Habilitation à diriger des recherches - Universtité Paris 7 Denis Diderot

Functions, rewriting and proofs:
termination and certification

Frédéric Blanqui

INVENTEURS DU MONDE NUMÉRIQUE

13 July 2012

Rewriting?

Monsieur,

Par décision en date du 13 juin 2012, vous avez été autorisé à présenter en soutenance vos travaux en vue de l'obtention du diplôme :

H.D.R. EN LETTRES ET SCIENCES HUMAINES

La soutenance aura lieu le 13 juillet 2012 à 9h00 à l'adresse suivante :

Laboratoire PPS - salle 1C06 - 175 rue du Chevaleret - 75013 Paris

La soutenance sera publique.

Je vous prie d'agréer, Monsieur, l'expression de mes salutations distinguées.

Rewriting!

Outline

Hardware/software bugs can have dramatic consequences

- 1993: Intel Pentium bug on floating point number division cost $475 millions
- 1996: Ariane V exploded because of an overflow
- 2000: 8 patients died because of miscalculated radiation dosage at the National Cancer Institute, Panama
- 2008: some investors lost 60% of their investment because of a bug in Moody's software
- 2012: Orange?
- ...

Goal of my research work

design tools and methodologies for helping hardware/software developers to write bug-free systems

How to prove the correctness of a program?

a program is a syntactic object (term) p

proving that p satisfies some property Q requires to have a clear semantics, *i.e.* a (partial) function $[\![p]\!] : \mathrm{IN} \to \mathrm{OUT}$

How to prove the correctness of a program?

a program is a syntactic object (term) p

proving that p satisfies some property Q requires to have a clear semantics, *i.e.* a (partial) function $[\![p]\!] : \mathrm{IN} \to \mathrm{OUT}$

$\Rightarrow$ proving the correctness of a program is a particular case of theorem proving

Is it decidable to find a proof?

In general: NO (Turing 1936)

BUT there are various decidable classes very important in practice: SAT, linear arithmetic, . . .

Is it decidable to ~~find~~ check a proof?

proof assistant: tool for defining mathematical objects, stating theorems and building proofs

- 1967: Automath (De Bruijn)
- 1972: LCF (Milner)
- 1973: Mizar (Trybulec)
- 1979: Nuprl (Bates and Constable)
- 1984: Coq (Coquand and Huet)
- 1986: HOL (Gordon)
- 1986: Isabelle (Paulson)
- 1992: Lego (Luo and Pollack)
- 1992: PVS (Owre, Rushby and Shankar)
- 2005: Matita (Asperti)
- 2007: Agda (Norell) *2009: Dedukti (Boespflug)*
- 2010: CoqMT (Strub)

Examples of machine-checked proofs

- 2000: fundamental theorem of algebra (Geuvers et al)
- 2005: 4-color theorem (Gonthier)
- 2006: formal verification of a C compiler back-end (Leroy et al)
- 2006: rewriting theory (CoLoR, Coccinelle, CeTA)
- 2009: formal verification of an OS kernel (NICTA)
- 2012?: 1998 Hales proof of Kepler conjecture (Flyspeck project)
- 2012?: 1962 Feit-Thompson odd order theorem (Gonthier et al)

What is a proof? Deduction vs Computation

- Purely axiomatic approach: every thing is defined using axioms

$$\begin{aligned}(\forall x)\ x+0 &= x\\(\forall x)(\forall y)\ x+(\mathsf{s}y) &= \mathsf{s}(x+y)\end{aligned}$$

Even a statement like "$\mathsf{s}0+\mathsf{s}0=\mathsf{ss}0$" requires a long proof

What is a proof? Deduction vs Computation

- Purely axiomatic approach: every thing is defined using axioms

$$\begin{aligned}(\forall x)\ x+0 &= x\\ (\forall x)(\forall y)\ x+(\mathsf{s}y) &= \mathsf{s}(x+y)\end{aligned}$$

  Even a statement like "$\mathsf{s}0+\mathsf{s}0=\mathsf{ss}0$" requires a long proof

- Mixed approach: deduction modulo some decidable congruence

  The proof of "$\mathsf{s}0+\mathsf{s}0=\mathsf{ss}0$" reduces to reflexivity
  (equality on closed arithmetic expressions is decidable)

