Programming Languages

Lambda Calculus

Introduction

Cálculo-λ^b: syntax and typing

Cálculo-λ^b: operational semantics

Cálculo-λ^{bn}: extension with natural numbers

What is the λ-calculus?

A programming language defined in a rigorous way.
It is based only on two operations: building functions and applying them.

Historically

- Conceived in the 1930s by Alonzo Church to formalize the notion of effectively computable function.
- Used since the 1960s to study formal semantics of programming languages.

Currently

- Core of functional programming languages and proof assistants.
  Lisp, OCaml, Haskell, Coq, Agda, Lean, ...
- Laboratory to investigate new language features.
- Strongly connected with proof theory, constructive mathematics, category theory, ...

Introduction

Cálculo-λ^b: syntax and typing

Cálculo-λ^b: operational semantics

Cálculo-λ^{bn}: extension with natural numbers

The λ^b calculus

Syntax of types

$$\begin{array}{rcl} \tau, \sigma, \rho, \ldots & ::= & \mathsf{bool} \\ & \mid & \tau \to \sigma \end{array}$$

We assume the type constructor "$\to$" is right-associative:

$$\tau \to \sigma \to \rho \quad = \quad \tau \to (\sigma \to \rho) \quad \neq \quad (\tau \to \sigma) \to \rho$$

The λ^b calculus

Assume a countably infinite set of variables is given:

$$\mathcal{X} = \{x, y, z, \ldots\}$$

Syntax of terms

$M, N, P, \ldots$	$::=$	x	variable
	$\mid$	$\lambda x : \tau.\, M$	abstraction
	$\mid$	$M\, N$	application
	$\mid$	true	true
	$\mid$	false	false
	$\mid$	if M then N else P	conditional

We assume application is left-associative:

$$M\, N\, P \quad = \quad (M\, N)\, P \quad \neq \quad M\, (N\, P)$$

Abstraction and "if" have lower precedence than application:

$$\lambda x : \tau.\, M\, N \quad = \quad \lambda x : \tau.\, (M\, N) \quad \neq \quad (\lambda x : \tau.\, M)\, N$$

The λ^b calculus

Examples of terms

- $\lambda x : \mathsf{bool}.\, x$
- $\lambda x : \mathsf{bool} \to \mathsf{bool}.\, x$
- $(\lambda x : \mathsf{bool}.\, x)\, \mathsf{false}$
- $(\lambda x : \mathsf{bool} \to \mathsf{bool}.\, x)\, (\lambda y : \mathsf{bool}.\, y)$
- $(\lambda x : \mathsf{bool}.\, \lambda y : \mathsf{bool} \to \mathsf{bool}.\, y\, x)\, \mathsf{true}$
- $\lambda x : \mathsf{bool}.\, \mathsf{if}\ x\ \mathsf{then}\, \mathsf{false}\ \mathsf{else}\ \mathsf{true}$
- $\mathsf{true}\, \mathsf{true}$
- $\mathsf{if}\ \lambda x : \mathsf{bool}.\, x\ \mathsf{then}\, \mathsf{false}\ \mathsf{else}\ \mathsf{true}$

Free and bound variables

An occurrence of x is **bound** if it appears inside an abstraction "$\lambda\, x$". An occurrence of x is **free** if it is not bound.

Example

Mark free and bound variable occurrences:

$$(\lambda x : \mathsf{bool} \to \mathsf{bool}.\, \lambda y : \mathsf{bool}.\, x\, y)\, (\lambda y : \mathsf{bool}.\, x\, y)\, y$$

Exercise

Define the set of free variables $\mathrm{fv}(M)$ of M.

Alpha equivalence

Terms that differ only in the name of *bound* variables are considered equal:

$$\begin{array}{lcl} \lambda x : \tau.\, \lambda y : \sigma.\, x & = & \lambda y : \tau.\, \lambda x : \sigma.\, y \;=\; \lambda a : \tau.\, \lambda b : \sigma.\, a \\ \lambda x : \tau.\, \lambda y : \sigma.\, x & \neq & \lambda x : \tau.\, \lambda y : \sigma.\, y \;=\; \lambda x : \tau.\, \lambda x : \sigma.\, x \end{array}$$

Type system

The notion of "typability" is formalized with a deductive system.

Problem

What type does x have?

