On Data-Structure Rewriting

Rachid Echahed
LIG Lab, Grenoble
France

June, 2010

Rewriting (reminder)

- A rewrite relation R is a binary relation

$$R \subseteq A \times A$$

- $(u, v) \in R$ is read "u rewrites into v" and written

$$u \rightarrow v$$

Rewriting (reminder)

$$R \subseteq A \times A$$

- $A =$ set of strings over a vocabulary (V^*)
- $A =$ set of states of the form (variables, valuation)
- $A =$ set of Turing Machine configurations
- $A =$ set of lambda-terms
- $A =$ set of trees (or terms)
- $A =$ set of clauses
- $A =$ set of process terms
- $A = \ldots$

Rewriting

$$R \subseteq A \times A$$

- How to define a rewrite relation R?
- How to define a run or the execution of a program?
  - A rewrite derivation : u_0 is the initial “call”

    $$u_0 \longrightarrow u_1 \longrightarrow \ldots \longrightarrow u_n$$

  - A narrowing derivation :
    w_0 is the initial goal (to solve):

    $$w_0 \leadsto_{\sigma_1} w_1 \ldots \leadsto_{\sigma_n} w_n$$

    Where

    $$w_i \leadsto_{\sigma} w_{i+1} \text{ iff } \sigma(w_i) \longrightarrow w_{i+1}$$

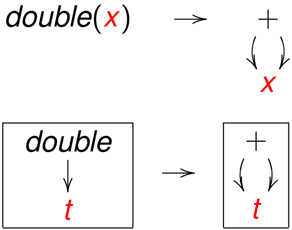
    w_i is an element of A with partial information
    σ instantiates w_i

Motivation : Extension of Term Rewriting ; sharing subterms

Function definitions by means of term rewrite rules

$$\begin{array}{lcl} 0 + x & \rightarrow & x \\ succ(x) + y & \rightarrow & succ(x + y) \\ double(x) & \rightarrow & x + x \end{array}$$

Very well established domain with several results : Confluence, Termination, Strategies, Proof methods (equational reasoning, induction) etc.



Sharing Subterms (information) and Term Rewriting

Consider the following rules:

$$
\begin{array}{lcl}
f(a,b) & \rightarrow & c \\
a & \rightarrow & b
\end{array}
$$

Sharing does not preserve properties of tree (term) rewriting !

$f(a,a) \rightarrow f(a,b) \rightarrow c$

$$
\begin{tikzcd}
f \arrow[d, bend right] \arrow[d, bend left] \arrow[r] & f \arrow[d, bend right] \arrow[d, bend left] \arrow[r, "\not\rightarrow" description, phantom] & {} \\
a & b &
\end{tikzcd}
$$

[Plump 99] survey on rewriting with “dags”.

Motivation (continued)

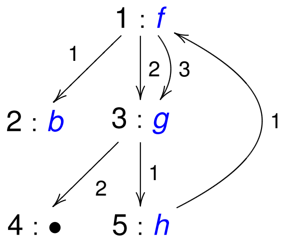
- Data-structure rewriting
  including cyclic data-structures with pointers
  such as circular lists, doubly-linked lists, etc.
- Data-structures are more complex than terms (Cycles, Sharing)
- Difficult to encode efficiently using terms
- Usually described by pointers ($\Rightarrow$ pointer rewriting)
- Formally described as term-graphs
  term-graphs = terms with cycles and sharing

Term-graphs

[Barendregt et al. 87]
[Plump 99, survey on *acyclic* term-graphs]

Let Ω be a set of operation symbols.
A *term-graph* t over Ω is defined by:

- a set of nodes N_t,
- a subset of labeled nodes $N_t^\Omega \subseteq N_t$,
- a labeling function $L_t : N_t^\Omega \to \Omega$,
- a successor function $S_t : N_t^\Omega \to N_t^*$,



Term-graphs

[Barendregt et al. 87]
[Plump 99, survey on *acyclic* term-graphs]

Let Ω be a set of operation symbols and $\mathcal{F}$ a set of feature symbols.
A *term-graph t* over Ω and $\mathcal{F}$ is defined by:

- a set of nodes N_t,
- a set of edges E_t
- a subset of labeled nodes $N_t^\Omega \subseteq N_t$,
- a node labeling function $L_t^n : N_t^\Omega \to \Omega$,
- an edge labeling function $L_t^e : E_t \to \mathcal{F}$
- a source function $\mathcal{S}_t : E_t \to N_t$,
- a target function $\mathcal{T}_t : E_t \to N_t$,

(Term-)Graph Rewriting

- Which graphs? (Term-Graphs)
- Which rules?
- Which rewrite relation?

Two main approaches

- Algorithmic approaches
- Algebraic approaches (DPO,SPO, ...)

