The graph rewriting calculus: confluence and expressiveness

Clara Bertolissi

LORIA

Soutenance de thèse

October 28, 2005

Term rewriting systems

A tool for reasoning about computation

- composed by a set of terms $\mathcal{T}$ and a set of rules $\mathcal{R}$
- use matching and substitutions for evaluation

Term rewriting systems

A tool for reasoning about computation

- composed by a set of terms $\mathcal{T}$ and a set of rules $\mathcal{R}$
- use matching and substitutions for evaluation

Modelling addition by means of rewrite rules:

$$\mathcal{R} = \begin{cases} R_0 : & 0 + x & \rightarrow & x \\ R_1 : & s(x) + y & \rightarrow & s(x+y) \end{cases}$$

Term reduction:

$$1+2 \ = \ s(0)+s(s(0)) \ \rightarrow_{R_1} \ s(0+s(s(0))) \ \rightarrow_{R_0} \ s(s(s(0))) \ = \ 3$$

λ-calculus

A calculus for modeling functionality

- functions are first-class citizens
- explicit application operator

$$(\lambda x.s\ x)\ (0 + s\ s\ 0) \quad \rightarrow_\beta \quad s\ (0 + s\ s\ 0)$$

λ-calculus

A calculus for modeling functionality

- functions are first-class citizens
- explicit application operator

$$(\lambda x.s\ x)\ (0 + s\ s\ 0) \quad \to_\beta \quad s\ (0 + s\ s\ 0)$$

Encoding of addition: $\lambda np.(\lambda fx.p\ f(n\ f\ x))$

Limits

Rewriting is nice, but

- the rewrite relation is difficult to control
- non-reducibility cannot be expressed syntactically

Lambda-calculus is great, but

- lacks of discrimination capabilities
- non trivial encoding of data

Higher-order rewriting

Combination of *TRS* and λ-calculus

- Algebraic extensions of λ-calculus [Breazu-Tannen, Gallier88] [Okada89]
- Term rewrite systems with abstraction [Klop80,Nipkow90,Wolfram93]

Higher-order rewriting

Combination of *TRS* and λ-calculus

- Algebraic extensions of λ-calculus
  [Breazu-Tannen, Gallier88] [Okada89]
- Term rewrite systems with abstraction
  [Klop80,Nipkow90,Wolfram93]

The Combinatory Reduction Systems (CRS) [Klop80]

The rewriting calculus [Cirstea,Kirchner00]

A higher-order calculus with more explicit features

- rules are first class objects
- application is explicit
- decision of redex reduction is explicit
- results are defined at the object level

The rewriting calculus [Cirstea,Kirchner00]

A higher-order calculus with more explicit features

- rules are first class objects
- application is explicit
- decision of redex reduction is explicit
- results are defined at the object level

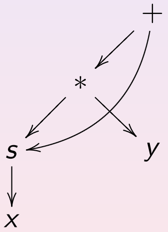
- expressiveness: λ-calculus, TRS[CLW03], objet calculi [CKL01], CRS [BCK03], ...
- extension with explicit substitutions: the ρ_x-calculus [CFK04]

From terms to term-graphs

improve efficiency

$\Rightarrow$ save space (sharing terms)

$\Rightarrow$ save time (reduce only once)



letrec $z = s(x)$ in $z * y + z$

From terms to term-graphs

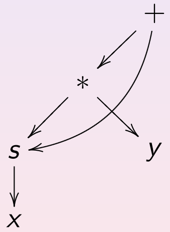
improve efficiency

⇒ save space (sharing terms)

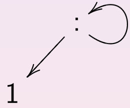
⇒ save time (reduce only once)

improve expressiveness

⇒ infinite regular data structures



letrec $z = s(x)$ in $z * y + z$

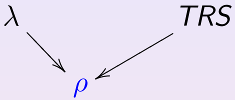


letrec $z = (1 : z)$ in z

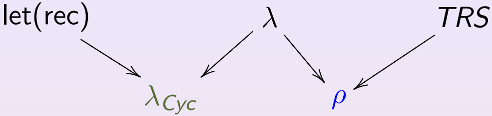
Term graph rewriting: different approaches

- implementation oriented approach *(pointers, redirections)* [Barendregt *et al.*87],[Plump98],[Kennaway94],...
- categorical approach *(push-out diagrams)* [CorradiniDrewes97],[Montanari,Corradini,Gadducci95], ...
- equational representation *(set of recursive equations)* [Ariola,Klop96], ...
  - Cyclic λ-calculus (λ_{Cyc}) [Ariola,Klop97]

