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:

$$\mathcal{R} = \left\{ \begin{array}{c}
R_0 : \quad 0 + x \rightarrow x \\
R_1 : \quad s(x) + y \rightarrow s(x + y)
\end{array} \right\}$$

Term reduction:

$$1 + 2 = s(0) + s(s(0)) \rightarrow_{R_1} s(0 + s(s(0))) \rightarrow_{R_0} s(s(s(0))) = 3$$
\[ (\lambda x. s \ x) \ (0 + s \ s \ 0) \quad \rightarrow_{\beta} \quad s \ (0 + s \ s \ 0) \]
\(\lambda\)-calculus

A calculus for modeling functionality

- functions are first-class citizens
- explicit application operator

\[(\lambda x.s\ x)\ (0 + s\ s\ 0) \rightarrow_\beta s\ (0 + s\ s\ 0)\]

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

- Algebraic extensions of $\lambda$-calculus
  [Breazu-Tannen, Gallier88] [Okada89]

- Term rewrite systems with abstraction
  [Klop80,Nipkow90,Wolfram93]
Higher-order rewriting

Combination of $TRS$ and $\lambda$-calculus

- Algebraic extensions of $\lambda$-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: $\lambda$-calculus, TRS[CLW03], objet calculi [CKL01], CRS [BCK03], ...

- extension with explicit substitutions: the $\rho_x$-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
Term graph rewriting: different approaches

- **implementation oriented approach** (*pointers, redirections*)
  [Barendregt *et al.* 87], [Plump 98], [Kennaway 94], . . .

- **categorical approach** (*push-out diagrams*)
  [CorradiniDrewes 97], [Montanari,Corradini,Gadducci 95], . . .

- **equational representation** (*set of recursive equations*)
  [Ariola,Klop 96], . . .
  - **Cyclic λ-calculus** (*$\lambda_{Cyc}$*) [Ariola,Klop 97]
Towards a $\rho$-calculus for term graphs
Towards a $\rho$-calculus for term graphs
Towards a $\rho$-calculus for term graphs

Aim: define a generalised calculus to deal with
- terms with sharing and cycles and pattern matching
Towards a $\rho$-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

*p-calculus*

*_ρ*-calculus

*ρ*-calculus

*ρg*-calculus

Syntax
Semantics
Properties
Expressiveness

Conclusions
The $\rho$-calculus syntax

**Terms**

$T ::= X$ (Variables)

<table>
<thead>
<tr>
<th>$K$ (Constants)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T \rightarrow T$ (Abstraction)</td>
</tr>
<tr>
<td>$T T$ (Application)</td>
</tr>
<tr>
<td>$T \triangleleft T$ (Structure)</td>
</tr>
<tr>
<td>$T[T \ll T]$ (Delayed matching constraint)</td>
</tr>
</tbody>
</table>
The $\rho$-calculus syntax

**Terms**  
$\mathcal{T} ::= \mathcal{X}$ (Variables)  
| $\mathcal{K}$ (Constants)  
| $\mathcal{T} \rightarrow \mathcal{T}$ (Abstraction)  
| $\mathcal{T} \mathcal{T}$ (Application)  
| $\mathcal{T} \triangleleft \mathcal{T}$ (Structure)  
| $\mathcal{T}[\mathcal{T} \triangleleft \mathcal{T}]$ (Delayed matching constraint)

$f(x) \rightarrow x$  
a standard rewrite rule

$(f(x) \rightarrow x) f(a)$  
an application of the rule $f(x) \rightarrow x$ to the term $f(a)$

$x[f(x) \triangleleft f(a)]$  
the term $x$ constrained by a matching problem
The Reduction Semantics

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

\[(\sigma) \quad \mathcal{T}_2[\mathcal{T}_1 \ll \mathcal{T}_3] \quad \mapsto_{\sigma} \quad \sigma(\mathcal{T}_1 \Leftrightarrow_{\emptyset} \mathcal{T}_3)(\mathcal{T}_2)\]

\[(\delta) \quad (\mathcal{T}_1 \mathrel{\wedge} \mathcal{T}_2)\mathcal{T}_3 \quad \mapsto_{\delta} \quad \mathcal{T}_1 \mathcal{T}_3 \mathrel{\wedge} \mathcal{T}_2 \mathcal{T}_3\]

