Formal Verification of Invariants for Attributed Graph Transformation Systems Based on Nested Attributed Graph Conditions

Sven Schneider(✉), Johannes Dyck, and Holger Giese

University of Potsdam, Hasso Plattner Institute, Potsdam, Germany
{sven.schneider,johannes.dyck,holger.giese}@hpi.de

Abstract. The behavior of various kinds of dynamic systems can be formalized using typed attributed graph transformation systems (GTSs). The states of these systems are then modelled using graphs and the evolution of the system from one state to another is described by a finite set of graph transformation rules. GTSs with small finite state spaces can be analyzed with ease but analysis is intractable/undecidable for GTSs inducing large/infinite state spaces due to the inherent expressiveness of GTSs. Hence, automatic analysis procedures do not terminate or return indefinite or incorrect results.

We propose an analysis procedure for establishing state-invariants for GTSs that are given by nested graph conditions (GCs). To this end, we formalize a symbolic analysis algorithm based on k-induction using Isabelle, apply it to GTSs and GCs over typed attributed graphs, develop support to single out some spurious counterexamples, and demonstrate the feasibility of the approach using our prototypical implementation.

Keywords: Formal static analysis · Symbolic state space abstraction · k-induction · Symbolic graphs · Isabelle

1 Introduction

The verification of formal models of complex dynamic systems w.r.t. to formal specifications is one of the grand challenges of model driven engineering. However, the expressiveness required to cover the multitude of complex actual and desired behaviors of such systems renders analysis often undecidable. Indeed, the formalism of graph transformation systems (GTSs) considered here is known to be Turing complete. Hence, fully-automatic procedures for establishing meaningful properties on the behavior of such systems are then guaranteed to be not terminating in general or to produce indefinite or even incorrect results.

Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) - 148420506.

F. Gadducci and T. Kehrer (Eds.): ICGT 2020, LNCS 12150, pp. 257–275, 2020.
https://doi.org/10.1007/978-3-030-51372-6_15

We subsequently focus on GTSs where an analysis using an explicit state space exploration using tools such as GROOVE [2] and HENSHIN [3] is not applicable due to infinite or intractably large sets of initial or reachable states.

We approach this problem by combining the symbolic static analysis techniques of k-induction and state abstractions to establish state invariants for dynamic systems with infinite state spaces modelled by GTSs. The idea of k-induction is to establish a state invariant by iteratively computing all shortest derivations from an initial state to a violating state. The use of state abstractions, which preserve and reflect the systems' behavior w.r.t. the invariant candidate, permits to handle GTSs with infinite sets of initial or violating states at the concrete level but finite (and sufficiently small) such sets at the abstract level.

As main contributions, we (a) formalize the principle of k-induction in the theorem prover Isabelle in the form of an analysis algorithm and (b) instantiate this analysis algorithm for the setting of (b1) invariant candidates formalized using the logic of nested graph conditions (GCs) and (b2) a suitable notion of typed attributed graph transformation. This instantiation based approach thereby also clearly separates aspects of k-induction from GTS related concepts.

To represent typed attributed graphs, we employ symbolic graphs [18–22], which are similar to E-GRAPHS [12]. These symbolic graphs also give rise to an instantiation of GCs that permits the specification of constraints on attributes throughout the GCs. We employ a graph transformation step relation on symbolic graphs that deviates from those formalized in [21,22] by being symmetric (allowing a backwards application used in the k-induction analysis algorithm) and by allowing for the removal of variables (not requiring that additional variables and their values must be guessed when computing backward steps).

As closest related work, approaches using k-induction have been used before without formal foundation in [4] and in [7–11] assuming $k = 1$, graphs without attributes, a single initial state, or a subclass of all GCs. Hence, we extend this line of research by formally treating the more general case of an arbitrary value of k, graphs with attributes, infinitely many initial states, and all GCs.

In [5,25,27,28], an abstraction of graphs results in shape graphs (which have limited expressiveness compared to GCs) where multiple nodes in the graph are represented by so called summary nodes in the shape graph and where multiplicity or even first-order logic constraints may further restrict this abstraction (see also [6]). Moreover, in [15], an abstraction of graphs is given in terms of consistent compasses (which can be encoded in GCs of depth one) containing a set of graphs of which one is matchable and a set of non-matchable graphs. Also, in [29], the tool ALLOY is used to establish state invariants for typed graphs.

Further related analysis approaches are as follows. The tool AUGUR2 [1] abstracts GTSs to Petri nets but imposes restrictions on graph transformation rules thereby limiting expressiveness. Lastly, static analysis of programs for GTSs w.r.t. pre/post conditions has been developed in [23] as well as [24].

In Sect. 2, we formalize the principle of k-induction in the form of an analysis algorithm. In Sect. 3, we discuss our running example, our notion of attributed graph transformation, and the logic of GCs. In Sect. 4, we instantiate the

analysis algorithm for attributed graph transformation and apply our prototypical implementation of it to our running example demonstrating its feasibility. In Sect. 5, we provide a conclusion and a discussion of future work.

2 Invariant Verification Using *k*-Induction

We now introduce our formalization of the technique of k-induction for the verification of (state) invariants. For this purpose, we introduce labelled transition systems (LTS) as an abstract framework, which is instantiated later on for graph transformation. The results of this section have been formalized in the interactive theorem prover Isabelle and we therefore omit all proofs. An LTS consists of a set S of states, a set L of labels, a relation δ of labelled steps between states, and initial states identified via a state predicate Z.

Definition 1 (Labelled Transition System (LTS)). *If S and L are sets of states and labels, $\delta \subseteq S \times L \times S$, $Z : S \rightarrow \mathbf{B}$, and $\Gamma = (S, L, \delta, Z)$, then Γ* is a labelled transition system, *written $\Gamma \in \mathcal{S}^{\text{lts}}$.*

Moreover, a finite path $\pi \in \mathsf{paths}(\Gamma, n)$ of Γ of length n is a sequence of n states from S interleaved with labels from L where $s{\cdot}l{\cdot}s'$ in π implies $(s, l, s') \in \delta$. Also, π_{S} and π_{L} map indices to the states and labels of the path π.

In Sect. 4, we restrict the states of an LTS resulting in a sub-LTS as follows.

Definition 2 (Sub-LTS). *If $\Gamma = (S, L, \delta, Z) \in \mathcal{S}^{\text{lts}}$, $S' \subseteq S$, and $\Gamma' = (S', L, \delta \cap (S' \times L \times S'), Z \cap (S' \times \mathbf{B})) \in \mathcal{S}^{\text{lts}}$, then Γ'* is a sub-LTS of Γ.

A predicate I on the states of an LTS is an invariant for the LTS, if all states that are reachable from an initial state of the LTS satisfy I.

Definition 3 (Invariant). *If $\Gamma = (S, L, \delta, Z) \in \mathcal{S}^{\text{lts}}$, $I : S \rightarrow \mathbf{B}$, and $\forall n \in \mathbf{N}.\ \forall \pi \in \mathsf{paths}(\Gamma, n).\ Z(\pi_{\mathsf{S}}(0)) \rightarrow I(\pi_{\mathsf{S}}(n))$, then Γ* has invariant I, *written* $\mathsf{invariant}(\Gamma, I)$.

Subsequently, we assume an invariant A (e.g. expressing earlier established invariants) for the LTS to improve applicability of the analysis approach as explained later on. For characterizing the k-induction algorithm below, we define shortest violations of a state predicate I as a finite path leading from an initial state to a state violating I visiting no further initial states and only passing through states satisfying I as well as A.

Definition 4 (Shortest Violation). *If $\Gamma = (S, L, \delta, Z) \in \mathcal{S}^{\text{lts}}$, $I : S \rightarrow \mathbf{B}$, $A : S \rightarrow \mathbf{B}$, $k \in \mathbf{N}$, $\pi \in \mathsf{paths}(\Gamma, k)$, $Z(\pi_{\mathsf{S}}(0))$, $\neg I(\pi_{\mathsf{S}}(k))$, $\forall 0 < j \leq k.\ \neg Z(\pi_{\mathsf{S}}(j))$, and $\forall j < k.\ I(\pi_{\mathsf{S}}(j)) \wedge A(\pi_{\mathsf{S}}(j))$, then π* is a shortest violation of I by Γ of length k under A, *written* $\pi \in \mathsf{SVio}(\Gamma, A, I, k)$.

The analysis algorithm $\mathcal{I}$ below checks for such shortest violations by (a) selecting all violating states s satisfying $\neg I(s)$ and by (b) computing up to k steps *backwards* ensuring that all k additional states s' visited on each of the paths

obtained satisfy $A(s') \wedge I(s')$. Firstly, when a state s', which is visited in this process, satisfies $Z(s')$, a shortest violation is obtained. Secondly, when no such path of k steps exists, there cannot be a shortest violation of greater length. Note that this analysis process benefits from employing the assumed invariant A, which is used to rule out paths with states that are known to be unreachable from an initial state by not satisfying A.

The analysis algorithm $\mathcal{I}$ returns a value b with three different values where $b = \mathsf{i}$ represents a successful verification of the given state predicate I as an invariant (when no paths are left that may be extended to shortest violations), where $b = \mathsf{v}$ represents that at least one shortest violation was determined, and where $b = \mathsf{u}$ represents the situation that the analysis was unable to return one of the two former definite results for the provided value of k that is decremented in each recursive application of $\mathcal{I}^{\text{inner}}$.

