAKE, thereby rapidly decreasing their velocity to 0 tracks per time unit. Moreover, some tracks of the track topology can be under construction. Passing a construction site, shuttles should always drive slowly to prevent derailing. To inform shuttles about such construction sites ahead, one or two yellow traffic lights are installed a few tracks ahead of construction sites. Fast-driving shuttles observing such yellow traffic lights slow down and then do not need to execute emergency brakes when the construction site is just ahead (cf. FT4 and FT5 in Figure 2(b)). After construction sites, green traffic lights may be installed permitting shuttles to increase their velocity (cf. FT6 in Figure 2(b)). However, shuttles missing yellow traffic lights and consequently noticing a construction site too late have to execute an emergency brake thereby changing into mode BRAKE. We consider such situations where shuttles do not appropriately take the yellow
traffic lights into account as failures on demand. For such failures on demand, we assume a probability of $10^{-6}$ and conjecture that their occurrence does not only depend on the visual observation by the train driver but also depends on a failure of the backup system.

To coordinate their driving behavior by forming convoys and to avoid emergency brakes, shuttles should communicate locally with other shuttles when approaching joins of track sequences (cf. FT9 in Figure 2(b)). To support this coordination, parallel tracks of a topology that are one, two, or three tracks ahead of such a join track are marked by conflicts and called conflicting tracks. Intuitively, communication between shuttles should resolve the conflict of which shuttle passes the join track first. Two shuttles can attempt to establish a connection if they are currently either both on a first or both on a second conflicting track (counted from left to right). If the connection attempt failed (which happens with the probability of 0.05), then shuttles continue to drive autonomously and try to connect one conflicting track later when the failure happened at the first conflicting track. Here, we assume an imperfect communication between shuttles meaning that the connection establishment prior to a join track is subject to a failure on demand due to potential interference in their wireless communication. If the connection attempt was successful (which happens with the probability of 0.95), then one of the shuttles becomes a convoy leader and the other one the convoy follower. The convoy leader is the shuttle designated to drive over the join track first to be followed subsequently by the convoy follower. A successfully established connection thereby ensures that neither shuttle executes an emergency brake, allowing both shuttles to drive safely and (if possible) without delay across the join track. To achieve this safe behavior, the convoy follower will adjust its speed (if possible) to ensure that it enters the join track not before the convoy leader has already left that track (and stops instead of executing an emergency brake when the convoy leader did not pass the join track fast enough). In this sense, the convoy follower gives up a part of its autonomy, whereas the convoy leader continues to drive forward under its own control.\footnote{In \cite{37}, we already considered a simple join fragment topology where shuttles did not communicate but rather just stopped to avoid collisions. In contrast, using communication, the shuttles are now attempting to avoid delays resulting from stopping when possible. Moreover, in \cite{36}, we already considered communication between shuttles in a simplified form without ensuring that the shuttles would perform two communication attempts independently of their starting position on the join fragment topology.}

In our compositional analysis approach, we distinguish between static and dynamic elements of the modeled large-scale system. For our running example, the static elements are the tracks, depots, installed traffic lights, construction sites, conflicts between tracks, and connections among these elements. These elements are static in the sense that the PTGTS modeling the behavior of the system never changes its underlying large-scale topology by adding or removing any static elements. Complementary, dynamic elements are shuttles, their attributes, their connections to tracks and to other shuttles as well as the attributes of traffic lights. For the considered shuttle scenario, we are interested in the properties P1–P6 given in the following. Using our compositional approach, we provide support for the analysis of these properties and discuss the analysis results in Section 9. Note that the properties P3–P5 are actor-based properties where individual actors, in our running example given by shuttles, are the focus of attention.

- **P1 (Timelock Freedom):** The PTGTS modeling the shuttle scenario never gets temporally stuck in a state where no steps (discrete steps of, e.g., driving shuttles or timed steps) are enabled.
- **P2 (Qualitative Reachability):** Shuttles never collide (i.e., for every reachable state, two distinct shuttles are never on a common track).
• P3 (Actor-based Deadlock Freedom): Shuttles are never prevented indefinitely from driving forward, e.g., after they stopped or executed emergency brakes (i.e., for every reachable state, for every shuttle, there is some continuing path in which that shuttle drives forward).

• P4 (Actor-based Livelock Freedom): Shuttles are permanently able to eventually exit the fragment topology on which they are currently located (i.e., for every reachable state, for every shuttle, there is a continuing path in which that shuttle eventually exits the fragment topology).

• P5 (Actor-based Fairness): Shuttles will eventually exit the fragment topology on which they are currently located (i.e., for every reachable state, for every continuing path, every shuttle eventually exits the fragment topology in that path).

• P6 (Quantitative Reachability): Occurrences of emergency brakes are improbable at a local level for a single shuttle but also improbable at the global level for the entire large-scale system and its possibly numerous shuttles.

4 PROBABILISTIC TIMED AUTOMATA

In this section, we recall the syntax and semantics of PTA [34] as well as briefly discuss how PTA can be model checked employing the so-called difference bound matrices [8, 11]. Based on these preliminaries, we then present PTGTSs and PTGTS state space generation in the subsequent section. As mentioned in Section 2, PTA are the underlying model of PTGTSs in the sense that every PTGTS can be translated into a semantically equivalent PTA according to [36]. A key reason for this translatability is that PTA and PTGTSs are by design similar regarding how they allow for the (a) specification of probabilities for results of discrete steps and for the (b) usage of clock variables to restrict the passage of time using clock invariants and to delay discrete steps using clock guards. As a direct consequence, state space generation techniques for PTA and PTGTSs as well as model checking techniques for PTA and PTGTSs w.r.t. probabilistic timed reachability properties also have strong similarities. For example, the tools Prism [31] and Uppaal [1] support model checking for foundational models such as PTA and TA, respectively. We introduce in this section PTA as a much simpler modeling formalism compared to PTGTSs introducing key terminology, syntax, and semantics to simplify the introduction of PTGTSs in the subsequent section. There, we also propose a novel state space generation technique for PTGTSs that is then used in conjunction with the novel analysis technique presented in Section 8, replacing the usage of Prism for analyzing PTA translations of PTGTSs or the usage of Uppaal for analyzing TA restrictions of such PTA.

4.1 Syntax and Semantics of Probabilistic Timed Automata

For a set of clock variables $X$, clock constraints $\psi \in \text{CC}(X)$ are finite conjunctions of clock comparisons of the form $c_1 \sim n$ and $c_1 - c_2 \sim n$ where $c_1, c_2 \in X, \sim \in \{<, >, \leq, \geq\}$, and $n \in \mathbb{N} \cup \{\infty\}$ (where $\mathbb{N}$ contains all natural numbers including 0). A clock valuation $v \in \text{CV}(X)$ of type $v : X \rightarrow R^+_0$ (where $R^+_0$ contains all positive real numbers including 0) satisfies a clock constraint $\psi$, written $v \models \psi$, as expected. The initial clock valuation $\text{ICV}(X)$ maps all clocks to 0. For a clock valuation $v$ and a set of clocks $X'$, $v[X' := 0]$ is the clock valuation mapping the clocks from $X'$ to 0 and all other clocks according to $v$. For a clock valuation $v$ and a duration $\delta \in R^+_0$, $v + \delta$ is the clock valuation mapping each clock $x$ to $v(x) + \delta$.

For a countable set $A$, $\mu : A \rightarrow [0, 1]$ is a Discrete Probability Distribution (DPD) over $A$, written $\mu \in \text{DPD}(A)$, if the probabilities assigned to elements of $A$ add up to 1, i.e., $\sum_{a \in A} \mu(a) = 1$ using summation over multisets. Moreover, the support of $\mu$, written $\text{supp}(\mu)$, contains all $a \in A$ for which the probability $\mu(a)$ is non-zero.

PTA [34] combine the usage of clocks to capture real-time phenomena and probabilism to approximate/describe the likelihood of outcomes of certain steps. A PTA (such as the one in
(a) PTA $A$ where circles represent locations, a vertical arrow without source indicates the initial location $\ell_1$, gray boxes attached to locations contain the invariant for that location (such as $c_0 \leq 5$) as well as the set of APs (such as \{done\}) with which that location is labeled. PTA edges are given by hyperedges where labels close to the source location of the edge (such as $a_2; c_0 \geq 0$) represent the action and guard of that PTA edge, respectively, and where labels close to the target locations of the edge (such as $0.5; \{c_0\}$) contain the probability to reach that target location and the set of clocks to be reset upon reaching that target location, respectively.

(b) Path of the PTA $A$ for some adversary where (Step 1) 3 time units elapse in $\ell_1$ (admissible would be even 5 time units according to the invariant $c_0 \leq 5$), which happens with probability 1 (as timed steps always have probability 1 when being performed), (Step 2) the location is changed to $\ell_2$, which happens with probability 1, using action $a_2$ and resetting the clock $c_0$ to 0 (the guard $c_0 \geq 0$ of the used PTA edge is satisfied since $c_0$ has the value 3 and the PTA edge leading to $\ell_2$ as well as the steps letting further time elapse were not selected by the adversary here), (Step 3) 2 time units elapse (admissible would be even 3 time units according to the invariant $c_0 \leq 5$), which happens with probability 1, and (Step 4) the location is changed to $\ell_3$, which happens with probability 0.5, using action $a_3$ and resetting the clock $c_0$ to 0 (the guard $c_0 \geq 2$ of the used PTA edge is satisfied since $c_0$ has the value 2 and the steps letting further time elapse were not selected by the adversary here).

(c) Symbolic state space induced by the PTA $A$ where the guards, invariants, and clock resets of $A$ have been encoded into the states. The PTA $A$ is isomorphic to its symbolic state space as the clock $c_0$ is reset in every step. However, when all locations of $A$ are reachable, the symbolic state space can be considerably larger compared to the PTA from which it was generated.

Fig. 3. PTA $A$, one of its paths, and its symbolic state space.

Figure 3(a) consists of (a) a set of locations with a distinguished initial location (such as $\ell_1$), (b) a set of clocks (such as $c_0$) which are initially set to 0, (c) an assignment of a set of Atomic Propositions (APs) (such as \{done\}) to each location (for subsequent analysis of, e.g., reachability properties), (d) an assignment of constraints on its clocks to each location as invariants (such as $c_0 \leq 5$), and (e) a set of probabilistic timed edges. Each probabilistic timed edge consists thereby of (i) a single source location, (ii) at least one target location, (iii) an action (such as $a_1$), (iv) a clock constraint (such as $c_0 \geq 1$) specifying as a guard when the edge is enabled based on the current values of the clocks, and (v) a DPD assigning a probability to each pair consisting of a set of clocks to be reset (such as \{c_0\}) and a target location to be reached.

**Definition 1 (PTA).** A probabilistic timed automaton (PTA) $A$ is a tuple with the following components:
- $\text{locs}(A)$ is a finite set of locations,
- $\text{iloc}(A)$ is the unique initial location from $\text{locs}(A)$,

---

2 Actions in PTA can be used, e.g., to document the meaning of edges or to define a parallel composition of PTA.
• acts(A) is a finite set of actions disjoint from \( R^+_0 \),
• clocks(A) is a finite set of clocks,
• invs(A) : locs(A) \rightarrow CC(clocks(A)) maps each location to an invariant for that location such that the initial clock valuation satisfies the invariant of the initial location (i.e., ICV(clocks(A)) \models invs(A)(iloc(A))),
• edges(A) \subseteq locs(A) \times acts(A) \times CC(clocks(A)) \times DPD(2^{\text{clocks}(A)} \times locs(A)) is a finite set of PTA edges of the form \((\ell_1, a, \psi, \mu)\) where \( \ell_1 \) is the source location, \( a \) is an action, \( \psi \) is a guard, and \( \mu \) is a DPD mapping pairs \((\text{Res}, \ell_2)\) of clocks to be reset and target locations to probabilities,
• aps(A) is a finite set of APs, and
• lab(A) : locs(A) \rightarrow 2^{\text{aps}(A)} maps each location to a set of APs.

The semantics of a PTA is given in terms of the induced Probabilistic Timed System (PTS). The states of the induced PTS are pairs \((\ell, v)\) of locations \( \ell \) and clock valuations \( v \). The steps of a PTA are either timed steps in which all clocks are advanced by a certain duration or discrete steps in which one PTA edge is implemented as a probabilistic step in the PTS. That is, clocks and timing constraints on them (guards, invariants, and resets) are encoded in the states and steps of the PTS already while the probabilistic aspect of PTA edges is preserved resulting in probabilistic steps in PTS containing DPDs on successor states. Resolving this probabilistic choice results in the semantics of a PTS in terms of its timed probabilistic paths. Considering a state space, which corresponds to these paths, instead would be disadvantageous for the definition of the satisfaction of probabilistic timed reachability properties (based on PTCTL satisfaction). However, for the iterative construction of a path, the non-determinism among PTS steps has to be resolved as well, which is technically achieved using a so-called adversary, which chooses each successive step based on the path constructed so far. See Figure 3(b) for an example of a path of the PTS induced by the PTA from Figure 3(a).

Definition 2 (PTS Induced by PTA). Every PTA A induces a unique probabilistic timed system (PTS) PTAt0PTS(A) = P consisting of the following components:

- states(P) = \{ (\ell, v) \in locs(A) \times CV(\text{clocks}(A)) \mid v \models invs(A)(\ell) \} contains as PTS states pairs of locations and clock valuations satisfying the location’s invariant,
- istate(P) = (iloc(A), ICV(\text{clocks}(A))) is the unique initial state from states(P),
- acts(P) = acts(A) is the same set of actions,
- steps(P) \subseteq states(P) \times (acts(P) \cup R^+_0) \times DPD(\text{states}(P)) is the set of PTS steps. A PTS step \(( (\ell, v), a, \mu ) \in \text{steps}(P)\) contains a source state \((\ell, v)\), an action \( a \) from acts(P) for a discrete step or a duration \( a \) from \( R^+_0 \) for a timed step, and a DPD \( \mu \) assigning a probability to each possible target state.

\(( (\ell, v), a, \mu ) \in \text{steps}(P)\), if one of the two following cases applies:

- **Timed step:**
  - \( a \in R^+_0 \) is a duration,
  - \( (\ell, v + t') \in \text{states}(P) \) for all \( t' \in [0, a] \), i.e., the invariant of \( \ell \) is also satisfied for all intermediate time points \( t' \), and
  - \( \mu(\ell, v + a) = 1 \) identifies the unique PTS target state \((\ell, v + a)\).
- **Discrete step:**
  - \( a \in \text{acts}(A) \) is an action of the PTA,
  - \( (\ell, a, \psi, \mu^+) \in \text{edges}(A) \) is an edge of the PTA,
  - \( v \models \psi \), i.e., the guard \( \psi \) of the edge is satisfied by the valuation \( v \), and
\[
\mu(\ell', \nu') = \sum \{ \mu'(X, \ell') \mid X \subseteq \text{clocks}(A), \nu' = \nu[X := 0] \}
\]
the DPD on the PTS target states \((\ell', \nu')\).

- \(\text{aps}(P) = \text{aps}(A)\) is the same set of APs, and
- \(\text{lab}(P)(\ell, \nu) = \text{lab}(A)(\ell)\) labels states in \(P\) according to the location labeling of \(A\).

According to the PTA semantics from Definition 2, (a) PTA with no clocks are essentially PA \([53]\), (b) PTA where each DPD \(\mu\) used in one of the PTA edges has only one element in its support (i.e., \(\text{supp}(\mu)\) is a singleton) are essentially TA \([2]\), and (c) PTA with both these restrictions are essentially finite automata.

Lastly, a PTA is called non-zeno when time diverges on each of its timed probabilistic paths, meaning that the sum of the durations of the timed steps in that timed probabilistic path must diverge. In here, assuming non-zeno is important for many real-time formalisms such as PTA to ensure the absence of unrealistic sequences of timed steps, e.g., with delays 1, 0.5, 0.25, \ldots converging at time 2, as these kinds of paths are not considered during analysis. However, the PTA induced by a PTA is infinite (for non-zeno PTA), since it i.a. contains all timed steps of durations \(\delta\) taken from \(R_0^+\).

### 4.2 Model Checking for Probabilistic Timed Automata

Since the PTS induced by a PTA is infinite (for non-zeno PTA), it cannot be explored and analyzed directly. Instead, symbolic analysis techniques for PTA have been devised such as those implemented in the Prism model checker \([31]\) for a fragment of Probabilistic Timed Computation Tree Logic (PCTL) \([34]\) operating on a symbolic finite representation of the induced PTS. To achieve finiteness of this symbolic representation (called symbolic state space subsequently), the syntax of clock constraints used for guards and invariants has been carefully chosen in \([34]\). Consider the symbolic state space in Figure 3(c) obtained for the PTA \(A\) from Figure 3(a). In such a symbolic state space, all timing constraints of the PTA given by guards, invariants, and clock resets have been converted into clock constraints called zones. Each state of the symbolic state space has then the form \((\ell, \psi)\) containing a location \(\ell\) and one of these zones \(\psi\). The zone \(\psi\) (such as \(0 \leq c_0 \leq 5\)) captures all clock valuations \(\nu\) that can be observed in the induced PTS in the location \(\ell\) (i.e., all clock valuations \(\nu\) satisfying \(\psi\), written \(\nu \models \psi\)). For example, the Prism model checker can then analyze properties such as “what is the worst-case probability that the PTA reaches a location labeled with the AP \(\text{done}\) within 5 time units.” For the PTA \(A\) in Figure 3(a), this property is satisfied with probability 0.75, since the non-determinism of the PTA would be resolved (by an adversary) such that the PTA first takes a step to \(\ell_2\) without letting time pass and then performs the probabilistic step (up to two times after waiting for not longer than 2 time units after the first attempt) until it reaches the location \(\ell_3\) labeled with the AP \(\text{done}\) (the probabilistic step cannot be taken a third time due to the deadline of at most 5 time units specified in the property). Thereby, the adversary does not result in a single path but a tree of probabilistic timed steps where each state has either an exiting timed or an exiting probabilistic step. Note that the deadline of 5 time units specified in the property would be encoded for model checking using an additional clock, which would change the structure of the resulting symbolic state space (compared to the state space in Figure 3(c)).

Other tools such as Uppaal \([1]\) designed for TA rely on the same underlying idea of a symbolic state space where clock valuations are given by zones. These tools use Difference Bound Matrices (DBMs) \([8, 11]\) to encode zones, which are given for \(n\) clocks by \((n + 1) \times (n + 1)\) matrices of pairs.

---

3 All probabilities assigned to pairs of clock resets \(X\) and target locations \(\ell'\) that result in the same target state \((\ell', \nu')\) must be added. Two such pairs can result in the same target state when some clocks \(c \in X\) were mapped by the valuation \(\nu\) of the source state to 0 already. For example, the pairs \((0, \ell')\) and \((1, c), \ell')\) have the same target state when \(\nu(c) = 0\). This implies that the probabilities assigned to these two pairs via the DPD \(\mu'\) must be added.
(\sim, v) for \sim \in \{\le, <\} and v \in \mathbb{Z} (where \mathbb{Z} is the set of all integers). Clock comparisons contained in a zone are encoded as follows: the cell \((0, i)\) encodes the clock constraint \(0 - c_i \sim v\), the cell \((i, 0)\) encodes the clock constraint \(c_i - 0 \sim v\), and the cell \((i, j)\) encodes the clock constraint \(c_i - c_j \sim v\) for \(i > 0\) and \(j > 0\). Transitivity between such DBM entries is resolved whenever the matrices are changed. For example, the clock constraints \(c_1 - c_2 \le 4\) and \(c_2 - c_3 \le 6\) imply (the tightest bound) \(c_1 - c_3 \le 10\) and, therefore, (a) if the DBM currently stores \(c_1 - c_3 \le 20\), then the entry \((\le, 20)\) is reduced to \((\le, 10)\) and (b) if the DBM currently stores \(c_3 - c_1 \le -20\), then the DBM is detected to be inconsistent (representing as a zone the empty set of clock valuations). During state space generation, DBMs are then used to determine the satisfaction of guards and are adapted for each new step by (i) including satisfied guards, (ii) resetting clocks, (iii) removing upper bounds on clocks to let time pass, and (iv) including the invariant of the target location. Whether a guard is satisfied by some valuation represented by the current DBM can be checked by adding the guard to the DBM, by resolving then transitivity as described before, and by, finally, checking the DBM for consistency. Moreover, to ensure finiteness of the symbolic state space, a clock ceiling given by a natural number \(n_c\) for each clock \(c\) is derived by determining the largest constant to which clock \(c\) is compared. Then, the DBM treats all values above that threshold equally, which means that a stored upper bound for clock \(c\) never exceeds \(n_c\). We refer to [8, 11] for an in-depth coverage of DBM operations as well as their memory and runtime-efficient implementation. Lastly, we point out that, for comparison, when simulating a TA or PTA, considerably more complex clock constraints could be used employing Satisfiability Modulo Theories (SMT) solvers such as Z3 [40] to decide whether a PTA edge is enabled.

5 PROBABILISTIC TIMED GRAPH TRANSFORMATION SYSTEMS

In this section, we recall the formalism of PTGTSs [36] allowing for modeling and analysis of probabilistic real-time systems, discuss the modeling details for our running example, and consider which kinds of state spaces can be generated for PTGTSs, pointing out their advantages and drawbacks.

The formalism of PTGTSs captures structure dynamics by means of rule-based graph transformation (cf. [17]), timed behavior (employing clocks as in TA or PTA), and probabilistic choices (as in PA or PTA) among alternative outcomes of graph transformation steps. The subformalisms of TGTSSs [6, 42] and PGTSs [30, 31], including tool support for their analysis, have been introduced before. In each case, tool-based analysis support is obtained by translating a PTGTS, TGTS, or PGTS into the corresponding PTA, TA, or PA, respectively, and by reusing then the existing model checking support of, e.g., Prism or Uppaal for the resulting automata. Since such resulting automata are usually much larger (in terms of edges) than the given PTGTS, TGTS, or PGTS (in terms of their rules), we understand PTGTS, TGTS, or PGTS to be high-level languages for their automata counterparts allowing for more concise representations of equivalent behavior.

5.1 Syntax and Semantics of Probabilistic Timed Graph Transformation Systems

We employ type graphs (cf. [17]) such as the type graph \(TG\) from Figure 4 for our running example. A type graph describes the set of all admissible (typed attributed) graphs by mentioning the allowed types of nodes, edges, and attributes. We assume typed attributed graphs in which attributes are connected to local variables and Attribute Conditions (ACs) over a many sorted first-order attribute logic [44] are used to specify the values of these local variables.\(^4\) Morphisms

This approach to attribution has also been used before to capture constraints on attributes in graph conditions in [50] and to describe attribute modifications in [45, 51]. In our tool AutoGraph, ACs are checked for satisfiability using the SMT solver Z3 [40] for cases where the AutoGraph built-in simplification is insufficient due to more complex ACs used.
m : G₁ → G₂ between graphs must ensure that the AC of G₂ is more restrictive compared to the AC of G₁ (w.r.t. the mapping of variables by m). For example, a graph with a local variable x and an AC x ≥ 2 can be matched to a graph with a local variable y and an AC y = 4 using a match m mapping x to y, because y = 4 implies m(x ≥ 2) = (y ≥ 2). Hence, the AC ⊥ (false) has been used for the type graph TG to ensure that TG does not restrict attribute values of instance graphs, because typing morphisms from instance graphs to TG trivially satisfy the required implication.

