Course 2D1453, 2006-07

Advanced Formal Methods

Lecture 3: Simply Typed Lambda calculus

Mads Dam
KTH/CSC

Some material from B. Pierce: TAPL + some from G. Klein, NICTA

Typing λ-terms

The uptyped λ-calculus allows "strange" terms to be formed:

- D D $=_\beta$ not(D D)
- succ (pair ff ff)

Solution: Rule out ill-formed terms using types (Church 1940)

Ill-formed term: Computation can "go wrong"

- succ (pair ff ff): Cannot complete computation to produce a sensible value = normal form
- Type unsafety – runtime error

Types

Simply typed λ-calculus, $\lambda_\rightarrow$:
Only two types, base types and function types

Syntax:
  T ::= A | T $\rightarrow$ T

- A: Base type, e.g. bool, int, float, array[int], ...
- $T_1 \rightarrow T_2$:
  Type of functions from T_1 to T_2

Type constructor $\rightarrow$ right-associative:
  $T_1 \rightarrow T_2 \rightarrow T_3 == T_1 \rightarrow (T_2 \rightarrow T_3)$

Typed λ-terms

$\lambda x.t$: Must be of function type $T_1 \rightarrow T_2$
But where to find T_1?

Alt. 1: Give domain type explicitly as typed λ-term $\lambda x{:}T_1.t$
  Example: $\lambda x{:}int.x + x : int \rightarrow int$
  Used here initially

Alt. 2: Keep untyped syntax
  Use types as well-formedness predicate
  $\lambda x.x + x : int \rightarrow int$
  $\lambda x.x: int \rightarrow int$, but also $\lambda x.x: bool \rightarrow bool$, etc.

The Typing Relation

Typing relation

$$\Gamma \vdash t : T$$

Γ: Type environment
Also: Type context, type assumptions
Finite function $x \mapsto T_x$
Must have $FV(t) \subseteq dom(\Gamma)$

$\boxed{\Gamma \text{ omitted if empty}}$ $\boxed{\text{Function update: } x \notin dom(\Gamma)}$

Typing rules:

$$\frac{x : T \in \Gamma}{\Gamma \vdash x : T} \qquad \frac{\Gamma, x : T_1 \vdash t : T_2}{\Gamma \vdash \lambda x : T_1.\ t : T_1 \rightarrow T_2}$$

$$\frac{\Gamma \vdash t : T_1 \rightarrow T_2 \quad \Gamma \vdash s : T_1}{\Gamma \vdash t\ s : T_2}$$

Base Types

Easy to extend to base types

Example: Booleans

Base type Bool

Terms t ::= x | λx : T . t | t t | true | false | if t then t else t | ...

New typing rules (+ one for false too):

$$\frac{-}{\Gamma \vdash true : bool} \qquad \frac{\Gamma \vdash t : bool \quad \Gamma \vdash s_1 : T \quad \Gamma \vdash s_2 : T}{\Gamma \vdash \text{if } t \text{ then } s_1 \text{ else } s_2 : T}$$

Terms, Notation, Reduction

Same syntactic conventions for typed terms:

- λ x : T_1 y : T_2 . T == λ x: T_1 . λ y : T_2 . T
  Sometimes use , as separator for clarity
- Similar for associativity

Alpha-conversion, substitution, free and bound variables

Reduction:

$$\frac{-}{(\lambda x : T\ .\ t)\ s \to_\beta t[s/x]} \qquad \frac{s \to_\beta s'}{s\ t \to_\beta s'\ t}$$

$$\frac{t \to_\beta t'}{s\ t \to_\beta s\ t'} \qquad \frac{t \to_\beta t'}{\lambda x : T\ .\ t \to_\beta \lambda x : T\ .\ t'}$$

Typing, Examples

Exercise 1: Give type derivations to show:

1. $\vdash \lambda x : A,\ y : B\ .\ x : A \to B \to A$
2. $\vdash \lambda x : A \to B,\ y : B \to C,\ z : A\ .\ y\ (x\ z) :$
   $(A \to B) \to (B \to C) \to A \to C$

Exercise 2: Find a context under which f x y has type A.
Can you give a simple description of *all* such contexts?