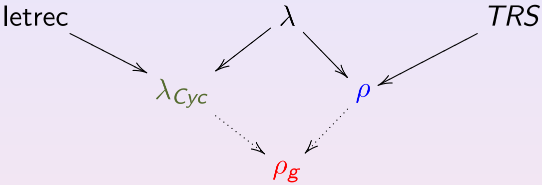
Towards a ρ-calculus for term graphs



Towards a ρ-calculus for term graphs



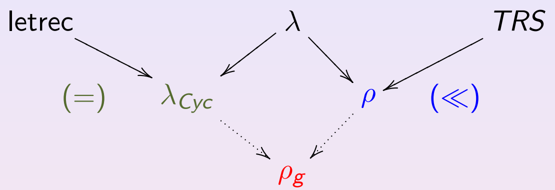
Towards a ρ-calculus for term graphs



➲ Aim: define a generalised calculus to deal with

- terms with sharing and cycles and pattern matching

Towards a ρ-calculus for term graphs



- Aim: define a generalised calculus to deal with
  - terms with sharing and cycles and pattern matching
- How: by means of
  - recursion equations and explicit matching constraints

Outline

ρ-calculus

ρ_g-calculus

- Syntax
- Semantics
- Properties
- Expressiveness

Conclusions

The ρ-calculus syntax

Terms	$\mathcal{T}$	$::=$	$\mathcal{X}$	(Variables)
		$\mid$	$\mathcal{K}$	(Constants)
		$\mid$	$\mathcal{T} \twoheadrightarrow \mathcal{T}$	(Abstraction)
		$\mid$	$\mathcal{T}\,\mathcal{T}$	(Application)
		$\mid$	$\mathcal{T} \wr \mathcal{T}$	(Structure)
		$\mid$	$\mathcal{T}[\mathcal{T} \ll \mathcal{T}]$	(Delayed matching constraint)

The ρ-calculus syntax

Terms	$\mathcal{T}$	$::=$	$\mathcal{X}$	(Variables)
		$\mid$	$\mathcal{K}$	(Constants)
		$\mid$	$\mathcal{T} \twoheadrightarrow \mathcal{T}$	(Abstraction)
		$\mid$	$\mathcal{T}\,\mathcal{T}$	(Application)
		$\mid$	$\mathcal{T} \wr \mathcal{T}$	(Structure)
		$\mid$	$\mathcal{T}[\mathcal{T} \ll \mathcal{T}]$	(Delayed matching constraint)

$f(x) \twoheadrightarrow x$	a standard rewrite rule
$(f(x) \twoheadrightarrow x)\ f(a)$	application of the rule $f(x) \twoheadrightarrow x$ to the term $f(a)$
$x[f(x) \ll f(a)]$	the term x constrained by a matching problem

The Reduction Semantics

$$(\rho) \quad (\mathcal{T}_1 \twoheadrightarrow \mathcal{T}_2)\mathcal{T}_3 \quad \mapsto_\rho \quad \mathcal{T}_2[\mathcal{T}_1 \ll \mathcal{T}_3]$$

$$(\sigma) \quad \mathcal{T}_2[\mathcal{T}_1 \ll \mathcal{T}_3] \quad \mapsto_\sigma \quad \sigma_{(\mathcal{T}_1 \prec\!\!\prec_\emptyset \mathcal{T}_3)}(\mathcal{T}_2)$$

$$(\delta) \quad (\mathcal{T}_1 \wr \mathcal{T}_2)\,\mathcal{T}_3 \quad \mapsto_\delta \quad \mathcal{T}_1\,\mathcal{T}_3 \wr \mathcal{T}_2\,\mathcal{T}_3$$

- (ρ) applying $\mathcal{T}_1 \twoheadrightarrow \mathcal{T}_2$ to $\mathcal{T}_3$ reduces to the delayed matching constraint $\mathcal{T}_2[\mathcal{T}_1 \ll \mathcal{T}_3]$
- (σ) computes $\mathcal{T}_1 \prec\!\!\prec_\emptyset \mathcal{T}_3$ and applies the result σ to the the term $\mathcal{T}_2$
- (δ) deals with the distributivity of the application on the structures built with the "$\wr$" constructor

Example of ρ-reduction

- $(x \rightarrow f(x))\ a \ \mapsto_\rho \ f(x)[x \ll a] \mapsto_\sigma f(a)$

Example of ρ-reduction

- $(x \rightharpoonup f(x))\ a \ \mapsto_\rho \ f(x)[x \ll a] \mapsto_\sigma f(a)$

