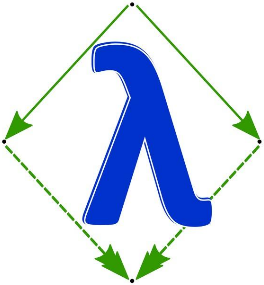


A relational λ-calculus

Pablo Barenbaum[1,2]
[1] Universidad de Buenos Aires
[2] Universidad Nacional de Quilmes

Ongoing work with Federico Lochbaum and Mariana Milicich.

2020-06-04

Table of Contents

Motivation

Logic programming

```
father(a, b).
father(b, c).

grandfather(A, B) :-
 father(A, C),
 father(C, B).

grandson(A, B) :-
 grandfather(B, A).
```

Motivation

Logic programming

```
father(a, b).
father(b, c).

grandfather(A, B) :-
 father(A, C),
 father(C, B).

grandson(A, B) :-
 grandfather(B, A).
```

Functional logic programming

(Curry, Mozart/OZ, ...)

```
father A = B
father B = C

grandfather = father . father

grandson = inverse grandfather

inverse :  (a -> b) -> b -> a
```

inverse f b = ν a. ((f a $\doteq$ b) ; a)

Motivation

Logic programming

```
father(a, b).
father(b, c).

grandfather(A, B) :-
 father(A, C),
 father(C, B).

grandson(A, B) :-
 grandfather(B, A).
```

Functional logic programming

(Curry, Mozart/OZ, ...)

```
father A = B
father B = C

grandfather = father . father

grandson = inverse grandfather

inverse :  (a -> b) -> b -> a
inverse f b = ν a. ((f a ≐ b) ; a)
```

Inversible programs — e.g. parser $\leftrightarrow$ pretty printer.

Motivation

λ-calculus

$$t \;::=\; x \;\mid\; \lambda x.\, t \;\mid\; t\; t$$

miniKanren

$$G \;::=\; T \doteq T \;\mid\; R(T_1, \ldots, T_n) \;\mid\; G \;;\; G \;\mid\; G \boxplus G \;\mid\; \nu x.\, G$$

(T, $T_1, \ldots$ are terms of a first-order language)

Motivation

λ-calculus

$$
\begin{array}{rcl}
t & ::= & x \\
  & \mid & \lambda x.\, t \\
  & \mid & t\; t
\end{array}
$$

miniKanren

$$
\begin{array}{rcl}
G & ::= & T \doteq T \\
  & \mid & R(T_1, \ldots, T_n) \\
  & \mid & G \,;\, G \\
  & \mid & G \boxplus G \\
  & \mid & \nu x.\, G
\end{array}
$$

(T, $T_1, \ldots$ are terms of a first-order language)

Related work

- Hanus et al. (2005)
  Operational Semantics for Declarative Multi-Paradigm Languages
- Rozplokhas, Vyatkin, Boulytchev (2019)
  Certified Semantics for miniKanren
- Lambda-calculi with stochastic/erratic choice.

Table of Contents

Our first approach

$$
\begin{array}{rcll}
t & ::= & x & \text{variable} \\
  & \mid & \lambda x.\, t & \text{abstraction} \\
  & \mid & t\ t & \text{application} \\
  & \mid & \mathbf{c} & \text{constructor} \\
  & \mid & \textsc{fail} & \text{explicit failure} \\
  & \mid & t \stackrel{\bullet}{=} t & \text{unification} \\
  & \mid & t\ ;\ t & \text{sequence} \\
  & \mid & t \boxplus t & \text{non-deterministic alternative} \\
  & \mid & \nu x.\, t & \text{fresh variable}
\end{array}
$$

Our first approach

t	$::=$	x	variable
	$\mid$	$\lambda x.\, t$	abstraction
	$\mid$	$t\; t$	application
	$\mid$	$\mathbf{c}$	constructor
	$\mid$	FAIL	explicit failure
	$\mid$	$t \overset{\bullet}{=} t$	unification
	$\mid$	$t \,;\, t$	sequence
	$\mid$	$t \boxplus t$	non-deterministic alternative
	$\mid$	$\nu x.\, t$	fresh variable