Graph Transformation

- Handbook of Graph Grammars and Computing by Graph Transformation (World Scientific)
  - Vol 1: Foundations, ed. G. Rozenberg, 1997
  - Vol 2: Applications, Languages and Tools
    eds. H. Ehrig, G. Engels, H.-J. Kreowski and G. Rozenberg, 1999
  - Vol 3: Concurrency, Parallelism and Distribution
    eds. H. Ehrig, H.-J. Kreowski, U. Montanari and G. Rozenberg, 1999
- A Monograph in Theoretical Computer Science (An EATCS series)
  H. Ehrig, K. Ehrig, U. Prange, G. Taentzer: Fundamentals of Algebraic Graph Transformation. Springer-Verlag, 2006

Outline

Algorithmic approach

[Barendregt et al. 87]
Shape of a rule:

$$L \to R$$

where L and R are rooted term-graphs.
A rule can be defined as one graph together with two roots

$$(L + R, r_1, r_2)$$

where r_1 and r_2 are the roots of L and R respectively
Let ρ be the rule $(L + R, r_1, r_2)$
We say that G rewrites to H using the rule ρ if

- L matches a subgraph of G ($h : L \to G \mid_n$)
- (build phase) Construct graph $G_1 = G + h(R)$
- (redirection phase) $G_2 = [h(r_1) \gg h(r_2)]G_1$
- (garbage collection phase) $H = G_2 \mid_{\text{root}}$

A cumbersome definition, hard to deal with in practice!

Rewrite Rules with actions

Shape of a rewrite rule :

$$[L \mid C] \to R$$

- L is a term-graph pattern
- C is a node constraint, $\bigwedge_{i=1}^{n}(\alpha_i \not\approx \beta_i)$.
- R is a sequence of actions $a_1; a_2; \ldots; a_n$

Actions

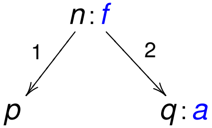
We consider three kinds of actions :

- Node definition $\alpha : f(\alpha_1, \ldots, \alpha_n)$
- Edge redirection $\alpha \gg_i \beta$
- Global redirection $\alpha \gg \beta$

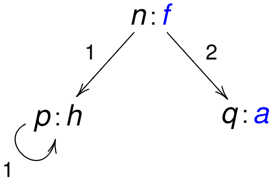
Application of actions

$a[t]$ denotes the application of action(s) a on the term-graph t

- Let $t = n : f(p, q : a)$



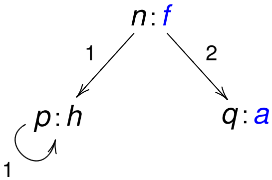
- Let $t_1 = p : h(p)[t] = n : f(p : h(p), q : a)$



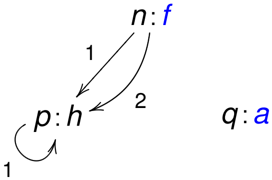
Application of actions

$a[t]$ denotes the application of action(s) a on the term-graph t

- Let $t_1 = p{:}h(p)[t] = n{:}f(p{:}h(p), q : a)$



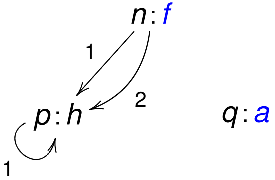
- Let $t_2 = n \gg_2 p[t_1] = n{:}f(p{:}h(p), p); q : a$



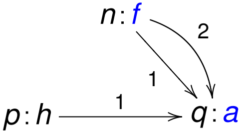
Application of actions

$a[t]$ denotes the application of action(s) a on the term-graph t

- Let $t_2 = n \gg_2 p[t_1] = n\!:\!f(p\!:\!h(p), p); q : a$



- Let $t_3 = p \gg q[t_2] = n\!:\!f(q, q); p\!:\!h(q)$



Rewrite Step

Let t be a term-graph

Let ρ be a rewrite rule $[L \mid C] \to R$

t rewrite to s at node α, $t \to_\alpha s$ iff:

- $\exists m : L \to t$ a homomorphism (ρ-matcher)
- $m(root_L) = \alpha$
- α is reachable from $root_t$
- $m(C)$ holds
- $s = m(R)[t]$

Term-Graph Rewrite Systems (tGRS) –Example–

Length of a circular list :

$r : length(p) \to r : length'(p, p)$

$r : length'(p_1 : cons(n, p_2), p_2) \to r : s(0)$

$[r : length'(p_1 : cons(n, p_2), p_3) \mid p_2 \not\approx p_3] \to r : s(q); q : length'(p_2, p_3)$

Remark: term rewrite systems are tGRS's.

Term-Graph Rewrite Systems –Example–

In-situ list reversal :

$o : reverse(p) \rightarrow o : rev(p, nil)$

$o : rev(p_1 : cons(n, nil), p_2) \rightarrow p_1 \gg_2 p_2; o \gg p_1$

$o : rev(p_1 : cons(n, p_2 : cons(m, p_3), p_4) \rightarrow p_1 \gg_2 p_4; o \gg_1 p_2; o \gg_2 p_1$

Visual Programming would help!

DPO approach of rewrite rules with actions

A categorical approach can be found in [TERMGRAPH 06, ENTCS07, RTA07]

$$
\begin{tikzcd}
L \arrow[d, "m"'] & K \arrow[l, "l"'] \arrow[r, "r"] \arrow[d, "d"] & R \arrow[d, "m'"] \\
G & D \arrow[l, "l'"'] \arrow[r, "r'"] & H
\end{tikzcd}
$$

Figure: Double pushout: a rewrite step ($G \to H$)

Redirections of edges (pointers) are handled by
$K = \mathit{disconnection}(L, E, N)$ and the morphisms l and r.
Remark: Morphisms l and r are not injective! D is not unique!

