Towards theorem proving graph grammars

Leila Ribeiro

joint work with Simone Costa, Fernando Dotti, Fabiane Dillenburg, Alfio Martini

Universidade Federal do Rio Grande do Sul (UFRGS), Brazil

IFIP WG .3 - July 2010 - Etelsen

Motivation

* **Theorem proving** is a **powerful technique** for the analysis of computational systems
* It is possible to prove properties of reachable states for **infinite state systems**
* This approach is **complementary** to other existing analysis techniques for graph grammars, like model checking and analysis based on approximations

Personal motivation: I kept trying to convince colleagues that using formal methods is nice, they allow to prove many relevant properties, hopefully using some tool, but I myself never used theorem provers before...

But...

- Theorem provers are typically hard to use because
  - ➡ the system under analysis must be faithfully encoded in a logical framework (in a rather low level way)
  - ➡ many existing tools are hard to install and use
  - ➡ proving properties needs a lot of user assistance
- Graphs (with types, attributes, ...) are complex structures, and the properties we wish to prove easily become cryptic logical formulae

The idea...

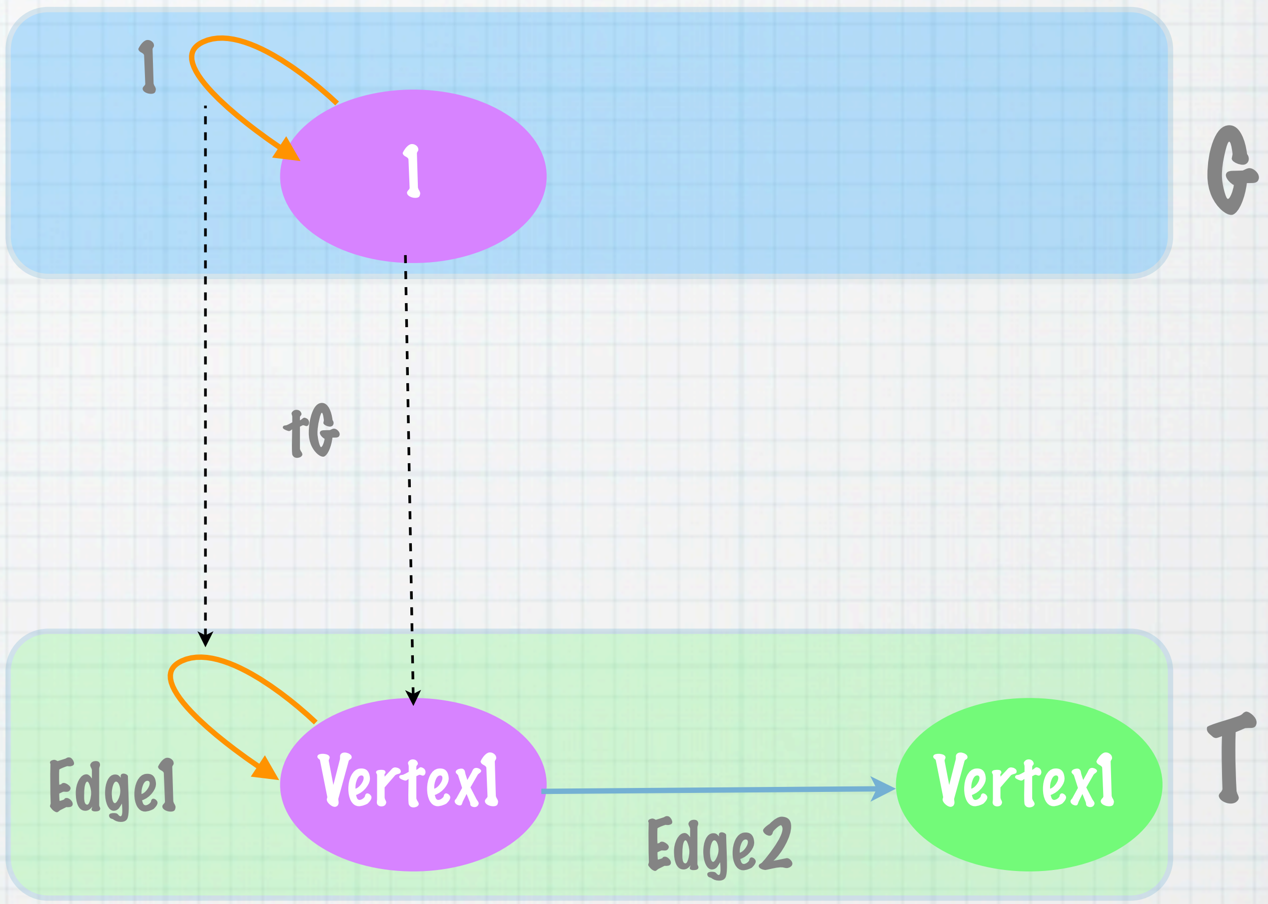
* Encode GGs in such a way they become closer to the input languages of theorem provers

