higher-order rewriting

Femke van Raamsdonk

July 4, 2007

combinatory logic

introduced by Moses Schönfinkel 1924

terms: \( M, N ::= x | S | K | I | (M N) \)

rules:

\[
\begin{align*}
S X Y Z & \rightarrow (X Z)(Y Z) \\
K X Y & \rightarrow X \\
I X & \rightarrow X
\end{align*}
\]

I can be defined using K and S:

\[
(S K) x \rightarrow (K x)(K x) \rightarrow x
\]
combinatory logic

terms:
\[ M, N ::= x | S | K | I | (M N) \]

rules:
\[
\begin{align*}
S X Y Z \rightarrow & (X Z) (Y Z) \\
K X Y \rightarrow & X \\
I X \rightarrow & X
\end{align*}
\]

I can be defined using K and S:
\[(SKK)x \rightarrow (Kx)(Kx) \rightarrow x\]

reduction may be infinite:
\[(SII)(SII) \rightarrow I(SII)(I(SII)) \rightarrow (SII)(I(SII)) \rightarrow (SII)(SII)\]

lambda calculus

terms:
\[ M, N ::= x | \lambda x. M | (M N) \]

rule:
\[(\lambda x. M) N \rightarrow_{\beta} M[x := N]\]

introduced by Alonzo Church 1936

\[
\begin{align*}
\text{reduction may be infinite:} \\
(\lambda x.x)(\lambda x.x) \rightarrow_{\beta} (\lambda x.x)(\lambda x.x)
\end{align*}
\]
expressivity

both CL and lambda calculus are Turing complete

extending lambda calculus

- lambda calculus with pairing
- lambda-calculus with arithmetic
- lambda calculus with explicit substitutions
- lambda calculus with patterns
- ...

towards higher-order rewriting

both patterns and bound variables

extending CL

to orthogonal (first-order) term rewriting systems (OTRSs), e.g.: 

\[
\begin{align*}
\text{map } F \text{ nil} & \rightarrow \text{ nil} \\
\text{map } F (\text{cons } H \ T) & \rightarrow \text{ cons } (F \ H) \ (\text{map } F \ T)
\end{align*}
\]

to (first-order) term rewriting systems (TRSs), e.g.: 

\[
\begin{align*}
\text{eq}(X, X) & \rightarrow \text{ true} \\
\text{por}(\text{true}, Z) & \rightarrow \text{ true} \\
\text{por}(Z, \text{true}) & \rightarrow \text{ true} \\
\text{por}(\text{false}, \text{false}) & \rightarrow \text{ false}
\end{align*}
\]
higher-order rewriting: rewriting with bound variables

- adds bound variables to first-order rewriting
- adds expressive power and algebraic data-types to λ-calculus

this morning

untyped lambda calculus: terms

- lambda calculus
- towards higher-order rewriting
- termination
- confluence

\[
M, N ::= \lambda x. M | (M N)
\]
untyped lambda calculus: reduction

rule:

\[(\lambda x. M) N \rightarrow_B M[x := N]\]

example:

\[(\lambda x.x x)(\lambda x.x x) \rightarrow_B (\lambda x.x x)(\lambda x.x x)\]

data types in lambda calculus: example

define:

true = \lambda x. y. x
false = \lambda x. y. y
not = \lambda x. x false true

then:

not true = (\lambda x. x false true) true
true false true
false

untyped lambda calculus: properties

- it is Turing complete
- \(\beta\)-reduction is confluent
- every \(\beta\)-reduction can be transformed into one where redexes are contracted in a standard (left-to-right) way
- all developments terminate

creation of redexes

the three ways of creating redexes (Jean-Jacques Lévy):

