Explicit Substitutions through the Eyes of the Suspension Calculus

Andrew Gacek
University of Minnesota

The Context of Interest

Abstractions can be used to capture binding structure as is present in formulas, proofs, programs, types, etc.

> *A representation for lambda terms is desired when these are used as data structures.*

Comparing lambda terms modulo lambda conversion rules is important in this context.

This issue has to be dealt with in metalanguage, logical framework and proof assistant implementations.

Need for Laziness

We can determine incompatibility of the terms

$$((\lambda x \lambda y \lambda z ((x\ z)\ t))\ (\lambda w\, w))$$

and

$$((\lambda x \lambda y \lambda z ((x\ y)\ s))\ (\lambda w\, w))$$

without calculating substitutions on t or s.

Desirable Properties for Rewriting Systems

Confluence

- But the λ-calculus is already confluent.
- Instead, we will look for confluence in the presence of variables representing open terms. So called graftable metavariables.

Termination

- But the λ-calculus already has nonterminating reductions.
- Instead, we will ask that the system does not introduce nonterminating computations in places where none previously existed.

Metavariable Confluence

Consider the term contracting the redex in the term

$$((\lambda x\ A)\ b)[y := t]$$

where A is a metavariable. We can contract the redex first,

$$A[x := b][y := t]$$

Or we distribute the substitution and then contract the redex,

$$A[y := t][x := b[y := t]]$$

Preservation of Strong Normalization

If all reduction paths for a term t terminate in the λ-calculus, then ideally an explicit substitution calculus *should not introduce a nonterminating reduction path for t*.

Consider if we tried to gain confluence by introducing a rule of the form

$$[x := b][y := t] \to [y := t][x := b[y := t]]$$

Then the calculus would not preserve strong normalization since we have

$$\begin{aligned} [x := b][y := t] &\to [y := t][x := b[y := t]] \\ &\to [x := b[y := t]][y := t[x := b[y := t]]] \\ &\to \ldots \end{aligned}$$

Combination of Traversals

In reducing the term

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

t_2 and t_3 should be substituted simultaneously into t_1.

This has shown to have very real practice benefit.

A Survey

	MC	PSN	CT
$\lambda\sigma$-calculus	Yes	No	Yes
Suspension calculus	Yes	?	Yes
$\lambda\upsilon$-calculus	No	Yes	No
λs-calculus	No	Yes	No
λs_e-calculus	Yes	No	No

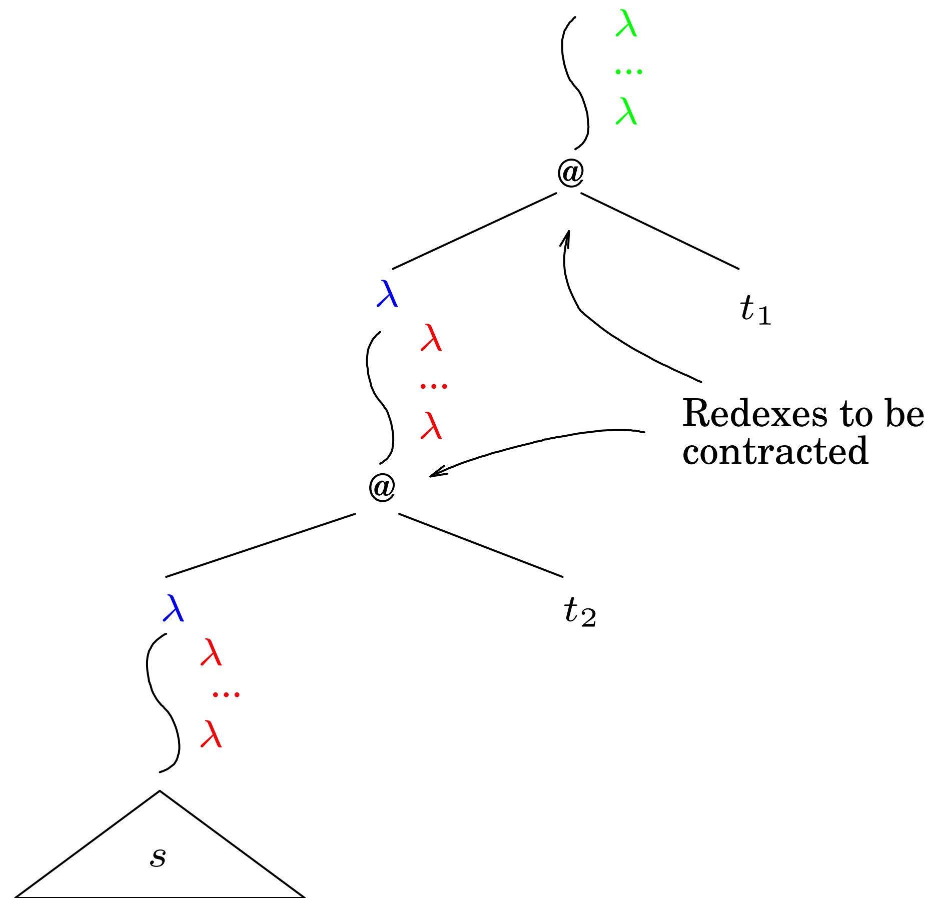
No system has been shown to have all three properties.