- $(f(x,y) \rightharpoonup g(x,y))\ f(a,b) \mapsto_\rho \ g(x,y)[f(x,y) \ll f(a,b)]$
  $\mapsto_\sigma \{a/x, b/y\}g(x,y) \ = \ g(a,b)$

Example of ρ-reduction

- $(x \rightharpoonup f(x))\ a \ \mapsto_\rho\ f(x)[x \ll a] \mapsto_\sigma f(a)$
- $(f(x,y) \rightharpoonup g(x,y)))\ f(a,b) \mapsto_\rho\ g(x,y)[f(x,y) \ll f(a,b)]$
  $\mapsto_\sigma \{a/x, b/y\} g(x,y) \ = \ g(a,b)$
- $(f(a) \rightharpoonup a \wr f(a) \rightharpoonup b)\ f(a)$
  $\mapsto_\delta\ (f(a) \rightharpoonup a)\ f(a) \wr (f(a) \rightharpoonup b)\ f(a) \ \mapsto\!\!\!\rightarrow_{\rho\sigma}\ a \wr b$

The ρ-calculus syntax

Terms

$$
\begin{array}{llll}
\mathcal{T} & ::= & \mathcal{X} & \text{(Variables)} \\
 & \mid & \mathcal{K} & \text{(Constants)} \\
 & \mid & \mathcal{T} \twoheadrightarrow \mathcal{T} & \text{(Abstraction)} \\
 & \mid & \mathcal{T}\ \mathcal{T} & \text{(Application)} \\
 & \mid & \mathcal{T} \wr \mathcal{T} & \text{(Structure)} \\
 & \mid & \mathcal{T}[\mathcal{T} \ll \mathcal{T}] & \text{(Matching constraint)}
\end{array}
$$

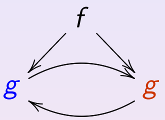
The ρ_g-calculus syntax [BBCK04]

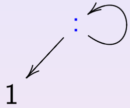
Terms	$\mathcal{G}$	$::=$	$\mathcal{X}$	(Variables)
		$\mid$	$\mathcal{K}$	(Constants)
		$\mid$	$\mathcal{G} \twoheadrightarrow \mathcal{G}$	(Abstraction)
		$\mid$	$\mathcal{G}\ \mathcal{G}$	(Application)
		$\mid$	$\mathcal{G} \wr \mathcal{G}$	(Structure)
		$\mid$	$\mathcal{G}\ [\mathcal{C}]$	(Constraint application)

Constraints	$\mathcal{C}$	$::=$	ϵ	(Empty constraint)
		$\mid$	$\mathcal{X} = \mathcal{G}$	(Recursion equation)
		$\mid$	$\mathcal{G} \ll \mathcal{G}$	(Match equation)
		$\mid$	$\mathcal{C}, \mathcal{C}$	(Conjunction of constraints)

where "," is *ACI* with neutral element ϵ.

Some ρ_g-terms

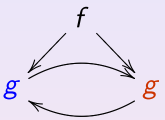




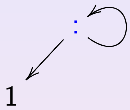
$f(x, y)\ [x = g(y), y = g(x)]$

$x\ [x = (1 : x)]$

Some ρ_{g}-terms

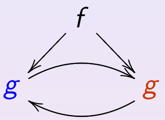


$$f(x,y)\ [x = g(y), y = g(x)]$$
$$\sim f(x,y)\ [y = g(x), x = g(y)]$$
$$\sim f(x,y)\ [y = g(x), x = g(y), \epsilon]$$

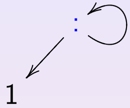


$$x\ [x = (1 : x)]$$

Some ρ_g-terms



$$f(x, y)\ [x = g(y), y = g(x)]$$
$$\sim f(x, y)\ [y = g(x), x = g(y)]$$
$$\sim f(x, y)\ [y = g(x), x = g(y), \epsilon]$$

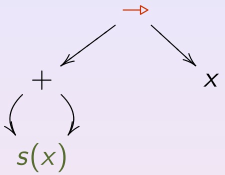


$$x\ [x = (1 : x)]$$

Remark:

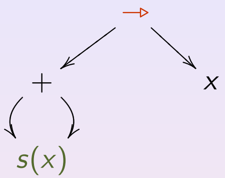
- we work on equivalence classes of terms.

