UNIVERSIDADE FEDERAL DE PELOTAS
Centro de Desenvolvimento Tecnológico
Programa de Pós-Graduação em Computação

Dissertação

PROPOSTA DE TÁTICAS PARA PROVA DE TEOREMAS DE GRAMÁTICA DE GRAFOS

Luiz Carlos Lemos Junior

Pelotas, 2014

Luiz Carlos Lemos Junior

PROPOSTA DE TÁTICAS PARA PROVA DE TEOREMAS DE GRAMÁTICA DE GRAFOS

Dissertação apresentada ao Programa de Pós-Graduação em Computação da Universidade Federal de Pelotas, como requisito parcial à obtenção do título de Mestre em Ciência da Computação

Orientadora: Profª. Drª. Simone André da Costa Cavalheiro
Co-orientadora: Profª. Drª. Luciana Foss

Pelotas, 2014

Universidade Federal de Pelotas / Sistema de Bibliotecas
Catalogação na Publicação

L555p Lemos Junior, Luiz Carlos

Proposta de táticas para prova de teoremas de gramática de grafos / Luiz Carlos Lemos Junior; Simone André da Costa Cavalheiro, orientadora; Luciana Foss, co-orientadora. – Pelotas, 2014.

166 f. : il.

Dissertação (Mestrado) – Programa de Pós-Graduação em Computação, Centro de Desenvolvimento Tecnológico Universidade Federal de Pelotas, 2014.

1. Métodos Formais. 2. Gramática de Grafos. 3. Prova de Teoremas. I. Cavalheiro, Simone André da Costa, orient. II. Foss, Luciana, coorient. III. Título.

CDD: 005

Elaborada por Aline Herbstrith Batista – CRB-10/1737

Dedico este trabalho a Profª. Simone André da Costa Cavalheiro, minha orientadora.

AGRADECIMENTOS

A meus pais Luiz Carlos Lemos (in memoriam) e Ana Girlei de Moura Lemos, por darem-me a vida, pelo incentivo e por proporcionarem-me o acesso a educação.

A minha Orientadora Simone André da Costa Cavalheiro, por guiar-me nesta trajetória tão importante. Pois sem seu apoio este trabalho não teria se concretizado. Por suas incansáveis explicações, revisões e incentivo nos momentos difíceis. Por ajudar-me a produzir artigos científicos, publicados em eventos relevantes nessa área. Por os mesmos motivos, agradecer a minha Co-orientadora Luciana Foss. Com vocês aprendi muitas coisas que certamente levarei por toda minha caminhada como docente e pessoa.

A minha companheira Caroline Islabão, pelo apoio e incentivo, principalmente no período final desta caminhada. Meu projeto inicial para inscrição ao Mestrado não seria o mesmo sem suas revisões e dicas. Pelas releituras de artigos escritos para eventos entre outras tantas coisas. Com carinho agradeço a "tia Biba", professora de Português, por revisar minha dissertação e aos demais familiares pelas palavras de motivação e incentivo.

A Profª. Paula Yamim e a Assistente Social Carmen Nascimento, autoras das cartas de recomendação de ingresso ao Mestrado, com as quais aprendi muito no período em que trabalhamos juntos, aprendizados que levarei por toda vida.

Ao Professor Marcelo Porto, pelos relatos de experiências, pelas dicas nas disciplinas obrigatórias fora de minha linha de pesquisa e pela disponibilidade para ajuda.

Aos Professores Coordenadores do PPGC-UFPel, Prof. Gerson Cavalheiro e o Prof. Felipe Marques, pela coragem de encarar a tarefa de coordenação. Sempre disponíveis e atuantes. Aos demais membros do Programa de Pós-Graduação em Computação desta Universidade.

Ao colega André Mello, pela troca de experiências e parceria, é possível dizer que aprendemos juntos e aos demais colegas da turma de 2011, valeu pelo companheirismo e união.

Ao pessoal da secretária, principalmente a secretária Martha Maia, pela disponibilidade e agilidade no atendimento. A "galera" do Lab. 6 da UFPel, por sempre manterem um clima agradável no laboratório.

Aos diretores das escolas em que trabalho ou trabalhei, principalmente o Professor Fernando Burkert, pela flexibilidade e compreensão em momentos difíceis, uma vez que a prefeitura de Pelotas, representada pela SMED não incentiva a capacitação de seus profissionais. Aos demais colegas e amigos que atuam na área da educação de Pelotas.

Aos colegas do Curso de Matemática a Distância da UFPel - CLMD, principalmente aos Profs. Neide Angelo e Patrícia Fantinel, pelas palavras de incentivo e relatos de

experiências. Aos demais colegas pelo companheirismo, certamente aprendo muito com vocês.

Ao povo brasileiro que paga seus impostos, os quais proporcionam o funcionamento da universidade. Aos gestores, que às vezes deixam de lado o convívio com suas famílias para dar funcionamento à instituição. A todos os envolvidos na criação do curso de Mestrado em Ciência da Computação da UFPel.

E a clássica, para todos aqueles que de uma forma ou de outra contribuíram nesta etapa importante na minha vida.

Não há vida sem correção, sem retificação.
— PAULO FREIRE

RESUMO

LEMOS JUNIOR, Luiz Carlos. **PROPOSTA DE TÁTICAS PARA PROVA DE TEOREMAS DE GRAMÁTICA DE GRAFOS**. 2014. 166 f. Dissertação (Mestrado em Ciência da Computação) – Programa de Pós-Graduação em Computação, Universidade Federal de Pelotas, Pelotas, 2014.

A computação é empregada diariamente no mundo moderno, através de sistemas que estão a cada dia mais complexos. A ocorrência de erros no funcionamento desses programas vem aumentando prejudicando as pessoas. Portanto, é necessário utilizar meios que aumentem o nível de confiança dos sistemas. Uma forma de desenvolver sistemas mais confiáveis é utilizando métodos formais. Dentro destas técnicas encontra-se a especificação e a verificação formal. Uma linguagem atraente para especificação formal é a gramática de grafos (GG), pois é visual e baseada em um mecanismo simples de reescrita de regras. Já a verificação pode ser realizada pela técnica de prova de teoremas, que permite a prova de propriedades para sistemas com espaço de estados grande ou até mesmo infinito. A abordagem para o uso da técnica de prova de teoremas para a verificação de sistemas descritos em GG já é conhecida. Nessa abordagem GG foi traduzida para a linguagem Event-B, o que possibilitou utilizar a plataforma Rodin para a prova de propriedades. No momento da verificação são geradas obrigações de prova, as quais devem ser demonstradas. Algumas delas são descarregadas de forma automática e outras necessitam da intervenção do usuário com certa experiência. Essa intervenção, na maioria das vezes, não é trivial e acaba por restringir o uso da técnica. Este trabalho visa otimizar o processo de verificação, propondo táticas de prova para demonstrar as obrigações interativas, as quais auxiliam o desenvolvedor no momento da verificação. Primeiramente são apresentadas táticas para os padrões de obrigações de prova de um sistema de GG. Logo após, táticas para demonstrar obrigações de propriedades específicas são propostas. A apresentação das táticas é feita através da descrição dos passos necessários para as demonstrações das propriedades consideradas, descrevendo as árvores de prova e propondo árvores de uso. Em qualquer uma das formas é possível realizar a prova da obrigação correspondente.

Palavras-chave: Métodos Formais, Gramática de Grafos, Prova de Teoremas.

ABSTRACT

LEMOS JUNIOR, Luiz Carlos. **PROPOSAL OF TACTICS FOR THEOREM PROVING OF GRAPH GRAMMARS**. 2014. 166 f. Dissertação (Mestrado em Ciência da Computação) – Programa de Pós-Graduação em Computação, Universidade Federal de Pelotas, Pelotas, 2014.

Computation is used daily in the modern world through systems that are becoming more complex each day. The occurrence of errors in these programs execution is increasing and causing many kinds of losses to the people. So, it is a need to use some ways to increase the reliability of those systems. One way to develop more reliable systems is by the use of formal methods. Inside these techniques there are the formal specification and verification. An attractive language for formal specification is the graph grammars (GG), since it is visual and based in a simple mechanism of rewriting rules. In the other side, the verification can be done by the technique of theorem proving, that allows the proof of properties for systems with wide space of states, or even infinite states. There is an approach that allows the use of the theorem proving technique for systems described by GG. In this proposal, GG was translated to Event-B language, which makes possible the utilization of Rodin platform to the proof of properties. At the verification step some statement obligations are generated and which must be demonstrated. Some of them are unloaded automatically and some others need a experienced user intervention. This intervention, in most of cases, is not trivial and restricts the usage of the technique. This work aims to optimize the verification process, proposing some proof tactics in order to demonstrate interactive obligations, which angle to help the developer in the moment of verification. Firstly, strategies to a GG system proof obligations patterns are presented. Just after, tactics to demonstrate specific properties obligations are suggested. The presentation of the tactics is done by describing the main steps to the demonstrations of the considered properties, describing the proof tree and proposing usage trees. For any of those forms it is possible to present the corresponding proof of the obligation.

Keywords: Formal Methods, Graph Grammars, Theorem Proving.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

GG	Gramática de Grafos
GTS	Graph Transformation Systems
IDE	Ambiente Integrado de Desenvolvimento
ML	Mono-Lemma Prover
MP	Modus Ponens
PP	Predicate Prover
SMT	Satisfiability Modulo Theories

LISTA DE SÍMBOLOS

$\in$	Pertence
$\rightarrow$	Função Total
$\pfun$	Função Parcial
$\mapsto$	Mapeamento
$\rightarrowtail$	Injeção Total
$\pinj$	Injeção Parcial
$\Rightarrow$	Implicação
$\rhd$	Restrição na Imagem
$:=$	Receber/Atribuir Valor
$\ndres$	Subtração no Domínio
$\setminus$	Subtração no Conjunto
$\cup$	União
$\top$	Verdade
$\vdash$	Martelo Sintático
$\subseteq$	Subconjunto
$\vee$	Ou Lógico
$\wedge$	E Lógico
$\varnothing$	Conjunto Vazio
$\mathbb{N}$	Conjunto dos Número Naturais
$\mathbb{Z}$	Conjunto dos Número Inteiros
$\forall$	Quantificador Universal
$\exists$	Quantificador Existencial
$\neg$	Negação

SUMÁRIO

1 INTRODUÇÃO

A computação está presente em todos os setores da sociedade. Diariamente bens ou serviços ligados à computação são utilizados em diversas atividades humanas, reforçando a importância da área tecnológica no mundo moderno. Sistemas de *software* e *hardware* são aplicados na produção de equipamentos, os quais são utilizados tanto no setor produtivo quanto no dia-a-dia das pessoas. O mau funcionamento destes sistemas acaba prejudicando os usuários. Por exemplo, um erro comum é a reinicialização inesperada de um telefone celular. Por outro lado, erros gerados no *software* que controla o voo de um avião podem causar danos irreparáveis. De qualquer forma, quem utiliza esses equipamentos espera um funcionamento correto e confiável. Os sistemas em geral estão a cada dia mais complexos, com diferentes tipos de funcionalidades e interações. Por isso, é necessário buscar meios que aumentem o nível de confiabilidade dos mesmos, desde o seu projeto.

Uma forma de aumentar a confiabilidade é utilizando métodos formais, que são técnicas baseadas em matemática que oferecem formas rigorosas e eficazes para modelar, projetar e analisar sistemas de *software* e *hardware*. Dentro destas técnicas, encontra-se a especificação e a verificação formal. A especificação formal consiste no uso de um modelo matemático, com sintaxe e semântica bem definidos, para a descrição do sistema. A adoção de uma linguagem de especificação formal permite reduzir a ocorrência de ambiguidades, especificando de forma precisa qual é a funcionalidade de cada componente do sistema (RIBEIRO, 2000). É possível dizer, que a linguagem formal descreve o que o sistema deve fazer. Já a verificação formal consiste na prova matemática da conformidade da especificação de um sistema com determinadas propriedades. Com essa técnica é possível assegurar propriedades do sistema.

Existem diversas linguagens de especificação, entre elas destaca-se gramática de grafos (GG) (EHRIG et al., 1997). Neste formalismo os estados de um sistema são representados por grafos e o seu comportamento é descrito por aplicações de regras. GG é ideal para sistemas em que os estados possuem descrições complexas, envolvendo diversos elementos e diferentes relações entre eles. Essa linguagem também

se enquadra para sistemas com comportamento orientado a dados, onde eventos são aplicados por configurações particulares do estado. Gramática de grafos é utilizada em diversas aplicações, como em sistemas concorrentes e distribuídos, linguagens visuais, reescrita de termos, programação em lógica, reconhecimento e geração de imagens, biologia, entre outros (RIBEIRO, 2000).

Uma GG é composta por: um grafo tipo, representando os vértices e arcos permitidos no sistema; um grafo inicial, especificando o estado inicial do sistema; e um conjunto de regras, que modela as transições que podem ocorrer no sistema. Uma das vantagens de utilizar GG é que essa linguagem é visual, o que facilita o entendimento da descrição de um sistema até mesmo para não especialistas. Uma regra em GG é composta por dois grafos e um morfismo. A aplicação de uma regra pode deletar, criar ou preservar elementos de um grafo-estado. Quando todos os elementos do grafo do lado esquerdo da regra possuírem correspondência (via morfismo) no grafo-estado, a regra pode ser aplicada, gerando um novo estado. Os elementos que estão no lado esquerdo e não possuem imagem no lado direito da regra são removidos. Os elementos que estão no lado esquerdo e são mapeados para o lado direito via morfismo são preservados. Já os elementos do lado direito que não possuem pré-imagem no lado esquerdo são criados no próximo estado do sistema descrito pela gramática. Se as regras não estão em conflito, i.e., se não atualizam a mesma porção do grafo-estado, elas podem ser aplicadas em paralelo, senão a escolha para aplicação é feita de forma não-determinística.

Como técnica de verificação formal aparece a prova de teoremas (ROBINSON; VORONKOV, 2001). Nessa técnica tanto o sistema quanto suas características desejadas são expressas em alguma lógica. Utilizando axiomas e lemas contidos no sistema é possível realizar a prova de propriedades. A prova de teoremas é adequada para sistemas com espaços de estados grande e até mesmo infinito (MELLO et al., 2011), mas em geral requer a intervenção do usuário para dar continuidade em uma demonstração.

Um trabalho previamente proposto (COSTA; RIBEIRO, 2010; RIBEIRO et al., 2010) introduziu uma abordagem lógica e relacional de GG, permitindo a verificação de sistemas distribuídos e assíncronos com espaço de estados grande ou até mesmo infinito, através da técnica de prova de teoremas. O trabalho apresenta uma codificação de grafos e regras de uma GG com relações. As relações que definem uma gramática determinam axiomas que podem ser utilizados no desenvolvimento de provas. Condições lógicas (usando lógica de primeira ordem e teoria dos conjuntos) impostas sobre as relações garantem que elas codificam grafos e morfismos de grafos. A aplicação de uma regra é descrita por um evento que pode ser visto como uma regra de inferência: quando um conjunto de variáveis satisfaz condições de guarda, a aplicação de regra é realizada. Essa abordagem de GG foi traduzida para estruturas

em Event-B, o que tornou possível utilizar os provadores disponíveis para essa linguagem, através da plataforma Rodin (ABRIAL et al., 2010) para a verificação formal de propriedades. As propriedades nessa abordagem especificam características sobre a estrutura do grafo-estado e devem ser declaradas como invariantes, indicando que são propriedades globais que devem ser válidas para todos os estados do sistema.

Ao modelar um sistema em Event-B, o Rodin faz uma verificação sintática (verificação estática) e uma verificação dinâmica. Nessas etapas a ferramenta gera obrigações de provas para garantir que invariantes são preservadas, condições de guarda e ações estão bem definidas, fórmulas têm sentido, entre outros. Essas obrigações objetivam garantir a consistência do sistema. Algumas dessas obrigações são descarregadas automaticamente, enquanto outras exigem intervenção do usuário. As provas são realizadas pelo método da indução matemática, no caso base se demonstra a propriedade para o grafo inicial e, no passo da indução, a propriedade é analisada para cada regra da gramática. O processo de prova é semi-automático, em muitos casos exigindo a interação do usuário. Nesse aspecto aparece a desvantagem dessa abordagem (assim como para prova de teoremas em geral), uma vez que o usuário deve possuir conhecimento do sistema, da linguagem de especificação e da ferramenta para concluir as provas.

1.1 Objetivos do Trabalho

O objetivo geral deste trabalho é propor estratégias de provas, que visam auxiliar o desenvolvedor no uso da técnica de prova de teoremas para a abordagem relacional de gramática de grafos (COSTA, 2010).

São objetivos específicos:

1. Identificar obrigações de provas, geradas pela tradução de cada um dos componentes de uma gramática de grafos em estruturas do Event-B;
2. Descrever as estruturas das árvores de prova, geradas nas demonstrações das obrigações de provas mais usuais;
3. Definir estratégias de provas para propriedades específicas.

Como resultado, espera-se motivar o uso da técnica de prova de teoremas para provar propriedades de sistemas descritos em gramática de grafos.

1.2 Organização do Texto

A apresentação deste trabalho é dividida nos seguintes capítulos, conforme segue:

Capítulo 2: Apresenta as definições básicas de uma GG e mostra como essa linguagem é especificada em Event-B.

Capítulo 3: Descreve o ambiente de prova da plataforma Rodin e detalha quais são as obrigações de prova geradas para um sistema de GG descrito em Event-B. As obrigações aqui consideradas referem-se àquelas que são sempre geradas após a especificação de qualquer sistema descrito em GG na linguagem Event-B.

Capítulo 4: Propõe estratégias de prova para demonstrar obrigações geradas por propriedades específicas de GG.

Capítulo 5: Detalha exemplos de aplicação das táticas propostas.

Capítulo 6: Sumariza as conclusões e propõe trabalhos futuros.

1.3 Trabalhos Relacionados

TRAN; PERCEBOIS (2012) propõe um método para provar que uma transformação de grafos preserva propriedades. O trabalho, que utiliza a abordagem proposta por COSTA (2010), consiste em identificar as condições gerais de uma GG para preservar propriedades do grafo-estado, mais precisamente propriedades estruturais. As noções relevantes de GG são implementadas no assistente de prova Isabelle/HOL (NIPKOW; PAULSON; WENZEL, 2002). Dado um grafo de entrada e um conjunto de regras de transformação de grafos, é utilizada a estratégia de indução matemática para verificar estaticamente se a transformação preserva uma propriedade particular do grafo inicial. No trabalho, os autores isolaram e formalizaram quais são as condições lógicas sobre as regras e o grafo inicial que preservam algumas características do grafo transformado.

No artigo publicado por STRECKER (2008), é apresentado os primeiros passos para a formalização de transformações de grafos em ambientes de provadores de teoremas. Grafos e transformações de grafos são formalizados com um modelo baseado em Teoria dos Conjuntos. A proposta foi desenvolvida com o assistente de prova Isabelle (NIPKOW; PAULSON; WENZEL, 2002), discutindo algumas estratégias iniciais para simplificar obrigações de prova. Nesse trabalho, não foram feitas verificações específicas.

Já no trabalho (STRECKER, 2012) é apresentado um meio de raciocinar localmente sobre propriedades globais em transformação de grafos, mais precisamente em propriedades que envolvam o fecho transitivo das relações dos arcos do grafo. É mostrado em quais condições é possível deixar de analisar todos os nodos de um grafo, para se deter a um conjunto finito de nodos. Um procedimento é apresentado

para transformar o problema reduzido a um problema de satisfatibilidade booleana. Nesse procedimento é possível provar propriedades de transformação de grafos em uma linguagem relacional restrita ao fecho transitivo.

No artigo de YAMAMOTO et al. (2008), os autores buscam propor uma abstração de sistemas de transformação de grafos utilizando a lógica temporal. Com o objetivo de utilizar *abstract model checking* para verificar sistemas com estados infinitos, os autores discutem extensões da lógica temporal que podem torná-la adequada a abstrair sistemas de transformações de grafos. De forma breve, os autores levantam alguns estudos de caso, ferramentas e aspectos de verificação formal, que podem ser considerados na abordagem proposta. Até o momento não se encontrou trabalhos posteriores dos autores que tratam desse tópico.

2 GRAMÁTICA DE GRAFOS

Neste capítulo será apresentada uma revisão das definições básicas de gramática de grafos (GG) (EHRIG et al., 1997) que são utilizadas no trabalho, bem como a especificação de GG na linguagem Event-B (ABRIAL, 2010). A gramática de grafos é composta por um grafo inicial, que define o estado inicial do sistema; um grafo tipo, que apresenta os tipos de vértices e arcos permitidos no sistema; e um conjunto de regras, que descreve as possíveis mudanças que podem ocorrer no sistema. Event-B é baseado no método B clássico (ABRIAL, 1996), utilizado para modelar e analisar sistemas. Entre suas características pode destacar-se: o uso da teoria dos conjuntos e lógica de primeira ordem na modelagem; representação do sistema em diferentes níveis; e a utilização de provas da matemática na verificação formal.

Os identificadores adotados nas definições de GG na Seção 2.1 (prefixados com `inv_`, `grd_`, `axm_` e `act_`) são aqueles utilizados em um modelo Event-B (respectivamente, invariante, condição de guarda, axioma e ação) na Seção 2.2.

2.1 Definições Básicas de Gramática de Grafos

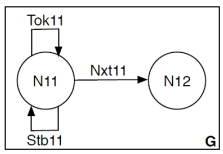
Gramática de grafos é uma linguagem de especificação adequada para representar situações complexas, uma vez que é simples e visual. Um grafo é definido por dois conjuntos e por duas funções. Um morfismo de grafos é definido por duas funções parciais.

Definição 1 ***(Grafo, Morfismo de Grafos):*** *Um grafo é uma tupla* $(vertG, edgeG, sourceG, targetG)$*, onde* $vertG$ *é um conjunto de vértices,* $edgeG$ *é um conjunto de arcos,* $sourceG$*,* $targetG$*:* $edgeG \to vertG$ *são funções totais que definem a origem e destino de cada arco, respectivamente. Dados dois grafos* $G = (vertG, edgeG, sourceG, targetG)$ *e* $H = (vertH, edgeH, sourceH, targetH)$*, um morfismo parcial de grafos* $g : G \nrightarrow H$ *é uma tupla* $g = (g\_V : vert\_G \nrightarrow vert\_H, g\_E : edgeG \nrightarrow edgeH)$*, de modo que* g *comuta com as funções de origem e destino, ou*

seja, as seguintes condições devem ser satisfeitas:

$$\texttt{grd\_srctgt:} \quad \forall e \in dom(g\_E) \cdot g\_V(source\_G(e)) = source\_H(g\_E(e)) \text{ e}$$
$$\forall e \in dom(g\_E) \cdot g\_V(target\_G(e)) = target\_H(g\_E(e))$$

Um morfismo de grafo é chamado total ou injetivo se ambos os seus componentes são funções totais ou injetivas, respectivamente.



$$vertG = \{N11, N12\}$$
$$edgeG = \{Tok11, Stb11, Nxt11\}$$
$$sourceG = \{Tok11 \mapsto N11, Stb11 \mapsto N11, Nxt11 \mapsto N11\}$$
$$targetG = \{Tok11 \mapsto N11, Stb11 \mapsto N11, Nxt11 \mapsto N12\}$$

Figura 1: Grafo G com sua Definição Formal

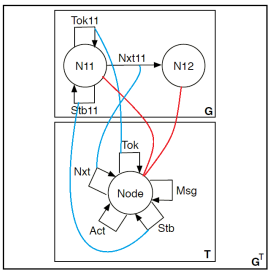
A Figura 1 mostra o grafo G com sua definição formal. As funções $sourceG$ e $targetG$ são totais, ou seja, todo arco está relacionado com um único vértice, tanto para a origem quanto para o destino.

Um grafo tipado é composto por dois grafos e por um morfismo total entre eles, chamado de morfismo de tipagem. Já um morfismo entre grafos tipados sobre o mesmo grafo é um morfismo de grafos, juntamente com as condições que garantem que o mapeamento é realizado entre elementos de mesmo tipo.

Definição 2 ***(Grafo Tipado, Morfismo de Grafos Tipados):*** *Um grafo tipado é uma tupla $G^T = (G, tG, T)$, onde G e T são grafos e $tG = (tG\_V, tG\_E)$ é um morfismo de tipagem de G sobre T, isto é, $tG : G \to T$ é um morfismo total de grafos (*`inv_tG_V` e `inv_tG_E`). Um morfismo de grafo tipado de G^T para H^T é definido por um morfismo $g = (g\_V, g\_E)$ de G para H, com a seguinte condição sendo satisfeita:

$$\texttt{grd\_vertices:} \quad \forall v \in dom(g\_V) \cdot tG\_V(v) = tH\_V(g\_V(v)) \text{ e}$$
$$\texttt{grd\_edges:} \quad \forall e \in dom(g\_E) \cdot tG\_E(e) = tH\_E(g\_E(e))$$

A Figura 2 mostra um exemplo de grafo tipado que é composto por dois grafos G e T com um morfismo total, o qual relaciona arcos e vértices de G com seu respectivo tipo em T, criando assim o grafo tipado G^T. É possível observar o mapeamento nas cores vermelho (vértices) e azul (arcos), bem como a definição explícita de tG_V e tG_E.



$$tG\_V = \{N11 \mapsto Node, N12 \mapsto Node\}$$
$$tG\_E = \{Tok11 \mapsto Tok, Stb11 \mapsto Stb, Nxt11 \mapsto Nxt\}$$

Figura 2: Exemplo de Grafo Tipado G^T

Em um morfismo de grafos tipados g, as condições `grd_vertices` e `grd_edges` garantem que se vértices (respectivamente arcos) são mapeados por g, então eles devem ser de mesmo tipo. Por exemplo, se considerar um vértice (tipo v) de um grafo G mapeado para um grafo H, via morfismo g, o tipo do vértice (grafo H) deve ser o mesmo (tipo v). Isto pode ser verificado com a comutatividade do seguinte diagrama:

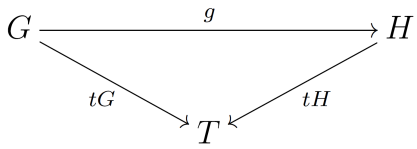


Figura 3: Diagrama de Morfismo de Grafos Tipados

No diagrama da Figura 3, H e T são grafos, g é o morfismo entre os grafos e tG e tH são funções totais (morfismo total) que relacionam os arcos e vértices dos grafos G e H com seu respectivo tipo no grafo T. A comutatividade deste diagrama garante que na definição de um morfismo entre grafos tipados, o mapeamento de vértices e arcos preservam o tipo.

O conjunto de regras de uma GG modela o comportamento do sistema. Uma regra é composta por dois grafos tipados sobre T, juntamente com um morfismo entre eles. O lado esquerdo da regra representa os itens que devem estar presentes no grafo-estado para a regra ser aplicada. Nas regras vértices ou arcos não podem ser identificados, elas são injetivas e não deletam vértices.

Definição 3 ***(Regra):*** *Uma regra α tipada sobre T é um morfismo de grafo tipado $\alpha = (\alpha\_V, \alpha\_E)\colon\ L^T \rightarrowtail R^T$, onde: L^T e R^T são grafos tipados sobre T; α é injetivo (`axm_alphaV` e `axm_alphaE`); $\alpha\_V : vertL \rightarrowtail vertR$ é uma função total (`axm_alphaV`).*

A Figura 4 mostra a regra $\alpha 1$ onde $L1$ e $R1$ são grafos tipados, representando o lado esquerdo e direito da regra, e os componentes do morfismo que representam o mapeamento de vértices (em azul $\alpha 1\_V$) e arcos (em vermelho $\alpha 1\_E$).

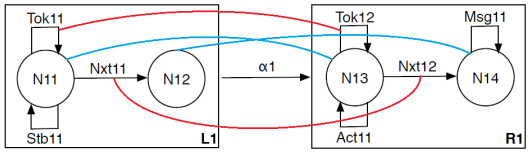
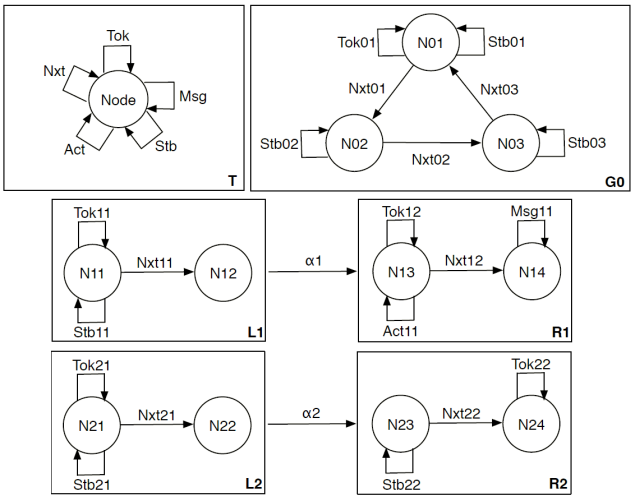


Figura 4: Regra $\alpha 1$

Gramática de grafos é composta por um grafo tipo; caracterizando os tipos de vértices e arcos permitidos no sistema; um grafo inicial, representando o estado inicial do sistema; e um conjunto de regras, que mostram as possíveis mudanças de estado que podem ocorrer no sistema.

Definição 4 ***(Gramática de Grafos):*** *Uma gramática de grafos tipada é uma tupla* $GG = (T, G0, r)$*, onde* T *é um grafo, chamado grafo tipo;* $G0$ *é um grafo tipado sobre* T*, chamado grafo inicial; e* r *é um conjunto de regras tipadas sobre* T*.*



$$\alpha 1 \begin{cases} \alpha 1\_V(N11) = N13 \\ \alpha 1\_V(N12) = N14 \\ \alpha 1\_E(Tok11) = Tok12 \\ \alpha 1\_E(Nxt11) = Nxt12 \end{cases} \qquad \alpha 2 \begin{cases} \alpha 2\_V(N21) = N23 \\ \alpha 2\_V(N22) = N24 \\ \alpha 2\_E(Stb21) = Stb22 \\ \alpha 2\_E(Nxt21) = Nxt22 \end{cases}$$

Figura 5: Gramática de Grafos

A Figura 5 mostra um exemplo de gramática de grafos, onde os morfismos entre as regras são apresentados de forma explícita. O Grafo tipo T define um vértice permitido (`Node`), e cinco tipos de arcos permitidos no sistema (`Nxt`, `Tok`, `Msg`, `Stb`, `Act`). O grafo $G0$ apresenta o estado inicial do sistema, neste caso foi definido com três vértices do tipo `Node` (`N01`, `N02`, `N03`) e com sete arcos: um arco do tipo `Tok` (`Tok01`); três arco do tipo `Stb` (`Stb01`, `Stb02`, `Stb03`); e três arcos do tipo `Nxt` (`Nxt01`, `Nxt02`, `Nxt03`). $\alpha 1$ e $\alpha 2$ representam o conjunto de regras da gramática, cada regra contém dois grafos tipados (tipagem expressa pelo prefixo), onde L e R representam o lado esquerdo e

direito da regra, respectivamente. Nesse exemplo o mapeamento de vértices e arcos para o grafo tipo foi omitido (o mapeamento é dado pelo prefixo) e o morfismo entre os grafos tipados de cada lado da regra foi explicitado por $\alpha 1$ ($\alpha 1\_V$ e $\alpha 1\_E$) e $\alpha 2$ ($\alpha 2\_V$ e $\alpha 2\_E$).

Uma regra é aplicável ao grafo-estado da gramática se ocorrer um *match*, isto é, uma imagem do lado esquerdo da regra no grafo-estado. O comportamento de uma GG é definida por aplicações de regras.

Definição 5 ***(Match):*** *Seja* $\alpha = (\alpha\_V, \alpha\_E)\colon\ L^T \rightarrowtail R^T$ *uma regra, onde* L^T *e* R^T *são grafos tipados sobre* T*. Seja* $G^T = (G, tG, T)$ *um grafo tipado, onde* $tG = (tG\_V, tG\_E)$*. Um* ***match*** m ***da regra*** r ***em*** G^T *é definido por um morfismo de grafos tipados total* $m = (m\_V, m\_E)\colon L^T \to G^T$*, onde* $m\_E\colon edgeL \rightarrowtail edgeG$ *é injetivo.*

A semântica operacional de uma GG é descrita por sucessivas aplicações de regras. Além disso, GG são inerentemente não-determinísticas, isto é, se várias regras são aplicáveis ao mesmo estado, então a escolha de qual será aplicada é não-determinística. Como se restringiu a abordagem para regras injetoras que não deletam vértices e *matches* que não identificam arcos, a aplicação de uma determinada regra através do *match* em um estado, remove todos os elementos que estão no lado esquerdo da regra e não são mapeados para o lado direito, e cria no estado todos os elementos que estão no lado direito da regra. O restante do estado permanece inalterado.

Definição 6 ***(Aplicação de Regra):*** *Seja* $\alpha = (\alpha\_V, \alpha\_E)\colon\ L^T \rightarrowtail R^T$ *uma regra e* $m = (m\_V, m\_E)$ *um match de* r *no grafo tipado* G^T*. Uma* ***aplicação de regra*** $G^T \overset{r,m}{\Rightarrow} H^T$*, ou aplicação de* r *a* G^T *em* m*, gera um grafo tipado* $H^T = (H, tH, T)$*, com* $H = (vertH, edgeH, sourceH, targetH)$*, da seguinte maneira:*

- $vertH = vertG \uplus (vertR - rng(\alpha\_V))$ *(*`act_vert`*);*
- $edgeH = (edgeG - rng(m\_E)) \uplus edgeR$ *(*`act_edge`*);*
- *para todo* $e \in edgeH$ *(*`act_src` *e* `act_tgt`*)*

$$sourceH(e) = \begin{cases} sourceG(e) & \text{se } e \in edgeG \\ \overline{m}(sourceR(e)) & \text{c.c} \end{cases}$$

$$targetH(e) = \begin{cases} targetG(e) & \text{se } e \in edgeG \\ \overline{m}(targetR(e)) & \text{c.c.} \end{cases}$$

onde $\overline{m}\colon vertR \to vertH$ *é definido por:*

$$\overline{m}(v) = \begin{cases} m\_V(v') & \text{se } v \in rng(\alpha\_V) \text{ e } v = \alpha\_V(v') \\ v & \text{c.c.} \end{cases}$$

- *para todo* $v \in vertH$ *e todo* $e \in edgeH$, $tH = (tH\_V, tH\_E)$ *é definido por (*`act_tV` *e* `act_tE`*):*

$$tH\_V(v) = \begin{cases} tG\_V(v) & \text{se } v \in vertG \\ tR\_V(v) & \text{c.c.} \end{cases}$$

$$tH\_E(e) = \begin{cases} tG\_E(v) & \text{se } e \in edgeG \\ tR\_E(v) & \text{c.c.} \end{cases}$$

Ao analisar o grafo inicial $G0$ e o conjunto de regras da GG (Figura 5) é possível observar que tanto $\alpha 1$ como $\alpha 2$ podem ser aplicadas no grafo inicial. Isto acontece pois todos os elementos do lado esquerdo dessas regras estão contidos no grafo inicial $G0$, logo existe a ocorrência dos *matches*.

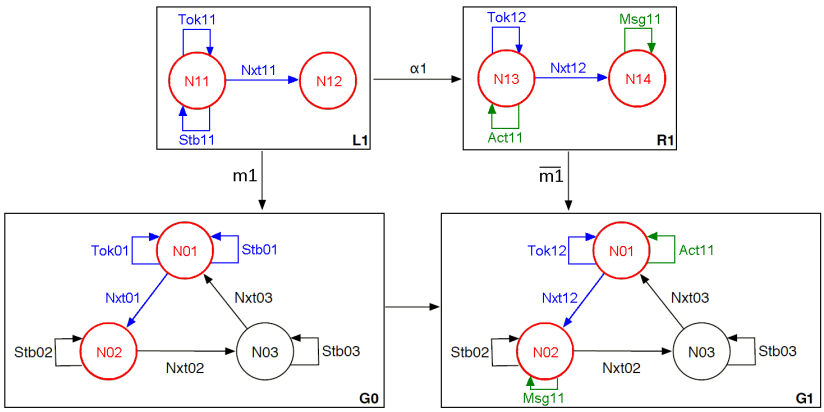


Figura 6: Aplicação da Regra $\alpha 1$ no Grafo Inicial $G0$ da Figura 5

A Figura 6 apresenta a aplicação da regra $\alpha 1$ em $G0$. É possível verificar que todos os elementos do lado esquerdo da regra foram mapeados para $G0$ via morfismo $m1$. $m1\_V$ mapeia os vértices de $L1$ em vértices de $G0$ (em vermelho $m1\_V = \{N11 \mapsto N01, N12 \mapsto N02\}$) e $m1\_E$ mapeia arcos de $L1$ em arcos de $G0$ (em azul $m1\_E = \{Tok11 \mapsto Tok01, Stb11 \mapsto Stb01, Nxt11 \mapsto Nxt01\}$). Com a ocorrência do *match* a regra pode ser aplicada gerando o grafo $G1$, que representa o novo estado da gramática. De forma intuitiva e geral, elementos que estão do lado esquerdo da regra e foram mapeados via morfismo $\alpha 1$ são preservados no novo estado (`N01`, `N02`, `Tok11`, `Nxt11`). Já os elementos que não foram mapeados via $\alpha 1$ serão deletados em $G1$ (`Stb11`) e os elementos que estão no lado direito da regra e não são imagem de nenhum elemento de $\alpha 1$ serão criados no novo estado (`Act11`, `Msg11`).

2.1.1 Exemplo: O Protocolo *Token Ring*

Para exemplificar o uso de GG para especificação formal é mostrada a aplicação em uma rede que usa o protocolo *token ring*. Esse protocolo é utilizado para controlar o acesso de várias estações a um meio de transmissão compartilhado, em uma rede que tem a topologia de anel. De acordo com esse protocolo, um padrão especial, denominado *token*, é transmitido de estação em estação em apenas uma direção. Quando uma estação quer enviar algum conteúdo pela rede, ela espera pelo *token*, retém ele e envia a mensagem pelo anel. A mensagem circula no anel de forma que todas as estações podem copiar o seu conteúdo. Quando a mensagem completa o ciclo, ela é recebida pela estação original, a qual remove a mensagem do anel volta para o estado em espera e envia o *token* para a próxima estação, reiniciando o ciclo. Se apenas um *token* existe, somente uma estação pode estar transmitindo num instante de tempo. Este protocolo foi modelado em um ambiente que novas estações podem ser adicionadas a qualquer instante. A Figura 7, ilustra a GG para este exemplo.

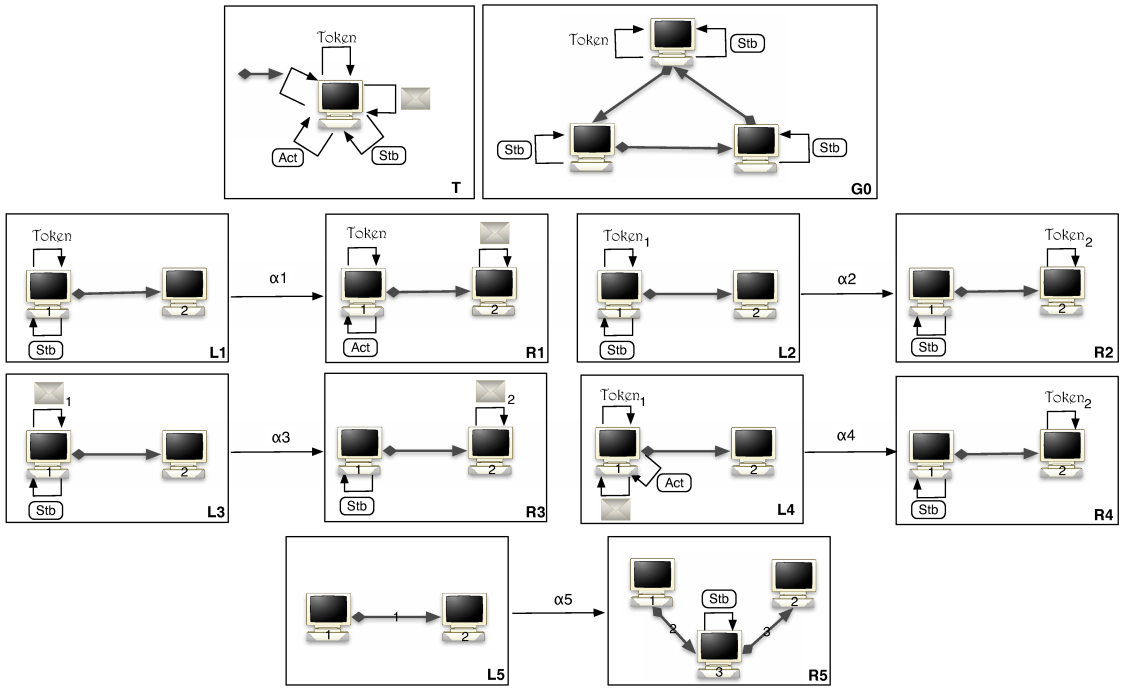


Figura 7: Gramática de Grafos - Token Ring

O grafo tipo T define um único tipo de vértice (estação), e cinco tipos de arcos, (mensagem), Token (token), (conexão), Act (estação ativa) e Stb (estação em espera). representa uma estação na rede e define uma porção de dados. As estações são conectadas por arcos do tipo . O Token representa um sinal especial que permite uma estação iniciar a transmissão. Cada estação ou está ativa (Act), significando que ela está transmitindo uma mensagem pela rede, ou está em espera (Stb). De acordo com esta especificação, só pode existir uma estação ativa no

anel a cada instante de tempo. O grafo inicial $G0$ define um anel com três estações. Inicialmente o *token* está em uma estação específica e nenhuma estação está transmitindo informação pela rede (todas as estações tem um arco (Stb)).

O comportamento do protocolo é modelado pelas regras. Na representação gráfica o morfismo não é representado explicitamente, se assume que os itens do grafo esquerdo são mapeados para itens com o mesmo nome no lado direito. Uma estação em espera com um arco *token* pode reter este arco e enviar uma mensagem, tornando-se uma estação ativa (regra $\alpha 1$), ou pode passar o *token* para a próxima estação (regra $\alpha 2$). Quando uma mensagem é recebida por uma estação em espera, a regra $\alpha 3$ pode ser aplicada e a mensagem é repassada para a próxima estação. Se a estação que recebe o *token* está ativa, então a regra $\alpha 4$ pode ser aplicada, removendo a mensagem do anel e enviando o *token* para a próxima estação. Regra $\alpha 5$ é aplicada para inserir novas estações no anel. Este modelo tem um espaço de estados infinito e gera um número infinito de computações. Isto porque sempre é possível adicionar (pela regra $\alpha 5$) uma nova estação no sistema.

2.2 Especificação de GG em Event-B

Nesta seção apresenta-se uma especificação de GG em Event-B, primeiramente será apresentado as definições gerais de um modelo Event-B e após como se especifica de fato um sistema descrito em GG na linguagem Event-B. Todos os exemplos apresentados fazem relação ao exemplo *token ring* da Figura 7 modelado em GG.

2.2.1 Event-B

Um modelo em Event-B (DEPLOY, 2011) é composto por duas partes, o contexto e a máquina. No contexto se definem os conjuntos, as constantes e os axiomas (componentes estáticos do modelo). Já na máquina são definidos as variáveis, as invariantes e os eventos.

Definição 7 ***(Modelo Event-B, Evento):*** *Um modelo Event-B é definido por uma tupla* $ModeloEB = (c, s, P, v, I, R_I, E)$*, onde* c *são constantes e* s *são conjuntos conhecidos no modelo;* $P(c, s)$ *é um conjunto de axiomas que restringem* c *e* s*;* v *são variáveis do modelo;* $I(c, s, v)$ *é uma invariante do modelo a qual limita os possíveis estados de* v*, tal que,* $\exists c, s, v \cdot P(c, s) \wedge I(c, s, v)$ *- isto é* P *e* I *caracterizam um conjunto não vazio de estados do modelo;* $R_I(c, s, v')$ *é uma ação de inicialização das variáveis do modelo; e* E *é um conjunto de eventos do modelo.*

Dados os estados v, v' *um evento é uma tupla* $e = (H, S)$*, onde* $H(c, s, v)$ *é a condição de guarda e* $S(c, s, v, v')$ *é o predicado que define uma relação entre os estados atuais e os seguintes. Também denota-se um evento de guarda por* $H(v)$*, o predicado antes-depois por* $S(v, v')$ *e a ação de inicialização por* $R_I(v')$*.*

O contexto define a tripla (c, s, P) e a máquina define os outros elementos (v, I, R_I, E).

A correção de um modelo é garantida através da geração e demonstração de um conjunto de obrigações de prova. A condição de consistência do modelo indica que sempre que um evento ou uma ação de inicialização é executada, existe um novo estado v', tal que a invariante do modelo é preservada - $I(v')$. Geralmente são declaradas duas obrigações de prova separadas: uma de viabilidade $(I(v) \wedge H(v) \Rightarrow \exists v' \cdot S(v, v'))$ e outra de preservação da invariante $(I(v) \wedge H(v) \wedge S(v, v') \Rightarrow I(v'))$. O comportamento de um modelo em Event-B é um sistema de transição definido como segue.

Definição 8 ***(Comportamento do Modelo Event-B):*** *Dado um* $ModeloEB = (c, s, P, v, I, R_I, E)$*, seu comportamento é dado por um sistema de transição* $BST = (BState, BS_0, \rightarrow)$*, onde:* $BState = \{\langle v \rangle | v$ *é um estado*$\} \cup Undef$*,* $BS_0 = Undef$*, e* $\rightarrow \subseteq BState \times BState$ *é a relação de transição descrita pelas regras:*

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

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

Figura 8: Comportamento de um Modelo Event-B

De acordo com a regra de inicialização, o modelo é inicializado em um estado que satisfaz $R_I \wedge I$ e sempre que existir um evento que satisfaz as condições de guarda (regra transição) o modelo pode evoluir disparando o evento e computando o próximo estado, de acordo com o predicado $S(v, v')$. Os eventos são atômicos e no caso de existir mais de um evento ativo a escolha é não-determinística.

A plataforma Rodin (ABRIAL et al., 2010) é um IDE (ambiente integrado de desenvolvimento), baseado no Eclipse (ECLIPSE, 2004), o qual utiliza a linguagem Event-B (ABRIAL, 2010). Ela fornece um auxílio para demonstrações matemáticas, é uma ferramenta *open source* aberta a outras extensões em forma de plugins e utiliza, como principal técnica de verificação, a prova de teoremas. Através do suporte para prova de teoremas é possível verificar as estruturas das provas, bem como hipóteses ou lemas utilizados nas demonstrações. Na ferramenta é possível instalar provadores.

2.2.2 GG em Event-B

O comportamento de um modelo Event-B é semelhante a de uma GG. Há um conceito de estado (dado por um conjunto de variáveis em Event-B e por um grafo em GG), uma transição é definida por uma operação atômica no estado atual (em

Event-B um evento que atualiza as variáveis e em GG uma aplicação de regra). Cada etapa deve preservar as propriedades do estado, em Event-B estas propriedades são declaradas como invariantes e em GG as propriedades garantidas estão relacionadas com a estrutura de um grafo.

Assumindo que uma GG tem n regras, nomeadas $\alpha i : Li^T \rightarrowtail Ri^T$ e $i \in \{1, ..., n\}$. Os componentes em Event-B que descrevem essa gramática são:

- Os conjuntos no modelo são $vertT$, $edgeT$ (os conjuntos de vértices e arestas do grafo tipo T), $vertLi$, $edgeLi$, $vertRi$ e $edgeRi$ (os conjuntos de vértices e arestas do lado esquerdo e direito da regra i).

- As constantes são os vértices e arcos do grafo tipo e das regras, os nomes das funções de tipagem tLi_V, tLi_E, tRi_V e tRi_E, os nomes das funções de origem e destino $sourceT$, $targetT$, $sourceLi$, $targetLi$, $sourceRi$ e $targetRi$ e os nomes das componentes das regras αi_V e αi_E.

- Os axiomas definem de forma explícita todos os conjuntos e funções do modelo (declarados acima). Os tipos das funções também são declaradas como axiomas.

- As variáveis do modelo são especificadas pelas relações $vertG$, $edgeG$, $sourceG$, $targetG$, tG_V e tG_E. Elas definem um estado alcançável do sistema pelo grafo tipado G.

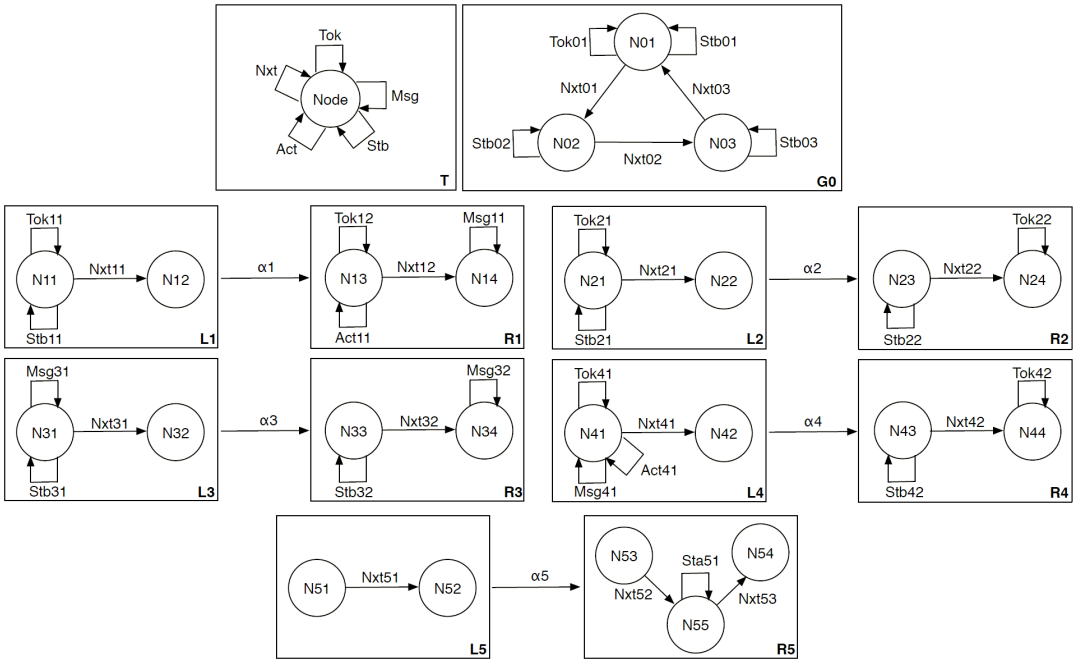
- As invariantes são utilizadas para definir os tipos das variáveis.

- A ação de inicialização define os valores iniciais para as variáveis $vertG$, $edgeG$, $sourceG$, $targetG$, tG_V e tG_E. Ela especifica o grafo inicial $G0$.

- O conjunto de eventos modela as aplicações de regras, um evento é definido para cada regra da gramática. A condição de guarda garante a ocorrência de um *match* do lado esquerdo da regra no grafo-estado da gramática.

A especificação de uma GG em Event-B também é realizada em dois componentes, um contexto e uma máquina. No contexto é especificado quase toda a parte estática do modelo (grafos e regras) e na máquina, além das variáveis de estado e invariantes, é especificada a parte dinâmica do modelo (aplicações de regras). Cabe salientar que existe uma dependência entre estes componentes, necessária para apresentar o comportamento do sistema (a máquina deve enxergar o contexto).

A seguir será descrito com mais detalhes como cada componente de uma GG é definido em Event-B. Se utilizou o exemplo *token ring*, no qual todos os arcos e vértices foram renomeados, conforme ilustra a Figura 9. Ela representa a mesma gramática da Figura 7 (*token ring*), com os vértices e arcos renomeados.



$$\alpha1 \begin{cases} \alpha1\_V(N11) = N13 \\ \alpha1\_V(N12) = N14 \\ \alpha1\_E(Tok11) = Tok12 \\ \alpha1\_E(Nxt11) = Nxt12 \end{cases} \quad \alpha2 \begin{cases} \alpha2\_V(N21) = N23 \\ \alpha2\_V(N22) = N24 \\ \alpha2\_E(Stb21) = Stb22 \\ \alpha2\_E(Nxt21) = Nxt22 \end{cases}$$

$$\alpha3 \begin{cases} \alpha3\_V(N31) = N33 \\ \alpha3\_V(N32) = N34 \\ \alpha3\_E(Stb31) = Stb32 \\ \alpha3\_E(Nxt31) = Nxt32 \end{cases} \quad \alpha4 \begin{cases} \alpha4\_V(N41) = N43 \\ \alpha4\_V(N42) = N44 \\ \alpha4\_E(Nxt41) = Nxt42 \end{cases}$$

$$\alpha1 \begin{cases} \alpha1\_V(N51) = N53 \\ \alpha1\_V(N52) = N54 \end{cases}$$

Figura 9: GG Token Ring

2.2.3 Contexto

Grafo tipo e regras são os elementos estáticos de uma GG, logo são definidos no contexto do modelo.

2.2.3.1 Definição de Grafo

Para definir-se um grafo é preciso que no contexto apareçam: como conjuntos, os nomes dos conjuntos de vértices e de arcos; como constantes, os nomes dos vértices e arcos e os nomes das funções de origem e destino; como axiomas, as atribuições dos conjuntos de vértices e arcos, bem como a definição das funções origem e destino.

As Figuras 10, 11 e 12 mostram exemplos de especificação de grafo simples na linguagem Event-B. A Figura 10 especifica o grafo tipo T e as Figuras 11 e 12 especificam, respectivamente, o lado esquerdo e direito da regra $\alpha1$. Na Figura 10,

CONTEXT ctx_T
SETS
vertT
edgeT
CONSTANTS
Node
Nxt Tok Msg Stb Act
sourceT
targetT
AXIOMS
axm_vertT : $partition(vertT, \{Node\})$
axm_edgeT : $partition(edgeT, \{Nxt\}, \{Tok\}, \{Msg\}, \{Stb\}, \{Act\})$
axm_srcTtype : $sourceT \in edgeT \rightarrow vertT$
axn_srcTdef : $partition(sourceT, \{Nxt \mapsto Node\}, \{Tok \mapsto Node\}, \{Msg \mapsto Node\}, \{Stb \mapsto Node\}, \{Act \mapsto Node\})$
axm_tgtTtype : $targetT \in edgeT \rightarrow vertT$
axn_tgtTdef : $partition(targetT, \{Nxt \mapsto Node\}, \{Tok \mapsto Node\}, \{Msg \mapsto Node\}, \{Stb \mapsto Node\}, \{Act \mapsto Node\})$
END

Figura 10: Grafo Tipo T em Event-B

CONTEXT ctx_L1
SETS
vertL1
edgeL1
CONSTANTS
N11 N12
Tok11 Stb11 Nxt11
sourceL1
targetL1
AXIOMS
axm_vertL1 : $partition(vertL1, \{N11\}, \{N12\})$
axm_edgeL1 : $partition(edgeL1, \{Tok11\}, \{Stb11\}, \{Nxt11\})$
axm_srcL1type : $sourceL1 \in edgeL1 \rightarrow vertL1$
axn_srcL1def : $partition(sourceL1, \{Tok11 \mapsto N11\}, \{Stb11 \mapsto N11\}, \{Nxt11 \mapsto N11\})$
axm_tgtL1type : $targetL1 \in edgeL1 \rightarrow vertL1$
axn_tgtL1def : $partition(targetL1, \{Tok11 \mapsto N11\}, \{Stb11 \mapsto N11\}, \{Nxt11 \mapsto N12\})$
END

Figura 11: Grafo $L1$ em Event-B

CONTEXT ctx_R1
SETS
vertR1
edgeR1
CONSTANTS
N13, N14
Tok12, Act11, Nxt12, Msg11
sourceR1
targetR1
AXIOMS
axm_vertR1 : $partition(vertR1, \{N13\}, \{N14\})$
axm_edgeR1 : $partition(edgeR1, \{Tok12\}, \{Act11\}, \{Nxt12\}, \{Msg11\})$
axm_srcR1type : $sourceR1 \in edgeR1 \rightarrow vertR1$
axn_srcR1def : $partition(sourceR1, \{Tok12 \mapsto N13\}, \{Act11 \mapsto N13\}, \{Nxt12 \mapsto N13\}, \{Msg11 \mapsto N14\})$
axm_tgtR1type : $targetR1 \in edgeR1 \rightarrow vertR1$
axn_tgtR1def : $partition(targetR1, \{Tok12 \mapsto N13\}, \{Act11 \mapsto N13\}, \{Nxt12 \mapsto N14\}, \{Msg11 \mapsto N14\})$
END

Figura 12: Grafo $R1$ em Event-B

$vertT$ e $edgeT$ são os nomes dos conjuntos de vértices e arcos, respectivamente; $Node$, Nxt, Msg, Stb, Act são os nomes das constantes; $sourceT$ e $targetT$ são os nomes das funções de origem e destino, respectivamente; axm_vertT e axm_edgeT são as definições dos conjuntos de vértices e arcos, respectivamente; axm_srcTtype e axm_tgtTtype definem os tipos das funções; e axm_srcTdef e axm_tgtTdef definem as atribuições das funções. Os grafos das Figuras 11 e 12 são definidos de forma análoga ao apresentado para a Figura 10.

Os grafos de uma GG, desconsiderando o grafo T, são grafos tipados sobre T. O grafo inicial, os grafos do lado esquerdo e direito das regras, o grafo-estado, todos são tipados sobre T. Na especificação um grafo tipado é definido por dois grafos e por um morfismo entre eles, chamado morfismo de tipagem.

2.2.3.2 Definição de Grafo Tipado

Para definir-se um grafo tipado na linguagem Event-B é necessário além de dois grafos, especificados conforme descrito na subseção anterior, um morfismo de tipagem. Para isso, no contexto devem ser acrescentados os seguintes elementos: como constantes, os nomes dos componentes do morfismo dos vértices e arcos; como axiomas, definir as funções de tipagem, as quais mapeiam elementos de um grafo em elementos do grafo tipo, ou seja, vértices (respectivamente arcos) do grafo serão ma-

peados para vértices (respectivamente arcos) do grafo tipo, resultando em um grafo tipado.

CONSTANTS

tL1_V

tL1_E

AXIOMS

axm_tL1_V : $tL1\_V \in vertL1 \rightarrow vertT$

axm_tL1_V_def : $partition(tL1\_V, \{N11 \mapsto Node\}, \{N12 \mapsto Node\})$

axm_tL1_E : $tL1\_E \in edgeL1 \rightarrow edgeT$

axm_tL1_E_def : $partition(tL1\_E, \{Tok11 \mapsto Tok\}, \{Stb11 \mapsto Stb\}, \{Nxt11 \mapsto Nxt\})$

Figura 13: Tipagem do Grafo $L1$

CONSTANTS

tR1_V

tR1_E

AXIOMS

axm_tR1_V : $tR1\_V \in vertR1 \rightarrow vertT$

axm_tR1_V_def : $partition(tR1\_V, \{N13 \mapsto Node\}, \{N14 \mapsto Node\})$

axm_tR1_E : $tR1\_E \in edgeR1 \rightarrow edgeT$

axm_tR1_E_def : $partition(tR1\_E, \{Tok12 \mapsto Tok\}, \{Act11 \mapsto Act\}, \{Nxt12 \mapsto Nxt\}, \{Msg11 \mapsto Msg\})$

Figura 14: Tipagem do Grafo $R1$

A Figura 13 mostra os acréscimos necessários para especificação do morfismo de tipagem do grafo $L1$ (representado na Figura 11) para o grafo T (representado na Figura 10). Onde, $tL1\_V$ e $tL1\_E$ são os nomes dos componentes do morfismo (declaradas como constantes) e axm_tL1_V, axm_tL1_V_def, axm_tL1_E e axm_tL1_E_def são os axiomas que definem os componentes. De forma análoga, a Figura 14 mostra os acréscimos necessários (códigos das Figuras 10 e 12) para se representar o morfismo de tipagem que define o grafo tipado $R1^T$.

Dados dois grafos tipados sobre um grafo tipo T, o morfismo entre esses grafos é chamado morfismo entre grafos tipados.

2.2.3.3 Definição de Morfismo entre Grafos Tipados

Para definir-se um morfismo entre grafos tipados, inclui-se na especificação a definição dos componentes do morfismo. No contexto, devem ser acrescentados os seguintes elementos: como constantes, os nomes dos componentes do morfismo; e como axiomas, as definições dos componentes do morfismo.

CONSTANTS

alpha1V

alpha1E

AXIOMS

axm_alpha1V : $alpha1V \in vertL1 \rightarrowtail vertR1$

axm_alpha1V_def : $partition(alpha1V, \{N11 \mapsto N13\}, \{N12 \mapsto N14\})$

axm_alpha1E : $alpha1E \in edgeL1 \rightarrowtail edgeR1$

axm_alpha1E_def : $partition(alpha1E, \{Tok11 \mapsto Tok12\}, \{Nxt11 \mapsto Nxt12\})$

Figura 15: Morfismo Regra $\alpha 1$

A Figura 15 mostra os acréscimos necessários para definir-se o morfismo (regra) entre os grafos $L1^T$ e $R1^T$. $\alpha 1\_V$ e $\alpha 1\_E$ são os nomes dos componentes do morfismo, declarados como constantes e axm_alpha1V, axm_alpha1V_def, axm_alpha1E e axm_alpha1E_def são os axiomas que definem esses componentes.

Em resumo os elementos especificados no componente contexto para definir-se uma GG em Event-B são os conjuntos, as constantes e os axiomas. Os demais elementos são especificados no componente máquina.

2.2.4 Máquina

As variáveis de estado, o grafo inicial e as aplicações de regras são definidos na máquina do modelo Event-B. Nesse componente são definidos as variáveis, as invariantes e os eventos. As variáveis representam o grafo-estado do modelo, as invariantes definem os tipos das variáveis (Figura 16), já os eventos definem o grafo inicial e as aplicações de regras.

Na Figura 16 as variáveis ($vertG$, $edgeG$, $sourceG$, $targetG$, tG_VG, tG_E) definem um grafo-estado e as invariantes (inv_vertG, inv_edgeG, inv_sourceG, inv_targetG, inv_tG_V, inv_tG_E) estabelecem seus tipos. Neste exemplo a variável $vertG$ foi declarada como um subconjunto dos números naturais ($vertG \in \mathbb{P}(\mathbb{N})$). Já $sourceG$, por exemplo, foi especificada como uma função total que mapeia arcos de $edgeG$ em vértices de $vertG$ ($sourceG \in edgeG \rightarrow vertG$).

Ainda no componente máquina é especificado o evento de inicialização, o qual representa o estado inicial da gramática de grafos.

2.2.4.1 Grafo Inicial

O evento de inicialização define os valores iniciais para as variáveis, ou seja, especifica o grafo inicial, conforme ilustrado na Figura 17.

A Figura 17 mostra a definição do evento de inicialização na linguagem Event-B (grafo inicial da GG). É observado que os nomes dos vértices e arcos foram renomeados para números naturais (o grafo $G0$ da Figuras 9 é especificado pelo grafo da

MACHINE mch_trAll
SEES ctx_GG
VARIABLES
vertG
edgeG
sourceG
targetG
tG_V
tG_E
INVARIANTS
inv_vertG : $vertG \in \mathbb{P}(\mathbb{N})$
inv_edgeG : $edgeG \in \mathbb{P}(\mathbb{N})$
inv_sourceG : $sourceG \in edgeG \rightarrow vertG$
inv_targetG : $targetG \in edgeG \rightarrow vertG$
inv_tG_V : $tG\_V \in vertG \rightarrow vertT$
inv_tG_E : $tG\_E \in edgeG \rightarrow edgeT$
END

Figura 16: Componente Máquina Event-B

Figura 18).

Nessa figura, os nomes dos vértices e arcos de $G0$ são números naturais, todos de acordo com a definição das invariantes. Por exemplo, o arco 3 é do tipo Tok, possui origem no vértice 1 e destino no vértice 2.

Os demais eventos determinam as possíveis mudanças de estado, que em GG são determinadas por aplicação de regras.