Moving to the Suspension Calculus

While names make λ-terms more readable for humans, de Bruijn indices are much more efficient for implementation purposes as it eliminates the need for α-conversion.

Example: We will write $\lambda x \lambda y\ x$ as $\lambda \lambda \#2$.

How Do We Make Substitutions Explicit?



We want to encode the effect on s of contracting shown redexes.

Encoding Substitutions

We encode the effect on s into a suspension $[\![s, ol, nl, e]\!]$, where

- ol is the number of abstraction encountered,
- nl is the number of abstractions that persist, and
- e is an environment list containing substitutions for the first ol de Bruijn indices of s.

The environment is a list of environment terms. Each environment term is a pair (t, n) where t is a term and n is an natural number called the index. The number n indicates the abstraction depth at which t originally occurred.

Examples Suspensions

Consider contracting a basic redex

$$(\lambda a)b \rightarrow [\![a, 1, 0, (b, 0) :: nil]\!]$$

Observe how this suspension should act based on different values of a,

$$[\![\#1, 1, 0, (b, 0) :: nil]\!] \rightarrow b$$

$$[\![\#2, 1, 0, (b, 0) :: nil]\!] \rightarrow \#1$$

$$[\![\#3, 1, 0, (b, 0) :: nil]\!] \rightarrow \#2$$

$$[\![\#4, 1, 0, (b, 0) :: nil]\!] \rightarrow \#3$$

Examples Suspensions (cont)

We can even consider if a is an application

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

Or an abstraction

$$[\![\lambda t, 1, 0, (b, 0) :: nil]\!] \rightarrow \lambda[\![t, 2, 1, (\#1, 1) :: (b, 0) :: nil]\!]$$

This last suspension is particularly interesting, and we should observe its effects on various values of t.

Examples Suspensions (cont)

To repeat,

$$[\![\lambda t, 1, 0, (b, 0) :: nil]\!] \rightarrow \lambda[\![t, 2, 1, (\#1, 1) :: (b, 0) :: nil]\!]$$

And for various values of t we have

$$[\![\#1, 2, 1, (\#1, 1) :: (b, 0) :: nil]\!] \rightarrow \#1$$
$$[\![\#2, 2, 1, (\#1, 1) :: (b, 0) :: nil]\!] \rightarrow b'$$
$$[\![\#3, 2, 1, (\#1, 1) :: (b, 0) :: nil]\!] \rightarrow \#2$$
$$[\![\#4, 2, 1, (\#1, 1) :: (b, 0) :: nil]\!] \rightarrow \#3$$

Where b' is b with all of its indices shifted up by one.