Some ρ_g-terms: patterns



$$(y + y)\ [y = s(x)] \rightharpoondown s(x)$$

Some ρ_g-terms: patterns



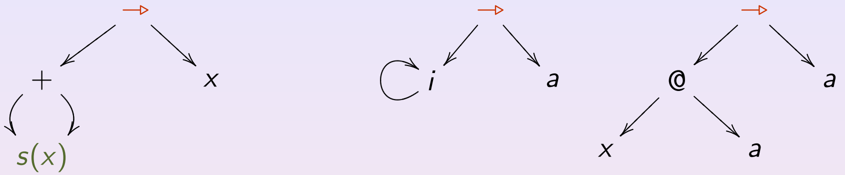
$$(y + y)\ [y = s(x)] \rightarrow s(x)$$

Remark:

- **patterns** are algebraic acyclic terms.

$$\mathcal{A} ::= \ \mathcal{X} \ \mid \ \mathcal{K} \ \mid \ (((f\ \mathcal{A})\ \mathcal{A})\ldots)\ \mathcal{A} \mid \ \mathcal{A}\ [\mathcal{X} = \mathcal{A}, \ldots, \mathcal{X} = \mathcal{A}]$$

Some ρ_g-terms: patterns



$$(y + y)\ [y = s(x)] \rightarrow s(x)$$

Remark:

- **patterns** are algebraic acyclic terms.

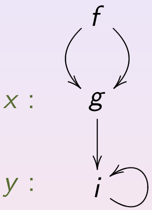
$$\mathcal{A} ::= \ \mathcal{X} \ \mid \ \mathcal{K} \ \mid \ (((f\ \mathcal{A})\ \mathcal{A}) \ldots)\ \mathcal{A} \mid \ \mathcal{A}\ [\mathcal{X} = \mathcal{A}, \ldots, \mathcal{X} = \mathcal{A}]$$

Graphical representation

- terms without constraints: trees

Graphical representation

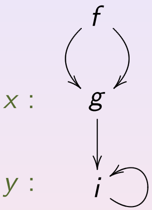
- terms without constraints: trees
- terms with recursion equations



$$f(x, x)\ [x = g(y), y = i(y)]$$

Graphical representation

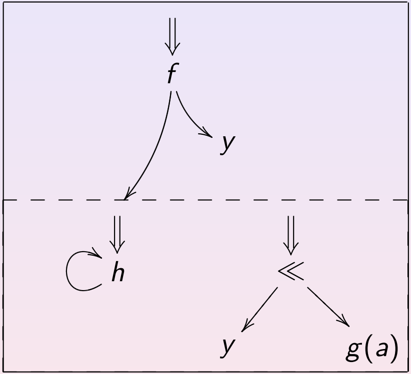
- terms without constraints: trees
- terms with recursion equations



$$f(x,x)\ [x = g(y), y = i(y)]$$

- terms with match equations ?

Graphical representation



$$f(x, y)\ [x = h(x), y \ll g(a)]$$

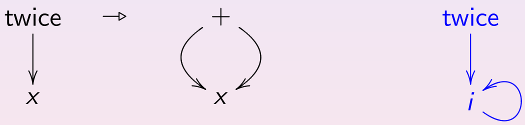
The main rules of the ρ_g-calculus semantics (1/3)

BASIC RULES:

$$(\rho) \quad (G_1 \rightarrow G_2)\ G_3 \quad \rightarrow_\rho \quad G_2\ [G_1 \ll G_3]$$

$$(\delta) \quad (G_1 \wr G_2)\ G_3 \quad \rightarrow_\delta \quad G_1\ G_3 \wr G_2\ G_3$$

Example:



$$(twice(x) \rightarrow x + x) \quad twice(z)\ [z = i(z)]$$

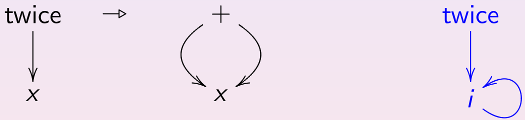
The main rules of the ρ_g-calculus semantics (1/3)

BASIC RULES:

$$(\rho) \quad (G_1 \rightarrow G_2)\ G_3 \quad \rightarrow_\rho \quad G_2\ [G_1 \ll G_3]$$

$$(\delta) \quad (G_1 \wr G_2)\ G_3 \quad \rightarrow_\delta \quad G_1\ G_3 \wr G_2\ G_3$$

Example:



$$(twice(x) \rightarrow x + x)\ \ twice(z)\ [z = i(z)]$$

$$\mapsto_\rho \quad x + x \quad [twice(x) \ll twice(z)\ [z = i(z)]]$$

The main rules of the ρ_g-calculus semantics (2/3)

