Vincent van Oostrom
Kees-Jan van de Looij
Marijn Zwitserlood
Universiteit Utrecht
Department of Philosophy
Contents

1 Goal

1.1 Named ................................................. 5
1.2 De Bruijn indexed ..................................... 7
1.3 Unary indexed ......................................... 9
1.4 Scoped .................................................. 10
1.5 $\beta$-reduction ....................................... 11
1.6 $\beta$-reduction: replication ....................... 12
1.7 $\beta$-reduction: yields generalised term ........ 13
1.8 $\beta$-reduction: extrusion ......................... 14
1.9 Generalised terms inductively .................... 15
1.10 Replication recursively .......................... 16
1.11 Extrusion iteratively/recursively ............... 17

2 Interaction net implementation .................. 18

2.1 From terms to nets ................................... 19
2.2 From terms to nets: translating zero ............ 20
2.3 From terms to nets: translating successor ...... 21
2.4 From terms to nets: translating application ... 22
2.5 From terms to nets: translating abstraction ... 23
2.6 From terms to nets: useless parts ............... 24
2.7 From terms to nets: useful part .................. 25
2.8 Net reduction: $x$-rules ............................. 26
1. Goal

Extend optimality theory and practice to HRS
1. Goal

Extend optimality theory and practice to HRS

Subgoals:

- TRS optimality theory done (Terese book)
- \(\lambda\)-calculus optimality here
1. Goal

Extend optimality theory and practice to HRS

Subgoals:

- TRS optimality
theory done (Terese book)

- $\lambda$-calculus optimality
here

Literature: Lévy, Lamping, Gonthier, Abadi, Danos, Mackie, Regnier, Asperti, Laneve, Guerrini, Mairson, Lawall, ... Here: analysis of De Bruijn’s calculational approach
1.1. Named

Church numeral $2 = \lambda x.\lambda y.x(y)$

\[\lambda x \quad \lambda y \quad @ \quad @ \quad x \quad x \quad y\]
1.1. Named

Church numeral $2 = \lambda x.\lambda y. x(xy)$

Application is exponentiation

$$2^2 \rightarrow^\beta \lambda y.2(2y)$$

$$\rightarrow^\beta \lambda y.2\lambda x.y(yx)$$

$$\rightarrow^\beta \lambda y.\lambda z.(\lambda x.y(yx))((\lambda x.y(yx))z)$$

$$\rightarrow^\beta \lambda y.\lambda z.(\lambda x.y(yx))(y(yz))$$

$$\rightarrow^\beta \lambda y.\lambda z.y(y(y(yz))) = 4$$
1.2. De Bruijn indexed

De Bruijn indexed Church numeral $2 = \lambda \lambda 1(10)$

\[
\begin{array}{c}
\lambda \\
\lambda \\
@ & @ \\
1 & 1 & 0 \\
\end{array}
\]
1.2. De Bruijn indexed

De Bruijn indexed Church numeral \( \overline{2} = \lambda \lambda 1(10) \)

\[
\begin{array}{c}
\lambda \\
\lambda \\
@ \quad @ \\
1 \quad 1 \quad 0
\end{array}
\]

De Bruijn indexed exponentiation

\[
\overline{2} \overline{2} \rightarrow_\beta \lambda 2(20) \\
\rightarrow_\beta \lambda 2 \lambda 1(10) \\
\rightarrow_\beta \lambda \lambda (\lambda 2(20))((\lambda 2(20))0) \\
\rightarrow_\beta \lambda \lambda (\lambda 2(20))(1(10)) \\
\rightarrow_\beta \lambda \lambda 1(1(10)))
\]
1.3. Unary indexed

Unary indexed $2 = \lambda \lambda (S0)((S0)0)$
1.4. Scoped

Scoped $2 = \lambda \lambda (S0)((S0)0)$
1.4. Scoped

Scoped $\bar{2} = \lambda\lambda(S0)((S0)0)$

Scoping is matching
binding is match with 0
end-of-scope is match with $S$
$\lambda$-terms are context-free trees
1.5. $\beta$-reduction

how to reduce?
1.6. $\beta$-reduction: replication

\[ \begin{align*}
\lambda & \lambda \\
\lambda & \lambda \\
S & S & 0 \\
0 & 0
\end{align*} \]

put argument if matching 0
erase argument if matching $S$
1.7. $\beta$-reduction: yields generalised term