Examples Suspensions (cont)

We can we encode b', the shifting of b, in the suspension calculus as $[\![b, 0, 1, nil]\!]$, which has the follow effects

$$[\![\#1, 0, 1, nil]\!] \rightarrow \#2$$

$$[\![\#2, 0, 1, nil]\!] \rightarrow \#3$$

$$[\![\#3, 0, 1, nil]\!] \rightarrow \#4$$

$$[\![\#4, 0, 1, nil]\!] \rightarrow \#5$$

The Rewriting Calculus

Beta Contraction

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

The Reading Rules

(r1) $[\![c, ol, nl, e]\!] \rightarrow c$ (c is a constant)

(r2) $[\![\#1, ol, nl, (t, l) :: e]\!] \rightarrow [\![t, 0, nl - l, nil]\!]$

(r3) $[\![\#i, 0, nl, e]\!] \rightarrow \#(i + nl)$

(r4) $[\![\#i, ol, nl, et :: e]\!] \rightarrow [\![\#(i-1), ol - 1, nl, e]\!]$, if $i > 1$

(r5) $[\![(t_1\ t_2), ol, nl, e]\!] \rightarrow [\![t_1, ol, nl, e]\!]\ [\![t_2, ol, nl, e]\!]$

(r6) $[\![\lambda\, t, ol, nl, e]\!] \rightarrow \lambda[\![t, ol + 1, nl + 1, (\#1, nl + 1) :: e]\!]$

Properties of the Simple Calculus

The calculus has several pleasing properties. In particular, let

- $\rhd_r$ denote the compatible extension of the reading rules, and
- $\rhd_{\beta_s}$ denote the compatible extension of all the rules.

Then, we have:

Prop 1. $\rhd_r$ is terminating and confluent.

Prop 2. $\rhd_{\beta_s}^*$ is capable of simulating beta contraction in the de Bruijn notation.

Prop 3. $\rhd_{\beta_s}$ is confluent.

Non-properties of the Simple Calculus

- The simple calculus does not allow combination of traversals.
- The simple calculus does not have metavariable confluence.

We will solve both of these problems by adding some way of composing substitutions.

Composing Substitutions

We desire a rule of the form

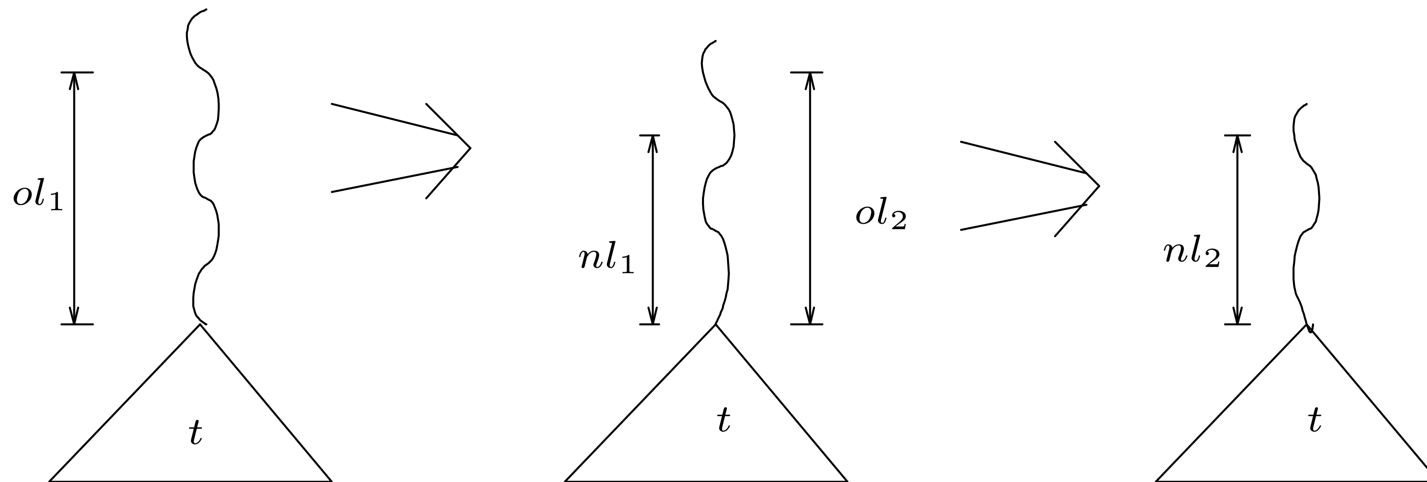
$$[\![[\![t_1, ol_1, nl_1, e_1]\!], ol_2, nl_2, e_2]\!] \rightarrow [\![t, ol', nl', e']\!]$$

Which leaves us to ask,

- What are ol' and nl'?
- What is e'?