Definition 5 ($\mathcal{I}$). *If* $\Gamma = (S, L, \delta, Z) \in \mathcal{S}^{\text{lts}}$, $A : S \rightarrow \mathbf{B}$, $I : S \rightarrow \mathbf{B}$, $k \in \mathbf{N}$, $i \in \mathbf{N}$, *paths* $\subseteq \mathsf{paths}(\Gamma, i)$, *then* $\mathcal{I}^{\text{inner}}(\Gamma, A, I, k, i, \mathit{paths}) \subseteq \{(b, \mathit{violations}) \mid b \in \{\mathsf{i}, \mathsf{v}, \mathsf{u}\} \wedge \mathit{violations} \subseteq \mathsf{paths}(\Gamma, k + i)\}$ *as follows.*

$$\mathcal{I}^{\text{inner}}(\Gamma, A, I, k, i, \mathit{paths}) = \begin{cases} \mathsf{if}\ \mathit{paths} = \emptyset\ \mathsf{then}\ (\mathsf{i}, \emptyset) \\ \mathsf{elseif}\ \mathsf{vio}(\mathit{paths}) \neq \emptyset\ \mathsf{then}\ (\mathsf{v}, \mathsf{vio}(\mathit{paths})) \\ \mathsf{elseif}\ k = 0\ \mathsf{then}\ (\mathsf{u}, \mathit{paths}) \\ \mathsf{else}\ \mathcal{I}^{\text{inner}}(\Gamma, A, I, k-1, i+1, \mathsf{ext}(\mathit{paths})) \end{cases}$$

where

$$\mathsf{vio}(\mathit{paths}) = \{\pi \in \mathit{paths} \mid Z(\pi_{\mathsf{S}}(0))\}$$

$$\mathsf{ext}(\mathit{paths}) = \{s \cdot \ell \cdot \pi \mid \pi \in \mathit{paths} \wedge (s, \ell, \pi_{\mathsf{S}}(0)) \in \delta \wedge A(s) \wedge I(s)\}$$

Moreover, if $k \in \mathbf{N}$ *and* $\mathit{paths}_0 = \{\pi \in \mathsf{paths}(\Gamma, 0) \mid \neg I(\pi_{\mathsf{S}}(0))\}$ *is the set of violating paths of length* 0, *then* $\mathcal{I}(\Gamma, A, I, k) = \mathcal{I}^{\text{inner}}(\Gamma, A, I, k, 0, \mathit{paths}_0)$.

The following theorem states that the analysis algorithm $\mathcal{I}$ performs a sound state invariant analysis as just described above.

Theorem 1 (Soundness of $\mathcal{I}$). *If* $\Gamma = (S, L, \delta, Z) \in \mathcal{S}^{\text{lts}}$, $A : S \rightarrow \mathbf{B}$, $I : S \rightarrow \mathbf{B}$, $\mathsf{invariant}(\Gamma, A)$, $k \in \mathbf{N}$, *and* $\mathcal{I}(\Gamma, A, I, k) = (b, \mathit{paths})$, *then there is* $j \leq k$ *s.t.* $\mathit{paths} \subseteq \mathsf{paths}(\Gamma, j)$ *and one of the following items holds.*

- $b = \mathsf{u}$, $j = k$, $\mathit{paths} \neq \emptyset$, *and* $\bigcup\{\mathsf{SVio}(\Gamma, A, I, i) \mid i \leq k\} = \emptyset$.
- $b = \mathsf{i}$, $\mathsf{invariant}(\Gamma, I)$ *and* $\mathit{paths} = \emptyset$.
- $b = \mathsf{v}$, $\neg\mathsf{invariant}(\Gamma, I)$, $\mathit{paths} = \mathsf{SVio}(\Gamma, A, I, j) \neq \emptyset$.

The analysis algorithm $\mathcal{I}$ is implementable when the set of paths considered is finite throughout its computation. This is guaranteed when the LTS has violations for at most finitely many states (finite initial set of paths handed to $\mathcal{I}^{\text{inner}}$) and when every state has at most finitely many predecessors (each path can only be extended backwards to finitely many paths in $\mathcal{I}^{\text{inner}}$).

Definition 6 (Finitely Backwards Branching LTS). *If* $\Gamma = (S, L, \delta, Z) \in \mathcal{S}^{\text{lts}}$, $I : S \rightarrow \mathbf{B}$, $\mathsf{finite}(\{s \in S \mid \neg I(s)\})$, *and* $\forall s' \in S.\, \mathsf{finite}(\{s \mid (s, \ell, s') \in \delta\})$, *then* Γ is finitely backwards branching for I.

The *concrete* instantiation of LTSs for GTSs in Sect. 4 is not finitely backwards branching in general because invariant candidates I may be violated by infinitely many states. Hence, we apply in Sect. 4 an abstraction leading to an *abstract* instantiation of LTSs for GTSs where the corresponding invariant candidate I' is violated by finitely many states. We then establish a connection between both instantiations in terms of an LTS abstraction relation (LTSAR), which permits to analyze the abstract instantiation using $\mathcal{I}$ instead of the concrete instantiation.

Intuitively, the paths considered using $\mathcal{I}$ for the concrete LTS are symbolically represented by the finite set of paths considered using $\mathcal{I}$ for the abstract LTS. Formally, an LTSAR consists of two subrelations R_S relating states and R_L relating labels of the underlying concrete and abstract LTSs. Note that we state suitable requirements on the relations R_S and R_L of an LTSAR in the following theorem and define only the type of an LTSAR here.

Definition 7 (LTS Abstraction Relation (LTSAR)). *If* $\Gamma = (S, L, \delta, Z) \in \mathcal{S}^{\text{lts}}$, $\Gamma' = (S', L', \delta', Z') \in \mathcal{S}^{\text{lts}}$, $R_S \subseteq S \times S'$, $R_L \subseteq L \times L'$, *then* (R_S, R_L) is an LTS Abstraction Relation from Γ to Γ', *written* $\Gamma \leq_{R_S,R_L} \Gamma'$.

For invariant candidates I and I' for Γ and Γ', the following theorem states six requirements on an LTSAR (R_S, R_L), which guarantee that (a) a violation of I' in Γ' implies the existence of a violation of I in Γ and (b) the absence of violations of I' in Γ' implies the absence of violations of I in Γ.

Theorem 2 (Preservation/Reflection of Invariants using LTS Abstraction Relations). *If* $\Gamma = (S, L, \delta, Z) \in \mathcal{S}^{\text{lts}}$, $\Gamma' = (S', L', \delta', Z') \in \mathcal{S}^{\text{lts}}$, $A : S \rightarrowtail \mathbf{B}$, $\mathsf{invariant}(\Gamma, A)$, $I : S \rightarrowtail \mathbf{B}$, $I' : S' \rightarrowtail \mathbf{B}$, *and* $\Gamma \leq_{R_S,R_L} \Gamma'$, *then both of the following items hold.*

- Part1*: R1, R2, R3, R4, R5, and not* $\mathsf{invariant}(\Gamma', I')$ *imply not* $\mathsf{invariant}(\Gamma, I)$.
- Part2*: R1, R2, R3, R4, R6, and* $\mathsf{invariant}(\Gamma', I')$ *imply* $\mathsf{invariant}(\Gamma, I)$.

The requirements R1–R6 used in these items are as follows.

- R1*:* $\forall (s, s') \in R_S.\ I(s) \leftrightarrow I'(s')$ (R_S is compatible with invariant satisfaction)
- R2*:* $\forall (s, s') \in R_S.\ Z(s) \leftrightarrow Z'(s')$ (R_S is compatible with initial states)
- R3*:* $\forall s' \in S'.\ \exists s \in S.\ (s, s') \in R_S$
  (R_S relates a concrete state $s \in S$ to each abstract state $s' \in S'$)
- R4*:* $\forall s \in S.\ \big(\exists k \in \mathbf{N}.\ \exists \pi \in \mathsf{SVio}(\Gamma, A, I, k).\ \pi_{\mathsf{S}}(k) = s\big) \rightarrow \big(\exists s' \in S'.\ (s, s') \in R_S\big)$
  (R_S relates an abstract state $s' \in S'$ to each concrete state $s \in S$ for which a shortest violation of I exists)
- R5*:* $\forall (s, s') \in R_S.\ \forall (s', l', \bar{s}') \in \delta'.\ \exists (s, l, \bar{s}) \in \delta.\ (l, l') \in R_L \wedge (\bar{s}, \bar{s}') \in R_S$
  (every forward step of the abstract LTS Γ' can be mimicked (forwards) by the concrete LTS Γ for two related source states (s, s') to allow for the concretization of a violating path)
- R6*:* $\forall (\bar{s}, \bar{s}') \in R_S.\ \neg Z(\bar{s}) \rightarrow \forall (s, l, \bar{s}) \in \delta.\ \exists (s', l', \bar{s}') \in \delta'.\ (l, l') \in R_L \wedge (s, s') \in R_S$
  (every backward step of the concrete LTS Γ (except for those leading to initial states) can be mimicked (backwards) by the abstract LTS Γ' for two related target states $(\bar{s}, \bar{s}')$ to allow for the abstraction of a violating path)

3 Modelling and Specifying Graph Transformation

As a running example, we consider a single shuttle travelling on a network of tracks (see Fig. 1a for the type graph used) where subsequent tracks are connected using *next* edges. The graph attribution stores the velocity v and acceleration a of the shuttle and, moreover, the constants for minimal, maximal, and safe velocities as well as the constant track length s in a *System* node. The rules refer to the attributes to describe the velocity v' of a shuttle after travelling over a track based on its current velocity v, acceleration a, and the constant track length s using the standard equation $v'^2 = v^2 + 2as$. The velocity of the shuttle should be below the safe velocity on tracks with flag *signal*, the velocity of the shuttle should be constant on tracks with flag *const*, and the flag *warning* on a track indicates that a track with flag *signal* is to be expected ahead. Analysis should establish the fact that the shuttle never violates *signal* and *const* flags as an invariant, which is formalized in Fig. 1c using graph conditions explained below. Note that tracks with flag *const* between tracks with flag *warning* and tracks with flag *signal* may prevent timely deceleration. We employ an assumed invariant to (a) specify the constant attribute values of the system node, (b) to rule out track networks with dead ends and loops, and (c) to ensure warnings n tracks ahead of signals for a parameter $n \in \mathbf{N}$ in all considered track networks.