2.2.4.2 Aplicação de Regras

A aplicação de uma regra é definida na máquina do modelo como um evento. A condição de guarda determina quando a regra pode ser aplicada, isto é, quando existir um *match* do lado esquerdo da regra no grafo-estado, e as ações definem o valor das variáveis após a aplicação da regra.

A Figura 19 mostra a definição da aplicação da regra $\alpha 1$. Sempre que houver valores para as variáveis mV, mE, $newEmsg$ e $newEact$, que satisfaçam as condições de guarda, o evento pode ocorrer. As condições de guarda grd_mV, grd_mE, grd_vertices, grd_edges e grd_srctrg asseguram que o par (mV, mE) é uma ocorrência de um *match* do lado esquerdo da regra no grafo-estado. As condições de guarda grd_newEmsg, grd_newEact e grd_E1E2 asseguram que $newEmsg$ e $newEact$ são novos arcos não pertencentes ao grafo-estado e ainda, garantem que os mesmos são diferentes. As ações act_E, act_src, act_tgt e act_tE alteram o grafo-estado, de acordo com a sua definição. Nesse caso um arco de tipo Stb será deletado e dois arcos dos

EVENTS
Initialisation
begin
act_vertG : $vertG := \{1, 2, 3\}$
act_edgeG : $edgeG := \{1, 2, 3, 4, 5, 6, 7\}$
act_srcG : $sourceG := \{1 \mapsto 1, 2 \mapsto 1, 3 \mapsto 1, 4 \mapsto 2, 5 \mapsto 2, 6 \mapsto 3, 7 \mapsto 3\}$
act_tgtG : $targetG := \{1 \mapsto 1, 2 \mapsto 1, 3 \mapsto 2, 4 \mapsto 2, 5 \mapsto 3, 6 \mapsto 3, 7 \mapsto 1\}$
act_tG_V : $tG\_V := \{1 \mapsto Node, 2 \mapsto Node, 3 \mapsto Node\}$
act_tG_E : $tG\_E := \{1 \mapsto Tok, 2 \mapsto Stb, 3 \mapsto Nxt, 4 \mapsto Stb, 5 \mapsto Nxt, 6 \mapsto Stb, 7 \mapsto Nxt\}$
end
END

Figura 17: Evento Grafo Inicial Event-B

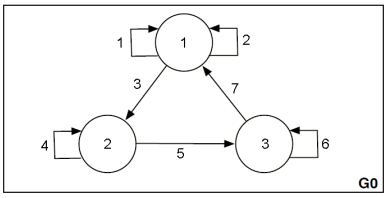


Figura 18: Grafo Inicial $G0$ com Números Naturais

tipos Msg e Act serão criados. Uma vez que o conjunto de vértices não é modificado pela regra, as variáveis relacionadas não são modificadas.

As condições de guarda são compostas por: grd_mV que define mV como uma função total nos vértices; grd_mE que define mE como uma função total e injetiva nos arcos; grd_newEmsg e grd_newEact garantem que os novos arcos pertencem aos $\mathbb{N}$ menos os números que já estão em uso; grd_E1E2 garante que o $newEmsg$ seja diferente de $newEact$; grd_vertices e grd_edges garantem a compatibilidade de tipo, respectivamente, dos vértices e arcos, mapeados pelo *match*; e grd_srctrg assegura que o mapeamento de origem e destino no mapeamento dos arcos pelo *match* será preservado. Já as ações são compostas por: act_E a qual exclui o arco $Stb11$ e adiciona os arcos $newEmsg$ e $newEact$; act_src e act_tgt que definem a origem e destino do novo conjunto de arcos, respectivamente; e act_tE a qual define a tipagem do novo conjunto de arcos.

Após especificação da GG em Event-B, dentro da plataforma Rodin se inicia o processo de prova, mais precisamente na perspectiva de prova da ferramenta. De maneira geral sempre que uma variável é modificada pelo evento de inicialização (estado inicial) e pelos demais eventos (aplicações de regras), serão geradas obrigações de prova, as quais devem ser demonstradas visando preservar os tipos das variáveis, que foram declaradas como invariantes. Existem vários tipos de obrigações de prova geradas para uma GG. Também é observado que as obrigações genéricas para uma GG seguem um mesmo padrão, isso também acontece com a prova de tais obrigações.

EVENTS
Event $alpha1$
any
mV
mE
$newEmsg$
$newEact$
where
grd_mV : $mV \in vertL1 \rightarrow vertG$
grd_mE : $mE \in edgeL1 \rightarrowtail edgeG$
grd_newEmsg : $newEmsg \in \mathbb{N} \setminus edgeG$
grd_newEact : $newEact \in \mathbb{N} \setminus edgeG$
grd_E1E2 : $newEmsg \neq newEact$
grd_vertices : $\forall v \cdot v \in vertL1 \Rightarrow tL1\_V(v) = tG\_V(mV(v))$
grd_edges : $\forall e \cdot e \in edgeL1 \Rightarrow tL1\_E(e) = tG\_E(mE(e))$
grd_srctgt : $\forall e \cdot e \in edgeL1 \Rightarrow mV(sourceL1(e)) = sourceG(mE(e)) \land$
$mV(targetL1(e)) = targetG(mE(e))$
then
act_E : $edgeG := (edgeG \setminus \{mE(Stb11)\}) \cup \{newEmsg, newEact\}$
act_src : $sourceG := (\{mE(Stb11)\} \lhd sourceG) \cup \{newEact \mapsto mV(N11),$
$newEmsg \mapsto mV(N12)\}$
act_tgt : $targetG := (\{mE(Stb11)\} \lhd targetG) \cup \{newEact \mapsto mV(N11),$
$newEmsg \mapsto mV(N12)\}$
act_tE : $tG\_E := (\{mE(Stb11)\} \lhd tG\_E) \cup \{newEact \mapsto Act,$
$newEmsg \mapsto Msg\}$
end
END

Figura 19: Evento Aplicação de Regra em Event-B

O objetivo do próximo capítulo é de tratar desses padrões, mas primeiramente será apresentada um resumo do ambiente de prova da plataforma Rodin.

3 O AMBIENTE DE PROVA RODIN E OBRIGAÇÕES DE PROVA GERADAS PELA ESPECIFICAÇÃO DE UMA GG

Neste capítulo descreve-se a perspectiva de prova da ferramenta Rodin (ABRIAL et al., 2010). Inicialmente, define-se regra de prova e descrevem-se os principais provadores disponíveis para a plataforma. Também neste capítulo são explanadas as obrigações de prova geradas para um sistema de GG descrito em Event-B. As obrigações consideradas neste capítulo são aquelas que serão sempre geradas após a especificação de um sistema nesse formalismo.

3.1 O Ambiente de Prova da Plataforma Rodin

Na ferramenta Rodin, após a especificação dos componentes contexto e máquina, é possível acessar o ambiente de prova, chamada de perspectiva de prova. Nessa área se encontra as obrigações de prova a serem demonstradas. Ao selecionar uma das obrigações de prova, por *default*, um conjunto de táticas de prova (DEPLOY, 2011) são aplicadas automaticamente pela ferramenta e, então, é possível visualizar as hipóteses selecionadas, o controle de prova e a árvore de prova. A Figura 20 apresenta a visão da perspectiva de prova da ferramenta. Na janela do controle de prova é possível identificar se a demonstração foi concluída ou não. Neste último caso o usuário deve interagir com a ferramenta (selecionando hipóteses, aplicando táticas, executando provadores, instanciando variáveis, entre outros) para a conclusão da prova.

O status de uma demonstração pode ser verificada através da cor do ícone *Smiley*, no ambiente de controle de prova: verde, representa que os sequentes foram demonstrados; vermelho, existem sequentes para ser demonstrados; e azul, indicando que algum sequente da árvore foi revisado e posteriormente deverá ser demonstrado.

Ao iniciar a prova de uma obrigação, independentemente de sua demonstração (completa ou parcial), uma árvore de prova é gerada. A partir de uma navegação na árvore, se acessa todas as regras e táticas que foram aplicadas em cada sequente. Uma forma de facilitar a compreensão das regras aplicadas está na utilização de *rule*

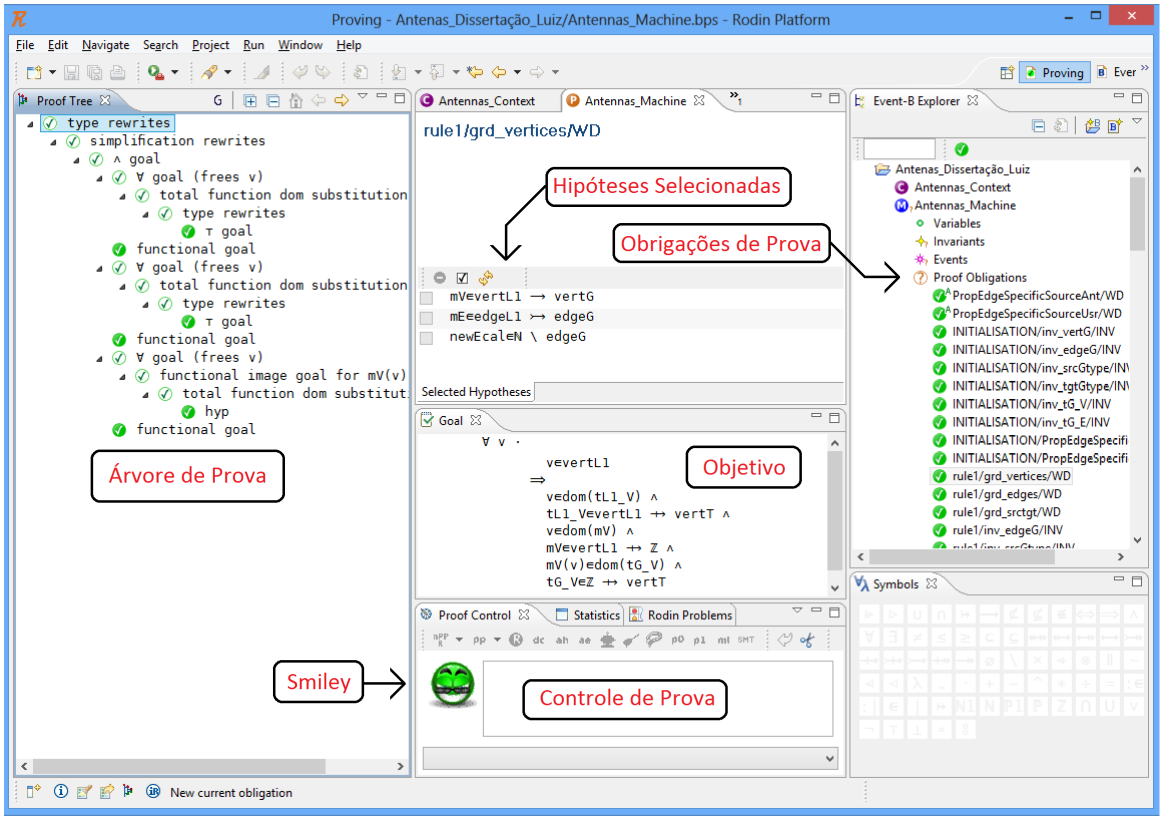


Figura 20: Perspectiva de Prova Plataforma Rodin

details, onde os detalhes para cada nodo são observados (tática/regra aplicada, antecedente, sequente de entrada, hipóteses que foram selecionadas ou desselecionadas). Cada nodo da árvore é um sequente, onde as hipóteses selecionadas representam o antecedente e o objetivo representa o consequente. Cada nodo da árvore é rotulado com um comentário que descreve como o sequente foi demonstrado.

A Figura 21 apresenta uma árvore de prova gerada por uma obrigação de prova.

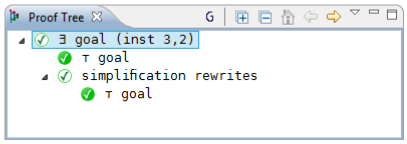


Figura 21: Árvore de Prova

Na ferramenta as árvores de prova são apresentadas conforme ilustrado na Figura 21. Uma linha é deslocada a direita quando o nodo correspondente é um descendente direto do nodo da linha anterior. Nesse caso, se apresenta uma árvore com quatro nodos, o primeiro é representado por ∃ goal (inst 3, 2). Esse nodo possui dois descendentes diretos ⊤ goal e simplification rewrites, que por sua vez possui um descendente (⊤ goal). O símbolo ✓ indica que o nodo foi demonstrado. As árvores neste trabalho serão apresentadas de duas maneiras distintas.

Em uma das representações adotadas, ilustrada na Figura 22, a árvore é descrita

pelas regras de inferência, onde o objetivo (sequente a ser demonstrado) é a raiz da árvore.

$$\dfrac{\dfrac{\text{True}}{\vdash \top}\,2 \qquad \dfrac{\dfrac{\text{True}}{\vdash \top}\,4}{A'}\,3}{A}\,1$$

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

Figura 22: Árvore de Prova com Sequentes

Na Figura 22, A é o sequente (ou o objetivo) a ser demonstrado. Após instanciar as variáveis com 3 e 2 no consequente descrito por uma propriedade existencial (regra $\exists$ goal) dois novos sequentes devem ser demonstrados: (i) $\vdash \top$ e (ii) A'. A demonstração de (i) é concluída com a aplicação da regra $\top$ goal. Já para a conclusão do sequente (ii) foram aplicadas as regras simplification rewrites (3) e $\top$ goal (4).

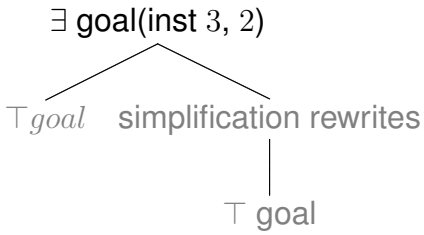


Figura 23: Árvore de Uso

A outra representação adotada é ilustrada na Figura 23, neste trabalho é chamada de Árvore de Uso. Ela mostra somente a informação das regras necessárias para demonstrar cada sequente (nodo). Os nodos em preto indicam que a aplicação da regra que rotula o nodo foi resultante de uma interação do usuário, já os nodos em cinza representam que as regras indicadas nos rótulos demonstraram automaticamente o sequente correspondente. Neste exemplo, para a demonstração da obrigação de prova se aplicou a regra $\exists$ goal(inst 3, 2) e os demais nodos foram demonstrados automaticamente.

A ferramenta conta com regras de prova, as quais estão divididas em regras de reescrita e regras de inferência. As regras de reescrita são unidirecionais, tratam igualdades e equivalências com a finalidade de simplificar as fórmulas e podem ser utilizadas no objetivo ou em uma única hipótese selecionada. Já as regras de inferência são aplicadas ao final de cada prova, realizando uma das seguintes ações: descarregar o objetivo; simplificar o objetivo e gerar uma hipótese (hipóteses selecionadas); simplificar o objetivo e gerar vários sub-objetivos; simplificar uma hipótese; ou simplificar uma hipótese e gerar várias hipóteses mais simples.

Essas regras são aplicadas de forma manual ou automática. A forma manual é realizada pelo usuário, assim exige um avançado conhecimento do funcionamento das regras e quando elas devem ser aplicadas. Já a forma automática é selecionada por *default* pela ferramenta, através das chamadas *auto* e *post-tactics*. A configuração *auto-tactic* aplica uma combinação de regras de reescrita, regras de inferência e provadores nas obrigações de prova recém geradas, visando sua demonstração. Já a *post-tactics* também aplica regras de reescrita, regras de inferência e provadores, mas após cada etapa da prova interativa. As regras de prova aplicadas no trabalho são descritas no Anexo A.

Uma regra de prova segue o padrão definido a seguir.

$$\frac{A}{C}\, R$$

Nessa regra, A é uma coleção (possivelmente vazia) de sequentes, chamado de antecedente. C é um sequente, representando o consequente e R é o nome de uma regra. A partir das hipóteses A, utilizando uma regra R se prova o sequente C.

Existem alguns provadores disponíveis para a plataforma Rodin. A ferramenta conta com um provador instalado chamado de NewPP, mas é recomendado que se instale os provadores da família Atelier B (CLEARSY, 2011) PP (predicate prover) e ML (mono-lemma). Estes provadores são os mais utilizados, mas há possibilidade de se instalar outros em forma de plugins (DEPLOY, 2011). Na maioria dos casos os provadores são de código fechado, assim não se conhece os métodos de prova por eles utilizados. A seguir segue uma descrição geral destes provadores.

O provador NewPP possui três configurações (forças). Na configuração *restricted* (nPP R), todas as hipóteses selecionadas e o objetivo são transferidos para NewPP. Na configuração *after lasso* (NewPP with lasso), uma operação de lasso é aplicada nas hipóteses selecionadas e no objetivo, e o resultado é transferido para NewPP. A operação de lasso seleciona qualquer hipótese não selecionada que possui algum símbolo em comum com o objetivo ou com hipótese já selecionada. Na configuração *unrestricted* (nPP), todas as hipóteses disponíveis são transferidas para NewPP. Este provador está incorporado na ferramenta por padrão e sua linguagem de entrada é a lógica de primeira ordem com o predicado $\in$. Primeiramente, todas as funções e símbolos de predicados que são diferentes de $\in$ e não relacionados com a aritmética são traduzidos. Por exemplo, $A \subseteq B$ é traduzido para $\forall x \cdot x \in A \Rightarrow x \in B$. Então NewPP traduz a obrigação de prova para forma normal conjuntiva e aplica uma combinação de princípio da resolução e o algoritmo Davis Putnam (DAVIS; PUTNAM, 1960; DAVIS; LOGEMANN; LOVELAND, 1962). Como ponto forte o provador gera o conjunto de hipóteses utilizadas e como pontos fracos não oferece suporte para aritmética, não considera alguns axiomas da teoria dos conjuntos e é limitado pela capacidade de

memória da máquina para entradas grandes. Se algumas hipóteses desnecessárias estão presentes, ele não descarrega as obrigações de prova.

O provador PP, disponível pela Atelier B (CLEARSY, 2011) como um provador externo, também possui três configurações (P0, P1, PP). Na configuração P0, todas as hipóteses selecionadas e o objetivo são transferidos para PP. Na configuração P1, uma operação de lasso é aplicada nas hipóteses selecionadas e no objetivo, e o resultado é transferido para PP. Na configuração PP, todas as hipóteses disponíveis são transferidas para PP. O sequente de entrada é transformado para linguagem clássica B e enviado para o provador PP do Atelier B (CLEARSY, 2011). PP é semelhante ao provador NewPP, mas com suporte para o raciocínio equacional e aritmético. Como ponto forte tem suporte limitado para argumentação aritmética e equacional. Como pontos fracos não gera o conjunto de hipóteses utilizadas e se hipóteses desnecessárias estão presentes, PP não consegue realizar a prova.

O provador ML também é disponibilizado pela Atelier B (CLEARSY, 2011), porém diferente dos outros provadores (PP e NewPP). ML aplica uma mistura dos métodos *forward*, *backward* e regras de reescrita a fim de demonstrar o objetivo (ou detectar uma contradição entre hipóteses). Como ponto forte tem o suporte limitado à argumentação equacional e aritmética. Já como pontos fracos não mostra o conjunto de hipóteses utilizadas e também não utiliza todos os axiomas da teoria dos conjuntos.

A forma de interação do usuário, necessária para demonstrar um obrigação de prova manual é realizada no controle de prova da Ferramenta. A Figura 24 apresenta a visão do controle de prova da ferramenta.

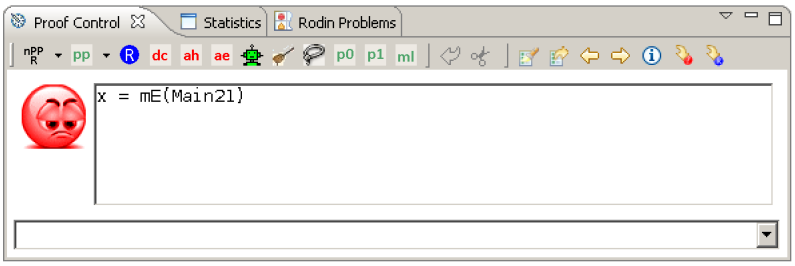


Figura 24: Controle de Prova Rodin

A descrição de cada item, da esquerda para a direita, é mostrado a seguir:

- Provador NewPP;
- Provador PP;
- Revisar o objetivo com as hipóteses selecionadas;
- Distinção de casos;
- Adicionar hipótese;

- Abstrair expressão;
- Executar a prova automática (*auto-tactics*);
- Executar *post-tactics*;
- Operação de Lasso;
- Provador P0;
- Provador P1;
- Provador ML;
- Retroceder a partir do nó atual;
- Poda o nó atual;
- Procura Hipóteses;
- Mostra a cache de hipóteses;
- Apresenta a obrigação de prova anterior;
- Apresenta a próxima obrigação de prova;
- Mostra informações da obrigação de prova;
- Avança para o próximo sub-objetivo pendente;
- Avança para o próximo sub-objetivo revisado.

Usualmente, para demonstrar uma obrigação de prova, adiciona-se hipóteses, instancia-se alguma variável, executa-se alguma regra de prova, ou ainda executa-se um provador. As *auto-tactics* e a operação de lasso são utilizadas de forma automática pela ferramenta e por alguns provadores.

Para incluir uma expressão como hipótese, se digita a mesma no controle de prova e após se pressiona o botão correspondente . Na Figura 24 é possível observar uma expressão digitada ($x = mE(Main21)$) no controle de prova, ao clicar no ícone que adiciona uma hipótese, a expressão é adicionada ficando disponível na árvore de prova. Geralmente, após adicionar uma hipótese, táticas automáticas são aplicadas.

No objetivo e nas hipóteses selecionadas ou não selecionadas, é possível encontrar ícones em vermelho (`partition`, $\cup$, $\in$, $\Rightarrow$,$\exists$, $\forall$, entre outros). Tais itens representam que uma regra de reescrita pode ser aplicada naquela expressão. Também no objetivo e nas hipóteses selecionas ou não, aparecem expressões com um retângulo em amarelo, onde é possível instanciar um elemento. Cabe salientar que em qualquer

interação um ou mais nodos são gerados na árvore de prova, indicando a tática ou regra de prova aplicada.

A Figura 25 mostra uma expressão onde é possível instanciar um elemento.

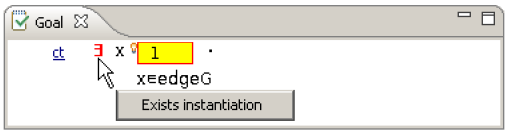


Figura 25: Instanciando um Elemento

Indo no ícone ∃ serão informadas quais regras de reescrita estão disponíveis para essa expressão, neste caso a regra disponível foi *Exists Instantiation* - ∃ goal (veja Anexo A). Ao clicar neste comando o elemento que se encontra dentro do retângulo em amarelo será instanciado. Neste exemplo, o número 1 será instanciado no objetivo $\exists x \cdot x \in edgeG$. As instanciações também podem ser realizadas nas hipóteses selecionadas.

Para uma GG especificada em Event-B, um conjunto de obrigações de prova é sempre gerado. Na próxima seção serão descritas essas obrigações e também serão indicados quais os provadores ou táticas que devem ser utilizados para a demonstração de cada uma delas.

3.2 Obrigações de Prova Geradas na Especificação de uma GG em Event-B

Ao especificar um sistema em Event-B, a ferramenta executa uma verificação estática e uma verificação dinâmica. Na verificação estática, a ferramenta realiza uma análise sintática da especificação, verificando se os nomes dos elementos e seus tipos estão de acordo, se não existe duplicidade, entre outros procedimentos relativos à sintaxe. Já na verificação dinâmica são geradas obrigações de prova, as quais devem ser descarregadas visando garantir que o sistema nunca alcance um estado inválido, ou seja, as variáveis satisfazem as invariantes. Algumas obrigações de prova são demonstradas de forma automática pela ferramenta e outras necessitam da intervenção do usuário.

As obrigações de prova geradas após a descrição de uma GG no Rodin são basicamente de dois tipos, para garantir que as especificações estão bem-definidas (rotuladas por WD) ou para preservar invariantes (rotuladas por INV).

Na especificação de uma GG em Event-B, o conjunto de variáveis representa o grafo-estado e as invariantes definem os tipos das variáveis. Obrigações de prova são geradas para garantir a preservação de seus tipos, tanto pelo evento de inicialização quanto pelos eventos que especificam as regras. Tais obrigações são geradas sempre que uma variável é modificada por estes eventos. Outras obrigações de prova visam

assegurar que as condições de guarda (condição para aplicação de uma regra) e ações (atualização de valores de variáveis) estejam bem definidas.

A Tabela 1 apresenta as obrigações de prova geradas quando se especifica uma GG em Event-B. Tais obrigações são facilmente descarregadas pela ferramenta (automaticamente) ou simplesmente executando um provador. Além do nome da obrigação de prova é apresentada uma breve descrição da mesma e ainda o provador ou tática que pode ser utilizado para sua prova. A ferramenta utiliza um padrão pré-definido para rotular as obrigações de prova (DEPLOY, 2011). Nesse trabalho aparecem obrigações com os seguintes rótulos: event/invariantlabel/INV, que garante a preservação da invariante *label* para o evento *event*; event/guardlabel/WD, que assegura que a condição de guarda *label* para o evento *event* está bem definido; e event/actionlabel/WD, que garante que a ação *label* do evento *event* está bem definido.

Tabela 1: Obrigações de Prova GG em Event-B

Identificação	**Descrição**	**Provador / Tática**
INITIALISATION/inv_vertG/INV	Garante que a variável $vertG$, no estado inicial (na sua inicialização), preserva o tipo (conjunto de Naturais)[1].	ML ou P0
INITIALISATION/inv_edgeG/INV	Garante que a variável $edgeG$, no estado inicial, preserva o tipo (conjunto de Naturais).	ML ou P0
INITIALISATION/inv_srcGtype/INV	Garante que a variável $sourceG$, no estado inicial, preserva o tipo (função total com domínio em $edgeG$ e contradomínio em $vertG$).	P0
INITIALISATION/inv_tgtGtype/INV	Garante que a variável $targetG$, no estado inicial, preserva o tipo (função total com domínio em $edgeG$ e contradomínio em $vertG$).	P0
INITIALISATION/inv_tG_V/INV	Garante que a variável tG_V, no estado inicial, preserva o tipo (função total com domínio em $vertG$ e contradomínio em $vertT$).	ML ou P0 ou NewPP

[1]Considera-se que essa invariante é genérica para qualquer GG uma vez que grafos na abordagem algébrica de GG são únicos a menos de isomorfismo, isto é, os nomes concretos dos seus vértices e arcos não são relevantes

INITIALISATION/inv_tG_E/INV	Garante que a variável tG_E, no estado inicial, preserva o tipo (função total com domínio em $edgeG$ e contradomínio em $edgeT$).	P0
rulei/grd_vertices/WD	Garante que a condição `grd_vertices` está bem definida, i.e., que $v \in dom(tLi\_V)$, $v \in dom(mV)$, $mV(v) \in dom(tG\_V)$, com tLi_V, tG_V e mV preservando seus tipos.	ML ou P1 / post tactics
rulei/grd_edges/WD	Garante que a condição `grd_edges` está bem definida, i.e., que $e \in dom(tLi\_E)$, $e \in dom(mE)$, $mE(e) \in dom(tG\_E)$, com tLi_E, tG_E e mE preservando seus tipos.	ML ou P1 ou NewPP / post tactics
rulei/grd_srctgt/WD	Garante que a condição `grd_srctgt` está bem definida, i.e., que $e \in dom(sourceLi)$, $sourceLi(e) \in dom(mV)$, $e \in dom(mE)$, $mE(e) \in dom(sourceG)$, $e \in dom(targetLi)$, $targetLi(e) \in dom(mV)$, $mE(e) \in dom(targetG)$, com $sourceLi$, mV, mE, $sourceG$, $targetLi$ e $targetG$ preservando seus tipos.	ML ou P1 / post tactics
rulei/inv_vertG/INV	Garante que a variável $vertG$, quando modificada pela aplicação da regra, preserva o tipo (conjunto de Naturais).	ML ou P0
rulei/inv_edgeG/INV	Garante que a variável $edgeG$, quando modificada pela aplicação da regra, preserva o tipo (conjunto de Naturais).	ML ou P0
rulei/inv_srcGtype/INV	Garante que a variável $sourceG$, quando modificada pela aplicação da regra, preserva o tipo (função total com domínio em $edgeG$ e contradomínio em $vertG$).	P0

rulei/inv_tgtGtype/INV	Garante que a variável $targetG$, quando modificada pela aplicação da regra, preserva o tipo (função total com domínio em $edgeG$ e contradomínio em $vertG$).	P0
rulei/inv_tG_V/INV	Garante que a variável tG_V, quando modificada pela aplicação da regra, preserva o tipo (função total com domínio em $vertG$ e contradomínio em $vertT$).	ML ou P0 ou NewPP
rulei/inv_tG_E/INV	Garante que a variável tG_E, quando modificada pela aplicação da regra, preserva o tipo (função total com domínio em $edgeG$ e contradomínio em $edgeT$).	P0
rulei/act_E/WD	Garante que a modificação da variável $edgeG$ está bem definida. Quando um arco e é deletado, garante que e pertence ao domínio da componente mE do *match* (componente que mapeia arcos) e que mE preserva o tipo. Isto é, garante que $e \in dom(mE)$ e $mE : edgeLi \rightarrowtail edgeG$.	Auto ou ML ou P1 ou NewPP / post tactics

rulei/act_src/WD	Garante que a modificação da variável $sourceG$ está bem definida. Quando um arco e é deletado, garante que e pertence ao domínio da componente mE do *match* (componente que mapeia arcos) e que mE preserva o tipo. Isto é, garante que $e \in dom(mE)$ e $mE : edgeLi \rightarrowtail edgeG$. Quando um arco é criado com origem (ou destino) em um vértice v preservado pela regra, garante que v pertence ao domínio da componente mV do *match* (componente que mapeia vértices) e que mV preserva o tipo. Isto é, garante que $v \in dom(mV)$ e $mV : vertLi \rightarrow vertG$.	Auto ou ML ou P1 ou NewPP / post tactics
rulei/act_tgt/WD	Garante que a modificação da variável $targetG$ está bem definida. Quando um arco e é deletado, garante que e pertence ao domínio da componente mE do *match* (componente que mapeia arcos) e que mE preserva o tipo. Isto é, garante que $e \in dom(mE)$ e $mE : edgeLi \rightarrowtail edgeG$. Quando um arco é criado com origem (ou destino) em um vértice v preservado pela regra, garante que v pertence ao domínio da componente mV do *match* (componente que mapeia vértices) e que mV preserva o tipo. Isto é, garante que $v \in dom(mV)$ e $mV : vertLi \rightarrow vertG$.	Auto ou ML ou P1 ou NewPP / post tactics

rulei/act_tE/WD	Garante que a modificação da variável tG_E está bem definida. Quando um arco e é deletado, garante que e pertence ao domínio da componente mE do *match* (componente que mapeia arcos) e que mE preserva o tipo. Isto é, garante que $e \in dom(mE)$ e $mE : edgeLi \rightarrowtail edgeG$.	Auto ou ML ou P1 ou NewPP / post tactics

Obrigações de prova identificadas com INITIALISATION garantem que as variáveis que definem o grafo-estado, quando inicializadas, preservem seus tipos. O valor inicial das variáveis descrevem o grafo inicial de uma gramática de grafos. Por exemplo, no sistema *token ring*, a obrigação INITIALISATION/inv_vertG/INV é utilizada para assegurar que $\{1, 2, 3\} \in \mathbb{P}(\mathbb{N})$ (veja Seção 2 Figura 16). De forma similar, obrigações de prova são geradas a fim de garantir a preservação dos tipos das outras variáveis ($edgeG$, $sourceG$, $targetG$, tG_V e tG_E) quando inicializadas. É possível observar que todas essas obrigações podem ser demonstradas executando apenas o provador P0.

Outras obrigações de prova asseguram que as condições de guarda de um determinado evento (ou regra) estão bem definidas, tais obrigações estão identificadas na Tabela 1 com o rótulo rulei/grd. Por exemplo, no *token ring*, a obrigação rulei/grd_vertices/WD assegura que a condição de guarda `grd_vertices` (veja Seção 2 Figura 19) está bem definida. Desta forma $v \in dom(tLi\_V)$, $v \in dom(mV)$ e $mV(v) \in dom(tG\_V)$, com tLi_V , tG_V e mV preservando seus tipos. Nesse caso $tLi\_V \in vertLi \rightarrow vertT$, $tG\_V \in vertG \rightarrow vertT$ e $mV \in vertLi \rightarrow vertG$.

Uma regra pode criar arcos, deletar arcos ou criar vértices no grafo-estado. Essa ação resulta na alteração do valor das variáveis correspondentes. Assim, obrigações de prova também são geradas para garantir que os novos valores das variáveis preservem seu tipo. Estas obrigações seguem o seguinte prefixo rulei/inv. No exemplo *token ring*, a regra 1 deleta um arco do tipo Stb e cria dois arcos, um do tipo Act e outro do tipo Msg (observe Seção 2 Figura 19). Nesse caso, as variáveis $edgeG$, $sourceG$, $targetG$ e tG_E são modificadas, então são geradas obrigações de prova para assegurar que as variáveis, depois de atualizadas, preservem seu tipo. P. ex. na regra 1, considerando a variável $edgeG$, deve ser garantido que $(edgeG \setminus \{mE(Stb11)\}) \cup \{newEmsg, newEact\} \in \mathbb{P}(\mathbb{N})$.

Obrigações também são geradas para assegurar que as ações (as quais definem o resultado da aplicação de uma regra) estão bem definidas. Estas obrigações são criadas com o rótulo rule/act. Quando uma regra deleta um arco e, as variáveis $edge$, $sourceG$, $targetG$ e tG_E são modificadas. Em tal caso, o arco que foi deletado no

grafo-estado é a imagem de e pelo componente mE do *match*, isto é, $mE(e)$ é deletado em $edgeG$. Da mesma forma são excluídos os elementos (pares) das funções $sourceG$, $targetG$ e tG_E que possuam $mE(e)$ como primeiro componente. Para as respectivas ações estarem bem definidas, o arco e deve pertencer ao domínio de mE e mE deve preservar seu tipo (estas são as obrigações de prova rule/act_E/WD, rule/act_src/WD, rule/act_tgt/WD, rule/act_tE/WD). Além disso quando uma regra adiciona um arco com origem (ou destino) em um vértice v, que é preservado pela regra, a origem (respectivamente destino) do arco adicionado deve ser imagem de v pelo componente mV do *match*. Nesse caso, para as variáveis $sourceG$ e $targetG$ estarem bem definidas, v deve pertencer ao domínio de mV, com mV preservando seu tipo (são as obrigações de prova rule/act_src/WD e rule/act_tgt/WD). Na Tabela 1 é observado que essas obrigações de prova são demonstradas automaticamente pela ferramenta.

As obrigações de prova descritas acima são geradas a partir da especificação de uma GG em Event-B. Segundo a abordagem (RIBEIRO et al., 2010), qualquer outra propriedade do sistema, deve ser declarada como uma invariante, assim se assume que ela será válida em todos os estado alcançáveis do sistema. As provas para tais propriedades são realizadas utilizando a indução: no caso base, uma obrigação de prova é gerada para garantir a preservação da propriedade para o grafo inicial e, no passo da indução, uma obrigação de prova é gerada para o grafo resultante da aplicação de uma regra da gramática. Em geral, a demonstração dessas obrigações de prova requer a intervenção do usuário, que deve ter o conhecimento tanto da ferramenta, quanto do sistema especificado. Apresentar algumas propriedades específicas para este formalismo, bem como as estratégias de prova para sua demonstração são os objetivos do próximo capítulo.

4 TÁTICAS DE PROVA PARA PROPRIEDADES SOBRE ESTADOS DE UMA GG

A tradução de GG em estruturas Event-B permitiu o uso da lógica de primeira ordem para expressar propriedades sobre estados alcançáveis de uma gramática de grafos. No trabalho (COSTA, 2010) foi possível observar que apesar do sistema descrito em GG ser intuitivo, a verificação de propriedades não é trivial. Propriedades sobre os estados são propriedades sobre os grafos, que geralmente são compostos por diferentes tipos de vértices e arcos. Em trabalhos anteriores (COSTA CAVALHEIRO; FOSS; RIBEIRO, 2012) foram propostos padrões para apresentação, codificação e reutilização destes tipos de propriedades. Neste Capítulo são apresentadas estratégias para demonstrar as obrigações de prova geradas por um subconjunto de propriedades pertencentes a este padrão, particularmente para as propriedades apresentadas na Figura 26. Os padrões previamente propostos são definidos a partir de uma biblioteca padrão de funções. Embora não sejam utilizadas essas funções nas propriedades descritas, as táticas aqui propostas priorizaram propriedades que descrevessem características típicas de grafos pré-definidas nesta biblioteca de funções. Além disso, optou-se por definir táticas para propriedades simples que pudessem ser reaproveitadas no caso de especificações mais complexas. Tais propriedades devem ser declaradas como invariantes no componente máquina. Importante também destacar que não foi feito um estudo sobre a otimização das táticas propostas. Isto é, nesse trabalho o foco é dado a definição da sequência de passos que demonstrem a obrigação de prova considerada, sem fazer uma análise posterior das possíveis simplificações.

A Figura 26 apresenta as propriedades declaradas como invariantes em sistemas de GG modelados em Event-B. Tais propriedades podem ser utilizadas para garantir alguma especificidade do sistema. A Tabela 2 apresenta o nome e a descrição de cada propriedade.

Para garantir a validade de uma propriedade, as obrigações de prova geradas pela ferramenta para a respectiva propriedade devem ser demonstradas. As propriedades são declaradas como invariantes, logo apresentam o rótulo INV (preservação da in-

INVARIANTS

propFin : $finite(tG\_E \rhd \{t\})$

propCard : $card(tG\_E \rhd \{t\}) = 1$

propExEdge : $\exists x \cdot x \in tG\_E \rhd \{t\}$

propExVert : $\exists x \cdot x \in tG\_V \rhd \{t\}$

propLoopT : $\exists x \cdot x \in edgeG \land sourceG(x) = targetG(x) \land tG\_E(x) = t$

propEdSpSrcT : $\exists x, y \cdot (x \in edgeG \land y \in vertG \land tG\_V(y) = t \land sourceG(x) = y)$

propEdSpTgtT : $\exists x, y \cdot (x \in edgeG \land y \in vertG \land tG\_V(y) = t \land targetG(x) = y)$

propNotIsoVT : $\forall x \cdot ((x \in vertG \land tG\_V(x) = t) \Rightarrow (\exists y \cdot (y \in edgeG \land ((sourceG(y) = x \land \neg targetG(y) = x) \lor (targetG(y) = x \land \neg sourceG(y) = x)))))$

propAllSVertSrcTEd : $\forall x \cdot ((x \in vertG \land tG\_V(x) = s) \Rightarrow (\exists y \cdot (y \in edgeG \land tG\_E(y) = t \land sourceG(y) = x)))$

propAllSVertTgtTEd : $\forall x \cdot ((x \in vertG \land tG\_V(x) = s) \Rightarrow (\exists y \cdot (y \in edgeG \land tG\_E(y) = t \land targetG(y) = x)))$

Figura 26: Propriedades Consideradas

Tabela 2: Descrição das Propriedade da Figura 26

Propriedade	Descrição
propFin	O conjunto de arcos do tipo t é finito.
propCard	O conjunto de arcos do tipo t é exatamente um.
propExEdge	Existe um arco do tipo t.
propExVert	Existe um vértice do tipo t.
propLoopT	Existe um arco loop do tipo t.
propEdSpSrcT	Existe um arco com origem em um vértice do tipo t.
propEdSpTgtT	Existe um arco com destino em um vértice do tipo t.
propNotIsoVT	Não existe um vértice isolado do tipo t.
propAllSVertSrcTEd	Todo vértice do tipo s é origem de um arco do tipo t.
propAllSVertTgtTEd	Todo vértice do tipo s é destino de um arco do tipo t.

variante). As obrigações de prova geradas para uma propriedade, seguem o mesmo padrão das obrigações geradas na especificação de uma GG em Event-B. Obrigações de prova são geradas para o evento de inicialização, garantindo que a propriedade é válida para o estado inicial e obrigações de prova são geradas para cada um dos demais eventos que atualizarem os valores das variáveis envolvidas definidas na propriedade, garantindo que a propriedade seja preservada pela aplicação das regras.

As demonstrações das obrigações de prova geradas por uma propriedade, em geral, envolvem interação do usuário. Além disso, muitas vezes, os passos necessários para a prova de uma obrigação gerada por um evento (regra), acabam se repetindo para os demais eventos (regras). Desta forma, são apresentadas estratégias/táticas de prova, que visam auxiliar o desenvolvedor no momento da verificação. As estratégias de demonstração das obrigações de prova geradas para as propriedades selecionadas (Figura 26), são apresentadas através dos passos necessários para sua

demonstração juntamente com a árvore de prova resultante. Tais estratégias são primeiramente apresentadas para o evento de inicialização (INITIALISATION/prop/INV) e após para as regras (rule_i/prop/INV). Considerando as regras, dependendo da propriedade, foi necessário apresentar subcasos de estratégias de prova. Ao abrir uma obrigação de prova, por *default*, a ferramenta aplica algumas regras como post ou auto tactis, gerando alguns nodos na árvore de prova. Para as táticas propostas, assume-se que o usuário sempre executa um *prune* (botão  do controle de prova) na árvore de prova, desfazendo as aplicações das táticas padrões e partindo do objetivo inicial da obrigação.

As árvores são apresentadas nas duas formas de visualização descritas no Capítulo 3. Uma delas mostra com detalhes os sequentes e regras que foram aplicadas. A outra, é uma árvore de uso, com atenção voltada apenas para a interação do usuário.

Nas árvores que mostram o sequente, mais precisamente no seu antecedente, se utiliza o símbolo H. Este símbolo é considerado como um conjunto de hipóteses selecionadas de forma automática pela ferramenta. Geralmente, quando uma tática ou regra é aplicada novas hipóteses são selecionadas, atualizando H. De forma a não tornar a notação muito carregada, optou-se por representar o conjunto de hipóteses de todos os sequentes por H. Também neste tipo de árvore, em virtude da limitação de espaço, nos sequentes podem aparecer letras. Estas letras, quando utilizadas, são apresentadas juntamente com seu significado na parte superior da figura da árvore de prova.

Por vezes, é necessário instanciar elementos em hipóteses geradas a partir de uma condição de guarda (prefixo `grd_`). A Tabela 3 apresenta estes identificadores juntamente com sua descrição.

Tabela 3: Descrição de Condições de Guarda

Identificador	Descrição
`grd_vertices`	$\forall v \cdot v \in vertLi \Rightarrow tLi\_V(v) = tG\_V(mV(v))$
`grd_edges`	$\forall e \cdot e \in edgeLi \Rightarrow tLi\_E(e) = tG\_E(mE(e))$
`grd_srctgt`	$\forall e \cdot e \in edgeLi \Rightarrow mV(sourceLi(e)) = sourceG(mE(e)) \wedge$ $\forall e \cdot e \in edgeLi \Rightarrow mV(targetLi(e)) = targetG(mE(e))$

Para `grd_srctgt` são geradas duas hipóteses distintas, uma considerando $source$ e outra $target$. No trabalho é informado em qual destas é necessário realizar a instanciação.

4.1 Propriedade propFin

A propriedade propFin ($finite(tG\_E \rhd \{t\})$) estabelece que o conjunto de arcos de tipo t em um grafo alcançável é finito. A propriedade propFin é necessária para a prova de qualquer propriedade sobre cardinalidade. Os passos para demonstrar a obrigação INITIALISATION/propFin/INV relativa ao evento de inicialização são os seguintes:

1. Adicionar como hipótese $tG\_E \rhd \{t\} = X$, substituindo tG_E pelo seu valor inicial e considerando X o resultado de tG_E restrito ao tipo t para o grafo inicial;
2. Executar o provador NewPP (with lasso);
3. Rodar o provador ML.

A Figura 27 apresenta a árvore de prova gerada para demonstrar essa obrigação.

$$\dfrac{\dfrac{\text{True}}{\vdash \top}\,2 \qquad \dfrac{\dfrac{\text{NewPP RULES}}{H \vdash tG\_E \rhd \{t\} = X}\,4}{\vdash tG\_E \rhd \{t\} = X}\,3 \qquad \dfrac{\text{ML RULES}}{H, tG\_E \rhd \{t\} = X \vdash finite(tG\_E \rhd \{t\})}\,5}{\vdash finite(tG\_E \rhd \{t\})}\,1$$

1 ah ($tG\_E \rhd \{t\} = X$); 2 $\top$ goal; 3 sl/ds; 4 NewPP (with lasso); 5 ML

Figura 27: Árvore de Prova INITIALISATION/propFin/INV

Descrição da Árvore de Prova: *O objetivo inicial a ser provado é* $\vdash finite(tG\_E \rhd \{t\})$*. A fim de demonstrar esse sequente adiciona-se* $tG\_E \rhd \{t\} = X$ *como hipótese (1), onde* X *é o resultado de* tG_E *restrito ao tipo* t*. Com essa ação três novos sub-objetivos são gerados: (i)* $\vdash \top$*, que é demonstrado automaticamente pela regra* $\top$ *goal (2); (ii)* $\vdash tG\_E \rhd \{t\} = X$ *que é descarregado aplicando o provador NewPP with lasso (4) (sl/ds (3) representa a operação de lasso realizada por NewPP with lasso); e (iii)* $H, tG\_E \rhd \{t\} = X \vdash finite(tG\_E \rhd \{t\})$*, que é comprovado pelo provador ML (5).*

A árvore de prova da Figura 28, apresenta a árvore de uso para demonstração da obrigação.

Descrição da Árvore de Uso: *A interação do usuário neste caso é adicionar* $tG\_E \rhd \{t\} = X$ *como hipótese, substituindo* tG_E *pelo seu valor inicial e considerando* X *o resultado da restrição. Nos próximos sub-objetivos executa-se o provador NewPP with lasso e ML, respectivamente.*

A fim de concluir a demonstração da propriedade PropFin, é necessário provar as obrigações de prova geradas para cada uma das regras que alterem o valor de tG_E.

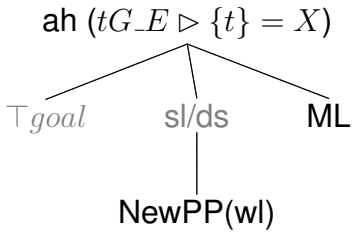


Figura 28: Árvore de Uso INITIALISATION/propFin/INV

Em geral uma regra pode deletar e criar novos arcos, então a obrigação de prova gerada segue este padrão $(\{mE(e_1), \ldots, mE(e_j)\} \lhd tG\_E) \cup \{ed_1 \mapsto t_1, \ldots, ed_k \mapsto t_k\} \rhd \{t\}$, onde j arcos são deletados e k pares de arcos com seus tipos são adicionados à variável tG_E. Para demonstrar essa obrigação aplicam-se as seguintes ações:

1. Aplicar a regra *Range Distribution Left Rewrites* no objetivo, com isso são aplicadas de forma automática algumas regras, gerando dois objetivos para se demonstrar: (i) $finite((\{mE(e_1), \ldots, mE(e_j)\} \lhd tG\_E) \rhd \{t\})$ e (ii) $finite(\{ed_1 \mapsto t_1, \ldots, ed_k \mapsto t_k\} \rhd \{t\})$;

2. Para provar (i), adicionar $(\{mE(e_1), \ldots, mE(e_j)\} \lhd tG\_E) \rhd \{t\} \subseteq tG\_E \rhd \{t\}$ como hipótese, e nos próximos sub-objetivos executar o provador ML;

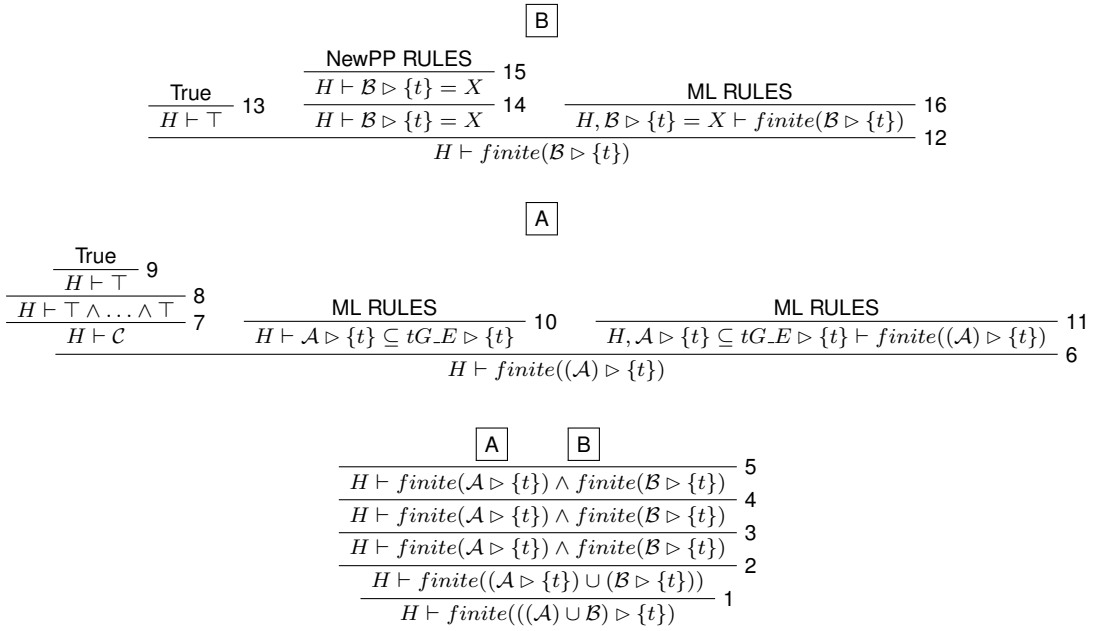
3. Para provar (ii), adicionar $\{ed_1 \mapsto t_1, \ldots, ed_k \mapsto t_k\} \rhd \{t\} = X$ como hipótese, considerando X o conjunto resultante da operação de restrição. Nos dois sub-objetivos gerados executar os provadores NewPP with lasso e ML, respectivamente.

A tática proposta é apresentada para o caso em que arcos são criados e deletados. No entanto, para regras que não criam ou não deletam arcos, a tática acaba se tornando mais simples. Para esses casos, pela árvore de prova é possível identificar qual foi o sub-objetivo particular gerado e a tática a ser aplicada deve ser a mesma proposta pela estratégia.

A Figura 29 apresenta a árvore de prova para demonstrar a obrigação rule_i/propFin/INV.

Descrição da Árvore de Prova para as Regras: *O objetivo inicial é* $H \vdash finite((((\{mE(e_1), \ldots, mE(e_j)\} \lhd tG\_E) \cup \{ed_1 \mapsto t_1, \ldots, ed_k \mapsto t_k\}) \rhd \{t\})$. *Primeiramente no objetivo aplica-se a regra* Range Ditribution Left Rewrites in Goal *(1), em seguida são aplicadas algumas regras de forma automática (regras 2-4 na árvore de prova). Após, a tática* $\wedge$ *goal (5) dividi o sequente em dois sub-objetivos: (I)* $finite((\{mE(e_1), \ldots, mE(e_j)\} \lhd tG\_E) \rhd \{t\})$ *e (II)* $finite(\{ed_1 \mapsto t_1, \ldots, ed_k \mapsto t_k\} \rhd \{t\})$. *A fim de descarregar (I), sub-árvore* $\boxed{A}$*, adiciona-se como hipótese* $(\{mE(e_1), \ldots, mE(e_j)\} \lhd tG\_E) \rhd \{t\} \subseteq tG\_E \rhd \{t\}$ *(6), com essa ação são gerados três*

$\mathcal{A}$ = $\{mE(e_1), \ldots, mE(e_j)\} \lhd tG\_E$
$\mathcal{B}$ = $\{ed_1 \mapsto t_1, \ldots, ed_k \mapsto t_k\}$
$\mathcal{C}$ = $e_1 \in dom(mE) \land \ldots \land e_j \in dom(mE) \land mE \in edgeLi \nrightarrow \mathbb{Z}$



1 Range Distribution Left Rewrites in Goal; 2 Simplification Rewrites; 3 Type Rewrites; 4 Simplification rewrites; 5 $\land$ goal; 6 ah ($\{mE(e_1), \ldots, mE(e_j)\} \lhd tG\_E \rhd \{t\} \subseteq tG\_E \rhd \{t\}$); 7 generalized MP; 8 simplification rewrites; 9 $\top$ goal; 10 ML; 11 ML; 12 ah ($\{ed_1 \mapsto t_1, \ldots, ed_k \mapsto t_k\} \rhd \{t\} = X$); 13 $\top$ goal; 14 sl/ds; 15 NewPP (with lasso); 16 ML

Figura 29: Árvore de Prova rule_i/propFin/INV

novos sub-objetivos: (i) $H \vdash e_1 \in dom(mE) \land \ldots \land e_j \in dom(mE) \land mE \in edgeLi \nrightarrow \mathbb{Z}$*, provado de forma automática (regras 7-9); (ii)* $H \vdash \{mE(e_1), \ldots, mE(e_j)\} \lhd tG\_E \rhd \{t\} \subseteq tG\_E \rhd \{t\}$ *e (iii)* $H, \{mE(e_1), \ldots, mE(e_j)\} \lhd tG\_E \rhd \{t\} \subseteq tG\_E \rhd \{t\} \vdash finite((\{mE(e_1), \ldots, mE(e_j)\} \lhd tG\_E) \rhd \{t\})$*, demonstrados pelo provador ML (regras 10 e 11). Para demonstrar (II), sub-árvore* B*, adiciona-se como hipótese* $\{ed_1 \mapsto t_1, \ldots, ed_k \mapsto t_k\} \rhd \{t\} = X$ *(12), onde* X *representa o resultado da operação de restrição. Essa instanciação, gera três novos sub-objetivos: (i)* $H \vdash \top$*, demonstrado automaticamente pela regra* $\top$ *goal (13); (ii)* $H \vdash \{ed_1 \mapsto t_1, \ldots, ed_k \mapsto t_k\} \rhd \{t\} = X$*, demonstrado pelo provador NewPP (with lasso) (15) e (iii)* $H, \{ed_1 \mapsto t_1, \ldots, ed_k \mapsto t_k\} \rhd \{t\} = X \vdash finite(\{ed_1 \mapsto t_1, \ldots, ed_k \mapsto t_k\} \rhd \{t\})$*, demonstrado executando o provador ML (16).*

A árvore de uso da estratégia de prova para demonstrar rule_i/propFin/INV é apresentada na Figura 30.

Descrição da Árvore de Uso: *A fim de utilizar essa estratégia, aplica-se a regra* Range Distribution Left Rewrites *no objetivo. Em seguida adiciona-se* $\{mE(e_1), \ldots, mE(e_j)\} \lhd tG\_E \rhd \{t\} \subseteq tG\_E \rhd \{t\}$ *como hipótese, nos dois sub-objetivos gerados executa-se o provador ML. Após adiciona-se* $\{ed_1 \mapsto t_1, \ldots, ed_k \mapsto t_k\} \rhd \{t\} =$

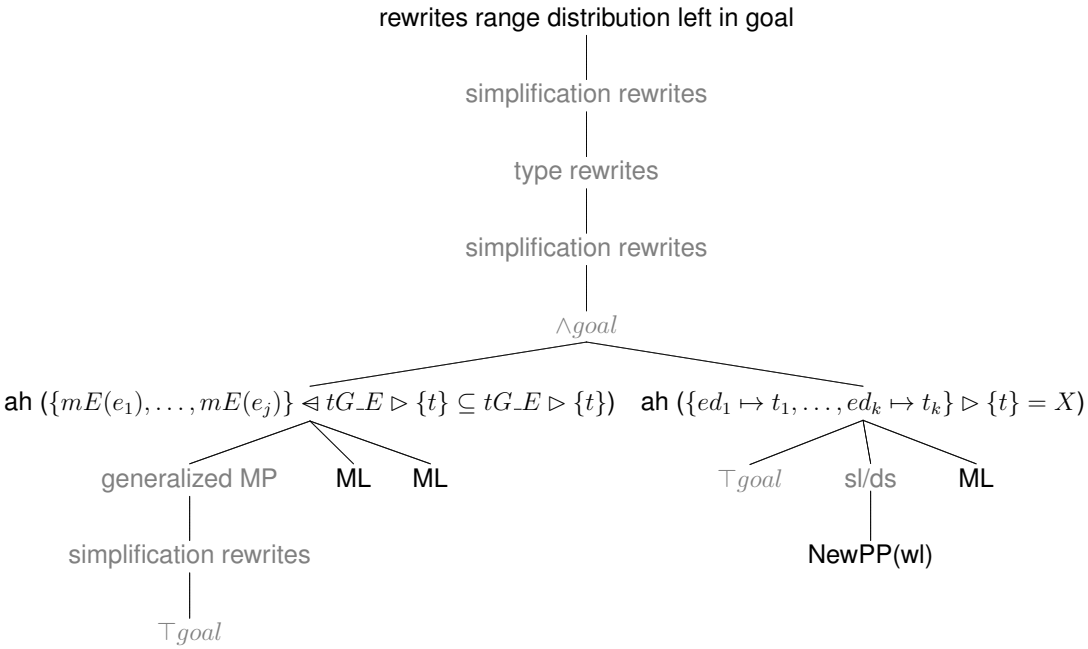


Figura 30: Árvore de Uso rule_i/propFin/INV

X como hipótese, onde X é o resultado da restrição. Nos sub-objetivos gerados executa-se o provador NewPP with lasso e ML, respectivamente.

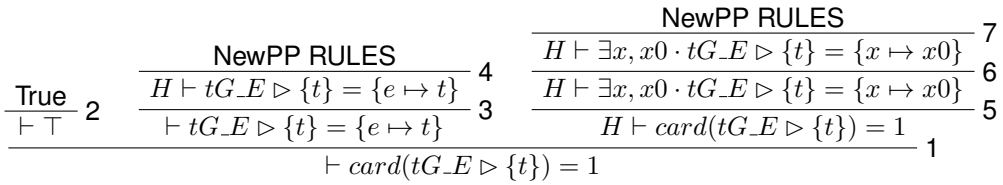
4.2 Propriedade `propCard`

A propriedade `propCard` ($card(tG\_E \rhd \{t\}) = 1$) especifica que a cardinalidade de arcos do tipo t no grafo-estado é igual a 1. Para o evento de inicialização, uma obrigação de prova é gerada substituindo tG_E por seu valor inicial. As etapas para demonstrar a obrigação INITIALISATION/propCard/INV, relativa ao evento de inicialização, são as seguintes:

1. Adicionar como hipótese $tG\_E \rhd \{t\} = \{e \mapsto t\}$, onde tG_E é substituído pelo seu valor inicial e $e \mapsto t$ é o par resultante de tG_E restrito ao tipo t;
2. Nos próximos sub-objetivos executar NewPP with lasso.

A Figura 31 apresenta a árvore de prova gerada na demonstração de INITIALISATION/propCard/INV.

Descrição da Árvore de Prova: *Visando demonstrar o objetivo* $\vdash card(tG\_E \rhd \{t\}) = 1$*, adiciona-se como hipótese* $tG\_E \rhd \{t\} = \{e \mapsto t\}$ *(1), onde* $e \mapsto t$ *é o par resultante de* tG_E *restrito ao tipo* t*. Após essa ação são gerados três novos sub-objetivos: (i)* $\vdash \top$*, descarregado de forma automática por* $\top$ *goal (2); (ii)* $\vdash tG\_E \rhd \{t\} = \{e \mapsto t\}$ *e (iii)* $H \vdash card(tG\_E \rhd \{t\}) = 1$*. O sequente (ii) é descarregado aplicando o provador NewPP*



1 ah($tG\_E \rhd \{t\} = \{e \mapsto t\}$); 2 $\top$ goal; 3 sl/ds; 4 NewPP (with lasso); 5 simplification rewrites; 6 sl/ds; 7 NewPP (with lasso)

Figura 31: Árvore de Prova INITIALISATION/propCard/INV

with lasso (4). Em (iii), primeiramente é aplicada a regra automática simplification rewrites (5), a qual transforma o objetivo em $H \vdash \exists x, x0 \cdot tG\_E \rhd \{t\} = \{x \mapsto x0\}$*. Este novo sequente é demonstrado executando o provador NewPP with lasso (7).*

Uma forma alternativa de apresentar a tática de demonstração para INITIALISATION/propCard/INV é através da árvore de uso, mostrada na Figura 32.

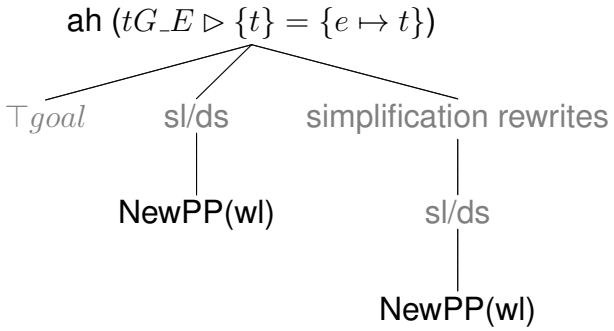


Figura 32: Árvore de Uso INITIALISATION/propCard/INV

Descrição da Árvore de Uso: *Para utilizar essa tática, adiciona-se* $tG\_E \rhd \{t\} = \{e \mapsto t\}$ *como hipótese, onde* $\{e \mapsto t\}$ *é o resultado da função* tG_E *restrita ao tipo* t*. Nos próximos sub-objetivos gerados executa-se o provador NewPP with lasso.*

Para as regras, `propCard` somente gera obrigações de prova para aquelas que atualizam o valor da variável tG_E. Como uma regra pode deletar, criar ou preservar arcos, se tem como obrigação geral para ser demonstrada $card((((\{mE(e_1), \ldots, mE(e_j)\} \lhd tG\_E) \cup \{ed_1 \mapsto t_1, \ldots, ed_k \mapsto t_k\}) \rhd \{t\}) = 1$, considerando que j arcos são deletados e k arcos são criados. Os possíveis casos de regra são: preserva o arco do tipo t; não envolve o arco do tipo t; e deleta e cria um arco do tipo t. Desta forma, as táticas são apresentadas para estes casos.

A - **Regra preserva o arco do tipo** t **ou não envolve o arco do tipo** t**.**

A tática para demonstrar rule_i/propCard/INV, nos casos em que um arco do tipo t

é preservado ou no caso em que a regra não envolve arcos do tipo t, segue os passos a seguir:

1. Aplicar as *post tactics default* da ferramenta;
2. Instanciar no objetivo os elementos x e $x0$ da hipótese de indução;
3. Executar o provador NewPP with lasso.