  - in dependent type systems, more terms are definable
  - reduce the gap with informal mathematical practice

What congruence?

if the object language contains λ-expressions (Church 1940):

$$x \mid \lambda xt \mid tu$$

one may consider the β-congruence:

$$(\lambda xt)u =_\beta t_x^u$$

What congruence?

if the object language contains first-order terms:

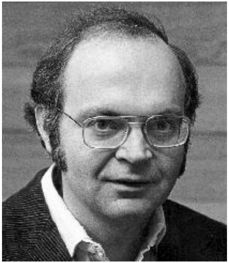
$$x \mid \mathsf{f}\, t_1 \ldots t_n$$

one may consider some equational theory E:

$$l_1 = r_1 \quad \ldots \quad l_n = r_n$$

How to prove that a congruence is decidable?

given a congruence E, find a relation R that is (Knuth 1967):



- decidable
- terminating: $\nexists$ infinite R-sequence
- confluent: R-congruent terms are R-joinable
- correct: R-congruent terms are E-congruent
- complete: E-congruent terms are R-congruent

Rewriting and completion

The basic idea is to orient equations $l = r$ into rewrite rules $l \rightarrow r$ (replacement becomes unidirectional)

> "Rewrite systems are directed equations used to compute by repeatedly replacing subterms of a given formula with equal terms until the simplest form possible is obtained." (DJ'90)

In 1967, Knuth devised a completion algorithm that, given a set of first-order equations E, tries to build a set of first-order rules R that is terminating, confluent, correct and complete

Remark: $\rightarrow_\beta$ has all the above properties except termination

Descendants

λ-calculus and first-order rewriting led to two important families of programming languages:

- functional programming languages: Lisp (1958), ML (1972), Haskell (1990), OCaml (1996), F# (2005), ...
- rewriting-based languages: OBJ (1976), Elan (1994), Maude (1996), ...

Descendants

λ-calculus and first-order rewriting led to two important families of programming languages:

- functional programming languages: Lisp (1958), ML (1972), Haskell (1990), OCaml (1996), F# (2005), ...
- rewriting-based languages: OBJ (1976), Elan (1994), Maude (1996), ...

"One framework to rule them all?"

Higher-order rewriting

higher-order rewriting is rewriting on λ-terms

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

- Combinatory Reduction Systems (CRS) (Klop 1980)
- Expression Reduction Systems (ERS) (Khasidashvili 1990)
- Higher-order Rewrite Systems (HRS) (Nipkow 1991)
  - simply-typed λ-terms in β-normal η-long form
  - matching modulo $\alpha\beta\eta$



Higher-order rewriting

- Higher-order Algebraic Specification Languages (HOASL) (Jouannaud, Okada 1991)
  - arbitrary terms
  - matching modulo α

"To infinity ... and beyond!"

- λ-calculus with patterns (van Oostrom 1990)
- ρ-calculus (Cirstea, Kirchner 1998)
- pattern calculus (Jay, Kesner 2004)

What congruence?

- β-reduction (Church 1940, ...)
  Automath, Coc, Isabelle
- β-reduction + induction (Tait 1967, ...)
  LCF, Nuprl, Coq, HOL, Lego, Matita, Agda
- β-reduction + first-order rewriting (Breazu-Tannen 1988, ...)
- β-reduction + higher-order rewriting
  (Barbanera, Fernández, Geuvers 1993, ...)
  Coq+CiME, Cac, Dedukti
- β-reduction + induction + FO decision procedures
  (Owre, Rushby and Shankar 1992, Stehr 2002, Strub 2008)
  PVS, CoqMT

Problem

how to prove the termination of $\rightarrow_\beta \cup \rightarrow_\mathcal{R}$?