For example, if $M \overset{\mathrm{def}}{=} \lambda x.\, \nu y.\, \Big(((x \overset{\bullet}{=} \mathbf{c}\, y) \,;\, y) \boxplus ((x \overset{\bullet}{=} \mathbf{d}) \,;\, x)\Big)$:

$$
\begin{array}{rcl}
M\,(\mathbf{c}\,\mathbf{e}) & \longrightarrow & \nu y.\, \Big(((\mathbf{c}\,\mathbf{e} \overset{\bullet}{=} \mathbf{c} y) \,;\, y) \boxplus ((\mathbf{c}\,\mathbf{e} \overset{\bullet}{=} \mathbf{d}) \,;\, x)\Big) \\
& \longrightarrow & ((\mathbf{c}\,\mathbf{e} \overset{\bullet}{=} \mathbf{c} y) \,;\, y) \boxplus ((\mathbf{c}\,\mathbf{e} \overset{\bullet}{=} \mathbf{d}) \,;\, x) \\
& \longrightarrow & (\mathbf{ok} \,;\, \mathbf{e}) \boxplus ((\mathbf{c}\,\mathbf{e} \overset{\bullet}{=} \mathbf{d}) \,;\, x) \\
& \longrightarrow & \mathbf{e} \boxplus ((\mathbf{c}\,\mathbf{e} \overset{\bullet}{=} \mathbf{d}) \,;\, x) \\
& \longrightarrow & \mathbf{e} \boxplus \mathrm{FAIL} \\
& \longrightarrow & \mathbf{e}
\end{array}
$$

Technical challenges

Fresh variables should be local to "threads" delimited by $\boxplus$

$$
\begin{array}{ccc}
\nu x.\ \Big(((x \overset{\bullet}{=} \mathbf{c})\ ;\ x) \boxplus ((x \overset{\bullet}{=} \mathbf{d})\ ;\ x)\Big) & & \\
\downarrow & & \\
((x \overset{\bullet}{=} \mathbf{c})\ ;\ x) \boxplus ((x \overset{\bullet}{=} \mathbf{d})\ ;\ x) & \longrightarrow & (\mathbf{ok}\ ;\ \mathbf{c}) \boxplus ((x \overset{\bullet}{=} \mathbf{d})\ ;\ x) \\
\not\downarrow & & \twoheaddownarrow \\
(\mathbf{ok}\ ;\ \mathbf{c}) \boxplus ((\mathbf{c} \overset{\bullet}{=} \mathbf{d})\ ;\ \mathbf{c}) & & \mathbf{c} \boxplus \mathbf{d}
\end{array}
$$

Technical challenges

Commutative conversions are needed to unblock redexes

$$(t \boxplus \lambda x.\, s)\, u \longrightarrow (t\, u) \boxplus ((\lambda x.\, s)\, u)$$

Technical challenges

Commutative conversions are needed to unblock redexes

$$(t \boxplus \lambda x.\, s)\, u \longrightarrow (t\, u) \boxplus ((\lambda x.\, s)\, u)$$

...so we must work up to associativity and commutativity of $\boxplus$

$$(t_1 \boxplus t_2)\,(s_1 \boxplus s_2)$$

$$\swarrow \qquad \searrow$$

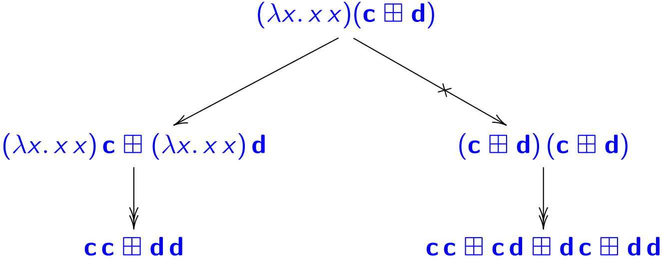
$$t_1(s_1 \boxplus s_2) \boxplus t_2(s_1 \boxplus s_2) \qquad\qquad (t_1 \boxplus t_2)\, s_1 \boxplus (t_1 \boxplus t_2)\, s_2$$

$$\downarrow \qquad\qquad \downarrow$$

$$(t_1\, s_1 \boxplus t_1\, s_2) \boxplus (t_2\, s_1 \boxplus t_2\, s_2) \;\cdots\cdots\; \equiv \;\cdots\cdots\; (t_1\, s_1 \boxplus t_2\, s_1) \boxplus (t_1\, s_2 \boxplus t_2\, s_2)$$

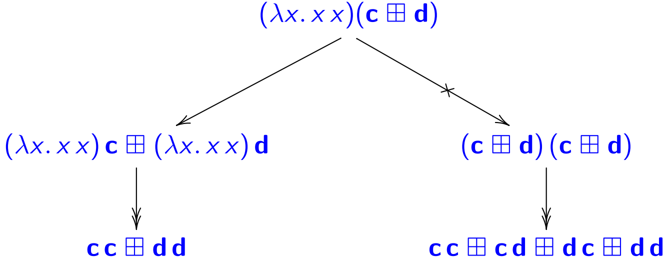
Technical challenges

Non-deterministic choice is an effect (not a value)



Technical challenges

Non-deterministic choice is an effect (not a value)



It does not commute with abstraction

$$(\lambda x.\, t) \boxplus (\lambda x.\, s) \not\equiv \lambda x.\, (t \boxplus s)$$

