Lambda-calculus and programming language semantics

Thibaut Balabonski @ UPSay
Fall 2023
https://www.lri.fr/~blsk/LambdaCalculus/

Chapter 3: simply-typed λ-calculus

1 Wrong programs

Wrong program in Python

```
p = (4, 2)
return p[1][0]
```

Runtime error

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'int' object has no attribute
           '__getitem__'
```

Wrong program in Caml

```
let p = (4, 2) in
fst(snd p)
```

Compile-time error

```
Error: This expression has type int but an
       expression was expected of type 'a * 'b
```

Wrong λ-term

$$\begin{array}{ll} & (\lambda x.\pi_1(\pi_2(x)))\ ((\lambda y.\langle y, (\lambda z.z)2\rangle)4) \\ \to_v & (\lambda x.\pi_1(\pi_2(x)))\ \langle 4, (\lambda z.z)2\rangle \\ \to_v & (\lambda x.\pi_1(\pi_2(x)))\ \langle 4, 2\rangle \\ \to_v & \pi_1(\pi_2(\langle 4, 2\rangle)) \\ \to_v & \pi_1(2) \end{array}$$

blocked term: not a value, yet not reducible

Motto

Well-typed programs do not go wrong

Connects a static analysis

- expressions have consistent types

with a semantic property

- the programs runs smoothly

2 Simple types

Typed syntax

Types

$$\begin{array}{rcll} \sigma, \tau & ::= & o & \text{base types} \\ & | & \sigma \longrightarrow \tau & \text{function types} \end{array}$$

Terms

$$\begin{array}{rcll} t & ::= & x & \text{variable} \\ & | & \lambda x^{\sigma}.t & \text{typed abstraction} \\ & | & t_1\ t_2 & \text{application} \end{array}$$

Notation $\tau_n \longrightarrow (\tau_{n-1} \dots (\tau_1 \longrightarrow \tau_0) \dots)$ is written $\tau_n \longrightarrow \tau_{n-1} \dots \tau_1 \longrightarrow \tau_0$

Simple types, *à la Church*

Typing judgment

$$\Gamma \vdash t : \sigma$$

the term t is well typed with type σ in the environment Γ with Γ: a set of typed variables $\{x_1^{\sigma_1}, \dots, x_n^{\sigma_n}\}$

$$\frac{x^{\tau} \in \Gamma}{\Gamma \vdash x : \tau} \qquad \frac{\Gamma, x^{\sigma} \vdash e : \tau}{\Gamma \vdash \lambda x^{\sigma}.e : \sigma \longrightarrow \tau} \qquad \frac{\Gamma \vdash e_1 : \sigma \longrightarrow \tau \qquad \Gamma \vdash e_2 : \sigma}{\Gamma \vdash e_1\ e_2 : \tau}$$

Simple types, without annotations

Typing judgment

$$\Gamma \vdash t : \sigma$$

the term t is well typed with type σ in the environment Γ with Γ: a function from variables to types $\{x_1 : \sigma_1, \dots, x_n : \sigma_n\}$

$$\frac{\Gamma(x) = \tau}{\Gamma \vdash x : \tau} \qquad \frac{\Gamma, x : \sigma \vdash e : \tau}{\Gamma \vdash \lambda x.e : \sigma \longrightarrow \tau} \qquad \frac{\Gamma \vdash e_1 : \sigma \longrightarrow \tau \qquad \Gamma \vdash e_2 : \sigma}{\Gamma \vdash e_1\ e_2 : \tau}$$

Exercise: examples and counter-examples

Give typing judgments for the following terms, or justify that this cannot be done

- $\lambda x.x$
- $\lambda xy.x$
- $\lambda xyz.x(yz)$
- $\lambda x.xx$

Extended types: integers

New type

$$\begin{array}{rcl} \sigma, \tau & ::= & \dots \\ & | & \mathsf{int} \end{array}$$

New typing rules

$$\frac{}{\Gamma \vdash n : \mathsf{int}} \qquad \frac{\Gamma \vdash t_1 : \mathsf{int} \qquad \Gamma \vdash t_2 : \mathsf{int}}{\Gamma \vdash t_1 \oplus t_2 : \mathsf{int}}$$

