Lecture 4: Typed Lambda Calculus

CS 181
Spring 2016
Kim Bruce

Some slide content taken from Unger and Michaelis

Office hours today: 11 - 12:30

Computation Rules

- Reduction rules for lambda calculus:

  (α) λx. M $\rightarrow_\alpha$ λy. ([y/x] M), if y $\notin$ FV(M).

  change name of parameters if new not capture old

  (β) (λx. M) N $\rightarrow_\beta$ [N/x] M.

  computation by substituting function argument for formal parameter

  (η) λx. (M x) $\rightarrow_\eta$ M.

  Optional rule to get rid of excess λ's

Keeping Out of Trouble!

- Use variable convention: In a term M, ensure:
  - all bound variables are distinct from all free ones, and
  - all lambdas bind different variables
  - E.g. if have (λx.(y (λy.(x (y (λy.(x y))))))), rewrite as: (λu.(y (λv.(u (v (λw.(u w))))))) using α-equivalence before doing any reductions.

Normal Forms

- A term M is in normal form if no reduction rules apply, even after applications of α.
- Not all terms have normal forms
  - $\Omega = (\lambda x.\ (x\ x))(\lambda x.\ (x\ x))$

How to evaluate

- Many strategies:
  - $(\lambda x.\ x + 32)(\underline{(\lambda y.\ y * 3)\ 5}) \Rightarrow \underline{(\lambda x.\ x + 32)\ 15} \Rightarrow 47$ *Inside-out*
  - versus
  - $\underline{(\lambda x.\ x + 32)((\lambda y.\ y * 3)\ 5)} \Rightarrow (\underline{(\lambda y.\ y * 3)\ 5}) + 32 \Rightarrow 47$ *Outside-in*
- Confluence: If M can be reduced to a normal form, then there is only one such normal form.
- However, not all strategies give a normal form:
  - $(\lambda x.\ 47)\ \Omega$

Types

- Types are a way of classifying expressions according to their use.
- Typically indicate operations available.
- Start with one or more base types & build more complex types

Type Expressions

- $\tau ::= b \mid (\tau \rightarrow \tau)$
  - *Specified as context free grammar!!*
  - where b represents one or more basic types (e.g., Integer, String, Boolean, etc.)
  - and $\tau' \rightarrow \tau$ represents functions with domain τ' & range τ.

Type*d* Expressions

- Idea: Every identifier introduced has an associated type. Two options:
  1. Every type has its own (potentially infinite) set of identifiers with that type.
  2. There is a (potentially infinite) collection of shared variables. When introduced, they are provided with a type annotation.
- Traditionally (& in the text) use the first.
- I use the second, as it is more flexible with richer languages.

Typed Lambda Calculus

- Terms of typed lambda calculus
  - $M := v \mid (M\ M) \mid \lambda v \mapsto \tau.\ M$
  - Also add primitive terms/operations on types
- Examples:
  - $\lambda v \mapsto$ Integer. $v + 2$ has type Integer $\to$ Integer
  - $\lambda x \mapsto$ Integer. $\lambda y \mapsto$ Integer. $\lambda z \mapsto$ Integer. $x * y + z$ has type Integer $\to$ Integer $\to$ Integer $\to$ Integer
  - $\lambda f \mapsto$ Integer $\to$ Integer. $\lambda x \mapsto$ Integer. $f(f(x))$ has type (Integer $\to$ Integer) $\to$ Integer $\to$ Integer

Typ*ing* Expressions

- Need to record types of variables in a symbol table, written E, which is a set of pairs associating variables with their types.
  - E.g., $E = \{x \mapsto \text{Integer}, y \mapsto \text{Integer} \to \text{Integer}, z \mapsto \text{Bool}\}$
  - *No duplicate entries for variables.*
- As long as E records types of all free variables in a term, then can determine the type of the term or determine that it has no type.

Typing Rules

- if $x \mapsto \tau$ is in E,
  then $E \vdash x: \tau$
- if $E \vdash M: \tau \to \tau'$ and $E \vdash M': \tau$,
  then $E \vdash (M\ M'): \tau'$
- if $E \cup \{x \mapsto \tau\} \vdash M: \tau'$,
  then $E \vdash \lambda x \mapsto \tau.\ M: \tau \to \tau'$
- Say M is well-typed with respect to E if can derive $E \vdash M: \tau$ for some τ

What are the types?

- cond = $\lambda b \mapsto$ Boolean. $\lambda t \mapsto$ e. $\lambda f \mapsto$ e.
  if b then t else f
- (cond true)
- csum = $\lambda m \mapsto$ Integer. $\lambda n \mapsto$ Integer. sum (m,n)
- (csum 7)
- (csum 7 2)

Questions?