Properties of the Typing Relation

Lemma 1:

1. If $\Gamma \vdash x : T$ then $x : T \in \Gamma$
2. If $\Gamma \vdash \lambda\ x : T_1.\ t : S$ then $S = T_1 \to T_2$ for some S such that $\Gamma, x : T_1 \vdash t : T_2$
3. If $\Gamma \vdash t\ s : T_2$ then there is some T_1 such that $\Gamma \vdash t : T_1 \to T_2$ and $\Gamma \vdash s : T_1$

Exercise 3: Prove this statement

Exercise 4: Is there any context Γ and type T such that $\Gamma \vdash x\ x : T$? If so, give a type derivation. If not, prove it.

Unique Typing and Normal Forms

Lemma 2: If $\Gamma \vdash t : T_1$ and $\Gamma \vdash t : T_2$ then $T_1 = T_2$

Exercise 5: Prove this statement.

Unique typing fails for many richer languages

Values:

$v \in Val ::= x \mid x\ v\ ...\ v \mid \lambda x : T\ .\ v$

Lemma 3: $t \not\to_\beta$ iff $t \in Val$

Exercise 6: Prove (or disprove) this statement.

Substitution Lemma

$\Gamma \le \Delta$: For all x, $\Gamma(x)$ is defined implied $\Delta(x)$ is defined and then $\Gamma(x) = \Delta(x)$

Proposition 1: If $\Gamma \vdash t : T$ and $\Gamma \le \Delta$ then $\Delta \vdash t : T$

Lemma 4 [Substitution]: If $\Gamma, x : S \vdash t : T$ and $\Gamma \vdash s : S$ then $\Gamma \vdash t[s/x] : T$

We'll prove this statement in class.

Theorem 1 [Subject Reduction]: If $\Gamma \vdash t : T$ and $t \to_\beta t'$ then $\Gamma \vdash t' : T$

Exercise 7: Prove this statement (hint: Use induction on the derivation of $\Gamma \vdash t : T$)

Extensions - Products

Many extensions possible, see TAPL for more

First: Product types

Types: $T ::= ... \mid T \times T$

Terms: $t ::= ... \mid (t, t) \mid fst \mid snd$

Reduction: Use generic $\to$ instead of $\to_\beta$

Can support different evaluation orders

Products – Reduction and Typing

Reduction rules: $\dfrac{-}{\text{fst}(t,s) \to t}$ $\dfrac{-}{\text{snd}(t,s) \to s}$

+ rules for context closure:

$\dfrac{t \to t'}{(t,s) \to (t',s)}$ $\dfrac{s \to s'}{(t,s) \to (t,s')}$ $\dfrac{t \to t'}{\text{fst } t \to \text{fst } t'}$ $\dfrac{t \to t'}{\text{snd } t \to \text{snd } t'}$

Typing rules:

$$\frac{\Gamma \vdash t : T \quad \Gamma \vdash s : S}{\Gamma \vdash (t, s) : T \times S}$$

$$\frac{-}{\Gamma \vdash \text{fst} : T \times S \to T} \qquad \frac{-}{\Gamma \vdash \text{snd} : T \times S \to S}$$

Sums

Types: T ::= ... | T + T

Terms: t ::= ... | in_1 | in_2 | cases in_1 => t || in_2 => t

Syntax slightly uncommon. Often use sugared version, something like:

case t of $in_1(x : T_1)$ => s_1 || $in_2(y : T_2)$ => s_2
== (cases in_1 => $\lambda x : T_1 . s_1$ ||$\lambda y{:}T_2 . s_2$) t

Sums – Reduction and Typing

Reduction rules:

$$\frac{-}{(\text{cases } in_1 \Rightarrow s_1 \parallel in_2 \Rightarrow s_2)\ (in_1\ t) \to s_1\ t}$$

$$\frac{-}{(\text{cases } in_1 \Rightarrow s_1 \parallel in_2 \Rightarrow s_2)\ (in_2\ t) \to s_2\ t}$$

Exercise: Give suitable context closure rules for sums

Typing:

$$\frac{-}{\Gamma \vdash in_1 : T \to T + S} \qquad \frac{-}{\Gamma \vdash in_2 : S \to T + S}$$

$$\frac{\Gamma \vdash s_1 : T_1 \to S \qquad \Gamma \vdash s_2 : T_2 \to S}{\Gamma \vdash \text{cases in1} \Rightarrow \text{s1} \parallel \text{in2} \Rightarrow \text{s2} : T_1 + T_2 \to S}$$

Exercise 8: Unique typing fails for the type system with sums. Why?

