The Suspension Calculus

Andrew Gacek

Department of Computer Science
University of Minnesota

November 2, 2006
Master's Thesis Defense

Outline

1. Using the Lambda Calculus for Representation
2. The Suspension Calculus
3. Other Explicit Substitution Calculi
4. Contributions and Future Work

Lambda Calculus as a Representational Device

Abstraction in the lambda calculus can capture binding in syntactic objects

Example

The formula

$$\forall x.(P(x) \vee Q)$$

can be encoded as

$$\mathit{forall}\ (\lambda x.\ (\mathit{or}\ (P\ x)\ Q))$$

Example

The expression

$$((\lambda x.\ x + 1)\ 2)$$

can be encoded as

$$\mathit{app}\ (\mathit{abs}\ \lambda x.\ (\mathit{plus}\ x\ (\mathit{const}\ 1)))\ (\mathit{const}\ 2)$$

Benefits of Such a Representation

Variable Renaming

$\forall y.(p(y) \vee q)$ is automatically equivalent to $\forall z.(p(z) \vee q)$

Quantifier Instantiation

$(\mathit{forall}\ P) \xrightarrow{t} (P\ t)$

Sophisticated Pattern Matching

We encode the pattern

$$\forall x.(P(x) \vee Q)$$

as

$$\mathit{forall}\ (\lambda x.\ (\mathit{or}\ (P\ x)\ Q))$$

which captures the notion that P can contain x but Q cannot

The De Bruijn Representation of Lambda Terms

Key Idea

Instead of using names to associate variable occurrences with their binders, we count the number of abstractions between a variable occurrence and its binder

Example

We represent the lambda term

$$(\lambda x.\ (\lambda y.\ x\ y)\ x)$$

by the de Bruijn term

$$(\lambda\,(\lambda\,\#2\ \#1)\ \#1)$$

In this representation α-convertible terms are identical

Explicit Substitutions and β-reduction

Key Idea

Laziness in substitution is important to implementations

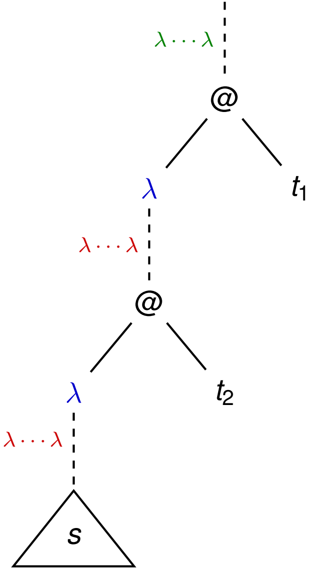
- Sometimes substitution can be avoided altogether, *e.g.*

$$(\lambda x.c\ t_1)\ t_2 \stackrel{?}{=} d\ t_3$$

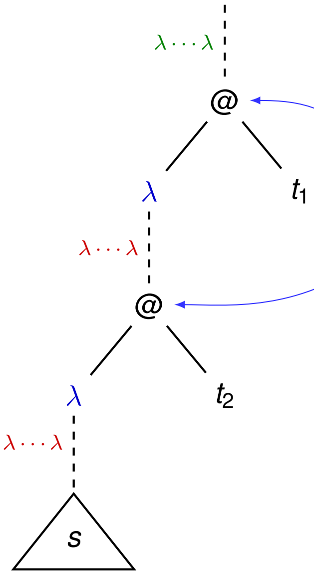
- Laziness is the basis for sharing substitution walks, *e.g.*

$$(\lambda x.\lambda y.t_1)\ t_2\ t_3$$

The General Scenario to be Treated

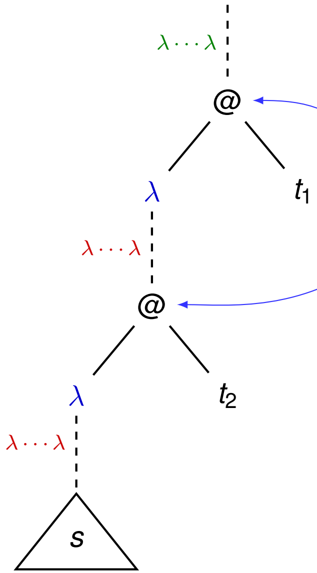


The General Scenario to be Treated



We want to contract these redexes but delay their effect on s

The General Scenario to be Treated



We want to contract these redexes but delay their effect on s

The result will have the form $[\![s, ol, nl, e]\!]$

Syntax of the Suspension Calculus

An explicit treatment of substitutions is gained by adding a new term of the form $[\![t, ol, nl, e]\!]$ where