* Use exiting theorem provers to prove properties of graph grammars

Typed Graphs

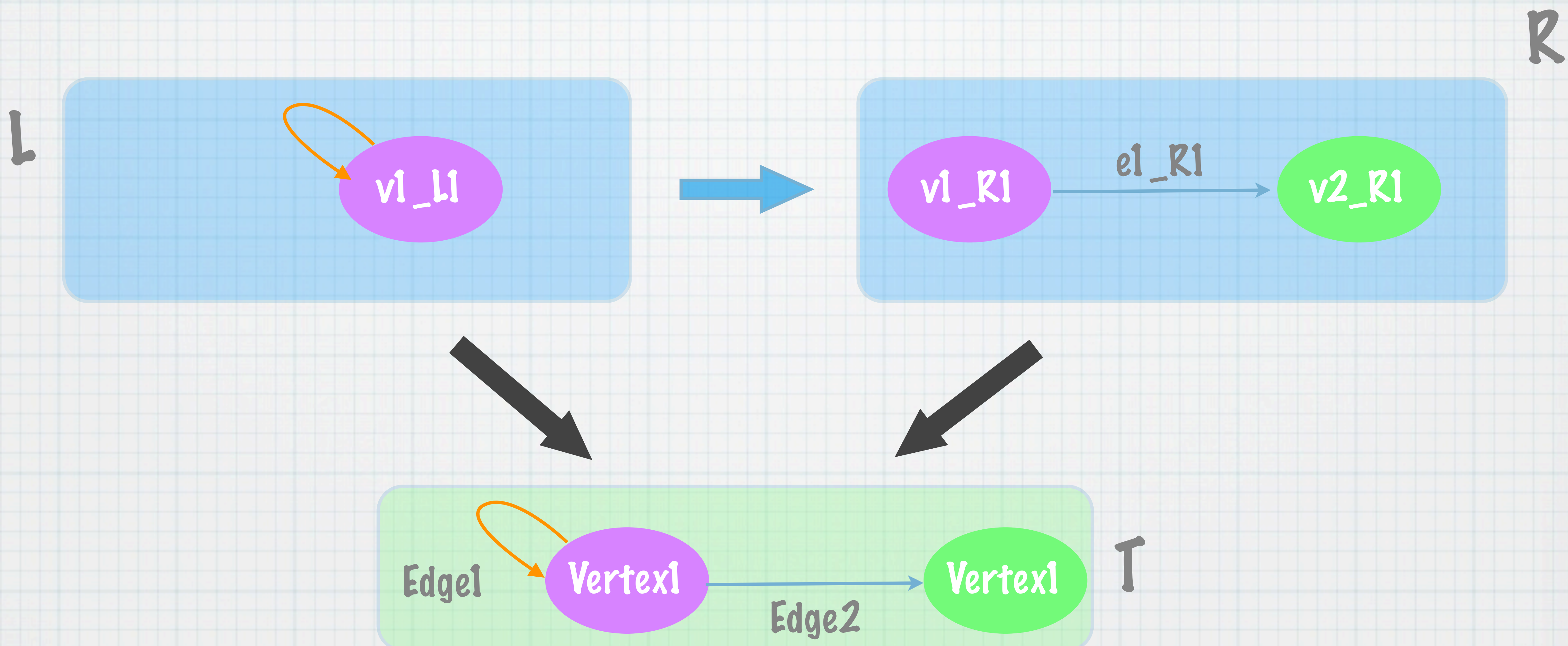
Typed Graph: tG

Type Graph: T



Category of typed graphs and partial graph morphisms : GraP(T)

