Lambda Calculus and Types

Programming Languages CS442

David Toman

School of Computer Science
University of Waterloo

Syntax

What happened to the typing rules?

Idea

We use Typing Rules for λ-abstraction/application

$\Rightarrow$ the same we used for parameters.

- $\lambda x.T$ (abstraction) and $(T_1\ T_2)$ (application):

$$\frac{\pi \uplus \{x : \theta\} \vdash T : \theta'}{\pi \vdash \lambda x.T : \theta \to \theta'} \qquad \frac{\pi \vdash T_1 : \theta \to \theta' \qquad \pi \vdash T_2 : \theta}{\pi \vdash (T_1\ T_2) : \theta'}$$

- $\langle T_1, T_2 \rangle$ (pair) and $(\mathbf{pr}_i\ T)$ (projection):

$$\frac{\pi \vdash T_1 : \theta_1 \qquad \pi \vdash T_2 : \theta_2}{\pi \vdash \langle T_1, T_2 \rangle : \theta_1 \times \theta_2} \qquad \frac{\pi \vdash T : \theta_1 \times \theta_2}{\pi \vdash (\mathbf{pr}_i\ T) : \theta_i}$$

Simply Typed Lambda Calculus

- Types:

$$\tau ::= \iota \mid \tau \to \tau \mid \tau \times \tau$$

- Terms

$$t ::= x \mid \lambda x : \tau . t \mid (t\ t') \mid \langle t, t' \rangle \mid \mathbf{pr}_1\, t \mid \mathbf{pr}_2\, t$$

- Reduction:

$$(\lambda x : \tau . t) t' \to [t'/x]t \qquad \mathbf{pr}_i\ \langle t_1, t_2 \rangle \to t_i$$

. . . all constrained by *typing rules!!*

Properties

- we inherit properties from the untyped λ:
  1. Confluence/Church-Rosser
  2. Uniqueness of normal forms
- and there are couple extras:

Theorem (Subject-Reduction)

If $\pi \vdash t : \tau$ and $t \to^* t'$ then $\pi \vdash t' : \tau$.

Theorem (Strong Normalization)

If $\pi \vdash t : \tau$ then every reduction sequence $t \to^* \ldots$ is finite.

$\Rightarrow$ we can't have Ω or Y!

Denotational Semantics

Usual "lazy evaluation" semantic rules:

$$\begin{aligned}
[\![\pi \vdash (\lambda x:\tau.t):\tau \to \theta]\!] &= \lambda e.\lambda u.[\![\pi \uplus \{x:\tau\} \vdash t:\theta]\!](e \uplus \{x = u\}) \\
[\![\pi \vdash (t_1\ t_2):\theta]\!] &= \lambda e.([\![\pi \vdash t_1:\tau \to \theta]\!]\ e)([\![\pi \vdash t_2:\tau]\!]\ e) \\
[\![x:\tau]\!] &= \lambda e.lookup(x, e)
\end{aligned}$$

why don't we consider "eager" evaluation?

Theorem (Soundness)

$[\![\pi \vdash t:\tau]\!]e \in [\![\tau]\!]$ *for all* $e \in [\![\pi]\!]$.

Theorem (Soundness of β-rule a.k.a. Substitution lemma)

$[\![\pi \vdash [t'/x]t:\tau]\!]e = [\![\pi \uplus \{I:\theta \vdash t:\tau]\!](e \uplus \{x = [\![\pi \vdash t':theta]\!]r\})$
for all $e \in [\![\pi]\!]$.

Constants and Operators

How do we *regain* Turing-completeness for the typed λ-calculus?

(or even weaker forms of iteration)?

Idea

We add new operators and reduction rules when needed.

- Booleans:

$$\mathit{true} : \mathit{Bool} \qquad \mathit{false} : \mathit{Bool}$$

$$\frac{\pi \vdash t : \mathit{Bool} \qquad \pi \vdash u : \tau \qquad \pi \vdash v : \tau}{\pi \vdash \textbf{if}\ t\ \textbf{then}\ u\ \textbf{else}\ v\ \textbf{fi} : \tau}$$

$$\textbf{if}\ \mathit{true}\ \textbf{then}\ u\ \textbf{else}\ v\ \textbf{fi} \longrightarrow u \qquad \textbf{if}\ \mathit{false}\ \textbf{then}\ u\ \textbf{else}\ v\ \textbf{fi} \longrightarrow v$$

Constants and Operators (cont.)

- Natural Numbers:

$$0 : Nat \qquad \frac{\pi \vdash n : Nat}{\pi \vdash n + 1 : Nat} \qquad \frac{\pi \vdash n : Nat \quad \pi \vdash u : \tau \quad \pi \vdash v : \tau \to (Nat \to \tau)}{\pi \vdash \mathbf{loop}\, n\, \mathbf{ifzero}\, u\, \mathbf{else}\, v : \tau}$$

$$\mathbf{loop}\, 0\, \mathbf{ifzero}\, u\, \mathbf{else}\, v \to u$$
$$\mathbf{loop}\, n + 1\, \mathbf{ifzero}\, u\, \mathbf{else}\, v \to v\ (\mathbf{loop}\, n\, \mathbf{ifzero}\, u\, \mathbf{else}\, v)n$$

$\Rightarrow$ this system is still *strongly normalizing* [Gödel's System T]

- General Recursion (the Y-substitute):

$$\frac{\pi \vdash t : \tau \to \tau}{\pi \vdash \mathbf{fix}\, t : \tau} \qquad \mathbf{fix}\, t \to t\ (\mathbf{fix}\, t)$$

$\Rightarrow$ finally a *Turing complete* system

Summary

- Simply-typed λ-calculus
  1. "function application" restricted by typing rules
  2. strong normalization
- Need for add-ons:
  1. basic data types ($Bool, Nat, \ldots$)
  2. fixpoint/recursion operators (to allow loops)

  $\Rightarrow$ basis of *typed* functional languages (PCF, ML, ...)
- Use of Typed λ-calculus for (operational) Semantics

  $\Rightarrow$ *denotational equations* can be used as *computational rules*
- Questions:
  1. additional "built-in" types and/vs. strong normalization?
  2. strong normalization and abstraction/parametrization principles?
  3. fixpoint operators—does *reduction strategy* matter?