Extended types: booleans

New type

$$\begin{array}{rcl}\sigma, \tau & ::= & \dots \\ & | & \mathsf{bool}\end{array}$$

New typing rules

$$\frac{}{\Gamma \vdash \mathsf{T} : \mathsf{bool}} \qquad \frac{}{\Gamma \vdash \mathsf{F} : \mathsf{bool}}$$

$$\frac{\Gamma \vdash t : \mathsf{int}}{\Gamma \vdash \mathsf{isZero}(t) : \mathsf{bool}}$$

$$\frac{\Gamma \vdash t_1 : \mathsf{bool} \qquad \Gamma \vdash t_2 : \tau \qquad \Gamma \vdash t_3 : \tau}{\Gamma \vdash \mathsf{if}\ t_1\ \mathsf{then}\ t_2\ \mathsf{else}\ t_3 : \tau}$$

Extended types: products

New type

$$\begin{array}{rcl}\sigma, \tau & ::= & \dots \\ & | & \tau_1 \times \tau_2\end{array}$$

New typing rules

$$\frac{\Gamma \vdash t_1 : \tau_1 \qquad \Gamma \vdash t_2 : \tau_2}{\Gamma \vdash \langle t_1, t_2 \rangle : \tau_1 \times \tau_2}$$

$$\frac{\Gamma \vdash t : \tau_1 \times \tau_2}{\Gamma \vdash \pi_1(t) : \tau_1} \qquad \frac{\Gamma \vdash t : \tau_1 \times \tau_2}{\Gamma \vdash \pi_2(t) : \tau_2}$$

Extended types: recursion

New typing rule

$$\frac{\Gamma \vdash t : (\sigma \longrightarrow \tau) \longrightarrow (\sigma \longrightarrow \tau)}{\Gamma \vdash \mathsf{Fix}(t) : \sigma \longrightarrow \tau}$$

3 Type preservation

Type preservation: β-reduction

If $\quad \Gamma \vdash t : \tau \quad$ and $\quad t \longrightarrow_\beta t' \quad$ then

$$\Gamma \vdash t' : \tau$$

Proof by induction on $t \longrightarrow_\beta t'$.

- Case $t_1 t_2 \longrightarrow t_1' t_2$ with $t_1 \longrightarrow t_1'$

  By inversion of the hypothesis $\Gamma \vdash t_1 t_2 : \tau$ there is σ such that $\Gamma \vdash t_1 : \sigma \longrightarrow \tau$ and $\Gamma \vdash t_2 : \sigma$

  By induction hypothesis $\Gamma \vdash t_1' : \sigma \longrightarrow \tau$ and one can conclude with the typing rule for applications.

$$\frac{\Gamma \vdash t_1' : \sigma \longrightarrow \tau \qquad \Gamma \vdash t_2 : \sigma}{\Gamma \vdash t_1' t_2 : \tau}$$

- Case $t_1 t_2 \longrightarrow t_1 t_2'$ with $t_2 \longrightarrow t_2'$ similar
- Case $\lambda x.t_0 \longrightarrow \lambda x.t_0'$ with $t_0 \longrightarrow t_0'$ similar
- Case $(\lambda x.t_1)t_2 \longrightarrow t_1\{x \leftarrow t_2\}$

  By inversion of the hypothesis $\Gamma \vdash (\lambda x.t_1)t_2 : \tau$ there is σ such that $\Gamma \vdash \lambda x.t_1 : \sigma \longrightarrow \tau$ and $\Gamma \vdash t_2 : \sigma$

  and by inversion of $\Gamma \vdash \lambda x.t_1 : \sigma \longrightarrow \tau$ we get $\Gamma, x : \sigma \vdash t_1 : \tau$

  Last step required: combine $\Gamma, x : \sigma \vdash t_1 : \tau$ *and* $\Gamma \vdash t_2 : \sigma$ *to conclude something about the type of* $t_1\{x \leftarrow t_2\}$

  Lemma: substitution preserves types

  If $\Gamma, x : \sigma \vdash t : \tau$ and $\Gamma \vdash u : \sigma$ then $\Gamma \vdash t\{x \leftarrow u\} : \tau$

Substitution and types
If $\Gamma, x : \sigma \vdash t : \tau$ and $\Gamma \vdash u : \sigma$ then

$$\Gamma \vdash t\{x \leftarrow u\} : \tau$$