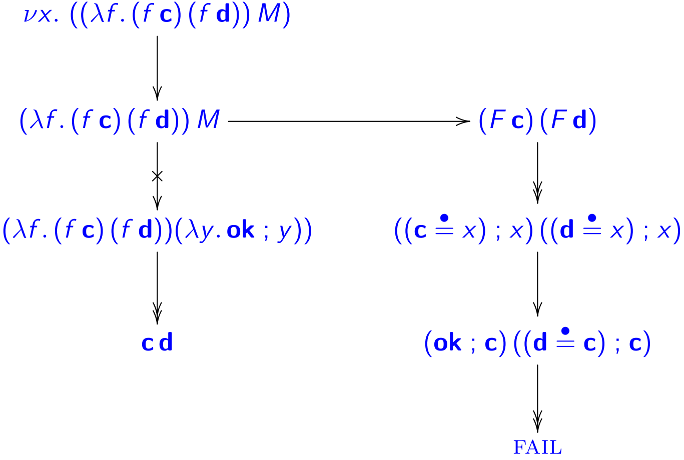
$$
\begin{array}{lcl}
(\lambda f.\, (f\, \mathbf{ok})\, (f\, \mathbf{ok}))((\lambda x.\, \mathbf{c}) \boxplus (\lambda x.\, \mathbf{d})) & \twoheadrightarrow & \mathbf{c}\,\mathbf{c} \boxplus \mathbf{d}\,\mathbf{d} \\
(\lambda f.\, (f\, \mathbf{ok})\, (f\, \mathbf{ok}))(\lambda x.\, (\mathbf{c} \boxplus \mathbf{d})) & \twoheadrightarrow & \mathbf{c}\,\mathbf{c} \boxplus \mathbf{c}\,\mathbf{d} \boxplus \mathbf{d}\,\mathbf{c} \boxplus \mathbf{d}\,\mathbf{d}
\end{array}
$$

Technical challenges

Unification should only be performed under weak contexts

Let $F \stackrel{\text{def}}{=} \lambda y.\,((y \stackrel{\bullet}{=} x)\ ;\ x)$.

If we allow reduction under lambdas, $F \rightarrow \lambda y.\,(\mathbf{ok}\ ;\ y)$.



Technical challenges

We cannot solve higher-order unification

$$\nu f.\,((f\,\mathbf{c} \stackrel{\bullet}{=} \mathbf{c})\ ;\ f) \longrightarrow\!\!\!\!\!\!\to\ ?$$

- There are no most general unifiers.
- Higher-order unification is undecidable.

Technical challenges

We cannot solve higher-order unification

$$\nu f.\,((f\,\mathbf{c} \doteq \mathbf{c}) \;;\; f) \longrightarrow\!\!\!\!\rightarrow\; ?$$

- There are no most general unifiers.
- Higher-order unification is undecidable.

...but we do want to pattern match against functions

$$(\mathbf{c}x \doteq \mathbf{c}(\lambda y.\, y)) \;;\; (x \doteq x) \longrightarrow\!\!\!\!\rightarrow (\lambda y.\, y) \doteq (\lambda y.\, y) \longrightarrow\!\!\!\!\rightarrow \mathbf{ok}$$

Technical challenges

Comparing functions by syntactic equality breaks confluence

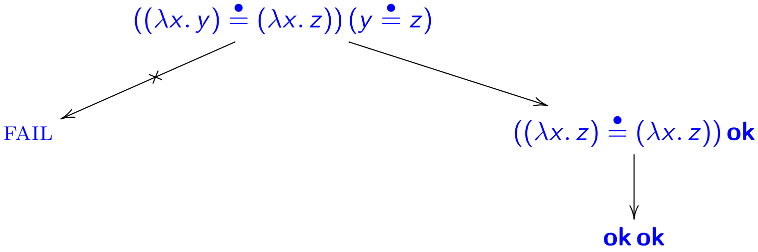


Table of Contents

The $\lambda^{\mathtt{U}}$-calculus

Terms	t	$::=$	x	variable
		$\mid$	$\lambda x.\, P$	abstraction
		$\mid$	$\lambda^{\ell} x.\, P$	allocated abstraction
		$\mid$	$t\ t$	application
		$\mid$	$\mathbf{c}$	constructor
		$\mid$	$t \overset{\bullet}{=} t$	unification
		$\mid$	$t\ ;\ t$	sequence
		$\mid$	$\nu x.\, t$	fresh variable
Programs	P	$::=$	$\mathtt{fail}$	empty program
		$\mid$	$t \oplus P$	non-deterministic alternative

Programs are of the form $P = t_1 \oplus \ldots \oplus t_n$.

$$\text{FAIL} \overset{\text{def}}{=} (\lambda x.\, \mathtt{fail})\, \mathbf{ok} \qquad (t \boxplus s) \overset{\text{def}}{=} (\lambda x.\, t \oplus s)\, \mathbf{ok}$$

Invariant

Two abstractions with the same location are equal.

