

Contents/Objectives of the lecture

- Definition and properties of rewriting
- Rule-based programming in ELAN
- **Logic and calculus for rewriting**
- Compilation or how to get efficiency?
- Applications and future developments

Let us come back on strategies...

rules for set

```
S: set ; i: int ;
```

```
[] Set(0) => Empty U (0) end
```

```
[] Set(i) => Set(i-1) U (i) end
```

```
[extractrule] (i) U S => [i U S] end
```

end

strategies for set

```
[] extractPos => dk(extractrule) end
```

end

Defined Strategies

Strategy operators with arguments (parameters).

$$\mathbf{map} : (\langle s \rightarrow s \rangle) \langle list[s] \rightarrow list[s] \rangle$$

$$\mathbf{map}(S) \Rightarrow \mathbf{dc}(\mathbf{nil}, S \cdot \mathbf{map}(S))$$

where $S : \langle s \rightarrow s \rangle$ and $s \in \mathcal{S}$.

$$[\mathbf{map}(S)](l)$$

equivalent to apply the two rules:

$$[\mathbf{map}(S)](\mathbf{nil}) \Rightarrow \mathbf{nil}$$

$$[\mathbf{map}(S)](x \cdot l') \Rightarrow [S](x) \cdot [\mathbf{map}(S)](l')$$

```

module map[X]

import global
  strat[X] strat[list[X]] X list[X]; end
stratop global
  map(@) : (<X->X) <list[X]->list[X]>;
end

rules for X
  x : X; n,m : int;
global
  [mul2] x => x*2          end
end

rules for list[X]
  s : <X->X>;
  [] map(s) => dc(nil,cons(s,map(s)))  end

```

Application of a strategy

`[map(mult2)] (3.4.5.6.nil)`

gives the result

`(6.8.10.12.nil)`

Strategy terms

- Elementary strategies: built from strategy constructors,
- Defined strategies: built from user's strategy operators, defined by rewrite rules applied with $[-]_-$ (meta-interpreter).

The Computational System Tower



Questions

What is a strategy?

Two points of views:

- A (set of) proof term(s) in rewrite logic
- A ρ -term in the ρ -calculus

How is defined strategy application?

Rule-based computation and deduction

The rewriting logic:
logical aspects of rewriting

Rewriting theory

$$\mathcal{R} = (\mathcal{F}, E, N, L, R)$$

- \mathcal{F} is a ranked alphabet of function symbols
- E is a set of \mathcal{F} -equalities (structural axioms)
- N is a set of confluent and terminating rewrite rules modulo E
- $A = E \cup N$ and $\langle t \rangle_A$: equational class of t modulo A
- L is a set of labels with arities
- R is a set of labelled conditional rewrite rules

$$\ell : l \Rightarrow r \text{ if } c$$

Rewriting Logic

[J. Meseguer TCS92]

Formulas are sequents of the form

$$\pi : \langle t \rangle_A \Rightarrow \langle t' \rangle_A$$

where π is a proof term, built on $\mathcal{F} \cup L \cup \{;\}$
recording the proof of the sequent.

Models are computation spaces (quotiented set of proof terms).

$$\mathcal{R} \vdash \pi : \langle t \rangle_A \Rightarrow \langle t' \rangle_A$$

if $\pi : \langle t \rangle_A \Rightarrow \langle t' \rangle_A$ can be obtained by finite application of the following deduction rules.

Reflexivity For any $t \in \mathcal{T}(\mathcal{F})$:

$$\langle t \rangle_A : \langle t \rangle_A \Rightarrow \langle t \rangle_A$$

Congruence For any $f \in \mathcal{F}$ with $\text{arity}(f) = n$:

$$\frac{\pi_1 : \langle t_1 \rangle_A \Rightarrow \langle t'_1 \rangle_A \quad \dots \quad \pi_n : \langle t_n \rangle_A \Rightarrow \langle t'_n \rangle_A}{f(\pi_1, \dots, \pi_n) : \langle f(t_1, \dots, t_n) \rangle_A \Rightarrow \langle f(t'_1, \dots, t'_n) \rangle_A}$$

Transitivity

$$\frac{\pi_1 : \langle t_1 \rangle_A \Rightarrow \langle t_2 \rangle_A \quad \pi_2 : \langle t_2 \rangle_A \Rightarrow \langle t_3 \rangle_A}{\pi_1; \pi_2 : \langle t_1 \rangle_A \Rightarrow \langle t_3 \rangle_A}$$