Proof by induction on the derivation of $\Gamma, x : \sigma \vdash t : \tau$

- Case where t is a variable
  - Case $\Gamma, x : \sigma \vdash x : \tau$ with $\sigma = \tau$
    Then $x\{x \leftarrow u\} = u$ and $\Gamma \vdash u : \sigma = \tau$
  - Case $\Gamma, x : \sigma \vdash y : \tau$ with $x \neq y$ and $\Gamma(y) = \tau$
    Then $y\{x \leftarrow u\} = y$ and $\Gamma \vdash y : \tau$
- Case $\Gamma, x : \sigma \vdash t_1\ t_2 : \tau$ with $\Gamma, x : \sigma \vdash t_1 : \sigma \longrightarrow \tau$ and $\Gamma, x : \sigma \vdash t_2 : \sigma$

  and induction hypotheses $\Gamma \vdash t_1\{x \leftarrow u\} : \sigma \longrightarrow \tau$ and $\Gamma \vdash t_2\{x \leftarrow u\} : \sigma$

  We deduce $\Gamma \vdash (t_1\{x \leftarrow u\})\ (t_2\{x \leftarrow u\}) : \tau$, which allows us to conclude since $(t_1\{x \leftarrow u\})\ (t_2\{x \leftarrow u\}) = (t_1\ t_2)\{x \leftarrow u\}$
- Case $\Gamma, x : \sigma \vdash \lambda y^{\tau'}.t : \tau' \longrightarrow \tau$ with $\Gamma, x : \sigma, y : \tau' \vdash t : \tau$

  and induction hypothesis $\Gamma, y : \tau' \vdash t\{x \leftarrow u\} : \tau$

  THen $\Gamma \vdash \lambda y^{\tau'}.(t\{x \leftarrow u\}) : \tau$

  By α-renaming we assume $y \neq x$ and $y \notin \mathrm{fv}(u)$, therefore $(\lambda y^{\tau'}.t)\{x \leftarrow u\} = \lambda y^{\tau'}.(t\{x \leftarrow u\})$, and we conclude with the former judgment

Reduction preserves types
Consequences

- If a term has a type, it will keep it along β-reduction
- If a term has a type and a normal form, the normal form has the same type

4 Type safety

Safety
Evaluation of a term should never see an inconsistent operation

- reduction never blocked before reaching a value

Simple statement: *if t is not a value, then there is t' such that $t \longrightarrow t'$*

Type safety

Progress lemma

$$\text{If} \vdash t : \tau \text{ and } t \text{ is not a value then there is } t' \text{ such that } t \longrightarrow t'$$

Using also the type preservation lemma we deduce $\vdash t' : \tau$, and we can go on

Safety theorem

If $\vdash t : \tau$, then

- either there is $t \longrightarrow t_1 \longrightarrow \ldots \longrightarrow t_n$ with t_n a value
- or there is an infinite reduction sequence $t \longrightarrow t_1 \longrightarrow t_2 \longrightarrow \ldots$

Progress lemma for λ-calculus + pairs (call by value)

The property

$$\text{If} \vdash t : \tau \text{ then either } t \text{ is a value or there is } t' \text{ with } t \longrightarrow_v t'$$

is proved by induction on the derivation of $\vdash t : \tau$

- Case $\Gamma \vdash x : \tau$ with $\Gamma(x) = \tau$

  Impossible since we consider only the empty environment

- Case $\vdash \lambda x.t_0 : \sigma \longrightarrow \tau$ with $x : \sigma \vdash t_0 : \tau$

  Then $t = \lambda x.t_0$ is a value

- Case $\vdash \langle t_1, t_2 \rangle : \tau_1 \times \tau_2$ with $\vdash t_1 : \tau_1$ and $\vdash t_1 : \tau_2$

  By induction hypothesis on $\vdash t_1 : \tau_1$ we have:

  - either there is t_1' with $t_1 \longrightarrow t_1'$ and then $\langle t_1, t_2 \rangle \longrightarrow_v \langle t_1', t_2 \rangle$
  - or t_1 is a value v_1

    Then by induction hypothesis on $\vdash t_2 : \tau_2$ we have:

    * either there is t_2' with $t_2 \longrightarrow t_2'$ and then $\langle v_1, t_2 \rangle \longrightarrow_v \langle v_1, t_2' \rangle$
    * or t_2 is a value v_2 and then $\langle v_1, v_2 \rangle$ is a value

- Case $\vdash t_1 t_2 : \tau$ with $\vdash t_1 : \sigma \longrightarrow \tau$ and $\vdash t_2 : \sigma$

  As in the previous case:

  - either there is t_1' with $t_1 \longrightarrow t_1'$, and then $t_1 t_2 \longrightarrow_v t_1' t_2$
  - or t_1 is a value v_1, and in this case

    * either there is t_2' with $t_2 \longrightarrow t_2'$, and then $v_1 t_2 \longrightarrow_v v_1 t_2'$
    * or t_2 is a value v_2 Then we want to prove that $v_1\ v_2$ reduces

      Classification lemma: if a is a value and $\Gamma \vdash t : \sigma \longrightarrow \tau$ then a has the shape $\lambda x.a'$

      By classification lemma, there are x, t_1' such that $v_1 = \lambda x.t_1'$ and therefore $(\lambda x.t_1')v_2 \longrightarrow_v t_1'\{x \leftarrow v_2\}$

- Case $\vdash \pi_1(t_0) : \tau_1$ with $\vdash t_0 : \tau_1 \times \tau_2$ By induction hypothesis we have:

  - either there is t_0' with $t_0 \longrightarrow t_0'$, and then $\pi_1(t_0) \longrightarrow_v \pi_1(t_0')$
  - or t_0 is a value v_0, and we want to prove that $\pi_1(v_0)$ reduces

    Classification lemma: if a is a value and $\Gamma \vdash a : \tau_1 \times \tau_2$ then a has the shape $\langle a_1, a_2 \rangle$

    By classification lemma there are v_1, v_2 such that $v_0 = \langle v_1, v_2 \rangle$ and therefore $\pi_1(\langle v_1, v_2 \rangle) \longrightarrow_v v_1$

- Case $\vdash \pi_2(t_0) : \tau_2$ with $\vdash t_0 : \tau_1 \times \tau_2$ is similar

5 Curry-Howard correspondence

Programs = proofs ***trailer***

types λ-calculus	Natural deduction
$\dfrac{\Gamma(x) = \tau}{\Gamma \vdash x : \tau}$	$\dfrac{\tau \in \Gamma}{\Gamma \vdash \tau}$
$\dfrac{\Gamma, x : \sigma \vdash e : \tau}{\Gamma \vdash \lambda x.e : \sigma \longrightarrow \tau}$	$\dfrac{\Gamma, \sigma \vdash \tau}{\Gamma \vdash \sigma \Longrightarrow \tau}$
$\dfrac{\Gamma \vdash e_1 : \sigma \longrightarrow \tau \qquad \Gamma \vdash e_2 : \sigma}{\Gamma \vdash e_1\ e_2 : \tau}$	$\dfrac{\Gamma \vdash \sigma \Longrightarrow \tau \qquad \Gamma \vdash \sigma}{\Gamma \vdash \tau}$
τ : type $\vdash$: typability	τ : formula $\vdash$: provability

Many proof assistants are built upon this correspondence

6 Normalization

Normalization

Does reduction actually make something smaller?

Theorm

If $\Gamma \vdash t : \tau$, then t is strongly normalizing.

Normalization theorem: a syntactic proof?

If $\Gamma \vdash t : \tau$, then t is strongly normalizing.

Proof attempt using structural induction on t

- Case of a variable: x is strongly normalizable
- Case of an abstraction: if t_0 is strongly normalizing, then so is $\lambda x.t_0$
- Case of an application: if t_1 and t_2 are both strongly normalizing, then...