1. \((\lambda x. \lambda y. M) P Q \rightarrow_B (\lambda y. M[x := P]) Q\)
2. \((\lambda x. x) (\lambda y. M) N \rightarrow_B (\lambda y. M) N\)
3. \((\lambda z. C[z Q]) (\lambda x. P) \rightarrow_B C[(\lambda x. P) Q]\)
creation of redexes: 1


devolution: idea

only residuals of redexes in the initial term are contracted
so created redexes are not contracted
development: example

\[(\lambda x. f x x)(\lambda y. y) a \rightarrow_\beta f((\lambda y. y) a)((\lambda y. y) a) \rightarrow_\beta f a((\lambda y. y) a) \rightarrow_\beta f a a\]

infinite behaviour is caused by created redexes

development: non-example

\[(\lambda x. \lambda y. f x y) a b \rightarrow_\beta (\lambda y. f a y) b \rightarrow_\beta f a b\]

infinite behaviour is caused by the 3rd type of created redexes

finiteness of developments

finiteness of superdevelopments
developments: via the underlined lambda calculus

the underlined lambda calculus is terminating (1)

the underlined lambda calculus is terminating (2)

the underlined lambda calculus is terminating (3)

theorem: σ terminating ⇒ M σ terminating

proof: Induction on the definition of terms.
1. M = x
2. M = λx. P
   IH ⇒ P σ is terminating

note: λx. M is not an underlined term
no interaction between P and Q
the underlined lambda calculus is terminating (4)

**Theorem:** $\sigma$ terminating $\Rightarrow M^\sigma$ terminating

**Proof:** Induction on the definition of terms.

1. $M = x$
2. $M = \lambda x. P$
3. $M = (P Q)$
4. $M = (\lambda x. P) Q$
   - head redex is contracted
     - IH $\Rightarrow P^\sigma[x:=Q^\sigma] = P^\sigma[x:=Q^\sigma]$ is terminating
   - head redex is not contracted
     - IH $\Rightarrow P^\sigma$ and $Q^\sigma$ are terminating

---

**Exercise**

how can redexes be created in CL?

---

**Simple Types**

$$A, B ::= b \mid (A \rightarrow B)$$

**Examples:**

- nat
- nat $\rightarrow$ nat $\rightarrow$ nat
- (nat $\rightarrow$ nat) $\rightarrow$ bool
simply typed terms

1. \(x : A\)
   if \(x\) is a variable of type \(A\)

2. \(f : A\)
   if \(f\) is a constant of type \(A\)

3. \(\lambda x. M : A \to B\)
   if \(M : B\) and \(x\) a variable of type \(A\)

4. \((F M) : B\)
   if \(F : A \to B\) and \(M : A\)
   there is no self-application as in \(\lambda x. xx\)

example: predicate logic

normal: \(\forall x. \text{plays}(x)\)
HOS: \(\forall (\lambda x.(\text{plays }x))\)
using:
\[
\begin{align*}
  x &: D \\
  \text{plays} &: D \to F \\
  \forall &: (D \to F) \to F
\end{align*}
\]

higher-order syntax

all bindings are expressed using \(\lambda\)

example: untyped lambda calculus

normal: \(\lambda x. x y\)
HOS: \(\text{abs} (\lambda x. (\text{app} x y))\)
using:
\[
\begin{align*}
  x &: T \\
  y &: T \\
  \text{app} &: (T \to (T \to T)) \\
  \text{abs} &: (T \to (T \to T)) \to T
\end{align*}
\]
beta reduction

$$(\lambda x. M) N \rightarrow_\beta M[x := N]$$

but now $\beta$-reduction terminates
(more this afternoon !)

different approaches

do we want to see the substitution explicitly ?

YES
then we work modulo $\alpha$

NO
then we work modulo $\alpha \beta$ (and $\eta$)

higher-order rewriting: systems

- modulo $\alpha$
  extensions of $\lambda$-calculus
  AFS (Jouannaud and Okada 1991)

