Strategic Port Graph Rewriting : An Interactive Modeling and Analysis Framework

Maribel Fernández

King's College London, UK

Hélène Kirchner

Inria, France

Bruno Pinaud

Bordeaux University, France

GRAPHITE 2014, April 4, 2014

Motivation and approach

In the context of software development and analysis, we address two challenges :

Motivation and approach

In the context of software development and analysis, we address two challenges :

- Provide a modeling framework for complex systems

$$\overbrace{\text{<State, Evolution Step, Control>}}^{\textbf{Complex Systems}}$$

Motivation and approach

In the context of software development and analysis, we address two challenges :

- Provide a modeling framework for complex systems

$$\overbrace{\text{<State, Evolution Step, Control>}}^{\textbf{Complex Systems}}$$

$$\underbrace{\text{<Graph, Set of Rules, Strategy>}}_{\textbf{Strategic Graph Program}}$$

Motivation and approach

In the context of software development and analysis, we address two challenges :

- Provide a modeling framework for complex systems

Complex Systems

<State, Evolution Step, Control>

<Graph, Set of Rules, Strategy>

Strategic Graph Program

- Preserve all computations and provide interactive visualisation tools to help analysis and debugging

Derivation tree analysis

Rewriting ingredients

- **In general, a rewriting process is:**
  - Non terminating,
  - Non confluent,
  - Highly concurrent

Rewriting ingredients

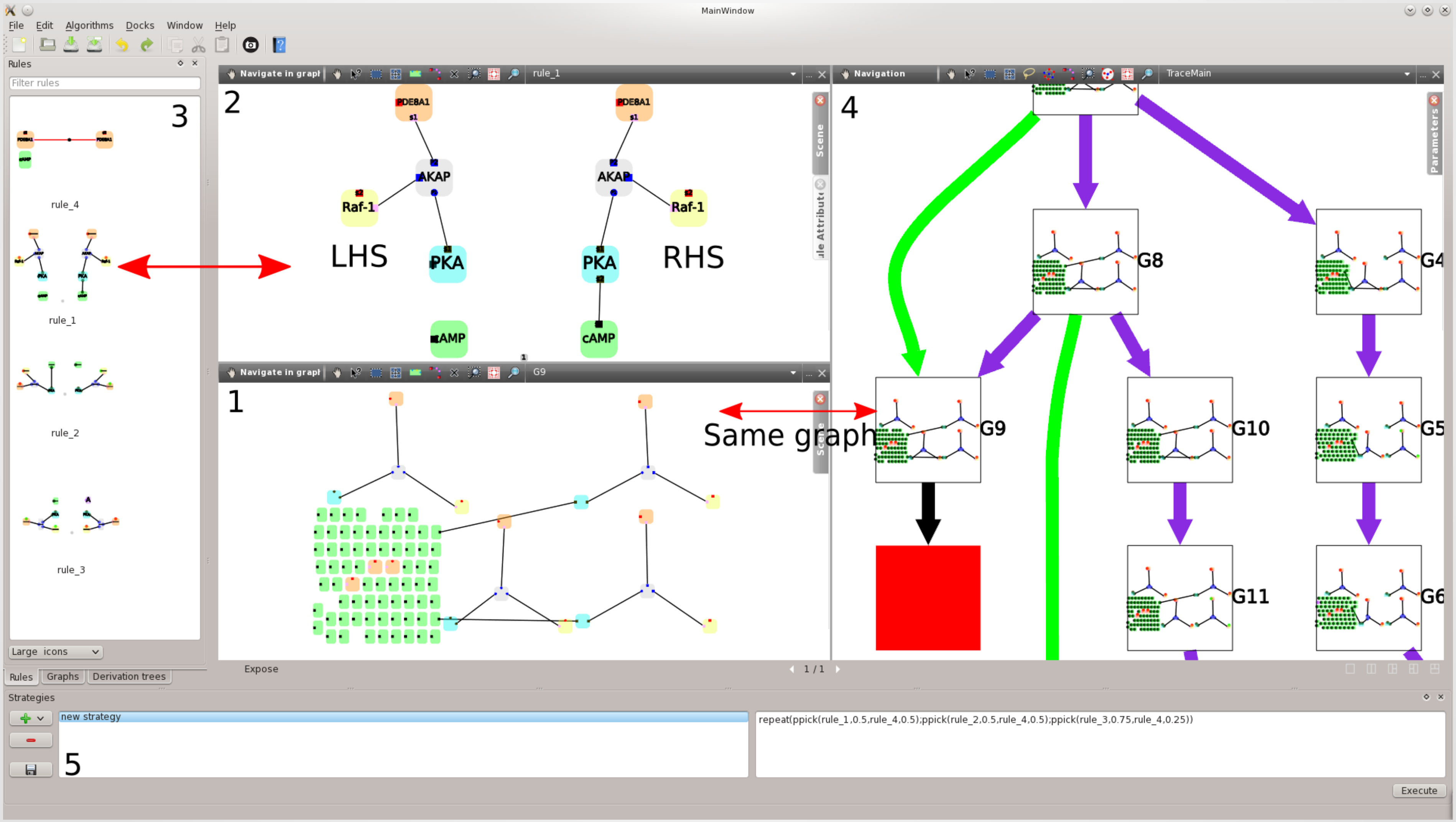
- **In general, a rewriting process is:**
  - Non terminating,
  - Non confluent,
  - Highly concurrent
- **A strategy language**
  - Some steps may be correlated (one followed by another),
  - Iterated until some condition is met,
  - or may occur only in some parts of the graph.
  - Non-deterministic choices: for simultaneous exploration of multiple rewriting scenarios and backtrack to test alternate strategies
- **A derivation tree**
  - History mechanism to record evolutions and choice points
  - Track properties along different scenarios