(successors of subterms!)
1.8. $\beta$-reduction: extrusion

extrude scopes to yield unary indexed terms
1.9. Generalised terms inductively

\[ t \in G\Lambda \ ::= \ 0 \ | \ St \ | \ \lambda t \ | \ tt \]

De Bruijn indexed \( \Lambda \)-terms: repeated successors of 0
1.9. Generalised terms inductively

\[ t \in G\Lambda ::= 0 \mid St \mid \lambda t \mid tt \]

De Bruijn indexed \(\Lambda\)-terms: repeated successors of 0

\[
\begin{align*}
S_i \vdash 0 & \quad S_i \vdash St \\
i \vdash t & \\
S_i \vdash t & \\
i \vdash \lambda t & \\
i \vdash t_1 t_2 & \quad @} \\
i \vdash t_1 & \\
i \vdash t_2 & \\
\end{align*}
\]

\(t\) well-formed under stack \(i\)
1.9. Generalised terms inductively

\[ t \in \mathcal{G}\Lambda ::= 0 \mid St \mid \lambda t \mid tt \]

De Bruijn indexed \( \Lambda \)-terms: repeated successors of 0

\[
\begin{align*}
S_i \vdash 0 & \quad S_i \vdash St \quad i \vdash \lambda t \quad i \vdash t_1 t_2 \quad \alpha \\
\hline
S_i \vdash t & \quad \lambda \\
S_i \vdash \lambda \lambda (S0)((S0)0) & \\
S0 \vdash \lambda (S0)((S0)0) & \\
SS0 \vdash (S0)((S0)0) & \quad \alpha \\
SS0 \vdash S0 & \\
S \quad \lambda \\
S0 \vdash 0 & \\
S \quad \lambda \\
SS0 \vdash S0 & \\
SS0 \vdash (S0)0 & \quad \alpha \\
SS0 \vdash 0 & \\
S0 \vdash 0 & \\
\end{align*}
\]

\( t \) well-formed under stack \( i \)
1.10. Replication recursively

\[(\lambda t)s \rightarrow t[s]^0\]

replication \(t[s]^i\) of argument \(s\) in \(t\) at depth \(i\)

\[
0[s]^0 = s \quad \text{put argument}
\]

\[
0[s]^{Si} = 0
\]

\[
(St)[s]^0 = t \quad \text{erase argument}
\]

\[
(St)[s]^{Si} = St[s]^i
\]

\[
(\lambda t)[s]^i = \lambda t[s]^{Si}
\]

\[
(t_1t_2)[s]^i = t_1[s]^it_2[s]^i \quad \text{duplicate argument}
\]

\[
2 \ 2 = (\lambda \lambda (S0)((S0)0))^2 \rightarrow \lambda (S2)((S2)0)
\]
1.11. Extrusion iteratively/recursively

\[ S\lambda t \rightarrow_\lambda \lambda t^{S0} \quad \text{minimal lifting } t^i \]

\[
\begin{align*}
t^0 &= St \\
0^{S_i} &= 0 \\
(S t)^{S_i} &= St^i \\
(\lambda t)^{S_i} &= \lambda t^{S S_i} \\
(t_1 t_2)^{S_i} &= t_1^{S_i} t_2^{S_i}
\end{align*}
\]

\[ \lambda(S_2)((S_2)0) \rightarrow \lambda2(20) \]
2. Interaction net implementation

applicator  abstractor  delimiter  duplicator  eraser

@  argument  λ  bind  i

function  body

•s are ports
•s are principal ports
i index (default 0)
2. Interaction net implementation

applicator  abstractor  delimiter  duplicator  eraser

\[ \text{@} \quad \text{argument} \quad \lambda \quad \text{bind} \quad \downarrow i \quad \downarrow i \quad \bigcirc \]

- \( \circ \)'s are ports
- \( \bullet \)'s are principal ports
- \( i \) index (default 0)