A Figura 33 apresenta a árvore de prova gerada para esses casos de obrigações.

$$\dfrac{\dfrac{\text{True}}{H \vdash \top}\,6 \qquad \dfrac{\dfrac{\text{NewPP RULES}}{H \vdash ((\{mE(e_1),\ldots,mE(e_j)\} \lhd tG\_E) \cup \{ed_1 \mapsto t_1,\ldots,ed_k \mapsto t_k\}) \rhd \{t\} = \{x \mapsto x0\}}\,8}{H \vdash ((\{mE(e_1),\ldots,mE(e_j)\} \lhd tG\_E) \cup \{ed_1 \mapsto t_1,\ldots,ed_k \mapsto t_k\}) \rhd \{t\} = \{x \mapsto x0\}}\,7}{\dfrac{H \vdash \exists x, x0 \cdot ((\{mE(e_1),\ldots,mE(e_j)\} \lhd tG\_E) \cup \{ed_1 \mapsto t_1,\ldots,ed_k \mapsto t_k\}) \rhd \{t\} = \{x \mapsto x0\}}{\dfrac{H \vdash \exists x, x0 \cdot ((\{mE(e_1),\ldots,mE(e_j)\} \lhd tG\_E) \cup \{ed_1 \mapsto t_1,\ldots,ed_k \mapsto t_k\}) \rhd \{t\} = \{x \mapsto x0\}}{\dfrac{H \vdash \exists x, x0 \cdot ((\{mE(e_1),\ldots,mE(e_j)\} \lhd tG\_E) \cup \{ed_1 \mapsto t_1,\ldots,ed_k \mapsto t_k\}) \rhd \{t\} = \{x \mapsto x0\}}{\dfrac{H \vdash \exists x, x0 \cdot ((\{mE(e_1),\ldots,mE(e_j)\} \lhd tG\_E) \cup \{ed_1 \mapsto t_1,\ldots,ed_k \mapsto t_k\}) \rhd \{t\} = \{x \mapsto x0\}}{H \vdash card(((\{mE(e_1),\ldots,mE(e_j)\} \lhd tG\_E) \cup \{ed_1 \mapsto t_1,\ldots,ed_k \mapsto t_k\}) \rhd \{t\}) = 1}\,1}\,2}\,3}\,4}\,5$$

1 simplification rewrites; 2 type rewrites; 3 simplification rewrites;
4 $\exists$ hyp ($\exists x, x0 \cdot tG\_E \rhd \{t\} = \{x \mapsto x0\}$); 5 $\exists$ goal (inst $x, x0$); 6 $\top$ goal; 7 sl/ds; 8 NewPP (with lasso)

Figura 33: Árvore de Prova rule_i/propCard/INV - Regra Preserva ou não Envolve o Arco do Tipo t

Descrição da Árvore de Prova para as Regras: *O objetivo dessa obrigação é* $H \vdash card(((\{mE(e_1),\ldots,mE(e_j)\} \lhd tG\_E) \cup \{ed_1 \mapsto t_1,\ldots,ed_k \mapsto t_k\}) \rhd \{t\}) = 1$. *A fim de demonstrar esse sequente, aplicam-se as* post tactics *(1) da ferramenta, após algumas regras são aplicadas de forma automática (regras 2-4), resultando no sequente* $H \vdash \exists x, x0 \cdot ((\{mE(e_1),\ldots,mE(e_j)\} \lhd tG\_E) \cup \{ed_1 \mapsto t_1,\ldots,ed_k \mapsto t_k\}) \rhd \{t\} = \{x \mapsto x0\}$. *Em seguida, instanciam-se os elementos* x *e* $x0$ *(da hipótese de indução) no objetivo (5), resultando em dois sub-objetivos para se demonstrar: (i)* $H \vdash \top$ *e (ii)* $H \vdash ((\{mE(e_1),\ldots,mE(e_j)\} \lhd tG\_E) \cup \{ed_1 \mapsto t_1,\ldots,ed_k \mapsto t_k\}) \rhd \{t\} = \{x \mapsto x0\}$. *(i) é descarregado automaticamente pela regra* $\top$ *goal (6). (ii) é descarregado executando o provador NewPP with lasso (8).*

Como forma alternativa de representar a tática para rule_i/propCard/INV, nos casos em que a regra preserva o arco do tipo t ou não envolve o arco do tipo t, se apresentada a árvore de uso na Figura 34.

Descrição da Árvore de Uso: *Primeiramente aplica-se a regra* post tactics *da ferramenta. Em seguida instanciam-se no objetivo os elementos* x *e* $x0$ *e ainda, executa-se o provador NewPP with lasso.*

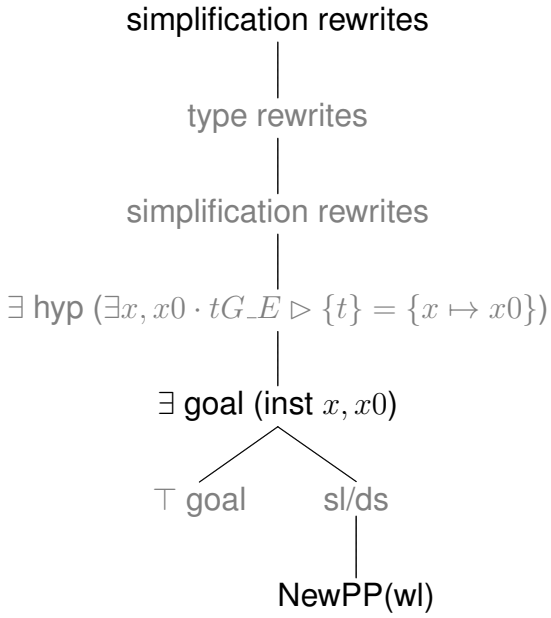


Figura 34: Árvore de Uso rule_i/propCard/INV - Regra Preserva ou não Envolve o Arco do Tipo t

B - Regra deleta e cria um arco do tipo t.

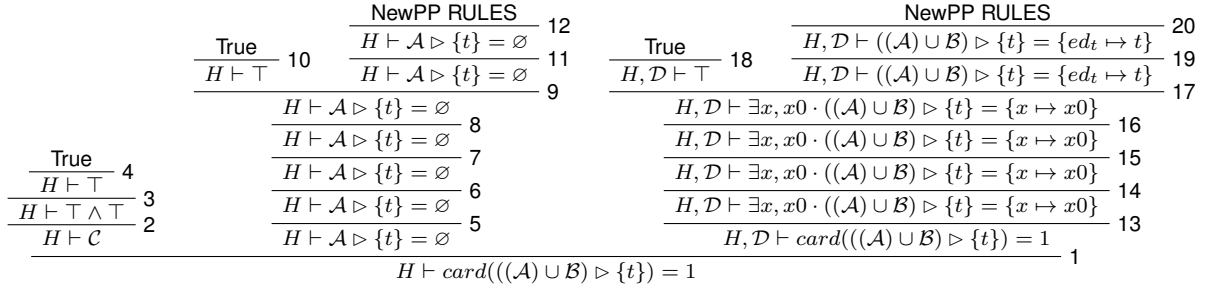
A fim de demonstrar rule_i/propCard/INV para os casos em que um arco do tipo t for deletado e criado pela regra, executam-se as seguintes ações:

1. Adicionar $\{mE(e_1), \ldots, mE(e_j)\} \lhd tG\_E \rhd \{t\} = \varnothing$ como hipótese;
2. Instanciar o nome do arco deletado e_j do tipo t na hipótese (`grd_edges`: $tLi\_E(e) = tG\_E(mE(e))$).
3. Executar o provador NewPP with lasso;
4. Instanciar no objetivo o nome do arco ed_t e seu tipo t criado pela regra;
5. Executar o provador NewPP with lasso.

A Figura 35 apresenta a árvore de prova para este caso.

Descrição da Árvore de Prova para as Regras: *O sequente inicial dessa obrigação é* $H \vdash card((((\{mE(e_1), \ldots, mE(e_j)\} \lhd tG\_E) \cup \{ed_1 \mapsto t_1, \ldots, ed_k \mapsto t_k\}) \rhd \{t\}) = 1$. *Visando sua demonstração, adiciona-se* $\{mE(e_1), \ldots, mE(e_j)\} \lhd tG\_E \rhd \{t\} = \varnothing$ *como hipótese (1), resultando em três novos sub-objetivos para se demonstrar: (I)* $H \vdash e_1 \in dom(mE) \land \ldots \land e_j \in dom(mE) \land mE \in edgeLi \nrightarrow \mathbb{Z}$*; (II)* $H \vdash \{mE(e_1), \ldots, mE(e_j)\} \lhd tG\_E \rhd \{t\} = \varnothing$*; e (III)* $H, \{mE(e_1), \ldots, mE(e_j)\} \lhd tG\_E \rhd \{t\} = \varnothing \vdash card((((\{mE(e_1), \ldots, mE(e_j)\} \lhd tG\_E) \cup \{ed_1 \mapsto t_1, \ldots, ed_k \mapsto t_k\}) \rhd \{t\}) = 1$. *O sequente (I) é demonstrado automaticamente por generalized MP, simplification rewrites*

$\mathcal{A}$ = $\{mE(e_1), \ldots, mE(e_j)\} \lhd tG\_E$
$\mathcal{B}$ = $\{ed_1 \mapsto t_1, \ldots, ed_k \mapsto t_k\}$
$\mathcal{C}$ = $e_1 \in dom(mE) \wedge \ldots \wedge e_j \in dom(mE) \wedge mE \in edgeLi \nrightarrow \mathbb{Z}$
$\mathcal{D}$ = $\{mE(e_1), \ldots, mE(e_j)\} \lhd tG\_E = \varnothing$



1 ah ($\{mE(e_1), \ldots, mE(e_j)\} \lhd tG\_E \rhd \{t\} = \varnothing$); 2 generalized MP; 3 simplification rewrites; 4 $\top$ goal; 5 simplification rewrites; 6 type rewrites; 7 simplification rewrites; 8 $\exists$ hyp ($\exists x, x0 \cdot tG\_E \rhd \{t\} = \{x \mapsto x0\}$; 9 $\forall$ hyp (inst $mE(e_j)$); 10 $\top$ goal; 11 sl/ds; 12 NewPP (with lasso); 13 simplification rewrites; 14 type rewrites; 15 simplification rewrites; 16 $\exists$ hyp ($\exists x, x0 \cdot tG\_E \rhd \{t\} = \{x \mapsto x0\}$); 17 $\exists$ goal (inst ed_t, t); 18 $\top$ goal; 19 sl/ds; 20 NewPP (with lasso)

Figura 35: Árvore de Prova rule_i/propCard/INV - Regra Deleta e Cria um Arco do Tipo t

e $\top$ goal (regras 2-4). Já o sequente (II), após serem aplicadas algumas regras automáticas (regras 5-8), instancia-se, o nome do arco deletado e_j do tipo t na hipótese que define a tipagem dos arcos do lado esquerdo da regra (9). Tal instanciação, resulta em dois sub-objetivos: (i) $H \vdash \top$ e (ii) $H \vdash \{mE(e_1), \ldots, mE(e_j)\} \lhd tG\_E \rhd \{t\} = \varnothing$. (i) é demonstrado automaticamente (10) e (ii) é descarregado pelo provador NewPP with lasso (12). Em (III), são aplicadas de forma automática as regras simplification rewrites, type rewrites, simplification rewrites e $\exists$ hyp (regras 13-16), gerando um novo objetivo para se demonstrar $H, \{mE(e_1), \ldots, mE(e_j)\} \lhd tG\_E \rhd \{t\} = \varnothing \vdash \exists x, x0 \cdot ((\{mE(e_1), \ldots, mE(e_j)\} \lhd tG\_E) \cup \{ed_1 \mapsto t_1, \ldots, ed_k \mapsto t_k\}) \rhd \{t\} = \{x \mapsto x0\}$. Visando sua demonstração, instancia-se no objetivo o nome do arco criado ed_t e seu tipo t (17), com isso são gerados dois novos sub-objetivos: (i) $H, \{mE(e_1), \ldots, mE(e_j)\} \lhd tG\_E \rhd \{t\} = \varnothing \vdash \top$, demonstrado automaticamente (18) e (ii) $H, \{mE(e_1), \ldots, mE(e_j)\} \lhd tG\_E \rhd \{t\} = \varnothing \vdash ((\{mE(e_1), \ldots, mE(e_j)\} \lhd tG\_E) \cup \{ed_1 \mapsto t_1, \ldots, ed_k \mapsto t_k\}) \rhd \{t\} = \{ed_t \mapsto t\}$, o qual é descarregado executando o provador NewPP with lasso (20).

A árvore de uso para a demonstração de rule_i/propCard/INV, nos casos em que a regra deleta e cria um arco do tipo t, está ilustrada na Figura 36.

Descrição da Árvore de Uso: *Para utilizar essa tática adiciona-se como hipótese $\{mE(e_1), \ldots, mE(e_j)\} \lhd tG\_E \rhd \{t\} = \varnothing$. No primeiro sub-objetivo, instancia-se o nome do arco deletado e_j do tipo t na hipótese que define a tipagem do lado esquerdo da regra. Após essa instanciação executa-se o provador NewPP with lasso. No outro sub-objetivo adiciona-se no objetivo o nome do arco criado ed_t pela regra, juntamente com*

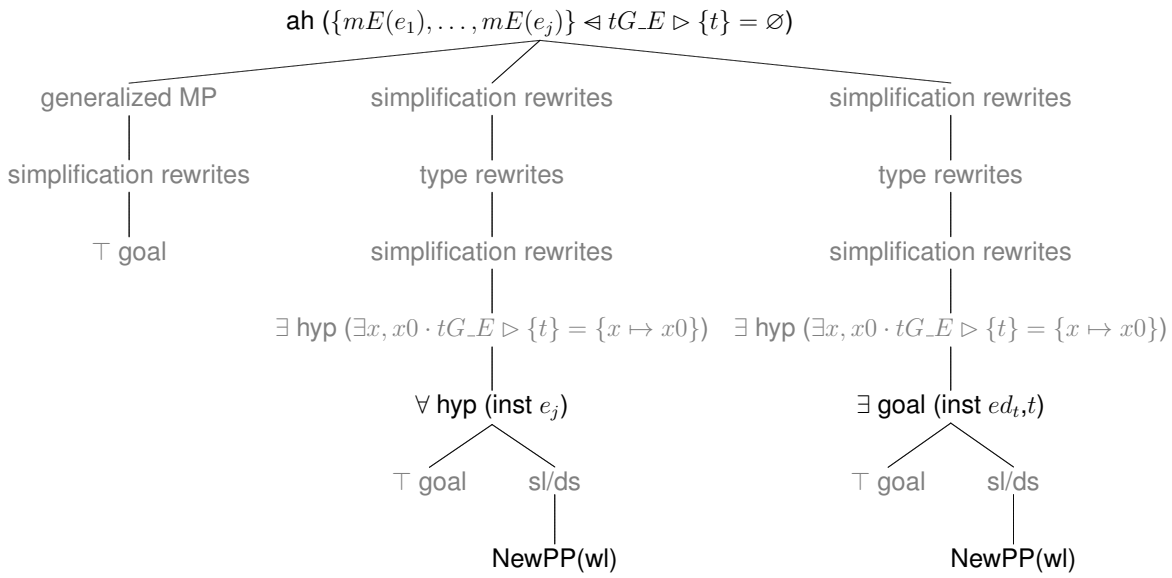


Figura 36: Árvore de Urso rule_i/propCard/INV - Regra Deleta e Cria um Arco do Tipo t

seu tipo t. Para concluir a demonstração executa-se o provador NewPP with lasso.

4.3 Propriedade propExEdge

A propriedade propExEdge ($\exists x \cdot x \in tG\_E \rhd \{t\}$) assegura que em todos os estados alcançáveis do sistema existe um arco do tipo t. Quando a variável tG_E for inicializada, é gerada uma obrigação de prova (INITIALISATION/propExEdge/INV) de forma a garantir a preservação dessa propriedade pelo grafo inicial. Visando demonstrar a obrigação gerada, executa-se o provador NewPP with lasso.

A Figura 37 apresenta a árvore de prova gerada pela demonstração de INITIALISATION/propExEdge/INV.

$$\dfrac{\dfrac{\text{NewPP RULES}}{H \vdash \exists x \cdot x \in tG\_E \rhd \{t\}}\,2}{\vdash \exists x \cdot x \in tG\_E \rhd \{t\}}\,1$$

1 sl/ds; 2 NewPP (with lasso)

Figura 37: Árvore de Prova INITIALISATION/propExEdge/INV

Descrição da Árvore de Prova: *O sequente inicial para se demonstrar é $\vdash \exists x \cdot x \in tG\_E \rhd \{t\}$. A fim de demonstrar esse sequente, executa-se o provador NewPP with lasso (2). A operação de lasso (sl/ds (1)), executada por NewPP with lasso, selecionou um conjunto de hipóteses H, as quais são consideradas no sequente demonstrado.*

A árvore de uso para essa obrigação é visualizada na Figura 38.

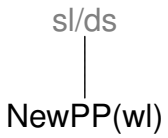


Figura 38: Árvore de Uso INITIALISATION/propExEdge/INV

Descrição da Árvore de Uso: *Para utilizar essa tática executa-se o provador NewPP with lasso.*

Além disso, para cada regra que atualizar o valor de tG_E é gerada uma obrigação de prova. Uma regra pode deletar e criar novos arcos, assim as obrigações de prova seguem este formato $\exists x \cdot x \in ((\{mE(e_1), \ldots, mE(e_j)\} \lhd tG\_E) \cup \{ed_1 \mapsto t_1, \ldots, ed_k \mapsto t_k\}) \rhd \{t\}$, onde j arcos são deletados e k arcos são criados pela regra. A definição de táticas para essa propriedade foi dividida nos seguintes casos: regra cria arcos do tipo t; regra preserva, não deleta e não cria arcos do tipo t; regra preserva, deleta e não cria arcos do tipo t; e regra não envolve arcos do tipo t.

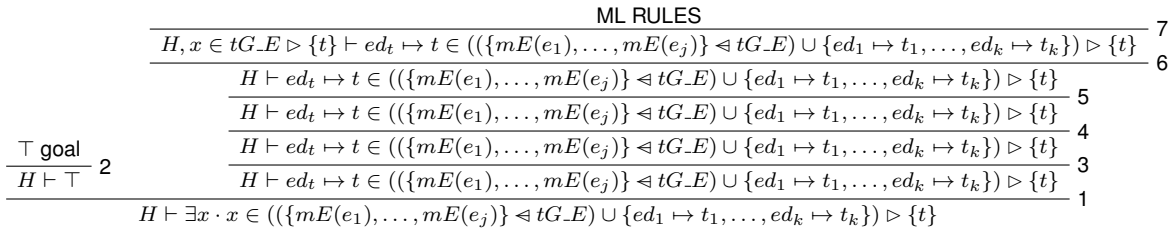
O caso em que a regra não preserva, deleta e não cria arcos do tipo t não é abordado nesse trabalho. Para esse caso não é possível definir uma tática genérica, pois a propriedade dependeria de outras propriedades do sistema ou mesmo de relação entre as regras. Particularmente, para esse caso, a existência de um arco com uma propriedade específica somente seria garantida se existissem outra(s) regra(s) da(s) qual(is) essa fosse dependente e que, em conjunto, garantissem a existência do arco. Pelo mesmo motivo esse caso não é considerado em propriedades subsequentes nesse trabalho.

A - Regra cria um arco do tipo t

Os passos para demonstrar rule_i/propExEdge/INV, para a regra que cria um arco do tipo t, são mostrados a seguir:

1. Instanciar no objetivo o par $ed_t \mapsto t$, onde ed_t é um arco do tipo t, criado pela regra;
2. Executar o provador ML.

A Figura 39 apresenta a árvore de prova gerada.



1 ∃ goal (inst $ed_t \mapsto t$); 2 $\top$ goal; 3 simplification rewrites; 4 type rewrites; 5 simplification rewrites; 6 ∃ hyp ($\exists x \cdot x \in tG\_E \rhd \{t\}$); 7 ML

Figura 39: Árvore de Prova rule_i/propExEdge/INV - Regra Cria um Arco do Tipo t

Descrição da Árvore de Prova para as Regras: *O objetivo inicial a ser demonstrado é* $H \vdash \exists x \cdot x \in ((\{mE(e_1), \ldots, mE(e_j)\} \lhd tG\_E) \cup \{ed_1 \mapsto t_1, \ldots, ed_k \mapsto t_k\}) \rhd \{t\}$*. Visando a prova desse sequente instancia-se no objetivo o par criado pela regra (1)* $ed_t \mapsto t$ *(de tipo* t*), o que resulta em dois novos sub-objetivos: (i)* $H \vdash \top$*, descarregado automaticamente (2) e (ii)* $H \vdash ed_t \mapsto t \in ((\{mE(e_1), \ldots, mE(e_j)\} \lhd tG\_E) \cup \{ed_1 \mapsto t_1, \ldots, ed_k \mapsto t_k\}) \rhd \{t\}$*. Após essa instanciação, a ferramenta aplica em (ii) uma sequência de regras de forma automática (regras 3-6), chegando a um novo sequente* $H, x \in tG\_E \rhd \{t\} \vdash ed_t \mapsto t \in ((\{mE(e_1), \ldots, mE(e_j)\} \lhd tG\_E) \cup \{ed_1 \mapsto t_1, \ldots, ed_k \mapsto t_k\}) \rhd \{t\}$*. Visando demonstrar esse sequente executa-se o provador ML (7).*

A forma alternativa de representar a tática é mostrada na Figura 40, através da árvore de uso.

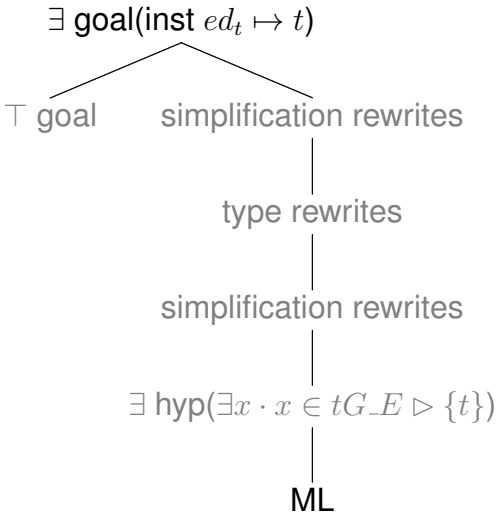


Figura 40: Árvore de Uso rule_i/propExEdge/INV - Regra Cria um Arco do Tipo t

Descrição da Árvore de Uso: *Para utilizar a estratégia instancia-se no objetivo o par* $ed_t \mapsto t$*, onde* ed_t *representa o arco criado pela regra e seu tipo* t*. Em seguida, executa-se o provador ML.*

B - Regra preserva, não deleta e não cria arcos do tipo t ou não envolve arcos do tipo t

Para demonstrar essa obrigação de prova executa-se o provador NewPP with lasso. A Figura 41 apresenta a árvore de prova gerada.

$$\dfrac{\dfrac{}{H \vdash \exists x \cdot x \in ((\{mE(e_1), \ldots, mE(e_j)\} \lhd tG\_E) \cup \{ed_1 \mapsto t_1, \ldots, ed_k \mapsto t_k\}) \rhd \{t\}}\ \text{NewPP RULES}}{H \vdash \exists x \cdot x \in ((\{mE(e_1), \ldots, mE(e_j)\} \lhd tG\_E) \cup \{ed_1 \mapsto t_1, \ldots, ed_k \mapsto t_k\}) \rhd \{t\}}\ \begin{matrix}2\\1\end{matrix}$$

1 sl/ds; 2 NewPP (with lasso)

Figura 41: Árvore de Prova rule_i/propExEdge/INV - Regra Preserva, não Deleta e não Cria Arcos do Tipo t ou não Envolve Arcos do Tipo t

Descrição da Árvore de Prova para as Regras: *O objetivo a ser demonstrado é* $H \vdash \exists x \cdot x \in ((\{mE(e_1), \ldots, mE(e_j)\} \lhd tG\_E) \cup \{ed_1 \mapsto t_1, \ldots, ed_k \mapsto t_k\}) \rhd \{t\}$*, sua demonstração é realizada executando o provador NewPP with lasso (2). NewPP realizou uma operação de lasso selecionando hipóteses através da regra sl/ds (1).*

A Figura 42 apresenta de forma alternativa a árvore de uso para a demonstração dessa obrigação.

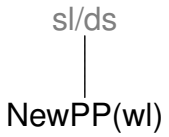


Figura 42: Árvore de Uso rule_i/propExEdge/INV - Regra Preserva, não Deleta e não Cria Arcos do Tipo t ou não Envolve Arcos do Tipo t

Descrição da Árvore de Uso: *Para utilizar essa estratégia executa-se o provador NewPP with lasso.*

C - Regra preserva, deleta e não cria arcos do tipo t

Nesse caso, a demonstração da obrigação de prova é realizada executando os seguintes passos:

1. Instanciar no objetivo o par $mE(e_j) \mapsto t$, onde e_j representa um arco do tipo t preservado pela regra;
2. Executar NewPP with lasso.

A Figura 43 apresenta a árvore de prova resultante da demonstração.

$\mathcal{A} = \{mE(e_1), \ldots, mE(e_j)\} \lhd tG\_E$
$\mathcal{B} = \{ed_1 \mapsto t_1, \ldots, ed_k \mapsto t_k\}$

$$
\dfrac{
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\text{True}}{H \vdash \top}\,6}{H \vdash e_j \in edgeLi}\,5}{H \vdash e_j \in dom(mE)}\,4}{H \vdash e_j \in dom(mE) \wedge \top}\,3}{H \vdash e_j \in dom(mE) \wedge mE \in edgeL1 \nrightarrow \mathbb{Z}}\,2
\qquad
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\text{NewPP RULES}}{H, x \in tG\_E \rhd \{t\} \vdash mE(e_j) \mapsto t \in (\mathcal{A}) \cup \mathcal{B} \rhd \{t\}}\,12}{H, x \in tG\_E \rhd \{t\} \vdash mE(e_j) \mapsto t \in (\mathcal{A}) \cup \mathcal{B} \rhd \{t\}}\,11}{H \vdash mE(e_j) \mapsto t \in (\mathcal{A}) \cup \mathcal{B} \rhd \{t\}}\,10}{H \vdash mE(e_j) \mapsto t \in (\mathcal{A}) \cup \mathcal{B} \rhd \{t\}}\,9}{H \vdash mE(e_j) \mapsto t \in (\mathcal{A}) \cup \mathcal{B} \rhd \{t\}}\,8}{H \vdash mE(e_j) \mapsto t \in (\mathcal{A}) \cup \mathcal{B} \rhd \{t\}}\,7
}{H \vdash \exists x \cdot x \in (\mathcal{A}) \cup \mathcal{B} \rhd \{t\}}\,1
$$

1 $\exists$ goal (inst $mE(e_j) \mapsto t$); 2 generalized MP; 3 simplification rewrites; 4 total function dom substitution in goal; 5 type rewrites; 6 $\top$ goal; 7 simplification rewrites; 8 type rewrites; 9 simplification rewrites; 10 $\exists$ hyp ($\exists x \cdot x \in tG\_E \rhd \{t\}$); 11 sl/ds; 12 NewPP (with lasso)

Figura 43: Árvore de Prova rule_i/propExEdge/INV - Regra Preserva, Deleta e não Cria Arcos do tipo t

Descrição da Árvore de Prova para as Regras: *O objetivo inicial é* $H \vdash \exists x \cdot x \in ((\{mE(e_1), \ldots, mE(e_j)\} \lhd tG\_E) \cup \{ed_1 \mapsto t_1, \ldots, ed_k \mapsto t_k\} \rhd \{t\}$*. Para demonstrar esse sequente, instancia-se o par* $mE(e_j) \mapsto t$*, onde* e_j *representa o arco do tipo* t *preservado pela regra (1). Essa ação resulta em dois sub-objetivos: (i)* $H \vdash e_j \in dom(mE) \wedge mE \in edgeL1 \nrightarrow \mathbb{Z}$ *e (ii)* $H \vdash mE(e_j) \mapsto t \in (\{mE(e_1), \ldots, mE(e_j)\} \lhd tG\_E) \cup \{ed_1 \mapsto t_1, \ldots, ed_k \mapsto t_k\} \rhd \{t\}$*. Em (i) são aplicadas algumas regras de forma automática (regras 2-5), chegando no objetivo* $H \vdash \top$*, que é descarregado automaticamente (6). Já em (ii) algumas regras são executadas de forma automática (regras 7-10). Para concluir a prova do objetivo executa-se o provador NewPP with lasso (12).*

A árvore de uso para esse caso é mostrada na Figura 44.

Descrição da Árvore de Uso: *Para utilizar essa tática, instancia-se no objetivo o par* $mE(e_j) \mapsto t$*, onde* e_j *representa o arco do tipo* t *preservado pela regra. Para concluir a demonstração executa-se o provador NewPP with lasso.*

4.4 Propriedade `propExVert`

Para garantir que no sistema exista um vértice de um determinado tipo t, se especifica a propriedade `propExVert` ($\exists x \cdot x \in tG\_V \rhd \{t\}$). Toda alteração na variável tG_V gera uma obrigação de prova a ser demonstrada. Para o evento de inicialização é gerada a obrigação INITIALISATION/propExVert/INV, e sua demonstração é realizada executando o provador P1. A Figura 45 apresenta a árvore de prova resultante desta

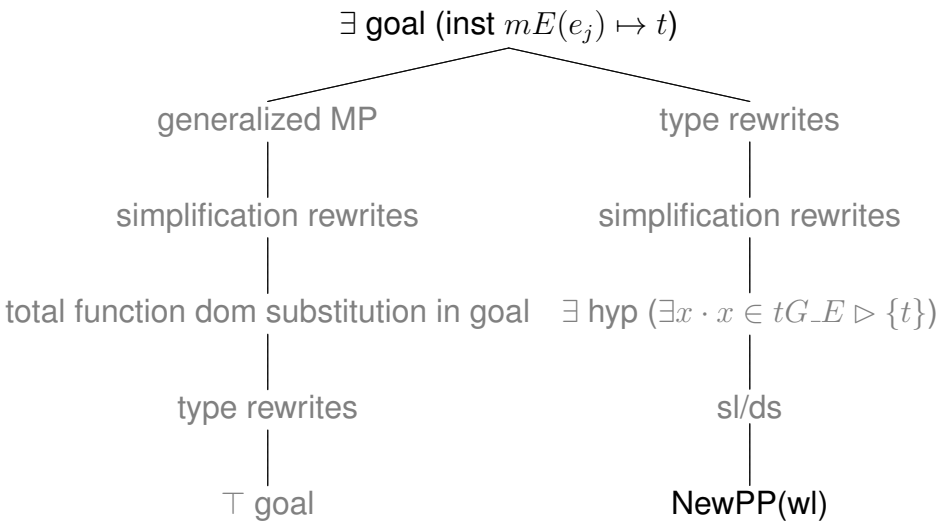


Figura 44: Árvore de Uso rule_i/propExEdge/INV - Regra Preserva, Deleta e não Cria Arcos do Tipo t

demonstração.

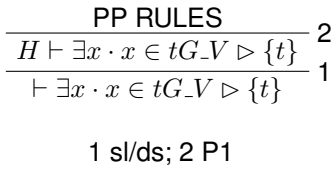


Figura 45: Árvore de Prova INITIALISATION/propExVert/INV

Descrição da Árvore de Prova: *O sequente inicial é* $\vdash \exists x \cdot x \in tG\_V \rhd \{t\}$*. Para demonstrar esse objetivo executa-se o provador P1 (2). P1 aplica uma operação de lasso (1), selecionando hipóteses para o novo sequente* $H \vdash \exists x \cdot x \in tG\_V \rhd \{t\}$*.*

A forma alternativa de apresentar a estratégia para demonstrar INITIALISATION/propExVert/INV, é mostrada através da árvore de uso da Figura 46.

Figura 46: Árvore de Uso INITIALISATION_i/propExVert/INV

Descrição da Árvore de Uso: *Para utilizar essa tática, aplica-se o provador P1.*

Essa abordagem considera que as regras não deletam vértices. Na propriedade `propExVert`, as obrigações de prova são geradas no seguinte formato $\exists x \cdot x \in tG\_V \cup \{v_1 \mapsto t_1, \ldots, v_l \mapsto t_l\} \rhd \{t\}$, considerando que l vértices podem ser criados pela

aplicação da regra. As possíveis ações realizadas por uma regra são: criar um vértice do tipo t; preservar e não criar um vértice do tipo t; e não envolver um vértice do tipo t. Deste modo, a apresentação das táticas são divididas nesses casos.

A - Regra cria um vértice do tipo t

A fim de demonstrar obrigações de prova desse formato executam-se os seguintes passos:

1. Instanciar no objetivo o par $v_t \mapsto t$, onde v_t é o vértice do tipo t criado pela regra;
2. Executar o provador ML.

A Figura 47 mostra a árvore de prova resultante na demonstração de rule_i/propExVertex/INV.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{H, x \in tG\_V \rhd \{t\} \vdash v_t \mapsto t \in (tG\_V \cup \{v_1 \mapsto t_1, \ldots, v_l \mapsto t_l\}) \rhd \{t\}}\,\text{ML RULES}\ 6}{H \vdash v_t \mapsto t \in (tG\_V \cup \{v_1 \mapsto t_1, \ldots, v_l \mapsto t_l\}) \rhd \{t\}}\ 5}{H \vdash v_t \mapsto t \in (tG\_V \cup \{v_1 \mapsto t_1, \ldots, v_l \mapsto t_l\}) \rhd \{t\}}\ 4}{H \vdash v_t \mapsto t \in (tG\_V \cup \{v_1 \mapsto t_1, \ldots, v_l \mapsto t_l\}) \rhd \{t\}}\ 3}{H \vdash \exists x \cdot x \in (tG\_V \cup \{v_1 \mapsto t_1, \ldots, v_l \mapsto t_l\}) \rhd \{t\}}\ 1 \qquad \dfrac{\text{True}}{H \vdash \top}\ 2$$

1 $\exists$ goal (inst $v_t \mapsto t$); 2 $\top$ goal; 3 type rewrites; 4 simplification rewrites; 5 $\exists$ hyp ($\exists x \cdot x \in tG\_V \rhd \{t\}$); 6 ML

Figura 47: Árvore de Prova rule_i/propExVert/INV - Regra Cria um Vértice do Tipo t

Descrição da Árvore de Prova para as Regras: *O sequente inicial para se demonstrar é $H \vdash \exists x \cdot x \in (tG\_V \cup \{v_1 \mapsto t_1, \ldots, v_l \mapsto t_l\}) \rhd \{t\}$. A fim de demonstrar esse sequente instancia-se no objetivo o par $v_t \mapsto t$ (1), onde v_t é o vértice do tipo t criado pela regra. Com essa ação são gerados dois novos objetivos: (i) $H \vdash \top$, demonstrado automaticamente (2); e (ii) $H \vdash v_t \mapsto t \in (tG\_V \cup \{v_1 \mapsto t_1, \ldots, v_l \mapsto t_l\}) \rhd \{t\}$. Em (ii) são executadas algumas regras de forma automática (regras 3-5), chegando no objetivo $H, x \in tG\_V \rhd \{t\} \vdash v_t \mapsto t \in (tG\_V \cup \{v_1 \mapsto t_1, \ldots, v_l \mapsto t_l\}) \rhd \{t\}$. Para demonstrar esse sequente executa-se o provador ML (6).*

A árvore de uso dessa obrigação é representada pela Figura 48.

Descrição da Árvore de Uso: *A fim de utilizar a tática, instancia-se o par $v_t \mapsto t$, onde v_t representa o vértice de tipo t criado pela regra. Em seguida, executa-se o provador ML.*

B - Regra preserva e não cria vértices do tipo t

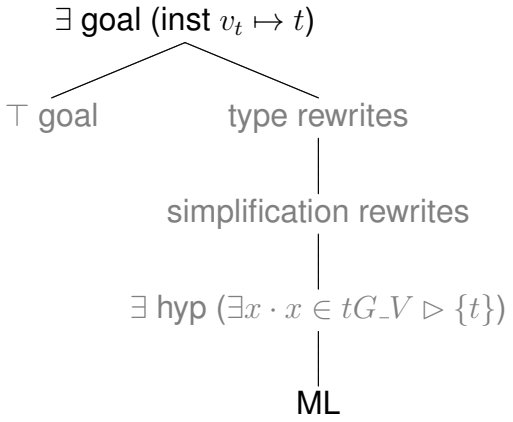
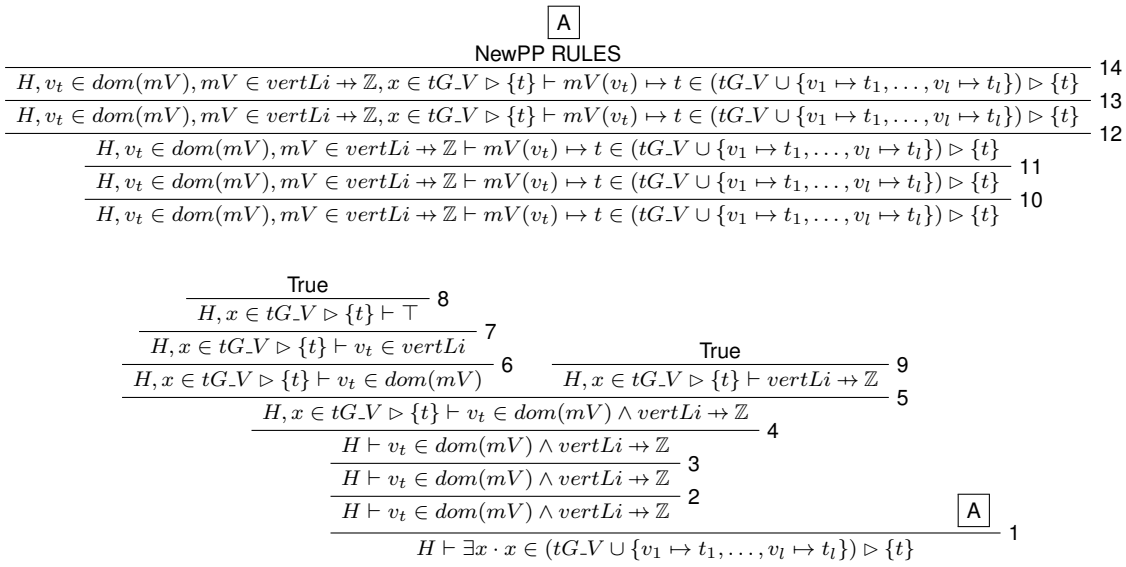


Figura 48: Árvore de Uso - rule_i/propExVert/INV - Regra Cria um Vértice do Tipo t

Visando demonstrar tais obrigações de prova, executam-se as seguintes ações:

1. Instanciar no objetivo o par $mV(v_t) \mapsto t$, onde v_t é o vértice do tipo t preservado pela regra;
2. Executar o provador NewPP with lasso.

A Figura 49 mostra a árvore de prova resultante da demonstração dessa obrigação.



∃ goal (inst $mV(v_t) \mapsto t$); 2 type rewrites; 3 simplification rewrites; 4 ∃ hyp ($\exists x \cdot x \in tG\_V \rhd \{t\}$); 5 ∧ goal; 6 total function dom substitution in goal; 7 type rewrites; 8 ⊤ goal; 9 functional goal; 10 type rewrites; 11 simplification rewrites; 12 ∃ hyp ($\exists x \cdot x \in tG\_V \rhd \{t\}$); 13 sl/ds; 14 NewPP with lasso

Figura 49: Árvore de Prova rule_i/propExVert/INV - Regra Preserva e não Cria Vértices do Tipo t

Descrição da Árvore de Prova para as Regras: *O sequente inicial para se demonstrar é* $H \vdash \exists x \cdot x \in (tG\_V \cup \{v_1 \mapsto t_1, \ldots, v_l \mapsto t_l\}) \rhd \{t\}$*. A fim de demonstrar esse sequente instancia-se no objetivo o par* $mV(v_t) \mapsto t$ *(1), onde* v_t *é o arco do tipo* t *preservado pela regra. Após essa instanciação, dois sub-objetivos são gerados: (i)* $H \vdash v_t \in dom(mV) \wedge vertLi \nrightarrow \mathbb{Z}$ *e (ii)* $H, v_t \in dom(mV), mV \in vertLi \nrightarrow \mathbb{Z} \vdash mV(v_t) \mapsto t \in (tG\_V\{v_1 \mapsto t_1, \ldots, v_l \mapsto t_l\}) \rhd \{t\}$*. O sequente (i) é descarregado automaticamente por uma sequência de regras (regras 2-9). Já em (ii), são aplicadas algumas regras de forma automática (regras 10-12), o que resulta em um novo objetivo* $H, v_t \in dom(mV), mV \in vertLi \nrightarrow \mathbb{Z}, x \in tG\_V \rhd \{t\} \vdash mV(v_t) \mapsto t \in (tG\_V\{v_1 \mapsto t_1, \ldots, v_l \mapsto t_l\}) \rhd \{t\}$*. A prova desse sequente é realizada executando o provador NewPP with lasso (14).*

A Figura 50 representa a árvore de uso para rule_i/propExVert/INV.

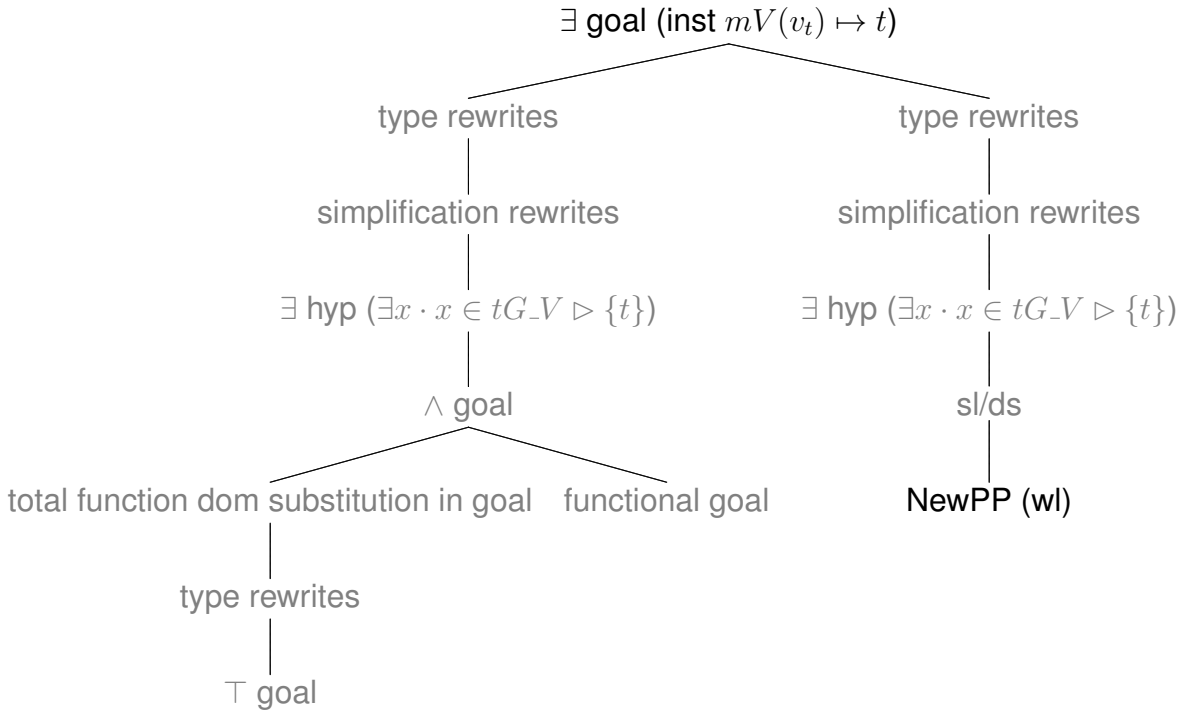


Figura 50: Árvore de Uso rule_i/propExVert/INV - Regra Preserva e não Cria Vértices do Tipo t

Descrição da Árvore de Uso: *Para utilizar tal estratégia instancia-se o par* $mV(v_t) \mapsto t$*, onde* v_t *é o nome do vértice do tipo* t*, preservado pela regra. Em seguida executa-se o provador NewPP with lasso.*

C - Regra não envolve vértices do tipo t

Para demonstrar essa obrigação executa-se o provador P1. A Figura 51 apresenta a árvore de prova gerada.

$$\dfrac{\dfrac{\text{PP RULES}}{H, x \in tG\_V \rhd \{t\} \vdash \exists x \cdot x \in (tG\_V \cup \{v_1 \mapsto t_1, \ldots, v_l \mapsto t_l\}) \rhd \{t\}}\,2}{H \vdash \exists x \cdot x \in (tG\_V \cup \{v_1 \mapsto t_1, \ldots, v_l \mapsto t_l\}) \rhd \{t\}}\,1$$

1 sl/ds; 2 P1

Figura 51: Árvore de Prova rule_i/propExVert/INV - Regra Não Envolve Vértices do Tipo t

Descrição da Árvore de Prova para as Regras: *O sequente para se demonstrar é* $H \vdash \exists x \cdot x \in (tG\_V \cup \{v_1 \mapsto t_1, \ldots, v_l \mapsto t_l\}) \rhd \{t\}$*, sua prova é realizada executando o provador P1 (2). A operação de lasso sl/ds (1) foi aplicada por P1 de forma automática gerando o objetivo* $H, x \in tG\_V \rhd \{t\} \vdash \exists x \cdot x \in (tG\_V \cup \{v_1 \mapsto t_1, \ldots, v_l \mapsto t_l\}) \rhd \{t\}$*.*

A árvore de uso para essa obrigação é apresentada na Figura 52.

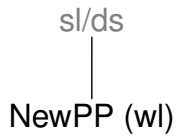


Figura 52: Árvore de Uso rule_i/propExVert/INV - Regra Não Envolve Vértices do Tipo t

Descrição da Árvore de Uso: *Para utilizar a estratégia executa-se o provador P1.*

4.5 Propriedade propLoopT

A propriedade propLoopT ($\exists x \cdot x \in edgeG \land sourceG(x) = targetG(x) \land tG\_E(x) = t$) estabelece que em qualquer grafo alcançável da gramática existe um arco loop de tipo t. Quando as variáveis $edgeG$, $sourceG$, $targetG$ e tG_E são iniciadas pelo evento de inicialização, uma obrigação de prova é gerada INITIALISATION/propLoopT/INV. Para demonstrar essa obrigação instancia-se no objetivo o nome do arco loop x do tipo t pertencente ao grafo inicial, onde $x \in \mathbb{N}$.

A Figura 53 apresenta a árvore de prova gerada pela demonstração.

$$\dfrac{\dfrac{\text{True}}{\vdash \top}\,2 \qquad \dfrac{\dfrac{\text{True}}{\vdash \top}\,4}{\vdash x \in edgeG \land sourceG(x) = targetG(x) \land tG\_E(x) = t}\,3}{\vdash \exists x \cdot x \in edgeG \land sourceG(x) = targetG(x) \land tG\_E(x) = t}\,1$$

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

Figura 53: Árvore de Prova INITIALISATION/propLoopT/INV

Descrição da Árvore de Prova: *O objetivo a ser demonstrado é* $\vdash \exists x \cdot x \in edgeG \wedge sourceG(x) = targetG(x) \wedge tG\_E(x) = t$*. A fim de descarregar esse sequente instancia-se no objetivo o nome do arco loop* x *do tipo* t *(1), pertencente ao grafo inicial da gramática. Após essa ação são gerados dois sub-objetivos: (i)* $\vdash \top$ *descarregado automaticamente (2) e (ii)* $\vdash x \in edgeG \wedge sourceG(x) = targetG(x) \wedge tG\_E(x) = t$*, que após executar a regra automática simplification rewrites (3), gera o objetivo* $\vdash \top$*, que é demonstrado de forma automática pela regra* $\top$ *goal (4).*

A Figura 54 apresenta a árvore de uso para essa obrigação.

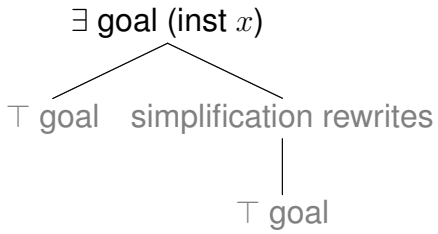


Figura 54: Árvore de Uso rule_i/propLoopT/INV

Descrição da Árvore de Uso: *Para utilizar tal estratégia, instancia-se no objetivo o nome do arco loop* x *de tipo* t *criado pela regra, pertencente ao grafo inicial.*

Considerando `propLoopT`, uma regra pode deletar e criar novos arcos, assim as obrigações de prova geradas para as regras seguem este padrão $\exists x \cdot x \in (edgeG \setminus \{mE(e_1), \ldots, mE(e_j)\}) \cup \{ed_1, \ldots ed_k\} \wedge (((\{mE(e_1), \ldots, mE(e_j)\} \lhd sourceG) \cup \{ed_1 \mapsto mV(v_1), \ldots, ed_k \mapsto v_l\})(x) = (((\{mE(e_1), \ldots, mE(e_j)\} \lhd targetG) \cup \{ed_1 \mapsto mV(v_1), \ldots, ed_k \mapsto v_l\})(x) \wedge (((\{mE(e_1), \ldots, mE(e_j)\} \lhd tG\_E) \cup \{ed_1 \mapsto t_1, \ldots, ed_k \mapsto t_k\})(x) = t$, considerando que j arcos são deletados e k arcos são criados pela regra. Uma regra se enquadra em uma das seguintes alternativas: cria arcos loop do tipo t; preserva, não deleta e não cria arcos loop do tipo t; preserva, deleta e não cria arcos loop do tipo t; não envolve arcos loop do tipo t; e não preserva, deleta e não cria arcos loop do tipo t. Desta forma, a apresentação das táticas são divididas nestes casos.

O caso em que a regra não preserva, deleta e não cria arcos loop do tipo t não é tratado neste trabalho.

A - Regra cria um arco loop do tipo t.

A fim de demonstrar este tipo de obrigação de prova instancia-se no objetivo o nome do arco loop ed_t do tipo t criado pela regra. A Figura 55 apresenta a árvore de prova gerada na demonstração.

$\mathcal{A}$ = $edgeG \setminus \{mE(e_1), \ldots, mE(e_j)\}$
$\mathcal{B}$ = $\{ed_1, \ldots, ed_k\}$
$\mathcal{C}$ = $\{mE(e_1), \ldots, mE(e_j)\} \lhd sourceG$
$\mathcal{D}$ = $\{ed_1 \mapsto mV(v_1), \ldots, ed_k \mapsto v_l\}$
$\mathcal{E}$ = $\{mE(e_1), \ldots, mE(e_j)\} \lhd targetG$
$\mathcal{F}$ = $\{ed_1 \mapsto mV(v_1), \ldots, ed_k \mapsto v_l\}$
$\mathcal{G}$ = $\{mE(e_1), \ldots, mE(e_j)\} \lhd tG\_E$
$\mathcal{H}$ = $\{ed_1 \mapsto t_1, \ldots, ed_k \mapsto t_k\}$

$$\dfrac{\dfrac{\text{True}}{H \vdash \top}\,2 \qquad \dfrac{\dfrac{\text{True}}{H \vdash \top}\,4}{H \vdash ed_t \in ((\mathcal{A}) \cup \mathcal{B}) \wedge ((\mathcal{C}) \cup \mathcal{D})(x) = ((\mathcal{E}) \cup \mathcal{F})(x) \wedge ((\mathcal{G}) \cup \mathcal{H})(x) = t}\,3}{H \vdash \exists x \cdot x \in ((\mathcal{A}) \cup \mathcal{B}) \wedge ((\mathcal{C}) \cup \mathcal{D})(x) = ((\mathcal{E}) \cup \mathcal{F})(x) \wedge ((\mathcal{G}) \cup \mathcal{H})(x) = t}\,1$$

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

Figura 55: Árvore de Prova rule_i/propLoopT/INV - Regra Cria um Arco Loop do Tipo t

Descrição da Árvore de Prova para as Regras: *O sequente a ser demonstrado é* $H \vdash \exists x \cdot x \in ((edgeG \setminus \{mE(e_1), \ldots, mE(e_j)\}) \cup \{ed_1, \ldots, ed_k\}) \wedge (((\{mE(e_1), \ldots, mE(e_j)\} \lhd sourceG) \cup \{ed_1 \mapsto mV(v_1), \ldots, ed_k \mapsto v_l\})(x) = (((\{mE(e_1), \ldots, mE(e_j)\} \lhd targetG) \cup \{ed_1 \mapsto mV(v_1), \ldots, ed_k \mapsto v_l\})(x) \wedge (((\{mE(e_1), \ldots, mE(e_j)\} \lhd tG\_E) \cup \{ed_1 \mapsto t_1, \ldots, ed_k \mapsto t_k\})(x) = t$. *A fim de demonstrar esse sequente, instancia-se no objetivo o nome do arco loop* ed_t*, de tipo* t*, criado pela aplicação da regra (1). Após são gerados dois sub-objetivos: (i)* $H \vdash \top$ *e (ii)* $H \vdash ed_t \in ((edgeG \setminus \{mE(e_1), \ldots, mE(e_j)\}) \cup \{ed_1, \ldots, ed_k\}) \wedge (((\{mE(e_1), \ldots, mE(e_j)\} \lhd sourceG) \cup \{ed_1 \mapsto mV(v_1), \ldots, ed_k \mapsto v_l\})(x) = (((\{mE(e_1), \ldots, mE(e_j)\} \lhd targetG) \cup \{ed_1 \mapsto mV(v_1), \ldots, ed_k \mapsto v_l\})(x) \wedge (((\{mE(e_1), \ldots, mE(e_j)\} \lhd tG\_E) \cup \{ed_1 \mapsto t_1, \ldots, ed_k \mapsto t_k\})(x) = t$. *(i) é descarregado de forma automática pela regra* $\top$ *goal (2). Em (ii), é aplicada de forma automática a regra simplification rewrites (3), gerando o sequente* $H \vdash \top$*, que é automaticamente demonstrado (4).*

A Figura 56 apresenta a árvore de uso para essa obrigação.

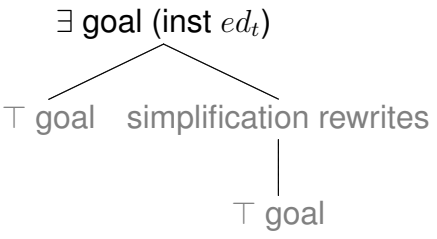


Figura 56: Árvore de Uso rule_i/propLoopT/INV - Regra Cria um Arco Loop do Tipo t

Descrição da Árvore de Uso: *Para utilizar tal estratégia, instancia-se no objetivo o nome do arco loop* ed_t *do tipo* t *criado pela regra.*

B - Regra preserva, não deleta e não cria arcos loop do tipo t ou a regra não envolve arcos loop do tipo t.

A demonstração das obrigações para estes dois casos são idênticas, assim uma única tática é apresentada. Para os casos em que nenhum outro arco (diferente do tipo t) for deletado, basta executar o provador NewPP with lasso. Já para os casos em que outros arcos são deletados, diferentes do tipo t, os passos necessários para demonstração são apresentados a seguir:

1. Aplicar a regra $\exists$ hyp na hipótese de indução;
2. Para cada arco e_i deletado pela regra, seguir a seguinte sequência de passos:
   (a) Adicionar como hipótese $\neg x = mE(e_i)$, onde e_i é um arco deletado pela regra diferente do tipo t;
   (b) Instanciar o arco deletado e_i na hipótese `grd_edges`;
   (c) Aplicar a regra partition rewrites na hipótese que define a tipagem dos arcos do lado esquerdo da regra e nos sub-objetivos gerados aplicar o provador NewPP with lasso.
3. Executar o provador NewPP with lasso.

Nessa demonstração, para cada arco deletado pela regra é necessário adicionar como hipótese que esse arco é diferente do elemento x. Após adicionar a diferenciação, a prova dos sub-objetivos gerados é realizada conforme mostrado na tática. Por exemplo, após instanciar o arco $\neg x = mE(e_i)$, os sub-objetivos gerados são demonstrados aplicando os passos 6 e 7. A Figura 57 apresenta a árvore de prova gerada na demonstração de rule_i/propLoopT/INV.

Descrição da Árvore de Prova para as Regras: *O primeiro objetivo para se demonstrar é* $H \vdash \exists x \cdot x \in (edgeG \setminus \{mE(e_1), \ldots, mE(e_j)\}) \cup \{ed_1, \ldots, ed_k\} \wedge (((\{mE(e_1), \ldots, mE(e_j)\} \lhd sourceG) \cup \{ed_1 \mapsto mV(v_1), \ldots, ed_k \mapsto v_l\})(x) = (((\{mE(e_1), \ldots, mE(e_j)\} \lhd targetG) \cup \{ed_1 \mapsto mV(v_1), \ldots, ed_k \mapsto v_l\})(x) \wedge (((\{mE(e_1), \ldots, mE(e_j)\} \lhd tG\_E) \cup \{ed_1 \mapsto t_1, \ldots, ed_k \mapsto t_k\})(x) = t$. *Visando demonstrar esse sequente aplica-se a regra* $\exists$ *hyp na hipótese de indução (1). Com isso são aplicadas algumas regras de forma automática (regras 2-5), gerando um novo objetivo para se demonstrar* $H \vdash \exists x0 \cdot x0 \in (edgeG \setminus \{mE(e_1), \ldots, mE(e_j)\}) \cup \{ed_1, \ldots, ed_k\} \wedge (((\{mE(e_1), \ldots, mE(e_j)\} \lhd sourceG) \cup \{ed_1 \mapsto mV(v_1), \ldots, ed_k \mapsto v_l\})(x0) = (((\{mE(e_1), \ldots, mE(e_j)\} \lhd targetG) \cup \{ed_1 \mapsto mV(v_1), \ldots, ed_k \mapsto v_l\})(x0) \wedge (((\{mE(e_1), \ldots, mE(e_j)\} \lhd tG\_E) \cup \{ed_1 \mapsto t_1, \ldots, ed_k \mapsto t_k\})(x0) = tG\_E(x)$. *Para demonstrar esse objetivo, adiciona-se como hipótese* $\neg x = mE(e_1)$ *(6), onde* e_1 *é o*

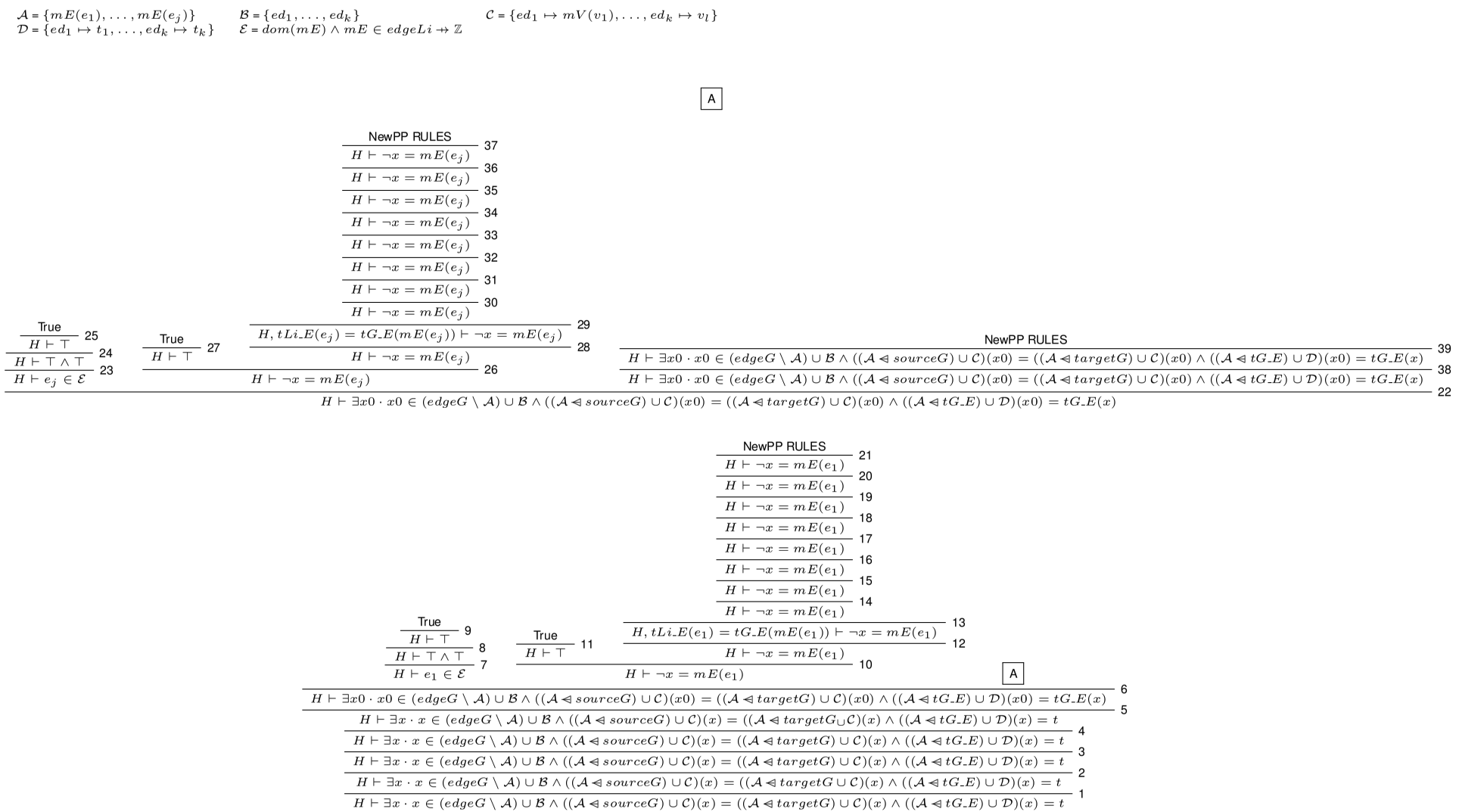


Figura 57: Árvore de Prova rule_i/propLoopT/INV - Regra Preserva, não Deleta e não Cria Arcos Loop do Tipo t ou não Envolve Arcos Loop do Tipo t.

primeiro arco deletado pela regra, essa ação gera três novos sub-objetivos: (I) $H \vdash e_1 \in dom(mE) \land mE \in edgeLi \nrightarrow \mathbb{Z}$*; (II)* $H \vdash \neg x = mE(e_1)$*; e (III)* $H \vdash \exists x0 \cdot x0 \in (edgeG \setminus \{mE(e_1), \ldots, mE(e_j)\}) \cup \{ed_1, \ldots, ed_k\} \land (((\{mE(e_1), \ldots, mE(e_j)\} \lhd sourceG) \cup \{ed_1 \mapsto mV(v_1), \ldots, ed_k \mapsto v_l\})(x0) = (((\{mE(e_1), \ldots, mE(e_j)\} \lhd targetG) \cup \{ed_1 \mapsto mV(v_1), \ldots, ed_k \mapsto v_l\})(x0) \land (((\{mE(e_1), \ldots, mE(e_j)\} \lhd tG\_E) \cup \{ed_1 \mapsto t_1, \ldots, ed_k \mapsto t_k\})(x0) = tG\_E(x)$*. (I) é descarregado automaticamente pelas regras generalized MP e simplification rewrites e* $\top$ *goal (regras 7-9). A fim de demonstrar (II) instancia-se o nome do primeiro arco deletado* e_1 *na hipótese* `grd_edges` *(10). Com isso, são gerados dois novos sub-objetivos para se demonstrar: (i)* $H \vdash \top$ *descarregado de forma automática (11) e (ii)* $H \vdash \neg x = mE(e_1)$*. Para demonstrar (ii), aplica-se a regra partition rewrites (13) na hipótese que define a tipagem dos arcos do lado esquerdo da regra. Após essa ação várias regras automáticas são aplicadas (regras 14-19), chegando no sequente* $H \vdash \neg x = mE(e_1)$ *que é demonstrado aplicando o provador NewPP with lasso (21). Esse processo anterior é repetido para cada um dos arcos deletados até o arco* e_j *(representado na árvore). A fim de demonstrar (III), sub-árvore* $\boxed{A}$*, adiciona-se* $\neg x = mE(e_j)$ *como hipótese (22), onde* e_j *é outro arco deletado pela regra, após são gerados três sub-objetivos para se demonstrar: (i)* $H \vdash e_j \in dom(mE) \land mE \in edgeLi \nrightarrow \mathbb{Z}$*, que é descarregado automaticamente (regras 23-25); (ii)* $H \vdash \neg x = mE(e_j)$*; e (iii)* $H \vdash \exists x0 \cdot x0 \in (edgeG \setminus \{mE(e_1), \ldots, mE(e_j)\}) \cup \{ed_1, \ldots, ed_k\} \land (((\{mE(e_1), \ldots, mE(e_j)\} \lhd sourceG) \cup \{ed_1 \mapsto mV(v_1), \ldots, ed_k \mapsto v_l\})(x0) = (((\{mE(e_1), \ldots, mE(e_j)\} \lhd targetG) \cup \{ed_1 \mapsto mV(v_1), \ldots, ed_k \mapsto v_l\})(x0) \land (((\{mE(e_1), \ldots, mE(e_j)\} \lhd tG\_E) \cup \{ed_1 \mapsto t_1, \ldots, ed_k \mapsto t_k\})(x0) = tG\_E(x)$*. Para demonstrar (ii) instancia-se o nome do outro arco deletado* e_j *na hipótese* `grd_edges` *(26). Após, aplica-se a regra partition rewrites (29) na hipótese que define a tipagem dos arcos do lado esquerdo da regra e, em seguida, executa-se o provador NewPP with lasso (37). Já o sequente (iii) é descarregado pelo provador NewPP with lasso (39).*

A árvore de uso para esses casos de obrigações é representada pela Figura 58.