We now recall attribute conditions (ACs) used by symbolic graphs and then revisit GTSs and GCs over symbolic graphs for describing actual and desired behavior in terms of a concrete LTS and state predicates from before.

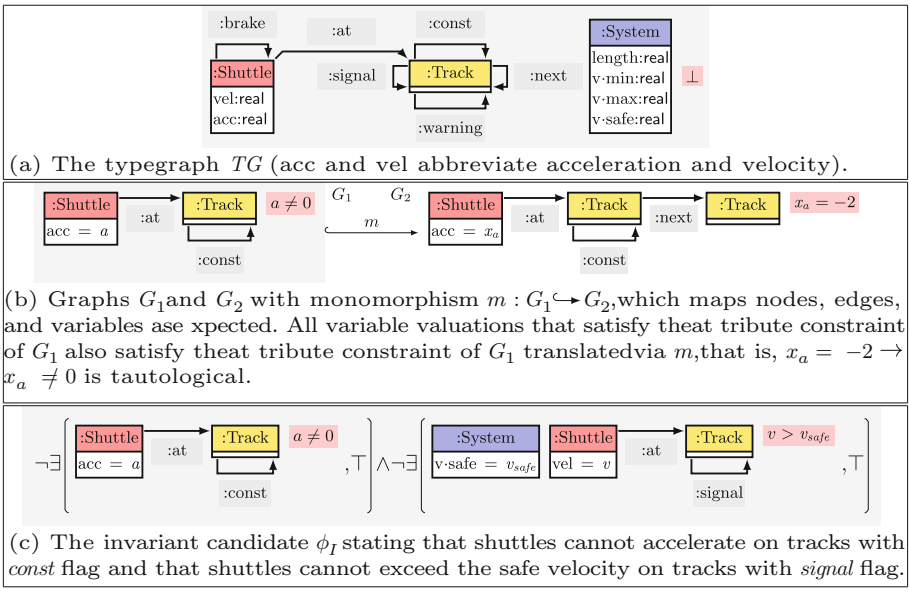


Fig. 1. Type graph and invariant candidate for the shuttle scenario.

The attribute logic AL contains ACs $\gamma \in \mathcal{S}_X^{\mathrm{AC}}$ of first-order logic (FOL) ranging over a set X of variables. The satisfaction of γ by a valuation $\alpha : X \to \mathcal{V}$ is denoted by $\alpha \models_{\mathrm{AC}} \gamma$. The SMT solver Z3 [17] supports ACs constructed using a restricted set of operators for the sorts bool, int, real, and string. When Z3 is unable to determine an answer to the satisfiability problem (note that AL satisfaction is undecidable), which does not occur for the examples considered here, we would notify the user in our prototypical implementation.

Symbolic graphs (called graphs subsequently) [18] are an adaptation of E-Graphs [12]. A finite graph G (such as those depicted in Fig. 1b) contains nodes, edges, variables $G.\mathrm{X}$, and an AC $G.\mathrm{ac}$ ranging over $G.\mathrm{X}$. Moreover, nodes and edges are equipped with node and edge attributes, which are connected to variables for which values are specified in the AC $G.\mathrm{ac}$. A morphism $m : G_1 \to G_2$ from graph G_1 to G_2 (see e.g. Fig. 1b) maps nodes, edges, variables, node attributes, and edge attributes of G_1 to those of G_2. The mappings of m must be compatible with the source and target functions of G_1 and G_2 as usual and $G_2.\mathrm{ac}$ must imply the translation $m(G_1.\mathrm{ac})$ of $G_1.\mathrm{ac}$ for all variable valuations to ensure that m characterizes a restriction of attributes (cf. Fig. 1b where this implication is discussed). Moreover, the class of all finite graphs typed (as usual using a typing morphism) over a given type graph TG is given by $\mathcal{S}^{\mathrm{graphs}}_{\mathrm{fin},TG}$ or simply $\mathcal{S}^{\mathrm{graphs}}_{\mathrm{fin}}$ when TG is known. In the remainder, we only employ monomorphisms, written $m : G_1 \hookrightarrow G_2$, with only injective mappings. The unique monomorphism from the empty graph $\emptyset$ to a graph G is denoted $\mathrm{i}(G) : \emptyset \hookrightarrow G$. Finally, the special monomorphism $\mathsf{a}(G) : G' \hookrightarrow G$ describes that G' is obtained from G by setting the AC $G.\mathrm{ac}$ to true (i.e., G' equals G except that $G'.\mathrm{ac} = \top$).

The graph logic GL [14,26] supports the specification of the (non)existence of certain subgraphs in a given host graph G. Besides propositional operators for (finite) conjunction and negation, GL features the *exists* operator $\exists$, which specifies for a given match $m : H \hookrightarrow G$ of a (context) graph H into the host graph G that m can be extended to a match $m' : H' \hookrightarrow G$ by using a monomorphism $f : H \hookrightarrow H'$ that explains how H is extended to the (context) graph H'.

The graph G_2 from Fig. 1b does not satisfy ϕ_I because the initial monomorphism $\mathrm{i}(G_2) : \emptyset \hookrightarrow G_2$ can be extended to m from Fig. 1b, which is forbidden by the left part $\exists(\mathrm{i}(G_1), \top)$ of ϕ_I.

Definition 8 (Graph Logic (GL)). *If $H \in \mathcal{S}^{\mathrm{graphs}}_{\mathrm{fin}}$ is a finite graph, $m : H \hookrightarrow G$ is a monomorphism, then ϕ'* is a graph condition over *H, written $\phi' \in \mathcal{S}^{\mathrm{GC}}_H$, which is satisfied by m, written $m \models \phi'$, if an item applies.*

- $\phi' = \wedge S$, $S \subseteq_{\mathrm{fin}} \mathcal{S}^{\mathrm{GC}}_H$, *and (for satisfaction)* $\forall \phi \in S.\, m \models \phi$.
- $\phi' = \neg\phi$, $\phi \in \mathcal{S}^{\mathrm{GC}}_H$, *and (for satisfaction)* $m \not\models \phi$.
- $\phi' = \exists(f : H \hookrightarrow H', \phi)$, $\phi \in \mathcal{S}^{\mathrm{GC}}_{H'}$ *is a GC over the extended graph H', and (for satisfaction) there is $m' : H' \hookrightarrow G$ s.t. $m = m' \circ f$ and $m' \models \phi$.*

Moreover, we define the following notions.

- *Derived operators: (true)* $\top$*, (false)* $\bot$*, (disjunction)* $\vee S$*, and (for all)* $\forall(f, \phi)$*.*
- *Graph Satisfaction: If* $\phi \in \mathcal{S}_{\emptyset}^{\mathrm{GC}}$ *is a GC over the empty graph satisfied by the initial morphism* $\mathrm{i}(G)$ *(i.e.,* $\mathrm{i}(G) \models \phi$*) then* ϕ *is satisfied by* G*, written* $G \models \phi'$*.*
- *Satisfying morphisms: If* $\phi \in \mathcal{S}_{H}^{\mathrm{GC}}$ *is a GC, then* $[\![\phi]\!] = \{m : H \hookrightarrow G \mid m \models \phi\}$*.*

Moreover, we define that two GCs ϕ_1 and ϕ_2 are consistent, when ϕ_1 only describes elements also described by ϕ_2 or none of them.

Definition 9 (Consistent GCs). *If* $\{\phi_1, \phi_2\} \subseteq \mathcal{S}_{\emptyset}^{\mathrm{GC}}$ *and* $[\![\phi_1]\!] \cap [\![\phi_2]\!] \neq \emptyset$ *implies* $[\![\phi_1]\!] \subseteq [\![\phi_2]\!]$*, then* ϕ_1 is consistent with ϕ_2*, written* $\mathsf{cons}(\phi_1, \phi_2)$*.*

To check satisfiability of a GC and consistency of two GCs, we employ the automated reasoning technique for GL in the form of the algorithm $\mathcal{A}$ for which tool support is available in AutoGraph as introduced in [26]. The algorithm $\mathcal{A}$ takes a GC ϕ as input, is known to terminate for unsatisfiable GCs (i.e., it is refutationally complete), and incrementally generates the set of minimal graphs satisfying ϕ (this set is empty for unsatisfiable GCs). As for the case of AL and Z3, we carefully handle cases where $\mathcal{A}$ does not terminate and also generates no minimal graph as discussed later on.