Confluence

$f(x) \to x$
$g(x) \to x$

The following term-graph

$$\begin{tikzcd} n : f \arrow[d] \\ q : g \arrow[u, bend right] \end{tikzcd}$$

rewrites to

$$\begin{tikzcd} n : f \arrow[loop above] & & q : g \arrow[loop above] \end{tikzcd}$$

Confluence

$$\alpha : f(\beta : c) \to \beta : a; \alpha \gg \beta$$

$$\alpha : g(\beta : c) \to \beta : b; \alpha \gg \beta$$

$$\begin{tikzcd}
p : f \arrow[dr] & & q : g \arrow[dl] \\
& q : c &
\end{tikzcd}$$

The label of node q may end as $q : a$ or $q : b$

Computing with non-confluent orthogonal Term-graph Rewrite Systems

How to evaluate the following term-graph ?

- $\mathit{addlast}(\mathit{length}(n : [1, 2]), n)$
- Two normal forms
  - $[1, 2, 2]$ (evaluate *addlast* after *length*)
  - $[1, 2, 3]$ (evaluate *length* after *addlast*)

Term-graphs with Priority

[PPDP06][RTA07][RTA08]

- Endow Term-graphs with priorities $(G, <_G)$ to express which node should be evaluated first
  - $m_1 : addlast(m_2 : length(n : [1, 2]), n); m_1 < m_2$
- Priorities should not be a total order (stay declarative)
- Which nodes should be ordered?
- Solution: Order only nodes producing a “side-effect”

Strategies and needed nodes

A strategy ϕ is a partial function which takes a rooted term-graph t and returns a node (position) n and a rule R,

$$\phi(t) = (n, R)$$

such that the term-graph t can be reduced at node n using the rule R,

$$t \rightarrow_n t'$$

Needed Nodes

Let ϕ be a rewrite strategy.
Let $\phi(t) = (p, R)$.
The node p is needed iff for all derivations

$$t \rightarrow_{\beta_1} t_1 \rightarrow_{\beta_2} \ldots t_{n-1} \rightarrow_{\beta_n} t_n$$

such that t_n is a value, there exists $i \in [1..n]$ s.t. $\beta_i = p$

Inductively sequential Term Rewrite Systems

- Constitute a subclass of TRSs for which efficient rewrite strategies are available [Antoy 92]
- Are as expressive as Strongly Sequential TRSs
- Are the basis of modern functional and logic programming languages.
- Are defined by means of data-structures called Definitional trees

Definitional Trees -case of terms-

Let $\mathcal{R}$ be the following TRS

`f(k,nil)` $\rightarrow$ `R1`

`f(0,cons(x,l))` $\rightarrow$ `R2`

`f(succ(n),cons(x,l))` $\rightarrow$ `R3`

A definitional tree of operator *f* is a hierarchical structure whose leaves are the rules defining *f*.

f(k, l)

  f(k, nil) $\rightarrow$ R1

  f(k, cons (x, u))

    f(0, cons (x,u)) $\rightarrow$ R2

    f(succ(y), cons (x,u)) $\rightarrow$ R3

Definitional trees
-case of term-graphs-

$$r : length'(p_1 : nil, p_2 : \bullet) \to rhs_1$$
$$r : length'(p_1 : cons(n : \bullet, p_2 : \bullet), p_2) \to rhs_2$$
$$[r : length'(p_1 : cons(n : \bullet, p_2 : \bullet), p_3 : \bullet) \mid p_2 \dot{\neq} p_3] \to rhs_3$$

A definitional tree T of the operation $length'$ is given bellow:

$r : length'(p_1 : \bullet, p_2 : \bullet)$

- $r : length'(p_1 : nil, p_2 : \bullet) \to rhs_1$
- $r : length'(p_1 : cons(n : \bullet, p_3 : \bullet), p_2 : \bullet)$
  - $r : length'(p_1 : cons(n : \bullet, p_2 : \bullet), p_2) \to rhs_2$
  - $[r : length'(p_1 : cons(n : \bullet, p_2 : \bullet), p_3 : \bullet) \mid p_2 \dot{\neq} p_3] \to rhs_3$

A Rewrite strategy ϕ

Consider the following definitional tree T of the operation g :

$r : g(p_1 : \bullet, p_2 : \bullet)$

$\quad r : g(p_1 : nil, p_2 : \bullet) \to rhs_1$

$\quad r : g(p_1 : cons(n : \bullet, p_3 : \bullet), p_2 : \bullet)$

$\quad\quad r : g(p_1 : cons(n : \bullet, p_2 : \bullet), p_2) \to rhs_2$

$\quad\quad [r : g(p_1 : cons(n : \bullet, p_2 : \bullet), p_3 : \bullet) \mid p_2 \dot{\neq} p_3] \to rhs_3$

$$
\begin{aligned}
&\phi(1 : g\ (2 : g(3 : g(nil, p), q), 4 : g(nil, o))) \\
&= \quad \phi(2 : g(3 : g(nil, p), q)) \\
&= \quad \phi(3 : g(nil, p)) \\
&= \quad (3, Rule1)
\end{aligned}
$$

Naive extension of TRS's

Contrary to term rewriting, Definitional trees are not enough to ensure the neededness of positions computed by the strategy ϕ, in the context of term-graph rewriting.

