Compositional Analysis of Probabilistic Timed Graph Transformation Systems

Maria Maximova, Sven Schneider, Holger Giese

Technische Berichte Nr. 133

des Hasso-Plattner-Instituts für
Digital Engineering an der Universität Potsdam

Technische Berichte des Hasso-Plattner-Instituts für
Digital Engineering an der Universität Potsdam

Technische Berichte des Hasso-Plattner-Instituts für Digital Engineering an der Universität Potsdam | 133

Maria Maximova | Sven Schneider | Holger Giese

Compositional Analysis of Probabilistic Timed Graph Transformation Systems

Universitätsverlag Potsdam

Bibliografische Information der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der
Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind
im Internet über http://dnb.dnb.de/ abrufbar.

Universitätsverlag Potsdam 2021
http://verlag.ub.uni-potsdam.de/

Am Neuen Palais 10, 14469 Potsdam
Tel.: +49 (0)331 977 2533 / Fax: 2292
E-Mail: verlag@uni-potsdam.de

Die Schriftenreihe **Technische Berichte des Hasso-Plattner-Instituts für Digital Engineering an der Universität Potsdam** wird herausgegeben von den Professoren des Hasso-Plattner-Instituts für Digital Engineering an der Universität Potsdam.

ISSN (print) 1613-5652
ISSN (online) 2191-1665

ISBN 978-3-86956-501-9

Zugleich online veröffentlicht auf dem Publikationsserver der Universität Potsdam:
https://doi.org/10.25932/publishup-49013
https://nbn-resolving.org/urn:nbn:de:kobv:517-opus4-490131

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 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 probabilistic timed graph transformation systems 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 we model shuttles driving on tracks of a large-scale topology and for which we verify that shuttles never collide and are unlikely to execute emergency brakes. In our evaluation, we apply an implementation of our approach to the running example.[1]

[1]Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) — 241885098, 148420506.

Contents

1 Introduction

Real-time cyber-physical systems often emit a complex behavior based on e.g. concurrent 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 models representing states of the system can be understood as graphs. In particular, the formalism of probabilistic timed graph transformation systems (PTGTSs) extends the standard rule-based transformation of graphs such that timed and probabilistic behavior is covered by supporting (a) non-deterministic choice among steps, (b) probabilistic choice among step results, and (c) steps representing the passage of time.

A model checking approach for PTGTSs w.r.t. probabilistic metric temporal properties was introduced in [19]. However, also this model checking approach is limited to systems with rather small state spaces due to the state space explosion problem. As a workaround, a selected set of small examples may be considered 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 examples do not reveal all the threatening behavior.

We present a decomposition-based approach for the analysis of large-scale systems modeled as PTGTSs to rule out violations of qualitative and quantitative safety properties.

As a first step, we capture the underlying static large-scale topology (LST) of a large-scale system as a subgraph that is not changed by graph transformation, describe how a fragment topology (FT) can be embedded into such an LST (see the left part of Figure 1.1), and specify how multiple such embeddings of FTs can overlap in their borders (see the right part of Figure 1.1).

As a second step, based on the decomposition described by such embeddings, we construct for each FT an adapted PTGTS. Such an adapted PTGTS is then ensured to (a) exhibit the same behavior on the non-overlapped part of the FT (named *core*) and to (b) simulate all possible behaviors that can happen for any occurrence of the FT in an LST. To obtain the mentioned simulation, we include modifications of the rules of the original PTGTS operating on the border of an FT into the adapted PTGTS. With this direct relationship between behaviors on the FTs and the LST, we obtain that the likelihood of an unwanted or forbidden graph pattern in one of the adapated PTGTS is an upper bound for its likelihood in its embedding in the large-scale PTGTS.

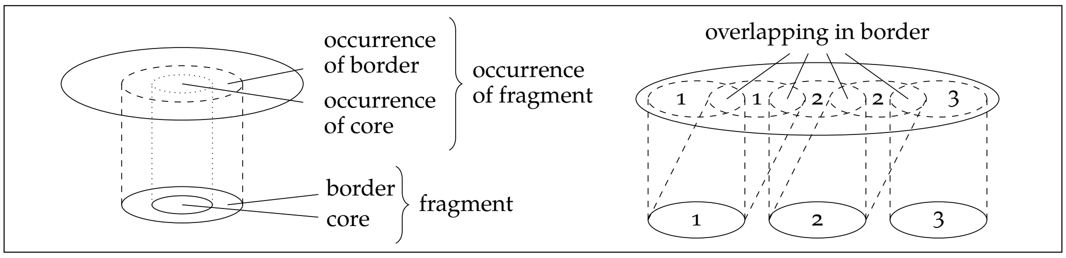


Figure 1.1: Occurrence of single FT with border and core in LST (left) and five occurrences of three FTs in LST overlapping in their borders (right)

As a last step, exploiting our decomposition to counter the state space explosion problem, we apply the model checking approach from [19] to the PTGTSs constructed for the FTs employing its reduction to probabilistic timed automata (PTA) instead of applying the model checking approach directly to the PTGTS modeling the large-scale system.

To illustrate our approach, we consider a running example in which we model shuttles driving on tracks of an LST and for which we verify that shuttles never collide and are unlikely to execute emergency brakes. In our evaluation, we apply an implementation of our approach to the running example.

The idea to decompose a system into subsystems or to compose it from subsystems for the analysis has been studied intensively [24] but our suggested compositional approach has distinguishing characteristics. Firstly, the vast majority of approaches (like 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. In contrast, we do not rely on a built-in decomposition operator but rather allow for a flexible derivation of an LST decomposition in terms of FTs, 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 graph transformation systems (GTSs) from [11, 23] 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 [14], 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 [12, 13] captured by timed GTSs (TGTSs) and their analysis via inductive invariant checking [3, 4]. Later on, this approach has been extended to role, component, and collaboration behavior, which is captured by TGTSs and hybrid GTSs in [5] and [2], 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.

This paper is structured as follows. In chapter 2, we introduce our running example from the domain of cyber-physical systems. In chapter 3, we recapitulate the necessary preliminaries related to PTA and PTGTSs also presenting the modeling of our running example. In chapter 4, we discuss the decomposition of static substructures of large-scale systems. In chapter 5, we present our decomposition-based approach allowing to split the model checking problem into more manageable parts. In chapter 6, we present an evaluation of the conceptual results for our running example. Finally, in chapter 7, we close the paper with a conclusion and an outlook on planned future work.

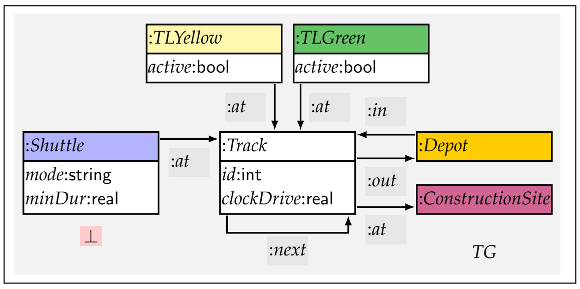
2 Running Example

We now informally introduce a scenario (based on the RailCab project [22]) of autonomous shuttles driving on an LST, which serves as a running example in the remainder of this paper. Based on this introduction, we will discuss how we model this shuttle scenario as a PTGTS in the next chapter.

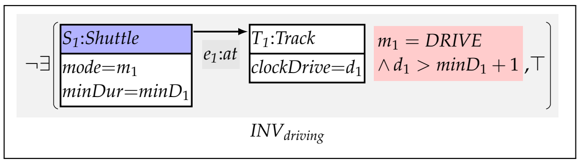
In the considered shuttle scenario, a track topology containing a large number of tracks of approximately equal length is given. Tracks are connected to the adjacent tracks via directed connections building in this manner track sequences. Two track sequences can be joined together (i.e., can end up in a common track with two predecessors) leading to a *join* fragment topology (see FT8 in Figure 4.1a) or can split up from a common track (i.e., a common track has then two successor tracks) leading to a *fork* fragment topology (see FT7 in Figure 4.1a). Moreover, depots may have a directed connection to a track allowing shuttles to enter or exit the track topology. Shuttles, which are always located on a single track, may be in mode *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). Regularly, shuttles change into mode *STOP*, which allows them to avoid coming too close to other shuttles. Moreover, shuttles should slow down before entering a track with a construction site on it. However, shuttles noticing the construction site too late have to execute an emergency brake thereby changing into the mode *BRAKE*. To reduce the likelihood of such emergency brakes, yellow traffic lights are installed a few tracks ahead of such construction sites to indicate to shuttles that they should slow down. After construction sites, green traffic lights may be installed permitting shuttles to increase their velocity. However, we also consider failures on demand where a traffic light that is passed by a shuttle is not recognized or, for some other reason, not appropriately taken into account by the shuttle. We assume a failure probability of 10^{-6} for this case assuming that the failure does not only depend on the visual observation by the train driver but also depends on a failure of the backup system.

In our running example, *static* elements are the tracks, depots, installed traffic lights, and construction sites as well as connections between these elements. The PTGTS modeling the behavior of the described scenario never changes this underlying LST. Complementary, *dynamic* elements are shuttles, their attributes, their connections to tracks of the LST as well as the attributes of traffic lights. Note that we use later a grammar to generate admissible LSTs.

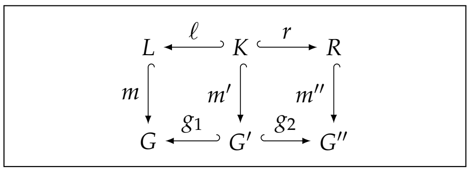
For the considered shuttle scenario, we are interested in various properties. Firstly, we need to verify that the behavior of the system never gets temporally stuck in a state where no steps (discrete steps of e.g. driving shuttles or timed steps) are



(a) Type graph

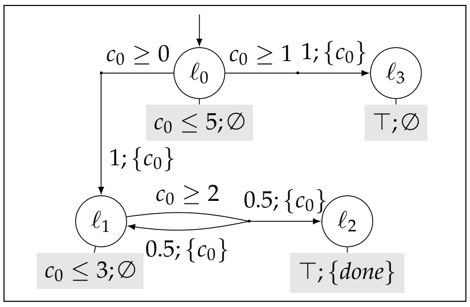


(b) Invariant

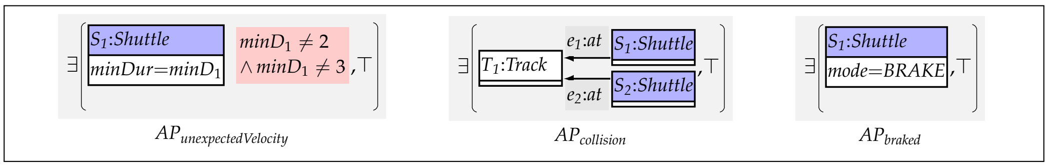


$$\begin{tikzcd}
L \arrow[d, "m"'] & K \arrow[l, "\ell"'] \arrow[r, "r"] \arrow[d, "m'"] & R \arrow[d, "m''"] \\
G & G' \arrow[l, "g_1"'] \arrow[r, "g_2"] & G''
\end{tikzcd}$$

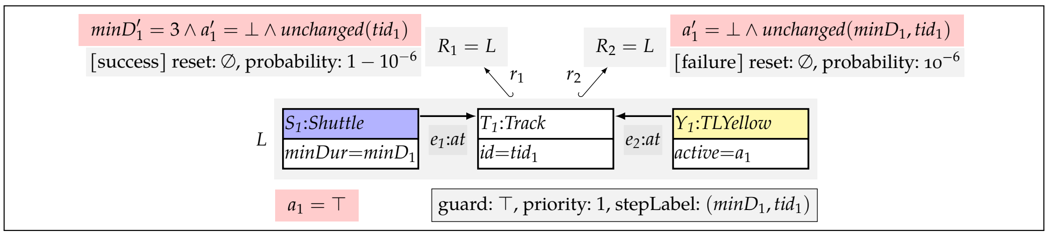
(c) DPO diagram



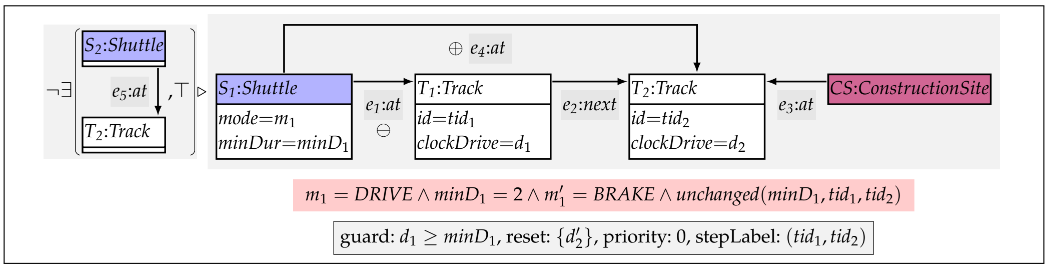
(d) Example of a PTA



(e) Atomic propositions

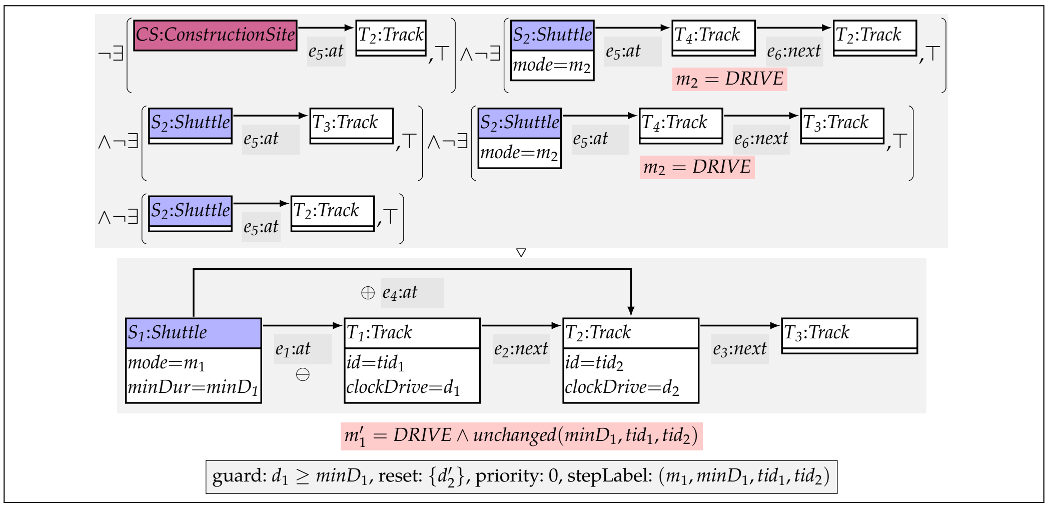

$minD_1' = 3 \wedge a_1' = \perp \wedge unchanged(tid_1)$
[success] reset: ∅, probability: $1 - 10^{-6}$
$R_1 = L$
r_1
$R_2 = L$
r_2
$a_1' = \perp \wedge unchanged(minD_1, tid_1)$
[failure] reset: ∅, probability: 10^{-6}
L
S1:Shuttle
minDur=minD1
e1:at
T1:Track
id=tid1
e2:at
Y1:TLYellow
active=a1
$a_1 = \top$
guard: ⊤, priority: 1, stepLabel: $(minD_1, tid_1)$

(f) The rule *SetSlow*: a shuttle may *successfully* decrease its velocity by setting its time per track to $[3, 4]$ (where only the lower end of the interval is stored in the graph) with probability $1 - 10^{-6}$ or may *fail* to decrease its velocity with probability 10^{-6}. Setting the *active* attribute to $\perp$ ensures that the rule cannot be applied twice.


¬∃
S2:Shuttle
e5:at
T2:Track
,⊤ ▷
⊕ e4:at
S1:Shuttle
mode=m1
minDur=minD1
e1:at
⊖
T1:Track
id=tid1
clockDrive=d1
e2:next
T2:Track
id=tid2
clockDrive=d2
e3:at
CS:ConstructionSite
$m_1 = DRIVE \wedge minD_1 = 2 \wedge m_1' = BRAKE \wedge unchanged(minD_1, tid_1, tid_2)$
guard: $d_1 \geq minD_1$, reset: $\{d_2'\}$, priority: 0, stepLabel: (tid_1, tid_2)

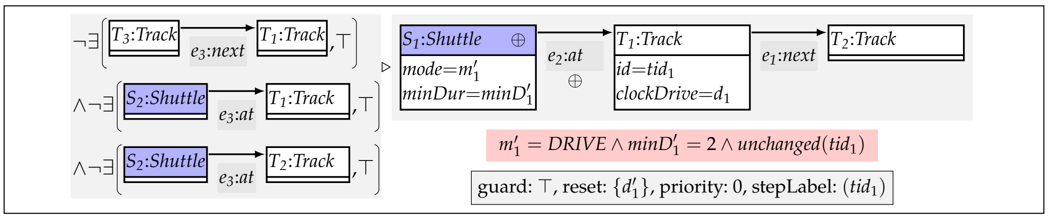
(g) The rule *ConstructionSiteBrake*: a shuttle with high velocity ($[2, 3]$ time units per track where only the lower end of the interval is stored in the graph) needs to execute an emergency brake to ensure that the track with a construction site on it is not entered with a too high velocity.

Figure 2.1: Details for our running example, DPO diagram, and PTA example



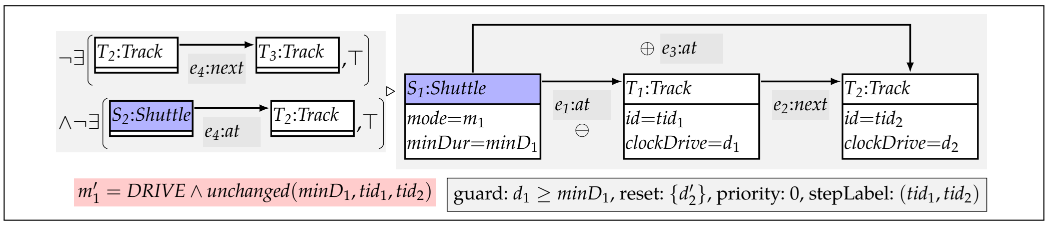
$\neg\exists$ CS:ConstructionSite e_5:at T_2:Track $,\top$
$\wedge\neg\exists$ S_2:Shuttle mode=m_2 e_5:at T_4:Track e_6:next T_2:Track $,\top$
$m_2 = DRIVE$
$\wedge\neg\exists$ S_2:Shuttle e_5:at T_3:Track $,\top$
$\wedge\neg\exists$ S_2:Shuttle mode=m_2 e_5:at T_4:Track e_6:next T_3:Track $,\top$
$m_2 = DRIVE$
$\wedge\neg\exists$ S_2:Shuttle e_5:at T_2:Track $,\top$
$\triangledown$
$\oplus$ e_4:at
S_1:Shuttle mode=m_1 minDur=$minD_1$
e_1:at $\ominus$
T_1:Track id=tid_1 clockDrive=d_1
e_2:next
T_2:Track id=tid_2 clockDrive=d_2
e_3:next
T_3:Track
$m_1' = DRIVE \wedge unchanged(minD_1, tid_1, tid_2)$
guard: $d_1 \geq minD_1$, reset: $\{d_2'\}$, priority: 0, stepLabel: $(m_1, minD_1, tid_1, tid_2)$

(**a**) The rule *Drive*: a shuttle may drive to the next track where the application condition is used to rule out situations that on the next track is a construction site or that the considered shuttle comes too close to another shuttle.



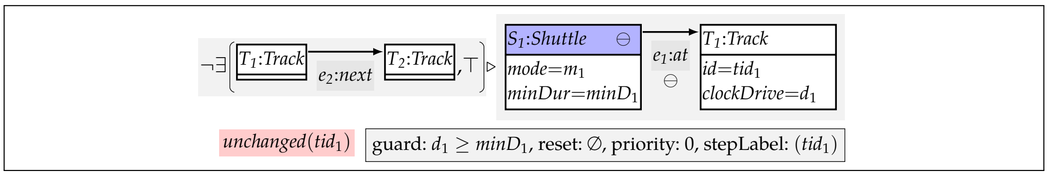
$\neg\exists$ T_3:Track e_3:next T_1:Track $,\top$
$\wedge\neg\exists$ S_2:Shuttle e_3:at T_1:Track $,\top$
$\wedge\neg\exists$ S_2:Shuttle e_3:at T_2:Track $,\top$
$\triangleright$
S_1:Shuttle $\oplus$ mode=m_1' minDur=$minD_1'$
e_2:at $\oplus$
T_1:Track id=tid_1 clockDrive=d_1
e_1:next
T_2:Track
$m_1' = DRIVE \wedge minD_1' = 2 \wedge unchanged(tid_1)$
guard: $\top$, reset: $\{d_1'\}$, priority: 0, stepLabel: (tid_1)

(**b**) The rule *DriveEnterFast*: adaptation of the rule *Drive* for the case that a new shuttle enters the current fragment topology with a high velocity (the similar rule for a shuttle with a low velocity has been omitted here for brevity) from a context track belonging to another fragment topology.