General Recursion

fix is not definable in $\lambda_\to$ (see later), but can be introduced as new constant

Terms: t ::= ... | fix

Reduction: fix f $\to$ f (fix f)

Typing: $\dfrac{-}{\Gamma \vdash \text{fix} : (T \to T) \to T}$

Exercise 9: Add a natural number base type, and define equal, plus, times, and factorial using fix

More Exercises

Exercise 10: Add the following constructs to simply typed lambda calculus, with reduction and typing rules:

t ::= ... | let x : T = t_1 in t_2 | letrec x : T = t_1 in t_2

The intention (of course) is that ”let” is used for non-recursive definitions, and ”letrec” for recursive ones. Give reduction and typing rules for ”let” and ”letrec”. Show how ”let” and ”letrec” can be coded in $\lambda_\to$. Do the same for mutually recursive definitions:

t ::= … | letrec $x_1 : T_1 = t_1$, … , $x_n : T_n = t_n$ in t

Note: In more realistic languages one will generally want type annotations T, T_1,… to be inferred automatically by the type checker

The ML Language

With the extensions above $\lambda_\to$ is a ”grandmother” of many typed functional languages

ML:

- Highly influential programming language
- Originally developed as a MetaLanguage for the LCF theorem prover [Gordon-Milner-Wadsworth-79]
- ML used for programming proof search in LCF

  Introduce base type ”theorem”

  The metalanguage must ensure type safety:

  The only values of type ”theorem” are those that really are theorems in the logic being represented
- ML main features: cbv semantics, automatic type inference, polymorphic types

ML, Haskell, PCF

ML and other languages:

- ML was influenced by Landin's ISWIM
- SML – Standard ML of 1997

  Comprehensive formal transition semantics and type system by [Milner-Tofte-Harper, 1990]
- Check out: SML of New Jersey, OCAML
- SML used in descendants of LCF: HOL, Isabelle
- Haskell is a descendant with cbn (lazy) semantics (and other twists)
- PCF [Plotkin-77]

  $\lambda_{\rightarrow}$ + naturals + more types + recursion

  Popular in theoretical studies

Strong Normalization

We are now addressing the base calculus $\lambda_{\rightarrow}$ with a single base type A

Strong normalization:

$t \in SN_n$ iff any $\rightarrow_\beta$-derivation $t = t_0 \rightarrow_\beta t_1 \rightarrow_\beta \cdots \rightarrow_\beta t_n \rightarrow_\beta \cdots$ has length at most n

$SN = \{t \mid \exists n.\ t \in SN_n\}$

Theorem 2 [Strong Normalization]: If $\vdash t : T$ then $t \in SN$

This immediately shows that all terms of functional type must express total functions on closed terms

Thus, general recursion cannot be encoded in $\lambda_{\rightarrow}$

Logical Relations

Exercise 11: Why is normalization tricky to prove?

As always, the trick is to find the right inductive argument

Proof here follows Tait [JSL-67] and Girard-Lafont-Taylor, Proofs and Types, CUP'89

Define predicate R_T on closed terms by:

- $R_A = \{t \mid t \in SN\}$
- $R_{S \rightarrow T} = \{t \mid \text{whenever } s \in R_S \text{ then } t\ s \in R_T\}$

Note: Do not require $t \in R_T$ implies $\vdash t : T$.

Proof of Normalization

Lemma 6: If $t \rightarrow_\beta t'$ and $t \in R_T$ then $t' \in R_T$

Proof: By structural induction on the structure of T

Exercise 12: Prove lemma 6.

Neutral term: Either a variable or an application

Lemma 7:

1. If $t \in R_T$ then $t \in SN$
2. If t is neutral and for all t', $t \rightarrow_\beta t'$ implies $t' \in R_T$, then $t \in R_T$

Proof of Lemma 7

Proof by simultaneous induction on T

T = A. Both 1 and 2 are immediate

$T = T_1 \rightarrow T_2$.

1: Let $t \in R_T$. By the induction hypothesis (2), $x \in R_{T_1}$, so $t\ x \in R_{T_2}$. Then $t\ x \in SN$, so $t \in SN$ as well.