Proposition: Let $SP = \langle \Omega, \mathcal{R} \rangle$ be tGRS such that Ω is constructor-based and the rules of every defined operation are stored in a definitional tree. Let t be a rooted term-graph. Then,

1. if $\phi(t) = (p, R)$, the node p is not needed in general.
2. if $\phi(t)$ is not defined, g can still have a constructor normal form.

Counter-examples

$r : f(p : 0) \to r \gg p$ $\qquad$ $r : h(p : 0, q : succ(n : \bullet)) \to q \gg p$

$r : f(p : succ(p' : \bullet)) \to r \gg p$

Let $t =$

$$\begin{tikzcd}
n : succ \arrow[d] \\
r : succ \arrow[d] \\
\color{red}{p} : f \arrow[d] \\
q : succ \arrow[d] \\
s : h \arrow[d] \arrow[uuu, bend right] \\
u : 0
\end{tikzcd}$$

$\phi(t) = (p, r : f(p : succ(p' : \bullet)) \to r \gg p)$.

However, the node p is not needed in t.

Counter-examples

$r : g(p : 0) \to r \gg p \qquad\qquad r : h(p : 0, q : succ(n : \bullet)) \to q \gg p$

Let $t =$

$$
\begin{tikzcd}
n : succ \arrow[d] \\
r : succ \arrow[d] \\
p : g \arrow[d] \\
q : succ \arrow[d] \\
s : h \arrow[d] \arrow[uuu, bend right] \\
u : 0
\end{tikzcd}
$$

$\phi(t)$ is not defined!.

However, the term-graph t rewrites to $n : succ(u : 0)$.

Inductively Sequential Term-Graph Rewrite Systems

Let $SP = \langle \Omega, \mathcal{R} \rangle$ be a tGRS.
SP is called inductively sequential iff

- The rules of every defined operation can be stored in a definitional tree and
- for all rules $[L \mid C] \rightarrow r$ in $\mathcal{R}$, for all global (respectively, local) redirections of the form $p \gg q$ (respectively, $p \gg_i q$ for some i), occurring in the right-hand side r, $p = Root_L$.

Main Properties of Strategy Φ

In presence of Inductively Sequential Term-Graph Rewrite Systems

- The positions computed by Φ are needed
- Φ is c-normalizing
- Φ is c-hyper-normalizing
- Derivations computed by Φ have minimal length

Confluence

Inductively sequential tGRS are not confluent!

$f(p : \bullet, p) \to 0$
$[f(p : \bullet, q : \bullet) \mid p \neq q] \to 1$
$r : g(q : \bullet) \to r \gg q$

Let t =
$$\begin{tikzcd}
 & n : f \arrow[dl] \arrow[dr] & \\
p : g \arrow[rr] & & q : 0
\end{tikzcd}$$

There are two different derivations starting from t :

$t \to_n 1$
$t \to_p f(q : 0, q) \to_n 0$

Admissible term-graphs

[JICSLP98]
Ω is contructor-based, i.e. $\Omega = D \cup C$ and $D \cap C = \emptyset$
D is a set of defined operations
C is a set of constructors

A term-graph is admissible if none of its cycles includes a defined operation.

$n : succ(n)$ is an admissible term-graph
$n : +(n, n)$ and $n : tail(n)$ are not admissible

Admissible term-graphs

The set of admissible term-graphs is not closed under rewriting

$n : f(m) \to q : g(n); n \gg m$

Let $\Omega = D \cup C$ with $C = \{0, succ\}$ and $D = \{f, g\}$

$$n_1 : f(m_1 : 0) \to q_1 : g(q_1)$$

Admissible Inductively sequential Term-Graph Rewrite Systems

Let $SP = \langle \Omega, \mathcal{R} \rangle$ be an inductively sequential tGRS. SP is called admissible iff for all rules $[\pi \mid C] \rightarrow r$ in $\mathcal{R}$ the following conditions are satisfied

- for all global (respectively, local) redirections of the form $p \gg q$ (respectively, $p \gg_i q$ for some i), occurring in the right-hand side r, we have $p = Root_\pi$ and $q \neq Root_\pi$.
- for all actions of the form $\alpha : f(\beta_1, \ldots, \beta_n)$, for all $i \in 1..n$, $\beta_i \neq Root_\pi$
- the set of actions of the form $\alpha : f(\beta_1, \ldots, \beta_n)$, appearing in r, do not construct a cycle including a defined operation.
- Constraint C includes disequations of the form $p \dot{\neq} q$ where p and q are labeled by constructor symbols.