- \((\rho)\) applying \(\mathcal{T}_1 \rightarrow \mathcal{T}_2\) to \(\mathcal{T}_3\) reduces to the delayed matching constraint \(\mathcal{T}_2[\mathcal{T}_1 \ll \mathcal{T}_3]\)
- \((\sigma)\) computes \(\mathcal{T}_1 \Leftrightarrow_{\emptyset} \mathcal{T}_3\) and applies the result \(\sigma\) to the term \(\mathcal{T}_2\)
- \((\delta)\) deals with the distributivity of the application on the structures built with the “\(\wedge\)” constructor
Example of $\rho$-reduction

$(x \mapsto f(x)) \ a \ \mapsto_{\rho} \ f(x)[x \ll a] \ \mapsto_{\sigma} \ f(a)$
Example of $\rho$-reduction

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

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

\begin{itemize}
\item $\left(x \rightarrow f(x)\right) a \xrightarrow{\rho} f(x)[x \ll a] \xrightarrow{\sigma} f(a)$

\item $\left(f(x, y) \rightarrow g(x, y)\right) f(a, b) \xrightarrow{\rho} g(x, y)[f(x, y) \ll f(a, b)]$
\hspace{1em} $\xrightarrow{\sigma} \{a/x, b/y\} g(x, y) = g(a, b)$

\item $\left(f(a) \rightarrow a \land f(a) \rightarrow b\right) f(a)$
\hspace{1em} $\xrightarrow{\delta} (f(a) \rightarrow a) f(a) \land (f(a) \rightarrow b) f(a) \xrightarrow{\rho \sigma} a \land b$
\end{itemize}
The ρ-calculus syntax

Terms

\[ T ::= \mathcal{X} \quad \text{(Variables)} \]

| \mathcal{K} \quad \text{(Constants)} |
| \rightarrow \rightarrow \text{(Abstraction)} |
| \rightarrow \rightarrow \text{(Application)} |
| \rightarrow \rightarrow \text{(Structure)} |
| \rightarrow \rightarrow \text{(Matching constraint)} |

Clara Bertolissi
The graph rewriting calculus: confluence and expressiveness
The $\rho_g$-calculus syntax [BBCK04]

**Terms**

$G ::= X$ \hspace{0.5cm} (Variables)

$\mid K$ \hspace{0.5cm} (Constants)

$\mid G \rightarrow G$ \hspace{0.5cm} (Abstraction)

$\mid GG$ \hspace{0.5cm} (Application)

$\mid G \triangleleft G$ \hspace{0.5cm} (Structure)

$\mid G [C]$ \hspace{0.5cm} (Constraint application)

**Constraints**

$C ::= \epsilon$ \hspace{0.5cm} (Empty constraint)

$\mid X = G$ \hspace{0.5cm} (Recursion equation)

$\mid G \ll G$ \hspace{0.5cm} (Match equation)

$\mid C, C$ \hspace{0.5cm} (Conjunction of constraints)

where “,” is ACI with neutral element $\epsilon$. 
Some $\rho_g$-terms

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

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

Remark: ▶ we work on equivalence classes of terms.
Some $\rho_g$-terms

\[
\begin{align*}
 f(x, y) \ [x = g(y), y = g(x)] \\
 \sim f(x, y) \ [y = g(x), x = g(y)] \\
 \sim f(x, y) \ [y = g(x), x = g(y), \epsilon]
\end{align*}
\]

Remark: we work on equivalence classes of terms.

\[ x \ [x = (1:x)] \]
Some $\rho_g$-terms

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

Remark:

► we work on equivalence classes of terms.
Some $\rho_g$-terms: patterns

$\begin{array}{c}
\rightarrow \\
+ \\
\text{x}
\end{array} \\
\begin{array}{c}
\leftarrow \\
s(x)
\end{array}
$  

$(y + y) \ [y = s(x)] \rightarrow s(x)$
Some $\rho_g$-terms: patterns

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

Remark:
- patterns are algebraic acyclic terms.

$$\mathcal{A} ::= \mathcal{X} \mid \mathcal{K} \mid (((f \ A) \ A) \ldots) \ A \mid \ A [\mathcal{X} = A, \ldots, \mathcal{X} = A]$$
Some $\rho_g$-terms: patterns

$\xrightarrow{\rightarrow}$

$\xrightarrow{\rightarrow}$

$\xrightarrow{\rightarrow}$

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 $\rho_g$-calculus semantics  

**Basic rules:**

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

\[(\delta) \quad (G_1 \bowtie G_2) G_3 \rightarrow_\delta G_1 G_3 \bowtie G_2 G_3\]

Example:

\[
\text{twice} \rightarrow \begin{align*}
\downarrow & \quad + \quad \downarrow \\
\downarrow & \quad \downarrow
\end{align*}
\]

\[
(twice(x) \rightarrow x + x) \quad twice(z) [z = i(z)]
\]
The main rules of the $\rho_g$-calculus semantics (1/3)