Porgy Overview

The PORGY environment features:

- Design port graphs and port rules and visualise them.
- Interactive application of a rule on a port graph.
- Creating and running a strategy.
- Exploration and analysis of a derivation tree.
  - Tooltips (get information)
  - Small multiples and animation (show the evolution of the graph)
  - Histograms (to follow graph parameter over rewriting operations)

Porgy Overview

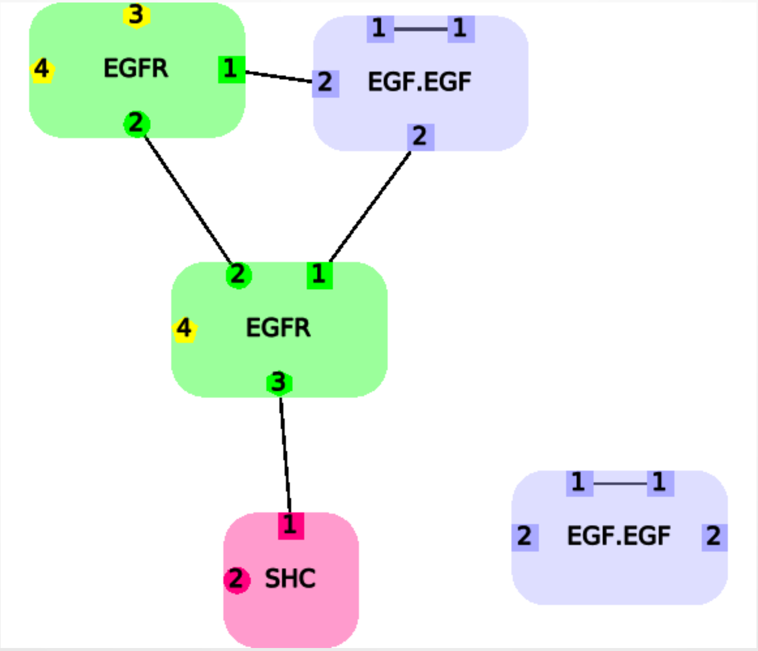


Port Graphs

[IbanescuBK03], [AndreiK07], κ-calculus [DanosL04], BioNetGen [BlinovYFH05]}

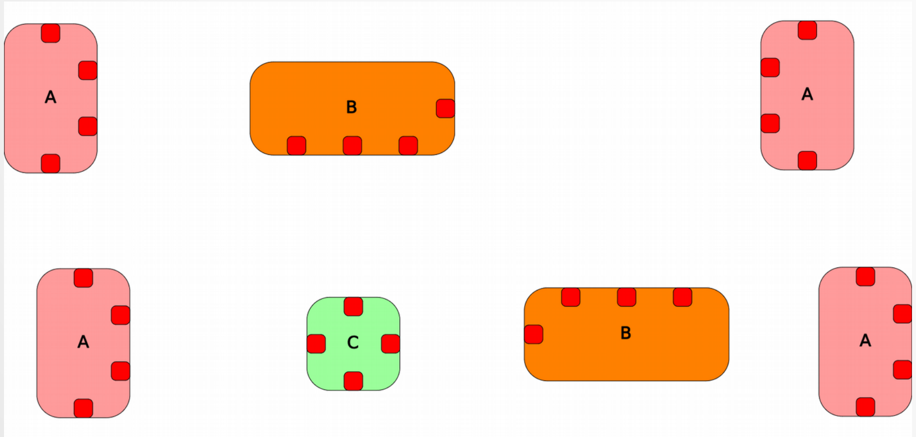
- Inspired by protein-protein interactions;
- Port graphs are graphs with multiple edges and loops, where:
  - Nodes have explicit connection points, called **ports**.
  - The edges attach only to ports of nodes.
  - Nodes, ports and edges have properties (ex: color, arity, boolean value, string, ...).

Actually equivalent to usual labeled graphs, but with more structure.

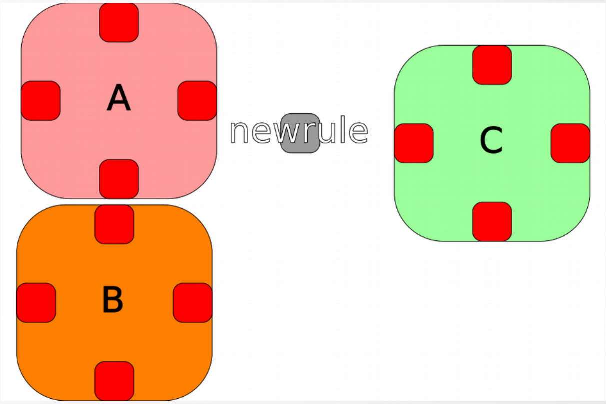


Notion of position: where to apply a rule in a graph?

