

Proof Tactics for Theorem Proving Graph Grammars through Rodin¹

Luiz Carlos Lemos Jr.²
Simone A. da Costa Cavalheiro²
Luciana Foss²

Data submissão: 20.09.2014

Data aceitação: 24.03.2015

Seção melhores artigos WEIT-2013, Rio Grande, Brasil, 2013.

Abstract: Graph grammar is a formal language suitable for the specification of distributed and concurrent systems. Theorem proving is a technique that allows the verification of systems with huge (and infinite) state space. One of the disadvantages of theorem proving graph grammars (and theorem proving in general) is the specific mathematical knowledge required from the user for concluding the proofs. Previous works have proposed proof strategies to help the developer in the verification process when adopting such approach, firstly establishing proof tactics for some properties and after proposing a visual representation for them. This paper extends the set of proposed tactics, with the aim of expanding the available strategies and encouraging the use of such a technique.

1 Introduction

Graph grammars (GG) [13] are one of the specification languages used to precisely describe a computer system. It is a visual language, based on simple operations of rewriting rules. The states of the system are modelled by means of graphs (state graphs) and the state changes are specified through graph rules.

Theorem proving [20] is a formal verification method that allows the developer to guarantee some properties of a formal specified system. In this method both the system and its properties are described using mathematical language and logic, and the proofs are carried out through axioms and intermediate lemmas of the system. The adoption of such technique allows the analysis of systems with huge (and even infinite) state space [10]. Proof assistants (or interactive theorem provers) are tools that assist the developers when adopting the theorem

¹This work is an extension of the paper published in WEIT 2013 indicated as one of the best articles of the event.

²Universidade Federal de Pelotas, Centro de Desenvolvimento Tecnológico Rua Gomes Carneiro – 1 – 96010-610 – Pelotas – RS – Brazil

{lclemos, simone.costa, lfoss@inf.ufpel.edu.br}

proving technique.

A previous translation [8] of GG into Event-B [11] structures has allowed the use of provers available for the Rodin platform [3] for theorem proving GG systems. After modelling a GG in Event-B, the tool makes a syntactic and dynamic verification, which generate proof obligations. These obligations are stated to ensure that invariants are preserved, guard conditions and actions are well defined, formulas are meaningful, among others. Some of these obligations are discharged automatically or simply running an external prover (that can be installed in the form of a plug-in), others need user interaction. Proof strategies have been developed [15] to help the users to discharge proof obligations generated by a GG specification in Rodin. Also, an alternative visual representation for the defined tactics was proposed [14], turning the use of them even more intuitive and user friendly.

Besides, in previous work [9], it was proposed patterns for the presentation, codification and reuse of property specifications over reachable states of graph grammars. Especially, in [15], it was presented the proof obligations generated by a GG specification in Rodin and proposed proof strategies to specific atomic properties defined in the previous patterns. In this paper, we extend the results presented in [14] and [15]. With respect to the original versions, we enlarge the set of proposed tactics, with the aim of expanding the available strategies and encouraging the use of such a technique, including strategies to new types of properties (which involve logical operators not used in previous specifications). We prioritise properties that describe typical characteristics of graphs (basic properties of the set of patterns), maximising the possibility of reuse in case of more complex specifications.

Other authors have investigated the analysis of graph transformation systems (GTSs) based on relational logic or set theory. Baresi and Spoletini [7] explore the formal language Alloy to find instances and counterexamples for models and GTSs. With Alloy, they only analyse a system for a finite scope, whose size is user-defined. Strecker [21, 22, 23] has proposed a formalisation of graph transformations in a set-theoretic model using the Isabelle/HOL [16] proof assistant. In [21] he started to define a language for writing graph transformation programs and reasoning about them. The language was defined with two statements, one to apply a rule repeatedly to a graph, and another to apply several rules in a specific order to a graph. In [22] and [23] he explores how to reason locally about global properties of graph transformations. In particular, he investigate under which conditions reasoning about a graph transformation can be reduced to reasoning about the shape of the transformation rules. Tran and Percebois [24] has also used a relational and logical representation of graph grammars in Isabelle to verify graph transformation systems. They statically verify if a transformation preserves a particularly property of a state graph, that is, they investigate general conditions for a graph grammar to preserve structural properties in a rule application. However, they only argue about the preservation of a property in one derivation step.

The remaining of this paper is organised as follows. In Section 2 we present an

overview on graph grammars and in Section 3 we briefly show the description of GG in Event-B. Section 4 details the visual representation for the tactics proposed in [15]. Section 5 establishes proof tactics for discharging a new set of properties and Section 6 contains final remarks.

2 Graph Grammars

Graph grammars are suitable for specifying complex situations, which present several elements and many relations between them. The basic idea of a GG consists on specifying the states of a system as graphs, called state graphs; and the possible state changes as graph rewriting rules. Graph rules are used to capture the dynamical aspects of the systems, that is, from the initial state, rule applications successively change the system state.

A graph grammar consists of a *type graph*, an *initial graph* and a *set of rules*. The *type graph* characterises the types of vertices and edges allowed in the system. The *initial graph* represents the initial state of the system and the *set of rules* describes the possible state changes that can occur. A rule has a left-hand side (LHS) and a right-hand side (RHS), which are both graphs, and a partial graph morphism that connects these graphs in a compatible way and determines what should be modified by the rule applications.

A rule is applicable in a graph if there is a match, that is, an image of the LHS of the rule in the graph. Roughly speaking, this means that all items (vertices and edges) belonging to the LHS must be present at the current state. Each rule application transforms a state graph in the following way: all items mapped from the LHS to the RHS (via a graph morphism) must be preserved; all items not mapped must be deleted from the current state; and all items present in the RHS that are not image of the LHS must be added to the current state to obtain the next one.

We show the use of GG specifying a simple mobile system depicted in Figure 1. The system consists of a network of interconnected antennas and mobile users. Each antenna has a maximal capacity of simultaneous connections, which blocks new connections. The type graph T describes two types of nodes: Ant (antenna) and Usr (user); and five types of edges: Acn (connections between antennas), Ucn (connection between users and antennas), Cal (connections between users), Cn (available user connections for antenna) and Main (main antenna). The initial graph G_0 specifies a system with two antennas and two users. Each antenna has two more available connections, given by Cn edges. The system behavior is described by the *set of rules*. Each user, connected to a single antenna, may start (rule r1) or finish (rule r4) a call. New antennas (rule r2) and users (rule r5) can be added to the system at any time. Also, a user may be switched to another antenna (rule r3) and new connections between antennas can be established at any moment (rule r6).

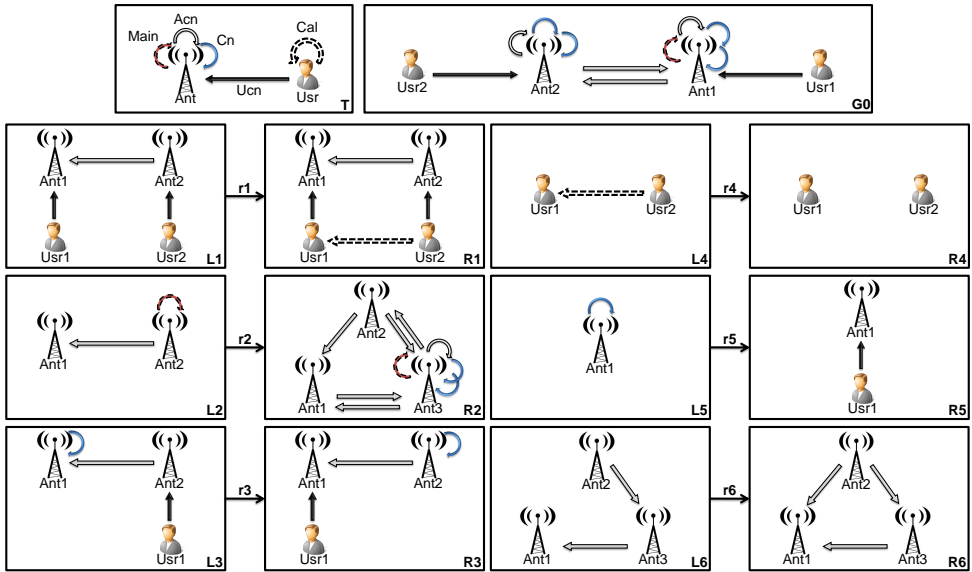


Figure 1. Mobile System GG

3 Graph Grammars in Event-B

A translation of graph grammar structures in Event-B language [19, 8] has allowed the use of Rodin platform [3] to prove properties of GG systems.

3.1 Mathematical Notation

In this section, we define the symbols and operations used in the remainder of the paper. Table 1 shows these symbols and corresponding meanings.

3.2 Event-B

Event-B [4, 2] is a state-based formalism closely related to Classical B [1] and Action Systems [5]. It is a method for system-level modelling and analysis that is intended for modelling and reasoning about systems that may consist of physical components, electronics and software. Moreover, it admits a notion of refinement, this means that complex interactions between subcomponents may be abstracted from in early stage modelling and then introduced through refinement in incremental stages.

Table 1. Definition of symbols and operations

Symbol/Operation	Meaning
\rightarrow	Partial functions
\rightarrow	Total functions
\rightsquigarrow	Total and injective functions
\mapsto	Mapping relation
\mathbb{N}	Set of natural numbers
\mathbb{Z}	Set of integer numbers
\mathbb{P}	The set of all subsets (power set)
\setminus or $-$	Set minus operator
$:=$	Becomes equal to
$rng(r)$	Range of a binary relation r
$dom(r)$	Domain of a binary relation r
$card(A)$	Number of elements of set A
\triangleleft	$A \triangleleft r \stackrel{def}{=} \{(a, b) \in r \mid a \notin A\}$ (Domain subtraction)
\triangleright	$r \triangleright B \stackrel{def}{=} \{(a, b) \in r \mid b \in B\}$ (Range restriction)

An Event-B specification contains two parts, a *context*, which represents the static part of the model and a *machine*, that represents the dynamic part of the model. In the context are defined sets, constants and axioms. While in the machine are defined variables, invariants and events.

Definition 1 (Event-B Model, Event) *An Event-B Model is defined by a tuple (c, s, P, v, I, R_I, E) , where c are constants and s are sets of the model; $P(c, s)$ is a collection of axioms constraining c and s ; v are the model variables; $I(c, s, v)$ is a model invariant limiting the possible states of v , s.t. $\exists c, s, v \cdot P(c, s) \wedge I(c, s, v)$; and E is a set of model events. A **context** defines the static part (c, s, P) and a **machine** defines the other elements (v, I, R_I, E) of an Event-B model.*

Given states v, v' **an event** is a tuple $e = (H, S)$ where $H(c, s, v)$ is the guard and $S(c, s, v, v')$ is the before-after predicate that defines a relation between current and next states.

In order to demonstrate the correctness of the model, several proof obligations must be discharged. The model consistency condition states that whenever an event or an initialisation action is attempted, there exists a suitable new state such that the model invariant is maintained.

The behaviour of an Event-B model is the transition system defined as follows.

Definition 2 (Event-B Model Behaviour) *Given an Event-B model $M = (c, s, P, v, I, R_I, E)$, its behaviour is given by a transition system $T = (St, S_0, \Rightarrow)$, where: $St = \{\langle v \rangle | v \text{ is a state}\} \cup Undef$, $S_0 = Undef$, and $\Rightarrow \subseteq St \times St$ is the transition relation given by the rules:*

$$\boxed{\text{start}} \frac{R_I(v') \wedge I(v')}{Undef \Rightarrow \langle v' \rangle}$$

$$\boxed{\text{transition}} \frac{\exists (H, S) \in E \cdot I(v) \wedge H(v) \wedge S(v, v') \wedge I(v')}{\langle v \rangle \Rightarrow \langle v' \rangle}$$

According to *start* rule, the model is initialised to a state satisfying $R_I \wedge I$ and then, as long as there is an enabled event (*transition* rule), the model may evolve by firing this event and computing the next state according to the event's before-after predicate. Events are atomic. In case there is more than one enabled event at a certain state, the demonic choice semantics applies. The semantics of an Event-B model is given in the form of proof semantics, based on Dijkstra's work on weakest preconditions [12].

3.3 Translation of Graph Grammars to Event-B Models

The behaviour of an Event-B model is similar to that of a GG: both have the concept of state (given by variables in Event-B and by a graph in a GG) and both have state transitions defined by atomic operations (defined by an event that updates variables in Event-B and by a rule application in a GG). Each transition should preserve properties of the state. In Event-B, these properties are stated as invariants and in a GG, they are related to the graph structure (only well-formed graphs can be generated). In both, GG and Event-B, a graph is defined by one set of vertices, one set of edges, and by four functions: source and target total functions, mapping each edge into source and target vertices, respectively; and two typing functions, mapping each vertex and edge to its type, respectively.

In order to specify a GG using Event-B, in the context are defined the type graph and the rules. In the machine we specify the variables that define a state graph, together with their types, the initial graph and the rule applications, the last two using events. In the initialisation event is stated the initial values of the variables (describing the initial state). Figure 2 illustrates the description of the state graph and the initial graph of the mobile system presented in Figure 1, renaming vertices and edges to natural numbers. For more details, see [19, 8].

Other events describe rule applications. The application of rule $r1$ of Figure 1 must be specified as detailed in Figure 3. Whenever there are concrete values for variables mV , mE ,

VARIABLES

```

vertG // (Graph) Vertices
edgeG // (Graph) Edges
sourceG // (Graph) Source Function
targetG // (Graph) Target Function
tG_V // Typing of vertices
tG_E // Typing of edges
    
```

INVARIANTS

```

inv_vertG : vertG ∈ ℙ(N)
inv_incG : edgeG ∈ ℙ(N)
inv_sourceG : sourceG ∈ edgeG → vertG
inv_targetG : targetG ∈ edgeG → vertG
inv_tG_V : tG_V ∈ vertG → vertT
inv_tG_E : tG_E ∈ edgeG → edgeT
    
```

EVENTS
Initialisation

```

act_vertG : vertG := {1, 2, 3, 4}
act_edgeG : edgeG := {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11}
act_srcG : sourceG := {1 ↦ 1, 2 ↦ 2, 3 ↦ 2, 4 ↦ 2, 5 ↦ 3, 6 ↦ 3, 7 ↦ 3, 8 ↦ 3, 9 ↦ 2, 10 ↦ 2, 11 ↦ 3}
act_tgtG : targetG := {1 ↦ 2, 2 ↦ 2, 3 ↦ 3, 4 ↦ 2, 5 ↦ 3, 6 ↦ 3, 7 ↦ 3, 8 ↦ 3, 9 ↦ 2, 10 ↦ 2, 11 ↦ 3}
act_tG_V : tG_V := {1 ↦ Ustr, 2 ↦ Ant, 3 ↦ Ant, 4 ↦ Ustr}
act_tG_E : tG_E := {1 ↦ Ucn, 2 ↦ Acn, 3 ↦ Acn, 4 ↦ Acn, 5 ↦ Acn, 6 ↦ Ucn, 7 ↦ Cn, 8 ↦ Cn, 9 ↦ Cn, 10 ↦ Cn, 11 ↦ Main}
    
```

Figure 2. State graph and Initial graph G_0 in Event-B

$newEcal$ that satisfy the guard conditions, the event may occur. Guard conditions assure that the pair mV and mE is actually a match from the LHS of the rule to the state graph. Guard condition $grd_newEcal$ guarantees that $newEcal$ is a new fresh element in the graph. The actions update the state graph according to the rule. In this case, one Cal edge is created.

Any change in the values of variables generates proof obligations to be demonstrated. The proof obligations generated by a GG specification are directly demonstrated just by running the available provers in Rodin. Besides the GG specification, we can also establish other properties to the model, stating them as invariants (considering that they are valid for all reachable states of the system). In order to verify each invariant, a proof obligation is generated for the initial state and for each rule of the grammar. However, in general, the discharging of them requires user interaction and it is not a trivial task.

```

Event rule1
  any
    mV
    mE
    newEcal
  where
    grd_mV : mV ∈ vertL1 → vertG
    grd_mE : mE ∈ edgeL1 ↦ edgeG
    grd_newEcal : newEcal ∈ ℕ \ edgeG
    grd_vertices : ∀v.v ∈ vertL1 ⇒ tL1_V(v) = tG_V(mV(v))
    grd_edges : ∀e.e ∈ edgeL1 ⇒ tL1_E(e) = tG_E(mE(e))
    grd_srctgt : ∀e.e ∈ edgeL1 ⇒ mV(sourceL1(e)) = sourceG(mE(e)) ∧
      ∧ mV(targetL1(e)) = targetG(mE(e))
  then
    act_E : edgeG := edgeG ∪ {newEcal}
    act_src : sourceG := sourceG ∪ {newEcal ↦ mV(Usr2)}
    act_tgt : targetG := targetG ∪ {newEcal ↦ mV(Usr1)}
    act_tG_E : tG_E := tG_E ∪ {newEcal ↦ Cal}
  end
    
```

Figure 3. Rule application in Event-B

3.4 Rodin Platform

The Rodin Platform [3] is an Eclipse-based IDE that provides a core functionality for syntactic analysis and proof-based verification of Event-B models. The platform is open source, contributes to the Eclipse framework and is further extensible with plug-ins which support features such as model checking, model animation, graphical front ends, additional proof capabilities and code generation.

The main verification technique is theorem proving supported by a collection of theorem provers. The main provers available for Rodin are NewPP, PP and ML. The NewPP prover has three forces. In the configuration restricted (nPP R), all selected hypotheses and the goal are passed to New PP. In the configuration after lasso, nPP with a lasso, a lasso operation is applied to the selected hypotheses and the goal and the result is passed to New PP. The lasso operation selects any unselected hypothesis that have a common symbol with the goal or a hypothesis that is currently selected. In the configuration unrestricted, nPP, all the available hypotheses are passed to New PP. This prover is embedded in the tool and its input language is first-order logic with the predicate \in . First, all function and predicate symbols that are different from \in and not related to arithmetic are translated away. Then New PP translates the proof obligation to CNF (conjunctive normal form) and applies a combination of unit resolution and the Davis Putnam algorithm. The prover PP (predicate prover), available in the Atelier-B as an external prover, also has three forces (P0, P1, PP). In the configuration P0, all selected hypotheses and the goal are passed to PP. In the configuration P1,

one lasso operation is applied to the selected hypotheses and the goal and the result is passed to PP. In the configuration PP, all the available hypotheses are passed to PP. The input sequent is translated to classical B and fed to the PP prover of Atelier B. PP works in a manner similar to newPP but with support for equational and arithmetic reasoning. The prover ML (mono-lemma) is also available in the Atelier-B, but different from others (PP and NewPP). ML applies a mix of forward, backward and rewriting rules in order to discharge the goal (or detect a contradiction among hypotheses). For more details see [2, 11].

Since some proof obligations need user interaction, proof tactics have been developed to help the users to discharge these obligations. A proof tactic, in the context of this work, is defined by a sequence of inference rule applications that must be performed to discharge proof obligations generated by specific properties defined for GG specifications in Rodin. The inference rules can be applied by running Rodin provers or applying rewritten rules, adding hypothesis, instantiating or eliminating quantifiers, and others. The complete list of rules used in this work can be seen in Table 2.

$$\frac{\frac{\frac{\text{True}}{\vdash \top} \quad 2 \quad \frac{\frac{\text{True}}{\vdash \top} \quad 4 \quad A'}{\vdash \top} \quad 3}{A} \quad 1}{A} \quad 1$$

1 \exists goal(inst 3, 2); 2 \top goal; 3 simplification rewrites; 4 \top goal

Figure 4. Proof Tree representation for tactics.

The proof tactics can be described by plain texts or tree-based representations. The use of tree-based representations is more intuitive and favour the understanding the proof structures, since a goal can be partitioned in several sub-goals during the proof process. Moreover, we can use two different tree-based representations to describe each tactic: proof trees and use trees. A proof tree describes each step in detail, showing the sub-goals and the corresponding rules applied to prove these sub-goals. On the other hand, a use tree give us a more abstract description of a proof tactic, just indicating the actions that a user must take to accomplish the proof. An example of proof tree can be seen in Figure 4, where the tree is described by the inference rules and the goal (sequent to be demonstrated) is the tree root. In this example, A is the goal. After to instantiate variables with 3 and 2 in the consequent quantified by an existential (rule \exists goal), two new sequents must be demonstrated: (i) $\vdash \top$ and (ii) A' . The demonstration of (i) is completed by applying the \top rule. While, in order to demonstrate (ii) must be applied the rewrites simplification (3) and \top goal (4) rules.

The corresponding use tree is shown in Figure 5. Each node in the use tree represents a sequent and it is labelled with a comment explaining how it can be discharged. Black labels

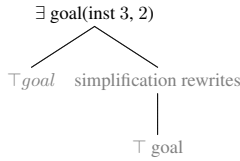


Figure 5. Use Tree representation for tactics.

are those that requires user intervention and gray labels are those discharged automatically by the tool. The use tree must be followed in depth from left to right. In this example, one can see that the proof obligation is discharged just applying the \exists goal (inst 3, 2) rule and the remaining of the nodes are automatically demonstrated.

4 Use Trees for Proof Tactics

In an Event-B model, properties stated as invariants are those that must be true for all reachable states of the specified system. Proofs for such properties are developed by induction: in the base case, a proof obligation is generated to guarantee that the initial graph satisfies the property and, at the inductive step, a proof obligation is generated for the graph resulting from the application of each rule of the grammar. In general, the discharging of such proof obligations requires intervention from the user, that must have knowledge of both the tool and the specification. Proof strategies to assist the developers at the time of discharging semi-automatic proof obligations generated by the specification of some atomic properties in the model were previously proposed [15]. Nonetheless, the description of such tactics was given by proof trees and textual explanations. In this section, we present an alternative representation for them, through use trees [14].

For each property, we first present the use tree for discharging the proof obligation for the initial graph and then for the rules. Labels of kind $ah(H)$ in a node indicates that H must be added as hypothesis, while labels \exists goal(inst I) means that I must be instantiated in an existential goal. Remaining labels are proof tactics and rewriting rules available in the tool.