Typing contexts

A **typing context** is a finite set of pairs $(x_i : \tau_i)$:

$$\{x_1 : \tau_1, \ldots, x_n : \tau_n\}$$

with no repeated variables ($i \neq j \Rightarrow x_i \neq x_j$).
They are denoted with uppercase Greek letters ($\Gamma, \Delta, \ldots$).
Sometimes we denote $\mathrm{dom}(\Gamma) = \{x_1, \ldots, x_n\}$.

Typing judgments

The type system predicates over **typing judgments**, of the form:

$$\Gamma \vdash M : \tau$$

Type system

Typing rules

$$\frac{}{\Gamma \vdash \mathsf{true} : \mathsf{bool}}\,\text{T-TRUE} \qquad \frac{}{\Gamma \vdash \mathsf{false} : \mathsf{bool}}\,\text{T-FALSE}$$

$$\frac{\Gamma \vdash M : \mathsf{bool} \quad \Gamma \vdash N : \tau \quad \Gamma \vdash P : \tau}{\Gamma \vdash \mathsf{if}\ M\ \mathsf{then}\ N\ \mathsf{else}\ P : \tau}\,\text{T-IF}$$

$$\frac{}{\Gamma, x : \tau \vdash x : \tau}\,\text{T-VAR} \qquad \frac{\Gamma, x : \tau \vdash M : \sigma}{\Gamma \vdash \lambda x : \tau.\, M : \tau \to \sigma}\,\text{T-ABS}$$

$$\frac{\Gamma \vdash M : \tau \to \sigma \quad \Gamma \vdash N : \tau}{\Gamma \vdash M\, N : \sigma}\,\text{T-APP}$$

Type system

Example — typing judgment derivations

Derive, if possible, typing judgments for the following terms:

1. $\lambda x : \mathsf{bool}.\ \mathsf{if}\ x\ \mathsf{then}\ \mathsf{false}\ \mathsf{else}\ x$
2. $\lambda y : \mathsf{bool} \to \mathsf{bool} \to \mathsf{bool}.\ \lambda z : \mathsf{bool}.\ y\ (y\ x\ z)$
3. $x\ z\ (y\ z)$
4. $\lambda x : \mathsf{bool} \to \mathsf{bool}.\ \lambda x : \mathsf{bool}.\ x$
5. $\mathsf{true}\ (\lambda x : \mathsf{bool}.\ x)$
6. $x\ x$

Properties of the type system

Theorem (Uniqueness of types)

If $\Gamma \vdash M : \tau$ and $\Gamma \vdash M : \sigma$ are derivable, then $\tau = \sigma$.

Theorem (Weakening + Strengthening)

If $\Gamma \vdash M : \tau$ is derivable and $\mathrm{fv}(M) \subseteq \mathrm{dom}(\Gamma \cap \Gamma')$ then $\Gamma' \vdash M : \tau$ is derivable.

Introduction

Cálculo-λ^b: syntax and typing

Cálculo-λ^b: operational semantics

Cálculo-λ^{bn}: extension with natural numbers

Formal semantics

The type system indicates how programs are constructed.
We also want to give them **meaning** (semantics).

Different ways of giving formal semantics

1. **Operational semantics.**
   Indicates how the program is executed until it reaches a result.

   Small-step semantics: step-by-step execution.
   Big-step semantics: direct evaluation to the result.

2. **Denotational semantics.**
   Interprets programs as mathematical objects.
3. **Axiomatic semantics.**
   Establishes logical relationships between the program state before and after execution.
4. ...

We will work with small-step operational semantics.

Small-step operational semantics

Programs

A **program** is a typable and *closed* term M ($\mathrm{fv}(M) = \emptyset$):

- The typing judgment $\vdash M : \tau$ must be derivable for some τ.

Evaluation judgments

The operational semantics predicates over **evaluation judgments**:

$$M \to N$$

where M and N are programs.