Lastly, monomorphisms (short monos), denoted by m : G₁ → G₂, are the morphisms that map the nodes, edges, attributes, and local variables contained in G₁ injectively to those in G₂.

**Graph Conditions (GCs)** [23, 51] of **Graph Logic (GL)** (such as the one in Figure 6(a)) are used to state structural properties on a graph requiring the presence or absence of certain subgraphs in that graph as well as ACs on attributes.

**Definition 3 (GCs).** For a graph H, ϕₖ is a graph condition (GC) over H defined as follows:

ϕₖ := ⊤ | ¬ϕₖ | ϕₖ ∧ ϕₖ ′ | ∃(f, ϕₖ ′)

where f : H ↣ H′ is a mono and where additional operators such as ⊥, ∨, and ∀ are derived as usual.

The satisfaction relation [23, 51] for GL defines when a mono satisfies a GC. For our running example, the GC from Figure 6(a) is satisfied by all graphs, in which two shuttles are at the same track.

**Definition 4 (Satisfaction of GCs).** A mono m : H ↣ G satisfies a GC ϕ over H, written m |= ϕ, if one of the following items applies:

• ϕ = ⊤.
• ϕ = ¬ϕ′ and m ̸|= ϕ′.
• ϕ = ϕ₁ ∧ ϕ₂, m |= ϕ₁, and m |= ϕ₂.
• ϕ = ∃(f : H ↣ H′, φ′) and ∃m′ : H′ ↣ G, m′ o f = m ∧ m′ |= ϕ′.

Moreover, if ϕ is a GC over the empty graph ∅, i(G) : ∅ ↣ G is an initial morphism, and i(G) |= ϕ, then the graph G satisfies ϕ, written G |= ϕ.

According to [17], a **Graph Transformation (GT)** step is performed by applying a GT rule ρ for a match m : L ↣ G on the graph G to be transformed. In that sense, the GT rule captures a modification pattern while the match m captures where the modification is to be applied in G. In this article, we employ GT rules of the form ρ = (ℓ : K ↣ L, r : K ↣ R, γ, φ, lab) consisting of two monos ℓ and r, an AC γ, an application condition φ (given by a GC), and a label lab. A GT rule ρ can only be applied if the match m satisfies its application condition φ. If the application condition φ is satisfied, then the GT rule ρ is applied by (a) deleting the graph elements in L − ℓ(K), (b) adding the graph elements in R − r(K), (c) deriving the values of variables in R from the values of variables in L using the AC γ (e.g., x′ = x + 2) in which the variables from L and R are used in
Fig. 5. DPO diagram and GT step for the running example.

(a) DPO diagram transforming a graph $G$ into a graph $G''$ by applying a rule $\rho$ for a match $m$.

Fig. 6. A PTGT AP and a PTGT invariant for the running example.

(b) GT step advancing the shuttle $S_1$ from track $T_1$ to track $T_2$.

WenowintroducePTGTSsintermsoftheirstates,PTGTAPs,PTGTinvariants,andPTGTrules.

PTGTS states (similarly to PTA) are given by pairs $(G, v)$ of a graph and a clock valuation. The initial state of a PTGTS is given by a distinguished initial graph and the initial valuation setting unprimed and primed form, respectively, and (d) generating a label describing the preformed GT step more precisely by replacing the variables contained in $lab$ (ranging over the variables from $L$ and $R$) by the values used for them in item (c). For the items (a) and (b), see Figure 5(a) for a diagram visualizing the GT step using the DPO approach [17] (where the two depicted squares are pushouts in the category of symbolic graphs). Moreover, see Figure 5(b) for a concrete example of a GT step where the edge $e_1$ is removed, the edge $e_4$ is added, and the variable $c_1$ is changed from $\top$ to $\bot$. In this example, the applied rule $\text{drive}$ is labeled with the values $(1, DRIVE, 3, 10, 11, 12)$ for the used variables $(sid_1, m_1, minD_1, tid_1, tid_2)$. We refer to [17] for further details on the DPO approach in general as well as to [51] for the here employed graph transformation approach based on symbolic graphs.

We now introduce PTGTSs in terms of their states, PTGT APs, PTGT invariants, and PTGT rules. PTGTS states (similarly to PTA) are given by pairs $(G, v)$ of a graph and a clock valuation. The initial state of a PTGTS is given by a distinguished initial graph and the initial valuation setting.

5Similarly, for readers familiar with the Henshin tool [26], we point out that Henshin also similarly generates labels for GT steps when generating a state space for a GTS.
all clocks contained in that initial graph to 0. We denote the set of all clocks of a graph \( G \) (given by variables of sort real) by \( C(G) \). In our running example, each variable used for the \( \text{clockDrive} \) attributes of \( \text{Track} \) nodes represents a clock. Since we omitted the attributes in the visualization of the fragment topologies in Figure 2(b), we refer to the fragment topologies given in unabbreviated notation in Appendix B) p. 56 showing the one-to-one connection between tracks and their \( \text{clockDrive} \) attributes representing clocks. Moreover, considering the concrete GT step in Figure 5(b) again, we note that the ACs of the two graphs (given in the red boxes) contain no information on the clocks \( d_1 \) and \( d_2 \), as these are stored in thevaluations of the corresponding states instead.

PTGT APs and PTGT invariants are specified for PTGTs by means of GCs. PTGT APs are employed, as for PTA, to label states and to specify properties, which can then be further analyzed. However, as for PTA and as required by the model checking algorithms, PTGT APs may not state restrictions on clocks, since the labeling of PTGT states with PTGT APs must be independent from the valuation of the state. For an example of a PTGT AP, see Figure 6(a) visualizing the PTGT AP \( \text{AP}_{\text{collision}} \), which is satisfied by every graph where two shuttles are at the same track. PTGT invariants are used to rule out inadmissible PTGT states. For an example of a PTGT invariant, see Figure 6(b) visualizing the PTGT invariant \( \text{INV}_{\text{driving}} \), which rules out PTGT states where a shuttle \( S_1 \) in mode \( \text{DRIVE} \) is at a track \( T_1 \) longer than permitted by the clock of \( T_1 \). Note that the \( \text{minDur} \) (minimal duration) attribute of a shuttle encodes the lower bound on how much time the shuttle can stay on a track in mode \( \text{DRIVE} \), whereas the upper bound is always given by the lower bound increased by 1.

PTGT rules of a PTGTS then correspond to edges of a PTA (in the sense of being used to derive steps) and contain (a) a common left-hand side graph \( L \), (b) an AC over the clocks contained in \( L \) to capture a guard, (c) a natural number representing a priority where higher numbers denote higher priorities, and (d) a non-empty set of tuples of the form \((\ell : K \rightarrow L, r : K \rightarrow R, \gamma, \phi, \text{lab}, C, p)\), where \((\ell, r, \gamma, \phi, \text{lab})\) is an underlying GT rule as introduced above not deleting or adding clock attributes, \( C \) is a set of clocks contained in \( R \) to be reset, and \( p \) is a real-valued probability from \([0,1]\) where the probabilities of all such tuples must add up to 1.

**Definition 5 (PTGT Rule).** A probabilistic timed graph transformation rule (PTGT rule) \( \sigma \) is a tuple with the following components:

- \( \text{lhs}(\sigma) \) is a common left-hand side graph of the underlying GT rules,
- \( \text{guard}(\sigma) \in \text{CC}(\text{C}\left(\text{lhs}(\sigma)\right)) \) is a guard defined as a clock constraint over the clocks from the left-hand side graph \( \text{lhs}(\sigma) \),
- \( \text{prio}(\sigma) \in \mathbb{N} \) is the priority assigned to \( \sigma \),
- \( \text{rules}(\sigma) \) is a finite set of GT rules \( \rho \) with \( \text{lhs}(\rho) = \text{lhs}(\sigma) \) with the trivial application condition \( \top \) over \( \text{lhs}(\sigma) \) where \( \text{lhs}(\rho) \) is the left-hand side graph of the GT rule \( \rho \),
- \( \text{appCond}(\sigma) \) is an application condition over \( \text{lhs}(\sigma) \),
- \( \text{reset}(\sigma)(\rho) \subseteq \text{C}(\text{rhs}(\rho)) \) identifies the clocks to be reset for each \( \rho \in \text{rules}(\sigma) \), and
- \( \text{dpd}(\sigma) \in \text{DPD}(\text{rules}(\sigma)) \) is a DPD on \( \text{rules}(\sigma) \) with \( \text{supp}(\text{dpd}(\sigma)) = \text{rules}(\sigma) \).

For an example of a PTGT rule, see Figure 7(a) depicting the PTGT rule \( \text{drive} \) that is used to advance a shuttle to the next track. This PTGT rule has a single underlying GT rule \( \text{drive}_{\text{done}} \) (indicated by [done] left to the GT rule), which was applied to derive the GT step given in Figure 5(b). For enhanced readability, we depict the graphs \( L, K, \) and \( R \) of such underlying GT rules in a single graph, where graph elements to be removed and to be added are annotated with \( \ominus \) and \( \oplus \), respectively. Moreover, the entire PTGT rule is annotated with (a) an AC, called \( \text{attribute guard} \), which is used to express attribute restrictions on matched attributes and which is here trivially true, (b) the \( \text{clock guard} \) (representing the guard of the PTGT rule) \( d_1 \geq \text{minD}_1 \), which is a clock constraint.
(a) PTGT rule \textit{drive} for advancing a shuttle to the next track (see Figure 33|p.63 for the entire application condition).

(b) PTGT rule \textit{set-slow} for slowing down a shuttle at a yellow traffic light.

(c) PTGT rule \textit{construction-site-brake} for executing an emergency brake when a fast shuttle approaches a construction site.

Fig. 7. Selection of PTGT rules for the running example.
when $\text{minDur}_1$ is replaced by a concrete value based on the match, (c) the priority of the PTGT rule, and (d) a list of variables contained in $L$ to be used to label resulting PTGT steps. Also, each underlying GT rule is annotated with (i) an AC, called attribute effect, describing how variables are changed by GT rule application, (ii) a list of unchanged variables (i.e., for each variable $x$ in this list, we basically require $x = x'$), (iii) the set of clocks to be reset after the GT rule has been applied, and (iv) the probability assigned to that underlying GT rule. Note that the attribute guard, the attribute effect, and the list of unchanged variables are formally encoded in the AC $\gamma$ of the underlying GT rule but are annotated here separately for improved readability. Finally, the PTGT rule $\text{drive}$ is equipped with an application condition$^6$ (where a triangle separates this application condition from the underlying GT rules). This application condition expresses that $\text{drive}$ is not applicable when tracks $T_2$ or $T_3$ are already occupied by another shuttle. If $T_2$ is already occupied, then the shuttle willing to drive will try to stay on $T_1$ longer and attempt to drive later onto $T_2$. If $T_3$ is already occupied, then another PTGT rule for stopping a shuttle should be used instead of $\text{drive}$.

For a PTGT rule with two underlying GT rules, see the PTGT rule $\text{set-slow}$ in Figure 7(b), which is used to slow down a fast shuttle at a yellow traffic light in the GT rule indicated by [success] (the velocity is decreased by increasing the $\text{minDur}$ attribute from 2 to 3) unless the shuttle does not recognize the traffic light keeping its current speed unchanged in the GT rule indicated by [failure] (representing a failure on demand). In both underlying GT rules, the active attribute of the yellow traffic light is changed from $\top$ (see the attribute guard) to $\bot$ (see both attribute effects) ensuring that the PTGT rule cannot be applied multiple times in direct succession for the same shuttle. Also note that both underlying GT rules of $\text{set-slow}$ are equipped with probabilities (distinct from 1) and that both these probabilities add up to 1, as required. In this particular PTGT rule, the use of the probability $10^{-6}$ for the failure case indicates that the probability for a failure on demand is very small, which can be the result of a rather reliable mechanism for observing traffic lights.

A PTGTS is then defined to be a tuple containing the components introduced before.

**Definition 6 (PTGTS).** A probabilistic timed graph transformation system (PTGTS) $S$ is a tuple with the following components:

- $iG(S)$ is a finite initial graph,
- $\text{rules}(S)$ is a finite set of PTGT rules,
- $\text{invs}(S)$ is a finite set of PTGT invariants, which are all satisfied for the initial graph and the initial clock valuation ($iG(S)$, $CIV(C(iG(S))))$, and
- $\text{aps}(S)$ is a finite set of PTGT APs.

We now define PTGT states and an equivalence relation on them based on graph isomorphisms that are compatible with clock valuations.

**Definition 7 (PTGT state).** A potential PTGT state $(G, v)$ given by a finite graph $G$ and a clock valuation $v \in CIV(C(G))$ is a PTGT state of $S$, written $(G, v) \in \text{States}(S)$, if $(G, v)$ satisfies all PTGT invariants $\phi \in \text{invs}(S)$ of $S$.

Moreover, two given PTGT states $(G_1, v_1)$ and $(G_2, v_2)$ are equivalent, written $(G_1, v_1) \equiv (G_2, v_2)$, if there is an isomorphism $\iota : G_1 \rightarrow G_2$ such that $v_2 \circ \iota = v_1$. The equivalence relation $\equiv$ also induces equivalence classes denoted by $[(G_1, v_1)]_{\equiv}$.

The semantics of a PTGTS is given by its induced PTS as in [36, 38] using here PTGT states instead of their equivalence classes for brevity.

---

$^6$Note that for better comprehension, we only show a simplified version of the used application condition here. For the entire application condition, see Figure 33 p. 61, where further rule applications are ruled out based on nearby shuttles on joining or forking track segments or nearby rigid elements such as construction sites or conflicts.
Definition 8 (PTS Induced by PTGTS). Every PTGTS $S$ induces a unique unique probabilistic timed system (PTS) $\text{PTGTS} \rightarrow \text{PTS}(S) = P$ consisting of the following components:

- **states** ($P$) contains as PTS states pairs $(G, v) \in \text{States}(S)$ where $G$ is a graph and $v$ is a valuation of the clocks of $G$ satisfying the PTGT invariants of $S$,
- **istate** ($P$) is the unique initial state from states($P$) consisting of the initial graph of $S$ and the initial clock valuation of its clocks,
- **acts** ($P$) contains tuples of the form $(\sigma, m, \rho, \text{values})$ consisting of the used PTGT rule $\sigma$, the used match $m$, the used GT rule $\rho$, and the values obtained for the variables listed as labels in $\rho$ as explained above,
- **steps** ($P$) is the set of PTS steps (see [36, 38] for a full definition of induced timed and discrete steps). A PTS step $((G, v), a, \mu) \in \text{steps}(P)$ contains a source state $(G, v)$, an action $a$ from acts($P$) for a discrete step or a duration $a$ from $R_0^+$ for a timed step, and a DPD $\mu$ assigning a probability to each possible target state.
- **aps** ($P$) = aps($S$) is the same set of PTGT APs, and
- **lab** ($P$)$(G, v)$ contains those PTGT APs $\phi \in \text{aps}(S)$ that are satisfied by $G$.

5.2 Modeling of Running Example as Probabilistic Timed Graph Transformation System

The PTGTS modeling our running example uses the type graph $TG$ from Figure 4 and contains 4 PTGT APs, 21 PTGT rules (note that the 8 PTGT rules with prefix border- in Table 2| p. 60 are used for the fragment PTGTSs only), and 5 PTGT invariants, which are given in detail in Appendix C| p. 59. We do not provide a concrete initial graph for that PTGTS but mention the rather small topology from Figure 2(a) as one potential choice.

In the following, we discuss in more detail the two undesirable situations where shuttles execute emergency brakes. To this end, we present, in addition to the PTGT rules drive and set-slow discussed in the previous subsection, further PTGT rules that are of central relevance for the running example along with paths demonstrating their application.

**Situation 1: Traffic Lights and Execution of Emergency Brakes at Construction Sites.**

The PTGT rule construction-site-brake (see Figure 7(c)) is used by a fast shuttle to execute an emergency brake when approaching a construction site. This PTGT rule has a single underlying GT rule that changes the mode of the shuttle from DRIVE to BRAKE (i.e., a shuttle that is in mode STOP will not be able to execute an emergency brake using this rule) and, as for the PTGT rule drive, the shuttle attempts to not enter the next track as long as it is occupied by another shuttle (as required by the rule’s application condition).

In Figure 8, we provide a path for the fragment topology FT4 (see Figure 2(b)), where a single fast shuttle drives forward, fails to observe the yellow traffic light, and consequently has to execute an emergency brake when reaching the construction site subsequently.

**Situation 2: Connection Attempts between Shuttles and Emergency Brakes Ahead of Joins.**

The PTGT rule connect (see Figure 9(a)), containing two underlying GT rules, is used to establish a connection between two shuttles. The first underlying GT rule describes a successful connection establishment where a connection is added between the two shuttles located on opposite conflicting tracks. The second underlying GT rule describes a failing connection establishment where no such connection is added between the shuttles. In both cases, by changing the attribute canConnect from $\top$ to $\bot$ for the second shuttle arrived at the conflicting track, we prevent that the PTGT rule connect is applicable multiple times in direct succession for the same two shuttles (similarly to the PTGT rule set-slow where also a Boolean variable was used to disable the PTGT
Fig. 8. A path of the running example for FT4 (see Figure 2(b)) where the shuttle $S_1$ does not slow down and executes an emergency brake at the construction site. The current values of the shuttle attributes \textit{mode}, \textit{canConnect}, and \textit{minDur} as well as the value of the clock \textit{clockDrive} of the connected track are given for $S_1$ in brackets (thereby \textit{DRIVE} is abbreviated by \textit{D} and \textit{BRAKE} by \textit{B}). The current value of the yellow traffic light attribute \textit{active} is given for \textit{Y} in brackets as well and the interleaved timed steps are omitted. In this path, $(s0\rightarrow s1)$ the fast shuttle $S_1$ drives onto the track with the yellow traffic light, $(s1\rightarrow s2)$ a failure on demand happens such that $S_1$ does not decrease its speed from $[2, 3]$ time units per track to $[3, 4]$ time units per track, $(s2\rightarrow s3)$ $S_1$ drives to the fifth track, $(s3\rightarrow s4)$ the attribute \textit{active} of the yellow traffic light is reset to \textit{T} indicating that the rule \textit{set-slow} can be applied at \textit{Y} again, $(s4\rightarrow s5)$ $S_1$ executes an emergency brake by rapidly reducing its velocity to 0 time units per track and stopping on the track with the construction site (this state is then labeled with \textit{AP} brake given in Figure 30| p. 59), and $(s5\rightarrow s6)$ after some unrestricted delay $S_1$ drives to the next track.

The application condition of \textit{connect} essentially states that connections can only be established for shuttles when they are not already connected. This is important when shuttles establish a connection and are then later on opposite conflicting tracks again before disconnecting in between. Moreover, the PTGT rule \textit{connect} is equipped with a higher priority of 1, ensuring that two shuttles located on opposite conflicting tracks will perform the connection attempt. Note that we assume that a successful connection attempt is much more probable than a failing connection attempt.

The PTGT rule \textit{emergency-brake} (see Figure 9(b)) is used to execute an emergency brake to avoid a collision at a join track. This PTGT rule has a single underlying GT rule in which the driving shuttle $S_1$ executes an emergency brake changing into mode \textit{BRAKE} when there is already the other shuttle $S_2$ just ahead of the join track. If the shuttle $S_1$ would not brake using this PTGT rule but would instead continue to stay in mode \textit{DRIVE} when advancing to track $T_2$, then a collision on the join track $T_4$ would be possible when both shuttles would then advance to that track. The application condition of \textit{emergency-brake} states (a) that there is no shuttle on the track to which the shuttle $S_1$ is driving to (ensuring that the shuttle $S_1$ stays on its current track as long as possible...
Fig. 9. Details for the running example.

(a) PTGT rule `connect` for establishing a connection between two shuttles (see Figure 48|p.71 for the entire application condition).

(b) PTGT rule `emergency-brake` for executing an emergency brake to avoid a collision (see Figure 45|p.69 for the entire application condition).
when there is a shuttle on track $T_2$) and (b) that both shuttles are not connected (meaning that the shuttle $S_1$ may observe the shuttle $S_2$ close to the join track at a too-late point in time being on track $T_1$ in contrast to the situation, where the shuttle $S_1$ would know, due to an established connection, that there is the shuttle $S_2$ driving towards the join track).

In Figure 10, we provide a path for the fragment topology FT9 (see Figure 2(b)), where two shuttles successfully establish a connection and pass by the join track without stopping or braking. As an alternative behavior, we provide a second path for FT9 in Figure 11 where connection attempts fail and where one of the shuttles executes an emergency brake to avoid a collision.

5.3 State Spaces of Probabilistic Timed Graph Transformation Systems

As discussed in Section 4.2, the PTS induced by a PTA is infinite, since it includes all timed steps with delays taken from $\mathbb{R}^+$. The PTS induced by a PTGTS is infinite for the same reason and is therefore as well neither computable nor directly analyzable necessitating development of symbolic analysis techniques for PTGTSs as for PTA.

Along with the introduction of PTGTSs [36], a model checking approach was presented. The first step of this model checking approach is the generation of the structural state space of the PTGTS by only applying the underlying GT rules of the PTGT rules (thereby ignoring all guards, invariants, clock resets, priorities, and probabilities). While each state of this structural state space is given by a single graph (thereby abstracting from valuations contained in the states of the induced PTS), we still refrain from calling this structural state space symbolic, since we refer to another kind of state space as symbolic below. This structural state space is then translated into a PTA (see [36, 38] for a more technical explanation) by (i) merging all sets of GT steps with a common match using a GT rule of a common PTGT rule into a single PTA edge $e$, (ii) equipping such a PTA edge $e$ with the guards given in the PTGT rule, (iii) equipping such a PTA edge $e$ with the clock reset sets for each underlying GT rule, (iv) extending the guard of such a PTA edge $e$ to prevent its application when another PTA edge (with identical source location) was obtained for a PTGT rule with higher priority (by essentially adding the negation of the guard of that other PTA edge to the guard of the PTA edge $e$), and (v) adding invariants to all states of the structural state space (i.e., to the locations of the resulting PTA) by evaluating the PTGT invariants of the PTGTS for these states. The resulting PTA is then guaranteed to induce the same PTS as the PTGTS (up to renaming). Hence, this resulting PTA can be used as a replacement for the considered PTGTS and the analysis results for probabilistic timed reachability properties obtained for this PTA using, e.g., the PRISM model checker are then also valid for the PTGTS. The first drawback of this translational approach is that the structural state space may be infinite due to infinitely many graphs being reachable when ignoring the timing constraints given by guards, invariants, and clock resets (see [38] for an example). Even if the structural state space is not infinite, there may be a considerable amount of locations and PTA edges that are not reachable or applicable, respectively, in any path of the PTA. This problem is to be expected also for our running example, where the forward progress of shuttles (e.g., by application of the PTGT rule drive) is restricted by guards and PTGT invariants to be delayed to a time points in the intervals $[2, 3]$ or $[3, 4]$, depending on the shuttle’s speed. Ignoring these restrictions of shuttle behavior allows for unrealizable paths where, e.g., some shuttle drives forward while all other shuttles are violating their PTGT invariants. In that sense, the usage of time in PTGTSs ensures a level of fairness between shuttles (that are in mode DRIVE) forcing them to drive forward (using some PTGT rule) after a certain amount of time.