Descrição da Árvore de Uso: *Para utilizar essa tática aplica-se a regra* $\exists$ *hyp na hipótese de indução. Em seguida são executadas as ações: adicionar como hipótese que o elemento* x *(da hipótese de indução) é diferente do arco deletado pela regra; instanciar o nome do arco deletado* e_1 *na hipótese* `grd_edges`; aplicar a regra partition rewrites na hipótese que define a tipagem dos arcos do lado esquerdo da regra tLi_E; executar o provador NewPP with lasso. Tais ações são realizadas para cada arco deletado (diferente do tipo t) pela regra. Após isso, o último objetivo é demonstrado aplicando o provador NewPP with lasso.

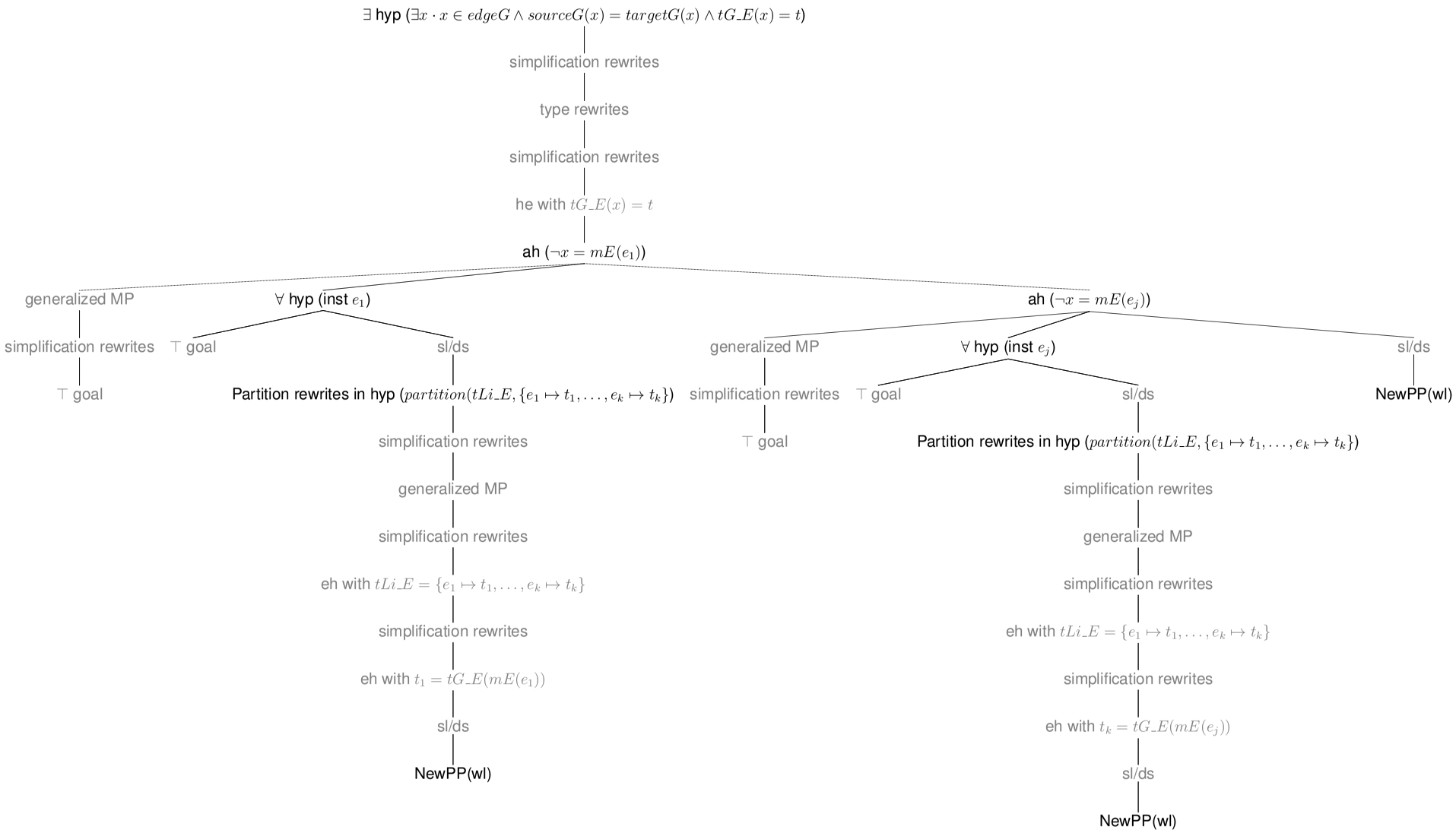


Figura 58: Árvore de Uso rule_i/PropLoopT/INV - Regra Preserva, não Deleta e não Cria Arcos Loop do Tipo t ou não Envolve Arcos Loop do Tipo t

C - Preserva, deleta e não cria arcos loop do tipo t.

Para demonstrar a obrigação gerada por esse caso de regra, executam-se as seguintes ações:

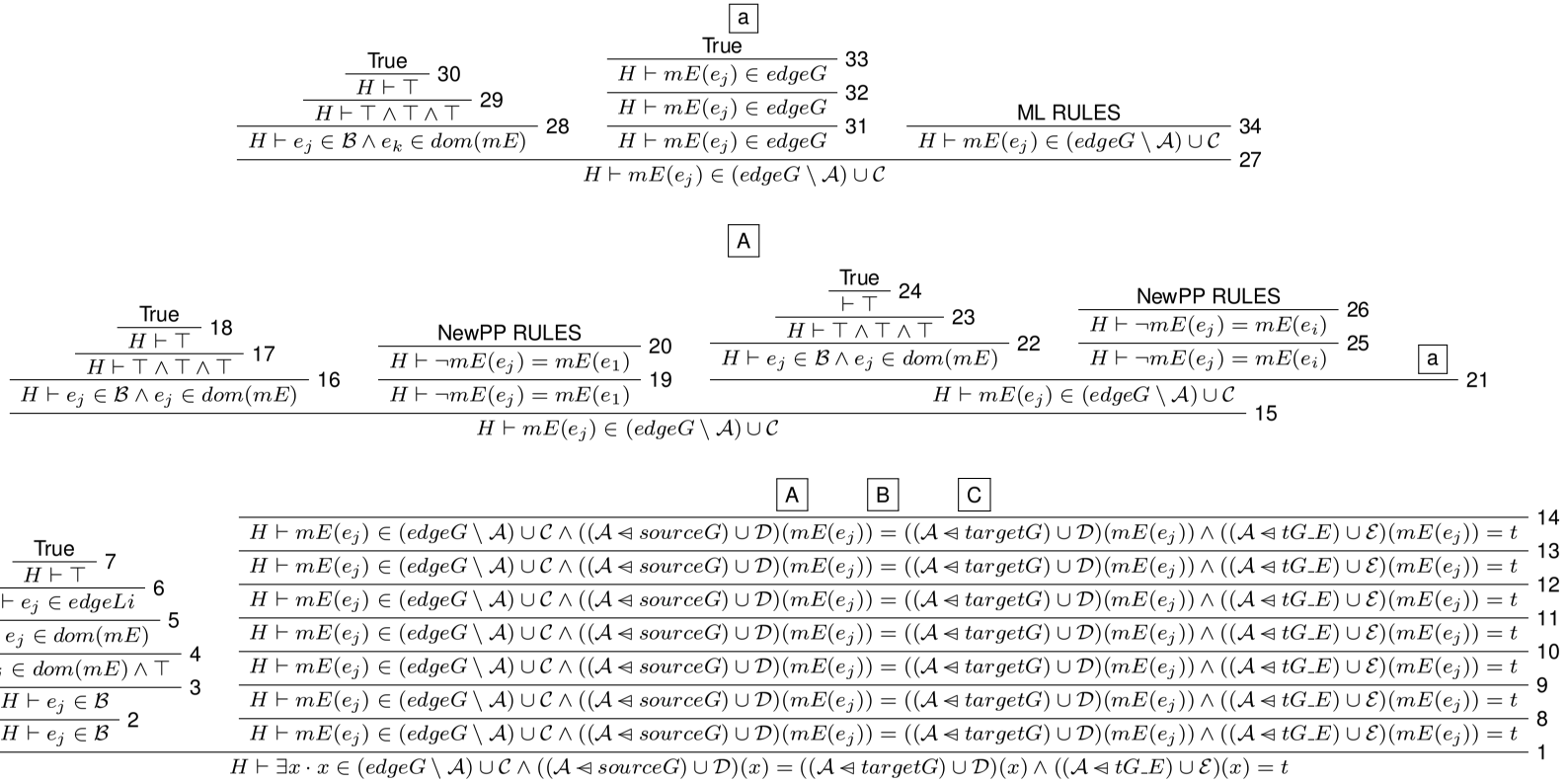
1. Instanciar no objetivo $mE(e_j)$, onde e_j é o nome do arco preservado do tipo t;
2. Para cada arco e_i deletado pela regra:
   (a) Adicionar $\neg mE(e_j) = mE(e_i)$ como hipótese, onde e_j é o nome do arco loop do tipo t (preservado) e e_i é um arco deletado pela regra. Após executar o provador NewPP with lasso.
3. Adicionar $mE(e_j) \in edgeG$ como hipótese e executar ML;
4. Adicionar como hipótese $sourceG(mE(e_j)) = targetG(mE(e_j))$;
5. Adicionar como hipótese $sourceLi(e_j) = targetLi(e_j)$;
6. Aplicar a regra partition rewrites na hipótese que define a origem dos arcos do lado esquerdo da regra;
7. Aplicar a regra partition rewrites na hipótese que define o destino dos arcos do lado esquerdo da regra;
8. Instanciar o nome do arco loop e_j de tipo t, preservado pela aplicação da regra, na hipótese `grd_srctgt` considerando $sourceG$;
9. Instanciar o nome do arco loop e_j de tipo t, preservado pela aplicação da regra, na hipótese `grd_srctgt` considerando $targetG$;
10. Executar o provador NewPP with lasso;
11. Executar o provador ML;
12. Instanciar o nome do arco loop e_j preservado pela regra na hipótese `grd_edges`;
13. Aplicar a tática partition rewrites na hipótese que define a tipagem dos arcos do lado esquerdo da regra;
14. Executar o provador ML.

Ao executar todos os passos mostrados acima, é gerada uma árvore de prova. Em virtude de limitação de espaço, essa árvore foi dividida e apresentada através das Figuras 59, 60, 61 e 62.

A Figura 59 é resultante da execução dos seguintes passos da tática proposta (1-4).

Descrição da Árvore de Prova para as Regras: *Conforme ilustrado na Figura 59, o primeiro sequente para se demonstrar é* $H \vdash \exists x \cdot x \in (edgeG \setminus \{mE(e_1), \ldots, mE(e_j)\}) \cup \{ed_1, \ldots, ed_k\} \land ((\{mE(e_1), \ldots, mE(e_j)\} \lhd sourceG) \cup \{ed_1 \mapsto mV(v_1), \ldots, ed_k \mapsto mV(v_l)\})(x) = ((\{mE(e_1), \ldots, mE(e_j)\} \lhd targetG) \cup \{ed_1 \mapsto mV(v_1), \ldots, ed_k \mapsto mV(v_l)\})(x) \land ((\{mE(e_1), \ldots, mE(e_j)\} \lhd tG\_E) \cup \{ed_1 \mapsto t_1, \ldots, ed_k \mapsto t_k\})(x) = t$. *A fim de demonstrar esse sequente, instancia-se* $mE(e_j)$ *no objetivo, onde* e_j *representa o nome do arco do tipo* t *preservado pela regra. Com isso dois novos sub-objetivos são gerados: (I)* $H \vdash e_j \in dom(mE) \land mE \in edgeLi \nrightarrow \mathbb{Z}$*; e (II)* $H \vdash mE(e_j) \in (edgeG \setminus \{mE(e_1), \ldots, mE(e_j)\}) \cup \{ed_1, \ldots, ed_k\} \land ((\{mE(e_1), \ldots, mE(e_j)\} \lhd sourceG) \cup \{ed_1 \mapsto mV(v_1), \ldots, ed_k \mapsto mV(v_l)\})(mE(e_j)) = ((\{mE(e_1), \ldots, mE(e_j)\} \lhd targetG) \cup \{ed_1 \mapsto mV(v_1), \ldots, ed_k \mapsto mV(v_l)\})(mE(e_j)) \land ((\{mE(e_1), \ldots, mE(e_j)\} \lhd tG\_E) \cup \{ed_1 \mapsto t_1, \ldots, ed_k \mapsto t_k\})(mE(e_j)) = t$. *Em (I), após a instanciação são aplicadas algumas regras automáticas (regras 2-6), chegando no objetivo* $H \vdash \top$ *que é demonstrado automaticamente (7). Da mesma forma, em* (II) *são aplicadas regras automáticas (regras 8-14), gerando três novos objetivos para se demonstrar: (I)* $H \vdash mE(e_j) \in (edgeG \setminus \{mE(e_1), \ldots, mE(e_j)\}) \cup \{ed_1, \ldots, ed_k\}$*; (II)* $H \vdash ((\{mE(e_1), \ldots, mE(e_j)\} \lhd sourceG) \cup \{ed_1 \mapsto mV(v_1), \ldots, ed_k \mapsto mV(v_l)\})(mE(e_j)) = ((\{mE(e_1), \ldots, mE(e_j)\} \lhd targetG) \cup \{ed_1 \mapsto mV(v_1), \ldots, ed_k \mapsto mV(v_l)\})(mE(e_j))$*; e (III)* $H \vdash ((\{mE(e_1), \ldots, mE(e_j)\} \lhd tG\_E) \cup \{ed_1 \mapsto mV(v_1), \ldots, ed_k \mapsto mV(v_l)\})(mE(e_j)) = tG\_E(x)$. *A fim de demonstrar (I), sub-árvore* $\boxed{A}$*, adiciona-se como hipótese* $\neg mE(e_j) = mE(e_1)$ *(15), onde* e_j *é o arco loop do tipo* t *preservado e* e_1 *é um dos arcos deletados pela regra. Após adicionar essa hipótese, são gerados três novos sub-objetivos: (i)* $H \vdash e_j \in dom(mE) \land mE \in edgeLi \nrightarrow \mathbb{Z} \land e_j \in dom(mE)$*; (ii)* $H \vdash \neg mE(e_j) = mE(e_1)$ *e (iii)* $H \vdash mE(e_j) \in (edgeG \setminus \{mE(e_1), \ldots, mE(e_j)\}) \cup \{ed_1, \ldots, ed_k\}$*. Em (i), são aplicadas regras automáticas (regras 16-17) chegando no sequente* $H \vdash \top$ *demonstrado automaticamente (18). Já o sequente (ii) é demonstrado pelo provador NewPP with lasso (20). Essa última sequência de táticas repete-se para cada arco deletado de* e_1 *até* e_j*. A fim de demonstrar (iii), adiciona-se como hipótese* $\neg mE(e_j) = mE(e_i)$ *(21), onde* e_j *é o arco loop preservado do tipo* t *e* e_i *é outro arco deletado pela regra. Em seguida são gerados três novos sequentes para se demonstrar:* (i) $H \vdash ed_t \in dom(mE) \land mE \in edgeLi \nrightarrow \mathbb{Z} \land ed_k \in dom(mE)$*; (ii)* $H \vdash \neg mE(e_j) = mE(e_j)$*; e (iii)* $H \vdash mE(e_j) \in (edgeG \setminus \{mE(e_1), \ldots, mE(e_j)\}) \cup \{ed_1, \ldots, ed_k\}$*. Em* (i) *são aplicadas de forma automática algumas regras (22-23), chegando no objetivo* $H \vdash \top$ *demonstrado automaticamente (24).* (ii) *é descarregado aplicando o provador NewPP with lasso. Para demonstrar* (iii)*, sub-árvore* $\boxed{a}$*, adiciona-se como hipótese* $mE(e_j) \in edgeG$*, gerando três sub-objetivos para se demonstrar: (i)* $H \vdash e_j \in dom(mE) \land mE \in edgeLi \nrightarrow \mathbb{Z} \land e_k \in dom(mE)$*; (ii)* $H \vdash mE(e_j) \in edgeG$*; e (iii)* $H \vdash mE(e_j) \in (edgeG \setminus \{mE(e_1), \ldots, mE(e_j)\}) \cup \{ed_1, \ldots, ed_k\}$*. Em (i) são aplicadas algumas regras, gerando o objetivo* $H \vdash \top$ *demonstrado automaticamente (30). (ii) é*

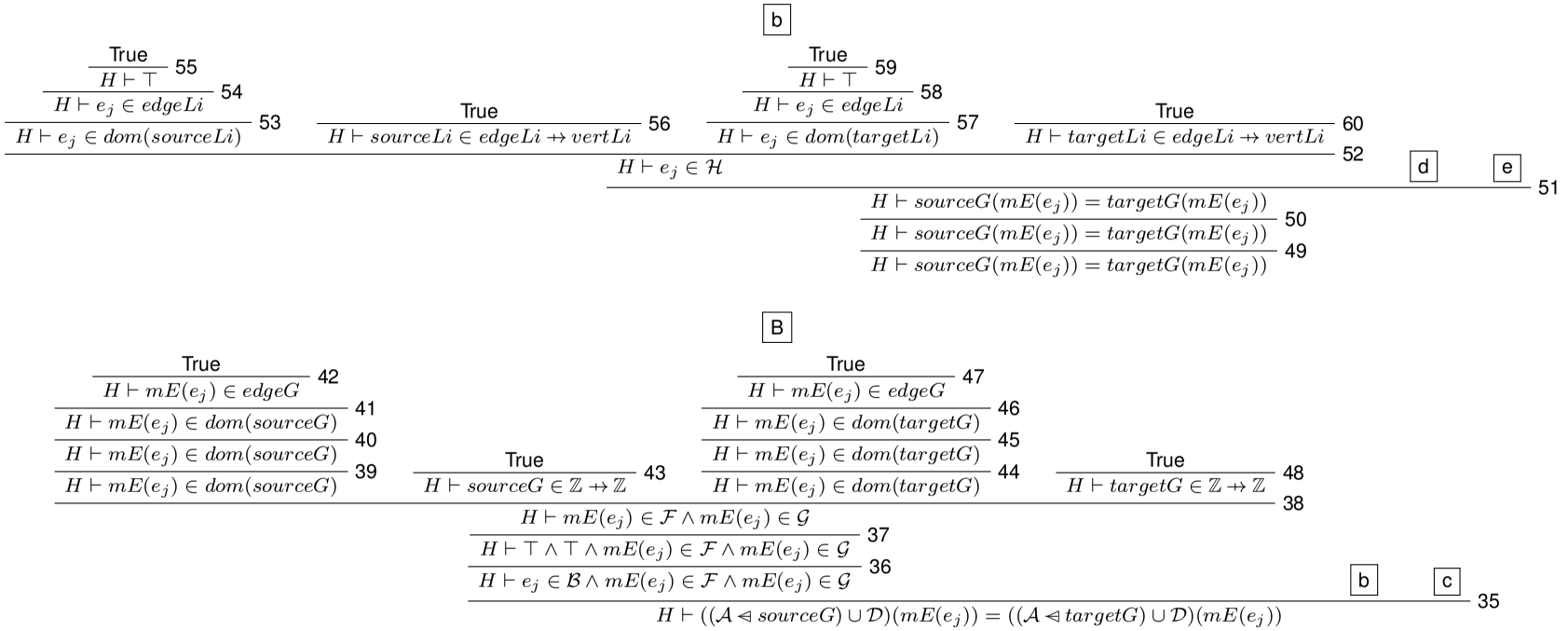
$\mathcal{A}$ = $\{mE(e_1), \ldots, mE(e_j)\}$ $\quad$ $\mathcal{B}$ = $dom(mE) \land mE \in edgeLi \nrightarrow \mathbb{Z}$ $\quad$ $\mathcal{C}$ = $\{ed_1, \ldots, ed_k\}$ $\quad$ $\mathcal{D}$ = $\{ed_1 \mapsto mV(v_1), \ldots, ed_k \mapsto mV(v_l)\}$ $\quad$ $\mathcal{E}$ = $\{ed_1 \mapsto t_1, \ldots, ed_k \mapsto t_k\}$



1 ∃ goal (inst $mE(e_j)$); 2 simplification rewrites; 3 generalized MP; 4 simplification rewrites; 5 total function dom substitution in goal; 6 type rewrites; 7 ⊤ goal; 8 generalized MP; 9 simplification rewrites; 10 type rewrites; 11 simplification rewrites; 12 ∃ hyp ($\exists x \cdot x \in edgeG \land sourceG(x) = targetG(x) \land tG\_E(x) = t$); 13 he with $tG\_E(x) = t$; 14 ∧ goal; 15 ah ($\neg mE(e_j) = mE(e_1)$); 16 generalized MP; 17 simplification rewrites; 18 ⊤ goal; 19 sl/ds; 20 NewPP with lasso; 21 ah ($\neg mE(e_j) = mE(e_i)$); 22 generalized MP; 23 simplification rewrites; 24 ⊤ goal; 25 sl/ds; 26 NewPP with lasso; 27 ah ($mE(e_j) \in edgeG$); 28 generalized MP; 29 simplification rewrites; 30 ⊤ goal; 31 functional image goal for $mE(e_j)$; 32 functional image goal for $mE(e_j)$; 33 hyp; 34 ML;

Figura 59: Árvore de Prova rule_i/propLoopT/INV - Regra Preserva, Deleta e Não Cria Arcos Loop do Tipo t (1).

$\mathcal{A}$ = $\{mE(e_1), \ldots, mE(e_j)\}$ $\mathcal{B}$ = $dom(mE) \land mE \in edgeLi \pfun \mathbb{Z}$ $\mathcal{C}$ = $\{ed_1, \ldots, ed_k\}$ $\mathcal{D}$ = $\{ed_1 \mapsto mV(v_1), \ldots, ed_k \mapsto v_l\}$
$\mathcal{E}$ = $\{ed_1 \mapsto t_1, \ldots, ed_k \mapsto t_k\}$ $\mathcal{F}$ = $dom(sourceG) \land sourceG \in \mathbb{Z} \pfun \mathbb{Z}$ $\mathcal{G}$ = $dom(targetG) \land targetG \in \mathbb{Z} \pfun \mathbb{Z}$ $\mathcal{H}$ = $dom(sourceLi) \land sourceLi \in edgeLi \pfun vertLi$
$\mathcal{I}$ = $dom(targetLi) \land targetLi \in edgeLi \pfun vertLi$



35 ah ($sourceG(mE(e_j)) = targetG(mE(e_j))$); 36 generalized MP; 37 simplification rewrites; 38 $\land$ goal; 39 functional image goal for $mE(e_j)$; 40 functional image goal for $mE(e_j)$; 41 total function dom substitution in goal; 42 hyp; 43 functional goal; 44 functional image goal for $mE(e_j)$; 45 functional image goal for $mE(e_j)$; 46 total function dom substitution in goal; 47 hyp; 48 functional goal; 49 generalized MP; 50 simplification rewrites; 51 ah ($sourceLi(e_j) = targetLi(e_j)$); 52 $\land$ goal; 53 total function dom substitution in goal; 54 type rewrites; 55 $\top$ goal; 56 functional goal; 57 total function dom substitution in goal; 58 type rewrites; 59 $\top$ goal; 60 functional goal;

Figura 60: Árvore de Prova rule_i/propLoopT/INV - Regra Preserva, Deleta e Não Cria Arcos Loop do Tipo t (2)

demonstrado por NewPP with lasso e (iii) é demonstrado por ML.

Na Figura 60 a apresentação da árvore de prova da Figura 59 tem continuidade. Nessa árvore executou-se os seguintes passos da tática proposta (5-6).

A fim de demonstrar o sequente (II), sub-árvore B *na Figura 60,* $H \vdash (((\{mE(e_1), \ldots, mE(e_j)\} \lhd sourceG) \cup \{ed_1 \mapsto mV(v_1), \ldots, ed_k \mapsto mV(v_l)\})(mE(e_j)) = (((\{mE(e_1), \ldots, mE(e_j)\} \lhd targetG) \cup \{ed_1 \mapsto mV(v_1), \ldots, ed_k \mapsto mV(v_l)\})(mE(e_j))$*, adiciona-se como hipótese* $sourceG(mE(e_j)) = targetG(mE(e_j))$ *(35), após a instanciação são gerados três novos sub-objetivos: (I)* $H \vdash sourceG(mE(e_j)) = targetG(mE(e_j))$*; (II)* $H \vdash e_j \in dom(mE) \land mE \in edgeLi \pfun \mathbb{Z} \land mE(ed_t) \in dom(sourceG) \land sourceG \in \mathbb{Z} \pfun \mathbb{Z} \land mE(ed_t) \in dom(targetG) \land targetG \in \mathbb{Z} \pfun \mathbb{Z}$*; (III)* $H \vdash (((\{mE(e_1), \ldots, mE(e_j)\} \lhd sourceG) \cup \{ed_1 \mapsto mV(v_1), \ldots, ed_k \mapsto mV(v_l)\})(mE(e_j)) = (((\{mE(e_1), \ldots, mE(e_j)\} \lhd targetG) \cup \{ed_1 \mapsto mV(v_1), \ldots, ed_k \mapsto mV(v_l)\})(mE(e_j))$*. Em (I), são aplicadas diversas regras de forma automática, que acabam demonstrando todos os sub-objetivos gerados (regras 36-48). Em (II), sub-árvore* b*, são aplicadas de forma automática algumas regras (49-50) e, visando demonstrar o objetivo, adiciona-se como hipótese* $sourceLi(e_j) = targetLi(e_j)$ *(51). Essa instanciação gera três novos objetivos para se demonstrar: (i)* $H \vdash ed_t \in dom(sourceLi) \land sourceLi \in edgeLi \pfun vertLi$*; (ii)* $H \vdash sourceLi(e_j) = targetLi(e_j)$*; (iii)* $H \vdash sourceG(mE(e_j)) = targetG(mE(e_j))$*. Em (i), são aplicadas algumas regras automáticas que descarregam todos os sub-objetivos gerados (regras 52-60).*

A Figura 61 apresenta a continuação da árvore de prova para a obrigação rule_i/propLoopT/INV, foram executados nessa parte da árvore os seguintes passos da tática proposta (7-11).

Para demonstrar o sequente (ii), sub-árvore d *na Figura 61, aplica-se a regra partition rewrites na hipótese que define a origem dos arcos do lado esquerdo da regra (62). Em seguida, são aplicadas algumas regras de forma automática (regras 63-66), de mesma forma aplica-se a regra partition rewrites na hipótese que define o destino dos arcos do lado esquerdo da regra (67). Após essa ação, regras são aplicadas de forma automática (regras 68-73) transformando o objetivo em* $H \vdash \top$*, que é demonstrado automaticamente (74). Para demonstrar (iii), sub-árvore* e*, instancia-se o nome do arco loop* e_j *(76) do tipo* t *preservado pela regra na hipótese* grd_srctgt*, considerando* $sourceG$*. Após essa instanciação são gerados dois novos sub-objetivos: (i)* $H \vdash \top$*, demonstrado de forma automática (77); (ii)* $H \vdash sourceG(mE(e_j)) = targetG(mE(e_j))$*. Para demonstrar (ii) instancia-se o nome do arco preservado* e_j *de tipo* t *na hipótese* grd_srctgt*, considerando* $targetG$ *(78). Com isso, são gerados dois sub-objetivos, o primeiro* $H \vdash \top$ *é demonstrado de forma automática (79). Já o segundo, relativo ao*

$\mathcal{A} = \{mE(e_1), \ldots, mE(e_j)\}$ $\quad$ $\mathcal{B} = dom(mE) \wedge mE \in edgeLi \rightarrowtail \mathbb{Z}$ $\quad$ $\mathcal{C} = \{ed_1, \ldots, ed_k\}$ $\quad$ $\mathcal{D} = \{ed_1 \mapsto mV(v_1), \ldots, ed_k \mapsto v_l\}$

$\mathcal{E} = \{ed_1 \mapsto t_1, \ldots, ed_k \mapsto t_k\}$ $\quad$ $\mathcal{F} = dom(sourceG) \wedge sourceG \in \mathbb{Z} \rightarrowtail \mathbb{Z}$ $\quad$ $\mathcal{G} = dom(targetG) \wedge targetG \in \mathbb{Z} \rightarrowtail \mathbb{Z}$ $\quad$ $\mathcal{H} = dom(sourceLi) \wedge sourceLi \in edgeLi \rightarrowtail vertLi$

$\mathcal{I} = dom(targetLi) \wedge targetLi \in edgeLi \rightarrowtail vertLi$

e

$$\dfrac{\dfrac{\dfrac{\text{True}}{H \vdash \top}\,77 \qquad \dfrac{\dfrac{\text{True}}{H \vdash \top}\,79 \qquad \dfrac{\dfrac{\text{NewPP RULES}}{H \vdash sourceG(mE(e_j)) = targetG(mE(e_j))}\,81}{H \vdash sourceG(mE(e_j)) = targetG(mE(e_j))}\,80}{H \vdash sourceG(mE(e_j)) = targetG(mE(e_j))}\,78}{H \vdash sourceG(mE(e_j)) = targetG(mE(e_j))}\,76}{H \vdash sourceG(mE(e_j)) = targetG(mE(e_j))}\,75$$

d

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\text{True}}{H \vdash \top}\,74}{H \vdash v_t = \{e_1 \mapsto v_1, \ldots, e_k \mapsto v_l\}(e_j)}\,73}{H \vdash v_t = targetLi(e_j)}\,72}{H \vdash v_t = targetLi(e_j)}\,71}{H \vdash v_t = targetLi(e_j)}\,70}{H \vdash v_t = targetLi(e_j)}\,69}{H \vdash v_t = targetLi(e_j)}\,68}{H \vdash v_t = targetLi(e_j)}\,67}{H \vdash \{e_1 \mapsto v_1, \ldots, e_k \mapsto v_l\}(e_j) = targetLi(e_j)}\,66}{H \vdash sourceLi(e_j) = targetLi(e_j)}\,65}{H \vdash sourceLi(e_j) = targetLi(e_j)}\,64}{H \vdash sourceLi(e_j) = targetLi(e_j)}\,63}{H \vdash sourceLi(e_j) = targetLi(e_j)}\,62}{H \vdash sourceLi(e_j) = targetLi(e_j)}\,61$$

61 sl/ds; 62 Partition rewrites in hyp ($partition(sourceLi, \{e_1 \mapsto v_1\}, \ldots, \{e_k \mapsto v_l\})$), 63 simplification rewrites; 64 remove $\neg$ in $\neg(e_1 = e_k \wedge v_1 = v_l)$); 65 eh with $sourceLi = \{e_1 \mapsto v_1, \ldots, e_k \mapsto v_l\}$; 66 simplification rewrites; 67 Partition rewrites in hyp ($partition(targetLi, \{e_1 \mapsto v_1\}, \ldots, \{e_j \mapsto v_t\})$); 68 simplification rewrites; 69 generalized MP; 70 simplification rewrites; 71 remove $\neg$ in $\neg(e_1 = e_k \wedge v_1 = v_l)$); 72 eh with $targetLi = \{e_1 \mapsto v_1, \ldots, e_k \mapsto v_l\}$; 73 simplification rewrites; 74 $\top$ goal; 75 sl/ds; 76 $\forall$ hyp (inst e_j); 77 $\top$ goal; 78 $\forall$ hyp (inst e_j); 79 $\top$ goal; 80 sl/ds; 81 NewPP with lasso;

Figura 61: Árvore de Prova rule_i/propLoopT/INV - Regra Preserva, Deleta e Não Cria Arcos Loop do Tipo t (3)

$\mathcal{A} = \{mE(e_1), \ldots, mE(e_j)\}$
$\mathcal{B} = dom(mE) \wedge mE \in edgeLi \rightarrow\!\!\!\!\!\!+ \mathbb{Z}$
$\mathcal{C} = \{ed_1, \ldots, ed_k\}$
$\mathcal{D} = \{ed_1 \mapsto mV(v_1), \ldots, ed_k \mapsto v_l\}$
$\mathcal{E} = \{ed_1 \mapsto t_1, \ldots, ed_k \mapsto t_k\}$
$\mathcal{F} = dom(sourceG) \wedge sourceG \in \mathbb{Z} \rightarrow\!\!\!\!\!\!+ \mathbb{Z}$
$\mathcal{G} = dom(targetG) \wedge targetG \in \mathbb{Z} \rightarrow\!\!\!\!\!\!+ \mathbb{Z}$
$\mathcal{H} = dom(sourceLi) \wedge sourceLi \in edgeLi \rightarrow\!\!\!\!\!\!+ vertLi$
$\mathcal{I} = dom(targetLi) \wedge targetLi \in edgeLi \rightarrow\!\!\!\!\!\!+ vertLi$

C

$$\cfrac{\cfrac{\text{True}}{H \vdash \top}\ 86 \qquad \cfrac{\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{\text{ML RULES}}{H \vdash ((\mathcal{A} \lhd tG\_E) \cup \mathcal{D})(mE(e_j)) = tG\_E(x)}\ 97}{H \vdash ((\mathcal{A} \lhd tG\_E) \cup \mathcal{D})(mE(e_j)) = tG\_E(x)}\ 96}{H \vdash ((\mathcal{A} \lhd tG\_E) \cup \mathcal{D})(mE(e_j)) = tG\_E(x)}\ 95}{H \vdash ((\mathcal{A} \lhd tG\_E) \cup \mathcal{D})(mE(e_j)) = tG\_E(x)}\ 94}{H \vdash ((\mathcal{A} \lhd tG\_E) \cup \mathcal{D})(mE(e_j)) = tG\_E(x)}\ 93}{H \vdash ((\mathcal{A} \lhd tG\_E) \cup \mathcal{D})(mE(e_j)) = tG\_E(x)}\ 92}{H \vdash ((\mathcal{A} \lhd tG\_E) \cup \mathcal{D})(mE(e_j)) = tG\_E(x)}\ 91}{H \vdash ((\mathcal{A} \lhd tG\_E) \cup \mathcal{D})(mE(e_j)) = tG\_E(x)}\ 90}{H \vdash ((\mathcal{A} \lhd tG\_E) \cup \mathcal{D})(mE(e_j)) = tG\_E(x)}\ 89}{H \vdash ((\mathcal{A} \lhd tG\_E) \cup \mathcal{D})(mE(e_j)) = tG\_E(x)}\ 88}{H \vdash ((\mathcal{A} \lhd tG\_E) \cup \mathcal{D})(mE(e_j)) = tG\_E(x)}\ 87}{H \vdash ((\mathcal{A} \lhd tG\_E) \cup \mathcal{D})(mE(e_j)) = tG\_E(x)}\ 85$$

c

$$\cfrac{\cfrac{\cfrac{\text{True}}{H \vdash ((\mathcal{A} \lhd sourceG) \cup \mathcal{D})(mE(e_j)) = ((\mathcal{A} \lhd targetG) \cup \mathcal{D})(mE(e_j))}\ 84}{H \vdash ((\mathcal{A} \lhd sourceG) \cup \mathcal{D})(mE(e_j)) = ((\mathcal{A} \lhd targetG) \cup \mathcal{D})(mE(e_j))}\ 83}{H \vdash ((\mathcal{A} \lhd sourceG) \cup \mathcal{D})(mE(e_j)) = ((\mathcal{A} \lhd targetG) \cup \mathcal{D})(mE(e_j))}\ 82$$

82 generalized MP; 83 simplification rewrites; 84 ML; 85 $\forall$ hyp (inst e_j); 86 $\top$ goal; 87 sl/ds; 88 he with $tG\_E(x) = t$;
89 Partition rewrites in hyp ($partition(tLi\_E, \{e_1 \mapsto t_1\}, \{e_j \mapsto tG\_E(x)\}, \ldots, \{e_k \mapsto t_k\})$); 90 simplification rewrites; 91 generalized MP; 92 simplification rewrites;
93 remove $\neg$ in $\neg(e_1 = e_k \wedge e_1 = tG\_E(x))$; 94 remove $\neg$ in $\neg(e_1 = e_k \wedge e_k = tG\_E(x))$; 95 eh with $tLi\_E = \{e_1 \mapsto t_1, e_j \mapsto tG\_E(x), \ldots e_k \mapsto t_k\}$; 96 simplification rewrites; 97 ML.

Figura 62: Árvore de Prova rule_i/propLoopT/INV - Regra Preserva, Deleta e Não Cria Arcos Loop do Tipo t (4)

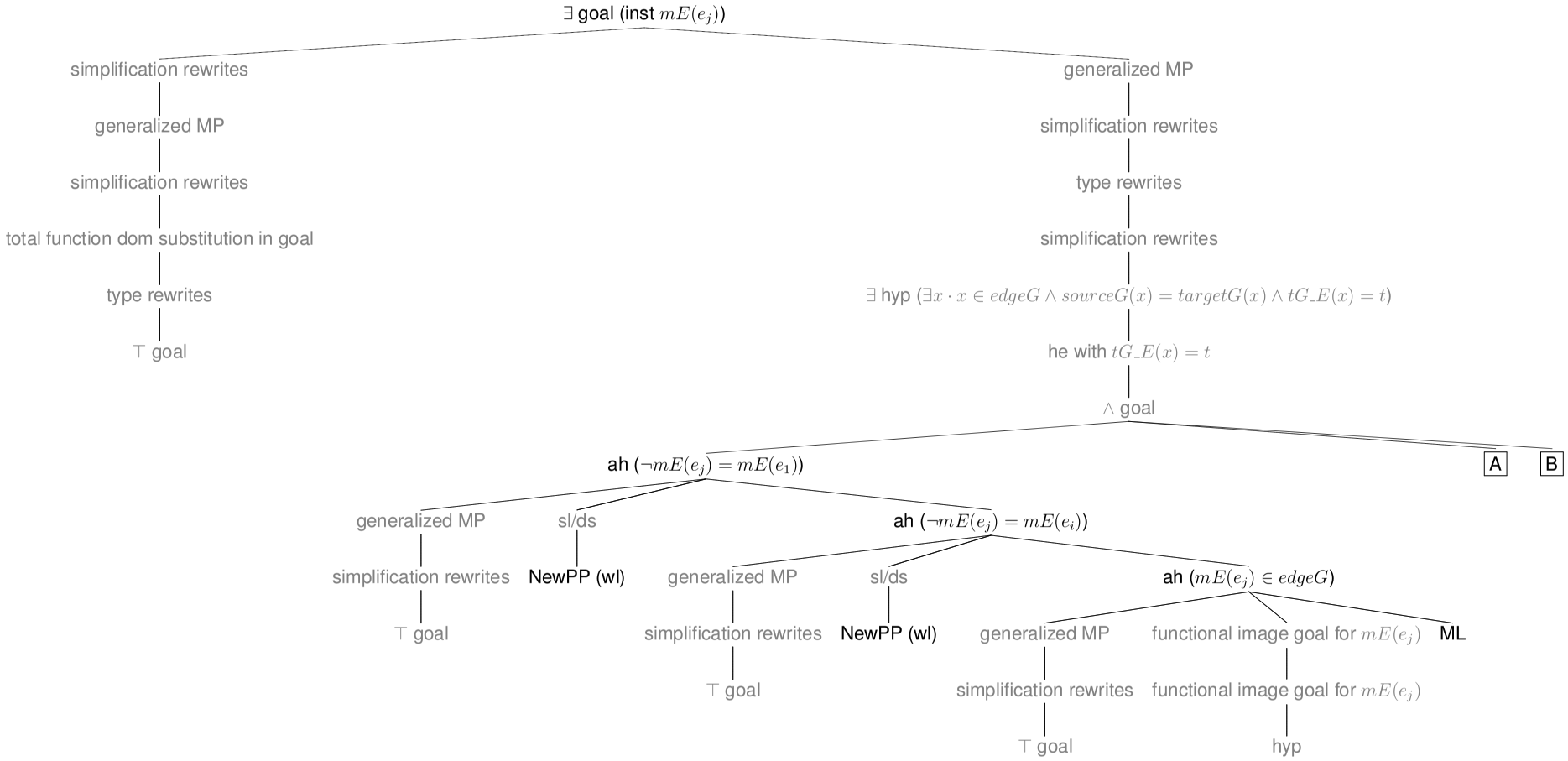


Figura 63: Árvore de Uso rule_i/propLoopT/INV - Regra Preserva, Deleta e não Cria Arcos Loop do Tipo t (1)

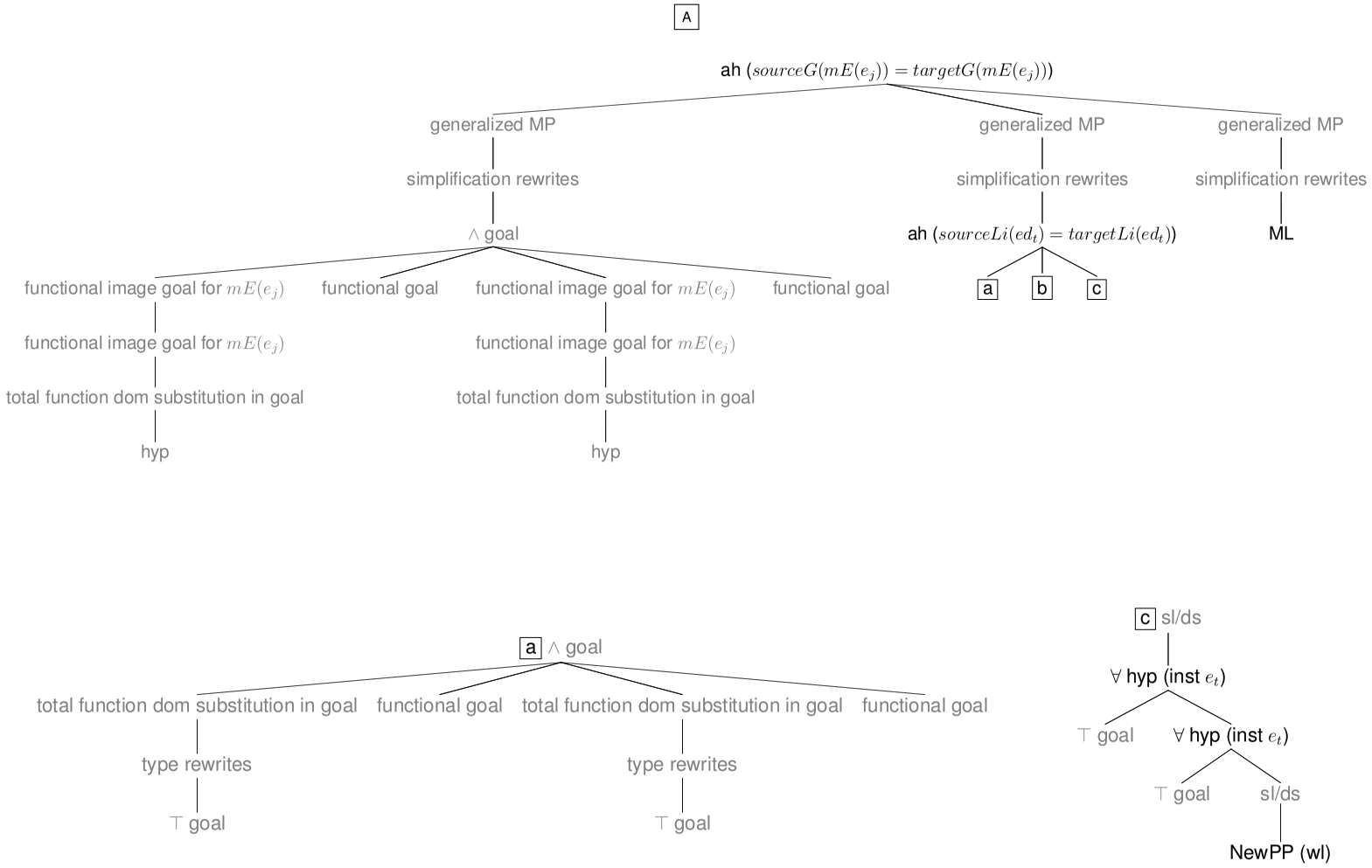


Figura 64: Árvore de Uso rule_i/propLoopT/INV - Regra Preserva, Deleta e não Cria Arcos Loop do Tipo t (2)

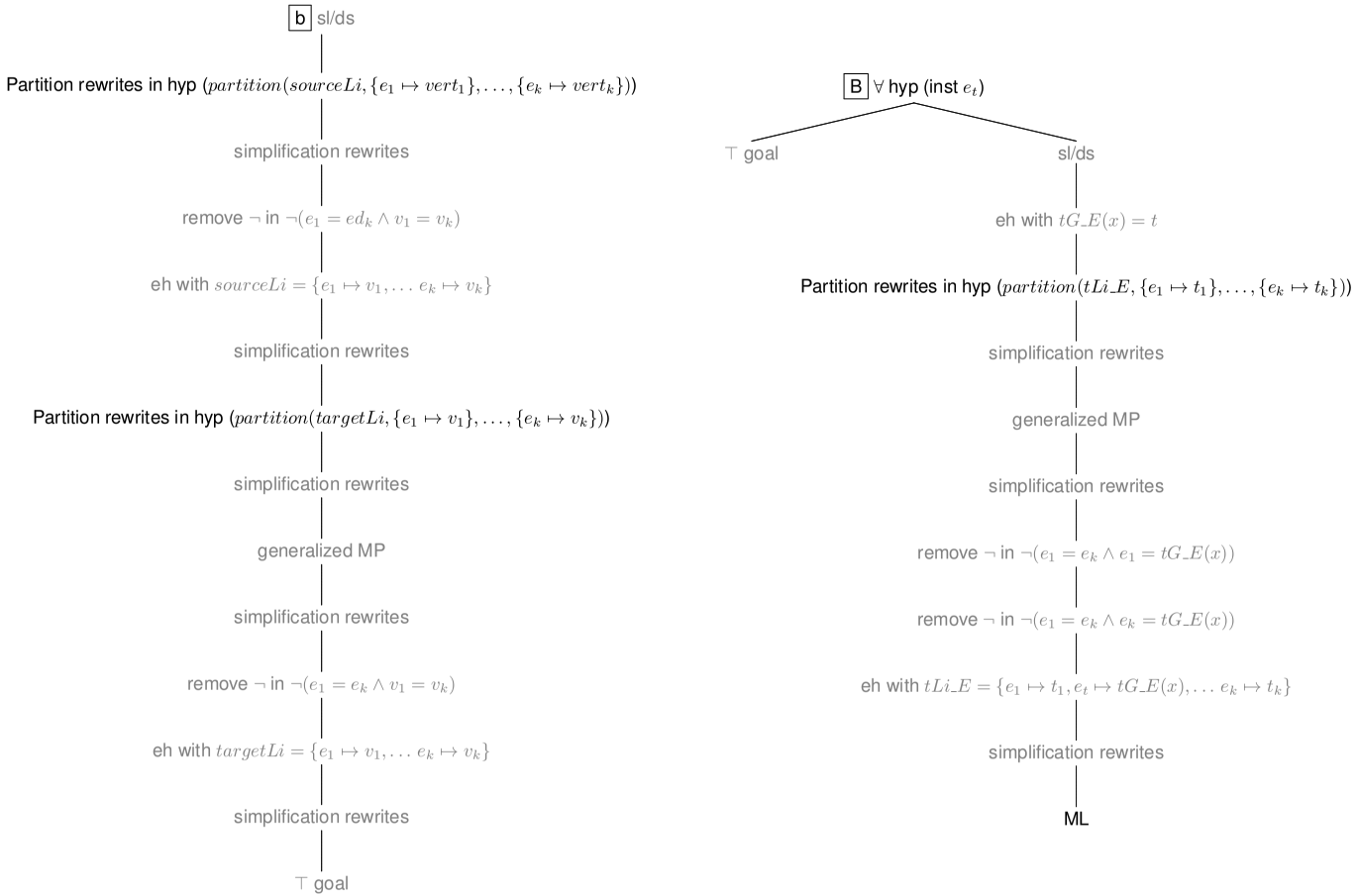


Figura 65: Árvore de Uso rule_i/propLoopT/INV - Regra Preserva, Deleta e não Cria Arcos Loop do Tipo t (3)

objetivo $H \vdash sourceG(mE(e_j)) = targetG(mE(e_j))$*, é demonstrado executando o provador NewPP with lasso (81).*

A última parte da árvore de prova é apresentada através da Figura 62, nessa parte da árvore são executadas as seguintes etapas da tática proposta (12-15).

Na Figura 62, em (III), sub-árvore $\boxed{c}$*,* $H \vdash (((\{mE(e_1), \ldots, mE(e_j)\} \lhd sourceG) \cup \{ed_1 \mapsto mV(v_1), \ldots, ed_k \mapsto mV(v_l)\})(mE(ed_t)) = (((\{mE(e_1), \ldots, mE(e_j)\} \lhd targetG) \cup \{ed_1 \mapsto mV(v_1), \ldots, ed_k \mapsto mV(v_l)\})(mE(ed_t))$*, são executadas algumas regras de forma automática (regras 82-83) e o sub-objetivo gerado é demonstrado executando o provador ML (84). Para descarregar o sequente (III), sub-árvore* $\boxed{C}$*,* $H \vdash (((\{mE(e_1), \ldots, mE(e_j)\} \lhd tG\_E) \cup \{ed_1 \mapsto mV(v_1), \ldots, ed_k \mapsto mV(v_l)\})(mE(e_j)) = tG\_E(x)$*, instancia-se o nome do arco loop* e_j *do tipo* t *na hipótese* `grd_edges`. Com essa ação, dois novos sub-objetivos são gerados: (i) $H \vdash \top$ que é demonstrado de forma automática (86); (ii) $H \vdash (((\{mE(e_i), \ldots, mE(e_j)\} \lhd tG\_E) \cup \{ed_1 \mapsto mV(v_1), \ldots, ed_k \mapsto mV(v_l)\})(mE(e_j)) = tG\_E(x)$. Visando demonstrar (ii), aplica-se a regra partition rewrites na hipótese que define a tipagem dos arcos do lado esquerdo da regra (89). Em seguida, algumas regras são aplicadas de forma automática (regras 90-96), e o sub-objetivo gerado é demonstrado aplicando o provador ML (97).

A árvore de uso desse tipo de obrigação de prova é apresentada nas Figuras 63, 64, 65. A árvore foi dividida em três figuras, assim a utilização da tática será explicada para cada uma delas.

Descrição da Árvore de Uso: *Para utilizar a tática apresentada na Figura 63, instancia-se* $mE(e_j)$ *no objetivo do sequente, onde* e_j *é o arco do tipo* t *preservado pela regra. Após adiciona-se uma sequência de hipóteses* $\neg mE(e_j) = mE(e_1), \ldots, \neg mE(e_j) = mE(e_i)$ *e* $mE(e_j) \in edgeG$*, onde* $\{e_1, \ldots, e_j\}$ *representa o conjunto de arcos deletados pela regra. Em cada objetivo gerado pela diferenciação executa-se o provador NewPP with lasso e no último sub-objetivo aplica-se o ML. Na sub-árvore* $\boxed{A}$ *da Figura 64, adiciona-se as hipóteses* $sourceG(mE(e_j)) = targetG(mE(e_j))$ *e* $sourceLi(e_j) = targetLi(e_j)$*. Sub-árvore* $\boxed{c}$*, instancia-se o nome do arco loop* e_j *na hipótese que define a origem dos arcos do lado esquerdo da regra, em seguida instancia-se novamente* e_j*, porém na hipótese que define o destino dos arcos do lado esquerdo da regra. Nos próximos dois sub-objetivos executam-se os provadores NewPP with lasso e ML, respectivamente. Já na sub-árvore* $\boxed{b}$ *representada pela Figura 65, é necessário aplicar a regra partition rewrites nas hipóteses* $sourceLi$ *e* $targetLi$*, respectivamente. Sub-árvore* $\boxed{B}$*, instanciar o nome do arco preservado* e_j *na hipótese* `grd_edges`*. Aplicar a regra partition rewrites na hipótese que define a tipagem dos arcos do lado esquerdo da regra e executar o provador ML.*

4.6 Propriedade propEdSpSrcT

A propriedade propEdSpSrcT ($\exists x, y \cdot (x \in edgeG \wedge y \in vertG \wedge tG\_V(y) = t \wedge sourceG(x) = y)$ assegura que em qualquer grafo alcançável do sistema existe um arco com origem em um vértice do tipo t. Nessa propriedade para toda regra que altera uma das variáveis $edgeG$, $vertG$, tG_V e $sourceG$ é gerada uma obrigação de prova. No evento de inicialização todas as variáveis são inicializadas, logo é gerada a obrigação INITIALISATION/propEdSpSrcT/INV. Para demonstrar esse tipo de obrigação instancia-se no objetivo x e y, onde x é um arco que possui origem no vértice y do tipo t (x e $y \in \mathbb{N}$). A Figura 66 apresenta a árvore de prova gerada pela demonstração.

$$\dfrac{\dfrac{\text{True}}{\vdash \top}\,2 \qquad \dfrac{\dfrac{\text{True}}{\vdash \top}\,4}{\vdash x \in edgeG \wedge y \in vertG \wedge tG\_V(y) = t \wedge sourceG(x) = y}\,3}{\vdash \exists x, y \cdot x \in edgeG \wedge y \in vertG \wedge tG\_V(y) = t \wedge sourceG(x) = y}\,1$$

1 $\exists$ goal (inst x, y); 2 $\top$ goal; 3 simplification rewrites; 4 $\top$ goal.

Figura 66: Árvore de Prova INITIALISATION/propEdSpSrcT/INV

Descrição da Árvore de Prova: *O sequente a ser demonstrado é* $\vdash \exists x, y \cdot x \in edgeG \wedge y \in vertG \wedge tG\_V(y) = t \wedge sourceG(x) = y$*. A fim de demonstrar esse sequente instancia-se no objetivo o nome do arco* x *e vértice* y*, onde* x *possui origem em* y *e* y *é do tipo* t *(1). Após essa instanciação, são gerados dois novos objetivos: (i)* $\vdash \top$*, demonstrado automaticamente pela regra* $\top$ *goal (2); e (ii)* $\vdash x \in edgeG \wedge y \in vertG \wedge tG\_V(y) = t \wedge sourceG(x) = y$*. Em (ii) é executada de forma automática a regra simplification rewrites (3), gerando o sequente* $\vdash \top$ *que é demonstrado de forma automática pela ferramenta (4).*

A Figura 67 apresenta a árvore de uso dessa tática.

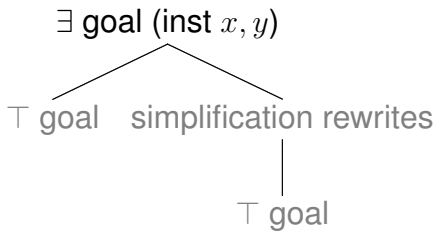


Figura 67: Árvore de Uso - INITIALISATION/propEdSpSrcT/INV

Descrição da Árvore de Uso: *Para utilizar essa estratégia instancia-se no objetivo o nome do arco* x *e o nome do vértice* y *que estão de acordo com a propriedade no*

grafo inicial. Ou seja, x *é um arco e* y *é um vértice do tipo* t*, ambos pertencentes ao grafo inicial e* x *possui origem em* y*.*

Considerando que as regras podem deletar e criar novos arcos e/ou criar novos vértices, assim as obrigações de prova geradas por `propEdSpSrcT` seguem este padrão $\exists x, y \cdot x \in (edgeG \setminus \{mE(e_1), \ldots, mE(e_j)\}) \cup \{ed_1, \ldots, ed_k\} \wedge y \in vertG \cup \{v_1, \ldots, v_l\} \wedge (tG\_V \cup \{v_1 \mapsto t_1, \ldots, v_l \mapsto t_l\})(y) = t \wedge ((\{mE(e_1), \ldots, mE(e_j)\}) \lhd sourceG\{ed_1 \mapsto mV(v_1), \ldots, ed_k \mapsto v_l\})(x) = y$, considerando que j arcos são deletados e k arcos são criados pela regra e/ou l vértices são criados pela aplicação da regra. Desta forma, as táticas para prova dessa propriedade foram divididas para os seguintes casos: regra cria um arco com origem em um vértice do tipo t; regra não envolve arcos com origem em vértices do tipo t; regra preserva, não deleta e não cria arcos com origem em vértices do tipo t; e regra preserva, deleta e não cria arcos com origem em vértices do tipo t.

Nesse trabalho não é considerado o caso em que a regra não preserva, deleta e não cria arcos com origem em vértices do tipo t.

A - Regra cria um arco com origem em um vértice do tipo t

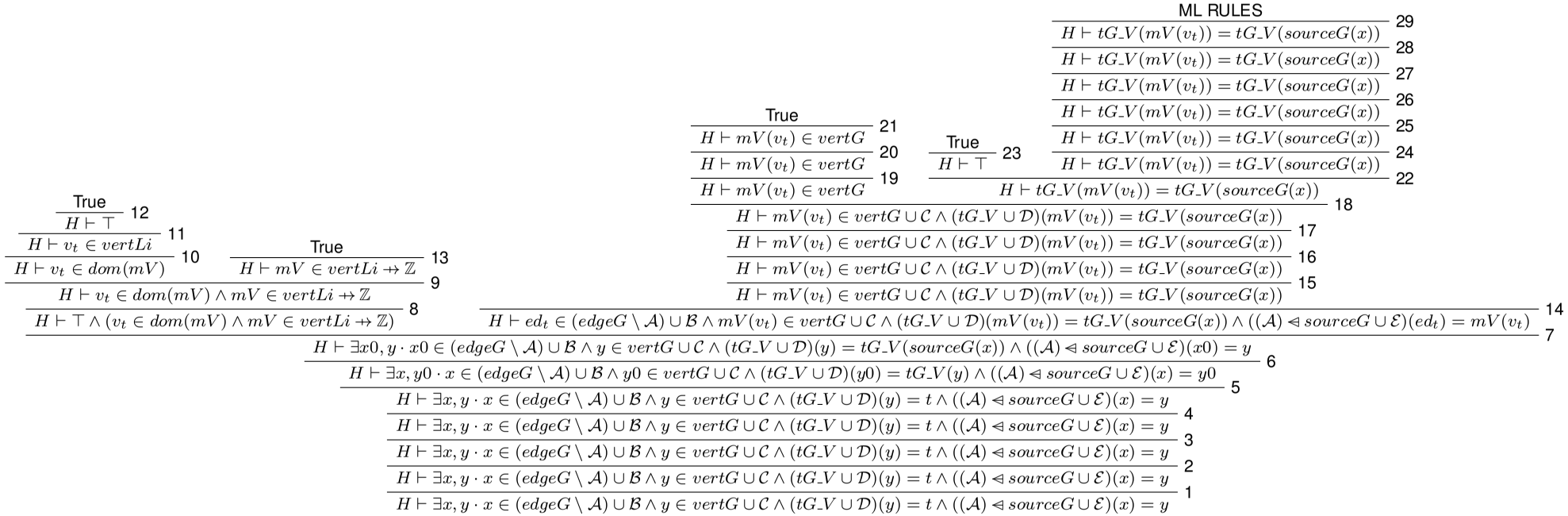
Neste tipo de obrigação de prova, o arco criado pode ter origem em um vértice do tipo t preservado ou criado pela aplicação da regra. No caso, em que o arco criado possui origem em um vértice do tipo t criado, a demonstração é realizada instanciando no objetivo o nome do arco criado ed_i e do vértice criado v_t do tipo t. Já para os casos em que a regra cria um arco em um vértice preservado do tipo t, a demonstração é realizada executando os seguintes passos:

1. Aplicar a regra $\exists$ hyp na hipótese de indução;
2. Instanciar no objetivo o nome do arco criado ed_t juntamente com seu vértice origem $mV(v_t)$ do tipo t que é preservado pela regra;
3. Instanciar o nome do vértice preservado v_t de tipo t na hipótese `grd_vertices`;
4. Aplicar a regra partition rewrites na hipótese que define a tipagem dos vértices do lado esquerdo da regra;
5. Executar o provador ML.

A Figura 68 exibe a árvore de prova resultante da demonstração.