Replacement For any $\ell : l(x_1, \dots, x_n) \Rightarrow r(x_1, \dots, x_n) \in R$,

$$\frac{\pi_1 : \langle t_1 \rangle_A \Rightarrow \langle t'_1 \rangle_A \quad \dots \quad \pi_n : \langle t_n \rangle_A \Rightarrow \langle t'_n \rangle_A}{\ell(\pi_1, \dots, \pi_n) : \langle l(t_1, \dots, t_n) \rangle_A \Rightarrow \langle r(t'_1, \dots, t'_n) \rangle_A}$$

An example of proof

$$\begin{aligned} \mathcal{S} &= \{ \mathit{nat} \} \\ \mathcal{F} &= \left\{ \begin{array}{l} \mathit{zero} : \vdash \mathit{nat}, \\ \mathit{s} : \mathit{nat} \vdash \mathit{nat}, \\ \mathit{plus} : \mathit{nat} \times \mathit{nat} \vdash \mathit{nat} \end{array} \right\} \\ \mathcal{A} &= \{ \mathit{plus}(X, Y) = \mathit{plus}(Y, X) \} \\ \mathcal{R} &= \left\{ \begin{array}{l} \ell_0 : \mathit{plus}(\mathit{zero}, \mathit{zero}) \rightarrow \mathit{zero}, \\ \ell_1 : \mathit{plus}(\mathit{s}(X), Y) \rightarrow \mathit{s}(\mathit{plus}(X, Y)) \end{array} \right\} \end{aligned}$$

How to get a proof of:

$$\langle \mathit{plus}(\mathit{zero}, \mathit{s}(\mathit{zero})) \rangle_A \rightarrow \langle \mathit{s}(\mathit{zero}) \rangle_A$$

Replacement using the rewrite rule ℓ_1 and commutativity of $plus$:

$$\frac{\langle zero \rangle_A : \langle zero \rangle_A \rightarrow \langle zero \rangle_A}{l_1(\langle zero \rangle_A, \langle zero \rangle_A) : \langle plus(zero, s(zero)) \rangle_A \rightarrow \langle s(plus(zero, zero)) \rangle_A}$$

Replacement using the rewrite rule ℓ_0 :

$$\frac{\langle zero \rangle_A : \langle zero \rangle_A \rightarrow \langle zero \rangle_A}{l_0 : \langle plus(zero, zero) \rangle_A \rightarrow \langle zero \rangle_A}$$

Congruence for the symbol s :

$$\frac{l_0 : \langle plus(zero, zero) \rangle_A \rightarrow \langle zero \rangle_A}{s(l_0) : \langle s(plus(zero, zero)) \rangle_A \rightarrow \langle s(zero) \rangle_A}$$

Transitivity:

$$\frac{l_1(\langle zero \rangle_A, \langle zero \rangle_A) : \langle plus(zero, s(zero)) \rangle_A \rightarrow \langle s(plus(zero, zero)) \rangle_A, \quad s(l_0) : \langle s(plus(zero, zero)) \rangle_A \rightarrow \langle s(zero) \rangle_A}{l_1(\langle zero \rangle_A, \langle zero \rangle_A); s(l_0) : \langle plus(zero, s(zero)) \rangle_A \rightarrow \langle s(zero) \rangle_A}$$

Models of rewriting logic

Computation space of the rewrite theory \mathcal{R} :

quotient set $\{\pi \mid \mathcal{R} \vdash \pi : \langle t \rangle_A \rightarrow \langle t' \rangle_A\} / (E \cup \mathcal{A}_{\mathcal{PT}(R)})$.

$$\forall \pi_1, \pi_2, \pi_3 \in \mathcal{PT} \quad \pi_1; (\pi_2; \pi_3) = (\pi_1; \pi_2); \pi_3$$

Associativity

$$\forall \pi : \langle t \rangle_A \rightarrow \langle t' \rangle_A, \quad \pi; \langle t' \rangle_A = \pi, \quad \text{and} \quad \langle t \rangle_A; \pi = \pi$$

