Termination of rewrite relations on λ-terms using the notion of computability closure

Frédéric Blanqui

INVENTEURS DU MONDE NUMÉRIQUE

Academia Sinica, 16 November 2012

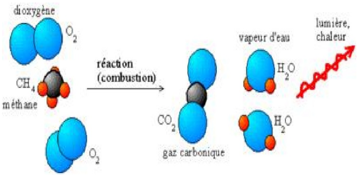
Rewriting

Rewriting is a simple yet Turing-complete framework for defining functions and proving equalities on terms.

Given a set $\mathcal{R} \subseteq \mathcal{T} \times \mathcal{T}$ of rewrite rules, $t \rightarrow_{\mathcal{R}} u$ if there are:

- a position p in t,
- a substitution σ,
- a rule $l \rightarrow r \in \mathcal{R}$

such that $t|_p = l\sigma$ ($t|_p$ matches l) and $u = t[r\sigma]_p$.



First-order rewriting

First-order rewriting is rewriting on first-order terms:

$$t = x \mid \mathsf{f}\, t_1 \ldots t_n$$

where f belongs to a fixed set of function symbols.

Rewriting theory has a long history: Thue (1914), Post, Markov (1947), Knuth (1967), Huet (1976), Dershowitz (1979), ...

$$\begin{array}{rcl} (x \cdot y) \cdot z & \rightarrow & x \cdot (y \cdot z) \\ x \cdot 1 & \rightarrow & x \\ x \cdot x^{-1} & \rightarrow & 1 \end{array}$$

λ-terms

λ-terms form a term algebra for functions (Church 1940)

$$t = x \mid \lambda x t \mid t t$$

⚠ Difference wrt first-order terms: substitution is defined modulo α-equivalence (renaming of bound variables):

$$(\lambda x y)_y^x =_\alpha \lambda x' x$$

$\Rightarrow$ termination techniques developed for FO rewriting do not generally apply to λ-calculus

λ-calculus

Function evaluation is obtained by using the β rule schema:

$$(\lambda x t)u \rightarrow_{\beta} t_x^u$$

It is Turing-complete but does not allow to represent many useful algorithms efficiently.

$\Rightarrow$ Hence the interest of extending it with function symbols f defined by rewrite rules $\mathsf{f}\, l_1 \ldots l_n \rightarrow r$.

Higher-order rewriting

Higher-order rewriting is rewriting on λ-terms:

$$t = x \mid \lambda x t \mid t t \mid \mathsf{f}$$

$$\begin{array}{rcl} D(\lambda x y) & \rightarrow & \lambda x 0 \\ D(\lambda x x) & \rightarrow & \lambda x 1 \\ D(\lambda x \sin(F x)) & \rightarrow & \lambda x D F x \times \cos(F x) \end{array}$$

Higher-order rewriting - Approach 1

- simply-typed λ-terms in β-normal η-long form
- matching modulo $\alpha\beta\eta$



Combinatory Reduction Systems (CRS) (Klop 1980)
Expression Reduction Systems (ERS) (Khasidashvili 1990)
Higher-order Rewrite Systems (HRS) (Nipkow 1991)

Simply-typed λ-calculus

simple types: $T = \mathsf{B} \mid T \Rightarrow T$

$$x^U : U \qquad \frac{t : T}{\lambda x^U t : U \Rightarrow T} \qquad \frac{v : U \Rightarrow T \quad u : U}{vu : T}$$

$\rightarrow_{\beta\eta}$ and $\rightarrow_{\beta\overline{\eta}}$ terminate and are confluent on typed λ-terms
$\Rightarrow$ every λ-term has a unique β-normal η-long (η-short) form

$$\begin{array}{rcll} \lambda x(tx) & \rightarrow_\eta & t & \text{if } x \notin \mathrm{Var}(t) \\ t & \rightarrow_{\overline{\eta}} & \lambda x(tx) & \text{if } x \notin \mathrm{Var}(t) \text{ and } t : U \Rightarrow V \text{ is not applied} \end{array}$$

Higher-order rewriting - Approach 1

can encode the untyped λ-calculus itself:

$$\mathsf{App}{:}\iota \Rightarrow \iota \Rightarrow \iota$$
$$\mathsf{Lam}{:}(\iota \Rightarrow \iota) \Rightarrow \iota$$

$$\mathsf{App}(\mathsf{Lam}X)Y \rightarrow_{\mathcal{R}} XY$$
$$\mathsf{Lam}(\lambda x \mathsf{App}Xx) \rightarrow_{\mathcal{R}} X$$

with $w = \mathsf{Lam}(\lambda x \mathsf{App}xx)$

$\mathsf{App}ww \rightarrow_{\mathcal{R}} (\lambda x \mathsf{App}xx)w \downarrow_{\beta\overline{\eta}} = \mathsf{App}ww \rightarrow_{\mathcal{R}} \ldots$