Descrição da Árvore de Prova: *O objetivo inicial da obrigação é* $\exists x, y \cdot x \in (edgeG \setminus \{mE(e_1), \ldots, mE(e_j)\}) \cup \{ed_1, \ldots, ed_k\} \wedge y \in vertG \cup \{v_1, \ldots, v_l\} \wedge (tG\_V \cup \{v_1 \mapsto t_1, \ldots, v_l \mapsto t_l\})(y) = t \wedge ((\{mE(e_1), \ldots, mE(e_j)\}) \lhd sourceG\{ed_1 \mapsto mV(v_1), \ldots, ed_k \mapsto v_l\})(x) = y$*. A fim de demonstrar esse sequente aplica-se a regra* $\exists$ *hyp na hipótese de indução (1), com isso são aplicadas algumas regras de forma automática (regras 2*

$\mathcal{A} = \{mE(e_1), \ldots, mE(e_j)\}$ $\mathcal{B} = \{ed_1, \ldots, ed_k\}$ $\mathcal{C} = \{v_1, \ldots, v_l\}$ $\mathcal{D} = \{v_1 \mapsto t_1, \ldots, v_l \mapsto t_l\}$ $\mathcal{E} = \{ed_1 \mapsto mV(v_1), \ldots, ed_k \mapsto v_l\}$



1 ∃ hyp ($\exists x, y \cdot x \in edgeG \wedge y \in vertG \wedge tG\_V(y) = t \wedge sourceG(x) = y$; 2 simplification rewrites; 3 type rewrites; 4 simplification rewrites; 5 he with $tG\_V(y) = t$; 6 he with $sourceG(x) = y$;
7 ∃ goal (inst ed_t, $mV(v_t)$); 8 simplification rewrites; 9 ∧ goal; 10 total function dom substitution in goal; 11 type rewrites; 12 ⊤ goal; 13 functional goal; 14 simplification rewrites; 15 generalized MP;
16 simplification rewrites; 17 he with $tG\_V(sourceG(x)) = t$; 18 ∧ goal; 19 functional image goal for $mV(v_t)$; 20 functional image goal for $mV(v_t)$; 21 hyp; 22 ∀ hyp (inst v_t); 23 ⊤ goal;
24 Partition rewrites in hyp ($partition(vertLi, \{v_1 \mapsto t_1\}, \ldots, \{v_l \mapsto t_l\})$); 25 simplification rewrites; 26 he with $tG\_V(sourceG(x)) = t$;
27 eh with $tLi\_V = \{v_1 \mapsto t_1, \ldots, v_t \mapsto tG\_V(sourceG(x)), \ldots, v_l \mapsto t_l\}$; 28 simplification rewrites; 29 ML

Figura 68: Árvore de Prova rule_i/propEdSpSrcT/INV - Regra Cria um Arco com Origem em um Vértice do Tipo t

-5), o que resulta em um novo sequente para se demonstrar $H \vdash \exists x0, y \cdot x0 \in (edgeG \setminus \{mE(e_1), \ldots, mE(e_j)) \cup \{ed_1, \ldots, ed_k\} \land y \in vertG \cup \{v_1, \ldots, v_l\} \land (tG\_V \cup \{v_1 \mapsto t_1, \ldots, v_l \mapsto t_l\})(y) = tG\_V(sourceG(x)) \land ((\{mE(e_1), \ldots, mE(e_j)\}) \lhd sourceG \cup \{ed_1 \mapsto mV(v_1), \ldots, ed_k \mapsto v_l\})(x0) = y$*. Para demonstrar esse sequente instancia-se o nome do arco criado* ed_t *pela regra juntamente com seu vértice origem* $mV(v_t)$*. A partir da instanciação são gerados dois novos objetivos: (I)* $H \vdash \top \land (v_t \in dom(mV) \land mV \in vertLi \pfun \mathbb{Z})$*; e (II)* $H \vdash ed_t \in (edgeG \setminus \{mE(e_1), \ldots, mE(e_j)\}) \cup \{ed_1, \ldots, ed_k\} \land mV(v_t) \in vertG \cup \{v_1, \ldots, v_l\} \land (tG\_V \cup \{v_1 \mapsto t_1, \ldots, v_l \mapsto t_l\})(mV(v_t)) = tG\_V(sourceG(x)) \land ((\{mE(e_1), \ldots, mE(e_j)\}) \lhd sourceG \cup \{ed_1 \mapsto mV(v_1), \ldots, ed_k \mapsto v_l\})(ed_t) = mV(v_t)$*. O sequente (I) é demonstrado automaticamente por uma sequência de regras (8-13). Da mesma forma em dois são aplicadas as regras de forma automática (regras 15-21), chegando no objetivo* $H \vdash tG\_V(mV(v_t)) = tG\_V(sourceG(x))$*. Para demonstrar o sequente instancia-se na hipótese* `grd_vertices` *(22) o nome do vértice* v_t*, do tipo* t*, que é origem do arco criado pela regra* ed_t*. Em seguida são gerados dois sequentes: (i)* $H \vdash \top$*, demonstrado automaticamente (23); e (ii)* $H \vdash tG\_V(mV(v_t)) = tG\_V(sourceG(x))$*. Em (ii) aplica-se a regra partition rewrites na hipótese que define a tipagem dos vértices do lado esquerdo da regra (24) e, em seguida, algumas regras são aplicadas automaticamente (regras 25-28) e o objetivo final é descarregado aplicando o provador ML (29).*

Uma forma alternativa de apresentar a tática é representada através da árvore de uso (Figura 69)

Descrição da Árvore de Uso: *Para utilizar a tática aplica-se a regra* $\exists$ *hyp na hipótese de indução. Em seguida instancia-se o nome do arco criado pela regra* ed_t *juntamente com seu vértice origem* $mV(v_t)$ *do tipo* t*, imagem do* match mV*. Então, o nome do arco* v_t *deve ser instanciado na hipótese* `grd_vertices` *e após aplica-se a regra partition rewrites na hipótese que define a tipagem dos vértices do lado esquerdo da regra e executa-se o provador ML.*

B - Regra não envolve arcos com origem em vértices do tipo t

Para esse caso, se a regra não deleta arcos a obrigação é descarregada executando o provador NewPP with lasso. Já para os casos em que a regra deleta arcos a demonstração é realizada executando as seguintes ações:

1. Aplicar a regra $\exists$ hyp na hipótese de indução;
2. Para cada arco e_i deletado pela regra, seguir os passos:
   (a) Adicionar como hipótese $\neg x = mE(e_i)$, onde e_i é um arco deletado pela regra;

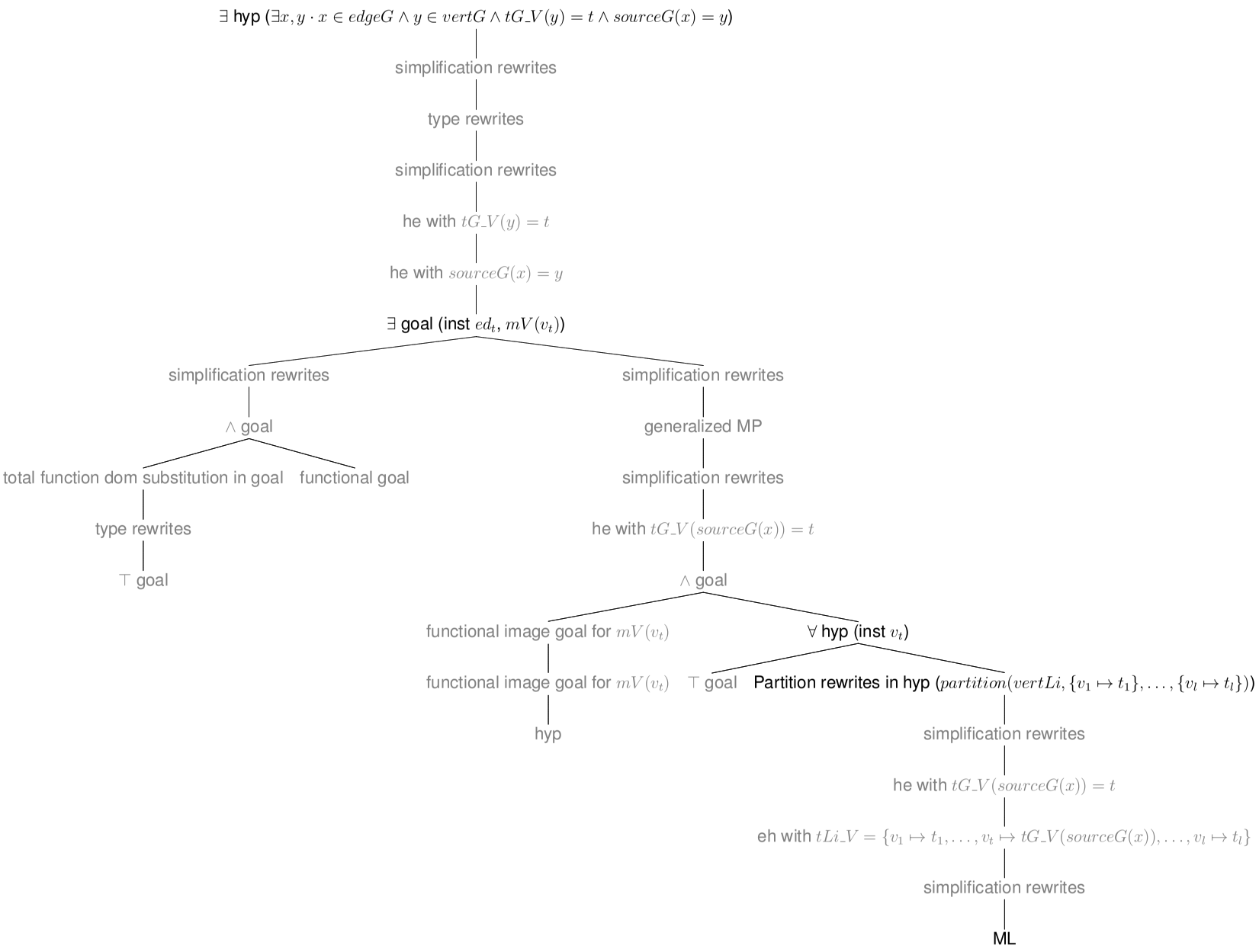


Figura 69: Árvore de Uso - rule_i/propEdSpSrcT/INV - Regra Cria um Arco com Origem em um Vértice do Tipo t

(b) Instanciar o nome do arco deletado e_i na hipótese grd_srctgt considerando $source$;

(c) Aplicar a regra Partition Rewrites na hipótese que define a origem dos arcos do lado esquerdo da regra;

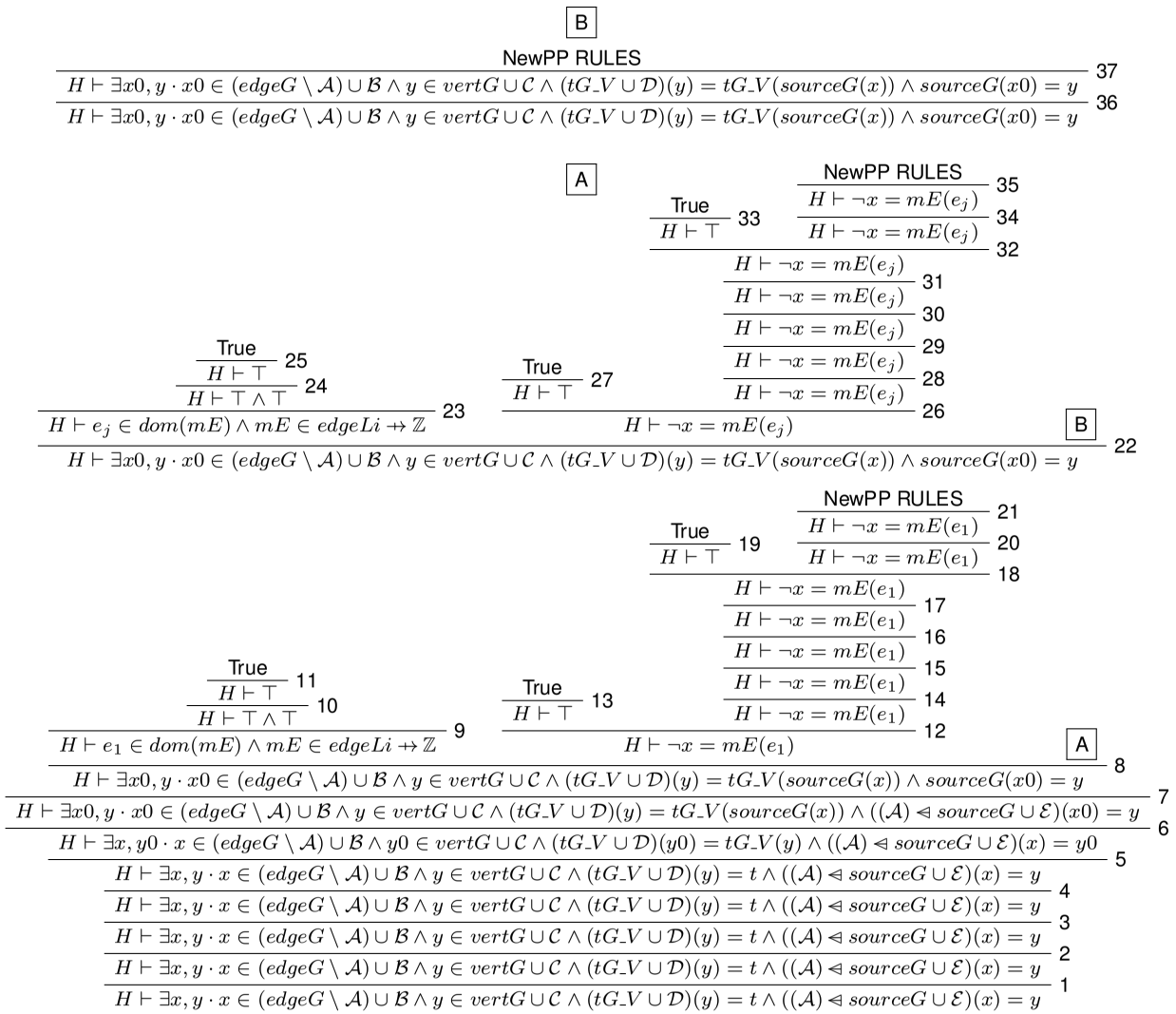
(d) Instanciar o nome do vértice v_i que é origem do arco deletado e_i, no componente grd_vertices;

3. Executar o provador NewPP with lasso.

Nesse caso, a regra não envolve arcos com origem em vértices do tipo t, desta forma a propriedade é assegurada pela hipótese de indução. Para isso, se diferencia o elemento x (arco que satisfaz a propriedade pela hipótese de indução) com cada arco deletado pela regra, ações (a-d) da tática proposta. A Figura 70 apresenta a árvore de prova resultante da demonstração.

Descrição da Árvore de Prova: *O primeiro sequente para se demonstrar é* $H \vdash \exists x, y \cdot x \in (edgeG \setminus \{mE(e_1), \ldots, mE(e_j)\}) \cup \{ed_1, \ldots, ed_k\} \land y \in vertG \cup \{v_1, \ldots, v_l\} \land (tG\_V \cup \{v_1 \mapsto t_1, \ldots, v_l \mapsto t_l\})(y) = t \land ((\{mE(e_1), \ldots, mE(e_j)\}) \lhd sourceG \cup \{ed_1 \mapsto mV(v_1), \ldots, ed_k \mapsto v_l\})(x) = y$. *Visando demonstrar esse sequente, aplica-se a regra* $\exists$ *hyp na hipótese de indução (1), algumas regras de forma automática são aplicadas (regras 2-7), resultando em um novo sequente* $H \vdash \exists x0, y \cdot x0 \in (edgeG \setminus \{mE(e_1), \ldots, mE(e_j)) \cup \{ed_1, \ldots, ed_k\} \land y \in vertG \cup \{v_1, \ldots, v_l\} \land (tG\_V \cup \{v_1 \mapsto t_1, \ldots, v_l \mapsto t_l\})(y) = tG\_V(sourceG(x)) \land sourceG(x0) = y$. *Para demonstrar esse objetivo adiciona-se como hipótese* $\neg x = m(e_1)$, *onde* e_1 *é o primeiro arco deletado pela regra, essa instanciação resulta em três novos objetivos para se demonstrar: (I)* $H \vdash e_1 \in dom(mE) \land mE \in edgeLi \nrightarrow \mathbb{Z}$; *(II)* $H \vdash \neg x = mE(e_1)$; *e (III)* $H \vdash \exists x0, y \cdot x0 \in (edgeG \setminus \{mE(e_1), \ldots, mE(e_j)) \cup \{ed_1, \ldots, ed_k\} \land y \in vertG \cup \{v_1, \ldots, v_l\} \land (tG\_V \cup \{v_1 \mapsto t_1, \ldots, v_l \mapsto t_l\})(y) = tG\_V(sourceG(x)) \land sourceG(x0) = y$. *Em (I) são aplicadas algumas regras que acabam demonstrando o sub-objetivo (regras 9-11). Em (II), instancia-se o nome do arco deletado* e_1 *na hipótese* grd_srctgt *, considerando* $source$, *e essa ação resulta em dois sequentes para se demonstrar: (i)* $H \vdash \top$, *descarregado automaticamente (13); e (ii)* $H \vdash \neg x = mE(e_1)$. *Para demonstrar (ii) aplica-se a regra partition rewrites na hipótese que define a origem dos arcos do lado esquerdo da regra (14), em seguida, algumas regras automaticamente são aplicadas (regras 15-17). Após instancia-se na hipótese* grd_vertices *o nome do vértice* v_1, *que é origem do arco deletado* e_1 *(18), gerando dois novos sub-objetivos: (i)* $H \vdash \top$, *demonstrado de forma automática (19); e (ii)* $H \vdash \neg x = mE(e_1)$, *descarregado pelo provador NewPP with lasso. A sequência de passos, a partir da diferenciação entre* x *e o primeiro arco deletado deve ser executada para cada arco deletado pela regra. Esse processo,*

$\mathcal{A} = \{mE(e_1), \ldots, mE(e_j)\}$ $\mathcal{B} = \{ed_1, \ldots, ed_k\}$ $\mathcal{C} = \{v_1, \ldots, v_l\}$ $\mathcal{D} = \{v_1 \mapsto t_1, \ldots, v_l \mapsto t_l\}$ $\mathcal{E} = \{ed_1 \mapsto mV(v_1), \ldots, ed_k \mapsto v_l\}$



1 ∃ hyp ($\exists x, y \cdot x \in edgeG \wedge y \in vertG \wedge tG\_V(y) = t \wedge sourceG(x) = y$; 2 simplification rewrites; 3 type rewrites; 4 simplification rewrites; 5 he with $tG\_V(y) = t$; 6 he with $sourceG(x) = y$;
7 Functional image simplification in goal; 8 ah $\neg x = m(e_1)$; 9 generalized MP; 10 simplification rewrites; 11 ⊤ goal; 12 ∀ hyp (inst e_1); 13 ⊤ goal;
14 Partition rewrites in hyp ($partition(sourceLi, \{e_1 \mapsto v_1\}, \ldots, \{e_k \mapsto v_l\})$); 15 simplification rewrites; 16 eh with $sourceLi = \{e_1 \mapsto v_1, \ldots, e_k \mapsto v_l\}$; 17 simplification rewrites; 18 ∀ hyp (inst v_1);
19 ⊤ goal; 20 sl/ds; 21 NewPP with lasso; 22 ah $\neg x = m(e_j)$; 23 generalized MP; 24 simplification rewrites; 25 ⊤ goal; 26 ∀ hyp (inst e_j); 27 ⊤ goal;
28 Partition rewrites in hyp ($partition(sourceLi, \{e_1 \mapsto v_1\}, \ldots, \{e_k \mapsto v_l\})$); 29 simplification rewrites; 30 eh with $sourceLi = \{e_1 \mapsto v_1, \ldots, e_k \mapsto v_l\}$; 31 simplification rewrites; 32 ∀ hyp (inst v_l);
33 ⊤ goal; 34 sl/ds; 35 NewPP with lasso; 36 sl/ds; 37 NewPP with lasso

Figura 70: Árvore de Prova rule_i/propEdSpSrcT/INV - Regra Não Envolve Arcos com Origem em Vértices do Tipo t

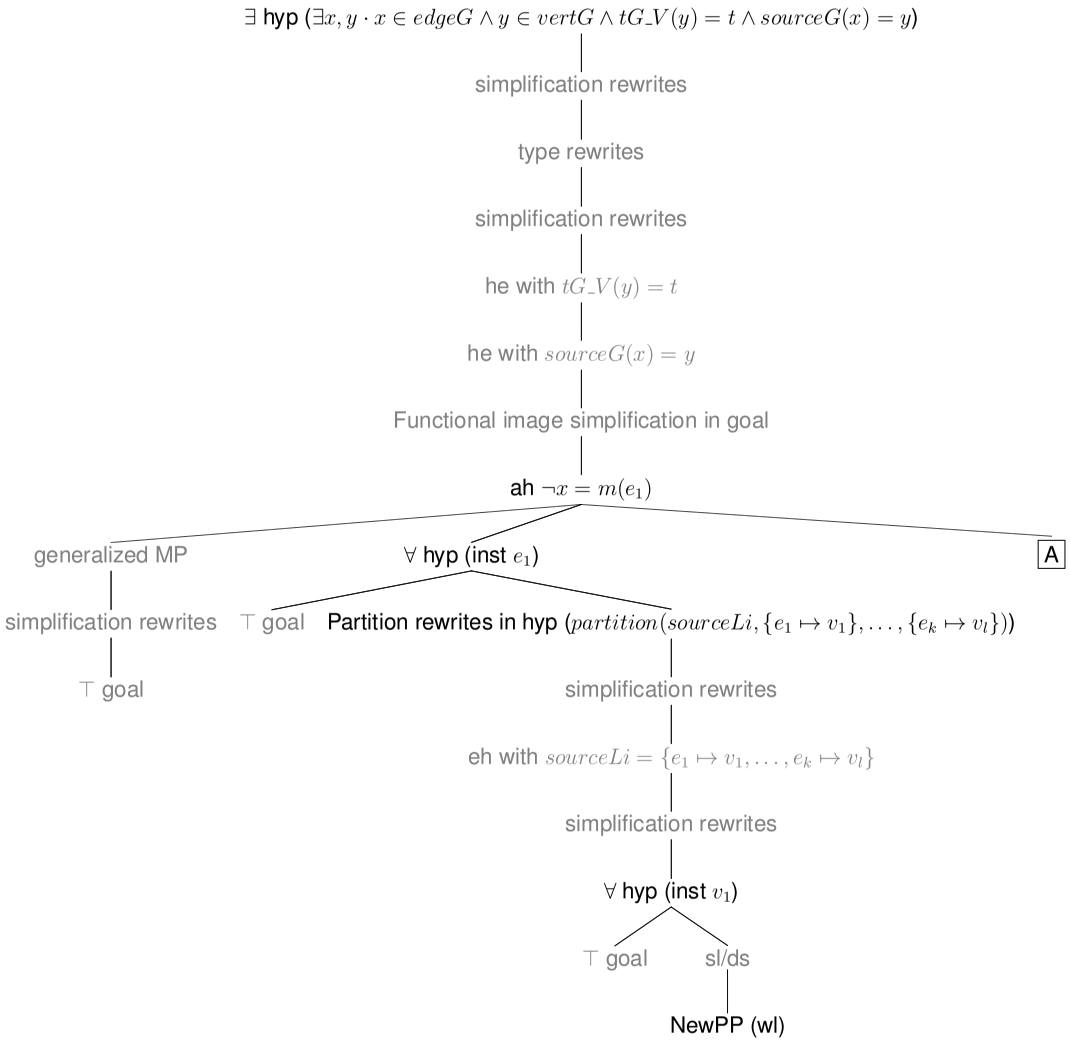


Figura 71: Árvore de Uso - rule_i/propEdSpSrcT/INV - Regra Não Envolve Arcos com Origem em Vértices do Tipo t (1)

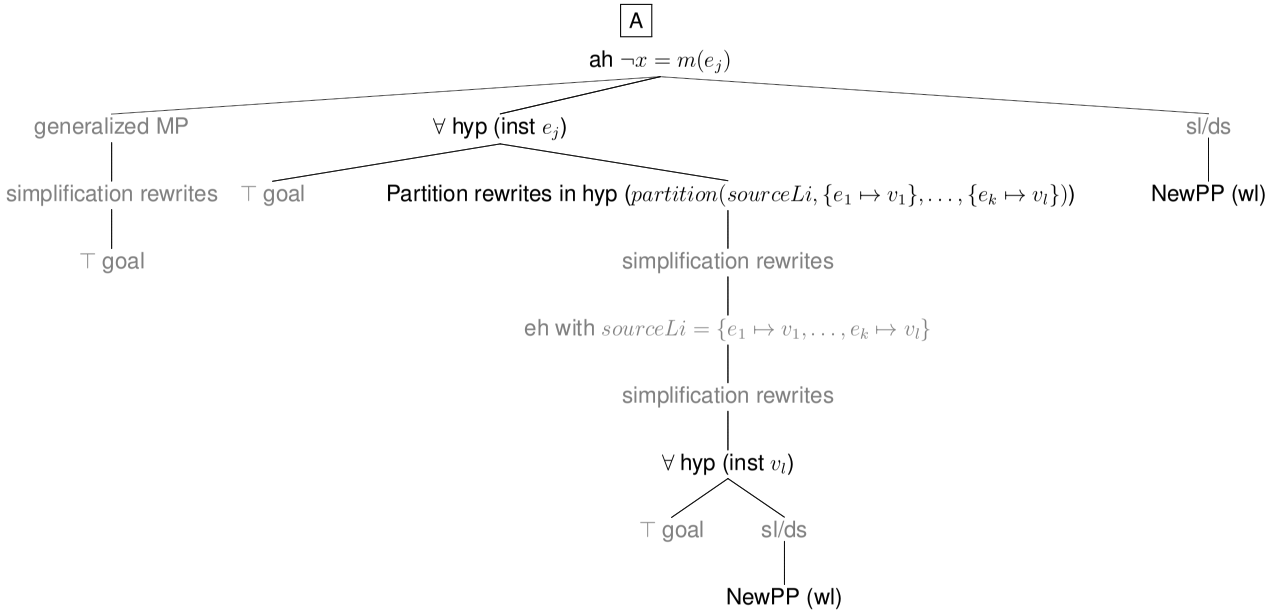


Figura 72: Árvore de Uso - rule_i/propEdSpSrcT/INV - Regra Não Envolve Arcos com Origem em Vértices do Tipo t (2)

representando os j arcos deletados, pode ser analisado na sub-árvore $\boxed{A}$ *(regras 22-35). Após realizadas todas as diferenciações é gerado um novo sequente, sub-árvore* $\boxed{B}$ $H \vdash \exists x0, y \cdot x0 \in (edgeG \setminus \{mE(e_1), \ldots, mE(e_j)) \cup \{ed_1, \ldots, ed_k\} \wedge y \in vertG \cup \{v_1, \ldots, v_l\} \wedge (tG\_V \cup \{v_1 \mapsto t_1, \ldots, v_l \mapsto t_l\})(y) = tG\_V(sourceG(x)) \wedge sourceG(x0) = y$, *que é demonstrado executando o provador NewPP with lasso (37).*

A árvore de uso para esse caso de regra é apresentada nas Figuras 71 e 72

Descrição da Árvore de Uso: *Para utilizar a tática, ilustrada na Figura 71, primeiramente aplica-se a regra $\exists$ hyp na hipótese de indução, em seguida adicionam-se, respectivamente, como hipóteses $\neg x = mE(e_1), \ldots, \neg x = mE(e_j)$, onde $\{e_1, \ldots, e_j\}$ representam os arcos deletados pela regra. Em cada diferenciação aplicam-se as seguintes ações: instancia-se o nome do arco deletado e_i na hipótese* `grd_srctgt`*, considerando $source$, em seguida, aplica-se a regra partition rewrites na hipótese que define a origem dos arcos do lado esquerdo da regra. Após instancia-se na hipótese* `grd_vertices` *o nome do vértice v_i que é origem do arco deletado e_i. Para descarregar o sub-objetivo gerado executa-se o provador NewPP with lasso. Realizadas todas as diferenciações, no último objetivo gerado executa-se o provador NewPP with lasso. A sub-árvore* $\boxed{A}$ *na Figura 72, representa a diferenciação para os j arcos deletados, juntamente com os passos necessários para demonstrar os sub-objetivos criados.*

C - Regra preserva, não deleta e não cria arcos com origem em vértices do tipo t

Existem dois casos distintos nesses tipos de obrigações de prova. O primeiro é relativo a regra que não deleta arcos. Nesse caso a demonstração é realizada executando o provador NewPP with lasso. No segundo caso, arcos com origem em vértices diferentes do tipo t são deletados pela regra. A tática para demonstrar obrigações desse tipo, é apresentada a seguir:

1. Instanciar no objetivo $mE(e_j)$ e $mV(v_t)$, onde e_j é o arco preservado pela regra, possuindo origem no vértice v_t do tipo t;
2. Aplicar a regra Partition Rewrites na hipótese que define o conjunto de arcos do lado esquerdo da regra e executar ML;
3. Instanciar o nome do vértice v_t que é origem do arco preservado e_j no componente `grd_vertices`;
4. Aplicar a regra Partition Rewrites na hipótese que define a tipagem dos vértices do lado esquerdo da regra e executar ML;

5. Instanciar o nome do arco e_j preservado na hipótese grd_srctgt, considerando $source$;
6. Aplicar a regra Partition Rewrites na hipótese que define a origem dos arcos do lado esquerdo da regra;
7. Executar o provador ML;

Após executar os passos da tática, uma árvore de prova é gerada, a qual pode ser analisada nas Figuras 73 e 74.

Descrição da Árvore de Prova: *Conforme Figura 73, o primeiro sequente para se demonstrar é* $H \vdash \exists x, y \cdot x \in (edgeG \setminus \{mE(e_1), \ldots, mE(e_j)\} \cup \{ed_1, \ldots, ed_k\}) \land y \in vertG \land tG\_V(y) = t \land ((\{mE(e_1), \ldots, mE(e_j)\} \lhd sourceG) \cup \{ed_1 \mapsto mV(v_1), \ldots, ed_k \mapsto v_l\})(x) = y$. *A fim de demonstrar esse sequente instancia-se* $mE(e_j)$ *e* $mV(v_t)$*, onde* e_j *é um arco preservado com origem no arco* v_t *do tipo* t *(1). Com essa ação dois novos sub-objetivos são gerados: (I)* $H \vdash (v_t \in dom(mV) \land mV \in vertLi \nrightarrow \mathbb{Z}) \land (ed_t \in dom(mE) \land mE \in edgeLi \nrightarrow \mathbb{Z})$ *e (II)* $H \vdash \exists x, y \cdot x \in (edgeG \setminus \{mE(e_1), \ldots, mE(e_j)\} \cup \{ed_1, \ldots, ed_k\}) \land y \in vertG \land tG\_V(mV(v_t)) = t \land ((\{mE(e_1), \ldots, mE(e_j)\} \lhd sourceG) \cup \{ed_1 \mapsto v_1, \ldots, ed_k \mapsto v_l\})(x) = y$. *Em (I) são aplicadas algumas regras automáticas, que acabam demonstrando todos os sub-objetivos gerados (regras 2-16). Em (II), sub-árvore* A*, a ferramenta aplica de forma automática algumas regras (17-25), gerando quatro sub-objetivos: (i)* $H \vdash mE(e_j) \in (edgeG \setminus \{mE(e_1), \ldots, mE(e_j)\}) \cup \{ed_1, \ldots, ed_k\}$*; (ii)* $H \vdash mV(v_t) \in vertG$*; (iii)* $H \vdash tG\_V(mV(v_t)) = tG\_V(sourceG(x))$*; e (iv)* $H \vdash ((\{mE(e_1), \ldots, mE(e_j)\} \lhd sourceG) \cup \{ed_1 \mapsto v_1, \ldots, ed_k \mapsto v_l\})(mE(ed_t)) = mV(v_t)$. *A fim de demonstrar (i) aplica-se a regra Partition Rewrites (26) na hipótese que define o conjunto dos arcos do lado esquerdo da regra, em seguida é aplicada de forma automática a regra use equality hypothesis - eh (27) e o sub-objetivo e demonstrado por ML (28). O sequente (ii) é demonstrado de forma automática pela regra hyp (31), conforme Figura 74, sub-árvore* B*. Para demonstrar (iii) instancia-se na hipótese* grd_vertices *o nome do vértice* v_t*, que é origem do arco* e_j *(32). Após essa ação são gerados dois novos sub-objetivos: (i)* $H \vdash \top$*, é descarregado de forma automática pela regra* $\top$ *goal (33) e (ii)* $H \vdash tG\_V(mV(v_t)) = tG\_V(sourceG(x))$*. A fim de demonstrar (ii) aplica-se a tática Partition Rewrites (34) na hipótese que define a tipagem dos arcos do lado esquerdo da regra. Em seguida algumas regras são aplicadas automaticamente (regras 34-39) e a demonstração é realizada por ML (40). Visando demonstrar (iv), sub-árvore* C*, instancia-se o nome do arco preservado* e_j *na hipótese* grd_srctgt*, considerando* $source$ *(41). Com isso, são gerados dois novos sub-objetivos: (i)* $H \vdash \top$*, que é descarregado automaticamente (42) e (ii)* $H \vdash ((\{mE(e_1), \ldots, mE(e_j)\} \lhd sourceG) \cup \{ed_1 \mapsto v_1, \ldots, ed_k \mapsto v_l\})(mE(e_j)) = mV(v_t)$.

$\mathcal{A} = \{mE(e_1), \ldots, mE(e_j)\}$ $\quad \mathcal{B} = dom(mE) \wedge mE \in edgeLi \rightarrowtail \mathbb{Z}$ $\quad \mathcal{C} = \{ed_1, \ldots, ed_k\}$ $\quad \mathcal{E} = \{ed_1 \mapsto mV(v_1), \ldots, ed_k \mapsto v_l\}$ $\quad \mathcal{F} = dom(mV) \wedge mV \in vertLi \rightarrowtail \mathbb{Z}$

ML RULES

$$\frac{\text{ML RULES}}{H \vdash mE(e_j) \in (edgeG \setminus \mathcal{A}) \cup \mathcal{C}}\ 28$$

$$\frac{H \vdash mE(e_j) \in (edgeG \setminus \mathcal{A}) \cup \mathcal{C}}{H \vdash mE(e_j) \in (edgeG \setminus \mathcal{A}) \cup \mathcal{C}}\ 27$$

$$\frac{H \vdash mE(e_j) \in (edgeG \setminus \mathcal{A}) \cup \mathcal{C}}{H \vdash mE(e_j) \in (edgeG \setminus \mathcal{A}) \cup \mathcal{C}}\ 26$$

A

$$\frac{\text{True}}{H \vdash mV(v_t) \in vertG}\ 31$$

$$\frac{H \vdash mV(v_t) \in vertG}{H \vdash mV(v_t) \in vertG}\ 30$$

$$\frac{H \vdash mV(v_t) \in vertG}{H \vdash mV(v_t) \in vertG}\ 29$$

B C

$$\frac{(26) \quad (29) \quad B \quad C}{H \vdash mE(e_j) \in (edgeG \setminus \mathcal{A} \cup \mathcal{C}) \wedge mV(v_t) \in vertG \wedge tG\_V(mV(v_t)) = tG\_V(sourceG(x)) \wedge ((\mathcal{A} \lhd sourceG) \cup \mathcal{E})(mE(e_j)) = mV(v_t)}\ 25$$

$$\frac{\ldots}{H \vdash mE(e_j) \in (edgeG \setminus \mathcal{A} \cup \mathcal{C}) \wedge mV(v_t) \in vertG \wedge tG\_V(mV(v_t)) = tG\_V(y) \wedge ((\mathcal{A} \lhd sourceG) \cup \mathcal{E})(mE(e_j)) = mV(v_t)}\ 24$$

$$\frac{\ldots}{H \vdash mE(e_j) \in (edgeG \setminus \mathcal{A} \cup \mathcal{C}) \wedge mV(v_t) \in vertG \wedge tG\_V(mV(v_t)) = t \wedge ((\mathcal{A} \lhd sourceG) \cup \mathcal{E})(mE(e_j)) = mV(v_t)}\ 23$$

$$\frac{\ldots}{H \vdash mE(e_j) \in (edgeG \setminus \mathcal{A} \cup \mathcal{C}) \wedge mV(v_t) \in vertG \wedge tG\_V(mV(v_t)) = t \wedge ((\mathcal{A} \lhd sourceG) \cup \mathcal{E})(mE(e_j)) = mV(v_t)}\ 22$$

$$\frac{\ldots}{H \vdash mE(e_j) \in (edgeG \setminus \mathcal{A} \cup \mathcal{C}) \wedge mV(v_t) \in vertG \wedge tG\_V(mV(v_t)) = t \wedge ((\mathcal{A} \lhd sourceG) \cup \mathcal{E})(mE(e_j)) = mV(v_t)}\ 21$$

$$\frac{\ldots}{H \vdash mE(e_j) \in (edgeG \setminus \mathcal{A} \cup \mathcal{C}) \wedge mV(v_t) \in vertG \wedge tG\_V(mV(v_t)) = t \wedge ((\mathcal{A} \lhd sourceG) \cup \mathcal{E})(mE(e_j)) = mV(v_t)}\ 20$$

$$\frac{\ldots}{H \vdash mE(e_j) \in (edgeG \setminus \mathcal{A} \cup \mathcal{C}) \wedge mV(v_t) \in vertG \wedge tG\_V(mV(v_t)) = t \wedge ((\mathcal{A} \lhd sourceG) \cup \mathcal{E})(mE(e_j)) = mV(v_t)}\ 19$$

$$\frac{\ldots}{H \vdash mE(e_j) \in (edgeG \setminus \mathcal{A} \cup \mathcal{C}) \wedge mV(v_t) \in vertG \wedge tG\_V(mV(v_t)) = t \wedge ((\mathcal{A} \lhd sourceG) \cup \mathcal{E})(mE(e_j)) = mV(v_t)}\ 18$$

$$\frac{\ldots}{H \vdash mE(e_j) \in (edgeG \setminus \mathcal{A} \cup \mathcal{C}) \wedge mV(v_t) \in vertG \wedge tG\_V(mV(v_t)) = t \wedge ((\mathcal{A} \lhd sourceG) \cup \mathcal{E})(mE(e_j)) = mV(v_t)}\ 17$$

$$\frac{\text{True}}{H \vdash \top}\ 12 \qquad \frac{H \vdash \top}{H \vdash v_t \in vertLi}\ 11 \qquad \frac{H \vdash v_t \in vertLi}{H \vdash v_t \in dom(mV)}\ 10$$

$$\frac{\text{True}}{H \vdash mV \in vertLi \rightarrowtail \mathbb{Z}}\ 13$$

$$\frac{\text{True}}{H \vdash \top}\ 16 \qquad \frac{H \vdash \top}{H \vdash e_j \in edgeLi}\ 15 \qquad \frac{H \vdash e_j \in edgeLi}{H \vdash e_j \in dom(mE)}\ 14$$

$$\frac{H \vdash v_t \in dom(mV) \quad H \vdash mV \in vertLi \rightarrowtail \mathbb{Z} \quad H \vdash e_j \in dom(mE)}{H \vdash v_t \in \mathcal{F} \wedge e_j \in dom(mE)}\ 9$$

$$\frac{H \vdash v_t \in \mathcal{F} \wedge e_j \in dom(mE)}{H \vdash v_t \in \mathcal{F} \wedge e_j \in dom(mE)}\ 8$$

$$\frac{H \vdash v_t \in \mathcal{F} \wedge e_j \in dom(mE)}{H \vdash v_t \in \mathcal{F} \wedge e_j \in dom(mE)}\ 7$$

$$\frac{H \vdash v_t \in \mathcal{F} \wedge e_j \in dom(mE)}{H \vdash v_t \in \mathcal{F} \wedge e_j \in dom(mE)}\ 6$$

$$\frac{H \vdash v_t \in \mathcal{F} \wedge e_j \in dom(mE)}{H \vdash v_t \in \mathcal{F} \wedge e_j \in dom(mE)}\ 5$$

$$\frac{H \vdash v_t \in \mathcal{F} \wedge e_j \in dom(mE)}{H \vdash v_t \in \mathcal{F} \wedge e_j \in dom(mE) \wedge \top}\ 4$$

$$\frac{H \vdash v_t \in \mathcal{F} \wedge e_j \in dom(mE) \wedge \top}{H \vdash v_t \in \mathcal{F} \wedge e_j \in \mathcal{B}}\ 3$$

$$\frac{H \vdash v_t \in \mathcal{F} \wedge e_j \in \mathcal{B}}{H \vdash (v_t \in \mathcal{F}) \wedge (e_j \in \mathcal{B})}\ 2$$

A

$$\frac{H \vdash (v_t \in \mathcal{F}) \wedge (e_j \in \mathcal{B}) \quad A}{H \vdash \exists x, y \cdot x \in (edgeG \setminus \mathcal{A} \cup \mathcal{C}) \wedge y \in vertG \wedge tG\_V(y) = t \wedge ((\mathcal{A} \lhd sourceG) \cup \mathcal{E})(x) = y}\ 1$$

1 $\exists$ goal (inst $mE(e_j), mV(v_t)$); 2 simplification rewrites; 3 generalized MP; 4 simplification rewrites; 5 type rewrites; 6 simplification rewrites; 7 $\exists$ hyp ($\exists x, y \cdot x \in edgeG \wedge y \in vertG \wedge tG\_V(y) = t \wedge sourceG(x) = y$); 8 he with $sourceG(x) = y$; 9 $\top$ goal; 10 total function dom substitution in goal; 11 type rewrites; 12 $\top$ goal; 13 functional goal; 14 total function dom substitution in goal; 15 type rewrites; 16 $\top$ goal; 17 simplification rewrites; 18 generalized MP; 19 simplification rewrites; 20 type rewrites; 21 simplification rewrites; 22 $\exists$ hyp ($\exists x, y \cdot x \in edgeG \wedge y \in vertG \wedge tG\_V(y) = t \wedge sourceG(x) = y$); 23 he with $tG\_V(y) = t$; 24 he with $sourceG(x) = y$; 25 $\wedge$ goal; 26 Partition rewrites in hyp ($partition(edgeLi, \{ed_1\}, \ldots, \{ed_k\})$); 27 eh with $edgeLi = \{ed_1, \ldots, ed_k\}$; 28 ML; 29 functional image goal for $mV(v_t)$; 30 functional image goal for $mV(v_t)$; 31 hyp;

Figura 73: Árvore de Prova rule_i/propEdSpSrcT/INV - Regra Preserva, não Deleta e não Cria Arcos com Origem em Vértices do Tipo t (1)

$\mathcal{A} = \{mE(e_1), \ldots, mE(e_j)\}$ $\mathcal{B} = dom(mE) \land mE \in edgeLi \nrightarrow \mathbb{Z}$ $\mathcal{C} = \{ed_1, \ldots, ed_k\}$ $\mathcal{D} = \{ed_1 \mapsto mV(v_1), \ldots, ed_k \mapsto mV(v_l)\}$
$\mathcal{E} = \{ed_1 \mapsto v_1, \ldots, ed_k \mapsto v_l\}$ $\mathcal{F} = dom(mV) \land mV \in vertLi \nrightarrow \mathbb{Z}$

C

$$\dfrac{\dfrac{\text{True}}{H \vdash \top}\,42 \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\text{ML RULES}}{H \vdash ((\mathcal{A} \lhd sourceG) \cup \mathcal{E})(mE(e_j)) = mV(v_t)}\,50}{H \vdash ((\mathcal{A} \lhd sourceG) \cup \mathcal{E})(mE(e_j)) = mV(v_t)}\,49}{H \vdash ((\mathcal{A} \lhd sourceG) \cup \mathcal{E})(mE(e_j)) = mV(v_t)}\,48}{H \vdash ((\mathcal{A} \lhd sourceG) \cup \mathcal{E})(mE(e_j)) = mV(v_t)}\,47}{H \vdash ((\mathcal{A} \lhd sourceG) \cup \mathcal{E})(mE(e_j)) = mV(v_t)}\,46}{H \vdash ((\mathcal{A} \lhd sourceG) \cup \mathcal{E})(mE(e_j)) = mV(v_t)}\,45}{H \vdash ((\mathcal{A} \lhd sourceG) \cup \mathcal{E})(mE(e_j)) = mV(v_t)}\,44}{H \vdash ((\mathcal{A} \lhd sourceG) \cup \mathcal{E})(mE(e_j)) = mV(v_t)}\,43}{H \vdash ((\mathcal{A} \lhd sourceG) \cup \mathcal{E})(mE(e_j)) = mV(v_t)}\,41$$

B

$$\dfrac{\dfrac{\text{True}}{H \vdash \top}\,33 \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\text{ML RULES}}{H \vdash tG\_V(mV(v_t)) = tG\_V(sourceG(x))}\,40}{H \vdash tG\_V(mV(v_t)) = tG\_V(sourceG(x))}\,39}{H \vdash tG\_V(mV(v_t)) = tG\_V(sourceG(x))}\,38}{H \vdash tG\_V(mV(v_t)) = tG\_V(sourceG(x))}\,37}{H \vdash tG\_V(mV(v_t)) = tG\_V(sourceG(x))}\,36}{H \vdash tG\_V(mV(v_t)) = tG\_V(sourceG(x))}\,35}{H \vdash tG\_V(mV(v_t)) = tG\_V(sourceG(x))}\,34}{H \vdash tG\_V(mV(v_t)) = tG\_V(sourceG(x))}\,32$$

32 ∀ hyp (inst v_t); 33 ⊤ goal; 34 Partition rewrites in hyp ($partition(tLi\_V, \{ed_1 \mapsto t_1\}, \ldots, \{ed_k \mapsto t_k\})$); 35 generalized MP; 36 simplification rewrites; 37 he with $tG\_V(sourceG(x)) = t$; 38 eh with $tLi\_V = \{ed_1 \mapsto t_1, e_j \mapsto tG\_V(sourceG(x)), \ldots, ed_k \mapsto t_k\}$; 39 simplification rewrites; 40 ML; 41 ∀ hyp (inst e_j); 42 ⊤ goal; 43 Partition rewrites in hyp ($partition(sourceLi, \{ed_1 \mapsto v_1\}, \ldots, \{ed_k \mapsto v_l\})$); 44 simplification rewrites; 45 remove ¬ in $\neg(e_j = ed_1 \land v_t = v_1)$; 46 remove ¬ in $\neg(e_j = ed_k \land v_t = v_l)$; 47 remove ¬ in $\neg(ed_1 = ed_k \land v_1 = v_l)$; 48 eh with $sourceLi = \{ed_1 \mapsto v_1, \ldots, ed_k \mapsto v_l\}$; 49 simplification rewrites; 50 ML

Figura 74: Árvore de Prova rule_i/propEdSpSrcT/INV - Regra Preserva, não Deleta e não Cria Arcos com Origem em Vértices do Tipo t (2)

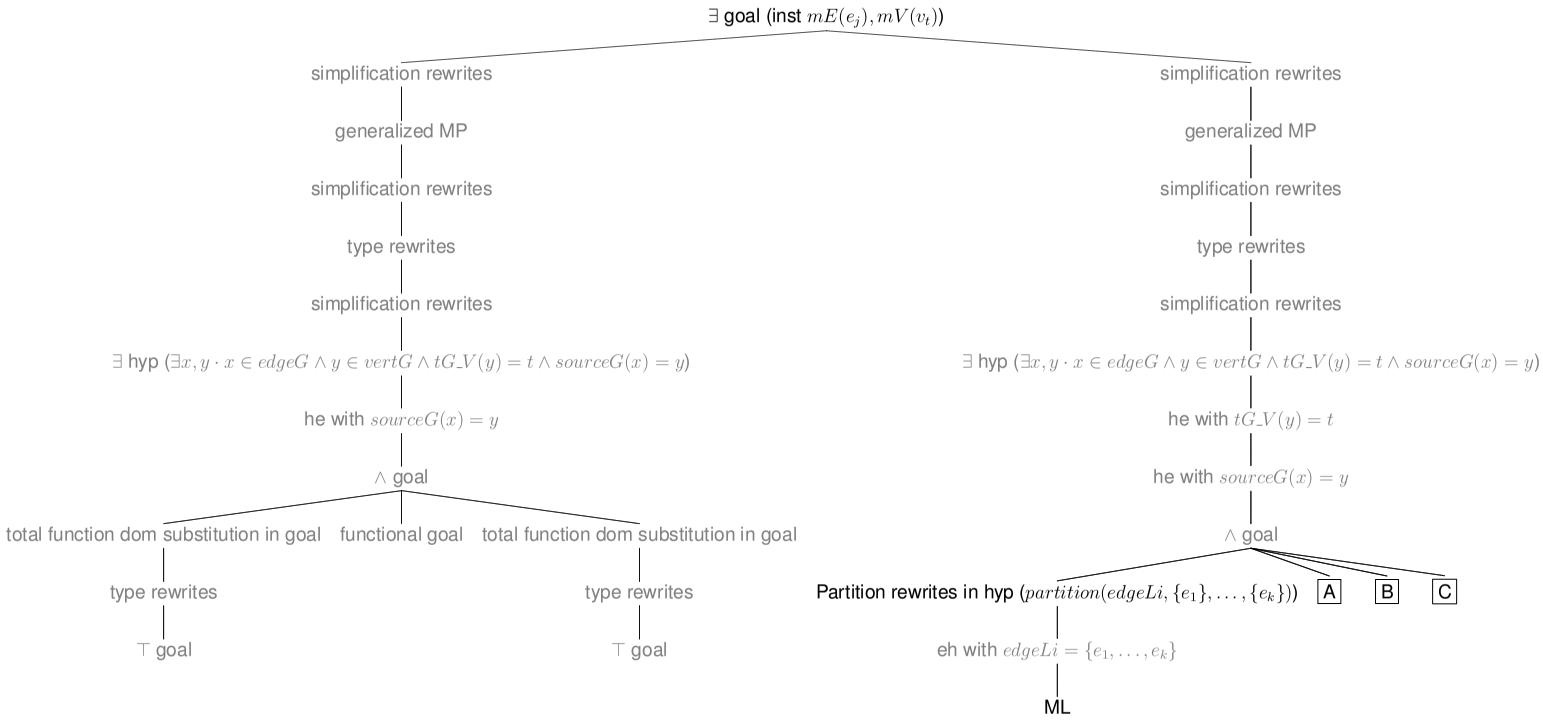


Figura 75: Árvore de Uso rule_i/propEdSpSrcT/INV - Regra Preserva, não Deleta e não Cria Arcos com Origem em Vértices do Tipo t (1)

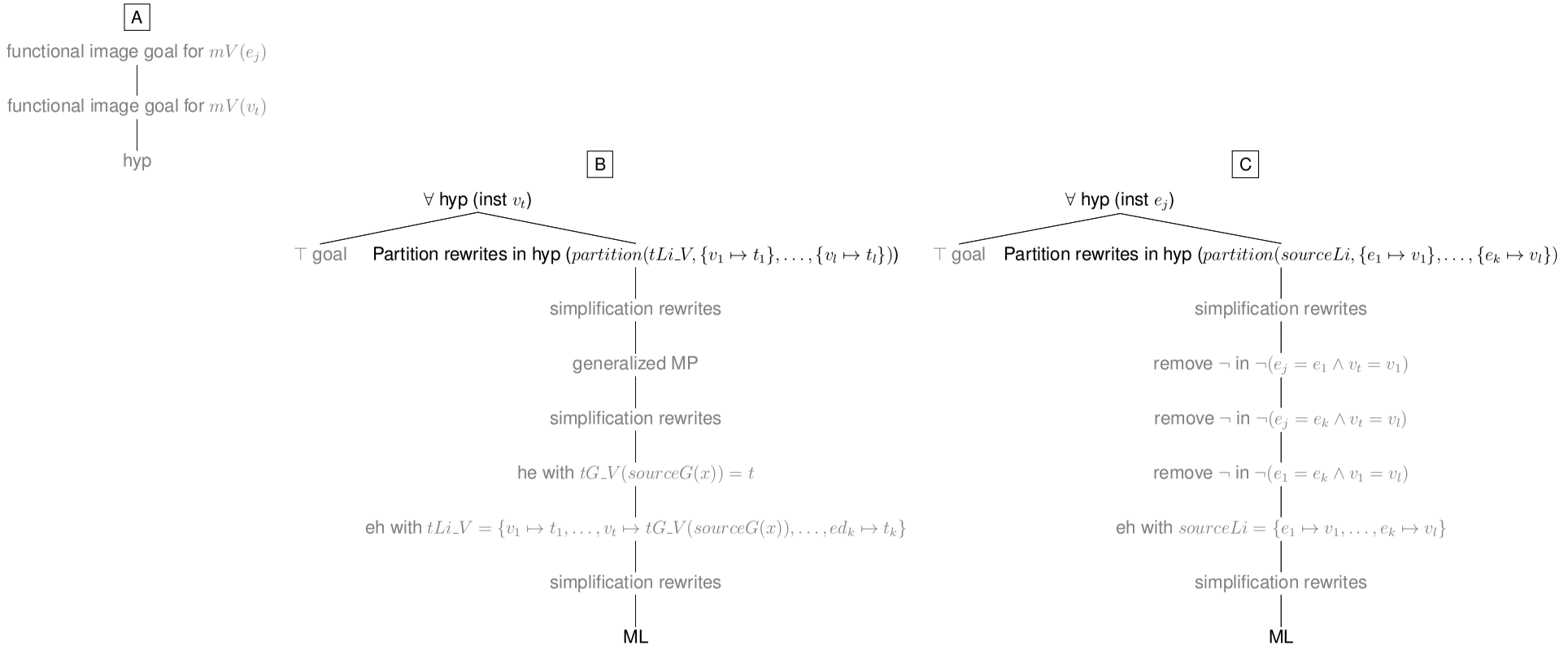


Figura 76: Árvore de Uso rule_i/propEdSpSrcT/INV - Regra Preserva, não Deleta e não Cria Arcos com Origem em Vértices do Tipo t (2)

Para demonstrar (ii) aplica-se a regra Partition Rewrites na hipótese que define a origem dos arcos do lado esquerdo da regra (43). De forma automática a ferramenta aplica algumas regras (44-49), para concluir a prova da obrigação executa-se o provador ML (50).

As Figuras 75 e 76 apresentam a árvore de uso para a obrigação rule_i/propEdSpSrcT/INV.

Descrição da Árvore de Uso: *Conforme Figura 75, para utilizar essa estratégia, instancia-se $mE(e_j)$ e $mV(v_t)$, onde e_j é o arco preservado com origem no vértice v_t do tipo t. Em seguida, é necessário aplicar a regra Partition Rewrites na hipótese que define os arcos do lado esquerdo da regra e executar ML. Na sub-árvore B da Figura 76, instancia-se o nome do vértice v_t que é origem do arco preservado e_j na hipótese* `grd_vertices`*. Aplica-se a regra Partition Rewrites na hipótese que define a tipagem dos vértices do lado esquerdo da regra e executa-se o provador ML. Na sub-árvore C, instancia-se o nome do arco e_j na hipótese* `grd_srctgt`*, considerando $source$, em seguida, aplica-se a regra Partition Rewrites na hipótese que define a origem dos arcos do lado esquerdo da regra e executa-se o provador ML.*

4.7 Propriedade `propEdSpTgtT`

A propriedade `propEdSpTgtT` ($\exists x, y \cdot (x \in edgeG \wedge y \in vertG \wedge tG\_V(y) = t \wedge targetG(x) = y)$ assegura que em qualquer grafo alcançável do sistema existe um arco com destino em um vértice do tipo t. Nessa propriedade para toda regra que altera uma das variáveis $edgeG$, $vertG$, tG_V e $targetG$ é gerada uma obrigação de prova. No evento de inicialização todas as variáveis são inicializadas, logo é gerada a obrigação INITIALISATION/propEdSpTgtT/INV. Para demonstrar esse tipo de obrigação instancia-se no objetivo x e y, onde x é um arco que possui destino no vértice y do tipo t (x e $y \in \mathbb{N}$). A Figura 77 apresenta a árvore de prova gerada pela demonstração.

$$\dfrac{\dfrac{\text{True}}{\vdash \top}\,2 \qquad \dfrac{\dfrac{\text{True}}{\vdash \top}\,4}{\vdash x \in edgeG \wedge y \in vertG \wedge tG\_V(y) = t \wedge targetG(x) = y}\,3}{\vdash \exists x, y \cdot x \in edgeG \wedge y \in vertG \wedge tG\_V(y) = t \wedge targetG(x) = y}\,1$$

1 $\exists$ goal (inst x, y); 2 $\top$ goal; 3 simplification rewrites; 4 $\top$ goal.

Figura 77: Árvore de Prova INITIALISATION/propEdSpTgtT/INV

Descrição da Árvore de Prova: *O sequente a ser demonstrado é $\vdash \exists x, y \cdot x \in edgeG \wedge y \in vertG \wedge tG\_V(y) = t \wedge targetG(x) = y$. A fim de demonstrar esse sequente*

instancia-se no objetivo o nome do arco x e vértice y, onde x possui destino em y e y é do tipo t (1). Após essa instanciação, são gerados dois novos objetivos: (i) $\vdash \top$, demonstrado automaticamente pela regra $\top$ goal (2); e (ii) $\vdash x \in edgeG \wedge y \in vertG \wedge tG\_V(y) = t \wedge targetG(x) = y$. Em (ii) é executada de forma automática a regra simplification rewrites (3), gerando o sequente $\vdash \top$ que é demonstrado de forma automática pela ferramenta (4).

A Figura 78 apresenta a árvore de uso dessa tática.

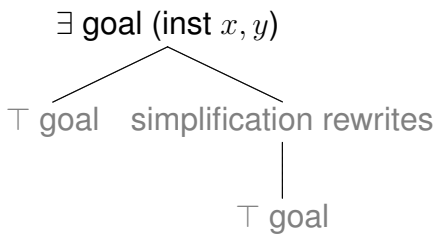


Figura 78: Árvore de Uso - INITIALISATION/propEdSpTgtT/INV

Descrição da Árvore de Uso: *Para utilizar essa estratégia instancia-se no objetivo o nome do arco x e o nome do vértice y que estão de acordo com a propriedade no grafo inicial. Ou seja, x é um arco e y é um vértice do tipo t, ambos pertencentes ao grafo inicial e x possui destino em y.*

Considerando que as regras podem deletar e criar novos arcos e/ou criar novos vértices, assim as obrigações de prova geradas por `propEdSpTgtT` seguem este padrão $\exists x, y \cdot x \in (edgeG \setminus \{mE(e_1), \ldots, mE(e_j)\}) \cup \{ed_1, \ldots, ed_k\} \wedge y \in vertG \cup \{v_1, \ldots, v_l\} \wedge (tG\_V \cup \{v_1 \mapsto t_1, \ldots, v_l \mapsto t_l\})(y) = t \wedge ((\{mE(e_1), \ldots, mE(e_j)\}) \lhd targetG\{ed_1 \mapsto mV(v_1), \ldots, ed_k \mapsto v_l\})(x) = y$, considerando que j arcos são deletados e k arcos são criados pela regra e/ou l vértices são criados pela aplicação da regra. Desta forma, as táticas para prova dessa propriedade foram divididas para os seguintes casos: regra cria um arco com destino em um vértice do tipo t; regra não envolve arcos com destino em vértices do tipo t; regra preserva, não deleta e não cria arcos com destino em vértices do tipo t; e regra preserva, deleta e não cria arcos com destino em vértices do tipo t.

Nesse trabalho não é considerado o caso em que a regra não preserva, deleta e não cria arcos com destino em vértices do tipo t.

A - Regra cria um arco com destino em um vértice do tipo t

Neste tipo de obrigação de prova, o arco criado pode ter destino em um vértice do tipo t preservado ou criado pela aplicação da regra. No caso, em que o arco criado possui destino em um vértice do tipo t criado, a demonstração é realizada instanciando no objetivo o nome do arco criado ed_i e do vértice criado v_t do tipo t. Já para os casos

em que a regra cria um arco em um vértice preservado do tipo t, a demonstração é realizada executando os seguintes passos:

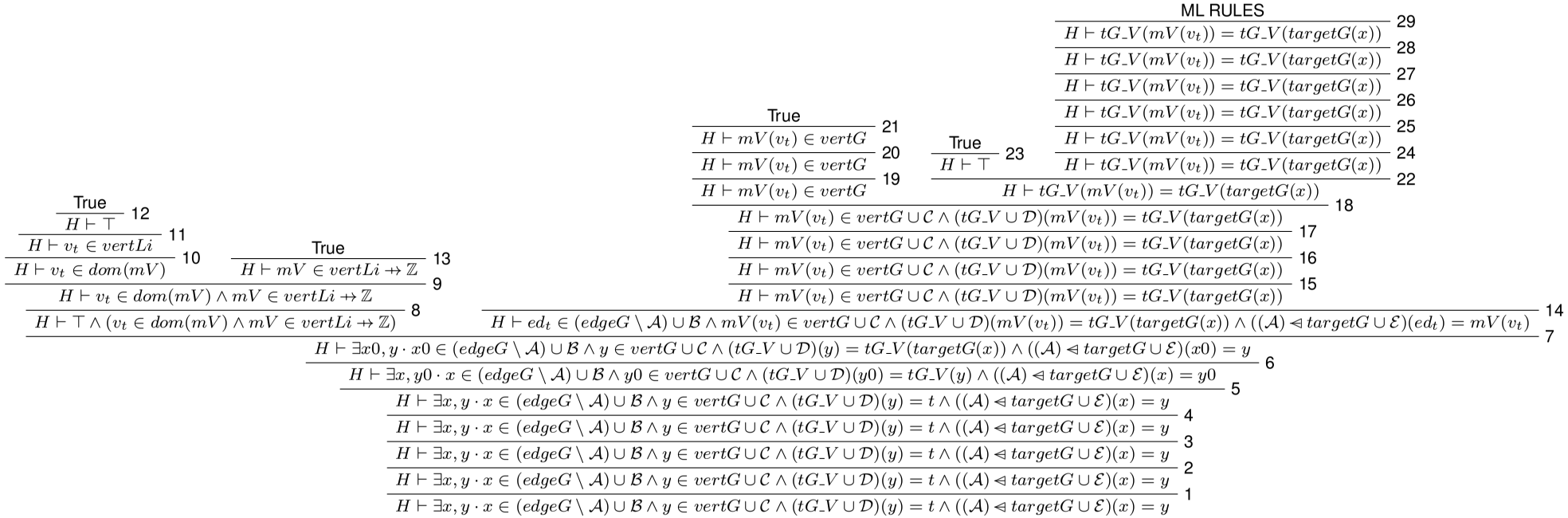
1. Aplicar a regra $\exists$ hyp na hipótese de indução;
2. Instanciar no objetivo o nome do arco criado ed_t juntamente com seu vértice destino $mV(v_t)$ do tipo t que é preservado pela regra;
3. Instanciar o nome do vértice preservado v_t de tipo t na hipótese `grd_vertices`;
4. Aplicar a regra partition rewrites na hipótese que define a tipagem dos vértices do lado esquerdo da regra;
5. Executar o provador ML.

A Figura 79 exibe a árvore de prova resultante da demonstração.

Descrição da Árvore de Prova: *O objetivo inicial da obrigação é* $\exists x, y \cdot x \in (edgeG \setminus \{mE(e_1), \ldots, mE(e_j)\}) \cup \{ed_1, \ldots, ed_k\} \wedge y \in vertG \cup \{v_1, \ldots, v_l\} \wedge (tG\_V \cup \{v_1 \mapsto t_1, \ldots, v_l \mapsto t_l\})(y) = t \wedge (\{mE(e_1), \ldots, mE(e_j)\}) \lhd targetG\{ed_1 \mapsto mV(v_1), \ldots, ed_k \mapsto v_l\})(x) = y$*. A fim de demonstrar esse sequente aplica-se a regra* $\exists$ *hyp na hipótese de indução (1), com isso são aplicadas algumas regras de forma automática (regras 2-5), o que resulta em um novo sequente para se demonstrar* $H \vdash \exists x0, y \cdot x0 \in (edgeG \setminus \{mE(e_1), \ldots, mE(e_j)\}) \cup \{ed_1, \ldots, ed_k\} \wedge y \in vertG \cup \{v_1, \ldots, v_l\} \wedge (tG\_V \cup \{v_1 \mapsto t_1, \ldots, v_l \mapsto t_l\})(y) = tG\_V(targetG(x)) \wedge (\{mE(e_1), \ldots, mE(e_j)\}) \lhd targetG \cup \{ed_1 \mapsto mV(v_1), \ldots, ed_k \mapsto v_l\})(x0) = y$*. Para demonstrar esse sequente instancia-se o nome do arco criado* ed_t *pela regra juntamente com seu vértice destino* $mV(v_t)$*. A partir da instanciação são gerados dois novos objetivos: (I)* $H \vdash \top \wedge (v_t \in dom(mV) \wedge mV \in vertLi \nrightarrow \mathbb{Z})$*; e (II)* $H \vdash ed_t \in (edgeG \setminus \{mE(e_1), \ldots, mE(e_j)\}) \cup \{ed_1, \ldots, ed_k\} \wedge mV(v_t) \in vertG \cup \{v_1, \ldots, v_l\} \wedge (tG\_V \cup \{v_1 \mapsto t_1, \ldots, v_l \mapsto t_l\})(mV(v_t)) = tG\_V(targetG(x)) \wedge (\{mE(e_1), \ldots, mE(e_j)\}) \lhd targetG \cup \{ed_1 \mapsto mV(v_1), \ldots, ed_k \mapsto v_l\})(ed_t) = mV(v_t)$*. O sequente (I) é demonstrado automaticamente por uma sequência de regras (8-13). Da mesma forma em dois são aplicadas as regras de forma automática (regras 15-21), chegando no objetivo* $H \vdash tG\_V(mV(v_t)) = tG\_V(targetG(x))$*. Para demonstrar o sequente instancia-se na hipótese* `grd_vertices` *(22) o nome do vértice* v_t*, do tipo* t*, que é destino do arco criado pela regra* ed_t*. Em seguida são gerados dois sequentes: (i)* $H \vdash \top$*, demonstrado automaticamente (23); e (ii)* $H \vdash tG\_V(mV(v_t)) = tG\_V(targetG(x))$*. Em (ii) aplica-se a regra partition rewrites na hipótese que define a tipagem dos vértices do lado esquerdo da regra (24) e, em seguida, algumas regras são aplicadas automaticamente (regras 25-28) e o objetivo final é descarregado aplicando o provador ML (29).*

Uma forma alternativa de apresentar a tática é representada através da árvore de uso (Figura 80).