Finding ol' and nl' (case 1)

Suppose that ol_2 is larger than or equal to nl_1. Then we have,

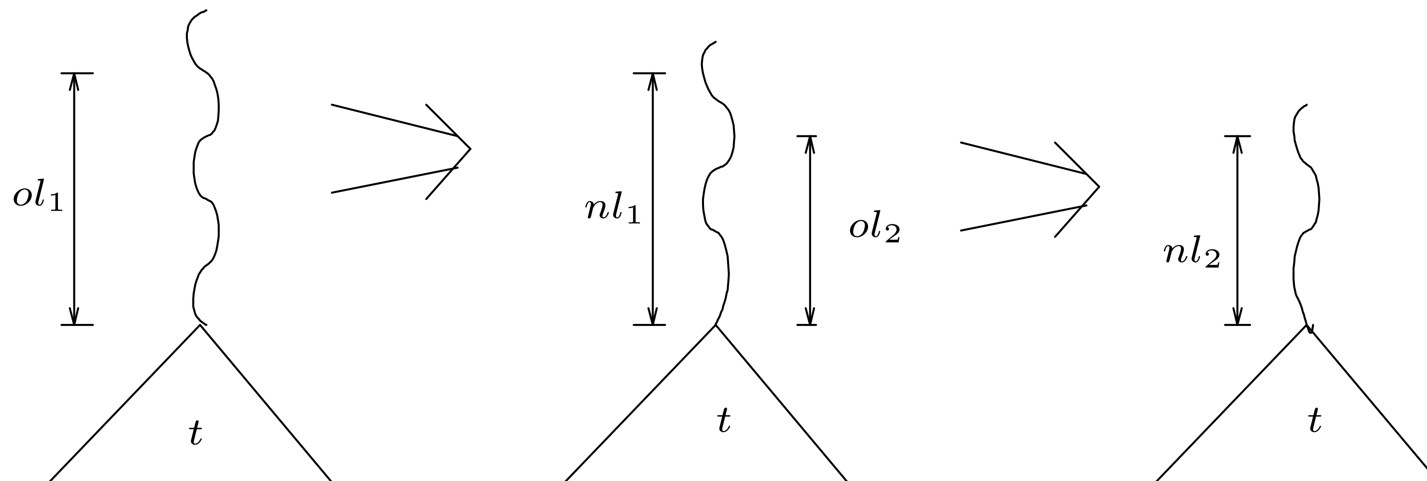


In this case,

- $ol' = ol_1 + (ol_2 - nl_1)$ and $nl' = nl_2$.
- Environment will be e_1 modified by e_2 plus an initial segment of e_2

Finding ol' and nl' (case 2)

On the other hand, suppose that ol_2 is smaller than nl_1. Then,



Now,

- $ol' = ol_1$ and $nl' = nl_2 + (nl_1 - ol_2)$.
- Environment will be e_1, with a final segment of it affected by e_2.

Finding e'

Rather than describing how to compute e', we will introduce a new syntactic form representing a merged environment. This will allow us to delay the computation of the merged environment. The new syntax is

$$\begin{array}{lcl} t & ::= & c \mid x \mid \#i \mid (t\ t) \mid (\lambda\, t) \mid [\![t, n, n, e]\!] \\ e & ::= & nil \mid et :: e \mid \{\!\{ e, n, n, e \}\!\} \end{array}$$

The new environment form $\{\!\{ e_1, nl_1, ol_2, e_2 \}\!\}$ represents the result of composing e_2 with e_1.