Local Identities

$$\text{For all } f \in \mathcal{F}_n, n \in \mathbf{N}, \quad \forall \pi_1, \dots, \pi_n, \pi'_1, \dots, \pi'_n:$$

$$f(\pi_1; \pi'_1, \dots, \pi_n; \pi'_n) = f(\pi_1, \dots, \pi_n); f(\pi'_1, \dots, \pi'_n)$$

Independence

For all $f \in \mathcal{F}_n, n \in \mathbf{N}$:

$$f(\langle t_1 \rangle_A, \dots, \langle t_n \rangle_A) = \langle f(t_1, \dots, t_n) \rangle_A$$

Preservation of A

$$\forall \ell : g \rightarrow d \in R, \forall \pi_1 : \langle t_1 \rangle_A \rightarrow \langle t'_1 \rangle_A, \dots, \pi_n : \langle t_n \rangle_A \rightarrow \langle t'_n \rangle_A$$

$$\ell(\pi_1, \dots, \pi_n) = \ell(\langle t_1 \rangle_A, \dots, \langle t_n \rangle_A); d(\pi_1, \dots, \pi_n) \text{ and}$$

$$\ell(\pi_1, \dots, \pi_n) = g(\pi_1, \dots, \pi_n); \ell(\langle t'_1 \rangle_A, \dots, \langle t'_n \rangle_A) .$$

Parallel Move Lemma

Strategies in rewriting logic

A strategy is a set of proof terms:

$$S = \{\pi \mid \pi \in \mathcal{PT}\}$$

Apply the strategy S to the term t :

find all terms t' such that

$$\pi : \langle t \rangle_A \rightarrow \langle t' \rangle_A \mid \pi \in S$$

Rule-based computation and deduction

The rewriting calculus:

A semantics for rule-based languages

Towards a new calculus

That gives a first class status to:

- ▶ rewrite rules — and therefore matching —

and

- ▶ strategies

The main ideas (1)

apply a **rule** at the top level of a **term**

$$[l \rightarrow r](t)$$

also denoted

$$(l \rightarrow r) \bullet t$$

The main ideas (2)

The application operator $\$ may return several results.

For example, if $+$ is commutative, what is the result of the application of the rule

$x + y \rightarrow x$ on $a + b$?

should it be a ?

or b ?

or $a \diamond b$?

The main ideas (3)

— rule application

— set of results

are **explicit** objects of the calculus

The calculus ingredients

Five components:

- 1** The syntax of terms and substitutions,
- 2** The description of the substitution application on terms,
- 3** The matching algorithm used to bind variables to their actual arguments,
- 4** The evaluation rules describing how the calculus operates locally.
- 5** The strategy describing how the evaluation rules operate globally.

The ρ -Calculus Syntax

- Elements in \mathcal{X} (variables) and in $\mathcal{T}(\mathcal{F}, \mathcal{X})$ are ρ -terms,

If t, u, t_1, \dots, t_m are ρ -terms and $f \in \mathcal{F}_m$ then the following expressions are ρ -terms:

- $f(t_1, \dots, t_m)$
- $\{t_1, \dots, t_m\}$ (if $m = 0$ we have the ρ -term \emptyset),
- $[t](u)$ (application of the ρ -term t to the ρ -term u),
- $t \rightarrow u$ (rewrite rule).

$$t ::= x \mid \{t, \dots, t\} \mid f(t, \dots, t) \mid t \mid t \rightarrow t \quad (x \in \mathcal{X}, f \in \mathcal{F})$$

Examples of ρ -terms

- $f(x, y)$ a first order term
- $f(x, x) \rightarrow x$ a “standard” rewrite rule
- $[a \rightarrow b](a)$ the application of the rule $a \rightarrow b$ to the term a
- $[f(x, y) \rightarrow g(x, y)](f(a, b))$ a classical rule application
- $[x \rightarrow x](a)$ the result is $\{a\}$
- $[x \rightarrow y](a)$ the result is $\{y\}$

More examples of ρ -terms

- $[x \rightarrow (x \rightarrow x)]([x \rightarrow y](a))$ similar to the λ -term $((\lambda x.\lambda x.x) ((\lambda x.y) a))$
- $[x \rightarrow x](x \rightarrow x)$ the well-known $(\Delta \Delta)$ λ -term
- $[(a \rightarrow b) \rightarrow c](a)$ a more complicated ρ -term
- $[\{a \rightarrow b, a \rightarrow c\}](a)$ apply several rules

Handling results

Handling several results via the set data structure

- has consequences on the calculus evaluation rules and their properties
- could be done in different ways using e.g. list, multisets, . . .