$$t_1 t_2 \longrightarrow_\beta^* (\lambda x.t_1')t_2 \longrightarrow_\beta^* (\lambda x.t_1')t_2' \longrightarrow_\beta t_1'\{x \longleftarrow t_2'\} \longrightarrow_\beta ???$$

Problem: $t_1'\{x \longleftarrow t_2'\}$ is not a subterm of t, so we have no induction hypothesis available

Lemma

If t and u are well-typed and strongly normalizing, then $t\{x \longleftarrow u\}$ is strongly normalizing

Exercise: preservation of normalization by reduction

If t is strongly normalizing and $t \longrightarrow^* t'$ then t' is strongly normalizing

If t is normalizable and $t \longrightarrow^* t'$ then t' is normalizable

If s and t are strongly normalizing and not st then there are x, s' such that $s \longrightarrow^* \lambda x.s'$ and $s'\{x \longleftarrow t\}$ is not strongly normalizing

Application lemma

Lemma

If s, t and $\vec{u}$ are strongly normalizing but $st\vec{u}$ is not, then there are x, s' such that $s \longrightarrow^* \lambda x.s'$ and $s'\{x \leftarrow t\}\vec{u}$ is not strongly normalizing

Rephrasing using contraposition If

- s, t and $\vec{u}$ are strongly normalizing
- $s \longrightarrow^* \lambda x.s'$
- $s'\{x \leftarrow t\}\vec{u}$ is strongly normalizing

then $st\vec{u}$ is strongly normalizing

Well-founded order

Order relation $(E, \leq)$: binary relation $\leq$ on the set E that is:

- reflexive $\forall x \in E,\ x \leq x$
- antisymmetric $\forall x, y \in E,\ x \leq y \wedge y \leq x \Longrightarrow x = y$
- transitive $\forall x, y, z \in E,\ x \leq y \wedge y \leq z \Longrightarrow x \leq z$

Strict order

$$x < y \quad \Longleftrightarrow x \leq y \wedge x \neq y$$

Well-founded order: no infinite strictly decreasing chain

$$x_0 > x_1 > x_2 > \dots$$

Alternative characterization: every non-empty subset of E has a minimal element

Well-founded induction

Context: well-founded order $(E, \leq)$

For any predicate P on E

$$(\forall x \in E, (\forall y \in E, y < x \Longrightarrow P(y)) \Longrightarrow P(x)) \Longrightarrow \forall x \in E, P(x)$$

Goal: proving a property of the shape $\forall x \in E, P(x)$

Let $x \in E$

- assume $P(y)$ true for all $y < x$ (induction hypotheses)
- show that $P(x)$ holds

Question: where is the base case of this induction?

Lexicographic order

Lexicographic product of two orders $(A, \leq_A)$ and $(B, \leq_B)$: order on $A \times B$ defined by the condition

$$(a, b) \leq (a', b') \quad \Longleftrightarrow \quad a <_A a' \vee (a = a' \wedge b \leq_B b')$$

Property

the lexicographic product of two well-founded orders is a well-founded order

Consequence: induction on a lexicographic order is valid

Exercise: Ackermann function

The Ackermann function is described by the following equations

$$\begin{array}{rcl} \mathsf{ack}(0,n) & = & n+1 \\ \mathsf{ack}(m+1,0) & = & \mathsf{ack}(m,1) \\ \mathsf{ack}(m+1,n+1) & = & \mathsf{ack}(m,\mathsf{ack}(m+1,n)) \end{array}$$

Show that $\mathsf{ack}(m,n)$ is indeed defined for any $m,n \in \mathbb{N}$

Lemma: preservation of normalization by substitution

Lemma

If t and u are well-typed and strong normalizing then $t\{x \longleftarrow u\}$ is strongly normalizing

By induction on the lexicographic product

$$(\mathsf{tl}(\mathsf{ty}(u)), \mathsf{ht}(t), \mathsf{tl}(t))$$

where

- $\mathsf{ty}(u)$ is the type of u
- $\mathsf{tl}(a)$ is the size of a (numbers of nodes in the syntactic tree)
- $\mathsf{ht}(t)$ is the length of the longest reduction sequence starting from t

Proof

By case on the shape of t

- Case of a variable
  - Case $t = x$ then $x\{x \longleftarrow u\} = u$, strongly normalizing by hypothesis
  - Case $t = y$ with $y \neq x$ then $y\{x \longleftarrow u\} = y$, strongly normalizing
- Case of an abstraction: $t = \lambda x.t_0$
  Then $\mathsf{ht}(t_0) = \mathsf{ht}(t)$ and $\mathsf{tl}(t_0) < \mathsf{tl}(t)$ and then $(\mathsf{tl}(\mathsf{ty}(u)), \mathsf{ht}(t_0), \mathsf{tl}(t_0)) < (\mathsf{tl}(\mathsf{ty}(u)), \mathsf{ht}(t), \mathsf{tl}(t))$
  By induction hypothesis, $t_0\{x \longleftarrow u\}$ is strongly normalizing
  Thus $t\{x \longleftarrow u\} = \lambda x.(t_0\{x \longleftarrow u\})$ is strongly normalizing.