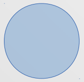
- Top-down or bottom-up traversals do not make sense.
- PORGY's solution is **located graphs** along with **located rewrite rules**.



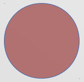
Goal : compute a morphism *g* of the left-hand side inside the graph



A rule called « newrule »



: position subgraph

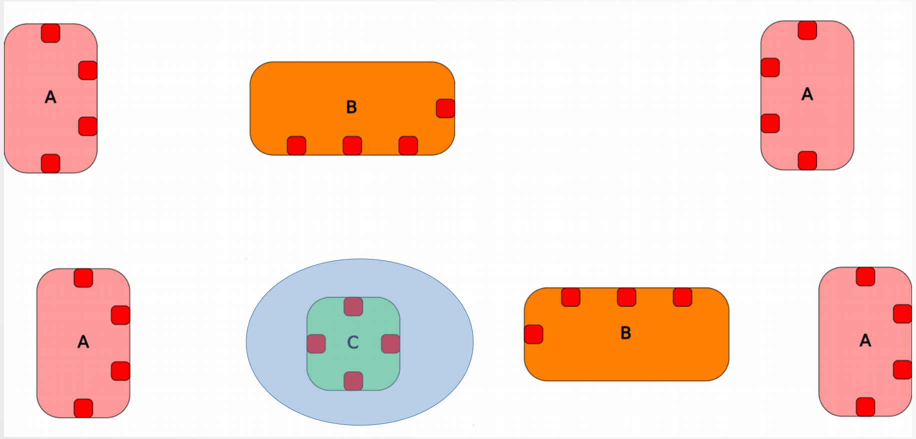


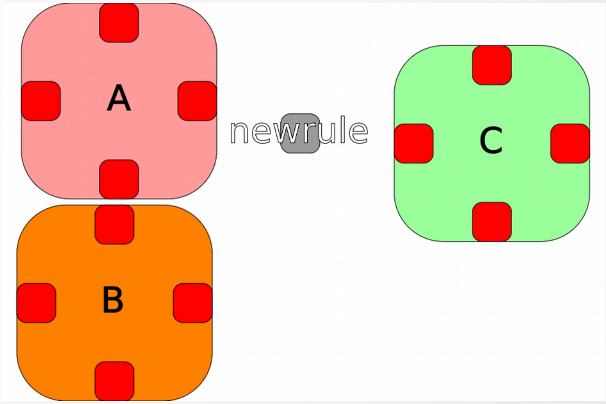
: ban subgraph

Notion of position: where to apply a rule in a graph?

- Top-down or bottom-up traversals do not make sense.
- PORGY's solution is **located graphs** along with **located rewrite rules**.

Goal : compute a morphism *g* of the left-hand side inside the graph





A rule called « newrule »

: position subgraph

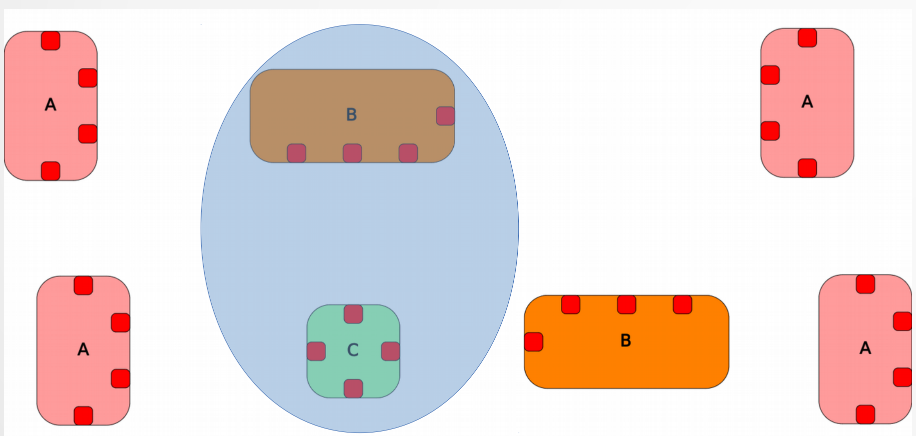
: ban subgraph

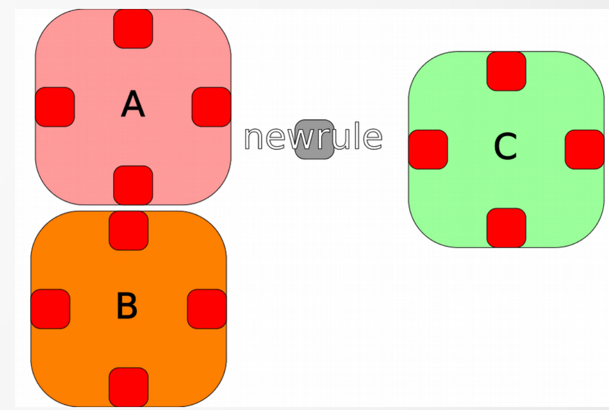
NOK! There is no A or B nodes in the position subgraph

Notion of position: where to apply a rule in a graph?