Example Combination

Consider

$$((\lambda\lambda a)\ b)\ c$$

which we saw earlier evaluates to

$$(\lambda[\![a, 2, 1, (\#1, 1) :: (b, 0) :: nil]\!])\ c$$

If we contract the redex we get

$$[\![[\![a, 2, 1, (\#1, 1) :: (b, 0) :: nil]\!], 1, 0, (c, 0) :: nil]\!]$$

which merges to

$$[\![a, 2, 0, \{\!\{(\#1, 1) :: (b, 0) :: nil, 1, 1, (c, 0) :: nil\}\!\}]\!]$$

Example Combination (cont)

To repeat, we have

$$((\lambda\lambda a)\ b)\ c$$

which evaluates to

$$[\![a, 2, 0, \{\!\{(\#1, 1) :: (b, 0) :: nil, 1, 1, (c, 0) :: nil\}\!\}]\!]$$

Using the combination rules we should be able to compute this merged environment out to get a normal environment,

$$[\![a, 2, 0, (c, 0) :: (b, 0) :: nil]\!]$$

Substitution Combination Rules

(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) $\{\!\{ nil, nl_1, ol_2, et :: e_2 \}\!\} \rightarrow \{\!\{ nil, nl_1 - 1, ol_2, e_2 \}\!\}$

(m5) $\{\!\{ (t, n) :: e_1, nl_1, ol_2, et :: e_2 \}\!\} \rightarrow$
$\{\!\{ (t, n) :: e_1, nl_1 - 1, ol_2 - 1, e_2 \}\!\}$,
provided $nl_1 > n$ and $ol_2 > 1$

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

Properties of the Enhanced Calculus

Let

- $\triangleright_{rm}$ denote the compatible extension of reading and composition rules, and
- $\triangleright_{\beta_s}$ denote the compatible extension of the entire ensemble.

Then, we have:

Prop 1. $\triangleright_{rm}$ is terminating.

Prop 2. $\triangleright_{rm}$ is locally confluent.

Prop 3. $\triangleright^*_{\beta_s}$ is capable of simulating beta contraction on de Bruijn terms and is also confluent on metavariables.

Another Calculus: $\lambda\sigma$-calculus

The $\lambda\sigma$-calculus is the only other system to support combination of traversals. Instead of maintaining a list of substitutions, however, the $\lambda\sigma$-calculus treats substitutions more as functions.

To encode the effect of various substitutions s on a term t, we write $t[s]$.

A Taste of the $\lambda\sigma$-calculus

Let's start with the $\uparrow$ substitution, which shifts all de Bruijn indices up by one. For example,

$$\#1[\uparrow] \rightarrow \#2$$
$$\#2[\uparrow] \rightarrow \#3$$
$$\#3[\uparrow] \rightarrow \#4$$
$$\#4[\uparrow] \rightarrow \#5$$

Composition in $\lambda\sigma$-calculus

The $\lambda\sigma$-calculus allows you to compose substitutions using $\circ$, just like you would compose functions. Consider the effect of the substitution $\uparrow \circ \uparrow$,

$$\#1[\uparrow \circ \uparrow] \rightarrow \#3$$
$$\#2[\uparrow \circ \uparrow] \rightarrow \#4$$
$$\#3[\uparrow \circ \uparrow] \rightarrow \#5$$
$$\#4[\uparrow \circ \uparrow] \rightarrow \#6$$

De Bruijn Indices in the $\lambda\sigma$-calculus

Let us abbreviate $\overbrace{\uparrow \circ \uparrow \circ \cdots \circ \uparrow}^{n}$ as $\uparrow^n$. Then we can encode all the de Bruijn indices using only $\#1$ and $\uparrow^n$.

$$\#1[\uparrow^0] \to \#1$$

$$\#1[\uparrow^1] \to \#2$$

$$\#1[\uparrow^2] \to \#3$$

$$\#1[\uparrow^3] \to \#4$$

In fact, this is exactly what the $\lambda\sigma$-calculus does. The term $\#n$ is just an abbreviation for $\#1[\uparrow^{n-1}]$.