To mitigate the problem of generating unrealizable steps, we proposed in [38] an adaptation of the state space generation approach from above where the generation of structural state space was interleaved with the analysis of location reachability and applicability of PTA.
Fig. 10. A path of the running example for FT9 (see Figure 2(b)) where no shuttle brakes or stops. We use the same annotations as in Figure 8. In this path, (s0→s1) the fast shuttle $S_1$ drives first, (s1→s2) $S_1$ is the first shuttle at the first conflicting track, (s2→s3) the slow shuttle $S_2$ drives to the first conflicting track on its side, (s3→s4) both shuttles connect (making $S_1$ the convoy leader, as that shuttle reached the conflicting track first, and $S_2$ the convoy follower), (s4→s6) the shuttle $S_1$ drives twice (s6→s7) the shuttle $S_2$ follows in convoy, (s7→s8) the shuttle $S_1$ drives to the join track, and (s8→s9) both shuttles disconnect.
Fig. 11. A path of the running example for FT9 (see Figure 2(b)) where connection attempts fail and one shuttle executes and emergency brake. The PTGT state s0 here can be reached from the PTGT state s3 in Figure 10 via a failing connection attempt. We use the same annotations as in Figure 8. In this path, the connection attempt has just failed, (s0→s1) the fast shuttle $S_1$ drives to the second conflicting track, (s1→s2) $S_1$ is the first shuttle at the second conflicting track, (s2→s3) the slow shuttle $S_2$ drives to the second conflicting track on its side, (s3→s4) both shuttles fail again to connect, (s4→s5) $S_1$ drives to the third conflicting track, (s5→s6) $S_2$ executes an emergency brake, since not braking would mean that both shuttles are just ahead of the join track, and (s6→s9) $S_1$ drives three times while $S_2$ is blocked until there are two empty tracks between both shuttles in the last state.
edges. However, this approach is very costly, since the analyzing computations performed in one analysis iteration for some batch of additional locations and PTA edges would not be reusable in the next iteration. Moreover, the size of the resulting PTA eventually prohibits the use of model checkers (such as Prism and Uppaal) altogether for deciding location reachability and PTA edge applicability. To reduce the effort for these interleaved analysis steps, we implemented an approach based on PTA simulation in AUTOGRAPH where the resulting TA or PTA is randomly simulated (which requires almost no memory and can be performed quite fast) making it superfluous to model check for the reachability of locations and applicability of PTA edges witnessed during simulation. However, (a) when most locations are unreachable and most PTA edges are inapplicable, not much analysis effort is saved, since the simulation approach is not able to detect unreachable locations or inapplicable PTA edges, and (b) further developments are required to improve this simulation approach to result in a better coverage of the reachable locations and applicable PTA edges by the simulated runs.

The second drawback of this translational approach is that even when every location is reachable and every PTA edge is applicable in the resulting PTA, there is no guarantee that any structural path through that PTA given by a sequence of PTA edges is realizable. Hereby, realizability of the structural path means that the PTS induced by the PTA has a path where the PTA edges contained in the structural path are applied in that order interleaved with timed steps. Hence, the analysis described above checking for the resulting PTA whether all its locations are reachable and all its PTA edges are applicable is insufficient when the resulting PTA is to be analyzed by considering such structural paths only (i.e., without checking these paths for realizability). For comparison, the PRISM model checker does not analyze structural paths through such a PTA. Instead, during model checking, it generates a symbolic state space for that PTA and then evaluates the given probabilistic timed reachability properties by considering structural paths through the generated symbolic state space, which are known to be realizable.

To improve and extend the analysis capabilities for our compositional analysis approach, we propose the direct generation of a symbolic state space for a PTGTS employing zones. Thereby, we follow naturally the approach discussed in Section 4.2 for generating zone-based symbolic state spaces for PTA. For this purpose, as a first step, we included in AUTOGRAPH the support for the DBM-based handling of zones following [8]. As a second step, we then devised the symbolic state space generation procedure resulting for the given PTGTS in a symbolic state space, which contains states given by a pair \((G, \psi)\) of a graph \(G\) and a clock constraint \(\psi\) representing as a zone a set of valuations with which the graph \(G\) can be reached.

In this symbolic state space generation procedure, we define the required clock ceilings (cf. Section 4.2) based on their types, which could require to increase the number of types used when different clock ceilings would need to be assigned to clock variables of a common type. For our running example, we employ the clock ceiling of 4 for all clockDrive clocks, since 4 is the highest value with which one of these clocks is compared.

Moreover, deviating from standard approaches for TA, to improve memory consumption, we drop all clock constraints mentioning clocks from a DBM when these clocks are not used anymore.
before they are reset. For our running example, this allows us to only record clock constraints over those \textit{clockDrive} clocks that are connected to tracks with shuttles in mode \textit{DRIVE}. When such a shuttle then advances to the next track, the \textit{clockDrive} clock of the target track is reset to 0 according to our PTGT rules for driving and this clock reset is then included into the DBM of the resulting state.

Finally, we only record those clocks in the DBM of a state that are constrained by a PTGT invariant. For our running example, the PTGT invariant $\text{INV}_{\text{driving}}$ from Figure 6(b) expresses an upper bound on the \textit{clockDrive} clock of the connected track in the required way. By storing only active clocks in the DBM, we are, e.g., able to reduce the size of the DBM for the case of 2 shuttles and the fragment topology FT9 to $(2 + 1) \times (2 + 1)$ cells in the DBM instead of $(16 + 1) \times (16 + 1)$ cells for the 16 clocks connected to the 16 tracks of that fragment topology. Note that for PTGTSs (not for our running example), where clocks are intermittently not constrained by PTGT invariants but are later used in PTGT invariants or guards, additional PTGT invariants are required to ensure that DBMs are not purged from these actually relevant clocks. Moreover, note that we already employ this approach of storing only active clocks in the DBM when simulating PTA and when generating the symbolic state space for a PTA in \textsc{AutoGraph} to detect reachable locations and applicable PTA edges, as described above.

The advantages of generating such a zone-based symbolic state space are that (a) all states and steps of that state space are actually reachable and applicable, respectively, (b) the state space is finite if and only if a finite set of graphs is reachable for the given PTGTS, and (c) the structural paths through that state space are known to be realizable in terms of a path of the PTS induced by the PTGTS. However, such a zone-based symbolic state space may be considerably larger compared to the PTA generated from the structural state space when the PTA contains high rates of reachable locations and applicable PTA edges (i.e., the number of states in the structural state space does not exceed the number of reachable graphs of the PTGTS by a lot) while the zone-based symbolic state space of the same PTGTS contains (on average) for each graph $G$ a large number of states $(G, \psi_i)$ (i.e., the number of states in the zone-based symbolic state space exceeds the number of reachable graphs of the PTGTS by far).

The translational approach and the zone-based approach both have their merits, and their suitability depends on the properties to be analyzed. The usefulness of the translational approach lies in the applicability of, e.g., the \textsc{Prism} model checker. However, especially the third mentioned advantage (see item (c) above) is essential for improving and extending our analysis capabilities compared to [37], as discussed in Section 8 in more detail.

6 COVERAGE OF LARGE-SCALE TOPOLOGIES AND LARGE-SCALE STATES

Based on the technical and theoretical developments from the previous sections, we present in the following sections our decomposition-based approach for analysis of a PTGTS $S_0$ modeling a large-scale cyber-physical system along the lines of the informal discussion from Section 1. As a first step, we present formal definitions of (a) the large-scale topology contained in a large-scale state, (b) how such a large-scale topology is covered by embeddings of fragment topologies (constrained by so-called \textit{overlapping specifications}), and (c) how these embeddings result in the corresponding coverage of the large-scale state by fragment states (constrained by GCs). This static relationship between a large-scale state and its fragments is visualized in Figure 1(b). For our running example, we provide an instance of the square from Figure 1(b) in Figure 13.

In our context, a \textit{large-scale topology} is given by a graph $\overline{G}$. This graph $\overline{G}$ must be a subgraph of every graph $G$ that occurs in a \textit{large-scale state} $(G, \nu)$ of the PTS induced by the given PTGTS $S_0$. Consequently, the large-scale topology $\overline{G}$ is also contained in the initial graph $G_0$ of $S_0$ and...
no PTGT rule application ever removes any graph element or attribute contained in $G$. Therefore, we also call $G$ the static or rigid underlying topology of $S_0$. In general, we aim at determining the largest rigid underlying topology. However, since graph transformation is Turing-complete and the PTGTS $S_0$ is expected to be equipped with a very large initial graph, we must rely on user input defining the large-scale topology to be used in subsequent steps. In principle, the user could provide the large-scale topology by providing a mono $\kappa : G_0 \hookrightarrow G$ into the initial graph $G_0$ of the PTGTS $S_0$ at the concrete instance level (de)selecting every graph element and attribute individually. However, to simplify the notion of an overlapping specification below, we require that a mono $\kappa_{TG} : \overline{TG} \hookrightarrow TG$ into the type graph of the PTGTS $S_0$ is provided instead. The graph $\overline{TG}$ is then a type graph as well and the mono $\kappa_{TG}$ can be used straightforwardly to restrict an arbitrary graph $G$ typed over $TG$ to its rigid subgraph $\overline{G}$ (represented using a mono $\kappa$ in Figure 1(b)) by removing all graph elements and attributes of a type contained in $TG$ but not contained in $\overline{TG}$. For our running example, we employ the rigid type graph $\overline{TG}$ from Figure 12 along with the obvious mono $\kappa_{TG} : \overline{TG} \hookrightarrow TG$, thereby implementing our identification of static and dynamic elements from Section 3. Note that we can verify that graph elements contained in the rigid type graph $\overline{TG}$ are never added or deleted when applying any PTGT rule used for our running example (cf. Appendix C | p. 59) by manual inspection or using the fully-automatic technique of 1-induction from [16, 49].

A coverage of the large-scale topology is given by embeddings of fragment topologies. For our running example, see Figure 14(c) for an example of three such embeddings into a single large-scale topology (also, see Figure 28 for an example of three such embeddings not respecting the overlapping specification). Such a coverage is required to ensure that (a) every graph element of the large-scale topology is reached by some embedding and (b) every two embeddings must satisfy an overlapping specification whenever they reach at least one common graph element of the large-scale topology. In Appendix A, for a given set of fragment topologies and an overlapping specification, we consider from a conceptual perspective two different application scenarios for our modeling and analysis approach: (i) the composition of a large-scale topology from the fragment topologies deriving the coverage alongside and (ii) the decomposition of a given large-scale topology resulting in a coverage by employing available graph parsing techniques.

As visualized in Figure 1(b), we use monos $\alpha_i : \overline{F_i} \hookrightarrow G$ to represent the embedding of a fragment topology $\overline{F_i}$ into the large-scale topology $G$. However, we first introduce the notion of an overlapping specification to restrict the set of viable embeddings. In particular, an overlapping specification captures how two fragment topologies can overlap when being embedded into a large-scale topology. Subsequently, $\mathcal{F}$ denotes a set of fragment topologies to be embedded into a large-scale topology. For our running example, this set is given by the fragment topologies from Figure 2(b). Technically, an overlapping specification contains for each two fragment topologies a set of spans.
Fig. 13. Embedding $\kappa$ of a large-scale topology $\overrightarrow{G}$ into the graph $G$ of a large-scale state (given in Figure 2(a)), embedding $\alpha_i$ of the fragment topology FT9 from Figure 2(b) into the large-scale topology $\overrightarrow{G}$, embedding $\kappa_i$ of the fragment topology FT9 into the graph $F_i$ of a fragment state, and embedding $e_i$ of the graph $F_i$ of the fragment state into the graph $G$ of the large-scale state.
of the form \((o_1 : O \leftarrow \overline{F}_1, o_2 : O \leftarrow \overline{F}_2)\), where \(O\) is the permitted overlapping graph that is embedded into the two fragment topologies via \(o_1\) and \(o_2\) and which may vary among the spans.

**Definition 9 (Overlapping Specification).** If \(\mathcal{F}\) is a set of graphs, \(o(\overline{F}_1, \overline{F}_2)\) contains for each \(\overline{F}_1\) and \(\overline{F}_2\) from \(\mathcal{F}\) a finite set of spans \((o_1 : O \leftarrow \overline{F}_1, o_2 : O \leftarrow \overline{F}_2)\), then \(o\) is an overlapping specification for \(\mathcal{F}\).

For our running example, see Figure 14(a) depicting that the fragment topologies \(FT_1\) and \(FT_5\) from Figure 2(b) may only overlap in the last three tracks of \(FT_1\) and the first three tracks of \(FT_5\).

An overlapping specification is satisfied by a concrete embedding of an overlapping of the two graphs \(\overline{F}_1\) and \(\overline{F}_2\) into a common rigid graph \(\overline{G}\) if the overlapping given by that concrete embedding is captured by one of the overappings from the overlapping specification.

**Definition 10 (Satisfaction of Overlapping Specification).** If \(\mathcal{F}\) is a set of graphs, \(o\) is an overlapping specification for \(\mathcal{F}, \overline{F}_1\) and \(\overline{F}_2\) are graphs contained in \(\mathcal{F}\), \(\alpha_1 : \overline{F}_1 \hookrightarrow \overline{G}\) and \(\alpha_2 : \overline{F}_2 \hookrightarrow \overline{G}\) are two embeddings into the same rigid graph \(\overline{G}\) matching at least one common graph element, then \(((\overline{F}_1, \alpha_1), (\overline{F}_2, \alpha_2))\) is satisfied by \(o\) if there is some span \((o_1 : O \leftarrow \overline{F}_1, o_2 : O \leftarrow \overline{F}_2) \in o(\overline{F}_1, \overline{F}_2)\) such that for the pushout \((g_1 : \overline{F}_1 \rightharpoonup P, g_2 : \overline{F}_2 \rightharpoonup P)\) of \((\alpha_1, \alpha_2)\) (i.e., for the overlapping of \(\overline{F}_1\) and \(\overline{F}_2\) w.r.t. \((o_1, o_2)\)) there is some \(h : P \rightharpoonup \overline{G}\) with \(\alpha_1 = h \circ g_1\) and \(\alpha_2 = h \circ g_2\).

\[
\begin{array}{c}
\overline{F}_1 \\
\alpha_1
\end{array} \\
\begin{array}{c}
\alpha_1 \\
h
\end{array} \\
\begin{array}{c}
\overline{F}_2 \\
\alpha_2
\end{array}
\]

For our running example, see Figure 14(b) showing that the overlapping specification for fragment topologies \(FT_1\) and \(FT_5\) from Figure 14(a) is satisfied for concrete embeddings of \(FT_1\) and \(FT_5\).

Based on Definition 10, specifying when two embeddings are permitted by the overlapping specification, we now define when a list of such embeddings entirely covers the given large-scale topology without violating the overlapping specification.

**Definition 11 (Coverage of Large-scale Topology).** If \(\overline{G}\) is a graph representing a large-scale topology, \(\mathcal{F}\) is a set of fragment topologies, and \(o\) is an overlapping specification for \(\mathcal{F}\), then \(M\) is a coverage of the large-scale topology \(\overline{G}\) w.r.t. \(\mathcal{F}\) and \(o\) when \(M\) is a list of pairs of the form \(((\overline{F}_1, \alpha_1)\) with \(\overline{F}_1 \in \mathcal{F}\) and \(\alpha_1 : \overline{F}_1 \hookrightarrow \overline{G}\) describing each an embedding of a fragment topology \(\overline{F}_1\) into the large-scale topology \(\overline{G}\), the monos \(\alpha_i\) contained in pairs of \(M\) are jointly epimorphic (i.e., each graph element and attribute of \(\overline{G}\) is reached by at least one of these monos), and \(((\overline{F}_1, \alpha_1), (\overline{F}_2, \alpha_2))\) is satisfied by \(o\) for all two pairs \(((\overline{F}_1, \alpha_1)\) and \((\overline{F}_2, \alpha_2)\) contained in \(M\).

Considering some large-scale state \((G, v)\) reachable in the induced PTS of the PTGTS and the mono \(\kappa : \overline{G} \rightharpoonup G\) capturing the rigid underlying topology \(\overline{G}\) of the full graph \(G\) (also containing dynamic elements), the extended embedding \(\kappa \circ \alpha_i\) specifies how a fragment topology \(\overline{F}_i\) is embedded into \(G\) via the embeddings \(\alpha_i : \overline{F}_i \rightharpoonup \overline{G}\) and \(\kappa\) (cf. Figure 1(b)). This extended embedding into the full graph \(G\) may satisfy important properties. For our running example, we observe, e.g., that each such extended embedding satisfies the property that every shuttle located on one of the embedded tracks has an attribute \(minDur\) of 2 (for fast shuttles) or of 3 (for slow shuttles). The negation of this property has been stated as a PTGT AP \(AP\) velocity-unexpected in Figure 31] p. 59 to detect states where this expected property is violated. These kinds of properties that are invariantly satisfied by extended embeddings into the full graphs \(G\) of the given PTGTS should also be
(a) The unique span \((\alpha_1, \alpha_2)\) contained in the overlapping specification used for the running example for FT1 and FT5.

(b) Embeddings of FT1 and FT5 respecting the overlapping specification (cf. Figure 14(a)).

(c) Three embeddings \(\alpha_1 – \alpha_3\) of fragment topologies into a large-scale topology where two pairs of these embeddings overlap.

(d) Three embeddings \(\epsilon_1 – \epsilon_3\) of the graph parts of fragment states into the graph part of a large-scale state where two pairs of these embeddings overlap. Note that these embeddings are compatible with the embeddings from Figure 14(c).

Fig. 14. Visualizations of embeddings of fragment topologies and fragment states into large-scale topologies and large-scale states, respectively. The indices of tracks indicate their mapping in the morphisms.
satisfied when analyzing the behavior on the fragment topologies. Hence, such properties should not only be satisfied by the extended embedding $\kappa \circ \alpha_i$ but also by the mono $\kappa_i : F_i \hookrightarrow F_i$ specifying how the fragment topology $F_i$ is embedded into the graph part $F_i$ of the fragment state $(F_i, v_i)$ of the fragment PTGTS $S_i$ (cf. Figure 1(b)). For this purpose, we define constrained fragment topologies by equipping fragment topologies with GCs for stating relevant properties.

**Definition 12 (Constrained Fragment Topology).** If $\mathcal{F}$ is a set of fragment topologies and $\mathcal{F}_c$ is a set of pairs $(\overline{F}, \phi)$ where $\phi$ is a GC over $\overline{F}$ and $\overline{F}$ is contained in $\mathcal{F}$, then $\mathcal{F}_c$ is a set of constrained fragment topologies.

A large-scale state is then covered by an extension of the coverage of its underlying large-scale topology when (a) the two ways of embedding of fragment topologies into the large-scale state correspond (i.e., the square in Figure 1(b) must commute for each $i$), (b) the additionally provided GC for each fragment topology is satisfied by the embeddings into the corresponding fragment states, and (c) clocks assigned to unique fragment states have the same clock values in the fragment state and large-scale state according to the clock valuations. For our running example, see Figure 14(d) for three embeddings $e_i$ of the graph parts of fragment states into the graph part of the common large-scale state, which are compatible in the required way with the three embeddings $\alpha_i$ from Figure 14(c) of fragment topologies into the common large-scale topology.

**Definition 13 (Coverage of Large-scale State).** If $M$ is a coverage of a large-scale topology $\overline{G}$ w.r.t. $\mathcal{F}$ and $\circ$ (according to Definition 11), $\mathcal{F}_c$ is a set of constrained fragment topologies obtained from $\mathcal{F}$ by adding a GC to each fragment topology in $\mathcal{F}$ (according to Definition 12), $\kappa : \overline{G} \hookrightarrow G$ embeds the large-scale topology $\overline{G}$ into the large-scale state $(G, v)$, then $M_c(G, v)$ is a coverage of the large-scale state $(G, v)$ w.r.t. $\kappa$, $\mathcal{F}_c$, and $\circ$ when

- $M_c(G, v)$ is the list of tuples of the form $((F_i, v_i), (\overline{F}_i, \phi_i), \alpha_i : \overline{F}_i \hookrightarrow \overline{G}, \kappa_i : F_i \hookrightarrow F_i, e_i : F_i \hookrightarrow G)$ obtained as an extension of the corresponding pairs $(\overline{F}_i, \alpha_i : \overline{F}_i \hookrightarrow \overline{G})$ in $M$ using the clock valuation $v_i$, the GC $\phi_i$ added to $\overline{F}_i$ in $\mathcal{F}_c$, and the additional embeddings $\kappa_i$ and $e_i$,
- the monos $e_i$ contained in tuples in $M_c(G, v)$ are jointly epimorphic (i.e., each graph element and attribute of the graph part $G$ of the large-scale state is reached by at least one of these monos),
- for each tuple in $M_c(G, v)$ the square $\kappa \circ \alpha_i = e_i \circ \kappa_i$ commutes and the GC $\phi_i$ is satisfied by $\kappa \circ \alpha_i$ as well as by $\kappa_i$ (cf. items (a) and (b) above), and
- $v_i(e^c) = v(c)$ whenever $e_i$ is the only embedding that maps $e^c$ to $c$ (cf. item (c) above).

Based on the notion of the coverage of a large-scale state by fragment states with their underlying fragment topologies, we discuss in the next section how large-scale states and their coverage change when PTGT steps are performed.

For our running example, we now summarize our explanations on the employed rigid type graph, overlapping specification, constrained fragment topologies, and embeddings used in the coverage of large-scale states.

**Example 1 (Coverage for Running Example).** For our running example, we use the mono $\kappa_{TG} : \overline{TG} \hookrightarrow TG$ restricting the type graph $TG$ from Figure 4 to the rigid type graph $\overline{TG}$ from Figure 12.

Moreover, we use the eight fragment topologies from Figure 2(b) (given in unabbreviated notation in Appendix B p. 56). These eight fragment topologies are constrained by stating that there is no shuttle on an entering track (indicated, in Figure 2(b), by a red entering arrow) that is neither fast (i.e., has the $\text{minDur}$ value of 2) nor slow (i.e., has the $\text{minDur}$ value of 3). The fact that shuttles
also satisfy this property on all other tracks is then a consequence of the PTGTS rules and of the satisfaction of this property for all entering tracks.

The used overlapping specification states for any two fragment topologies FT1 and FTj (where i and j may be equal), as in Figure 14(a) for FT1 and FT5, that overlapping embeddings must overlap in the chain of three entering or exiting tracks. In Figure 2(b), these two kinds of entering and exiting track chains have been marked with a red arrow.

The embeddings 𝜅i and 𝜖j essentially assign all dynamic elements to fragment topologies resulting in fragment states. Firstly, they assign shuttles S connected to a track T in the large-scale state to all fragment topologies for which the embedding 𝜅i maps to the track T. Secondly, connections between two shuttles S1 and S2 are included in those fragment states that contain S1 and S2. Lastly, attributes of traffic lights are added to those fragment states that contain their corresponding traffic light nodes. Note, as visualized in Figure 14(d), dynamic elements may be assigned to multiple fragment states. For our running example, this applies only to shuttles with their attributes and edges connecting them to tracks in which multiple fragment topologies overlap.

In the next section, we introduce behavioral coverage relations to extend coverages from this section to also take steps of the involved PTGTSs into account.

7 BEHAVIORAL COVERAGE RELATION

We employ the coverage 𝐌(𝐺, 𝑣) of a large-scale state (𝐺, 𝑣) (introduced in Definition 13) capturing a decomposition of a single large-scale state into n fragment states as a foundation to establish a behavioral relationship between a given PTGTS S0 and n PTGTSs 𝑆i. These PTGTSs 𝑆i then operate on the n embeddings of fragment states into a common large-scale state of the PTGTS S0. For this purpose, we define a behavioral coverage relation, which establishes a relationship between the PTGTS S0 and the PTGTSs 𝑆i in terms of coverages 𝐌(𝐺, 𝑣i) for all reachable large-scale states (𝐺𝑖, 𝑣𝑖) of 𝑆0. We first focus on this behavioral relationship and then discuss the construction of the n PTGTSs 𝑆i for our running example.