- Top-down or bottom-up traversals do not make sense.
- PORGY's solution is **located graphs** along with **located rewrite rules**.

Goal : compute a morphism *g* of the left-hand side inside the graph





A rule called « newrule »

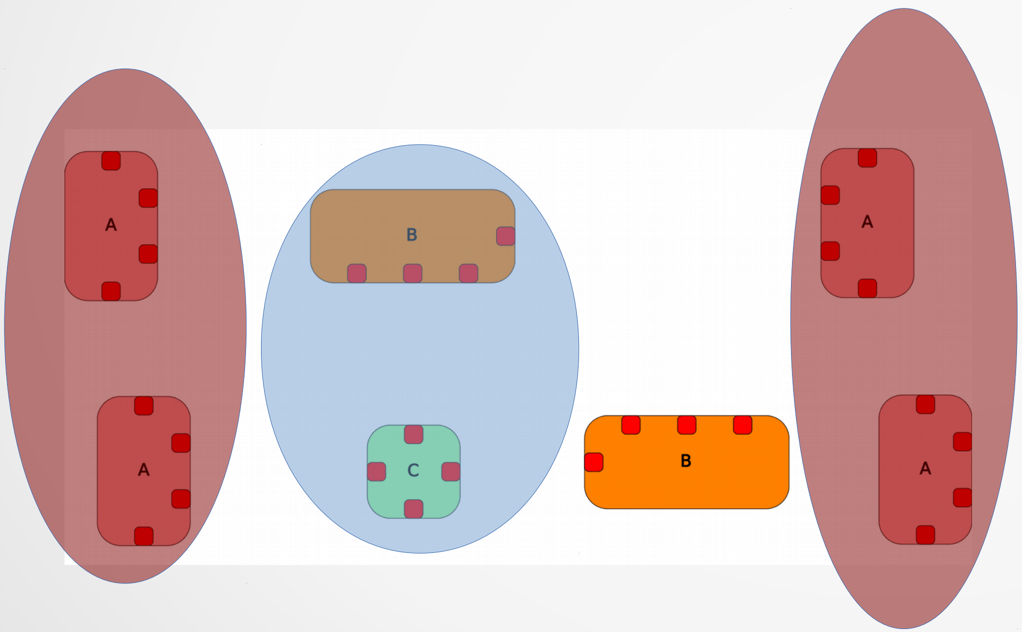
: position subgraph

: ban subgraph

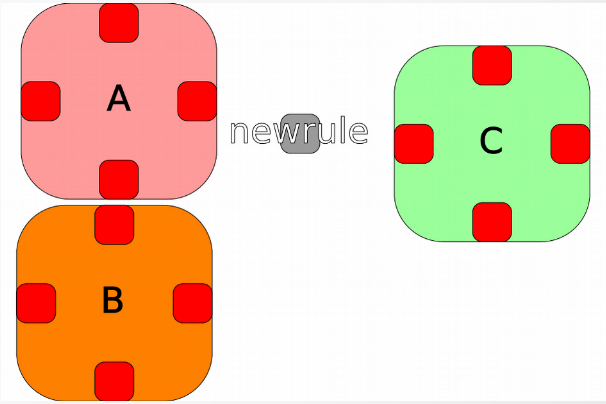
OK! The matching will only be possible with the B node of the position subgraph

Notion of position: where to apply a rule in a graph?

- Top-down or bottom-up traversals do not make sense.
- PORGY's solution is **located graphs** along with **located rewrite rules**.



Goal : compute a morphism *g* of the left-hand side inside the graph



A rule called « newrule »

: position subgraph

: ban subgraph

NOK! All A nodes are banned

Located Graph and Located Rules

A ***located graph*** G_Q^P consists of a port graph *G* and two distinguished subgraphs *P* and *Q* of *G*, called resp. the ***position subgraph***, or simply ***position***, and the ***banned subgraph***.

A ***located rewrite rule*** consists of a port graph rewrite rule $L \Rightarrow R$ and two disjoints subgraphs *M* and *N* of *R*. It is denoted $L \Rightarrow R_M^N$.

- Rewriting must take place at least partially in *P* and not in *Q*.
- To apply a rule, $g(L) \cap P \neq \emptyset$ and $g(L) \cap Q = \emptyset$
- *G* is updated to $(G \setminus g(L)) \cup g(R)$
- *P* is updated to $(P \setminus g(L)) \cup g(M)$
- *Q* is updated to $Q \cup g(N)$

Grammar: rule applications and strategies

Let L, R be port graphs; M, N positions;

$$n \in \mathbb{N}; \, p_{i=1 \ldots n} \in [0,1]; \sum_{i=1}^{n} p_i = 1$$

(Transformations) $T ::= L \Rightarrow R_M^N$

(Applications) $A ::= \mathtt{Id} \mid \mathtt{Fail} \mid \mathtt{all(T)} \mid \mathtt{one(T)}$

(Strategies) $S ::= A \mid S \,;\, S \mid \mathtt{repeat}(S) \mid \mathtt{while}(S)\ \mathtt{do}\ (S)$
$\mid (S)\mathtt{orelse}(S) \mid \mathtt{if}(S)\ \mathtt{then}\ (S)\ \mathtt{else}\ (S)$
$\mid \mathtt{ppick}(S_1, p_1, \ldots, S_n, p_n)$
$\mid U$

Grammar 2/2: position updates

Let *attribute* be an attribute label;
n a valid value for the given attribute label;
function-name the name of a built-in or user-defined function.