The `propFin` property stated as $finite(tG_E \triangleright \{t\})$ establishes that the set of edges of type t of a reachable graph is finite. Finiteness property is required whenever a cardinality property must be demonstrated. Figures 6(a) presents the proof tactic for discharging the proof obligation generated by the initialisation event. Particularly, in order to discharge INITIALISATION/propFin/INV, first $tG_E \triangleright \{t\} = \{x\}$ must be added as hypothesis, replacing tG_E by its value and considering x the result of tG_E restricted to the type t for the initial graph. Then, we must execute the prover PP in force P1 and finally run the ML prover.

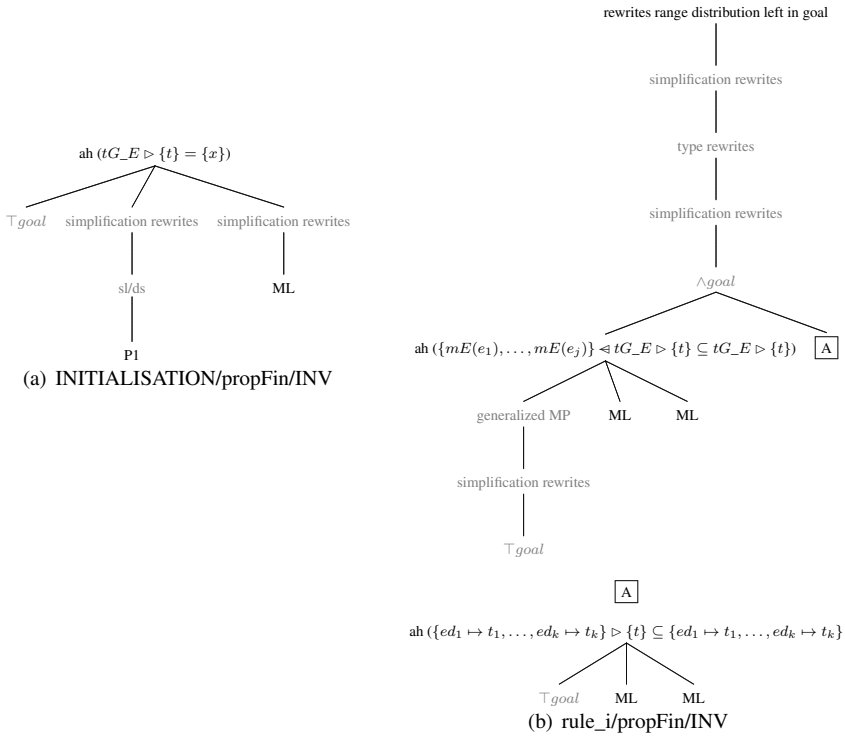


Figure 6. Use Trees for `propFin`

In general, a rule can both delete and create new edges, then the obligation to be discharged for this property will be of the form $finite(\{mE(e_1), \dots, mE(e_j)\} \triangleleft tG_E \cup \{ed_1 \mapsto t_1, \dots, ed_k \mapsto t_k\}) \triangleright \{t\}$, considering that j edges are deleted and a set of k edges are included in tG_E . Figure 6(b) describes the tactic for each rule.

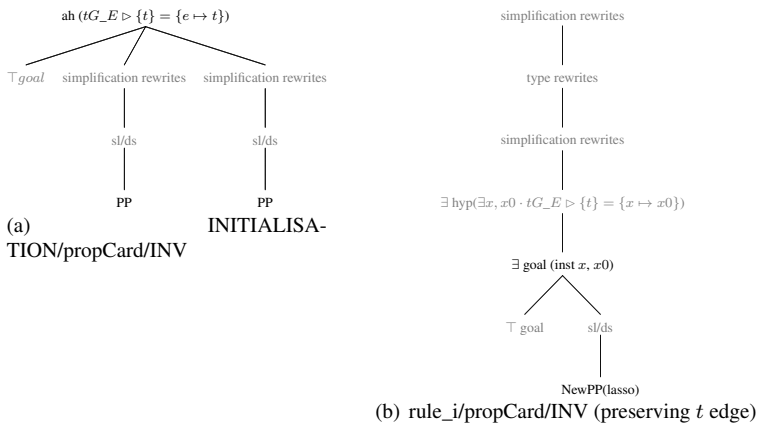


Figure 7. Use Trees for `propCard`

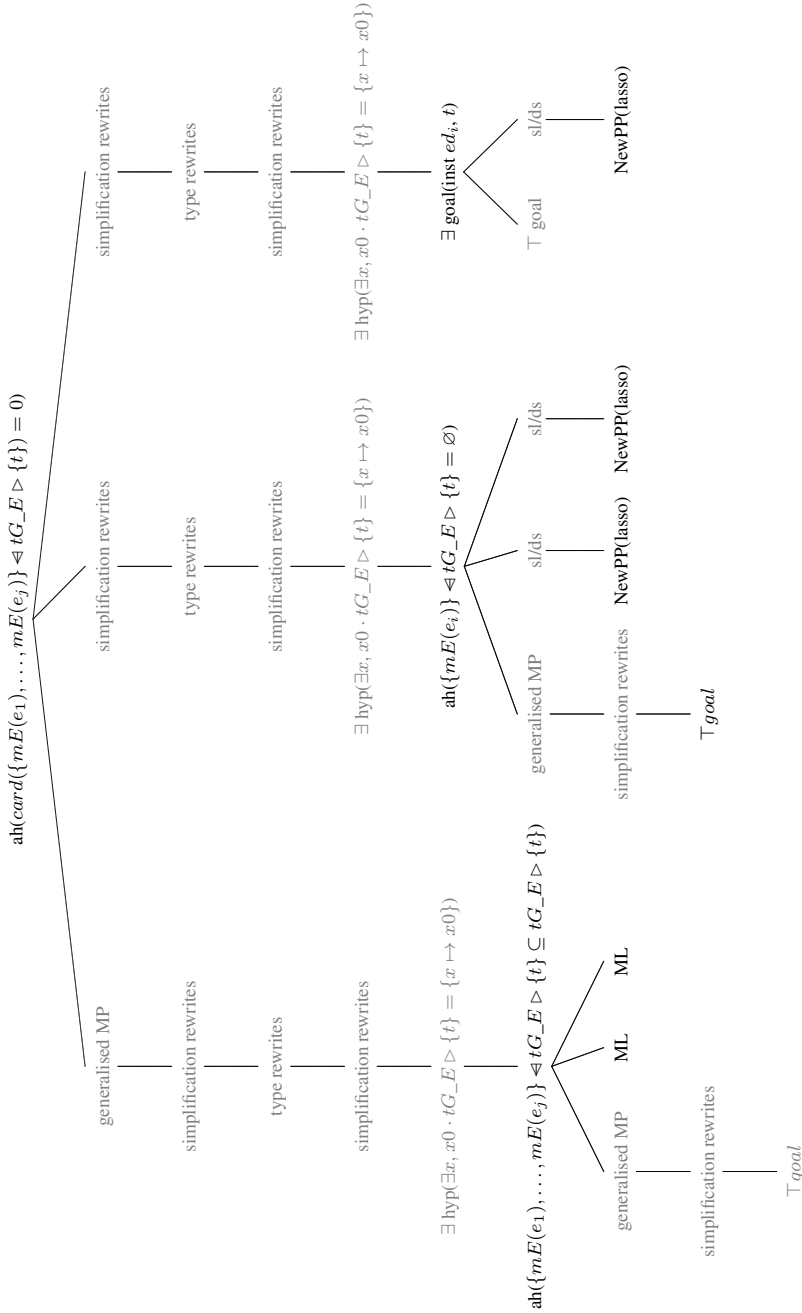


Figure 8. Use tree rule `i/propCard/INV` (deleting and creating t edge) for `propCard`

The `propCard` property, $\text{card}(tG_E \triangleright \{t\}) = 1$, states that any reachable graph has exactly one edge of type t . Figures 7(a), 7(b) and 8 illustrate the proof tactics for the initialisation event and for the rules, when a t edge is preserved and when a t edge is deleted and another t edge is created, respectively. In the initialisation, tG_E must be replaced by its initial value and $e \mapsto t$ is the pair resultant of tG_E restricted to the type t .

The general obligation to be discharged for each rule to `propCard` will be $\text{card}(((\{mE(e_1), \dots, mE(e_j)\} \triangleleft tG_E) \cup \{ed_1 \mapsto t_1, \dots, ed_k \mapsto t_k\}) \triangleright \{t\}) = 1$, considering that j edges are deleted and k edges are created. In fact, if this property is valid, the t edge or is preserved, or is deleted and created again by a rule application. Then, we divide our tactic in two cases. In case that a t edge is preserved, the existential goal must be instantiated with x and $x0$ (with the preserved edge and its type). In the other case, we consider that $mE(e_i)$ is the deleted t edge and ed_i is the created one.

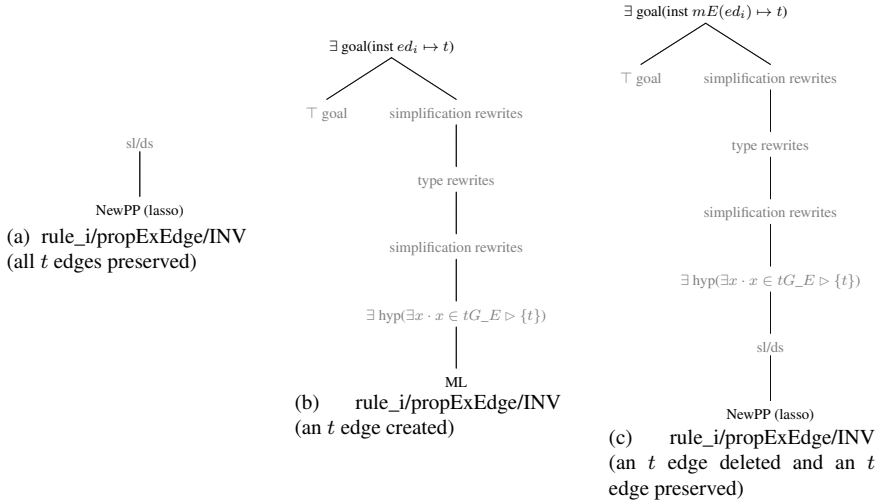


Figure 9. Use Trees for `propExEdge`

The `propExEdge` property, $\exists x \cdot x \in tG_E \triangleright \{t\}$, states that any reachable graph has an edge of type t . To discharge the proof obligation for the initial graph just run NewPP. Tactics for rules are depicted in Figure 9. Again, since a rule can preserve, delete and create edges, then we divide our proof strategies in three cases: (a) all t edges are preserved (and no edge t is created), (b) a t edge is created and (c) a t edge is deleted, no t edge is created and a t edge is preserved. In (b) we consider that $ed_i \mapsto t$ is the created edge and in (c) we assume that $mE(ed_i) \mapsto t$ is the preserved edge. The `propExVertex` property, $\exists x \cdot x \in tG_V \triangleright \{t\}$, states that any reachable graph has a vertex of type t . In order to discharge the

proof obligations for the initialisation and rules we must just run NewPP (lasso).

Properties in Figure 10 were stated for the mobile system in Rodin and all discharged using the proposed tactics. For example, the `propFin` property can be discharged with the assistance of the use trees depicted in Figure 6. In order to discharge the proof obligation generated by the initialisation event (use tree showed in Figure 6(a)), $tG_E \triangleright \{Main\} = \{11\}$ must be added as hypothesis (see Figure 2 to definition of tG_E). Then, we must execute the PP prover in force P1 and finally run the ML prover. One proof obligation is generated to each rule, that can be discharged with the assistance of the use tree depicted in Figure 6(b). Considering the `rule1` event (see Figure 3), the obligation generated for this property will be $finite(\{newEcal \mapsto Cal\} \triangleright \{Main\})$, since it do not delete anything and create an edge named `newEcal`. In order to discharge this obligation the *range distribution to the left rewrites* must be applied in the goal. Then, $\{newEcal \mapsto Cal\} \triangleright \{Main\} \subseteq \{newEcal \mapsto Cal\}$ must be added as hypothesis. Finally, the ML prover must be run twice. Note that, for `rule1` event, only one branch is generated, since there is no deleted element.

INVARIANTS

```

propFin : finite(tG_E ▷ {Main}) // The set of edges of type Main of a reachable graph is finite.
propCard : card(tG_E ▷ {Main}) = 1 // Any reachable graph has exactly one edge of type Main.
propExEdge : ∃x · x ∈ tG_E ▷ {Acn} // Any reachable graph has an edge of type Acn.
propExVert : ∃x · x ∈ tG_V ▷ {Ant} // Any reachable graph has a vertex of type Ant.
    
```

Figure 10. Verified Properties for the Mobile System

5 Extending the Set of Proof Tactics

The focus of our research has been on properties about reachable states for (infinite) state verification. Properties over states are properties over graphs, typically composed of different kinds of edges and vertices. In previous work [9], we have proposed patterns for the presentation, codification and reuse of property specifications and in [15] he have developed proof strategies for the demonstration of specific atomic properties belonging to such patterns. Here we extend the set of tactics describing proof strategies for discharging the properties presented in Figure 11.

The previously proposed patterns are defined from a standard library of functions. Although these functions are not used in the described properties, the tactics proposed here prioritise properties that describe typical characteristics of graphs, predefined in this library of functions. Furthermore, we choose to set tactics for simple properties that could be reused in case of more complex specifications.

The proof obligations generated by a `prop` property stated as invariant are labelled

INVARIANTS

```

propLoopT :  $\exists x \cdot x \in \text{edge}G \wedge \text{source}G(x) = \text{target}G(x) \wedge tG\_E(x) = t$ 
// Any reachable graph has a loop edge of type t.

propEdSpSrcT :  $\exists x, y \cdot (x \in \text{edge}G \wedge y \in \text{vert}G \wedge tG\_V(y) = t \wedge \text{source}G(x) = y)$ 
// Any reachable graph has an edge with source in a vertex of type t.

propEdSpTgtT :  $\exists x, y \cdot (x \in \text{edge}G \wedge y \in \text{vert}G \wedge tG\_V(y) = t \wedge \text{target}G(x) = y)$ 
// Any reachable graph has an edge with target in a vertex of type t.

propNotIsoVT :  $\forall x \cdot ((x \in \text{vert}G \wedge tG\_V(x) = t) \Rightarrow (\exists y \cdot (y \in \text{edge}G \wedge ((\text{source}G(y) = x \wedge \neg \text{target}G(y) = x) \vee (\text{target}G(y) = x \wedge \neg \text{source}G(y) = x))))))$ 
// Any reachable graph has not an isolated vertex of type t.


propAllSVertSrcTEd :  $\forall x \cdot ((x \in \text{vert}G \wedge tG\_V(x) = s) \Rightarrow (\exists y \cdot (y \in \text{edge}G \wedge tG\_E(y) = t \wedge \text{source}G(y) = x)))$ 
// In any reachable graph, all vertex of type s is source of an edge of type t.

propAllSVertTgtTEd :  $\forall x \cdot ((x \in \text{vert}G \wedge tG\_V(x) = s) \Rightarrow (\exists y \cdot (y \in \text{edge}G \wedge tG\_E(y) = t \wedge \text{target}G(y) = x)))$ 
// In any reachable graph, all vertex of type s is target of an edge of type t.
    
```

Figure 11. Properties as Invariants in Event-B

with INITIALISATION/prop/INV for the initialisation event and with rule_i/prop/INV for each rule rule_i. For each property, we first present the steps for discharging the proof obligation for the initial graph and then for the rules. In addition to a textual description of the steps required to complete the proofs, we also detail the proof trees.

In a proof tree, each node represents a sequent and each number (from 1 to 82) represents the rule or the prover used to discharge the corresponding sequent. The symbol H in the antecedents represents a set of hypotheses selected automatically by the tool. Generally, when a tactic is applied, new hypotheses are selected, updating H. Aiming not to overload the notation, we represent the set of automatically selected hypotheses always by H. A set of proof tactics, rewriting rules and provers are available for the Rodin platform [3]. Those used in this work are listed in Table 2. For details see [11, 2].

When opening a proof obligation, by default, the tool applies some auto- and post-tactics, generating some nodes in the proof tree. In order to use the proposed strategies, it is assumed that the user always starts a proof executing a prune in the proof tree (button  at the proof control), undoing any application of rule and starting the proof from the original goal. We also assume that the rules applied in the auto- and post- tactics during the proof process are those configured by default in the tool.

In the following we first discuss the development process of the proof tactics and after we detail the strategies for discharging the proof obligations generated by the properties listed in Figure 11. The strategies for discharging `propEdSpTgtT` and `propAllVertTgtTEd` are respectively very similar to `propEdSpSrcT` and `propAllVertSrcTEd`. We have just to replace source by target and consider the component target in the `grd_vertices` hypothesis instead of source in the proof steps. In order to not extend the text, we omit their

Table 2. Description of Rules

1 - \exists goal (inst x)	2 - \top goal
3 - simplification rewrites	4 - \exists goal (inst ed_t)
5 - \exists hyp ($\exists x \cdot x \in edgeG \wedge sourceG(x) = targetG(x) \wedge tG_E(x) = t$)	6 - type rewrites
7 - eh with $tG_E(x) = t$	8 - ah ($\neg x = mE(e_1)$)
9 - generalised MP	10 - \forall hyp (inst e_1)
11 - sl/ds	12 - Partition rewrites in hyp ($partition(tLi_E\{e_1 \mapsto t_1\}, \dots, \{e_k \mapsto t_k\})$)
13 - eh with $tLi_E = \{e_1 \mapsto t_1, \dots, e_k \mapsto t_k\}$	14 - eh with $t_1 = tG_E(mE(e_1))$
15 - NewPP (with lasso)	16 - ah ($\neg x = mE(e_j)$)
17 - \forall hyp (inst e_j)	18 - eh with $t_k = tG_E(mE(e_j))$
19 - \exists goal (inst $mE(e_p)$)	20 - total function dom substitution in goal
21 - ah ($\neg mE(e_p) = mE(e_1)$)	22 - ah ($\neg mE(e_p) = mE(e_i)$)
23 - ah ($mE(e_p) \in edgeG$)	24 - functional image goal for $mE(e_j)$
25 - hyp	26 - ML
27 - ah ($sourceG(mE(e_p)) = targetG(mE(e_p))$)	28 - \wedge goal
29 - functional goal	30 - ah ($sourceLi(e_p) = targetLi(e_p)$)
31 - Partition rewrites in hyp ($partition(sourceLi, \{e_1 \mapsto v_1\}, \dots, \{e_k \mapsto v_l\})$)	32 - remove \neg in $\neg(e_1 = e_k \wedge v_1 = v_l)$
33 - eh with $sourceLi = \{e_1 \mapsto v_1, \dots, e_k \mapsto v_l\}$	34 - Partition rewrites in hyp $partition(targetLi, \{e_1 \mapsto v_1\}, \dots, \{e_j \mapsto v_t\})$
35 - eh with $targetLi = \{e_1 \mapsto v_1, \dots, e_k \mapsto v_l\}$	36 - remove \neg in $\neg(e_1 = e_k \wedge e_1 = tG_E(x))$
37 - remove \neg in $\neg(e_1 = e_k \wedge e_k = tG_E(x))$	38 - $tLi_E = \{e_1 \mapsto t_1, e_j \mapsto tG_E(x), \dots, e_k \mapsto t_k\}$
39 - \exists goal (inst x, y)	40 - \exists hyp ($\exists x, y \cdot x \in edgeG \wedge y \in vertG \wedge tG_V(y) = t \wedge sourceG(x) = y$)
41 - eh with $tG_V(y) = t$	42 - eh with $sourceG(x) = y$
43 - \exists goal (inst $ed_t, mV(v_t)$)	44 - eh with $tG_V(sourceG(x)) = t$
45 - functional image goal for $mV(v_t)$	46 - \forall hyp (inst v_t)
47 - Partition rewrites in hyp ($partition(vertLi, \{v_1 \mapsto t_1\}, \dots, \{v_l \mapsto t_l\})$)	48 - eh with $tLi_V = \{v_1 \mapsto t_1, \dots, v_t \mapsto tG_V(sourceG(x)), \dots, v_l \mapsto t_l\}$
49 - Functional image simplification in goal	50 - ah $\neg x = m(e_1)$
51 - \forall hyp (inst v_1)	52 - ah $\neg x = m(e_j)$
53 - \forall hyp (inst v_l)	54 - \exists goal (inst $mE(e_p), mV(v_t)$)
55 - Partition rewrites in hyp ($partition(edgeLi, \{e_1\}, \dots, \{e_k\})$)	56 - eh with $edgeLi = \{ed_1, \dots, ed_k\}$
57 - Partition rewrites in hyp ($partition(tLi_V, \{v_1 \mapsto t_1\}, \dots, \{v_k \mapsto t_k\})$)	58 - eh with $tLi_V = \{v_1 \mapsto t_1, \dots, v_j \mapsto tG_V(sourceG(x)), \dots, v_k \mapsto t_k\}$
59 - remove \neg in $\neg(e_j = e_1 \wedge v_t = v_1)$	60 - remove \neg in $\neg(e_j = e_k \wedge v_t = v_l)$
61 - \forall goal (free x)	62 - \Rightarrow goal
63 - remove \in in $x \in \{v_1, \dots, v_n\}$	64 - \forall hyp ($x = v_1 \vee \dots \vee x = v_n$)
65 - eh with $x = v_i$	66 - eh with $x = v_j$
67 - remove \in in $x \in vertG \cup \{v_1, \dots, v_l\}$	68 - \forall hyp ($x \in vertG \vee x = v_1 \vee \dots \vee x = v_l$)
69 - \forall hyp (inst x)	70 - \Rightarrow hyp mp ($tG_V(x) = t \Rightarrow (\exists y \cdot y \in edgeG \wedge ((sourceG(y) = x \wedge \neg targetG(y) = x) \vee (targetG(y) = x \wedge \neg sourceG(y) = x)))$)
71 - \exists hyp ($\exists y \cdot y \in edgeG \wedge ((sourceG(y) = x \wedge \neg targetG(y) = x) \vee (targetG(y) = x \wedge \neg sourceG(y) = x))$)	72 - \exists goal (inst y)
73 - eh with $x = v_i$	74 - eh with $tG_V(x) = t$
75 - eh with $(tG_V \cup \{v_1 \mapsto t_1, \dots, v_l \mapsto t_l\})(x) = t$	76 - remove \in in $x \in \{v_1, \dots, v_n\}$
77 - eh with $x = v_s$	78 - \exists goal (inst e_j)
79 - ah ($tLi_E(e_1) = tG_E(mE(e_1))$)	80 - functional image goal for $mE(e_1)$
81 - ah ($tLi_E(e_j) = tG_E(mE(e_j))$)	82 - \forall hyp (inst e_p)

descriptions. Sometimes, we need to instantiate elements in hypotheses generated from a guard condition (prefix `grd_`). Table 3 presents these identifiers along with their description.