Substitution application on ρ -terms

Takes care of variable capture.

| Grafting | Substitution |
|---|---|
| $x \mapsto u$ | x/u |
| $\{x \mapsto a + y\}(y \rightarrow y + x)$ $= y \rightarrow y + (a + y)$ | $\{x/a + y\}(y \rightarrow y + x)$ $= y' \rightarrow y' + (a + y)$ |

Evaluation Rules of the ρ -Calculus

The main rule in general:

$$\mathbf{Fire} \quad [l \rightarrow r](t) \quad \mapsto \quad r \langle\langle \mathit{Solution}(l \ll_T^? t) \rangle\rangle$$

For example

$$\mathbf{Fire} \quad [f(x) \rightarrow x](f(a)) \quad \mapsto \quad x \langle\langle \mathit{Solution}(f(x) \ll_T^? f(a)) \rangle\rangle$$

In case of syntactic matching:

$$\begin{array}{l} \mathbf{Fire} \quad [l \rightarrow r](\sigma(l)) \quad \mapsto \quad \{\sigma(r)\} \\ \mathbf{Fire} \quad [l \rightarrow r](t) \quad \mapsto \quad \emptyset \\ \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \text{if } l \text{ does not match } t \end{array}$$

Congruence rules on operators

$$\begin{array}{lll} \mathbf{Congruence} & [f(u_1, \dots, u_n)](f(v_1, \dots, v_n)) & \mapsto \{f([u_1](v_1), \dots, [u_n](v_n))\} \\ \mathbf{Congruence_fail} & [f(u_1, \dots, u_n)](g(v_1, \dots, v_m)) & \mapsto \emptyset \end{array}$$

Equivalent to the reduction of

$$[f(x_1, \dots, x_n) \rightarrow f([u_1](x_1), \dots, [u_n](x_n))](t)$$

Congruence rules on sets

| | | | |
|----------------|--|-----------|---|
| Distrib | $[\{u_1, \dots, u_n\}](v)$ | \mapsto | $\{[u_1](v), \dots, [u_n](v)\}$ |
| Batch | $[v](\{u_1, \dots, u_n\})$ | \mapsto | $\{[v](u_1), \dots, [v](u_n)\}$ |
| SwitchL | $\{u_1, \dots, u_n\} \rightarrow v$ | \mapsto | $\{u_1 \rightarrow v, \dots, u_n \rightarrow v\}$ |
| SwitchR | $u \rightarrow \{v_1, \dots, v_n\}$ | \mapsto | $\{u \rightarrow v_1, \dots, u \rightarrow v_n\}$ |
| OpOnSet | $f(v_1, \dots, \{u_1, \dots, u_m\}, \dots, v_n)$ | \mapsto | $\{f(v_1, \dots, u_1, \dots, v_n),$ $\dots,$ $f(v_1, \dots, u_m, \dots, v_m)\}$ |

Handling sets of sets

Flat $\{u_1, \dots, \{v_1, \dots, v_n\}, \dots, u_m\} \rightsquigarrow \{u_1, \dots, v_1, \dots, v_n, \dots, u_m\}$

Applying substitutions

Meta-rules in this version

$$\begin{array}{ll} \text{Propagate} & r\langle\langle\{s_1, \dots, s_n\}\rangle\rangle \rightsquigarrow \{s_1(r), \dots, s_n(r)\} \\ \text{Clash} & r\langle\langle\emptyset\rangle\rangle \rightsquigarrow \emptyset \end{array}$$

Could be made explicit: $\rho\sigma$ -calculus

Remarks

- ρ -calculus describes rewrite rules application at the top level of a term

$$[f(x) \rightarrow g(x)](f(b)) \xrightarrow[\text{Fire}]{} \{g(b)\}$$

but

$$[x + 0 \rightarrow x](f(3 + 0)) \xrightarrow[\text{Fire}]{} \emptyset$$

and

$$[f(x + 0 \rightarrow x)](f(3 + 0)) \xrightarrow[\text{Congruence}]{} f([x + 0 \rightarrow x](3 + 0)) \xrightarrow[\text{Fire}]{} f(\{3\})$$

- The evaluation rules expressing the behavior of ρ -calculus are applied *everywhere* in the term (e.g. like β -reduction).