Problem

how to prove the termination of $\rightarrow_\beta \cup \rightarrow_{\mathcal{R}}$?

remark: termination is not modular! (Toyama 1987)

Problem

how to prove the termination of $\rightarrow_\beta \cup \rightarrow_\mathcal{R}$?

remark: termination is not modular! (Toyama 1987)

if $\mathcal{R}$ is first-order, $\mathcal{R}$ cannot create new β-redexes and $\rightarrow_\beta \cup \rightarrow_\mathcal{R}$ terminates on all $\mathcal{R}$*-stable* subset of $\mathrm{SN}(\rightarrow_\beta)$ (a weak form of typing) (Dougherty 1991)

Termination of β-reduction alone?

in the simply-typed λ-calculus:

- $\rightarrow_\beta$ can be proved terminating by a direct induction on the type of the substituted variable (Sanchis 1967, van Daalen 1980)
  does not extend to rewriting where the type of substituted variables can increase, *e.g.* $\mathsf{f}(\mathsf{c}x) \rightarrow x$ with $x : \mathsf{A} \Rightarrow \mathsf{B}$

Termination of β-reduction alone?

in the simply-typed λ-calculus:

- $\rightarrow_\beta$ can be proved terminating by a direct induction on the type of the substituted variable (Sanchis 1967, van Daalen 1980)
  does not extend to rewriting where the type of substituted variables can increase, *e.g.* $\mathsf{f}(\mathsf{c}x) \rightarrow x$ with $x : \mathsf{A} \Rightarrow \mathsf{B}$
- λI-terms can be interpreted by hereditarily monotone functions on $\mathbb{N}$ (Gandy 1980)
  can be used to build interpretations but these interpretations can also be obtained from an extended computability proof (van de Pol 1996)

Outline

Computability

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

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

Computability predicates

there are different definitions of computability (Tait Sat, Girard Red, Parigot SatInd, Girard $\mathrm{Bi}\bot$) but Girard's definition Red is better suited for handling *arbitrary* rewriting

Computability predicates

there are different definitions of computability (Tait Sat, Girard Red, Parigot SatInd, Girard $\mathrm{Bi}\bot$) but Girard's definition Red is better suited for handling *arbitrary* rewriting

let Red be the set of P such that:

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

neutral $=$ not head-reducible after application (λxu is *not* neutral)

Computable terms

Red is a complete lattice for set inclusion 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 $u : U$, 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:

- termination: $P \subseteq \mathrm{SN}(\rightarrow)$
- stability by reduction: $\rightarrow(P) \subseteq P$
- if t is neutral and $\rightarrow(t) \subseteq P$ then $t \in P$

  (taking $\mathsf{f}\vec{t}$ neutral if $|\vec{t}| \geq \sup\{|\vec{l}| \mid \mathsf{f}\vec{l} \rightarrow r \in \mathcal{R}\}$)

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:

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

Theorem: Given a set $\mathcal{R}$ of rules, the relation $\rightarrow_\beta \cup \rightarrow_\mathcal{R}$ terminates if every rule of $\mathcal{R}$ is of the form $\mathsf{f}\vec{l} \rightarrow r$ with $r \in \mathrm{CC}_{\mathcal{R},\mathsf{f}}(\vec{l})$, where $\mathrm{CC}_{\mathcal{R},\mathsf{f}}(\vec{l})$ is a set of terms that are $\mathcal{R}$-computable whenever $\vec{l}$ 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. . . :-(

example: with $\mathsf{c} : (\mathsf{B} \Rightarrow \mathsf{A}) \Rightarrow \mathsf{B}$ and $\mathsf{f} : \mathsf{B} \Rightarrow (\mathsf{B} \Rightarrow \mathsf{A})$, $\rightarrow_\beta \cup \rightarrow_\mathcal{R}$ with $\mathcal{R} = \{\mathsf{f}(\mathsf{c}x) \rightarrow x\}$ does not terminate (Mendler 1987)

with $w = \lambda x \mathsf{f} x x : \mathsf{B} \Rightarrow \mathsf{A}$, $w(\mathsf{c}w) \rightarrow_\beta \mathsf{f}(\mathsf{c}w)(\mathsf{c}w) \rightarrow_\mathcal{R} w(\mathsf{c}w)$

