Sharing in the weak lambda-calculus (2)

Jean-Jacques Lévy
INRIA

Joint work with Tomasz Blanc and Luc Maranget
Happy birthday Henk!
Happy birthday Henk!
Happy birthday Henk!
HISTORY
Sharing in the lambda-calculus

• goal:
  - efficient implementations of functional languages
  - by “functional languages”, we mean here logical systems (Coq, Isabelle, etc)
  - although real functional languages use more environment machines
  - but it could be useful for partial evaluation
Sharing in the lambda-calculus

• Lamping’s algorithm [91]:
  - **optimal** in total number of beta-reductions
  - sharing **contexts**
  - complex treatment of fan-in and fan-out nodes (geometry of interaction [Gonthier 92])
  - **inefficient** in practice (not elementary recursive [Mairson 96])
Sharing in the lambda-calculus

application

\[(\lambda x. yx)z\]
Sharing in the lambda-calculus

\[ G \]

\[ y \]

\[ x \]

\[ (\lambda x. yx)z \]

abstraction
Sharing in the lambda-calculus

fan rules

\((\lambda x. y x) z\)
Sharing in the lambda-calculus

bracket rules

(\lambda x. y x) z
Sharing in the lambda-calculus

croissant rules

$(\lambda x.yx)z$
Wadsworth’s algorithm

• sharing subterms [Wadsworth 72]:
  - arguments of beta-redexes are shared
  - easy to implement with dags (directed acyclic graphs)
Wadsworth’s algorithm

- Algorithm 1:
  - need duplication steps (abstractions on left of beta-redexes with reference counter greater than 1)
  - not optimal in total number of beta-reductions
Wadsworth’s algorithm

• Algorithm 2:
  - only duplicate **nodes on paths to the bound variable** of abstractions on left of beta-redexes
  - and share subterms not containing the bound variable
Wadsworth’s algorithm

• Algorithm 2 [Shivers–Wand 04]:
  - bottom-up traversal of abstraction $\lambda t.M$ to find nodes and paths to the bound variable $t$
Strong labeled lambda-calculus

- catch **history** of creations of redexes
- names (labels) of redexes are structured
- **confluent** calculus
Strong labeled lambda-calculus

- catch **history** of creations of redexes
- names (labels) of redexes are structured
- **confluent** calculus
Strong labeled lambda-calculus

- all redexes with same name are contracted in **single** step
- these **complete** normal order reductions are optimal
- theory of **redex families**
Weak labeled lambda-calculus

- under lambdas compute subterms with **no occurrence** of bound variable
- strong labeled theory + **tagging paths** to occurrences of the bound variable
- confluent calculus
Weak labeled lambda-calculus

- problem with K-terms.
- name of creating redex does not appear in name of created redex

\[ \lambda x \alpha y \]
\[ \lambda t M \]

name\(_1\) \(\alpha\)
name\(_2\) \(b\)
Weak labeled lambda-calculus\(^{(1)}\)

- \([\text{Klop 60}]\)
- tag lefts of application nodes when K-terms
- special tag since corresponding node is not duplicated

\[
\begin{array}{c}
\text{name}_1 \alpha \\
\text{name}_2 \langle \alpha, b \rangle
\end{array}
\]

\[
\begin{array}{c}
\langle \alpha, b \rangle \\
\langle \alpha, \alpha, a \rangle \\
\langle \alpha, d \rangle
\end{array}
\]
Weak labeled lambda-calculus (2)

[Barendregt 60]

- **add** the atomic label of applications to the names of redexes.
- and redo all theory of (weak) labelled calculus

\[
\lambda t \Rightarrow M \\
\lambda t \Rightarrow M
\]

\[
\alpha \xrightarrow{f} e \\
\alpha \xrightarrow{y} e
\]

\[
\alpha' = f \alpha \\
[\alpha', a]b
\]

\[
\lambda t \Rightarrow M \\
\lambda t \Rightarrow M
\]

\[
\alpha' \xrightarrow{\beta} e \\
\alpha' \xrightarrow{\beta} e
\]

\[
\lambda t \Rightarrow M \\
\lambda t \Rightarrow M
\]

\[
\alpha' \xrightarrow{\beta} e \\
\alpha' \xrightarrow{\beta} e
\]
Weak labeled lambda-calculus (3)

- [Geuvers 50 ?]
  - keep $[\alpha]$ because of $K$-redexes
  - $[\alpha]$ are useless

\[
\begin{align*}
\lambda t & \quad M \\
\alpha & \quad e \\
\lambda t & \quad y \\
\alpha' & = f \alpha \\
[\alpha', a] & b
\end{align*}
\]
THEOREMS
Weak lambda-calculus

• Terms

\[ M, N ::= x \mid MN \mid \lambda x. M \]