Matching

In general:

For a theory T a T -*match-equation* is a formula of the form $t \ll_T^? t'$, where t and t' are ρ -terms.

A substitution σ is a T -solution of $t \ll_T^? t'$ when

$$\sigma(t) =_T t'$$

T -matching is in general **undecidable**.

Matching

Decidable cases:

- higher-order pattern matching [Miller-89]
- higher-order matching (up to the fourth order [PadovaniThese-96])
- many first-order equational theories and their combinations

Solutions set

We define the function *Solution* on a T -matching system \mathcal{S} as returning the set of:

- — all T -matches of \mathcal{S} when \mathcal{S} is not trivial
- — $\{Id\}$ (the identity substitution) when \mathcal{S} is trivial
- — \emptyset when the matching algorithm fails.

Examples

If $+$ is commutative (C), the set of C -matches of $x + y \ll_T^? a + b$ is made of two substitutions:

$$\mathcal{S} = \{(x/a, y/b), (x/b, y/a)\}$$

If \cdot is associative commutative (AC), the set of C -matches of $x \cdot y \ll_T^? 1 \cdot 2 \cdot 3$ is made of 3 substitutions:

$$\mathcal{S} = \{(x/1, y/2 \cdot 3), (x/2, y/1 \cdot 3), (x/3, y/1 \cdot 2)\}$$

Specificities of the ρ_T -calculus

- It is a calculus of explicit rule application.
- It captures the “matching power”.
- It is parameterized by the matching algorithm.
- It explicitly handles result sets.
- It allows the full control of the rewrite rule application.
- It provides a uniform combination of higher-order and first-order features.
- It allows expressing strategies (e.g. search strategies).
- It should not be confused with the rewrite relation generated by a TRS.

The ρ_{\emptyset} -calculus

An instance of ρ_T -calculus where:

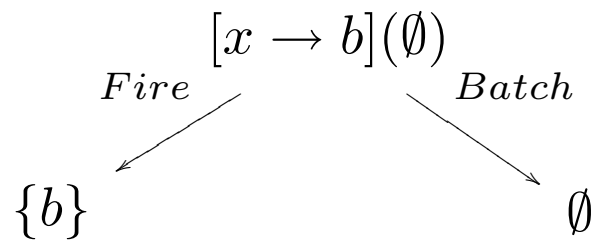
- — $T = \emptyset$
- — the left members of matching equations are composed only of first-order terms (i.e. not containing arrows or applications).

From now on we only consider ρ_{\emptyset} -calculus.

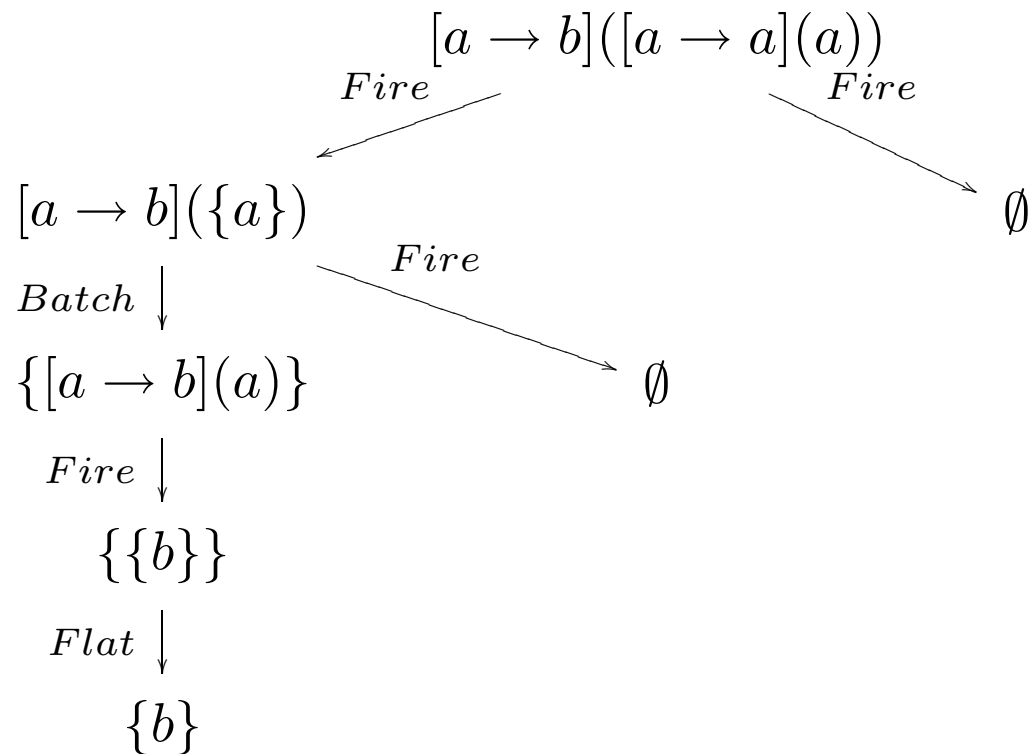
No difficulty to extend to the case of first-order equational matching (e.g. AC-matching)