Dealing with subterms

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

example: with $\mathsf{c} : (\mathsf{B} \Rightarrow \mathsf{A}) \Rightarrow \mathsf{B}$ and $\mathsf{f} : \mathsf{B} \Rightarrow (\mathsf{B} \Rightarrow \mathsf{A})$, $\rightarrow_\beta \cup \rightarrow_\mathcal{R}$ with $\mathcal{R} = \{\mathsf{f}(\mathsf{c}x) \rightarrow x\}$ does not terminate (Mendler 1987)

with $w = \lambda x \mathsf{f} x x : \mathsf{B} \Rightarrow \mathsf{A}$, $w(\mathsf{c}w) \rightarrow_\beta \mathsf{f}(\mathsf{c}w)(\mathsf{c}w) \rightarrow_\mathcal{R} w(\mathsf{c}w)$

$\Rightarrow$ 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 matching modulo $\beta\eta$

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

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* 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 $v \in \mathcal{X}$

Dealing with higher-order pattern-matching

Dale Miller (1991): if l is an *higher-order pattern* 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 $C[(\lambda xu)v] \rightarrow_{\beta_0} C[u^v_x]$ 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 $l \rightarrow \lambda xr \in \mathcal{R}$

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

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

$\Rightarrow$ 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

RPO

RPO is a well-founded quasi-ordering (WFQO) on terms extending a WFQO on symbols (Plaisted, Dershowitz 1978)

$$(1)\ \frac{t_i \geq_{\mathrm{rpo}} u}{\mathsf{f}\vec{t} >_{\mathrm{rpo}} u} \qquad (2)\ \frac{(\mathsf{f},\vec{t}) \sqsupset (\mathsf{g},\vec{u}) \quad \mathsf{f}\vec{t} >_{\mathrm{rpo}} \vec{u}}{\mathsf{f}\vec{t} >_{\mathrm{rpo}} \mathsf{g}\vec{u}}$$

where $(\mathsf{f},\vec{t}) \sqsupset (\mathsf{g},\vec{u})$ if $\mathsf{f} > \mathsf{g} \vee (\mathsf{f} \simeq \mathsf{g} \wedge \vec{t}\ (>_{\mathrm{rpo}})_{\mathrm{stat}[\mathsf{f}]}\ \vec{u})$

HORPO

HORPO is a (non-transitive) extension of RPO to λ-terms (Jouannaud, Rubio 1999)

Revisiting (HO)RPO

What is the relation between CC and HORPO?

- both are based on computability
- there are even extensions of HORPO using CC
- CC is defined for a fixed $\mathcal{R}$

Revisiting (HO)RPO

What is the relation between CC and HORPO?

- both are based on computability
- there are even extensions of HORPO using CC
- CC is defined for a fixed $\mathcal{R}$

but CC itself is a relation!

replace $t \in \mathrm{CC}_{\mathcal{R},\mathsf{f}}(\vec{l})$ by $\mathsf{f}\vec{l} >_{\mathrm{CC}(\mathcal{R})} t$

Revisiting (HO)RPO

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

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

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

$$(\mathsf{f},\vec{l}) \sqsupset (\mathsf{g},\vec{t}) \text{ if } \mathsf{f} > \mathsf{g} \vee (\mathsf{f} \simeq \mathsf{g} \wedge \vec{l}\ ((\rhd \cup \rightarrow_\beta \cup \rightarrow_\mathcal{R})^+)_{\mathrm{stat}[\mathsf{f}]}\ \vec{t})$$

...

Revisiting (HO)RPO

$$\mathcal{R} \mapsto \{(\mathsf{f}\vec{l}, r) \mid r \in \mathrm{CC}^{\emptyset}_{\mathcal{R},\mathsf{f}}, \mathrm{type}(\mathsf{f}\vec{l}) = \mathrm{type}(r)\}$$

is a monotone function on the complete lattice of relations

Revisiting (HO)RPO

$$\mathcal{R} \mapsto \{(\mathsf{f}\vec{l}, r) \mid r \in \mathrm{CC}^{\emptyset}_{\mathcal{R},\mathsf{f}}, \mathrm{type}(\mathsf{f}\vec{l}) = \mathrm{type}(r)\}$$

is a monotone function on the complete lattice of relations

the monotone closure of its fixpoint (Tarski 1955):

- contains HORPO
- is equal to RPO when restricted to FO terms!