To ease the subsequent presentation of the required behavioral relationship, we now recall the basic notions of simulations and bisimulations 𝑅 ⊆ 𝑄1 × 𝑄2 from, e.g., [41], introduced also for relating two graph transformation systems in [35]. Simulations and bisimulations relate the states of two systems and require that this relationship is preserved by the step behavior of these two systems, which are given in terms of their labeled step relations 𝑆1 × 𝑄1 × 𝐿1 × 𝑄1 and 𝑆2 × 𝑄2 × 𝐿2 × 𝑄2. A further relation 𝛿 ⊆ 𝐿1 × 𝐿2 is used to define corresponding labels in the two systems. While a simulation only requires the second system to mimic the steps of the first system, a bisimulation requires that the first system also mimicks the steps of the second system using the same relation between states. See Figure 15 for a simple example of a simulation relation.

Definition 14 (Simulation, Bisimulation). A relation 𝑅 ⊆ 𝑄1 × 𝑄2 is a simulation for 𝑆1 ⊆ 𝑄1 × 𝐿1 × 𝑄1, 𝑆2 ⊆ 𝑄2 × 𝐿2 × 𝑄2, and 𝛿 ⊆ 𝐿1 × 𝐿2, if for all (𝑞1, 𝑞2) ∈ 𝑅 and (𝑞1, 𝑙1, 𝑞1′) ∈ 𝑆1 there is (𝑞2, 𝑙2, 𝑞2′) ∈ 𝑆2 such that (𝑙1, 𝑙2) ∈ 𝛿 and (𝑞1′, 𝑞2′) ∈ 𝑅.

Moreover, 𝑅 is a bisimulation for 𝑆1, 𝑆2, and 𝛿, if 𝑅 is a simulation for 𝑆1, 𝑆2, and 𝛿 and 𝑅−1 is a simulation for 𝑆2, 𝑆1, and 𝛿−1.

When the step relations of the two systems are given by GTSs, the usage of a relation 𝛿 is in our context insufficient to capture corresponding steps, as step labels not only contain the employed GT rule but also the employed match. To suitably capture corresponding step labels, we therefore go a step further by employing the embeddings contained in a coverage of a large-scale state. See Figure 16 for a visualization where a discrete step of the PTGTS S0 is related to a corresponding step...
Fig. 15. Example of a simulation relation between two systems $T$ and $S$. A dotted edge from $t_i$ to $s_j$ represents a tuple $(t_i, s_j)$ in the simulation relation. The relation $\Delta$ used for the mimicking of step labels is given by $\{(a, a), (b, x), (c, y)\}$ and allows, e.g., for the simulation of a $b$ step by an $x$ step. The depicted simulation is no bisimulation, because for the tuple $(t_1, s_1)$ in the relation, there is no step from $t_1$ mimicking the $y$ step from $s_1$, which would need to be a $c$ step from $t_1$. In conclusion, when $t_0$ and $s_0$ are understood to be the start states of $T$ and $S$, then $S$ simulates $T$ but not vice versa.

Fig. 16. Correspondence of discrete steps between the large-scale system $S_0$ and one of its fragment systems $S_i$, which are preserving the respective rigid structure given by $G$ and $F_i$.

of the PTGTSs $S_i$ where the initially given embeddings $(\alpha_i, \kappa, e_i, \kappa_i)$ are replaced by $(\alpha_i, \kappa'', e_i'', \kappa_i'')$ only keeping the embedding of the fragment topology unchanged.

Based on the idea of simulation and bisimulation, we now present the notion of a behavioral coverage relation in two steps: We first present its set theoretic or syntactic structure and then present a sequence of five semantic simulation constraints that must be satisfied by a behavioral coverage relation.

Behavioral coverage relations are defined for a given PTGTS $S_0$ (PTGTS operating on large-scale topology) and the PTGTSs $S_i$ (PTGTSs operating on fragment topologies). Also, behavioral coverage relations rely on the syntactic relationship given by coverages of large-scale states of $S_0$ by the corresponding small-scale states as introduced in Definition 13; to define the starting point for the subsequently defined semantic simulation constraints, we require the existence of a coverage $M_c(s_0)$ of the initial large-scale state $s_0$ of $S_0$ based on Definition 13 (PTGTSs operating on fragment topologies). Moreover, to allow to derive results for the PTGTS $S_0$ from the analysis based on model checking of the PTGTSs $S_i$, we employ a set of PTGT APs $\mathcal{A}$ that is a subset of the PTGT APs of $S_0$. 

Formal Aspects of Computing, Vol. 35, No. 3, Article 16. Publication date: September 2023.
and of each $S_i$ (Common set of atomic propositions). For our running example, this set $\mathcal{A}$ contains the first three PTGT APs given in Table 2] p. 60 to analyze, e.g., the collision of shuttles.\textsuperscript{8}

**Definition 15 (Shape of Behavioral Coverage Relation).** Let the following be given:

- (PTGTS operating on large-scale topology) $S_0$ is a PTGTS with initial large-scale state $s_0 = (G_0, v_0)$, where the underlying large-scale topology is identified via $\kappa_0 : \mathcal{G} \leftrightarrow G_0$ (and preserved by all steps of the PTGTS $S_0$).
- (PTGTSs operating on fragment topologies) For each $1 \leq i \leq n$, $S_i$ is a PTGTS with initial fragment state $s_{0,i} = (F_{0,i}, v_{0,i})$, where the underlying fragment topology is identified via $\kappa_{0,i} : \mathcal{F}_{0,i} \leftrightarrow F_{0,i}$ (and preserved by all steps of the PTGTS $S_i$).
- (Coverage of the initial large-scale state) $M_\kappa(s_0)$ is a coverage of the large-scale state $s_0$ w.r.t. $\kappa_0$, the set of constrained fragment topologies $\mathcal{F}_c = \{(\mathcal{F}_{0,i}, \phi_i) \mid 1 \leq i \leq n\}$, and some overlapping specification $o$ (cf. Definition 13), and
- (Common set of atomic propositions) For each $0 \leq i \leq n$, $\mathcal{A}$ is a set of APs contained in $S_i$. 

$S_{BCR}$ is a behavioral coverage relation between $S_0$ and $(S_1, \ldots, S_n)$ containing tuples of the form $((G, v), \kappa : \mathcal{G} \leftrightarrow G, w)$, where $s = (G, v)$ is a large-scale state of $S_0$, $\kappa$ identifies the large-scale topology of $G$, and $w$ is a list of coverages $M_\kappa(s)$ of the large-scale state $s$ w.r.t. $\kappa$, $\mathcal{F}_c$, and $o$ when the subsequently presented five simulation constraints are satisfied.

We now introduce the five semantic constraints that must be satisfied by every behavioral coverage relation.

In contrast to [37], we tailored here the subsequent definitions of the semantic constraints to the special case that the PTGTS $S_0$ (as in our running example) only employs GCs of a restricted form. In particular, we assume that, for some graph $P$, PTGT APs are of the form $\exists(f : \emptyset \hookrightarrow P, \tau)$, PTGT invariants are of the form $\neg \exists(f : \emptyset \hookrightarrow P, \tau)$, and application conditions in PTGT rules (that are conjunctions of GCs) are of the form $\neg \exists(f : \emptyset \hookrightarrow P, \tau)$. This restriction simplifies the identification of parts of fragment states and large-scale states that are considered for an evaluation of such GCs in the subsequent definitions. While a more general formalization of the following semantic constraints covering the general case of PTGTS making use of arbitrary nested GCs is feasible, we believe that the restriction in these definitions simplifies the application of our definitions for concrete examples because less complex properties have to be verified, since the aspect of match identification for GC evaluation is already included in the following definitions.

Intuitively, the five semantic constraints require the following:

- In Definition 16, we require that the initial states of $S_0$ and the $n$ PTGTSs $S_i$ are contained in the behavioral coverage relation.
- In Definition 17, we require that PTGT APs to be considered for analysis are correspondingly evaluated in the PTGTSs.
- In Definition 18, we require that timed steps are performed correspondingly (in terms of a bisimulation) in the PTGTSs, which primarily translates to the corresponding evaluation of PTGT invariants in the PTGTSs.
- In Definition 19, we require that discrete steps of $S_0$ can be mimicked (in terms of a simulation) by the PTGTSs $S_i$ responsible for representing the part of the large-scale state of $S_0$ where the step was applied.

\textsuperscript{8}To satisfy the requirement that related states are labeled with the same PTGT APs does not require all PTGTSs to use the same set of PTGT APs. For our running example, the PTGT AP AP\textsubscript{disconnected-on-border} is not used for the PTGTSs for those fragment topologies where shuttles cannot establish a connection, which are all fragment topologies except for FT9.
In Definition 20, we require that discrete steps performed by some PTGTS $S_i$ in isolation (on a part of the large-scale topology where the fragment topology $F_i$ does not overlap with the fragment topology $F_j$ of another PTGTS $S_j$ with $i \neq j$) can be mimicked by $S_0$ (in terms of simulations).

We now present these constraints separately.

We now define that initial states of the considered PTGTSs must be related in a behavioral coverage relation $S_{BCR}$ based on the coverages assumed in Definition 15. Technically, this means that the initial state $s_0$ of the PTGTS $S_0$ must be contained in some tuple $(s_0, \kappa_0, w)$ in $S_{BCR}$ and that it is related (for each $i$) with the initial state $s_{0,i}$ of the PTGTS $S_i$. 

**Definition 16 (Init Constraint).** In any behavioral coverage relation, the initial large-scale state $s_0$ of $S_0$ is related. That is, $(s_0, \kappa_0, w) \in S_{BCR}$ for some $w$ where the $i$th coverage $M_i(s_0)$ in $w$ given by $((F_i, \nu_i), (\overline{F}_i, \phi_i), \alpha_i : \overline{F}_i \rightarrow \overline{G}, \kappa_i : \overline{F}_i \rightarrow F_i, e_i : F_i \rightarrow G_0)$ satisfies $(F_i, \nu_i) = s_{0,i}$ and $\kappa_i = \kappa_{0,i}$.

We now define that states of $S_0$ must be labeled with a PTGT AP if a $S_{BCR}$-related state of $S_i$ is labeled with that PTGT AP. For this, we employ the set $\mathcal{A}$ of common PTGT APs employed in Definition 15, as already discussed before Definition 15. Technically, the behavioral coverage relation $S_{BCR}$ requires that only those fragment states and large-scale states are related in $S_{BCR}$ for which a labeling of the large-scale state using one of these PTGT APs from $\mathcal{A}$ implies the corresponding labeling of a fragment state. As an example of such an AP, we refer to $AP_{coll}$ from Figure 6(a) for identifying states in which two shuttles collided on a track. In the following definition, we exploit the restricted syntax of PTGT APs $ap \in \mathcal{A}$ and how such a restricted PTGT AP is satisfied using a mono $k$ to require that and how this satisfying match is correspondingly available in the fragment state. For general nested GCs, this definition would be much more concise, but its application would be more involved.

**Definition 17 (Labeling Constraint).** In any behavioral coverage relation, a labeling in a large-scale state can only be caused by a labeling in a fragment state. That is, if $((G, \nu), \kappa : \overline{G} \rightarrow G, w) \in S_{BCR}$, $ap = \exists (f : \emptyset \rightarrow P, \tau) \in \mathcal{A}$, and the mono $k : P \rightarrow G$ shows that $ap$ is satisfied by $G$, then there is some $1 \leq i \leq n$ such that $((F_i, \nu_i), (\overline{F}_i, \phi_i), \alpha_i : \overline{F}_i \rightarrow \overline{G}, \kappa_i : \overline{F}_i \rightarrow F_i, e_i : F_i \rightarrow G)$ is the $i$th element of $w$, and there is some $k_i : P \rightarrow F_i$ with $k = e_i \circ k_i$ showing that $ap$ is satisfied by $F_i$ as well.

We now define that timed steps from related states in a behavioral coverage relation $S_{BCR}$ can always be mimicked in terms of a bisimulation. Since timed steps are restricted by PTGT invariants only, we need to require that PTGT invariants restrict timed steps correspondingly in the involved PTGTSs. Similarly to the previous case of PTGT APs, we rely here on the restricted form of PTGT invariants to derive a more explicit constraint.

**Definition 18 (Temporal Constraint).** In any behavioral coverage relation, if $((G, \nu), \kappa : \overline{G} \rightarrow G, w) \in S_{BCR}$ and $S_0$ has a timed step with duration $\delta$ (not involving any PTGT rule by definition) from $(G, \nu)$ to $(G, \nu + \delta)$, then there is some tuple $((G, \nu + \delta), \kappa : \overline{G} \rightarrow G, w') \in S_{BCR}$ where $w'$ is obtained pointwise from $w$ by applying the corresponding timed step to each $((F_i, \nu_i), (\overline{F}_i, \phi_i), \alpha_i : \overline{F}_i \rightarrow \overline{G}, \kappa_i : \overline{F}_i \rightarrow F_i, e_i : F_i \rightarrow G) \in w$ resulting in $((F_i, \nu_i + \delta), (\overline{F}_i, \phi_i), \alpha_i : \overline{F}_i \rightarrow \overline{G}, \kappa_i : \overline{F}_i \rightarrow F_i, e_i : F_i \rightarrow G) \in w'$. Vice versa, the PTGTS $S_0$ has a timed step with duration $\delta$ for any common timed step of each $S_i$ with duration $\delta$.

For concrete applications of behavioral coverage relations, we point out that evaluations of PTGT APs and PTGT invariants spanning across more than one fragment state in a large-scale state are likely to be incompatible with the presented approach. Hence, PTGT APs and PTGT invariants should only perform local evaluations in a way that is compatible with the decomposition of the...
large-scale state as in our running example. Employing restricted GCs for PTGT APs and PTGT invariants certainly supports the process of ensuring that the evaluation of such GCs in refined to individual small-scale states in every case.

We now define that discrete steps of the large-scale PTGTS $S_0$ (from a state related in the behavioral coverage relation $S_{BCR}$) can be mimicked by the corresponding ($S_{BCR}$-related) small-scale PTGTS $S_1$ that is responsible for the part of the large-scale state, where the discrete step was applied. Refer to Figure 16 for a diagram representing how a step of $S_0$ is correspondingly mimicked by some $S_1$ while the embeddings are transferred resulting again in $S_{BCR}$-related states. Since discrete steps do not modify the underlying rigid structure, the decomposition of the large-scale topology is preserved in terms of embeddings $(\kappa, \kappa_i)$ into embeddings $(\kappa'', \kappa_i'')$. In the following definition, we distinguish between the two cases of a PTGTS $S_1$ either (i) being not affected by the discrete step of the PTGTS $S_0$ in which case the state belonging to that PTGTS does not need to be modified or (ii) being affected by the discrete step of the PTGTS $S_0$ in which case the state belonging to that PTGTS is modified by the application of a corresponding discrete step of $S_1$. In the latter case, Figure 16 explains already how such a discrete step is mimicked locally by the PTGTS $S_1$. For our running example, a discrete step of the large-scale PTGTS is mimicked by at most two small-scale PTGTSs, since, according to the employed overlapping specification, three or more fragment topologies cannot overlap and matches of left-hand sides of underlying GT rules are also sufficiently localized such that they never involve more than two fragment states. However, when the step of the PTGTS $S_0$ is performed on the core of a some fragment state (i.e., not on the border of a fragment state, as depicted in Figure 1), it is mimicked by modifying only this single affected fragment state. For our running examples, only driving and stopping steps of shuttles are performed on borders of FTs (and thereby also by more than one fragment state), while these and all further PTGT rules can be applied also on the core of some fragment state.

**Definition 19 (Large-to-small Constraint).** In any behavioral coverage relation, if

- $(G, v, \kappa : G \rightarrow w) \in S_{BCR}$ and
- $S_0$ performs the discrete step from $(G, v)$ to $(G'', v'')$ using an underlying GT rule $\rho = (\ell : K \leftarrow L, r : K \leftarrow R, y, \phi_{ac, lab})$ (given in Figure 16) where, since the discrete step preserves the large-scale topology $G$, there are unique $\kappa' : G \rightarrow G'$ and $\kappa'' : G \rightarrow G''$ such that $\ell \circ \kappa' = \kappa$ and $\kappa'' = r \circ \kappa'$, then
- $((G'', v'', \kappa'') : G'' \rightarrow w'') \in S_{BCR}$ for some $w''$ that is obtained pointwise from $w$ by adapting each tuple $((F_i, v_i), (\bar{F}_i, \phi_i), \alpha_i : \bar{F}_i \leftarrow \bar{G}, \kappa_i : \bar{F}_i \leftarrow F_i, e_i : F_i \rightarrow G) \in w$ to a resulting tuple $((F_i'', v_i''), (\bar{F}_i, \phi_i), \alpha_i : \bar{F}_i \leftarrow \bar{G}, \kappa''_i : \bar{F}_i \leftarrow F_i', e_i' : F_i' \rightarrow G'') \in w''$ as follows:
  - If $m(L) \cap e_i(F_i) = \emptyset$ (i.e., the discrete step does not match a part of the fragment state $F_i$), then the tuple $((F_i, v_i), (\bar{F}_i, \phi_i), \alpha_i : \bar{F}_i \leftarrow \bar{G}, \kappa_i : \bar{F}_i \leftarrow F_i, e_i : F_i \rightarrow G)$ remains unchanged.
  - Otherwise, the PTGTS $S_1$ must simulate the discrete step and the tuple $((F_i, v_i), (\bar{F}_i, \phi_i), \alpha_i : \bar{F}_i \leftarrow \bar{G}, \kappa_i : \bar{F}_i \leftarrow F_i, e_i : F_i \rightarrow G)$ needs to be updated according to the following items:
    - There must be a discrete step of $S_1$ from $F_i$ to $F_i''$ as given in Figure 16 for some underlying GT rule $\rho_1 = (\ell_i : K_i \leftarrow L_i, r_i : K_i \leftarrow R_i, y_i, \phi_{ac, i, lab_i})$ with the same probability and priority as $\rho$.
    - Since the discrete step of $S_1$ preserves the fragment topology $\bar{F}_i$, there are unique $\kappa'_i : \bar{F}_i \leftarrow \bar{F}_i'$ and $\kappa''_i : \bar{F}_i \leftarrow \bar{F}_i''$ such that $\ell_i \circ \kappa'_i = \kappa_i$ and $\kappa''_i = r_i \circ \kappa''_i$.
    - Moreover, the discrete step of $S_1$ must lead to monos $e'_i : F_i' \rightarrow G'$ and $e''_i : F_i'' \rightarrow G''$ such that $\ell \circ e'_i = e_i \circ \ell_i$ and $r \circ e''_i = e_i' \circ r_i$.

We now define that discrete steps of any small-scale PTGTS $S_1$ (from a state related in the behavioral coverage relation $S_{BCR}$) on its core (but not discrete steps on the border) can be mimicked by...
the corresponding \((S_{\text{BCR}} \text{-related})\) large-scale PTGTS \(S_0\). Again, refer to Figure 16 for a diagram representing how a step of \(S_1\) is correspondingly mimicked by \(S_0\) while the embeddings are transferred resulting again in \(S_{\text{BCR}}\)-related states. In the following definition, we ensure that the fragment states of all other small-scale PTGTSs are not modified.

Definition 20 (Small-to-large Constraint). If

\(\bullet\) \(\{(G, v), \kappa : \overline{G} \rightarrow G, w\} \in S_{\text{BCR}},\)
\(\bullet\) \(\{(F_i, v_i), (\overline{F_i}, \phi_i), \alpha_i : \overline{F_i} \rightarrow \overline{G}, \kappa_i : \overline{F_i} \rightarrow F_i, e_i : F_i \rightarrow G\} \in w,\)
\(\bullet\) \(S_i\) performs the discrete step from \((F_i, v_i)\) to \((F''_i, v''_i)\) using an underlying GT rule \(\rho_1 = (\ell_i : K_i \rightarrow L_i, r_i : K_i \rightarrow R_i, \phi_{\text{ac}, i}, \text{lab}_i)\) (given in Figure 16) where, since the discrete step preserves the fragment topology \(\overline{F_i}\), there are unique \(\kappa'_i : \overline{F_i} \rightarrow F_i'\) and \(\kappa''_i : \overline{F_i} \rightarrow F''_i\) such that \(\hat{\ell}_i \circ \kappa'_i = \kappa_i\) and \(\kappa''_i = \hat{r}_i \circ \kappa'_i\), and

\(\bullet\) \(e_i(m_i(L_i))\) does not overlap with any \(e_j(F_j)\) for \(i \neq j\), then

\(\bullet\) there is some tuple \(\{(G'', v''), \kappa'' : \overline{G} \rightarrow G'', w''\} \in S_{\text{BCR}}\) for some \(G'', v''\), \(\kappa''\), and \(w''\) as follows:

\(\circ\) There must be a discrete step of \(S_0\) from \(G\) to \(G''\) as given in Figure 16 for some underlying GT rule \(\rho = (\ell : K \rightarrow L, r : K \rightarrow R, \gamma, \phi_{\text{ac}, \gamma}, \text{lab})\) with the same probability and priority as \(\rho_1\).

\(\circ\) Since the discrete step of \(S_0\) preserves the large-scale topology \(\overline{G}\), there are unique \(\kappa' : \overline{G} \rightarrow G'\) and \(\kappa'' : \overline{G} \rightarrow G''\) such that \(\hat{\ell} \circ \kappa' = \kappa\) and \(\kappa'' = \hat{r} \circ \kappa'\).

\(\circ\) Moreover, the discrete step of \(S_0\) must lead to monos \(e'_i : F'_i \rightarrow G'\) and \(e''_i : F''_i \rightarrow G''\) such that \(\hat{\ell} \circ e'_i = e_i \circ \hat{\ell}_i\) and \(\hat{r} \circ e''_i = e''_i \circ \hat{r}_i\).

\(\circ\) Finally, \(w''\) is obtained from \(w\) by adapting the tuple \(((F_i, v_i), (\overline{F_i}, \phi_i), \alpha_i : \overline{F_i} \rightarrow \overline{G}, \kappa_i : \overline{F_i} \rightarrow F_i, e_i : F_i \rightarrow G)\) \(\in w\) to the resulting tuple \(((F'_i, v'_i), (\overline{F'_i}, \phi_i), \alpha_i : \overline{F'_i} \rightarrow \overline{G}, \kappa'_i : \overline{F'_i} \rightarrow F'_i, e'_i : F'_i \rightarrow G')\) \(\in w''\).

Intuitively, the last two constraints require that a behavioral coverage relation contains for each PTGTS \(S_i\) a simulation for the steps performed by \(S_0\) and a bisimulation for the steps performed by \(S_i\). However, note that the behavioral coverage relation does not require that steps performed on an overlapping of two fragment topologies can be mimicked by \(S_0\). This means that the step behavior of \(S_0\) is overapproximated by the PTGTSs \(S_i\) in the sense that these PTGTSs perform steps that are not happening in the PTGTS \(S_0\). For our running example, the step behavior of \(S_0\) is overapproximated in the sense of additional steps in which shuttles enter a fragment topology (by driving fast or slow or by stopping) and in the sense of steps in which shuttles exit a fragment topology (by driving or stopping), while these steps are not possible in the PTGTS \(S_0\). For a given large-scale state of \(S_0\), such entering and exiting steps of one of the PTGTSs \(S_i\) may not be mimicked by the PTGTS \(S_0\), since (i) there may not be a shuttle \(S_1\) in the large-scale state that can currently enter a fragment topology or (ii) there can be a shuttle \(S_2\) that exits a fragment topology using a PTGT rule for stopping while there is currently no other shuttle \(S_1\) two tracks ahead outside the fragment topology in the large-scale state. Intuitively, the additional rules only used for the PTGTSs \(S_i\) are used for steps where some dynamic elements (shuttles in the running example) are assumed to be present just outside the fragment topology; however, these assumed elements may not actually be present in the large-scale state, which prevents such a step to be mimickable by \(S_0\).