Technical specificities

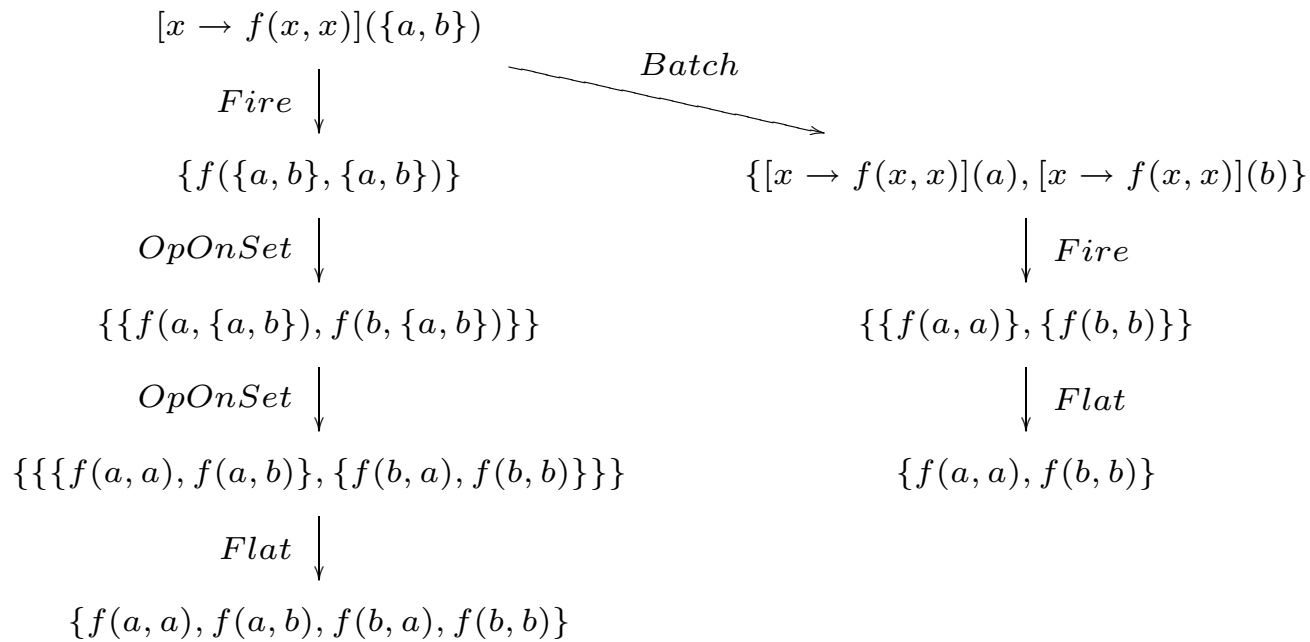
- A rule application always fires ... but the result may be the empty set (when the rule does not match an under-evaluated term),
- Thus the wild ρ_\emptyset -calculus is trivially non-confluent:



Undesired matching failures



Non-right-linear rewrite rules



The price of getting confluence

Under which conditions can we get the ρ_\emptyset -calculus confluent?

Evaluation strategy

When using a call-by-value evaluation strategy, the ρ_\emptyset -calculus is confluent.

The term $[l \rightarrow r](t)$ is reduced using the evaluation rule **Fire** only if $t \in \mathcal{T}(\mathcal{F})$.

Other sufficient conditions are given in H.Cirstea's Phd Thesis.