The $\lambda^{\mathtt{U}}$-calculus

$$
\begin{array}{llcl}
\text{Values} & \mathtt{v} & ::= & x \\
 & & \mid & \lambda^{\ell} x.\, P \\
 & & \mid & \mathbf{c}\, \mathtt{v}_1 \ldots \mathtt{v}_n \\
\text{Weak contexts} & \mathsf{W} & ::= & \Box \\
 & & \mid & \mathsf{W}\, t \\
 & & \mid & t\, \mathsf{W} \\
 & & \mid & \mathsf{W} \overset{\bullet}{=} t \\
 & & \mid & t \overset{\bullet}{=} \mathsf{W} \\
 & & \mid & \mathsf{W} \,;\, t \\
 & & \mid & t \,;\, \mathsf{W}
\end{array}
$$

Usual operation to plug a term into a context:

$$\mathsf{W}\langle t \rangle$$

Plus an operation to plug a program into a context:

$$\mathsf{W}\langle t_1 \oplus \ldots \oplus t_n \rangle \stackrel{\text{def}}{=} \mathsf{W}\langle t_1 \rangle \oplus \ldots \oplus \mathsf{W}\langle t_n \rangle$$

Reduction rules

Rules operate on the toplevel program.

$$P_1 \oplus \mathsf{W}\langle \lambda x.\, Q \rangle \oplus P_2 \quad \xrightarrow{\texttt{alloc}} \quad P_1 \oplus \mathsf{W}\langle \lambda^{\ell} x.\, Q \rangle \oplus P_2 \qquad \ell \text{ fresh}$$

Reduction rules

Rules operate on the toplevel program.

$$P_1 \oplus \mathsf{W}\langle \lambda x.\, Q \rangle \oplus P_2 \xrightarrow{\texttt{alloc}} P_1 \oplus \mathsf{W}\langle \lambda^{\ell} x.\, Q \rangle \oplus P_2 \qquad \ell \text{ fresh}$$

$$P_1 \oplus \mathsf{W}\langle (\lambda^{\ell} x.\, Q)\, \mathtt{v} \rangle \oplus P_2 \xrightarrow{\texttt{beta}} P_1 \oplus \mathsf{W}\langle Q\{x := \mathtt{v}\} \rangle \oplus P_2$$

Reduction rules

Rules operate on the toplevel program.

$$P_1 \oplus \mathsf{W}\langle \lambda x.\, Q \rangle \oplus P_2 \xrightarrow{\texttt{alloc}} P_1 \oplus \mathsf{W}\langle \lambda^{\ell} x.\, Q \rangle \oplus P_2 \qquad \ell \text{ fresh}$$

$$P_1 \oplus \mathsf{W}\langle (\lambda^{\ell} x.\, Q)\, \mathtt{v} \rangle \oplus P_2 \xrightarrow{\texttt{beta}} P_1 \oplus \mathsf{W}\langle Q\{x := \mathtt{v}\} \rangle \oplus P_2$$

$$P_1 \oplus \mathsf{W}\langle \mathtt{v} \,;\, t \rangle \oplus P_2 \xrightarrow{\texttt{seq}} P_1 \oplus \mathsf{W}\langle t \rangle \oplus P_2$$

Reduction rules

Rules operate on the toplevel program.

$$P_1 \oplus \mathsf{W}\langle \lambda x.\, Q \rangle \oplus P_2 \xrightarrow{\texttt{alloc}} P_1 \oplus \mathsf{W}\langle \lambda^\ell x.\, Q \rangle \oplus P_2 \qquad \ell \text{ fresh}$$

$$P_1 \oplus \mathsf{W}\langle (\lambda^\ell x.\, Q)\, \mathtt{v} \rangle \oplus P_2 \xrightarrow{\texttt{beta}} P_1 \oplus \mathsf{W}\langle Q\{x := \mathtt{v}\} \rangle \oplus P_2$$

$$P_1 \oplus \mathsf{W}\langle \mathtt{v}\ ;\ t \rangle \oplus P_2 \xrightarrow{\texttt{seq}} P_1 \oplus \mathsf{W}\langle t \rangle \oplus P_2$$

$$P_1 \oplus \mathsf{W}\langle \nu x.\, t \rangle \oplus P_2 \xrightarrow{\texttt{fresh}} P_1 \oplus \mathsf{W}\langle t\{x := y\} \rangle \oplus P_2 \qquad y \text{ fresh}$$

Reduction rules

Rules operate on the toplevel program.