5.1 Development Process of Proof Tactics

A tactic proposition requires knowledge of the specified system and the rewriting rules and proof rules available for the Rodin tool. Similarly, it is crucial to understand the formal proof methods (e.g. direct proof, proof by mathematical induction, proof by contradiction, among others) and have experience in the demonstration of mathematical statements.

Since in our event-B model properties are declared as invariants, proofs are developed by induction. The proof obligation generated for the initialisation event (which defines the initial graph) is demonstrated by instantiating the variables with the elements that guarantee the property. For instance, if a property states that any reachable graph has a loop edge of type t , we must instantiate the goal with a loop edge of type t of the initial graph.

The proof obligations generated in the induction step depend on the changes that the events held in the state. Particularly, these modifications in GG correspond to the rule application effect: graph elements can be created, deleted or preserved. For each event (or rule application), the goal to be demonstrated is generated replacing the variables specified in the model by its new values updated by the event. If the event does not update any variable of a property, no proof obligation is generated for such case.

The proof tactic is developed according to kind of property to be discharged. For instance, if it is a universal quantification, usually, we eliminate the universal quantifier, remove the membership operator, dividing the proof in cases. The cases depend on the effect of the rule application: vertices can be added or preserved, edges can be added, deleted or preserved. If it is an existential quantification, we must instantiate the element that has the property, and often the property is assured by the induction hypothesis.

Table 3. Guard Conditions

Identifier	Description
<code>grd_vertices</code>	$\forall v \cdot v \in vertLi \Rightarrow tLi_V(v) = tG_V(mV(v))$
<code>grd_edges</code>	$\forall e \cdot e \in edgeLi \Rightarrow tLi_E(e) = tG_E(mE(e))$
<code>grd_srctgt</code>	$\forall e \cdot e \in edgeLi \Rightarrow mV(sourceLi(e)) = sourceG(mE(e)) \wedge$ $\forall e \cdot e \in edgeLi \Rightarrow mV(targetLi(e)) = targetG(mE(e))$

5.2 propLoopT property

The `propLoopT` property states that any reachable graph has a loop edge of type t . When the variables $edgeG$, $sourceG$, $targetG$ and tG_E are initialised by the initialisation event, one proof obligation labelled with INITIALISATION/propLoopT/INV is generated. In order to discharge this obligation we must simply instantiate in the goal the name of the loop edge of type t contained in the initial graph. Figure 12 presents the generated proof tree.

$$\frac{\frac{\text{True}}{\vdash \top} \quad 2 \quad \frac{\frac{\text{True}}{\vdash \top} \quad 2}{\vdash x \in edgeG \wedge sourceG(x) = targetG(x) \wedge tG_E(x) = t} \quad 3}{\vdash \exists x \cdot x \in edgeG \wedge sourceG(x) = targetG(x) \wedge tG_E(x) = t} \quad 1$$

Figure 12. Proof Tree INITIALISATION/propLoopT/INV

Proof Tree Description: the goal to be demonstrated is $\vdash \exists x \cdot x \in edgeG \wedge sourceG(x) = targetG(x) \wedge tG_E(x) = t$. In order to discharge this sequent, we must instantiate in the goal the name of the loop edge x of type t (1) belonging to the initial graph. Then, two new sub-goals are generated: (i) $\vdash \top$ that is automatically discharged (by 2) and (ii) $\vdash x \in edgeG \wedge sourceG(x) = targetG(x) \wedge tG_E(x) = t$, which after the automatic execution of simplification rewrites (3), generates the goal $\vdash \top$, also automatically demonstrated by the \top goal rule (2). ■

The corresponding use tree can be seen in Figure 13. By the use tree, one can abstract each detail from the proof tree and realise that only the instantiation of the loop edge name must be performed. From now on, the use trees will be omitted, but they can be constructed from the presented proof trees, by following the numbering of each level of the proof trees.

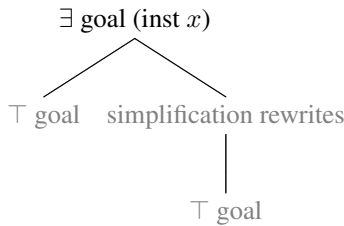


Figure 13. Use tree rule_i/propLoopT/INV

Considering that a rule can delete and create new edges, the proof obligations generated for rules are described as follows $\exists x \cdot x \in (edgeG \setminus \{mE(e_1), \dots, mE(e_j)\}) \cup \{ed_1, \dots, ed_k\} \wedge ((\{mE(e_1), \dots, mE(e_j)\} \triangleleft sourceG) \cup \{ed_1 \mapsto mV(v_1), \dots, ed_k \mapsto$

$v_l\})(x) = ((\{mE(e_1), \dots, mE(e_j)\} \triangleleft targetG) \cup \{ed_1 \mapsto mV(v_1), \dots, ed_k \mapsto v_l\})(x) \wedge$
 $((\{mE(e_1), \dots, mE(e_j)\} \triangleleft tG_E) \cup \{ed_1 \mapsto t_1, \dots, ed_k \mapsto t_k\})(x) = t$, considering that j edges are deleted and k edges are created by the rule. In the case of this property, a rule falls into one of the following cases: it creates loop edges of type t ; it just preserves (it does not delete and neither create) loop edges of type t ; it preserves and deletes (but it does not create) loop edges of type t ; it does not involve loop edges of type t ; it just deletes (it does not preserve and neither create) loop edges of type t . Thus, the presentation of the tactics are divided in these cases, excepting the last one that is not treated in this work. This is because it is not possible to define a generic tactic for this property in such case. Its warranty would depend on other system properties or relations between rules.

A - Rule creates a loop edge of type t

In order to demonstrate this kind of proof obligation we must instantiate, in the goal, the name of the loop edge of type t that is created by the rule. Figure 14 presents the corresponding proof tree.

$$\begin{array}{l}
 \mathcal{A} = edgeG \setminus \{mE(e_1), \dots, mE(e_j)\} \\
 \mathcal{B} = \{ed_1, \dots, ed_k\} \\
 \mathcal{C} = \{mE(e_1), \dots, mE(e_j)\} \triangleleft sourceG \\
 \mathcal{D} = \{ed_1 \mapsto mV(v_1), \dots, ed_k \mapsto v_l\} \\
 \mathcal{E} = \{mE(e_1), \dots, mE(e_j)\} \triangleleft targetG \\
 \mathcal{F} = \{ed_1 \mapsto mV(v_1), \dots, ed_k \mapsto v_l\} \\
 \mathcal{G} = \{mE(e_1), \dots, mE(e_j)\} \triangleleft tG_E \\
 \mathcal{H} = \{ed_1 \mapsto t_1, \dots, ed_k \mapsto t_k\}
 \end{array}$$

$$\frac{\frac{\frac{True}{H \vdash \top} \quad 2 \quad \frac{\frac{True}{H \vdash \top} \quad 2}{H \vdash ed_t \in ((\mathcal{A}) \cup \mathcal{B}) \wedge ((\mathcal{C}) \cup \mathcal{D})(x) = ((\mathcal{E}) \cup \mathcal{F})(x) \wedge ((\mathcal{G}) \cup \mathcal{H})(x) = t} \quad 3}}{H \vdash \exists x \cdot x \in ((\mathcal{A}) \cup \mathcal{B}) \wedge ((\mathcal{C}) \cup \mathcal{D})(x) = ((\mathcal{E}) \cup \mathcal{F})(x) \wedge ((\mathcal{G}) \cup \mathcal{H})(x) = t} \quad 4}$$

Figure 14. Proof Tree rule_i/propLoopT/INV - Rule Creates a Loop Edge of Type t

Proof Tree Description: The sequent to be proved is $H \vdash \exists x \cdot x \in ((edgeG \setminus \{mE(e_1), \dots, mE(e_j)\}) \cup \{ed_1, \dots, ed_k\}) \wedge ((\{mE(e_1), \dots, mE(e_j)\} \triangleleft sourceG) \cup \{ed_1 \mapsto mV(v_1), \dots, ed_k \mapsto v_l\})(x) = ((\{mE(e_1), \dots, mE(e_j)\} \triangleleft targetG) \cup \{ed_1 \mapsto mV(v_1), \dots, ed_k \mapsto v_l\})(x) \wedge ((\{mE(e_1), \dots, mE(e_j)\} \triangleleft tG_E) \cup \{ed_1 \mapsto t_1, \dots, ed_k \mapsto t_k\})(x) = t$. In order to discharge this sequent, we must instantiate, in the goal, the name of the loop edge ed_t , of type t , created by the rule application (4). Then, two new sub-goals are generated: (i) $H \vdash \top$ and (ii) $H \vdash ed_t \in ((edgeG \setminus \{mE(e_1), \dots, mE(e_j)\}) \cup \{ed_1, \dots, ed_k\}) \wedge ((\{mE(e_1), \dots, mE(e_j)\} \triangleleft sourceG) \cup \{ed_1 \mapsto mV(v_1), \dots, ed_k \mapsto v_l\})(x) = ((\{mE(e_1), \dots, mE(e_j)\} \triangleleft targetG) \cup \{ed_1 \mapsto mV(v_1), \dots, ed_k \mapsto v_l\})(x) \wedge$

$((\{mE(e_1), \dots, mE(e_j)\} \triangleleft tG_E) \cup \{ed_1 \mapsto t_1, \dots, ed_k \mapsto t_k\})(x) = t$. (i) is automatically discharged by the \top goal rule (2). In (ii), the simplification rewrites rule (3) is automatically applied, resulting the sequent $H \vdash \top$, which is also automatically demonstrated (by 2). ■

B - Rule just preserves or does not involve loop edges of type t

For cases that no other edge (different from type t) is deleted, we must just run the NewPP with lasso prover. For cases in which other edges, different from type t , are deleted, we must carry out the following steps to conclude the proof:

1. Apply the \exists hyp rule in the induction hypothesis;
2. For each edge e_i , which is deleted by the rule, the next sequence of steps must be followed:
 - (a) Add $\neg x = mE(e_i)$ as hypothesis, where e_i is an edge deleted by the rule, but does not have type t ;
 - (b) Instantiate the deleted edge e_i in the `grd_edges` hypothesis;
 - (c) Apply the partition rewrites rule in the hypothesis that defines the typing of the edges in the left-hand side of the rule and apply the NewPP with lasso prover, in the generated sub-goals.
3. Run the NewPP with lasso prover.

Figure 15 presents the proof tree generated for this case.

Proof Tree Description: The goal to be demonstrated is $H \vdash \exists x \cdot x \in (edgeG \setminus \{mE(e_1), \dots, mE(e_j)\}) \cup \{ed_1, \dots, ed_k\} \wedge ((\{mE(e_1), \dots, mE(e_j)\} \triangleleft sourceG) \cup \{ed_1 \mapsto mV(v_1), \dots, ed_k \mapsto v_l\})(x) = ((\{mE(e_1), \dots, mE(e_j)\} \triangleleft targetG) \cup \{ed_1 \mapsto mV(v_1), \dots, ed_k \mapsto v_l\})(x) \wedge ((\{mE(e_1), \dots, mE(e_j)\} \triangleleft tG_E) \cup \{ed_1 \mapsto t_1, \dots, ed_k \mapsto t_k\})(x) = t$. In order to discharge this sequent we must apply the \exists hyp rule in the induction hypothesis (5). After this, some rules are automatically applied (rules 3, 6, 3 and 7), remaining the following sub-goal $H \vdash \exists x0 \cdot x0 \in (edgeG \setminus \{mE(e_1), \dots, mE(e_j)\}) \cup \{ed_1, \dots, ed_k\} \wedge ((\{mE(e_1), \dots, mE(e_j)\} \triangleleft sourceG) \cup \{ed_1 \mapsto mV(v_1), \dots, ed_k \mapsto v_l\})(x0) = ((\{mE(e_1), \dots, mE(e_j)\} \triangleleft targetG) \cup \{ed_1 \mapsto mV(v_1), \dots, ed_k \mapsto v_l\})(x0) \wedge ((\{mE(e_1), \dots, mE(e_j)\} \triangleleft tG_E) \cup \{ed_1 \mapsto t_1, \dots, ed_k \mapsto t_k\})(x0) = tG_E(x)$. In order to discharge such sub-goal, we must add $\neg x = mE(e_i)$ as hypothesis (8), for each edge e_i deleted by the rule application. We assume that $\{e_1, \dots, e_j\}$ represents the set of

$$\begin{array}{c}
 \mathcal{A} = \{mE(e_1), \dots, mE(e_j)\} \quad \mathcal{B} = \{ed_1, \dots, ed_k\} \quad \mathcal{C} = \{ed_1 \mapsto mV(v_1), \dots, ed_k \mapsto v_j\} \\
 \mathcal{D} = \{ed_1 \mapsto t_1, \dots, ed_k \mapsto t_k\} \quad \mathcal{E} = \text{dom}(mE) \wedge mE \in \text{edge}Li \mapsto Z
 \end{array}$$

$$\begin{array}{c}
 \boxed{\text{B}} \\
 \frac{H \vdash \exists x_0 \cdot x_0 \in (\text{edge}G \setminus \mathcal{A}) \cup \mathcal{B} \wedge ((\mathcal{A} \Leftarrow \text{source}G) \cup \mathcal{C})(x_0) \wedge ((\mathcal{A} \Leftarrow tG_E) \cup \mathcal{D})(x_0) = tG_E(x)}{\text{NewPPRULES}} \quad 11,15
 \end{array}$$

$$\begin{array}{c}
 \boxed{\text{A}} \\
 \frac{\frac{\frac{\text{True}}{H \vdash \top} \quad 2}{H \vdash \top \wedge \top} \quad 3}{H \vdash e_j \in \mathcal{E}} \quad 9 \quad \frac{\text{True}}{H \vdash \top} \quad 2}{H \vdash \neg x = mE(e_j)} \quad 17}{\frac{\text{NewPPRULES} \quad 3,9,3,13,3,18,11,15}{H \vdash \neg x = mE(e_j)} \quad 12}{H, tL_i_E(e_j) = tG_E(mE(e_j)) \vdash \neg x = mE(e_j)} \quad 11}{H \vdash \neg x = mE(e_j)} \quad 16}
 \end{array}$$

$$\begin{array}{c}
 \boxed{\text{A}} \\
 \frac{\frac{\frac{\text{True}}{H \vdash \top} \quad 2}{H \vdash \top \wedge \top} \quad 3}{H \vdash e_1 \in \mathcal{E}} \quad 9 \quad \frac{\text{True}}{H \vdash \top} \quad 2}{H \vdash \neg x = mE(e_1)} \quad 10}{\frac{\text{NewPPRULES} \quad 3,9,3,13,3,14,11,15}{H \vdash \neg x = mE(e_1)} \quad 12}{H, tL_i_E(e_1) = tG_E(mE(e_1)) \vdash \neg x = mE(e_1)} \quad 11}{H \vdash \neg x = mE(e_1)} \quad 8}
 \end{array}$$

$$\frac{H \vdash \exists x_0 \cdot x_0 \in (\text{edge}G \setminus \mathcal{A}) \cup \mathcal{B} \wedge ((\mathcal{A} \Leftarrow \text{source}G) \cup \mathcal{C})(x_0) = ((\mathcal{A} \Leftarrow tG_E) \cup \mathcal{D})(x_0) = tG_E(x)}{H \vdash \exists x \cdot x \in (\text{edge}G \setminus \mathcal{A}) \cup \mathcal{B} \wedge ((\mathcal{A} \Leftarrow \text{source}G) \cup \mathcal{C})(x) = ((\mathcal{A} \Leftarrow tG_E) \cup \mathcal{D})(x) = t} \quad 8, 5,3,6,3,7$$

Figure 15. Proof Tree rule_i/propLoopT/INV - Rule just preserves or does not involve loop edges of type t

deleted edges. In the proof tree is illustrated the sub-trees for e_1 and e_j , omitting the intermediary ones. The addition of such hypothesis produces three new sub-goals: (I) $H \vdash e_i \in \text{dom}(mE) \wedge mE \in \text{edgeLi} \mapsto \mathbb{Z}$; (II) $H \vdash \neg x = mE(e_i)$ and (III) $H \vdash \exists x0. x0 \in (\text{edgeG} \setminus \{mE(e_1), \dots, mE(e_j)\}) \cup \{ed_1, \dots, ed_k\} \wedge ((\{mE(e_1), \dots, mE(e_j)\} \triangleleft \text{sourceG}) \cup \{ed_1 \mapsto mV(v_1), \dots, ed_k \mapsto v_l\})(x0) = ((\{mE(e_1), \dots, mE(e_j)\} \triangleleft \text{targetG}) \cup \{ed_1 \mapsto mV(v_1), \dots, ed_k \mapsto v_l\})(x0) \wedge ((\{mE(e_1), \dots, mE(e_j)\} \triangleleft tG_E) \cup \{ed_1 \mapsto t_1, \dots, ed_k \mapsto t_k\})(x0) = tG_E(x)$. (I) is automatically discharged by generalised MP rule (9), the simplification rewrites rule (3) and the \top goal rule (2). In order to demonstrate (II) we must instantiate the name e_i in the `grd_edges` hypothesis (17). As a result, two new sub-goals must be demonstrated: (i) $H \vdash \top$, automatically discharged (by 2) and (ii) $H \vdash \neg x = mE(e_i)$. In order to conclude (ii), we must apply the partition rewrites rule (12) in the hypothesis that defines the typing of edges in the left-hand side of the rule. After, some automatic rules are applied (rules 3, 9, 3, 13, 3, 18, 11), and then the sequent is demonstrated applying the NewPP with lasso prover (15). This process must be repeated for each deleted edge (from e_1 to e_j). Finally, after some automatic selection/ deselection of hypothesis (11), the sequent (III) is discharged by the NewPP with lasso prover (15). ■

C - Rule preserves and deletes (but does not create) loop edges of type t

The steps to discharge the proof obligation in this case are the following:

1. Instantiate in the goal $mE(e_p)$, such that e_p is the name of the preserved t edge;
2. For each deleted edge e_i :
 - (a) Add $\neg mE(e_p) = mE(e_i)$ as hypothesis, such that e_p is the name of the preserved loop edge of type t and e_i is a deleted edge by the rule. After, run the NewPP with lasso prover.
3. Add $mE(e_p) \in \text{edgeG}$ as hypothesis and run ML;
4. Add $\text{sourceG}(mE(e_p)) = \text{targetG}(mE(e_p))$ as hypothesis;
5. Add $\text{sourceLi}(e_p) = \text{targetLi}(e_p)$ as hypothesis;
6. Apply the partition rewrites rule in the hypothesis that defines the source of the edges in the left-hand side of the rule;
7. Apply partition rewrites rule in the hypothesis that defines the target of the edges in the left-hand side of the rule;
8. Instantiate the name of the preserved loop edge e_p of type t , in the `grd_srctgt` hypothesis considering sourceG ;

9. Instantiate the name of the preserved loop edge e_p of type t , in the `grd_srctgt` hypothesis considering `targetG`;
10. Run NewPP with lasso;
11. Run ML;
12. Instantiate the name of the preserved loop edge e_p of type t , in the `grd_edges` hypothesis;
13. Apply partition rewrites rule in the hypothesis that defines the typing of the edges in the left-hand side of the rule;
14. Run ML.

Due to space limitations the generated proof tree was divided and presented in Figures 16, 17, and 18.