Fact 1 (Algorithm $\mathcal{A}$). *If* $\phi \in \mathcal{S}_{\emptyset}^{\mathrm{GC}}$ *is a GC over the empty graph and* $\mathcal{A}$ *terminates for* ϕ*, it returns the finite set of all minimal graphs satisfying* ϕ*.*

The standard operation shift from [13] is also applicable to symbolic graphs [26]. It defines an adaptation of a GC ϕ with context graph H for a monomorphism $m : H \hookrightarrow H'$ resulting in an equivalent GC with context graph H' in the sense of the following fact (by considering how additional elements of H' may be used in a satisfaction proof for the given GC ϕ).

Fact 2 (Operation shift). *If* $m_1 : H \hookrightarrow H'$*,* $m_2 : H' \hookrightarrow H''$*, and* $\phi \in \mathcal{S}_{H}^{\mathrm{GC}}$*, then* $m_2 \circ m_1 \models \phi$ *iff* $m_2 \models \mathsf{shift}(m_1, \phi)$*.*

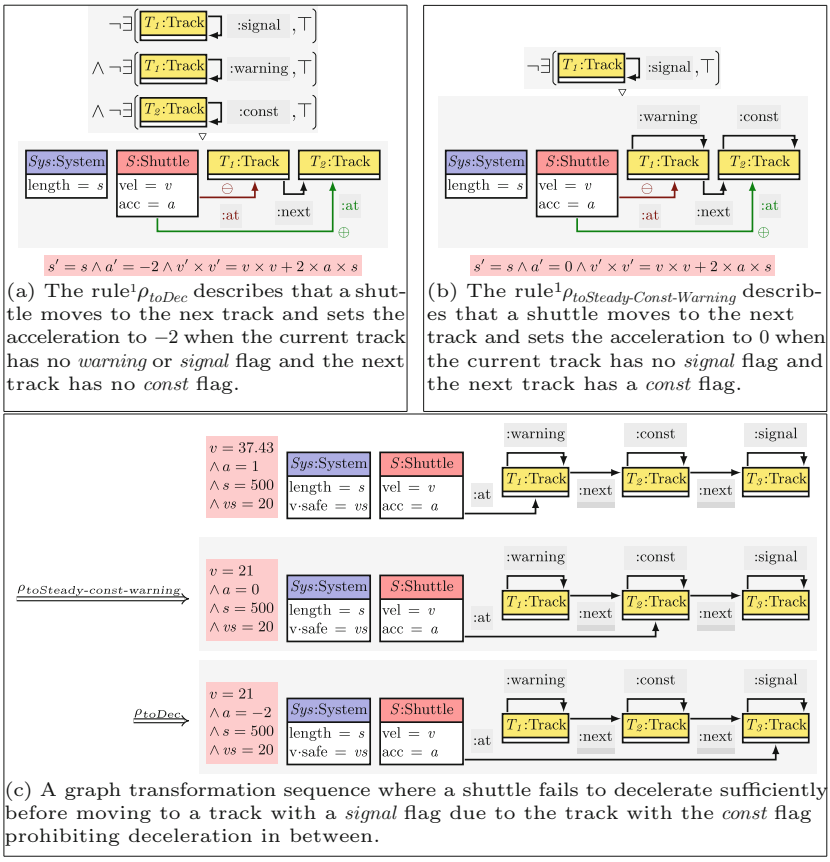
Graph transformation steps are defined using rules specifying structural and attribute transformations. A rule ρ contains for the structural part (as in the DPO approach) two monomorphisms $\rho.\mathrm{del} : K \hookrightarrow L$ and $\rho.\mathrm{add} : K \hookrightarrow R$ where K, $L - \rho.\mathrm{del}(K)$, and $R - \rho.\mathrm{add}(K)$ contain the preserved/deleted/added elements. For the attribute part, L, K, and R have the trivial ACs $\top$ and a rule ρ contains an AC $\rho.\mathrm{ac}$ instead, which is defined over the disjoint union V (i.e., the coproduct, written $\amalg$ where $\rho.\mathrm{lX}$ and $\rho.\mathrm{rX}$ map variables to the disjoint union V) of the variables of L and R. Intuitively, variables originating from L are used as unprimed variables and variables originating from R are used as primed variables. Finally, a rule contains left and right hand side application conditions $\rho.\mathrm{lC}$ and $\rho.\mathrm{rC}$ defined over the graphs L and R and checked during the transformation

as in the DPO approach. See Fig. 3 for two simple rules[1] and, for our running example, Fig. 2 for two of the total nine rules (see [8, Section C.1.6, p. 336] for a full description of the assumed invariants and rules of the considered GTS).

Definition 10 (Graph Transformation Rules). *If* $\rho.\text{del} : K \hookrightarrow L$, $\rho.\text{add} : K \hookrightarrow R$ *are to monomorphisms,* $\Pi(\rho.\text{lX} : L.\text{X} \hookrightarrow V, \rho.\text{rX} : R.\text{X} \hookrightarrow V)$ *is a coproduct,* $\rho.\text{ac} \in \mathcal{S}_V^{\text{AC}}$, $\rho.\text{lC} \in \mathcal{S}_L^{\text{GC}}$, $\rho.\text{rC} \in \mathcal{S}_R^{\text{GC}}$, *and* $L.\text{ac} = K.\text{ac} = R.\text{ac} = \top$, *then* $\rho = (\rho.\text{del}, \rho.\text{add}, \rho.\text{lX}, \rho.\text{rX}, \rho.\text{ac}, \rho.\text{lC}, \rho.\text{rC})$ is a rule, *written* $\rho \in \mathcal{S}^{\text{rules}}$. *Moreover, we define the following abbreviations.*

- $\rho.\text{lG} = L$ *and* $\rho.\text{rG} = R$ *are the left and right hand side graphs of the rule* ρ.
- $\mathcal{S}_{\text{fin}}^{\text{rules}}$ *is the set of all rules where* L, K, *and* R *are finite.*

Graph transformations systems then contain a finite set of finite rules (used for graph transformation steps) and initial states described by a GC.



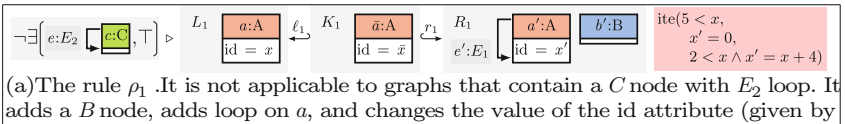
(a) The rule[1] ρ_{toDec} describes that a shuttle moves to the nex track and sets the acceleration to -2 when the current track has no *warning* or *signal* flag and the next track has no *const* flag.

(b) The rule[1] $\rho_{toSteady\text{-}Const\text{-}Warning}$ describes that a shuttle moves to the next track and sets the acceleration to 0 when the current track has no *signal* flag and the next track has a *const* flag.

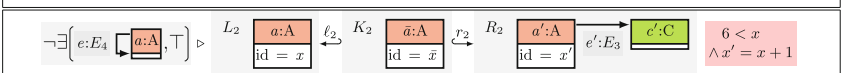
(c) A graph transformation sequence where a shuttle fails to decelerate sufficiently before moving to a track with a *signal* flag due to the track with the *const* flag prohibiting deceleration in between.

Fig. 2. Two rules and a graph transformation sequence for our shuttle scenario.

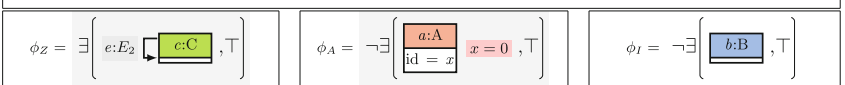
[1] Here, L, K, and R are given in a single graph and preserved/deleted/added elements are colored black/red/green and deleted/added elements are marked with $\ominus/\oplus$.



(a)The rule ρ_1 .It is not applicable to graphs that contain a C node with E_2 loop. It adds a B node, adds loop on a, and changes the value of the id attribute (given by variable x) of node a using an AC that uses the *if-then-else* operation ite.



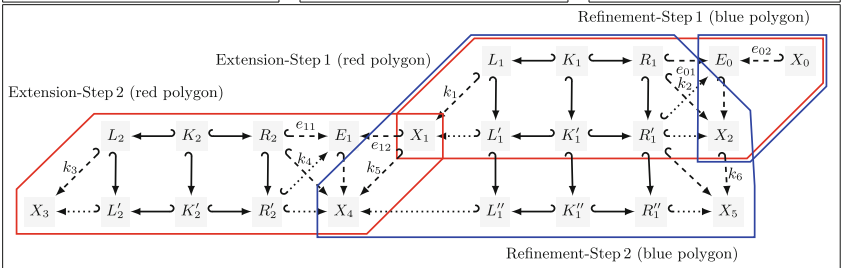
(b) The rule ρ_2. It is not applicable to graphs where the matched A node has an E_4 loop or an id attribute of at most 6. It adds a C node c', adds an edge from the matched A node to c',and increases the id attribute (given by variable x) of node a.



