Semantics of a Relational λ-Calculus

ICTAC 2020

December 2nd, 2020

Pablo Barenbaum[1,2]
Federico Lochbaum[2]
Mariana Milicich[1]

[1] Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
Argentina

[2] Universidad Nacional de Quilmes
Argentina

Outline

Motivation

Functional programming	**Logic/relational programming**
Inductive datatypes	First order terms, symbolic variables
Pattern matching	Unification
Higher-order functions	Relations, invertible predicates
Lazy evaluation	Non-deterministic search

Motivation

Functional programming

Inductive datatypes
Pattern matching
Higher-order functions
Lazy evaluation

Logic/relational programming

First order terms, symbolic variables
Unification
Relations, invertible predicates
Non-deterministic search

Functional logic programming

```
mother Hera     = Rhea
mother Demeter  = Hera

grandmother     = mother ∘ mother

granddaughter   = inv grandmother

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

Motivation — Formal semantics

Functional programming
λ-**calculus**

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

Logic/relational programming
miniKanren

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

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

Motivation — Formal semantics

Functional programming
λ-calculus

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

Logic/relational programming
miniKanren

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

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

Functional logic programming

???

Motivation — Formal semantics

Functional programming
λ-calculus

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

Logic/relational programming
miniKanren

$$G ::= T \overset{\bullet}{=} T \mid R(T_1, \ldots, T_n) \mid G \mathbin{;} G \mid G \oplus G \mid \nu x.G$$

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

Functional logic programming

Goal:

Confluent λ-calculus with relational constructs.

Related work

Functional logic programming languages

λProlog (Miller et al., 1986), **Mercury** (Somogyi et al., 1995),
Curry (Hanus et al., 1997), **Makam** (Stampoulis, 2018), **...**

Related formalisms

- **Pattern calculi** **No full unification, no non-determinism.**
  Jay & Kesner (2006), Klop et al. (2008), Petit (2011), ...

- **λ-calculi with non-deterministic choice** **No unification.**
  Schmidt-Schauß et al. (2000), Faggian & Della Rocca (2019), ...

- **miniKanren** **No λ-abstractions/applications.**
  Rozhplokas et al. (2019)

- **λ-calculi with non-deterministic choice and unification**
  Smolka (1997), Chakravarty et al. (1998) **Not confluent.**
  Albert et al. (2002) **Big-step semantics, no confluence.**

Outline

Our first approach

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

Difficulty

We cannot solve higher-order unification

$$f\,\mathbf{c} \doteq \mathbf{c}$$

Higher-order unification is undecidable. (Only semi-decidable).

`Huet, 1973`

Higher-order unification problems have no most general unifiers.
Existence of mgu's is key for confluence.

There are well-known restrictions of higher-order unification:

Higher-order pattern unification. `Miller, 1991`

Nominal unification. `Urban et al., 2004`

They require strong reduction (under abstractions).
They fall back on full higher-order unification.

Difficulty

We cannot solve higher-order unification

...but we **do** want to match functions

$$(x \doteq \lambda y.\, y) \mathbin{;} (x \doteq x) \to (\lambda y.\, y) \doteq (\lambda y.\, y) \to \text{(should succeed)}$$

...but functions cannot be compared by syntactic equality

This is not stable under substitution.

E.g.:

$$(\lambda x.\, y) \doteq (\lambda x.\, z) \qquad \text{fails}$$

but if $y \mapsto z$,

$$(\lambda x.\, z) \doteq (\lambda x.\, z) \qquad \text{succeeds}$$

Outline

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
		$\mid$	$t \doteq t$	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 $P = t_1 \oplus \ldots \oplus t_n$.

Invariant

Any two abstractions with the same location are closed and equal.

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

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

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.

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

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

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

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

Reduction rules