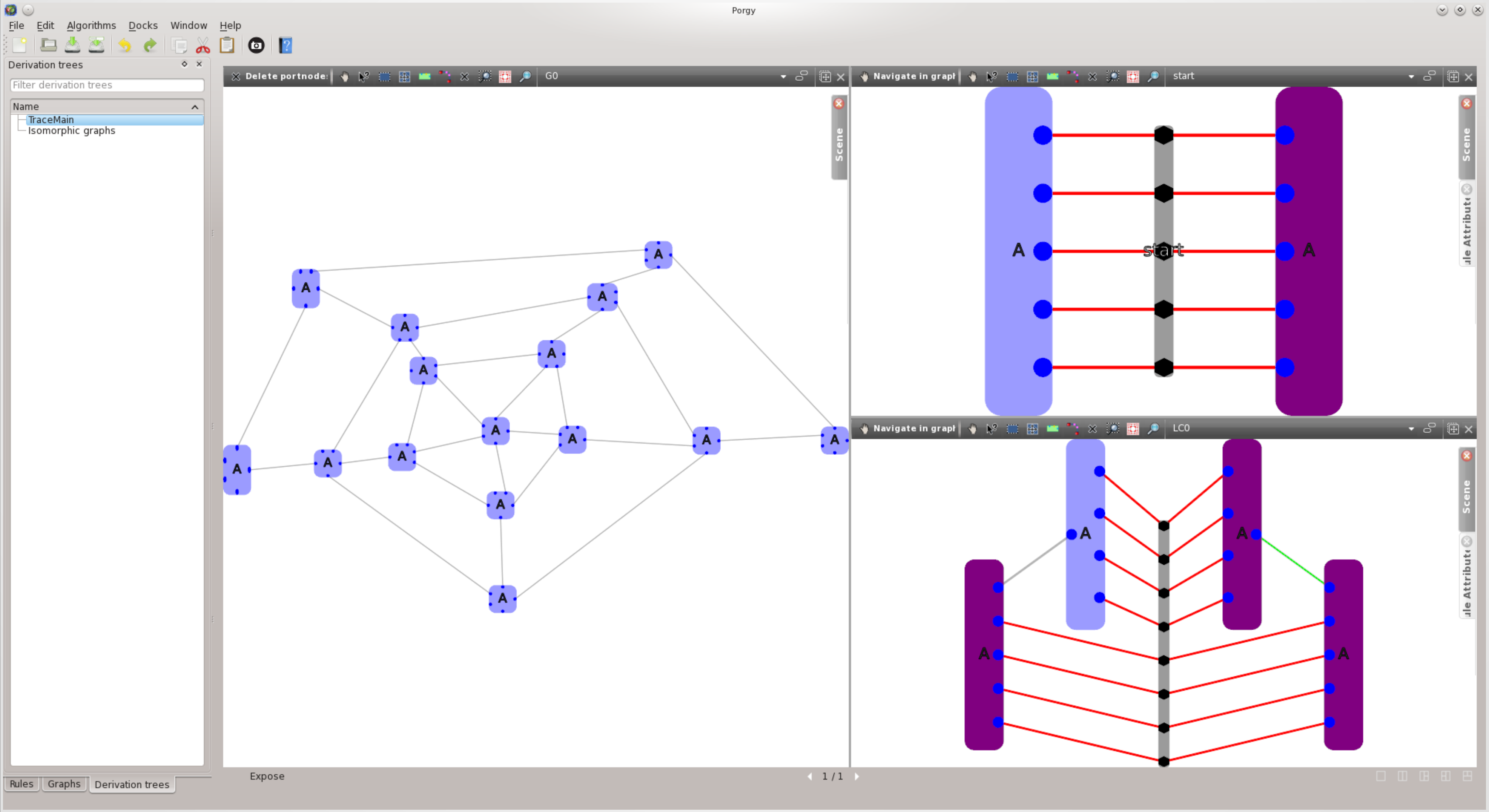
(Position Update) *U* ::= `setPos`(*F*) | `setBan`(*F*) | `isEmpty`(*F*)

(Focusing) *F*::= `CrtGraph` | `CrtPos` | `CrtBan` | `AllNgb`(*F*)
| `OneNgb`(*F*) | `NextNgb`(*F*)
| $F \cup F$ | $F \cap F$ | $F \setminus F$ | $\emptyset$
| `Property`(ρ, *F*)

(Properties) ρ := (*Elem*, *Expr*) | (Function,*function-name*)
Elem := Node | Edge | Port
Expr := Label == *n* | Label ! = *n* | *attribute Relop attribute*
| *attribute* Relop *n*
Relop := == | ! = | > | < | >= | <=