$\mathcal{A}$ = $\{mE(e_1), \ldots, mE(e_j)\}$ $\quad$ $\mathcal{B}$ = $\{ed_1, \ldots, ed_k\}$ $\quad$ $\mathcal{C}$ = $\{v_1, \ldots, v_l\}$ $\quad$ $\mathcal{D}$ = $\{v_1 \mapsto t_1, \ldots, v_l \mapsto t_l\}$ $\quad$ $\mathcal{E}$ = $\{ed_1 \mapsto mV(v_1), \ldots, ed_k \mapsto v_l\}$



1 ∃ hyp ($\exists x, y \cdot x \in edgeG \land y \in vertG \land tG\_V(y) = t \land targetG(x) = y$; 2 simplification rewrites; 3 type rewrites; 4 simplification rewrites; 5 he with $tG\_V(y) = t$; 6 he with $targetG(x) = y$;
7 ∃ goal (inst ed_t, $mV(v_t)$); 8 simplification rewrites; 9 ∧ goal; 10 total function dom substitution in goal; 11 type rewrites; 12 ⊤ goal; 13 functional goal; 14 simplification rewrites; 15 generalized MP;
16 simplification rewrites; 17 he with $tG\_V(targetG(x)) = t$; 18 ∧ goal; 19 functional image goal for $mV(v_t)$; 20 functional image goal for $mV(v_t)$; 21 hyp; 22 ∀ hyp (inst v_t); 23 ⊤ goal;
24 Partition rewrites in hyp ($partition(vertLi, \{v_1 \mapsto t_1\}, \ldots, \{v_l \mapsto t_l\})$); 25 simplification rewrites; 26 he with $tG\_V(targetG(x)) = t$;
27 eh with $tLi\_V = \{v_1 \mapsto t_1, \ldots, v_t \mapsto tG\_V(targetG(x)), \ldots, v_l \mapsto t_l\}$; 28 simplification rewrites; 29 ML

Figura 79: Árvore de Prova rule_i/propEdSpTgtT/INV - Regra Cria um Arco com Destino em um Vértice do Tipo t

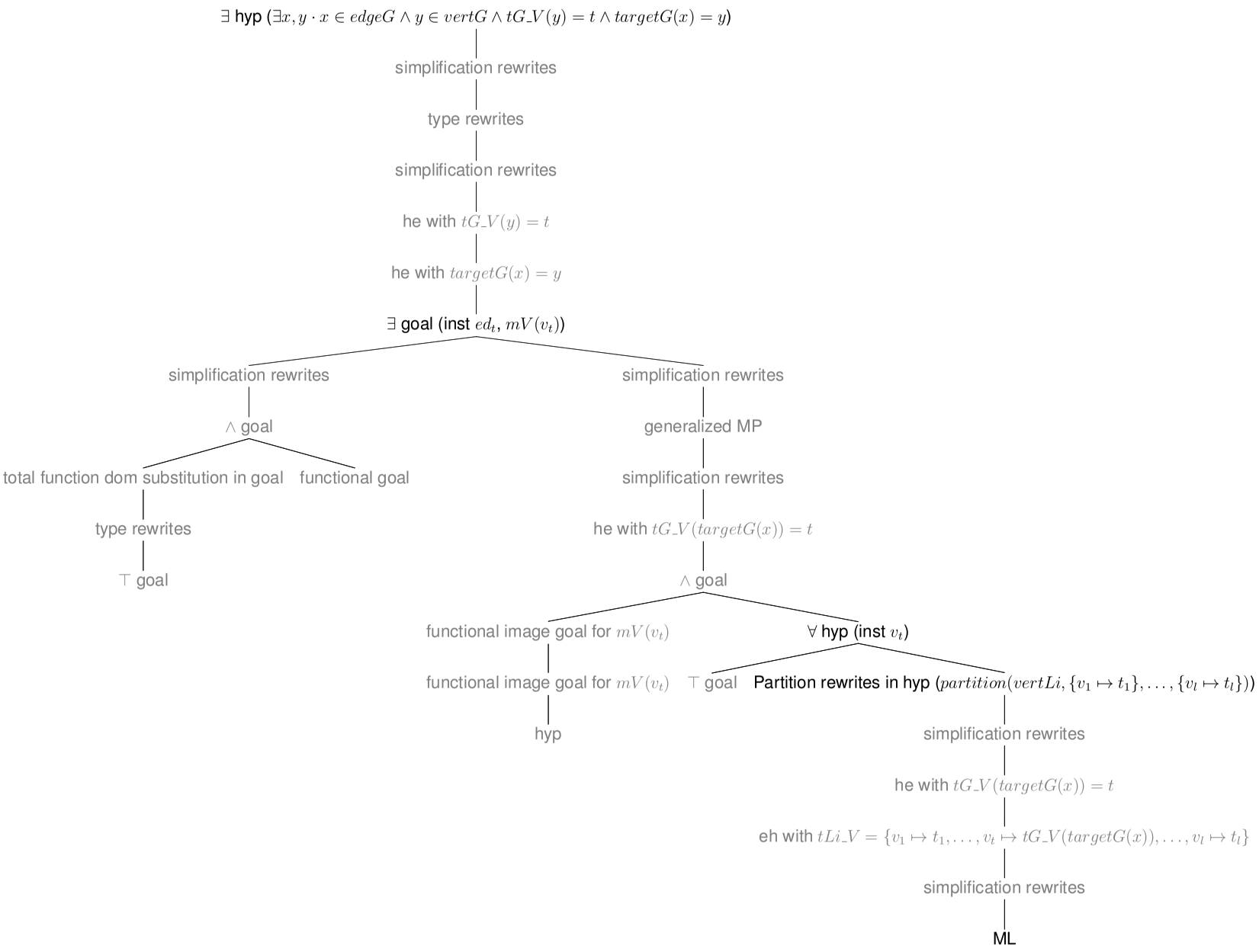


Figura 80: Árvore de Uso - rule_i/propEdSpTgtT/INV - Regra Cria um Arco com Destino em um Vértice do Tipo t

Descrição da Árvore de Uso: *Para utilizar a tática aplica-se a regra* $\exists$ *hyp na hipótese de indução. Em seguida instancia-se o nome do arco criado pela regra* ed_t *juntamente com seu vértice destino* $mV(v_t)$ *do tipo* t*, imagem do* match mV*. Então, o nome do arco* v_t *deve ser instanciado na hipótese* `grd_vertices` *e após aplica-se a regra partition rewrites na hipótese que define a tipagem dos vértices do lado esquerdo da regra e executa-se o provador ML.*

B - Regra não envolve arcos com destino em vértices do tipo t

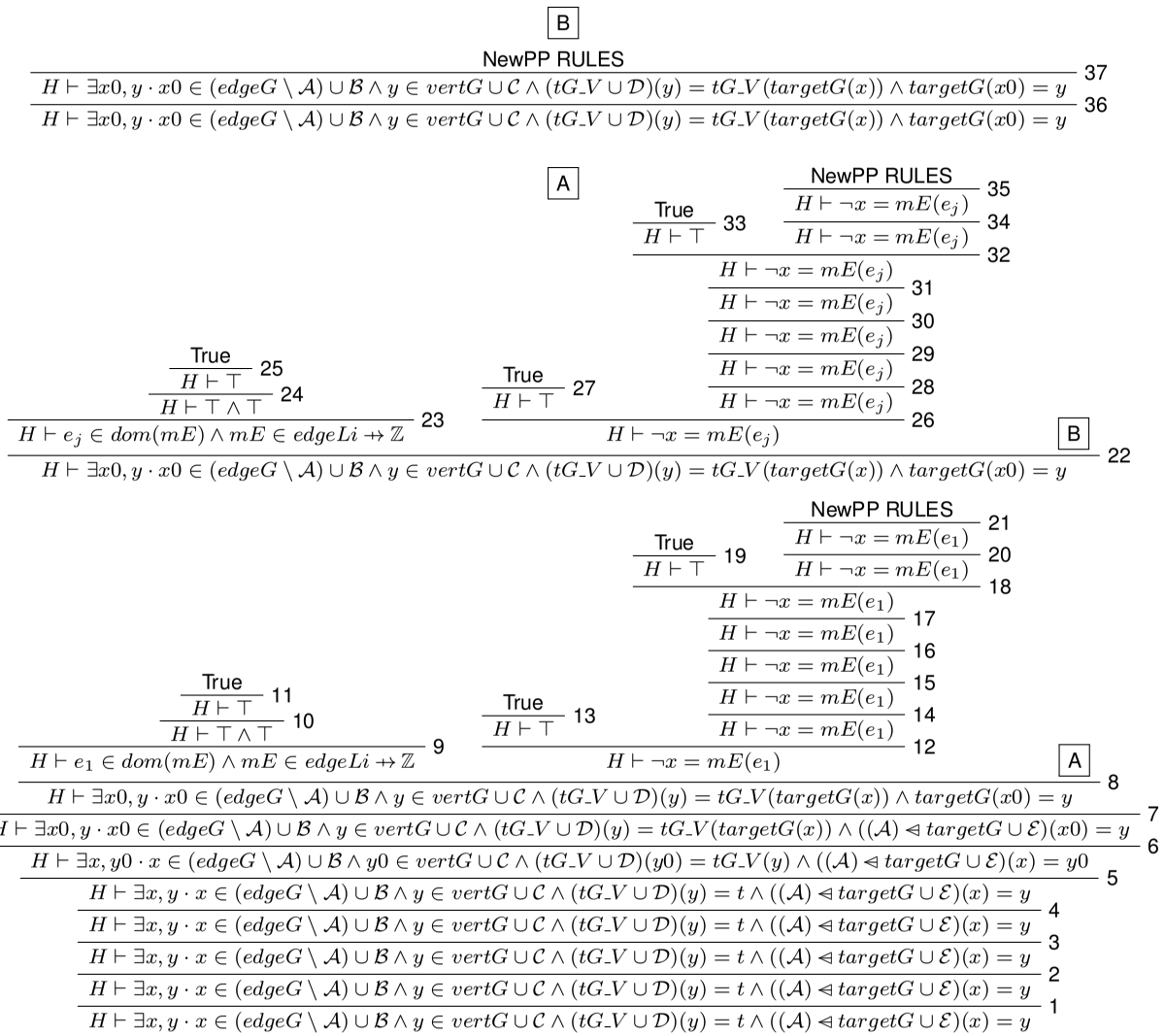
Para esse caso, se a regra não deleta arcos a obrigação é descarregada executando o provador NewPP with lasso. Já para os casos em que a regra deleta arcos a demonstração é realizada executando as seguintes ações:

1. Aplicar a regra $\exists$ hyp na hipótese de indução;
2. Para cada arco e_i deletado pela regra, seguir os passos:
   (a) Adicionar como hipótese $\neg x = mE(e_i)$, onde e_i é um arco deletado pela regra;
   (b) Instanciar o nome do arco deletado e_i na hipótese `grd_srctgt` considerando $target$;
   (c) Aplicar a regra Partition Rewrites na hipótese que define o destino dos arcos do lado esquerdo da regra;
   (d) Instanciar o nome do vértice v_i que é destino do arco deletado e_i, no componente `grd_vertices`;
3. Executar o provador NewPP with lasso.

Nesse caso, a regra não envolve arcos com destino em vértices do tipo t, desta forma a propriedade é assegurada pela hipótese de indução. Para isso, se diferencia o elemento x (arco que satisfaz a propriedade pela hipótese de indução) com cada arco deletado pela regra, ações (a-d) da tática proposta. A Figura 81 apresenta a árvore de prova resultante da demonstração.

Descrição da Árvore de Prova: *O primeiro sequente para se demonstrar é* $H \vdash \exists x, y \cdot x \in (edgeG \setminus \{mE(e_1), \ldots, mE(e_j)\}) \cup \{ed_1, \ldots, ed_k\} \wedge y \in vertG \cup \{v_1, \ldots, v_l\} \wedge (tG\_V \cup \{v_1 \mapsto t_1, \ldots, v_l \mapsto t_l\})(y) = t \wedge ((\{mE(e_1), \ldots, mE(e_j)\}) \lhd targetG \cup \{ed_1 \mapsto mV(v_1), \ldots, ed_k \mapsto v_l\})(x) = y$*. Visando demonstrar esse sequente, aplica-se a regra* $\exists$ *hyp na hipótese de indução (1), algumas regras de forma automática são aplicadas (regras 2-7), resultando em um novo sequente* $H \vdash \exists x0, y \cdot x0 \in (edgeG \setminus \{mE(e_1), \ldots, mE(e_j)\}) \cup \{ed_1, \ldots, ed_k\} \wedge y \in vertG \cup \{v_1, \ldots, v_l\} \wedge (tG\_V \cup \{v_1 \mapsto t_1,$

$\mathcal{A} = \{mE(e_1), \ldots, mE(e_j)\}$ $\mathcal{B} = \{ed_1, \ldots, ed_k\}$ $\mathcal{C} = \{v_1, \ldots, v_l\}$ $\mathcal{D} = \{v_1 \mapsto t_1, \ldots, v_l \mapsto t_l\}$ $\mathcal{E} = \{ed_1 \mapsto mV(v_1), \ldots, ed_k \mapsto v_l\}$



1 $\exists$ hyp ($\exists x, y \cdot x \in edgeG \land y \in vertG \land tG\_V(y) = t \land targetG(x) = y$; 2 simplification rewrites; 3 type rewrites; 4 simplification rewrites; 5 he with $tG\_V(y) = t$; 6 he with $targetG(x) = y$;
7 Functional image simplification in goal; 8 ah $\neg x = m(e_1)$; 9 generalized MP; 10 simplification rewrites; 11 $\top$ goal; 12 $\forall$ hyp (inst e_1); 13 $\top$ goal;
14 Partition rewrites in hyp ($partition(targetLi, \{e_1 \mapsto v_1\}, \ldots, \{e_k \mapsto v_l\})$); 15 simplification rewrites; 16 eh with $targetLi = \{e_1 \mapsto v_1, \ldots, e_k \mapsto v_l\}$; 17 simplification rewrites; 18 $\forall$ hyp (inst v_1);
19 $\top$ goal; 20 sl/ds; 21 NewPP with lasso; 22 ah $\neg x = m(e_j)$; 23 generalized MP; 24 simplification rewrites; 25 $\top$ goal; 26 $\forall$ hyp (inst e_j); 27 $\top$ goal;
28 Partition rewrites in hyp ($partition(targetLi, \{e_1 \mapsto v_1\}, \ldots, \{e_k \mapsto v_l\})$); 29 simplification rewrites; 30 eh with $targetLi = \{e_1 \mapsto v_1, \ldots, e_k \mapsto v_l\}$; 31 simplification rewrites; 32 $\forall$ hyp (inst v_l);
33 $\top$ goal; 34 sl/ds; 35 NewPP with lasso; 36 sl/ds; 37 NewPP with lasso

Figura 81: Árvore de Prova rule_i/propEdSpTgtT/INV - Regra Não Envolve Arcos com Destino em Vértices do Tipo t

$\ldots, v_l \mapsto t_l\})(y) = tG\_V(targetG(x)) \land targetG(x0) = y$. *Para demonstrar esse objetivo adiciona-se como hipótese* $\neg x = m(e_1)$*, onde* e_1 *é o primeiro arco deletado pela regra, essa instanciação resulta em três novos objetivos para se demonstrar: (I)* $H \vdash e_1 \in dom(mE) \land mE \in edgeLi \nrightarrow \mathbb{Z}$*; (II)* $H \vdash \neg x = mE(e_1)$*; e (III)* $H \vdash \exists x0, y \cdot x0 \in (edgeG \setminus \{mE(e_1), \ldots, mE(e_j)\}) \cup \{ed_1, \ldots, ed_k\} \land y \in vertG \cup \{v_1, \ldots, v_l\} \land (tG\_V \cup \{v_1 \mapsto t_1, \ldots, v_l \mapsto t_l\})(y) = tG\_V(targetG(x)) \land targetG(x0) = y$*. Em (I) são aplicadas algumas regras que acabam demonstrando o sub-objetivo (regras 9-11). Em (II), instancia-se o nome do arco deletado* e_1 *na hipótese* `grd_srctgt` *, considerando* $target$*, e essa ação resulta em dois sequentes para se demonstrar: (i)* $H \vdash \top$*, descarregado automaticamente (13); e (ii)* $H \vdash \neg x = mE(e_1)$*. Para demonstrar (ii) aplica-se a regra partition rewrites na hipótese que define o destino dos arcos do lado esquerdo da regra (14), em seguida, algumas regras automaticamente são aplicadas (regras 15-17). Após instancia-se na hipótese* `grd_vertices` *o nome do vértice* v_1*, que é destino do arco deletado* e_1 *(18), gerando dois novos sub-objetivos: (i)* $H \vdash \top$*, demonstrado de forma automática (19); e (ii)* $H \vdash \neg x = mE(e_1)$*, descarregado pelo provador NewPP with lasso. A sequência de passos, a partir da diferenciação entre* x *e o primeiro arco deletado deve ser executada para cada arco deletado pela regra. Esse processo, representando os* j *arcos deletados, pode ser analisado na sub-árvore* A *(regras 22-35). Após realizadas todas as diferenciações é gerado um novo sequente, sub-árvore* B $H \vdash \exists x0, y \cdot x0 \in (edgeG \setminus \{mE(e_1), \ldots, mE(e_j)) \cup \{ed_1, \ldots, ed_k\} \land y \in vertG \cup \{v_1, \ldots, v_l\} \land (tG\_V \cup \{v_1 \mapsto t_1, \ldots, v_l \mapsto t_l\})(y) = tG\_V(targetG(x)) \land targetG(x0) = y$*, que é demonstrado executando o provador NewPP with lasso (37).*

A árvore de uso para este caso de regra é apresentada nas Figuras 82 e 83

Descrição da Árvore de Uso: *Para utilizar a tática, ilustrada na Figura 82, primeiramente aplica-se a regra* $\exists$ *hyp na hipótese de indução, em seguida adicionam-se, respectivamente, como hipóteses* $\neg x = mE(e_1), \ldots, \neg x = mE(e_j)$*, onde* $\{e_1, \ldots, e_j\}$ *representam os arcos deletados pela regra. Em cada diferenciação aplicam-se as seguintes ações: instancia-se o nome do arco deletado* e_i *na hipótese* `grd_srctgt`*, considerando* $target$*, em seguida, aplica-se a regra partition rewrites na hipótese que define o destino dos arcos do lado esquerdo da regra. Após instancia-se na hipótese* `grd_vertices` *o nome do vértice* v_i *que é destino do arco deletado* e_i*. Para descarregar o sub-objetivo gerado executa-se o provador NewPP with lasso. Realizadas todas as diferenciações, no último objetivo gerado executa-se o provador NewPP with lasso. A sub-árvore* A *na Figura 83, representa a diferenciação para os* j *arcos deletados, juntamente com os passos necessários para demonstrar os sub-objetivos criados.*

C - Regra preserva, não deleta e não cria arcos com destino em vértices do tipo t

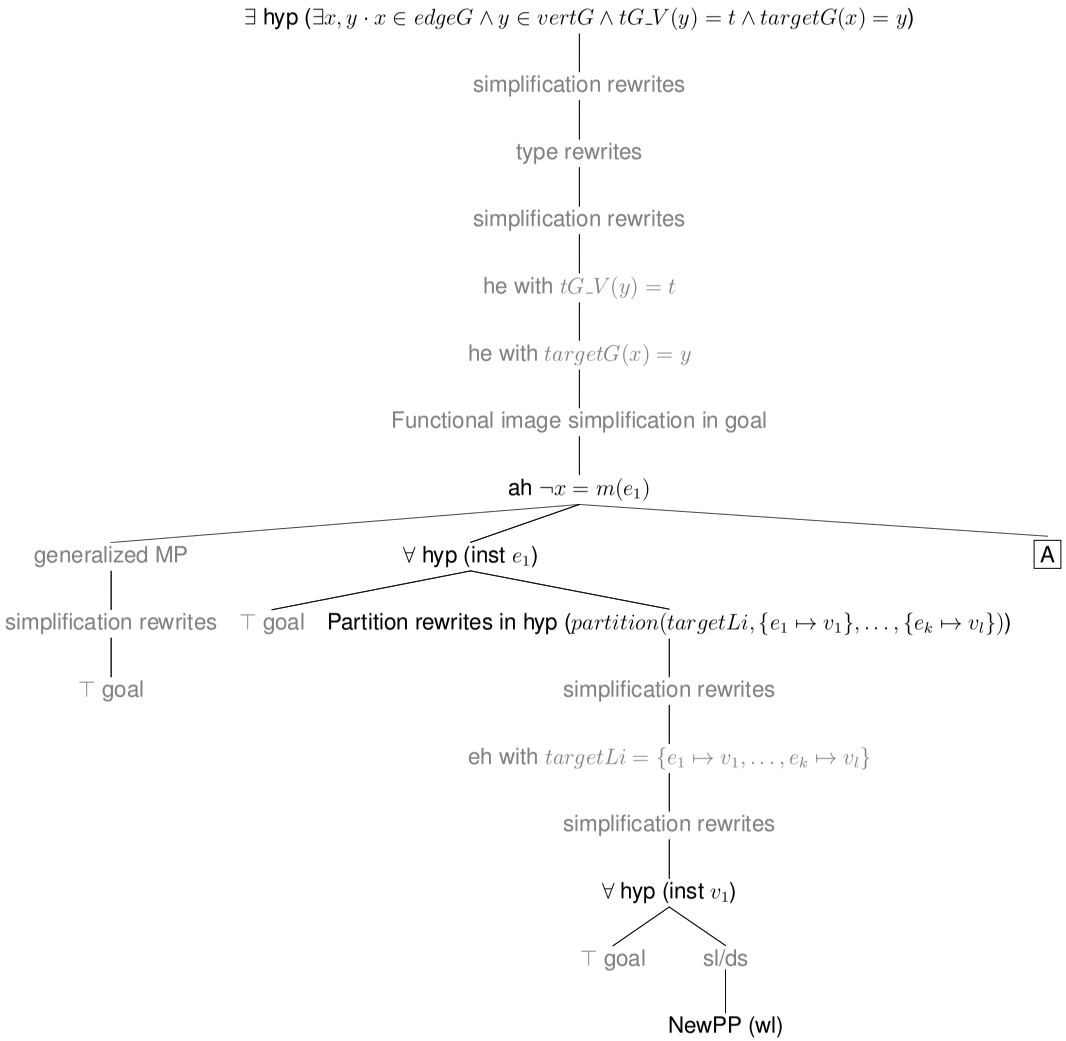


Figura 82: Árvore de Uso - rule_i/propEdSpTgtT/INV - Regra Não Envolve Arcos com Destino em Vértices do Tipo t (1)

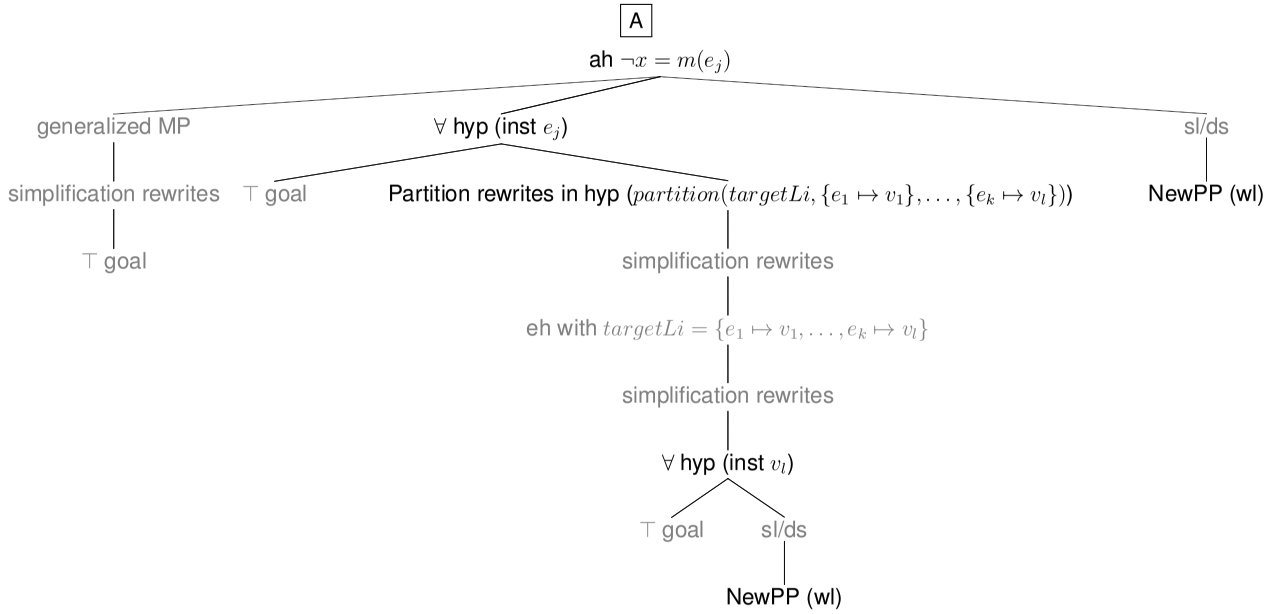


Figura 83: Árvore de Uso - rule_i/propEdSpTgtT/INV - Regra Não Envolve Arcos com Destino em Vértices do Tipo t (2)

Existem dois casos distintos nesses tipos de obrigações de prova. O primeiro é relativo a regra que não deleta arcos. Nesse caso a demonstração é realizada executando o provador NewPP with lasso. No segundo caso, arcos com destino em vértices diferentes do tipo t são deletados pela regra. A tática para demonstrar obrigações desse tipo, é apresentada a seguir:

1. Instanciar no objetivo $mE(e_j)$ e $mV(v_t)$, onde e_j é o arco preservado pela regra, possuindo destino no vértice v_t do tipo t;
2. Aplicar a regra Partition Rewrites na hipótese que define o conjunto de arcos do lado esquerdo da regra e executar ML;
3. Instanciar o nome do vértice v_t que é destino do arco preservado e_j no componente `grd_vertices`;
4. Aplicar a regra Partition Rewrites na hipótese que define a tipagem dos vértices do lado esquerdo da regra e executar ML;
5. Instanciar o nome do arco e_j preservado na hipótese `grd_srctgt`, considerando $target$;
6. Aplicar a regra Partition Rewrites na hipótese que define o destino dos arcos do lado esquerdo da regra;
7. Executar o provador ML;

Após executar os passos da tática, uma árvore de prova é gerada, a qual pode ser analisada nas Figuras 84 e 85.

Descrição da Árvore de Prova: *Conforme Figura 84, o primeiro sequente para se demonstrar é* $H \vdash \exists x, y \cdot x \in (edgeG \setminus \{mE(e_1), \ldots, mE(e_j)\} \cup \{ed_1, \ldots, ed_k\}) \land y \in vertG \land tG\_V(y) = t \land ((\{mE(e_1), \ldots, mE(e_j)\} \lhd targetG) \cup \{ed_1 \mapsto mV(v_1), \ldots, ed_k \mapsto v_l\})(x) = y$. *A fim de demonstrar esse sequente instancia-se* $mE(e_j)$ *e* $mV(v_t)$*, onde* e_j *é um arco preservado com destino no arco* v_t *do tipo* t *(1). Com essa ação dois novos sub-objetivos são gerados: (I)* $H \vdash (v_t \in dom(mV) \land mV \in vertLi \pfun \mathbb{Z}) \land (ed_t \in dom(mE) \land mE \in edgeLi \pfun \mathbb{Z})$ *e (II)* $H \vdash \exists x, y \cdot x \in (edgeG \setminus \{mE(e_1), \ldots, mE(e_j)\} \cup \{ed_1, \ldots, ed_k\}) \land y \in vertG \land tG\_V(mV(v_t)) = t \land ((\{mE(e_1), \ldots, mE(e_j)\} \lhd targetG) \cup \{ed_1 \mapsto v_1, \ldots, ed_k \mapsto v_l\})(x) = y$. *Em (I) são aplicadas algumas regras automáticas, que acabam demonstrando todos os sub-objetivos gerados (regras 2-16). Em (II), sub-árvore* A*, a ferramenta aplica de forma automática algumas regras (17-25), gerando quatro sub-objetivos: (i)* $H \vdash mE(e_j) \in (edgeG \setminus \{mE(e_1), \ldots, mE(e_j)\}) \cup \{ed_1, \ldots, ed_k\}$*; (ii)* $H \vdash mV(v_t) \in vertG$*; (iii)* $H \vdash tG\_V(mV(v_t)) = tG\_V(targetG(x))$*; e (iv)* $H \vdash ((\{mE(e_1), \ldots, mE(e_j)\} \lhd targetG) \cup \{ed_1 \mapsto v_1, \ldots, ed_k \mapsto v_l\})(mE(ed_t)) = mV(v_t)$.

$\mathcal{A} = \{mE(e_1), \ldots, mE(e_j)\}$ $\quad$ $\mathcal{B} = dom(mE) \land mE \in edgeLi \nrightarrow \mathbb{Z}$ $\quad$ $\mathcal{C} = \{ed_1, \ldots, ed_k\}$ $\quad$ $\mathcal{E} = \{ed_1 \mapsto mV(v_1), \ldots, ed_k \mapsto v_l\}$ $\quad$ $\mathcal{F} = dom(mV) \land mV \in vertLi \nrightarrow \mathbb{Z}$

Branch ML RULES:

$$\frac{\frac{\frac{\text{ML RULES}}{H \vdash mE(e_j) \in (edgeG \setminus \mathcal{A}) \cup \mathcal{C}}\,28}{H \vdash mE(e_j) \in (edgeG \setminus \mathcal{A}) \cup \mathcal{C}}\,27}{H \vdash mE(e_j) \in (edgeG \setminus \mathcal{A}) \cup \mathcal{C}}\,26$$

Branch A:

$$\frac{\frac{\frac{\text{True}}{H \vdash mV(v_t) \in vertG}\,31}{H \vdash mV(v_t) \in vertG}\,30}{H \vdash mV(v_t) \in vertG}\,29$$

Branches B, C.

$$\frac{\ldots}{H \vdash mE(e_j) \in (edgeG \setminus \mathcal{A} \cup \mathcal{C}) \land mV(v_t) \in vertG \land tG\_V(mV(v_t)) = tG\_V(targetG(x)) \land ((\mathcal{A} \lhd targetG) \cup \mathcal{E})(mE(e_j)) = mV(v_t)}\,25$$

$$\frac{}{H \vdash mE(e_j) \in (edgeG \setminus \mathcal{A} \cup \mathcal{C}) \land mV(v_t) \in vertG \land tG\_V(mV(v_t)) = tG\_V(y) \land ((\mathcal{A} \lhd targetG) \cup \mathcal{E})(mE(e_j)) = mV(v_t)}\,24$$

$$\frac{}{H \vdash mE(e_j) \in (edgeG \setminus \mathcal{A} \cup \mathcal{C}) \land mV(v_t) \in vertG \land tG\_V(mV(v_t)) = t \land ((\mathcal{A} \lhd targetG) \cup \mathcal{E})(mE(e_j)) = mV(v_t)}\,23$$

$$\frac{}{H \vdash mE(e_j) \in (edgeG \setminus \mathcal{A} \cup \mathcal{C}) \land mV(v_t) \in vertG \land tG\_V(mV(v_t)) = t \land ((\mathcal{A} \lhd targetG) \cup \mathcal{E})(mE(e_j)) = mV(v_t)}\,22$$

$$\frac{}{H \vdash mE(e_j) \in (edgeG \setminus \mathcal{A} \cup \mathcal{C}) \land mV(v_t) \in vertG \land tG\_V(mV(v_t)) = t \land ((\mathcal{A} \lhd targetG) \cup \mathcal{E})(mE(e_j)) = mV(v_t)}\,21$$

$$\frac{}{H \vdash mE(e_j) \in (edgeG \setminus \mathcal{A} \cup \mathcal{C}) \land mV(v_t) \in vertG \land tG\_V(mV(v_t)) = t \land ((\mathcal{A} \lhd targetG) \cup \mathcal{E})(mE(e_j)) = mV(v_t)}\,20$$

$$\frac{}{H \vdash mE(e_j) \in (edgeG \setminus \mathcal{A} \cup \mathcal{C}) \land mV(v_t) \in vertG \land tG\_V(mV(v_t)) = t \land ((\mathcal{A} \lhd targetG) \cup \mathcal{E})(mE(e_j)) = mV(v_t)}\,19$$

$$\frac{}{H \vdash mE(e_j) \in (edgeG \setminus \mathcal{A} \cup \mathcal{C}) \land mV(v_t) \in vertG \land tG\_V(mV(v_t)) = t \land ((\mathcal{A} \lhd targetG) \cup \mathcal{E})(mE(e_j)) = mV(v_t)}\,18$$

$$\frac{}{H \vdash mE(e_j) \in (edgeG \setminus \mathcal{A} \cup \mathcal{C}) \land mV(v_t) \in vertG \land tG\_V(mV(v_t)) = t \land ((\mathcal{A} \lhd targetG) \cup \mathcal{E})(mE(e_j)) = mV(v_t)}\,17$$

$$\frac{\frac{\frac{\frac{\text{True}}{H \vdash \top}\,12}{H \vdash v_t \in vertLi}\,11}{H \vdash v_t \in dom(mV)}\,10 \qquad \frac{\text{True}}{H \vdash mV \in vertLi \nrightarrow \mathbb{Z}}\,13 \qquad \frac{\frac{\frac{\text{True}}{H \vdash \top}\,16}{H \vdash e_j \in edgeLi}\,15}{H \vdash e_j \in dom(mE)}\,14}{H \vdash v_t \in \mathcal{F} \land e_j \in dom(mE)}\,9$$

$$\frac{}{H \vdash v_t \in \mathcal{F} \land e_j \in dom(mE)}\,8$$

$$\frac{}{H \vdash v_t \in \mathcal{F} \land e_j \in dom(mE)}\,7$$

$$\frac{}{H \vdash v_t \in \mathcal{F} \land e_j \in dom(mE)}\,6$$

$$\frac{}{H \vdash v_t \in \mathcal{F} \land e_j \in dom(mE)}\,5$$

$$\frac{}{H \vdash v_t \in \mathcal{F} \land e_j \in dom(mE) \land \top}\,4$$

$$\frac{}{H \vdash v_t \in \mathcal{F} \land e_j \in \mathcal{B}}\,3$$

$$\frac{}{H \vdash (v_t \in \mathcal{F}) \land (e_j \in \mathcal{B})}\,2$$

$$\frac{\ldots \qquad \boxed{A}}{H \vdash \exists x, y \cdot x \in (edgeG \setminus \mathcal{A} \cup \mathcal{C}) \land y \in vertG \land tG\_V(y) = t \land ((\mathcal{A} \lhd targetG) \cup \mathcal{E})(x) = y}\,1$$

1 $\exists$ goal (inst $mE(e_j), mV(v_t)$); 2 simplification rewrites; 3 generalized MP; 4 simplification rewrites; 5 type rewrites; 6 simplification rewrites; 7 $\exists$ hyp ($\exists x, y \cdot x \in edgeG \land y \in vertG \land tG\_V(y) = t \land targetG(x) = y$); 8 he with $targetG(x) = y$; 9 $\top$ goal; 10 total function dom substitution in goal; 11 type rewrites; 12 $\top$ goal; 13 functional goal; 14 total function dom substitution in goal; 15 type rewrites; 16 $\top$ goal; 17 simplification rewrites; 18 generalized MP; 19 simplification rewrites; 20 type rewrites; 21 simplification rewrites; 22 $\exists$ hyp ($\exists x, y \cdot x \in edgeG \land y \in vertG \land tG\_V(y) = t \land targetG(x) = y$); 23 he with $tG\_V(y) = t$; 24 he with $targetG(x) = y$; 25 $\land$ goal; 26 Partition rewrites in hyp ($partition(edgeLi, \{ed_1\}, \ldots, \{ed_k\})$); 27 eh with $edgeLi = \{ed_1, \ldots, ed_k\}$; 28 ML; 29 functional image goal for $mV(v_t)$; 30 functional image goal for $mV(v_t)$; 31 hyp;

Figura 84: Árvore de Prova rule_i/propEdSpTgtT/INV - Regra Preserva, não Deleta e não Cria Arcos com Destino em Vértices do Tipo t (1)

$\mathcal{A} = \{mE(e_1), \ldots, mE(e_j)\}$ $\mathcal{B} = dom(mE) \wedge mE \in edgeLi \rightarrow \mathbb{Z}$ $\mathcal{C} = \{ed_1, \ldots, ed_k\}$ $\mathcal{D} = \{ed_1 \mapsto mV(v_1), \ldots, ed_k \mapsto mV(v_l)\}$

$\mathcal{E} = \{ed_1 \mapsto v_1, \ldots, ed_k \mapsto v_l\}$ $\mathcal{F} = dom(mV) \wedge mV \in vertLi \rightarrow \mathbb{Z}$

C

ML RULES

$$\frac{\frac{\text{True}}{H \vdash \top}\,42 \qquad \frac{\frac{\frac{\frac{\frac{\frac{\frac{\frac{\frac{\text{ML RULES}}{H \vdash ((\mathcal{A} \lhd targetG) \cup \mathcal{E})(mE(e_j)) = mV(v_t)}\,50}{H \vdash ((\mathcal{A} \lhd targetG) \cup \mathcal{E})(mE(e_j)) = mV(v_t)}\,49}{H \vdash ((\mathcal{A} \lhd targetG) \cup \mathcal{E})(mE(e_j)) = mV(v_t)}\,48}{H \vdash ((\mathcal{A} \lhd targetG) \cup \mathcal{E})(mE(e_j)) = mV(v_t)}\,47}{H \vdash ((\mathcal{A} \lhd targetG) \cup \mathcal{E})(mE(e_j)) = mV(v_t)}\,46}{H \vdash ((\mathcal{A} \lhd targetG) \cup \mathcal{E})(mE(e_j)) = mV(v_t)}\,45}{H \vdash ((\mathcal{A} \lhd targetG) \cup \mathcal{E})(mE(e_j)) = mV(v_t)}\,44}{H \vdash ((\mathcal{A} \lhd targetG) \cup \mathcal{E})(mE(e_j)) = mV(v_t)}\,43}{}}{H \vdash ((\mathcal{A} \lhd targetG) \cup \mathcal{E})(mE(e_j)) = mV(v_t)}\,41$$

B

$$\frac{\frac{\text{True}}{H \vdash \top}\,33 \qquad \frac{\frac{\frac{\frac{\frac{\frac{\frac{\text{ML RULES}}{H \vdash tG\_V(mV(v_t)) = tG\_V(targetG(x))}\,40}{H \vdash tG\_V(mV(v_t)) = tG\_V(targetG(x))}\,39}{H \vdash tG\_V(mV(v_t)) = tG\_V(targetG(x))}\,38}{H \vdash tG\_V(mV(v_t)) = tG\_V(targetG(x))}\,37}{H \vdash tG\_V(mV(v_t)) = tG\_V(targetG(x))}\,36}{H \vdash tG\_V(mV(v_t)) = tG\_V(targetG(x))}\,35}{H \vdash tG\_V(mV(v_t)) = tG\_V(targetG(x))}\,34}{H \vdash tG\_V(mV(v_t)) = tG\_V(targetG(x))}\,32$$

32 ∀ hyp (inst v_t); 33 ⊤ goal; 34 Partition rewrites in hyp ($partition(tLi\_V, \{ed_1 \mapsto t_1\}, \ldots, \{ed_k \mapsto t_k\})$); 35 generalized MP; 36 simplification rewrites; 37 he with $tG\_V(targetG(x)) = t$; 38 eh with $tLi\_V = \{ed_1 \mapsto t_1, e_j \mapsto tG\_V(targetG(x)), \ldots, ed_k \mapsto t_k\}$; 39 simplification rewrites; 40 ML; 41 ∀ hyp (inst e_j); 42 ⊤ goal; 43 Partition rewrites in hyp ($partition(targetLi, \{ed_1 \mapsto v_1\}, \ldots, \{ed_k \mapsto v_l\})$); 44 simplification rewrites; 45 remove ¬ in $\neg(e_j = ed_1 \wedge v_t = v_1)$; 46 remove ¬ in $\neg(e_j = ed_k \wedge v_t = v_l)$; 47 remove ¬ in $\neg(ed_1 = ed_k \wedge v_1 = v_l)$; 48 eh with $targetLi = \{ed_1 \mapsto v_1, \ldots, ed_k \mapsto v_l\}$; 49 simplification rewrites; 50 ML

Figura 85: Árvore de Prova rule_i/propEdSpTgtT/INV - Regra Preserva, não Deleta e não Cria Arcos com Destino em Vértices do Tipo t (2)

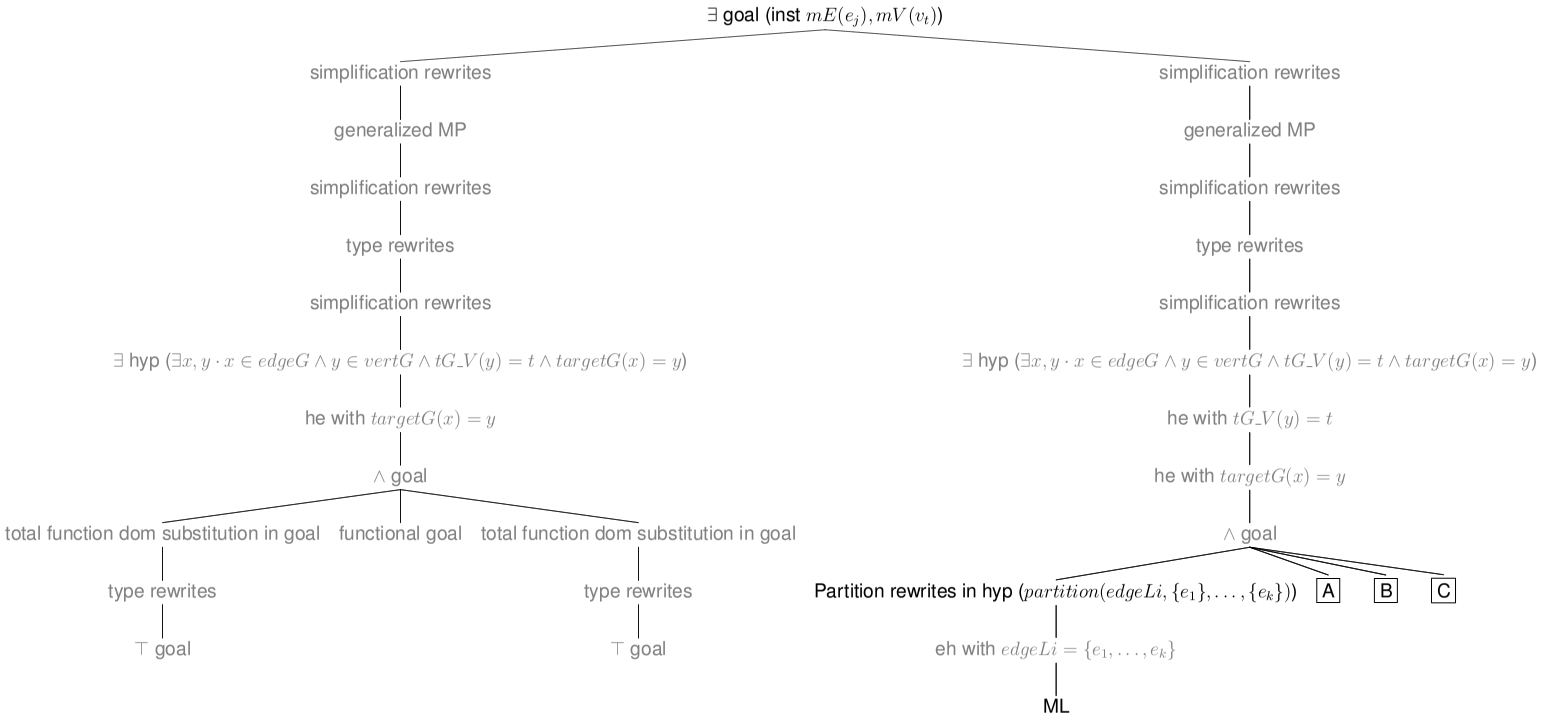


Figura 86: Árvore de Uso rule_i/propEdSpTgtT/INV - Regra Preserva, não Deleta e não Cria Arcos com Destino em Vértices do Tipo t (1)

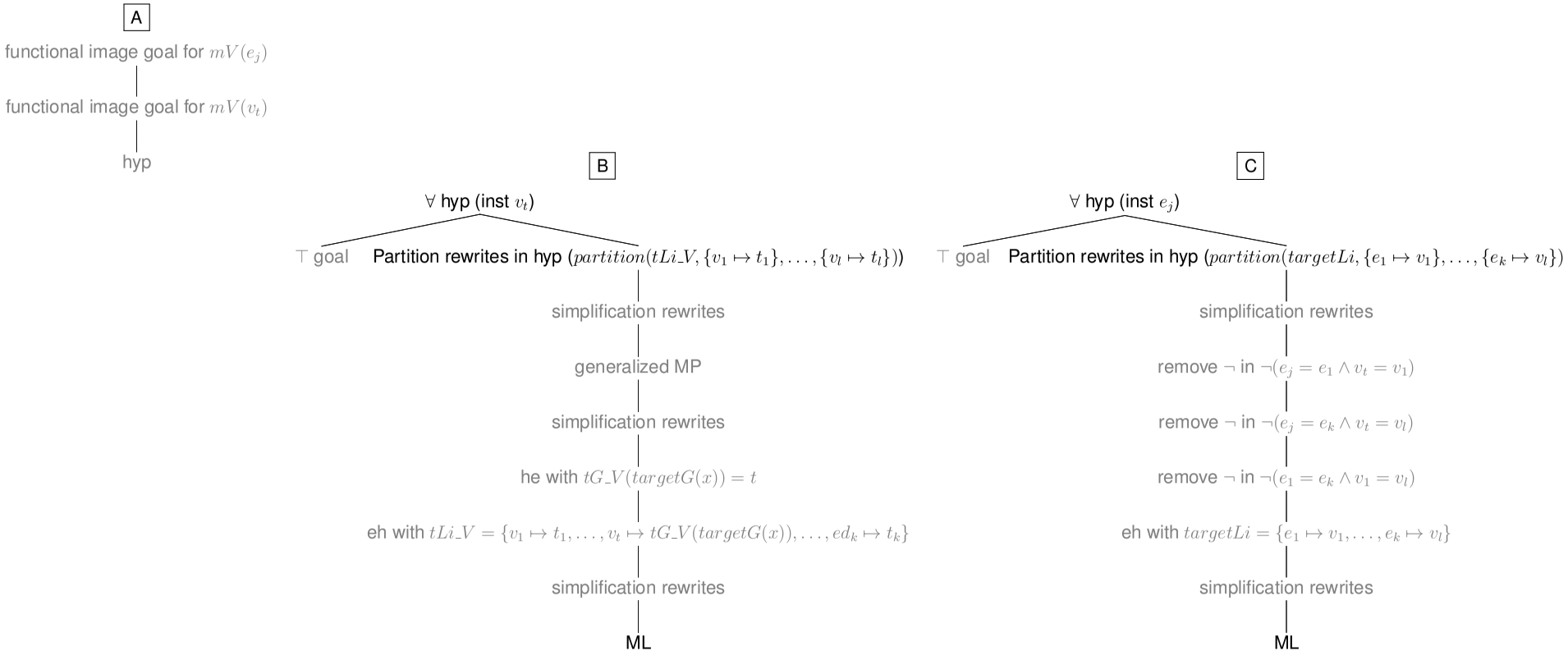


Figura 87: Árvore de Uso rule_i/propEdSpTgtT/INV - Regra Preserva, não Deleta e não Cria Arcos com Destino em Vértices do Tipo t (2)

A fim de demonstrar (i) aplica-se a regra Partition Rewrites (26) na hipótese que define o conjunto dos arcos do lado esquerdo da regra, em seguida é aplicada de forma automática a regra use equality hypothesis - eh (27) e o sub-objetivo e demonstrado por ML (28). O sequente (ii) é demonstrado de forma automática pela regra hyp (31), conforme Figura 85, sub-árvore B*. Para demonstrar (iii) instancia-se na hipótese* grd_vertices *o nome do vértice* v_t*, que é destino do arco* e_j *(32). Após essa ação são gerados dois novos sub-objetivos: (i)* $H \vdash \top$*, é descarregado de forma automática pela regra* $\top$ *goal (33) e (ii)* $H \vdash tG\_V(mV(v_t)) = tG\_V(targetG(x))$*. A fim de demonstrar (ii) aplica-se a tática Partition Rewrites (34) na hipótese que define a tipagem dos arcos do lado esquerdo da regra. Em seguida algumas regras são aplicadas automaticamente (regras 34-39) e a demonstração é realizada por ML (40). Visando demonstrar (iv), sub-árvore* C*, instancia-se o nome do arco preservado* e_j *na hipótese* grd_srctgt*, considerando* $target$ *(41). Com isso, são gerados dois novos sub-objetivos: (i)* $H \vdash \top$*, que é descarregado automaticamente (42) e (ii)* $H \vdash ((\{mE(e_1), \ldots, mE(e_j)\} \lhd targetG) \cup \{ed_1 \mapsto v_1, \ldots, ed_k \mapsto v_l\})(mE(e_j)) = mV(v_t)$*. Para demonstrar (ii) aplica-se a regra Partition Rewrites na hipótese que define o destino dos arcos do lado esquerdo da regra (43). De forma automática a ferramenta aplica algumas regras (44-49), para concluir a prova da obrigação executa-se o provador ML (50).*

As Figuras 86 e 87 apresentam a árvore de uso para a obrigação rule_i/propEdSpTgtT/INV.

Descrição da Árvore de Uso: *Conforme Figura 86, para utilizar essa estratégia, instancia-se* $mE(e_j)$ *e* $mV(v_t)$*, onde* e_j *é o arco preservado com destino no vértice* v_t *do tipo* t*. Em seguida, é necessário aplicar a regra Partition Rewrites na hipótese que define os arcos do lado esquerdo da regra e executar ML. Na sub-árvore* B *da Figura 87, instancia-se o nome do vértice* v_t *que é destino do arco preservado* e_j *na hipótese* grd_vertices*. Aplica-se a regra Partition Rewrites na hipótese que define a tipagem dos vértices do lado esquerdo da regra e executa-se o provador ML. Na sub-árvore* C*, instancia-se o nome do arco* e_j *na hipótese* grd_srctgt*, considerando* $target$*, em seguida, aplica-se a regra Partition Rewrites na hipótese que define o destino dos arcos do lado esquerdo da regra e executa-se o provador ML.*

4.8 Propriedade propNotIsoVT

A propriedade propNotIsoVT $\forall x \cdot ((x \in vertG \wedge tG\_V(x) = t) \Rightarrow (\exists y \cdot (y \in edgeG \wedge ((sourceG(y) = x \wedge \neg targetG(y) = x) \vee (targetG(y) = x \wedge \neg sourceG(y) = x)))))$ assegura que em qualquer grafo alcançável do sistema, não existe um vértice isolado do tipo t. Quando as variáveis $vertG$, tG_V, $edgeG$, $sourceG$ e $targetG$ são ini-

cializadas pelo evento de inicialização, é gerada a obrigação de prova INITIALISATION/propNotIsoVT/INV. Essa obrigação pode ser provada executando os seguintes passos:

1. Eliminar o quantificador universal do objetivo;
2. Remover o operador $\in$ na hipótese $x \in vertG$;
3. Iniciar a prova por casos na hipótese $x = v_1 \vee \ldots \vee x = v_n$;
4. Para cada caso, aplicar uma das ações até concluir a demonstração:
   (a) Para vértices v_i, $i \in \{1, \ldots, n\}$ que são do tipo t, instanciar no objetivo o arco nele incidente (não loop);
   (b) Para vértices v_j, $j \in \{1, \ldots, n\}$ que não são do tipo t, executar o provador ML;

A estratégia para essa propriedade relativa a obrigação de prova INITIALISATION/propNotIsoVT/INV é representada através da árvore de prova da Figura 88.

Descrição da Árvore de Prova: *O objetivo inicial a ser demonstrado na obrigação de prova é* $\vdash \forall x \cdot x \in vertG \wedge tG\_V(x) = t \Rightarrow (\exists y \cdot y \in edgeG \wedge ((sourceG(y) = x \wedge \neg targetG(y) = x) \vee (targetG(y) = x \wedge \neg sourceG(y) = x)))$*. A fim de demonstrar esse sequente elimina-se o quantificador universal no objetivo (1). Essa ação resulta em um novo objetivo* $tG\_V(x) = t, x \in \{v_1, \ldots, v_n\} \vdash \exists y \cdot y \in edgeG \wedge ((sourceG(y) = x \wedge \neg targetG(y) = x) \vee (targetG(y) = x \wedge \neg sourceG(y) = x))$*. Para demonstrar esse sequente, primeiramente remove-se o operador* $\in$ *na hipótese* $x \in vertG$ *(3). Então se inicia a prova por casos, a partir da hipótese* $x = v_1 \vee \ldots \vee x = v_n$*. Com isso, primeiramente são aplicadas algumas regras automáticas (5-8 e 13-16), chegando nos sub-objetivos de cada caso: (i)* $x = v_i \vdash \exists y \cdot y \in edgeG \wedge ((sourceG(y) = v_i \wedge \neg targetG(y) = v_i) \vee (targetG(y) = v_i \wedge \neg sourceG(y) = v_i))$*; (ii)* $x = v_j, t = t_k \vdash \exists y \cdot y \in edgeG \wedge ((sourceG(y) = v_j \wedge \neg targetG(y) = v_j) \vee (targetG(y) = v_j \wedge \neg sourceG(y) = v_j))$*. O sequente (i) representa o caso em que o vértice considerado* v_i *é do tipo* t*. Para demonstrar esse sequente instancia-se no objetivo o nome do arco* ed_t *incidente neste vértice e a conclusão da prova é realizada de forma automática pelas regras* $\top$ *goal e simplification rewrites (regras 10-12). Já o sequente (ii) representa o caso em que o vértice* v_j *não é do tipo* t*, para demonstrar este tipo de caso executa-se o provador ML (17). Cabe salientar que sequentes semelhantes a (i) serão gerados* i *vezes se no estado inicial existir* i *vértices do tipo* t*. Da mesma forma são gerados* j *sequentes semelhantes a (ii) caso existir* j *vértices diferentes do tipo* t *no grafo inicial da gramática. A prova desses sequentes seguem o mesmo padrão apresentado em cada caso.*

1 $\forall$ goal (free x); 2 $\Rightarrow$ goal; 3 remove $\in$ in $x \in \{v_1, \ldots, v_n\}$; 4 $\forall$ hyp ($x = v_1 \vee \ldots \vee x = v_n$); 5 generalized MP; 6 simplification rewrites; 7 eh with $x = v_i$; 8 simplification rewrites; 9 $\exists$ goal (inst ed_t); 10 $\top$ goal; 11 simplification rewrites; 12 $\top$ goal; 13 generalized MP; 14 simplification rewrites; 15 eh with $x = v_j$; 16 simplification rewrites; 17 ML

Figura 88: Árvore de Prova INITIALISATION/propNotIsoVT/INV

A árvore de uso para essa obrigação de prova é apresentada na Figura 89.

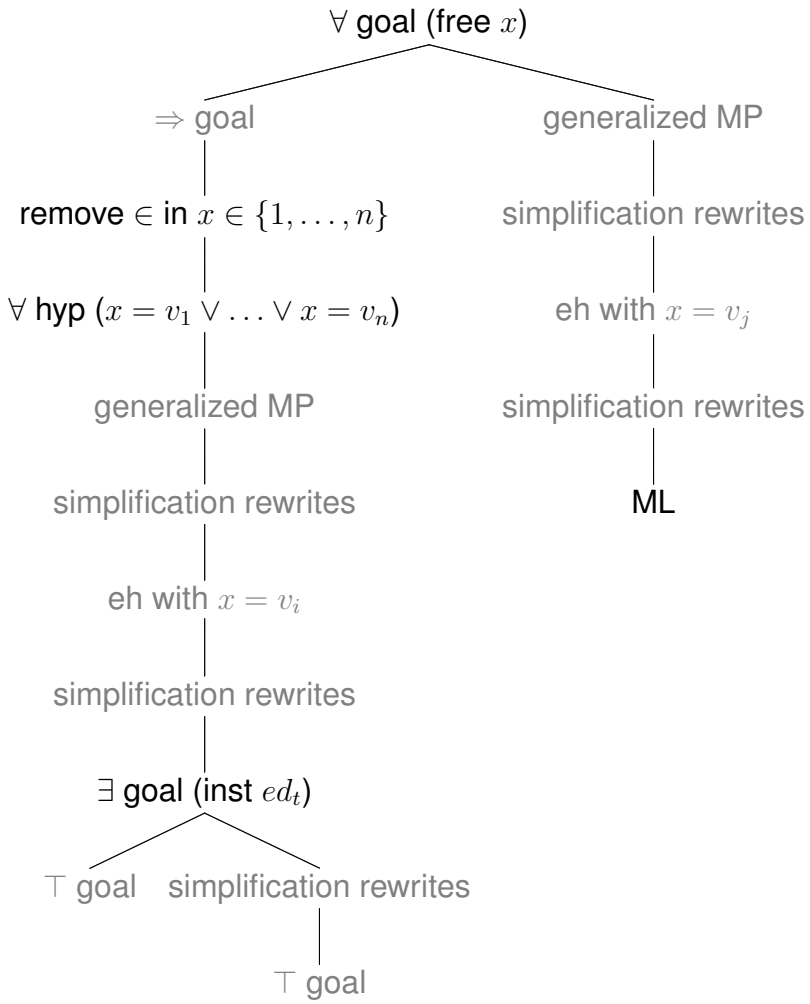


Figura 89: Árvore de Uso INITIALISATION/propNotIsoVT/INV

Descrição da Árvore de Uso: *Para utilizar essa tática, se elimina o quantificador universal do objetivo e após remove-se o operador* $\in$ *da hipótese* $x \in \{1, \ldots, n\}$*. Se inicia a prova por casos a partir da hipótese* $x = v_1 \vee \ldots \vee x = v_n$*. Para cada caso: se o vértice solicitado no antecedente é do tipo* t *(*v_i*), então instancia-se no objetivo o nome do arco (não loop) nele incidente (*ed_t*). Caso contrário, executa-se o provador ML. Essas ações repetem-se para cada vértice do grafo inicial.*

Considerando a propriedade `propNotIsoVT`, uma regra pode deletar e criar arcos e/ou criar ou não novos vértices, assim as obrigações geradas para essas regras, seguem o seguinte formato $\forall x \cdot x \in vertG \cup \{v_1, \ldots, v_l\} \wedge (tG\_V \cup \{v_1 \mapsto t_1, \ldots, v_l \mapsto t_l\})(x) = t \Rightarrow (\exists y \cdot y \in (edgeG \setminus \{mE(e_1), \ldots, mE(e_j)\}) \cup \{ed_1, \ldots, ed_k\} \wedge (((((\{mE(e_1), \ldots, mE(e_j)\} \lhd sourceG \cup \{ed_1 \mapsto t_1, \ldots, ed_k \mapsto t_k\})(y) = x \wedge \neg((\{mE(e_1), \ldots, mE(e_j)\} \lhd targetG \cup \{ed_1 \mapsto t_1, \ldots, ed_k \mapsto t_k\})(y) = x) \vee ((((\{mE(e_1), \ldots, mE(e_j)\} \lhd targetG \cup \{ed_1 \mapsto t_1, \ldots, ed_k \mapsto t_k\})(y) = x \wedge$

$\neg((\{mE(e_1),\ldots,mE(e_j)\} \lhd sourceG \cup \{ed_1 \mapsto t_1,\ldots,ed_k \mapsto t_k\})(y) = x)))$, considerando que j arcos são deletados, k arcos são criados e l vértices são criados pela regra. Para essa propriedade considera-se os seguintes casos para as regras: regra cria e não preserva vértices do tipo t; regra cria e preserva vértices do tipo t; regra não cria e preserva vértices do tipo t; e regra não envolve vértices do tipo t. Desta forma a apresentação das táticas são divididas nesses casos.

Para essa propriedade, não é considerado o caso em que a regra deleta arcos que possuam origem ou destino em um vértice do tipo t.

A- Regra cria e não preserva vértices do tipo t

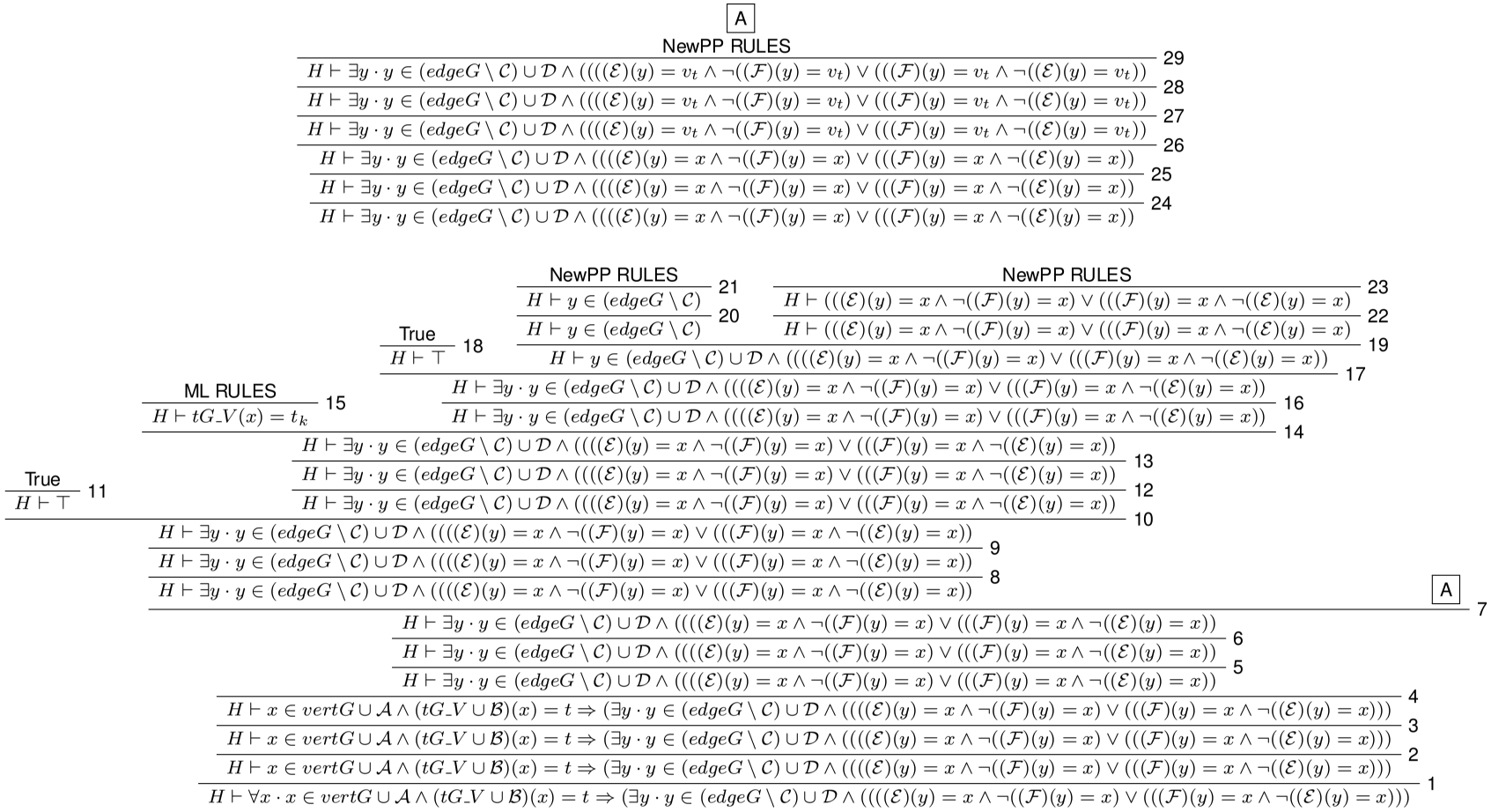
Para as regras que criam e não preservam vértices do tipo t e não deletam arcos, a obrigação de prova é demonstrada executando o provador NewPP with lasso. Já em regras que deletam arcos (não possuindo origem ou destino em vértices do tipo t) a demonstração é realizada executando os seguintes passos:

1. Instanciar o quantificador universal do objetivo;
2. Remover o operador $\in$ na hipótese $x \in vertG \cup \{v_1,\ldots,v_l\}$;
3. Iniciar a prova por casos a partir da hipótese $x \in vertG \vee x = v_1 \vee \ldots \vee x = v_l$;
4. Instanciar o elemento x na hipótese de indução $\forall x \cdot x \in vertG \wedge tG\_V(x) = t \Rightarrow (\exists y \cdot y \in edgeG \wedge ((sourceG(y) = x \wedge \neg targetG(y) = x) \vee (targetG(y) = x \wedge \neg sourceG(y) = x)))$;
5. Aplicar a regra Modus Ponens a partir de $\Rightarrow$ na hipótese $tG\_V(x) = t \Rightarrow (\exists y \cdot y \in edgeG \wedge ((sourceG(y) = x \wedge \neg targetG(y) = x) \vee (targetG(y) = x \wedge \neg sourceG(y) = x)))$ e executar o provador ML;
6. Instanciar no objetivo o elemento y (representando o arco criado com incidência no vértice criado do tipo t, pela hipótese de indução) e nos dois próximos sub-objetivos executar o provador NewPP with lasso;
7. Executar o provador NewPP with lasso;

Essa estratégia, após aplicada na obrigação correspondente, gera uma árvore de prova conforme Figura 90.