Rules operate on the toplevel program.

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

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

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

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

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

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

Reduction rules

Rules operate on the toplevel program.

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

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

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

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

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

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

Two abstractions unify iff they have the same location.

Confluence

The $\lambda^{\mathtt{U}}$-calculus is **confluent**, up to a notion of structural equivalence $\equiv$.

$$
\begin{tikzcd}
 & t_1 \arrow[dl] \arrow[dr] & \\
t_2 \arrow[d, dotted, two heads] & & t_3 \arrow[d, dotted, two heads] \\
t_2' \arrow[r, dotted, no head] & \equiv \arrow[r, dotted, no head] & t_3'
\end{tikzcd}
$$

Confluence

Example

$$
\begin{tikzcd}
(\mathtt{v}_1 \stackrel{\bullet}{=} \mathtt{v}_2)\,(\mathtt{w}_1 \stackrel{\bullet}{=} \mathtt{w}_2)\; t \arrow[r, "\sigma = \mathrm{mgu}(\{\mathtt{v}_1 \stackrel{\bullet}{=} \mathtt{v}_2\})"] \arrow[dd, "\tau = \mathrm{mgu}(\{\mathtt{w}_1 \stackrel{\bullet}{=} \mathtt{w}_2\})"'] & \mathbf{ok}\,({\mathtt{w}_1}^{\sigma} \stackrel{\bullet}{=} {\mathtt{w}_2}^{\sigma})\; t^{\sigma} \arrow[d, dotted, "\tau' = \mathrm{mgu}(\{{\mathtt{w}_1}^{\sigma} \stackrel{\bullet}{=} {\mathtt{w}_2}^{\sigma}\})"] \\
 & \mathbf{ok}\,\mathbf{ok}\,(t^{\sigma})^{\tau'} \arrow[d, dotted, dash, "\equiv"] \\
({\mathtt{v}_1}^{\tau} \stackrel{\bullet}{=} {\mathtt{v}_2}^{\tau})\; \mathbf{ok}\; t^{\tau} \arrow[r, dotted, "\sigma' = \mathrm{mgu}(\{{\mathtt{v}_1}^{\tau} \stackrel{\bullet}{=} {\mathtt{v}_2}^{\tau}\})"] & \mathbf{ok}\,\mathbf{ok}\,(t^{\tau})^{\sigma'}
\end{tikzcd}
$$

The equivalence relies on the fact that:

$\tau' \circ \sigma$ and $\sigma' \circ \tau$ are both most general unifiers of $\{\{\mathtt{v}_1 \stackrel{\bullet}{=} \mathtt{v}_2, \mathtt{w}_1 \stackrel{\bullet}{=} \mathtt{w}_2\}\}$

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

Type system

We have formulated a system of **simple types** for $\lambda^{\mathtt{U}}$.

Subject reduction

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

Outline

Motivation

Difficulties

Operational semantics

Denotational semantics

Future work

A denotational semantics

We have defined a naive **denotational semantics** for $\lambda^{\mathtt{U}}$:

$$
\begin{array}{rcl}
[\![A \to B]\!] & \stackrel{\text{def}}{=} & [\![A]\!] \to \mathcal{P}([\![B]\!]) \\[2em]
[\![x^A]\!]_\rho & \stackrel{\text{def}}{=} & \{\rho(x^A)\} \\
[\![\mathbf{c}]\!]_\rho & \stackrel{\text{def}}{=} & \{\mathsf{R}_{\mathbf{c}}\} \\
[\![\lambda x^A.\, P]\!]_\rho & \stackrel{\text{def}}{=} & \{\curlywedge a^{[\![A]\!]}.\, [\![P]\!]_{\rho[x^A \mapsto a]}\} \\
[\![\lambda^\ell x^A.\, P]\!]_\rho & \stackrel{\text{def}}{=} & \{\curlywedge a^{[\![A]\!]}.\, [\![P]\!]_{\rho[x^A \mapsto a]}\} \\
[\![t\; s]\!]_\rho & \stackrel{\text{def}}{=} & \{b \mid \exists f \in [\![t]\!]_\rho, \exists a \in [\![s]\!]_\rho, b \in f(a)\} \\
[\![t \stackrel{\bullet}{=} s]\!]_\rho & \stackrel{\text{def}}{=} & \{\mathsf{R}_{\mathbf{ok}} \mid \exists a \in [\![t]\!]_\rho, \exists b \in [\![s]\!]_\rho, a = b\} \\
[\![t \,;\, s]\!]_\rho & \stackrel{\text{def}}{=} & \{a \mid \exists b \in [\![t]\!]_\rho, a \in [\![s]\!]_\rho\} \\
[\![\nu x^A.\, t]\!]_\rho & \stackrel{\text{def}}{=} & \{b \mid \exists a \in [\![A]\!], b \in [\![t]\!]_{\rho[x^A \mapsto a]}\} \\
[\![\mathtt{fail}^A]\!]_\rho & \stackrel{\text{def}}{=} & \varnothing \\
[\![t \oplus P]\!]_\rho & \stackrel{\text{def}}{=} & [\![t]\!]_\rho \cup [\![P]\!]_\rho \\[2em]
[\![P]\!] & \stackrel{\text{def}}{=} & \bigcup_\rho [\![P]\!]_\rho
\end{array}
$$

A naive denotational semantics

Correctness

If $P \to Q$ then $[\![P]\!] \supseteq [\![Q]\!]$.

A naive denotational semantics

Correctness

If $P \to Q$ then $[\![P]\!] \supseteq [\![Q]\!]$.

(Completeness fails).

Outline

Future work

- We have a working **prototype** programming language (`Ñuflo`).

- Relate $\lambda^{\mathtt{U}}$ with pattern calculi.
- Study evaluation strategies and abstract machines.
- Formulate a complete denotational semantics.
- ...