BASIC RULES +

MATCHING RULES:

The main rules of the ρ_g-calculus semantics (2/3)

BASIC RULES +

MATCHING RULES:

propagate	$G_1 \ll (G_2\ [E_2])$	$\rightarrow_p$	$G_1 \ll G_2, E_2 \quad \text{if } G_1 \notin \mathcal{X}$
decomp	$K(G_1, \ldots, G_n) \ll K(G'_1, \ldots, G'_n)$	$\rightarrow_{dk}$	$G_1 \ll G'_1, \ldots, G_n \ll G'_n$
eliminate	$K \ll K, E$	$\rightarrow_e$	E
solved	$x \ll G, E$	$\rightarrow_s$	$x = G, E \quad \text{if } x \notin \mathcal{DV}(E)$

Example (continue):

$$(twice(x) \twoheadrightarrow x + x)\ \ twice(z)\ [z = i(z)]$$

$$\mapsto_\rho \quad x + x \quad [twice(x) \ll twice(z)\ [z = i(z)]]$$

The main rules of the ρ_{g}-calculus semantics (2/3)

Basic rules +

Matching rules:

propagate	$G_1 \ll (G_2\ [E_2])$	$\rightarrow_p$	$G_1 \ll G_2, E_2$	*if* $G_1 \notin \mathcal{X}$
decomp	$K(G_1,\ldots,G_n) \ll K(G'_1,\ldots,G'_n)$	$\rightarrow_{dk}$	$G_1 \ll G'_1,\ldots,G_n \ll G'_n$	
eliminate	$K \ll K, E$	$\rightarrow_e$	E	
solved	$x \ll G, E$	$\rightarrow_s$	$x = G, E$	*if* $x \notin \mathcal{DV}(E)$

Example (continue):

$$
\begin{array}{ll}
 & (twice(x) \twoheadrightarrow x + x)\ \ twice(z)\ [z = i(z)] \\
\mapsto_\rho & x + x\ \ [twice(x) \ll twice(z)\ [z = i(z)]] \\
\mapsto_p & x + x\ \ [twice(x) \ll twice(z),\ z = i(z)] \\
\mapsto_{dk} & x + x\ \ [x \ll z,\ z = i(z)] \\
\mapsto_s & x + x\ \ [x = z,\ z = i(z)]
\end{array}
$$

The main rules of the ρ_g-calculus semantics (3/3)

BASIC RULES + MATCHING RULES +

GRAPH RULES:

The main rules of the ρ_g-calculus semantics (3/3)

BASIC RULES + MATCHING RULES +

GRAPH RULES:

external sub	$\mathsf{Ctx}[y]\ [y = G, E]$	$\rightarrow_{es}$	$\mathsf{Ctx}[G]\ [y = G, E]$
acyclic sub	$G\ [G_0 \lll \mathsf{Ctx}[y], y = G_1, E]$	$\rightarrow_{ac}$	$G\ [G_0 \lll \mathsf{Ctx}[G_1], y = G_1, E]$ *where* $\lll \in \{=, \ll\}$
garbage	$G\ [E, x = G']$	$\rightarrow_{gc}$	$G\ [E]$ *if* $x \notin \mathcal{FV}(E) \cup \mathcal{FV}(G)$

Example:

$$(twice(x) \rightharpoonup x + x)\ \ twice(z)\ [z = i(z)]$$

$$\mapsto\!\!\!\rightarrow_{\rho g}\ \ x + x\ \ [x = z,\ z = i(z)]$$

The main rules of the ρ_g-calculus semantics (3/3)

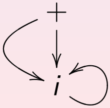
BASIC RULES + MATCHING RULES +

GRAPH RULES:

external sub	$\mathsf{Ctx}[y]\ [y = G, E]$	$\to_{es}$	$\mathsf{Ctx}[G]\ [y = G, E]$
acyclic sub	$G\ [G_0 \lll \mathsf{Ctx}[y], y = G_1, E]$	$\to_{ac}$	$G\ [G_0 \lll \mathsf{Ctx}[G_1], y = G_1, E]$ where $\lll \in \{=, \ll\}$
garbage	$G\ [E, x = G']$	$\to_{gc}$	$G\ [E]$ if $x \notin \mathcal{FV}(E) \cup \mathcal{FV}(G)$

