On Data-Structure Rewriting

Rachid Echahed
LIG Lab, Grenoble
France

June, 2010
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 \((\text{variables, valuation})\)
- \( A = \) 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”
    \[ u_0 \rightarrow u_1 \rightarrow \ldots \rightarrow u_n \]
  ▶ A narrowing derivation:
    \( w_0 \) is the initial goal (to solve):
    \[ w_0 \rightsquigarrow_{\sigma_1} w_1 \ldots \rightsquigarrow_{\sigma_n} w_n \]

Where

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

\( w_i \) is an element of \( A \) with partial information
\( \sigma \) instantiates \( w_i \)
Motivation: Extension of Term Rewriting; sharing subterms

Function definitions by means of term rewrite rules

\[
0 + x \rightarrow x \\
succ(x) + y \rightarrow succ(x + y) \\
double(x) \rightarrow x + x
\]

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

\[
\begin{align*}
  f(a, b) & \rightarrow c \\
  a & \rightarrow b
\end{align*}
\]

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

\[
\begin{align*}
  f(a, a) & \rightarrow f(a, b) \rightarrow c
\end{align*}
\]

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

---

**Sharing Subterms (information) and Term Rewriting**
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 (⇒ **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 \( \Omega \) be a set of operation symbols. A *term-graph* \( t \) over \( \Omega \) is defined by:

- a set of nodes \( N_t \),
- a subset of labeled nodes \( N^\Omega_t \subseteq N_t \),
- a labeling function \( L_t : N^\Omega_t \rightarrow \Omega \),
- a successor function \( S_t : N^\Omega_t \rightarrow N_t^* \),
Term-graphs

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

Let $\Omega$ be a set of operation symbols and $\mathcal{F}$ a set of feature symbols.

A term-graph $t$ over $\Omega$ and $\mathcal{F}$ is defined by:

- a set of nodes $N_t$,
- a set of edges $E_t$,
- a subset of labeled nodes $N^\Omega_t \subseteq N_t$,
- a node labeling function $L^n_t : N^\Omega_t \to \Omega$,
- an edge labeling function $L^e_t : E_t \to \mathcal{F}$,
- a source function $S_t : E_t \to N_t$,
- a target function $T_t : E_t \to N_t$.  

(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 2: Applications, Languages and Tools
  - Vol 3: Concurrency, Parallelism and Distribution
- A Monograph in Theoretical Computer Science (An EATCS series)
Outline

Introduction

Motivations

Termgraph Rewrite Systems

Confluence and Rewrite Strategies

Narrowing

A Modal Logic for Graph Transformation

Conclusion
Algorithmic approach

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

\[ L \rightarrow R \]

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 \( \rho \) be the rule \((L + R, r_1, r_2)\)
We say that \( G \) rewrites to \( H \) using the rule \( \rho \) if

- **L matches** a subgraph of \( G \) \((h : L \rightarrow G \mid_n)\)
- (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:

\[ [L \mid C] \rightarrow R \]

- \( L \) is a term-graph pattern
- \( C \) is a node constraint, \( \bigwedge_{i=1}^{n} (\alpha_i \not\sim \beta_i) \).
- \( R \) is a sequence of actions \( a_1; a_2; \ldots; a_n \)
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 \( t = n \cdot f(p, q : a) \)

- Let \( t_1 = p \cdot h(p)[t] = n \cdot f(p \cdot h(p), q : a) \)
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 $\rho$ be a rewrite rule $[L \mid C] \rightarrow R$

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

- $\exists m : L \rightarrow t$ a homomorphism ($\rho$-matcher)
- $m(\text{root}_L) = \alpha$
- $\alpha$ is reachable from $\text{root}_t$
- $m(C)$ holds
- $s = m(R)[t]$
Term-Graph Rewrite Systems (tGRS)

