ISR 2014
Strategies

Hélène KIRCHNER
Inria

August 2014

Strategy languages : Examples

Strategy language

A *strategy language* gives syntactic means to describe strategies.
Several attempts:

ELAN, Stratego, Strafunski, TOM, Maude, PORGY...

Two main purposes:

- build derivation step and derivations
- operationaly compute the next acceptable strategic steps.

Strategy language

A *strategy language* gives syntactic means to describe strategies.
Several attempts:

ELAN, Stratego, Strafunski, TOM, Maude, PORGY...

Two main purposes:

- build derivation step and derivations
- operationaly compute the next acceptable strategic steps.

Languages references

- OBJ3, CafeOBJ, Maude
- Elan, Tom
- ASF+SDF, Stratego, Strafunski
- Porgy

Elan

`http://elan.loria.fr/elan.html`

The ELAN system provides an environment for specifying and prototyping deduction systems in a language based on rewrite rules controlled by strategies. It offers a natural and simple logical framework for the combination of the computation and deduction paradigms as it is backed up by the concepts of rewriting calculus and rewriting logic. It permits to support the design of theorem provers, logic programming languages, constraint solvers and decision procedures and to offer a modular framework for studying their combination.

Tom

https://gforge.inria.fr/projects/tom/

Tom is a programming language particularly well-suited for programming various transformations on tree structures and XML based documents. Tom is a language extension which adds new matching primitives to C and Java as well as support for rewrite rules systems. The rules can be controlled using a strategy language.

Tom is good for:

programming by pattern matching

developing compilers and DSL

transforming XML documents

implementing rule based systems

describing algebraic transformations

Stratego

`http://strategoxt.org/Stratego/WebHome`

Stratego/XT is a language and toolset for program transformation. The Stratego language provides rewrite rules for expressing basic transformations, programmable rewriting strategies for controlling the application of rules, concrete syntax for expressing the patterns of rules in the syntax of the object language, and dynamic rewrite rules for expressing context-sensitive transformations, thus supporting the development of transformation components at a high level of abstraction. The XT toolset offers a collection of extensible, reusable transformation tools, such as powerful parser and pretty-printer generators and grammar engineering tools. Stratego/XT supports the development of program transformation infrastructure, domain-specific languages, compilers, program generators, and a wide range of meta-programming tasks.

Maude

`http://maude.cs.uiuc.edu/`

Maude is a high-performance reflective language and system supporting both equational and rewriting logic specification and programming for a wide range of applications. Maude has been influenced in important ways by the OBJ3 language, which can be regarded as an equational logic sublanguage. Besides supporting equational specification and programming, Maude also supports rewriting logic computation.

Maude supports in a systematic and efficient way logical reflection. This makes Maude remarkably extensible and powerful, supports an extensible algebra of module composition operations, and allows many advanced metaprogramming and metalanguage applications. Indeed, some of the most interesting applications of Maude are metalanguage applications, in which Maude is used to create executable environments for different logics, theorem provers, languages, and models of computation.

Porgy

tulip.labri.fr/TulipDrupal/?q=porgy

PORGY aims at designing relevant graphical representations and adequate interactions on dynamic graphs emerging from graph rewriting systems. Graph rewriting systems appear as a powerful formalism to capture and study phenomena occurring in complex systems, such as the evolution of bio-molecular networks, adhoc communication networks or interaction nets. The ability to act on the simulation of the rewriting calculus will offer the expert a unique mean of interacting with the systems they design and study, turning interactive visualisation of graph rewriting systems into a high-level visual programming environment.

ISR 2014
Strategies

Hélène KIRCHNER
Inria

August 2014

Strategy languages : Constructs

Strategy language

Notation:
t or G denotes a syntactic expression (term, graph,...)
S is a strategy expression in a strategy language on a rewrite rule system $\mathcal{R}$
Application of S to G is denoted $S \bullet G$.

Elementary strategies

- *Identity*, *Failure*
  Id and Fail are two constant strategies that respectively denote cess and failure.
- Rule $l \Rightarrow r$
  A (labelled) rule R is a strategy.

Strategy language

Application of rules

- Select one match
  If a rule R has several possible applications (due to several matchings for instance), $One(R)$ non-deterministically computes only one of them and ignore the others.
- Apply all :
  If a rule R has several possible applications $All(R)$ applies R in all possible ways, giving different branches.
- Parallel application :
  $R_1 \parallel R_2$ applies rules R_1 and R_2 at disjoint positions.
  (generalized to $R_1 \parallel \ldots \parallel R_n$)

Strategy language

Application of rules

- Probabilistic application :
  When probabilities $p_1, \ldots, p_n \in [0, 1]$ are associated to rules $R_1, \ldots, R_n$ such that $p_1 + \ldots + p_n = 1$, the strategy $\texttt{ppick}(R_1, p_1, \ldots, R_n, p_n)$ picks one of the rules for application, according to the given probabilities.
- Don't care:
  In ELAN, the construct $dc(R_1, \ldots, R_n)$ non-deterministically chooses one of the rules for application (with an equiprobability).