Based on the introduced behavioral coverage relation, we now derive results on the simulation of paths to show how, e.g., state reachability (in terms of AP labeled states) can be translated between the large-scale system and the corresponding small-scale systems.

As a first step, we state that behavioral coverage relations allow for the simulation of each path of the PTGTS \(S_0\) by the PTGTSs \(S_i\).
Lemma 1 (Existence of Simulating Paths). If $S_{BCR}$ is a behavioral coverage relation between $S_0$ and $(S_1, \ldots, S_n)$ and $\pi$ is a path of length $m$ in the PTGTS $S_0$ from the initial large-scale state $s_0$ to a large-scale state $s_m$, then, for each $1 \leq i \leq n$, there is a path $\pi_i$ in the PTGTS $S_i$ (of length $k_i \leq m$) ending in a fragment state $s_{i,k_i}$ such that $(s_{m}, \kappa, w) \in S_{BCR}$ for some embedding $\kappa : \overline{G} \rightarrow G$ and list of coverages $w$ where the $i$th element of $w$ is of the form $(s_{i,k_i}, (\overline{F}_i, \phi_i), \alpha_i : \overline{F}_i \rightarrow \overline{G}, \kappa_i : \overline{F}_i \rightarrow F_i, \epsilon_i : F_i \rightarrow G)$. Moreover, the probability of each such path $\pi_i$ is at least as high as the probability of the path $\pi$.

Proof (By Induction on $m$). Any path $\pi$ of length $m$ is simulated stepwise by each PTGTS $S_i$ using a possibly shorter path $\pi_i$ of length $k_i$ shorter than $m$. For the induction base, we note that, due to Definition 16, the initial states of the involved PTGTSs are related. For the induction step, we consider the different steps possibly according to the PTGTS step relation. Firstly, due to Definition 18, for each timed step of the PTGTS $S_0$ there is a corresponding timed step of each PTGTS $S_i$ to be appended to the path constructed for $S_i$ such that the reached states are again in $S_{BCR}$. Secondly, due to Definition 19, for each discrete step of $S_0$ with priority $\text{prio}$ there is a corresponding discrete step of each affected $S_i$ such that the reached states are again in $S_{BCR}$. Thirdly, due to Definition 20, if there would be an additional discrete step enabled in some $S_i$ with a priority $\text{prio}_i > \text{prio}$, then there would also be a discrete step with the same priority $\text{prio}_i$ in $S_0$, which would be a contradiction to the existence of the discrete step with priority $\text{prio}$ in the PTGTS $S_0$ from before. Hence, every timed and discrete step of the PTGTS $S_0$ can be mimicked by each PTGTS $S_i$ resulting again in states related by the behavioral coverage relation $S_{BCR}$ (where some discrete steps may not be required to be mimicked, as the large-scale state not managed by the PTGTS $S_i$ is modified).

Moreover, since $S_i$ applies PTGT rules with the same probability as $S_0$ (but not all discrete steps of $S_0$ are simulated by $S_i$ according to Definition 19), $\pi_i$ has at least the probability of $\pi$. 

Based on the existence of such simulating paths, we state that a PTGTS $S_0$ satisfies a safety property given by an AP, when safety w.r.t. this AP can be established for each PTGTS $S_i$. Safety means here that no path contains a state that is labeled with that AP.

Theorem 1 (Safety Verification). If $S_{BCR}$ is a behavioral coverage relation between $S_0$ and $(S_1, \ldots, S_n)$ w.r.t. the set of PTGT APs $\mathcal{A}$ and $ap \in \mathcal{A}$, then the PTGTS $S_0$ is safe w.r.t. the occurrence of an $ap$-labeled state when the PTGTS $S_i$ (for each $1 \leq i \leq n$) is safe w.r.t. the occurrence of an $ap$-labeled state. Moreover, the probability for observing an $ap$-labeled large-scale state in a path $\pi$ starting in some large-scale state $s$ of $S_0$ is smaller than the probability for observing an $ap$-labeled fragment state in a path $\pi_i$ starting in some $S_{BCR}$-related fragment state $s_i$ of $S_i$.

Proof (By Contraposition). Assume that the PTGTS $S_0$ is not safe by having a reachable large-scale state labeled with $ap \in \mathcal{A}$. Let $\pi$ be a path of length $m$ in $S_0$ from the initial large-scale state $s_0$ to such an $ap$-labeled large-scale state $s_m$. By applying Lemma 1, we are able to construct for each $1 \leq i \leq n$ a path $\pi_i$ of the PTGTS $S_i$ (of length $k_i \leq m$) ending in a fragment state $s_{i,m}$ such that at least one of these fragment states $s_{i,m}$ is also labeled with $ap$ due to Definition 17. However, the existence of such a path $\pi_i$ then contradicts the safety of $S_i$. The upper bound on the probabilistic occurrence of an $ap$-labeled state is then a direct consequence of Lemma 1 and the existence of such a path $\pi_i$ of the PTGTS $S_i$. 

We now apply the proposed methodology for establishing a behavioral relationship between the PTGTS $S_0$ and the PTGTSs $S_i$ to our running example. For this purpose, we describe how the
fragment state of each $S_i$ is embedded into the large-scale state of $S_0$ and, based on this embedding, how the PTGTs $S_i$ are derived from the PTGTS $S_0$.

**Example 2 (Construction of Embeddings and Simulating PTGTs for Running Example).** Additionally to the embeddings $\alpha_i$ and $\kappa$ already constructed for an arbitrary large-scale state of the PTGTS $S_0$, we obtain the fragment state $F_i$ (together with the embeddings $e_i$ and $\kappa_i$) from the fragment topology $\overline{F}_i$ by adding (a) all shuttles (with their attributes) that are connected to tracks contained in the fragment topology, (b) all active attributes of yellow and green traffic lights contained in the fragment topology, and (c) all connections between shuttles that have been added in item (a). This construction of fragment states $F_i$ also naturally applies to the initial large-scale state of $S_0$. Two embeddings $e_i$ and $e_j$ (for $i \neq j$) overlap in graph elements of their fragment topologies according to the overlapping specification (allowing for the overlapping on the chain of three entering/exiting tracks, as explained before). Moreover, $e_i$ and $e_j$ may also overlap in shuttles, their attributes, and their connections to tracks but not in further dynamic elements. To this end, we use, e.g., the AP $\text{AP}_{\text{disconnected-on-border}}$ from Figure 32 to verify that connections between shuttles are removed early enough to guarantee that connected shuttles are never on one of the tracks where two such embeddings may overlap.

Moreover, we adapt the given PTGTS $S_0$ to obtain for each of the eight fragment topologies one PTGTS $S_i$ by (a) changing the initial graph to the source graph of $e_i$ capturing the fragment topology as well as the additional dynamic elements of the initial large-scale state of $S_0$ connected to it and (b) adding eight PTGT rules for overapproximating the step behavior of $S_0$ on the tracks that may overlap with tracks of other fragment topologies.

For the latter point (b), we observe that all but two of the PTGT rules of $S_0$ are never applicable on the parts of fragment topologies that may overlap with other fragment topologies (called borders of fragment topologies). The remaining two PTGT rules that are applicable on the borders are the PTGT rule $\text{drive}$ from Figure 33| p. 61 and the PTGT rule $\text{stop1}$ from Figure 34| p. 62 (for which we proceed analogously to the PTGTS rule $\text{drive}$). For the PTGT rule $\text{drive}$, to allow shuttles to enter fragment topologies, we add the four PTGT rules $\text{border-enter-drive-fast}$ from Figure 54| p. 72, $\text{border-enter-drive-slow}$ from Figure 55| p. 72, $\text{border-exit-drive1}$ from Figure 58| p. 74, and $\text{border-exit-drive2}$ from Figure 59| p. 74. Similarly, for the PTGT rule $\text{stop1}$, to allow shuttles to stop on an exiting track and to exit the fragment topology by stopping, we add the four PTGT rules $\text{border-enter-stop-fast}$ from Figure 56| p. 73, $\text{border-enter-stop-slow}$ from Figure 57| p. 73, $\text{border-exit-stop1}$ from Figure 60| p. 74, and $\text{border-exit-stop2}$ from Figure 61| p. 75.

The additional PTGT rule $\text{border-enter-drive-fast}$ is used to simulate $\text{drive}$ steps where a shuttle in $S_0$ drives from a track not covered by $S_i$ to a track covered by $S_i$. This PTGT rule is constructed from the PTGT rule $\text{drive}$ by (i) omitting the source track $T_j$ from the PTGT rule $\text{drive}$, as this track is not covered by the PTGTS $S_i$, (ii) adding a fast shuttle (adding a slow shuttle results in the PTGT rule $\text{border-enter-drive-slow}$), and (iii) omitting application conditions that refer to graph elements outside the fragment topology and fragment states. Here, in step (ii), we rely on the constraints on the eight fragment topologies (cf. Example 1) expressing that all shuttles (including shuttles entering a fragment topology) drive fast or slow.

Similarly, the additional PTGT rules $\text{border-exit-drive1}$ and $\text{border-exit-drive2}$ are used to simulate the two $\text{drive}$ steps in which a shuttle in $S_0$ drives to an exiting track of the fragment topology and then exits the fragment topology to a track not covered by $S_i$. These two PTGT rules are also constructed from the PTGT rule $\text{drive}$ by similarly, (a) omitting the track $T_j$ for $\text{border-exit-drive1}$ and the tracks $T_k$ and $T_l$ for $\text{border-exit-drive2}$ from the PTGT rule $\text{drive}$ as these tracks are not covered by the PTGTS $S_i$, (b) removing the shuttle with its attributes for the PTGT rule...
Note that these additional PTGT rules overapproximate the step behavior that is possible in $S_0$, as they may be applied when analyzing $S_i$ also when (i) no corresponding shuttle in $S_0$ is able to enter a fragment topology or (ii) the PTGT rule \textit{drive} is disabled in the PTGTS $S_0$ because of shuttles too close ahead. In the latter case, the additional PTGT rules \textit{border-exit-drive1} and \textit{border-exit-drive2} are not disabled in the PTGTS $S_i$, since the application conditions matching the too closely located shuttles are omitted in these two PTGT rules.

While highly desirable, the information given by the set of fragment topologies and the overlapping specification is insufficient to automatically synthesize the additional PTGT rules for the borders of fragment systems. Firstly, attributes need to be instantiated with values (e.g., in our running example, the \textit{minDur} attribute is equipped with one of two possible values in \textit{border-enter-drive-fast}, \textit{border-enter-drive-slow}, \textit{border-enter-stop-fast}, and \textit{border-enter-stop-slow}) but potential values cannot be derived automatically and must therefore be provided. Secondly, dynamic elements preserved, created, or deleted by underlying GT rules of a given PTGT rule may be deleted (e.g., the driving shuttle from \textit{drive} is deleted instead of preserved in \textit{border-exit-drive1}) or created (e.g., the driving shuttle from \textit{drive} is created instead of preserved in \textit{border-enter-drive-slow}) in the additional PTGT rules, since a dynamic element may enter or exit the scope managed by a fragment system and the definition of which dynamic elements such as shuttles are managed by a fragment system must therefore be provided. Lastly, the adaptation of PTGT rules (by removing static elements from underlying GT rules that may not be part of the fragment topology) must also be extended to the application conditions. Based on this discussion, we expect that a fully or partially automatic generation of correct-by-construction additional PTGT rules requires at least some further information on potential attribute values on borders and of how dynamic elements of the large-scale system are split up amongst the fragment systems. Nevertheless, without automatic support for the generation of these additional PTGT rules, careful design of the decomposition as in our running example can greatly simplify the task of generating such additional PTGT rules, since it does not depend on potentially complex behavior on the core of fragment topologies.

For our running example, we now describe the construction of a suitable behavioral coverage relation relying on the coverage of the large-scale topology discussed in Example 1.

**Lemma 2 (Existence of Behavioral Coverage Relation for Running Example).** For the PTGTS $S_0$ of our running example with an arbitrary initial large-scale topology such that $M$ is a coverage of that large-scale topology w.r.t. some embedding $\kappa : G \leftarrow G$, the set of eight constrained fragment topologies, and the overlapping specification $o$ from Example 1, there is a behavioral coverage relation $S_{BCR}$ between $S_0$ and the $n$ PTGTSs $S_i$ constructed in Example 2.

**Proof.** We argue that a suitable behavioral coverage relation $S_{BCR}$ can be constructed for our running example by (base) starting with a candidate for $S_{BCR}$ first containing only a tuple to satisfy the constraint from Definition 16 about initial states and (closure) then showing that arbitrary timed and discrete steps can be mimicked appropriately satisfying the four further constraints of behavioral coverage relations where these mimicking steps then extend the candidate for $S_{BCR}$ in each case. These two construction steps thereby result in the smallest possible behavioral coverage relation $S_{BCR}$ for the running example at hand. We note that all tuples that are added in these two construction steps (a) use the embeddings described in Example 2 and (b) contain consistent clock valuations among the PTGTSs $S_i$.

Satisfaction of the labeling constraint from Definition 17: $S_{BCR}$ satisfies this constraint, because all APs of $S_0$ only match a single track and, hence, such a match can be obtained analogously.
Compositional Analysis of Probabilistic Timed Graph Transformation Systems

in every $S_i$. That is, the PTGT APs employed only match a suitably small part of the fragment states, ensuring that these matches cannot span beyond what is covered by single fragment state. In particular, the PTGT AP $AP_{collision}$ matches a single track with two shuttles on it; the matched part of a rigid part of a large-scale state satisfying this PTGT AP therefore only contains this single track, and this matched part is therefore always contained in any single fragment state of some PTGTS $S_i$. The same reasoning of a single track matched applied for the PTGTS AP $AP_{brake}$.

Satisfaction of the temporal constraint from Definition 18: $S_{BCR}$ satisfies this constraint because of the item (b) on consistent clock valuations from above and the fact that the same PTGT invariants are used in the PTGTS $S_0$ and the PTGTSs $S_i$. Moreover, these PTGT invariants can be matched only on the core of fragment states also ensuring that they do not span across the borders of a fragment state or, as for the PTGT invariant $INV_{driving}$, again match only a single track as for the PTGT APs from the previous paragraph.

Satisfaction of the large-to-small constraint from Definition 19: $S_{BCR}$ satisfies this constraint on the simulation of discrete steps of $S_0$ by the affected $S_i$, since (i) all steps on the non-overlapped parts of the fragment topology (i.e., the core) can be performed in $S_i$ using the same PTGT rules and matches, (ii) each further step of $S_0$ matching tracks that are jointly covered by $S_i$ and $S_j$ (i.e., on the border) can be mimicked by $S_i$ and $S_j$ using the additional PTGT rules (along the detailed discussion of these additional PTGT rules in Example 2), and (iii) the additional PTGT rules have all the priority 0 not preventing the also required PTGT rule applications on the core from the previous item (ii).

Satisfaction of the small-to-large constraint from Definition 20: $S_{BCR}$ satisfies the constraint on the simulation of discrete steps of $S_i$ on its core, because $S_i$ contains the same PTGT rules as $S_0$ (except for fragment systems, where these omitted rules are inapplicable, because the rigid fragment topology used for that fragment system does not contain certain nodes/edges such as traffic light nodes or conflict nodes to be matched by the omitted PTGT rules).

Based on this behavioral coverage relation and Theorem 1, we obtain the desired overapproximation results for the qualitative safety w.r.t. collisions and the quantitative unlikeliness of emergency brakes for our running example. Note that analysis results for the remaining four properties introduced in Section 3 do not directly carry over from the PTGTS $S_i$ to the PTGTS $S_0$, since the PTGTS $S_i$ overapproximates the step behavior of the PTGTS $S_0$, thereby possibly not violating properties that are in fact violated by the PTGTS $S_0$. However, to analyze $S_0$ w.r.t. these properties, we envision to extend our decomposition-based approach to exploit the fact that there is a bisimulation between the PTGTS $S_0$ and the steps of the PTGTSs $S_i$ on their cores, implying that undetected violations in $S_i$ can only occur due to the overapproximation on its borders.

**Theorem 2 (Qualitative and Quantitative Safety for Running Example).** The PTGTS $S_0$ exhibits no collisions when this is the case for each PTGTS $S_i$. Moreover, an emergency brake is performed in $S_0$ with a probability not higher than the probability of an emergency brake in any $S_i$.

**Proof.** By direct application of Theorem 1 and Lemma 2, since the PTGT APs $AP_{collision}$ and $AP_{brake}$ for checking for collisions and for checking for emergency brakes are contained in the set $\mathcal{A}$ for our running example and also used in the PTGTS $S_0$ as well as in the PTGTSs $S_i$. Hence, any violation in one of the PTGTSs $S_i$ will be translated to the PTGTS $S_0$ via Theorem 1 on the basis of the behavioral coverage relation $S_{BCR}$ established in Lemma 2 for our running example.

For our running example, we assume that the initial large-scale state is given by the large-scale topology containing (a) static elements (as discussed before), (b) no shuttles, but (c) attributes of yellow and green traffic lights. This assumption ensures that the initial fragment states for a
common fragment topology are identical. Hence, the PTGTSs obtained for these initial fragment states are identical as well and can be analyzed in a single analysis run. Consequently, any number \( n \) of embeddings of instances of the eight fragment topologies allows to perform only eight analysis runs on the resulting eight PTGTSs.

When this assumption on the initial large-scale state is not satisfied, e.g., because there is already a shuttle on the large-scale topology, we may end up with, in the worst case, \( m \) different initial fragment states to be considered in \( m \) model checking runs for a PTGTS when the fragment topology is embedded into the initial large-scale topology resulting in \( m \) different (initial) fragment states. However, even in this case, later model checking runs may be determined to be already covered when their initial (fragment) states occurred in any earlier model checking run. For example, when there are two embeddings of FT1 into the initial large-scale state: (e1) a single shuttle is at the track next to the depot and (e2) no shuttle is in the fragment state. In this case, the first model checking run for embedding (e1) will also reach the state given by embedding (e2), since the shuttle may just drive forward exiting the fragment topology. Consequently, the second model checking run then does not need to be executed.

In any case, analyzing the PTGTS behavior on the fragment topologies instead of the large-scale topology decreases the number of active components (given by shuttles in our running example) to be considered in a single model checking run, therefore also decreasing the size of the state space to a hopefully computable and analyzable size.

8 ANALYSIS OF BEHAVIOR ON FRAGMENT TOPOLOGIES

We now discuss how the behavior of fragment PTGTSs can be analyzed using property-specific analysis procedures. For this purpose, we consider again the six properties for our running example listed in Section 3, discuss their relevance, and present our analysis procedures for them. In the next section, we then extend these analysis procedures to large-scale PTGTSs and evaluate them.

For the analysis of properties, we rely on the zone-based symbolic state space generation for PTGTSs (see Section 5.3) implemented in our tool AUTOGRAPH. Recall that the states of this symbolic state space are given by pairs \((G, \psi)\) of a graph \(G\) and a clock constraint \(\psi\) representing as a zone a non-empty set of valuations of the clocks of \(G\) (where we employ DBMs to represent zones at the technical level). Also, recall that every structural path of steps through that symbolic state space is realizable in the induced PTS of the PTGTS used to generate this symbolic state space.

As an alternative to the generation of the symbolic state space based on zones, we could obtain an equivalent PTA for the given PTGTS and could employ the PRISM model checker to analyze that PTA. However, besides the problem of a potentially intractably large PTA that cannot be analyzed using PRISM, most properties discussed below cannot be analyzed using this approach as the analysis in PRISM always starts at the (unique) initial state of the PTA. In contrast, we need to (a) start analysis at different states of the state space and (b) monitor active components across multiple discrete steps.

To monitor active components across discrete steps of the PTGTS, we rely on labels that are generated for each PTGT step from the matched variables. For our running example, we monitor single shuttles driving across the tracks of the fragment topology to analyze their driving behavior. For example, consider the PTGT rule drive given in Figure 7(a) and one of its applications in Figure 5(b), where the label \((1, \text{DRIVE}, 3, 10, 11)\) indicates that the shuttle with ID 1 drives slowly from track 10 to track 11 (cf. Section 5.1 for our earlier explanations). Hence, when considering steps in the symbolic state space generated for our PTGTS, we can inspect whether a currently monitored shuttle (given by the ID of the track it is currently located on) is involved in some specific step.
Technically, we derive from the symbolic state space a transition relation (which is more abstract compared to the state space) to be able to monitor a single active component of interest across multiple steps where the monitored active component will only be involved in some of these steps (because in the other steps other active components or no active components perform steps). To achieve this, we obtain for each state $s$ in the symbolic state space a so-called label-state $\hat{s}$ that may contain one or several values. For a PTGT step from $s$ to $s'$, a label-state $\hat{s}$ of $s$ is adapted to a label-state $\hat{s}'$ for $s'$ based on the label of that PTGT step.

For our running example, we use label-states given by the ID of the track to which the currently monitored shuttle is connected to. For the PTGT step using the PTGT rule $\text{drive}$ with label $(1, \text{DRIVE}, 3, 10, 11)$, we would then either change $\hat{s} = 10$ into $\hat{s}' = 11$ (since we monitor the shuttle on track 10 and this shuttle moved in that PTGT step) or would obtain $\hat{s} = \hat{s}'$ (since we monitor a shuttle on another track, which did not move in that PTGT step).

Note that this monitoring approach using labels easily allows to determine steps that are of particular relevance later on such as PTGT steps for (a) new shuttles entering the fragment topology, (b) the monitored shuttle exiting the fragment topology, and (c) the monitored shuttle executing an emergency brake. Also note that the past behavior of shuttles can be monitored as well by considering PTGT steps in the symbolic state space in reverse direction and by generating the transition relation on label-states from above in reverse direction alongside. To generate this transition relation, we define how label-states are adapted in terms of how the label of a PTGT rule affects the label-state. In comparison, we introduced the Metric Temporal Graph Logic (MTGL) in \cite{20, 51, 52}, which allows to monitor graph elements across steps in a more general way and to state properties on paths where such monitoring is then successful when all of the monitored graph elements have been preserved in all considered steps and unsuccessful when some of the monitored graph elements have been deleted in some of the considered steps. However, for the running example, the presented monitoring using the labels of PTGT steps is sufficient and far less technically involved.

For our running example, we are interested in the analysis of the following six properties:

**Property 1: Timelock Detection.** The property P1 from Section 3 stating that “The PTGTS modeling the shuttle scenario never gets temporally stuck in a state where no steps are enabled” expresses a timelock detection problem.