The Cons Operator in the $\lambda\sigma$-calculus

The only other operator in the $\lambda\sigma$-calculus is the cons operator represented by $\cdot$ and allows you to cons a term a onto a substitution s as $a \cdot s$.

There are two important rules for evaluating this operator,

$$\#1[a \cdot s] \rightarrow a$$

$$\uparrow \circ (a \cdot s) \rightarrow s$$

Lastly, we need to know about the identity substitution id.

A Beta Contraction in the $\lambda\sigma$-calculus

The Beta rule in the $\lambda\sigma$-calculus say,

$$(\lambda a)\ b \rightarrow a[b \cdot id]$$

Observe how this substitution should act based on different values of a,

$$\#1[b \cdot id] \rightarrow b$$

$$\#2[b \cdot id] \rightarrow \#1$$

$$\#3[b \cdot id] \rightarrow \#2$$

$$\#4[b \cdot id] \rightarrow \#3$$

More on Composition in the $\lambda\sigma$-calculus

To see exactly how that works, we need to know the rule

$$a[s][t] \to a[s \circ t]$$

Then by expanding the abbreviation for $\#n$ to $\#1[\uparrow^{n-1}]$, we see for example,

$$\begin{aligned} \#4[b \cdot id] &= \#1[\uparrow^3][b \cdot id] \\ &\to \#1[\uparrow^3 \circ (b \cdot id)] \\ &\to \#1[\uparrow^2 \circ id] \\ &\to \#1[\uparrow^2] \\ &= \#3 \end{aligned}$$

Associativity in the $\lambda\sigma$-calculus

We saw that

$$\uparrow \circ (a \cdot s) \rightarrow s$$

but what about

$$(\uparrow \circ \uparrow) \circ (a \cdot s)$$

We cannot evaluate this term without knowing another rule,

$$(s \circ t) \circ u \rightarrow s \circ (t \circ u)$$

which allows us to do

$$(\uparrow \circ \uparrow) \circ (a \cdot s) \rightarrow \uparrow \circ (\uparrow \circ (a \cdot s)) \rightarrow \uparrow \circ s$$

Moving Underneath Abstractions in $\lambda\sigma$

The rule for moving a substitution underneath an abstraction is,

$$(\lambda a)[s] \rightarrow \lambda a[\#1 \cdot (s \circ \uparrow)]$$

Notice that we modify the substitution with $\uparrow$ as we push it down.

Primary Differences

- The way renumbering for environment terms is handled
  - In $\lambda\sigma$ we apply $\uparrow$ each time we go down
  - In the suspension calculus we increment nl, and use the difference between nl and the index to compute a shift amount when we extract a term from the environment
- Interactions between combined environments
  - In $\lambda\sigma$ we have $(s \circ t) \circ u \rightarrow s \circ (t \circ u)$
  - In the suspension calculus, $\{\!\{e_1, nl_1, ol_2, e_2\}\!\}$ cannot interact with other merged environments

Translation to the $\lambda\sigma$-calculus

Using the new merging rules for the suspension calculus, we have been able to define a mapping to the $\lambda\sigma$-calculus.

For example,

$$T(\llbracket a, 1, 0, (b, 0) :: nil \rrbracket) = T(a)[T(b) \cdot id]$$
$$T(\llbracket a, 2, 1, (\#1, 1) :: (b, 0) :: nil \rrbracket) = T(a)[\#1 \cdot ((T(b) \cdot id) \circ \uparrow)]$$

$$T(\llbracket a, 2, 0, \{\!\{(\#1, 1) :: (b, 0) :: nil, 1, 1, (c, 0) :: nil\}\!\} \rrbracket) = \\ T(a)[(\#1 \cdot ((T(b) \cdot id) \circ \uparrow) \circ (T(c) \cdot id)]$$
$$T(\llbracket a, 2, 0, (c, 0) :: (b, 0) :: nil \rrbracket) = T(a)[T(c) \cdot T(b) \cdot id]$$

Moreover, the translation preserves the rewrite relationship.