Strategy language

Building derivations

- Composition : $Sequence(S_1, S_2)$ or $seq(S_1, S_2)$ or S_1 *Then* S_2 or S_1 ; S_2
  applies S_1 then S_2.

Strategy language

Choices (selection of branches in the derivation tree)

- $First(S_1, S_2)$ or $(S_1)\texttt{orelse}(S_2)$ or $S_1 <^+ S_2$
  selects the first strategy that does not fail; it fails if both fail
- $Try(S)$ tries the strategy S but never fails

$$Try(S) = First(S, \mathrm{Id})$$

- $One(S)$ selects one possible application of the strategy S
  $One(S_1, \ldots, S_n)$ selects one possible application of one strategy.
- Probabilistic choice $\texttt{ppick}(S_1, p_1, \ldots, S_n, p_n)$
  When probabilities $p_1, \ldots, p_n \in [0, 1]$ are associated to strategies $S_1, \ldots, S_n$ such that $p_1 + \ldots + p_n = 1$, $\texttt{ppick}(S_1, p_1, \ldots, S_n, p_n)$ picks one of the strategies for application, according to the given probabilities.

Strategy language

Conditionals and Tests

- *If − Then − Else*
  `if`(S)`then`(S')`else`(S'') checks if application of S returns Id, in which case S' is applied, otherwise S'' is applied
- *Match*
  In Maude, `match`(S) matches the term S to t and returns t if success or Fail otherwise.
- *Not*
  `not`$(S) \triangleq$ `if`(S)`then`(Fail)`else`(Id)
  fails if S succeeds and succeeds if S fails.

Strategy language

Exploiting the structure of objects: terms

- On a term t, $AllSuc(S)$ applies the strategy S on all immediate subterms:

$$AllSuc(S)\bullet f(t_1, ..., t_n) = f(t'_1, ..., t'_n)$$

if $S\bullet t_1 = t'_1, ..., S\bullet t_n = t'_n$; it fails if there exists i such that $S\bullet t_i$ fails.

- On a term t, $OneSuc(S)$ applies the strategy S on the first immediate subterm (if it exists) where S does not fail:

$$OneSuc(S)\bullet f(t_1, ..., t_n) = f(t_1, ..., t'_i, ..., t_n)$$

if for all $1 \leq j < i$, $S\bullet t_j$ fails, and $S\bullet t_i = t'_i$;

it fails if f is a constant or if for all i, $S\bullet t_i$ fails.

Strategy language

Exploiting the structure of objects: graphs

- On a graph G, $AllNbg(S)$ applies the strategy S on all immediate successors of the nodes in G, where an immediate successor of a node v is a node connected to v.
- $OneNbg(S)$ applies the strategy S on one immediate successor of a node in G, chosen non-deterministically.

Strategy language

Recursive strategies and iterations

- Fixpoint: $\mu x.S = S[x \leftarrow \mu x.S]$
- Repeat: *Repeat*(S) keeps on sequentially applying S until it fails and returns the last result

$$\mathit{Repeat}(S) \quad = \quad \mu x.\mathit{First}(\mathit{Sequence}(S, x), \mathit{Identity})$$

- While:
  `while`(S)`do`(S') keeps on sequentially applying S' while the expression S is successful; if S fails, then Id is returned.

Strategy language

Traversal strategies(Tom)

$$
\begin{array}{lcl}
\mathit{OnceBottomUp}(S) & = & \mu x.\mathit{First}(\mathit{OneSuc}(x), S) \\
\mathit{BottomUp}(S) & = & \mu x.\mathit{Sequence}(\mathit{AllSuc}(x), S) \\
\mathit{TopDown}(S) & = & \mu x.\mathit{Sequence}(S, \mathit{AllSuc}(x)) \\
\mathit{Innermost}(S) & = & \mu x.\mathit{Sequence}(\mathit{AllSuc}(x), \mathit{Try}(\mathit{Sequence}(S, x)))
\end{array}
$$

Strategy language

Recursive closure strategies (Stratego [Visser01])
The recursive closure $recx(S)$ of the strategy S attempts to apply S to the entire subject term and the strategy $recx(S)$ to each occurrence of the variable x in S.