$\neg\exists$ T_2:Track e_4:next T_3:Track $,\top$
$\wedge\neg\exists$ S_2:Shuttle e_4:at T_2:Track $,\top$
$\triangleright$
$\oplus$ e_3:at
S_1:Shuttle mode=m_1 minDur=$minD_1$
e_1:at $\ominus$
T_1:Track id=tid_1 clockDrive=d_1
e_2:next
T_2:Track id=tid_2 clockDrive=d_2
$m_1' = DRIVE \wedge unchanged(minD_1, tid_1, tid_2)$
guard: $d_1 \geq minD_1$, reset: $\{d_2'\}$, priority: 0, stepLabel: (tid_1, tid_2)

(**c**) The rule *DriveExit1*: adaptation of the rule *Drive* for the case that a shuttle drives onto the last track of the current fragment topology.



$\neg\exists$ T_1:Track e_2:next T_2:Track $,\top$
$\triangleright$
S_1:Shuttle $\ominus$ mode=m_1 minDur=$minD_1$
e_1:at $\ominus$
T_1:Track id=tid_1 clockDrive=d_1
$unchanged(tid_1)$
guard: $d_1 \geq minD_1$, reset: $\emptyset$, priority: 0, stepLabel: (tid_1)

(**d**) The rule *DriveExit2*: adaptation of the rule *Drive* for the case that a shuttle exits the current fragment topology towards a track belonging to another fragment topology.

Figure 2.2: The rule *Drive* and the three adapted rules *DriveEnterFast*, *DriveExit1*, and *DriveExit2* for fragment topologies where parts of the application condition of the rule *Drive* are omitted due to the overlay specification of the running example

enabled. Secondly, we need to verify whether the rules have been constructed in a way ensuring the absence of collisions between shuttles (i.e., two shuttles should not be on a common track). Thirdly, emergency brakes should be improbable at a local level for a single shuttle but also at the global level for the entire LST and its possible numerous number of shuttles.

3 Preliminaries

We now briefly introduce the subsequently required details for graph transformation systems (GTSs) [10], probabilistic timed automata (PTA) [17], and probabilistic timed graph transformation systems (PTGTSs) [18, 19] in our notation. Along this presentation, we also discuss the modeling details for our running example from the previous chapter.

We employ type graphs (cf. [10]) such as the type graph *TG* from Figure 2.1a 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 specified using a many sorted first-order attribute logic as proposed in [20] (the attribute constraint $\bot$ (false) in *TG* means that the type graph does not restrict attribute values). This approach to attribution has been used to capture constraints on attributes in graph conditions in [26] and to describe attribute modifications in [21, 27].

Graph transformation is then performed by applying a graph transformation rule (short, rule) $\rho = (\ell : K \hookrightarrow L, r : K \hookrightarrow R)$ consisting of two monomorphisms (i.e., all components of the morphisms are injective). The rule specifies that the graph elements in $L - \ell(K)$ are to be deleted, the graph elements in K are to be preserved, and the graph elements in $R - r(K)$ are to be added during graph transformation. Such a rule is applied to a graph G for a given match $m : L \hookrightarrow G$ resulting in a graph G'' by constructing the double pushout (DPO) diagram (see Figure 2.1c) where the first and the second pushout squares describe the removal and the addition of graph elements specified in the rule, respectively. Moreover, a rule may additionally contain an application condition ϕ (denoted by $\rho = (\ell, r, \phi)$) to rule out certain matches specifying e.g. graph elements that may not be connected to graph elements matched by m. For further details on the graph transformation approach, we refer to [10].

PTA [17] combine the use 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 Figure 2.1d consists of (a) a set of locations with a distinguished initial location such as ℓ_0, (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 $c_0 \leq 3$, and (e) a set of probabilistic timed edges each consisting of (e1) a single source location, (e2) at least one target location, (e3) a clock constraint such as $c_0 \geq 2$ specifying as a guard when the edge is enabled based on the current values of the clocks, (e4) for each target location a probability such as 0.5 that this target is reached (the sum of all the probabilities for the target locations of the edge must add up to 1 as a probability distribution is required), and (e5) for each target location a set of clocks such as $\{c_0\}$ to be reset to 0 when that target location is reached.

States of a PTA are given by pairs (ℓ, v) where ℓ is a location and v is the variable valuation mapping each clock of the PTA to a real number. Nondeterminism arises in PTA since a step for advancing time as well as multiple steps applying rules may be enabled in a single state. The logic PTCTL [17] then allows to specify properties such as "what is the worst-case probability that the PTA reaches a location labeled with the AP *done* within 5 time units", which can be analyzed by the PRISM model checker [16]. For the example PTA from Figure 2.1d, the given condition is satisfied with probability 0.75 since the nondeterminism of the PTA would be resolved (by a so-called adversary) such that the PTA first takes a step to ℓ_1 without letting time pass and then performs the probabilistic step (up to two times after waiting for not longer than 2 time units) until it reaches the location ℓ_2 labeled with the AP *done* (the probabilistic step cannot be taken a third time due to the requirement of at most 5 time units in the quoted property above).

PTGTSs have been introduced in [18, 19] as a probabilistic real-time extension of GTSs. It has been shown that PTGTSs can be translated to PTA and, hence, PTGTSs can be understood as a high-level language for PTA as discussed below in more detail and can be analyzed using PRISM as well.

Similarly to PTA, a PTGTS state is given by a pair (G, v) of a graph and a clock valuation. The initial state is given by a distinguished initial graph and a valuation setting all clocks to 0. In our running example, each attribute of type *clockDrive* of a *Track* node (cf. Figure 2.1a) represents one clock. Invariants and APs are specified for PTGTSs by means of graph conditions as in Figure 2.1b and Figure 2.1e, respectively, for our running example. We use the single invariant $INV_{driving}$ requiring that shuttles in mode *DRIVE* cannot be on a track longer than the value of their *minDur* (minimal duration) attribute plus 1. Moreover, we consider three APs to specify properties that we want to analyze later on. The AP $AP_{unexpectedVelocity}$ is used to detect graphs in which a shuttle does not have an expected velocity of $[2, 3]$ or $[3, 4]$ time units per track where only the lower end of the interval is stored in the graph in the *minDur* attribute. The AP $AP_{collision}$ is used to detect graphs in which two shuttles are on a common track to capture their collision. Finally, the AP AP_{braked} is used to detect graphs in which a shuttle has just executed an emergency brake.

PTGT rules of a PTGTS then correspond to edges of a PTA and contain (a) a left-hand side graph L, (b) an attribute constraint on the clock attributes contained in L to capture a guard, (c) a natural number describing a priority where higher numbers denote higher priorities, and (d) a nonempty set of tuples of the form $(\ell : K \hookrightarrow L, r : K \hookrightarrow R, \phi, C, p)$ where (ℓ, r, ϕ) is an underlying GT rule with application condition ϕ[1], C is a set of clock attributes contained in L 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. See Figure 2.1f, Figure 2.1g, and Figure 2.2a for three PTGT rules *SetSlow*, *ConstructionSiteBrake*, and *Drive* from our running example where the last

[1]The underlying GT rule may not delete or add clock attributes.

two PTGT rules have a unique underlying GT rule with probability 1 and where the first PTGT rule has a higher priority as well as two underlying GT rules with probabilities 10^{-6} and $1 - 10^{-6}$. For the PTGT rules *ConstructionSiteBrake* and *Drive*, we depict the graphs L, K, and R in a single graph (subsequently called *LKR*-graph) where graph elements to be removed and to be added are annotated with $\ominus$ and $\oplus$, respectively. In the PTGT rule *SetSlow*, no graph elements are removed or added (i.e., the graphs L and R of the underlying GT rules coincide). Nevertheless, for this PTGT rule, we depict the two right-hand side morphisms r_1 and r_2 as they describe PTGT steps with different attribute modifications and probabilities. Also, the PTGT rules *ConstructionSiteBrake* and *Drive* have application conditions, which are depicted left to the $\triangleright$ symbol or above the $\triangledown$ symbol. The attribute preconditions and attribute modifications are given for each PTGT rule in the red box below the *LKR*-graph (or are split into multiple red boxes as for the PTGT rule *SetSlow*). In these attribute preconditions and attribute modifications, unprimed (primed) variables denote the values of attributes before (after) GT rule application. Note that if variables are not changed by the GT rule application, we denote this using the operator *unchanged* (see e.g. Figure 2.1g where $unchanged(minD_1, tid_1, tid_2)$ denotes that the variables $minD_1$, tid_1, and tid_2 remain unchanged). Moreover, further information about the PTGT rule (i.e., the guard and the priority) but also further information about the probabilistic choices (i.e., the sets of clocks to be reset and probabilities) are depicted in gray boxes. Lastly, we also allow to annotate a PTGT step in the induced state space with (a) a name chosen for the probabilistic choice such as *success* and *failure* in Figure 2.1f and (b) the values of the variables contained in the list stepLabel (which may contain variables from L and R).

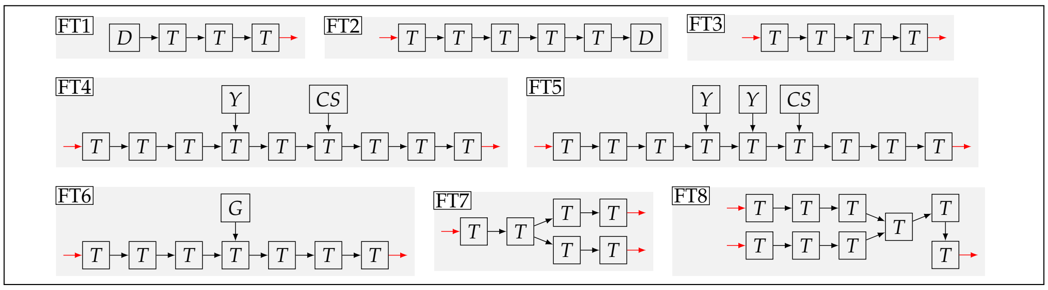
When comparing PTA and PTGTSs, we observe that PTA edges are either enabled for the current valuation or not whereas PTGT rules may be applicable for many matches at the same time (e.g. allowing to apply the *Drive* for one of multiple shuttles). Priorities used in PTGTSs can be encoded in GTSs (including PTGTSs) by adding the left-hand side graphs of rules with higher priorities as negative application conditions to all rules with a lower priority. Similarly, priorities, if integrated into PTA, could be encoded by refining the guards. However, for our running example, we can exchange the underlying track topology without effort, while this would require a fundamental adaptation of the corresponding PTA. Also, as in [19], we observe in chapter 6 that small PTGTSs result in PTA of considerable size and we therefore conclude that PTGTSs are typically much more concise compared to PTA.

4 Decomposition of Large-Scale Topologies

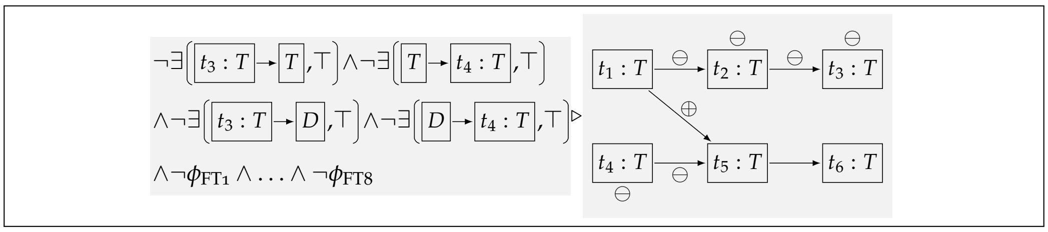
We now present our decomposition-based approach to analyze a PTGTS $\mathcal{S}_0$ modeling a large-scale cyber-physical system along the lines of the informal presentation from the introduction. For our running example, such a PTGTS is given by an initial graph typed over the type graph from Figure 2.1a that is restricted later on in a suitable way, 13 PTGT rules of which we present three in Figure 2.1f, Figure 2.1g, and Figure 2.2a (further rules are given in Appendix B), the invariant from Figure 2.1b, and the three APs from Figure 2.1e.

As a first step, we identify a substructure of the initial graph of $\mathcal{S}_0$ that is static in the sense that this substructure is preserved and also never extended throughout all PTGT steps of $\mathcal{S}_0$. For large-scale cyber-physical systems such as our running example, the existence of such a static substructure may be justified by a logical or spatial distribution. The embedding of a static substructure $\overline{G}$ in a given graph G is then captured by a monomorphism $\kappa : \overline{G} \hookrightarrow G$ describing how $\overline{G}$ is embedded into G. As a special case, such an embedding κ can be derived for arbitrary graphs G by a monomorphism $\kappa_{TG} : \overline{TG} \hookrightarrow TG$ describing how the given type graph TG is restricted to a smaller type graph $\overline{TG}$. That is, $\overline{G}$ then contains only those elements from G that are typed over the smaller type graph $\overline{TG}$. For our running example, we restrict the type graph TG from Figure 2.1a to such a smaller type graph $\overline{TG}$ by removing the *Shuttle* node with its attributes, the *at* edge connected to the *Shuttle* node, and the *active* attributes from the *TLYellow* and *TLGreen* nodes. The graphs $\overline{G}$ obtained from graphs G of $\mathcal{S}_0$ using this restriction are then called *large-scale topologies* (*LSTs*) and contain for our running example a track topology with depots, traffic lights, and construction sites. Note that the fact that such an underlying LST is indeed preserved and never extended by arbitrary rule applications can be verified (at least for our running example) by inspecting each rule individually using the technique of 1-induction [9, 25].

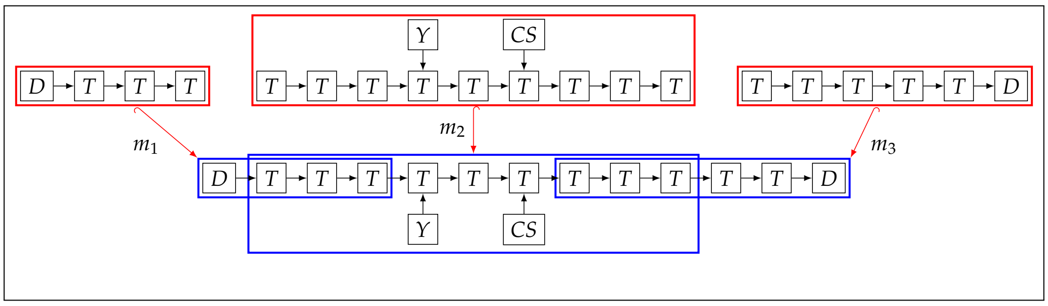
As a second step, we now introduce the notion of a decomposition of the LST into a small set of (constrained) *fragment topologies* (*FTs*). Such (constrained) FTs are given by (a) a graph that is typed over the type graph used for the LST and (b) a graph condition describing constraints on how the graph of the FT may be embedded into graphs of $\mathcal{S}_0$. Moreover, an *overlapping specification o* is required to describe how the *embeddings α_i* of the graphs of two FTs may overlap in the LST. Such an overlapping specification is given by a set of spans $(o_1 : O \hookrightarrow \overline{T}_1, o_2 : O \hookrightarrow \overline{T}_2)$ where O is the *permitted overlapping graph* that is embedded into the two FTs. A decomposition of an LST (in the following definition, we simply consider the LST contained in the initial graph G_0 of $\mathcal{S}_0$) is then given by embeddings of selected FTs



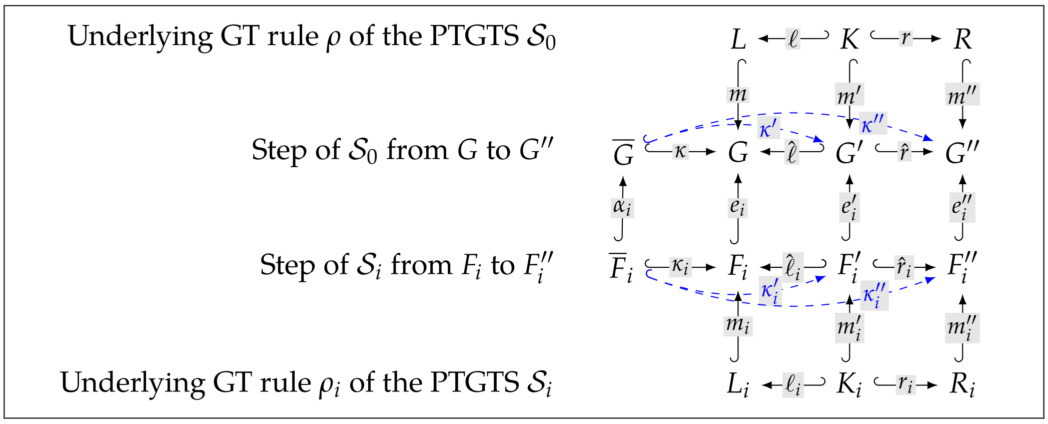
(**a**) FTs for our running example where the red arrows indicate points for topology (de)composition.


$\neg\exists(t_3 : T \to T, \top) \wedge \neg\exists(T \to t_4 : T, \top)$
$\wedge\neg\exists(t_3 : T \to D, \top) \wedge \neg\exists(D \to t_4 : T, \top)$
$\wedge\neg\phi_{FT1} \wedge \ldots \wedge \neg\phi_{FT8}$
$t_1 : T$
$t_2 : T$
$t_3 : T$
$t_4 : T$
$t_5 : T$
$t_6 : T$
⊖
⊕

(**b**) Rule *Merge* for binary overlapping of two instances of FTs.



(**c**) Decomposition $M = \{m_1, m_2, m_3\}$ of an LST w.r.t. FT1–FT8.


Underlying GT rule ρ of the PTGTS $\mathcal{S}_0$
Step of $\mathcal{S}_0$ from G to G''
Step of $\mathcal{S}_i$ from F_i to F_i''
Underlying GT rule ρ_i of the PTGTS $\mathcal{S}_i$
$L \xleftarrow{\ell} K \xrightarrow{r} R$
m, m', m''
$\overline{G} \xrightarrow{\kappa} G \xleftarrow{\hat{\ell}} G' \xrightarrow{\hat{r}} G''$
κ', κ''
α_i, e_i, e_i', e_i''
$\overline{F}_i \xrightarrow{\kappa_i} F_i \xleftarrow{\hat{\ell}_i} F_i' \xrightarrow{\hat{r}_i} F_i''$
κ_i', κ_i''
m_i, m_i', m_i''
$L_i \xleftarrow{\ell_i} K_i \xrightarrow{r_i} R_i$

(**d**) Correspondence of the graph transformation based steps between the large-scale system $\mathcal{S}_0$ and one of its fragment systems $\mathcal{S}_i$, which are preserving the respective static structure given by $\overline{G}$ and $\overline{F_i}$.

Figure 4.1: FTs for our running example, rule *Merge*, example for topology composition, and correspondence between steps in the large-scale system and a fragment system

$$\begin{tikzcd}
\overline{F}_1 \arrow[dr, hook, "g_1"] \arrow[ddr, hook, "\alpha_1"'] & O \arrow[l, hook', "o_1"'] \arrow[r, hook, "o_2"] & \overline{F}_2 \arrow[dl, hook', "g_2"'] \arrow[ddl, hook', "\alpha_2"] \\
& P \arrow[d, hook, "h"] & \\
& G'_0 &
\end{tikzcd}$$

Figure 4.2: Satisfaction of overlapping specification

into the LST (cf. Figure 1.1) such that the overlapping specification is satisfied (the constraints of the FTs are checked for $\mathcal{S}_0$ later on). In applications, to reduce the state space explosion problem for the model checking phase later on, it is advantageous to employ a low number of small FTs that are strictly constrained and are allowed to overlap in a manageable number of ways.

Definition 1 (Decomposition of LST).
If

- $\mathcal{S}_0$ is a PTGTS with initial state $s_0 = (G_0, v_0)$,
- $\kappa : \overline{G}_0 \hookrightarrow G_0$ is a monomorphism identifying the LST of $\mathcal{S}_0$ contained in G_0,
- $\mathcal{F}$ is a set of (constrained) FTs of the form $(\overline{F}_i, \phi_i)$,
- $o((\overline{F}_1, \phi_1), (\overline{F}_2, \phi_2)) \subseteq \{(o_1, o_2) \mid o_1 : O \hookrightarrow \overline{F}_1, o_2 : O \hookrightarrow \overline{F}_2\}$ is an overlapping specification, which describes how two FTs from $\mathcal{F}$ may overlap,
- M is a list of tuples of the form $(\overline{F}, \phi, \alpha)$ where $(\overline{F}, \phi) \in \mathcal{F}$ and $\alpha : \overline{F} \hookrightarrow \overline{G}_0$,
- the monomorphisms in M respect the overlapping specification o, i.e., (see Figure 4.2) for all $(\overline{F}_1, \phi_1, \alpha_1 : \overline{F}_1 \hookrightarrow \overline{G}_0), (\overline{F}_2, \phi_2, \alpha_2 : \overline{F}_2 \hookrightarrow \overline{G}_0) \in M$ there is some pair $(o_1 : O \hookrightarrow \overline{F}_1, o_2 : O \hookrightarrow \overline{F}_2) \in o((\overline{F}_1, \phi_1), (\overline{F}_2, \phi_2))$ such that for the pushout $(g_1 : \overline{F}_1 \hookrightarrow P, g_2 : \overline{F}_2 \hookrightarrow P)$ of (o_1, o_2) (i.e., the overlapping of $\overline{F}_1$ and $\overline{F}_2$ w.r.t. (o_1, o_2)) there is some $h : P \hookrightarrow G'_0$ such that $\alpha_1 = h \circ g_1$ and $\alpha_2 = h \circ g_2$.

then M is a decomposition of the LST of $\mathcal{S}_0$ w.r.t. κ, $\mathcal{F}$, and o. ♦

To provide a better intuition for this definition, we now present the decomposition of the LST considered for our running example.