About the expressiveness of the ρ -calculus

Encoding the λ -calculus

$t ::= x \mid \{t\} \mid f(t, \dots, t) \mid t \mid x \rightarrow t$

FirePropagate $[x \rightarrow r](t) \rightsquigarrow \{\{x/t\}r\}$

$\varphi(\lambda x.t) = x \rightarrow t$

$\varphi((s t)) = [s](t)$

Given t and t' two λ -terms. Then, $t \longrightarrow_{\beta} t'$ iff $\varphi(t) \longrightarrow_{\rho} \{\varphi(t')\}$.

Proof: For any λ -term $(\lambda x.t u)$ that reduces to $\{x/u\}t$ the corresponding ρ -reduction is $[x \rightarrow t](u) \longrightarrow_{\text{FirePropagate}^*} \{\{x/u\}t\}$. \square

Encoding rewriting

$t ::= x \mid \{t\} \mid f(t, \dots, t) \mid [t](l) \mid l \rightarrow l$

Given t and t' two terms in $\mathcal{T}(\mathcal{F}, \mathcal{X})$ and \mathcal{R} a first order term rewrite system.

If $t \xrightarrow{*}_{\mathcal{R}} t'$ then there exists a ρ -term u constructed using the rules in \mathcal{R} such that $[u](t) \xrightarrow{*}_{\rho} \{t'\}$.

Proof: Let us consider a rewrite rule $(l \rightarrow r)$ that applies on the term t at position p and thus, $t[\theta l]_p \xrightarrow{\mathcal{R}} t[\theta r]_p = t'$.

The ρ -term u to be applied in the ρ -calculus is $t[l \rightarrow r]_p$ and we get:

$$[t[l \rightarrow r]_p](t[\theta l]_p) \xrightarrow{*}_{\rho} \{t[\theta r]_p\}$$

□

(Normalization) Strategies

Given a rewrite theory \mathcal{R} ,

does it exist a ρ -term $\xi_{\mathcal{R}}$ such that for any term u ,

if u reduces to the term v in the rewrite theory \mathcal{R} ,

then $[\xi_{\mathcal{R}}](u)$ ρ -reduces to $\{\dots, v, \dots\}$?

Given a rewrite theory \mathcal{R} ,

does it exist a ρ -term $\xi_{\mathcal{R}}$ such that for any term u ,

if u **normalizes** to the term v in the rewrite theory \mathcal{R} ,

then $[\xi_{\mathcal{R}}](u)$ **ρ -normalizes** to $\{\dots, v, \dots\}$?

Extending the calculus: with syntactic sugar

$$t ::= id \mid fail \mid t; t \mid dk(t, t)$$

Evaluation rules

Identity id \mapsto $x \rightarrow x$

Fail $fail$ \mapsto $x \rightarrow \emptyset$

Compose $[s_1; s_2](t)$ \mapsto $[s_2]([s_1](t))$

DK $[dk(s_1, s_2)](t)$ \mapsto $\{[s_1](t), [s_2](t)\}$

The *first* operator

First $[first(s_1, \dots, s_n)](t) \quad \mapsto \quad \langle [s_1](t), \dots, [s_n](t) \rangle$

First_fail $\langle \emptyset, t_1, \dots, t_n \rangle \quad \mapsto \quad \langle t_1, \dots, t_n \rangle$

First_success $\langle t, t_1, \dots, t_n \rangle \quad \mapsto \quad \{t\}$

if $t \neq \emptyset$ is closed and contains no redex

First_single $\langle \rangle \quad \mapsto \quad \{ \}$

Fixpoint operator

$$\Theta = A \text{ with } A = x \rightarrow (y \rightarrow [y]([x](y)))$$

Thus

$$[\Theta](G) \xrightarrow{*}_{\rho} \{[G]([\Theta](G))\}$$

Repeat operator

$$\text{repeat}^*(r) \equiv [\Theta](J(r))$$

where

$$J(r) \equiv f \rightarrow (x \rightarrow [first(r; f, id)](x))$$

Term traversal

Tseq $[\Phi(r)](f(u_1, \dots, u_n)) \quad \mapsto \quad \langle \{f([r](u_1), \dots, u_n)\}, \dots, \{f(u_1, \dots, [r](u_n))\} \rangle$