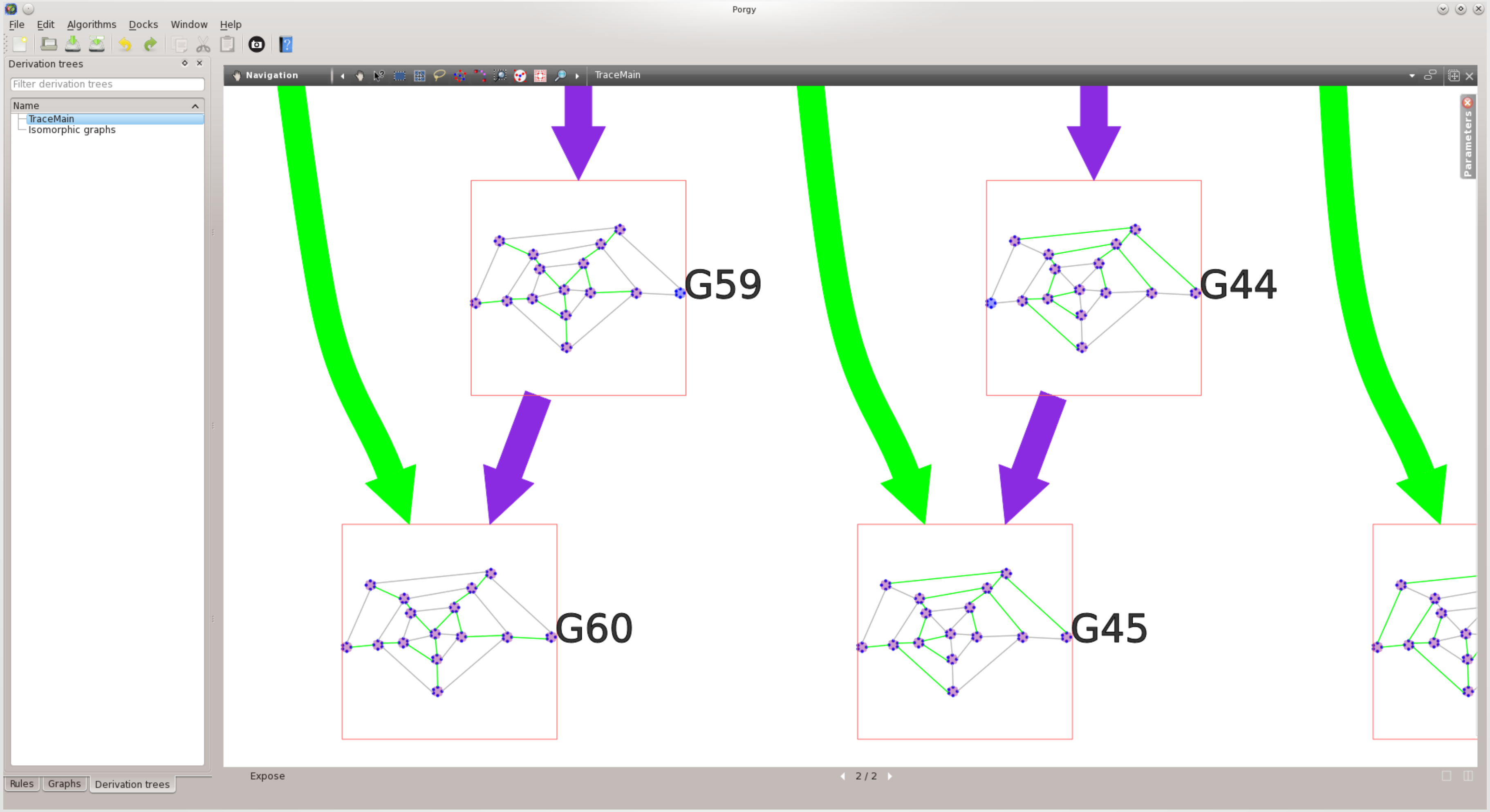
Other constructs : not(S):= if(S)then(Fail)else(Id)
try(S):= (S) orelse (Id)

Examples : spanning tree computation



Examples : spanning tree computation

Very simple strategy for one solution: ***one(start);repeat(LC0)***
Strategy to find all solutions (*all()* not fully implemented yet) : ***all(start);repeat(LC0)***