Proof Tree Description: Figure 16 corresponds to steps 1 to 3 of the proposed tactic. The sequent to be demonstrated is $H \vdash \exists x \cdot x \in (\text{edge}G \setminus \{mE(e_1), \dots, mE(e_j)\}) \cup \{ed_1, \dots, ed_k\} \wedge ((\{mE(e_1), \dots, mE(e_j)\} \triangleleft \text{source}G) \cup \{ed_1 \mapsto mV(v_1), \dots, ed_k \mapsto mV(v_l)\})(x) = ((\{mE(e_1), \dots, mE(e_j)\} \triangleleft \text{target}G) \cup \{ed_1 \mapsto mV(v_1), \dots, ed_k \mapsto mV(v_l)\})(x) \wedge ((\{mE(e_1), \dots, mE(e_j)\} \triangleleft tG_E) \cup \{ed_1 \mapsto t_1, \dots, ed_k \mapsto t_k\})(x) = t$. With the aim of discharging it, we must first instantiate $mE(e_p)$ in the goal, such that e_p represents the name of the edge of type t preserved by the rule. Then two new sub-goals are generated: (I) $H \vdash e_p \in \text{dom}(mE) \wedge mE \in \text{edge}Li \mapsto \mathbb{Z}$; and (II) $H \vdash mE(e_p) \in (\text{edge}G \setminus \{mE(e_1), \dots, mE(e_j)\}) \cup \{ed_1, \dots, ed_k\} \wedge ((\{mE(e_1), \dots, mE(e_j)\} \triangleleft \text{source}G) \cup \{ed_1 \mapsto mV(v_1), \dots, ed_k \mapsto mV(v_l)\})(mE(e_p)) = ((\{mE(e_1), \dots, mE(e_j)\} \triangleleft \text{target}G) \cup \{ed_1 \mapsto mV(v_1), \dots, ed_k \mapsto mV(v_l)\})(mE(e_p)) \wedge ((\{mE(e_1), \dots, mE(e_j)\} \triangleleft tG_E) \cup \{ed_1 \mapsto t_1, \dots, ed_k \mapsto t_k\})(mE(e_p)) = t$. In (I), after the instantiation, some rules (3, 9, 3, 20, 6, respectively) are automatically applied, reaching the $H \vdash \top$ goal, which is automatically discharged by (2). In the same way, in (II), some rules are automatically applied (9, 3, 6, 3, 5, 7 and 2, respectively), generating three new sequents to be discharged: (I) $H \vdash mE(e_p) \in (\text{edge}G \setminus \{mE(e_1), \dots, mE(e_j)\}) \cup \{ed_1, \dots, ed_k\}$; (II) $H \vdash ((\{mE(e_1), \dots, mE(e_j)\} \triangleleft \text{source}G) \cup \{ed_1 \mapsto mV(v_1), \dots, ed_k \mapsto mV(v_l)\})(mE(e_p)) = ((\{mE(e_1), \dots, mE(e_j)\} \triangleleft \text{target}G) \cup \{ed_1 \mapsto mV(v_1), \dots, ed_k \mapsto mV(v_l)\})(mE(e_p))$; and (III) $H \vdash ((\{mE(e_1), \dots, mE(e_j)\} \triangleleft tG_E) \cup \{ed_1 \mapsto mV(v_1), \dots, ed_k \mapsto mV(v_l)\})(mE(e_p)) = tG_E(x)$. In order to proof (I) (sub-tree \boxed{A}), we must add $\neg mE(e_p) = mE(e_i)$ as hypothesis, for each deleted edge e_i , with $i \in \{1, \dots, j\}$. After, three new sub-goals must be demonstrated: (i) $H \vdash e_p \in \text{dom}(mE) \wedge mE \in \text{edge}Li \mapsto \mathbb{Z} \wedge e_p \in \text{dom}(mE)$; (ii) $H \vdash \neg mE(e_p) = mE(e_i)$ and (iii) $H \vdash mE(e_p) \in (\text{edge}G \setminus \{mE(e_1), \dots, mE(e_j)\}) \cup \{ed_1, \dots, ed_k\}$.

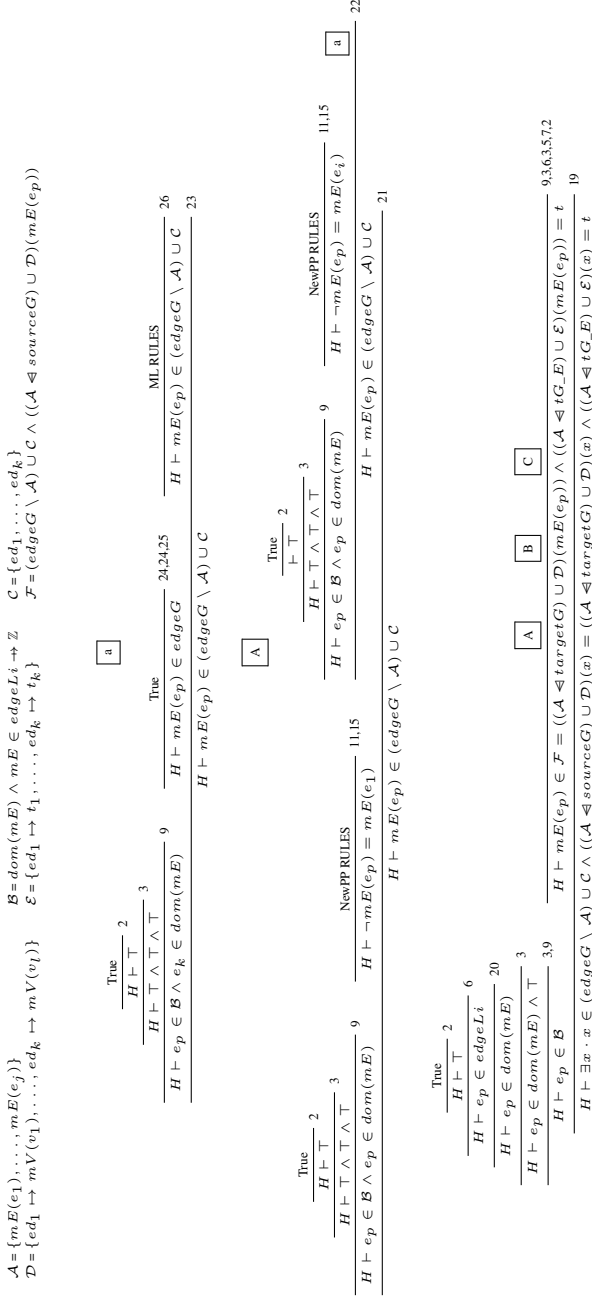


Figure 16. Proof Tree rule_i/propLoopT/INV - Rule preserves and deletes (but does not create) loop edges of type t (1/3)

$$\begin{array}{c}
 \begin{array}{l}
 \mathcal{A} = \{mE(e_1), \dots, mE(e_j)\} \\
 \mathcal{D} = \{ed_1 \mapsto mV(v_1), \dots, ed_k \mapsto v_l\} \\
 \mathcal{G} = \text{dom}(\text{target}G) \wedge \text{target}G \in \mathbb{Z} \rightarrow \mathbb{Z} \\
 \mathcal{J} = mE(ep) \in \text{dom}(\text{source}G) \\
 \mathcal{M} = \text{target}Li \in \text{edge}Li \mapsto \text{vert}Li
 \end{array}
 &
 \begin{array}{l}
 \mathcal{B} = \text{dom}(mE) \wedge mE \in \text{edge}Li \mapsto \mathbb{Z} \\
 \mathcal{E} = \{ed_1 \mapsto t_1, \dots, ed_k \mapsto t_k\} \\
 \mathcal{H} = \text{dom}(\text{source}Li) \wedge \text{source}Li \in \text{edge}Li \mapsto \text{vert}Li \\
 \mathcal{K} = mE(ep) \in \text{dom}(\text{target}G)
 \end{array}
 &
 \begin{array}{l}
 \mathcal{C} = \{ed_1, \dots, ed_k\} \\
 \mathcal{F} = \text{dom}(\text{source}G) \wedge \text{source}G \in \mathbb{Z} \rightarrow \mathbb{Z} \\
 \mathcal{I} = \text{dom}(\text{target}Li) \wedge \text{target}Li \in \text{edge}Li \mapsto \text{vert}Li \\
 \mathcal{L} = \text{source}Li \in \text{edge}Li \mapsto \text{vert}Li
 \end{array}
 \\
 \\
 \begin{array}{c}
 \boxed{\text{b}} \\
 \frac{\frac{\frac{\text{True}}{H \vdash \mathbb{T}} \quad 2}{H \vdash ep \in \text{edge}Li} \quad 6}{H \vdash ep \in \text{dom}(\text{source}Li)} \quad 20 \quad \frac{\frac{\text{True}}{H \vdash \mathbb{T}} \quad 2}{H \vdash ep \in \text{edge}Li} \quad 6}{H \vdash ep \in \text{dom}(\text{target}Li)} \quad 20 \quad \frac{\text{True}}{H \vdash \mathcal{M}} \quad 29}{H \vdash ep \in \mathcal{H}} \quad 28 \quad \boxed{\text{d}} \quad \boxed{\text{e}} \quad 9,3,30 \\
 \frac{}{H \vdash \text{source}G(mE(ep)) = \text{target}G(mE(ep))}
 \end{array}
 &
 \begin{array}{c}
 \boxed{\text{B}} \\
 \frac{\frac{\frac{\text{True}}{H \vdash \mathbb{T}} \quad 25}{H \vdash \mathcal{J}} \quad 24,24,20}{H \vdash \text{source}G \in \mathbb{Z} \rightarrow \mathbb{Z}} \quad 29 \quad \frac{\frac{\text{True}}{H \vdash \mathbb{T}} \quad 25}{H \vdash \mathcal{K}} \quad 24,24,20}{H \vdash \text{target}G \in \mathbb{Z} \rightarrow \mathbb{Z}} \quad 29 \quad 28}{H \vdash mE(ep) \in \mathcal{F} \wedge mE(ep) \in \mathcal{G}} \quad 3}{H \vdash \mathbb{T} \wedge \mathbb{T} \wedge mE(ep) \in \mathcal{F} \wedge mE(ep) \in \mathcal{G}} \quad 9}{H \vdash ep \in \mathcal{B} \wedge mE(ep) \in \mathcal{F} \wedge mE(ep) \in \mathcal{G}} \quad 9}{H \vdash ((\mathcal{A} \Leftarrow \text{source}G) \cup \mathcal{D})(mE(ep)) = ((\mathcal{A} \Leftarrow \text{target}G) \cup \mathcal{D})(mE(ep))} \quad 27 \\
 \boxed{\text{c}}
 \end{array}
 \end{array}$$

Figure 17. Proof Tree rule_i/propLoopT/INV - Rule preserves and deletes (but does not create) loop edges of type t (2/3)

In (i), some rules (9 and 3) are automatically applied, remaining the sequent $H \vdash \top$ that is automatically discharged by (2). On the other hand, after some selection and deselection of hypothesis, the sequent (ii) is proved running NewPP with lasso (15). This last sequence of steps must be repeated for each deleted edge. In order to demonstrate (iii) (sub-tree a), we must add $mE(e_p) \in \text{edge}G$ as hypothesis, reaching three sub-goals: (i) $H \vdash e_p \in \text{dom}(mE) \wedge mE \in \text{edge}Li \mapsto \mathbb{Z} \wedge e_k \in \text{dom}(mE)$; (ii) $H \vdash mE(e_p) \in \text{edge}G$; and (iii) $H \vdash mE(e_p) \in (\text{edge}G \setminus \{mE(e_1), \dots, mE(e_j)\}) \cup \{ed_1, \dots, ed_k\}$. In (i), after some automatic simplifications, the $H \vdash \top$ sequent is automatically discharged by (2). (ii) is automatically proved and (iii) is demonstrated running ML. Figure 17 corresponds to steps 4 and 5 of the proposed tactic. In order to discharge (II) (sub-tree B), we must add $\text{source}G(mE(e_p)) = \text{target}G(mE(e_p))$ (27) as hypothesis, which generates three new sub-goals: (I) $H \vdash e_p \in \text{dom}(mE) \wedge mE \in \text{edge}Li \mapsto \mathbb{Z} \wedge mE(ed_t) \in \text{dom}(\text{source}G) \wedge \text{source}G \in \mathbb{Z} \mapsto \mathbb{Z} \wedge mE(ed_t) \in \text{dom}(\text{target}G) \wedge \text{target}G \in \mathbb{Z} \mapsto \mathbb{Z}$; (II) $H \vdash \text{source}G(mE(e_p)) = \text{target}G(mE(e_p))$; (III) $H \vdash ((\{mE(e_1), \dots, mE(e_j)\} \triangleleft \text{source}G) \cup \{ed_1 \mapsto mV(v_1), \dots, ed_k \mapsto mV(v_l)\})(mE(e_p)) = ((\{mE(e_1), \dots, mE(e_j)\} \triangleleft \text{target}G) \cup \{ed_1 \mapsto mV(v_1), \dots, ed_k \mapsto mV(v_l)\})(mE(e_p))$. (I) is automatically discharged after the application of several rules. In (II) (sub-tree b), rules (9) and (3) are automatically applied. In order to complete the proof, we must add $\text{source}Li(e_p) = \text{target}Li(e_p)$ (30) as hypothesis, which generates three new sub-goals: (i) $H \vdash e_p \in \text{dom}(\text{source}Li) \wedge \text{source}Li \in \text{edge}Li \mapsto \text{vert}Li$; (ii) $H \vdash \text{source}Li(e_p) = \text{target}Li(e_p)$; and (iii) $H \vdash \text{source}G(mE(e_p)) = \text{target}G(mE(e_p))$. After the automatic application of several rules, the sub-goal (i) is discharged. Figure 18 corresponds to the remaining steps (6 - 14) of the proposed tactic. Sequent (ii) (sub-tree d) is discharged applying the partition rewrites rule on the hypothesis that defines the source of the edges in the left-hand side of the rule (31). Then, after the automatic application of some rules (3, 32, 33 and 3), we must apply the partition rewrites rule on the hypothesis that defines the target of the edges in the left-hand side of the rule (34). Then, automatic applied rules discharge the remaining sub-goals. In order to discharge (iii) (sub-tree e), we must instantiate the name of the preserved loop edge e_p of type t (82) in the `grd_srctgt` hypothesis, considering $\text{source}G$. Next, two new sub-goals must be proved: (i) $H \vdash \top$, automatically discharged (2); and (ii) $H \vdash \text{source}G(mE(e_p)) = \text{target}G(mE(e_p))$, which is demonstrated instantiating the name of the preserved edge e_p of type t in `grd_srctgt` hypothesis, considering $\text{target}G$ (82). Therewith, two new sequents must be demonstrated: $H \vdash \top$, automatically discharged (2) and $H \vdash \text{source}G(mE(e_p)) = \text{target}G(mE(e_p))$, demonstrated running NewPP with lasso (15). In (III) (sub-tree c), $H \vdash ((\{mE(e_1), \dots, mE(e_j)\} \triangleleft \text{source}G) \cup \{ed_1 \mapsto mV(v_1), \dots, ed_k \mapsto mV(v_l)\})(mE(e_p)) = ((\{mE(e_1), \dots, mE(e_j)\} \triangleleft \text{target}G) \cup \{ed_1 \mapsto mV(v_1), \dots, ed_k \mapsto mV(v_l)\})(mE(e_p))$, some rules are automatically applied (9 and 3) and the remaining sequent is proved running ML (26). With the aim of proving (III) (sub-tree C), we must instantiate the name of the loop edge e_p of type t in `grd_edges`

$$\begin{array}{c}
 \mathcal{A} = \{mE(e_1), \dots, mE(e_j)\} \\
 \mathcal{D} = \{ed_1 \mapsto mV(v_1), \dots, ed_k \mapsto v_l\} \\
 \mathcal{G} = \text{dom}(\text{targetG}) \wedge \text{targetG} \in \mathbb{Z} \rightarrow \mathbb{Z} \\
 \\
 \mathcal{B} = \text{dom}(mE) \wedge mE \in \text{edgeLi} \rightarrow \mathbb{Z} \\
 \mathcal{E} = \{ed_1 \mapsto t_1, \dots, ed_k \mapsto t_k\} \\
 \mathcal{H} = \text{dom}(\text{sourceLi}) \wedge \text{sourceLi} \in \text{edgeLi} \rightarrow \text{vertLi} \\
 \\
 \mathcal{C} = \{ed_1, \dots, ed_k\} \\
 \mathcal{F} = \text{dom}(\text{sourceG}) \wedge \text{sourceG} \in \mathbb{Z} \rightarrow \mathbb{Z} \\
 \mathcal{I} = \text{dom}(\text{targetLi}) \wedge \text{targetLi} \in \text{edgeLi} \rightarrow \text{vertLi} \\
 \\
 \boxed{c} \\
 \\
 \frac{\text{True}}{H \vdash \top} \quad 2 \quad \frac{\text{True}}{H \vdash \top} \quad 2 \quad \frac{\text{NewPP RULES}}{H \vdash \text{sourceG}(mE(ep)) = \text{targetG}(mE(ep))} \quad 11,15 \\
 \frac{\text{True}}{H \vdash \top} \quad 2 \quad \frac{\text{True}}{H \vdash \text{sourceG}(mE(ep)) = \text{targetG}(mE(ep))} \quad 11,82 \quad \frac{\text{True}}{H \vdash \text{sourceG}(mE(ep)) = \text{targetG}(mE(ep))} \quad 82 \\
 \\
 \boxed{d} \\
 \\
 \frac{\text{True}}{H \vdash \top} \quad 2 \quad \frac{\text{True}}{H \vdash v_t = \text{targetLi}(ep)} \quad 3 \quad \frac{\text{True}}{H \vdash v_t = \text{targetLi}(ep)} \quad 3 \\
 \frac{\text{True}}{H \vdash \{e_1 \mapsto v_1, \dots, e_k \mapsto v_l\}(ep) = \text{targetLi}(ep)} \quad 34,3,9,3,32,35 \\
 \frac{\text{True}}{H \vdash \{e_1 \mapsto v_1, \dots, e_k \mapsto v_l\}(ep) = \text{targetLi}(ep)} \quad 3 \quad \frac{\text{True}}{H \vdash \text{sourceLi}(ep) = \text{targetLi}(ep)} \quad 11,31,3,32,33 \\
 \\
 \boxed{C} \\
 \\
 \frac{\text{True}}{H \vdash \top} \quad 2 \quad \frac{\text{ML RULES}}{H \vdash ((\mathcal{A} \Leftarrow tG_E) \cup \mathcal{D})(mE(ep)) = tG_E(x)} \quad 11,7,12,3,9,3,36,37,38,3,26 \\
 \frac{\text{True}}{H \vdash ((\mathcal{A} \Leftarrow tG_E) \cup \mathcal{D})(mE(ep)) = tG_E(x)} \quad 82 \\
 \\
 \boxed{c} \\
 \frac{\text{True}}{H \vdash ((\mathcal{A} \Leftarrow \text{sourceG}) \cup \mathcal{D})(mE(ep)) = ((\mathcal{A} \Leftarrow \text{targetG}) \cup \mathcal{D})(mE(ep))} \quad 9,3,26
 \end{array}$$

Figure 18. Proof Tree rule_i/propLoopI/INV - Rule preserves and deletes (but does not create) loop edges of type t (3/3)

hypothesis. After, two new sub-goals must be demonstrated: (i) $H \vdash \top$, which is automatically discharged by (2); and (ii) $H \vdash ((\{mE(e_i), \dots, mE(e_j)\} \triangleleft tG_E) \cup \{ed_1 \mapsto mV(v_1), \dots, ed_k \mapsto mV(v_l)\})(mE(e_p)) = tG_E(x)$, which is discharged applying partition rewrites rule in the hypothesis that define the typing of the edges in the left-hand side of the rule (12). Next, some rules are automatically applied (3, 9, 3, 36, 37, 38 and 3, respectively) and the remaining sequent is proved running ML (26). ■

5.3 propEdSpSrcT property

The `propEdSpSrcT` property, $\exists x, y \cdot (x \in edgeG \wedge y \in vertG \wedge tG_V(y) = t \wedge sourceG(x) = y)$, states that any reachable graph has an edge with source in a vertex of type t . Any rule that updates the value of one of the variables `edgeG`, `vertG`, `tG_V` or `sourceG` generates a proof obligation. In the initialisation event all variables are initialised, so is generated a obligation labelled as INITIALISATION/propEdSpSrcT/INV. In order to demonstrate this obligation, we must instantiate x and y in the goal, such that x is an edge that has source in a vertex y of type t ($x \in y \in \mathbb{N}$). Figure 19 presents the proof tree generated by the demonstration.

$$\frac{\frac{\frac{\text{True}}{\vdash \top} \quad 2}{\vdash x \in edgeG \wedge y \in vertG \wedge tG_V(y) = t \wedge sourceG(x) = y} \quad 3}{\vdash \exists x, y \cdot x \in edgeG \wedge y \in vertG \wedge tG_V(y) = t \wedge sourceG(x) = y} \quad 39$$

Figure 19. Proof Tree INITIALISATION/propEdSpSrcT/INV

Proof Tree Description: The sequent to be stated is $\vdash \exists x, y \cdot x \in edgeG \wedge y \in vertG \wedge tG_V(y) = t \wedge sourceG(x) = y$. In order to discharge such sequent we must instantiate in the goal the name of the edge x and of the vertex y , such that x has y as source vertex and y is of type t (39). Next, two new sub-goals are generated: (i) $\vdash \top$, automatically proved by \top goal (2); and (ii) $\vdash x \in edgeG \wedge y \in vertG \wedge tG_V(y) = t \wedge sourceG(x) = y$, which is automatically discharged (by 3 and 2). ■

Assuming that a rule can delete or create edges and that it can create vertices, the proof obligations generated by `propEdSpSrcT` are stated as follows $\exists x, y \cdot x \in (edgeG \setminus \{mE(e_1), \dots, mE(e_j)\}) \cup \{ed_1, \dots, ed_k\} \wedge y \in vertG \cup \{v_1, \dots, v_l\} \wedge (tG_V \cup \{v_1 \mapsto t_1, \dots, v_l \mapsto t_l\})(y) = t \wedge ((\{mE(e_1), \dots, mE(e_j)\} \triangleleft sourceG\{ed_1 \mapsto mV(v_1), \dots, ed_k \mapsto v_l\})(x) = y)$, considering that j edges are deleted, and k edges and l vertices are created. Thus, the tactics for discharging this property were divided in the following cases: the rule creates an edge with source in a vertex of type t ; the rule does not involve edges with source in a vertex of type t ; the rule just preserves (it does not delete and neither create) edges with

source in a vertex of type t ; and the rule preserves and deletes (but it does not create) edges with source in a vertex of type t . Again, we do not consider the case in which the rule just, deletes (it does not preserve and neither create) edges with source in vertices of type t .

A - Rule creates an edge with source in a vertex of type t

In this case, the source of the created edge can be preserved or created. If the source vertex is also created the proof obligation is discharged instantiating the goal with the name ed_t of the created edge and with the name v_t of its source created vertex of type t . In the cases in which a rule creates the edge, but preserves the source vertex of type t , we must follow the next steps:

1. Apply \exists hyp rule in the induction hypothesis;
2. Instantiate, in the goal, the name of the created edge ed_t together with its source vertex $mV(v_t)$ of type t , which is preserved by the rule;
3. Instantiate the name of the preserved vertex v_t of type t in the `grd_vertices` hypothesis;
4. Apply partition rewrites rule in the hypothesis that defines the typing of vertices in the left-hand side of the rule;
5. Run ML.

Figure 20 exhibits the resultant proof tree.