–Example–

Length of a circular list:

\[ r : \text{length}(p) \rightarrow r : \text{length}'(p, p) \]

\[ r : \text{length}'(p_1 : \text{cons}(n, p_2), p_2) \rightarrow r : s(0) \]

\[ [r : \text{length}'(p_1 : \text{cons}(n, p_2), p_3) \mid p_2 \not\approx p_3] \rightarrow r : s(q); q : \text{length}'(p_2, p_3) \]

**Remark:** term rewrite systems are tGRS’s.
In-situ list reversal:

\[ o : reverse(p) \rightarrow o : \text{rev}(p, nil) \]

\[ o : \text{rev}(p_1 : \text{cons}(n, nil), p_2) \rightarrow p_1 \gg 2 p_2; o \gg p_1 \]

\[ o : \text{rev}(p_1 : \text{cons}(n, p_2 : \text{cons}(m, p_3), p_4) \rightarrow p_1 \gg 2 p_4; o \gg 1 p_2; o \gg 2 p_1 \]

Visual Programming would help!
DPO approach of rewrite rules with actions

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

\[
\begin{array}{c}
L & \xleftarrow{l} & K & \xrightarrow{r} & R \\
\downarrow{m} & & \downarrow{d} & & \downarrow{m'} \\
G & \xleftarrow{l'} & D & \xrightarrow{r'} & H
\end{array}
\]

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

Redirections of edges (pointers) are handled by 
\(K = \text{disconnection}(L, E, N)\) and the morphisms \(l\) and \(r\).

Remark: Morphisms \(l\) and \(r\) are not injective! \(D\) is not unique!
Confluence

\[ f(x) \rightarrow x \]
\[ g(x) \rightarrow x \]

The following term-graph

\[ n:f \]
\[ q:g \]

rewrites to

\[ n:f \]
\[ q:g \]
Confluence

\[
\alpha : f(\beta : c) \rightarrow \beta : a; \quad \alpha \gg \beta
\]

\[
\alpha : g(\beta : c) \rightarrow \beta : b; \quad \alpha \gg \beta
\]

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?

- \texttt{addlast(length(n : [1, 2]), n)}
- Two normal forms
  - \([1, 2, 2]\) (evaluate \texttt{addlast} after \texttt{length})
  - \([1, 2, 3]\) (evaluate \texttt{length} after \texttt{addlast})
Term-graphs with Priority

[PPDP06][RTA07][RTA08]

- Endow Term-graphs with priorities \((G, <_G)\) to express which node should be evaluated first
  - \(m_1 : \text{addlast}(m_2 : \text{length}(n : [1, 2]), n); m_1 < m_2\)
- 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 $\phi$ 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 $R$,

$$t \rightarrow_n t'$$
Let $\phi$ 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

\[
\begin{align*}
  f(k,\text{nil}) & \rightarrow R_1 \\
  f(0,\text{cons}(x, l)) & \rightarrow R_2 \\
  f(\text{succ}(n),\text{cons}(x, l)) & \rightarrow R_3
\end{align*}
\]

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

\[
\begin{align*}
  f(k, l) \\
  & \rightarrow R_1 \\
  & \rightarrow R_2 \\
  & \rightarrow R_3
\end{align*}
\]
Definitional trees
-case of term-graphs-

\[ r : \text{length}'(p_1 : \text{nil}, p_2 : \bullet) \rightarrow \text{rhs}_1 \]
\[ r : \text{length}'(p_1 : \text{cons}(n : \bullet, p_2 : \bullet), p_2) \rightarrow \text{rhs}_2 \]
\[ [r : \text{length}'(p_1 : \text{cons}(n : \bullet, p_2 : \bullet), p_3 : \bullet) \mid p_2 \neq p_3] \rightarrow \text{rhs}_3 \]

A definitional tree \( T \) of the operation \textit{length}' is given below:

\[ r : \text{length}'(p_1 : \bullet, p_2 : \bullet) \]
\[ r : \text{length}'(p_1 : \text{nil}, p_2 : \bullet) \rightarrow \text{rhs}_1 \]
\[ r : \text{length}'(p_1 : \text{cons}(n : \bullet, p_3 : \bullet), p_2 : \bullet) \]
\[ r : \text{length}'(p_1 : \text{cons}(n : \bullet, p_2 : \bullet), p_2) \rightarrow \text{rhs}_2 \]
\[ [r : \text{length}'(p_1 : \text{cons}(n : \bullet, p_2 : \bullet), p_3 : \bullet) \mid p_2 \neq p_3] \rightarrow \text{rhs}_3 \]
A Rewrite strategy $\phi$