(SPO) Rule



: partial injective graph homomorphism, no vertex deletion

SPO Graph Grammar

* Type Graph T
* Initial (start) graph G typed over T
* Set of rules typed over T

Semantics based on rule applications (POs in GraP(T))

Relational Graph Grammars

* Definition of graph grammars based on relations and restrictions on these relations
* Faithful encoding of SPO, considering injective rules that do not delete vertices and matches that are injective on edges
* Based on Courcelle's relational structures: domain+relations, transduction

Relational Graph

V_{GG} : *set of vertex ids*
E_{GG} : *set of edge ids*

Graph T

$$vertT \subseteq V_{GG}$$
$$edgeT \subseteq E_{GG}$$
$$sourceT \in edgeT \to vertT$$
$$targetT \in edgeT \to vertT$$

tG

$$tG\_V \in vertG \to vertT$$
$$tG\_E \in edgeG \to edgeT$$

source/target compat.

Graph G

$$vertG \subseteq V_{GG}$$
$$edgeG \subseteq E_{GG}$$
$$sourceG \in edgeG \to vertG$$
$$targetG \in edgeG \to vertG$$

Relational Rule

$\alpha : L \longrightarrow R$

$$\alpha_V : vertL \rightarrowtail vertR$$
$$\alpha_E : edgeL \rightarrowtail\!\!\!\!\!+ \ edgeR$$

type compat.

Match is analogous, but total .

Rule Application

$$
\begin{array}{ccc}
L & \xrightarrow{\alpha} & R \\
{\scriptstyle m}\downarrow & (PO) & \downarrow \\
G & \longrightarrow & H
\end{array}
$$

$$vertH = vertG \uplus (vertR - \alpha_V(vertL))$$
$$edgeH = (edgeG - m_E(edgeL)) \uplus E_R$$

+ source, target, typing functions...

Now, implementing....

* First attempt: Use Isabelle

* Second attempt: Event-B (Rodin platform)

Event-B

* Formalism based on B
* Event-B models are composed by
  - **CONTEXT** : de…
    facts that may be used in the model
  - **MACHINE** : de…
    (transitions)

Definition of type graph and rules

Definition of start graph and rule application

Type Graph in Event-B

CONTEXT ctx_gg

SETS

`vertT` (Type Graph T) Verteces

`edgeT` (Type Graph T) Edges

CONSTANTS

`sourceT`

`targetT`

`Vertex1`

`Edge1`

`Vertex2`

`Edge2`

AXIOMS

`axm1` : $partition(vertT, \{Vertex1\}, \{Vertex2\})$

`axm2` : $partition(edgeT, \{Edge1\}, \{Edge2\})$

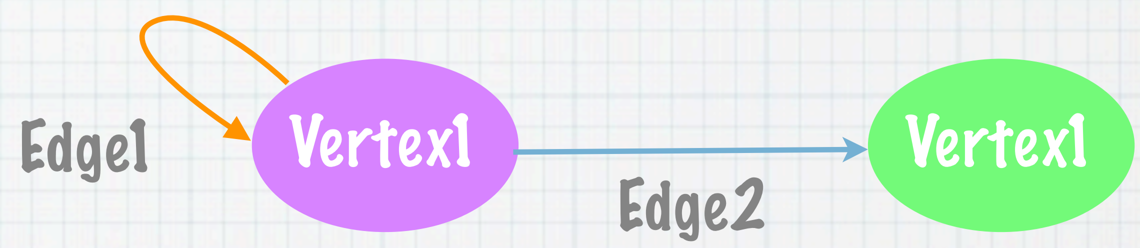
`axm_src` : $sourceT \in edgeT \rightarrow vertT$

`axm_srcD` : $partition(sourceT, \{Edge1 \mapsto Vertex1\}, \{Edge2 \mapsto Vertex1\})$