Proof Tree Description: The initial goal is $\exists x, y \cdot x \in (edgeG \setminus \{mE(e_1), \dots, mE(e_j)\}) \cup \{ed_1, \dots, ed_k\} \wedge y \in vertG \cup \{v_1, \dots, v_l\} \wedge (tG_V \cup \{v_1 \mapsto t_1, \dots, v_l \mapsto t_l\})(y) = t \wedge ((\{mE(e_1), \dots, mE(e_j)\} \triangleleft sourceG \{ed_1 \mapsto mV(v_1), \dots, ed_k \mapsto v_l\}))(x) = y$. In order to discharge this sequent we must apply the \exists in rule the induction hypothesis (40), and then some rules are automatically applied (rules 2, 6, 3, 41 and 42 respectively), which results in the following sequent to be demonstrated: $H \vdash \exists x_0, y \cdot x_0 \in (edgeG \setminus \{mE(e_1), \dots, mE(e_j)\}) \cup \{ed_1, \dots, ed_k\} \wedge y \in vertG \cup \{v_1, \dots, v_l\} \wedge (tG_V \cup \{v_1 \mapsto t_1, \dots, v_l \mapsto t_l\})(y) = tG_V(sourceG(x)) \wedge ((\{mE(e_1), \dots, mE(e_j)\} \triangleleft sourceG \cup \{ed_1 \mapsto mV(v_1), \dots, ed_k \mapsto v_l\}))(x_0) = y$. In order to discharge it, we must instantiate the name of the created edge ed_t together with its source vertex $mV(v_t)$. Then, two new sub-goals must be demonstrated: (I) $H \vdash \top \wedge (v_t \in dom(mV) \wedge mV \in vertLi \mapsto \mathbb{Z})$; and (II) $H \vdash ed_t \in (edgeG \setminus \{mE(e_1), \dots, mE(e_j)\}) \cup \{ed_1, \dots, ed_k\} \wedge mV(v_t) \in vertG \cup \{v_1, \dots, v_l\} \wedge (tG_V \cup \{v_1 \mapsto t_1, \dots, v_l \mapsto t_l\})(mV(v_t)) = tG_V(sourceG(x)) \wedge$

$$\begin{array}{c}
 \mathcal{A} = \{mE(e_1), \dots, mE(e_j)\} \quad \mathcal{B} = \{ed_1, \dots, ed_k\} \quad C = \{v_1, \dots, v_t\} \\
 \mathcal{D} = \{v_1 \mapsto t_1, \dots, v_j \mapsto t_j\} \quad \mathcal{E} = \{ed_1 \mapsto mV(v_1), \dots, ed_k \mapsto v_t\} \quad \mathcal{F} = \text{dom}(mV) \wedge mV \in \text{vertLi} \mapsto Z \\
 \\
 \boxed{A} \\
 \\
 \text{ML RULES} \\
 \\
 \frac{\frac{\text{True}}{H \vdash mV(v_t)} \quad 45,45,25}{\frac{\text{True}}{H \vdash \top} \quad 2} \quad \frac{H \vdash tG_V(mV(v_t)) = tG_V(\text{sourceG}(x))}{H \vdash tG_V(mV(v_t)) = tG_V(\text{sourceG}(x))} \quad \frac{47,3,44,48,3,26}{46}}{H \vdash mV(v_t) \in \text{vertG} \cup C \wedge (tG_V \cup \mathcal{D})(mV(v_t)) = tG_V(\text{sourceG}(x))} \quad \frac{9,3,44,28}{46}}{H \vdash mV(v_t) \in \text{vertG} \cup C \wedge (tG_V \cup \mathcal{D})(mV(v_t)) = tG_V(\text{sourceG}(x))} \quad \frac{3}{46}}{H \vdash ed_t \in (\text{edgeG} \setminus \mathcal{A}) \cup \mathcal{B} \wedge mV(v_t) \in \text{vertG} \cup C \wedge (tG_V \cup \mathcal{D})(mV(v_t)) = tG_V(\text{sourceG}(x)) \wedge ((\mathcal{A}) \triangleleft \text{sourceG} \cup \mathcal{E})(ed_t) = mV(v_t)} \\
 \\
 \frac{\frac{\text{True}}{H \vdash \top} \quad 2}{\frac{H \vdash v_t \in \text{vertLi}}{H \vdash v_t \in \text{dom}(mV)} \quad 6} \quad \frac{\text{True}}{H \vdash mV \in \text{vertLi} \mapsto Z} \quad 29}{\frac{H \vdash v_t \in \mathcal{F}}{H \vdash \top \wedge (v_t \in \mathcal{F})} \quad 3} \quad \frac{28}{3}}{H \vdash \exists x, y \cdot x \in (\text{edgeG} \setminus \mathcal{A}) \cup \mathcal{B} \wedge y \in \text{vertG} \cup C \wedge (tG_V \cup \mathcal{D})(y) = tG_V(\text{sourceG}(x)) \wedge ((\mathcal{A}) \triangleleft \text{sourceG} \cup \mathcal{E})(x) = y} \quad \frac{43}{42}}{H \vdash \exists x, y \cdot x \in (\text{edgeG} \setminus \mathcal{A}) \cup \mathcal{B} \wedge y \in \text{vertG} \cup C \wedge (tG_V \cup \mathcal{D})(y) = tG_V(y) \wedge ((\mathcal{A}) \triangleleft \text{sourceG} \cup \mathcal{E})(x) = y} \quad \frac{40,2,6,3,41}{42}}{H \vdash \exists x, y \cdot x \in (\text{edgeG} \setminus \mathcal{A}) \cup \mathcal{B} \wedge y \in \text{vertG} \cup C \wedge (tG_V \cup \mathcal{D})(y) = t \wedge ((\mathcal{A}) \triangleleft \text{sourceG} \cup \mathcal{E})(x) = y}
 \end{array}$$

Figure 20. Proof Tree rule_i/propEdSrcT/INV - Rule creates an edge with source in a vertex of type t

$((\{mE(e_1), \dots, mE(e_j)\}) \triangleleft_{sourceG} \cup \{ed_1 \mapsto mV(v_1), \dots, ed_k \mapsto v_l\})(ed_t) = mV(v_t)$.

Sequent (I) is automatically discharged by a sequence of rules (3, 28, 20, 6, 2, 29). In the same way, in (II) some rules are automatically applied (9, 3, 44, 28, 45, 45, 25, respectively) reaching the goal $H \vdash tG_V(mV(v_t)) = tG_V(sourceG(x))$. In order to demonstrate the last sequent, we must instantiate, in the `grd_vertices` hypothesis (46), the name of the vertex v_t , of type t , that is source of the created edge ed_t . Then, two new sequents must be discharged: (i) $H \vdash \top$, automatically proved (2); and (ii) $H \vdash tG_V(mV(v_t)) = tG_V(sourceG(x))$. For proving (ii) we must apply the partition rewrites rule in the hypothesis that defines the typing of the vertices in the left-hand side of the rule (47) and then, after the automatic application of some rules (3, 44, 48, 3), the final goal is discharged running ML (26). ■

B - Rule does not involve edges with source in vertices of type t

For this case, if the rule does not delete edges, the proof obligation is discharged just running NewPP with lasso. If the considered rule deletes edges, the demonstration must follow the next steps:

1. Apply \exists hyp rule in induction hypothesis;
2. For each edge e_i deleted by the rule, we must follow the next actions:
 - (a) Add $\neg x = mE(e_i)$ as hypothesis, such that e_i is a deleted edge;
 - (b) Instantiate the name of the deleted edge e_i in the `grd_srctgt` hypothesis considering *source*;
 - (c) Apply partition rewrites rule in the hypothesis that defines the source of edges in the left-hand side of the rule;
 - (d) Instantiate the name of the vertex v_i that is source of the deleted edge e_i , in the `grd_vertices` statement;
3. Run NewPP with lasso.

Once the rule does not involve edges with source in vertices of type t , the property is ensured by induction hypothesis. We just have to differentiate the element x (edge that satisfies the property by induction hypothesis) with each edge deleted by the rule, steps (a-d) of the proposed tactic. Figure 21 presents the proof tree after the demonstration.

$$\begin{array}{c}
 \mathcal{A} = \{mE(e_1), \dots, mE(e_j)\} \quad \mathcal{B} = \{ed_1, \dots, ed_k\} \quad \mathcal{C} = \{v_1, \dots, v_l\} \quad \mathcal{D} = \{v_1 \mapsto t_1, \dots, v_l \mapsto t_l\} \quad \mathcal{E} = \{ed_1 \mapsto mV(v_1), \dots, ed_k \mapsto v_l\} \\
 \boxed{\text{B}} \\
 \frac{H \vdash \exists x_0, y \cdot x_0 \in (\text{edge}G \setminus \mathcal{A}) \cup \mathcal{B} \wedge y \in \text{vert}G \cup \mathcal{C} \wedge (tG_V \cup \mathcal{D})(y) = tG_V(\text{source}G(x)) \wedge \text{source}G(x_0) = y}{\text{NewPP RULES}} \quad 11,15 \\
 \boxed{\text{A}} \\
 \frac{H \vdash e_j \in \text{dom}(mE) \wedge mE \in \text{edge}Li \rightarrow \mathbb{Z}}{9} \quad \frac{\text{True}}{H \vdash \top} \quad 2 \quad \frac{\text{True}}{H \vdash \neg x = mE(e_j)} \quad 2 \quad \frac{\text{NewPP RULES}}{H \vdash \neg x = mE(e_j)} \quad 11,15 \quad \frac{3,1,3,3,3,53}{H \vdash \neg x = mE(e_j)} \quad 17 \\
 \frac{H \vdash \exists x_0, y \cdot x_0 \in (\text{edge}G \setminus \mathcal{A}) \cup \mathcal{B} \wedge y \in \text{vert}G \cup \mathcal{C} \wedge (tG_V \cup \mathcal{D})(y) = tG_V(\text{source}G(x)) \wedge \text{source}G(x_0) = y}{52} \\
 \frac{\text{True}}{H \vdash \top} \quad 2 \quad \frac{\text{True}}{H \vdash \top \wedge \top} \quad 3 \quad \frac{\text{True}}{H \vdash \neg x = mE(e_1)} \quad 2 \quad \frac{\text{NewPP RULES}}{H \vdash \neg x = mE(e_1)} \quad 11,15 \quad \frac{3,1,3,3,3,51}{H \vdash \neg x = mE(e_1)} \quad 10 \\
 \frac{H \vdash e_1 \in \text{dom}(mE) \wedge mE \in \text{edge}Li \rightarrow \mathbb{Z}}{9} \quad \frac{H \vdash \exists x_0, y \cdot x_0 \in (\text{edge}G \setminus \mathcal{A}) \cup \mathcal{B} \wedge y \in \text{vert}G \cup \mathcal{C} \wedge (tG_V \cup \mathcal{D})(y) = tG_V(\text{source}G(x)) \wedge \text{source}G(x_0) = y}{50} \\
 \frac{H \vdash \exists x_0, y \cdot x_0 \in (\text{edge}G \setminus \mathcal{A}) \cup \mathcal{B} \wedge y \in \text{vert}G \cup \mathcal{C} \wedge (tG_V \cup \mathcal{D})(y) = tG_V(\text{source}G(x)) \wedge \text{source}G(x_0) = y}{49} \\
 \frac{H \vdash \exists x_0, y \cdot x_0 \in (\text{edge}G \setminus \mathcal{A}) \cup \mathcal{B} \wedge y \in \text{vert}G \cup \mathcal{C} \wedge (tG_V \cup \mathcal{D})(y) = tG_V(\text{source}G(x)) \wedge ((\mathcal{A}) \Leftarrow \text{source}G \cup \mathcal{E})(x_0) = y}{42} \\
 \frac{H \vdash \exists x, y \cdot x \in (\text{edge}G \setminus \mathcal{A}) \cup \mathcal{B} \wedge y_0 \in \text{vert}G \cup \mathcal{C} \wedge (tG_V \cup \mathcal{D})(y_0) = tG_V(y) \wedge ((\mathcal{A}) \Leftarrow \text{source}G \cup \mathcal{E})(x) = y_0}{40,3,6,3,41} \\
 \frac{H \vdash \exists x, y \cdot x \in (\text{edge}G \setminus \mathcal{A}) \cup \mathcal{B} \wedge y \in \text{vert}G \cup \mathcal{C} \wedge (tG_V \cup \mathcal{D})(y) = t \wedge ((\mathcal{A}) \Leftarrow \text{source}G \cup \mathcal{E})(x) = y}{40,3,6,3,41}
 \end{array}$$

Figure 21. Proof Tree rule_i/propEdSpSrcT/INV - Rule does not Involve edges with source in vertices of type t

Proof Tree Description: The first sequent to be demonstrated is $H \vdash \exists x, y \cdot x \in (\text{edge}G \setminus \{mE(e_1), \dots, mE(e_j)\}) \cup \{ed_1, \dots, ed_k\} \wedge y \in \text{vert}G \cup \{v_1, \dots, v_l\} \wedge (tG_V \cup \{v_1 \mapsto t_1, \dots, v_l \mapsto t_l\})(y) = t \wedge ((\{mE(e_1), \dots, mE(e_j)\}) \triangleleft \text{source}G \cup \{ed_1 \mapsto mV(v_1), \dots, ed_k \mapsto v_l\})(x) = y$. First we must apply the \exists hyp rule in the induction hypothesis (40), and, after the automatic application of some rules (3, 6, 3, 41, 42, 49, respectively), the resultant sequent is $H \vdash \exists x_0, y \cdot x_0 \in (\text{edge}G \setminus \{mE(e_1), \dots, mE(e_j)\}) \cup \{ed_1, \dots, ed_k\} \wedge y \in \text{vert}G \cup \{v_1, \dots, v_l\} \wedge (tG_V \cup \{v_1 \mapsto t_1, \dots, v_l \mapsto t_l\})(y) = tG_V(\text{source}G(x_0)) \wedge \text{source}G(x_0) = y$. In order to discharge it, we must add $\neg x = m(e_i)$ as hypothesis, such that e_i is an edge deleted by the rule, which results in three sub-goals: (I) $H \vdash e_i \in \text{dom}(mE) \wedge mE \in \text{edge}Li \mapsto \mathbb{Z}$; (II) $H \vdash \neg x = mE(e_i)$; and (III) $H \vdash \exists x_0, y \cdot x_0 \in (\text{edge}G \setminus \{mE(e_1), \dots, mE(e_j)\}) \cup \{ed_1, \dots, ed_k\} \wedge y \in \text{vert}G \cup \{v_1, \dots, v_l\} \wedge (tG_V \cup \{v_1 \mapsto t_1, \dots, v_l \mapsto t_l\})(y) = tG_V(\text{source}G(x_0)) \wedge \text{source}G(x_0) = y$. (I) is automatically discharged after the automatic application of some rules (9, 3, 2). In order to demonstrate (II), we must instantiate the name of the deleted edge e_i in `grd_srctgt` hypothesis, considering *source*. And this action results in two sequents to be discharged: (i) $H \vdash \top$, automatically discharged (2); and (ii) $H \vdash \neg x = mE(e_i)$. To discharge (ii) we must apply the partition rewrites rule in the hypothesis that defines the source of the edges in the left-hand side of the rule (31), and then, some rules are automatically applied (3, 33 and 3, respectively). Next, we must instantiate in `grd_vertices` hypothesis the name of the vertex v , which is source of the deleted edge e_i , generating two new sub-goals: (i) $H \vdash \top$, automatically discharged (2); and (ii) $H \vdash \neg x = mE(e_i)$, discharged running NewPP with lasso. In the proof tree, the last sequence of steps is illustrated for e_1 and e_j , omitting the sub-trees for the remaining deleted edges. After all differentiations, the sequent (in sub-tree $\boxed{\text{B}}$) $H \vdash \exists x_0, y \cdot x_0 \in (\text{edge}G \setminus \{mE(e_1), \dots, mE(e_j)\}) \cup \{ed_1, \dots, ed_k\} \wedge y \in \text{vert}G \cup \{v_1, \dots, v_l\} \wedge (tG_V \cup \{v_1 \mapsto t_1, \dots, v_l \mapsto t_l\})(y) = tG_V(\text{source}G(x_0)) \wedge \text{source}G(x_0) = y$ is proved running the NewPP with lasso prover (15). ■

C - Rule preserves and does not create edges with source in vertices of type t

We distinguish the proof obligation to be demonstrated in two cases. First considering that the rule does not delete edges. For such case, the goal is demonstrated just running NewPP with lasso. In the second case, edges (possibly with source in vertices of type t) are deleted by the rule. The tactic to discharge the goal in this case is described next:

1. Instantiate $mE(e_p)$ and $mV(v_t)$ in the goal, such that e_p is the preserved edge that has source in a vertex v_t of type t ;
2. Apply partition rewrites rule in the hypothesis that defines the set of edges in the left-hand side of the rule and run ML;

3. Instantiate the name of the vertex v_t that is source of the preserved edge e_p in `grd_vertices` hypothesis;
4. Apply partition rewrites rule in the hypothesis that defines the typing of vertices in the left-hand side of the rule and run ML;
5. Instantiate the name of the preserved edge e_p in the `grd_srctgt` hypothesis, considering *source*;
6. Apply partition rewrites rule in the hypothesis that defines the source of the edges in the left-hand side of the rule;
7. Run ML.

The generated proof tree is presented in Figure 22.

Proof Tree Description: The sequent to be demonstrated is $H \vdash \exists x, y \cdot x \in (edgeG \setminus \{mE(e_1), \dots, mE(e_j)\} \cup \{ed_1, \dots, ed_k\}) \wedge y \in vertG \wedge tG_V(y) = t \wedge ((\{mE(e_1), \dots, mE(e_j)\} \triangleleft sourceG) \cup \{ed_1 \mapsto mV(v_1), \dots, ed_k \mapsto v_l\})(x) = y$. In order to discharge it, we must first instantiate $mE(e_p)$ and $mV(v_t)$ in the goal, such that e_p is the preserved edge with source in a vertex v_t of type t (54). Then, two new sub-goals are generated: (I) $H \vdash (v_t \in dom(mV) \wedge mV \in vertLi \mapsto \mathbb{Z}) \wedge (e_p \in dom(mE) \wedge mE \in edgeLi \mapsto \mathbb{Z})$ and (II) $H \vdash \exists x, y \cdot x \in (edgeG \setminus \{mE(e_1), \dots, mE(e_j)\} \cup \{ed_1, \dots, ed_k\}) \wedge y \in vertG \wedge tG_V(mV(v_t)) = t \wedge ((\{mE(e_1), \dots, mE(e_j)\} \triangleleft sourceG) \cup \{ed_1 \mapsto v_1, \dots, ed_k \mapsto v_l\})(x) = y$. (I) is automatically discharged after the automatic application of some rules (respectively, 3, 9, 3, 6, 3, 40, 42, 2, 20, 6, 2, 29, 20, 6 and 2). To discharge (II) (sub-tree A), some rules (3, 9, 3, 6, 3, 40, 41, 42 and 28, respectively) are automatically applied generating four sub-goals: (i) $H \vdash mE(e_p) \in (edgeG \setminus \{mE(e_1), \dots, mE(e_j)\} \cup \{ed_1, \dots, ed_k\})$; (ii) $H \vdash mV(v_t) \in vertG$; (iii) $H \vdash tG_V(mV(v_t)) = tG_V(sourceG(x))$; and (iv) $H \vdash ((\{mE(e_1), \dots, mE(e_j)\} \triangleleft sourceG) \cup \{ed_1 \mapsto v_1, \dots, ed_k \mapsto v_l\})(mE(e_p)) = mV(v_t)$. With the aim of discharging (i), we must apply the partition rewrites rule (55) in the hypothesis that defines the set of edges in the left-hand side of the rule. Next, the use equality hypothesis - eh rule (56) is automatically applied and the remaining sub-goal is proved by running ML (26). Sequent (ii) is automatically discharged (by rules 45 and 25). In order to demonstrate (iii) (sub-tree B), we must instantiate in `grd_vertices` hypothesis the name of the vertex v_t , which is source of edge e_p (46). Next, two new sub-goals must be discharged: (i) $H \vdash \top$, automatically discharged by \top goal (2) and (ii) $H \vdash tG_V(mV(v_t)) = tG_V(sourceG(x))$, which is discharged applying the partition rewrites rule (57) in the hypothesis that defines the typing of the vertices in the left-hand side of the rule. Then, after the automatic application of some rules (57, 9, 3, 44, 58 and 3, respectively), the proof is concluded running ML (26). In order to demonstrate (iv) (sub-tree

$$\begin{array}{c}
 \mathcal{A} = \{mE(e_1), \dots, mE(e_j)\} \\
 \mathcal{E} = \{ed_1 \mapsto mV(v_1), \dots, ed_k \mapsto v_l\} \\
 \mathcal{B} = \text{dom}(mE) \wedge mE \in \text{edgeLi} \rightarrow \mathcal{Z} \\
 \mathcal{F} = \text{dom}(mV) \wedge mV \in \text{vertLi} \rightarrow \mathcal{Z} \\
 \mathcal{C} = \{ed_1, \dots, ed_k\} \quad \mathcal{D} = \{ed_1 \mapsto mV(v_1), \dots, ed_k \mapsto mV(v_l)\} \\
 \boxed{\text{C}} \\
 \frac{\text{True} \quad 2}{H \vdash \top} \quad \frac{\text{ML RULES} \quad 31,3,59,60,32,33,3,26}{H \vdash ((\mathcal{A} \Leftarrow \text{sourceG}) \cup \mathcal{E})(mE(ep)) = mV(vt)} \quad \frac{82}{H \vdash ((\mathcal{A} \Leftarrow \text{sourceG}) \cup \mathcal{E})(mE(ep)) = mV(vt)} \\
 \boxed{\text{B}} \\
 \frac{\text{True} \quad 2}{H \vdash \top} \quad \frac{\text{ML RULES} \quad 57,9,3,44,58,3,26}{H \vdash tG_V(mV(vt)) = tG_V(\text{sourceG}(x))} \quad \frac{46}{H \vdash tG_V(mV(vt)) = tG_V(\text{sourceG}(x))} \\
 \boxed{\text{A}} \\
 \frac{\text{ML RULES} \quad 55,56,26}{H \vdash mE(ep) \in (\text{edgeG} \setminus \mathcal{A}) \cup \mathcal{C}} \quad \frac{\text{True} \quad 45,45,25}{H \vdash mV(vt) \in \text{vertG}} \quad \frac{\boxed{\text{B}} \quad \boxed{\text{C}}}{H \vdash mE(ep) \in (\text{edgeG} \setminus \mathcal{A} \cup \mathcal{C}) \wedge mV(vt) \in \text{vertG} \wedge tG_V(mV(vt)) = tG_V(\text{sourceG}(x)) \wedge ((\mathcal{A} \Leftarrow \text{sourceG}) \cup \mathcal{E})(mE(ep)) = mV(vt)} \quad \frac{28}{H \vdash mE(ep) \in (\text{edgeG} \setminus \mathcal{A} \cup \mathcal{C}) \wedge mV(vt) \in \text{vertG} \wedge tG_V(mV(vt)) = tG_V(\text{sourceG}(x)) \wedge ((\mathcal{A} \Leftarrow \text{sourceG}) \cup \mathcal{E})(mE(ep)) = mV(vt)} \quad \frac{42}{H \vdash mE(ep) \in (\text{edgeG} \setminus \mathcal{A} \cup \mathcal{C}) \wedge mV(vt) \in \text{vertG} \wedge tG_V(mV(vt)) = tG_V(\text{sourceG}(x)) \wedge ((\mathcal{A} \Leftarrow \text{sourceG}) \cup \mathcal{E})(mE(ep)) = mV(vt)} \quad \frac{3,9,3,6,3,40,41}{H \vdash mE(ep) \in (\text{edgeG} \setminus \mathcal{A} \cup \mathcal{C}) \wedge mV(vt) \in \text{vertG} \wedge tG_V(mV(vt)) = t \wedge ((\mathcal{A} \Leftarrow \text{sourceG}) \cup \mathcal{E})(mE(ep)) = mV(vt)} \\
 \frac{\text{True} \quad 2}{H \vdash \top} \quad \frac{6}{H \vdash vt \in \text{vertLi}} \quad \frac{20}{H \vdash vt \in \text{dom}(mV)} \quad \frac{\text{True} \quad 20}{H \vdash mV \in \text{vertLi} \rightarrow \mathcal{Z}} \quad \frac{6,3,40,42}{H \vdash vt \in \mathcal{F} \wedge ep \in \text{dom}(mE)} \quad \frac{3}{H \vdash vt \in \mathcal{F} \wedge ep \in \text{dom}(mE) \wedge \top} \quad \frac{9}{H \vdash vt \in \mathcal{F} \wedge ep \in \mathcal{B}} \quad \frac{3}{H \vdash (vt \in \mathcal{F}) \wedge (ep \in \mathcal{B})} \\
 \frac{54}{H \vdash \exists x, y. x \in (\text{edgeG} \setminus \mathcal{A} \cup \mathcal{C}) \wedge y \in \text{vertG} \wedge tG_V(y) = t \wedge ((\mathcal{A} \Leftarrow \text{sourceG}) \cup \mathcal{E})(x) = y} \\
 \boxed{\text{A}}
 \end{array}$$