Example:

$$(twice(x) \twoheadrightarrow x + x)\ \ twice(z)\ [z = i(z)]$$
$$\mapsto\!\!\!\!\to_{\rho g}\ x + x\ \ [x = z,\ z = i(z)]$$
$$\mapsto\!\!\!\!\to_{es}\ z + z\ \ [x = z,\ z = i(z)]$$
$$\mapsto\!\!\!\!\to_{gc}\ (z + z)\ [z = i(z)]$$



Sharing reduction strategy

Perform a step of reduction using (*external sub*) or (*acyclic sub*) if:

- it instantiates a variable in active position by an abstraction or a structure,

$$x\ a\ [x = f(x) \rightharpoonup x]$$

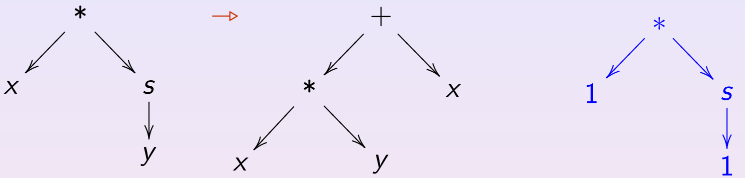
- or it instantiates a variable in a stuck match equation,

$$a\ [a \ll y, y = a]$$

- or it instantiates a variable by a variable.

$$z + z\ [z = x, x = 1]$$

Multiplication example: the ρ-reduction

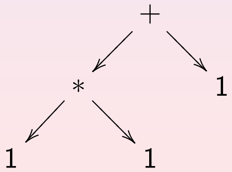


$$(x * s(y) \twoheadrightarrow x * y + x) \;\; 1 * s(1)$$

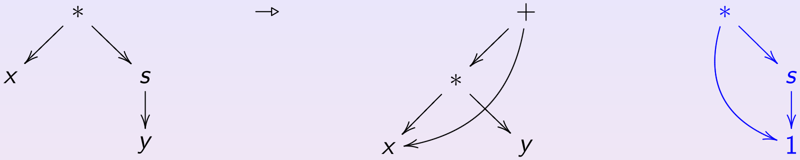
$$\mapsto_\rho \quad [x * s(y) \ll 1 * s(1)] \; (x * y + x)$$

$$\mapsto_\sigma \quad \{1/x, 1/y\}(x * y + x)$$

$$= \quad 1 * 1 + 1$$

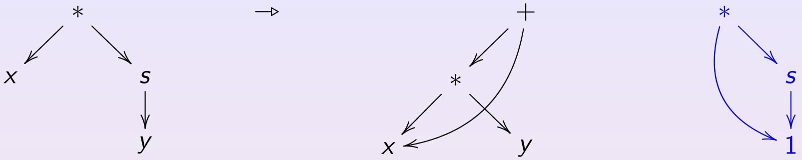


Multiplication in the ρ_g-calculus



$$(x * s(y) \to x * y + x) \quad z * s(z)\ [z = 1]$$

Multiplication in the ρ_g-calculus



$$(x * s(y) \mathrel{-\!\!\rhd} x * y + x) \quad z * s(z) \; [z = 1]$$

$$\mapsto_{\rho} \quad x * y + x \;\; [x * s(y) \ll z * s(z) \; [z = 1]]$$

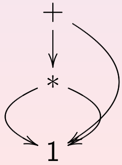
$$\mapsto_{p} \quad x * y + x \;\; [x * s(y) \ll z * s(z) \;, z = 1]$$

$$\mapsto\!\!\!\twoheadrightarrow_{dk} \quad x * y + x \;\; [x \ll z, y \ll z, \; z = 1]$$

$$\mapsto\!\!\!\twoheadrightarrow_{s} \quad x * y + x \;\; [x = z, y = z, \; z = 1]$$

$$\mapsto\!\!\!\twoheadrightarrow_{es} \quad (z * z + z) \;\; [x = z, y = z, z = 1]$$

$$\mapsto\!\!\!\twoheadrightarrow_{gc} \quad (z * z + z) \; [z = 1]$$



Matching example - Non-linearity

Success:

$$f(y,y) \ll f(a,a)$$

$$\mapsto_{dk} \quad y \ll a, y \ll a$$

$$= \quad y \ll a \quad (by\ idempotency)$$

$$\mapsto_{s} \quad y = a$$

Failure:

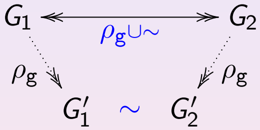
$$f(x,x) \ll f(a,b)$$

$$\mapsto_{dk} \quad x \ll a, x \ll b$$

The reduction is stuck: the condition $x \notin \mathcal{DV}(E)$ is not satisfied.