Higher-order rewriting - Approach 2

- arbitrary λ-terms
- matching modulo α

Higher-order Algebraic Specification Languages (Jouannaud-Okada 1991)

Problem

Sufficient conditions for the termination of $\rightarrow_{\mathcal{R}}$ or $\rightarrow_\beta \cup \rightarrow_{\mathcal{R}}$?

- Toyama 1988: $\mathrm{SN}(R_1) \wedge \mathrm{SN}(R_2) \not\Rightarrow \mathrm{SN}(R_1 \uplus R_2)$

$$\mathcal{R}_1 = \{\mathsf{fab}x \rightarrow \mathsf{f}xxx\} \quad \mathcal{R}_2 = \left\{ \begin{array}{ccc} \mathsf{g}xy & \rightarrow & x \\ \mathsf{g}xy & \rightarrow & y \end{array} \right\}$$

$$\mathsf{f(gab)(gab)(gab)} \rightarrow^2_{\mathcal{R}} \mathsf{fab(gab)} \rightarrow_{\mathcal{R}} \mathsf{f(gab)(gab)(gab)} \rightarrow_{\mathcal{R}} \ldots$$

- Dougherty 1992: $\rightarrow_\beta \cup \rightarrow_{\mathcal{R}}$ terminates on any $\mathcal{R}$-stable set if $\mathcal{R}$ is FO and $\rightarrow_{\mathcal{R}}$ terminates on FO terms

  (because FO rewriting cannot create β-redexes)

Method 1 for $\rightarrow_\beta$ alone

On simply-typed λ-terms:

$\rightarrow_\beta$ can be proved terminating by a direct induction on the type of the substituted variable (Sanchis 1967, van Daalen 1980)

$$(\lambda x^{A \Rightarrow U} xv)(\lambda y^A u) \rightarrow_\beta (\lambda y^A u)v$$

this extends neither to polymorphic types nor to rewriting since, in these cases, the type of substituted variables may not decrease

$$\mathsf{f}(\mathsf{c}x) \rightarrow x \text{ with } \mathsf{f} : \mathsf{B} \Rightarrow (\mathsf{B} \Rightarrow \mathsf{A}) \text{ and } \mathsf{c} : (\mathsf{B} \Rightarrow \mathsf{A}) \Rightarrow \mathsf{B}$$

Method 2 for $\rightarrow_\beta$ alone

On simply-typed λ-terms:

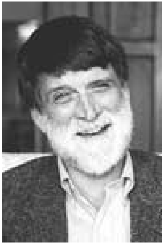
λI-terms ($x \in \mathrm{Var}(t)$ in $\lambda x t$) can be interpreted by hereditarily monotone functions on $\mathbb{N}$ (Gandy 1980)

this can be used to build interpretations (van de Pol 1996, Hamana 2006) but these interpretations can also be obtained from an extended computability proof

Outline

Computability

Computability has been introduced for proving termination of β-reduction in typed λ-calculi by Tait (1967) and Girard (1970)



- every type T is mapped to a set $[\![T]\!]$ of computable terms
- every $t : T$ is proved to be computable, *i.e.* $t \in [\![T]\!]$

Computability predicates

There are different definitions of computability (Tait, Girard, Parigot) but Girard's definition Red is better suited for rewriting.

Let Red be the set of P such that:

- $P \subseteq \mathrm{SN}(\rightarrow_\beta)$
- $\rightarrow_\beta (P) \subseteq P$
- if t is neutral and $\rightarrow_\beta(t) \subseteq P$ then $t \in P$

Main idea of neutrality: if t is neutral then the reduction of tu does not create new redexes ($\Rightarrow \lambda xu$ is not neutral).