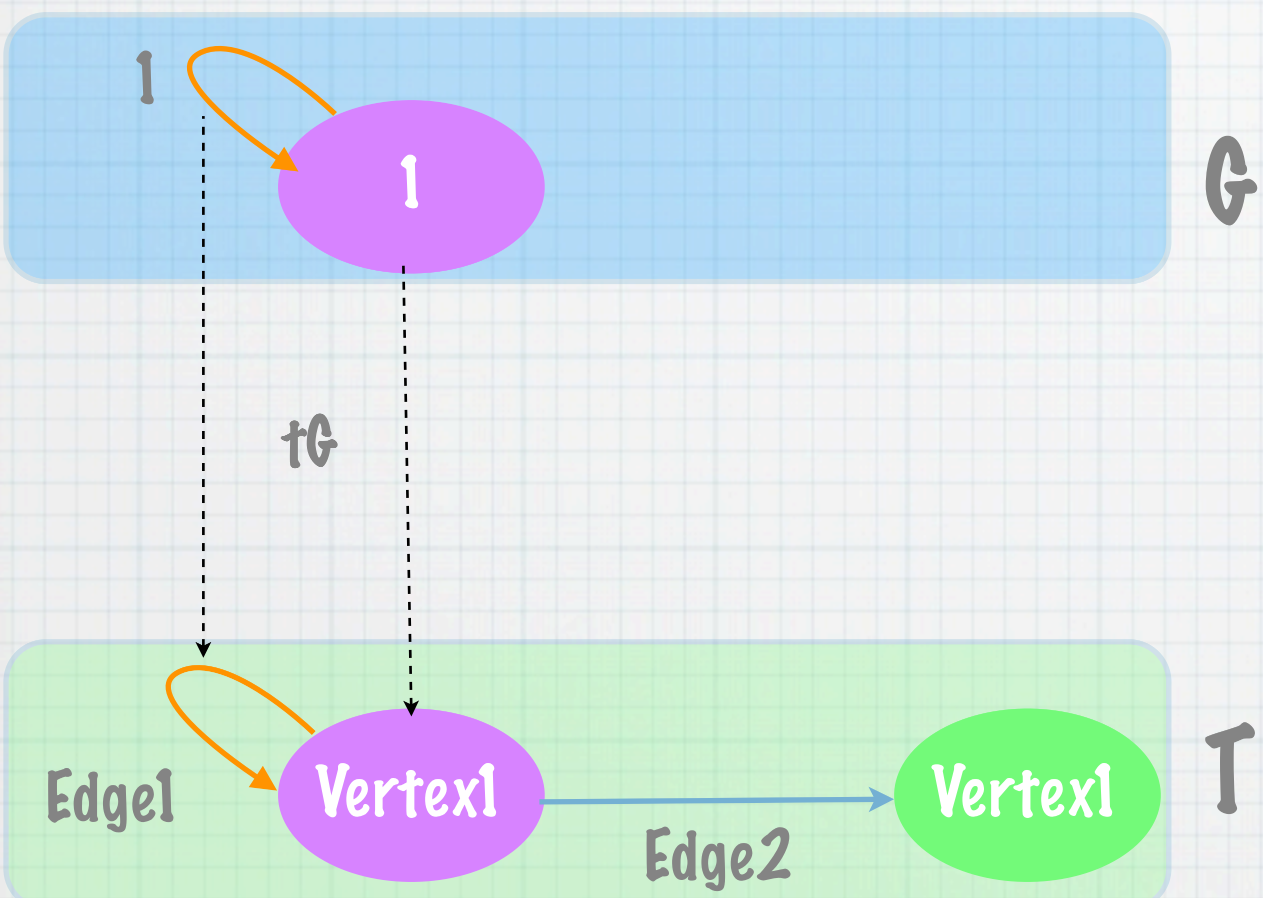
`axm_trg` : $targetT \in edgeT \rightarrow vertT$

`axm10` : $partition(targetT, \{Edge1 \mapsto Vertex1\}, \{Edge2 \mapsto Vertex2\})$