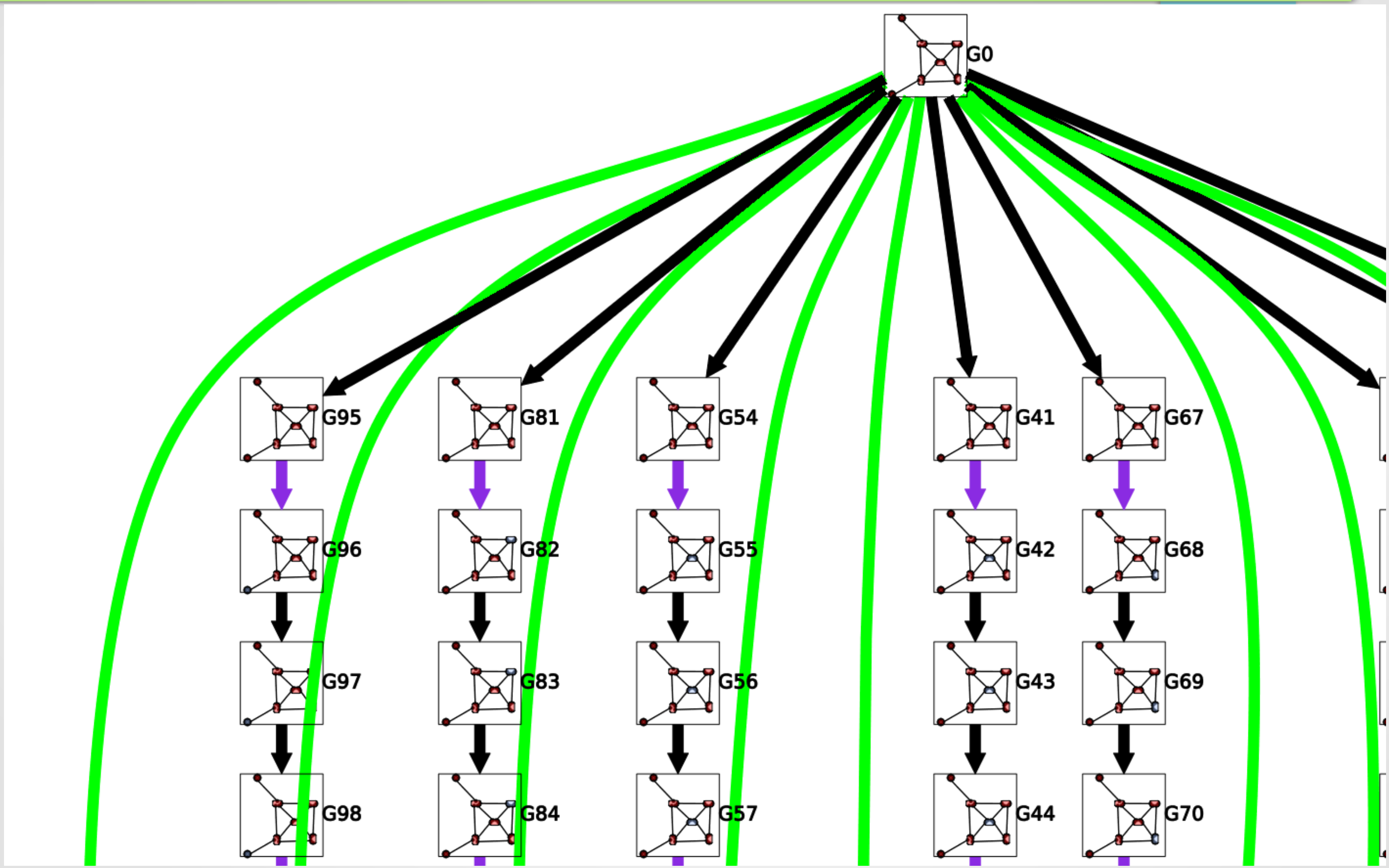


Examples: simple connectivity test

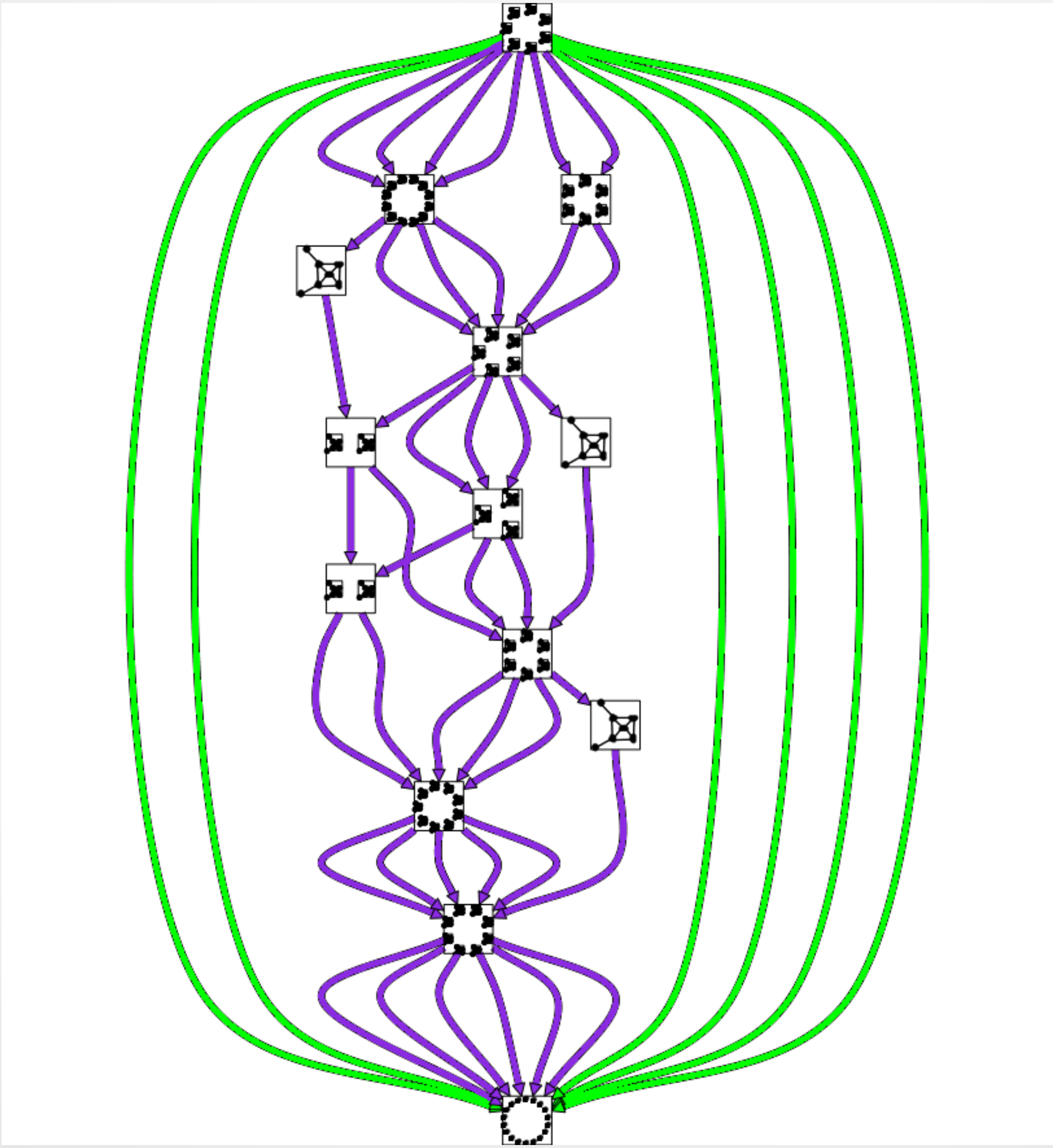
The strategy code with only one rule which mark a visited node :

```
setPos(CrtGraph);
one(newrule);

setPos(Property(Node, "state"=="true", CrtGraph));
setPos(AllNgb(CrtPos));

while(not(isEmpty(CrtPos))) do (
if(newrule) then (
newrule
) else (
setPos(AllNgb(CrtPos)\Property(Node, "state"=="true", CrtGraph))
)
);

setPos(CrtGraph);
not(newrule)
```

Examples: simple connectivity test



Examples: simple connectivity test



Semantics and Properties 1/2

- A *strategic graph program* is given by a set of port graph rewrite rules $\mathfrak{R}$, a strategy expression *S* (built from $\mathfrak{R}$) and a located graph G_P^Q. We denote it $[S_{\mathfrak{R}}, G_P^Q]$, or simply $[S, G_P^Q]$ when $\mathfrak{R}$ is clear from the context.
- In the paper, we give rule based semantics of the strategy language: small step operational semantics, specified using transition rules on configurations (multisets of strategic graph programs)
- Some properties:
  - $[S, G_P^Q]$ is *terminating* (Id or Fail) if there is no infinite transition sequence from the initial configuration $[S, G_P^Q]$.
  - The sublanguage that excludes the *while* and *repeat* constructs is terminating.

Semantics and Properties 2/2

Characterisation of Terminal Configurations:
For every strategic graph program $[S, G_P^Q]$ where $S \neq Id$ and $S \neq Fail$, there exists a configuration *C* such that $[S, G_P^Q] \to C$.