Example 1 (Decomposition for Running Example).
Let $\mathcal{F}$ contain the constrained FTs (FT$i, \phi_i$) for $1 \leq i \leq 8$ where each FTi is given in Figure 4.1a (here we use an abbreviated notation where D, T, Y, G, and CS are the obvious abbreviations for the node types of the type graph) and where ϕ_i states in each case that shuttles must have a velocity of $[2, 3]$ or $[3, 4]$ time units per track.[1]

Let $o((\overline{F}_1, \phi_1), (\overline{F}_2, \phi_2))$ be the overlapping specification stating that overlappings $(o_1 : O \hookrightarrow \overline{F}_1, o_2 : O \hookrightarrow \overline{F}_2)$ of two FTs are always (for any of the 8×8 combinations)

[1]For each FT from Figure 4.1a, this constraint can be formalized as a graph condition.

of the form $O = T_1 \rightarrow T_2 \rightarrow T_3$ where T_1 and T_3 are mapped to a *Track* node in $\overline{F}_1$ and $\overline{F}_2$ with an entering and an exiting red arrow by o_1 and o_2, respectively.

An example of a decomposition of an LST employing the previously mentioned FTs and overlapping specification is given in Figure 4.1c where three FTs are embedded into an LST. To be appropriate later on, the decomposition must ensure that all tracks of the LST are covered by embedding morphisms to which *Shuttle* nodes may be connected (e.g. due to *Shuttle* nodes in the initial graph of $\mathcal{S}_0$ or due to connected *Depot* nodes from which *Shuttle* nodes may enter the LST). In fact, the eight chosen FTs limit the reasoning for our running example to LSTs that can be decomposed using these FTs. $\Diamond$

In general, we consider the two use cases: (a) a given PTGTS with underlying LST is to be analyzed and (b) LSTs are to be constructed based on the selected and analyzed FTs. Both use cases are supported but require a different handling. For the use case (a) a parsing of the LST w.r.t. the given FTs and overlapping specification must be performed to obtain a decomposition of the LST. Efficient parsing algorithms have been devised for the special case of hyperedge replacement (HR) grammars (which require that nodes are not deleted) in [6, 7, 8]. A suitable graph transformation based grammar for our running example with 25 rules is given in Appendix D. For the use case (b) in which we need to construct some LST, we may employ node deleting rules. For our running example, consider the rule *Merge* from Figure 4.1b that can be used to iteratively overlap two FTs starting with a disjoint union of copies of FTs. The rule *Merge* overlaps two instances of three successive *Track* nodes following the overlapping specification where the application condition ensures that the rule is applied at entry and exit points also excluding the possibility that the six matched *Track* nodes belong to an instance of FTi using $\neg\phi_{\mathrm{FT}i}$.

5 Overapproximation of Behavior

The decompositions of LSTs introduced in the previous chapter are now used as a foundation to establish a behavioral relationship between a given PTGTS $\mathcal{S}_0$ and n PTGTs $\mathcal{S}_i$ that operate on the instances of FTs that are embedded into the LST of $\mathcal{S}_0$ according to the given LST decomposition.

For this purpose, we extend the structural embeddings given by the α monomorphisms from FTs to the LST in Definition 1 to embeddings of the entire graph (including the static but also the dynamic parts) of a state of some $\mathcal{S}_i$ called *fragment topology state* (*FTS*) into the entire graph of a state of $\mathcal{S}_0$ called *large-scale state* (*LSS*). Consider the left middle square in Figure 4.1d where the embedding α_i together with the FT and LST embeddings κ_i and κ is complemented with an embedding e_i of the FTS F_i into the LSS G. Note that e_i must be an extension of α_i in the sense that the square commutes (i.e., $\kappa \circ \alpha_i = e_i \circ \kappa_i$ is required). Also, $e_i \circ \kappa_i$ must satisfy the constraint ϕ_i of the FT used for $\mathcal{S}_i$.

To simplify our presentation, we assume that the PTGTS $\mathcal{S}_0$ (as in our running example) only employs APs of the form $\exists(f : \emptyset \hookrightarrow P, \top)$, invariants of the form $\neg\exists(f : \emptyset \hookrightarrow P, \top)$, and application conditions in PTGT rules that are conjunctions of graph conditions of the form $\neg\exists(f : \emptyset \hookrightarrow P, \top)$ for some graph P. This restriction simplifies the identification of parts of FTSs and LSSs that are considered for an evaluation of such graph conditions.

As a next step, we present a decomposition relation, which establishes a relationship between $\mathcal{S}_0$ and the PTGTSs $\mathcal{S}_i$ in terms of embedding monomorphisms κ, α_i, e_i, and κ_i for all reachable states of $\mathcal{S}_0$. Moreover, the decomposition relation requires that (a) the timed and discrete steps of $\mathcal{S}_0$ can be mimicked by each affected $\mathcal{S}_i$ and (b) that discrete steps performed by some PTGTS $\mathcal{S}_i$ in isolation on a part of the LST where the FT $\overline{F}_i$ does not overlap with the FT $\overline{F}_j$ of another PTGTS $\mathcal{S}_j$ with $i \neq j$ can be mimicked by $\mathcal{S}_0$. That is, the decomposition relation is a simulation for the steps performed by $\mathcal{S}_0$ and a bisimulation on those steps that are performed in isolation by a single PTGTS $\mathcal{S}_i$. Also, to allow to derive results for $\mathcal{S}_0$ from a model checking based analysis of the PTGTSs $\mathcal{S}_i$, we require a set of APs $\mathcal{A}$ that is part of the APs of $\mathcal{S}_0$ and of each $\mathcal{S}_i$. Based on this set $\mathcal{A}$, the decomposition relation also requires that only those FTSs and LSSs are related that satisfy the same sets of APs in $\mathcal{A}$. For our running example, this set will contain all three APs of $\mathcal{S}_0$ (see Figure 2.1e). Finally, we require that the initial states of $\mathcal{S}_0$ and the n PTGTSs $\mathcal{S}_i$ are covered by the decomposition relation.

Definition 2 (Decomposition Relation).
Given

- (PTGTS FOR LARGE-SCALE SYSTEM) $\mathcal{S}_0$ is a PTGTS with initial LSS $s_0 = (G_0, v_0)$ where the LST is identified via $\kappa_0 : \overline{G}_0 \hookrightarrow G_0$ (and preserved by all steps of the PTGTS),
- (PTGTSs FOR FTs) for each $1 \leq i \leq n$: $\mathcal{S}_i$ is a PTGTS with initial FTS $s_{0,i} = (F_{0,i}, v_{0,i})$ where the underlying FT is identified via $\kappa_i : \overline{F}_{0,i} \hookrightarrow F_{0,i}$ (and preserved by all steps of the PTGTS),
- (PRESERVED ATOMIC PROPOSITIONS) $\mathcal{A}$ is a set of APs contained in each $\mathcal{S}_i$, and
- (DECOMPOSITION OF THE LST) M is a decomposition of size n of the LST of $\mathcal{S}_0$ w.r.t. κ_0, $\mathcal{F} = \{\overline{F}_{0,i} \mid 1 \leq i \leq n\}$, and some overlapping specificiation o (cf. Definition 1).

S is a *decomposition relation* between $\mathcal{S}_0$ and $(\mathcal{S}_1, \ldots, \mathcal{S}_n)$ containing tuples of the form $((G, v), \kappa : \overline{G} \hookrightarrow G, w)$ where (G, v) is a state of $\mathcal{S}_0$, κ identifies the LST of G, and w is a tuple of length n of tuples of the form $(s_i, \overline{F}_i, \phi_i, \alpha_i, \kappa_i, e_i)$ when the following items are satisfied.

1. (ELEMENTS OF DECOMPOSITION RELATION) The relation S explains how the FTS of the PTGTS $\mathcal{S}_i$ is embedded into the LSS of $\mathcal{S}_0$, i.e., (see Figure 4.1d) if $((G, v), \kappa : \overline{G} \hookrightarrow G, w) \in S$ and $((F_i, v_i), \overline{F}_i, \phi_i, \alpha_i, \kappa_i, e_i)$ is the ith element of w, then $s_i = (F_i, v_i)$ is a state of $\mathcal{S}_i$, $(\overline{F}_i, \phi_i, \alpha_i)$ is the ith element of M, $\kappa_i : \overline{F}_i \hookrightarrow G'$ (embedding of FT into LST), $e_i : F_i \hookrightarrow G$ (embedding of FTS into LSS), $e_i \circ \kappa_i$ satisfies ϕ_i, and $\kappa \circ \alpha_i = e_i \circ \kappa_i$ (embedding e_i is an extension of embedding κ_i),
2. (CONSISTENT VALUATIONS) The clock valuations of each FTS agree with the LSS, i.e., if $((G, v), \kappa : \overline{G} \hookrightarrow G, w) \in S$, $((F_i, v_i), \overline{F}_i, \phi_i, \alpha_i, \kappa_i, e_i) \in w$, and $\kappa_i(c_i) = c$, then $v_i(c_i) = v(c)$.
3. (INITIAL STATES RELATED) The initial LSS of $\mathcal{S}_0$ is related, i.e., $(s_0, \kappa_0, w) \in S$ for some w where the ith element $(s_i, \overline{F}_i, \phi_i, \alpha_i, \kappa_i, e_i)$ of w satisfies $s_i = s_{0,i}$.
4. (ATOMIC PROPOSITIONS) The labeling with APs is in agreement w.r.t. $\mathcal{A}$, i.e., if $((G, v), \kappa : \overline{G} \hookrightarrow G, w) \in S$, $ap = \exists(f : \emptyset \hookrightarrow P, \top) \in \mathcal{A}$, the monomorphism $k : P \hookrightarrow G$ shows that ap is satisfied by G, then there is some $1 \leq i \leq n$ such that $((F_i, v_i), \overline{F}_i, \phi_i, \alpha_i, \kappa_i, e_i)$ is the ith element of w, and there is some $k_i : P \hookrightarrow F_i$ showing that ap is satisfied by F_i and $k = e_i \circ k_i$.
5. (BISIMULATION OF TIMED STEPS) If $((G, v), \kappa : \overline{G} \hookrightarrow G, w) \in S$ and $\mathcal{S}_0$ has a timed step (not involving a PTGTS rule) from (G, v) to $(G, v + \delta)$ then there is some $((G, v + \delta), w') \in S$ where w' is obtained pointwise from w by applying the corresponding timed step to each $((F_i, v_i), \overline{F}_i, \phi_i, \alpha_i, \kappa_i, e_i) \in w$ resulting in $((F_i, v_i + \delta), \overline{F}_i, \phi_i, \alpha_i, \kappa_i, e_i)$ and vice versa for a common timed step of each $\mathcal{S}_i$ of duration δ.

6. (SIMULATION OF STRUCTURAL STEPS OF $\mathcal{S}_0$ BY $\mathcal{S}_i$) if
   - $((G, v), \kappa : \overline{G} \hookrightarrow G, w) \in S$ and
   - $\mathcal{S}_0$ performs the structural step from (G, v) to (G'', v'') using an underlying GT rule $\rho = (\ell : K \hookrightarrow L, r : K \hookrightarrow R, \phi_{ac})$ given in Figure 4.1d where, since the step of $\mathcal{S}_0$ preserves the LST, there are unique $\kappa' : \overline{G} \hookrightarrow G'$ and $\kappa'' : \overline{G} \hookrightarrow G''$ such that $\hat{\ell} \circ \kappa' = \kappa$ and $\kappa'' = \hat{r} \circ \kappa'$, then
   - $((G'', v''), \kappa'' : \overline{G} \hookrightarrow G, w'') \in S$ for some w'' that is obtained pointwise from w by adapting each tuple $((F_i, v_i), \overline{F}_i, \phi_i, \alpha_i, \kappa_i, e_i) \in w$ into a resulting tuple $((F''_i, v''_i), \overline{F}_i, \phi_i, \alpha_i, \kappa''_i, e''_i)$ as follows. If $m(L) \cap e_i(F_i) = \emptyset$, then all components of the tuple remain unchanged. Otherwise, the PTGTS $\mathcal{S}_i$ must simulate the step and the tuple needs the updating described in the following steps.
     - There must be a step of $\mathcal{S}_i$ as given in Figure 4.1d from F_i to F''_i for some underlying rule $\rho_i = (\ell_i : K_i \hookrightarrow L_i, r_i : K_i \hookrightarrow R_i, \phi_{ac,i})$ with the same probability and priority as ρ.
     - Since the step of $\mathcal{S}_i$ preserves the FT, there are unique $\kappa'_i : \overline{F}_i \hookrightarrow F'_i$ and the required $\kappa''_i : \overline{F}_i \hookrightarrow F''_i$ such that $\hat{\ell}_i \circ \kappa'_i = \kappa_i$ and $\kappa''_i = \hat{r}_i \circ \kappa'_i$.
     - The step of $\mathcal{S}_i$ must allow for $e'_i : F'_i \hookrightarrow G'$ and $e''_i : F''_i \hookrightarrow G''$ such that $\hat{\ell} \circ e'_i = e_i \circ \hat{\ell}_i$ and $\hat{r} \circ e'_i = e''_i \circ \hat{r}_i$.

7. (SIMULATION OF STRUCTURAL STEPS OF $\mathcal{S}_i$ ON ITS CORE BY $\mathcal{S}_0$) if
   - $((G, v), \kappa : \overline{G} \hookrightarrow G, w) \in S$,
   - $((F_i, v_i), \overline{F}_i, \phi_i, \alpha_i, \kappa_i, e_i) \in w$,
   - $\mathcal{S}_i$ performs the structural step from (F_i, v_i) to (F''_i, v''_i) using an underlying GT rule $\rho_i = (\ell_i : K_i \hookrightarrow L_i, r_i : K_i \hookrightarrow R_i, \phi_{ac,i})$ given in Figure 4.1d where, since the step of $\mathcal{S}_i$ preserves the FT, there are unique $\kappa'_i : \overline{F}_i \hookrightarrow F'_i$ and $\kappa''_i : \overline{F}_i \hookrightarrow F''_i$ such that $\hat{\ell}_i \circ \kappa'_i = \kappa_i$ and $\kappa''_i = \hat{r}_i \circ \kappa'_i$,
   - $e_i(m_i(L_i))$ does not overlap with any $e_j(F_j)$ for $i \neq j$, then
   - there is some $((G'', v''), \kappa'' : \overline{G} \hookrightarrow G, w'') \in S$ for some G'', v'', κ'', and w'' as follows.
     - There must be a step of $\mathcal{S}_0$ as given in Figure 4.1d from G to G'' for some underlying rule $\rho = (\ell : K \hookrightarrow L, r : K \hookrightarrow R, \phi_{ac})$ with the same probability and priority as ρ_i.
     - Since the step of $\mathcal{S}_0$ preserves the LST, there are unique $\kappa' : \overline{G} \hookrightarrow G'$ and the required $\kappa'' : \overline{G} \hookrightarrow G''$ such that $\hat{\ell} \circ \kappa' = \kappa$ and $\kappa'' = \hat{r}_i \circ \kappa'$.
     - The step of $\mathcal{S}_0$ must allow for $e'_i : F'_i \hookrightarrow G'$ and $e''_i : F''_i \hookrightarrow G''$ such that $\hat{\ell} \circ e'_i = e_i \circ \hat{\ell}_i$ and $\hat{r} \circ e'_i = e''_i \circ \hat{r}_i$.
     - Finally, w'' is obtained from w by only adapting the above chosen tuple $((F_i, v_i), \overline{F}_i, \phi_i, \alpha_i, \kappa_i, e_i)$ into the tuple $((F''_i, v''_i), \overline{F}_i, \phi_i, \alpha_i, \kappa''_i, e''_i)$.

◆

We now state that decomposition relations allow for the simulation of each path of the PTGTS $\mathcal{S}_0$ by the PTGTSs $\mathcal{S}_i$.

Lemma 1 (Existence of Simulating Paths).
If S is a decomposition relation between $\mathcal{S}_0$ and $(\mathcal{S}_1, \dots, \mathcal{S}_n)$, and π is a path of length m in $\mathcal{S}_0$ from the initial state to a state s_m, then, for each $1 \leq i \leq n$, there is a path π_i of $\mathcal{S}_i$ (of length $k_i \leq m$) ending in a state s_{i,k_i} such that $(s_m, \kappa, w) \in S$ for some κ and w where the ith element of w is of the form $(s_{i,k_i}, \overline{F}_i, \phi_i, \alpha_i, \kappa_i, e_i)$. Moreover, the probability of each such path π_i is at least as high as the probability of the path π.

Proof. By induction on m. Firstly, due to item 5 in Definition 2, each timed step of $\mathcal{S}_0$ results in an additional corresponding timed step of each $\mathcal{S}_i$ to be appended to the path constructed for the PTGTS $\mathcal{S}_i$. Secondly, due to item 6 in Definition 2, each graph transformation based step with priority p of $\mathcal{S}_0$ results in an additional corresponding graph transformation of each affected $\mathcal{S}_i$. Moreover, due to item 7 in Definition 2, if there would be a graph transformation based step with a priority $p_i > p$ enabled in $\mathcal{S}_i$, there would also be a step with the same priority in $\mathcal{S}_0$, which would be a contradiction to the existence of the step with priority p from before. Also, in both cases, the resulting states are in the decomposition relation. Lastly, since $\mathcal{S}_i$ applies rules with the same probability as $\mathcal{S}_0$ (but not all steps of $\mathcal{S}_0$ are simulated by $\mathcal{S}_i$), π_i has at least the probability of π. □

We now state that a PTGTS satisfies a safety property given by an AP, when safety w.r.t. this AP can be established for each $\mathcal{S}_i$.

Theorem 1 (Safety Verification).
If S is a decomposition relation between $\mathcal{S}_0$ and $(\mathcal{S}_1, \dots, \mathcal{S}_n)$ w.r.t $\mathcal{A}$ and $ap \in \mathcal{A}$, then $\mathcal{S}_0$ is safe w.r.t. the occurrence of an *ap*-labeled graph when (for each $1 \leq i \leq n$) $\mathcal{S}_i$ is safe w.r.t. the occurrence of an *ap*-labeled graph. Moreover, the probability of an occurrence of an *ap*-labeled graph from some state s in $\mathcal{S}_0$ is smaller than the probability of an occurrence of an *ap*-labeled graph from some S-related state s_i in $\mathcal{S}_i$.

Proof. By contraposition. Assume that $\mathcal{S}_0$ is not safe by having a reachable state labeled with $ap \in \mathcal{A}$. Let π be a path of length m in $\mathcal{S}_0$ from the initial state to such a state s_m. By applying Lemma 1, we are able to construct for each $1 \leq i \leq n$ a path π_i of $\mathcal{S}_i$ (of length $k_i \leq m$) ending in a state $s_{i,m}$ such that at least one of these states is also labeled by *ap* due to item 4 in Definition 2. However, the existence of such a path π_i then contradicts the safety of $\mathcal{S}_i$. The upper bound on the probabilistic occurrence of an *ap*-labeled graph is then a direct consequence of Lemma 1. □

We now apply the proposed methodology of establishing a behavioral relationship between the PTGTS $\mathcal{S}_0$ and the PTGTSs $\mathcal{S}_i$ to our running example. For this purpose, we now describe how the FTS of each $\mathcal{S}_i$ is embedded into the LSS of $\mathcal{S}_0$ and, based on this embedding, how the $\mathcal{S}_i$ is derived from $\mathcal{S}_0$.

Example 2 (Construction of Embeddings and Simulating PTGTSs).
Firstly, the embeddings e_i of FTSs into the LSS are obtained as extensions of the structural embeddings κ_i by also matching (a) all *Shuttle* nodes (with their attributes)

that are connected to *Track* nodes contained in the FT via *next* edges and (b) all *active* attributes of *TLYellow* and *TLGreen* nodes contained in the FT. This extension also naturally applies to the initial state of $\mathcal{S}_0$. Clearly, two embeddings e_i and e_j (for $i \neq j$) only overlap in elements of their FTs but not in the additionally matched dynamic elements.