- t is a term whose skeleton we substitute over
- ol is the old embedding level of t
- nl is the new embedding level of t
- e is an environment of substitutions of the form

$$(t_1, l_1) :: (t_2, l_2) :: \ldots :: (t_n, l_n) :: nil$$

where t_i is the substitution for the index $\#i$ and l_i is its embedding level

A Simple Rewriting Calculus

$$(\beta_s) \qquad ((\lambda\, t_1)\ t_2) \rightarrow [\![t_1, 1, 0, (t_2, 0) :: nil]\!]$$

A Simple Rewriting Calculus

(β_s) $((\lambda\, t_1)\ t_2) \to [\![t_1, 1, 0, (t_2, 0) :: nil]\!]$

(r1) $[\![(t_1\ t_2), ol, nl, e]\!] \to ([\![t_1, ol, nl, e]\!]\ [\![t_2, ol, nl, e]\!])$

(r2) $[\![(\lambda\, t), ol, nl, e]\!] \to (\lambda\, [\![t, ol', nl', (\#1, nl') :: e]\!])$,
where $ol' = ol + 1$ and $nl' = nl + 1$

(r3) $[\![\#1, ol, nl, (t, l) :: e]\!] \to [\![t, 0, nl', nil]\!]$, where $nl' = nl - l$

(r4) $[\![\#i, ol, nl, (t, l) :: e]\!] \to [\![\#i', ol', nl, e]\!]$,
where $i' = i - 1$ and $ol' = ol - 1$, provided $i > 1$

(r5) $[\![\#i, 0, nl, nil]\!] \to \#j$, where $j = i + nl$

(r6) $[\![c, ol, nl, e]\!] \to c$, provided c is a constant

A Simple Example

$$(\lambda\,(\lambda\,(\#1\ \#2)))\ t$$

$$\rhd_{\beta_s}\ [\![\lambda\,(\#1\ \#2), 1, 0, (t, 0) :: nil]\!]$$

$$\rhd_{r2}\ \lambda\,[\![(\#1\ \#2), 2, 1, (\#1, 1) :: (t, 0) :: nil]\!]$$

$$\rhd_{r1}\ \lambda\,([\![\#1, 2, 1, (\#1, 1) :: (t, 0) :: nil]\!]\ [\![\#2, 2, 1, (\#1, 1) :: (t, 0) :: nil]\!])$$

$$\rhd_{r3}\ \lambda\,([\![\#1, 0, 0, nil]\!]\ [\![\#2, 2, 1, (\#1, 1) :: (t, 0) :: nil]\!])$$

$$\rhd_{r5}\ \lambda\,(\#1\ [\![\#2, 2, 1, (\#1, 1) :: (t, 0) :: nil]\!])$$

$$\rhd_{r4}\ \lambda\,(\#1\ [\![\#1, 1, 1, (t, 0) :: nil]\!])$$

$$\rhd_{r3}\ \lambda\,(\#1\ [\![t, 0, 1, nil]\!])$$

Motivation for Merging

We have a mechanism for multiple non-trivial substitutions, but our system doesn't yet have them

Example

The term

$$(\lambda\,\lambda\,t_1)\ t_2\ t_3$$

reduces to

$$[\![[\![t_1, 2, 1, (\#1, 1) :: (t_2, 0) :: nil]\!], 1, 0, (t_3, 0) :: nil]\!]$$

but no rule applies to the outer substitution

In order to merge these two we need to generate the merging of two environments

Rules for Merging Environments

(m1) $[\![[\![t, ol_1, nl_1, e_1]\!], ol_2, nl_2, e_2]\!] \to [\![t, ol', nl', \{\!\{e_1, nl_1, ol_2, e_2\}\!\}]\!]$,
where $ol' = ol_1 + (ol_2 \dot{-} nl_1)$ and $nl' = nl_2 + (nl_1 \dot{-} ol_2)$