END

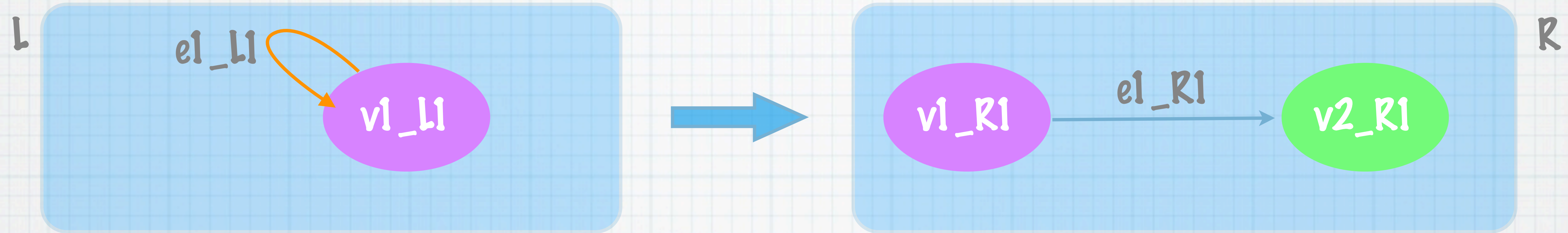


State Graph



MACHINE mch_gg
SEES ctx_gg
VARIABLES
  vertG
  edgeG
  sourceG
  targetG
  tG_V Typing vertices, tG_V
  tG_E Typing edges, tG_E
INVARIANTS
  type_vertG : $vertG \in \mathbb{P}(\mathbb{N})$
  type_edgeG : $edgeG \in \mathbb{P}(\mathbb{N})$
  type_sourceG : $sourceG \in edgeG \rightarrow vertG$
  type_targetG : $targetG \in edgeG \rightarrow vertG$
  type_tG_V : $tG\_V \in vertG \rightarrow vertT$
  type_tG_E : $tG\_E \in edgeG \rightarrow edgeT$
EVENTS
Initialisation
  begin
    act1 : $vertG := \{1\}$
    actE : $edgeG := \{1\}$
    acts : $sourceG := \{1 \mapsto 1\}$
    actt : $targetG := \{1 \mapsto 1\}$
    act3 : $tG\_V := \{1 \mapsto Vertex1\}$
    act4 : $tG\_E := \{1 \mapsto Edge1\}$
  end
END

Rule



CONTEXT ctx_gg

SETS

- vertL1
- edgeL1
- vertR1
- edgeR1

CONSTANTS

- v1_L1
- e1_L1
- v1_R1
- v2_R1
- e1_R1
- tL1_V (LHS1) Typing vertices, tL1_V
- tL1_E (LHS1) Typing edges, tL1_E
- tR1_V (RHS1) Typing vertices, tR1_V
- tR1_E (RHS1) Typing edges, tR1_E
- alpha1V (Rule 1) Rule alpha1: mapping vertices
- alpha1E (Rule 1) Rule alpha1: mapping edges

AXIOMS

L

- axm15 : $partition(vertL1, \{v1\_L1\})$
- axm16 : $partition(edgeL1, \{e1\_L1\})$
- axm_srcL1 : $sourceL1 \in edgeL1 \rightarrow vertL1$
- axm_srcDL1 : $partition(sourceL1, \{e1\_L1 \mapsto v1\_L1\})$
- axm_tarL1 : $targetL1 \in edgeL1 \rightarrow vertL1$
- axmtarDL1 : $partition(targetL1, \{e1\_L1 \mapsto v1\_L1\})$
- axmtL1V : $tL1\_V \in vertL1 \rightarrow vertT$
- act9 : $partition(tL1\_V, \{v1\_L1 \mapsto Vertex1\})$
- axmtL1E : $tL1\_E \in edgeL1 \rightarrow edgeT$
- act10 : $partition(tL1\_E, \{e1\_L1 \mapsto Edge1\})$

R

- axm17 : $partition(vertR1, \{v1\_R1\}, \{v2\_R1\})$
- axm18 : $partition(edgeR1, \{e1\_R1\})$
- axm_srcR1 : $sourceR1 \in edgeR1 \rightarrow vertR1$
- axm_srcDR1 : $partition(sourceR1, \{e1\_R1 \mapsto v1\_R1\})$
- axm_tarR1 : $targetR1 \in edgeR1 \rightarrow vertR1$
- axmrafDR1 : $partition(targetR1, \{e1\_R1 \mapsto v2\_R1\})$
- axmtR1 : $tR1\_V \in vertR1 \rightarrow vertT$
- act13 : $partition(tR1\_V, \{v1\_R1 \mapsto Vertex1\}, \{v2\_R1 \mapsto Vertex2\})$
- axmtR1E : $tR1\_E \in edgeR1 \rightarrow edgeT$
- act14 : $partition(tR1\_E, \{e1\_R1 \mapsto Edge2\})$