Admissible Inductively sequential Term-Graph Rewrite Systems

[ICGT08][JICSLP98]
In presence of Admissible Inductively sequential Term-Graph Rewrite Systems

- The set of admissible term-graphs is closed under the rewrite relation defined by admissible rules.
- Φ computes needed positions
- Admissible term-graphs admit unique normal forms

Narrowing

$$w_i \rightsquigarrow_\sigma w_{i+1} \text{ iff } \sigma(w_i) \rightarrow w_{i+1}$$

- Rewriting = Matching + Transformation
- Narrowing = Unification + Transformation

Narrowing
–Motivation–

- Automated deduction [Slagle 74] [Fay 79][Hullot 80]
- Functional and Logic Programming [Goguen and Meseguer 84, ...]
- Security verification [Meadows 89, ...]
- Reachability Analysis [Meseguer and Thati 05, ...]
- ...

Narrowing

Instantiate goal variables and apply a reduction step

$$\begin{array}{rcl} 0 + X & \to & X \\ s(X) + Y & \to & s(X + Y) \end{array}$$

$$\begin{array}{lll} U + s(0) = s(s(0)) & \rightsquigarrow_{\{U \mapsto s(V)\}} & s(V + s(0)) = s(s(0)) \\ & \rightsquigarrow_{\{V \mapsto 0\}} & s(s(0)) = s(s(0)) \end{array}$$

Computed answer: $\{U \mapsto s(0)\}$

Some Results

Needed Term narrowing [POPL04][JACM2000]
(main operational semantics of current functionalogic programming languages)
Needed Graph Narrowing [JICSLP98]
Needed Collapsing Narrowing [Gratra 2000]
Narrowing-based algorithm for data-structure rewriting [ICGT06]

- Goal
  $o : equal(p : length(q), s(s(0))) = true$

- Solution : a circular list of length two
  $[q : cons(n_1, r : cons(n_2, q)) \mid q \not\approx r]$

Narrowing: What do we transform?

Rule

$$o : f(p : a, q, r) \longrightarrow p : b; o \gg_3 q$$

Rewrite Steps

o_1, p_1, q_1 and r_1 are constants (names or addresses)

$$o_1 : f(p_1 : a, q_1 : a, r_1) \longrightarrow o_1 : f(p_1 : b, q_1 : a, q_1)$$

$$o_1 : f(p_1 : a, p_1, r_1) \longrightarrow o_1 : f(p_1 : b, p_1, p_1)$$

Narrowing: What do we transform?

Rule

$$o : f(p : a, q, r) \longrightarrow p : b; o \gg_3 q$$

Rewrite Steps (o_1, p_1, q_1 and r_1 are constants)

$o_1 : f(p_1 : a, q_1 : a, r_1) \longrightarrow o_1 : f(p_1 : b, q_1 : a, q_1)$

$o_1 : f(p_1 : a, p_1, r_1) \longrightarrow o_1 : f(p_1 : b, p_1, p_1)$

Narrowing steps (o_2, p_2, q_2 and r_2 are variables)

$$o_2 : f(p_2, q_2 : a, r_2) \rightsquigarrow ?$$

σ labels node p_2 with symbol a.

$o_2 : f(p_2, q_2 : a, r_2) \rightsquigarrow_\sigma o_2 : f(p_2 : b, q_2 : a, q_2) \mid p_2 \not\approx q_2$

$o_2 : f(p_2, q_2 : a, r_2) \rightsquigarrow_{\sigma \cup \{q_2 \mapsto p_2\}} o_2 : f(p_2 : b, p_2, p_2)$

Narrowing: What do we transform?

Rule

$$o : f(p : a, q, r) \longrightarrow p : b; o \gg_3 q$$

Narrowing steps

(o_2, p_2, q_2 and r_2 are variables)

$$\begin{aligned}
& o_2 : f(p_2, q_2 : a, r_2) \\
& \quad \rightsquigarrow_\sigma [apply(o_2 : f(p_2 : a, q_2 : a, r_2), p_2 : b; o_2 \gg_3 q_2)] \\
& \quad \rightsquigarrow [apply(o_2 : f(p_2 : a, q_2 : a, r_2), p_2 : b; o_2 \gg_3 q_2) \mid p_2 \not\approx q_2] \\
& \quad \rightsquigarrow [apply(o_2 : f(p_2 : b, q_2 : a, r_2), o_2 \gg_3 q_2) \mid p_2 \not\approx q_2] \\
& \quad \rightsquigarrow [apply(o_2 : f(p_2 : a, q_2 : a, q_2), \epsilon) \mid p_2 \not\approx q_2] \\
& \quad \rightsquigarrow [o_2 : f(p_2 : a, q_2 : a, q_2) \mid p_2 \not\approx q_2]
\end{aligned}$$

Narrowing: What do we transform?