Consider the following definitional tree $T$ of the operation $g$:

\[ r : g(p_1 : \bullet, p_2 : \bullet) \]
\[ r : g(p_1 : \text{nil}, p_2 : \bullet) \rightarrow \text{rhs}_1 \]
\[ r : g(p_1 : \text{cons}(n : \bullet, p_3 : \bullet), p_2 : \bullet) \]

\[ r : g(p_1 : \text{cons}(n : \bullet, p_2 : \bullet), p_2) \rightarrow \text{rhs}_2 \]
\[ [r : g(p_1 : \text{cons}(n : \bullet, p_2 : \bullet), p_3 : \bullet) \mid p_2 \neq p_3] \rightarrow \text{rhs}_3 \]

\[ \phi(1 : g \ (2 : g(3 : g(\text{nil}, p), q), 4 : g(\text{nil}, o))) \]
\[ = \phi(2 : g(3 : g(\text{nil}, p), q)) \]
\[ = \phi(3 : g(\text{nil}, p)) \]
\[ = (3, \text{Rule1}) \]
Contrary to term rewriting, Definitional trees are not enough to ensure the neededness of positions computed by the strategy \( \phi \), in the context of term-graph rewriting.

 Proposition: Let \( SP = \langle \Omega, R \rangle \) be tGRS such that \( \Omega \) 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

\[ r : f(p : 0) \to r \gg p \]
\[ r : f(p : \text{succ}(p' : \bullet)) \to r \gg p \]
\[ r : h(p : 0, q : \text{succ}(n : \bullet)) \to q \gg p \]

Let \( t = \)

\[ n : \text{succ} \]
\[ r : \text{succ} \]
\[ p : f \]
\[ q : \text{succ} \]
\[ s : h \]
\[ u : 0 \]

\[ \phi(t) = (p, r : f(p : \text{succ}(p' : \bullet)) \to r \gg p). \]

However, the node \( p \) is not needed in \( t \).
Counter-examples

\[ r : g(p : 0) \rightarrow r \gg p \]  
\[ r : h(p : 0, q : \text{succ}(n : \bullet)) \rightarrow q \gg p \]

Let \( t = n : \text{succ} \)

\[ \phi(t) \text{ is not defined!} \]

However, the term-graph \( t \) rewrites to \( n : \text{succ}(u : 0) \).
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 $[L | C] \rightarrow r$ 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 $r$, $p = \text{Root}_L$. 
Main Properties of Strategy $\Phi$

In presence of Inductively Sequential Term-Graph Rewrite Systems

- The positions computed by $\Phi$ are needed
- $\Phi$ is c-normalizing
- $\Phi$ is c-hyper-normalizing
- Derivations computed by $\Phi$ have minimal length
Inductively sequential tGRS are not confluent!

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

Let \( t = n : f \) \[ \begin{array}{c}
  p : g \\
  \rightarrow \\
  q : 0
\end{array} \]

There are two different derivations starting from \( t : \)

\[ t \rightarrow_n 1 \]
\[ t \rightarrow_p f(q : 0, q) \rightarrow_n 0 \]
Admissible term-graphs

[JICSLP98]
\( \Omega \) is constructor-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

\[ n: f(m) \rightarrow q: g(n); \ n \gg m \]

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

\[ n_1 : f(m_1 : 0) \rightarrow q_1 : g(q_1) \]
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 $p = \text{Root}_\pi$ and $q \neq \text{Root}_\pi$.
- for all actions of the form $\alpha : f(\beta_1, \ldots, \beta_n)$, for all $i \in 1..n$, $\beta_i \neq \text{Root}_\pi$
- 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 $p \neq q$ 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.
- $\Phi$ computes needed positions
- Admissible term-graphs admit unique normal forms
Narrowing

\[ w_i \xrightarrow{\sigma} w_{i+1} \text{ iff } \sigma(w_i) \rightarrow w_{i+1} \]