This problem can be analyzed by checking all states $s$ of the generated symbolic state space as follows: If $s$ has no exiting (discrete) step and a non-trivial invariant has been assigned to $s$, then $s$ represents a state in timelock. For our running example, the non-trivial invariant $d_1 \leq 3$ is obtained when a fast shuttle in mode $\text{DRIVE}$ is at a track with clock $d_1$. This invariant prevents the progress of time once the clock $d_1$ has reached the value of 3 and is therefore non-trivial (the invariant $d_1 \geq 0$ would be, in contrast, trivial, as it would not prevent the progress of time). Essentially, such an invariant should force the execution of some discrete step instead of a further timed step. The timelock then expresses that such a discrete step is missing. In that sense, timelocks are always modeling errors and their detection is of great importance. For our running example, such a missing discrete step would mean that the shuttle would still advance to the next track in the modeled cyber-physical system, but this driving step would not be covered by the PTGTS modeling this system. This would lead to an inconsistency between the system state and the model state having as a consequence that analysis results for the model are then of no further value. For our running example, see Figure 17(a) for a state in timelock, which is unreachable for our PTGTS. In this state, the shuttle $S_1$ must advance to the next track (due to the PTGT invariant) but cannot do so, since the PTGT rule $\text{drive}$ is disabled due to the shuttle $S_2$ being on the next track.
Property 2: Qualitative Reachability Analysis. The property $P_2$ from Section 3 stating that “Shuttles never collide” expresses a qualitative reachability analysis problem.

This problem can be analyzed by defining a PTGT AP $\phi$, which specifies the desired property, and by then labeling each state $(G, \psi)$ in the symbolic state space with $\phi$ where the graph $G$ satisfies $\phi$. For our running example, we employ the PTGT AP $AP_{\text{collision}}$ from Figure 6(a) labeling all states where two shuttles are on a common track. Note that for such labeling, we could even employ PTGT APs where clock constraints are used, because the limitation of PTGT APs mentioned in Section 5.1 is only relevant when probabilistic reachability of such labeled states is then to be analyzed using, e.g., the Prism model checker for the PTA that could be derived from the PTGTS.

For our running example, we never observed collisions, which is ensured by the used PTGT rules (see Figure 17(b) for an example state with a collision, which is not reachable for our PTGTS). Collisions are avoided, since the PTGT rules advancing shuttles, such as drive, are equipped with application conditions ruling out forward steps of shuttles onto tracks on which a shuttle is already located. As pointed out before, the application condition of the PTGT rule drive thereby forces a shuttle to stay on its current track as long as possible but may result in a timelock when no PTGT

Fig. 17. Unreachable/undesirable fragment states for our running example (the used annotations are the same as in Figure 8).
rule is available to advance the shuttle when it would need to leave the track according to the PTGT invariant \( \text{INV}_{\text{driving}} \) from Figure 6(b).

**Property 3: Actor-based Deadlock Detection.** The property \( \text{P3} \) from Section 3 stating that “Shuttles are never prevented indefinitely from driving ahead” expresses an actor-based deadlock detection problem.

While a coordination mechanism outside the scope of the PTGTS could instruct shuttles to stay on a certain track in mode \( \text{STOP} \) or \( \text{BRAKE} \), we assume here that such shuttles should eventually be able to leave their current track, since the PTGT rules have been designed with that possibility in mind. Therefore, when a shuttle ends up in a situation, where it could never perform a step afterwards, we would consider our PTGTS to be faulty, since we only employ fragment topologies where a congestion (i.e., a traffic jam among the shuttles) is expected to be resolved eventually.

The deadlock detection problem can be analyzed using the monitoring of shuttles from above as follows: Firstly, we define a mapping \( R \), which maps pairs of the form \( ((G, \psi), \text{tid}) \) to a Boolean value where \( (G, \psi) \) is a state of the symbolic state space and \( \text{tid} \) is the ID of a track with a shuttle on it contained in \( G \) (i.e., a label-state as discussed above). Initially, the value, to which \( R \) maps to, is false for each pair of arguments, i.e., \( R(s, \text{tid}) = \perp \) (see item Init below). Secondly, we execute a fixed-point operation on that mapping \( R \), which terminates once \( R \) is not changed anymore. In each iteration of this fixed-point operation, we consider all states of the symbolic state space. Clearly, due to the modifications performed by this fixed-point operation, only states \( s \) must be considered in an iteration if the mapping \( R \) has been adjusted in the previous iteration for some state \( s' \) directly reachable from \( s \) thereby improving efficiency.

The fixed-point operation adapts \( R \) in two cases called Base and Propagation as follows:

- **Init:** \( R(s, \text{tid}) = \perp \) for every state \( s \) and every ID \( \text{tid} \) of a track on which a shuttle is located in state \( s \).
- **Base:** Condition: There is a step from the considered state \( s \) to some state \( s' \) where a shuttle advances from track \( \text{tid} \) to some other track. Modification: \( R(s, \text{tid}) \) is set to \( \top \).
- **Propagation:** Condition: There is a step from the considered state \( s \) to some state \( s'' \) where no shuttle advances from track \( \text{tid} \) to some other track. Modification: \( R(s, \text{tid}) \) is set to \( R(s'', \text{tid}) \).

Once a fixed point has been reached, each mapping \( R(s, \text{tid}) = \perp \) identifies a state \( s \) where a shuttle on the track with ID \( \text{tid} \) was unable to ever drive forward for any resolution of the non-determinism.

For our running example, the absence of deadlocks is ensured, since the reason for a shuttle \( S_1 \) to have stopped or braked is always another shuttle \( S_2 \). Such a shuttle \( S_2 \) is always ahead of the stopped/braked shuttle \( S_1 \) (or on the opposite conflicting track in mode \( \text{DRIVE} \)) and can eventually leave the fragment topology. Once the shuttle \( S_2 \) has exited the fragment topology or is far enough ahead of shuttle \( S_1 \), the shuttle \( S_1 \) can drive forward.

Incorrectly defined PTGT rules could lead to a deadlock, e.g., for FT9 when both shuttles brake/stop just ahead of the join track not allowing either shuttle to advance because the other shuttle could do the same (see Figure 17(c)). In that sense, the absence of deadlocks in this situation is the consequence of only the second shuttle reaching the track preceding the join track by braking. Incorrectly constructed fragment topologies could also lead to a deadlock, e.g., when such a fragment topology contains a circle of tracks and a depot from which fresh shuttles can enter that circle (see Figure 17(d)). For such a fragment topology, also our correct PTGTS, will eventually reach a state in which no shuttle can advance. The problem of this fragment topology is obvious.
and it would therefore never be employed in a large-scale system but in other PTGTSs modeling, e.g., scheduling algorithms where the presence of deadlocks is more complex to foresee.

**Property 4: Actor-based Livelock Detection.** The property \( P4 \) from Section 3 stating that “Shuttles are permanently able to eventually exit the fragment topology on which they are currently located” expresses an actor-based livelock detection problem.

For this problem, in addition to detecting all states \( s \) satisfying a certain condition (here, that a shuttle is located on some track) as for \( P2 \), we also need to determine whether a path exists that starts in \( s \) and ends in a state where the shuttle has (just) exited the fragment topology.

The livelock detection problem can be analyzed employing an adapted version of the fixed-point operation used for \( P3 \). In this adaptation, we keep INIT and PROPAGATION unchanged and modify only the BASE case.

- **INIT:** \( R(s, \, tid) = \bot \) for every state \( s \) and every ID \( tid \) of a track on which a shuttle is located in state \( s \).
- **BASE:** **Condition:** There is a step from the considered state \( s \) to some state \( s' \) where a shuttle exits the fragment topology from track \( tid \).
  **Modification:** \( R(s, \, tid) \) is set to \( \top \).
- **PROPAGATION:** **Condition:** There is a step from the considered state \( s \) to some state \( s'' \) where no shuttle advances from track \( tid \) to some other track.
  **Modification:** \( R(s, \, tid) \) is set to \( R(s'', \, tid) \).

Once a fixed point has been reached, each mapping \( R(s, \, tid) = \bot \) identifies a state \( s \) where a shuttle on the track with ID \( tid \) was unable to ever exit the fragment topology for any resolution of the non-determinism.

Again, for our running example, an incorrectly constructed fragment topology could allow shuttles to enter a part of a track topology from which no exit is ever reachable. See Figure 17(d) for an example of such an ill-formed fragment topology where the shuttle \( S_1 \) is essentially trapped in the circle even though it is currently not in a deadlock.

**Property 5: Actor-based Fairness Analysis.** The property \( P5 \) from Section 3 stating that “Shuttles will eventually exit the fragment topology on which they are currently located” expresses a actor-based fairness analysis problem.

The difference to \( P4 \) is that we now require that a shuttle is not only permanently able to exit the fragment topology but that it will also eventually do so. This problem amounts to detecting whether there is a cycle in the symbolic state space with a shuttle \( S_1 \) on a certain track where \( S_1 \) never advances in the steps of that cycle. If this is the case, then there is an infinite path (repeating the cycle) on which the shuttle never exits the fragment topology.

The fairness analysis problem can be analyzed employing an adapted version of the fixed-point operation used for \( P3 \) and \( P4 \). In this adaptation, we keep INIT and BASE as for \( P4 \) and modify only the PROPAGATION case. Note that for \( P5 \) the fixed-point operation additionally differs from those used for \( P3 \) and \( P4 \) by checking in the PROPAGATION case not a single state but all steps exiting a common state at once.

- **INIT:** \( R(s, \, tid) = \bot \) for every state \( s \) and every ID \( tid \) of a track on which a shuttle is located in state \( s \).
- **BASE:** **Condition:** There is a step from the considered state \( s \) to some state \( s' \) where a shuttle exits the fragment topology from track \( tid \).
  **Modification:** \( R(s, \, tid) \) is set to \( \top \).
• **PROPAGATION:** Condition: The considered state $s$ has the set $S'$ of successor states where each step from $s$ to any successor state $s'' \in S'$ does not advance the shuttle from track $tid$ to some other track.

Modification: $R(s, tid)$ is set to $\land_{s'' \in S'} R(s'', tid)$.

Once a fixed point has been reached, each mapping $R(s, tid) = \perp$ identifies a state $s$ where a shuttle on the track with ID $tid$ was not able to exit the fragment topology for every resolution of the non-determinism.

When analyzing this property for our running example, we determine that this property is not satisfied for FT7 and FT9, since it is possible that a shuttle stops for some reason while other shuttles are able to enter and exit the fragment topology. See Figure 17(e) for an example of such a fragment state where the shuttle $S_1$ is able to never drive forward while being in a cycle of the symbolic state space in which it could at least sometimes advance. For this fragment state, such a cycle can contain the steps of the shuttle $S_2$ driving through the fragment topology, exiting the fragment topology, another shuttle entering the fragment topology and then reaching the given fragment state again.

**Property 6: Quantitative Reachability Analysis.** The property $P6$ from Section 3 stating that "Occurrences of emergency brakes are improbable at a local level for a single shuttle but also improbable at the global level for the entire large-scale system and its possibly numerous shuttles" expresses a quantitative reachability analysis problem.

This problem can be analyzed by adapting the backward analysis algorithm employed by, e.g., the PRISM model checker to determine the worst-case probability (across all possible adversaries) to reach a state labeled with a certain AP (see [29, 31, 39] for more technical details of the subsequently adapted analysis approach). Intuitively, we extract the relevant part of the symbolic state space by determining a set of target states where shuttles just performed an emergency brake and by monitoring these shuttles then backwards across steps until they entered the track topology. All these states where the monitored shuttles just entered the topology can then be understood to be initial states for analysis (note that there may be many states where a shuttle executed an emergency brake and from each of these states we may find also many states backwards where such a shuttle entered the fragment topology based on different interleavings with steps of other shuttles). However, PRISM cannot handle multiple initial states. Therefore, we discuss now shortly the combination of our monitoring approach from above (used to determine the states/steps relevant for the analysis) and the backward analysis algorithm performed by PRISM on the symbolic state space obtained for a given PTA.

We may identify all those states $s$ where the graph part satisfies the PTGT AP $AP_{brake}$ (see Figure 30| p. 59) by containing a shuttle in mode $BRAKE$. However, we would then also derive states $s$ where the shuttle did not brake in the most recent step leading to state $s$. Therefore, we alternatively identify the set $S$ of pairs $(s, tid)$ of all states $s$ from which a step using the PTGT rule $emergency-brake$ or the PTGT rule $construction-site-brake$ exists in the symbolic state space and in which a shuttle advances from the track with ID $tid$ to some other track. We then define a mapping $R$, which maps pairs of the form $((G, \psi), tid)$ to a probability between 0 and 1 where $(G, \psi)$ is a state of the symbolic state space and $tid$ is the ID of a track with a shuttle on it contained in $G$ (i.e., a label-state as discussed above). Initially, the probability, to which $R$ maps to, is 1 for each pair contained in $S$ and 0 for each other pair of a state and a track ID. We then apply a fixed-point operation adapting $R$ and an initially empty set $I$ of argument pairs $(s, tid)$ of $R$ until a fixed-point is reached in which $R$ and $I$ are not changed anymore. Hereby, the set $I$ records all those pairs $(s, tid)$ where a shuttle, which was identified to have braked later on, entered the fragment topology onto the track with ID $tid$. 
The fixed-point operation adapts $R$ and $I$ as follows:

- **INIT**: $R(s, \text{tid}) = 1$ for every pair $(s, \text{tid}) \in S$ and 0 for every other pair.
- **ADVANCE-STEP**: Condition: There is a step from the considered state $s$ to some state $s'$ where a shuttle advances from track $\text{tid}$ to track $\text{tid}'$ and where $(s, \text{tid}) \notin I$. 
  Modification: $R(s, \text{tid})$ is set to $\max(R(s, \text{tid}), R(s', \text{tid}'))$ and $R(s, \text{tid}'')$ is set to $\max(R(s, \text{tid}''), R(s', \text{tid}''))$ for every $\text{tid}'' \neq \text{tid}$.
- **ENTER-STEP**: Condition: There is a step from the considered state $s$ to some state $s'$ where a shuttle enters the topology onto track $\text{tid}$.
  Modification: $(s', \text{tid})$ is added to $I$.
- **OTHER-STEP**: Condition: There is a step from the considered state $s$ with DPD $\mu$ describing the target states $S'$ of that step where no shuttle advances or enters the topology.
  Modification: $R(s, \text{tid})$ is set to $\max(R(s, \text{tid}), \sum_{s' \in S} \mu(s') \times R(s', \text{tid}))$ for every $\text{tid}$ for which $(s, \text{tid}) \notin I$.

Once a fixed point has been reached, we determine $\max_{(s, \text{tid}) \in I} R(s, \text{tid})$ as the worst-case probability with which a shuttle entering the fragment topology can eventually execute an emergency brake.

Note that the analysis algorithm, which we obtained as an adaptation of the backward analysis algorithm employed by PRISM, ensures, e.g., for FT4 and FT5, that probabilities occurring in paths due to probabilistic steps of other shuttles do not impact the ultimately obtained probability for the monitored shuttle. To achieve this, paths resulting from alternative outcomes of probabilistic steps of other shuttles are merged suitably. For example, in Figure 18, we provide a simplified part of a symbolic state space where in (action $a_1$) a fast shuttle $S_1$ enters the fragment topology, (action $a_2$) a shuttle $S_2$ passes by a yellow traffic light, (actions $a_3$ and $a_4$) the shuttle $S_2$ stops due to a shuttle ahead, (action $a_5$) the shuttle $S_1$ passes by a yellow traffic light, and (action $a_6$) the shuttle $S_1$ executes an emergency brake. For this example, in which the steps of shuttle $S_2$ should be insignificant to the probability for $S_1$ executing an emergency brake, we expect a worst-case probability of 0.75 to be obtained for $S_1$ executing the emergency brake (according to the probabilities used in Figure 18), since the step (action $a_2$) had no impact on shuttle $S_1$. Technically, we (a) obtain the probability of 0.75 for $s_4$ as the result of $0.75 \times 1 + 0.25 \times 0$, where 1 is the initial probability assigned to state $s_5$ and 0 is the initial probability assigned to state $s_6$, (b) propagate that probability to $s_2$ and $s_5$, and (c) obtain again the probability of 0.75 for $s_1$ as the result of $0.6 \times 0.75 + 0.4 \times 0.75$.

Based on Theorem 2, we can conclude that a worst-case probability obtained using the described analysis procedure is an upper bound for the probability of an emergency brake of the same shuttle in the large-scale system. However, the number of expected emergency brakes executed in the large-scale system by any shuttle during, e.g., a single day thereby depends directly on the number of shuttles passing by joins and construction sites during that time period.

In the next section, we present the analysis results obtained for the six discussed properties and evaluate the applicability of our decomposition-based analysis approach.

---

We omit several further steps that are not of relevance here and use different probabilities to disambiguate the presentation.
9 EVALUATION

In this section, we evaluate the use of PTGTSs to model large-scale cyber-physical systems as well as the suitability of our decomposition-based analysis approach for such PTGTSs. To this end, we firstly discuss our running example as an extension of the example from [37] in terms of supporting the additional FT9 and its modeling using a PTGTS with additional PTGT rules. Secondly, we review the improved analysis procedures as introduced in the previous section and discuss relevant assumptions for our running example. Lastly, we apply the analysis procedures to the running example presenting the individual steps of analysis and the obtained analysis results including a discussion of efficiency regarding memory and time consumption. For this last step, we employ our novel implementation in AUTOGRAPH of the DBM-based state space generation technique introduced in Section 5.3, avoiding the generation and integration of unreachable states and steps and the analysis procedures based on the monitoring technique of active components presented in Section 8. The tool AUTOGRAPH does not rely, besides Z3 for the management of ACs for the handling of attributes, on further tools or noteworthy libraries for the use case presented in this article. Instead, the presented steps of our analysis procedure have been integrated directly as an extension of AUTOGRAPH. However, as discussed in detail in Section 5.3, e.g., the implementation in AUTOGRAPH of our earlier work in [37, 38] depends on PRISM and UPPAAL for the analysis of generated PTA and TA to perform probabilistic analysis and to detect states and steps prohibited by the temporal clock-based constraints employed in the PTGTS when only generating the structural state space.

Modeling of Running Example as PTGTS. In our running example, we included the fragment topology FT9 from [36] for two shuttles attempting to connect when approaching a join track. In comparison, one of the two shuttles would simply stop when approaching a join track in the fragment topology FT8, which we used in our earlier work in [37].

However, when applying our current decomposition-based approach to large-scale PTGTSs featuring instances of the fragment topology FT9, we had to modify and add several PTGT rules to ensure that shuttles must perform two failing connection attempts (using the PTGT rule connect in Figure 9(a)) before executing an emergency brake (using the PTGT rule emergency-brake in Figure 9(b)). In [36], we were only able to analyze the fragment PTGTS for FT9 for a small number of fixed initial PTGT states and had to perform analysis runs for each such initial PTGT state all over again. In particular, we were only able to consider initial PTGT states in which the clocks of the tracks on which the shuttles are located are both 0 ruling out other clock values to begin with. Using the current decomposition-based approach, the behavior of the PTGTS is analyzed more appropriately, since shuttles enter the critical area close to a join track (identified by conflicts connected to tracks) from many different PTGT states (given by the placement of shuttles on tracks and the current clock values of such tracks).

When applying the current decomposition-based approach, we realized that when shuttles enter the fragment topology FT9 at different time points (using two different entry tracks) with varying speeds (slow or fast, which was also not considered in [36]), timelocks could occur. For example, once a connection was established where a slow shuttle was the convoy follower and a fast shuttle was the convoy leader, it became possible for the fast shuttle to establish a distance to its follower where the PTGT rule disconnect1 (see Figure 49| p. 70) would not be applicable, and we therefore had to include another PTGT rule disconnect2 (see Figure 50| p. 70) for shuttles with a larger distance in between when disconnecting. Also, because of the varying speeds among shuttles and slightly skewed positions when entering the critical area close to a join track, it was possible that one shuttle was just far enough ahead of the other shuttle such that the two shuttles did not manage to communicate at the first and second conflicting tracks. As a consequence, the shuttle that
was a little bit behind was able to close the gap to the other shuttle and had to execute an emergency brake just before reaching the join track. To rule out this situation, we introduced the PTGT rule stop2 (see Figure 47 p. 68), where a shuttle already stops when spotting another shuttle on the opposite side at the next conflicting track. Also, we had to include two additional GCs into the application condition of the PTGT rule drive (see Figure 33 p. 61) to ensure that the shuttle driving behind would (a) stop using the PTGT rule stop2 when possible and (b) not restart driving once it stopped using the PTGT rule stop2 in an earlier step while the other shuttle is still in range.

Note that further velocities to be used by shuttles or higher number of conflicting tracks could require additional PTGT rules to accommodate for longer distances between connected shuttles (to avoid livelocks when such PTGT rules are missing). Our prototypical implementation of the analysis procedures described in the previous section in the tool AUTOGRAPH was crucial to detect previously unknown and unexpected problems in the PTGT rules employed for our running example. We used extension of the tool AUTOGRAPH (a) to detect timelocks in the symbolic state space and to generate example paths leading to such timelocks as well as (b) to detect cases where a shuttle executed an emergency brake with an unexpected probability of, e.g., 1 (this probability of 1 indicates that the shuttle had to execute an emergency brake without ever performing a connection attempt beforehand) and to generate the paths leading to such an emergency brake. Considering such undesirable paths was helpful to determine the underlying problems and to devise adaptations of the affected PTGT rules to obtain the PTGT rules in the form presented here.

**Improved Analysis Capabilities.** Compared to [37], we have presented additional analysis support for the PTGTS-independent properties of timelock freedom (P1) and actor-based deadlock freedom (P3) as well as the PTGTS-specific properties of actor-based livelock freedom (such as P4) and actor-based fairness (such as P5). Moreover, quantitative reachability analysis support (as for P6) for the calculation of the worst-case probability to reach a certain state/step is improved, since, due to the usage of symbolic state space, each path that contributes to a worst-case probability is indeed realizable. Hence, there is no overapproximation of this worst-case probability (for FT4, FT5, and FT9) beyond the fact that not every fragment state will occur in the actual large-scale system. Recall that certain fragment states can be expected not to occur in the large-scale system, because shuttles may enter the fragment topology even when they do not exist in the actual large-scale state. Note that, in our evaluation, we consider only at most two shuttles by using a System node for each fragment topology, which stores in its counter attribute the number of shuttles that may enter that fragment topology; the value of this counter attribute is checked and increased/decreased whenever a shuttle enters/exits the fragment topology using one of the PTGT rules for the border given in Table 2 p. 60. In contrast to this limitation to two shuttles, we refer to [37] for our previous benchmark evaluation, where we allowed shuttles to enter the fragment topology at all times, also permitting shuttles to be on every track of the fragment topology. The restriction to at most two shuttles is realistic for our running example when there is a global control system ensuring that shuttles keep a certain distance from each other at a global level (guaranteeing that not more than two shuttles enter a common fragment topology) while the shuttles use communication to locally coordinate their driving behavior (as for FT9). For FT9, this assumption is of especial importance, since the PTGT rules designed for that fragment topology would need to be considerably more complex when more than two shuttles could enter that fragment topology, also resulting in an intractable state space for FT9 due to its 16 contained tracks and the resulting state space explosion.