$$
\begin{array}{lcl}
try(S) & = & S <^{+} id \\
repeat(S) & = & recx(try(S; x)) \\
while(c, S) & = & recx(try(c; S; x)) \\
do - while(S, c) & = & recx(S; try(c; x)) \\
while - not(c, S) & = & recx(c <^{+} S; x) \\
for(i, c, S) & = & i; while - not(c, S)
\end{array}
$$

Exercise : KB completion

```
[Delete] (E U s=s ; R) => (E ; R)
[Compose] (E ; R U s->t) => (E ; R U s->u)
                            if reduce(t->u)
[Simplify] (E U s=t ; R) => (E U s=u ; R)
                            if reduce(t->u)
[Orient] (E U s=t ; R) => (E ; R U s->t)
                          if s > t
[Collapse] (E ; R U s->t) => (E U u=t ; R)
                             if reduce(s->u)
[Deduce] (E ; R) => (E U s=t ; R)
                     if s=t in CP(R)
```

Exercise : KB completion

```
completion =>
  repeat(repeat(repeat(Collapse);
                       repeat(Compose) ;
                       repeat(Simplify) ;
                       repeat(Delete) ;
                       repeat(Orient)) ;
                 Deduce)
```

Exercise : Propositional formulas

```
module prop-laws
imports libstrategolib prop
rules
  DefI : Impl(x, y) -> Or(Not(x), y)
  DefE : Eq(x, y)   -> And(Impl(x, y), Impl(y, x))

  DN   : Not(Not(x)) -> x

  DMA  : Not(And(x, y)) -> Or(Not(x), Not(y))
  DMO  : Not(Or(x, y))  -> And(Not(x), Not(y))

  DAOL : And(Or(x, y), z) -> Or(And(x, z), And(y, z))
  DAOR : And(z, Or(x, y)) -> Or(And(z, x), And(z, y))

  DOAL : Or(And(x, y), z) -> And(Or(x, z), Or(y, z))
  DOAR : Or(z, And(x, y)) -> And(Or(z, x), Or(z, y))
```

Exercise : Propositional formulas

```
strategies

nf = innermost (
DefI <+ DefE <+ DAOL <+ DAOR <+ DN <+ DMA <+ DMO
                                )
```

ISR 2014
Strategies

Hélène KIRCHNER
Inria

August 2014

Operational semantics of strategic programs

Strategic programming

A *strategic rewrite program* consists of a finite set of rewrite rules $\mathcal{R}$, a strategy expression S (built from $\mathcal{R}$ using a strategy language $\mathcal{L}(\mathcal{R})$) and a given structure G.

A *configuration* C is a multiset $\{O_1, \ldots, O_n\}$ where each O_i is a strategic program $[S', G']$.

$$C_0 = \{[S, G]\} \xrightarrow{*} \ldots \xrightarrow{*} C_k = \{\ldots [S'_k, G'_k] \ldots\}$$

Let $Reach([S, G], C_k)$ the set of all intermediate generated structures G' occurring in $C_0, \ldots, C_k$.

Correctness and Completeness

w.r.t. rewriting derivations

$\mathcal{L}(\mathcal{R})$ strategy language, $S \in \mathcal{L}(\mathcal{R})$
$C_0 = \{[S, G]\} \xrightarrow{*} \ldots \xrightarrow{*} C_k = \{...[S'_k, G'_k]...\}$

- Correctness:
  If $C_0 = \{[S, G]\} \xrightarrow{*} \ldots \xrightarrow{*} C_k = \{...[S'_k, G'_k]...\}$
  and if $G' \in \mathit{Reach}([S, G], C_k)$, then $G \rightarrow^*_{\mathcal{R}} G'$.
- Completeness:
  If $G \rightarrow^*_{\mathcal{R}} G'$, there exists $S \in \mathcal{L}(\mathcal{R})$ such that
  $C_0 = \{[S, G]\} \xrightarrow{*} \ldots \xrightarrow{*} C_k = \{...[S'_k, G'_k]...\}$
  and $G' \in \mathit{Reach}([S, G], C_k)$.

Correctness and Completeness

w.r.t. semantics

$\mathcal{L}(\mathcal{R})$ strategy language, $S \in \mathcal{L}(\mathcal{R})$
$C_0 = \{[S, G]\} \xrightarrow{*} \ldots \xrightarrow{*} C_k = \{...[S'_k, G'_k]...\}$
$[\![S]\!] \bullet G$ is a set of derivations

- Correctness:
  If $C_0 = \{[S, G]\} \xrightarrow{*} \ldots \xrightarrow{*} C_k = \{...[S'_k, G'_k]...\}$
  and if $G' \in \mathit{Reach}([S, G], C_k)$, then $G \rightarrow^*_{\mathcal{R}} G' \in [\![S]\!] \bullet G$.
- Completeness:
  If $G \rightarrow^*_{\mathcal{R}} G' \in [\![S]\!] \bullet G$, there exists $S \in \mathcal{L}(\mathcal{R})$ such that
  $C_0 = \{[S, G]\} \xrightarrow{*} \ldots \xrightarrow{*} C_k = \{...[S'_k, G'_k]...\}$
  and $G' \in \mathit{Reach}([S, G], C_k)$.