Values

Values are the possible results of evaluating programs:

$$V ::= \mathsf{true} \mid \mathsf{false} \mid \lambda x : \tau.\, M$$

Small-step operational semantics

Evaluation rules for boolean expressions

$$\frac{}{\text{if true then } M \text{ else } N \to M}\,\text{E-IF}\textsc{True}$$

$$\frac{}{\text{if false then } M \text{ else } N \to N}\,\text{E-IF}\textsc{False}$$

$$\frac{M \to M'}{\text{if } M \text{ then } N \text{ else } P \to \text{if } M' \text{ then } N \text{ else } P}\,\text{E-IF}$$

Small-step operational semantics

Example

1. Derive the following judgment:

$$\begin{array}{ll} & \text{if } (\text{if } \text{false} \text{ then } \text{false} \text{ else } \text{true}) \text{ then } \text{false} \text{ else } \text{true} \\ \rightarrow & \text{if } \text{true} \text{ then } \text{false} \text{ else } \text{true} \end{array}$$

2. For which terms M does $\text{true} \rightarrow M$ hold?
3. Is it possible to derive the following judgment?

$$\begin{array}{ll} & \text{if } \text{true} \text{ then } (\text{if } \text{false} \text{ then } \text{false} \text{ else } \text{false}) \text{ else } \text{true} \\ \rightarrow & \text{if } \text{true} \text{ then } \text{false} \text{ else } \text{true} \end{array}$$

Small-step operational semantics

Evaluation rules for functions (abstraction and application)

$$\frac{M \to M'}{M\ N \to M'\ N}\ \text{E-APP1}$$

$$\frac{N \to N'}{(\lambda x : \tau.\ M)\ N \to (\lambda x : \tau.\ M)\ N'}\ \text{E-APP2}$$

$$\frac{}{(\lambda x : \tau.\ M)\ V \to M\{x := V\}}\ \text{E-APPABS}$$

Substitution

The **substitution** operation:

$$M\{x := N\}$$

denotes the term resulting from replacing all free occurrences of x in M by N.

Substitution

Definition of substitution

$$
\begin{aligned}
x\{x := N\} &\stackrel{\text{def}}{=} N \\
a\{x := N\} &\stackrel{\text{def}}{=} a \text{ if } a \in \{\mathsf{true}, \mathsf{false}\} \cup \mathcal{X} \setminus \{x\} \\
(\text{if } M \text{ then } P \text{ else } Q)\{x := N\} &\stackrel{\text{def}}{=} \text{if } M\{x := N\} \\
&\qquad \text{then } P\{x := N\} \\
&\qquad \text{else } Q\{x := N\} \\
(M_1\, M_2)\{x := N\} &\stackrel{\text{def}}{=} M_1\{x := N\}\, M_2\{x := N\} \\
(\lambda y : \tau.\, M)\{x := N\} &\stackrel{\text{def}}{=}
\end{aligned}
$$

$$
\begin{cases}
\lambda y : \tau.\, M & \text{if } x = y \\
\lambda y : \tau.\, M\{x := N\} & \text{if } x \neq y,\ y \notin \mathsf{fv}(N) \\
\lambda z : \tau.\, M\{y := z\}\{x := N\} & \text{if } x \neq y,\ y \in \mathsf{fv}(N), \\
 & \quad z \notin \{x, y\} \cup \mathsf{fv}(M) \cup \mathsf{fv}(N)
\end{cases}
$$

Substitution

Definition of substitution (alternative)

$$
\begin{array}{rcl}
x\{x := N\} & \stackrel{\text{def}}{=} & N \\
a\{x := N\} & \stackrel{\text{def}}{=} & a \text{ if } a \in \{\mathsf{true}, \mathsf{false}\} \cup \mathcal{X} \setminus \{x\} \\
(\text{if } M \text{ then } P \text{ else } Q)\{x := N\} & \stackrel{\text{def}}{=} & \text{if } M\{x := N\} \\
 & & \text{then } P\{x := N\} \\
 & & \text{else } Q\{x := N\} \\
(M_1\, M_2)\{x := N\} & \stackrel{\text{def}}{=} & M_1\{x := N\}\, M_2\{x := N\} \\
(\lambda y : \tau.\, M)\{x := N\} & \stackrel{\text{def}}{=} & \lambda y : \tau.\, M\{x := N\} \\
 & & \text{assuming } y \notin \{x\} \cup \mathsf{fv}(N)
\end{array}
$$

The assumption can always be satisfied by renaming the bound variable "y" in case of conflict.

Small-step operational semantics

We define a **multi-step evaluation judgment**:

$$M \twoheadrightarrow N$$

where M and N are programs.