Rule

$$o : f(p : a, q, r) \longrightarrow p : b; o \gg_3 q$$

Narrowing steps

o_2, p_2, q_2 and r_2 are variables

$$\begin{aligned}
& o_2 : f(p_2, q_2 : a, r_2) \\
& \quad \rightsquigarrow_\sigma [\mathit{apply}(o_2 : f(p_2 : a, q_2 : a, r_2), p_2 : b; o_2 \gg_3 q_2)] \\
& \quad \rightsquigarrow_{\{q_2 \mapsto p_2\}} [\mathit{apply}(o_2 : f(p_2 : a, p_2, r_2), p_2 : b; o_2 \gg_3 q_2)] \\
& \quad \rightsquigarrow [\mathit{apply}(o_2 : f(p_2 : b, p_2, r_2), o_2 \gg_3 q_2)] \\
& \quad \rightsquigarrow [\mathit{apply}(o_2 : f(p_2 : b, p_2, p_2), \epsilon)] \\
& \quad \rightsquigarrow o_2 : f(p_2 : b, p_2, p_2)
\end{aligned}$$

g-terms

Symbolic handling of actions
G is a term-graph
ϕ is a conjunction of disequations
τ is a sequence of actions

$$[G \mid \phi]$$

$$[apply(G, \tau) \mid \phi]$$

Example:

$$[o_2 : f(p_2 : a, q_2 : a, q_2) \mid p_2 \not\approx q_2]$$

$$[apply(o_2 : f(p_2 : b, q_2 : a, r_2), o_2 \gg_3 q_2) \mid p_2 \not\approx q_2]$$

Graph Narrowing Rules

Superposition rule (SUP)

$$[G \mid \psi]^{\tau} \rightsquigarrow_{SUP,\rho,\theta} [H \mid \psi']^{\sigma(\tau)}$$

If:

- G is a term-graph
- ρ is rewrite rule $[L \mid \phi] \to R$
- σ is a most general unifier of L and G such that:
  - $\sigma(L)$ and $\sigma(G)$ are compatible
  - The root of L unifies with a labeled node in G (non-variable unification)
- $H = apply(\sigma(G) \cup \sigma(L), \sigma(R))$
- $\theta = (\sigma, K)$ with $K = \sigma(L) \backslash \sigma(G)$
- $\psi' = \sigma(\psi) \wedge \sigma(\phi) \wedge \bigwedge_{p \in \text{affected\_by}(\sigma(\tau)), q \in K^{\Omega}} p \dot{\neq} q.$

Graph Narrowing Rules

Action rule: simplify (SIM)

$$[\mathit{apply}(G, \epsilon) \mid \psi]^{\tau} \rightsquigarrow_{\mathit{SIM}} [G \mid \psi]^{\tau}$$

Graph Narrowing Rules

Action rule: execute (EXE)

$$[\mathit{apply}(G, \alpha.u) \mid \psi]^{\tau} \rightsquigarrow_{\mathit{EXE}} [\mathit{apply}(\alpha[G], u) \mid \psi']^{\tau.\alpha}$$

If:

- action α is not a node creation and
- $[G \mid \psi]$ is ready for action α.

Graph Narrowing Rules

Action rule: new node (NEW)

$$[\mathit{apply}(G, n^+.u) \mid \psi]^\tau \rightsquigarrow_{\mathit{NEW},\sigma} [\mathit{apply}(n^+[G], \sigma(u)) \mid \psi']^{\tau.n^+}$$

If:

- $\sigma = \{n \mapsto n'\}$ where n' is a fresh effective node
- $\psi' = \psi \wedge \bigwedge_{p \in \mathcal{V} \cap N_G} (p \not\approx n')$

Graph Narrowing Rules

Isolation rule with equality (EQU)

$$[apply(G, \alpha.u) \mid \psi]^{\tau} \rightsquigarrow_{EQU,\sigma} [apply(\sigma(G), \sigma(\alpha).\sigma(u)) \mid \sigma(\psi)]^{\sigma(\tau)}$$

If:

- there exists a node $n \in \text{affected\_by}(\alpha)$,
- m is not an α-isolated node in G and
- σ is a substitution (compatible with G) such that $\sigma(n) = \sigma(m)$

$$\begin{aligned} o_2 : f(p_2, q_2 : a, r_2) & \\ &\rightsquigarrow_{\sigma} [apply(o_2 : f(p_2 : a, q_2 : a, r_2), p_2 : b; o_2 \gg_3 q_2)] \\ &\rightsquigarrow_{\{q_2 \mapsto p_2\}} [apply(o_2 : f(p_2 : a, p_2, r_2), p_2 : b; o_2 \gg_3 q_2)] \end{aligned}$$

Graph Narrowing Rules

Isolation rule with disequality (DIS)

$$[apply(G, \alpha.u) \mid \psi]^\tau \rightsquigarrow_{DIS} [apply(G, \alpha.u) \mid \psi \wedge n \dot{\neq} m]^\tau$$

If:

- there exists a node $n \in$ affected_by(α),
- m is not an α-isolated node in G

$$\begin{aligned} & o_2 : f(p_2, q_2 : a, r_2) \\ & \quad \rightsquigarrow_\sigma [apply(o_2 : f(p_2 : a, q_2 : a, r_2), p_2 : b; o_2 \gg_3 q_2)] \\ & \quad \rightsquigarrow [apply(o_2 : f(p_2 : a, q_2 : a, r_2), p_2 : b; o_2 \gg_3 q_2) \mid p_2 \not\approx q_2] \end{aligned}$$