Operational semantics

- Given a strategic graph program $[S_{\mathcal{R}}, G]$, transitions are performed, according to the strategy S, starting from the initial configuration $\{[S, G]\}$.
- A *result* is of the form $[\mathrm{Id}, G]$ or $[\mathrm{Fail}, G]$.
- A strategic graph program $[S, G]$ is *(strongly) terminating* if there is no infinite transition sequence out of the initial configuration $\{[S, G]\}$, otherwise it is non-terminating.
- It is *weakly terminating* if we can reach a configuration having at least one result.

Operational semantics

Given a strategic graph program $[S, G]$,
- A configuration is terminal if no transition can be performed from it.
- The *result set* associated to a sequence of transitions out of the initial configuration $\{[S, G]\}$ is the set of all the results in the configurations in the sequence.
- If the sequence of transitions out of the initial configuration $\{[S, G]\}$ ends in a terminal configuration, then the result set of the sequence is a *complete result set* for the program $\{[S, G]\}$.
- If a strategic graph program does not reach a terminal configuration (in case of non-termination) then the complete result set is undefined ($\perp$).

Operational semantics

The transition relation $\longrightarrow$ is a binary relation on configurations defined as follows:

$$\{O_1, \ldots, O_k, V_1, \ldots, V_j\} \longrightarrow \{O'_{11}, \ldots, O'_{1m_1}, \ldots, O'_{km_k}, V_1, \ldots, V_j\}$$

if $O_i \rightarrow \{O'_{i1}, \ldots, O'_{im_i}\}$, for $1 \leq i \leq k$, where $k \geq 1$ and some of the O'_{ij} might be results.

Operational semantics

$$\frac{}{[\mathtt{one}(l \Rightarrow r), G] \to \{[\mathsf{Id}, G']\}}\ G' \in LS_{l \Rightarrow r}(G)$$

where LS is the legal set of reducts

$$\frac{}{[\mathtt{one}(l \Rightarrow r), G] \to \{[\mathsf{Fail}, G]]\}}\ LS_{l \Rightarrow r}(G) = \varnothing$$

Operational semantics

$$\frac{}{[\mathsf{Id}; S, G] \to \{[S, G]\}}$$

$$\frac{}{[\mathsf{Fail}; S, G] \to \{[\mathsf{Fail}, G]\}}$$

$$\frac{[S_1, G] \to \{[S_1^1, G_1], \ldots, [S_1^k, G_k]\}}{[S_1; S_2, G] \to \{[S_1^1; S_2, G_1], \ldots, [S_1^k; S_2, G_k]\}}$$

Operational semantics

$$\frac{\exists G', M \; s.t. \; \{[S_1, G]\} \longrightarrow^* \{[\mathsf{Id}, G'], M\}}{[\texttt{if}(S_1)\texttt{then}(S_2)\texttt{else}(S_3), G] \rightarrow \{[S_2, G]\}}$$

$$\frac{\nexists G', M \; s.t. \; \{[S_1, G]\} \longrightarrow^* \{[\mathsf{Id}, G'], M\}}{[\texttt{if}(S_1)\texttt{then}(S_2)\texttt{else}(S_3), G] \rightarrow \{[S_3, G]\}}$$

Operational semantics

Semantics of while/repeat loops:

$$\frac{}{[\texttt{while}(S_1)\texttt{do}(S_2), G_P^Q] \rightarrow \{[\texttt{if}(S_1)\texttt{then}(S_2;\texttt{while}(S_1)\texttt{do}(S_2))\texttt{else}(\mathsf{Id}), G]\}}$$

Termination

Strategic programs are not terminating in general, however it may be suitable to identify a terminating sublanguage (i.e. a sublanguage for which the transition relation is terminating)
Pb: characterise the strategic programs that yield terminal configurations.

- **Termination property:**
  The sublanguage that excludes iterators (such as the while/repeat construct) is strongly terminating.

Lemma:
If $[S_1, G]$ is strongly terminating and S_2 is such that $[S_2, G']$ is strongly terminating for any G', then $[S_1; S_2, G]$ is strongly terminating.

Properties to prove

- **Progress property: Characterisation of Terminal Configurations**
  For every strategic graph program $[S, G]$ that is not a result (i.e., $S \neq \mathrm{Id}$ and $S \neq \mathrm{Fail}$), there exists a configuration C such that $\{[S, G]\} \rightarrow C$.