**Analysis Results for Running Example.** The fragment PTGTSs constructed for the fragment topologies from Figure 2(b) employ the PTGT APs, PTGT rules, and PTGT invariants given in Table 2 p. 60.
We used the eight PTGT rules with the prefix border- to overapproximate the step behavior of the large-scale PTGTSs allowing shuttles to enter and exit the fragment topology at hand in all ways this could be observable in the actual large-scale system for some discrete step between two large-scale states. For example, (i) the PTGT rule border-enter-drive-fast in Figure 54 p. 72 is used when a shuttle enters the fragment topology with a fast velocity,\(^{10,11}\) (ii) the PTGT rule border-exit-stop1 in Figure 60 p. 74 is used when a shuttle approaches the exiting track of the fragment topology (thereby its look-ahead is restricted to the next track only, as opposed to the PTGT rule stop1 in Figure 34 p. 62, which has a look-ahead of 2 tracks) and stops because there could be a shuttle on the track subsequent to track \(T_2\), and (iii) the PTGT rule border-exit-stop2 in Figure 61 p. 75 is used when a shuttle exits the fragment topology (thereby having no look-ahead) and stops because there could be a shuttle two tracks ahead (if there would be a shuttle one track ahead, then the shuttle would also not drive forward in the large-scale system).

For the considered fragment PTGTSs, we then performed the symbolic state space generation using a machine with 256 GB DDR4 and \(2 \times E5-2643\) Xeon @ 3.4 GHz \(\times 6\) cores \(\times 2\) threads. See Table 1 for details of our evaluation, where we provide average values over 10 runs for which we limited the Java runtime environment to 230 GB memory usage. We provide the resource usage in terms of runtime and memory consumption, information on the size of the resulting symbolic state space in terms of number of states and steps as well as the number of states labeled with the PTGT AP \(AP_{\text{collision}}\) from Figure 29 p. 59, the number of states labeled with the PTGT AP \(AP_{\text{velocity-unexpected}}\) from Figure 31 p. 59, and the worst-case probability calculated for a shuttle entering the fragment topology to eventually execute an emergency brake.

Firstly, we point out that we verified the absence of timelocks (P1), actor-based deadlocks (P3), and actor-based livelocks (P4).

Secondly, we determined that each of the PTGT rules assigned to the fragment PTGTSs according to Table 2 p. 60 was actually used when generating the symbolic state space, which rules out certain modeling errors such as states being unexpectedly unreachable or PTGT rules being unexpectedly not applicable anywhere.

Thirdly, for qualitative reachability (P2), we observed that no state is labeled with the PTGT APs \(AP_{\text{collision}}\) or \(AP_{\text{velocity-unexpected}}\). For \(AP_{\text{collision}}\), this means that, according to Theorem 2, our large-scale PTGTS with a large-scale topology constructed from the eight fragment topologies as described in Section 6 is safe w.r.t. the occurrence of collisions. For \(AP_{\text{velocity-unexpected}}\) this means

\[^{10}\text{To ensure that a certain track matched by one of these PTGT border rules is an entering track (i.e., a track without predecessor in the fragment topology) or an exiting track (i.e., a track without a successor in the fragment topology), we employ for tracks the attributes kind where an entering track has kind = −1, an exiting track has kind = 2, and the track preceding an exiting track has kind = 1. All other tracks have kind = 0. For example, the border PTGT rules for entering a fragment topology therefore check that the track \(T_i\) onto which a shuttle is created has the attribute kind = −1. Alternatively, the checking for the absence of predecessors/successors of tracks can be encoded using application conditions, but this would lead to more complex visualizations of the PTGT rules and can be expected to slow down the state space generation, as the evaluation of these application conditions requires further graph matching calls (e.g., in our tool AutoGraph).}\]

\[^{11}\text{In all PTGT border rules for the entering of fragment topologies, as already mentioned before, we use a System node to limit the number of shuttles per fragment topology. Also, we allow shuttles to get unique IDs assigned (where a shuttle always gets the smallest free ID to improve debugging). However, these additional IDs lead to larger symbolic state spaces, since, for many states of the fragment PTGTS with two shuttles, the same state is reached with swapped ID assignments. Technically, the IDs attribute of a System node is a list storing a binary encoding of the currently free and taken IDs where the ID \(i\) is free if and only if the element at position \(i\) in this list is 1. For an entering shuttle, the operation replace substitutes the first occurrence of 1 by 0 at index \(i\) to change the ID \(i\) in the list from free to taken, the operation indexof determines the index \(i\) of the first occurrence of 1 in the current list, and this index \(i\) is then assigned to the entering shuttle as its ID. For an exiting shuttle with ID \(i\), the operation replace substitutes the value 0 at index \(i\) by 1 to change the ID \(i\) in the list from taken to free. To disable the ID assignment, we use the empty IDs list in initial graphs, resulting in all IDs to be −1.}\]
Table 1. Results of Our Evaluation for the Running Example

| fragment topology | runtime [s] | memory [GB] | states | steps | collisions | velocity unexpected for emergency brake |
|-------------------|-------------|-------------|--------|-------|------------|-----------------------------------------|
| FT1               | 2           | 0.32        | 10     | 18    | 0          | 0                                       |
| FT2               | 11          | 2.36        | 183    | 296   | 0          | 0                                       |
| FT3               | 9           | 2.21        | 132    | 267   | 0          | 0                                       |
| FT4               | 127         | 9.75        | 2,140  | 3,923 | 0          | 0                                       |
| FT5               | 167         | 9.58        | 2,824  | 4,860 | 0          | $1.0 \times 10^{-6}$                     |
| FT6               | 74          | 9.33        | 1,238  | 2,266 | 0          | 0                                       |
| FT7               | 69          | 5.28        | 963    | 2,720 | 0          | 0                                       |
| FT9               | 1,420       | 16.92       | 12,319 | 20,461| 0          | $2.5 \times 10^{-3}$                    |

that no shuttle can enter any fragment topology with an unexpected velocity and that the used PTGT rules never change the velocity of a shuttle to an unexpected value. Hence, the additional PTGT border rules used in the fragment PTGTSs for overapproximating the entering of shuttles (border-enter-drive-fast in Figure 54| p. 72, border-enter-drive-slow in Figure 55| p. 72, border-enter-stop-fast in Figure 56| p. 73, and border-enter-stop-slow in Figure 57| p. 73) are sufficient, as they cover all possible velocities of shuttles.

Fourthly, the worst-case probability for a shuttle to execute an emergency brake in FT4 and FT5 is $10^{-6}$ and $10^{-12}$, respectively. Hence, the usage of FT5 for the construction of actual large-scale topologies is quantitatively more desirable compared to FT4 and, moreover, according to Theorem 2, the installations of yellow traffic lights as in FT4 and FT5 suitably decrease the likelihood of emergency brakes also for our large-scale PTGTS. However, the probabilities that some shuttle executes an emergency brake in a given time span in FT4/FT5 (obtained by combining the maximal throughput of shuttles for FT4/FT5 with the worst-case probability obtained for FT4/FT5) can be expected to be a too-coarse upper bound when the maximal throughput is not to be expected for the actual large-scale system. Moreover, the rate at which emergency brakes occur in the large-scale PTGTS further increases with the number of embeddings of instances of FT4 and FT5. For example, when shuttles pass $10^{12}$ times by a construction site with two preceding yellow traffic lights (as in the fragment topology FT5) in the actual large-scale system, we expect that at least one of these shuttles executes an emergency brake with probability of about $(1 - 10^{-12})^{10^{12}} \approx 0.37$.

Fifthly, for the fragment topology FT9, we have verified that shuttles entering the fragment topology execute an emergency brake with a probability of at most $0.05^2$, which demonstrates that each such execution of an emergency brake required the shuttle to have performed two failing connection attempts (each with a probability of 0.05) beforehand. As for FT4 and FT5, the probability for an emergency brake depends on the number of shuttles approaching the join track. However, the use of a general coordination mechanism at the global level may lead to much smaller probabilities (e.g., by allowing two shuttles to enter such a join fragment topology only rarely at once or by ensuring that one shuttle enters the fragment topology much earlier compared to the other shuttle).

Lastly, we pointed out in the previous section that actor-based fairness (P5) is violated for our PTGTS, since a shuttle S may, once it has stopped or braked, not perform a step ever again. To tackle this problem, we may include an additional PTGT rule in our PTGTS in the future, which first detects with higher priority as early as possible that the shuttle S may advance to a next track using the PTGT rule drive, then changes the mode of S to a new mode START, and, finally, resets the clock of the track connected to the shuttle S to 0. Employing an additional PTGT invariant, the shuttle S can then be forced to apply the PTGT rule drive not later than after a certain duration. However, further PTGT rules may then be required to reset the mode of S back to STOP, when the PTGT rule
drive becomes inapplicable before the shuttle then actually advances. With such a modification of the behavior of the large-scale PTGTS and the fragment PTGTSs, we could then attempt to verify fairness (i.e., that every shuttle will eventually exit the current fragment topology no matter how non-determinism is resolved). Furthermore, we may then even be able to quantitatively analyze the worst-case duration for (a) shuttles driving through fragment topologies and (b) shuttles not advancing temporarily.

The runtime and memory consumption of our prototypical implementation in AUTOGRAPH have not been an issue for the presented running example. However, this was not the case when allowing for an arbitrary number of shuttles per track topology (as in [37]), leading to an exponential increase of the size of the symbolic state space (generated for a fragment PTGTS) with every further shuttle/track. Moreover, there is a vast potential for optimizations of our current implementation regarding memory consumption (by only storing subsequently relevant information on states and steps) and runtime (by facilitating concurrency during state space generation).

10 CONCLUSION AND FUTURE WORK
We presented an analysis approach for a large-scale system modeled as a PTGTS for which analysis using model checking is not feasible. In our approach, we rely on a decomposition of the underlying static large-scale topology of the considered PTGTS into fragment topologies of manageable size. For every fragment topology obtained by decomposition, analysis is then performed for the fragment PTGTS obtained as an adaptation of the large-scale PTGTS to the considered fragment topology. As analysis results, we thereby determined (a) overapproximations of state reachability (expressing qualitative safety properties) and (b) upper bounds for probabilistic state reachability (expressing quantitative safety properties). Moreover, using a mechanism for monitoring the behavior of active components across steps, we were able to improve the analysis capabilities of our approach (compared to our earlier work in [37]) (a) by allowing for the analysis of the behavior of fragment PTGTSs w.r.t. the further important properties of timelock freedom, actor-based deadlock freedom, actor-based livelock freedom, and actor-based fairness as well as (b) by enhancing the precision for the analysis of quantitative safety properties employing a novel symbolic state space generation procedure for PTGTSs.

As future work, we intend to extend the formal relationship between the large-scale PTGTSs and its fragment PTGTSs to cover further behavioral properties of interest. This extension will also pertain to properties stated using the metric temporal graph logic (MTGL) [52], which allows to express more complex properties on timed graph transformation sequences, compared to those considered in this article, using a more elaborate monitoring mechanism at the graph level. Also, to cover further aspects of the RailCab project [47] considered in our running example, we will (i) develop more general decomposition schemes where dynamic elements may be connected across fragment topology borders and (ii) consider critical areas of a track topology that are protected by a semaphore-like regional control mechanism to be verified. Lastly, to further evaluate the applicability of our approach, we intend to apply it to other case studies as, e.g., the one discussed in [3].

APPENDICES
A COMPOSITION AND DECOMPOSITION OF LARGE-SCALE TOPOLOGIES
We consider two scenarios in which our analysis approach for large-scale PTGTSs may be used. In the first scenario, we now consider how topologies can be decomposed into instances of fragment topologies. In the second scenario, we consider how instances of fragment topologies can be composed into large-scale topologies. These two scenarios, therefore, capture general applications of our approach at a conceptual level. For our running example, we distinguish here between the
scenarios where the large-scale track topology is already given (and needs to be decomposed) and the case where a large-scale track topology is constructed based on existing and analyzed building blocks.

In the first scenario, a given large-scale system should be modeled as a PTGTS and then analyzed. Once the PTGTS is obtained, the large-scale topology of the initial state, a set $\mathcal{F}$ of fragment topologies, and a corresponding overlapping specification are to be defined by the user. Then, as the next step, a decomposition must be determined for each of the reachable large-scale states. All such decompositions can be obtained as discussed in the previous section by (a) decomposing the underlying large-scale topology contained in all large-scale states into fragment topologies and (b) then devising a procedure for assigning the dynamic elements contained in an arbitrary large-scale state to the fragment topologies leading to fragment states. The remaining problem is then the derivation of the coverage of the large-scale topology (cf. Definition 11) for a user-provided set $\mathcal{F}$ of fragment topologies. This problem can be understood to be a parsing problem, where the parsing process needs to determine how the fragment topologies can be used to cover the large-scale topology satisfying the overlapping specification. Intuitively, parsing thereby extends the matching of a single graph to the matching of an unknown number of copies of the fragment topologies from $\mathcal{F}$ with the additional requirement of satisfaction of the overlapping specification. Considering the problem of developing a parsing solution that works for arbitrary large-scale topologies, sets of fragment topologies, and overlapping specifications, we point out that graph parsing is NP-complete, implying that, while a coverage can always be determined when one exists, scenario-specific solutions may be more appropriate compared to general solutions. Note that for restricted settings, rather efficient parsing algorithms have been proposed for, e.g., hyperedge replacement grammars in [12–15].

For our running example, we conjecture that the parsing problem can be solved efficiently, since each fragment topology from Figure 2(b) (except for FT3) contains a unique structure compared to the other fragment topologies. In particular, (i) FT1 is the only fragment topology that has a depot with an exiting edge, (ii) FT2 is the only fragment topology that has a depot with an entering edge, (iii) FT4 is the only fragment topology that has a yellow traffic light and a construction site with an empty track in between, (iv) FT5 is the only fragment topology that has yellow traffic lights on two successive tracks, (v) FT6 is the only fragment topology that has a green traffic light, (vi) FT7 is the only fragment topology that has a track with two successors, and (vii) FT9 is the only fragment topology that has a track with two predecessors. Hence, all embeddings of these fragment topologies can be determined efficiently and unambiguously. All remaining tracks that have not been matched at this point must be covered using copies of fragment topology FT3. Due to the overlapping specification, there is always a unique way to extend the partial coverage obtained so far by overlapping an already determined embedding with an embedding of FT3 (unless there is a circle of tracks where any match of FT3 can be used to extend the partial coverage). Hence, we conclude that, in general, fragment topologies should ideally be defined to cover syntactically different parts of the large-scale topology at hand, allowing at best for such a backtracking-free parsing procedure.

Besides this example-specific parsing procedure exploiting the diversity of the fragment topologies, we can use a generic simple format of GT rules for parsing. For the fragment topology FT9, such GT rules are given in Figure 19. The idea for their construction is to match a subset of the 3

---

Some tools successfully derive graph morphisms by encoding this search problem as an SMT solver input. Each returned model then describes one graph morphism. The same approach could be used, in principle, for the problem of determining coverages. However, without any knowledge on the number of copies required for each fragment topology in the coverage, we expect that too-large input formulas would be generated especially for large-scale topologies.
Fig. 19. The GT rules for the fragment topology FT9 from Figure 2(b) for the composition/decomposition of large-scale topologies. For each GT rule, we assume an application condition requiring that the tracks $T_3$ and $T_4$ have no exiting edge and the track $T_7$ has no entering edge. Each of the eight depicted GT rules deletes no graph elements and adds a different part of the fragment topology. The left-hand side graph (left of the arrow) specifies which graph elements must exist before the GT rule is applied, and the right-hand side graph (right of the arrow) specifies which graph elements are added by rule application.

track chains (each consisting of 3 tracks) contained in FT9 and then to add the non-matched graph elements by applying one of the GT rules from Figure 19. Since FT9 has 3 entries/exits marked with the red arrows in Figure 2(b), there are $2^3 = 8$ GT rules defined for FT9. For all 8 fragment topologies from Figure 2(b), we obtain using the same construction idea $2^1 + 2^1 + 2^2 + 2^2 + 2^2 + 2^3 + 2^3 = 36$
GT rules in total. However, employing these 36 generically constructed GT rules backwards for parsing in an unguided manner is rather inefficient compared to the example-specific parsing described above.

In the second scenario, a large-scale system is developed in a model-driven manner by requiring the construction of a large-scale topology along with the PTGT rules to obtain the PTGTS modeling the large-scale system to be developed. In this scenario, composition operations ensuring the satisfaction of the overlapping specification must be determined to obtain the desired large-scale topology. For our running example, such a model-driven approach could involve the composition of the fragment topologies as basic building blocks by a track topology designer. The generically constructed GT rules from above could then be straightforwardly employed by such a designer for the manual generation of large-scale track topologies.

In both scenarios, a coverage $M$ of the large-scale topology is generated, explaining which fragment topologies have been embedded into the large-scale topology and where they have been embedded. Moreover, each incremental extension of $M$ in both scenarios ensures that embeddings are only added to the coverage when this does not result in a violation of the overlapping specification. Finally, in each case, the generation of the coverage $M$ is completed successfully when the monos in $M$ are jointly epimorphic, which means that they jointly map to all graph elements of the large-scale topology.

B  TOPOLOGY FRAGMENTS FOR RUNNING EXAMPLE

In addition to presenting the fragment topologies in our abbreviated notation in Figure 2(b), we depict them in Figures 20, 21, 22, 23, 24, 25, 26, and 27 (according to the type graph $TG$ from Figure 4) with all their attributes. Also, see Figure 28 for an embedding of three fragment topologies where the overlapping specification is not respected.

![Diagram 20: The fragment topology FT1 in unabbreviated form.](image)

![Diagram 21: The fragment topology FT2 in unabbreviated form.](image)

![Diagram 22: The fragment topology FT3 in unabbreviated form.](image)

Formal Aspects of Computing, Vol. 35, No. 3, Article 16. Publication date: September 2023.
Fig. 23. The fragment topology FT4 in unabbreviated form.

Fig. 24. The fragment topology FT5 in unabbreviated form.

Fig. 25. The fragment topology FT6 in unabbreviated form.
Fig. 26. The fragment topology FT7 in unabbreviated form.

Fig. 27. The fragment topology FT9 in unabbreviated form.
Fig. 28. Three embeddings $\alpha'_1$–$\alpha'_3$ of fragment topologies not respecting the overlapping specification (cf. Figure 14(a)) into a large-scale topology where two pairs of these embeddings overlap. For FT1 and FT5, not enough tracks are overlapped, and for FT5 and FT2, the overlapping introduces an unintended fork.

Fig. 29. The PTGT AP $\mathcal{A}_{\text{collision}}$ (already given in Figure 6(a)): It detects two shuttles on a common track representing a collision.

Fig. 30. The PTGT AP $\mathcal{A}_{\text{brake}}$: It detects a shuttle that has executed an emergency brake, as indicated by the value $\text{BRAKE}$ of its $\text{mode}$ attribute.

Fig. 31. The PTGT AP $\mathcal{A}_{\text{velocity-unexpected}}$: It detects a shuttle that has an unexpected minimal velocity.

Fig. 32. The PTGT AP $\mathcal{A}_{\text{disconnected-on-border}}$: It detects when a shuttle with a connection to another shuttle (being either convoy leader or convoy follower in that connection) is on an exiting track, as indicated by the $\text{kind}$ attribute.

C  COMPONENTS OF PTGTSS MODELING RUNNING EXAMPLE

We now present all PTGT APs, PTGT rules, and PTGT invariants used in our large-scale PTGTS $S_0$ and the fragment PTGTSs $S_i$ constructed for the fragment topologies $FT_i$ from Figure 2(b). An overview of these components is given in Table 2.
Table 2. PTGT APs, PTGT Rules, and PTGT Invariants Used for the Large-scale PTGTs $S_0$ and the Fragment PTGTs $S_i$ Constructed for the Fragment Topologies $F_{Ti}$ from Figure 2(b)

| PTGT AP                  | Figure       | Page | $S_0$ | $S_1$ | $S_2$ | $S_3$ | $S_4$ | $S_5$ | $S_6$ | $S_7$ | $S_9$ |
|--------------------------|--------------|------|-------|-------|-------|-------|-------|-------|-------|-------|-------|
| $AP_{collision}$         | Figure 29    | p. 59|       |       |       |       |       |       |       |       |       |
| $AP_{brake}$             | Figure 30    | p. 59|       |       |       |       |       |       |       |       |       |
| $AP_{velocity-unexpected}$| Figure 31    | p. 59|       |       |       |       |       |       |       |       |       |
| $AP_{disconnected-on-border}$ | Figure 32   | p. 59|       |       |       |       |       |       |       |       |       |

| PTGT Rule                | Figure       | Page | Priority | $S_0$ | $S_1$ | $S_2$ | $S_3$ | $S_4$ | $S_5$ | $S_6$ | $S_7$ | $S_9$ |
|--------------------------|--------------|------|----------|-------|-------|-------|-------|-------|-------|-------|-------|-------|
| $drive$                  | Figure 33    | p. 61| 0        |       |       |       |       |       |       |       |       |       |
| $stop1$                  | Figure 34    | p. 62| 0        |       |       |       |       |       |       |       |       |       |
| $drive-from-depot$       | Figure 35    | p. 62| 0        |       |       |       |       |       |       |       |       |       |
| $drive-to-depot1$        | Figure 36    | p. 63| 0        |       |       |       |       |       |       |       |       |       |
| $drive-to-depot2$        | Figure 37    | p. 63| 0        |       |       |       |       |       |       |       |       |       |
| $construction-site-brake$ | Figure 38   | p. 63| 0        |       |       |       |       |       |       |       |       |       |
| $construction-site-stop$ | Figure 39    | p. 64| 0        |       |       |       |       |       |       |       |       |       |
| $set-slow$               | Figure 40    | p. 64| 0        |       |       |       |       |       |       |       |       |       |
| $reset-yellow-traffic-light$ | Figure 41 | p. 65| 1        |       |       |       |       |       |       |       |       |       |
| $reset-green-traffic-light$ | Figure 42  | p. 65| 1        |       |       |       |       |       |       |       |       |       |
| $emergency-brake$        | Figure 43    | p. 66| 0        |       |       |       |       |       |       |       |       |       |
| $first-at-conflict$      | Figure 44    | p. 67| 0        |       |       |       |       |       |       |       |       |       |
| $stop2$                  | Figure 45    | p. 67| 1        |       |       |       |       |       |       |       |       |       |
| $connect$                | Figure 46    | p. 67| 1        |       |       |       |       |       |       |       |       |       |
| $disconnect1$            | Figure 47    | p. 68| 0        |       |       |       |       |       |       |       |       |       |
| $disconnect2$            | Figure 48    | p. 69| 1        |       |       |       |       |       |       |       |       |       |
| $stop-convoy$            | Figure 49    | p. 70| 1        |       |       |       |       |       |       |       |       |       |
| $drive-convoy1$          | Figure 50    | p. 70| 1        |       |       |       |       |       |       |       |       |       |
| $drive-convoy2$          | Figure 51    | p. 70| 0        |       |       |       |       |       |       |       |       |       |
| $border-enter-drive-fast$ | Figure 52    | p. 71| 0        |       |       |       |       |       |       |       |       |       |
| $border-enter-drive-slow$ | Figure 53    | p. 71| 0        |       |       |       |       |       |       |       |       |       |
| $border-enter-stop-fast$ | Figure 54    | p. 72| 0        |       |       |       |       |       |       |       |       |       |
| $border-enter-stop-slow$ | Figure 55    | p. 72| 0        |       |       |       |       |       |       |       |       |       |
| $border-exit-drive1$     | Figure 56    | p. 73| 0        |       |       |       |       |       |       |       |       |       |
| $border-exit-drive2$     | Figure 57    | p. 73| 0        |       |       |       |       |       |       |       |       |       |
| $border-exit-stop1$      | Figure 58    | p. 74| 0        |       |       |       |       |       |       |       |       |       |
| $border-exit-stop2$      | Figure 59    | p. 74| 0        |       |       |       |       |       |       |       |       |       |
| $border-exit-stop3$      | Figure 60    | p. 74| 0        |       |       |       |       |       |       |       |       |       |
| $border-exit-stop4$      | Figure 61    | p. 75| 0        |       |       |       |       |       |       |       |       |       |

| PTGT Invariant           | Figure       | Page | $S_0$ | $S_1$ | $S_2$ | $S_3$ | $S_4$ | $S_5$ | $S_6$ | $S_7$ | $S_9$ |
|--------------------------|--------------|------|-------|-------|-------|-------|-------|-------|-------|-------|-------|
| $INV_{driving}$          | Figure 62    | p. 75|       |       |       |       |       |       |       |       |       |
| $INV_{connect}$          | Figure 63    | p. 75|       |       |       |       |       |       |       |       |       |
| $INV_{first-at-conflict}$ | Figure 64    | p. 75|       |       |       |       |       |       |       |       |       |
| $INV_{disconnect}$       | Figure 65    | p. 75|       |       |       |       |       |       |       |       |       |
| $INV_{disconnect2}$      | Figure 66    | p. 75|       |       |       |       |       |       |       |       |       |
Fig. 33. The PTGT rule drive (given in simplified form in Figure 7(a)): A shuttle drives forward unless this would result in a dangerous situation or another PTGT rule has been designed for a more special case of driving.
Fig. 34. The PTGT rule stop1: A shuttle stops due to a shuttle too close ahead.