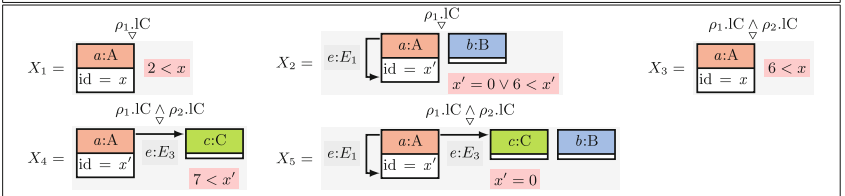
(c)The initial states ϕ_Z have a C node with E_2 loop.

(d)The assumed invariant ϕ_A states that no A node may have an id of 0.

(e)The invariant candidate ϕ_I states reachable graph contain no B node.



(f)The analysis using $\mathcal{I}$ starts with the path $\pi_0 = X_0$ of length 0 where $X0$=($\emptyset$,$\neg\phi I$). Using Ext, a path $\pi_1 = X_1 \cdot (k_1, \rho_1, k_2) \cdot X2$ of length 1 is constructed by extending π_0. Using Ext, a path $\pi_2 = X_3 \cdot (k_3, \rho_2, k_4) \cdot X_4 \cdot (k_5 \circ k_1, \rho_1, k_6 \circ k_2) \cdot X_5$ of length 2 is constructed by extending $\pi_1$where the second step of π_2 is obtained by refinement of π_1 via Ref.



(g) The abstract states $X_i = (G_i, \phi_i)$ from Figure 3f. To ease presentation, we use GCs such as ρ_1. lC on graphs different from L_1. The ACs of G_i are obtained according to the step relation considering the AC of the given source/target graph. (G_1) G_1.ac $\equiv \exists x' \,.\, \rho_1.\text{ac} \equiv 5 < x \vee 2 < x \equiv 2 < x$. ($G_2$) G_2.ac $\equiv \exists\, x.\rho_1.\,\text{ac} \wedge 2 < x \equiv x' = 0 \vee 6 < x'$. ($G_3$) G_3.ac $\equiv \exists x'.\ \rho_2.\text{ac} \wedge 2 < x' \equiv 6 < x$. ($G_4$) G_4.ac $\equiv \exists x.\ \rho_2.\text{ac} \wedge 6 < x \equiv 7 < x'$. ($G_5$) G_5.ac $\equiv \exists x.\ \rho_1.\text{ac} \wedge 7 < x \equiv x' = 0$.

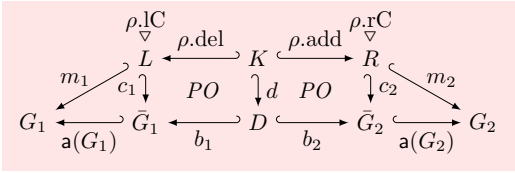
Fig. 3. Example of invariant analysis for abstract LTS.

Definition 11 (Graph Transformation System (GTS)). *If* $P \subseteq_{\text{fin}} \mathcal{S}^{\text{rules}}_{\text{fin}}$ *and* $\phi_Z \in \mathcal{S}^{\text{GC}}_{\emptyset}$*, then* (P, ϕ_Z) is a graph transformation system.

Deviating from [22], we now introduce a notion of graph transformation steps in which structural and attribute transformations are decoupled. The defined step relation is symmetric and supports the removal as well as addition of variables, which is also relevant when attribute values are to be modified.

Definition 12 (Steps). *There is a step* $G_1 \overset{\sigma}{\Longrightarrow} G_2$ *with label* σ*, whenever*

- *there is a rule* $\rho \in \mathcal{S}^{\text{rules}}_{\text{fin}}$ *with* $\rho.\text{lG} = L$ *and* $\rho.\text{rG} = R$ *as depicted below,*
- *the graph* L *can be matched to* G_1 *using* $m_1 : L \hookrightarrow G_1$ *that satisfies the left-hand side application condition* $\rho.\text{lC}$*,*
- *the graph* $\bar{G}_1$ *is obtained from* G_1 *by setting the AC of* G_1 *to* $\top$ *inducing the morphisms* c_1 *and* $\mathsf{a}(G_1)$ *compatible with* m_1*,*
- *the graphs* D *and* $\bar{G}_2$ *are constructed according to the double pushout approach as pushout complement and pushout from left to right,*
- *the graph* G_2 *is obtained from* $\bar{G}_2$ *by setting the AC of* G_2 *according to the AC* $\rho.\text{ac}$ *of the rule inducing morphisms* m_2 *and* $\mathsf{a}(G_2)$ *compatible with* c_2*, and*
- *the morphism* m_2 *satisfies the right-hand side application condition* $\rho.\text{rC}$*.*



For this construction, $\sigma = (\sigma.\text{rule}, \sigma.\text{drule}, \sigma.\text{match}, \sigma.\text{comatch}) = (\rho, \bar{\rho}, m_1, m_2)$ *is the used label where* $\bar{\rho}$ *is the* derived rule *(cf. [13]) with* $\bar{\rho}.\text{del} = b_1$*,* $\bar{\rho}.\text{add} = b_2$*,* $\bar{\rho}.\text{lC} = \mathsf{shift}(c_1, \rho.\text{lC})$*,* $\bar{\rho}.\text{rC} = \mathsf{shift}(c_2, \rho.\text{lC})$*, and where the AC* $\bar{\rho}.\text{ac}$ *is adapted from* $\rho.\text{ac}$ *according to the renamings of* c_1 *and* c_2*.*

For our running example, see Fig. 2c for a graph transformation sequence applying the two rules from Fig. 2a and Fig. 2b. Note that the last graph of this sequence violates the invariant candidate from Fig. 1c as the shuttle exceeds the permitted velocity on a track with *signal* flag.

The steps defined by this construction immediately induce a concrete LTS (see Definition 1) for a given GTS where the initial states are given by all graphs satisfying the GC characterizing initial graphs of the GTS.

Definition 13 (Concrete LTS of Graph Transformation). *If* (P, ϕ_Z) *is a GTS then* $\mathsf{cLTS}((P, \phi_Z)) = \Gamma = (S, L, \delta, Z)$ is the concrete LTS of (P, ϕ_Z) *with*

- $S = \mathcal{S}^{\text{graphs}}_{\text{fin}}$ *is the set of all finite graphs,*
- $L = \mathcal{S}^{\text{steps}}$ *is the set of all step labels,*
- $\delta = \{(G, \sigma, H) \mid G \overset{\sigma}{\Longrightarrow} H\}$ *is given by graph transformation steps of* (P, ϕ_Z)*,*
- $Z(\bar{G}) = \bar{G} \models \phi_Z$ *uses the GC satisfaction relation,*

Moreover, a state G of Γ (i.e., a finite graph) satisfies a state predicate (cf. the last item above) given by a GC ϕ defined over the empty graph $\emptyset$ iff $G \models \phi$.

Finally, the operations left and the reverse operation right introduced in [13] can be adapted to symbolic graphs. The operation left inductively propagates a GC ϕ over the right hand side graph $\rho.\mathrm{rG} = R$ (such as the application condition $\rho.\mathrm{lC}$) of a rule ρ to the left hand side graph $\rho.\mathrm{lG} = L$ of ρ by applying the renaming of graph elements according to $\rho.\mathrm{del}$ and $\rho.\mathrm{add}$ to the graphs in the GC ϕ. The two operations ensure the following compatibility with steps (cf. [13]).

Fact 3 (Operations left and right). *If $\rho \in \mathcal{S}^{\mathrm{rules}}_{\mathrm{fin}}$ is a finite rule with the left and right hand side graphs L and R, $\phi_L \in \mathcal{S}^{\mathrm{GC}}_{L}$ and $\phi_R \in \mathcal{S}^{\mathrm{GC}}_{R}$ are GCs over L and R, and $G \xRightarrow{(\rho,\bar{p},m,\bar{m})} H$ is a graph transformation step, then $\bar{m} \models \phi_R$ iff $m \models \mathsf{left}(\rho, \phi_R)$ and $m \models \phi_L$ iff $\bar{m} \models \mathsf{right}(\rho, \phi_L)$.*

4 Invariant Analysis for Graph Transformation Systems

Based on the preliminaries from the previous section on graph transformation and graph specification using GCs, we now apply our theory on k-induction from Sect. 2. Note that the instantiation presented here is specific to the step relation for graph transformation presented in the previous section due to the decoupling of transformation of structure and ACs. For this instantiation, we construct an LTS that is finitely backwards branching (see Definition 6) and that is related to the concrete LTS Γ from the previous section via a suitable LTSAR (see Definition 7) to permit an application of Theorem 2 for enabling the analysis of the GTS using $\mathcal{I}$ according to Theorem 1. For this purpose, we assume a fixed GTS (P, ϕ_Z), the induced LTS $\mathsf{cLTS}((P, \phi_Z)) = \Gamma$ (see Definition 13), an assumed invariant $\phi_A \in \mathcal{S}^{\mathrm{GC}}_{\emptyset}$, and an invariant candidate $\phi_I \in \mathcal{S}^{\mathrm{GC}}_{\emptyset}$.

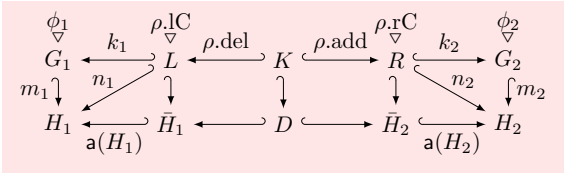
For demonstration purposes, we consider the GTS $(\{\rho_1, \rho_2\}, \phi_Z)$ with assumed invariant ϕ_A and invariant candidate ϕ_I from Fig. 3.