- **Uniqueness/ Determinism property:**
  Il the language contains only non-deterministic operators (excluding *One* for instance):
  every strategic graph program has at most one *result set*.

- **Computational Completeness property:**
  The set of all strategic programs $[S_R, G]$ is Turing complete, i.e. can simulate any Turing machine. (Sequential composition and iteration are enough [HaberPlump2001])

ISR 2014
Strategies

Hélène KIRCHNER
Inria

August 2014

PORGY

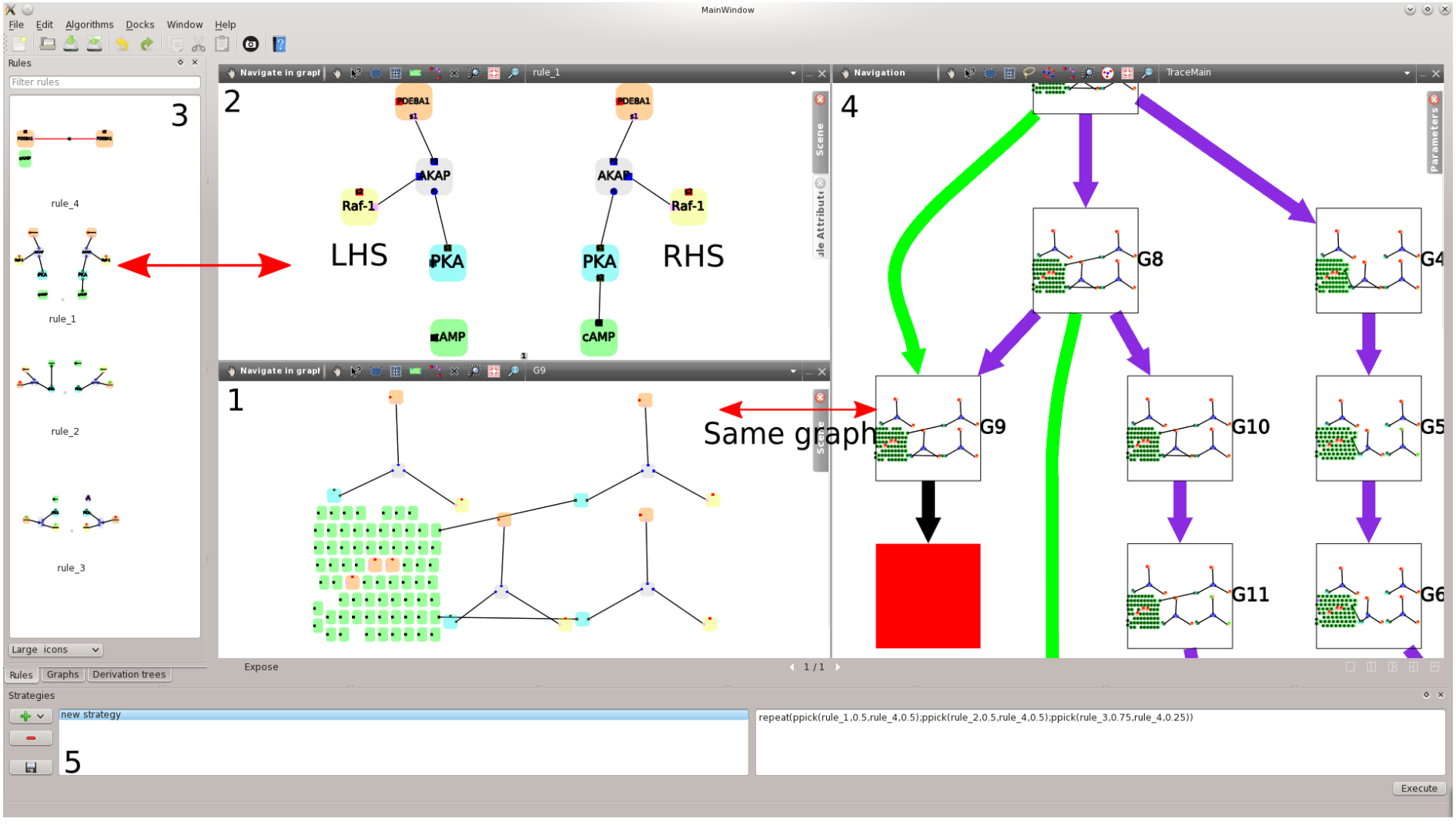
Graph rewriting strategies
PORGY strategy language

Goal: to facilitate the specification, analysis and simulation of complex systems, using port graphs.

A system is represented by
- an initial graph,
- a collection of graph rewriting rules,
- a user-defined strategy to control the application of rules.