- axmA1V : $alpha1V \in vertL1 ⇸ vertR1$
- act15 : $partition(alpha1V, \{v1\_L1 \mapsto v1\_R1\})$
- axmA1E : $alpha1E \in edgeL1 ⇸ edgeR1$
- act16 : $alpha1E = \varnothing$

Rule (Behavior)

EVENTS

Event $alpha1 \widehat{=}$

any

mV

mE

$newV$

$newE$

where

grd1 : $mV \in vertL1 \to vertG$

grd2 : $mE \in edgeL1 \rightarrowtail edgeG$

grd3 : $newV \in \mathbb{N} \setminus vertG$

grd4 : $newE \in \mathbb{N} \setminus edgeG$

grd5 : $\forall v \cdot v \in vertL1 \Rightarrow tL1\_V(v) = tG\_V(mV(v))$

grd6 : $\forall e \cdot e \in edgeL1 \Rightarrow tL1\_E(e) = tG\_E(mE(e))$

grd7 : $\forall e \cdot e \in edgeL1 \Rightarrow mV(sourceL1(e)) = sourceG(mE(e)) \wedge mV(targetL1(e)) = targetG(mE(e))$

m is a match

vertex type compatibility

edge type compatibility

source/target compatibility

then

act3 : $vertG := vertG \cup \{newV\}$

actE : $edgeG := (edgeG \setminus \{mE(e1\_L1)\}) \cup \{newE\}$

acts : $sourceG := (\{mE(e1\_L1)\} \lhd sourceG) \cup \{newE \mapsto (sourceG(mE(e1\_L1)))\}$

actt : $targetG := (\{mE(e1\_L1)\} \lhd targetG) \cup \{newE \mapsto newV\}$

acttV : $tG\_V := tG\_V \cup \{newV \mapsto Vertex2\}$

act6 : $tG\_E := (\{mE(e1\_L1)\} \lhd tG\_E) \cup \{newE \mapsto Edge2\}$

end

Properties

* Properties are stated as invariants:

INVARIANTS

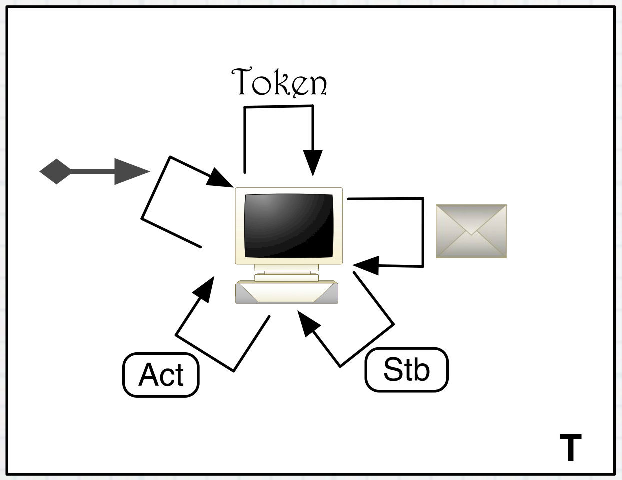
$$\ldots : \ldots$$

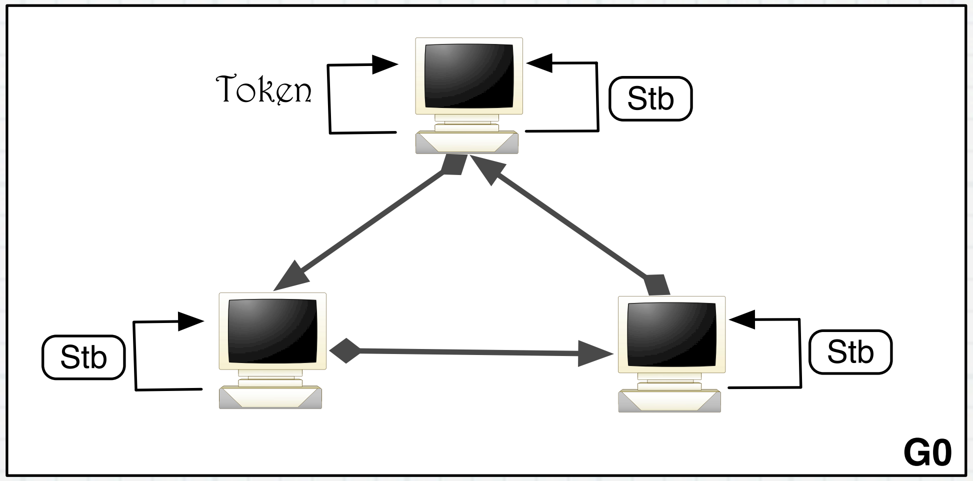
$$\texttt{prop1} : \ \mathit{finite}(\mathit{edgeG})$$