**Basic rules:**

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

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

Example:

\[\begin{align*}
\text{twice} & \rightarrow \\
\downarrow x & \quad + \\
\downarrow x & \quad \downarrow x \\
\text{twice} & \quad \text{i} \\
\rightarrow_\rho x + x & \quad [\text{twice}(x) \ll \text{twice}(z) [z = i(z)]]
\end{align*}\]
The main rules of the $\rho_g$-calculus semantics (2/3)

**Basic rules**

**Matching rules:**

Basic rules +

Example (continue):

$\Rightarrow_{\rho} \frac{\text{twice}(x)}{\frac{\text{twice}(z)}{z = i(z)}}$
The main rules of the $\rho_g$-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):

$\rho$-calculus rule:

$\quad \quad \quad \quad \quad \quad \quad \quad \quad (\text{twice}(x) \rightarrow x + x) \quad \text{twice}(z) \begin{array}{c} [z = i(z)] \end{array}$

$\quad \quad \quad \quad \quad \quad \quad \quad \quad \rightarrow_{\rho} x + x \quad \begin{array}{c} \text{twice}(x) \ll \text{twice}(z) \begin{array}{c} [z = i(z)] \end{array} \end{array}$
The main rules of the $\rho_g$-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 \( G_1 \not\in \mathcal{X} \)

**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 \( x \not\in \text{DV}(E) \)

Example (continue):

\[
\begin{align*}
(twice(x) \rightarrow x + x) & \quad twice(z) [z = i(z)] \\
\rightarrow_{\rho} & \quad x + x & [twice(x) \ll twice(z) [z = i(z)]] \\
\rightarrow_{\rho} & \quad x + x & [twice(x) \ll twice(z), z = i(z)] \\
\rightarrow_{dk} & \quad x + x & [x \ll z, z = i(z)] \\
\rightarrow_{s} & \quad x + x & [x = z, z = i(z)]
\end{align*}
\]
The main rules of the $\rho_g$-calculus semantics (3/3)

Basic rules + Matching rules +

Graph rules:

Example:

\[
\frac{\text{twice}(x) \rightarrow \triangle x + x)}{\rho_g \frac{x + x}{x = z, z = i(z)}} \rightarrow \rho_g \frac{z + z}{z = i(z)} \rightarrow \rho_g (z + z) \frac{z = i(z)}{\downarrow \downarrow \downarrow i \downarrow \downarrow}
\]
The main rules of the $\rho_g$-calculus semantics (3/3)

**Basic rules + Matching rules +**

**Graph rules:**

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

**Example:**

- $\rho_g$ $(\text{twice}(x) \rightarrow x + x) \text{twice}(z) [z = i(z)]$
- $\rho_g$ $x + x [x = z, z = i(z)]$
The main rules of the $\rho_g$-calculus semantics (3/3)

**Basic rules + Matching rules +**

**Graph rules:**

- **external sub** $\text{Ctx}[y] [y = G, E] \to_{\text{es}} \text{Ctx}[G] [y = G, E]$
- **acyclic sub** $G [G_0 \lll \text{Ctx}[y], y = G_1, E] \to_{\text{ac}} G [G_0 \lll \text{Ctx}[G_1], y = G_1, E]$
  where $\lll \in \{=, \lll\}$
- **garbage** $G [E, x = G'] \to_{\text{gc}} G [E]$
  if $x \not\in \mathcal{FV}(E) \cup \mathcal{FV}(G)$

Example:

$$
(\text{twice}(x) \to x + x) \xrightarrow{\rho_g} x + x [x = z, z = i(z)]
\xrightarrow{\rho_g} z + z [x = z, z = i(z)]
\xrightarrow{\text{es}} z + z [x = z, z = i(z)]
\xrightarrow{\text{gc}} (z + z) [z = i(z)]
$$

$+$

$\downarrow$

$i$
Sharing reduction strategy

Perform a step of reduction using \textit{(external sub)} or \textit{(acyclic sub)} if:

- it instantiates a variable in active position by an abstraction or a structure,
  \[ x \ a \ [x = f(x) \rightarrow x] \]

- 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 $\rho$-reduction

\[(x * s(y) \rightarrow x * y + x) \quad 1 * s(1)\]