Descrição da Árvore de Prova: *O sequente inicial a ser demonstrado é* $H \vdash \forall x \cdot x \in vertG \cup \{v_1,\ldots,v_l\} \wedge (tG\_V \cup \{v_1 \mapsto t_1,\ldots,v_l \mapsto t_l\})(x) = t \Rightarrow (\exists y \cdot y \in (edgeG \setminus \{mE(e_1),\ldots,mE(e_j)\}) \cup \{ed_1,\ldots,ed_k\} \wedge ((((\{mE(e_1),\ldots,mE(e_j)\} \lhd sourceG \cup \{ed_1 \mapsto t_1,\ldots,ed_k \mapsto t_k\})(y) = x \wedge \neg((\{mE(e_1),\ldots,mE(e_j)\} \lhd targetG \cup \{ed_1 \mapsto t_1,\ldots,ed_k \mapsto t_k\})(y) = x) \vee (((\{mE(e_1),\ldots,mE(e_j)\} \lhd targetG \cup \{ed_1 \mapsto t_1,\ldots,ed_k \mapsto t_k\})(y) = x \wedge \neg((\{mE(e_1),\ldots,mE(e_j)\} \lhd sourceG \cup \{ed_1 \mapsto t_1,\ldots,ed_k \mapsto t_k\})(y) = x)))$. *A fim de*

$\mathcal{A}$ = $\{v_1, \ldots, v_l\}$ $\quad$ $\mathcal{B}$ = $\{v_1 \mapsto t_1, \ldots, v_l \mapsto t_l\}$ $\quad$ $\mathcal{C}$ = $\{mE(e_1), \ldots, mE(e_j)\}$ $\quad$ $\mathcal{D}$ = $\{ed_1, \ldots, ed_k\}$
$\mathcal{E}$ = $\{mE(e_1), \ldots, mE(e_j)\} \lhd sourceG \cup \{ed_1 \mapsto mV(v_1), \ldots, ed_k \mapsto v_l\}$ $\quad$ $\mathcal{F}$ = $\{mE(e_1), \ldots, mE(e_j)\} \lhd targetG \cup \{ed_1 \mapsto mV(v_1), \ldots, ed_k \mapsto v_l\}$



1 $\forall$ goal (frees x); 2 type rewrites; 3 simplification rewrites; 4 $\Rightarrow$ goal; 5 remove $\in$ in $x \in vertG \cup \{v_1, \ldots, v_l\}$; 6 simplification rewrites; 7 $\forall$ hyp ($x \in vertG \vee x = v_1 \vee \ldots \vee x = v_l$); 8 generalized MP; 9 simplification rewrites; 10 $\forall$ hyp (inst x); 11 $\top$ goal; 12 generalized MP; 13 simplification rewrites; 14 $\Rightarrow$ hyp mp ($tG\_V(x) = t \Rightarrow (\exists y \cdot y \in edgeG \wedge ((sourceG(y) = x \wedge \neg targetG(y) = x) \vee (targetG(y) = x \wedge \neg sourceG(y) = x)))$); 15 ML; 16 $\exists$ hyp ($\exists y \cdot y \in \wedge((sourceG(y) = x \wedge \neg targetG(y) = x) \vee (targetG(y) = x \wedge \neg sourceG(y) = x))$); 17 $\exists$ goal (inst y); 18 $\top$ goal; 19 $\wedge$ goal; 20 sl/ds; 21 NewPP with lasso; 22 sl/ds; 23 NewPP with lasso; 24 generalized MP; 25 simplification rewrites; 26 eh with $x = v_t$; 27 simplification rewrites; 28 sl/ds; 29 NewPP with lasso

Figura 90: Árvore de Prova rule_i/propNotIsoVT/INV - Regra Cria e Não Preserva Vértices do Tipo t

demonstrar esse sequente instancia-se o quantificador universal do objetivo (1). Em seguida, algumas regras de forma automática são aplicadas (regras 2-4) gerando um novo objetivo $H \vdash \exists y \cdot y \in (edgeG \setminus \{mE(e_1), \ldots, mE(e_j)\}) \cup \{ed_1, \ldots, ed_k\} \wedge ((((\{mE(e_1), \ldots, mE(e_j)\} \lhd sourceG \cup \{ed_1 \mapsto t_1, \ldots, ed_k \mapsto t_k\})(y) = x \wedge \neg((\{mE(e_1), \ldots, mE(e_j)\} \lhd targetG \cup \{ed_1 \mapsto t_1, \ldots, ed_k \mapsto t_k\})(y) = x) \vee ((((\{mE(e_1), \ldots, mE(e_j)\} \lhd targetG \cup \{ed_1 \mapsto t_1, \ldots, ed_k \mapsto t_k\})(y) = x \wedge \neg((\{mE(e_1), \ldots, mE(e_j)\} \lhd sourceG \cup \{ed_1 \mapsto t_1, \ldots, ed_k \mapsto t_k\})(y) = x))$*. Para dar continuidade na prova, é necessário remover o operador* $\in$ *na hipótese* $x \in vertG \cup \{v_1, \ldots, v_l\}$ *(5), após se inicia uma prova por casos a partir da hipótese* $x \in vertG \vee x = v_1 \vee \ldots \vee x = v_l$ *(7). Aqui considera-se dois sub-objetivos gerados, o primeiro considerando um vértice* x *pertencente a* $vertG$ *e o segundo considerando um vértice criado pela regra: (I)* $H \vdash \exists y \cdot y \in (edgeG \setminus \{mE(e_1), \ldots, mE(e_j)\}) \cup \{ed_1, \ldots, ed_k\} \wedge ((((\{mE(e_1), \ldots, mE(e_j)\} \lhd sourceG \cup \{ed_1 \mapsto t_1, \ldots, ed_k \mapsto t_k\})(y) = x \wedge \neg((\{mE(e_1), \ldots, mE(e_j)\} \lhd targetG \cup \{ed_1 \mapsto t_1, \ldots, ed_k \mapsto t_k\})(y) = x) \vee ((((\{mE(e_1), \ldots, mE(e_j)\} \lhd targetG \cup \{ed_1 \mapsto t_1, \ldots, ed_k \mapsto t_k\})(y) = x \wedge \neg((\{mE(e_1), \ldots, mE(e_j)\} \lhd sourceG \cup \{ed_1 \mapsto t_1, \ldots, ed_k \mapsto t_k\})(y) = x))$ *e (II)* $H \vdash \exists y \cdot y \in (edgeG \setminus \{mE(e_1), \ldots, mE(e_j)\}) \cup \{ed_1, \ldots, ed_k\} \wedge ((((\{mE(e_1), \ldots, mE(e_j)\} \lhd sourceG \cup \{ed_1 \mapsto t_1, \ldots, ed_k \mapsto t_k\})(y) = v_t \wedge \neg((\{mE(e_1), \ldots, mE(e_j)\} \lhd targetG \cup \{ed_1 \mapsto t_1, \ldots, ed_k \mapsto t_k\})(y) = v_t) \vee ((((\{mE(e_1), \ldots, mE(e_j)\} \lhd targetG \cup \{ed_1 \mapsto t_1, \ldots, ed_k \mapsto t_k\})(y) = v_t \wedge \neg((\{mE(e_1), \ldots, mE(e_j)\} \lhd sourceG \cup \{ed_1 \mapsto t_1, \ldots, ed_k \mapsto t_k\})(y) = v_t))$*. Em (I) algumas regras de forma automática foram aplicadas (regras 8-9), em seguida instancia-se o elemento* x *na hipótese de indução (10). Essa instanciação gera o objetivo* $H \vdash \top$ *demonstrado automaticamente (11) e* $H \vdash \exists y \cdot y \in (edgeG \setminus \{mE(e_1), \ldots, mE(e_j)\}) \cup \{ed_1, \ldots, ed_k\} \wedge ((((\{mE(e_1), \ldots, mE(e_j)\} \lhd sourceG \cup \{ed_1 \mapsto t_1, \ldots, ed_k \mapsto t_k\})(y) = x \wedge \neg((\{mE(e_1), \ldots, mE(e_j)\} \lhd targetG \cup \{ed_1 \mapsto t_1, \ldots, ed_k \mapsto t_k\})(y) = x) \vee ((((\{mE(e_1), \ldots, mE(e_j)\} \lhd targetG \cup \{ed_1 \mapsto t_1, \ldots, ed_k \mapsto t_k\})(y) = x \wedge \neg((\{mE(e_1), \ldots, mE(e_j)\} \lhd sourceG \cup \{ed_1 \mapsto t_1, \ldots, ed_k \mapsto t_k\})(y) = x))$*. Visando demonstrar esse sequente, após algumas regras automática serem aplicadas (regras 12-13), aplica-se a regra Modus Ponens na hipótese gerada pela instanciação (14). Com essa ação são gerados dois novos sequentes para se demonstrar: (i)* $H \vdash tG\_V(x) = t_k$ *e (ii)* $H \vdash \exists y \cdot y \in (edgeG \setminus \{mE(e_1), \ldots, mE(e_j)\}) \cup \{ed_1, \ldots, ed_k\} \wedge ((((\{mE(e_1), \ldots, mE(e_j)\} \lhd sourceG \cup \{ed_1 \mapsto t_1, \ldots, ed_k \mapsto t_k\})(y) = x \wedge \neg((\{mE(e_1), \ldots, mE(e_j)\} \lhd targetG \cup \{ed_1 \mapsto t_1, \ldots, ed_k \mapsto t_k\})(y) = x) \vee ((((\{mE(e_1), \ldots, mE(e_j)\} \lhd targetG \cup \{ed_1 \mapsto t_1, \ldots, ed_k \mapsto t_k\})(y) = x \wedge \neg((\{mE(e_1), \ldots, mE(e_j)\} \lhd sourceG \cup \{ed_1 \mapsto t_1, \ldots, ed_k \mapsto t_k\})(y) = x))$*. Para demonstrar (i) executa-se o provador ML (15). Em (ii) é aplicada de forma automática a regra* $\exists$ *hyp (16), em seguida instancia-se no objetivo o elemento* y *(representando o arco criado incidente no vértice criado do tipo* t*, pela hipótese de indução (17)) gerando vários sub-objetivos, os quais são demonstrados pela regra* $\top$ *goal (18) e*

executando o provador NewPP with lasso (21) e (23). No sequente (II), sub-árvore $\boxed{A}$*, a ferramenta aplica algumas regras de forma automática (regras 24-27), e o objetivo final é demonstrado pelo provador NewPP with lasso (29).*

A forma alternativa de apresentar a tática é utilizando a árvore de uso, exibida na Figura 91.

Descrição da Árvore de Uso: *Para utilizar a estratégia instancia-se o quantificador universal do objetivo. Em seguida remove-se o operador* $\in$ *na hipótese* $x \in vertG \cup \{v_1, \ldots, v_l\}$ *e se inicia a prova por casos na hipótese* $x \in vertG \vee x = v_1 \vee \ldots \vee x = v_l$*. Então, instancia-se o elemento* x *na hipótese de indução. Após, aplicar a regra Modus Ponens na hipótese resultante da instanciação e executar o provador ML. Na sequência, instancia-se no objetivo o elemento* y *e nos sub-objetivos gerados executa-se o provador NewPP with lasso. Já no último objetivo gerado o provador NewPP with lasso deve ser executado.*

B- Regra cria e preserva vértices do tipo t.

Na propriedade `propNotIsoVT` uma regra pode criar e preservar vértices do tipo t, deletando ou não arcos. A fim de demonstrar as obrigações de prova desse tipo, se apresenta a tática conforme segue:

1. Instanciar o quantificador universal no objetivo;
2. Remover o operador $\in$ na hipótese $x \in vertG \cup \{v_1, \ldots, v_l\}$;
3. Iniciar a prova por casos na hipótese $x \in vertG \vee x = v_1 \vee \ldots \vee x = v_l$;
4. Instanciar o elemento x na hipótese de indução;
5. Aplicar a regra Modus Ponens a partir da hipótese selecionada $tG\_V(x) = t \Rightarrow (\exists y \cdot y \in edgeG \wedge ((sourceG(y) = x \wedge \neg targetG(y) = x) \vee (targetG(y) = x \wedge \neg sourceG(y) = x)))$ e nos próximos sub-objetivos executar o provador ML;
6. Instanciar no objetivo o elemento y;
7. Executar o provador ML.

Essa estratégia, após executada na obrigação correspondente, gera uma árvore de prova. Os passos para demonstrar são semelhantes ao caso anterior (**caso A**), sendo a única diferença na parte final da demonstração, onde executa-se o provador ML ao invés de NewPP with lasso. A árvore de prova pode ser visualizada na Figura 90. Da mesma forma, a árvore de uso é observada na Figura 91.

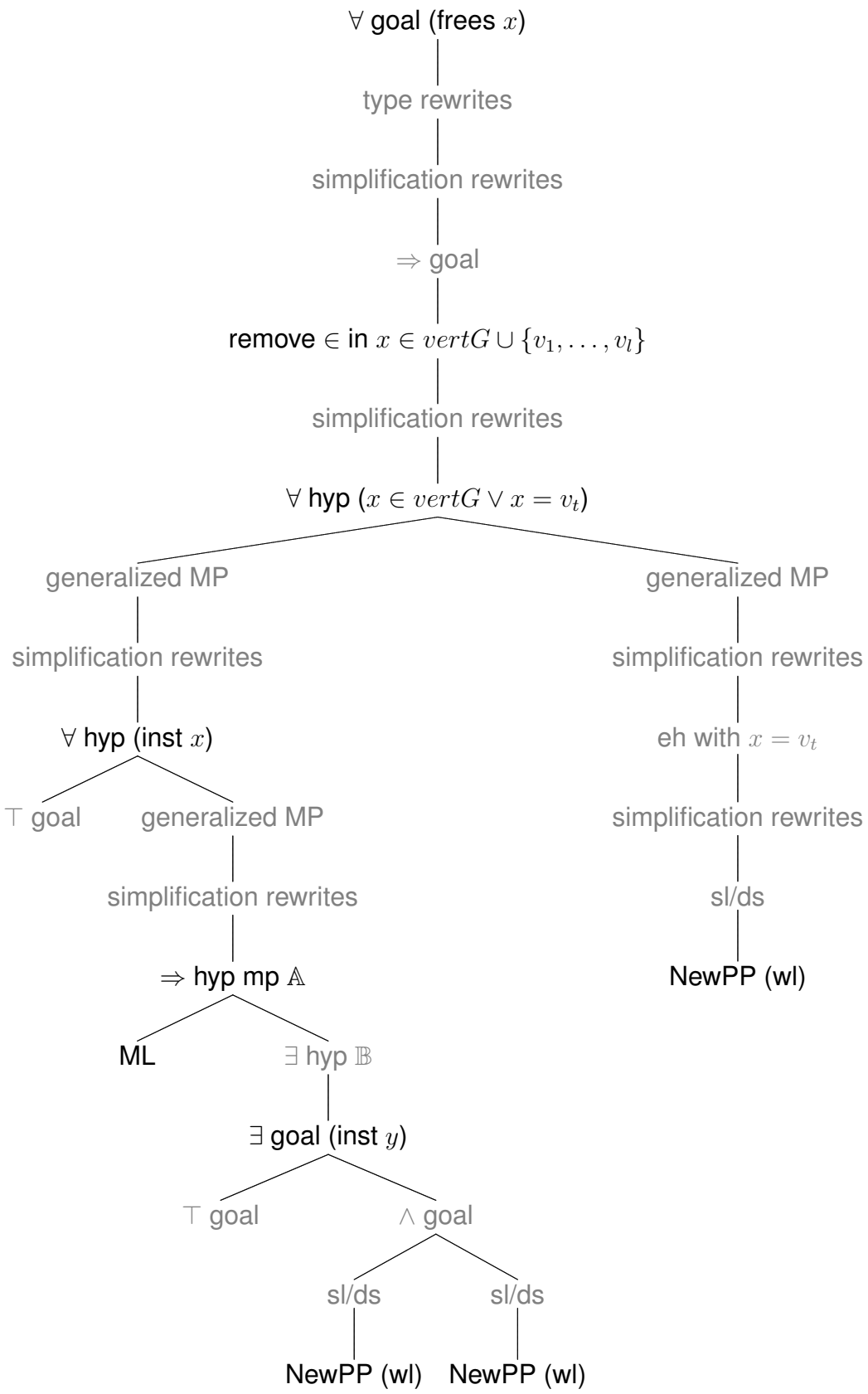


Figura 91: Árvore de Uso rule_i/propNotIsoVT/INV - Regra Cria e não Preserva Vértices do Tipo t

C- Regra não cria e preserva vértices do tipo t

Quando nenhum arco é deletado a prova dessa obrigação é realizada executando o provador NewPP with lasso. Já em regras nas quais algum arco (que não possua origem ou destino em um vértice do tipo t) for deletado a forma de demonstrar essa obrigação, é executando as seguintes ações:

1. Instanciar o quantificador universal do objetivo;
2. Instanciar o elemento x na hipótese de indução;
3. Instanciar o elemento y no objetivo;
4. Aplicar a operação de lasso no objetivo;
5. Executar NewPP with lasso;
6. Executar o provador ML.

Através da árvore de prova da Figura 92, é possível analisar cada regra utilizada na demonstração de rule_i/propNotIsoVT/INV (regra preserva e não cria vértices do tipo t).

Descrição da Árvore de Prova: *O primeiro sequente para se demonstrar é* $H \vdash \forall x \cdot x \in vertG \wedge tG\_V(x) = t \Rightarrow (\exists y \cdot y \in (edgeG \setminus \{mE(e_1), \ldots, mE(e_j)\}) \cup \{ed_1, \ldots, ed_k\} \wedge ((((\{mE(e_1), \ldots, mE(e_j)\} \lhd sourceG \cup \{ed_1 \mapsto t_1, \ldots, ed_k \mapsto t_k\})(y) = x \wedge \neg((\{mE(e_1), \ldots, mE(e_j)\} \lhd targetG \cup \{ed_1 \mapsto t_1, \ldots, ed_k \mapsto t_k\})(y) = x) \vee ((((\{mE(e_1), \ldots, mE(e_j)\} \lhd targetG \cup \{ed_1 \mapsto t_1, \ldots, ed_k \mapsto t_k\})(y) = x \wedge \neg((\{mE(e_1), \ldots, mE(e_j)\} \lhd sourceG \cup \{ed_1 \mapsto t_1, \ldots, ed_k \mapsto t_k\})(y) = x))$*. A fim de demonstrar esse sequente instancia-se o quantificador universal do objetivo (1). Após essa instanciação, são aplicadas de forma automática algumas regras (2-5), gerando um novo sequente* $H \vdash \exists y \cdot y \in (edgeG \setminus \{mE(e_1), \ldots, mE(e_j)\}) \cup \{ed_1, \ldots, ed_k\} \wedge ((((\{mE(e_1), \ldots, mE(e_j)\} \lhd sourceG \cup \{ed_1 \mapsto t_1, \ldots, ed_k \mapsto t_k\})(y) = x \wedge \neg((\{mE(e_1), \ldots, mE(e_j)\} \lhd targetG \cup \{ed_1 \mapsto t_1, \ldots, ed_k \mapsto t_k\})(y) = x) \vee ((((\{mE(e_1), \ldots, mE(e_j)\} \lhd targetG \cup \{ed_1 \mapsto t_1, \ldots, ed_k \mapsto t_k\})(y) = x \wedge \neg((\{mE(e_1), \ldots, mE(e_j)\} \lhd sourceG \cup \{ed_1 \mapsto t_1, \ldots, ed_k \mapsto t_k\})(y) = x))$*. Visando demonstrar este objetivo instancia-se o elemento* x *na hipótese de indução (6), resultando em dois sub-objetivos para se demonstrar: (I)* $H \vdash \top$*; (II)* $H \vdash \exists y \cdot y \in (edgeG \setminus \{mE(e_1), \ldots, mE(e_j)\}) \cup \{ed_1, \ldots, ed_k\} \wedge ((((\{mE(e_1), \ldots, mE(e_j)\} \lhd sourceG \cup \{ed_1 \mapsto t_1, \ldots, ed_k \mapsto t_k\})(y) = x \wedge \neg((\{mE(e_1), \ldots, mE(e_j)\} \lhd targetG \cup \{ed_1 \mapsto t_1, \ldots, ed_k \mapsto t_k\})(y) = x) \vee ((((\{mE(e_1), \ldots, mE(e_j)\} \lhd targetG \cup \{ed_1 \mapsto t_1, \ldots, ed_k \mapsto t_k\})(y) = x \wedge \neg((\{mE(e_1), \ldots, mE(e_j)\} \lhd sourceG \cup \{ed_1 \mapsto t_1, \ldots, ed_k \mapsto t_k\})(y) = x))$*. (I) é demonstrado automaticamente pela regra* $\top$ *goal (7). Em (II) são aplicadas algumas*

$\mathcal{A} = \{v_1, \ldots, v_l\}$
$\mathcal{B} = \{v_1 \mapsto t_1, \ldots, v_l \mapsto t_l\}$
$\mathcal{C} = \{mE(e_1), \ldots, mE(e_j)\}$
$\mathcal{D} = \{ed_1, \ldots, ed_k\}$
$\mathcal{E} = \{mE(e_1), \ldots, mE(e_j)\} \lhd sourceG \cup \{ed_1 \mapsto t_1, \ldots, ed_k \mapsto t_k\}$
$\mathcal{F} = \{mE(e_1), \ldots, mE(e_j)\} \lhd targetG \cup \{ed_1 \mapsto t_1, \ldots, ed_k \mapsto t_k\}$

NewPP RULES

$$\frac{\text{NewPP RULES}}{H \vdash y \in (edgeG \setminus \mathcal{C}) \cup \mathcal{D}}\ 17$$

$$\frac{H \vdash y \in (edgeG \setminus \mathcal{C}) \cup \mathcal{D}}{H \vdash y \in (edgeG \setminus \mathcal{C}) \cup \mathcal{D}}\ 16$$

$$\frac{H \vdash y \in (edgeG \setminus \mathcal{C}) \cup \mathcal{D}}{H \vdash y \in (edgeG \setminus \mathcal{C}) \cup \mathcal{D}}\ 15$$

$$\frac{H \vdash y \in (edgeG \setminus \mathcal{C}) \cup \mathcal{D}}{H \vdash y \in (edgeG \setminus \mathcal{C}) \cup \mathcal{D}}\ 14$$

ML RULES

$$\frac{\text{ML RULES}}{H \vdash (((\mathcal{E})(y) = x \land \neg((\mathcal{F})(y) = x) \lor (((\mathcal{F})(y) = x \land \neg((\mathcal{E})(y) = x)}\ 18$$

$$\frac{H \vdash y \in (edgeG \setminus \mathcal{C}) \cup \mathcal{D} \qquad H \vdash (((\mathcal{E})(y) = x \land \neg((\mathcal{F})(y) = x) \lor (((\mathcal{F})(y) = x \land \neg((\mathcal{E})(y) = x)}{H \vdash y \in (edgeG \setminus \mathcal{C}) \cup \mathcal{D} \land ((((\mathcal{E})(y) = x \land \neg((\mathcal{F})(y) = x) \lor (((\mathcal{F})(y) = x \land \neg((\mathcal{E})(y) = x))}\ 13$$

$$\frac{\text{True}}{H \vdash \top}\ 12$$

$$\frac{H \vdash \top \qquad H \vdash y \in (edgeG \setminus \mathcal{C}) \cup \mathcal{D} \land ((((\mathcal{E})(y) = x \land \neg((\mathcal{F})(y) = x) \lor (((\mathcal{F})(y) = x \land \neg((\mathcal{E})(y) = x))}{H \vdash \exists y \cdot y \in (edgeG \setminus \mathcal{C}) \cup \mathcal{D} \land ((((\mathcal{E})(y) = x \land \neg((\mathcal{F})(y) = x) \lor (((\mathcal{F})(y) = x \land \neg((\mathcal{E})(y) = x))}\ 11$$

$$\frac{H \vdash \exists y \cdot y \in (edgeG \setminus \mathcal{C}) \cup \mathcal{D} \land ((((\mathcal{E})(y) = x \land \neg((\mathcal{F})(y) = x) \lor (((\mathcal{F})(y) = x \land \neg((\mathcal{E})(y) = x))}{H \vdash \exists y \cdot y \in (edgeG \setminus \mathcal{C}) \cup \mathcal{D} \land ((((\mathcal{E})(y) = x \land \neg((\mathcal{F})(y) = x) \lor (((\mathcal{F})(y) = x \land \neg((\mathcal{E})(y) = x))}\ 10$$

$$\frac{H \vdash \exists y \cdot y \in (edgeG \setminus \mathcal{C}) \cup \mathcal{D} \land ((((\mathcal{E})(y) = x \land \neg((\mathcal{F})(y) = x) \lor (((\mathcal{F})(y) = x \land \neg((\mathcal{E})(y) = x))}{H \vdash \exists y \cdot y \in (edgeG \setminus \mathcal{C}) \cup \mathcal{D} \land ((((\mathcal{E})(y) = x \land \neg((\mathcal{F})(y) = x) \lor (((\mathcal{F})(y) = x \land \neg((\mathcal{E})(y) = x))}\ 9$$

$$\frac{H \vdash \exists y \cdot y \in (edgeG \setminus \mathcal{C}) \cup \mathcal{D} \land ((((\mathcal{E})(y) = x \land \neg((\mathcal{F})(y) = x) \lor (((\mathcal{F})(y) = x \land \neg((\mathcal{E})(y) = x))}{H \vdash \exists y \cdot y \in (edgeG \setminus \mathcal{C}) \cup \mathcal{D} \land ((((\mathcal{E})(y) = x \land \neg((\mathcal{F})(y) = x) \lor (((\mathcal{F})(y) = x \land \neg((\mathcal{E})(y) = x))}\ 8$$

$$\frac{\text{True}}{H \vdash \top}\ 7$$

$$\frac{H \vdash \top \qquad H \vdash \exists y \cdot y \in (edgeG \setminus \mathcal{C}) \cup \mathcal{D} \land ((((\mathcal{E})(y) = x \land \neg((\mathcal{F})(y) = x) \lor (((\mathcal{F})(y) = x \land \neg((\mathcal{E})(y) = x))}{H \vdash \exists y \cdot y \in (edgeG \setminus \mathcal{C}) \cup \mathcal{D} \land ((((\mathcal{E})(y) = x \land \neg((\mathcal{F})(y) = x) \lor (((\mathcal{F})(y) = x \land \neg((\mathcal{E})(y) = x))}\ 6$$

$$\frac{H \vdash \exists y \cdot y \in (edgeG \setminus \mathcal{C}) \cup \mathcal{D} \land ((((\mathcal{E})(y) = x \land \neg((\mathcal{F})(y) = x) \lor (((\mathcal{F})(y) = x \land \neg((\mathcal{E})(y) = x))}{H \vdash \exists y \cdot y \in (edgeG \setminus \mathcal{C}) \cup \mathcal{D} \land ((((\mathcal{E})(y) = x \land \neg((\mathcal{F})(y) = x) \lor (((\mathcal{F})(y) = x \land \neg((\mathcal{E})(y) = x))}\ 5$$

$$\frac{H \vdash \exists y \cdot y \in (edgeG \setminus \mathcal{C}) \cup \mathcal{D} \land ((((\mathcal{E})(y) = x \land \neg((\mathcal{F})(y) = x) \lor (((\mathcal{F})(y) = x \land \neg((\mathcal{E})(y) = x))}{H \vdash x \in vertG \land tG\_V(x) = t \Rightarrow (\exists y \cdot y \in (edgeG \setminus \mathcal{C}) \cup \mathcal{D} \land ((((\mathcal{E})(y) = x \land \neg((\mathcal{F})(y) = x) \lor (((\mathcal{F})(y) = x \land \neg((\mathcal{E})(y) = x)))}\ 4$$

$$\frac{H \vdash x \in vertG \land tG\_V(x) = t \Rightarrow (\exists y \cdot y \in (edgeG \setminus \mathcal{C}) \cup \mathcal{D} \land ((((\mathcal{E})(y) = x \land \neg((\mathcal{F})(y) = x) \lor (((\mathcal{F})(y) = x \land \neg((\mathcal{E})(y) = x)))}{H \vdash x \in vertG \land tG\_V(x) = t \Rightarrow (\exists y \cdot y \in (edgeG \setminus \mathcal{C}) \cup \mathcal{D} \land ((((\mathcal{E})(y) = x \land \neg((\mathcal{F})(y) = x) \lor (((\mathcal{F})(y) = x \land \neg((\mathcal{E})(y) = x)))}\ 3$$

$$\frac{H \vdash x \in vertG \land tG\_V(x) = t \Rightarrow (\exists y \cdot y \in (edgeG \setminus \mathcal{C}) \cup \mathcal{D} \land ((((\mathcal{E})(y) = x \land \neg((\mathcal{F})(y) = x) \lor (((\mathcal{F})(y) = x \land \neg((\mathcal{E})(y) = x)))}{H \vdash x \in vertG \land tG\_V(x) = t \Rightarrow (\exists y \cdot y \in (edgeG \setminus \mathcal{C}) \cup \mathcal{D} \land ((((\mathcal{E})(y) = x \land \neg((\mathcal{F})(y) = x) \lor (((\mathcal{F})(y) = x \land \neg((\mathcal{E})(y) = x)))}\ 2$$

$$\frac{H \vdash x \in vertG \land tG\_V(x) = t \Rightarrow (\exists y \cdot y \in (edgeG \setminus \mathcal{C}) \cup \mathcal{D} \land ((((\mathcal{E})(y) = x \land \neg((\mathcal{F})(y) = x) \lor (((\mathcal{F})(y) = x \land \neg((\mathcal{E})(y) = x)))}{H \vdash \forall x \cdot x \in vertG \land tG\_V(x) = t \Rightarrow (\exists y \cdot y \in (edgeG \setminus \mathcal{C}) \cup \mathcal{D} \land ((((\mathcal{E})(y) = x \land \neg((\mathcal{F})(y) = x) \lor (((\mathcal{F})(y) = x \land \neg((\mathcal{E})(y) = x)))}\ 1$$

1 $\forall$ goal (frees x); 2 type rewrites; 3 simplification rewrites; 4 $\Rightarrow$ goal; 5 he with $tG\_V(x) = t$; 6 $\forall$ hyp (inst x); 7 $\top$ goal; 8 generalized MP; 9 simplification rewrites; 10 $\exists$ hyp $(\exists y \cdot y \in edgeG \land ((sourceG(y) = x \land \neg targetG(y) = x) \lor (targetG(y) = x \land \neg sourceG(y) = x)))$; 11 $\exists$ goal (inst y); 12 $\top$ goal; 13 $\land$ goal; 14 sl/ds; 15 he with $tG\_V(x) = t$; 16 sl/ds; 17 NewPP with lasso; 18 ML

Figura 92: Árvore de Prova rule_i/propNotIsoVT/INV - Regra Preserva e não Cria Vértices do Tipo t.

regras automáticas (regras 8-10), em seguida instancia-se o elemento y no objetivo (11). Após essa instanciação são gerado dois novos objetivos: (i) $H \vdash \top$, demonstrado automaticamente (12) e (ii) $H \vdash y \in (edgeG \setminus \{mE(e_1), \ldots, mE(e_j)\}) \cup \{ed_1, \ldots, ed_k\} \wedge ((((\{mE(e_1), \ldots, mE(e_j)\} \lhd sourceG \cup \{ed_1 \mapsto t_1, \ldots, ed_k \mapsto t_k\})(y) = x \wedge \neg((\{mE(e_1), \ldots, mE(e_j)\} \lhd targetG \cup \{ed_1 \mapsto t_1, \ldots, ed_k \mapsto t_k\})(y) = x) \vee ((((\{mE(e_1), \ldots, mE(e_j)\} \lhd targetG \cup \{ed_1 \mapsto t_1, \ldots, ed_k \mapsto t_k\})(y) = x \wedge \neg((\{mE(e_1), \ldots, mE(e_j)\} \lhd sourceG \cup \{ed_1 \mapsto t_1, \ldots, ed_k \mapsto t_k\})(y) = x))$. Em (ii), primeiramente de forma automática é aplica a regra $\wedge$ goal (13), gerando dois sub-objetivos para demonstrar: (i) $H \vdash y \in (edgeG \setminus \{mE(e_1), \ldots, mE(e_j)\}) \cup \{ed_1, \ldots, ed_k\}$; (ii) $H \vdash ((((\{mE(e_1), \ldots, mE(e_j)\} \lhd sourceG \cup \{ed_1 \mapsto t_1, \ldots, ed_k \mapsto t_k\})(y) = x \wedge \neg((\{mE(e_1), \ldots, mE(e_j)\} \lhd targetG \cup \{ed_1 \mapsto t_1, \ldots, ed_k \mapsto t_k\})(y) = x) \vee ((((\{mE(e_1), \ldots, mE(e_j)\} \lhd targetG \cup \{ed_1 \mapsto t_1, \ldots, ed_k \mapsto t_k\})(y) = x \wedge \neg((\{mE(e_1), \ldots, mE(e_j)\} \lhd sourceG \cup \{ed_1 \mapsto t_1, \ldots, ed_k \mapsto t_k\})(y) = x)$. Em (i) realiza-se uma operação de lasso (regras 14-16) e o último sub-objetivo é demonstrado executando o provador NewPP with lasso (17). O sequente (ii) é demonstrado aplicando o provador ML.

A Figura 93 apresenta a árvore de uso para essa demonstração.

Descrição da Árvore de Uso: *Para utilizar a estratégia de prova, primeiramente instancia-se o quantificador universal do objetivo. Em seguida, instancia-se o elemento x na hipótese de indução. O próximo passo é instanciar o elemento y no objetivo e executar uma operação de lasso. Rodar o provador NewPP with lasso e em seguida executar o provador ML.*

D- Regra não envolve vértices do tipo t

Quando nenhum arco é deletado pela regra a demonstração é realizada executando o provador NewPP with lasso. Quando a regra deleta arcos que não possuam origem ou destino em vértices do tipo t, a prova é executada seguindo as seguintes ações:

1. Instanciar o quantificador universal do objetivo;
2. Remover o operador $\in$ na hipótese $x \in vertG \cup \{v_1, \ldots, v_l\}$;
3. Iniciar a prova por casos na hipótese $x \in vertG \vee x = v_1 \vee \ldots \vee x = v_l$;
4. Instanciar o elemento x na hipótese de indução;
5. Aplicar a regra Modus Ponens a partir da hipótese $tG\_V(x) = t \Rightarrow (\exists y \cdot y \in edgeG \wedge ((sourceG(y) = x \wedge \neg targetG(y) = x) \vee (targetG(y) = x \wedge \neg sourceG(y) = x)))$ e nos próximos sub-objetivos executar o provador NewPP with lasso;

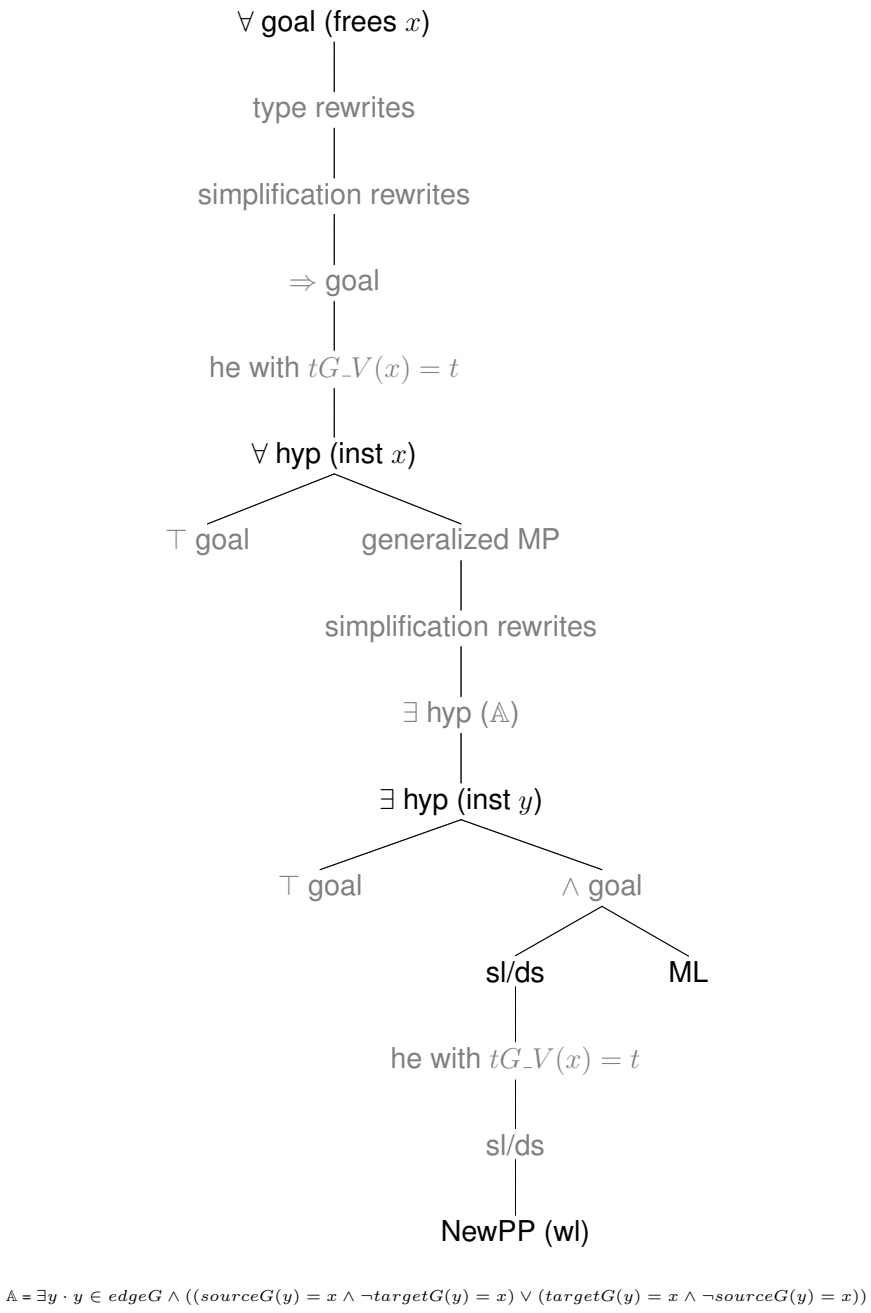


Figura 93: Árvore de Uso rule_i/propNotIsoVT/INV - Regra Preserva e não Cria vértices do tipo t

6 Instanciar no objetivo o elemento y e nos sub-objetivos gerados executar o provador NewPP with lasso.

A Figura 94 apresenta a árvore de prova resultante da demonstração de rule_i/propNotIsoVT/INV, quando a regra não envolve vértices do tipo t.

Descrição da Árvore de Prova: *O primeiro sequente para se demonstrar é* $H \vdash \forall x \cdot x \in vertG \wedge tG\_V(x) = t \Rightarrow (\exists y \cdot y \in (edgeG \setminus \{mE(e_1), \ldots, mE(e_j)\}) \cup \{ed_1, \ldots, ed_k\} \wedge$

$\mathcal{A}$ = $\{v_1, \ldots, v_l\}$ $\quad$ $\mathcal{B}$ = $\{v_1 \mapsto t_1, \ldots, v_l \mapsto t_l\}$ $\quad$ $\mathcal{C}$ = $\{mE(e_1), \ldots, mE(e_j)\}$ $\quad$ $\mathcal{D}$ = $\{ed_1, \ldots, ed_k\}$
$\mathcal{E}$ = $\{mE(e_1), \ldots, mE(e_j)\} \lhd sourceG \cup \{ed_1 \mapsto t_1, \ldots, ed_k \mapsto t_k\}$ $\quad$ $\mathcal{F}$ = $\{mE(e_1), \ldots, mE(e_j)\} \lhd targetG \cup \{ed_1 \mapsto t_1, \ldots, ed_k \mapsto t_k\}$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{H \vdash x \in vertG \cup \mathcal{A} \wedge (tG\_V \cup \mathcal{B})(x) = t \Rightarrow (\exists y \cdot y \in (edgeG \setminus \mathcal{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)))}\ 4}{H \vdash x \in vertG \cup \mathcal{A} \wedge (tG\_V \cup \mathcal{B})(x) = t \Rightarrow (\exists y \cdot y \in (edgeG \setminus \mathcal{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)))}\ 3}{H \vdash x \in vertG \cup \mathcal{A} \wedge (tG\_V \cup \mathcal{B})(x) = t \Rightarrow (\exists y \cdot y \in (edgeG \setminus \mathcal{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)))}\ 2}{H \vdash \forall x \cdot x \in vertG \cup \mathcal{A} \wedge (tG\_V \cup \mathcal{B})(x) = t \Rightarrow (\exists y \cdot y \in (edgeG \setminus \mathcal{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)))}\ 1}{H \vdash y \in (edgeG \setminus \mathcal{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))}\ 5}{H \vdash \exists y \cdot y \in (edgeG \setminus \mathcal{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))}\ 6}{H \vdash \exists y \cdot y \in (edgeG \setminus \mathcal{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))}\ 8}{H \vdash x \in vertG \wedge tG\_V(x) = (tG\_V \cup \mathcal{B})(x)}\ 9}{H \vdash x \in vertG}\ 10}{H \vdash x \in vertG}\ 11 \qquad \dfrac{\dfrac{H \vdash x \in vertG \wedge tG\_V(x) = (tG\_V \cup \mathcal{B})(x)}{H \vdash x \in vertG \wedge tG\_V(x) = (tG\_V \cup \mathcal{B})(x)}\ 12}{}\ 13 \qquad \boxed{\text{A}} \qquad \dfrac{\dfrac{}{H \vdash \top}\ 7}{}\ \text{True}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{H \vdash y \in (edgeG \setminus \mathcal{C})}\ 19}{H \vdash y \in (edgeG \setminus \mathcal{C})}\ 18}{H \vdash y \in (edgeG \setminus \mathcal{C}) \cup \mathcal{D} \wedge ((((\mathcal{E})(y) = x \wedge \neg((\mathcal{F})(y) = x) \vee (((\mathcal{F})(y) = x \wedge \neg((\mathcal{E})(y) = x))}\ 15}{H \vdash \exists y \cdot y \in (edgeG \setminus \mathcal{C}) \cup \mathcal{D} \wedge ((((\mathcal{E})(y) = x \wedge \neg((\mathcal{F})(y) = x) \vee (((\mathcal{F})(y) = x \wedge \neg((\mathcal{E})(y) = x))}\ 14}{H \vdash \exists y \cdot y \in (edgeG \setminus \mathcal{C}) \cup \mathcal{D} \wedge ((((\mathcal{E})(y) = x \wedge \neg((\mathcal{F})(y) = x) \vee (((\mathcal{F})(y) = x \wedge \neg((\mathcal{E})(y) = x))}\ 16}{H \vdash \top}\ \text{True} \qquad \dfrac{\dfrac{}{H \vdash y \in (edgeG \setminus \mathcal{C})}\ 17}{}\ \text{NewPP RULES} \qquad \dfrac{\dfrac{H \vdash (((\mathcal{E})(y) = x \wedge \neg((\mathcal{F})(y) = x) \vee (((\mathcal{F})(y) = x \wedge \neg((\mathcal{E})(y) = x)}\ 20}{H \vdash (((\mathcal{E})(y) = x \wedge \neg((\mathcal{F})(y) = x) \vee (((\mathcal{F})(y) = x \wedge \neg((\mathcal{E})(y) = x)}\ 21}{}\ \text{NewPP RULES} \qquad \boxed{\text{A}}$$

1 $\forall$ goal (frees x); 2 type rewrites; 3 simplification rewrites; 4 $\Rightarrow$ goal; 5 he with $(tG\_V \cup \{v_1 \mapsto t_1, \ldots, v_l \mapsto t_l\})(x) = t$); 6 $\forall$ hyp (inst x); 7 $\top$ goal; 8 $\Rightarrow$ hyp mp $(tG\_V(x) = t \Rightarrow (\exists y \cdot y \in edgeG \wedge ((sourceG(y) = x \wedge \neg targetG(y) = x) \vee (targetG(y) = x \wedge \neg sourceG(y) = x))))$; 9 $\wedge$ goal; 10 sl/ds; 11 NewPP with lasso; 12 sl/ds; 13 NewPP with lasso; 14 $\exists$ hyp $(\exists y \cdot y \in \wedge((sourceG(y) = x \wedge \neg targetG(y) = x) \vee (targetG(y) = x \wedge \neg sourceG(y) = x)))$; 15 $\exists$ goal (inst y); 16 $\top$ goal; 17 $\wedge$ goal; 18 sl/ds; 19 NewPP with lasso; 20 sl/ds; 21 NewPP with lasso

Figura 94: Árvore de Prova rule_i/propNotIsoVT/INV - Regra não Envolve Vértices do Tipo t

$((((\{mE(e_1),\ldots,mE(e_j)\} \lhd sourceG \cup \{ed_1 \mapsto t_1,\ldots,ed_k \mapsto t_k\})(y) = x \wedge \neg((\{mE(e_1),\ldots,mE(e_j)\} \lhd targetG \cup \{ed_1 \mapsto t_1,\ldots,ed_k \mapsto t_k\})(y) = x) \vee (((\{mE(e_1),\ldots,mE(e_j)\} \lhd targetG \cup \{ed_1 \mapsto t_1,\ldots,ed_k \mapsto t_k\})(y) = x \wedge \neg((\{mE(e_1),\ldots,mE(e_j)\} \lhd sourceG \cup \{ed_1 \mapsto t_1,\ldots,ed_k \mapsto t_k\})(y) = x)))$. *Para demonstrar esse sequente instancia-se o quantificador universal do objetivo (1). Em seguida, são aplicadas algumas regra automaticamente (regras 2-5), gerando um novo sequente para se demonstrar* $H \vdash \exists y \cdot y \in (edgeG \setminus \{mE(e_1),\ldots,mE(e_j)\}) \cup \{ed_1,\ldots,ed_k\} \wedge ((((\{mE(e_1),\ldots,mE(e_j)\} \lhd sourceG \cup \{ed_1 \mapsto t_1,\ldots,ed_k \mapsto t_k\})(y) = x \wedge \neg((\{mE(e_1),\ldots,mE(e_j)\} \lhd targetG \cup \{ed_1 \mapsto t_1,\ldots,ed_k \mapsto t_k\})(y) = x) \vee (((\{mE(e_1),\ldots,mE(e_j)\} \lhd targetG \cup \{ed_1 \mapsto t_1,\ldots,ed_k \mapsto t_k\})(y) = x \wedge \neg((\{mE(e_1),\ldots,mE(e_j)\} \lhd sourceG \cup \{ed_1 \mapsto t_1,\ldots,ed_k \mapsto t_k\})(y) = x))$. *A fim de demonstrar este objetivo instancia-se o elemento* x *na hipótese de indução (6), com isso é gerado o objetivo* $H \vdash \top$*, que é demonstrado de forma automática (7), em seguida aplica-se a regra Modus Ponens na hipótese resultante da instanciação, gerando dois novos objetivos para se demonstrar: (I)* $H \vdash x \in vertG \wedge tG\_V(x) = (tG\_V \cup \{v_1 \mapsto t_1,\ldots,v_l \mapsto t_l\})(x)$ *; (II)* $H \vdash \exists y \cdot y \in (edgeG \setminus \{mE(e_1),\ldots,mE(e_j)\}) \cup \{ed_1,\ldots,ed_k\} \wedge ((((\{mE(e_1),\ldots,mE(e_j)\} \lhd sourceG \cup \{ed_1 \mapsto t_1,\ldots,ed_k \mapsto t_k\})(y) = x \wedge \neg((\{mE(e_1),\ldots,mE(e_j)\} \lhd targetG \cup \{ed_1 \mapsto t_1,\ldots,ed_k \mapsto t_k\})(y) = x) \vee (((\{mE(e_1),\ldots,mE(e_j)\} \lhd targetG \cup \{ed_1 \mapsto t_1,\ldots,ed_k \mapsto t_k\})(y) = x \wedge \neg((\{mE(e_1),\ldots,mE(e_j)\} \lhd sourceG \cup \{ed_1 \mapsto t_1,\ldots,ed_k \mapsto t_k\})(y) = x))$. *Em (I) é aplicada a regra* $\wedge$ *goal de forma automática (9), o que resulta em dois sub-objetivos: (i)* $H \vdash x \in vertG$ *e (ii)* $H \vdash x \in vertG \wedge tG\_V(x) = (tG\_V \cup \{v_1 \mapsto t_1,\ldots,v_l \mapsto t_l\})(x)$*, ambos demonstrados por NewPP with lasso (11 e 13). A fim de demonstrar (II), é aplicada a regra* $\exists$ *hyp de forma automática (14). Em seguida, instancia-se o elemento* y *no objetivo, gerando o sequente* $H \vdash \top$*, demonstrado automaticamente (16). Após é aplicada a regra automática* $\wedge$ *goal (17), gerando dois sub-objetivos: (i)* $H \vdash y \in (edgeG \setminus \{mE(e_1),\ldots,mE(e_j)\})$*; e (ii)* $H \vdash ((((\{mE(e_1),\ldots,mE(e_j)\} \lhd sourceG \cup \{ed_1 \mapsto t_1,\ldots,ed_k \mapsto t_k\})(y) = x \wedge \neg((\{mE(e_1),\ldots,mE(e_j)\} \lhd targetG \cup \{ed_1 \mapsto t_1,\ldots,ed_k \mapsto t_k\})(y) = x) \vee (((\{mE(e_1),\ldots,mE(e_j)\} \lhd targetG \cup \{ed_1 \mapsto t_1,\ldots,ed_k \mapsto t_k\})(y) = x \wedge \neg((\{mE(e_1),\ldots,mE(e_j)\} \lhd sourceG \cup \{ed_1 \mapsto t_1,\ldots,ed_k \mapsto t_k\})(y) = x)$*. (i) e (ii) são descarregados por NewPP with lasso.*

A árvore de uso representando essa demonstração é mostrada na Figura 95.

Descrição da Árvore de Uso: *A fim de utilizar a tática, instancia-se o quantificador universal do objetivo, após instancia-se o elemento* x *na hipótese de indução. Aplica-se a regra Modus Ponens na hipótese gerada pela instanciação anterior e nos próximos sub-objetivos executa-se o provador NewPP with lasso. Após instancia-se o elemento* y *no objetivo do sequente e nos próximos sub-objetivos gerados executa-se o provador NewPP with lasso.*

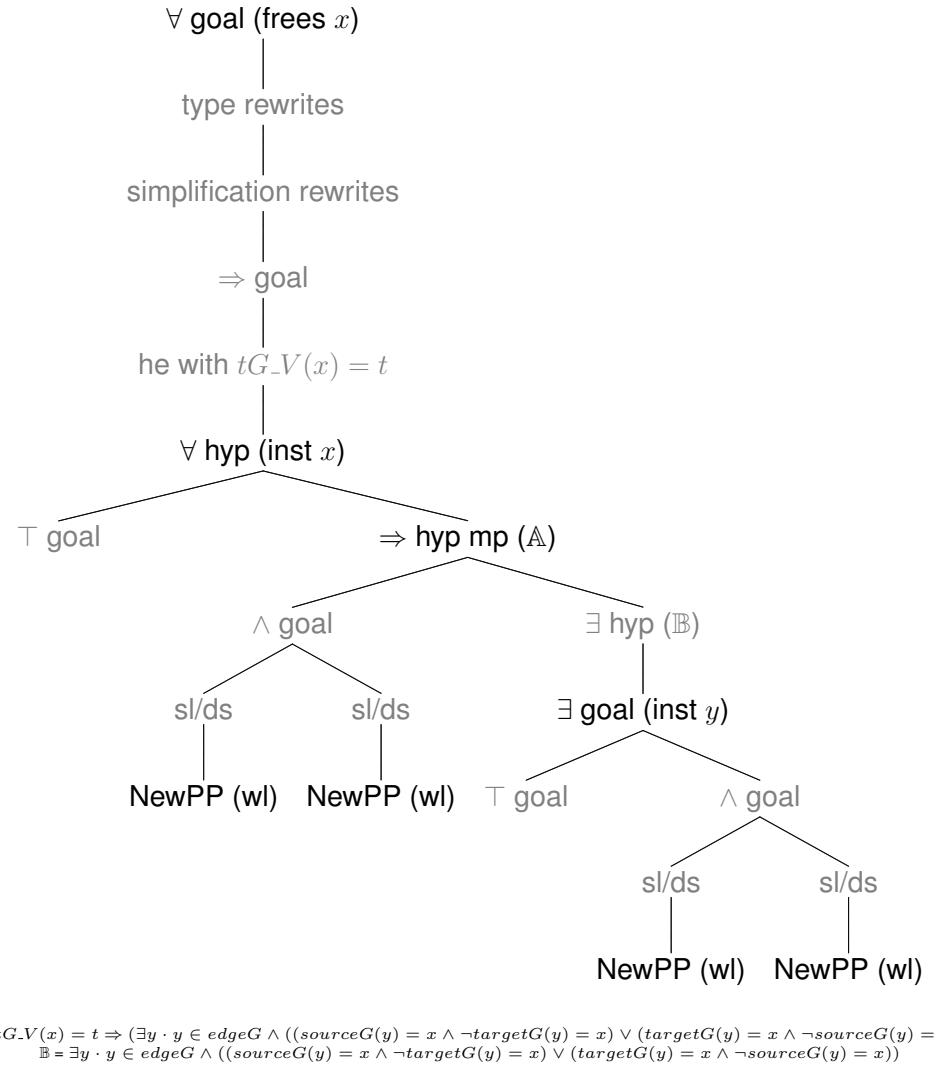


Figura 95: Árvore de Uso rule_i/propNotIsoVT/INV - Regra não Envolve Vértices do Tipo t

4.9 Propriedade propAllSVertSrcTEd

A propriedade propAllSVertSrcTEd ($\forall x \cdot ((x \in vertG \wedge tG\_V(x) = s) \Rightarrow (\exists y \cdot (y \in edgeG \wedge tG\_E(y) = t \wedge sourceG(y) = x)))$) assegura que em todos os estados do sistema, todo o vértice do tipo s é origem de um arco do tipo t. As alterações nas variáveis $vertG$, tG_V, $edgeG$, tG_E e $sourceG$ acabam gerando uma obrigação de prova a ser demonstrada. Para descarregar a obrigação INITIALISATION/propAllSVertSrcTEd/INV, gerada para o evento de inicialização, aplicam-se as seguintes ações:

1. Instanciar o quantificador universal do objetivo;
2. Remover o operador $\in$ na hipótese $x \in vertG$;
3. Iniciar a prova por casos na hipótese $x = v_1 \vee \ldots \vee x = v_n$, considerando $vertG$

inicializada com n vértices;

4. Em cada caso, aplicar uma das ações até concluir a demonstração:
   (a) Para vértices v_s (tipo s) instanciar o arco e_j (tipo t) que possui origem em v_s;
   (b) Para vértices v_i (não do tipo t) executar o provador ML;

A Figura 96 apresenta a árvore de prova resultante da demonstração.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{x = v_i, s = s_k \vdash \exists y{\cdot} y \in edgeG \wedge tG\_E(y) = t \wedge sourceG(y) = x}\,\text{ML RULES}}{x = v_n, tG\_V(v_i) = s \vdash \exists y{\cdot} y \in edgeG \wedge tG\_E(y) = t \wedge sourceG(y) = x}\,17}{tG\_V(x) = s, x = v_i \vdash \exists y{\cdot} y \in edgeG \wedge tG\_E(y) = t \wedge sourceG(y) = x}\,16}{tG\_V(x) = s, x = v_i, \top \vee x = v_n \vdash \exists y{\cdot} y \in edgeG \wedge tG\_E(y) = t \wedge sourceG(y) = x}\,15}{tG\_V(x) = s, x = v_i \vdash \exists y{\cdot} y \in edgeG \wedge tG\_E(y) = s \wedge sourceG(y) = x}\,14}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{x = v_s \vdash \top}\,\text{True}}{x = v_s \vdash e_j \in edgeG \wedge tG\_E(e_j) = t \wedge sourceG(e_j) = x}\,10 \quad \dfrac{\dfrac{}{x = v_s \vdash \top}\,\text{True}}{x = v_s \vdash e_j \in edgeG \wedge tG\_E(e_j) = t \wedge sourceG(e_j) = x}\,12}{x = v_s \vdash \exists y{\cdot} y \in edgeG \wedge tG\_E(y) = t \wedge sourceG(y) = x}\,11}{x = v_s, tG\_V(v_s) = s \vdash \exists y{\cdot} y \in edgeG \wedge tG\_E(y) = t \wedge sourceG(y) = x}\,9}{tG\_V(x) = s, x = v_s \vdash \exists y{\cdot} y \in edgeG \wedge tG\_E(y) = t \wedge sourceG(y) = x}\,8}{tG\_V(x) = s, x = v_s, \top \vee x = v_n \vdash \exists y{\cdot} y \in edgeG \wedge tG\_E(y) = t \wedge sourceG(y) = x}\,7}{tG\_V(x) = s, x = v_s \vdash \exists y{\cdot} y \in edgeG \wedge tG\_E(y) = t \wedge sourceG(y) = x}\,6}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{}{H \vdash \exists y{\cdot} y \in edgeG \wedge tG\_E(y) = t \wedge sourceG(y) = x}\,4}{H \vdash \exists y{\cdot} y \in edgeG \wedge tG\_E(y) = t \wedge sourceG(y) = x}\,3}{\vdash x \in vertG \wedge tG\_V(x) = s \Rightarrow (\exists y{\cdot} y \in edgeG \wedge tG\_E(y) = t \wedge sourceG(y) = x)}\,2}{\vdash \forall x{\cdot} x \in vertG \wedge tG\_V(x) = s \Rightarrow (\exists y{\cdot} (y \in edgeG \wedge tG\_E(y) = t \wedge sourceG(y) = x))}\,1$$

1 $\forall$ goal (free x); 2 $\Rightarrow$ goal; 3 remove $\in$ in $x \in \{v_1, \ldots, v_n\}$; 4 $\forall$ hyp ($x = v_1 \vee \ldots \vee x = v_n$);
5 generalized MP; 6 simplification rewrites; 7 eh with $x = v_s$; 8 simplification rewrites; 9 $\exists$ goal (inst e_j);
10 $\top$ goal; 11 simplification rewrites; 12 $\top$ goal; 13 generalized MP; 14 simplification rewrites;
15 eh with $x = v_i$; 16 simplification rewrites; 17 ML

Figura 96: Árvore de Prova INITIALISATION/propAllSVertSrcTEd/INV

Descrição da Árvore de Prova: *O objetivo inicial para se demonstrar é* $\vdash \forall x{\cdot} x \in vertG \wedge tG\_V(x) = s \Rightarrow (\exists y{\cdot}(y \in edgeG \wedge tG\_E(y) = t \wedge sourceG(y) = x))$. *Para demonstrar esse sequente instancia-se o quantificador universal do objetivo (1), após é aplicada de forma automática a regra* $\Rightarrow$ *goal (2). Em seguida, remove-se o operador* $\in$ *na hipótese* $x \in vertG$ *(3) e se inicia a prova por casos através da hipótese* $x = v_1 \vee \ldots \vee x = v_n$ *(4). Para cada vértice do grafo inicial, primeiramente é aplicada algumas*

regras automáticas (ações 5-8 e 13-16), em seguida são gerados os sub-objetivos dos casos: (I) $tG\_V(x) = s, x = v_s \vdash \exists y \cdot y \in edgeG \wedge tG\_E(y) = t \wedge sourceG(y) = x$*; (II)* $tG\_V(x) = s, x = v_i \vdash \exists y \cdot y \in edgeG \wedge tG\_E(y) = t \wedge sourceG(y) = x$*. O sequente (I) representa o objetivo gerado para um vértice do tipo* s*, para demonstrar este tipo de sequente instancia-se no objetivo o nome do arco* e_j *do tipo* t *que possui origem no vértice do tipo* s *após instanciação são gerados dois sub-objetivos: (i)* $x = v_s \vdash \top$ *e (ii)* $x = v_s \vdash e_j \in edgeG \wedge tG\_E(e_j) = t \wedge sourceG(e_j) = x$*. (i) e (ii) são demonstrados automaticamente (10-12). Já o sequente (II) representa o objetivo gerado para um vértice diferente do tipo* s*, para demonstrar esse sequente executa-se o provador ML (17). Essas ações são necessárias em cada vértice do grafo inicial do sistema.*

Também é possível representar a tática proposta através da árvore de uso ilustrada na Figura 97.

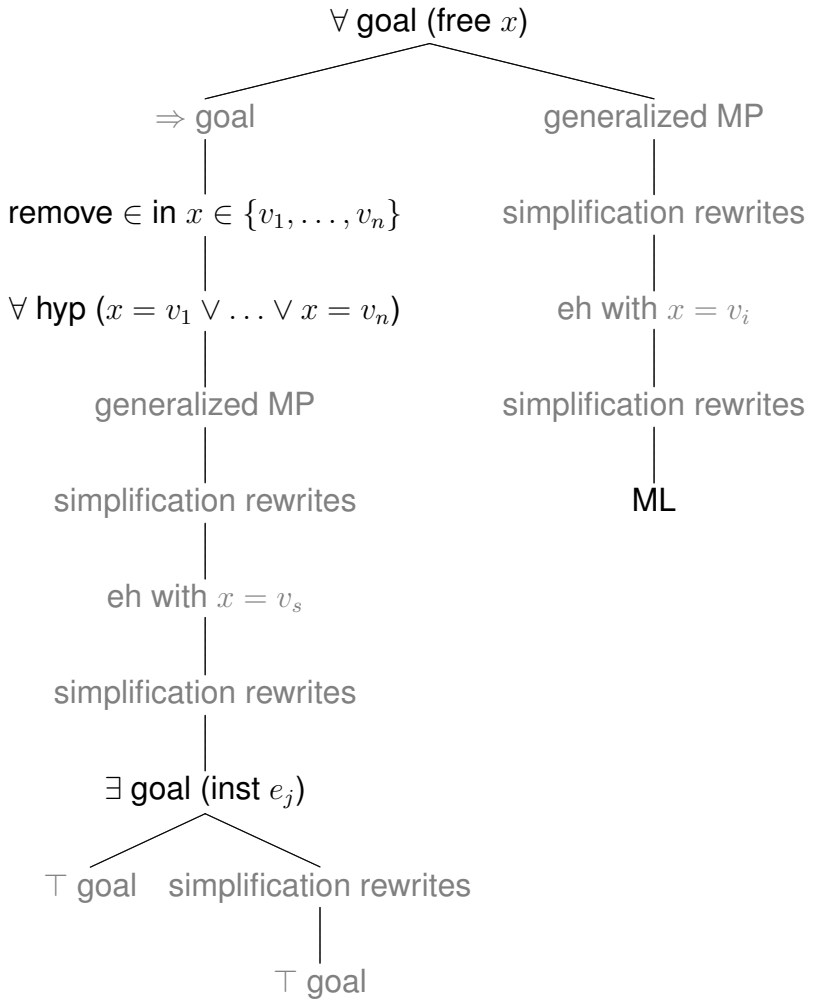


Figura 97: Árvore de Uso INITIALISATION/propAllSVertSrcTEd/INV

Descrição da Árvore de Uso: *Para utilizar essa tática, se elimina o quantificador universal do objetivo e após remove-se o operador* $\in$ *da hipótese* $x \in \{1, \ldots, n\}$*. Se*

inicia a prova por casos a partir da hipótese $x = v_1 \vee \ldots \vee x = v_n$*. Para cada caso: se o vértice solicitado nas hipóteses é do tipo* s *(*v_s*), então instancia-se no objetivo o nome do arco do tipo* t *que possui origem neste vértice. Caso contrário, executa-se o provador ML. Essas ações repetem-se para cada vértice do grafo inicial.*

Nessa propriedade as obrigações de prova para as regras são geradas no seguinte formato $\forall x \cdot x \in vertG \cup \{v_1, \ldots, v_l\} \wedge (tG\_V \cup \{v_1 \mapsto t_1, \ldots, v_l \mapsto t_l\})(x) = s \Rightarrow (\exists y \cdot y \in (edgeG \setminus \{mE(e_1), \ldots, mE(e_j)\}) \cup \{ed_1, \ldots, ed_k\} \wedge ((\{mE(e_1), \ldots, mE(e_j)\} \lhd tG\_E) \cup \{ed_1 \mapsto t_1, \ldots, ed_k \mapsto t_k\})(y) = t \wedge ((\{mE(e_1), \ldots, mE(e_j)\} \lhd sourceG) \cup \{ed_1 \mapsto mV(v_1), \ldots, ed_k \mapsto v_l\})(y) = x)$, considerando que j arcos são deletados, k arcos e l vértices são criados pela aplicação da regra. Neste sentido, considera-se as seguintes ações de uma regra: não preserva e cria vértice do tipo s que é origem de arco do tipo t; preserva e não cria vértice do tipo s que é origem de arco do tipo t; preserva vértices do tipo s e ainda deleta e cria um arco do tipo t; e não envolve vértices do tipo s e arcos do tipo t.