Secondly, we adapt the given PTGTS $\mathcal{S}_0$ to obtain for each of the eight FTs one PTGTS $\mathcal{S}_i$ by (a) changing the initial graph to the source of e_i capturing the FT as well as the additional dynamic elements of the initial state of $\mathcal{S}_0$ connected to it, (b) adding eight rules for overapproximating the behavior of $\mathcal{S}_0$ on the tracks that may overlap with tracks of other FTs. For the latter point, we observe that all but three of the rules of $\mathcal{S}_0$ (including *SetSlow* and *ConstructionSiteBrake* from Figure 2.1) are never applicable on the parts of FTs that may overlap with other FTs (i.e., borders of FTs). The remaining three rules are *Drive* from Figure 2.2a as well as two similar rules for stopping the shuttle that we do not consider in detail here. Three of the four derived rules for rule *Drive* are given in Figure 2.2.

The additional rule *DriveEnterFast* is used to simulate *Drive* steps where a shuttle in $\mathcal{S}_0$ drives from a track not covered by $\mathcal{S}_i$ to a track covered by $\mathcal{S}_i$. The rule *DriveEnterFast* is essentially constructed by omitting the source track T_1 from the rule *Drive*, by adding the shuttle with one of the two expected velocities (the other velocity results in the omitted rule *DriveEnterSlow*)[1], and by omitting application conditions that may not be satisfied due to the overlapping specification and the structure of FTs.

Similarly, the additional rules *DriveExit1* and *DriveExit2* are constructed from rule *Drive* to allow for the simulation of the two steps in which a shuttle in $\mathcal{S}_0$ drives using rule *Drive* on two tracks covered by $\mathcal{S}_i$ to a track not covered by $\mathcal{S}_i$. These two rules are then constructed similarly, by omitting the tracks T_3 (for *DriveExit1*) and T_3 and T_4 (for *DriveExit2*) from rule *Drive* as these are not covered by the $\mathcal{S}_i$, by removing the shuttle with its attributes in rule *DriveExit2*, by omitting application conditions that may not be satisfied due to the overlapping specification and the structure of FTs, and by omitting application conditions that refer to the removed tracks.

Note that these additional rules overapproximate the behavior that is possible in $\mathcal{S}_0$ as they may be used when analyzing $\mathcal{S}_i$ also when no corresponding shuttle in $\mathcal{S}_0$ is able to enter the FT or when rule *Drive* would be disabled due to the omitted application conditions for the case of rules *DriveExit1* and *DriveExit2*. $\diamondsuit$

For our running example, we now describe the construction of a suitable decomposition relation relying on the LST decomposition introduced before.

Lemma 2 (Existence of Decomposition Relation for Running Example).
For the PTGTS $\mathcal{S}_0$ of our running example with an arbitrary initial LST such that M is a decomposition of that LST w.r.t. some monomorphism κ, the set of eight FTs, and the overlapping specification o from Example 1 there is a decomposition relation S between $\mathcal{S}_0$ and the n PTGTSs $\mathcal{S}_i$ from Example 2.

[1]Here, we rely on the constraints on the eight FTs (cf. Example 1) requiring that the AP $AP_{unexpectedVelocity}$ is never labeled in the large-scale system $\mathcal{S}_0$.

Proof. We construct S to contain all tuples (a) that use embeddings as described in Example 2 and (b) that contain consistent clock valuations among the PTGTSs. The relation S satisfies the requirement on APs as all APs of $\mathcal{S}_0$ only match a single track and, hence, such a match can be obtained analogously in some $\mathcal{S}_i$ in each case. The relation S satisfies the requirement of timed steps by assumption (b) on consistent clock valuations above. The relation S satisfies the simulation of discrete steps of $\mathcal{S}_0$ by the affected $\mathcal{S}_i$ since all rules on the non-overlapped parts of the FT can be performed in $\mathcal{S}_i$ using the same rules and matches and because each further step of $\mathcal{S}_0$ matching tracks that are jointly covered by $\mathcal{S}_i$ and $\mathcal{S}_j$ can be mimicked by $\mathcal{S}_i$ and $\mathcal{S}_j$ using the additional rules as discussed in Example 2. Also, the additional rules have all the priority 0 not preventing the required rule applications on the core. Lastly, the relation S satisfies the simulation of discrete steps of $\mathcal{S}_i$ on its core because $\mathcal{S}_i$ contains the rules of $\mathcal{S}_0$. □

Based on this decomposition relation and Theorem 1, we can obtain the desired overapproximation result for $\mathcal{S}_0$ for the qualitative safety w.r.t. collisions and the quantitative unlikeliness of emergency brakes.

Corollary 1 (Qualitative and Quantitative Safety for Running Example)**.**
$\mathcal{S}_0$ exhibits no collisions when this is the case for each $\mathcal{S}_i$. Moreover, emergency brakes are performed in $\mathcal{S}_0$ with a probability not higher than the probability of such an occurrence in any $\mathcal{S}_i$.

Note that we only need to analyze one PTGTS for each of the eight permitted FTs w.r.t. the occurrence of collisions and the probability of emergency brakes.

6 Evaluation

To analyze the eight PTGTSs constructed for our running example in chapter 5 (see Table 6.1 for the results), we have employed the methodology from [19] generating the state spaces for these PTGTSs without timed steps and then generated the corresponding PTA from these state spaces. We then restricted these PTA to timed automata (TA) essentially removing the information on probabilities, applied UPPAAL [15] to determine the edges of the TA that can never be applied due to unsatisfiable guards, and removed the corresponding edges from the previously generated PTA. The entire analysis using our prototypical implementation required less than three days on a machine using up to 250 GB memory where the state space generation required most of the time. However, there is a vast potential for optimizations regarding memory consumption (by only storing subsequently relevant information on states and steps) and runtime (by facilitating concurrency during state space generation).

Firstly, using UPPAAL, we have verified that each of the eight TA (hence, also the eight PTA) have no reachable deadlock (where also timed steps are disabled). Hence, we obtain that the PTGTS $\mathcal{S}_0$ also does not contain this particular modeling error since, using the decomposition relation, we also obtain that every deadlock reachable in $\mathcal{S}_0$ can be reached analogously in each $\mathcal{S}_i$.

Secondly, we have observed that the obtained PTA do not label any location with $AP_{unexpectedVelocity}$ or $AP_{collision}$. For $AP_{unexpectedVelocity}$ this means that the additional rules such as *DriveEnterFast* and *DriveEnterSlow* for overapproximating the steps of entering shuttles entirely cover all possible velocities of shuttles. For $AP_{collision}$ this means that Corollary 1 implies that the PTGTS $\mathcal{S}_0$ with an LST constructed in the described way from the eight FTs is safe w.r.t. the occurrence of collisions.

Thirdly, to verify that yellow traffic lights suitably slow down the shuttles before construction sites, we have identified locations ℓ_i in the resulting PTA that are labeled with AP_{braked} (occurring only in FT4 and FT5). In each case, we were able to track using a custom analysis algorithm (since the PRISM model checker was too slow for the large PTA at hand) the shuttle backwards over all possible paths leading to such a location ℓ_i up to the step where the shuttle entered the FT. We then determined the maximal probability of any such path obtaining a worst-case emergency brake probability of 10^{-6} and 10^{-12} for any entering shuttle in FT4 and FT5, respectively. On the one hand, FT5 is thereby verified to be quantitatively more desirable compared to FT4. On the other hand, Corollary 1 implies that installations of yellow traffic lights as in FT4 and FT5 suitably decrease the likelihood of emergency brakes also for $\mathcal{S}_0$. 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

Table 6.1: Results of our evaluation for the running example

fragment topology	states	steps	collisions	max. probability for violating the velocity limit at a construction site
FT1	9	18	0	0
FT2	335	693	0	0
FT3	216	503	0	0
FT4	109 379	312 915	0	1×10^{-6}
FT5	106 122	284 102	0	1×10^{-12}
FT6	12 473	31 812	0	0
FT7	4048	16 314	0	0
FT8	121 953	452 340	0	0

to be too coarse upper bounds when the maximal throughput is not to be expected for the real system.

7 Conclusion and Future Work

We presented an analysis approach for large-scale systems modeled as PTGTSs for which model checking is not feasible. In this approach, we rely on a decomposition of an underlying static large-scale topology into fragment topologies of manageable size. Model checking is then applied for each fragment topology and an adaptation of the PTGTS to such a fragment topology. We thereby determine (a) overapproximations of reachability properties important for qualitative safety properties and (b) upper bounds for probabilistic reachability properties important for quantitative safety properties.

As future work, we intend to extend our analysis to fairness properties and conditions of the metric temporal graph logic (MTGL) [28]. Also, to cover further aspects of the RailCab project [22], we will develop more general decomposition schemes where dynamic components (such as connected shuttles driving in convoys) may be covered by multiple fragment topologies. Lastly, to further evaluate applicability of our approach, we intend to apply it to other case studies as e.g. the one discussed in [1].

Bibliography

[1] P. Baldan, A. Corradini, and B. König. “Static Analysis of Distributed Systems with Mobility Specified by Graph Grammars—A Case Study”. In: *Proc. of Int. Conf. on Integrated Design & Process Technology*. Edited by Ehrig, Krämer, et al. SDPS, 2002.

[2] B. Becker. “Architectural modelling and verification of open service-oriented systems of systems”. PhD thesis. Hasso-Plattner-Institut für Softwaresystemtechnik, Universität Potsdam, 2014.

[3] B. Becker, D. Beyer, H. Giese, F. Klein, and D. Schilling. “Symbolic invariant verification for systems with dynamic structural adaptation”. In: *28th International Conference on Software Engineering (ICSE 2006), Shanghai, China, May 20-28, 2006*. Edited by L. J. Osterweil, H. D. Rombach, and M. L. Soffa. ACM, 2006, pages 72–81. DOI: 10.1145/1134285.1134297.

[4] B. Becker and H. Giese. “On Safe Service-Oriented Real-Time Coordination for Autonomous Vehicles”. In: *11th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC 2008), 5-7 May 2008, Orlando, Florida, USA*. IEEE Computer Society, 2008, pages 203–210. DOI: 10.1109/ISORC.2008.13.

[5] B. Becker, H. Giese, and S. Neumann. *Correct dynamic service-oriented architectures : modeling and compositional verification with dynamic collaborations*. Technical report 29. Hasso Plattner Institute at the University of Potsdam, 2009.

[6] F. Drewes, B. Hoffmann, and M. Minas. “Formalization and correctness of predictive shift-reduce parsers for graph grammars based on hyperedge replacement”. In: *J. Log. Algebraic Methods Program.* 104 (2019), pages 303–341. DOI: 10.1016/j.jlamp.2018.12.006.

[7] F. Drewes, B. Hoffmann, and M. Minas. “Graph Parsing as Graph Transformation—Correctness of Predictive Top-Down Parsers”. In: *Graph Transformation - 13th International Conference, ICGT 2020, Held as Part of STAF 2020, Bergen, Norway, June 25-26, 2020, Proceedings*. Edited by F. Gadducci and T. Kehrer. Volume 12150. Lecture Notes in Computer Science. Springer, 2020, pages 221–238. DOI: 10.1007/978-3-030-51372-6_13.

[8] F. Drewes, B. Hoffmann, and M. Minas. “Predictive Top-Down Parsing for Hyperedge Replacement Grammars”. In: *Graph Transformation - 8th International Conference, ICGT 2015, Held as Part of STAF 2015, L'Aquila, Italy, July 21-23, 2015. Proceedings*. Edited by F. Parisi-Presicce and B. Westfechtel. Volume 9151. Lecture Notes in Computer Science. Springer, 2015, pages 19–34. DOI: 10.1007/978-3-319-21145-9_2.

[9] J. Dyck. “Verification of Graph Transformation Systems with k-Inductive Invariants”. PhD thesis. University of Potsdam, Hasso Plattner Institute, Potsdam, Germany, 2020. DOI: 10.25932/publishup-44274.

[10] H. Ehrig, K. Ehrig, U. Prange, and G. Taentzer. *Fundamentals of Algebraic Graph Transformation*. Springer-Verlag, 2006.

[11] A. H. Ghamarian and A. Rensink. "Generalised Compositionality in Graph Transformation". In: *Graph Transformations - 6th International Conference, ICGT 2012, Bremen, Germany, September 24-29, 2012. Proceedings*. Edited by H. Ehrig, G. Engels, H.-J. Kreowski, and G. Rozenberg. Volume 7562. Lecture Notes in Computer Science. Springer, 2012, pages 234–248. DOI: 10.1007/978-3-642-33654-6_16.

[12] H. Giese. "Modeling and Verification of Cooperative Self-adaptive Mechatronic Systems". In: *Reliable Systems on Unreliable Networked Platforms - 12th Monterey Workshop 2005, Laguna Beach, CA, USA, September 22-24, 2005. Revised Selected Papers*. Edited by F. Kordon and J. Sztipanovits. Volume 4322. Lecture Notes in Computer Science. Springer, 2005, pages 258–280. DOI: 10.1007/978-3-540-71156-8_14.

[13] H. Giese and W. Schäfer. "Model-Driven Development of Safe Self-optimizing Mechatronic Systems with MechatronicUML". In: *Assurances for Self-Adaptive Systems - Principles, Models, and Techniques*. Edited by J. Cámara, R. de Lemos, C. Ghezzi, and A. Lopes. Volume 7740. Lecture Notes in Computer Science. Springer, 2013, pages 152–186. DOI: 10.1007/978-3-642-36249-1_6.

[14] H. Giese, M. Tichy, S. Burmester, and S. Flake. "Towards the compositional verification of real-time UML designs". In: *Proceedings of the 11th ACM SIGSOFT Symposium on Foundations of Software Engineering 2003 held jointly with 9th European Software Engineering Conference, ESEC/FSE 2003, Helsinki, Finland, September 1-5, 2003*. Edited by J. Paakki and P. Inverardi. ACM, 2003, pages 38–47. DOI: 10.1145/940071.940078.

[15] E.-Y. Kang, D. Mu, and L. Huang. "Probabilistic Verification of Timing Constraints in Automotive Systems Using UPPAAL-SMC". In: *Integrated Formal Methods - 14th International Conference, IFM 2018, Maynooth, Ireland, September 5-7, 2018, Proceedings*. Edited by C. A. Furia and K. Winter. Volume 11023. Lecture Notes in Computer Science. Springer, 2018, pages 236–254. DOI: 10.1007/978-3-319-98938-9_14.

[16] M. Z. Kwiatkowska, G. Norman, and D. Parker. "PRISM 4.0: Verification of Probabilistic Real-Time Systems". In: *Computer Aided Verification - 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings*. Edited by G. Gopalakrishnan and S. Qadeer. Volume 6806. Lecture Notes in Computer Science. Springer, 2011, pages 585–591. ISBN: 978-3-642-22109-5. DOI: 10.1007/978-3-642-22110-1_47.

[17] M. Z. Kwiatkowska, G. Norman, J. Sproston, and F. Wang. "Symbolic Model Checking for Probabilistic Timed Automata". In: *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems, Joint International Conferences on Formal Modelling and Analysis of Timed Systems, FORMATS 2004 and Formal Techniques in Real-Time and Fault-Tolerant Systems, FTRTFT 2004, Grenoble, France, September 22-24, 2004, Proceedings*. Edited by Y. Lakhnech and S. Yovine. Volume 3253. Lecture Notes in Computer Science. Springer, 2004, pages 293–308. ISBN: 3-540-23167-6. DOI: 10.1007/978-3-540-30206-3_21.

[18] M. Maximova, H. Giese, and C. Krause. "Probabilistic timed graph transformation systems". In: *Graph Transformation - 10th International Conference, ICGT 2017, Held as Part of STAF 2017, Marburg, Germany, July 18-19, 2017, Proceedings*. Edited by J. de Lara and D. Plump. Volume 10373. Lecture Notes in Computer Science. Springer, 2017, pages 159–175. ISBN: 978-3-319-61469-4. DOI: 10.1007/978-3-319-61470-0_10.

[19] M. Maximova, H. Giese, and C. Krause. "Probabilistic timed graph transformation systems". In: *J. Log. Algebr. Meth. Program.* 101 (2018), pages 110–131. DOI: 10.1016/j.jlamp.2018.09.003.

[20] F. Orejas. "Symbolic graphs for attributed graph constraints". In: *J. Symb. Comput.* 46.3 (2011), pages 294–315. DOI: 10.1016/j.jsc.2010.09.009.

[21] F. Orejas and L. Lambers. “Lazy Graph Transformation”. In: *Fundam. Inform.* 118.1-2 (2012), pages 65–96. DOI: `10.3233/FI-2012-706`.

[22] *RailCab Project*. URL: `https://www.hni.uni-paderborn.de/cim/projekte/railcab`.

[23] A. Rensink. “Compositionality in Graph Transformation”. In: *Automata, Languages and Programming, 37th International Colloquium, ICALP 2010, July 6-10, Bordeaux, France, 2010, Proceedings, Part II*. Edited by S. Abramsky, C. Gavoille, C. Kirchner, F. M. auf der Heide, and P. G. Spirakis. Volume 6199. Lecture Notes in Computer Science. Springer, 2010, pages 309–320. DOI: `10.1007/978-3-642-14162-1_26`.

[24] W. P. de Roever, H. Langmaack, and A. Pnueli, editors. *Compositionality: The Significant Difference, International Symposium, COMPOS'97, Bad Malente, Germany, September 8-12, 1997. Revised Lectures*. Volume 1536. Lecture Notes in Computer Science. Springer, 1998. ISBN: 3-540-65493-3. DOI: `10.1007/3-540-49213-5`.

[25] S. Schneider, J. Dyck, and H. Giese. “Formal Verification of Invariants for Attributed Graph Transformation Systems Based on Nested Attributed Graph Conditions”. In: *Graph Transformation - 13th International Conference, ICGT 2020, Held as Part of STAF 2020, Bergen, Norway, June 25-26, 2020, Proceedings*. Edited by F. Gadducci and T. Kehrer. Volume 12150. Lecture Notes in Computer Science. Springer, 2020, pages 257–275. DOI: `10.1007/978-3-030-51372-6_15`.

[26] S. Schneider, L. Lambers, and F. Orejas. “Automated reasoning for attributed graph properties”. In: *STTT* 20.6 (2018), pages 705–737. DOI: `10.1007/s10009-018-0496-3`.

[27] S. Schneider, M. Maximova, L. Sakizloglou, and H. Giese. “Formal Testing of Timed Graph Transformation Systems using Metric Temporal Graph Logic”. In: *STTT* (2019). Accepted.

[28] S. Schneider, L. Sakizloglou, M. Maximova, and H. Giese. “Optimistic and Pessimistic On-the-fly Analysis for Metric Temporal Graph Logic”. In: *Graph Transformation - 13th International Conference, ICGT 2020, Held as Part of STAF 2020, Bergen, Norway, June 25-26, 2020, Proceedings*. Edited by F. Gadducci and T. Kehrer. Volume 12150. Lecture Notes in Computer Science. Springer, 2020, pages 276–294. DOI: `10.1007/978-3-030-51372-6_16`.

In Appendix A and Appendix B, we present all elements of the PTGTS $\mathcal{S}_0$ of our running example in our full notation. In Appendix C, we provide the eight FTs in our full notation. In Appendix D, we present graph transformation rules for parsing of an LST w.r.t. the FTs from Figure 4.1a and the overlapping specification from Example 1.

A Extended Type Graph of Running Example

For debugging purposes, we employ the extended type graph from Figure A.1.

- We may limit the number of shuttles that may enter an initially empty FT: for this purpose, we add a single *System* node to the FT, add a single *count* attribute to the *System* node, and assign the maximal number of shuttles that should be allowed on the FT to that attribute. We have used the maximal number of 100 shuttles for our experiments, which is no limitation since the FTs have much fewer *Track* nodes and therefore a possible collision is not prevented.

- We may assign unique identifiers (given by *id* attributes) to shuttles when they enter the FT: for this purpose, we add a single *System* node to the FT, add a single *freeIds* attribute to the *System* node, and assign e.g. the string "11111", which is a binary encoding of the already taken shuttle identifiers. The first shuttle will take the identifier 1 and set the *freeIds* attribute to the string "01111". The second shuttle will take the identifier 2 and set the *freeIds* attribute to the string "00111". If the first shuttle then exits the FT, it will release the identifier 1 by setting the *freeIds* attribute to the string "10111". We have used the empty string for our experiments, which results in all shuttles having the same identifier -1.

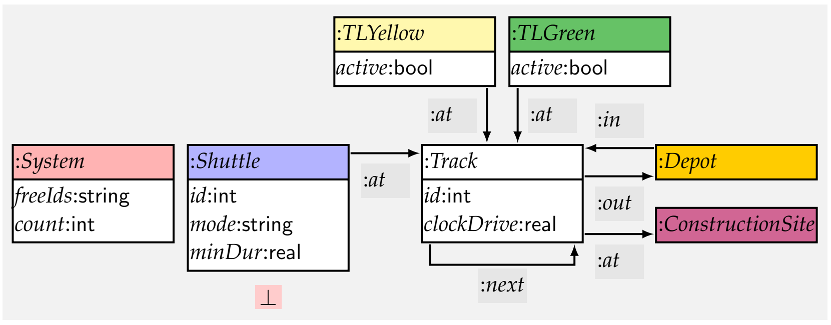


Figure A.1: Extended type graph of running example

B Rules of Running Example

All PTGT rules of our running example are given in Figure B.1, Figure B.2, Figure B.3, Figure B.4, Figure B.5, Figure B.6, Figure B.7, Figure B.8, Figure B.9, Figure B.10, Figure B.11, Figure B.12, Figure B.13, Figure B.14, Figure B.15, Figure B.16, Figure B.17, Figure B.18, Figure B.19, Figure B.20, and Figure B.21.