The strategy language includes constructs to deal with graph traversal and management of rewriting positions in the graph.

PORGY Overview



(1) one state of the graph (2) a rule; (3) all rules; (4) derivation tree; (5) the strategy editor

Graph rewriting strategies
PORGY strategy language

Specific case of positions

- Where to apply a rule in a graph?
- Top-down or bottom-up traversals do not make sense.
- Need for a strategy language which includes operators to select rules and the positions where the rules are applied, and also to change the positions along the derivation.

Graph rewriting strategies

PORGY solution (Porgy2011):
A *located graph* G_P^Q consists of a graph G, a subgraph P of G called the *position subgraph* and a subgraph Q of G called the *banned subgraph*.

- Rewriting must take place fully or partially in P.
- No rewriting can happen fully or partially in Q.

Graph rewriting strategies
New constructs

A strategy expression combines

- *applications* of located rewrite rules, generated by the non-terminal A,
- *position updates*, generated by the non-terminal U,
- *focusing expressions* generated by F.

PORGY Strategy language

Usual constructs

Let L, R be port graphs; M, N positions; $n \in \mathbb{N}$;

$\pi_{i=1\ldots n} \in [0,1]; \sum_{i=1}^{n} \pi_i = 1$

$$
\begin{array}{rcl}
S & ::= & A \mid U \mid S; S \mid \texttt{repeat}(S) \mid \texttt{while}(S)\texttt{do}(S) \\
 & \mid & (S)\texttt{orelse}(S) \mid \texttt{if}(S)\texttt{then}(S)\texttt{else}(S) \\
 & \mid & \texttt{ppick}(S_1, \pi_1, \ldots, S_n, \pi_n) \\
A & ::= & \mathsf{Id} \mid \mathsf{Fail} \mid \texttt{all}\,T \mid \texttt{one}\,T \\
T & ::= & L_W \Rightarrow R_M^N \mid (T \parallel T)
\end{array}
$$

Graph rewriting strategies
New constructs

Focusing Strategies

$$
\begin{array}{rcl}
U & ::= & \texttt{setPos}(F) \mid \texttt{setBan}(F) \mid \texttt{isEmpty}(F) \\
F & := & \texttt{CrtGraph} \mid \texttt{CrtPos} \mid \texttt{CrtBan} \mid \texttt{AllNbg}(F) \\
 & \mid & \texttt{OneNbg}(F) \mid \texttt{NextNbg}(F) \mid \texttt{Property}(\rho, F) \\
 & \mid & F \cup F \mid F \cap F \mid F \setminus F \mid \varnothing
\end{array}
$$

Properties

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

$$
\begin{array}{rcl}
\rho & := & (\mathit{Elem}, \mathit{Expr}) \mid (\texttt{Function},\ \mathit{function\text{-}name}) \\
\mathit{Elem} & := & \texttt{Node} \mid \texttt{Edge} \mid \texttt{Port} \\
\mathit{Expr} & := & \texttt{Label} == a \mid \texttt{Label} \mathrel{!}= a \mid \mathit{attribute}\ \mathit{Relop}\ \mathit{attribute} \\
 & \mid & \mathit{attribute}\ \mathit{Relop}\ a \\
\mathit{Relop} & := & == \mid \mathrel{!}= \mid > \mid < \mid >= \mid <=
\end{array}
$$

Outermost rewriting on trees

start $\triangleq$ `Property`((*Function*, *Root*), `CrtGraph`) selects the subgraph containing just the root of the tree.
The *next* ports for each node are defined to be the ones that connect with their children.

The strategy for outermost rewriting with a rule *R* is:

```
setPos(start);
while(not(isEmpty(CrtPos)))do(
if(R)then(R;setPos(start))else(setPos(AllNbg(CrtPos))))
```

Innermost rewriting on trees

start $\triangleq$ `Property`((*Function*, *Leaf*), `CrtGraph`) selects the leaves of the tree
rest $\triangleq$ `CrtGraph` $\setminus$ *start*
For each node, the *next* port connects with the parent node.

```
setPos(start);setBan(rest);
while(not(isEmpty(CrtPos)))do(
  if(R)then(R ;setPos(start) ;setBan(rest))else(
  setPos(AllNbg(CrtPos)) ;setBan(CrtBan\ CrtPos)))
```