Figure 22. Proof Tree rule_i/propEdsSrcT/INV - Rule preserves and does not create edges with source in vertices of type t

\boxed{C}), we must instantiate the name of the preserved edge e_p in `grd_srctgt` hypothesis, considering `source` (82). Then, two new sub-goals are generated: (i) $H \vdash \top$, automatically discharged (2) and (ii) $H \vdash ((\{mE(e_1), \dots, mE(e_j)\} \triangleleft sourceG) \cup \{ed_1 \mapsto v_1, \dots, ed_k \mapsto v_l\})(mE(e_p)) = mV(v_t)$, which is demonstrated following the next steps: first, applying the partition rewrites rule in the hypothesis that defines the source of the edges in the left-hand side of the rule (31); then some rules are automatically applied (3, 59, 60, 32, 33 and 3, respectively); and finally, we must run ML (26). ■

5.4 `propNotIsoVT` property

The `propNotIsoVT` property, $\forall x \cdot ((x \in vertG \wedge tG_V(x) = t) \Rightarrow (\exists y \cdot (y \in edgeG \wedge ((sourceG(y) = x \wedge \neg targetG(y) = x) \vee (targetG(y) = x \wedge \neg sourceG(y) = x))))))$, states that any reachable graph does not have an isolated vertex of type t . The proof obligation INITIALISATION/propNotIsoVT/INV generated by the initialisation event is discharged following the next steps. The corresponding proof tree is represented in Figure 23.

1. Eliminate the universal quantifier in the goal;
2. Remove the \in operator in the hypothesis $x \in vertG$;
3. Start the proof by cases in the hypothesis $x = v_1 \vee \dots \vee x = v_n$;
4. For each case, apply the following actions:
 - (a) For each vertex $v_i, i \in \{1, \dots, n\}$ of type t , we must instantiate in the goal the name of its (no loop) incident edge;
 - (b) For each vertex $v_j, j \in \{1, \dots, n\}$ of type different from t , run ML.

Proof Tree Description: The goal to be demonstrated is $\vdash \forall x \cdot x \in vertG \wedge tG_V(x) = t \Rightarrow (\exists y \cdot y \in edgeG \wedge ((sourceG(y) = x \wedge \neg targetG(y) = x) \vee (targetG(y) = x \wedge \neg sourceG(y) = x)))$. In order to discharge it, we must eliminate the universal quantifier in the goal (61). This action together with the automatic application of rule (62) results in the sub-goal $tG_V(x) = t, x \in \{v_1, \dots, v_n\} \vdash \exists y \cdot y \in edgeG \wedge ((sourceG(y) = x \wedge \neg targetG(y) = x) \vee (targetG(y) = x \wedge \neg sourceG(y) = x))$. With the aim of demonstrating this sequent, first we must remove the \in operator in $x \in vertG$ (63) hypothesis. Then, we must select proof by cases in $x = v_1 \vee \dots \vee x = v_n$ hypothesis. Therewith, after the automatic application of some rules (9, 3, 65 or 66 and 3), remains some sub-goals: (i) $x = v_i \vdash \exists y \cdot y \in edgeG \wedge ((sourceG(y) = v_i \wedge \neg targetG(y) = v_i) \vee (targetG(y) = v_i \wedge \neg sourceG(y) = v_i))$; and (ii) $x = v_j, t = t_k \vdash \exists y \cdot y \in edgeG \wedge ((sourceG(y) = v_j \wedge \neg targetG(y) = v_j) \vee (targetG(y) = v_j \wedge \neg sourceG(y) = v_j))$. Sequent (i) represents the case in that the considered vertex v_i is of type t . To discharge it, we must instantiate in the goal the name of the edge ed_t incident in this vertex and the proof is automatically concluded

B	
ML RULES	26
$x = v_j, t = t_k \vdash \exists y. y \in \text{edgeG} \wedge ((\text{sourceG}(y) = v_j \vee \neg \text{targetG}(y) = v_j) \vee (\text{targetG}(y) = v_j \wedge \neg \text{sourceG}(y) = v_j))$	3
$x = v_j, tG_V(v_j) = t \vdash \exists y. y \in \text{edgeG} \wedge ((\text{sourceG}(y) = v_j \wedge \neg \text{targetG}(y) = v_j) \vee (\text{targetG}(y) = v_j \wedge \neg \text{sourceG}(y) = v_j))$	9,3,66
$tG_V(x) = t, x = v_j \vdash \exists y. y \in \text{edgeG} \wedge ((\text{sourceG}(y) = x \wedge \neg \text{targetG}(y) = x) \vee (\text{targetG}(y) = x \wedge \neg \text{sourceG}(y) = x))$	3
A	2
True	2
$x = v_i \vdash \neg T$	3
$x = v_i \vdash \text{ed}_t \in \text{edgeG} \wedge ((\text{sourceG}(\text{ed}_t) = v_i \wedge \neg \text{targetG}(\text{ed}_t) = v_i) \vee (\text{targetG}(\text{ed}_t) = v_i \wedge \neg \text{sourceG}(\text{ed}_t) = v_i))$	4
$x = v_i \vdash \exists y. y \in \text{edgeG} \wedge ((\text{sourceG}(y) = v_i \wedge \neg \text{targetG}(y) = v_i) \vee (\text{targetG}(y) = v_i \wedge \neg \text{sourceG}(y) = v_i))$	3
$x = v_t, tG_V(v_i) = t \vdash \exists y. y \in \text{edgeG} \wedge ((\text{sourceG}(y) = v_t \wedge \neg \text{targetG}(y) = v_t) \vee (\text{targetG}(y) = v_t \wedge \neg \text{sourceG}(y) = v_t))$	9,3,65
$tG_V(x) = t, x = v_i \vdash \exists y. y \in \text{edgeG} \wedge ((\text{sourceG}(y) = x \wedge \neg \text{targetG}(y) = x) \vee (\text{targetG}(y) = x \wedge \neg \text{sourceG}(y) = x))$	3
A	64
$tG_V(x) = t, x = v_1 \vee \dots \vee x = v_n \vdash \exists y. y \in \text{edgeG} \wedge ((\text{sourceG}(y) = x \wedge \neg \text{targetG}(y) = x) \vee (\text{targetG}(y) = x \wedge \neg \text{sourceG}(y) = x))$	63
$tG_V(x) = t, x \in \{v_1, \dots, v_n\} \vdash \exists y. y \in \text{edgeG} \wedge ((\text{sourceG}(y) = x \wedge \neg \text{targetG}(y) = x) \vee (\text{targetG}(y) = x \wedge \neg \text{sourceG}(y) = x))$	62
$\vdash x \in \text{vertG} \wedge tG_V(x) = t \Rightarrow (\exists y. y \in \text{edgeG} \wedge ((\text{sourceG}(y) = x \wedge \neg \text{targetG}(y) = x) \vee (\text{targetG}(y) = x \wedge \neg \text{sourceG}(y) = x)))$	61
$\vdash \forall x. x \in \text{vertG} \wedge tG_V(x) = t \Rightarrow (\exists y. y \in \text{edgeG} \wedge ((\text{sourceG}(y) = x \wedge \neg \text{targetG}(y) = x) \vee (\text{targetG}(y) = x \wedge \neg \text{sourceG}(y) = x)))$	61

Figure 23. Proof Tree for INITIALISATION/propNotIso VT/INV

with \top goal and simplification rewrites rules (2 and 3). In turn, sequent (ii) represents the case that vertex v_j is not of type t , which is discharged running ML (26). It should be noted that similar sequents to (i) and (ii) can be generated for each different vertex of type t or of type different from t , respectively. All of them (conforming the case) are discharged following the described steps. ■

Considering the `propNotIsoVT` property, a rule can delete and create edges and/or create vertices. Then, proof obligations follows the following pattern $\forall x \cdot x \in \text{vert}G \cup \{v_1, \dots, v_l\} \wedge (tG_V \cup \{v_1 \mapsto t_1, \dots, v_l \mapsto t_l\})(x) = t \Rightarrow (\exists y \cdot y \in (\text{edge}G \setminus \{mE(e_1), \dots, mE(e_j)\}) \cup \{ed_1, \dots, ed_k\} \wedge (((\{mE(e_1), \dots, mE(e_j)\} \triangleleft \text{source}G \cup \{ed_1 \mapsto t_1, \dots, ed_k \mapsto t_k\})(y) = x \wedge \neg((\{mE(e_1), \dots, mE(e_j)\} \triangleleft \text{target}G \cup \{ed_1 \mapsto t_1, \dots, ed_k \mapsto t_k\})(y) = x) \vee (((\{mE(e_1), \dots, mE(e_j)\} \triangleleft \text{target}G \cup \{ed_1 \mapsto t_1, \dots, ed_k \mapsto t_k\})(y) = x \wedge \neg((\{mE(e_1), \dots, mE(e_j)\} \triangleleft \text{source}G \cup \{ed_1 \mapsto t_1, \dots, ed_k \mapsto t_k\})(y) = x))))$, considering that j edges are deleted, k edges and l vertices are created by a rule application. Thus, we consider the next cases: the rule creates and does not preserve vertices of type t ; the rule creates and preserves vertices of type t ; the rule preserves and does not create vertices of type t ; and rule does not involve vertices of type t . We do not consider cases that the rule deletes edges with source or target in vertices of type t .

A - Rule creates, preserving or not, vertices of type t

For rules that do not delete edges, the proof obligation is discharged running NewPP with lasso. For rules that delete edges (which have no source or target in vertices of type t), the demonstration of the corresponding proof obligation follows the next steps. The generated proof tree is illustrated in Figure 24.

1. Eliminate the universal quantifier in the goal;
2. Remove the \in operator in $x \in \text{vert}G \cup \{v_1, \dots, v_l\}$ hypothesis;
3. Start the proof by cases from $x \in \text{vert}G \vee x = v_1 \vee \dots \vee x = v_l$ hypothesis and for case $x \in \text{vert}G$ we must first follow the next steps:
 - (a) Instantiate element x in the induction hypothesis $\forall x \cdot x \in \text{vert}G \wedge tG_V(x) = t \Rightarrow (\exists y \cdot y \in \text{edge}G \wedge ((\text{source}G(y) = x \wedge \neg \text{target}G(y) = x) \vee (\text{target}G(y) = x \wedge \neg \text{source}G(y) = x)))$;
 - (b) Apply Modus Ponens from \Rightarrow in $tG_V(x) = t \Rightarrow (\exists y \cdot y \in \text{edge}G \wedge ((\text{source}G(y) = x \wedge \neg \text{target}G(y) = x) \vee (\text{target}G(y) = x \wedge \neg \text{source}G(y) = x)))$ hypothesis and run ML;
 - (c) Instantiate in the goal the element y (representing the edge that is incident in the vertex of type t) and in the next two sub-goals run NewPP with lasso;
4. Run NewPP with lasso;

Proof Tree Description: The sequent to be discharged is $H \vdash \forall x \cdot x \in \text{vert}G \cup \{v_1, \dots, v_l\} \wedge (tG_V \cup \{v_1 \mapsto t_1, \dots, v_l \mapsto t_l\})(x) = t \Rightarrow (\exists y \cdot y \in (\text{edge}G \setminus \{mE(e_1), \dots, mE(e_j)\}) \cup \{ed_1, \dots, ed_k\} \wedge (((\{mE(e_1), \dots, mE(e_j)\} \triangleleft \text{source}G \cup \{ed_1 \mapsto t_1, \dots, ed_k \mapsto t_k\})(y) = x \wedge \neg((\{mE(e_1), \dots, mE(e_j)\} \triangleleft \text{target}G \cup \{ed_1 \mapsto t_1, \dots, ed_k \mapsto t_k\})(y) = x) \vee (((\{mE(e_1), \dots, mE(e_j)\} \triangleleft \text{target}G \cup \{ed_1 \mapsto t_1, \dots, ed_k \mapsto t_k\})(y) = x \wedge \neg((\{mE(e_1), \dots, mE(e_j)\} \triangleleft \text{source}G \cup \{ed_1 \mapsto t_1, \dots, ed_k \mapsto t_k\})(y) = x))))$. With the aim of demonstrating it, we must eliminate the universal quantifier in the goal (61). Next, some rules are automatically applied (6, 3 and 62, respectively) generating the sub-goal $H \vdash \exists y \cdot y \in (\text{edge}G \setminus \{mE(e_1), \dots, mE(e_j)\}) \cup \{ed_1, \dots, ed_k\} \wedge (((\{mE(e_1), \dots, mE(e_j)\} \triangleleft \text{source}G \cup \{ed_1 \mapsto t_1, \dots, ed_k \mapsto t_k\})(y) = x \wedge \neg((\{mE(e_1), \dots, mE(e_j)\} \triangleleft \text{target}G \cup \{ed_1 \mapsto t_1, \dots, ed_k \mapsto t_k\})(y) = x) \vee (((\{mE(e_1), \dots, mE(e_j)\} \triangleleft \text{target}G \cup \{ed_1 \mapsto t_1, \dots, ed_k \mapsto t_k\})(y) = x \wedge \neg((\{mE(e_1), \dots, mE(e_j)\} \triangleleft \text{source}G \cup \{ed_1 \mapsto t_1, \dots, ed_k \mapsto t_k\})(y) = x)))$. After, it is necessary to remove the \in operator in $x \in \text{vert}G \cup \{v_1, \dots, v_l\}$ hypothesis (67). Then, after a simplification (3), we must select proof by cases in the $x \in \text{vert}G \vee x = v_1 \vee \dots \vee x = v_l$ hypothesis (68). In what follows, we consider two kinds of generated sub-goals, first considering a vertex x belonging to $\text{vert}G$ and second considering a created vertex: (I) $H \vdash \exists y \cdot y \in (\text{edge}G \setminus \{mE(e_1), \dots, mE(e_j)\}) \cup \{ed_1, \dots, ed_k\}$

$\wedge (((\{mE(e_1), \dots, mE(e_j)\} \triangleleft \text{source}G \cup \{ed_1 \mapsto t_1, \dots, ed_k \mapsto t_k\})(y) = x \wedge \neg((\{mE(e_1), \dots, mE(e_j)\} \triangleleft \text{target}G \cup \{ed_1 \mapsto t_1, \dots, ed_k \mapsto t_k\})(y) = x) \vee (((\{mE(e_1), \dots, mE(e_j)\} \triangleleft \text{target}G \cup \{ed_1 \mapsto t_1, \dots, ed_k \mapsto t_k\})(y) = x \wedge \neg((\{mE(e_1), \dots, mE(e_j)\} \triangleleft \text{source}G \cup \{ed_1 \mapsto t_1, \dots, ed_k \mapsto t_k\})(y) = x)))$ and (II) $H \vdash \exists y \cdot y \in (\text{edge}G \setminus \{mE(e_1), \dots, mE(e_j)\}) \cup \{ed_1, \dots, ed_k\} \wedge (((\{mE(e_1), \dots, mE(e_j)\} \triangleleft \text{source}G \cup \{ed_1 \mapsto t_1, \dots, ed_k \mapsto t_k\})(y) = v_t \wedge \neg((\{mE(e_1), \dots, mE(e_j)\} \triangleleft \text{target}G \cup \{ed_1 \mapsto t_1, \dots, ed_k \mapsto t_k\})(y) = v_t) \vee (((\{mE(e_1), \dots, mE(e_j)\} \triangleleft \text{target}G \cup \{ed_1 \mapsto t_1, \dots, ed_k \mapsto t_k\})(y) = v_t \wedge \neg((\{mE(e_1), \dots, mE(e_j)\} \triangleleft \text{source}G \cup \{ed_1 \mapsto t_1, \dots, ed_k \mapsto t_k\})(y) = v_t)))$. In (I) (sub-tree A), some rules are automatically applied (9 and 3), then we must instantiate the element x in induction hypothesis (69). Such instantiation generates the sub-goals $H \vdash \top$, which is automatically discharged (2) and $H \vdash \exists y \cdot y \in (\text{edge}G \setminus \{mE(e_1), \dots, mE(e_j)\}) \cup \{ed_1, \dots, ed_k\} \wedge (((\{mE(e_1), \dots, mE(e_j)\} \triangleleft \text{source}G \cup \{ed_1 \mapsto t_1, \dots, ed_k \mapsto t_k\})(y) = x \wedge \neg((\{mE(e_1), \dots, mE(e_j)\} \triangleleft \text{target}G \cup \{ed_1 \mapsto t_1, \dots, ed_k \mapsto t_k\})(y) = x) \vee (((\{mE(e_1), \dots, mE(e_j)\} \triangleleft \text{target}G \cup \{ed_1 \mapsto t_1, \dots, ed_k \mapsto t_k\})(y) = x \wedge \neg((\{mE(e_1), \dots, mE(e_j)\} \triangleleft \text{source}G \cup \{ed_1 \mapsto t_1, \dots, ed_k \mapsto t_k\})(y) = x)))$. In the last sequent, after some automatic rule applications (9 and 3, respectively), we must apply Modus Ponens in the generated hypothesis (70). After this action two new sequents must be demonstrated: (i) $H \vdash tG_V(x) = t_k$ and (ii) $H \vdash \exists y \cdot y \in (\text{edge}G \setminus \{mE(e_1), \dots, mE(e_j)\}) \cup \{ed_1, \dots, ed_k\} \wedge (((\{mE(e_1), \dots, mE(e_j)\} \triangleleft \text{source}G \cup \{ed_1 \mapsto t_1, \dots, ed_k \mapsto t_k\})(y) = x \wedge \neg((\{mE(e_1), \dots, mE(e_j)\} \triangleleft \text{target}G \cup \{ed_1 \mapsto t_1, \dots, ed_k \mapsto t_k\})(y) = x) \vee (((\{mE(e_1), \dots, mE(e_j)\} \triangleleft \text{target}G \cup \{ed_1 \mapsto t_1, \dots, ed_k \mapsto t_k\})(y) = x \wedge \neg((\{mE(e_1), \dots, mE(e_j)\} \triangleleft \text{source}G \cup \{ed_1 \mapsto t_1, \dots, ed_k \mapsto t_k\})(y) = x)))$

x). In order to demonstrate (i), we must run ML (26). In (ii) the \exists hyp rule (71) is automatically applied. Then, we must instantiate in the goal the element y (that represents the created edge that is incident in the created vertex of type t , from induction hypothesis (72)), generating various sub-goals, which are discharged by the \top goal rule (2) and running NewPP with lasso (15). In sequent (II) (sub-tree $\boxed{\mathbf{B}}$), after the automatic application of some rules (9, 3, 73, 3 and 11, respectively), the final goal is discharged running NewPP with lasso (15). ■

B - Rule preserves and does not create vertices of type t

In case that no edge is deleted by the rule, the proof obligation is demonstrated running NewPP with lasso. When a rule deletes an edge (with no source or target in a vertex of type t) the proof must follow the next steps:

1. Eliminate the universal quantifier in the goal;
2. Instantiate the element x in the induction hypothesis;
3. Instantiate the element y in the goal;
4. Apply one lasso operation in the goal;
5. Run NewPP with lasso;
6. Run ML.