- modulo $\alpha / \beta / \eta$
  CRS (Klop 1980)
  ERS (Khasidashvili 1990)
  HOER (Wolfram 1991)
  HRS (Nipkow 1991)

beta reduction implements substitution

$$\forall x. P(x) \quad \forall (\lambda x. (P x))$$

$$\downarrow$$

$$\forall (\lambda x. ((\lambda u. \text{plays } u) \cdot x))$$

$$\downarrow$$

$$\forall (\lambda x. \text{plays } x)$$

$$\forall x. \text{plays } x \quad \forall (\lambda x. \text{plays } x)$$
restricted eta expansion

the \( \eta \)-expansion rule:
\[ M \rightarrow_\eta \lambda x. (M x) \]

side conditions:
- types ok
- fresh variables
- do not create \( \beta \)-redexes

unique representatives

every \( \beta \eta \) equivalence class has a unique representative:
its \( \beta \) long normal form

long normal form

a \( \beta \)-normal form:
\[ \lambda x_1 \ldots \lambda x_m. (c M_1 \ldots M_n) \]

its long normal form:
\[ \lambda x_1 \ldots \lambda x_m \lambda x_{m+1} \ldots \lambda x_p. (c M'_1 \ldots M'_n x'_{m+1} \ldots x'_p) \]

requirements on a rewrite rule \( l \rightarrow r \)?

as in TRSs:
- all free variables of \( r \) occur in \( l \)
- \( l \) is not a free variable

in addition:
- \( l \) and \( r \) have the same type
- \( l \) and \( r \) are in \( \beta \eta \) normal form
decidability

is it decidable whether a rule can be applied?
are critical pairs decidable?

is matching decidable?

higher-order matching problem:
\[ s =_{\beta, \eta} l'^{\tau} \]
results:
- 3rd order: decidable (Dowek 1994)
- 4th order: decidable (Padovani 2001)
- general: decidable (Stirling 2006)
- modulo \( \beta \): undecidable (Loader 2003)

unification is undecidable

higher-order unification problem:
\[ g^\sigma =_{\beta, \eta} l'^{\tau} \]
result:
- undecidable (Huet 1976)

restriction of left-hand sides to patterns

unification is decidable for patterns (Miller 1991)
pattern:
all arguments of a free variable are different bound variables
not a pattern:
\[ f(Z(Z(x))) \]
why is it called higher-order?

order of a rewrite system:
maximum order of a free variable in a rule

order of a type:
1. order(b) = 1
2. order(A₁ → ... → Aₙ → b) = max{order(Aᵢ)} + 1

order of a rewrite system: examples

0th order:
   a → b

1st order:
   plus(0, Y) → Y

2nd order:
   map(λx. F(x), nil) → nil

3rd order:
   f(λx. Z(x)) → Z(λa. a)

outermost-fair rewriting: idea

Part IV
termination

every outermost redex is eventually contracted

example of a non-outermost-fair reduction:

```plaintext
a  →  a
h(x)  →  h'(x)
f(x, h'(y))  →  c
```
outermost-fair rewriting is normalizing

for TRSs that are almost orthogonal
(O’Donnell 1977)

for HRSs that are weakly orthogonal and fully extended
(van Oostrom 1999)

why is higher-order more difficult?

a non-outermost redex can create an outermost redex
so we will need an additional restriction

proof idea

induction on the maximum length to normal form
outermost-fair is stable under projection

normal form

example

rules:

\[ \begin{align*}
  f(\lambda x. Z) & \rightarrow a \\
  g(X) & \rightarrow g(X) \\
  h(X) & \rightarrow a
\end{align*} \]

an infinite outermost-fair reduction of a normalizing term:

\[ \begin{align*}
  \lambda x. Z & \rightarrow f \\
  \lambda x. Z & \rightarrow \lambda x. Z \\
  \lambda x. Z & \rightarrow \lambda x. Z \\
  \vdots
\end{align*} \]
fully applied: idea

avoid the occur check:

- a free variable has as arguments
- all variables that are at that position bound

why is weakly orthogonal difficult?

