Lambda Calculus

Lecture (3): Simply Typed Lambda Calculus

Sriram Rajamani

Microsoft Research

Last class: Syntax & semantics of

Syntax:

$$e ::= x \quad \leftarrow \text{variable}$$

terms

$$\mid \lambda x.\, e \quad \leftarrow \text{function abstraction}$$

$$\mid e_1\ e_2 \quad \leftarrow \text{function application}$$

Operational semantics

$$(\lambda x.\ e_1)\ e_2 \longrightarrow_{\beta} \underbrace{[x \mapsto e_2]\ e_1}_{\text{term obtained by replacing all free occ. of } x \text{ in } e_1 \text{ by } e_2}$$

Examples of bad λ-calculus programs

$x \ (\lambda x. \ x)$ ← We say that this term is "wrong"

Not a value!

Cannot reduce any further

because x is not a function

Q: Can we find out at "compile time" if a λ-calculus program is bad?

Today ...

Program verification for λ-calculus

- Add types to λ-calculus terms
- Set up a type system that ensures
  "well-typed terms cannot go wrong"

Reference: Ben Pierce's book
"Types and Programming Languages"

First ....

We will consider a simpler language

Languge of arithmetic expressions

Terms:

t ::= true

false

if t then t else t

0

pred t

succ t

iszero t

values:

v : true

false

NV

NV : 0

succ NV

Operational Semantics

$\text{if true then } t_1 \text{ else } t_2 \longrightarrow t_1$ (E-IFTRUE)

$\text{if false then } t_1 \text{ else } t_2 \longrightarrow t_2$ (E-IFFALSE)

$$\frac{t_1 \to t_1'}{\text{if } t_1 \text{ then } t_2 \text{ else } t_3 \to \text{if } t_1' \text{ then } t_2 \text{ else } t_3} \quad \text{(E-IF)}$$

$$\frac{t_1 \to t_1'}{\text{succ } t_1 \to \text{succ } t_1'} \quad \text{(E-SUCC)}$$

$$\frac{t_1 \to t_1'}{\text{pred } t_1 \to \text{pred } t_1'} \quad \text{(E-PRED)}$$

$\text{pred } 0 \to 0$ (E-PREDZ)

$\text{pred } (\text{succ } nv_1) \to nv_1$ (E-PREDSUCC)

Operational semantics continued ....

$$\text{iszero } 0 \longrightarrow \text{true} \quad \text{(E-ISZEROZERO)}$$

$$\text{iszero}(\text{succ } nv_1) \longrightarrow \text{false} \quad \text{(E-ISZEROSUCC)}$$

$$\frac{t_1 \longrightarrow t_1'}{\text{iszero } t_1 \longrightarrow \text{iszero } t_1'} \quad \text{(E-ISZERO)}$$

Example 1

$$\text{iszero}\ (\ \text{if}\ \ (\text{iszero}\ (\text{succ}\ \ 0))\ \ \text{then}\ \ (\text{succ}\ \ 0)\ \ \text{else}\ \ 0\ )$$

Example 1

iszero (if (iszero (succ 0)) then (succ 0) else 0)

$\rightarrow$ iszero (if false then (succ 0) else 0)

$\rightarrow$ iszero (0)

$\rightarrow$ true.

Examples of bad programs (i.e. they get "stuck")

① succ true

② iszero false

③ if (succ zero) then true else false

④ if (if true then (succ zero) else false)
then true else false

Goal: Come up with a "static" verification technique to rule out "bad" programs

Typed arithmetic expressions

Types

$T ::= \text{Bool}$
$\quad\quad | \ \text{Nat}$

Typing Relation

$t : T$

read "Term t has type T"

Typing rules

$$\text{true} : \text{Bool} \qquad \text{[T-TRUE]}$$

$$\text{false} : \text{Bool} \qquad \text{[T-FALSE]}$$

$$\frac{t_1 : \text{Bool} \quad t_2 : T \quad t_3 : T}{\text{if } t_1 \text{ then } t_2 \text{ else } t_3 : T} \qquad \text{[T-IF]}$$

$$0 : \text{Nat} \qquad \text{[T-ZERO]}$$

$$\frac{t_1 : \text{Nat}}{\text{succ } t_1 : \text{Nat}} \qquad \text{[T-SUCC]}$$

$$\frac{t_1 : \text{Nat}}{\text{pred } t_1 : \text{Nat}} \qquad \text{[T-PRED]}$$

$$\frac{t_1 : \text{Nat}}{\text{iszero } t_1 : \text{Bool}} \qquad \text{[T-ISZERO]}$$