\( \text{@} \) applicator, \( \lambda \) abstractor for local \( \beta \)
\( \nabla_i \) duplicator, \( \bigcirc \) eraser for explicit local replication
\( \downarrow_i \) scope delimiter for explicit local extrusion
2.1. From terms to nets

\[ \Lambda \rightarrow \text{IN} \] recursively maps closed terms to ‘closed’ nets

\[ i \vdash t \] mapped to net with \( i + 1 \) free ports
2.2. From terms to nets: translating zero
2.3. From terms to nets: translating successor
2.4. From terms to nets: translating application
2.5. From terms to nets: translating abstraction
2.6. From terms to nets: useless parts
2.7. From terms to nets: useful part
2.8. Net reduction: x-rules

\[ f' = f' \]

Drawing convention

\[ f', g' \] identical to or updates of \( f, g \) (distinct)

Update: increment of \( i \) iff other \( \lambda \) or \( \sqcup_j (i \geq j) \)
2.8. Net reduction: x-rules

\[ f, g \text{ identical to or updates of } f, g \text{ (distinct)} \]

Update: increment of \( i \) iff other \( \lambda \) or \( \sqcup j \) \( (i \geq j) \)
2.9. Net reduction: Beta

\[ \lambda @ \rightarrow \Box @ \rightarrow \Box = \]

\text{disintegrate} \quad \text{annihilate}

\[ B = \text{Beta} + x \]
2.10. Net reduction: example Beta-step
2.11. Net reduction: example x-normalisation
2.12. Net reduction: example B-normalisation
2.13. Net reduction: example normal form

\[ \lambda \]

\[ 1 \quad 1 \quad 1 \quad 1 \]

\[ @ \quad @ \]

\[ 1 \]

\[ 1 \]
2.14. From nets to terms

Why does this net represent 4?
2.14. From nets to terms

Why does this net represent $4$?

read-back map $\triangle : \text{IN} \rightarrow \Lambda$ in three phases:

1. **unwind**: replicate, extrude applications
2. **scope remove**: remove redundant scopes
3. **cut loop**: elide duplicators yielding tree
2.15. Read-back: rotating port of application
2.16. Read-back: replicate, extrude applications
2.17. Read-back: rotating port of delimiter
2.18. Read-back: remove redundant scopes
2.19. Read-back: cut loops
2.20. Read-back: elide duplicators
3. Correctness

No syntactic proof yet . . .
3. Correctness

No syntactic proof yet . . .

Semantic proof via stack-based read-back

- Invariant under reduction
- Coincides with normal form (tree)
3.1. Push down automaton

Transitions between (ports on) edges of net

\[ b \in B ::= i\delta \] blocks with replication info

\[ \ell \in L ::= b\ell \] levels with scoping info

\[ \sigma \in S ::= i\ell \] stack with read-back info

\( \delta \) ranges over directors \( \{L, R\} \)
3.1. Push down automaton

Transitions between (ports on) edges of net

\[ b \in B ::= i\delta \] blocks with replication info

\[ \ell \in L ::= b\ell \] levels with scoping info

\[ \sigma \in S ::= i\ell \] stack with read-back info

\[ \delta \] ranges over directors \( \{L, R\} \)

Constraints on PDA transitions:
3.2. Example path read-back

So term of shape $\lambda\lambda (?)(?(?0)))$ (in fact $?$s are $1$s)
4. Optimality

Implementation is **optimal** in sense of Lévy

- **No needless work** via outermost strategy
- **No double work** because of local replication

Why?
4. Optimality

Implementation is optimal in sense of Lévy

- No needless work via outermost strategy
- No double work because of local replication

Why?

Same abstract algorithm as extant implementations
5. Efficiency

- Prototype implementation `lambdascope`
- Trivial: 1 day of programming (in Java)
- As efficient as (optimised version of) BOHM (outperforms functional languages on same examples)
- Solves oracle problems (brackets/croissants)
- Many possibilities for optimisation closedness, types etc.
6. Future work

- Fast implementation
- Ideal explicit substitution calculus (orthogonal, preservation of termination etc.)
- Type system for graphs
- Extension to higher-order rewriting
6. Future work

• Fast implementation

• Ideal explicit substitution calculus
  (orthogonal, preservation of termination etc.)

• Type system for graphs

• Extension to higher-order rewriting

Attend RTA 2004

Registration is open now