As an initial candidate for the LTS to be constructed, we define the LTS Γ' in which each state $(\bar{G}, \bar{\phi})$ is given by a GC $\bar{\phi}$ and the graph $\bar{G}$ over which $\bar{\phi}$ is defined for improved readability. The LTS Γ' induces an LTSAR in which the relation R_S contains pairs $(G, (\bar{G}, \bar{\phi}))$ for which some monomorphism $m : \bar{G} \hookrightarrow G$ with $m \models \bar{\phi}$ exists. The steps of Γ' adapt states (G_1, ϕ_1) to states (G_2, ϕ_2) using a rule ρ of the GTS (P, ϕ_Z) for matches $k_1 : \rho.\mathrm{lG} \hookrightarrow G_1$ and $k_2 : \rho.\mathrm{rG} \hookrightarrow G_2$ at the abstract level by considering all concrete steps of graphs H_1 and H_2 that are related to G_1 and G_2 via R_S (by means of instantiation morphisms m_1 and m_2). That is, the same rule ρ can be applied to each graph covered by (G_1, ϕ_1) and, vice versa, (G_2, ϕ_2) covers only the graphs reachable using such steps.

Definition 14 (Abstract LTS of GC Transformation). *If (P, ϕ_Z) is a GTS then $\mathsf{aLTS}((P, \phi_Z)) = \Gamma' = (S', L', \delta', Z')$ is the abstract LTS of (P, ϕ_Z) with*

- $S' = \{(\bar{G}, \bar{\phi}) \mid \bar{G} \in \mathcal{S}^{\mathrm{graphs}}_{\mathrm{fin}} \wedge \bar{\phi} \in \mathcal{S}^{\mathrm{GC}}_{\bar{G}}\}$,
- $L' = \{(k_1 : \rho.\mathrm{lG} \hookrightarrow G_1, \rho, k_2 : \rho.\mathrm{rG} \hookrightarrow G_2) \mid \rho \in P, \{G_1, G_2\} \subseteq \mathcal{S}^{\mathrm{graphs}}_{\mathrm{fin}}\}$,

- $((G_1,\phi_1),(k_1,\rho,k_2),(G_2,\phi_2)) \in \delta'$ *iff* $\rho \in P$, $k_1 : L \hookrightarrow G_1$, $k_2 : R \hookrightarrow G_2$,
  - $\forall m_1 \in [\![\phi_1]\!]. \exists m_2 \in [\![\phi_2]\!]. P(m_1 \circ k_1, m_2 \circ k_2, \rho)$ *and*
  - $\forall m_2 \in [\![\phi_2]\!]. \exists m_1 \in [\![\phi_1]\!]. P(m_1 \circ k_1, m_2 \circ k_2, \rho)$ *using the abbreviation* P*:*
  - $P(n_1,n_2,\rho) = (\exists \sigma.\, H_1 \overset{\sigma}{\Longrightarrow} H_2 \wedge \sigma.\mathrm{rule} = \rho \wedge \sigma.\mathrm{match} = n_1 \wedge \sigma.\mathrm{comatch} = n_2)$,



- $Z'((\bar{G},\bar{\phi})) = [\![\exists(\mathrm{i}(\bar{G}),\bar{\phi}) \wedge \phi_Z]\!] \neq \emptyset$.

Moreover, a state $(\bar{G},\bar{\phi})$ *of* Γ' *satisfies a state predicate (cf. the last item above) given by a GC* ϕ *defined over the empty graph* $\emptyset$ *iff* $[\![\exists(\mathrm{i}(\bar{G}),\bar{\phi}) \wedge \phi]\!] \neq \emptyset$.[2]

We state that each sub-LTS Γ'' of Γ' induces a certain LTSAR for the LTS Γ.

Lemma 1 (LTSAR for GTS). *If* (P,ϕ_Z) *is a GTS,* $\Gamma = \mathsf{cLTS}((P,\phi_Z))$, Γ'' *is a sub-LTS of* $\Gamma' = \mathsf{aLTS}((P,\phi_Z))$, $R_S = \{(G,(\bar{G},\bar{\phi})) \mid \exists m : \bar{G} \hookrightarrow G.\, m \models \bar{\phi}\}$, *and* $R_L = \{(\sigma,(k_1,\rho,k_2)) \mid \sigma.\mathrm{rule} = \rho \in P\}$, *then* $\Gamma \leq_{R_S,R_L} \Gamma''$ *by Definition 7.* □

Selecting the entire LTS $\Gamma'' = \Gamma'$ results in an LTSAR, which does not satisfy the requirements of Theorem 2 in general. Instead, we obtain a suitable sub-LTS Γ'' of Γ' in an on-the-fly manner during an application of $\mathcal{I}$ (see Definition 5): Γ'' then describes precisely the paths maintained by $\mathcal{I}^{\mathrm{inner}}$ in its parameter *paths* at any point in the computation. Hence, the initial candidate is the sub-LTS Γ''_0 that contains the single state $(\emptyset,\neg\phi_I)$ violating ϕ_I. Note that Γ''_0 induces an LTSAR satisfying the requirements R1–R5 already. See Fig. 3f where node X_0 represents this initial state inducing the path π_0 of length 0.

Inside an application of $\mathcal{I}^{\mathrm{inner}}(\Gamma',\phi_A,\phi_I,k,i,\mathit{paths})$ (see Definition 5), we extend paths in *paths* w.r.t. Γ' and thereby adapt Γ''_i to Γ''_{i+1} such that the LTSAR for Γ''_{i+1} (see Lemma 1) also satisfies the requirements R1–R5 of Theorem 2. The satisfaction of requirement R6 for the backwards simulation may require that further path extensions are computed in subsequent iterations of $\mathcal{I}^{\mathrm{inner}}$. In Fig. 3f, the path π_0 is extended to paths π_1 and π_2 where the last nodes X_2 and X_5 are then incrementally more specific than X_0 (w.r.t. the monomorphisms that satisfy their GCs).

When the application of $\mathcal{I}$ terminates with a definite result $b \in \{\mathsf{i},\mathsf{v}\}$, the obtained sub-LTS Γ''_i constructed up to this point induces an LTSAR, which meets the relevant requirements listed in Theorem 2. In particular (see also Theorem 3 later on), (a) for the result $(\mathsf{v},\mathit{paths})$ meaning that the invariant candidate ϕ_I is violated by Γ', we can apply *Part1* of Theorem 2 because R1–R5 are satisfied and (b) for the result $(\mathsf{i},\emptyset)$ meaning that ϕ_I is established as an invariant for Γ', we can apply *Part2* of Theorem 2 because there are no further backward

[2] Definition 16 resolves cases where $\exists(\mathrm{i}(\bar{G}),\bar{\phi})$ and ϕ are not consistent (Definition 8).

steps that require consideration since all paths constructed so far were discarded for not having any further relevant step implying also R6 as required.

In the remainder, we discuss the backwards construction of paths of Γ' for a GTS using the operation Ext. This extension operation (see Definition 16) entails a refining operation Ref (see Definition 15) used to adapt paths in line with the operation $\mathsf{ext}(\mathit{paths})$ used in $\mathcal{I}$ to ensure that the requirements R1–R5 are satisfied by the corresponding sub-LTS Γ''_{i+1} constructed so far.

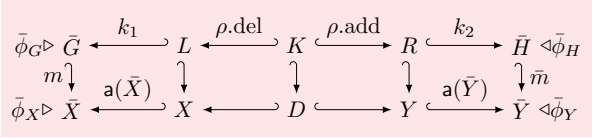
Extending a path π of Γ' starting in a state $(\bar{G}, \bar{\phi})$ adding a backwards step for a rule ρ may result in a refinement due to (a) additional graph elements when the comatch of the step does not only match elements of $\bar{G}$, (b) additional restrictions originating from the application conditions of ρ, and (c) fewer variable valuations satisfying the AC of the start graph of the path.

For example, in Fig. 3f, the path $\pi_1 = X_1 \cdot (k_1, \rho_1, k_2) \cdot X_2$ (in the second line) is refined to $X_4 \cdot (k_5 \circ k_1, \rho_1, k_6 \circ k_2) \cdot X_5$ according to the monomorphism $k_5 : G_1 \hookrightarrow G_4$ for the application of ρ_2 in *Extension-Step 2*. Considering the elements X_1 and X_4 given in more detail in Fig. 3g, we see that X_4 is much more specific than X_2 due to the additional GC originating from ρ_2, the inclusion of node c and edge e, and a more restrictive AC.

The following operation Ref refines the path π starting in $(\bar{G}, \bar{\phi}_G)$ to a path π' starting in $(\bar{X}, \bar{\phi}_X)$ for a monomorphism $m : \bar{G} \hookrightarrow \bar{X}$ and a GC $\bar{\phi}_X$ defined on $\bar{X}$, which describe the effect of the backwards step on π. It does so by adapting the monomorphisms contained in the labels of the steps in π, performs a step leading to a graph $\bar{Y}$ to propagate attribute restrictions given by the AC of $\bar{X}$, and propagates the additional GC $\bar{\phi}_X$ to the resulting graph $\bar{Y}$.