$$P_1 \oplus \mathsf{W}\langle \lambda x.\, Q \rangle \oplus P_2 \xrightarrow{\texttt{alloc}} P_1 \oplus \mathsf{W}\langle \lambda^\ell x.\, Q \rangle \oplus P_2 \qquad \ell \text{ fresh}$$

$$P_1 \oplus \mathsf{W}\langle (\lambda^\ell x.\, Q)\ \mathtt{v} \rangle \oplus P_2 \xrightarrow{\texttt{beta}} P_1 \oplus \mathsf{W}\langle Q\{x := \mathtt{v}\} \rangle \oplus P_2$$

$$P_1 \oplus \mathsf{W}\langle \mathtt{v}\ ;\ t \rangle \oplus P_2 \xrightarrow{\texttt{seq}} P_1 \oplus \mathsf{W}\langle t \rangle \oplus P_2$$

$$P_1 \oplus \mathsf{W}\langle \nu x.\, t \rangle \oplus P_2 \xrightarrow{\texttt{fresh}} P_1 \oplus \mathsf{W}\langle t\{x := y\} \rangle \oplus P_2 \qquad y \text{ fresh}$$

$$P_1 \oplus \mathsf{W}\langle \mathtt{v} \doteq \mathtt{w} \rangle \oplus P_2 \xrightarrow{\texttt{unif}} P_1 \oplus \mathsf{W}\langle \mathbf{ok} \rangle^\sigma \oplus P_2$$

$$\sigma = \mathrm{mgu}(\{\mathtt{v} \doteq \mathtt{w}\})$$

Reduction rules

Rules operate on the toplevel program.

$$P_1 \oplus W\langle \lambda x.\, Q \rangle \oplus P_2 \xrightarrow{\texttt{alloc}} P_1 \oplus W\langle \lambda^{\ell} x.\, Q \rangle \oplus P_2 \qquad \ell \text{ fresh}$$

$$P_1 \oplus W\langle (\lambda^{\ell} x.\, Q)\ \mathtt{v} \rangle \oplus P_2 \xrightarrow{\texttt{beta}} P_1 \oplus W\langle Q\{x := \mathtt{v}\} \rangle \oplus P_2$$

$$P_1 \oplus W\langle \mathtt{v}\ ;\ t \rangle \oplus P_2 \xrightarrow{\texttt{seq}} P_1 \oplus W\langle t \rangle \oplus P_2$$

$$P_1 \oplus W\langle \nu x.\, t \rangle \oplus P_2 \xrightarrow{\texttt{fresh}} P_1 \oplus W\langle t\{x := y\} \rangle \oplus P_2 \qquad y \text{ fresh}$$

$$P_1 \oplus W\langle \mathtt{v} \doteq \mathtt{w} \rangle \oplus P_2 \xrightarrow{\texttt{unif}} P_1 \oplus W\langle \mathbf{ok} \rangle^{\sigma} \oplus P_2 \qquad \sigma = \mathrm{mgu}(\{\mathtt{v} \doteq \mathtt{w}\})$$

$$P_1 \oplus W\langle \mathtt{v} \doteq \mathtt{w} \rangle \oplus P_2 \xrightarrow{\texttt{fail}} P_1 \oplus P_2 \qquad \text{if } \mathrm{mgu}(\{\mathtt{v} \doteq \mathtt{w}\}) \text{ fails}$$

Unification

The most general unifier:

$$\mathrm{mgu}(\{\mathtt{v}_1 \stackrel{\bullet}{=} \mathtt{w}_1, \ldots, \mathtt{v}_n \stackrel{\bullet}{=} \mathtt{w}_n\})$$

can be computed as usual, with a few tweaks on the algorithm:

$$\{\lambda^{\ell} x.\, P \stackrel{\bullet}{=} \lambda^{\ell'} y.\, Q\} \uplus \mathsf{G} \quad \rightsquigarrow \quad \begin{cases} \mathsf{G} & \text{if } \ell = \ell' \\ \text{fails} & \text{otherwise} \end{cases}$$

If mgu(G) succeeds, it is an *idempotent most general unifier* for G.

Example

$$\begin{aligned}
\texttt{father} &\stackrel{\text{def}}{=} \lambda x.\, ((x \stackrel{\bullet}{=} \mathbf{a} \,;\, \mathbf{b}) \oplus (x \stackrel{\bullet}{=} \mathbf{b} \,;\, \mathbf{c})) \\
\texttt{grandfather} &\stackrel{\text{def}}{=} \lambda x.\, \texttt{father}\,(\texttt{father}\, x) \\
\texttt{grandson} &\stackrel{\text{def}}{=} \texttt{inverse}\,\texttt{grandfather} \\
\texttt{inverse} &\stackrel{\text{def}}{=} \lambda f.\, \lambda y.\, \nu x.\, (f\, x \stackrel{\bullet}{=} y \,;\, x)
\end{aligned}$$