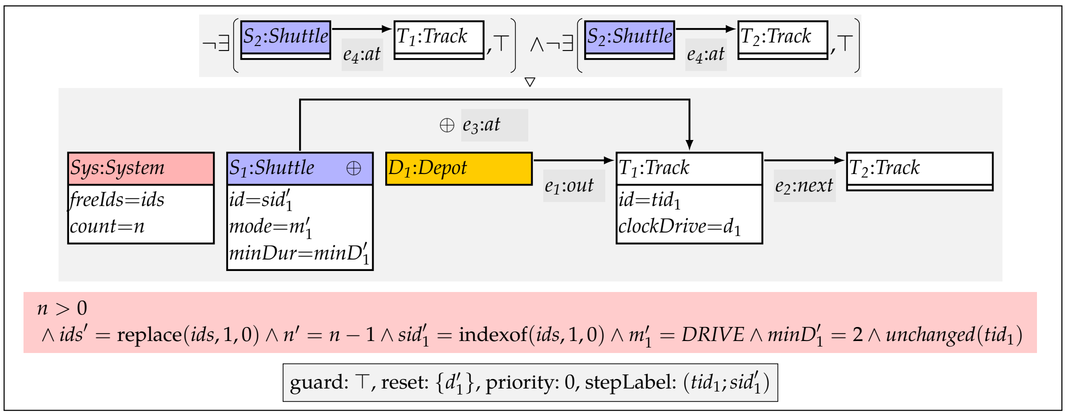

$\neg\exists$ (S_2:Shuttle e_4:at T_1:Track , $\top$) $\wedge\neg\exists$ (S_2:Shuttle e_4:at T_2:Track , $\top$)
$\oplus$ e_3:at
Sys:System
freeIds=ids
count=n
S_1:Shuttle $\oplus$
id=sid_1'
mode=m_1'
minDur=$minD_1'$
D_1:Depot
e_1:out
T_1:Track
id=tid_1
clockDrive=d_1
e_2:next
T_2:Track
$n > 0$
$\wedge\, ids' = \text{replace}(ids, 1, 0) \wedge n' = n - 1 \wedge sid_1' = \text{indexof}(ids, 1, 0) \wedge m_1' = DRIVE \wedge minD_1' = 2 \wedge unchanged(tid_1)$
guard: $\top$, reset: $\{d_1'\}$, priority: 0, stepLabel: $(tid_1; sid_1')$

Figure B.1: The rule *DriveFromDepot*: a shuttle enters the track topology from a depot

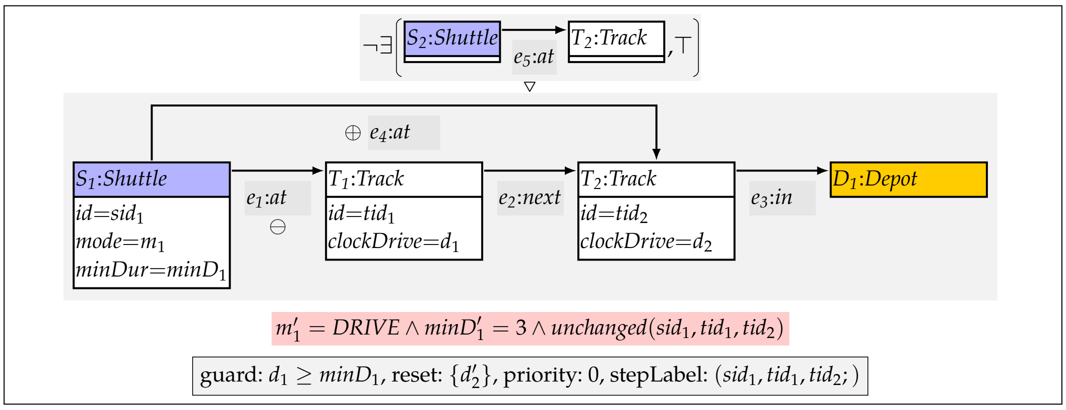

$\neg\exists$ (S_2:Shuttle e_5:at T_2:Track , $\top$)
$\oplus$ e_4:at
S_1:Shuttle
id=sid_1
mode=m_1
minDur=$minD_1$
e_1:at $\ominus$
T_1:Track
id=tid_1
clockDrive=d_1
e_2:next
T_2:Track
id=tid_2
clockDrive=d_2
e_3:in
D_1:Depot
$m_1' = DRIVE \wedge minD_1' = 3 \wedge unchanged(sid_1, tid_1, tid_2)$
guard: $d_1 \geq minD_1$, reset: $\{d_2'\}$, priority: 0, stepLabel: $(sid_1, tid_1, tid_2;)$

Figure B.2: The rule *DriveToDepot1*: a shuttle approaches a depot and slows down if necessary

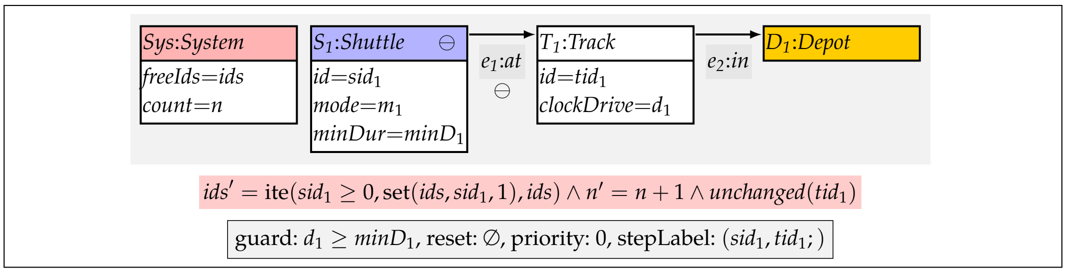

Sys:System
freeIds=ids
count=n
S1:Shuttle ⊖
id=sid1
mode=m1
minDur=minD1
e1:at ⊖
T1:Track
id=tid1
clockDrive=d1
e2:in
D1:Depot
ids′ = ite(sid1 ≥ 0, set(ids, sid1, 1), ids) ∧ n′ = n + 1 ∧ unchanged(tid1)
guard: d1 ≥ minD1, reset: ∅, priority: 0, stepLabel: (sid1, tid1;)

Figure B.3: The rule *DriveToDepot2*: a shuttle exits the track topology to a depot

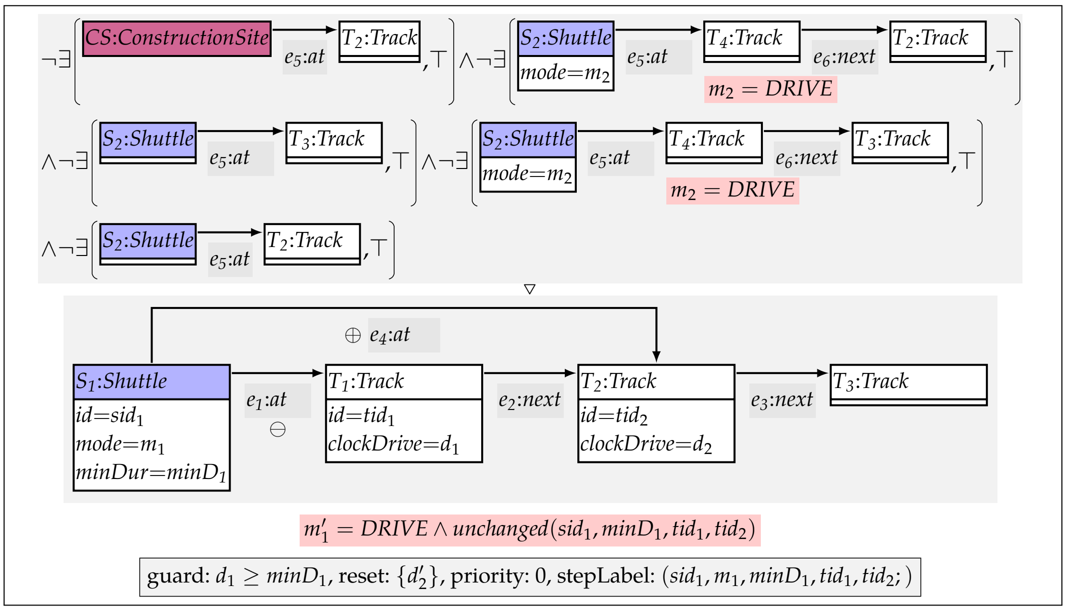

¬∃ CS:ConstructionSite e5:at T2:Track ,⊤
∧¬∃ S2:Shuttle mode=m2 e5:at T4:Track e6:next T2:Track ,⊤ m2 = DRIVE
∧¬∃ S2:Shuttle e5:at T3:Track ,⊤
∧¬∃ S2:Shuttle mode=m2 e5:at T4:Track e6:next T3:Track ,⊤ m2 = DRIVE
∧¬∃ S2:Shuttle e5:at T2:Track ,⊤
▽
⊕ e4:at
S1:Shuttle
id=sid1
mode=m1
minDur=minD1
e1:at ⊖
T1:Track
id=tid1
clockDrive=d1
e2:next
T2:Track
id=tid2
clockDrive=d2
e3:next
T3:Track
m′1 = DRIVE ∧ unchanged(sid1, minD1, tid1, tid2)
guard: d1 ≥ minD1, reset: {d′2}, priority: 0, stepLabel: (sid1, m1, minD1, tid1, tid2;)

Figure B.4: The rule *Drive*: the most commonly used rule allowing a shuttle to drive

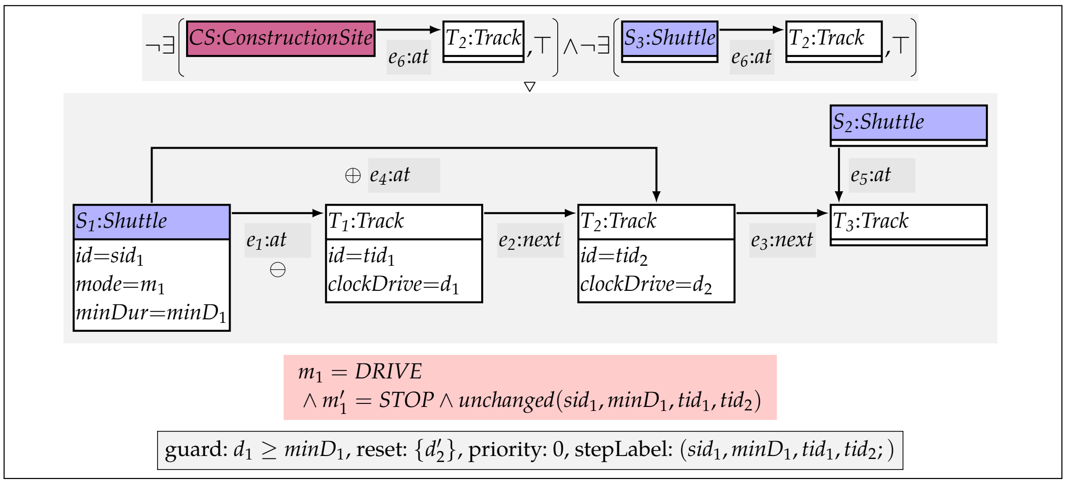
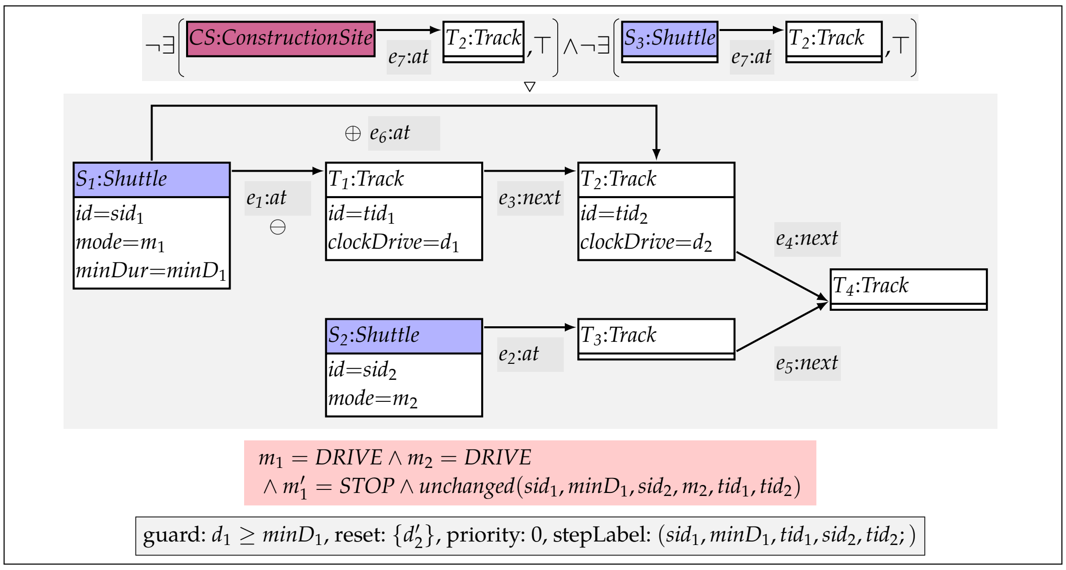
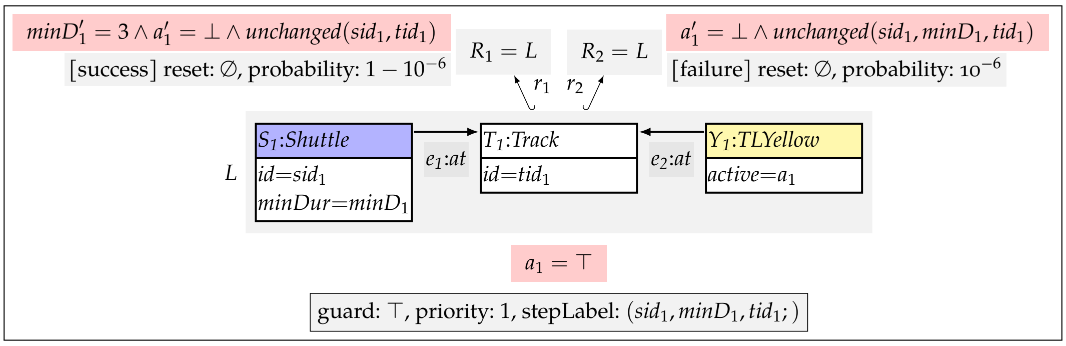

¬∃ CS:ConstructionSite e6:at T2:Track ,⊤
∧¬∃ S3:Shuttle e6:at T2:Track ,⊤
▽
S2:Shuttle
e5:at
⊕ e4:at
S1:Shuttle
id=sid1
mode=m1
minDur=minD1
e1:at ⊖
T1:Track
id=tid1
clockDrive=d1
e2:next
T2:Track
id=tid2
clockDrive=d2
e3:next
T3:Track
m1 = DRIVE
∧ m′1 = STOP ∧ unchanged(sid1, minD1, tid1, tid2)
guard: d1 ≥ minD1, reset: {d′2}, priority: 0, stepLabel: (sid1, minD1, tid1, tid2;)

Figure B.5: The rule *Stop1*: a shuttle stops due to a shuttle too close ahead



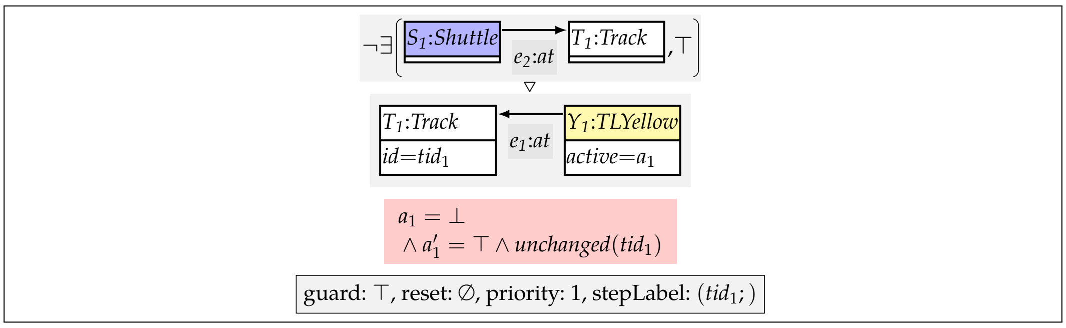
$\neg\exists$ CS:ConstructionSite e_7:at T_2:Track $\top$ $\wedge\neg\exists$ S_3:Shuttle e_7:at T_2:Track $\top$
$\oplus$ e_6:at
S_1:Shuttle id=sid_1 mode=m_1 minDur=$minD_1$
e_1:at $\ominus$
T_1:Track id=tid_1 clockDrive=d_1
e_3:next
T_2:Track id=tid_2 clockDrive=d_2
e_4:next
T_4:Track
S_2:Shuttle id=sid_2 mode=m_2
e_2:at
T_3:Track
e_5:next
$m_1 = DRIVE \wedge m_2 = DRIVE \wedge m_1' = STOP \wedge unchanged(sid_1, minD_1, sid_2, m_2, tid_1, tid_2)$
guard: $d_1 \geq minD_1$, reset: $\{d_2'\}$, priority: 0, stepLabel: $(sid_1, minD_1, tid_1, sid_2, tid_2;)$

Figure B.6: The rule *Stop2*: a shuttle stops due to another shuttle that would be just ahead after driving forward



$minD_1' = 3 \wedge a_1' = \bot \wedge unchanged(sid_1, tid_1)$
[success] reset: $\emptyset$, probability: $1 - 10^{-6}$
$R_1 = L$
$R_2 = L$
$a_1' = \bot \wedge unchanged(sid_1, minD_1, tid_1)$
[failure] reset: $\emptyset$, probability: 10^{-6}
r_1
r_2
L
S_1:Shuttle id=sid_1 minDur=$minD_1$
e_1:at
T_1:Track id=tid_1
e_2:at
Y_1:TLYellow active=a_1
$a_1 = \top$
guard: $\top$, priority: 1, stepLabel: $(sid_1, minD_1, tid_1;)$

Figure B.7: The rule *SetSlow*: a shuttle may slow down due to a yellow traffic light



$\neg\exists$ S_1:Shuttle e_2:at T_1:Track $\top$
T_1:Track id=tid_1
e_1:at
Y_1:TLYellow active=a_1
$a_1 = \bot \wedge a_1' = \top \wedge unchanged(tid_1)$
guard: $\top$, reset: $\emptyset$, priority: 1, stepLabel: $(tid_1;)$

Figure B.8: The rule *ResetYellow*: the *active* attribute of the yellow traffic light is reset once a shuttle has exited the track with the traffic light on it

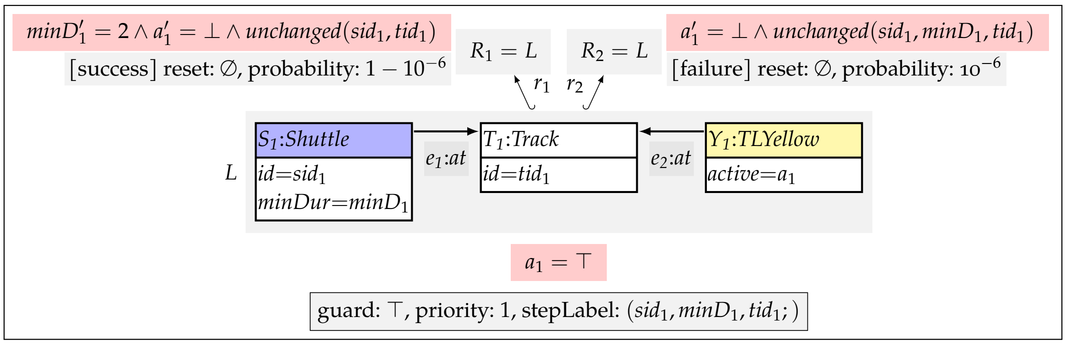

$minD'_1 = 2 \wedge a'_1 = \bot \wedge unchanged(sid_1, tid_1)$
[success] reset: $\emptyset$, probability: $1 - 10^{-6}$
$R_1 = L$
$R_2 = L$
r_1
r_2
$a'_1 = \bot \wedge unchanged(sid_1, minD_1, tid_1)$
[failure] reset: $\emptyset$, probability: 10^{-6}
L
S_1:Shuttle
id=sid_1
minDur=$minD_1$
e_1:at
T_1:Track
id=tid_1
e_2:at
Y_1:TLYellow
active=a_1
$a_1 = \top$
guard: $\top$, priority: 1, stepLabel: $(sid_1, minD_1, tid_1;)$

Figure B.9: The rule *SetFast*: a shuttle may increase its velocity due to a green traffic light