- 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

\[
0 + X \rightarrow X \\
\text{s}(X) + Y \rightarrow \text{s}(X + Y)
\]

\[
U + \text{s}(0) = \text{s}(\text{s}(0)) \quad \leadsto \{U \mapsto \text{s}(V)\} \quad \text{s}(V + \text{s}(0)) = \text{s}(\text{s}(0)) \\
\quad \leadsto \{V \mapsto 0\} 
\]

Computed answer: \(\{U \mapsto \text{s}(0)\}\)
Some Results

Needed Term narrowing [POPL04][JACM2000]
(main operational semantics of current functional logic
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

\[ o : f(p : a, q, r) \rightarrow p : b; o \gg_3 q \]

Rewrite Steps

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

\[ o_1 : f(p_1 : a, q_1 : a, r_1) \rightarrow o_1 : f(p_1 : b, q_1 : a, q_1) \]

\[ o_1 : f(p_1 : a, p_1, r_1) \rightarrow o_1 : f(p_1 : b, p_1, p_1) \]
Narrowing: What do we transform?

Rule

\[ o : f(p : a, q, r) \rightarrow p : b; o \gg_3 q \]

Rewrite Steps \((o_1, p_1, q_1 \text{ and } r_1 \text{ are constants})\)

\[ o_1 : f(p_1 : a, q_1 : a, r_1) \rightarrow o_1 : f(p_1 : b, q_1 : a, q_1) \]
\[ o_1 : f(p_1 : a, p_1, r_1) \rightarrow o_1 : f(p_1 : b, p_1, p_1) \]

Narrowing steps \((o_2, p_2, q_2 \text{ and } r_2 \text{ are variables})\)

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

\(\sigma\) 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) \rightarrow p : b; o \gg 3 q}

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

{o_2 : f(p_2, q_2 : a, r_2)}
\leadsto_o^\sigma [apply(o_2 : f(p_2 : a, q_2 : a, r_2), p_2 : b; o_2 \gg 3 q_2)]
\leadsto [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]
\leadsto [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]
\leadsto [apply(o_2 : f(p_2 : a, q_2 : a, q_2), \epsilon) \mid p_2 \not\approx q_2]
\leadsto [o_2 : f(p_2 : a, q_2 : a, q_2) \mid p_2 \not\approx q_2]
Narrowing: What do we transform?

Rule

\[ o : f(p : a, q, r) \rightarrow p : b; o \gg_{3} q \]

Narrowing steps

\( o_2, p_2, q_2 \) and \( r_2 \) are variables

\[ o_2 : f(p_2, q_2 : a, r_2) \]

\[ \leadsto_{\sigma} [\text{apply}(o_2 : f(p_2 : a, q_2 : a, r_2), p_2 : b; o_2 \gg_{3} q_2)] \]

\[ \leadsto\{q_2 \rightarrow p_2\} [\text{apply}(o_2 : f(p_2 : a, p_2, r_2), p_2 : b; o_2 \gg_{3} q_2)] \]

\[ \leadsto [\text{apply}(o_2 : f(p_2 : b, p_2, r_2), o_2 \gg_{3} q_2)] \]

\[ \leadsto [\text{apply}(o_2 : f(p_2 : b, p_2, p_2), \epsilon)] \]

\[ \leadsto o_2 : f(p_2 : b, p_2, p_2) \]
Symbolic handling of actions

$G$ is a term-graph

$\phi$ is a conjunction of disequations

$\tau$ 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)

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

If:

▷ \(G\) is a term-graph
▷ \(\rho\) is rewrite rule \([L \mid \phi] \rightarrow R\)
▷ \(\sigma\) 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 \(K = \sigma(L) \setminus \sigma(G)\)
▷ \(\psi' = \sigma(\psi) \land \sigma(\phi) \land \bigwedge_{p \in \text{affected by } (\sigma(\tau)), q \in K_\Omega} p \neq q\).
Graph Narrowing Rules