- an outermost redex may (as such) disappear
  - because it is contracted
  - because it is not a redex anymore
  - because it is not outermost anymore

Part V

confluence

two coinitial rewrite sequences can be joined
why confluence?

confluence yields uniqueness of normal forms

\[ \begin{align*}
    a & \quad \to^* \quad b_1 \\
    a & \quad \to^* \quad b_2 \\
\end{align*} \]

methods to prove confluence of HRSs

- orthogonality \Rightarrow \text{confluence} \\

- joinability of critical pairs \Rightarrow \text{local confluence} \\
  (Nipkow 1991) \\
  Joinability of critical pairs and termination \Rightarrow \text{confluence}

- critical pairs development closed \Rightarrow \text{confluence} \\
  (van Oostrom 1996)

orthogonality

rewrite steps do not destroy each other

orthogonality: definition

rewrite steps do not destroy each other
is guaranteed by two conditions on the rewrite rules:

- left-hand sides are linear
- left-hand sides are non-overlapping
left-linearity: idea

no check for equality

\[ f = \left\{ \begin{array}{ll}
\infty & \rightarrow S(\infty) \\
\text{eq}(X, X) & \rightarrow \text{true} \\
\text{eq}(X, S(X)) & \rightarrow \text{false}
\end{array} \right. \]

is not confluent:

\[ \left\{ \begin{array}{l}
\infty \quad \rightarrow \text{true} \\
\infty \quad \rightarrow S(\infty)
\end{array} \right. \]

left-linearity: definition

a free variable occurs at most once in a left-hand side

non-left-linearity may yield non-confluence

non-overlapping: definition

two left-hand sides are not unifiable on a non-variable position

unification is decidable because left-hand sides are patterns
overlap may yield non-confluence

rules:

\[ a \rightarrow b \]
\[ a \rightarrow c \]

divergence:

orthogonality implies confluence: TML proof method
define relation \( \rightsquigarrow \) with

\[ \rightsquigarrow \text{ has the diamond property} \]
\[ \rightsquigarrow^* = \rightarrow^* \]

orthogonality implies confluence: proof methods

- using developments and the parallel moves lemma
- Tait-Martin-Löf method using an inductive definition

what do we take for \( \rightsquigarrow \)?

for TRSs: parallel reduction \( \parallel \rightarrow \)
parallel reduction: example

rules:

\[ f(X) \rightarrow g(X, X) \]

\[ a \rightarrow b \]

parallel step:

\[ f \quad g \]

\[ a \quad a \]

\[ b \quad b \]

parallel reduction for HRSs: no diamond property

\[ f(\lambda x. Z(x)) \rightarrow Z(Z(a)) \]

\[ g(X) \rightarrow h(X) \]

parallel reduction for lambda: no diamond property

\[ \lambda x. (\lambda y. x) \rightarrow (\lambda y. 1) \]

\[ \lambda x. (\lambda y. x) \rightarrow (\lambda x. 1) \]

parallel reduction does not have the diamond property

because residuals of parallel redexes may be nested
what do we take for $\rightarrow$?

for HRSs: multistep reduction $\rightarrow$

how do we proceed?

- define (inductively) $\rightarrow^*$
- prove $\rightarrow^* = \rightarrow^*$
- define (inductively) $s^*$ with $s \rightarrow^* s^*$
- prove (inductively) if $s \rightarrow^* s'$ then $s' \rightarrow^* s^*$ (triangle)

multistep reduction: example

rule:

$f(X) \rightarrow g(X)$

multistep:

$h(f(f(a)), f(f(a))) \rightarrow h(g(g(a)), g(g(a)))$

parallel step:

$h(f(f(a)), f(f(a))) \rightarrow h(g(f(a)), g(f(a)))$

multistep reduction for lambda calculus

1. variable

$x \rightarrow x$

2. abstraction

$M \rightarrow M'$

$\lambda x. M \rightarrow \lambda x. M'$

3. application

$M \rightarrow M' \quad N \rightarrow N'$

$(M N) \rightarrow (M' N')$

4. beta

$M \rightarrow M' \quad N \rightarrow N'$

$(\lambda x. M) N \rightarrow M'[x := N']$
multistep reduction for lambda calculus: example