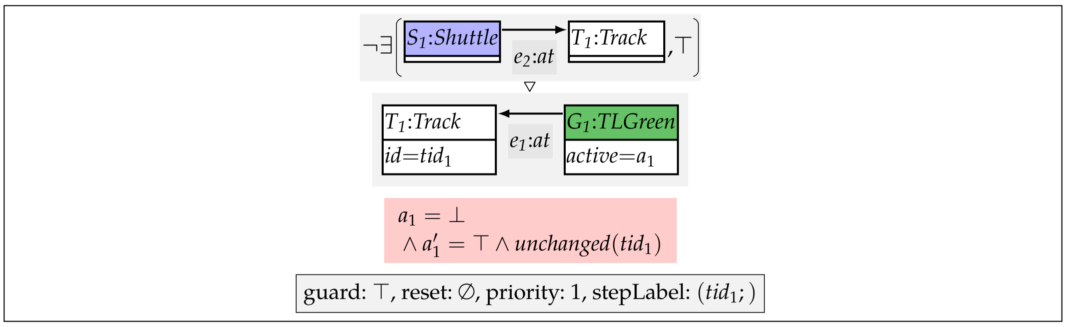


Figure B.10: The rule *ResetGreen*: the *active* attribute of the green traffic light is reset once a shuttle has exited the track with the traffic light on it

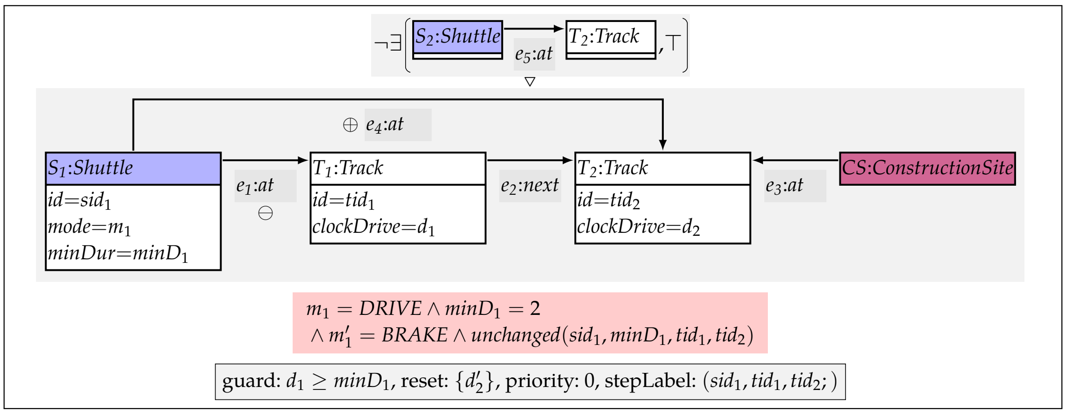
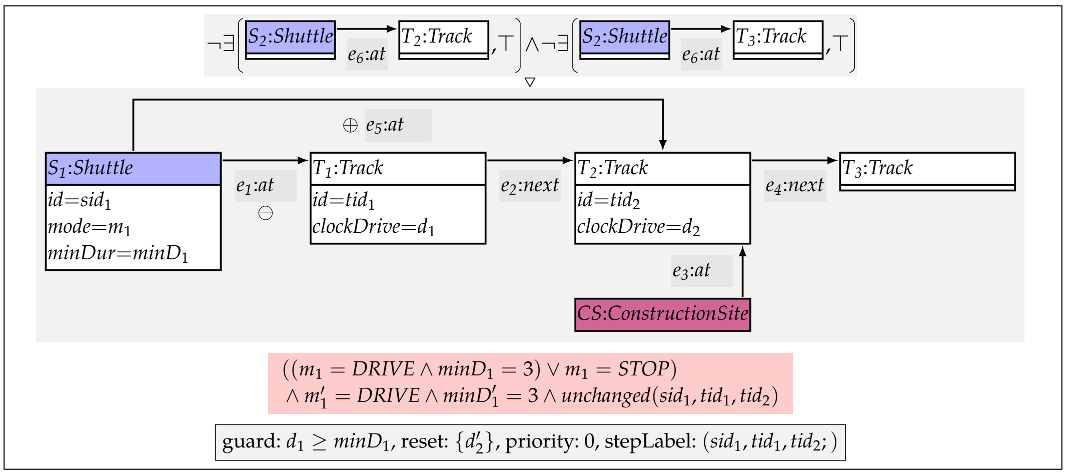
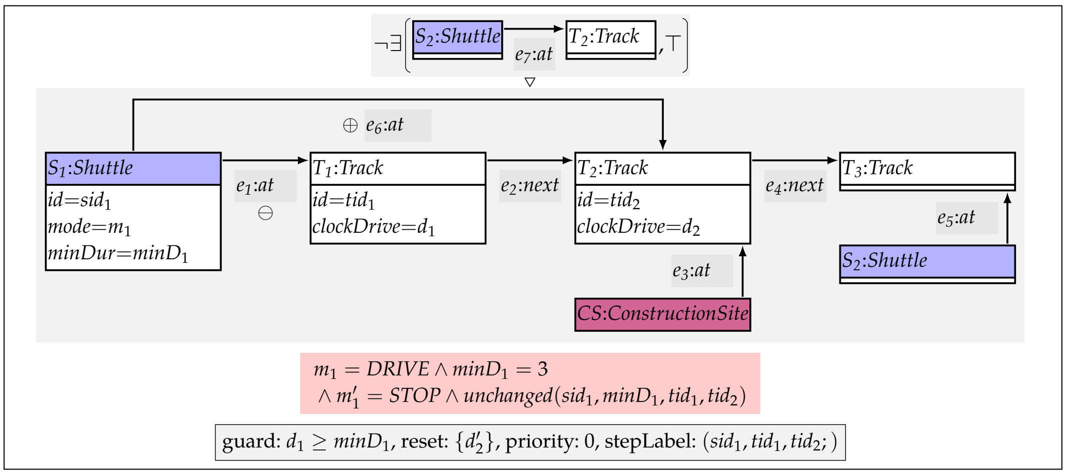


Figure B.11: The rule *ConstructionSiteBrake*: a fast shuttle executes an emergency brake approaching a construction site



$\neg\exists$ S_2:Shuttle e_6:at T_2:Track $,\top$ $\wedge\neg\exists$ S_2:Shuttle e_6:at T_3:Track $,\top$
$\oplus$ e_5:at
S_1:Shuttle
id=sid_1
mode=m_1
minDur=$minD_1$
e_1:at $\ominus$
T_1:Track
id=tid_1
clockDrive=d_1
e_2:next
T_2:Track
id=tid_2
clockDrive=d_2
e_4:next
T_3:Track
e_3:at
CS:ConstructionSite
$((m_1 = DRIVE \wedge minD_1 = 3) \vee m_1 = STOP)$
$\wedge\, m_1' = DRIVE \wedge minD_1' = 3 \wedge unchanged(sid_1, tid_1, tid_2)$
guard: $d_1 \geq minD_1$, reset: $\{d_2'\}$, priority: 0, stepLabel: $(sid_1, tid_1, tid_2;)$

Figure B.12: The rule *ConstructionSiteDrive*: a slow shuttle drives over a construction site



$\neg\exists$ S_2:Shuttle e_7:at T_2:Track $,\top$
$\oplus$ e_6:at
S_1:Shuttle
id=sid_1
mode=m_1
minDur=$minD_1$
e_1:at $\ominus$
T_1:Track
id=tid_1
clockDrive=d_1
e_2:next
T_2:Track
id=tid_2
clockDrive=d_2
e_4:next
T_3:Track
e_5:at
S_2:Shuttle
e_3:at
CS:ConstructionSite
$m_1 = DRIVE \wedge minD_1 = 3$
$\wedge\, m_1' = STOP \wedge unchanged(sid_1, minD_1, tid_1, tid_2)$
guard: $d_1 \geq minD_1$, reset: $\{d_2'\}$, priority: 0, stepLabel: $(sid_1, tid_1, tid_2;)$

Figure B.13: The rule *ConstructionSiteStop*: a shuttle may need to stop due to shuttles ahead also on a construction site

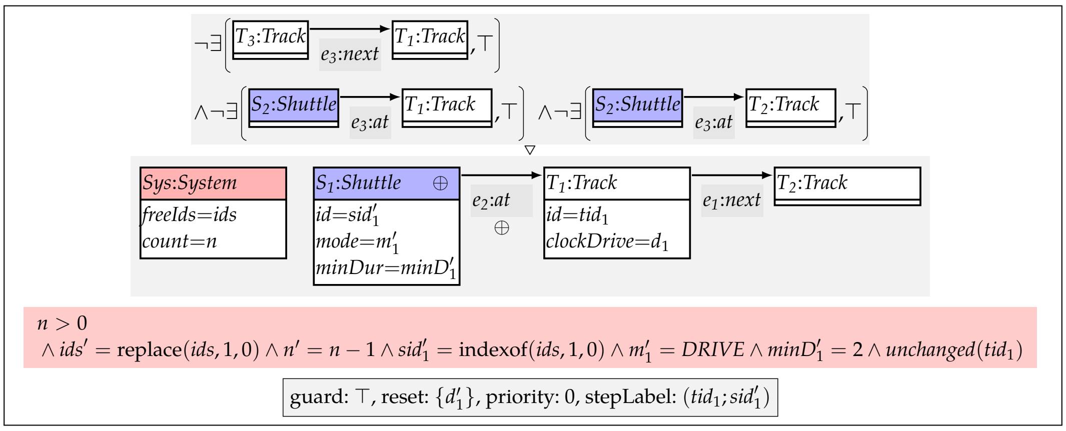

$\neg\exists$ T_3:Track e_3:next T_1:Track , $\top$
$\wedge\neg\exists$ S_2:Shuttle e_3:at T_1:Track , $\top$
$\wedge\neg\exists$ S_2:Shuttle e_3:at T_2:Track , $\top$
Sys:System
freeIds=ids
count=n
S_1:Shuttle $\oplus$
id=sid'_1
mode=m'_1
minDur=$minD'_1$
e_2:at $\oplus$
T_1:Track
id=tid_1
clockDrive=d_1
e_1:next
T_2:Track
$n > 0$
$\wedge\, ids' = \text{replace}(ids, 1, 0) \wedge n' = n - 1 \wedge sid'_1 = \text{indexof}(ids, 1, 0) \wedge m'_1 = DRIVE \wedge minD'_1 = 2 \wedge unchanged(tid_1)$
guard: $\top$, reset: $\{d'_1\}$, priority: 0, stepLabel: $(tid_1; sid'_1)$

Figure B.14: The rule *DriveEnterFast*: additional rule for FT4 and FT5 for a fresh fast shuttle entering the FT

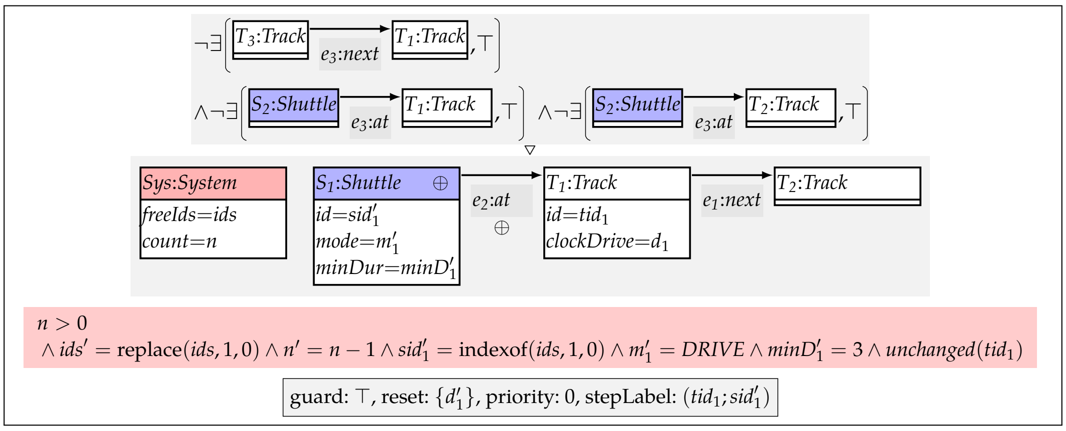

$\neg\exists$ T_3:Track e_3:next T_1:Track , $\top$
$\wedge\neg\exists$ S_2:Shuttle e_3:at T_1:Track , $\top$
$\wedge\neg\exists$ S_2:Shuttle e_3:at T_2:Track , $\top$
Sys:System
freeIds=ids
count=n
S_1:Shuttle $\oplus$
id=sid'_1
mode=m'_1
minDur=$minD'_1$
e_2:at $\oplus$
T_1:Track
id=tid_1
clockDrive=d_1
e_1:next
T_2:Track
$n > 0$
$\wedge\, ids' = \text{replace}(ids, 1, 0) \wedge n' = n - 1 \wedge sid'_1 = \text{indexof}(ids, 1, 0) \wedge m'_1 = DRIVE \wedge minD'_1 = 3 \wedge unchanged(tid_1)$
guard: $\top$, reset: $\{d'_1\}$, priority: 0, stepLabel: $(tid_1; sid'_1)$

Figure B.15: The rule *DriveEnterSlow*: additional rule for FT4 and FT5 for a fresh slow shuttle entering the FT

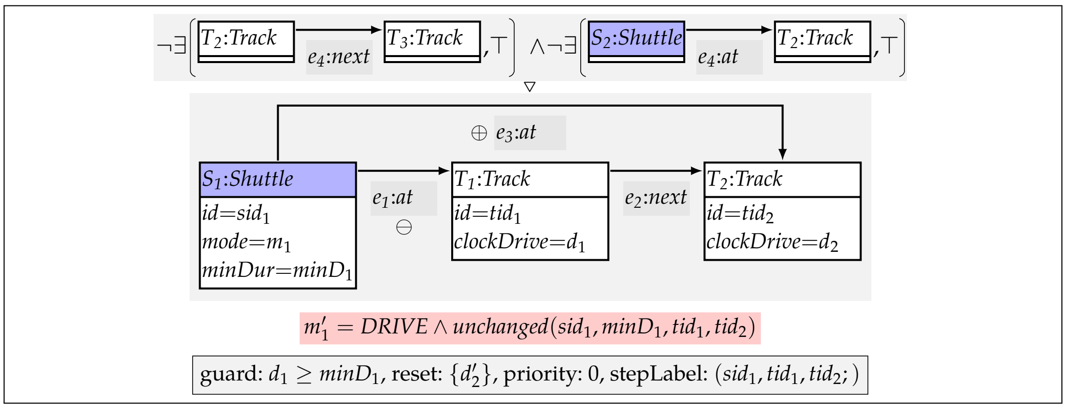

$\neg\exists$ T_2:Track e_4:next T_3:Track , $\top$
$\wedge\neg\exists$ S_2:Shuttle e_4:at T_2:Track , $\top$
$\oplus$ e_3:at
S_1:Shuttle
id=sid_1
mode=m_1
minDur=$minD_1$
e_1:at $\ominus$
T_1:Track
id=tid_1
clockDrive=d_1
e_2:next
T_2:Track
id=tid_2
clockDrive=d_2
$m'_1 = DRIVE \wedge unchanged(sid_1, minD_1, tid_1, tid_2)$
guard: $d_1 \geq minD_1$, reset: $\{d'_2\}$, priority: 0, stepLabel: $(sid_1, tid_1, tid_2;)$

Figure B.16: The rule *DriveExit1*: additional rule for FT4 and FT5 for a shuttle approaching the end of the FT

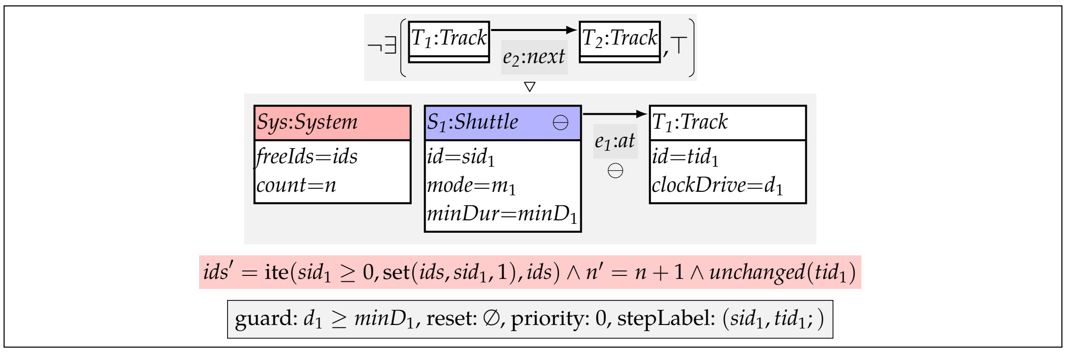

$\neg\exists$ (T_1:Track e_2:next T_2:Track , $\top$)
Sys:System
freeIds=ids
count=n
S_1:Shuttle ⊖
id=sid_1
mode=m_1
minDur=$minD_1$
e_1:at ⊖
T_1:Track
id=tid_1
clockDrive=d_1
$ids' = \mathrm{ite}(sid_1 \geq 0, \mathrm{set}(ids, sid_1, 1), ids) \wedge n' = n + 1 \wedge unchanged(tid_1)$
guard: $d_1 \geq minD_1$, reset: $\emptyset$, priority: 0, stepLabel: $(sid_1, tid_1;)$

Figure B.17: The rule *DriveExit2*: additional rule for FT4 and FT5 for a shuttle exiting the FT

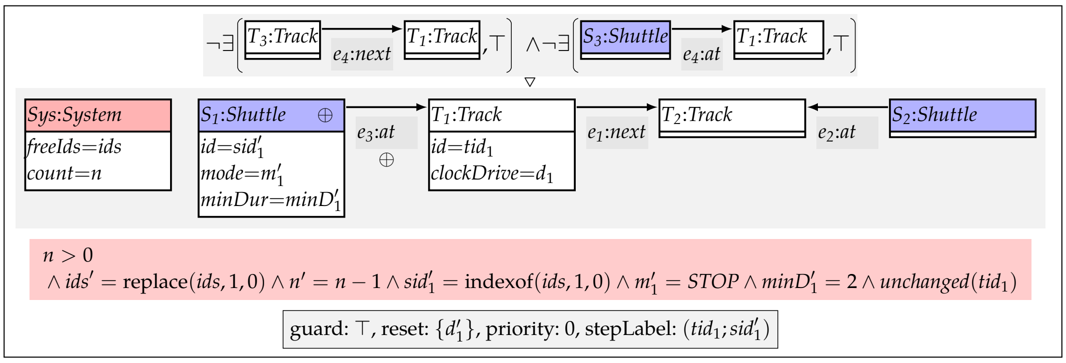

$\neg\exists$ (T_3:Track e_4:next T_1:Track , $\top$) $\wedge\neg\exists$ (S_3:Shuttle e_4:at T_1:Track , $\top$)
Sys:System
freeIds=ids
count=n
S_1:Shuttle ⊕
id=sid_1'
mode=m_1'
minDur=$minD_1'$
e_3:at ⊕
T_1:Track
id=tid_1
clockDrive=d_1
e_1:next
T_2:Track
e_2:at
S_2:Shuttle
$n > 0$
$\wedge\ ids' = \mathrm{replace}(ids, 1, 0) \wedge n' = n - 1 \wedge sid_1' = \mathrm{indexof}(ids, 1, 0) \wedge m_1' = STOP \wedge minD_1' = 2 \wedge unchanged(tid_1)$
guard: $\top$, reset: $\{d_1'\}$, priority: 0, stepLabel: $(tid_1; sid_1')$

Figure B.18: The rule *StopEnterFast*: additional rule for FT4 and FT5 for a fresh fast shuttle entering the FT and immediately stopping due to a shuttle ahead

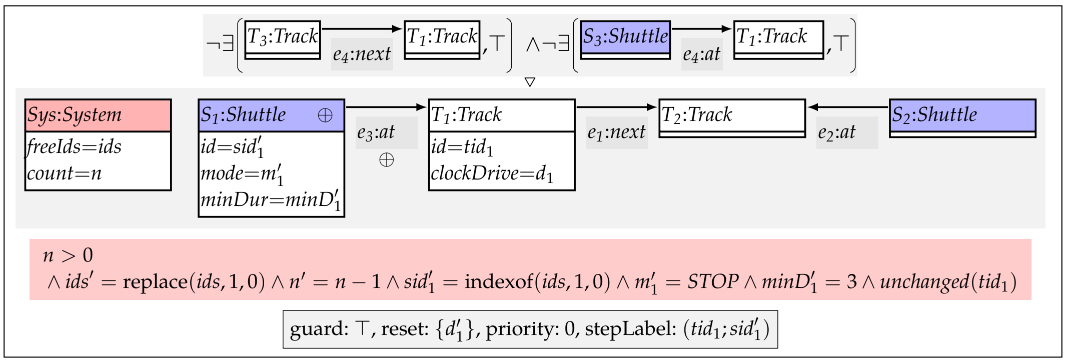

$\neg\exists$ (T_3:Track e_4:next T_1:Track , $\top$) $\wedge\neg\exists$ (S_3:Shuttle e_4:at T_1:Track , $\top$)
Sys:System
freeIds=ids
count=n
S_1:Shuttle ⊕
id=sid_1'
mode=m_1'
minDur=$minD_1'$
e_3:at ⊕
T_1:Track
id=tid_1
clockDrive=d_1
e_1:next
T_2:Track
e_2:at
S_2:Shuttle
$n > 0$
$\wedge\ ids' = \mathrm{replace}(ids, 1, 0) \wedge n' = n - 1 \wedge sid_1' = \mathrm{indexof}(ids, 1, 0) \wedge m_1' = STOP \wedge minD_1' = 3 \wedge unchanged(tid_1)$
guard: $\top$, reset: $\{d_1'\}$, priority: 0, stepLabel: $(tid_1; sid_1')$