Fig. 35. The PTGT rule drive-from-depot: A shuttle enters the track topology from a depot.
Fig. 36. The PTGT rule drive-to-depot1: A shuttle approaches a depot and slows down if it drives fast.

Fig. 37. The PTGT rule drive-to-depot2: A shuttle exits the track topology to a depot.

Fig. 38. The PTGT rule construction-site-brake (already given in Figure 7(c)): A fast shuttle executes an emergency brake approaching a construction site.
Fig. 39. The PTGT rule construction-site-stop: A shuttle stops on a construction site due to a shuttle ahead.

requirement: m̄₁ = DRIVE \∧ minD₁ = 3

attribute guard: m₁ = DRIVE \∧ minD₁ = 3

clock guard: d_i ≥ minD₁

priority: 0

label: (sid_i, tid_i, tid_d)

attribute effect: m'₁ = STOP

unchanged variables: sid_i, minD₁, tid_i, tid_d

reset: (d_i)

probability: 1

\[ \neg \exists \{ S₂_Shuttle \} \text{at } T₂ \text{ Track } \]
Fig. 41. The PTGT rule \textit{set-slow} (already given in Figure 7(b)): A shuttle reaches a yellow traffic light and (a) observes the yellow traffic light and therefore slows down when it was driving fast or (b) does not observe the yellow traffic light and keeps its velocity.

Fig. 42. The PTGT rule \textit{reset-yellow-traffic-light}: The \textit{active} attribute of the yellow traffic light is reset once a shuttle has exited the track with the yellow traffic light on it.
Fig. 43. The PTGT rule *set-fast*: A shuttle reaches a green traffic light and (a) observes the green traffic light and therefore increases its velocity when it was driving slow or (b) does not observe the green traffic light and keeps its velocity.

Fig. 44. The PTGT rule *reset-green-traffic-light*: The *active* attribute of the green traffic light is reset once a shuttle has exited the track with the green traffic light on it.
Fig. 45. The PTGTrule emergency-brake (given in simplified form in Figure 9(b)): A shuttle executes an emergency brake just before a join track as it notices the other shuttle at this late point in time with which it could otherwise collide on the join track.

Fig. 46. The PTGTrule first-at-conflict: A shuttle determines that there is no shuttle on the opposite side of the conflicting track and it therefore knows that it is the first shuttle to reach this conflicting track.
Fig. 47. The PTGT rule stop2: A shuttle stops at a conflicting track because it notices a shuttle on the conflicting track ahead with which it did not attempt to establish a connection on the conflicting track to which it currently advanced.
Fig. 48. The PTGT rule `connect` (given in simplified form in Figure 9(a)): Two shuttles (that are not connected yet) attempt to establish a connection when being on opposite sides of the same conflicting track (when they have not yet attempted to establish a connection already, as indicated by their `canConnect` attributes); if a connection is established, then the convoy leader is the shuttle that first reached the conflicting track.
Fig. 49. The PTGT rule `disconnect1`: A connection between two shuttles is terminated when the shuttles are on a common track sequence with one track in between.

Fig. 50. The PTGT rule `disconnect2`: A connection between two shuttles is terminated when the shuttles are on a common track sequence with two tracks in between.

Fig. 51. The PTGT rule `stop-convoy`: A shuttle that is a convoy follower stops to ensure that it does not collide subsequently with its convoy leader.
Fig. 52. The PTGT rule *drive-convoy1*: A shuttle that is a convoy follower drives forward and reaches the conflicting track where its convoy leader is currently at.

Fig. 53. The PTGT rule *drive-convoy2*: A shuttle that is a convoy follower drives forward and is then one track away from the conflicting track where its convoy leader is currently at.
Fig. 54. The PTGT rule \texttt{border-enter-drive-fast}: An additional PTGT rule for a fresh fast shuttle entering the fragment topology.

Fig. 55. The PTGT rule \texttt{border-enter-drive-slow}: An additional PTGT rule for a fresh slow shuttle entering the fragment topology.
Fig. 56. The PTGT rule `border-enter-stop-fast`: An additional PTGT rule for a fresh fast shuttle entering the fragment topology and immediately stopping due to a shuttle ahead.

Fig. 57. The PTGT rule `border-enter-stop-slow`: An additional PTGT rule for a fresh slow shuttle entering the fragment topology and immediately stopping due to a shuttle ahead.
Fig. 58. The PTGT rule `border-exit-drive1`: An additional PTGT rule for a shuttle approaching an exiting border of the fragment topology.

Fig. 59. The PTGT rule `border-exit-drive2`: An additional PTGT rule for a shuttle exiting the fragment topology.

Fig. 60. The PTGT rule `border-exit-stop1`: An additional PTGT rule for a shuttle stopping when approaching an exiting border of the fragment topology.
Fig. 61. The PTGT rule `border-exit-stop2`: An additional PTGT rule for a shuttle stopping when exiting the fragment topology.

Fig. 62. The PTGT invariant `INV\text{\text{driving}}` (already given in Figure 6(b)): This PTGT invariant states that a driving shuttle must advance to the next track according to its `minDur` attribute.

Fig. 63. The PTGT invariant `INV\text{\text{connect}}`: This PTGT invariant states that a connection attempt must be performed as soon as this is possible.

Fig. 64. The PTGT invariant `INV\text{\text{first-at-conflict}}`: This PTGT invariant states that a shuttle must notice that it is the first at a conflicting track as soon as this is possible.

Fig. 65. The PTGT invariant `INV\text{\text{disconnect1}}`: This PTGT invariant states that a connection between two shuttles must be terminated as soon as this is possible using the PTGT rule `disconnect1` from Figure 49.

Fig. 66. The PTGT invariant `INV\text{\text{disconnect2}}`: This PTGT invariant states that a connection between two shuttles must be terminated as soon as this is possible using the PTGT rule `disconnect2` from Figure 50.
GLOSSARY

AC Attribute Condition.
AP Atomic Proposition.
CTMC Continuous Time Markov Chain.
DPD Discrete Probability Distribution.
GC Graph Condition.
GL Graph Logic.
GTS Graph Transformation System.
MDP Markov Decision Process.
MTGL Metric Temporal Graph Logic.
PA Probabilistic Automaton.
PGTS Probabilistic Graph Transformation System.
PTA Probabilistic Timed Automata.
PTCTL Probabilistic Timed Computation Tree Logic.
PTGTS Probabilistic Timed Graph Transformation System.
PTS Probabilistic Timed System.
SMT Satisfiability Modulo Theories.
TA Timed Automaton.
TGTS Timed Graph Transformation System.

REFERENCES

[1] Department of Information Technology at Uppsala University, Sweden and Department of Computer Science at Aalborg University, Denmark. 2021. UPPAAL. Department of Information Technology at Uppsala University, Sweden and Department of Computer Science at Aalborg University, Denmark. Retrieved from https://uppaal.org/.

[2] Rajeev Alur and David L. Dill. 1994. A theory of timed automata. Theor. Comput. Sci. 126, 2 (1994), 183–235. DOI: https://doi.org/10.1016/0304-3975(94)90010-8

[3] Paolo Baldan, Andrea Corradini, and Barbara König. 2002. Static analysis of distributed systems with mobility specified by graph grammars—A case study. In International Conference on Integrated Design & Process Technology. SDPS.

[4] Basil Becker. 2014. Architectural Modelling and Verification of Open Service-oriented Systems of Systems. Ph.D. Dissertation. Hasso-Plattner-Institut für Softwaresystemtechnik, Universität Potsdam. Retrieved from http://opus.kobv.de/ubp/volltexte/2014/7015/.

[5] Basil Becker, Dirk Beyer, Holger Giese, Florian Klein, and Daniela Schilling. 2006. Symbolic invariant verification for systems with dynamic structural adaptation. In 28th International Conference on Software Engineering (ICSE’06). ACM, 72–81. DOI: https://doi.org/10.1145/1134285.1134297

[6] Basil Becker and Holger Giese. 2008. On safe service-oriented real-time coordination for autonomous vehicles. In 11th IEEE International Symposium on Object-Oriented Real-time Distributed Computing (ISORC’08). IEEE Computer Society, 203–210. DOI: https://doi.org/10.1109/ISORC.2008.13

[7] Basil Becker, Holger Giese, and Stefan Neumann. 2009. Correct Dynamic Service-oriented Architectures: Modeling and Compositional Verification with Dynamic Collaborations. Technical Report 29. Hasso Plattner Institute at the University of Potsdam.

[8] Johan Bengtsson and Wang Yi. 2003. Timed automata: Semantics, algorithms and tools. In Lectures on Concurrency and Petri Nets, Advances in Petri Nets (Lecture Notes in Computer Science, Vol. 3098). Springer, 87–124. DOI: https://doi.org/10.1007/978-3-540-27755-2_3

[9] Juan de Lara, Esther Guerra, Artur Boronat, Reiko Heckel, and Paolo Torrini. 2014. Domain-specific discrete event modelling and simulation using graph transformation. Softw. Syst. Model. 13, 1 (2014), 209–238. DOI: https://doi.org/10.1007/s10270-012-0242-3

[10] Willem P. de Roever, Hans Langmaack, and Amir Pnueli (Eds.). 1998. International Symposium on Compositionality: The Significant Difference. (Lecture Notes in Computer Science, Vol. 1536). Springer. DOI: https://doi.org/10.1007/3-540-49213-5

[11] David L. Dill. 1989. Timing assumptions and verification of finite-state concurrent systems. In International Workshop on Automatic Verification Methods for Finite State Systems (Lecture Notes in Computer Science, Vol. 407). Springer, 197–212. DOI: https://doi.org/10.1007/3-540-52148-8_17

Formal Aspects of Computing, Vol. 35, No. 3, Article 16. Publication date: September 2023.
[12] Frank Drewes, Berthold Hoffmann, and Mark Minas. 2015. Predictive top-down parsing for hyperedge replacement grammars. In 8th International Conference on Graph Transformation, Held as Part of STAF 2015 (Lecture Notes in Computer Science, Vol. 9151). Springer, 19–34. DOI: https://doi.org/10.1007/978-3-319-21145-9_2

[13] Frank Drewes, Berthold Hoffmann, and Mark Minas. 2019. Formalization and correctness of predictive shift-reduce parsers for graph grammars based on hyperedge replacement. J. Log. Algebr. Meth. Program. 104 (2019), 303–341. DOI: https://doi.org/10.1016/j.jlamp.2018.12.006

[14] Frank Drewes, Berthold Hoffmann, and Mark Minas. 2020. Graph parsing as graph transformation—correctness of predictive top-down parsers. In 13th International Conference on Graph Transformation, Held as Part of STAF 2020 (Lecture Notes in Computer Science, Vol. 12150). Springer, 221–238. DOI: https://doi.org/10.1007/978-3-030-51372-6_13

[15] Frank Drewes, Berthold Hoffmann, and Mark Minas. 2021. Rule-based top-down parsing for acyclic contextual hyperedge replacement grammars. In 14th International Conference on Graph Transformation, Held as Part of STAF 2021 (Lecture Notes in Computer Science, Vol. 12741). Springer, 164–184. DOI: https://doi.org/10.1007/978-3-030-78946-6_9

[16] Johannes Dyck. 2020. Verification of Graph Transformation Systems with k-Inductive Invariants. Ph. D. Dissertation. University of Potsdam, Hasso Plattner Institute, Potsdam, Germany. DOI: https://doi.org/10.25932/publshup-44274

[17] Hartmut Ehrig, Karsten Ehrg, Ulrike Prange, and Gabriele Taentzer. 2006. Fundamentals of Algebraic Graph Transformation. Springer.

[18] Amir Hossein Ghamarian and Arend Rensink. 2012. Generalised compositionality in graph transformation. In 6th International Conference on Graph Transformation (Lecture Notes in Computer Science, Vol. 7562). Springer, 234–248. DOI: https://doi.org/10.1007/978-3-642-33654-6_16

[19] Holger Giese. 2005. Modeling and verification of cooperative self-adaptive mechatronic systems. In 12th Monterey Workshop on Reliable Systems on Unreliable Networked Platforms (Lecture Notes in Computer Science, Vol. 4322). Springer, 258–280. DOI: https://doi.org/10.1007/978-3-540-71556-8_14

[20] Holger Giese, Maria Maximova, Lucas Sakizloglou, and Sven Schneider. 2019. Metric temporal graph logic over typed attributed graphs. In 22nd International Conference on Fundamental Approaches to Software Engineering, Held as Part of the European Joint Conferences on Theory and Practice of Software (Lecture Notes in Computer Science, Vol. 11424). Springer, 282–298. DOI: https://doi.org/10.1007/978-3-030-16722-6_16

[21] Holger Giese and Wilhelm Schäfer. 2013. Model-driven development of safe self-optimizing mechatronic systems with MechatronicUML. In Assurances for Self-Adaptive Systems—Principles, Models, and Techniques. (Lecture Notes in Computer Science, Vol. 7740). Springer, 152–186. DOI: https://doi.org/10.1007/978-3-642-36249-1_6

[22] Holger Giese, Matthias Tichy, Sven Burmester, and Stephan Flake. 2003. Towards the compositional verification of real-time UML designs. In 11th ACM SIGSOFT Symposium on Foundations of Software Engineering 2003 Held Jointly with 9th European Software Engineering Conference. ACM, 38–47. DOI: https://doi.org/10.1145/940071.940078

[23] Anne-gret Habel and Karl-Heinz Pennemann. 2009. Correctness of high-level transformation systems relative to nested conditions. Math. Struct. Comput. Sci. 19, 2 (2009), 245–296. DOI: https://doi.org/10.1017/S0960129508007202

[24] Reiko Heckel, Georgios Lajios, and Sebastian Menge. 2006. Stochastic graph transformation systems. In 2nd International Conference on Graph Transformation (Lecture Notes in Computer Science, Vol. 326). Springer, 210–225. DOI: https://doi.org/10.1007/11537280_16

[25] Reiko Heckel, Georgios Lajios, and Sebastian Menge. 2006. Stochastic graph transformation systems. Fun dam. Inform. 74, 1 (2006), 63–84. Retrieved from http://content.iospress.com/articles/fundamenta-informatica/fi74-1-04.

[26] Henshin Team. 2021. Henshin. Retrieved from https://www.eclipse.org/henshin/.

[27] Kenneth Johnson, Radu Calinescu, and Shinji Kikuchi. 2013. An incremental verification framework for component-based software systems. In 16th ACM SIGSOFT Symposium on Component Based Software Engineering, Part of Component Place ‘13. ACM, 33–42. DOI: https://doi.org/10.1145/2465449.2465456

[28] Joost-Pieter Katoen, Marta Z. Kwiatkowska, Gethin Norman, and David Parker. 2001. Faster and symbolic CTMC model checking. In Joint International Workshop on Process Algebra and Probabilistic Methods, Performance Modeling and Verification (Lecture Notes in Computer Science, Vol. 2165). Springer, 23–38. DOI: https://doi.org/10.1007/3-540-44804-7_2

[29] Christian Krause and Holger Giese. 2011. Model checking probabilistic real-time properties for service-oriented systems with service level agreements. In 13th International Workshop on Verification of Infinite-State Systems (EPTCS 11, Vol. 73). 64–78. DOI: https://doi.org/10.4204/EPTCS.73.8

[30] Christian Krause and Holger Giese. 2012. Probabilistic graph transformation systems. In 6th International Conference on Graph Transformation (Lecture Notes in Computer Science, Vol. 7562). Springer, 311–325. DOI: https://doi.org/10.1007/978-3-642-33654-6_21

[31] Marta Z. Kwiatkowska, Gethin Norman, and David Parker. 2011. PRISM 4.0: Verification of probabilistic real-time systems. In 23rd International Conference on Computer Aided Verification (Lecture Notes in Computer Science, Vol. 6806). Springer, 585–591. DOI: https://doi.org/10.1007/978-3-642-22110-1_47

Formal Aspects of Computing, Vol. 35, No. 3, Article 16. Publication date: September 2023.
[32] Marta Z. Kwiatkowska, Gethin Norman, David Parker, and Hongyang Qu. 2013. Compositional probabilistic verification through multi-objective model checking. *Inf. Comput.* 232 (2013), 38–65. DOI: https://doi.org/10.1016/j.ic.2013.10.001

[33] Marta Z. Kwiatkowska, Gethin Norman, Roberto Segala, and Jeremy Sproston. 2002. Automatic verification of real-time systems with discrete probability distributions. *Theor. Comput. Sci.* 282, 1 (2002), 101–150. DOI: https://doi.org/10.1016/S0304-3975(01)00046-9

[34] Marta Z. Kwiatkowska, Gethin Norman, Jeremy Sproston, and Fuzhi Wang. 2004. Symbolic model checking for probabilistic timed automata. In *Joint International Conferences on Formal Modelling and Analysis of Timed Systems and Formal Techniques in Real-Time and Fault-Tolerant Systems (Lecture Notes in Computer Science, Vol. 3253)*. Springer, 293–308. DOI: https://doi.org/10.1007/978-3-540-30206-3_21

[35] Maria Maximova. 2019. *Behavior and Confluence Analysis of M-adhesive Transformation Systems Using M-functors*. Ph. D. Dissertation. Technical University of Berlin, Germany. Retrieved from https://nbn-resolving.org/urn:nbn:de:101:2019122500571906965554.

[36] Maria Maximova, Holger Giese, and Christian Krause. 2018. Probabilistic timed graph transformation systems. *J. Log. Algebr. Meth. Program.* 101 (2018), 110–131. DOI: https://doi.org/10.1016/j.jlamp.2018.09.003

[37] Maria Maximova, Sven Schneider, and Holger Giese. 2021. Compositional analysis of probabilistic timed graph transformation systems. In *24th International Conference on Fundamental Approaches to Software Engineering, Held as Part of the European Joint Conferences on Theory and Practice of Software (Lecture Notes in Computer Science, Vol. 12649)*. Springer, 196–217. DOI: https://doi.org/10.1007/978-3-030-71500-7_10

[38] Maria Maximova, Sven Schneider, and Holger Giese. 2021. Interval probabilistic timed graph transformation systems. In *14th International Conference on Graph Transformation (Lecture Notes in Computer Science, Vol. 12741)*. Springer, 221–239. DOI: https://doi.org/10.1007/978-3-030-78946-6_12

[39] Maria Maximova, Sven Schneider, and Holger Giese. 2021. *Interval Probabilistic Timed Graph Transformation Systems*. Technical Report 134. Hasso Plattner Institute at the University of Potsdam, Potsdam, Germany.

[40] Microsoft Corporation. 2021. Z3. Retrieved from https://github.com/Z3Prover/z3.

[41] Robin Milner. 1989. *Communication and Concurrency*. Prentice Hall.

[42] Stefan Neumann. 2007. *Modellierung und Verifikation zeitbehafteter Graphtransformationssysteme mittels Groove*. Master’s Thesis. University of Paderborn.

[43] J. R. Norris. 1997. *Markov Chains*. Cambridge University Press. DOI: https://doi.org/10.1017/CBO9780511810633

[44] Fernando Orejas. 2011. Symbolic graphs for attributed graph constraints. *J. Symb. Comput.* 46, 3 (2011), 294–315. DOI: https://doi.org/10.1016/j.jsc.2010.09.009

[45] Fernando Orejas and Leen Lambers. 2012. Lazy graph transformation. *Fundam. Inform.* 118, 1-2 (2012), 65–96. DOI: https://doi.org/10.3233/FI-2012-706

[46] Michael Oser Rabin. 1963. Probabilistic automata. *Inf. Contr.* 6, 3 (1963), 230–245. DOI: https://doi.org/10.1016/S0019-9958(63)90290-0

[47] RailCab Team. 2023. RailCab Project. Retrieved from https://www.hni.uni-paderborn.de/cim/projekte/railcab.

[48] Arend Rensink. 2010. Compositionality in graph transformation. In *37th International Colloquium on Automata, Languages and Programming (Lecture Notes in Computer Science, Vol. 6199)*. Springer, 309–320. DOI: https://doi.org/10.1007/978-3-642-14162-1_26

[49] Sven Schneider, Johannes Dyck, and Holger Giese. 2020. Formal verification of invariants for attributed graph transformation systems based on nested attributed graph conditions. In *13th International Conference on Graph Transformation, Held as Part of STAF 2020 (Lecture Notes in Computer Science, Vol. 12150)*. Springer, 257–275. DOI: https://doi.org/10.1007/978-3-030-51372-6_15

[50] Sven Schneider, Leen Lambers, and Fernando Orejas. 2018. Automated reasoning for attributed graph properties. *Int. J. Softw. Tools Technol. Transf.* 20, 6 (2018), 705–737. DOI: https://doi.org/10.1007/s10009-018-0496-3

[51] Sven Schneider, Maria Maximova, Lucas Sakizloglou, and Holger Giese. 2021. Formal testing of timed graph transformation systems using metric temporal graph logic. *Int. J. Softw. Tools Technol. Transf.* 23, 3 (2021), 411–488. DOI: https://doi.org/10.1007/s10009-020-00585-w

[52] Sven Schneider, Lucas Sakizloglou, Maria Maximova, and Holger Giese. 2020. Optimistic and pessimistic on-the-fly analysis for metric temporal graph logic. In *13th International Conference on Graph Transformation, Held as Part of STAF 2020 (Lecture Notes in Computer Science, Vol. 12150)*. Springer, 276–294. DOI: https://doi.org/10.1007/978-3-030-51372-6_16

[53] Roberto Segala. 1995. *Modeling and Verification of Randomized Distributed Real-time Systems*. Ph. D. Dissertation. Massachusetts Institute of Technology, Cambridge, MA. Retrieved from http://hdl.handle.net/1721.1/36560.
[54] Shuling Wang, Naijun Zhan, and Lijun Zhang. 2017. A compositional modelling and verification framework for stochastic hybrid systems. *Formal Aspects Comput.* 29, 4 (2017), 751–775. DOI: https://doi.org/10.1007/s00165-017-0421-7

[55] Junhua Zhang, Jun Zhao, Zhiqiu Huang, and Zining Cao. 2009. Model checking interval probabilistic timed automata. In *1st International Conference on Information Science and Engineering*. 4936–4940. DOI: https://doi.org/10.1109/ICISE.2009.749

Received 27 January 2022; revised 9 November 2022; accepted 13 November 2022