Computed Solutions

$$[G_0 \mid \mathit{True}] \rightsquigarrow_{\sigma_1} \cdots \rightsquigarrow_{\sigma_n} [G_n \mid \phi]$$

Computed Solution is : $(\sigma_1 \cdots \sigma_n, \phi)$

Example:

- Goal
  $o : \mathit{equal}(p : \mathit{length}(q), s(s(0))) = \mathit{true}$

- Solution
  $[q : \mathit{cons}(n_1, r : \mathit{cons}(n_2, q)) \mid q \not\approx r]$

Graph Narrowing: Soundness and Completeness

Proposition 1: The proposed narrowing rules are sound.
If $[G \mid \mathit{True}] \rightsquigarrow_{\sigma} [H \mid \phi]$
then there exists a ground substitution θ satisfying ϕ such that:
$G\sigma\theta \longrightarrow^{*} H\theta$

Proposition 2: The proposed narrowing rules are complete.
If $G\sigma \longrightarrow^{*} H$, σ being irreducible.
Then, there exist two substitutions θ and γ and a term-graph G' such that :

- $[G \mid \mathit{True}] \rightsquigarrow^{*}{}_{\theta} [G' \mid \phi]$
- γ satifies ϕ
- $\sigma = \theta\gamma$
- $G'\gamma = H$

Modal Logic and Graph Transformation – motivations–

- Specify graph shapes (data-structures)
  - Circular list
  - Balanced tree

  Graph properties can be specified within several logics, such as :
  - Separation Logic,
  - Monadic second order logic,
  - Modal logics (e.g., LTL, CTL, μ-calculus, etc).

- Verification of graph transformation:
  - Invariant
  - Reachability

  - Need to define new logics able to specify rule application and graph transformation.

Dynamic Logic

- Agents
- Knowledge
- Actions

Evaluate a formula in a model $\Rightarrow$ Transform the considered model

A Modal Logic for Graph Rewriting

- $G \models \phi$
- $G!_{\mathcal{R}} \models \phi$ where $G!_{\mathcal{R}}$ is the normal form of G
- $G!_{\mathcal{R}} \models \phi$ iff $G \models [\mathcal{R}^*]\phi$

A Modal Logic for Graph Rewriting : $\mathcal{L}_{gr}$

Language

- Formulas :

$$\phi ::= p \mid \perp \mid \neg\phi \mid \phi \vee \phi \mid [\alpha]\phi$$

- Actions :

$$\alpha ::= a \mid \alpha^* \mid \alpha;\alpha \mid \alpha \vee \alpha \mid \text{modifiers.}$$

$[\alpha]\Phi$:"After performing" actions α, formula Φ holds.

Modifiers

- Add a new node
- Add a new label (to current node)
- remove a label from the current node
- Add the label “a” to the edges going from a Φ-node to a Ψ-node.
- ...
- Graph modifiers :
  - U
  - n
  - $\vec{n}$
  - $\phi?$
  - $(\omega :=_g \phi)$
  - $(\omega :=_l \phi)$
  - $(a + (\phi, \psi))$
  - $(a - (\phi, \psi))$

Modifiers
–Example–

$$[p :=_g \bot][p :=_l \top](p \land [a](\neg p \land q))$$

$$\begin{tikzcd} p \arrow[r, "a"] \arrow[d, "a"'] & p, q \\ q & \end{tikzcd} \quad \Rightarrow \quad \begin{tikzcd} \bullet \arrow[r, "a"] \arrow[d, "a"'] & q \\ q & \end{tikzcd} \quad \Rightarrow \quad \begin{tikzcd} p \arrow[r, "a"] \arrow[d, "a"'] & q \\ q & \end{tikzcd}$$

Modal Logic: $\mathcal{L}_{gr}$

Semantics (informally)

- $G^r \models p$
  iff p holds at node r
- $G^r \models [a]\varphi$
  iff $G^n \models \varphi$ for all nodes n such that the edge $r \xrightarrow{a} n \in G$.
- $G^r \models [\omega :=_g \bot][\omega :=_l \top]\, \varphi$
  iff $H^r \models \varphi$, H is obtained from G by tagging the node r by ω (ω does not hold outside node r).

Modal Logic: $\mathcal{L}_{gr}$

Semantics

- $G^r \models [a - (\phi, \psi)]\varphi$
  iff $H^r \models \varphi$, H is obtained from G by erasing the edges $n \xrightarrow{a} m$, such that $G^n \models \phi$ and $G^m \models \psi$.

- $G^r \models [a + (\phi, \psi)]\varphi$
  iff $H^r \models \varphi$, H is obtained from G by adding the edges $n \xrightarrow{a} m$, such that $G^n \models \phi$ and $G^m \models \psi$.

- $G^r \models [f?]\varphi$
  iff $G^r \models \varphi$ and f holds at node r.

- $G^r \models [\vec{nw}]\varphi$
  iff $H^{nw} \models \varphi$. H is obtained from G by adding a new node nw.