A typing derivation is a tree of instances of typing rules

$$\frac{\qquad}{\text{if iszero } 0 \text{ then } 0 \text{ else pred } 0 : \text{Nat}}$$

A typing *derivation* is a *tree* of instances of typing rules

$$\frac{\text{iszero } 0 : \text{Bool} \qquad 0 : \text{Nat} \qquad \text{pred } 0 : \text{Nat}}{\text{if iszero } 0 \text{ then } 0 \text{ else pred } 0 : \text{Nat}} \; [\text{T-IF}]$$

A typing derivation is a tree of instances of typing rules

$$\dfrac{\dfrac{0 : \mathrm{Nat}}{\mathrm{iszero}\ 0 : \mathrm{Bool}}\,[\text{T-ISZERO}] \qquad \dfrac{}{0 : \mathrm{Nat}}\,[\text{T-ZERO}] \qquad \dfrac{0 : \mathrm{Nat}}{\mathrm{pred}\ 0 : \mathrm{Nat}}\,[\text{T-PRED}]}{\mathrm{if}\ \mathrm{iszero}\ 0\ \mathrm{then}\ 0\ \mathrm{else}\ \mathrm{pred}\ 0 : \mathrm{Nat}}\,[\text{T-IF}]$$

A typing <u>derivation</u> is a <u>tree</u> of instances of typing rules

$$\dfrac{\dfrac{\dfrac{}{0 : \mathrm{Nat}}\,[\text{T-ZERO}]}{\mathrm{iszero}\ 0 : \mathrm{Bool}}\,[\text{T-ISZERO}] \qquad \dfrac{}{0 : \mathrm{Nat}}\,[\text{T-ZERO}] \qquad \dfrac{\dfrac{}{0 : \mathrm{Nat}}\,[\text{T-ZERO}]}{\mathrm{pred}\ 0 : \mathrm{Nat}}\,[\text{T-PRED}]}{\text{if iszero } 0 \text{ then } 0 \text{ else pred } 0 : \mathrm{Nat}}\,[\text{T-IF}]$$

Any term t that has a derivation t : T is "<u>WELL-TYPED</u>"

Type safety = Progress + Preservation

Progress: A well-typed term is not stuck (either it is a value, or it can take a step according to operational semantics)

Preservation: If a well-typed term takes a step, the resulting term is also well-typed

$0 : \text{Nat}$ [T-ZERO]

$\text{true} : \text{Bool}$ [T-TRUE]

$\text{false} : \text{Bool}$ [T-FALSE]

$$\frac{t_1 : \text{Bool} \quad t_2 : T \quad t_3 : T}{\text{if } t_1 \text{ then } t_2 \text{ else } t_3 : T} \text{ [T-IF]}$$

$$\frac{t_1 : \text{Nat}}{\text{succ } t_1 : \text{Nat}} \text{ [T-SUCC]}$$

$$\frac{t_1 : \text{Nat}}{\text{pred } t_1 : \text{Nat}} \text{ [T-PRED]}$$

$$\frac{t_1 : \text{Nat}}{\text{iszero } t_1 : \text{Bool}} \text{ [T-ISZERO]}$$

Lemma [canonical forms]:

1. If v (a value) $: \text{Bool}$ then v is either true or false

2. If $v : \text{Nat}$ then v is 0, or (succ 0), or succ(succ 0) or, succ (succ (succ 0)) ....

$0 : Nat$ [T-ZERO]

$true : Bool$ [T-TRUE]

$false : Bool$ [T-FALSE]

$$\frac{t_1 : Bool \quad t_2 : T \quad t_3 : T}{\text{if } t_1 \text{ then } t_2 \text{ else } t_3 : T} \text{ [T-IF]}$$

$$\frac{t_1 : Nat}{succ\ t_1 : Nat} \text{ [T-SUCC]}$$

$$\frac{t_1 : Nat}{pred\ t_1 : Nat} \text{ [T-PRED]}$$

$$\frac{t_1 : Nat}{iszero\ t_1 : Bool} \text{ [T-ISZERO]}$$

Progress Thm:

Suppose t is well typed (i.e., $\exists T$ s.t. $t : T$)

Then, either t is a value

or $\exists t'$ s.t. $t \rightarrow t'$

$0 : Nat$ [T-ZERO]

$true : Bool$ [T-TRUE]

$false : Bool$ [T-FALSE]

$$\frac{t_1 : Bool \quad t_2 : T \quad t_3 : T}{\text{if } t_1 \text{ then } t_2 \text{ else } t_3 : T} \text{ [T-IF]}$$

$$\frac{t_1 : Nat}{succ\ t_1 : Nat} \text{ [T-SUCC]}$$

$$\frac{t_1 : Nat}{pred\ t_1 : Nat} \text{ [T-PRED]}$$

$$\frac{t_1 : Nat}{iszero\ t_1 : Bool} \text{ [T-ISZERO]}$$

Progress Thm:

Suppose t is well typed
(i.e., $\exists T$ st $t : T$)
Then, either t is a value
or $\exists t'$ s.t $t \to t'$

Proof:

By induction on derivation of $t : T$.