Tseq_ct $[\Phi(r)](c) \quad \mapsto \quad \emptyset$

Tpar $[\Psi(r)](f(u_1, \dots, u_n)) \quad \mapsto \quad \{f([r](u_1), \dots, [r](u_n))\}$

Tpar_ct $[\Psi(r)](c) \quad \mapsto \quad \{c\}$

Normalisation strategies

Bottom-up application

$$Once_{bu}(r) \equiv [\Theta](H_{bu}(r))$$

where

$$H_{bu}(r) \equiv f \rightarrow (x \rightarrow [first(\Phi(f), r)](x))$$

Ex: $[Once_{bu}(a \rightarrow b)](f(a, g(a))) \rightsquigarrow \{f(b, g(a))\}$

leftmost-innermost/outermost strategies

$$lmim(r) \equiv repeat^*(Once_{bu}(r))$$

$$lmom(r) \equiv repeat^*(Once_{td}(r)).$$

Encoding Conditional Rewriting

The conditional rewrite rule:

$$l \rightarrow r \text{ if } c$$

is simply encoded by

$$l \rightarrow [\{\mathbf{T} \rightarrow r, \mathbf{F} \rightarrow \emptyset\}](\text{lmim}(c))$$

Or even simpler

$$l \rightarrow [\mathbf{T} \rightarrow r](\text{lmim}(c))$$

ELAN's semantics

Using rewriting logic [Meseguer 92]

Using ρ -calculus

The ELAN basic object: rewrite rules with where

Rules are labelled conditional rewrite rules with local variable affectations

```
[lab]     $l \Rightarrow r(x, y)$   
         where  $x := [S_1]u_1$   
         if  $c_1(x)$   
         where  $y := [S_2]u_2$   
         if  $c_2(x, y)$ 
```

- lab is the rule label,
- l and r are the respective left and right-hand sides,
- c_1, c_2 are the conditions and
- $z := [S]u$ are local affectations, assigning to the local variable z one of the result of the strategy S applied to the term u .

Example of ELAN's rule

$$\begin{aligned} [\text{deriveSum}] \quad p_1 + p_2 &\Rightarrow p'_1 + p'_2 \\ &\text{where } p'_1 := (\text{derive})p_1 \\ &\text{where } p'_2 := (\text{derive})p_2 \end{aligned}$$

can be represented by the following two ρ -terms

$$p_1 + p_2 \rightarrow [\text{derive}](p_1) + [\text{derive}](p_2)$$

$$p_1 + p_2 \rightarrow [p'_1 \rightarrow [p'_2 \rightarrow p'_1 + p'_2]([\text{derive}](p_2))][[\text{derive}](p_1)]$$

An expression of ELAN's rules in ρ -calculus

[lab] $l \Rightarrow r(x)$
where $x := [S_1]u_1$
if $c_1(x)$

The rule is expressed as the ρ -term:

$$l \rightarrow [x \rightarrow [True \rightarrow r(x)]([lmim](c_1(x)))]([S_1](u_1))$$

With two where if

```
[lab]    l ⇒  r(x)
           where x := [S1]u1
           if  c1(x)
           where y := [S2]u2
           if  c2(x, y)
```

The rule is expressed as the ρ -term:

$$l \rightarrow [x \rightarrow [True \rightarrow [y \rightarrow [True \rightarrow r(x, y)]([lmim](c_2(x, y)))]([S_2](u_2))]([lmim](c_1(x)))]([S_1](u_1))$$

To sum-up

- ▶ ρ_T -calculus is a simple, elegant and powerful calculus
 - explicit rule application (and thus “matching power”)
 - explicit handling of result sets
 - parameterized by a matching theory T
- ▶ Clear distinction between the rewrite relation and the rewrite calculus.
- ▶ It allows uniform combination of first and higher-order computations
- ▶ ρ -calculus gives a simple semantics to ELAN programs

To know more

- Horatiu Cirstea's PhD, en Francais
- IGPL'2001, H.Cirstea and C.Kirchner : ρ -calculus (I and II)
- FOSSACS'2001, H.Cirstea, C.Kirchner, L.Liquori: ρ -cube
- RTA'2001, H.Cirstea, C.Kirchner, L.Liquori: matching power
- ESSLLI'2001, introduction to ρ -calculus
- www.loria.fr/~ckirchne/=rho/, all papers on the ρ -calculus