2: Suppose t is neutral and whenever $t \rightarrow_\beta t'$ then $t' \in R_T$. Let $t_1 \in R_{T_1}$. We show $t\ t_1 \in R_{T_2}$. By the induction hypothesis (1), $t_1 \in SN_n$ for some n. We proceed by nested induction on n. It is sufficient to show $t_2 \in R_{T_2}$ whenever $t\ t_1 \rightarrow_\beta t_2$, by the induction hypothesis (2), and since $t\ t_1$ is neutral. Since t is neutral, either $t \rightarrow_\beta t'$ and $t_2 = t'\ t_1$, or else $t_1 \rightarrow_\beta t_1'$, and $t_2 = t\ t_1'$. In the first case, $t_2 \in R_{T_2}$ by the assumptions, and in the second, $t_1' \in R_{T_1}$. Then $t_1' \in SN_{n'}$, $n' < n$. So by the inner i.h. $T\ t_1' \in R_{T_2}$.

Abstraction Lemma

Lemma 8: If $t_1[t/x] \in R_{T_2}$ whenever $t \in R_{T_1}$ then $\lambda x : T_1.\ t_1 \in R_{T_1 \rightarrow T_2}$

Proof: Assume $t \in R_{T_1}$. We must show

$$t' = (\lambda x : T_1.\ t_1)\ t \in R_{T\text{-}1}\ .$$

By 7.1, $t \in SN_{n_2}$ and $t_1 \in SN_{n_1}$ for some n_1, n_2. Then n_1+n_2 is an upper bound on the number of reduction steps that can be performed before the outermost redex in t' must be reduced, so we proceed by induction on n_1+n_2. By 7.2 it is sufficient to show $t'' \in R_{T_2}$ whenever $t' \rightarrow_\beta t''$. Check out the possible cases:

- $t'' = t_1[t/x]$. We are done by the assumptions.
- $t'' = (\lambda x : T_1.\ t_1)\ s$ and $t \rightarrow_\beta s$. Then $s \in R_{T_1}$ by Lemma 6 and $s \in SN_{n_2'}$, $n_2' < n_2$ so we're done by the induction hypothesis.
- $t'' = (\lambda\ x : T_1.\ t_1')\ t$ and $t_1 \rightarrow_\beta t_1'$. By lemma 6, $t_1'[t/x] \in R_{T_2}$, and $t_1' \in SN_{n_1'}$ for some $n_1' < n_1$. So $t'' \in R_{T\text{-}2}$.

Fundamental Lemma

Lemma 9: Suppose $x_1 : T_1,\dots,x_n : T_n \vdash t : T$. If $t_i \in R_{T_i}$ for all i: $1 \leq i \leq n$, then $t[t_1/x_1,\dots,t_n/x_n] \in R_T$.

Note: This proves theorem 2, for n = 0.

Proof: By induction on size of the type derivation. Let $\Gamma = x_1 : T_1,\dots,x_n : T_n$ and $\underline{t}/\underline{x}$ abbreviate $t_1/x_1,\dots,t_n/x_n$.

- $t = x_i$: Then $t[\underline{t}/\underline{x}] = t_i$, $T = T_i$, and $t_i \in R_{T_i}$ by the assumptions.
- $t = t'\ t''$: By the induction hypothesis, $t'[\underline{t}/\underline{x}] \in R_{T' \to T}$ and $t''[\underline{t}/\underline{x}] \in R_T'$. Then $t[\underline{t}/\underline{x}] = (t'\ t'')[\underline{t}/\underline{x}] = (t'[\underline{t}/\underline{x}])\ (t''[\underline{t}/\underline{x}]) \in R_T$.
- $t = \lambda x'' : T''.\ t'$. Then $T = T'' \to T'$. Let $t'' \in R_{T''}$ be arbitrary. By the induction hypothesis, $t'[\underline{t}/\underline{x}, t''/x''] \in R_{T'}$. But then $\lambda x : T''.\ t'[\underline{t}/\underline{x}] = t[\underline{t}/\underline{x}] \in R_T$ as desired.

Exercise

Exercise 13: We did not require that $t \in R_T$ only if $\vdash t : T$. Why was that?