- Case of an application: $t = t_0 t_1 t_2 \dots t_n$ with t_0 not an application
  Case on t_0
  - Case $t_0 = y$ with $y \neq x$
    Each reductions of t is in one of the t_i with $i \geq 1$ thus $\mathsf{ht}(t_i) \leq \mathsf{ht}(t)$ for all $i \geq 1$, moreover $\mathsf{tl}(t_i) < \mathsf{tl}(t)$ for all $i \geq 1$.
    Thus by induction hypothesis $t_i\{x \longleftarrow u\}$ is strongly normalizing $i \geq 1$,
    and $y\ t_1\{x \longleftarrow u\} \dots t_n\{x \longleftarrow u\}$ is strongly normalizing as well
    Finally $t\{x \longleftarrow u\}$ is strongly normalizing
  - Case $t_0 = \lambda y.t_0'$
    Then $t \longrightarrow t' = t_0'\{y \longleftarrow t_1\}t_2 \dots t_n$ and $\mathsf{ht}(t') < \mathsf{ht}(t)$
    Then by induction hypothesis $t'\{x \longleftarrow u\}$ is strongly normalizing
    We have

    $$\begin{array}{ll} & t'\{x \longleftarrow u\} \\ = & (t_0'\{y \longleftarrow t_1\}t_2 \dots t_n)\{x \longleftarrow u\} \\ = & t_0'\{x \longleftarrow u\}\{y \longleftarrow t_1\{x \longleftarrow u\}\}\ t_2\{x \longleftarrow u\}\ \dots\ t_n\{x \longleftarrow u\} \end{array}$$

    By induction hypothesis $t_i\{x \longleftarrow u\}$ is strongly normalizing for any i
    Thus by application lemma $t\{x \longleftarrow u\}$ is strongly normalizing

- Case $t_0 = x$ We have to show that $u\ t_1\{x \leftarrow u\}\ \ldots\ t_n\{x \leftarrow u\}$ If $u \longrightarrow^* y$ then we conclude as above

  Otherwise $u \longrightarrow^* \lambda y.u_0$

  By induction hypothesis $t_i\{x \leftarrow u\}$ is strongly normalizing for any $i \geq 1$

  To apply the lemma, we have to show that $t' = u_0\{y \leftarrow t_1\{x \leftarrow u\}\}\ t_2\{x \leftarrow u\}\ \ldots\ t_n\{x \leftarrow u\}$ is strongly normalizing Show that $t' = u_0\{y \leftarrow t_1\{x \leftarrow u\}\}\ t_2\{x \leftarrow u\}\ \ldots\ t_n\{x \leftarrow u\}$ is strongly normalizing

  Trick: $t' = (z\ t_2\{x \leftarrow u\}\ \ldots\ t_n\{x \leftarrow u\})\{z \leftarrow u_0\{y \leftarrow t_1\{x \leftarrow u\}\}\}$ Then we can conclude by induction hypothesis by just checking that:

  * $z\ t_2\{x \leftarrow u\}\ \ldots\ t_n\{x \leftarrow u\}$ is strongly normalizing
  * $u_0\{y \leftarrow t_1\{x \leftarrow u\}\}$ is strongly normalizing
  * $\mathsf{ty}(u_0\{y \leftarrow t_1\{x \leftarrow u\}\}) < \mathsf{ty}(u)$
  * $z\ t_2\{x \leftarrow u\}\ \ldots\ t_n\{x \leftarrow u\}$ is strongly normalizing since the $t_i\{x \leftarrow u\}$ are strongly normalizing
  * We have $\mathsf{ty}(u) = \mathsf{ty}(\lambda x.u_0) = \sigma \longrightarrow \tau$ and $\mathsf{ty}(t_1\{x \leftarrow u\}) = \mathsf{ty}(t_1) = \sigma < \mathsf{ty}(u)$ Since u_0 and $t_1\{x \leftarrow u\}$ are strongly normalizing we deduce by induction hypothesis that $u_0\{y \leftarrow t_1\{x \leftarrow u\}\}$ is strongly normalizing
  * Moreover $\mathsf{ty}(u_0\{y \leftarrow t_1\{x \leftarrow u\}\}) = \mathsf{ty}(u_0) = \tau < \mathsf{ty}(u)$

  Then we can apply induction hypothesis to prove that t' is strongly normalizing, and the lemma to deduce that t is strongly normalizing.