Confluence of the linear ρ_g-calculus [Ber05]

Any reductions starting from two joinable terms converge to two equivalent terms.



- Linearity: we restrict to a ρ_g-calculus with linear patterns.
- The congruence $\sim$ is induced by $AC1$, avoiding I.

Non triviality of the proof

- non termination of the system.
- reductions on equivalent classes of terms.
- need of adapting and combining existing techniques
  - properties of equational rewriting adapted to terms with constraints.
  - "finite developments method" of the classical λ-calculus.
  - Compatibility property:

$$
\begin{array}{ccc}
G_1 & \overset{\rho_{\mathrm{g}}}{\longmapsto} & G_2 \\
\wr & & \wr \\
G_1' & \overset{\rho_{\mathrm{g}}}{\longmapsto} & G_2'
\end{array}
$$

Proof sketch (1/2)

- the Σ-rules: (δ) $\cup$ (*external sub*) $\cup$ (*acyclic sub*)
- t he τ-rules: (ρ) $\cup$ Matching rules $\cup$ (*garbage*)

prove $CON_\sim$ for Σ

prove $CON_\sim$ for τ

Proof sketch (1/2)

- the Σ-rules: (δ) $\cup$ (*external sub*) $\cup$ (*acyclic sub*)
- the τ-rules: (ρ) $\cup$ MATCHING RULES $\cup$ (*garbage*)

prove $CON_{\sim}$ for Σ

prove $CON_{\sim}$ for τ

deduce $CON_{\sim}$ for $(\Sigma \cup \tau)$

Proof sketch (2/2)

1. $CON_{\sim}$ for τ: using *local confluence* and *termination* of the relation and the *compatibility* property
2. $CON_{\sim}$ for Σ: using the *finite developments* method of the λ-calculus adapted to Σ
3. $CON_{\sim}$ for $(\Sigma \cup \tau)$: using a *commutation* lemma for the two relations and the *compatibility* property

Proof sketch (2/2)

1. *CON*$_\sim$ for τ: using *local confluence* and *termination* of the relation and the *compatibility* property
2. *CON*$_\sim$ for Σ: using the *finite developments* method of the λ-calculus adapted to Σ
3. *CON*$_\sim$ for $(\Sigma \cup \tau)$: using a *commutation* lemma for the two relations and the *compatibility* property

Theorem: The linear ρ_{g}-calculus is *Church-Rosser* modulo $AC1$.

Expressiveness of the ρ_g-calculus

- Conservativity of the ρ_g-calculus vs ρ-calculus
- Conservativity of the ρ_g-calculus vs cyclic lambda
- Relationship with term graph rewriting

Conservativity of the ρ_{g}-calculus vs ρ-calculus

- Matching: Given a matching problem $T \ll U$ with T a linear ρ-term, and a substitution $\sigma = \{x_1/U_1, \ldots, x_n/U_n\}$.

$$\sigma(U) = T \quad \textit{if and only if} \quad T \ll U \mapsto\!\!\!\!\to_{\mathcal{M}} x_1 = U_1, \ldots, x_n = U_n$$

Conservativity of the ρ_g-calculus vs ρ-calculus

- Matching: Given a matching problem $T \ll U$ with T a linear ρ-term, and a substitution $\sigma = \{x_1/U_1, \ldots, x_n/U_n\}$.

  $$\sigma(U) = T \quad \textit{if and only if} \quad T \ll U \mapsto\!\!\!\!\rightarrow_{\mathcal{M}} x_1 = U_1, \ldots, x_n = U_n$$

- Completeness:
  If $T \mapsto\!\!\!\!\rightarrow_{\rho\sigma\delta} T'$ in the ρ-calculus then $T \mapsto\!\!\!\!\rightarrow_{\rho g} T'$ in the ρ_g-calculus.

- Soundness: Given a ρ-term T.
  If $T \mapsto\!\!\!\!\rightarrow_{\rho g} T'$ in the ρ_g-calculus and T' contains no constraints, then $T \mapsto\!\!\!\!\rightarrow_{\rho\sigma\delta} T'$ in the ρ-calculus.