Computable terms

Red is a complete lattice for set inclusion that is closed by:

$$a(P, Q) = \{t \mid \forall u \in P, tu \in Q\}$$

By taking $[\![U \Rightarrow V]\!] := a([\![U]\!], [\![V]\!])$,

a term $t : U \Rightarrow V$ is computable if:
for every computable term $u : U$, tu is computable

Application to rewriting (Jouannaud-Okada 1991)

Given a set $\mathcal{R}$ of rewrite rules, let $\rightarrow = \rightarrow_\beta \cup \rightarrow_\mathcal{R}$ and $\mathrm{Red}_\mathcal{R}$ be the set of P such that:

- $P \subseteq \mathrm{SN}(\rightarrow)$
- $\rightarrow(P) \subseteq P$
- if t is neutral and $\rightarrow(t) \subseteq P$ then $t \in P$
  $\mathsf{f}\vec{t}$ is neutral if $|\vec{t}| \geq \sup\{|\vec{l}| \mid \mathsf{f}\vec{l} \rightarrow r \in \mathcal{R}\}$

Theorem: $\rightarrow_\beta \cup \rightarrow_\mathcal{R}$ terminates if every rule of $\mathcal{R}$ is of the form $\mathsf{f}\vec{l} \rightarrow r$ with $r \in \mathrm{CC}_{\mathcal{R},\mathsf{f}}(\vec{l})$, set of terms computable when $\vec{l}$ so are.

Computability closure

By what operation $\mathrm{CC}_{\mathcal{R},\mathsf{f}}(\vec{l})$ can be closed?

$$\text{(arg)}\quad l_i \in \mathrm{CC}_{\mathcal{R},\mathsf{f}}(\vec{l})$$

$$\text{(app)}\quad \frac{t : U \Rightarrow V \in \mathrm{CC}_{\mathcal{R},\mathsf{f}}(\vec{l}) \qquad u : U \in \mathrm{CC}_{\mathcal{R},\mathsf{f}}(\vec{l})}{tu \in \mathrm{CC}_{\mathcal{R},\mathsf{f}}(\vec{l})}$$

$$\text{(red)}\quad \frac{t \in \mathrm{CC}_{\mathcal{R},\mathsf{f}}(\vec{l}) \qquad t \rightarrow_\beta \cup \rightarrow_\mathcal{R} t'}{t' \in \mathrm{CC}_{\mathcal{R},\mathsf{f}}(\vec{l})}$$

Dealing with bound variables

Annotate $\mathrm{CC}_{\mathcal{R},\mathsf{f}}(\vec{l})$ with a set X of (bound) variables:

$$\text{(var)}\ \frac{x \in X}{x \in \mathrm{CC}^{X}_{\mathcal{R},\mathsf{f}}(\vec{l})}$$

$$\text{(lam)}\ \frac{t \in \mathrm{CC}^{X \cup \{x\}}_{\mathcal{R},\mathsf{f}}(\vec{l}) \quad x \notin \mathrm{FV}(\vec{l})}{\lambda x t \in \mathrm{CC}^{X}_{\mathcal{R},\mathsf{f}}(\vec{l})}$$

Dealing with subterms

Problem: computability is not preserved by subterm... :-(

with $\mathsf{c} : (\mathsf{B} \Rightarrow \mathsf{A}) \Rightarrow \mathsf{B}$, $\mathsf{f} : \mathsf{B} \Rightarrow (\mathsf{B} \Rightarrow \mathsf{A})$ and $\mathcal{R} = \{\mathsf{f}(\mathsf{c}x) \to x\}$,
$\to_\beta \cup \to_\mathcal{R}$ does not terminate (Mendler1987):

with $w = \lambda x^\mathsf{B} \mathsf{f} x x$, $w(\mathsf{c}w) \to_\beta \mathsf{f}(\mathsf{c}w)(\mathsf{c}w) \to_\mathcal{R} w(\mathsf{c}w) \to_\beta \ldots$

$\Rightarrow$ restrictions on subterms (based on types) are necessary:

$$\text{(sub-app-fun)}\quad \frac{\mathsf{g}\vec{t} \in \mathrm{CC}^X_{\mathcal{R},\mathsf{f}}(\vec{l}) \quad \mathsf{g} : \vec{T} \Rightarrow \mathsf{B} \quad \mathrm{Pos}(\mathsf{B}, T_i) \subseteq \mathrm{Pos}^+(T_i)}{t_i \in \mathrm{CC}^X_{\mathcal{R},\mathsf{f}}(\vec{l})}$$