$$\begin{aligned}
\texttt{grandson}\,\mathbf{c} \;&\twoheadrightarrow\; \nu x.\, ((\texttt{grandfather}\, x \stackrel{\bullet}{=} \mathbf{c}) \,;\, x) \\
&\twoheadrightarrow\; (\texttt{father}\,(\texttt{father}\, x) \stackrel{\bullet}{=} \mathbf{c}) \,;\, x \\
&\twoheadrightarrow\; (((\texttt{father}\, x \stackrel{\bullet}{=} \mathbf{a}) \,;\, \mathbf{b}) \stackrel{\bullet}{=} \mathbf{c}) \,;\, x \\
&\oplus\; (((\texttt{father}\, x \stackrel{\bullet}{=} \mathbf{b}) \,;\, \mathbf{c}) \stackrel{\bullet}{=} \mathbf{c}) \,;\, x \\
&\twoheadrightarrow\; ((((x \stackrel{\bullet}{=} \mathbf{a} \,;\, \mathbf{b}) \stackrel{\bullet}{=} \mathbf{a}) \,;\, \mathbf{b}) \stackrel{\bullet}{=} \mathbf{c}) \,;\, x \\
&\oplus\; ((((x \stackrel{\bullet}{=} \mathbf{b} \,;\, \mathbf{c}) \stackrel{\bullet}{=} \mathbf{a}) \,;\, \mathbf{b}) \stackrel{\bullet}{=} \mathbf{c}) \,;\, x \\
&\oplus\; ((((x \stackrel{\bullet}{=} \mathbf{a} \,;\, \mathbf{b}) \stackrel{\bullet}{=} \mathbf{b}) \,;\, \mathbf{c}) \stackrel{\bullet}{=} \mathbf{c}) \,;\, x \\
&\oplus\; ((((x \stackrel{\bullet}{=} \mathbf{b} \,;\, \mathbf{c}) \stackrel{\bullet}{=} \mathbf{b}) \,;\, \mathbf{c}) \stackrel{\bullet}{=} \mathbf{c}) \,;\, x \\
&\twoheadrightarrow\; \mathbf{a}
\end{aligned}$$

Example

Type inference algorithm for the simply-typed λ-calculus:

$$
\begin{aligned}
\mathbb{W}[x] &\stackrel{\text{def}}{=} a_x \\
\mathbb{W}[\lambda x.\, t] &\stackrel{\text{def}}{=} \nu a_x.\, \mathbf{fun}\, a_x\, \mathbb{W}[t] \\
\mathbb{W}[t\, s] &\stackrel{\text{def}}{=} \nu a.\, ((\mathbb{W}[t] \doteq \mathbf{fun}\, \mathbb{W}[s]\, a) \;;\; a)
\end{aligned}
$$

$$
\begin{aligned}
\mathbb{W}[\lambda x.\, \lambda y.\, y\, x] &= \nu a.\, \mathbf{fun}\, a\, (\nu b.\, \mathbf{fun}\, b\, (\nu c.\, (b \doteq \mathbf{fun}\, a\, c) \;;\; c)) \\
&\twoheadrightarrow \mathbf{fun}\, a\, (\mathbf{fun}\, (\mathbf{fun}\, a\, c)\, c)
\end{aligned}
$$

Example

Dynamic patterns:

$$(\lambda c.\, \lambda x.\, \nu y.\, (x \doteq (c\, y))\ ;\ y)\, \mathbf{d}\, (\mathbf{d}\, \mathbf{c})$$

$$\twoheadrightarrow \quad \nu y.\, ((\mathbf{d}\, \mathbf{c}) \doteq (\mathbf{d}\, y))\ ;\ y$$

$$\twoheadrightarrow \quad \mathbf{c}$$

Structural equivalence

Structural equivalence (between toplevel programs)

Reflexive, symmetric, and transitive closure of:

$$P \oplus t \oplus s \oplus Q \quad \equiv \quad P \oplus s \oplus t \oplus Q$$

$$P \oplus t \oplus Q \quad \equiv \quad P \oplus t\{x := y\} \oplus Q \quad \text{if } y \notin \mathrm{fv}(t)$$

$$P \oplus t \oplus Q \quad \equiv \quad P \oplus t\{\ell := \ell'\} \oplus Q \quad \text{if } \ell' \notin \mathrm{locs}(t)$$

Structural equivalence

Structural equivalence (between toplevel programs)

Reflexive, symmetric, and transitive closure of:

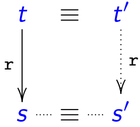
$$P \oplus t \oplus s \oplus Q \quad \equiv \quad P \oplus s \oplus t \oplus Q$$

$$P \oplus t \oplus Q \quad \equiv \quad P \oplus t\{x := y\} \oplus Q \quad \text{if } y \notin \mathsf{fv}(t)$$

$$P \oplus t \oplus Q \quad \equiv \quad P \oplus t\{\ell := \ell'\} \oplus Q \quad \text{if } \ell' \notin \mathsf{locs}(t)$$

Lemma

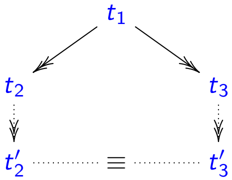
$\equiv$ is a strong bisimulation with respect to $\xrightarrow{\mathtt{r}}$ for each reduction rule $\mathtt{r}$.