• Rules

\[(\beta) \quad R = (\lambda x. M)N \xrightarrow{R} M[x\backslash N] \]

\[(\nu) \quad \frac{M \xrightarrow{R} M'}{MN \xrightarrow{R} M'N} \]

\[(\xi') \quad \frac{M \xrightarrow{R} M' \quad x \not\in R}{\lambda x. M \xrightarrow{R} \lambda x. M'} \]

\[(\mu) \quad \frac{N \xrightarrow{R} N'}{MN \xrightarrow{R} MN'} \]

\[(\omega) \quad \frac{M \xrightarrow{R} N}{M \rightarrow N} \]
Weak labeled lambda-calculus

• Terms
  \[ U, V ::= \alpha : X \] labeled terms
  \[ X, Y ::= S | U \] clipped or labeled terms
  \[ S, T ::= x | UV | \lambda x.U \] clipped terms

• Labels
  \[ \alpha, \beta ::= a | [\alpha'] | [\alpha'] | [\alpha', \beta] \] labels
  \[ \alpha', \beta' ::= \alpha_1 \alpha_2 \cdots \alpha_n \ (n \geq 1) \] compound labels

• Rules
  \[ R = \beta:((\alpha' \cdot \lambda x.U)V) \xrightarrow{R} [\beta \alpha']:(\beta \alpha' \otimes U)[x \setminus [\beta \alpha'] : V] \]

\[ \alpha_1 \alpha_2 \cdots \alpha_n \cdot X = \alpha_1 : \alpha_2 : \cdots : \alpha_n : X \]
Weak labeled lambda-calculus

• Diffusion

\[ \alpha' \odot X = \begin{cases} X & \text{if } x \not\in X \\ x & \text{if } x \in X \end{cases} \]

\[ \alpha' \odot x = x \]

\[ \alpha' \odot UV = (\alpha' \odot U \alpha' \odot V) \text{ if } x \in UV \]

\[ \alpha' \odot \lambda y.U = \lambda y. \alpha' \odot U \text{ if } x \in \lambda y.U \]

\[ \alpha' \odot \beta : X = [\alpha', \beta] : \alpha' \odot X \text{ if } x \in X \]

• Substitution

\[ x[x \backslash W] = W \]

\[ y[x \backslash W] = y \]

\[ (UV)[x \backslash W] = U[x \backslash W] V[x \backslash W] \]

\[ (\lambda y.U)[x \backslash W] = \lambda y.U[x \backslash W] \]

\[ (\beta : X)[x \backslash W] = \beta : X[x \backslash W] \]
Weak labeled lambda-calculus

• Labels containment:

\[ \alpha' \prec [\alpha'] \]
\[ \alpha' \prec [\alpha'] \]
\[ \alpha' \prec [\alpha', \beta] \]
\[ \alpha' \prec \beta_i \Rightarrow \alpha' \prec \beta_1 \cdots \beta_n \]
\[ \alpha' \prec \beta' \prec \gamma' \Rightarrow \alpha' \prec \gamma' \]
Weak labeled lambda-calculus

• Maximality invariant

\[ Q(W) ::= \text{we have } \alpha' \not\subsetneq \beta \text{ for every redex } R \text{ with name } \alpha' \text{ and any subterm } \beta : X \text{ in } W. \]

• Lemma 1

If \( Q(W) \) and \( W \xrightarrow{\gamma'} W' \), then \( Q(W') \).
Weak labeled lambda-calculus

• Maximality invariant

\[ Q(W) ::= \text{we have } \alpha' \not\prec \beta \text{ for every redex } R \text{ with name } \alpha' \text{ and any subterm } \beta : X \text{ in } W. \]

• Lemma 1

If \( Q(W) \) and \( W \xrightarrow{\gamma'} W' \), then \( Q(W') \).

• Lexical scope invariant

\[ R(W) ::= \text{for any pair of subterms } \alpha : x \text{ and } \alpha \vdash y \text{ in } W, \text{ we have } x \text{ free in } W \text{ iff } y \text{ free in } W. \]

• Lemma 2

If \( R(W) \) and \( W \xrightarrow{} W' \), then \( R(W') \).
Weak labeled lambda-calculus

- Maximality invariant
  \[ \mathcal{P}(W) ::= \text{for any pair of subterms } \alpha : X \text{ and } \alpha : Y \text{ in } W, \]
  
  \[ \text{we have } X = Y. \]

- Sharing lemma
  If \( \mathcal{P}(W) \land Q(W) \land R(W) \) and \( W \xrightarrow{\gamma'} W' \), then \( \mathcal{P}(W') \).

- Sharing theorem
  \[ \text{Init}(U) ::= \text{every subterm of } U \text{ is labeled with a distinct letter.} \]
  
  Let \( \text{Init}(U) \) and \( U \xrightarrow{\Rightarrow} V \), then \( \mathcal{P}(V) \).
Conclusion

• weak lambda calculus implemented with dags
• useful for programming languages?
• do theory for weak labeled lambda calculus (3)
• and if explicit substitutions?
• do theory as particular case of term rewriting systems
• big difference between weak and strong calculus (POPL mark)
Conclusion

• weak lambda calculus implemented with dags
• useful for programming languages?
• do theory for weak labeled lambda calculus (3)
• and if explicit substitutions?
• do theory as particular case of term rewriting systems
• big difference between weak and strong calculus (POPL mark)

Rendez-vous in 2017...