Figure 25 presents the corresponding proof tree.

Proof Tree Description: The sequent to be demonstrated is $H \vdash \forall x \cdot x \in \text{vert}G \wedge tG_V(x) = t \Rightarrow (\exists y \cdot y \in (\text{edge}G \setminus \{mE(e_1), \dots, mE(e_j)\}) \cup \{ed_1, \dots, ed_k\}) \wedge (((\{mE(e_1), \dots, mE(e_j)\} \triangleleft \text{source}G \cup \{ed_1 \mapsto t_1, \dots, ed_k \mapsto t_k\})(y) = x \wedge \neg((\{mE(e_1), \dots, mE(e_j)\} \triangleleft \text{target}G \cup \{ed_1 \mapsto t_1, \dots, ed_k \mapsto t_k\})(y) = x) \vee (((\{mE(e_1), \dots, mE(e_j)\} \triangleleft \text{target}G \cup \{ed_1 \mapsto t_1, \dots, ed_k \mapsto t_k\})(y) = x \wedge \neg((\{mE(e_1), \dots, mE(e_j)\} \triangleleft \text{source}G \cup \{ed_1 \mapsto t_1, \dots, ed_k \mapsto t_k\})(y) = x)))$. In order to discharge it, we must first eliminate the universal quantifier in the goal (61). Then, after the automatic application of some rules (6, 3, 62 and 74, respectively), the following sequent must be discharged $H \vdash \exists y \cdot y \in (\text{edge}G \setminus \{mE(e_1), \dots, mE(e_j)\}) \cup \{ed_1, \dots, ed_k\} \wedge (((\{mE(e_1), \dots, mE(e_j)\} \triangleleft \text{source}G \cup \{ed_1 \mapsto t_1, \dots, ed_k \mapsto t_k\})(y) = x \wedge \neg((\{mE(e_1), \dots, mE(e_j)\} \triangleleft \text{target}G \cup \{ed_1 \mapsto t_1, \dots, ed_k \mapsto t_k\})(y) = x) \vee (((\{mE(e_1), \dots, mE(e_j)\} \triangleleft \text{target}G \cup \{ed_1 \mapsto t_1, \dots, ed_k \mapsto t_k\})(y) = x \wedge \neg((\{mE(e_1), \dots, mE(e_j)\} \triangleleft \text{source}G \cup \{ed_1 \mapsto t_1, \dots,$

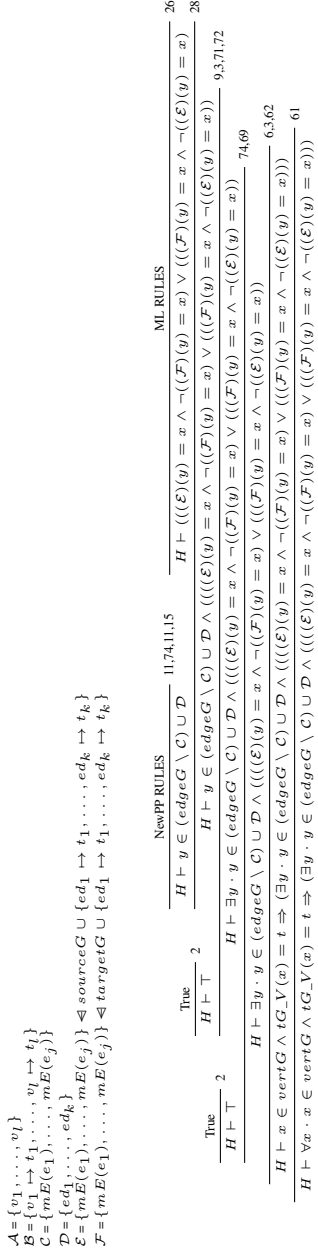


Figure 25. Proof Tree for rule `_i/propNotIsoVT/INV` - Rule preserves and does not create vertices of type `t`

$ed_k \mapsto t_k\})(y = x)$). Aiming to proof it, we must instantiate x in the induction hypothesis (69), remaining two sub-goals: (I) $H \vdash \top$ automatically discharged by \top goal (2) and (II) $H \vdash \exists y \cdot y \in (\text{edge}G \setminus \{mE(e_1), \dots, mE(e_j)\}) \cup \{ed_1, \dots, ed_k\} \wedge (((\{mE(e_1), \dots, mE(e_j)\} \triangleleft \text{source}G \cup \{ed_1 \mapsto t_1, \dots, ed_k \mapsto t_k\})(y = x) \wedge \neg((\{mE(e_1), \dots, mE(e_j)\} \triangleleft \text{target}G \cup \{ed_1 \mapsto t_1, \dots, ed_k \mapsto t_k\})(y = x) \vee (((\{mE(e_1), \dots, mE(e_j)\} \triangleleft \text{target}G \cup \{ed_1 \mapsto t_1, \dots, ed_k \mapsto t_k\})(y = x) \wedge \neg((\{mE(e_1), \dots, mE(e_j)\} \triangleleft \text{source}G \cup \{ed_1 \mapsto t_1, \dots, ed_k \mapsto t_k\})(y = x))$). In (II) some rules are automatically applied (9, 3 and 71, respectively), and next we must instantiate y in the remaining goal (72). Such instantiation takes to two sub-goals: (i) $H \vdash \top$, automatically discharged (2) and (ii) $H \vdash y \in (\text{edge}G \setminus \{mE(e_1), \dots, mE(e_j)\}) \cup \{ed_1, \dots, ed_k\} \wedge (((\{mE(e_1), \dots, mE(e_j)\} \triangleleft \text{source}G \cup \{ed_1 \mapsto t_1, \dots, ed_k \mapsto t_k\})(y = x) \wedge \neg((\{mE(e_1), \dots, mE(e_j)\} \triangleleft \text{target}G \cup \{ed_1 \mapsto t_1, \dots, ed_k \mapsto t_k\})(y = x) \vee (((\{mE(e_1), \dots, mE(e_j)\} \triangleleft \text{target}G \cup \{ed_1 \mapsto t_1, \dots, ed_k \mapsto t_k\})(y = x) \wedge \neg((\{mE(e_1), \dots, mE(e_j)\} \triangleleft \text{source}G \cup \{ed_1 \mapsto t_1, \dots, ed_k \mapsto t_k\})(y = x))$. In (ii), the \wedge goal rule is automatically applied (28), generating the following sequents: (i) $H \vdash y \in (\text{edge}G \setminus \{mE(e_1), \dots, mE(e_j)\}) \cup \{ed_1, \dots, ed_k\}$; and (ii) $H \vdash (((\{mE(e_1), \dots, mE(e_j)\} \triangleleft \text{source}G \cup \{ed_1 \mapsto t_1, \dots, ed_k \mapsto t_k\})(y = x) \wedge \neg((\{mE(e_1), \dots, mE(e_j)\} \triangleleft \text{target}G \cup \{ed_1 \mapsto t_1, \dots, ed_k \mapsto t_k\})(y = x) \vee (((\{mE(e_1), \dots, mE(e_j)\} \triangleleft \text{target}G \cup \{ed_1 \mapsto t_1, \dots, ed_k \mapsto t_k\})(y = x) \wedge \neg((\{mE(e_1), \dots, mE(e_j)\} \triangleleft \text{source}G \cup \{ed_1 \mapsto t_1, \dots, ed_k \mapsto t_k\})(y = x))$. In (i) we must execute a lasso operation, automatically discharging some rules (11, 74 and 11, respectively) and the last sub-goal is demonstrated running NewPP with lasso (15). Sequent (ii) is demonstrated running ML. ■

C - Rule does not involve vertices of type t

If no edge is deleted by the rule, the generated proof obligation is discharged running NewPP with lasso. If the rule delete edges (with no source or target in vertices of type t), the proof can be executed as follows. Figure 26 presents the proof tree.

1. Eliminate the universal quantifier in the goal;
2. Remove the \in in operator in the $x \in \text{vert}G \cup \{v_1, \dots, v_l\}$ hypothesis;
3. Select proof by cases in the $x \in \text{vert}G \vee x = v_1 \vee \dots \vee x = v_l$ hypothesis;
4. Instantiate x in the induction hypothesis;
5. Apply Modus Ponens in $tG_V(x) = t \Rightarrow (\exists y \cdot y \in \text{edge}G \wedge ((\text{source}G(y) = x \wedge \neg \text{target}G(y) = x) \vee (\text{target}G(y) = x \wedge \neg \text{source}G(y) = x)))$ hypothesis and in the next sub-goals run NewPP with lasso;
6. Instantiate y in the goal and run NewPP with lasso in the remaining sub-goals.

$$\begin{aligned} \mathcal{A} &= \{v_1, \dots, v_l\} & \mathcal{B} &= \{v_1 \mapsto t_1, \dots, v_l \mapsto t_l\} & \mathcal{C} &= \{mE(e_1), \dots, mE(e_j)\} & \mathcal{D} &= \{ed_1, \dots, ed_k\} \\ \mathcal{E} &= \{mE(e_1), \dots, mE(e_j)\} \triangleleft \text{source}G \cup \{ed_1 \mapsto t_1, \dots, ed_k \mapsto t_k\} & \mathcal{F} &= \{mE(e_1), \dots, mE(e_j)\} \triangleleft \text{target}G \cup \{ed_1 \mapsto t_1, \dots, ed_k \mapsto t_k\} \end{aligned}$$

A

$$\begin{array}{c} \text{NewPP RULES} \quad 11,15 \quad \text{NewPP RULES} \quad 11,15 \\ \frac{H \vdash y \in (\text{edge}G \setminus C)}{H \vdash \top} \quad 2 \quad \frac{H \vdash (((\mathcal{E})(y) = x \wedge \neg((\mathcal{F})(y) = x) \vee (((\mathcal{F})(y) = x \wedge \neg((\mathcal{E})(y) = x) \wedge \neg((\mathcal{E})(y) = x))) \vee (((\mathcal{F})(y) = x) \wedge \neg((\mathcal{E})(y) = x))) \vee (((\mathcal{F})(y) = x) \wedge \neg((\mathcal{E})(y) = x)))}{H \vdash \exists y . y \in (\text{edge}G \setminus C) \cup \mathcal{D} \wedge (((\mathcal{E})(y) = x \wedge \neg((\mathcal{F})(y) = x) \vee (((\mathcal{F})(y) = x) \wedge \neg((\mathcal{E})(y) = x))) \vee (((\mathcal{F})(y) = x) \wedge \neg((\mathcal{E})(y) = x)))} \quad 28, 71,72 \\ \\ \text{NewPP RULES} \quad 11,15 \quad \text{NewPP RULES} \quad 11,15 \\ \frac{H \vdash x \in \text{vert}G}{H \vdash x \in \text{vert}G \wedge tG_V(x) = (tG_V \cup B)(x)} \quad 28 \quad \frac{H \vdash x \in \text{vert}G \wedge tG_V(x) = (tG_V \cup B)(x)}{H \vdash \exists y . y \in (\text{edge}G \setminus C) \cup \mathcal{D} \wedge (((\mathcal{E})(y) = x \wedge \neg((\mathcal{F})(y) = x) \vee (((\mathcal{F})(y) = x) \wedge \neg((\mathcal{E})(y) = x))) \vee (((\mathcal{F})(y) = x) \wedge \neg((\mathcal{E})(y) = x)))} \quad 70, 75,69 \\ \\ \text{NewPP RULES} \quad 11,15 \quad \text{NewPP RULES} \quad 11,15 \\ \frac{H \vdash x \in \text{vert}G \wedge tG_V(x) = (tG_V \cup B)(x)}{H \vdash \exists y . y \in (\text{edge}G \setminus C) \cup \mathcal{D} \wedge (((\mathcal{E})(y) = x \wedge \neg((\mathcal{F})(y) = x) \vee (((\mathcal{F})(y) = x) \wedge \neg((\mathcal{E})(y) = x))) \vee (((\mathcal{F})(y) = x) \wedge \neg((\mathcal{E})(y) = x)))} \quad 28 \quad \frac{H \vdash \exists y . y \in (\text{edge}G \setminus C) \cup \mathcal{D} \wedge (((\mathcal{E})(y) = x \wedge \neg((\mathcal{F})(y) = x) \vee (((\mathcal{F})(y) = x) \wedge \neg((\mathcal{E})(y) = x))) \vee (((\mathcal{F})(y) = x) \wedge \neg((\mathcal{E})(y) = x)))}{H \vdash \forall x . x \in \text{vert}G \cup \mathcal{A} \wedge (tG_V \cup B)(x) = t \Rightarrow (\exists y . y \in (\text{edge}G \setminus C) \cup \mathcal{D} \wedge (((\mathcal{E})(y) = x \wedge \neg((\mathcal{F})(y) = x) \vee (((\mathcal{F})(y) = x) \wedge \neg((\mathcal{E})(y) = x))) \vee (((\mathcal{F})(y) = x) \wedge \neg((\mathcal{E})(y) = x)))} \quad 61 \end{array}$$

Figure 26. Proof Tree for rule_i/propNotIsoVT/INV - Rule does not involve vertices of type t

Proof Tree Description: The goal to be demonstrated is $H \vdash \forall x \cdot x \in (\text{vert}G \cup \{v_1, \dots, v_l\}) \wedge (tG_V \cup \{v_1 \mapsto t_1, \dots, v_l \mapsto t_l\})(x) = t \Rightarrow (\exists y \cdot y \in (\text{edge}G \setminus \{mE(e_1), \dots, mE(e_j)\}) \cup \{ed_1, \dots, ed_k\} \wedge (((\{mE(e_1), \dots, mE(e_j)\} \triangleleft \text{source}G \cup \{ed_1 \mapsto t_1, \dots, ed_k \mapsto t_k\})(y) = x \wedge \neg(\{mE(e_1), \dots, mE(e_j)\} \triangleleft \text{target}G \cup \{ed_1 \mapsto t_1, \dots, ed_k \mapsto t_k\})(y) = x) \vee (((\{mE(e_1), \dots, mE(e_j)\} \triangleleft \text{target}G \cup \{ed_1 \mapsto t_1, \dots, ed_k \mapsto t_k\})(y) = x \wedge \neg(\{mE(e_1), \dots, mE(e_j)\} \triangleleft \text{source}G \cup \{ed_1 \mapsto t_1, \dots, ed_k \mapsto t_k\})(y) = x)))$. First, we must eliminate the universal quantifier in the goal (61). Next, some rules are automatically applied (6, 3, 62 and 75 respectively), generating the $H \vdash \exists y \cdot y \in (\text{edge}G \setminus \{mE(e_1), \dots, mE(e_j)\}) \cup \{ed_1, \dots, ed_k\} \wedge (((\{mE(e_1), \dots, mE(e_j)\} \triangleleft \text{source}G \cup \{ed_1 \mapsto t_1, \dots, ed_k \mapsto t_k\})(y) = x \wedge \neg(\{mE(e_1), \dots, mE(e_j)\} \triangleleft \text{target}G \cup \{ed_1 \mapsto t_1, \dots, ed_k \mapsto t_k\})(y) = x) \vee (((\{mE(e_1), \dots, mE(e_j)\} \triangleleft \text{target}G \cup \{ed_1 \mapsto t_1, \dots, ed_k \mapsto t_k\})(y) = x \wedge \neg(\{mE(e_1), \dots, mE(e_j)\} \triangleleft \text{source}G \cup \{ed_1 \mapsto t_1, \dots, ed_k \mapsto t_k\})(y) = x))$ sub-goal. Then, x must be instantiated in the induction hypothesis (69). The first remaining sub-goal $H \vdash \top$ is automatically discharged (2). Next we must apply Modus Ponens in the $tG_V(x) = t \Rightarrow (\exists y \cdot y \in \text{edge}G \wedge ((\text{source}G(y) = x \wedge \neg \text{target}G(y) = x) \vee (\text{target}G(y) = x \wedge \neg \text{source}G(y) = x)))$ hypothesis, which generates two sub-goals to be discharged: (I) $H \vdash x \in \text{vert}G \wedge tG_V(x) = (tG_V \cup \{v_1 \mapsto t_1, \dots, v_l \mapsto t_l\})(x)$; and (II) $H \vdash \exists y \cdot y \in (\text{edge}G \setminus \{mE(e_1), \dots, mE(e_j)\}) \cup \{ed_1, \dots, ed_k\} \wedge (((\{mE(e_1), \dots, mE(e_j)\} \triangleleft \text{source}G \cup \{ed_1 \mapsto t_1, \dots, ed_k \mapsto t_k\})(y) = x \wedge \neg(\{mE(e_1), \dots, mE(e_j)\} \triangleleft \text{target}G \cup \{ed_1 \mapsto t_1, \dots, ed_k \mapsto t_k\})(y) = x) \vee (((\{mE(e_1), \dots, mE(e_j)\} \triangleleft \text{target}G \cup \{ed_1 \mapsto t_1, \dots, ed_k \mapsto t_k\})(y) = x \wedge \neg(\{mE(e_1), \dots, mE(e_j)\} \triangleleft \text{source}G \cup \{ed_1 \mapsto t_1, \dots, ed_k \mapsto t_k\})(y) = x)))$. In (I), the \wedge goal rule is automatically applied (28), resulting in two sequents: (i) $H \vdash x \in \text{vert}G$ and (ii) $H \vdash x \in \text{vert}G \wedge tG_V(x) = (tG_V \cup \{v_1 \mapsto t_1, \dots, v_l \mapsto t_l\})(x)$, both demonstrated with NewPP with lasso (15). In (II), the \exists hyp rule is automatically applied (71). After, we must instantiate y in the goal, generating $H \vdash \top$, which is automatically discharged (2). Then, the \wedge goal rule (28) is automatically applied, remaining the sub-goals: (i) $H \vdash y \in (\text{edge}G \setminus \{mE(e_1), \dots, mE(e_j)\})$; and (ii) $H \vdash (((\{mE(e_1), \dots, mE(e_j)\} \triangleleft \text{source}G \cup \{ed_1 \mapsto t_1, \dots, ed_k \mapsto t_k\})(y) = x \wedge \neg(\{mE(e_1), \dots, mE(e_j)\} \triangleleft \text{target}G \cup \{ed_1 \mapsto t_1, \dots, ed_k \mapsto t_k\})(y) = x) \vee (((\{mE(e_1), \dots, mE(e_j)\} \triangleleft \text{target}G \cup \{ed_1 \mapsto t_1, \dots, ed_k \mapsto t_k\})(y) = x \wedge \neg(\{mE(e_1), \dots, mE(e_j)\} \triangleleft \text{source}G \cup \{ed_1 \mapsto t_1, \dots, ed_k \mapsto t_k\})(y) = x))$, which, after some automatic selection and deselection of hypothesis, are discharged running NewPP with lasso (15). ■

5.5 `propAllSVertSrcTED` property

The `propAllSVertSrcTED` property, $\forall x \cdot ((x \in \text{vert}G \wedge tG_V(x) = s) \Rightarrow (\exists y \cdot (y \in \text{edge}G \wedge tG_E(y) = t \wedge \text{source}G(y) = x)))$, states that in any reachable graph all vertices of type s is source of an edge of type t .

In order to demonstrate the obligation generated by the initialisation event, labelled with INITIALISATION/propAllSrcTEd/INV, we must follow the next steps:

1. Eliminate the universal quantifier of the goal;
2. Remove the operator \in in the $x \in vertG$ hypothesis;
3. Select the proof by cases in the $x = v_1 \vee \dots \vee x = v_n$ hypothesis, considering $vertG$ initialised with n vertices;
4. For each case, we must apply the following actions:
 - (a) For vertices v_s (of type s), we must instantiate the edge e_j (of type t) that has source in v_s ;
 - (b) For vertices v_i (not of type s) we must run ML;

Figure 27 presents the proof tree after the demonstration.