Confluence

Theorem

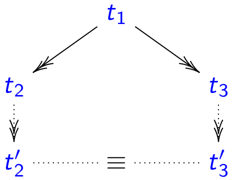
The $\lambda^{\mathtt{U}}$-calculus is confluent up to $\equiv$.



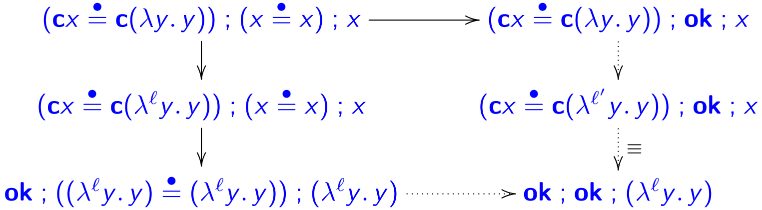
Confluence

Theorem

The $\lambda^{\mathtt{U}}$-calculus is confluent up to $\equiv$.

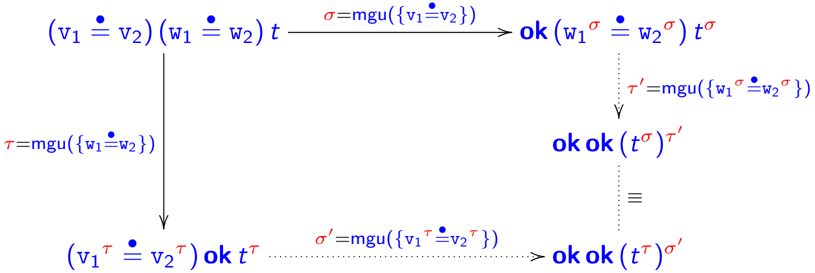


Example



Confluence

Another example



The equivalence relies on the fact that:

$\tau' \circ \sigma$ and $\sigma' \circ \tau$ are both most general unifiers of $\{\{v_1 \stackrel{\bullet}{=} v_2, w_1 \stackrel{\bullet}{=} w_2\}\}$

hence $\tau' \circ \sigma \equiv \sigma' \circ \tau$, up to renaming.

Proof of confluence

Simultaneous reduction $t \overset{G}{\Rightarrow} P$ collects all the unification goals G.

$$\frac{t \overset{G_1}{\Longrightarrow} \bigoplus_{i=1}^{n} t_i \qquad s \overset{G_2}{\Longrightarrow} \bigoplus_{j=1}^{m} s_j}{t\, s \xrightarrow{G_1 \cup G_2} \bigoplus_{i=1}^{n} \bigoplus_{j=1}^{m} t_i\, s_j} \qquad \frac{}{\mathtt{v} \stackrel{\bullet}{=} \mathtt{w} \xrightarrow{\{\mathtt{v} \stackrel{\bullet}{=} \mathtt{w}\}} \textbf{ok}}$$

Moreover:

$$\frac{t_i \overset{G_i}{\Longrightarrow} P_i \qquad Q_i := \begin{cases} {P_i}^{\sigma} & \text{if } \sigma = \mathrm{mgu}(G_i) \\ \mathtt{fail} & \text{if } \mathrm{mgu}(G_i) \text{ fails} \end{cases} \qquad \text{for each } i = 1..n}{\bigoplus_{i=1}^{n} t_i \Rightarrow \bigoplus_{i=1}^{n} Q_i}$$

Proof of confluence

Key lemma

If $t \overset{\mathsf{G}}{\Rightarrow} P$ then $t^\sigma \overset{\mathsf{G}^\sigma}{\Longrightarrow} P^\sigma$.

Tait–Martin-Löf's technique, up to $\equiv$

1. $\rightarrow\ \subseteq\ \Rightarrow\equiv$
2. $\Rightarrow\ \subseteq\ \rightarrow^*\equiv$
3. $\Rightarrow$ has the diamond property, up to $\equiv$.