Revisiting (HO)RPO

$$\mathcal{R} \mapsto \{(\mathsf{f}\vec{l}, r) \mid r \in \mathrm{CC}^{\emptyset}_{\mathcal{R},\mathsf{f}}, \mathrm{type}(\mathsf{f}\vec{l}) = \mathrm{type}(r)\}$$

is a monotone function on the complete lattice of relations

the monotone closure of its fixpoint (Tarski 1955):

- contains HORPO
- is equal to RPO when restricted to FO terms!

$\Rightarrow$ provide a general method to get a powerful termination ordering for any type system

What else?

- rewriting modulo some equational theory
- conditional rewriting (Riba 2006)
- size-based termination
- semantic labelling (Roux 2009)
- dependency pairs

Outline

Type theory and rewriting

Computability closure

- Computability
- Dealing with matching modulo $\beta\eta$
- Revisiting (HO)RPO

Conclusion and perspectives

Conclusion

- deduction modulo is essential for doing large proofs
- deduction modulo rewriting is simple and powerful
- we have criteria/tools for checking termination and confluence
  (see results of last termination competition!)

$\Rightarrow$ we can check the decidability of proof-checking

How to increase our confidence in such a proof system?

- use a machine-checked proof-checker kernel

  Coq (Barras 97), CoqMT (Strub 2010), ...

  $\Rightarrow$ one can use unproved tools to build proofs

How to increase our confidence in such a proof system?

- use a machine-checked proof-checker kernel

  Coq (Barras 97), CoqMT (Strub 2010), ...

  $\Rightarrow$ one can use unproved tools to build proofs

- one can check system properties (termination, confluence, ...) by using external tools providing certificates

  and use machine-checked certificate verifiers

  Rainbow, CiME3 (2006), CeTA (2009)

How to increase our confidence in such a proof system?

- use a machine-checked proof-checker kernel

  Coq (Barras 97), CoqMT (Strub 2010), ...

  $\Rightarrow$ one can use unproved tools to build proofs

- one can check system properties (termination, confluence, ...) by using external tools providing certificates

  and use machine-checked certificate verifiers

  Rainbow, CiME3 (2006), CeTA (2009)

can we go further?

Modules and computation

```
Module Type Group_Sig.
  Parameter t : Type.
  Parameter zero : t.
  Parameter opp : t -> t.
  Parameter add : t -> t -> t.
  Parameter law1 : forall x, add x (opp x) = zero.
  ...
End Nat_Sig.

Module Group_Theory (G : Group_Sig).
  (* the equational properties of add are not part of the congruence! *)
  Theorem Feit_Thompson : ...
  ...
End Group_Theory.

Module Group_X <: Group_Sig.
  Definition t := ...
  ...
  Lemma law1 : forall x, add x (opp x) = zero. Proof. ... Qed.
  ...
End Group_X.

Module Group_X_Theory := Group_Theory Group_X.
```

Modules and computation

```
Module Type Group_Sig.
  Parameter t : Type.
  Parameter zero : t.
  Parameter opp : t -> t.
  Parameter add : t -> t -> t.
  Parameter law1 : forall x, add x (opp x) = zero.
  ...
End Nat_Sig.

Module Group_Theory (G : Group_Sig).
  (* the equational properties of add are not part of the congruence! *)
  Theorem Feit_Thompson : ...
  ...
End Group_Theory.

Module Group_X <: Group_Sig.
  Definition t := ...
  ...
  Lemma law1 : forall x, add x (opp x) = zero. Proof. ... Qed.
  ...
End Group_X.

Module Group_X_Theory := Group_Theory Group_X.
```

Use completion! $\Rightarrow$ the congruence becomes dynamic [Dedukti]

Unorientable equations

some equations may be unorientable (commutativity/associativity)

Unorientable equations

some equations may be unorientable (commutativity/associativity)

$\Rightarrow$ use rewriting with matching modulo some equational theory

Unorientable equations

some equations may be unorientable (commutativity/associativity)

$\Rightarrow$ use rewriting with matching modulo some equational theory

and/or canonical elements only (by construction)

related works:

- canonizers (Shostak 1984)
- normalized types (Courtieu 2001)
- the open calculus of constructions (Stehr 2002)
- construction functions for quotient types [Moca!] (B., Hardin, Weis 2007)

Questions?