Matching failures in ρ-calculus and ρ_{g}-calculus

ρ-calculus

$$(f(a) \twoheadrightarrow b)\ f(c)$$

$$\mapsto_{\rho} \quad b[f(a) \ll f(c)]$$

Matching failures in ρ-calculus and ρ_g-calculus

ρ-calculus

$$(f(a) \twoheadrightarrow b)\ f(c)$$
$$\mapsto_\rho \quad b[f(a) \ll f(c)]$$

ρ_g-calculus

$$(f(a) \twoheadrightarrow b)\ f(c)$$
$$\mapsto_\rho \quad b\ [f(a) \ll f(c)]$$
$$\mapsto_{dk} \quad b\ [a \ll c]$$

Conservativity of the ρ_{g}-calculus vs cyclic lambda

- Translation from a cyclic λ-term t to a ρ_g-term $[\![t]\!]$;
- Completeness:
  If $t_1 \mapsto\!\!\!\rightarrow_{\lambda c} t_2$ in the cyclic λ-calculus, then $[\![t_1]\!] \mapsto\!\!\!\rightarrow_{\rho g} [\![t_2]\!]$ in the ρ_{g}-calculus.
- Soundness:
  If $T_1 \mapsto\!\!\!\rightarrow_{\rho g} T_2$ in the ρ_{g}-calculus,
  with $T_1 = [\![t_1]\!]$ and T_2 without matching constraints,
  then we have $t_1 \mapsto\!\!\!\rightarrow_{\lambda c} t_2$ with $[\![t_2]\!] = T_2$.

ρ_g-calculus vs TGR

- Matching: the *Matching rules* well-behaves *w.r.t.* the notion of graph homomorphism

- Completeness: If $G_0 \mapsto\!\!\!\twoheadrightarrow G_n$ in a *TGR*, then there exist n ρ_g-terms $H_1, \ldots, H_n$, built from the *TGR* reduction, such that $(H_1 \ldots (H_n\ G_0)) \mapsto\!\!\!\twoheadrightarrow_{\rho g} G'_n$ with G'_n homomorphic to G_n

- Soundness:
  If $G_{\lceil (L \twoheadrightarrow R)\ G' \rceil} \mapsto\!\!\!\twoheadrightarrow_{\rho g} G_{\lceil H \rceil}$ with G, G', H, L, R term graphs and L linear, then $G[G'] \mapsto\!\!\!\twoheadrightarrow G[H']$ using the rule (L, R) in the *TGR*, with H' homomorphic to H.

General soundness *w.r.t.* TGR does not hold

Consider the ρ_g-term $f((a \twoheadrightarrow b)\, x, (a \twoheadrightarrow c)\, x)\ [x = a]$

General soundness *w.r.t.* TGR does not hold

Consider the ρ_g-term $\quad f((a \rightarrow b)\, x, (a \rightarrow c)\, x)\ [x = a]$

$$\longmapsto\!\!\!\rightarrow_{\rho g} \quad f(b, c)$$

General soundness *w.r.t.* TGR does not hold

Consider the ρ_g-term $f((a \twoheadrightarrow b)\, x, (a \twoheadrightarrow c)\, x)\ [x = a]$

$\mapsto_{\rho g}\ f(b, c)$

In a *TGR* we have no corresponding reduction

Conclusions

Expressive capabilities of the rewriting calculus:

- ρ-calculus and higher-order rewriting (CRSs)
- ρ-calculus with graph-like structures

ρ-calculus vs CRS

- Characterisation of CRS matching and all its solutions.
  - Treat CRS matching as λ-calculus higher-order matching
  - Translations from a CRS to simply typed λ-calculus and back
  - Completeness and correctness of the approach
    ➲ uniqueness and decidability of CRS pattern matching
- Encoding of CRS derivations into the ρ-calculus.
  - Translation function $[\![\_]\!]$
  - Preservation of matching solutions
  - Given a CRS-derivation $t_0 \mapsto\!\!\!\rightarrow_{\mathcal{R}} t_n$ there exists a ρ-term T, built from this derivation, such that any reduction of T terminates and converges to $[\![t_n]\!]$

ρ-calculus vs CRS: perspectives

- encoding a CRS in the ρ-calculus directly from its set of rewrite rules (following [CLW03])
- encoding the ρ-calculus into CRSs

Conclusions on the ρ_g-calculus

A generalisation of the cyclic λ-calculus with matching facilities

- representation of regular infinite entities
- higher-order capabilities
- explicit matching at the object-level

- Properties: Confluence of the linear ρ_g-calculus,
- Relation with other formalisms:
  - Conservativity *w.r.t.* the standard ρ-calculus and the cyclic λ-calculus
  - Simulation of first-order term-graph rewriting

Perspectives

- Matching: generalisation to cyclic left-hand sides
- Adequacy *w.r.t.* an infinitary version of the ρ-calculus
- Implementation in TOM (*http://tom.loria.fr*)
- Applications: semantic web, telecom network, bio-informatics, ...