Normal forms

Normal programs

$$P^\star ::= \bigoplus_{i=1}^{n} t_i^\star$$

Normal terms

$$t^\star ::= \mathtt{v} \mid S$$

Stuck terms

$$\begin{array}{rcll} S & ::= & x\, t_1^\star \ldots t_n^\star & n > 0 \\ & | & \mathbf{c}\, t_1^\star \ldots t_n^\star & \text{if } t_i^\star \text{ stuck for some } i = 1..n \\ & | & (t_1^\star \,;\, t_2^\star)\, s_1^\star \ldots s_n^\star & \text{if } t_1^\star \text{ stuck} \\ & | & (t_1^\star \stackrel{\bullet}{=} t_2^\star)\, s_1^\star \ldots s_n^\star & \text{if } t_i^\star \text{ stuck for some } i = 1..2 \\ & | & (\lambda^\ell x.\, P)\, t^\star\, s_1^\star \ldots s_n^\star & \text{if } t^\star \text{ stuck} \end{array}$$

Proposition

Normal forms of $\rightsquigarrow$ are given exactly by the grammar $P^\star$.

Example

$$f\,\mathbf{c} \stackrel{\bullet}{=} \mathbf{c}$$

Type system

Providing a system of **simple types** is straightforward.

The rule for $\nu x.\, t$ is logically unsound.

$$\frac{\Gamma, x : B \vdash t : A}{\Gamma \vdash \nu x.\, t : A}$$

(Weak) subject reduction

- If $\Gamma \vdash P : A$ and $P \xrightarrow{\neg\texttt{fresh}} Q$ then $\Gamma \vdash Q : A$.
- If $\Gamma \vdash P : A$ and $P \xrightarrow{\texttt{fresh}(x)} Q$ then $\Gamma, x : B \vdash Q : A$ for some B.

Normalization (?)

We have **not** been able to prove **strong normalization** for the simply typed variant of $\lambda^{\mathtt{U}}$ using:

- Reducibility candidates (Tait/Girard)
- Decreasing degrees of created redexes (Turing, Prawitz, ...)
- Increasing functionals (de Vrijer)
- Stratified regions (Amadio)

Types of constructors should verify a positivity condition.

E.g. if $\mathbf{c} : (A \to A) \to A$ we can type a non-terminating term:

$$\omega \stackrel{\text{def}}{=} \lambda x^A. \nu y^{A \to A}. (\mathbf{c} y \stackrel{\bullet}{=} x \,;\, y\, x)$$

$$\omega(\mathbf{c}\,\omega) \to^+ \omega(\mathbf{c}\,\omega)$$

Table of Contents

A naive denotational semantics

$$[\![A \to B]\!] \stackrel{\text{def}}{=} [\![A]\!] \to \mathcal{P}([\![B]\!])$$

$$\begin{aligned}
[\![x^A]\!]_\rho &\stackrel{\text{def}}{=} \{\rho(x^A)\} \\
[\![\mathbf{c}]\!]_\rho &\stackrel{\text{def}}{=} \{\mathrm{R}_{\mathbf{c}}\} \\
[\![\lambda x^A.\, P]\!]_\rho &\stackrel{\text{def}}{=} \{\curlywedge a^{[\![A]\!]}.\, [\![P]\!]_{\rho[x^A \mapsto a]}\} \\
[\![\lambda^\ell x^A.\, P]\!]_\rho &\stackrel{\text{def}}{=} \{\curlywedge a^{[\![A]\!]}.\, [\![P]\!]_{\rho[x^A \mapsto a]}\} \\
[\![t\ s]\!]_\rho &\stackrel{\text{def}}{=} \{b \mid f \in [\![t]\!]_\rho, a \in [\![s]\!]_\rho, b \in f(a)\} \\
[\![t \doteq s]\!]_\rho &\stackrel{\text{def}}{=} \{\mathrm{R}_{\mathbf{ok}} \mid a \in [\![t]\!]_\rho, b \in [\![s]\!]_\rho, a = b\} \\
[\![t \,;\, s]\!]_\rho &\stackrel{\text{def}}{=} \{a \mid b \in [\![t]\!]_\rho, a \in [\![s]\!]_\rho\} \\
[\![\nu x^A.\, t]\!]_\rho &\stackrel{\text{def}}{=} \{b \mid a \in [\![A]\!], b \in [\![t]\!]_{\rho[x^A \mapsto a]}\} \\
[\![\mathtt{fail}^A]\!]_\rho &\stackrel{\text{def}}{=} \varnothing \\
[\![t \oplus P]\!]_\rho &\stackrel{\text{def}}{=} [\![t]\!]_\rho \cup [\![P]\!]_\rho
\end{aligned}$$

$$[\![P]\!] \stackrel{\text{def}}{=} \bigcup_\rho [\![P]\!]_\rho$$

A naive denotational semantics

Correctness

If $P \to Q$ then $[\![P]\!] = [\![Q]\!]$. (Work in progress)

Table of Contents

Future work

Translation from a pattern calculus (*e.g.* PPC)	★★
Extend with new constructs (*e.g.* "$\forall(P)$")	★★★
Evaluation strategies / abstract machines	★★★
Richer type systems (*e.g.* instantiation patterns)	★★★★
Strong normalization	★★★★ *"Truly frightening"*