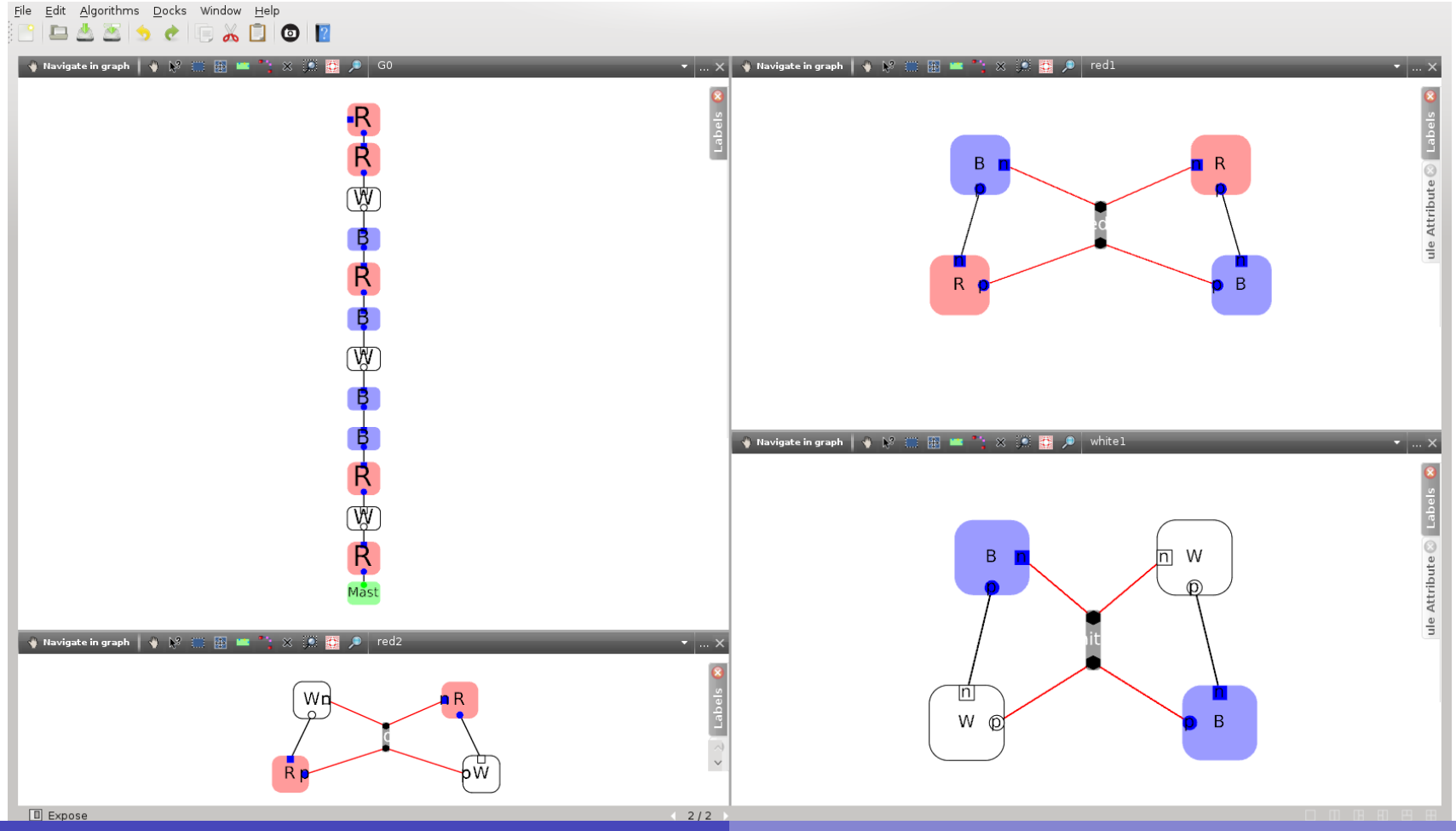
Sorting Exercise: french flag

Sort a list of three colours (Blue, Red and White) to represent the French flag (Blue first, then White and finally Red)

- three port nodes representing each colour that have two ports each: a *previous* port (to the left of the node) and a *next* port (to the right of the node).
- a *Mast* port node at the beginning of the list
- three rules to swap two colours if they are in the wrong order.

Sorting Exercise: french flag

- the list to sort (top left)
- the three flag sorting rules *white1* (bottom right), *red1* (top right) and *red2* (bottom left) in Porgy



Sorting Exercise: french flag

Define:

$swap \triangleq ((white1)\texttt{orelse}(red1))\texttt{orelse}(red2)$

$backToMast \triangleq \texttt{Property}((Node, mast), \texttt{CrtGraph})$

Strategy:

```
setPos(backToMast);
while(not(isEmpty(CrtPos)))do(
  if(swap)then(
    swap;
    setPos(backToMast)
)else(
  setPos(AllNbg(CrtPos)))
)
```

Sorting Exercise: french flag

Write a strategy to test if the list is correctly sorted.