\((\lambda y. y) x \rightarrow x x\)
\((\lambda z. z) a \rightarrow a\)
\((\lambda x. (\lambda y. y) x)((\lambda z. z) a) \rightarrow a a\)

multistep reduction for HRSs: example

rules:

\[ a \rightarrow b \]
\[ f(x) \rightarrow g(x) \]

multisteps:

\[ a \rightarrow b \]
\[ f(a) \rightarrow g(b) \]

multistep reduction for HRSs

1. variable
\[
\frac{\bar{z} \rightarrow \bar{t}}{x(\bar{z}) \rightarrow x(\bar{t})}
\]
2. function
\[
\frac{\bar{z} \rightarrow \bar{t}}{f(\bar{z}) \rightarrow f(\bar{t})}
\]
3. abstraction
\[
\frac{s \rightarrow t}{\lambda x. s \rightarrow \lambda x. t}
\]
4. rule
\[
\frac{s \rightarrow t}{\sigma \rightarrow \tau}
\]

difference between parallel reduction and multistep

parallel reduction:

\[
\frac{f^p \rightarrow r^p}{f^p \rightarrow r^p}
\]

multistep:

\[
\frac{\sigma \rightarrow \tau}{f^p \rightarrow r^p}
\]
multistep simulates rewriting

\[ \rightarrow^* = \rightarrow^* \]

\[ M^* \text{ for lambda calculus} \]

intuition: contract all redexes in \( M \)
goal: universal common reduct

1. \( (x(s_1, \ldots, s_n))^* = x(s_1^*, \ldots, s_n^*) \)
2. \( (f(s_1, \ldots, s_n))^* = f(s_1^*, \ldots, s_n^*) \)
3. \( (\lambda x.s)^* = \lambda x. s^* \)
4. \( (r)^* = r^* \)

\[ s^* \text{ for HRSs} \]

intuition: contract all redexes in \( s \)
goal: universal common reduct

1. \( (x(s_1, \ldots, s_n))^* = x(s_1^*, \ldots, s_n^*) \)
2. \( (f(s_1, \ldots, s_n))^* = f(s_1^*, \ldots, s_n^*) \)
3. \( (\lambda x.s)^* = \lambda x. s^* \)
4. \( (r)^* = r^* \)

\[ \triangle: s^* \text{ is a universal common reduct} \]

lemma: if \( s \rightarrow s' \) then \( s' \rightarrow s^* \)
proof:
induction on the definition of \( \rightarrow^* \)

\[ \begin{array}{cc}
\text{s} & \text{s}^* \\
\downarrow & \downarrow \\
\text{s'} & \text{s'}^* \\
\end{array} \]
orthogonality implies confluence

- multistep reduction instead of parallel reduction
- alternative: use developments instead of inductive definitions
- extends to the weakly orthogonal case

exercise

is it possible to define $s^*$ for parallel reduction?
prove the diamond property of $\rightarrow\Rightarrow$ without $s^*$

critical pairs: idea

critical pairs arise from most general overlaps

rules:

\[
\begin{align*}
&f(X) \rightarrow a \\
&h(\lambda x. f(g(x))) \rightarrow b
\end{align*}
\]

critical pair:

\[
\begin{array}{c}
\text{critical pairs: lemma} \\
\text{all critical pairs joinable } \Rightarrow \text{local confluence}
\end{array}
\]

rules:

\[
\begin{align*}
&f(X) \rightarrow a \\
&h(\lambda x. f(g(x))) \rightarrow b \\
&b \rightarrow h(\lambda x. a)
\end{align*}
\]

joinable critical pair:
development closed: idea

rules:

- $f(g(\lambda x. Z(x))) \rightarrow f(Z(a))$
- $g(\lambda x. h'(x)) \rightarrow h'(h(a))$
- $h(X) \rightarrow h'(X)$

concluding remarks

some concepts for higher-order rewriting:
- multistep reduction instead of parallel reduction
- fully applied rewrite rules

some results for higher-order rewriting:
- critical pair lemma
- orthogonality yields confluence
- standardization
- recursive path ordering
- outermost-fair rewriting is normalizing

for HRSs:
all critical pairs development closed $\Rightarrow$ confluence