**Action rule: simplify (SIM)**

\[
[\text{apply}(G, \epsilon) \mid \psi]^{\tau} \sim_{SIM} [G \mid \psi]^{\tau}
\]
Action rule: execute (EXE)

\[
[\text{apply}(G, \alpha.u) | \psi]^T \rightsquigarrow_{\text{EXE}} [\text{apply}(\alpha[G], u) | \psi']^T.\alpha
\]

If:

- action \(\alpha\) is not a node creation and
- \([G | \psi]\) is ready for action \(\alpha\).
Graph Narrowing Rules

**Action rule: new node (NEW)**

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

If:

- \(\sigma = \{n \mapsto n'\}\) where \(n'\) is a fresh effective node
- \(\psi' = \psi \land \bigwedge_{p \in \mathcal{V} \cap N_G} (p \not\approx n')\)
Isolation rule with equality (EQU)

\[
[\text{apply}(G, \alpha.u) \mid \psi] \xrightarrow{EQU, \sigma} [\text{apply}(\sigma(G), \sigma(\alpha) \cdot \sigma(u)) \mid \sigma(\psi)]^{\sigma(\tau)}
\]

If:

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

\[
o_2 : f(p_2, q_2 : a, r_2) \\
\xrightarrow{\sigma} [\text{apply}(o_2 : f(p_2 : a, q_2 : a, r_2), p_2 : b; o_2 \gg 3 q_2)] \\
\xrightarrow{\{q_2 \mapsto p_2\}} [\text{apply}(o_2 : f(p_2 : a, p_2, r_2), p_2 : b; o_2 \gg 3 q_2)]
\]
Graph Narrowing Rules

Isolation rule with disequality (DIS)

$$[\text{apply}(G, \alpha.u) \mid \psi]^\top \leadsto_{\text{DIS}} [\text{apply}(G, \alpha.u) \mid \psi \land n \neq m]^\top$$

If:

- there exists a node $n \in \text{affected by}(\alpha)$,
- $m$ is not an $\alpha$-isolated node in $G$

$$o_2 : f(p_2, q_2 : a, r_2)$$

$$\leadsto_\sigma [\text{apply}(o_2 : f(p_2 : a, q_2 : a, r_2), p_2 : b; o_2 \gg_3 q_2)]$$

$$\leadsto [\text{apply}(o_2 : f(p_2 : a, q_2 : a, r_2), p_2 : b; o_2 \gg_3 q_2) \mid p_2 \not\approx q_2]$$
Computed Solutions

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

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

Example:

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

- **Solution**
  
  \([q : cons(n_1, r : cons(n_2, q)) \mid q \not\approx r]\)
Graph Narrowing: Soundness and Completeness

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

Proposition 2: The proposed narrowing rules are complete. If \(G\sigma \longrightarrow^* H\), \(\sigma\) being irreducible. Then, there exist two substitutions \(\theta\) and \(\gamma\) and a term-graph \(G'\) such that:

- \([G \mid True] \leadsto^* \theta [G' \mid \phi]\)
- \(\gamma\) satisfies \(\phi\)
- \(\sigma = \theta\gamma\)
- \(G'\gamma = H\)
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, $\mu$-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:
  \[ \phi ::= p \mid \perp \mid \neg \phi \mid \phi \lor \phi \mid [\alpha] \phi \]

- Actions:
  \[ \alpha ::= a \mid \alpha^* \mid \alpha; \alpha \mid \alpha \lor \alpha \mid \text{modifiers} \]

$[\alpha] \Phi$ : “After performing” actions $\alpha$, formula $\Phi$ 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–

\[ [p := g \perp] [p := l \top] (p \land [a](\neg p \land q)) \]

\[ \begin{align*}
 p & \xrightarrow{a} p, q & \Rightarrow & \bullet \xrightarrow{a} q & \Rightarrow & p \xrightarrow{a} q \\
 q & \downarrow a & q & \downarrow a & q & \downarrow a
\end{align*} \]
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 $r \overset{a}{\rightarrow} n \in G$.