$[S, G_P^Q]$, if terminating, reduces to configurations that contain graph programs of the form $[Id, G'^{Q'}_{P'}]$ or $[Fail, G'^{Q'}_{P'}]$ (called values).

Each strategic graph program in the sublanguage that excludes non-deterministic operators (*OneNgb*, *one*, *ppick*, *orelse*) and r*epeat* has at most one *program result* which is a set of values (*Id* / *Fail*).

Conclusion

PORGY, an interactive visual environment for port graph transformation.
Programming with rules and strategies, including focusing capabilities.
Simple and intuitive visualisation and interaction.

Porgy is a set of Tulip plugins, about 20 000 lines of C++ code (with Qt library for GUI).
Tulip is a graph visualisation and manipulation framework (see http://tulip.labri.fr)

Strategy developed with the Boost Spirit library (only 1 500 lines of code).

Conclusion

PORGY, an interactive visual environment for port graph transformation.
Programming with rules and strategies, including focusing capabilities.
Simple and intuitive visualisations and interactions.

Porgy is a set of Tulip plugins, about 20 000 lines of C++ code (with Qt library for GUI).
Tulip is a graph visualisation and manipulation framework (see http://tulip.labri.fr)

Strategy developed with the Boost Spirit library (only 1 500 lines of code).

Other on-going applications:

Encoding Interaction Nets [laffont:90] programs

Specification/modelling biochemical systems and other complex systems.

Social Network Analysis (SNA, propagation models)

Strategic Port Graph Rewriting : An Interactive Modeling and Analysis Framework

QUESTIONS????

Feel free to ask for a live demo!!!