Figure B.19: The rule *StopEnterSlow*: additional rule for FT4 and FT5 for a fresh slow shuttle entering the FT and immediately stopping due to a shuttle ahead

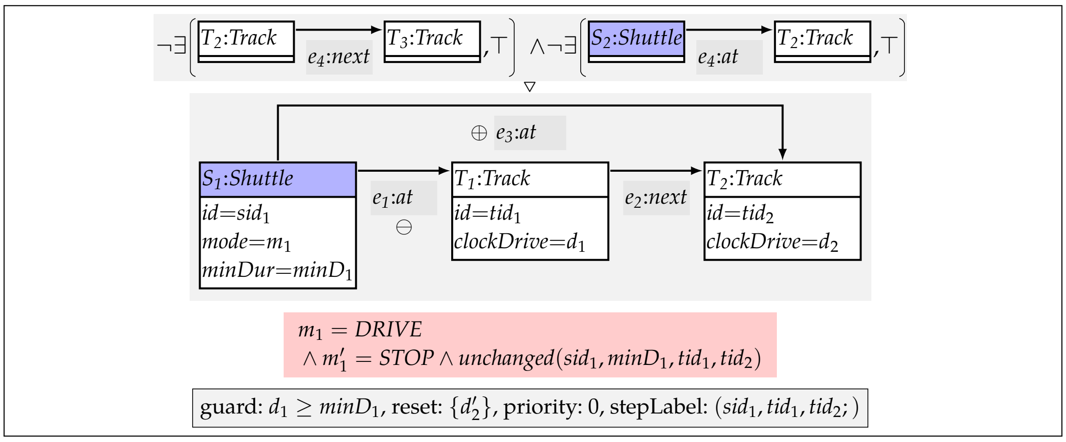

$\neg\exists$ T_2:Track e_4:next T_3:Track $,\top$
$\wedge\neg\exists$ S_2:Shuttle e_4:at T_2:Track $,\top$
$\oplus$ e_3:at
S_1:Shuttle
$id=sid_1$
$mode=m_1$
$minDur=minD_1$
e_1:at $\ominus$
T_1:Track
$id=tid_1$
$clockDrive=d_1$
e_2:next
T_2:Track
$id=tid_2$
$clockDrive=d_2$
$m_1 = DRIVE$
$\wedge\, m_1' = STOP \wedge unchanged(sid_1, minD_1, tid_1, tid_2)$
guard: $d_1 \geq minD_1$, reset: $\{d_2'\}$, priority: 0, stepLabel: $(sid_1, tid_1, tid_2;)$

Figure B.20: The rule *StopExit1*: additional rule for FT4 and FT5 for a shuttle approaching the end of the FT and stopping due to a shuttle that may be on the next track in $\mathcal{S}_0$

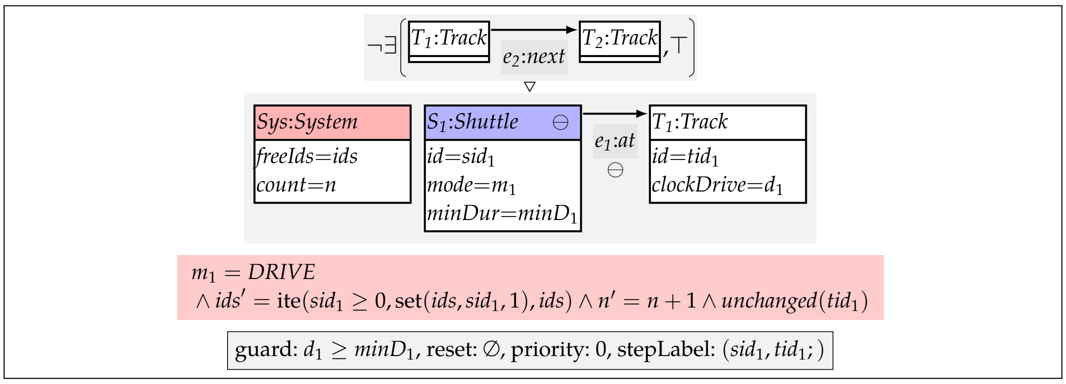

$\neg\exists$ T_1:Track e_2:next T_2:Track $,\top$
Sys:System
$freeIds=ids$
$count=n$
S_1:Shuttle $\ominus$
$id=sid_1$
$mode=m_1$
$minDur=minD_1$
e_1:at $\ominus$
T_1:Track
$id=tid_1$
$clockDrive=d_1$
$m_1 = DRIVE$
$\wedge\, ids' = \text{ite}(sid_1 \geq 0, \text{set}(ids, sid_1, 1), ids) \wedge n' = n + 1 \wedge unchanged(tid_1)$
guard: $d_1 \geq minD_1$, reset: $\emptyset$, priority: 0, stepLabel: $(sid_1, tid_1;)$

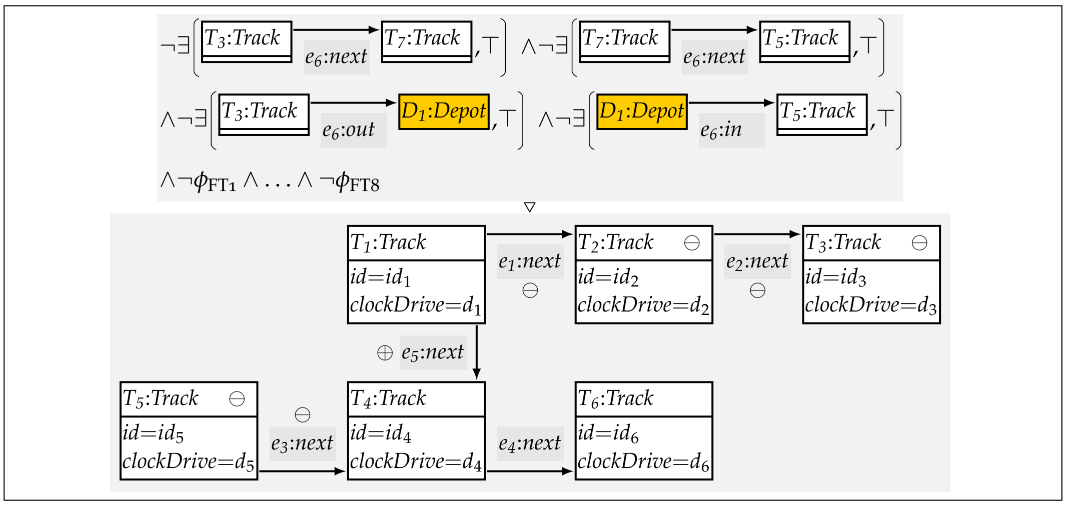
Figure B.21: The rule *StopExit2*: additional rule for FT4 and FT5 for a shuttle exiting the FT

C Fragment Topologies of Running Example

In this appendix, we present the FTs for our running example as well as the rule *Merge* for merging two FTs. Recall that these FTs and the rule *Merge* have been presented in abbreviated notation in Figure 4.1. An LST can be generated by (a) constructing the disjoint union of copies of the FTs and (b) by applying the rule *Merge* until it cannot be applied again.

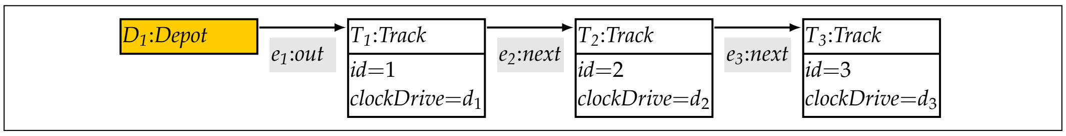
Also note that we rely on the *id* attributes of *Track* nodes only during the analysis of FTs to track the movement of shuttles. However, for the (de)composition of LSTs, it is obviously more appropriate to strip the *id* attributes from the FTs and the rule *Merge*.

The rule *Merge* is given in Figure C.1 and the FTs are given in Figure C.2, Figure C.3, Figure C.4, Figure C.5, Figure C.6, Figure C.7, Figure C.8, and Figure C.9.



$\neg\exists$ T_3:Track e_6:next T_7:Track , $\top$
$\wedge\neg\exists$ T_7:Track e_6:next T_5:Track , $\top$
$\wedge\neg\exists$ T_3:Track e_6:out D_1:Depot , $\top$
$\wedge\neg\exists$ D_1:Depot e_6:in T_5:Track , $\top$
$\wedge\neg\phi_{FT1} \wedge \ldots \wedge \neg\phi_{FT8}$
T_1:Track $id=id_1$ $clockDrive=d_1$
e_1:next ⊖
T_2:Track ⊖ $id=id_2$ $clockDrive=d_2$
e_2:next ⊖
T_3:Track ⊖ $id=id_3$ $clockDrive=d_3$
⊕ e_5:next
T_5:Track ⊖ $id=id_5$ $clockDrive=d_5$
⊖ e_3:next
T_4:Track $id=id_4$ $clockDrive=d_4$
e_4:next
T_6:Track $id=id_6$ $clockDrive=d_6$

Figure C.1: The rule *Merge* for merging of two FTs



D_1:Depot
e_1:out
T_1:Track $id=1$ $clockDrive=d_1$
e_2:next
T_2:Track $id=2$ $clockDrive=d_2$
e_3:next
T_3:Track $id=3$ $clockDrive=d_3$

Figure C.2: The fragment topology FT1

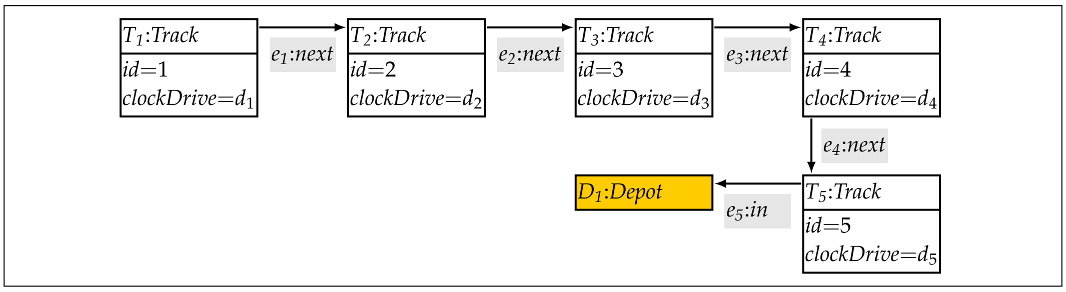


Figure C.3: The fragment topology FT2

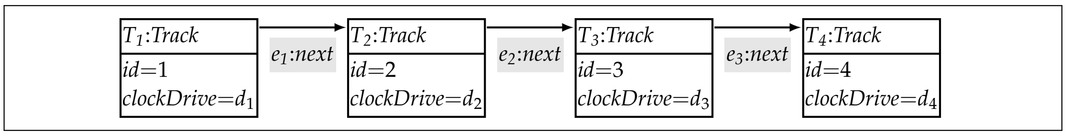


Figure C.4: The fragment topology FT3

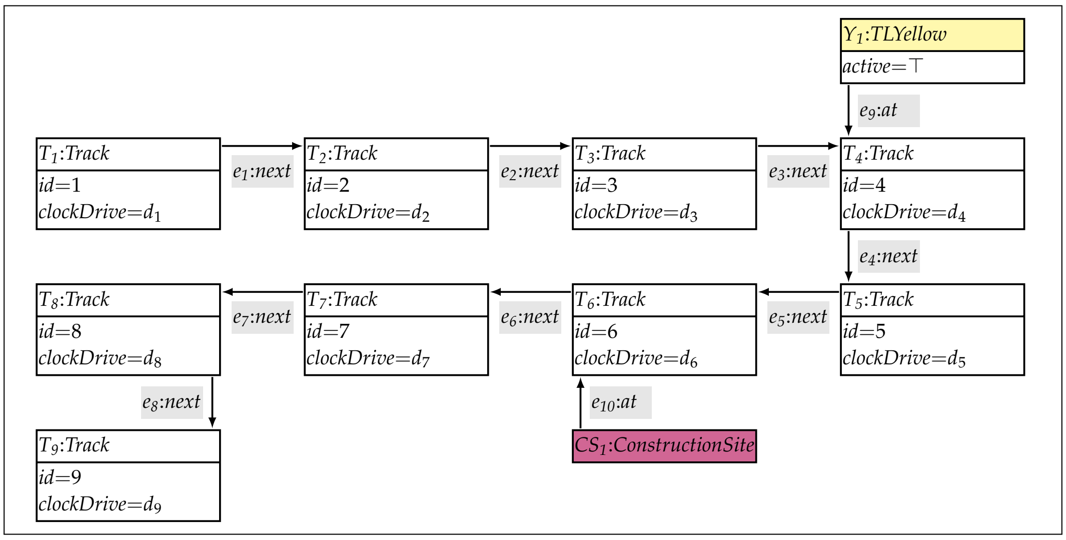


Figure C.5: The fragment topology FT4

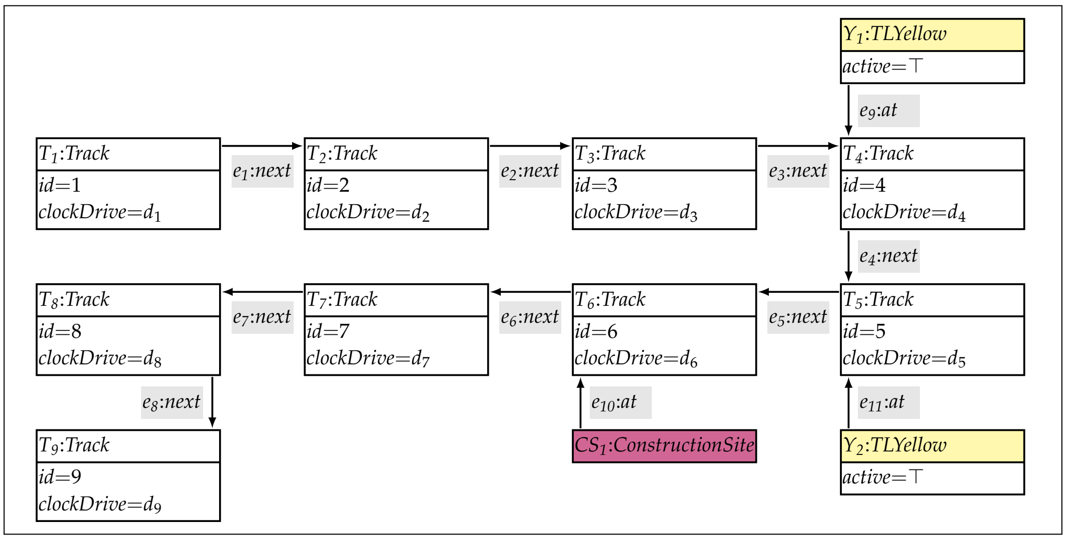


Figure C.6: The fragment topology FT5

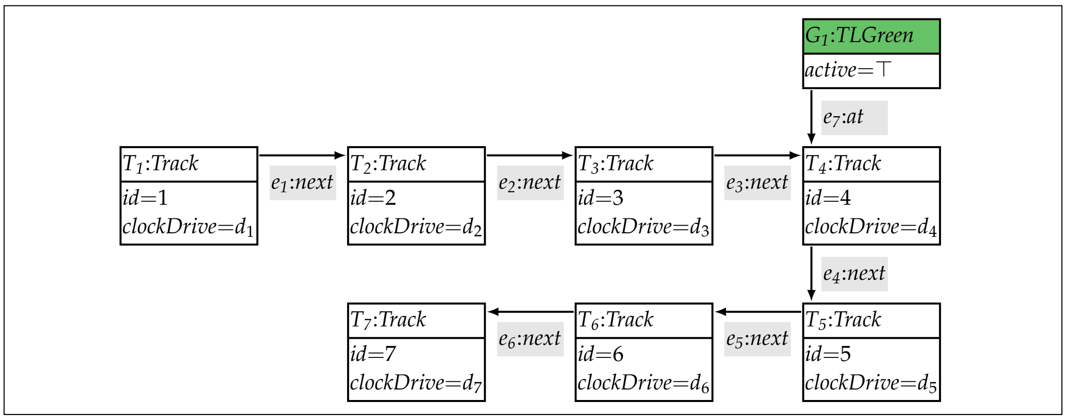


Figure C.7: The fragment topology FT6

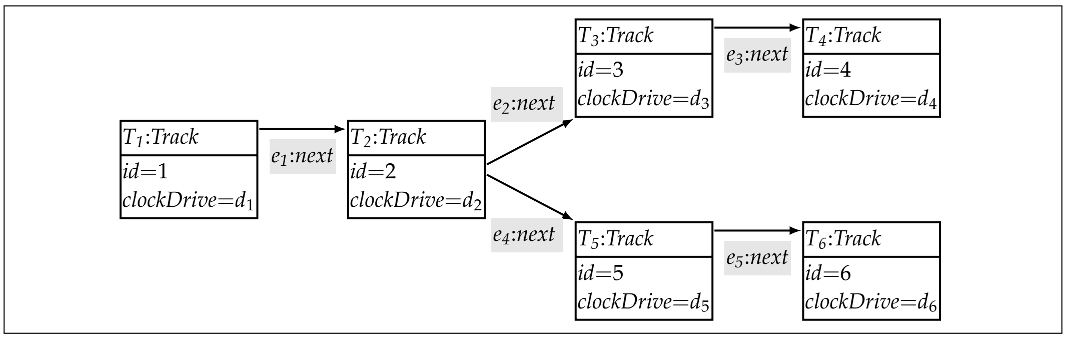


Figure C.8: The fragment topology FT7

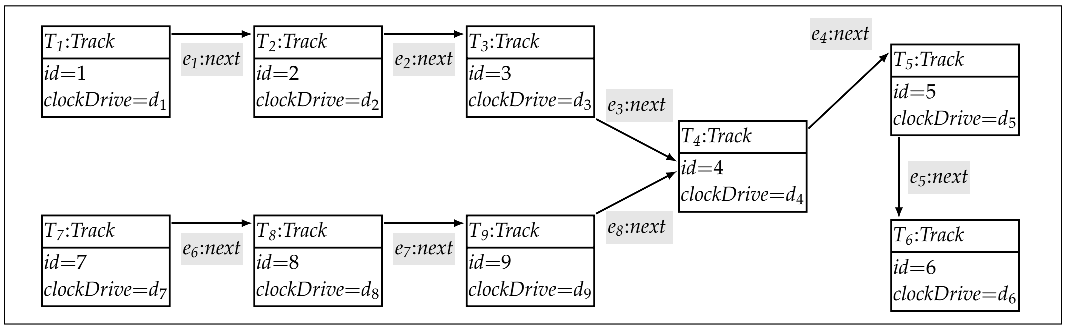


Figure C.9: The fragment topology FT8

D Construction/Parsing of Topologies

In this appendix, we present GT rules for the construction of LSTs. These rules also allow, since they are non-deleting, for the parsing using approaches such as [6, 7, 8]. In these rules, we make use of abbreviations in application conditions such as *noExitingEdgeFrom*(T_3) meaning that the node T_3, which is a *Track* node, has no exiting *next* edge to some *Track* node and no exiting *out* edge to some *Depot* node as well as *noEnteringEdgeTo*(T_5) meaning that the node T_5, which is a *Track* node, has no entering *next* edge from some *Track* node and no entering *in* edge from some *Depot* node. In comparison to the FTs from Appendix C, we are not generating *id* attributes for the *Track* nodes initially. Such additional attributes as well as a suitable *System* node could be generated once the basic structure of the LST has been constructed using the following rules. The idea of these rules, in comparison to the *Merge* rule is that each FT can be added to the current graph and that it can be overlapped (following the overlapping specification of three *Track* nodes connected via *next* edges) with a part of a previously generated FT.

The rules are given in Figure D.1, Figure D.2, Figure D.3, Figure D.4, Figure D.5, Figure D.6, Figure D.7, and Figure D.8.