Rules for Merging Environments

(m1) $[\![[\![t, ol_1, nl_1, e_1]\!], ol_2, nl_2, e_2]\!] \rightarrow [\![t, ol', nl', \{\!\{ e_1, nl_1, ol_2, e_2 \}\!\}]\!]$,
where $ol' = ol_1 + (ol_2 \dot{-} nl_1)$ and $nl' = nl_2 + (nl_1 \dot{-} ol_2)$

(m2) $\{\!\{ e_1, nl_1, 0, nil \}\!\} \rightarrow e_1$

(m3) $\{\!\{ nil, 0, ol_2, e_2 \}\!\} \rightarrow e_2$

(m4) $\{\!\{ nil, nl_1, ol_2, (t, l) :: e_2 \}\!\} \rightarrow \{\!\{ nil, nl_1', ol_2', e_2 \}\!\}$,
where $nl_1' = nl_1 - 1$ and $ol_2' = ol_2 - 1$, provided $nl_1 \geq 1$

(m5) $\{\!\{ (t, n) :: e_1, nl_1, ol_2, (s, l) :: e_2 \}\!\} \rightarrow \{\!\{ (t, n) :: e_1, nl_1', ol_2', e_2 \}\!\}$,
where $nl_1' = nl_1 - 1$ and $ol_2' = ol_2 - 1$, provided $nl_1 > n$

(m6) $\{\!\{ (t, n) :: e_1, n, ol_2, (s, l) :: e_2 \}\!\} \rightarrow$
$([\![t, ol_2, l, (s, l) :: e_2]\!], m) :: \{\!\{ e_1, n, ol_2, (s, l) :: e_2 \}\!\}$,
where $m = l + (n \dot{-} ol_2)$

Properties of the Suspension Calculus

Theorem

The reading and merging rules define a terminating and confluent system

Proof structure:

- Termination used an extended recursive path ordering [Der82, FZ95] — I also verified this using Coq
- Confluence used weak confluence and termination

Theorem

The full system is confluent

Proof structure: Used the technique from [CHL96]

Graftable Meta Variables and Confluence

For X a graftable meta variable, $[\![X, ol, nl, e]\!]$ is irreducible which makes confluence an issue

Example

$((\lambda\,((\lambda\,X)\ t_1))\ t_2)$ can be rewritten to either of the following

$$[\![[\![X, 1, 0, (t_1, 0) :: nil]\!], 1, 0, (t_2, 0) :: nil]\!]$$

$$[\![[\![X, 2, 1, (\#1, 1) :: (t_2, 0) :: nil]\!], 1, 0, (t_1', 0) :: nil]\!]$$

where $t_1' = [\![t_1, 1, 0, (t_2, 0) :: nil]\!]$

The reading rules do not suffice to ensure a common reduct

Graftable Meta Variables and Confluence

For X a graftable meta variable, $[\![X, ol, nl, e]\!]$ is irreducible which makes confluence an issue

Example

$((\lambda\,((\lambda\,X)\ t_1))\ t_2)$ can be rewritten to either of the following

$$[\![[\![X, 1, 0, (t_1, 0) :: nil]\!], 1, 0, (t_2, 0) :: nil]\!]$$

$$[\![[\![X, 2, 1, (\#1, 1) :: (t_2, 0) :: nil]\!], 1, 0, (t_1', 0) :: nil]\!]$$

where $t_1' = [\![t_1, 1, 0, (t_2, 0) :: nil]\!]$

The reading rules do not suffice to ensure a common reduct

But the merging rules guarantee one exists

Comparison at Two Levels

1. Property based comparison

	Combination	Confluence	PSN
Suspension calculus	Yes	Yes	?
$\lambda\sigma$-calculus	Yes	Yes	No
λs-calculus	No	No	Yes
λs_e-calculus	No	Yes	No
λ_{ws}-calculus	No	Yes	Yes

2. Behavior based comparison
   - Describe information preserving translations
   - Translations provide insight into behavior

Contributions

- A modified version of the suspension calculus
  - Merging rules which are usable in practice
  - Structure for composition which retains logical properties
  - New proofs for termination and confluence
- A comparison of explicit substitution calculi
  - Translations between calculi
  - Proofs of formal properties of the translations

Future Work

- Preservation of strong normalization
- New methods of higher-order unification
- Compilation of strong reduction