7 Denotational semantics

Semantic domains (λ-calculus with simple types)

Denotational semantics

- associate to each λ-term t a mathematical object s

where the nature of s depends on the type of t

We associate to each type τ a set of mathematical values D^τ called the *semantic domain* of τ

$$\begin{array}{rcl} D^{\mathsf{bool}} & = & \mathbb{B} \\ D^{\mathsf{int}} & = & \mathbb{N} \\ D^{\sigma \to \tau} & = & (D^\sigma \longrightarrow D^\tau) \end{array}$$

where $A \longrightarrow B$ is the set of mathematical functions from A to B

Semantics of terms

Translation by induction on the structure of the term

$$\begin{array}{rcl} [\![x]\!]_\rho & = & \rho(x) \\ [\![\lambda x.t_0]\!]_\rho & = & a \longmapsto [\![t_0]\!]_{\rho[x \leftarrow a]} \\ [\![t_1\ t_2]\!]_\rho & = & [\![t_1]\!]_\rho\ [\![t_2]\!]_\rho \end{array}$$

where ρ is an *environment*: a function from λ-variables towards semantic values, with $\rho[x \leftarrow a]$ defined by

$$\begin{array}{rcll} \rho[x \leftarrow a](x) & = & a & \\ \rho[x \leftarrow a](y) & = & \rho(y) & \quad \text{si } y \neq x \end{array}$$

Note: here x is a λ-variable and a is a mathematical variable

Examples

$$
\begin{array}{rcl}
[\![\lambda x.x]\!]_\rho & = & a \mapsto [\![x]\!]_{\rho[x \leftarrow a]} \\
& = & a \mapsto (\rho[x \leftarrow a])(x) \\
& = & a \mapsto a \\
[\![\lambda x.\lambda y.x]\!]_\rho & = & a \mapsto [\![\lambda y.x]\!]_{\rho[x \leftarrow a]} \\
& = & a \mapsto (b \mapsto [\![x]\!]_{\rho[x \leftarrow a][y \leftarrow b]}) \\
& = & a \mapsto (b \mapsto (\rho[x \leftarrow a][y \leftarrow b])(x)) \\
& = & a \mapsto (b \mapsto a)
\end{array}
$$

Reduction preserves the semantics

Theorem

$$\text{If } t \longrightarrow t', \text{ then } [\![t]\!]_\rho = [\![t']\!]_\rho \text{ for all } \rho$$

the other direction is subtle

Proof: by induction on $t \longrightarrow t'$

Extended denotational semantics

Most of the extensions can be added directly

$$
\begin{array}{rcl}
[\![\mathsf{T}]\!]_\rho & = & \mathsf{vrai} \\
[\![\mathsf{F}]\!]_\rho & = & \mathsf{faux} \\
[\![n]\!]_\rho & = & n \\
[\![t_1 \oplus t_2]\!]_\rho & = & [\![t_1]\!]_\rho + [\![t_2]\!]_\rho \\
[\![\mathsf{isZero}(t)]\!]_\rho & = & \begin{cases} \mathsf{vrai} & \text{si } [\![t]\!]_\rho = 0 \\ \mathsf{faux} & \text{si } [\![t]\!]_\rho \neq 0 \end{cases} \\
& \dots &
\end{array}
$$

What about fixpoints?

Scott domains *trailer*

Extended semantic domaines

- with partially defined values
- with an order on the information level of a partially defined value *min: undefined, max: fully defined*
- completes: any increasing sequence has a limit

Any function can be completed

Interesting functions: *monotone* and *continuous*

- more information on the argument gives more information on the result
- image of a limit = limit of the images

Then we find a semantical fixpoint to interpret any term *using Knaster-Tarski theorem*