Definition 15 (Refinement of Abstract Paths). *If* $\Gamma' = (S', L', \delta', Z') \in \mathcal{S}^{\mathrm{lts}}$, $\pi \in \mathsf{paths}(\Gamma', n)$, $m : \bar{G} \hookrightarrow \bar{X}$, $\bar{\phi}_X \in \mathcal{S}^{\mathrm{GC}}_{\bar{X}}$, $\pi' \in \mathsf{paths}(\Gamma', n)$, *then* π' is the refinement of π via m and $\bar{\phi}_X$, *written* $\pi' = \mathsf{Ref}(\pi, m, \bar{\phi}_X)$, *if an item applies.*

- $n = 0$, $\pi = (\bar{G}, \bar{\phi}_G)$, *and* $\pi' = (\bar{X}, \bar{\phi}_X \wedge \mathsf{shift}(m, \bar{\phi}_G))$.
- $n > 0$, $\pi = (\bar{G}, \bar{\phi}_G) \cdot (k_1, \rho, k_2) \cdot \tilde{\pi}$, $\pi_{\mathsf{S}}(1) = (\bar{H}, \bar{\phi}_H)$, $\bar{X} \xRightarrow{(\rho, \bar{\rho}, m \circ k_1, \bar{m} \circ k_2)} \bar{Y}$, $\bar{\phi}_Y = \mathsf{shift}(\mathsf{a}(\bar{Y}), \mathsf{right}(\bar{\rho}, \exists(\mathsf{a}(\bar{X}), \bar{\phi}_X)))$, *and* $\pi' = (\bar{X}, \bar{\phi}_X \wedge \mathsf{shift}(m, \bar{\phi}_G)) \cdot (m \circ k_1, \rho, \bar{m} \circ k_2) \cdot \mathsf{Ref}(\tilde{\pi}, \bar{m}, \bar{\phi}_Y)$.



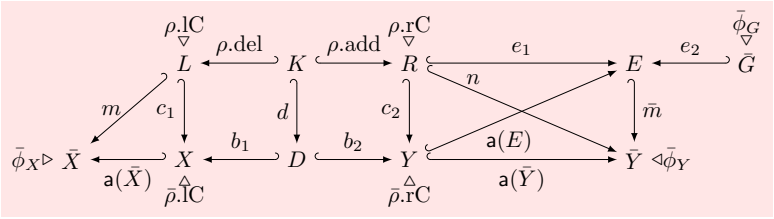
Concrete violating paths of Γ (such as in Fig. 2c for our running example) can be constructed from symbolic violating paths of Γ' starting in $(\bar{G}, \bar{\phi}_G)$ by (a) running the algorithm $\mathcal{A}$ from Fact 1 to obtain some monomorphism $m : \bar{G} \hookrightarrow \bar{X}$ satisfying $\bar{\phi}_G$, (b) employing Z3 to determine a variable valuation satisfying the AC of $\bar{X}$ resulting in some monomorphism $m' : \bar{G} \hookrightarrow \bar{Y}$, and (c) applying the operation Ref for m' and $\bar{\phi}_Y = \top$. Besides such concrete violating paths, we

return all *symbolic* violating paths to the user for which $\mathcal{A}$ or Z3 fail to determine definite results (which does not occur in the examples considered here).

We now introduce the operation Ext for extending a path of Γ' by adding a further backwards step. To ensure that we construct all paths, we follow the definition of E-concurrent rules from [13] to generate all minimal overlaps for each successive rule application and to adjust GCs to the application conditions of the rules. Moreover, in item (8), we employ the operation Ref to adapt the given path of Γ' to the additional step. Finally, in item (9), item (10), and item (11), we further split and filter the constructed paths to ensure that the state predicate satisfaction is compatible with R_S (see Theorem 3).

Definition 16 (Extension of Abstract Paths). *If (P, ϕ_Z) is a GTS, $\Gamma' = \mathsf{aLTS}((P, \phi_Z))$, $\pi \in \mathsf{paths}(\Gamma', n)$, then $\mathsf{Ext}(\pi)$ computes the possibly empty set of all path extensions $\pi' \in \mathsf{paths}(\Gamma', n+1)$ of π using the following procedure.*

(1) $(\bar{G}, \bar{\phi}_G)$ *is the first state of* π.
(2) $\rho \in P$ *is some rule of the GTS with* $\rho.\mathrm{lG} = L$ *and* $\rho.\mathrm{rG} = R$.
(3) $(e_1 : R \hookrightarrow E, e_2 : \bar{G} \hookrightarrow E) \in \mathcal{E}'$ *is a minimal overlapping of* R *and* $\bar{G}$ *(cf. [13]).*[3]
(4) $E \xRightarrow{(\mathsf{rev}(\rho), \bar{\rho}, e_1, m)} \bar{X}$ *is a step of the GTS where* ρ *is reversed using* rev *and applied forwards to* E *using match* e_1 *to obtain the required backwards step.*
(5) $\bar{\phi}_X = \mathsf{shift}(\mathsf{a}(\bar{X}), \bar{\rho}.\mathrm{lC} \wedge \mathsf{left}(\bar{\rho}, \bar{\rho}.\mathrm{rC} \wedge \exists(\mathsf{a}(E), \mathsf{shift}(e_2, \bar{\phi}_G))))$ *is the GC for* $\bar{X}$ *obtained using GC propagation as in [13].*
(6) $\bar{X} \xRightarrow{(\rho, \bar{\rho}, m, \bar{m} \circ e_1)} \bar{Y}$ *is a step of the GTS using the rule* ρ *possibly further restricting the AC from* E *to* $\bar{Y}$.
(7) $\bar{\phi}_Y = \mathsf{shift}(\mathsf{a}(\bar{Y}), \bar{\rho}.\mathrm{rC} \wedge \mathsf{right}(\bar{\rho}, \bar{\rho}.\mathrm{lC})) \wedge \mathsf{shift}(\bar{m} \circ e_2, \bar{\phi}_G)$ *is the GC for* $\bar{Y}$ *obtained using GC propagation as in [13].*
(8) $\tilde{\pi}_0 = (\bar{X}, \bar{\phi}_X) \cdot (m, \rho, n = \bar{m} \circ e_1) \cdot \mathsf{Ref}(\pi, \bar{m} \circ e_2, \bar{\phi}_Y)$ *is obtained by prepending the new step to the path refinement of* π *according to* $\bar{m} \circ e_2$ *and* $\bar{\phi}_Y$.



(9) (Disambiguation of Abstraction for ϕ_I) If $\exists(\mathrm{i}(\bar{X}), \bar{\phi}_X)$ is consistent with ϕ_I (see Definition 8), which can be checked using $\mathcal{A}$, we know that $(\bar{X}, \bar{\phi}_X)$ either only covers graphs satisfying ϕ_I or no such graphs. In this case, $\tilde{\pi}_1$ is $\mathsf{Ref}(\tilde{\pi}, \mathrm{id}(\bar{X}), \phi_I)$ or $\mathsf{Ref}(\tilde{\pi}, \mathrm{id}(\bar{X}), \neg\phi_I)$ (where $\mathrm{id}(\bar{X})$ is the identity morphism on $\bar{X}$) and $\tilde{\pi}_1 = \tilde{\pi}$ otherwise.
(10) (Disambiguation of Abstraction for ϕ_Z) Analogous to item (9) for the GC ϕ_Z representing the initial state of the GTS at hand obtaining $\tilde{\pi}_2$ from $\tilde{\pi}_1$.

[3] $\mathcal{E}'$ denotes the set of pairs of monomorphisms that are jointly epimorphic, that is, two monomorphisms that map to each graph element of their common target graph.

Table 1. Results of invariant analysis for the abstract LTS for shuttle scenario.

			Outcome $(b, paths)$ of analysis algorithm $\mathcal{I}$	
Lookahead n	Path length k	Duration	Element b	Size of element *paths*
2	2	1 s	u	6
3	3	2 s	u	12
4	4	12 s	u	8
5	5	63 s	i	0

(11) (Nonemptyness of Abstraction) If $\mathcal{A}$ and Z3 *determine that $\tilde{\pi}_2$ represents at least one concrete violation (as discussed subsequent to Definition 15) compatible with ϕ_A, then π' is equal to $\tilde{\pi}_2$ (otherwise $\tilde{\pi}_2$ results in no path extension).*

Figure 3f depicts two applications of Ext both requiring applications of Ref (the first refinement regarding the empty path π_0 is trivial and the second has been discussed above). The first extension uses ρ_1 from Fig. 3a, constructs the overlapping E_0 where the two B nodes are identified (not explicitly depicted), applies the reversal of rule ρ_1 using the match e_{01} to obtain X_1, and then applies ρ_1 to obtain the AC refinement $X_2 = (G_2, \phi_2)$ of E_0 depicted in Fig. 3g. Note that $X_2 = (G_2, \phi_2)$ still violates the invariant candidate ϕ_I (for all monomorphisms $m : G_2 \hookrightarrow H$). The further extension using ρ_2 then results in path π_2 ending in $X_5 = (G_5, \phi_5)$, which does not need to be considered further as X_5 violates the assumed invariant ϕ_A (for all monomorphisms $m : G_5 \hookrightarrow H$).

Finally, $\mathcal{I}$ from Definition 5 can be used to check a GTS against an invariant candidate ϕ_I by applying $\mathcal{I}$ using the described instantiation.