$\rightarrow_{\rho} \quad [x * s(y) \ll 1 * s(1)] \quad (x * y + x)$

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

$= \quad 1 * 1 + 1$
Multiplication in the $\rho_g$-calculus

\[(x \ast s(y) \rightarrow x \ast y + x) \ast s(z) [z = 1]\]
Multiplication in the $\rho_g$-calculus

\[
\begin{align*}
(x \ast s(y) & \rightarrow x \ast y + x) \\
\rightarrow_{\rho} & x \ast y + x [x \ast s(y) \ll z \ast s(z) [z = 1]] \\
\rightarrow_{p} & x \ast y + x [x \ll z, y \ll z, z = 1] \\
\rightarrow_{dk} & x \ast y + x [x = z, y = z, z = 1] \\
\rightarrow_{s} & x \ast y + x [x = z, y = z, z = 1] \\
\rightarrow_{es} & (z \ast z + z) [x = z, y = z, z = 1] \\
\rightarrow_{gc} & (z \ast z + z) [z = 1]
\end{align*}
\]
Matching example - Non-linearity

Success:

\[ f(y, y) \ll f(a, a) \]

\[ \mapsto_{dk} y \ll a, y \ll a \]

\[ = y \ll a \quad (\text{by idempotency}) \]

\[ \mapsto_{s} y = a \]

Failure:

\[ f(x, x) \ll f(a, b) \]

\[ \mapsto_{dk} x \ll a, x \ll b \]

The reduction is stuck: the condition \( x \notin \mathcal{DV}(E) \) is not satisfied.
Confluence of the linear $\rho_g$-calculus [Ber05]

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

$$G_1 \xleftarrow{\rho_g \cup \sim} G_2$$

$G_1' \sim G_2'$

- **Linearity**: we restrict to a $\rho_g$-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 $\lambda$-calculus.
- **Compatibility** property:

\[
\begin{align*}
G_1 & \xrightarrow{\rho_g} G_2 \\
G_1' & \xrightarrow{\rho_g} G_2'
\end{align*}
\]
Proof sketch (1/2)

- the $\Sigma$-rules: $(\delta) \cup (\text{external sub}) \cup (\text{acyclic sub})$
- the $\tau$-rules: $(\rho) \cup \text{Matching rules} \cup (\text{garbage})$

prove $\text{CON}_{\sim}$ for $\Sigma$  
prove $\text{CON}_{\sim}$ for $\tau$
Proof sketch (1/2)

- the $\Sigma$-rules: $(\delta) \cup (\text{external sub}) \cup (\text{acyclic sub})$
- the $\tau$-rules: $(\rho) \cup \text{Matching rules} \cup (\text{garbage})$

prove $\text{CON}_\sim$ for $\Sigma$
prove $\text{CON}_\sim$ for $\tau$
deduce $\text{CON}_\sim$ for $(\Sigma \cup \tau)$
Proof sketch (2/2)

1. $CON\sim$ for $\tau$: using local confluence and termination of the relation and the compatibility property

2. $CON\sim$ for $\Sigma$: using the finite developments method of the $\lambda$-calculus adapted to $\Sigma$

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 $\tau$: using *local confluence* and *termination* of the relation and the *compatibility* property

2. $CON_\sim$ for $\Sigma$: using the *finite developments* method of the $\lambda$-calculus adapted to $\Sigma$

3. $CON_\sim$ for $(\Sigma \cup \tau)$: using a *commutation* lemma for the two relations and the *compatibility* property

**Theorem**: The linear $\rho_g$-calculus is *Church-Rosser* modulo $AC1$. 
Expressiveness of the $\rho_g$-calculus

- Conservativity of the $\rho_g$-calculus vs $\rho$-calculus
- Conservativity of the $\rho_g$-calculus vs cyclic lambda
- Relationship with term graph rewriting
Conservativity of the $\rho_g$-calculus vs $\rho$-calculus

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

  $$\sigma(U) = T \text{ if and only if } T \ll U \xrightarrow{M} x_1 = U_1, \ldots, x_n = U_n$$
Conservativity of the $\rho_g$-calculus vs $\rho$-calculus

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

  $$\sigma(U) = T \iff T \ll U \xrightarrow{\mathcal{M}} x_1 = U_1, \ldots, x_n = U_n$$

- **Completeness**: If $T \xrightarrow{\rho} T'$ in the $\rho$-calculus then $T \xrightarrow{\rho_g} T'$ in the $\rho_g$-calculus.

- **Soundness**: Given a $\rho$-term $T$.