$$\texttt{prop2} : \ \mathit{card}(\mathit{edgeG}) \leq 2$$

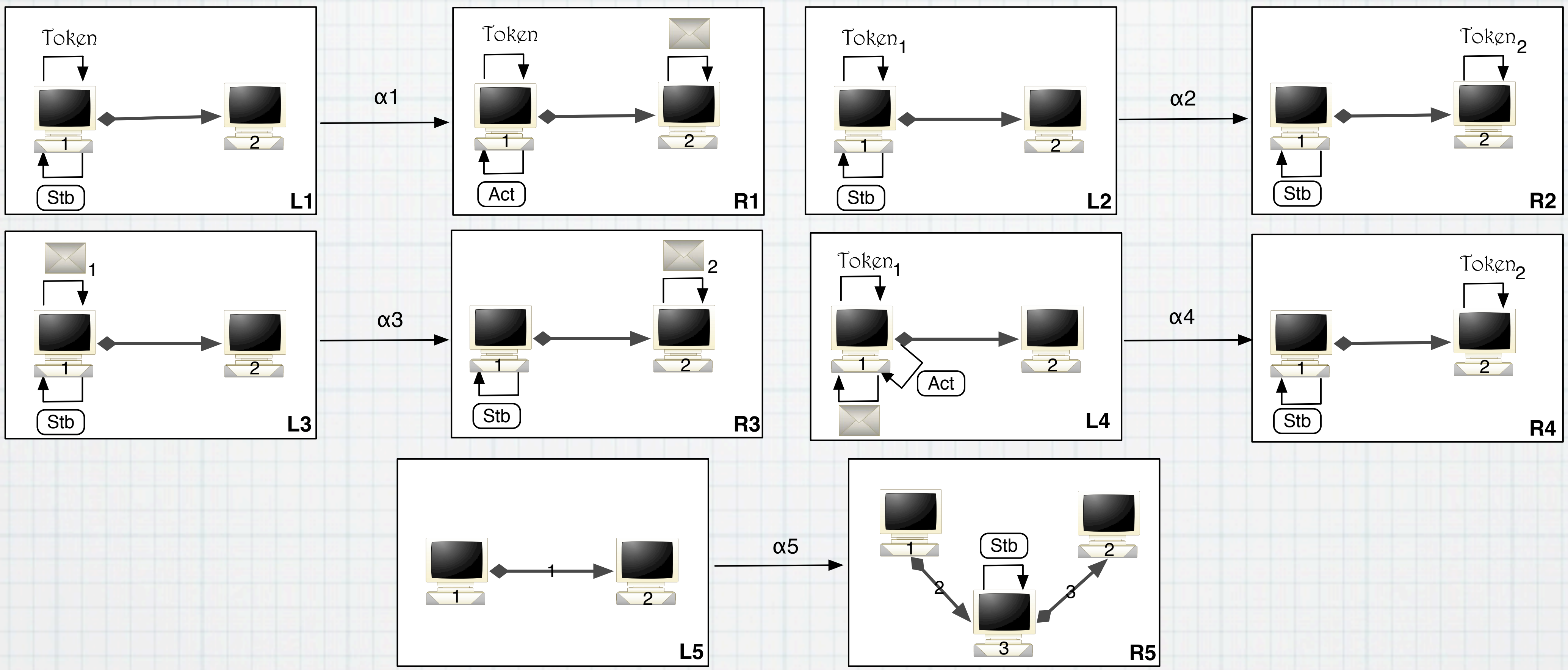
* In Event-B, proof obligations are generated to guarantee that all **invariants are well-defined**, are **valid in any initial state** and are **preserved by all events**