Theorem 3 (Instantiation of k-Induction for GTSs). *If (P, ϕ_Z) is a GTS, $\phi_A \in \mathcal{S}_\emptyset^{\mathrm{GC}}$ is an assumed invariant, $\phi_I \in \mathcal{S}_\emptyset^{\mathrm{GC}}$ is an invariant candidate, $k \in \mathbf{N}$, and the application of the algorithm $\mathcal{I}$ using the described instantiation Γ' for Γ,* Ext *(from Definition 16) for* ext*, and $\{(\emptyset, \neg\phi_I)\}$ for $paths_0$ terminates with $(b, paths)$, then Theorem 1 and Theorem 2 are applicable and $(b, paths)$ is a sound judgement on whether ϕ_I is an invariant for (P, ϕ_Z).*

Proof. The used operation Ext for path extension ensures that the last computed sub-LTS Γ'' of Γ' results in an LTSAR (see Lemma 1) meeting the requirements R1–R5 from Theorem 2 as follows (by induction on the parameter k for R4).

- Requirements *R1* and *R2* (preservation of invariant and initial state): Ensured by item (9) and item (10) in Definition 16.
- Requirement *R3* (R_S is right total): Ensured by item (11) in Definition 16.
- Requirement *R4* (R_S is left total on violating states): R4 means that each state G that violates ϕ_I in Γ via some shortest violation is covered by some state $(\bar{G}, \bar{\phi}_G)$ of Γ''. R4 is obviously satisfied by the initial LTS candidate that has the only state $(\emptyset, \neg\phi_I)$. Moreover, every extension (entailing the described refinement) of the set of paths in each iteration preserves this property because the refinement only excludes paths that are known to be only covering paths not representing shortest violations.

- Requirement *R5* (forward steps of Γ' are simulated by Γ): Ensured by applying the path refinement operation Ref in the operation Ref.

Lastly, the requirement R6 is satisfied for all states that are not at the beginning of a path in Γ'' since Ext considers all possible backward steps. □

For our running example, we applied our prototypical implementation of the analysis algorithm $\mathcal{I}$. For $k = 2$, we obtained the indefinite result $(\mathsf{u}, \mathit{paths})$ where the sequence from Fig. 2c is a concretization of a path in *paths* that could not be ruled out. As stated in Table 1, a path length of $k = 5$ (i.e., 5-induction) was required to establish that ϕ_I is an invariant. While the time required for invariant analysis increases exponentially with longer values of k due to the exponentially increasing number of paths of that length, we believe that the analysis times required for the running example already demonstrate feasibility albeit a potential for further optimizations of our prototypical implementation. Also note that the number of path extensions in each step grows exponentially with the size of the rules.

5 Conclusion and Future Work

We formalized the static analysis approach of k-induction using Isabelle for the abstract setting of LTSs establishing sufficient conditions for the preservation/reflection of invariants by means of an abstraction relation. We then applied this analysis approach to typed attributed GTSs by abstracting graphs by nested graph conditions (GCs) and by applying k-induction on these GCs. Our results extend the state of the art by permitting *attributes* as well as *nested* GCs for the specification of initial states, assumed invariants, and invariant candidates.

In the future, we want to develop support for probabilistic/timed GTSs such as [16]. Moreover, we strive to develop further abstractions to improve support for GTSs with multiple active components such as shuttles. Finally, heuristics guiding the computation of paths in the analysis procedure using parameterizations may improve performance by e.g. prioritizing path extension over checking for violations of attribute constraints of assumed invariants.

References

1. Augur 2 (2008). http://www.ti.inf.uni-due.de/en/research/tools/augur2
2. Graphs for Object-Oriented Verification (GROOVE) (2011). http://groove.cs.utwente.nl
3. EMF Henshin (2013). http://www.eclipse.org/modeling/emft/henshin
4. Becker, B., Giese, H.: 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, pp. 203–210. IEEE Computer Society (2008). https://doi.org/10.1109/ISORC.2008.13

5. Boneva, I., Rensink, A., Kurbán, M.E., Bauer, J.: Graph abstraction and abstract graph transformation. Technical report LNCS4549/TR-CTIT-07-50, July 2007
6. Corradini, A., Heindel, T., König, B., Nolte, D., Rensink, A.: Rewriting abstract structures: materialization explained categorically. In: Bojańczyk, M., Simpson, A. (eds.) FoSSaCS 2019. LNCS, vol. 11425, pp. 169–188. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-17127-8_10
7. Dyck, J.: Increasing expressive power of graph rules and conditions and automatic verification with inductive invariants. Master's thesis, University of Potsdam, Hasso Plattner Institute, Potsdam, Germany (2012)
8. Dyck, J.: Verification of graph transformation systems with k-inductive invariants. Ph.D. thesis, University of Potsdam, Hasso Plattner Institute, Potsdam, Germany (2020). https://doi.org/10.25932/publishup-44274
9. Dyck, J., Giese, H.: Inductive invariant checking with partial negative application conditions. In: Parisi-Presicce, F., Westfechtel, B. (eds.) ICGT 2015. LNCS, vol. 9151, pp. 237–253. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-21145-9_15
10. Dyck, J., Giese, H.: k-inductive invariant checking for graph transformation systems. In: de Lara, J., Plump, D. (eds.) ICGT 2017. LNCS, vol. 10373, pp. 142–158. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-61470-0_9
11. Dyck, J., Giese, H.: k-inductive invariant checking for graph transformation systems. Technical report 119, Hasso Plattner Institute at the University of Potsdam, Potsdam, Germany (2017)
12. Ehrig, H., Ehrig, K., Prange, U., Taentzer, G.: Fundamentals of Algebraic Graph Transformation. Springer, Berlin (2006). https://doi.org/10.1007/3-540-31188-2
13. Ehrig, H., Golas, U., Habel, A., Lambers, L., Orejas, F.: $\mathcal{M}$-adhesive transformation systems with nested application conditions. part 1: parallelism, concurrency and amalgamation. Math. Struct. Comput. Sci. **24**(4) (2014). https://doi.org/10.1017/S0960129512000357
14. Habel, A., Pennemann, K.: Correctness of high-level transformation systems relative to nested conditions. Math. Struct. Comput. Sci. **19**(2), 245–296 (2009). https://doi.org/10.1017/S0960129508007202
15. Kulcsár, G.: A compass to controlled graph rewriting. Ph.D. thesis, Technische Universität Darmstadt, January 2019. http://tuprints.ulb.tu-darmstadt.de/9304/
16. Maximova, M., Giese, H., Krause, C.: Probabilistic timed graph transformation systems. J. Log. Algebr. Meth. Program. **101**, 110–131 (2018). https://doi.org/10.1016/j.jlamp.2018.09.003
17. Microsoft Corporation: Z3. https://github.com/Z3Prover/z3
18. Orejas, F.: Attributed graph constraints. In: Ehrig, H., Heckel, R., Rozenberg, G., Taentzer, G. (eds.) ICGT 2008. LNCS, vol. 5214, pp. 274–288. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-87405-8_19
19. Orejas, F.: Symbolic graphs for attributed graph constraints. J. Symb. Comput. **46**(3), 294–315 (2011). https://doi.org/10.1016/j.jsc.2010.09.009
20. Orejas, F., Lambers, L.: Delaying constraint solving in symbolic graph transformation. In: Ehrig, H., Rensink, A., Rozenberg, G., Schürr, A. (eds.) ICGT 2010. LNCS, vol. 6372, pp. 43–58. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15928-2_4
21. Orejas, F., Lambers, L.: Symbolic attributed graphs for attributed graph transformation. ECEASST **30** (2010). http://journal.ub.tu-berlin.de/index.php/eceasst/article/view/405
22. Orejas, F., Lambers, L.: Lazy graph transformation. Fundam. Inform. **118**(1–2), 65–96 (2012). https://doi.org/10.3233/FI-2012-706

23. Pennemann, K.: Development of correct graph transformation systems. Ph.D. thesis, University of Oldenburg, Germany (2009). http://oops.uni-oldenburg.de/884/. URN http://nbn-resolving.de/urn:nbn:de:gbv:715-oops-9483
24. Poskitt, C.M., Plump, D.: Verifying monadic second-order properties of graph programs. In: Giese, H., König, B. (eds.) ICGT 2014. LNCS, vol. 8571, pp. 33–48. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-09108-2_3
25. Rensink, A.: Canonical graph shapes. In: Schmidt, D. (ed.) ESOP 2004. LNCS, vol. 2986, pp. 401–415. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24725-8_28
26. Schneider, S., Lambers, L., Orejas, F.: Automated reasoning for attributed graph properties. STTT **20**(6), 705–737 (2018). https://doi.org/10.1007/s10009-018-0496-3
27. Steenken, D.: Verification of infinite-state graph transformation systems via abstraction. Ph.D. thesis, University of Paderborn (2015). https://nbn-resolving.de/urn:nbn:de:hbz:466:2--15768
28. Steenken, D., Wehrheim, H., Wonisch, D.: Sound and complete abstract graph transformation. In: Simao, A., Morgan, C. (eds.) SBMF 2011. LNCS, vol. 7021, pp. 92–107. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-25032-3_7
29. Wang, X., Büttner, F., Lamo, Y.: Verification of graph-based model transformations using alloy. ECEASST **67** (2014). https://doi.org/10.14279/tuj.eceasst.67.943