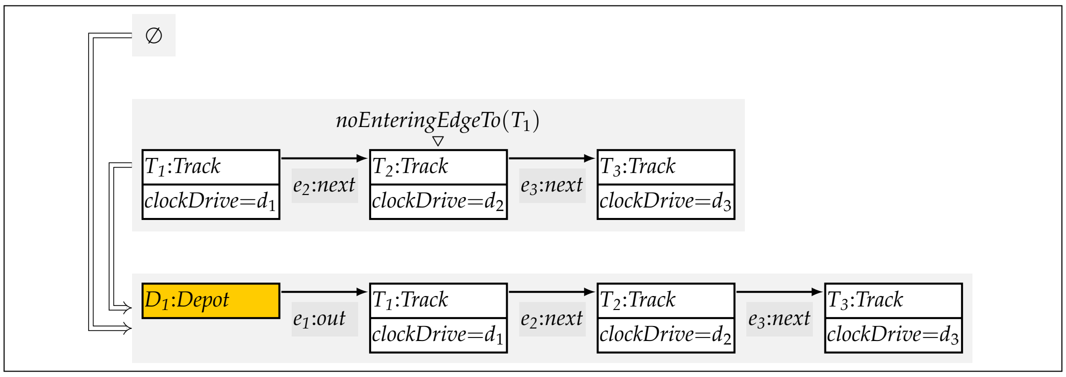


Figure D.1: Construction/parsing rules for FT1

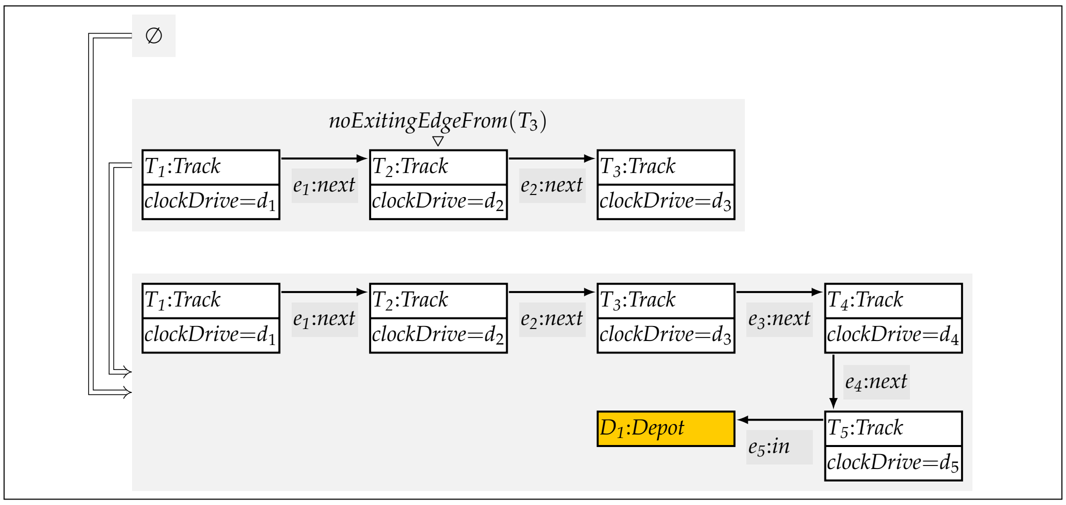


Figure D.2: Construction/parsing rules for FT2

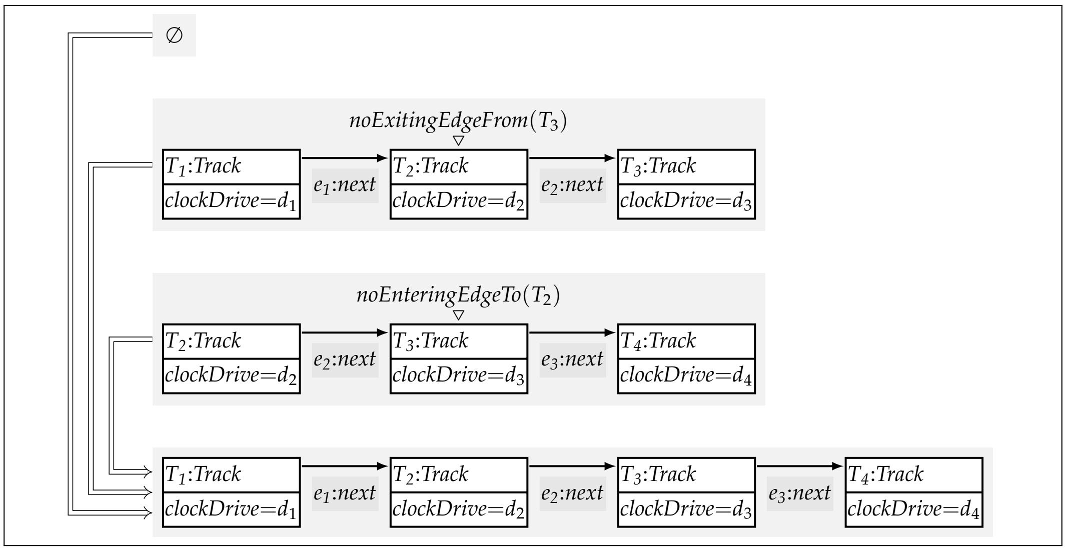


Figure D.3: Construction/parsing rules for FT3

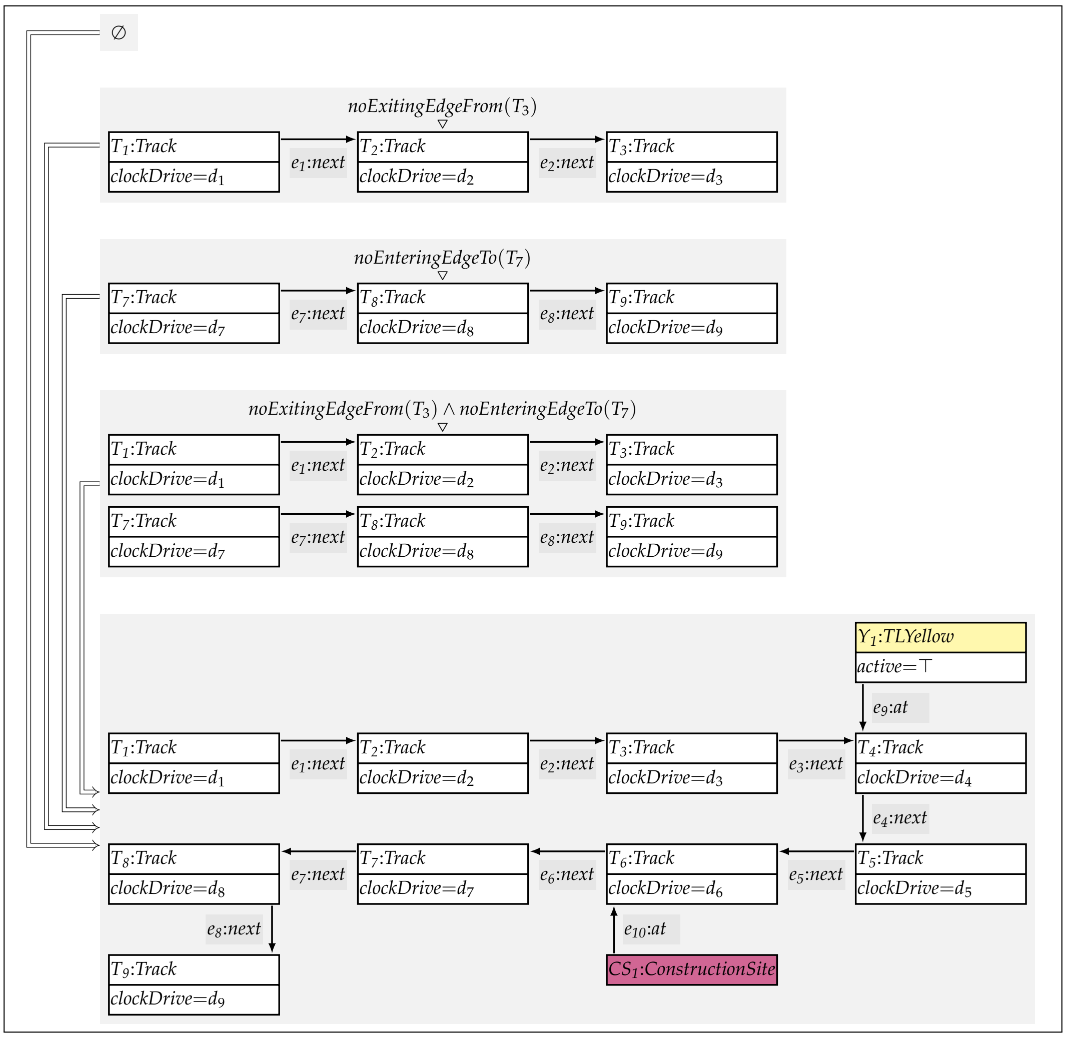


Figure D.4: Construction/parsing rules for FT4

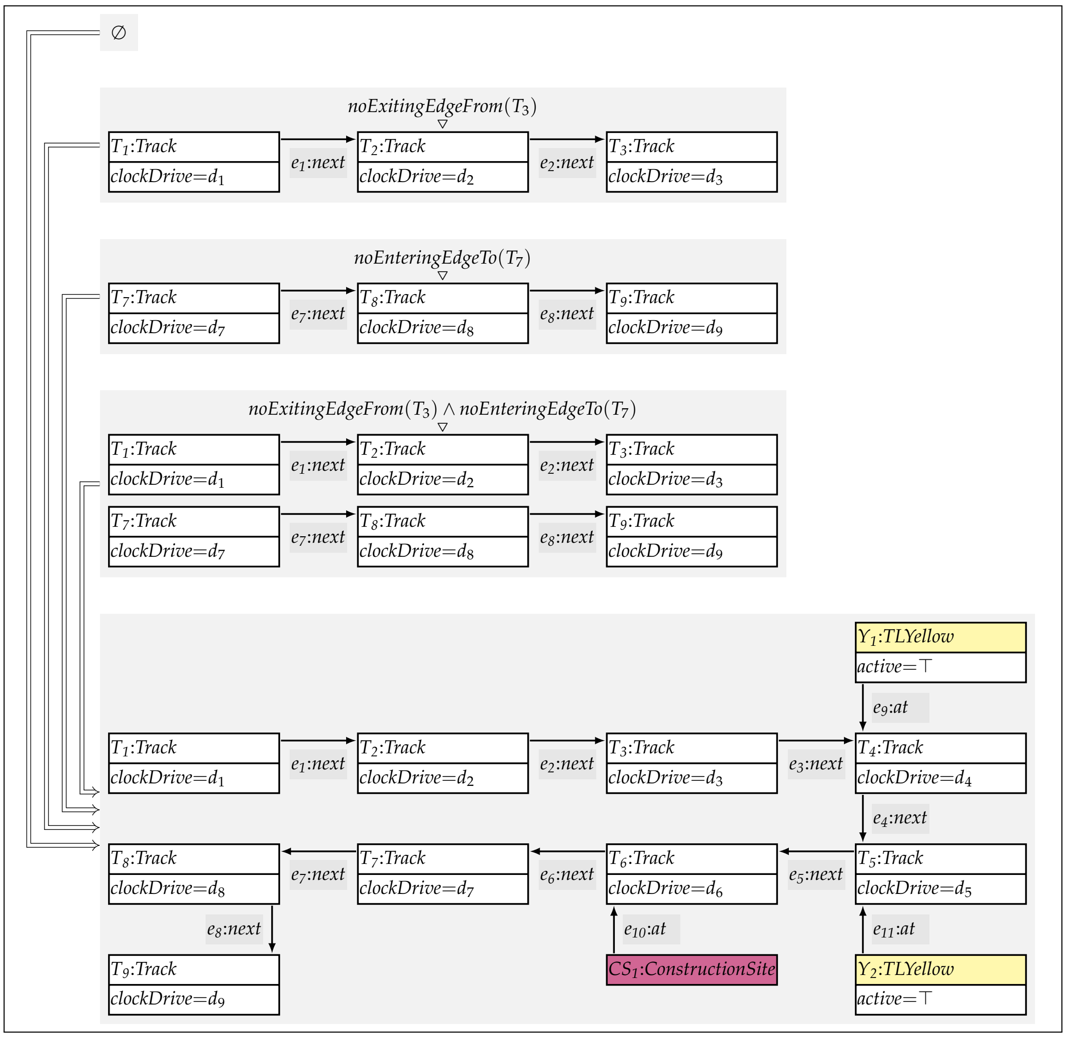


Figure D.5: Construction/parsing rules for FT5

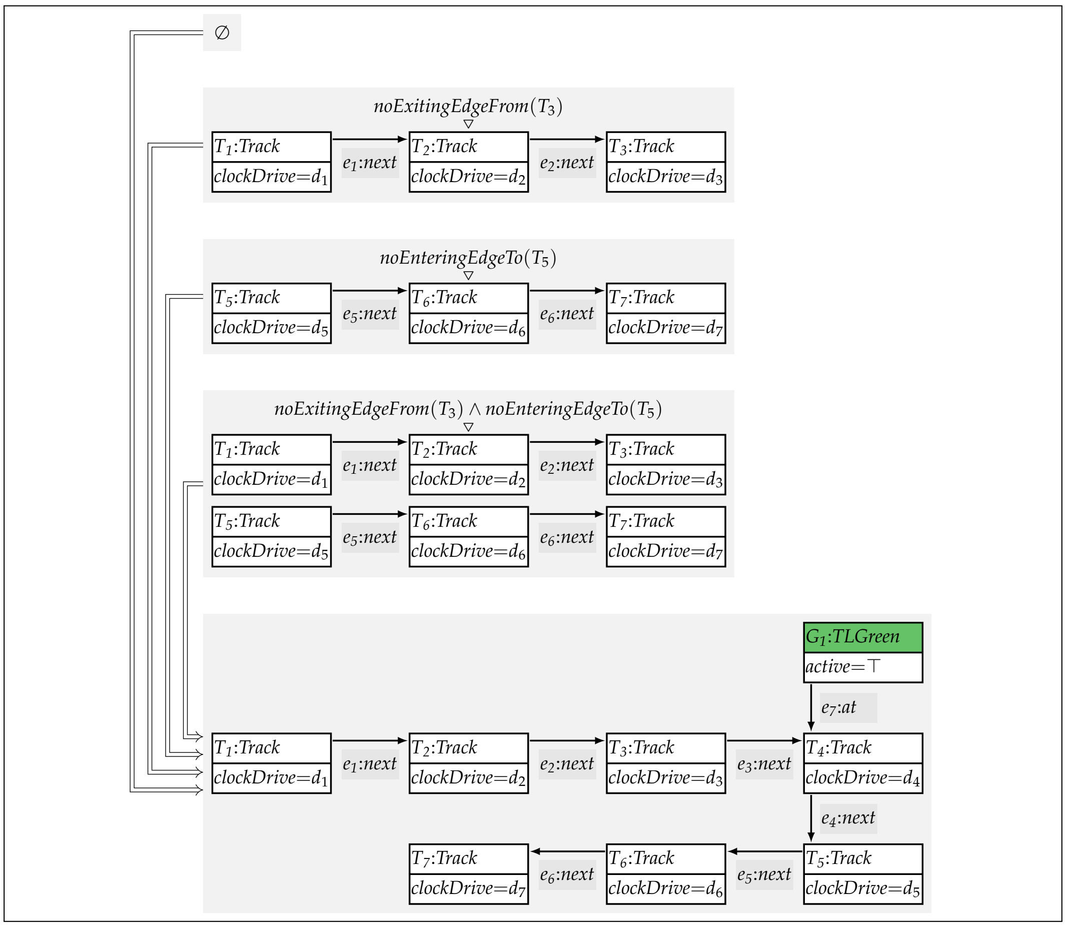


Figure D.6: Construction/parsing rules for FT6

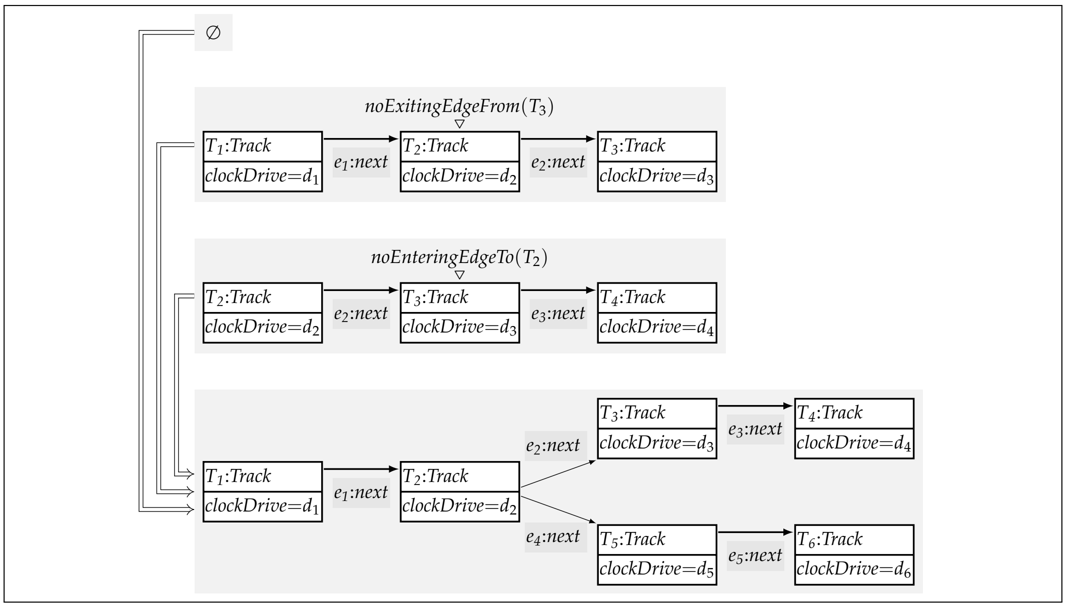


Figure D.7: Construction/parsing rules for FT7

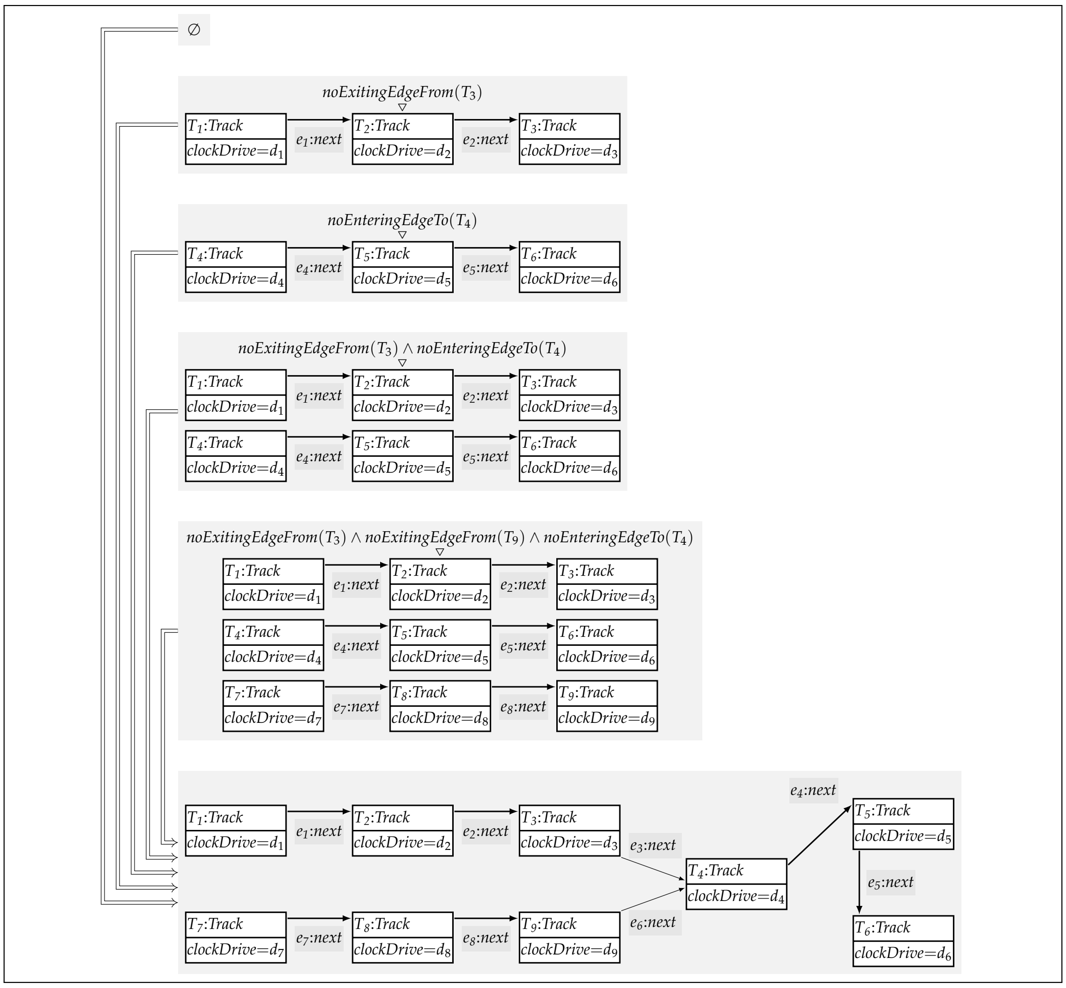


Figure D.8: Construction/parsing rules for FT8

ISBN 978-3-86956-501-9
ISSN 1613-5652