Dealing with subterms

$$\text{(sub-app-var-l)}\ \frac{tu \in \mathrm{CC}^X_{\mathcal{R},\mathsf{f}}(\vec{l}) \quad u{\downarrow_\eta} \in X}{t \in CC^X_{\mathsf{f}}(\vec{l})}$$

$$\text{(sub-app-var-r)}\ \frac{tu \in \mathrm{CC}^X_{\mathcal{R},\mathsf{f}}(\vec{l}) \quad t{\downarrow_\eta} \in X \quad t : U \Rightarrow \vec{U} \Rightarrow U}{u \in CC^X_{\mathsf{f}}(\vec{l})}$$

$$\text{(sub-lam)}\ \frac{\lambda xt \in \mathrm{CC}^X_{\mathcal{R},\mathsf{f}}(\vec{l}) \quad x \notin \mathrm{FV}(\vec{l})}{t \in \mathrm{CC}^{X\cup\{x\}}_{\mathcal{R},\mathsf{f}}(\vec{l})}$$

$$\text{(sub-SN)}\ \frac{t \in \mathrm{CC}^X_{\mathcal{R},\mathsf{f}}(\vec{l}) \quad u : \mathsf{B} \trianglelefteq t \quad \mathrm{FV}(u) \subseteq \mathrm{FV}(t) \quad [\![\mathsf{B}]\!] = \mathrm{SN}}{u \in \mathrm{CC}^X_{\mathcal{R},\mathsf{f}}(\vec{l})}$$

Dealing with function calls

Consider a relation $\sqsupset$ on pairs $(\mathsf{h}, \vec{v})$, where $\vec{v}$ are computable arguments of h, such that $\sqsupset \cup \rightarrow_{\mathrm{prod}}$ is well-founded.

$$\text{(app-fun)}\ \frac{(\mathsf{f}, \vec{l}) \sqsupset (\mathsf{g}, \vec{t}) \quad \vec{t} \in \mathrm{CC}_{\mathcal{R},\mathsf{f}}(\vec{l})}{\mathsf{g}\vec{t} \in \mathrm{CC}_{\mathcal{R},\mathsf{f}}(\vec{l})}$$

Example: $(\mathsf{f}, \vec{l}) \sqsupset (\mathsf{g}, \vec{t})$ if either:

- $\mathsf{f} > \mathsf{g}$
- $\mathsf{f} \simeq \mathsf{g}$ and $\vec{l}\ ((\rhd \cup \rightarrow)^{+})_{\mathrm{stat}[\mathsf{f}]}\ \vec{t}$

where $\geq$ is a well-founded quasi-ordering on symbols and $\mathrm{stat}[\mathsf{f}] = \mathrm{stat}[\mathsf{g}] \in \{\mathrm{lex}, \mathrm{mul}\}$

Outline

Dealing with higher-order pattern-matching

$$\mathsf{f}\vec{t} =_{\beta\eta} \mathsf{f}\vec{l}\sigma \rightarrow_{\mathcal{R}} r\sigma$$

Problem: $\vec{t}$ computable $\Rightarrow$ $\vec{l}\sigma$ computable?

Dealing with higher-order pattern-matching

Dale Miller (1991): if l is an *higher-order pattern* (free variables are applied to distinct bound variables) and $l\sigma =_{\beta\eta} t$ with σ and t in β-normal η-long form, then $l\sigma \rightarrow^*_{\beta_0} =_\eta t$ where $C[(\lambda x u)v] \rightarrow_{\beta_0} C[u^v_x]$ if $v \in \mathcal{X}$

$\Rightarrow$ consider β_0-normalized rewriting with matching modulo $\beta_0\eta$ (subsumes CRS and HRS rewriting)!

Theorem: assuming that $\leftarrow_{\beta_0\eta}\rightarrow_{\mathcal{R},\beta_0\eta} \subseteq \rightarrow_{\mathcal{R},\beta_0\eta} =_{\beta_0\eta}$,
if t is computable and $t =_{\beta_0\eta} l\sigma$ with l an higher-order pattern, then $l\sigma$ is computable.

Dealing with higher-order pattern-matching

Theorem: $\leftarrow_{\beta_0\eta}\rightarrow_{\mathcal{R},\beta_0\eta} \subseteq \rightarrow_{\mathcal{R},\beta_0\eta}=_{\beta_0\eta}$ if:

- every rule is of the form $\mathsf{f}\vec{l} \rightarrow r$ with $\mathsf{f}\vec{l}$ an higher-order pattern
- if $l \rightarrow r \in \mathcal{R}$, $l : T \Rightarrow U$ and $x \notin \mathrm{FV}(l)$, then $lx \rightarrow rx \in \mathcal{R}$
- if $lx \rightarrow r \in \mathcal{R}$ and $x \notin \mathrm{FV}(l)$, then $l \rightarrow \lambda xr \in \mathcal{R}$

$$s \leftarrow_{\beta_0} (\lambda xs)x =_{\beta_0\eta} l\sigma x \rightarrow_{\mathcal{R}} r\sigma x$$

$$s \leftarrow_{\eta} \lambda xsx =_{\beta_0\eta} \lambda xl\sigma \rightarrow_{\mathcal{R}} \lambda xr\sigma$$