  If $T \xrightarrow{\rho_g} T'$ in the $\rho_g$-calculus and $T'$ contains no constraints, then $T \xrightarrow{\rho} T'$ in the $\rho$-calculus.
Matching failures in $\rho$-calculus and $\rho_g$-calculus

\[ \rho\text{-calculus} \quad \frac{f(a) \rightarrow b}{\mapsto \rho} f(c) \quad \mapsto \rho \quad b[f(a) \ll f(c)] \]
Matching failures in $\rho$-calculus and $\rho_g$-calculus

$\rho$-calculus

\[
(f(a) \rightarrow b) \ f(c) \quad \mapsto \rho \quad b[f(a) \ll f(c)]
\]

$\rho_g$-calculus

\[
(f(a) \rightarrow b) \ f(c) \quad \mapsto \rho \quad b \ [f(a) \ll f(c)]
\quad \mapsto \text{dk} \quad b \ [a \ll c]
\]
Conservativity of the $\rho_g$-calculus vs cyclic lambda

- Translation from a cyclic $\lambda$-term $t$ to a $\rho_g$-term $[t]$;

- Completeness:
  If $t_1 \overset{\lambda_c}{\rightarrow} t_2$ in the cyclic $\lambda$-calculus, then $[t_1] \overset{\rho_g}{\rightarrow} [t_2]$ in the $\rho_g$-calculus.

- Soundness:
  If $T_1 \overset{\rho_g}{\rightarrow} T_2$ in the $\rho_g$-calculus, with $T_1 = [t_1]$ and $T_2$ without matching constraints, then we have $t_1 \overset{\lambda_c}{\rightarrow} t_2$ with $[t_2] = T_2$. 

Clara Bertolissi
The graph rewriting calculus: confluence and expressiveness
Matching: the Matching rules well-behaves w.r.t. the notion of graph homomorphism

Completeness: If $G_0 \rightarrow G_n$ in a TGR, then there exist $n$ $\rho_g$-terms $H_1, \ldots, H_n$, built from the TGR reduction, such that $(H_1 \ldots (H_n G_0)) \rightarrow \rho_g G'_n$ with $G'_n$ homomorphic to $G_n$

Soundness:
If $G[(L \rightarrow R) G'] \rightarrow \rho_g G[H]$ with $G$, $G'$, $H$, $L$, $R$ term graphs and $L$ linear, then $G[G'] \rightarrow G[H']$ using the rule $(L, R)$ in the TGR, with $H'$ homomorphic to $H$. 

$\rho_g$-calculus vs TGR
General soundness \textit{w.r.t.} TGR does not hold

Consider the $\rho_g$-term $f((a \rightarrow b)x, (a \rightarrow c)x) [x = a]$
General soundness \textit{w.r.t.} TGR does not hold

Consider the $\rho_g$-term $f((a \rightarrow b)x, (a \rightarrow c)x) [x = a] \red{\rightarrow_{\rho_g}} f(b, c)$

Clara Bertolissi

The graph rewriting calculus: confluence and expressiveness
Consider the $\rho_g$-term $f((a \rightarrow b)x, (a \rightarrow c)x) [x = a]$.

$\mapsto_{\rho_g} f(b, c)$

In a TGR we have no corresponding reduction.
Conclusions

Expressive capabilities of the rewriting calculus:

- $\rho$-calculus and higher-order rewriting ($\text{CRS}_s$)

- $\rho$-calculus with graph-like structures
\(\rho\)-calculus vs CRS

- Characterisation of CRS matching and all its solutions.
  - Treat CRS matching as \(\lambda\)-calculus higher-order matching
  - Translations from a CRS to simply typed \(\lambda\)-calculus and back
  - Completeness and correctness of the approach
    - Universality and decidability of CRS pattern matching

- Encoding of CRS derivations into the \(\rho\)-calculus.
  - Translation function \([\cdot]\)
  - Preservation of matching solutions
  - Given a CRS-derivation \(t_0 \mapsto_R t_n\) there exists a \(\rho\)-term \(T\), built from this derivation, such that any reduction of \(T\) terminates and converges to \([t_n]\)
\(\rho\)-calculus vs \textsc{Crs}: perspectives

- encoding a \textsc{Crs} in the \(\rho\)-calculus directly from its set of rewrite rules (following [CLW03])

- encoding the \(\rho\)-calculus into \textsc{Crs}
Conclusions on the $\rho_g$-calculus

A generalisation of the cyclic $\lambda$-calculus with matching facilities

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

Properties: Confluence of the linear $\rho_g$-calculus,

Relation with other formalisms:
- Conservativity w.r.t. the standard $\rho$-calculus and the cyclic $\lambda$-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 $\rho$-calculus
- Implementation in TOM ($http://tom.loria.fr$)
- Applications: semantic web, telecom network, bio-informatics, ...