Multi-step evaluation rules

$$\frac{}{M \twoheadrightarrow M} \qquad \frac{M \rightarrow N \quad N \twoheadrightarrow P}{M \twoheadrightarrow N}$$

Small-step operational semantics

Example — evaluation

Repeatedly reduce the following term until reaching a value:

$$(\lambda x : \mathsf{bool}.\, \lambda f : \mathsf{bool} \to \mathsf{bool}.\, f\,(f\,x))\,\mathsf{true}\,(\lambda x : \mathsf{bool}.\, x)$$

Properties of evaluation

Theorem (Determinism)

If $M \to N_1$ and $M \to N_2$ then $N_1 = N_2$.

Theorem (Type preservation)

If $\vdash M : \tau$ and $M \to N$ then $\vdash N : \tau$.

Theorem (Progress)

If $\vdash M : \tau$ then:

1. Either M is a value.
2. Or there exists N such that $M \to N$.

Theorem (Termination)

If $\vdash M : \tau$, then there is no infinite sequence of steps:

$$M \to M_1 \to M_2 \to \dots$$

Properties of evaluation

Corollary (Canonicity)

1. If $\vdash M : \text{bool}$ is derivable, then the evaluation of M terminates and the result is true or false.
2. If $\vdash M : \tau \to \sigma$ is derivable, then the evaluation of M terminates and the result is an abstraction.

Slogan

Well typed programs cannot go wrong. (Robin Milner)

The λ^{bn} calculus

Syntax: types

$$\tau, \sigma, \ldots \quad ::= \quad \ldots \mid \mathsf{nat}$$

Syntax: terms

			Informal semantics
M	$::=$	$\ldots$	
	$\mid$	zero	the number zero
	$\mid$	succ(M)	the successor of the number represented by M
	$\mid$	pred(M)	the predecessor of the number represented by M
	$\mid$	isZero(M)	represents a boolean true/false, depending on whether M represents zero or not

The λ^{bn} calculus: typing rules

$$\frac{}{\Gamma \vdash \mathsf{zero} : \mathsf{nat}}\,\textsc{T-Zero}$$

$$\frac{\Gamma \vdash M : \mathsf{nat}}{\Gamma \vdash \mathsf{succ}(M) : \mathsf{nat}}\,\textsc{T-Succ} \qquad \frac{\Gamma \vdash M : \mathsf{nat}}{\Gamma \vdash \mathsf{pred}(M) : \mathsf{nat}}\,\textsc{T-Pred}$$

$$\frac{\Gamma \vdash M : \mathsf{nat}}{\Gamma \vdash \mathsf{isZero}(M) : \mathsf{bool}}\,\textsc{T-IsZero}$$

The λ^{bn} calculus: values

We extend the set of **values**:

$$V ::= \ldots \mid \mathsf{zero} \mid \mathsf{succ}(V)$$

The λ^{bn} calculus: operational semantics

$$\frac{M \to M'}{\mathsf{succ}(M) \to \mathsf{succ}(M')}\,\textsc{E-Succ}$$

$$\frac{M \to M'}{\mathsf{pred}(M) \to \mathsf{pred}(M')}\,\textsc{E-Pred}$$

$$\frac{}{\mathsf{pred}(\mathsf{succ}(V)) \to V}\,\textsc{E-PredSucc}$$

$$\frac{M \to M'}{\mathsf{isZero}(M) \to \mathsf{isZero}(M')}\,\textsc{E-IsZero}$$

$$\frac{}{\mathsf{isZero}(\mathsf{zero}) \to \mathsf{true}}\,\textsc{E-IsZeroZero}$$

$$\frac{}{\mathsf{isZero}(\mathsf{succ}(V)) \to \mathsf{false}}\,\textsc{E-IsZeroSucc}$$

The λ^{bn} calculus: operational semantics

Example

1. Evaluate isZero(succ(pred(succ(zero)))).
2. Evaluate isZero(pred(pred(succ(zero)))). (What happens?)

Normal form ("n.f.")

A program M is in **n.f.** if there is no M' such that $M \to M'$.

Are all closed typable n.f.'s values?

In the λ^{b} calculus **yes**.
In the λ^{bn} calculus **no**. (Which property stops holding?)

Normal forms that are not values are called **error terms**.

¿ ¿ ¿ ¿ ¿ ¿ ¿ ¿ ¿ ? ? ? ? ? ? ? ? ?

Recommended reading

Chapters 5 and 9 of Pierce's book.

Benjamin C. Pierce. *Types and Programming Languages*
The MIT Press, 2002.