$\Rightarrow$ every set of rules of the form $\mathsf{f}\vec{l} \rightarrow r$ with $\mathsf{f}\vec{l}$ an higher-order pattern can be completed into a set compatible with $\rightarrow_{\beta_0\eta}$

Outline

Dealing with rewriting modulo some equational theory

$$\mathsf{f}\vec{t} =_{\mathcal{E}} u \rightarrow_{\mathcal{R}} v$$

Problem: $\vec{t}$ computable $\Rightarrow$ v computable?

Dealing with rewriting modulo some equational theory

First, we need $\mathrm{SN}(\rightarrow_\beta)$ to be closed by $=_\mathcal{E}$. For instance:

Theorem: $\rightarrow_\beta =_\mathcal{E} \subseteq =_\mathcal{E} \rightarrow_\beta$ if:

- $\mathcal{E}$ is linear (no variable occurs twice)
- $\mathcal{E}$ is regular ($\forall l = r \in \mathcal{E}, \mathrm{FV}(l) = \mathrm{FV}(r)$)
- $\mathcal{E}$ is algebraic (no abstraction nor applied variable)

✗ $x \times 0 = 0$

✗ $x \times (y + z) = (x \times y) + (x \times z)$

✓ $\forall(\lambda x \forall(\lambda y P x y)) = \forall(\lambda y \forall(\lambda x P x y))$

Dealing with rewriting modulo some equational theory

Given a set $\mathcal{E}$ of equations and a set $\mathcal{R}$ of rewrite rules, let now $\rightarrow = \rightarrow_\beta \cup =_\mathcal{E}\rightarrow_\mathcal{R}$ and $\mathrm{Red}^\mathcal{E}_\mathcal{R}$ be the set of P such that:

- $P \subseteq \mathrm{SN}(\rightarrow)$
- $\rightarrow(P) \subseteq P$ and $=_\mathcal{E}(P) \subseteq P$
- if t is neutral and $\rightarrow(t) \subseteq P$ then $t \in P$

Dealing with rewriting modulo some equational theory

Theorem: assuming that $\to_\beta =_\mathcal{E} \subseteq =_\mathcal{E} \to_\beta$, the relation $\to_\beta \cup =_\mathcal{E} \to_\mathcal{R}$ terminates if:

- every rule of $\mathcal{R}$ is of the form $\mathsf{h}\vec{n} \to r$ with $r \in \mathrm{CC}^{\mathcal{E}}_{\mathcal{R},\mathsf{h}}(\vec{n})$,
- every equation of $\mathcal{E}$ is of the form $\mathsf{f}\vec{l} = \mathsf{g}\vec{m}$ with $\vec{m} \in \mathrm{CC}^{\mathcal{E}}_{\mathcal{R},\mathsf{f}}(\vec{l})$ and $\vec{l} \in \mathrm{CC}^{\mathcal{E}}_{\mathcal{R},\mathsf{g}}(\vec{m})$.

$$\mathsf{f}\vec{t} = \mathsf{f}\vec{l}\sigma \leftrightarrow_\mathcal{E} \mathsf{g}\vec{m}\sigma \leftrightarrow_\mathcal{E} \ldots \leftrightarrow_\mathcal{E} \mathsf{h}\vec{n}\theta \to_\mathcal{R} r\theta = v$$

$$\vec{t} \text{ computable} \Rightarrow \vec{m}\sigma \text{ computable} \Rightarrow \ldots \Rightarrow v \text{ computable}$$

Dealing with rewriting modulo some equational theory

Examples:

- commutativity: $+xy = +yx$

$$\{y, x\} \subseteq \mathrm{CC}_+(xy)$$

- associativity: $+(+xy)z = +x(+yz)$

$$\{x, +yz\} \subseteq \mathrm{CC}_+((+xy)z)$$
$$\{+xy, z\} \subseteq \mathrm{CC}_+(x(+yz))$$

To know more on computability closure

- how to deal with constructors having functional arguments
- how to deal with conditional rewriting
- what is the relation with RPO
- what is the relation with dependency pairs
- what is the relation with semantic labelling

see `https://who.rocq.inria.fr/Frederic.Blanqui/`

Thank you!