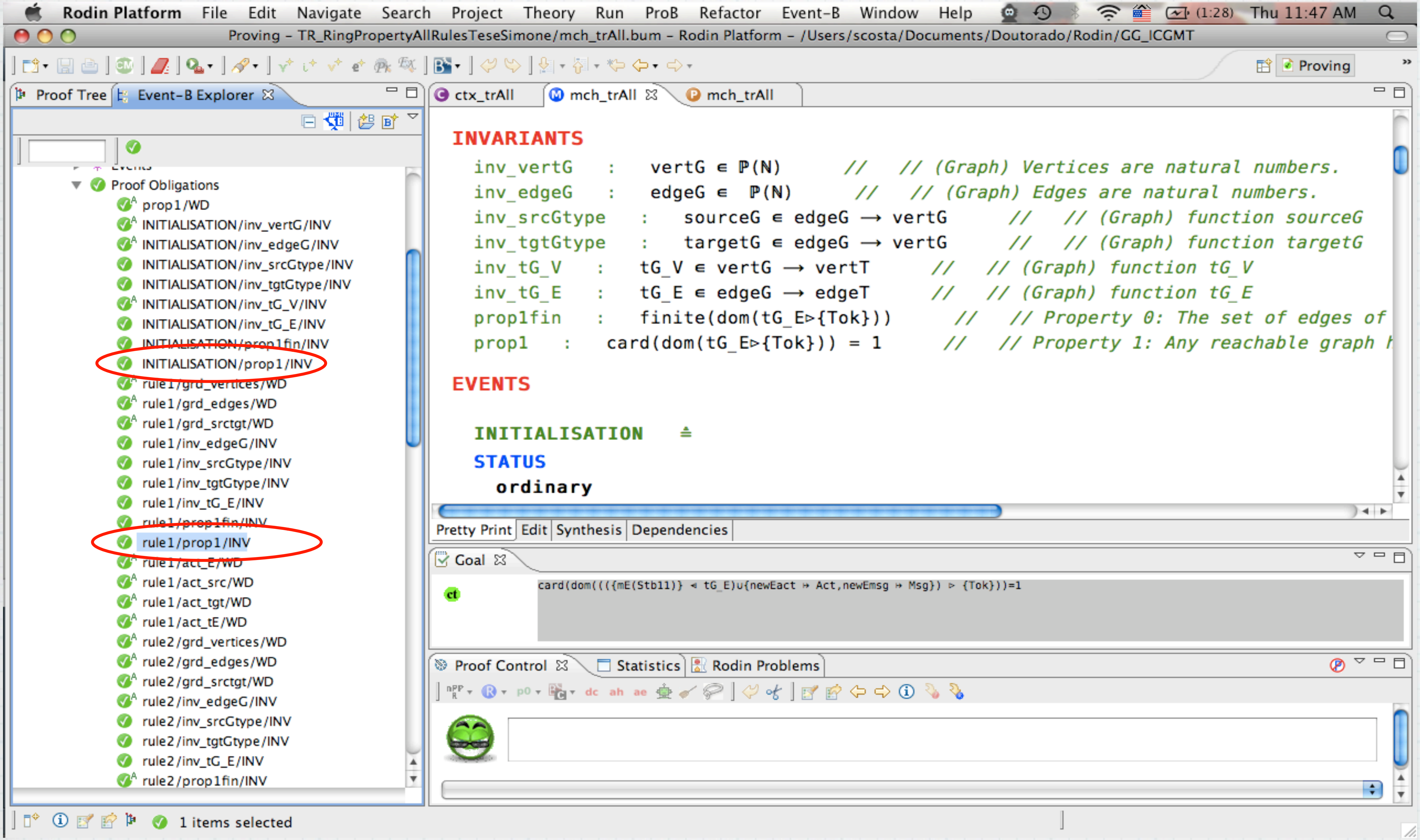
Another Example: Token Ring





Another Example: Token Ring





86 POs: 57 (A) + 29 (SA)

Final Remarks

- ☑ GGs can be faithfully encoded in Event-B
- ☑ Rodin offers a set of theorem provers that can be used to verify graph grammars
- ☐ However, we need
  - * libraries for commonly used operations;
  - * theories for graph structures;
  - * property patterns (tactics patters);
  - * integration of data types (attributed GTS), NACs, ...
  - * more "intelligent" theorem prover: Isabelle???

Thank you!!!