Proof Tree Description: The sequent to be demonstrated is $\vdash \forall x.x \in vertG \wedge tG_V(x) = s \Rightarrow (\exists y.y \in edgeG \wedge tG_E(y) = t \wedge sourceG(y) = x)$. In order to discharge it, we must eliminate the universal quantifier of the goal (61), and then \Rightarrow goal rule (62) is automatically applied. Next, we must remove the \in operator from the $x \in vertG$ hypothesis (73) and select proof by cases in the $x = v_1 \vee \dots \vee x = v_n$ hypothesis (64). For each vertex of the initial graph, some rules are automatically applied (9, 3, 65 or 77, and 3, respectively), and then one sub-goal of one of the following cases is generated : (I) $tG_V(x) = s, x = v_s \vdash \exists y.y \in edgeG \wedge tG_E(y) = t \wedge sourceG(y) = x$; (II) $tG_V(x) = s, x = v_i \vdash \exists y.y \in edgeG \wedge tG_E(y) = t \wedge sourceG(y) = x$. The sequent (I) represents the goal generated by a vertex of type s . In such case, we must instantiate in the goal the name of the edge e_j of type t that has source in this vertex of type s , generating two new sub-goals that must be proved: (i) $x = v_s \vdash \top$ and (ii) $x = v_s \vdash e_j \in edgeG \wedge tG_E(e_j) = t \wedge sourceG(e_j) = x$, both automatically discharged (by rules 2 and 3). The sequent (II) details the goal generated by a vertex that is not of type s . For discharging it, we must just run ML (26). These last actions must be repeated for each vertex of the initial graph. ■

For rules, proof obligations follow the next format $\forall x.x \in vertG \cup \{v_1, \dots, v_l\} \wedge (tG_V \cup \{v_1 \mapsto t_1, \dots, v_l \mapsto t_l\})(x) = s \Rightarrow (\exists y.y \in (edgeG \setminus \{mE(e_1), \dots, mE(e_j)\}) \cup \{ed_1, \dots, ed_k\} \wedge ((\{mE(e_1), \dots, mE(e_j)\} \triangleleft tG_E) \cup \{ed_1 \mapsto t_1, \dots, ed_k \mapsto t_k\})(y) = t \wedge ((\{mE(e_1), \dots, mE(e_j)\} \triangleleft sourceG) \cup \{ed_1 \mapsto mV(v_1), \dots, ed_k \mapsto v_l\})(y) = x)$, considering that j edges are deleted, k edges and l vertices are created by a rule application. Thus, a rule could match in one of the following situations: it creates and does not preserve a vertex of type s that is source of an edge of type t ; it preserves and does not create a vertex of

$$\begin{array}{c}
 \boxed{B} \\
 \hline
 \text{ML RULES} \\
 \frac{x = v_i, s = s_k \vdash \exists y \cdot y \in \text{edge}G \wedge tG_E(y) = t \wedge \text{source}G(y) = x}{x = v_n, tG_V(v_i) = s \vdash \exists y \cdot y \in \text{edge}G \wedge tG_E(y) = t \wedge \text{source}G(y) = x} \frac{26}{3} \\
 \frac{x = v_n, tG_V(v_i) = s, x = v_i \vdash \exists y \cdot y \in \text{edge}G \wedge tG_E(y) = t \wedge \text{source}G(y) = x}{tG_V(x) = s, x = v_i, \top \vee x = v_n \vdash \exists y \cdot y \in \text{edge}G \wedge tG_E(y) = t \wedge \text{source}G(y) = x} \frac{65}{3} \\
 \frac{tG_V(x) = s, x = v_i, \top \vee x = v_n \vdash \exists y \cdot y \in \text{edge}G \wedge tG_E(y) = t \wedge \text{source}G(y) = x}{tG_V(x) = s, x = v_i \vdash \exists y \cdot y \in \text{edge}G \wedge tG_E(y) = t \wedge \text{source}G(y) = x} \frac{3}{9} \\
 \\
 \boxed{A} \\
 \frac{\text{True}}{x = v_s \vdash \top} \frac{2}{\frac{x = v_s \vdash e_j \in \text{edge}G \wedge tG_E(e_j) = t \wedge \text{source}G(e_j) = x}{x = v_s \vdash \exists y \cdot y \in \text{edge}G \wedge tG_E(y) = t \wedge \text{source}G(y) = x} \frac{3}{78}} \frac{\text{True}}{x = v_s \vdash \top} \frac{2}{3} \\
 \frac{x = v_s, tG_V(v_s) = s \vdash \exists y \cdot y \in \text{edge}G \wedge tG_E(y) = t \wedge \text{source}G(y) = x}{tG_V(x) = s, x = v_s \vdash \exists y \cdot y \in \text{edge}G \wedge tG_E(y) = t \wedge \text{source}G(y) = x} \frac{3}{77} \\
 \frac{tG_V(x) = s, x = v_s, \top \vee x = v_n \vdash \exists y \cdot y \in \text{edge}G \wedge tG_E(y) = t \wedge \text{source}G(y) = x}{tG_V(x) = s, x = v_s \vdash \exists y \cdot y \in \text{edge}G \wedge tG_E(y) = t \wedge \text{source}G(y) = x} \frac{3}{9} \\
 \\
 \frac{\boxed{A} \quad \boxed{B}}{H \vdash \exists y \cdot y \in \text{edge}G \wedge tG_E(y) = t \wedge \text{source}G(y) = x} \frac{73,64}{62} \\
 \frac{\vdash x \in \text{vert}G \wedge tG_V(x) = s \Rightarrow (\exists y \cdot y \in \text{edge}G \wedge tG_E(y) = t \wedge \text{source}G(y) = x)}{\vdash \forall x \cdot x \in \text{vert}G \wedge tG_V(x) = s \Rightarrow (\exists y \cdot (y \in \text{edge}G \wedge tG_E(y) = t \wedge \text{source}G(y) = x))} \frac{61}{62}
 \end{array}$$

Figure 27. Proof Tree INITIALISATION/propAllSVertSrcTED/INV

type s that is source of an edge of type t ; it preserves a vertex of type s , deletes and creates an edge of type t that has such vertex as source vertex; it does not involves vertices of type s and edges of type t . The case that a rule preserves or creates vertices of type s and deletes edges of type t is not treated in this work.

For all the other cases, the tactic to be applied is the same. If the rule does not delete edges, the obligation is discharged just running NewPP with lasso. If the rule deletes edges, all different from type t , the demonstration is concluded with the next actions:

1. Eliminate the universal quantifier of the goal;
2. For each deleted edge e_i , follow the next steps:
 - (a) Select from the list of hypothesis the guard $tLi_E(e_i) = tG_E(mE(e_i))$;
 - (b) Run NewPP with lasso.
3. Run NewPP with lasso.

$$\begin{array}{c}
 \mathcal{A} = \{v_1, \dots, v_l\} \\
 \mathcal{C} = \{mE(e_1), \dots, mE(e_j)\} \\
 \mathcal{E} = \{mE(e_1), \dots, mE(e_j)\} \triangleleft \text{sourceG} \cup \{ed_1 \mapsto mV(v_1), \dots, ed_k \mapsto v_l\} \\
 \mathcal{G} = tLi_E \in \text{edgeLi} \mapsto \text{edgeT} \\
 \mathcal{B} = \{v_1 \mapsto t_1, \dots, v_l \mapsto t_l\} \\
 \mathcal{D} = \{ed_1, \dots, ed_k\} \\
 \mathcal{F} = \{mE(e_1), \dots, mE(e_j)\} \triangleleft tG_E \cup \{ed_1 \mapsto t_1, \dots, ed_k \mapsto t_k\} \\
 \mathcal{H} = mE \in \text{edgeLi} \mapsto \mathbb{Z}
 \end{array}$$

$$\boxed{\text{C}} \quad \frac{}{H \vdash \exists y. y \in (\text{edgeG} \setminus \mathcal{C}) \cup \mathcal{D} \wedge (\mathcal{F})(y) = t \wedge (\mathcal{E})(y) = x} \text{NewPP RULES} \quad 11.15$$

$$\boxed{\text{B}} \quad \frac{\frac{\text{True}}{H \vdash \top} \quad 2 \quad \frac{\text{True}}{H \vdash \overline{\top}} \quad 2 \quad \frac{\text{True}}{H \vdash e_j \in \text{edgeLi}} \quad 6 \quad \frac{\text{True}}{H \vdash e_j \in \text{edgeLi}} \quad 6 \quad \frac{\text{True}}{H \vdash mE(e_j) \in \text{edgeG}} \quad 25}{\frac{\text{True}}{H \vdash e_j \in \text{dom}(tLi_E)} \quad 20 \quad \frac{\text{True}}{H \vdash e_j \in \text{dom}(mE)} \quad 20 \quad \frac{\text{True}}{H \vdash mE(e_j) \in \text{dom}(tG_E)} \quad 24,24,20} \quad 28}{\frac{\text{True}}{H \vdash e_j \in \text{dom}(tLi_E) \wedge e_j \in \text{dom}(mE) \wedge mE(e_j) \in \text{dom}(tG_E)} \quad 3} \quad 9} \quad \text{NewPP RULES} \quad 11.15$$

$$\boxed{\text{C}} \quad \frac{}{H \vdash e_j \in \text{dom}(tLi_E) \wedge \mathcal{G} \wedge e_j \in \text{dom}(mE) \wedge \mathcal{H} \wedge mE(e_j) \in \text{dom}(tG_E) \wedge t \wedge mE(e_j) \in \text{dom}(tG_E) \wedge t \wedge (\mathcal{F})(y) = t \wedge (\mathcal{E})(y) = x} \quad 81$$

$$\boxed{\text{A}} \quad \frac{\frac{\text{True}}{H \vdash \top} \quad 2 \quad \frac{\text{True}}{H \vdash \overline{\top}} \quad 2 \quad \frac{\text{True}}{H \vdash e_1 \in \text{edgeLi}} \quad 6 \quad \frac{\text{True}}{H \vdash e_1 \in \text{edgeLi}} \quad 6 \quad \frac{\text{True}}{H \vdash mE(e_1) \in \text{edgeG}} \quad 25}{\frac{\text{True}}{H \vdash e_1 \in \text{dom}(tLi_E)} \quad 20 \quad \frac{\text{True}}{H \vdash e_1 \in \text{dom}(mE)} \quad 20 \quad \frac{\text{True}}{H \vdash mE(e_1) \in \text{dom}(tG_E)} \quad 80,20} \quad 29}{\frac{\text{True}}{H \vdash e_1 \in \text{dom}(tLi_E) \wedge \mathcal{G} \wedge e_1 \in \text{dom}(mE) \wedge \mathcal{H} \wedge mE(e_1) \in \text{dom}(tG_E) \wedge t \wedge mE(e_1) \in \text{dom}(tG_E) \wedge t \wedge (\mathcal{F})(y) = t \wedge (\mathcal{E})(y) = x} \quad 28} \quad \text{NewPP RULES} \quad 11.15$$

$$\boxed{\text{A}} \quad \frac{\frac{\text{True}}{H \vdash \top} \quad 2 \quad \frac{\text{True}}{H \vdash \overline{\top}} \quad 2 \quad \frac{\text{True}}{H \vdash e_1 \in \text{edgeLi}} \quad 6 \quad \frac{\text{True}}{H \vdash e_1 \in \text{edgeLi}} \quad 6 \quad \frac{\text{True}}{H \vdash mE(e_1) \in \text{edgeG}} \quad 25}{\frac{\text{True}}{H \vdash e_1 \in \text{dom}(tLi_E)} \quad 20 \quad \frac{\text{True}}{H \vdash e_1 \in \text{dom}(mE)} \quad 20 \quad \frac{\text{True}}{H \vdash mE(e_1) \in \text{dom}(tG_E)} \quad 80,20} \quad 29}{\frac{\text{True}}{H \vdash e_1 \in \text{dom}(tLi_E) \wedge \mathcal{G} \wedge e_1 \in \text{dom}(mE) \wedge \mathcal{H} \wedge mE(e_1) \in \text{dom}(tG_E) \wedge t \wedge mE(e_1) \in \text{dom}(tG_E) \wedge t \wedge (\mathcal{F})(y) = t \wedge (\mathcal{E})(y) = x} \quad 28} \quad \text{NewPP RULES} \quad 11.15$$

$$\boxed{\text{B}} \quad \frac{\frac{\text{True}}{H \vdash \top} \quad 2 \quad \frac{\text{True}}{H \vdash \overline{\top}} \quad 2 \quad \frac{\text{True}}{H \vdash e_1 \in \text{edgeLi}} \quad 6 \quad \frac{\text{True}}{H \vdash e_1 \in \text{edgeLi}} \quad 6 \quad \frac{\text{True}}{H \vdash mE(e_1) \in \text{edgeG}} \quad 25}{\frac{\text{True}}{H \vdash e_1 \in \text{dom}(tLi_E)} \quad 20 \quad \frac{\text{True}}{H \vdash e_1 \in \text{dom}(mE)} \quad 20 \quad \frac{\text{True}}{H \vdash mE(e_1) \in \text{dom}(tG_E)} \quad 80,20} \quad 29}{\frac{\text{True}}{H \vdash e_1 \in \text{dom}(tLi_E) \wedge \mathcal{G} \wedge e_1 \in \text{dom}(mE) \wedge \mathcal{H} \wedge mE(e_1) \in \text{dom}(tG_E) \wedge t \wedge mE(e_1) \in \text{dom}(tG_E) \wedge t \wedge (\mathcal{F})(y) = t \wedge (\mathcal{E})(y) = x} \quad 28} \quad \text{NewPP RULES} \quad 11.15$$

$$\frac{}{H \vdash x \in \text{vertG} \cup \mathcal{A} \wedge (tG_V \cup \mathcal{B})(x) = s \Rightarrow (\exists y. y \in (\text{edgeG} \setminus \mathcal{C}) \cup \mathcal{D} \wedge (\mathcal{F})(y) = t \wedge (\mathcal{E})(y) = x)} \quad 6,3,62} \quad 61} \quad \frac{}{H \vdash \forall x. x \in \text{vertG} \cup \mathcal{A} \wedge (tG_V \cup \mathcal{B})(x) = s \Rightarrow (\exists y. y \in (\text{edgeG} \setminus \mathcal{C}) \cup \mathcal{D} \wedge (\mathcal{F})(y) = t \wedge (\mathcal{E})(y) = x)} \quad 61}$$

Figure 28. Proof Tree for rule_i/propAllISVertSrcTEd/INV

The generated proof tree is presented in Figure 28.

Proof Tree Description: The initial goal to be demonstrated is $H \vdash \forall x \cdot x \in \text{vert}G \cup \{v_1, \dots, v_l\} \wedge (tG_V \cup \{v_1 \mapsto t_1, \dots, v_l \mapsto t_l\})(x) = s \Rightarrow (\exists y \cdot y \in (\text{edge}G \setminus \{ed_1, \dots, ed_k\}) \cup \{ed_1, \dots, ed_k\} \wedge (\{mE(e_1), \dots, mE(e_j)\} \triangleleft tG_E \cup \{ed_1 \mapsto t_1, \dots, ed_k \mapsto t_k\})(y) = t \wedge (\{mE(e_1), \dots, mE(e_j)\} \triangleleft \text{source}G \cup \{ed_1 \mapsto mV(v_1), \dots, ed_k \mapsto v_l\})(y) = x)$. With the aim of demonstrating it, first we must eliminate the universal quantifier of the goal (61), and then, some rules are automatically applied (6, 3 and 62, respectively), generating the sequent $H \vdash \exists y \cdot y \in (\text{edge}G \setminus \{mE(e_1), \dots, mE(e_j)\}) \cup \{ed_1, \dots, ed_k\} \wedge (\{mE(e_1), \dots, mE(e_j)\} \triangleleft tG_E \cup \{ed_1 \mapsto t_1, \dots, ed_k \mapsto t_k\})(y) = t \wedge (\{mE(e_1), \dots, mE(e_j)\} \triangleleft \text{source}G \cup \{ed_1 \mapsto mV(v_1), \dots, ed_k \mapsto v_l\})(y) = x$. Then, it is necessary to guarantee that the deleted edges are not of type t . Considering that e_i is a deleted edge, we must apply the following steps: add as hypothesis the guard $tLi_E(e_i) = tG_E(mE(e_i))$, and after run NewPP with lasso. This must be repeated for each deleted edge (in the proof tree is being considered that j edges are deleted, from e_1 to e_j , omitting the intermediary ones). Finally, a last sequent is generated $H \vdash \exists y \cdot y \in (\text{edge}G \setminus \{mE(e_1), \dots, mE(e_j)\}) \cup \{ed_1, \dots, ed_k\} \wedge (\{mE(e_1), \dots, mE(e_j)\} \triangleleft tG_E \cup \{ed_1 \mapsto t_1, \dots, ed_k \mapsto t_k\})(y) = t \wedge (\{mE(e_1), \dots, mE(e_j)\} \triangleleft \text{source}G \cup \{ed_1 \mapsto mV(v_1), \dots, ed_k \mapsto v_l\})(y) = x$, which is proved running NewPP with lasso (15). ■

6 Final Remarks

Theorem proving as verification technique requires user interaction during the development of the proofs, but on the other hand, it allows the verification of systems with huge or infinite state spaces. The use of theorem provers is not simple, requires knowledge of the tool and the system to be checked. This work constitutes one more step towards the reduction of expertise required from the user when adopting such an approach for graph grammars specifications. Particularly, we proposed proof tactics (strategies) for discharging a specific set of invariants. The proposed tactics are a roadmap to demonstrate structural properties of graphs, which usually are of interest for systems specified in GG. These tactics are presented by proof trees, which describe in detail the proof process. The application of tactics by non-specialists is quite simple, it is enough to follow the steps in the use trees. Besides these tactics can be directly applied to instances of the considered properties, its use may show sub-tactics that can be reused in sub-properties, which may appear in other verifications. This work not only extends the previously proposed set of tactics [15], but also includes properties that involve new logic operators.

Currently, various approaches allow the verification of graph grammars models through model checking [17, 18, 6]. Although model checking is an established way of verification, it

has as disadvantage the need to build the complete state space (or at least large portions of it), which may lead to the state explosion problem. This problem may occur even if the system is finite state, when the notion of state is complex. One of the strengths of graph grammars is their ability to model complex states (as graphs), and therefore it is to expect that even small specifications may lead to a large number of complex reachable graphs, restricting the possibilities to use model checking techniques. There are few works [21, 22, 23] that are using theorem provers or proof assistants for graph grammars, but as far we know, none of them are actually performing verification.

Strategies for discharging other kind of properties are under development, particularly, tactics for all patterns proposed in [9]. Concerning the kind of properties to be proven, in this work we only considered invariants, but it would also be possible to use the proposed translation to prove properties using variants (like termination, for example). This is also supported by event-B and corresponding tools.

Acknowledgement

The authors gratefully acknowledge financial support received from FAPERGS (PRO-NEM 11/2016-2).

References

- [1] J. R. Abrial. *The B-Book: Assigning Programs to Meanings*. Cambridge University Press, 2005.
- [2] J.-R. Abrial. *Modeling in Event-B: System and Software Engineering*. Cambridge University Press, New York, NY, USA, 1st edition, 2010.
- [3] J.-R. Abrial et al. Rodin: An open toolset for modelling and reasoning in Event-B. *International Journal on Software Tools for Technology Transfer (STTT)*, 12(6):447–466, April 2010.
- [4] J.-R. Abrial and S. Hallerstede. Refinement, decomposition, and instantiation of discrete models: Application to event-b. *Fundam. Inform.*, 77(1-2):1–28, 2007.
- [5] R.-J. Back and K. Sere. Stepwise Refinement of Action Systems. In J. L. A. van de Snepscheut, editor, *Proceedings of the International Conference on Mathematics of Program Construction, 375th Anniversary of the Groningen University*, pages 115–138, London, UK, 1989. Springer.
- [6] P. Baldan, A. Corradini, and B. König. A framework for the verification of infinite-state graph transformation systems. *Inf. and Comp.*, 206:869–907, 2008.
- [7] L. Baresi and P. Spoletini. On the use of Alloy to analyze graph transformation systems. In A. Corradini, H. Ehrig, U. Montanari, L. Ribeiro, and G. Rozenberg, editors, *ICGT*, volume 4178 of *LNCS*, pages 306–320. Springer, 2006.
- [8] S. A. da Costa. *Relational approach of graph grammars*. PhD thesis, UFRGS, Brazil, 2010.
- [9] S. A. da Costa Cavalheiro, L. Foss, and L. Ribeiro. Specification patterns for properties over reachable states of graph grammars. In *Proceedings of the 15th Brazilian conference on Formal Methods: foundations and applications*, SBMF’12, pages 83–98, Berlin, 2012. Springer.

- [10] A. M. de Mello, L. C. L. Junior, L. Foss, and S. A. da Costa Cavalheiro. Graph grammars: A comparison between verification methods. *WEIT*, pages 88–94, 2011.
- [11] DEPLOY. Event-B and the Rodin platform, Aug 2013. <http://www.event-b.org/> (last acc. Sept 2014).
- [12] E. Dijkstra. *A Discipline of Programming*. Prentice-Hall International, 1976.
- [13] H. Ehrig et al. Algebraic approaches to graph transformation. *Handbook of graph grammars and computing by graph transformation: volume I. foundations*, pages 247–312, 1997.
- [14] L. Lemos, S. Da Costa Cavalheiro, and L. Foss. Towards the use and description of proof tactics for theorem proving graph grammars through rodin. In *Theoretical Computer Science (WEIT), 2013 2nd Workshop-School on*, pages 51–58, Oct 2013.
- [15] L. C. Lemos Junior, S. A. da Costa Cavalheiro, and L. Foss. Theorem proving graph grammars: Strategies for discharging proof obligations. In J. Iyoda and L. de Moura, editors, *Formal Methods: Foundations and Applications*, volume 8195 of *Lecture Notes in Computer Science*, pages 147–162. Springer Berlin Heidelberg, 2013.
- [16] T. Nipkow, M. Wenzel, and L. C. Paulson. *Isabelle/HOL: A Proof Assistant for Higher-order Logic*. Springer-Verlag, Berlin, Heidelberg, 2002.
- [17] A. Rensink. The GROOVE simulator: A tool for state space generation. In J. Pfalz, M. Nagl, and B. Böhlen, editors, *Applications of Graph Transformations with Industrial Relevance (AGTIVE)*, volume 3062 of *LNCS*, pages 479–485. Springer, 2004.
- [18] L. Ribeiro, F. L. Dotti, and R. Bardohl. A formal framework for the development of concurrent object-based systems. In H.-J. Kreowski, U. Montanari, F. Orejas, G. Rozenberg, and G. Taentzer, editors, *Formal Methods in Software and Systems Modeling*, volume 3393 of *Lecture Notes in Computer Science*, pages 385–401. Springer, 2005.
- [19] L. Ribeiro, F. L. Dotti, S. A. da Costa, and F. C. Dillenburg. Towards theorem proving graph grammars using Event-B. *ECEASST*, 30, 2010.
- [20] J. A. Robinson and A. Voronkov, editors. *Handbook of Automated Reasoning (in 2 volumes)*. Elsevier and MIT Press, 2001.
- [21] M. Strecker. Modeling and verifying graph transformations in proof assistants. *Electronic Notes in Theoretical Computer Science*, 203(1):135–148, 2008.
- [22] M. Strecker. Locality in reasoning about graph transformations. In *Proceedings of the 4th International Conference on Applications of Graph Transformations with Industrial Relevance, AGTIVE'11*, pages 169–181, Berlin, Heidelberg, 2012. Springer-Verlag.
- [23] M. Strecker. Interactive and automated proofs for graph transformations. *Mathematical Structures in Computer Science (MSCS)*, page 31 pages, 2014. accepted to appear in special issue on Term and Graph Rewriting.
- [24] H. N. Tran and C. Percebois. Towards a rule-level verification framework for property-preserving graph transformations. In G. Antoniol, A. Bertolino, and Y. Labiche, editors, *2012 IEEE Fifth International Conference on Software Testing, Verification and Validation, Montreal, QC, Canada, April 17-21, 2012*, pages 946–953. IEEE, 2012.