Lack of PSN in the $\lambda\sigma$-calculus

Mellies showed that the $\lambda\sigma$-calculus does not preserve strong normalization, by demonstrating a strongly normalizing term with an infinite reduction path in the $\lambda\sigma$-calculus. He did this by noting that in the expression

$$\uparrow \circ (a \cdot s),$$

the term a is vacuous, but the lax way the $\lambda\sigma$-calculus handles merged substitutions, still allows a to interact with other substitutions:

$$(\uparrow \circ (a \cdot s)) \circ u \rightarrow \uparrow \circ ((a \cdot s) \circ u)$$

Hope for PSN in the Suspension Calculus

One of the primary differences listed before was

- Interactions between combined environments
  - In $\lambda\sigma$ we have $(s \circ t) \circ u \rightarrow s \circ (t \circ u)$
  - In the suspension calculus, $\{\!\{e_1, nl_1, ol_2, e_2\}\!\}$ cannot interact with other merged environments

Because of this, because of the careful way that the substitution calculus handles merged substitutions, we believe that we will be able to prove preservation of strong normalization.

Summary

- Simplified merging rules for the suspension calculus
- Rewrite preserving translation from the suspension calculus to the $\lambda\sigma$-calculus
- The translation and differences between the systems give us hope for PSN in the suspension calculus

Questions?

Appendix

Beta and Reading Rles

Beta Contraction

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

The Reading Rules

(r1) $[\![c, ol, nl, e]\!] \to c \quad (c$ **is a constant)**

(r2) $[\![\#1, ol, nl, (t, l) :: e]\!] \to [\![t, 0, nl - l, nil]\!]$

(r3) $[\![\#i, 0, nl, e]\!] \to \#(i + nl)$

(r4) $[\![\#i, ol, nl, et :: e]\!] \to [\![\#(i-1), ol - 1, nl, e]\!]$**, if** $i > 1$

(r5) $[\![(t_1\ t_2), ol, nl, e]\!] \to [\![t_1, ol, nl, e]\!]\ [\![t_2, ol, nl, e]\!]$

(r6) $[\![\lambda\, t, ol, nl, e]\!] \to \lambda[\![t, ol + 1, nl + 1, (\#1, nl + 1) :: e]\!]$

Merging Rules

(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) $\{\!\{ nil, nl_1, ol_2, et :: e_2 \}\!\} \rightarrow \{\!\{ nil, nl_1 - 1, ol_2, e_2 \}\!\}$

(m5) $\{\!\{ (t, n) :: e_1, nl_1, ol_2, et :: e_2 \}\!\} \rightarrow$
$\{\!\{ (t, n) :: e_1, nl_1 - 1, ol_2 - 1, e_2 \}\!\}$,
provided $nl_1 > n$ and $ol_2 > 1$

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

$\lambda\sigma$-calculus Rules

$$(\lambda a)\ b \to a[b \cdot id]$$

$$(a\ b)[s] \to a[s]\ b[s]$$

$$(\lambda a)[s] \to \lambda a[\#1 \cdot (s \circ \uparrow)]$$

$$a[s][t] \to a[s \circ t]$$

$$(a \cdot s) \circ t \to a[t] \cdot (s \circ t)$$

$$(s \circ t) \circ u \to s \circ (t \circ u)$$

$$\#1[id] \to \#1$$

$$\#1[a \cdot s] \to a$$

$$id \circ s \to s$$

$$\uparrow \circ\, id \to \uparrow$$

$$\uparrow \circ (a \cdot s) \to s$$

Translation

Terms

$$T(\#i) = \#1[\uparrow^{i-1}]$$

$$T(\lambda a) = \lambda T(a)$$

$$T(a\ b) = T(a)\ T(b)$$

$$T([\![t, ol, nl, e]\!]) = T(t)[E(e, nl)]$$

Environments

$$E(nil, j) = \uparrow^{j}$$

$$E((t, n) :: e, j) = (T(t) \cdot E(e, n)) \circ \uparrow^{j-n}$$

$$E(\{\!\{e_1, nl_1, ol_2, e_2\}\!\}, j) = E(e_1, nl_1) \circ E(e_2, j - (nl_1 \dot{-} ol_2))$$