Examples of $\mathcal{L}_{gr}$ Specified Properties

- Class of all a-cycle-free rooted termgraphs.
  $[\omega :=_g \top][U][\omega :=_l \bot][a^+]\omega$
- Class of all a-circular rooted termgraphs
  $[\omega :=_g \bot][U][\omega :=_l \top]\langle a^+\rangle\omega$.
- Class of all (a, b)-binary rooted termgraphs
  $[\omega :=_g \bot][U][\omega :=_l \top][a][\pi :=_g \top][(a \cup b)^\star][\pi :=_l \bot][U](\omega \rightarrow [b][(a \cup b)^\star]\pi)$.
- Let $R_G(a) = \{(n_1, n_2)$: the edge $n_1 \xrightarrow{a} n_2 \in G\}$.
  $G \models [\omega :=_g \bot][U][\omega :=_l \top][a]\neg\omega$ iff $R_G(a)$ is irreflexive.
- Classes of circular lists, balanced trees, ...

Hamiltonian Graphs

The following formula expresses the existence of a Hamiltonian cycle.
α stands for $a_1 \cup \ldots \cup a_n$, where the a_i's are the possible features used in the graph ($\mathcal{F} = \{a_1, \ldots, a_n\}$):

$$\langle \omega :=_g \top; \pi :=_g \bot; \omega :=_l \bot; \pi :=_l \top; (\alpha; \omega?; \omega :=_l \bot)^\star \rangle (\pi \wedge [U]\neg\omega).$$

Decidability

- With * and without “nw” : the problem of “model checking” ($G \models \Phi$) is decidable.

Modal Logic $\mathcal{L}_{gr}$ and Graph Rewriting

Expressing pattern-matching in $\mathcal{L}_{gr}$

Proposition: Let G^r be a term-graph with root r (a distinguished node). There exists a *-free action α_G and a *-free formula ϕ_G such that for all finite rooted termgraphs $G'^{r'}$, $G'^{r'} \models \langle\alpha_G\rangle\phi_G$ iff there exists a graph homomorphism from G to $G'^{r'}$.

We define the action α_G and the formula ϕ_G as follows:

- $\beta_G = (\pi_0 :=_g \perp); \ldots; (\pi_{N-1} :=_g \perp)$, ($N$ being the number of nodes in G)
- for all non-negative integers i, if $i < N$ then $\gamma_G^i = (\neg\pi_0 \wedge \ldots \wedge \neg\pi_{i-1})?; (\pi_i :=_l \top); U$,
- $\alpha_G = \beta_G; \gamma_G^0; \ldots; \gamma_G^{N-1}$,

Modal Logic $\mathcal{L}_{gr}$ and Graph Rewriting

We define the formula ϕ_G as follows:

- for all non-negative integers i, if $i < N$ then $\psi^i_G =$ if $\mathcal{L}^n(i)$ is defined then $\langle U\rangle(\pi_i \wedge \mathcal{L}^n(i))$ else $\top$,
- for all non-negative integers i, j, if $i, j < N$ then $\chi^{i,j}_G =$ if there exists an edge $e \in \mathcal{E}$ such that $\mathcal{S}(e) = i$ and $\mathcal{T}(e) = j$ then $\langle U\rangle(\pi_i \wedge \langle\mathcal{L}^e(e)\rangle\pi_j)$ else $\top$,
- $\phi_G = \psi^0_G \wedge \ldots \wedge \psi^{N-1}_G \wedge \chi^{0,0}_G \wedge \ldots \wedge \chi^{N-1,N-1}_G$.

Modal Logic $\mathcal{L}_{gr}$ and Graph Rewriting

Actions representing the right-hand sides can be expressed by the following elementary formulas :

- Action $n : f(a_1 \Rightarrow n_1, \ldots, a_k \Rightarrow n_k)$
  $U; \pi_n?; (f :=_l \top); (a_1 + (\pi_n, \pi_{n_1})); \ldots; (a_k + (\pi_n, \pi_{n_k}))$.
- Action $n \gg_a m$
  $(a - (\pi_n, \top)); (a + (\pi_n, \pi_m))$.
- Action $n \gg m)$ (for a-edges)
  $(\lambda_a :=_g \bot); (\lambda_a :=_g \langle a \rangle \pi_n); (a - (\top, \pi_n)); (a + (\lambda_a, \pi_m))$.

Modal Logic $\mathcal{L}_{gr}$ and Graph Rewriting

- Firing a rule $\rho = L \rightarrow R$

$$\beta_\rho = \alpha_L ; \alpha_R$$

- Normal form of graph G^r satisfies φ: Let $\mathcal{R} = (\bigvee \beta_{\rho_i})$

$$G^r \models [\mathcal{R}^*]([\mathcal{R}]\bot \Rightarrow \varphi)$$

- Rule ρ preserves the property φ:

$$\models (\varphi \Rightarrow [\beta_\rho]\varphi)$$

Conclussion and perspectives

- Admissible termgraphs seem to be a good trade-off to ensure confluence and efficient strategies
- Cloning and algebraic approaches (sesqui-pushout)
- Narrowing
- Visual Programming and Termgraph Rewriting
- Proof Techniques
- Applications