- $G^r \models [\omega := g \bot][\omega := l \top] \varphi$
  - iff $H^r \models \varphi$, $H$ is obtained from $G$ by tagging the node $r$ by $\omega$
  - (\(\omega\) 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 $r$.

- $G^r \models [n\overrightarrow{w}]\varphi$ 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
  \[ [\omega := g \bot][U][\omega := l \top][a^+]\omega. \]

- Class of all $(a, b)$-binary rooted termgraphs
  \[ [\omega := g \bot][U][\omega := l \top][a][\pi := g \top][(a \cup b)^*][\pi := l \bot][U](\omega \rightarrow [b][(a \cup b)^*]_{\pi}). \]

- Let $R_G(a) = \{(n_1, n_2) : \text{the edge } n_1 \xrightarrow{a} n_2 \in G\}.$
  \[ G \models [\omega := g \bot][U][\omega := l \top][a][\neg \omega \text{ iff } R_G(a) \text{ is irreflexive.}] \]

- Classes of circular lists, balanced trees, ...
Hamiltonian Graphs

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

$$\langle \omega := g \top; \pi := g \bot; \omega := l \bot; \pi := l \top; (\alpha; \omega?; \omega := l \bot)^* \rangle$$

$$\left( \pi \land [U] \neg \omega \right).$$
Decidability

- With * and without “nw” : the problem of “model checking” 
  \((G \models \Phi)\) is decidable.
Expressing pattern-matching in $\mathcal{L}_{gr}$

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

We define the action $\alpha_G$ and the formula $\phi_G$ as follows:

- $\beta_G = (\pi_0 := g \bot); \ldots; (\pi_{N-1} := g \bot)$,  
  ($N$ being the number of nodes in $G$)
- for all non-negative integers $i$, if $i < N$ then $\gamma^i_G = (\neg \pi_0 \land \ldots \land \neg \pi_{i-1})?; (\pi_i := l \top); U$,
- $\alpha_G = \beta_G; \gamma^0_G; \ldots; \gamma^{N-1}_G$. 

Modal Logic $\mathcal{L}_{gr}$ and Graph Rewriting
We define the formula $\phi_G$ as follows:

- for all non-negative integers $i$, if $i < N$ then $\psi^i_G = \text{if } \mathcal{L}^n(i) \text{ is defined then } \langle U \rangle (\pi_i \land \mathcal{L}^n(i)) \text{ else } \top$,

- for all non-negative integers $i, j$, if $i, j < N$ then $\chi^{i,j}_G = \text{if there exists an edge } e \in \mathcal{E} \text{ such that } S(e) = i \text{ and } T(e) = j \text{ then } \langle U \rangle (\pi_i \land \langle \mathcal{L}^e(e) \rangle \pi_j) \text{ else } \top$,

- $\phi_G = \psi^0_G \land \ldots \land \psi^{N-1}_G \land \chi^{0,0}_G \land \ldots \land \chi^{N-1,N-1}_G$. 

Modal Logic $\mathcal{L}_{gr}$ and Graph Rewriting
Actions representing the right-hand sides can be expressed by the following elementary formulas:

- **Action** $\text{n : f(a}_1 \Rightarrow n_1, \ldots, a_k \Rightarrow n_k)$
  $U; \pi_n?; (f := l \top); (a_1 + (\pi_n, \pi_{n_1})); \ldots; (a_k + (\pi_n, \pi_{n_k}))$.  

- **Action** $\text{n \gg_a m}$
  $(a - (\pi_n, \top)); (a + (\pi_n, \pi_m))$.  

- **Action** $\text{n \gg m}$ (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 $G'$ satisfies $\varphi$: Let $\mathcal{R} = (\bigvee \beta_{\rho_i})$

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

- Rule $\rho$ preserves the property $\varphi$:

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