Hint: From the begining of the list, the strategy traverses all nodes with the corresponding color. It fails on the first node of the wrong color.
Define:
CheckXNodes $\triangleq$

```
        if(isEmpty(Property((Node, Colour=X),CrtPos)))
then(
          Fail);
        setPos(NextNbg(CrtPos));
        while(not(isEmpty(Property((Node,
Colour=X),CrtPos))))do(
          setPos(NextNbg(CrtPos)));
```

where X defines the color to check.

Sorting Exercise: french flag

Strategy:

```
setPos(backToMast);
  setPos(NextNbg(CrtPos));
CheckBlueNodes;
CheckWhiteNodes;
CheckRedNodes;
CheckEnd
```

where CheckEnd checks if the list is terminated, i.e. the current position set is empty:

```
if(not(isEmpty(CrtPos))) then(
  Fail)
```

Graph testing exercise

Test if a graph is connected. If not, the strategy ends on a failure.

Hint: Assuming that in the initial graph all nodes have the Boolean attribute *state* set to false, we need just one rewriting rule, which simply sets *state* to *true* on a node.
The strategy pick-one-node selects a node *n*, non-deterministically, as a starting point. The current position set is set to all neighbours of *n*. Then, the rule is applied as long as possible. If the rule cannot be applied, the position subgraph is set to all neighbours of its nodes which are not already tagged (visit-neighbours-at-any-distance). When there is no more nodes in the position subgraph, if the rule can still be applied, there is another connected component in the graph, so the strategy ends on a failure (check-all-nodes-visited).

Graph testing exercise

pick-one-node:

```
setPos(CrtGraph);
oneR;
setPos(Property((Node, state==true,CrtGraph));
```

visit-neighbours-at-any-distance:

```
setPos(AllNbg(CrtPos));
while(not(isEmpty(CrtPos)))do(
  if(R)then(R)else(
    setPos(AllNbg(CrtPos) \ Property((Node,
state==true),CrtGraph))
  ));
```

check-all-nodes-visited:

```
setPos(CrtGraph);
if(R) then(Fail)
```

ISR 2014
Strategies

Hélène KIRCHNER
Inria

August 2014

Verification - Open problems

ARS versus other formalisms

ARS coincide mathematically with STS - state transition systems

Comparison with other formalisms:
- labeled transition systems LTS
- Kripke structures KS
- Buchi automata BA
- Petri Nets PN
- Concurrent Game Structure CGS

STS differ however from finite state automata in several ways:
- In STS, the set of states and the set of transitions are not necessarily finite, or even countable.
- A finite-state automaton distinguishes a special "start" state and a set of special "final" states.

Labeled transition systems

In LTS, a labeling function maps each state to the set of atomic propositions that are true in this state.

One can define:

- trace equivalence
- simulation
- bisimilarity

Concurrent Game Structure

CGSs provides a generalization of labeled transition systems, modeling multi-agent systems, viewed as multi-player games in which players perform concurrent actions, chosen strategically as a function of the history of the game. [MogaveroMV-FSTTCS10]

A strategy is a plan for an agent that contains all choices of moves as a function of the history of the current outcome.

Concurrent Game Structure

[MogaveroMV-FSTTCS10]
A *Concurrent Game Structure (CGS)*

$$\mathcal{G} = \langle AP, Ag, Ac, St, \lambda, \tau, s_0 \rangle$$

- AP finite non-empty set of atomic propositions
- Ag finite non-empty set of agents
- Ac enumerable non-empty set of actions
- St enumerable non-empty set of states, $s_0 \in St$ initial state
- $\lambda : St \mapsto 2^{AP}$ labeling function that maps each state to the set of atomic propositions that are true in this state.
- Decisions or action profiles $Dc = Ac^{Ag}$ are functions representing the choices of an action for each agent.
- $\tau : St \times Dc \mapsto St$ is the transition function

Concurrent Game Structure

- *Tracks* and *paths* are legal sequences (resp. finite and infinite) of reachable states in $\mathcal{G}$ that can be seen as possible outcomes of the game modeled by $\mathcal{G}$.
  Let $Trk \subseteq ST^+$ and $Pth \subseteq ST^\omega$ be the sets of non-trivial tracks and paths.
- A *strategy* is a partial function $f : Trk \rightarrow Ac$, non associated to any particular agent, mapping each non-trivial track in its domain to an action.
- A *play* is the outcome of a game determined by all the agent strategies participating to the game.

Concurrent Game Structure

[MogaveroMV-FSTTCS10]
Strategy Logic SL extends LTL with existential and universal strategy quantifiers an agent binding (a, x) meaning "bind agent a to the strategy associated with variable x".

In SL the model checking problem is decidable (2ExpTime complete)

Conclusion

Further topics:

- Game theory for strategies
- Proving properties of strategies and strategic reductions
- Strategies for autonomic computing and runtime verification