O caso em que a regra preserva ou cria vértices do tipo s e deleta arcos t não será tratado neste trabalho.

Para todos os casos de regras, observou-se que a tática utilizada para demonstração é idêntica. No caso em que a regra não deleta arcos, independente do tipo, a prova é realizada executando o provador NewPP with lasso. Para os casos em que arcos diferentes do tipo t são deletados, a demonstração é realizada aplicando as seguintes ações:

1. Instanciar o quantificador universal do objetivo;
2. Para cada arco e_i deletado pela regra, executar as seguintes ações:
   (a) Adicionar como hipótese $tLi\_E(e_i) = tG\_E(mE(e_i))$;
   (b) Executar o provador NewPP with lasso.
3. Executar NewPP with lasso.

Após executar os passos apresentados acima, uma árvore de prova é gerada e representada pela Figura 98.

Descrição da Árvore de Prova: *O objetivo inicial para se demonstrar é* $H \vdash \forall x \cdot x \in vertG \cup \{v_1, \ldots, v_l\} \wedge (tG\_V \cup \{v_1 \mapsto t_1, \ldots, v_l \mapsto t_l\})(x) = s \Rightarrow (\exists y \cdot y \in (edgeG \setminus \{ed_1, \ldots, ed_k\}) \cup \{ed_1, \ldots, ed_k\} \wedge (\{mE(e_1), \ldots, mE(e_j)\} \lhd tG\_E \cup \{ed_1 \mapsto t_1, \ldots, ed_k \mapsto$

$\mathcal{A}$ = $\{v_1, \ldots, v_l\}$
$\mathcal{B}$ = $\{v_1 \mapsto t_1, \ldots, v_l \mapsto t_l\}$
$\mathcal{C}$ = $\{mE(e_1), \ldots, mE(e_j)\}$
$\mathcal{D}$ = $\{ed_1, \ldots, ed_k\}$
$\mathcal{E}$ = $\{mE(e_1), \ldots, mE(e_j)\} \lhd sourceG \cup \{ed_1 \mapsto mV(v_1), \ldots, ed_k \mapsto v_l\}$
$\mathcal{F}$ = $\{mE(e_1), \ldots, mE(e_j)\} \lhd tG\_E \cup \{ed_1 \mapsto t_1, \ldots, ed_k \mapsto t_k\}$

1 $\forall$ goal (frees x); 2 type rewrites; 3 simplification rewrites; 4 $\Rightarrow$ goal; 5 ah ($tLi\_E(e_1) = tG\_E(mE(e_1))$); 6 $\wedge$ goal; 7 total function dom substitution in goal; 8 type rewrites; 9 $\top$ goal; 10 functional goal; 11 total function dom substitution in goal; 12 type rewrites; 13 $\top$ goal; 14 functional goal; 15 functional image goal for $mE(e_1)$; 16 total function dom substitution in goal; 17 hyp; 18 functional goal; 19 sl/ds; 20 NewPP with lasso; 21 ah ($tLi\_E(e_j) = tG\_E(mE(e_j))$); 22 generalized MP; 23 simplification rewrites; 24 $\wedge$ goal; 25 total function dom substitution in goal; 26 type rewrites; 27 $\top$ goal; 28 total function dom substitution in goal; 29 type rewrites; 30 $\top$ goal; 31 functional image goal for $mE(e_j)$; 32 functional image goal for $mE(e_j)$; 33 total function dom substitution in goal; 34 hyp; 35 sl/ds; 36 NewPP with lasso; 37 sl/ds; 38 NewPP with lasso

Figura 98: Árvore de Prova rule_i/propAllSVertSrcTEd/INV

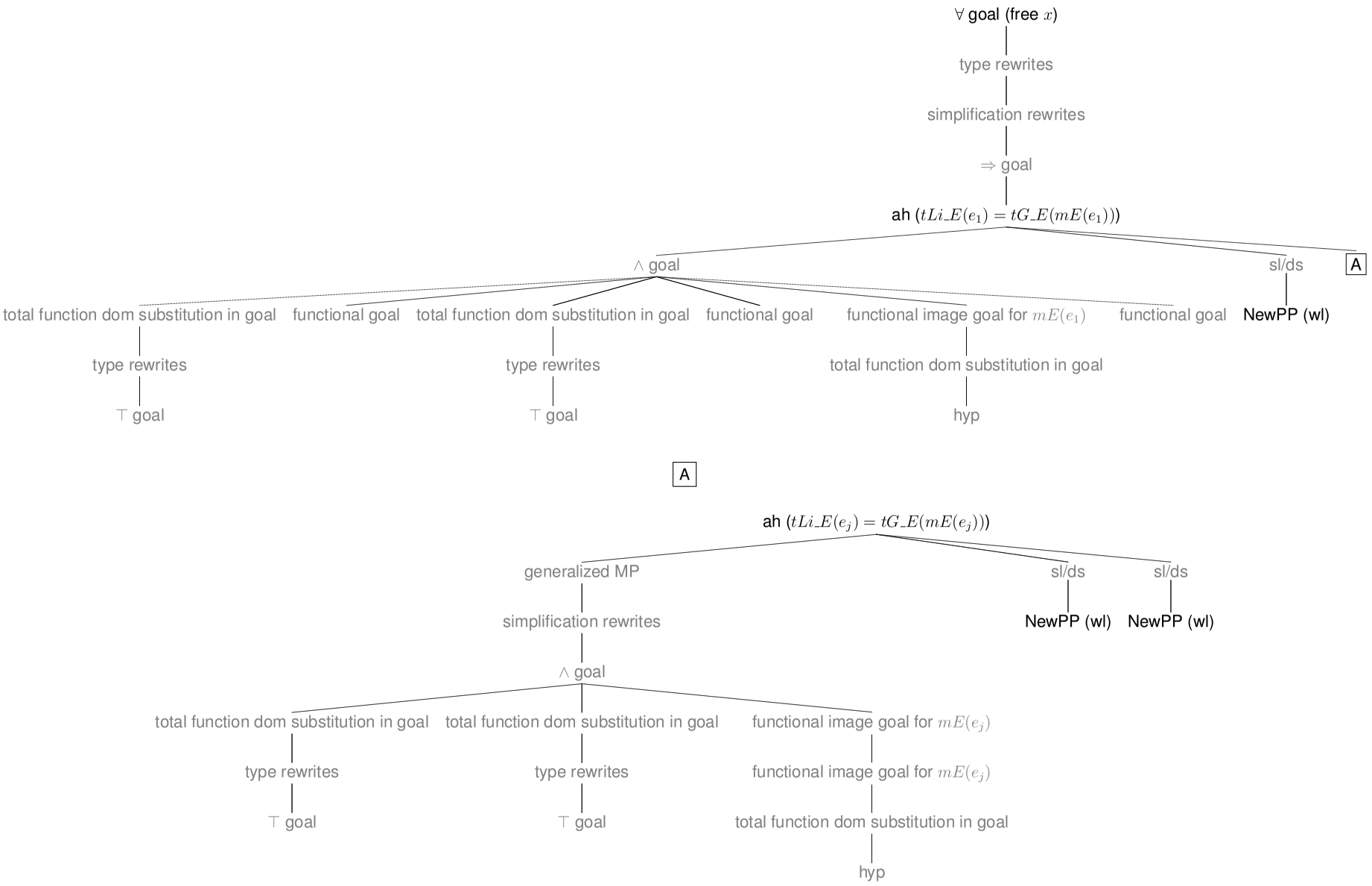


Figura 99: Árvore de Uso rule_i/propAllSVertSrcTEd/INV

$t_k\})(y) = t \wedge (\{mE(e_1), \ldots, mE(e_j)\} \lhd sourceG \cup \{ed_1 \mapsto mV(v_1), \ldots, ed_k \mapsto v_l\})(y) = x)$. *Para demonstrar esse sequente instancia-se o quantificador universal do objetivo, em seguida são aplicadas algumas regras de forma automática (regras 2-4), resultando no sequente* $H \vdash \exists y \cdot y \in (edgeG \setminus \{mE(e_1), \ldots, mE(e_j)\}) \cup \{ed_1, \ldots, ed_k\} \wedge (\{mE(e_1), \ldots, mE(e_j)\} \lhd tG\_E \cup \{ed_1 \mapsto t_1, \ldots, ed_k \mapsto t_k\})(y) = t \wedge (\{mE(e_1), \ldots, mE(e_j)\} \lhd sourceG \cup \{ed_1 \mapsto mV(v_1), \ldots, ed_k \mapsto v_l\})(y) = x$. *Nesse tipo de obrigação é necessário informar que os tipos dos arcos deletados são diferentes do tipo* t. *Para isso, considerando que o arco* e_i *é deletado pela regra, aplicam-se as ações: adicionar como hipótese* $tLi\_E(e_i) = tG\_E(mE(e_i))$, *em seguida executar o provador NewPP with lasso. Considerando os* j *arcos deletados pela regra, os passos necessários para demonstrar cada caso, é visualizado na árvore de prova através das regras (5-36). Após a execução dos passos, o último objetivo gerado,* $H \vdash \exists y \cdot y \in (edgeG \setminus \{mE(e_1), \ldots, mE(e_j)\}) \cup \{ed_1, \ldots, ed_k\} \wedge (\{mE(e_1), \ldots, mE(e_j)\} \lhd tG\_E \cup \{ed_1 \mapsto t_1, \ldots, ed_k \mapsto t_k\})(y) = t \wedge (\{mE(e_1), \ldots, mE(e_j)\} \lhd sourceG \cup \{ed_1 \mapsto mV(v_1), \ldots, ed_k \mapsto v_l\})(y) = x$ *é demonstrado aplicando o provador NewPP with lasso (38).*

A árvore de uso para essa demonstração é apresentada na Figura 99.

Descrição da Árvore de Uso: *Para utilizar essa tática, instancia-se o quantificador universal no objetivo. Em seguida, para cada arco deletado* e_i, *executam-se os passos: adiciona-se como hipótese* $tLi\_E(e_i) = tG\_E(mE(e_i))$ *e executa-se o provador NewPP with lasso. Depois de adicionar todas as hipóteses para os arcos deletados, a demonstração final é realizada por NewPP with lasso.*

4.10 Propriedade `propAllSVertTgtTEd`

A propriedade `propAllSVertTgtTEd` ($\forall x \cdot ((x \in vertG \wedge tG\_V(x) = s) \Rightarrow (\exists y \cdot (y \in edgeG \wedge tG\_E(y) = t \wedge targetG(y) = x))))$ assegura que em todos os estados do sistema, todo o vértice do tipo s é destino de um arco do tipo t. As alterações nas variáveis $vertG$, tG_V, $edgeG$, tG_E e $targetG$ acabam gerando uma obrigação de prova a ser demonstrada. Para descarregar a obrigação INITIALISATION/propAllSVertTgtTEd/INV, gerada para o evento de inicialização, aplicam-se as seguintes ações:

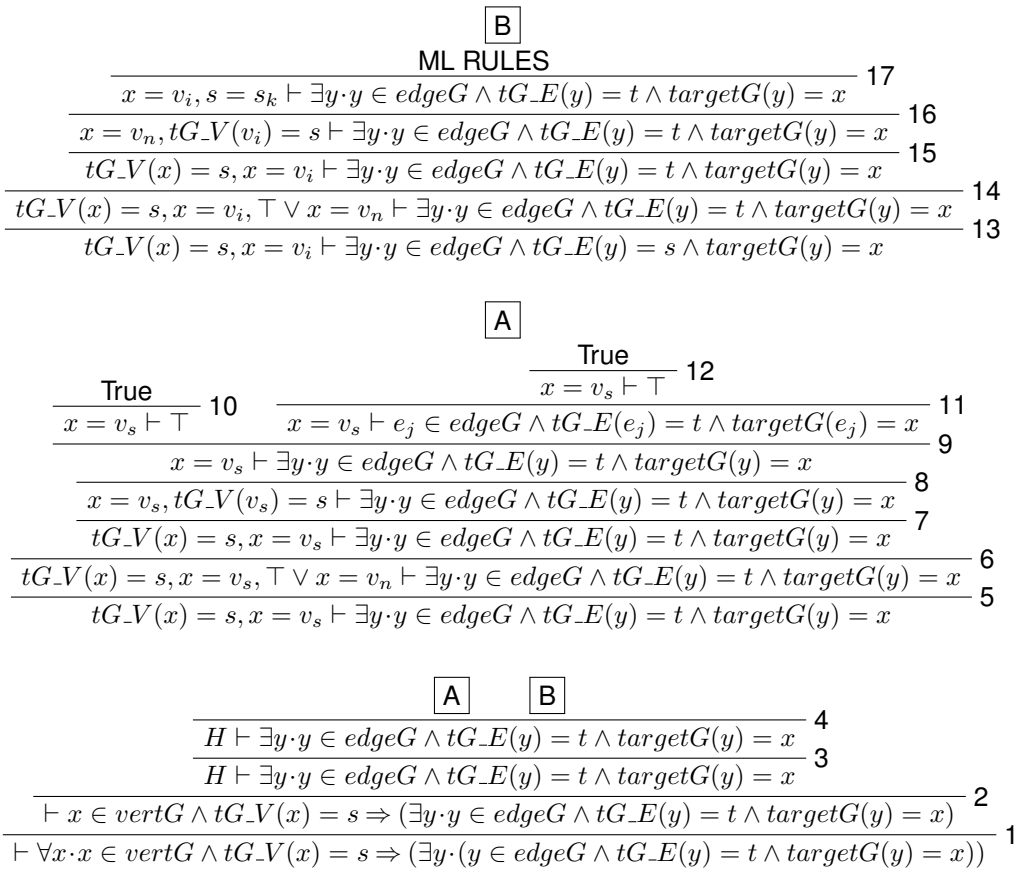
1. Instanciar o quantificador universal do objetivo;
2. Remover o operador $\in$ na hipótese $x \in vertG$;
3. Iniciar a prova por casos na hipótese $x = v_1 \vee \ldots \vee x = v_n$, considerando $vertG$ inicializada com n vértices;

4. Em cada caso, aplicar uma das ações até concluir a demonstração:

   (a) Para vértices v_s (tipo s) instanciar o arco e_j (tipo t) que possui destino em v_s;

   (b) Para vértices v_i (não do tipo t) executar o provador ML;

A Figura 100 apresenta a árvore de prova resultante da demonstração.



1 $\forall$ goal (free x); 2 $\Rightarrow$ goal; 3 remove $\in$ in $x \in \{v_1, \ldots, v_n\}$; 4 $\forall$ hyp ($x = v_1 \vee \ldots \vee x = v_n$);
5 generalized MP; 6 simplification rewrites; 7 eh with $x = v_s$; 8 simplification rewrites; 9 $\exists$ goal (inst e_j);
10 $\top$ goal; 11 simplification rewrites; 12 $\top$ goal; 13 generalized MP; 14 simplification rewrites;
15 eh with $x = v_i$; 16 simplification rewrites; 17 ML

Figura 100: Árvore de Prova INITIALISATION/propAllSVertTgtTEd/INV

Descrição da Árvore de Prova: *O objetivo inicial para se demonstrar é* $\vdash \forall x \cdot x \in vertG \wedge tG\_V(x) = s \Rightarrow (\exists y \cdot (y \in edgeG \wedge tG\_E(y) = t \wedge targetG(y) = x))$. *Para demonstrar esse sequente instancia-se o quantificador universal do objetivo (1), após é aplicada de forma automática a regra* $\Rightarrow$ *goal (2). Em seguida, remove-se o operador* $\in$ *na hipótese* $x \in vertG$ *(3) e se inicia a prova por casos através da hipótese* $x = v_1 \vee \ldots \vee x = v_n$ *(4). Para cada vértice do grafo inicial, primeiramente é aplicada algumas*

regras automáticas (ações 5-8 e 13-16), em seguida são gerados os sub-objetivos dos casos: (I) $tG\_V(x) = s, x = v_s \vdash \exists y \cdot y \in edgeG \wedge tG\_E(y) = t \wedge targetG(y) = x$*; (II)* $tG\_V(x) = s, x = v_i \vdash \exists y \cdot y \in edgeG \wedge tG\_E(y) = t \wedge targetG(y) = x$*. O sequente (I) representa o objetivo gerado para um vértice do tipo* s*, para demonstrar esse tipo de sequente instancia-se no objetivo o nome do arco* e_j *do tipo* t *que possui destino no vértice do tipo* s *após instanciação são gerados dois sub-objetivos: (i)* $x = v_s \vdash \top$ *e (ii)* $x = v_s \vdash e_j \in edgeG \wedge tG\_E(e_j) = t \wedge targetG(e_j) = x$*. (i) e (ii) são demonstrados automaticamente (10-12). Já o sequente (II) representa o objetivo gerado para um vértice diferente do tipo* s*, para demonstrar esse sequente executa-se o provador ML (17). Essas ações são necessárias em cada vértice do grafo inicial do sistema.*

Também é possível representar a tática proposta através da árvore de uso ilustrada na Figura 101.

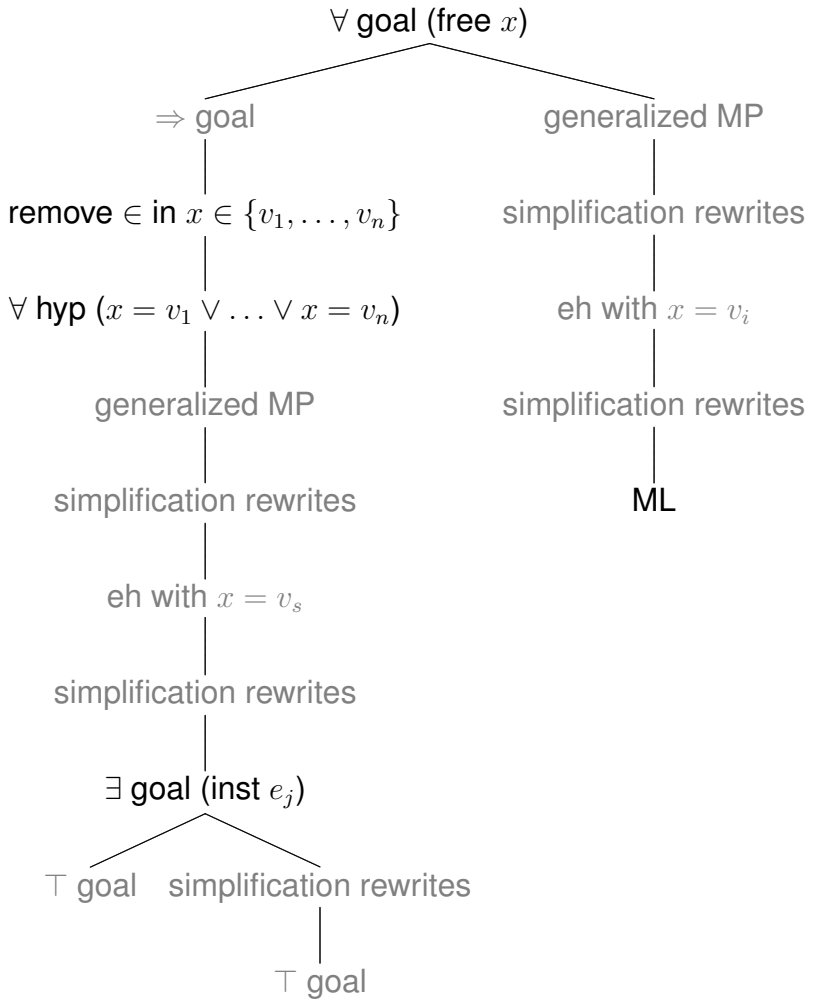


Figura 101: Árvore de Uso INITIALISATION/propAllSVertTgtTEd/INV

Descrição da Árvore de Uso: *Para utilizar essa tática, se elimina o quantificador universal do objetivo e após remove-se o operador* $\in$ *da hipótese* $x \in \{1, \ldots, n\}$*. Se*

inicia a prova por casos a partir da hipótese $x = v_1 \vee \ldots \vee x = v_n$*. Para cada caso: se o vértice solicitado nas hipóteses é do tipo* s *(*v_s*), então instancia-se no objetivo o nome do arco do tipo* t *que possui destino neste vértice. Caso contrário, executa-se o provador ML. Essas ações repetem-se para cada vértice do grafo inicial.*

Nessa propriedade as obrigações de prova para as regras são geradas no seguinte formato $\forall x \cdot x \in vertG \cup \{v_1, \ldots, v_l\} \wedge (tG\_V \cup \{v_1 \mapsto t_1, \ldots, v_l \mapsto t_l\})(x) = s \Rightarrow (\exists y \cdot y \in (edgeG \setminus \{mE(e_1), \ldots, mE(e_j)\}) \cup \{ed_1, \ldots, ed_k\} \wedge ((\{mE(e_1), \ldots, mE(e_j)\} \lhd tG\_E) \cup \{ed_1 \mapsto t_1, \ldots, ed_k \mapsto t_k\})(y) = t \wedge ((\{mE(e_1), \ldots, mE(e_j)\} \lhd targetG) \cup \{ed_1 \mapsto mV(v_1), \ldots, ed_k \mapsto v_l\})(y) = x)$, considerando que j arcos são deletados, k arcos e l vértices são criados pela aplicação da regra. Neste sentido, considera-se as seguintes ações de uma regra: não preserva e cria vértice do tipo s que é destino de arco do tipo t; preserva e não cria vértice do tipo s que é destino de arco do tipo t; preserva vértices do tipo s e ainda deleta e cria um arco do tipo t; e não envolve vértices do tipo s e arcos do tipo t.

O caso em que a regra preserva ou cria vértices do tipo s e deleta arcos t não é abordado neste trabalho.

Para todos os casos de regras, observou-se que a tática utilizada para demonstração é idêntica. No caso em que a regra não deleta arcos, independente do tipo, a prova é realizada executando o provador NewPP with lasso. Para os casos em que arcos diferentes do tipo t são deletados, a demonstração é realizada aplicando as seguintes ações:

1. Instanciar o quantificador universal do objetivo;
2. Para cada arco e_i deletado pela regra, executar as seguintes ações:
   (a) Adicionar como hipótese $tLi\_E(e_i) = tG\_E(mE(e_i))$;
   (b) Executar o provador NewPP with lasso.
3. Executar NewPP with lasso.

Após executar os passos apresentados acima, uma árvore de prova é gerada e representada pela Figura 102.

Descrição da Árvore de Prova: *O objetivo inicial para se demonstrar é* $H \vdash \forall x \cdot x \in vertG \cup \{v_1, \ldots, v_l\} \wedge (tG\_V \cup \{v_1 \mapsto t_1, \ldots, v_l \mapsto t_l\})(x) = s \Rightarrow (\exists y \cdot y \in (edgeG \setminus \{ed_1, \ldots, ed_k\}) \cup \{ed_1, \ldots, ed_k\} \wedge (\{mE(e_1), \ldots, mE(e_j)\} \lhd tG\_E \cup \{ed_1 \mapsto t_1, \ldots, ed_k \mapsto t_k\})(y) = t \wedge (\{mE(e_1), \ldots, mE(e_j)\} \lhd targetG \cup \{ed_1 \mapsto mV(v_1), \ldots, ed_k \mapsto v_l\})(y) = x)$*. Para demonstrar esse sequente instancia-se o quantificador universal do objetivo, em*

$\mathcal{A} = \{v_1, \ldots, v_l\}$
$\mathcal{B} = \{v_1 \mapsto t_1, \ldots, v_l \mapsto t_l\}$
$\mathcal{C} = \{mE(e_1), \ldots, mE(e_j)\}$
$\mathcal{D} = \{ed_1, \ldots, ed_k\}$
$\mathcal{E} = \{mE(e_1), \ldots, mE(e_j)\} \lhd targetG \cup \{ed_1 \mapsto mV(v_1), \ldots, ed_k \mapsto v_l\}$
$\mathcal{F} = \{mE(e_1), \ldots, mE(e_j)\} \lhd tG\_E \cup \{ed_1 \mapsto t_1, \ldots, ed_k \mapsto t_k\}$

1 $\forall$ goal (frees x); 2 type rewrites; 3 simplification rewrites; 4 $\Rightarrow$ goal; 5 ah ($tLi\_E(e_1) = tG\_E(mE(e_1))$); 6 $\wedge$ goal; 7 total function dom substitution in goal; 8 type rewrites; 9 $\top$ goal; 10 functional goal; 11 total function dom substitution in goal; 12 type rewrites; 13 $\top$ goal; 14 functional goal; 15 functional image goal for $mE(e_1)$; 16 total function dom substitution in goal; 17 hyp; 18 functional goal; 19 sl/ds; 20 NewPP with lasso; 21 ah ($tLi\_E(e_j) = tG\_E(mE(e_j))$); 22 generalized MP; 23 simplification rewrites; 24 $\wedge$ goal; 25 total function dom substitution in goal; 26 type rewrites; 27 $\top$ goal; 28 total function dom substitution in goal; 29 type rewrites; 30 $\top$ goal; 31 functional image goal for $mE(e_j)$; 32 functional image goal for $mE(e_j)$; 33 total function dom substitution in goal; 34 hyp; 35 sl/ds; 36 NewPP with lasso; 37 sl/ds; 38 NewPP with lasso

Figura 102: Árvore de Prova rule_i/propAllSVertTgtTEd/INV

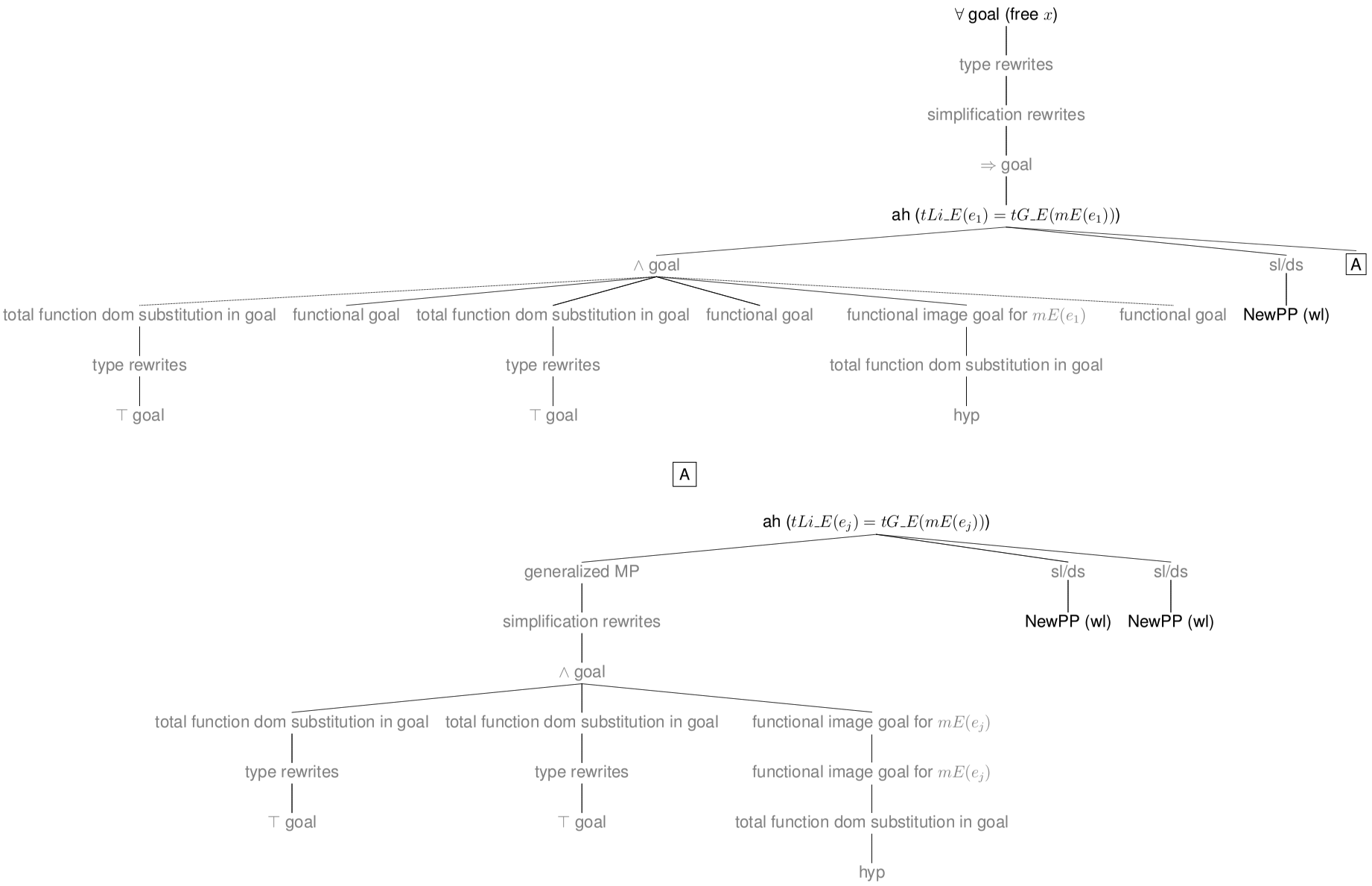


Figura 103: Árvore de Uso rule_i/propAllSVertTgtTEd/INV

seguida são aplicadas algumas regras de forma automática (regras 2-4), resultando no sequente $H \vdash \exists y \cdot y \in (edgeG \setminus \{mE(e_1), \ldots, mE(e_j)\}) \cup \{ed_1, \ldots, ed_k\} \wedge (\{mE(e_1), \ldots, mE(e_j)\} \lhd tG\_E \cup \{ed_1 \mapsto t_1, \ldots, ed_k \mapsto t_k\})(y) = t \wedge (\{mE(e_1), \ldots, mE(e_j)\} \lhd targetG \cup \{ed_1 \mapsto mV(v_1), \ldots, ed_k \mapsto v_l\})(y) = x$*. Nesse tipo de obrigação é necessário informar que os tipos dos arcos deletados são diferentes do tipo* t*. Para isso, considerando que o arco* e_i *é deletado pela regra, aplicam-se as ações: adicionar como hipótese* $tLi\_E(e_i) = tG\_E(mE(e_i))$*, em seguida executar o provador NewPP with lasso. Considerando os* j *arcos deletados pela regra, os passos necessários para demonstrar cada caso, é visualizado na árvore de prova através das regras (5-36). Após a execução dos passos, o último objetivo gerado,* $H \vdash \exists y \cdot y \in (edgeG \setminus \{mE(e_1), \ldots, mE(e_j)\}) \cup \{ed_1, \ldots, ed_k\} \wedge (\{mE(e_1), \ldots, mE(e_j)\} \lhd tG\_E \cup \{ed_1 \mapsto t_1, \ldots, ed_k \mapsto t_k\})(y) = t \wedge (\{mE(e_1), \ldots, mE(e_j)\} \lhd targetG \cup \{ed_1 \mapsto mV(v_1), \ldots, ed_k \mapsto v_l\})(y) = x$ *é demonstrado aplicando o provador NewPP with lasso (38).*

A árvore de uso para essa demonstração é apresentada na Figura 103.

Descrição da Árvore de Uso: *Para utilizar essa tática, instancia-se o quantificador universal no objetivo. Em seguida, para cada arco deletado* e_i*, executam-se os passos: adiciona-se como hipótese* $tLi\_E(e_i) = tG\_E(mE(e_i))$ *e executa-se o provador NewPP with lasso. Depois de adicionar todas as hipóteses para os arcos deletados, a demonstração final é realizada por NewPP with lasso.*

5 APLICANDO AS TÁTICAS PROPOSTAS

Neste capítulo serão apresentados exemplos de aplicação das táticas propostas. Inicialmente são consideradas propriedades para o exemplo Token Ring, descrito no Capítulo 2. Na sequência é apresentado um exemplo de especificação em GG, a descrição de um Sistema Móvel, e então serão detalhadas as propriedades verificadas. Ainda neste capítulo, será mostrado a aplicação concreta de uma tática de prova. Importante destacar que todas as táticas propostas foram testadas utilizando estes dois exemplos. Para os casos em que não se tinha nenhuma regra nos exemplos que se enquadrasse na tática considerada, modificou-se a descrição de algumas regras, apenas com o objetivo de validar a tática. O texto que segue restringe-se a descrever exemplos de propriedades interessantes de serem verificadas nos exemplos considerados.

5.1 Propriedades Verificadas para o Protocolo Token Ring

Para o exemplo do protocolo Token Ring, detalhados na Seção 2.1, as seguintes propriedades foram verificadas utilizando as táticas propostas:

`propExEdgeStb:` $\exists x \cdot x \in tG\_E \rhd \{Stb\}$

Existe um arco do tipo standby. Essa propriedade indica que sempre existe uma estação em espera no anel.

`propExEdgeTok:` $\exists x \cdot x \in tG\_E \rhd \{Tok\}$

Existe um arco do tipo Token. Essa propriedade indica que sempre existe o Token em alguma estação no anel.

`propLoopTok:` $\exists x \cdot x \in edgeG \land sourceG(x) = targetG(x) \land tG\_E(x) = Tok$

Existe um arco loop do tipo Token. Essa propriedade adiciona a propriedade anterior a informação de que o arco Token é do tipo loop.

`propFinTok:` $finite(tG\_E \rhd \{Tok\})$

O conjunto de arcos do tipo Token é finito. Essa propriedade é estabelecida para auxiliar a prova da propriedade sobre cardinalidade.

propCardTok: $card(tG\_E \rhd \{Tok\}) = 1$
Existe exatamente um arco do tipo Token. Essa propriedade estabelece que somente uma estação está com o Token, indicando que é possível ter apenas uma estação ativa no anel em qualquer instante de tempo.

propExVertNode: $\exists x \cdot x \in tG\_V \rhd \{Node\}$
Existe um vértice do tipo Node. Essa propriedade especifica que sempre existe pelo menos uma estação no anel.

propEdSpSrcNode: $\exists x, y \cdot (x \in edgeG \wedge y \in vertG \wedge tG\_V(y) = Node \wedge sourceG(x) = y)$
Existe um arco com origem em Node. Propriedade utilizada para testar a tática correspondente.

propEdSpTgtNode: $\exists x, y \cdot (x \in edgeG \wedge y \in vertG \wedge tG\_V(y) = Node \wedge targetG(x) = y)$
Existe um arco com destino em Node. Propriedade utilizada para testar a tática correspondente.

propNotIsoVNode: $\forall x \cdot ((x \in vertG \wedge tG\_V(x) = Node) \Rightarrow (\exists y \cdot (y \in edgeG \wedge ((sourceG(y) = x \wedge \neg targetG(y) = x) \vee (targetG(y) = x \wedge \neg sourceG(y) = x)))))$
Não existe um nodo isolado do tipo Node.

propAllNodeVertSrcNextEd: $\forall x \cdot ((x \in vertG \wedge tG\_V(x) = Node) \Rightarrow (\exists y \cdot (y \in edgeG \wedge tG\_E(y) = Next \wedge sourceG(y) = x)))$
Todo vértice do tipo Node é origem de um arco do tipo Next.

propAllNodeVertTgtNextEd: $\forall x \cdot ((x \in vertG \wedge tG\_V(x) = Node) \Rightarrow (\exists y \cdot (y \in edgeG \wedge tG\_E(y) = Next \wedge targetG(y) = x)))$
Todo vértice do tipo Node é destino de um arco do tipo Next.

Essas 3 últimas propriedades indicam que estações estão sempre conectadas a outras estações.

5.2 Sistema Móvel

Nesse trabalho se utilizou GG para modelar um sistema simples de telefonia móvel, ilustrado na Figura 104.

Este sistema consiste em um conjunto de antenas e usuários de telefonia móvel, onde os usuários podem conectar-se com antenas e realizar chamadas. Cada antena tem uma capacidade máxima para conexões de usuários, esgotada essa capacidade, o usuário não consegue conectar-se a antena correspondente. O grafo tipo T descreve dois tipo de vértices Ant (antenas) e Usr (usuário) e cinco tipos de arcos Acn (conexão entre antenas), Ucn (conexão entre usuários e antenas), Cal (chamada entre usuários), Cn (disponibilidade para um usuário conectar-se a uma antena) e Main

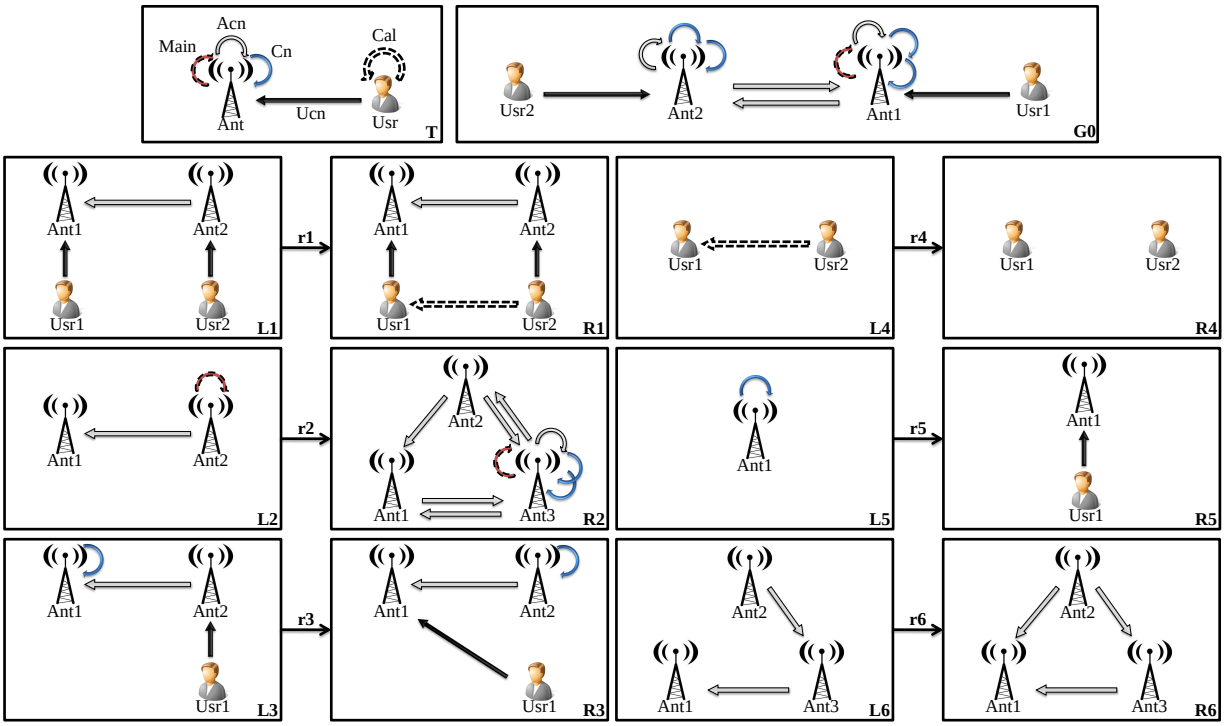


Figura 104: GG *Sistema Móvel*

(antena principal). O grafo inicial $G0$ especifica um sistema com duas antenas e dois usuários. Cada antena possui duas conexões disponíveis para novos usuários, indicada pelos arcos do tipo Cn. O comportamento do sistema é descrito pelo conjunto de regras $r1$ a $r6$. Usuários conectados a antenas podem iniciar (regra $r1$) ou terminar (regra $r4$) uma chamada. Novas antenas e novos usuários podem ser acrescentados no sistema em qualquer instante de tempo (regra $r2$ e $r5$, respectivamente). Um usuário pode trocar de antena (regra $r3$) e novas conexões entre as antenas podem ser estabelecidas (regra $r6$).

5.3 Propriedades Verificadas para o Sistema Móvel

As seguintes propriedades foram verificadas utilizando as táticas propostas para o Sistema Móvel:

`propExEdgeAcn:` $\exists x \cdot x \in tG\_E \rhd \{Acn\}$

Existe um arco do tipo Acn. Essa propriedade indica que sempre é possível estabelecer uma chamada na rede.

`propLoopMain:` $\exists x \cdot x \in edgeG \wedge sourceG(x) = targetG(x) \wedge tG\_E(x) = Main$

Existe um arco loop do tipo Main. Essa propriedade estabelece que o sistema sempre tem uma antena principal.

`propFinMain:` $finite(tG\_E \rhd \{Main\})$

O conjunto de arcos do tipo Main é finito. Essa propriedade é estabelecida para auxiliar a prova da propriedade sobre cardinalidade.

propCardMain: $card(tG\_E \rhd \{Main\}) = 1$

Existe exatamente um arco do tipo Main. Essa propriedade estabelece que somente uma antena é a principal.

propExVertAnt: $\exists x \cdot x \in tG\_V \rhd \{Ant\}$

Existe um vértice do tipo Antena. Essa propriedade especifica que sempre existe pelo menos uma antena no sistema móvel.

propEdSpSrcAnt: $\exists x, y \cdot (x \in edgeG \land y \in vertG \land tG\_V(y) = Ant \land sourceG(x) = y)$

Existe um arco com origem em Antena. Propriedade utilizada para testar a tática correspondente.

propEdSpTgtAnt: $\exists x, y \cdot (x \in edgeG \land y \in vertG \land tG\_V(y) = Ant \land targetG(x) = y)$

Existe um arco com destino em Ant. Propriedade utilizada para testar a tática correspondente.

propNotIsoVUsr: $\forall x \cdot ((x \in vertG \land tG\_V(x) = Usr) \Rightarrow (\exists y \cdot (y \in edgeG \land ((sourceG(y) = x \land \neg targetG(y) = x) \lor (targetG(y) = x \land \neg sourceG(y) = x)))))$

Não existe um nodo isolado do tipo Usuário. Essa propriedade estabelece que não existem usuários desconectados.

propAllUsrVertSrcUcnEd: $\forall x \cdot ((x \in vertG \land tG\_V(x) = Usr) \Rightarrow (\exists y \cdot (y \in edgeG \land tG\_E(y) = Ucn \land sourceG(y) = x)))$

Todo vértice do tipo Usuário é origem de um arco do tipo Ucn. Essa propriedade indica que usuários estão sempre conectados a antenas.

propAllAntVertTgtAcnEd: $\forall x \cdot ((x \in vertG \land tG\_V(x) = Ant) \Rightarrow (\exists y \cdot (y \in edgeG \land tG\_E(y) = Acn \land targetG(y) = x)))$

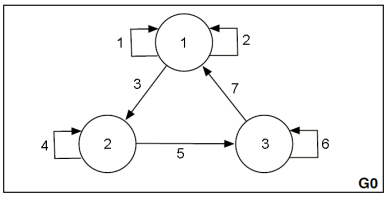
Todo vértice do tipo Antena é destino de um arco do tipo Acn. Essa propriedade especifica que antenas sempre habilitam chamadas.

5.4 Utilizando as Táticas Propostas

Nesta seção serão apresentados exemplos de aplicação das táticas propostas para demonstrar a obrigação de prova gerada para uma propriedade específica. O primeiro exemplo é relativo ao evento de inicialização, já o outro é para o evento que representa a aplicação de uma regra. Na tática proposta os nomes de vértices, arcos e ações realizadas por uma regra, na obrigação de prova gerada, são tratados de forma genérica. Já nos exemplos apresentados estes elementos aparecem de acordo com sua especificação (Token Ring ou Sistema Móvel).

5.4.1 Evento de Inicialização

No sistema *token ring* (Figura 7 do Capítulo 2) é possível especificar a propriedade `propCardTok` ($card(tG\_E \rhd \{Tok\}) = 1$). Essa propriedade assegura que o conjunto de arcos do tipo Tok é exatamente 1. Considerando que no grafo inicial do sistema $G0$, os nomes dos vértices e arcos são atualizados para os números naturais (Figura 105), então uma obrigação de prova é gerada para garantir a preservação da propriedade após a inicialização de tG_E. A obrigação de prova gerada segue o seguinte formato $card(\{1 \mapsto Tok, 2 \mapsto Stb, 3 \mapsto Nxt, 4 \mapsto Stb, 5 \mapsto Nxt, 6 \mapsto Stb, 7 \mapsto Nxt\} \rhd \{Tok\}) = 1$.



$$vertG := \{1, 2, 3\}$$
$$edgeG := \{1, 2, 3, 4, 5, 6, 7\}$$
$$sourceG := \{1 \mapsto 1, 2 \mapsto 1, 3 \mapsto 1, 4 \mapsto 2, 5 \mapsto 2, 6 \mapsto 3, 7 \mapsto 3\}$$
$$targetG := \{1 \mapsto 1, 2 \mapsto 1, 3 \mapsto 2, 4 \mapsto 2, 5 \mapsto 3, 6 \mapsto 3, 7 \mapsto 1\}$$
$$tG\_V := \{1 \mapsto Node, 2 \mapsto Node, 3 \mapsto Node\}$$
$$tG\_E := \{1 \mapsto Tok, 2 \mapsto Stb, 3 \mapsto Nxt, 4 \mapsto Stb, 5 \mapsto Nxt, 6 \mapsto Stb, 7 \mapsto Nxt\}$$

Figura 105: Grafo Inicial $G0$ com Números Naturais

Para demonstrar essa obrigação de prova aplicam-se as seguintes ações dentro do controle de prova da ferramenta:

1. Adicionar como hipótese $\{1 \mapsto Tok, 2 \mapsto Stb, 3 \mapsto Nxt, 4 \mapsto Stb, 5 \mapsto Nxt, 6 \mapsto Stb, 7 \mapsto Nxt\} \rhd \{Tok\} = \{1 \mapsto Tok\}$;
2. Nos próximos sub-objetivos executar NewPP with lasso.

Após demonstração, na ferramenta é gerada uma árvore de prova, conforme Figura 106. Observe que a árvore gerada corresponde a árvore da Figura 32, apresentada na Seção 4.2.

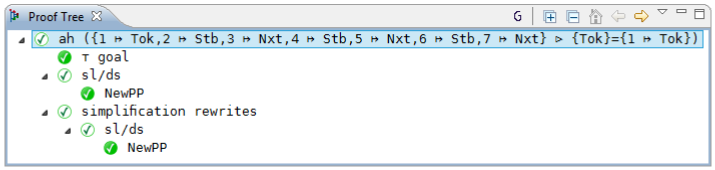


Figura 106: Árvore de Prova INITIALISATION/propCardTok/INV

5.4.2 Evento Aplicação de Regra

Para assegurar que no sistema móvel existe um arco do tipo Acn, utiliza-se a propriedade `propExEdgeAcn` ($\exists x \cdot x \in tG\_E \rhd \{Acn\}$). Na especificação do sistema móvel, todos os elementos recebem nomes distintos. Na regra $r6$, por exemplo os nomes de cada vértice e arco são especificados conforme ilustrado na Figura 107. Essa regra altera os valores das variáveis, assim sua aplicação resulta na seguinte obrigação de prova $\exists x \cdot x \in (tG\_E \cup \{newEacn \mapsto Acn\}) \rhd \{Acn\}$.

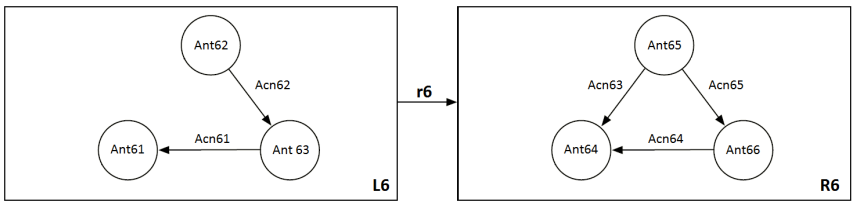


Figura 107: Regra $r6$ *Mobile System*

A fim de demonstrar esse tipo de obrigação de prova, executam-se os seguintes passos:

1. Instanciar no objetivo o par $newEacn \mapsto Acn$, onde $newEacn$ é o arco criado pela regra do tipo Acn;
2. Executar o provador ML.

Após demonstração, na ferramenta é gerada uma árvore de prova conforme Figura 108. Observe que a árvore gerada corresponde a árvore da Figura 40, apresentada na Seção 4.3.

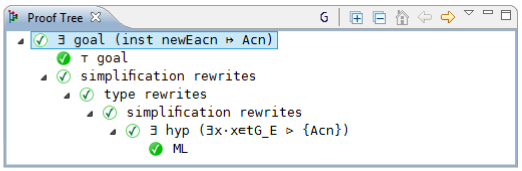


Figura 108: Árvore de Prova rule6/propExEdgeAcn/INV

6 CONCLUSÃO

A presente dissertação utiliza uma abordagem (COSTA, 2010) que permite aplicar a técnica de prova de teoremas para gramática de grafos. Em tal proposta, foi realizada uma tradução de GG para Event-B (DEPLOY, 2011), sendo possível utilizar os provadores disponíveis para essa linguagem, através da plataforma Rodin (ABRIAL et al., 2010), para a verificação formal de propriedades. Após uma especificação em Event-B, a plataforma Rodin realiza uma verificação estática e uma dinâmica, gerando obrigações de prova, as quais visam garantir que a especificação do sistema está bem-definida e não possui inconsistências. Algumas obrigações são demonstradas de forma automática pela ferramenta e outras necessitam de interação. Como o processo de prova é semiautomático, o desenvolvedor precisa estar familiarizado com o sistema, com a linguagem de especificação e com a ferramenta, além de possuir o conhecimento lógico-matemático para auxiliar na conclusão das provas. Essa interação é considerada desvantagem nessa abordagem, bem como na técnica de prova de teoremas em geral, e acaba restringindo a adoção desse método.

Nesse sentido aparece este trabalho que apresenta táticas de prova, que auxiliam o usuário nas demonstrações das obrigações de prova geradas na especificação de uma GG em Event-B. Primeiramente são apresentadas as definições básicas de uma GG e como é realizada sua especificação em Event-B (DEPLOY, 2011). Em seguida apresenta-se o ambiente de prova da ferramenta Rodin (ABRIAL et al., 2010), juntamente com seus provadores e regras de inferência utilizados nas demonstrações. Inicialmente, identificou-se quais são as obrigações de prova que sempre são geradas na descrição de uma GG em Event-B e indicou-se os provadores que as descarregam (LEMOS JUNIOR; COSTA CAVALHEIRO; FOSS, 2013). Após, tomando como base alguns padrões de propriedades propostos anteriormente (COSTA CAVALHEIRO; FOSS; RIBEIRO, 2012), realizou-se uma análise nas obrigações de prova geradas pela especificação de algumas propriedades específicas. E a partir dessa análise apresenta-se táticas de prova para demonstrar as obrigações geradas por essas propriedades. A apresentação das táticas é feita descrevendo-se os passos necessários para as demonstrações das propriedades consideradas, descrevendo as

árvores de prova e propondo árvores de uso. Em qualquer uma das formas, é possível realizar a prova da obrigação correspondente. E foram descritas ainda, as regras de prova utilizadas nas demonstrações.

Como trabalhos futuros pretende-se ampliar o conjunto de táticas propostas, visando englobar mais casos de aplicações. Também espera-se identificar semelhanças nas demonstrações das propriedades já propostas, com o intuito de gerar padrões de prova para conjuntos de sub-objetivos. E ampliar o número de provadores considerados nas táticas já propostas, como os provadores da família Satisfiability Modulo Theories (SMT-Solvers), mais precisamente o CVC3 (BARRETT; TINELLI, 2007) e veriT (BOUTON et al., 2009).

REFERÊNCIAS

ABRIAL, J.-R. **The B-book**: Assigning Programs to Meanings. New York, NY, USA: Cambridge University Press, 1996.

ABRIAL, J.-R. **Modeling in Event-B**: System and Software Engineering. 1st.ed. New York, NY, USA: Cambridge University Press, 2010.

ABRIAL, J.-R.; BUTLER, M.; HALLERSTEDE, S.; HOANG, T. S.; MEHTA, F.; VOISIN, L. Rodin: An Open Toolset for Modelling and Reasoning in Event-B. **International Journal on Software Tools for Technology Transfer (STTT)**, [S.l.], April 2010.

BARRETT, C.; TINELLI, C. CVC3. In: INTERNATIONAL CONFERENCE ON COMPUTER AIDED VERIFICATION (CAV '07), 19., 2007. **Proceedings...** Springer-Verlag, 2007. p.298–302. (Lecture Notes in Computer Science, v.4590). Berlin, Germany.

BOUTON, T.; CAMINHA, D.; OLIVEIRA, B.; DÉHARBE, D.; FONTAINE, P. veriT: an open, trustable and efficient SMT-solver. In: CONFERENCE ON AUTOMATED DEDUCTION (CADE), VOLUME 5663 OF LECTURE NOTES IN COMPUTER SCIENCE, 2009. **Proceedings...** Springer, 2009. p.151–156.

CLEARSY. **Atelier B**. http://www.atelierb.eu/ (last accessed Jan 2014). Atelier B is an industrial tool that allows for the operational use of the B Method to develop defect-free proven software (formal software).

COSTA CAVALHEIRO, S. A. da; FOSS, L.; RIBEIRO, L. Specification patterns for properties over reachable states of graph grammars. In: BRAZILIAN CONFERENCE ON FORMAL METHODS: FOUNDATIONS AND APPLICATIONS, 15., 2012, Berlin. **Proceedings...** Springer, 2012. p.83–98. (SBMF'12).

COSTA, S. A. da. **Relational Approach of Graph Grammars**. 2010. Tese (Doutorado em Ciência da Computação) — INF, UFRGS, Brazil.

COSTA, S. A. da; RIBEIRO, L. Verification of graph grammars using a logical approach. **Science of Computer Programming**, [S.l.], v.In Press, Corrected Proof, p.–, 2010.

DAVIS, M.; LOGEMANN, G.; LOVELAND, D. A Machine Program for Theorem-proving. **Commun. ACM**, New York, NY, USA, v.5, n.7, p.394–397, July 1962.

DAVIS, M.; PUTNAM, H. A Computing Procedure for Quantification Theory. **J. ACM**, New York, NY, USA, v.7, n.3, p.201–215, July 1960.

DEPLOY. **Event-B and the Rodin Platform**. http://www.event-b.org/ (last accessed Jan 2014). Rodin Development is supported by European Union ICT Projects DEPLOY (2008 to 2012) and RODIN (2004 to 2007).

ECLIPSE. **Eclipse Projects**. http://www.eclipse.org/ (last accessed Jan 2014). The Eclipse Foundation was created in January 2004 as an independent not-for-profit corporation to act as the steward of the Eclipse community.

EHRIG, H.; HECKEL, R.; KORFF, M.; LöWE, M.; RIBEIRO, L.; WAGNER, A.; CORRADINI, A. Algebraic approaches to graph transformation. Part II: single pushout approach and comparison with double pushout approach. **Handbook of graph grammars and computing by graph transformation: volume I. foundations**, River Edge, NJ, USA, p.247–312, 1997.

LEMOS JUNIOR, L.; COSTA CAVALHEIRO, S. André da; FOSS, L. Theorem Proving Graph Grammars: Strategies for Discharging Proof Obligations. In: IYODA, J.; MOURA, L. (Ed.). **Formal Methods**: Foundations and Applications. [S.l.]: Springer Berlin Heidelberg, 2013. p.147–162. (Lecture Notes in Computer Science, v.8195).

MELLO, A. Moura de; LEMOS JUNIOR, L.; FOSS, L.; COSTA CAVALHEIRO, S. da. Graph Grammars: A Comparison between Verification Methods. In: THEORETICAL COMPUTER SCIENCE (WEIT), 2011 WORKSHOP-SCHOOL ON, 2011. **Anais...** [S.l.: s.n.], 2011. p.88–94.

NIPKOW, T.; PAULSON, L. C.; WENZEL, M. **Isabelle/HOL — A Proof Assistant for Higher-Order Logic**. [S.l.]: Springer, 2002. (LNCS, v.2283).

RIBEIRO, L. Métodos formais de especificação: Gramáticas de grafos. In: VIII ESCOLA DE INFORMÁTICA DA SBC-SUL, 2000, Porto Alegre. **Anais...** Editora da UFRGS, 2000. p.1–33.

RIBEIRO, L.; DOTTI, F. L.; COSTA, S. A. da; DILLENBURG, F. C. Towards Theorem Proving Graph Grammars using Event-B. **ECEASST**, [S.l.], v.30, 2010.

ROBINSON, J. A.; VORONKOV, A. (Ed.). **Handbook of Automated Reasoning (in 2 volumes)**. [S.l.]: Elsevier and MIT Press, 2001.

STRECKER, M. Modeling and Verifying Graph Transformations in Proof Assistants. **Electron. Notes Theor. Comput. Sci.**, Amsterdam, The Netherlands, The Netherlands, v.203, n.1, p.135–148, Mar. 2008.

STRECKER, M. Locality in Reasoning about Graph Transformations. In: SCHÜRR, A.; VARRÓ, D.; VARRÓ, G. (Ed.). **Applications of Graph Transformations with Industrial Relevance**. [S.l.]: Springer Berlin Heidelberg, 2012. p.169–181. (Lecture Notes in Computer Science, v.7233).

TRAN, H.; PERCEBOIS, C. Towards a Rule-Level Verification Framework for Property-Preserving Graph Transformations. In: SOFTWARE TESTING, VERIFICATION AND VALIDATION (ICST), 2012 IEEE FIFTH INTERNATIONAL CONFERENCE ON, 2012. **Anais...** [S.l.: s.n.], 2012. p.946–953.

YAMAMOTO, M.; TANABE, Y.; TAKAHASHI, K.; HAGIYA, M. Abstraction of Graph Transformation Systems by Temporal Logic and Its Verification. In: MEYER, B.; WOODCOCK, J. (Ed.). **Verified Software**: Theories, Tools, Experiments. [S.l.]: Springer Berlin Heidelberg, 2008. p.518–527. (Lecture Notes in Computer Science, v.4171).

ANEXO A REGRAS DE PROVA UTILIZADAS

∃ **goal:** Regra de inferência manual

Nome: EXISTS_INST

Regra: $$\frac{H \vdash WD(E)\ H \vdash P(E)}{H \vdash \exists x \cdot P(x)}$$

Descrição: Instancia elementos no objetivo do sequente.

∃ **hyp:** Regra de inferência automática

Nome: XST_L

Regra: $$\frac{H,\ P(x) \vdash Q}{H,\ \exists x \cdot P(x) \vdash Q}$$

Descrição: Simplifica qualquer sequente contendo hipóteses existencialmente quantificadas, liberando suas variáveis ligadas.

∀ **goal:** Regra de inferência manual

Nome: Forall istantiation - FORALL_INST

Regra: $$\frac{H \ \vdash \ WD(E) \qquad H, [x := E]P \ \vdash \ G}{H, \forall x \cdot P \ \vdash \ G}$$

Descrição: Instancia elementos no objetivo do sequente.

$\forall$ **hyp:** Regra de inferência automática

Nome: ALL_R

Regra: $$\frac{H \vdash P(x)}{H \vdash \forall x \cdot P(x)}$$

Descrição: Simplifica qualquer sequente contendo hipóteses universalmente quantificadas, liberando todas as variáveis ligadas.

$\Rightarrow$ **goal:** Regra de inferência automática

Nome: IMP_R

Regra: $$\frac{H, P \ \vdash \ Q}{H \ \vdash \ P \Rightarrow Q}$$

Descrição: Simplifica qualquer sequente com um objetivo implicativo adicionando o lado direito da implicação como uma hipótese, no novo objetivo.

$\Rightarrow$ **hyp mp:** Regra de inferência manual

Nome: Forall Modus Ponens - FORALL_INST_MP

Regra: $$\frac{H \vdash WD(E) \quad H, WD(E) \vdash [x := E]P \quad H, WD(E), [x := E]Q \vdash G}{H, \forall x \cdot P \Rightarrow Q \ \vdash \ G}$$

Descrição: x é instanciado com E e a regra Modus Ponens é aplicada.

$\top$ **goal:** Regra de inferência automática

Nome: TRUE_GOAL

Regra: $\overline{H \vdash \top}$

Descrição: Descarrega qualquer sequente cujo objetivo seja $\top$ (verdade lógica).

$\wedge$ **goal:** Regra de inferência automática

Nome: Conjunctive Goal - AND_R

Regra: $$\frac{H \vdash P \ H \vdash Q}{H \vdash P \wedge Q}$$

Descrição: Divide um sequente com objetivo conjuntivo em vários sub-objetivos.

eh: Regra de inferência automática

Nome: Use Equality Hypothesis from Left to Right - EQL_LR

Regra: $$\frac{H(E) \vdash P(E)}{H(x), x = E \vdash P(x)}$$

Descrição: Simplifica um sequente reescrevendo todas as hipóteses selecionadas e o objetivo usando a hipótese que é uma igualdade entre uma variável livre e uma expressão que não contenha a variável livre. Nesse caso a variável é livre no lado esquerdo da expressão.

functional goal: Regra de inferência automática

Nome: FUN_GOAL

Regra: $$\frac{}{H,\, f \in E \; op \; F \vdash f \in T_1 \nrightarrow T_2}$$

Descrição: Tenta descarregar um sequente cujo objetivo estabelece que uma expressão é uma função.

Generalized MP: Regra de inferência automática

Nome: Generalized Modus Ponens - prefixadas com GENMP

Regra: Visualização disponível em DEPLOY (2011)

Descrição: Encontrando ocorrências de uma hipótese em outras hipóteses pode simplificar o objetivo e substituí-lo por $\top$ ou $\bot$.

he: Regra de inferência automática

Nome: Use Equality Hypothesis from Right to Left - EQL_RL

Regra: $$\frac{H(E) \vdash P(E)}{H(x), E = x \vdash P(x)}$$

Descrição: Simplifica um sequente reescrevendo todas as hipóteses selecionadas e o objetivo usando a hipótese que é uma igualdade entre uma variável livre e uma expressão que não contenha a variável livre. Nesse caso a variável é livre no lado direito da expressão.

hyp: Regra de inferência automática

Nome: Goal in Hypotheses - HYP

Regra: $\overline{H, P \vdash P}$

Descrição: Descarrega qualquer sequente cujo objetivo está contido em suas hipóteses.

Partition rewrites in hyp: Regra de reescrita automática e manual

Nome: Partition Rewriter - DEF_PARTITION

Regra: $\mathrm{partition}(s, s_1, s_2, \ldots, s_n) \mathrel{\widehat{=}} \begin{array}{ll} & s = s_1 \cup s_2 \cup \cdots \cup s_n \\ \wedge & s_1 \cap s_2 = \varnothing \\ & \vdots \\ \wedge & s_1 \cap s_n = \varnothing \\ & \vdots \\ \wedge & s_{n-1} \cap s_n = \varnothing \end{array}$

Descrição: Simplifica todos os predicados da forma $partition(S, \ldots)$ para o formato expandido no objetivo e todas as hipóteses se tornam visíveis.

Range Distribution Left Rewrites in Goal: Regra de reescrita manual

Nome: DISTRI_RANRES_BUNION_L

Regra: $(s \cup t) \rhd r \mathrel{\widehat{=}} (s \rhd r) \cup (t \rhd r)$

Descrição: Aplica a restrição da imagem à esquerda nos elementos de uma união.

remove $\in$: Regra de reescrita manual

Nome: Remove Membership - DEF_IN_SETENUM

Regra: $E \in \{A, \ldots, B\} \mathrel{\widehat{=}} E = A \vee \ldots \vee E = B$

Descrição: Remove o operador $\in$ da expressão $E \in \{A, \ldots, B\}$.

remove ¬: Regra de reescrita automática

Nome: Remove Negation - DISTRI_NOT_AND

Regra: $\neg\,(P \wedge Q) \mathrel{\widehat{=}} \neg\,P \vee \neg\,Q$

Descrição: Disponível em DEPLOY (2011)

Simplification Rewriter: Regra de reescrita automática

Nome: Prefixadas com SIM

Regra: $$\frac{H, H' \vdash P'}{H \vdash P}$$

Descrição: Tenta simplificar todos os predicados de um sequente usando regras de reescrita de simplificação.

Type Rewrites: Regra de reescrita automática

Nome: SIMP_TYPE_EQUAL_EMPTY e SIMP_TYPE_IN

Regra: $Ty = \varnothing \mathrel{\widehat{=}} \bot$ e $t \in Ty \mathrel{\widehat{=}} \top$

Descrição: Simplifica os predicados que contenham expressões de tipo.