[By induction hypothesis, theorem holds for all subterms used in the derivation]

$0 : \text{Nat}$ [T-ZERO]

$\text{true} : \text{Bool}$ [T-TRUE]

$\text{false} : \text{Bool}$ [T-FALSE]

$$\frac{t_1 : \text{Bool} \quad t_2 : T \quad t_3 : T}{\text{if } t_1 \text{ then } t_2 \text{ else } t_3 : T} \text{ [T-IF]}$$

$$\frac{t_1 : \text{Nat}}{\text{succ } t_1 : \text{Nat}} \text{ [T-SUCC]}$$

$$\frac{t_1 : \text{Nat}}{\text{pred } t_1 : \text{Nat}} \text{ [T-PRED]}$$

$$\frac{t_1 : \text{Nat}}{\text{iszero } t_1 : \text{Bool}} \text{ [T-ISZERO]}$$

Preservation Thm:

Suppose $t : T$ and $t \to t'$,
then $t' : T$

Proof:

By induction on derivation of $t : T$

Recall .. untyped lambda calculus

Syntax:

$$e ::= x \quad \text{variable}$$
$$\mid \lambda x.\, e \quad \text{function abstraction}$$
$$\mid e_1\ e_2 \quad \text{function application}$$

terms

Let us add booleans ...

Syntax:

$$
\begin{aligned}
e ::= \; & x && \leftarrow \text{variable} \\
\mid \; & \lambda x.\, e && \leftarrow \text{function abstraction} \\
\mid \; & e_1 \; e_2 && \leftarrow \text{function application} \\
\mid \; & \text{true} && \\
\mid \; & \text{false} && \Big\} \leftarrow \text{boolean consts} \\
\mid \; & \text{if } e_1 \text{ then } e_2 \text{ else } e_3 && \} \leftarrow \text{conditional}
\end{aligned}
$$

terms

Next.. let us add types

Typed lambda calculus

Syntax:

$$
\begin{aligned}
e \; ::= \;\; & x && \leftarrow \text{variable} \\
\mid \; & \lambda x : T .\, e && \leftarrow \text{function abstraction} \\
\mid \; & e_1 \; e_2 && \leftarrow \text{function application} \\
\mid \; & \text{true} && \\
\mid \; & \text{false} && \leftarrow \text{boolean consts} \\
\mid \; & \text{if } e_1 \text{ then } e_2 \text{ else } e_3 && \leftarrow \text{conditional}
\end{aligned}
$$

terms

Next .. let us add types

Typed lambda calculus

Syntax:

$$
\begin{aligned}
e ::= \ & x \\
| \ & \lambda x : T .\ e \\
| \ & e_1\ e_2 \\
| \ & \text{true} \\
| \ & \text{false} \\
| \ & \text{if } e_1 \text{ then } e_2 \text{ else } e_3
\end{aligned}
$$

Values:

$$
\begin{aligned}
v : \ & \text{true} \\
| \ & \text{false} \\
| \ & \lambda x : T .\ e
\end{aligned}
$$

Types:

$$
\begin{aligned}
T : \ & \text{bool} \\
| \ & T \to T
\end{aligned}
$$

Typed lambda calculus

Syntax:

$$
\begin{aligned}
e ::= \;& x \\
\mid\;& \lambda x{:}T.\,e \\
\mid\;& e_1\, e_2 \\
\mid\;& \text{true} \\
\mid\;& \text{false} \\
\mid\;& \text{if } e_1 \text{ then } e_2 \text{ else } e_3
\end{aligned}
$$

Values:

$$
\begin{aligned}
v : \;& \text{true} \\
\mid\;& \text{false} \\
\mid\;& \lambda x{:}T.\,e
\end{aligned}
$$

Types:

$$
\begin{aligned}
T : \;& \text{bool} \\
\mid\;& T \to T
\end{aligned}
$$

Typing relation
or
Typing judgment

$\Gamma \vdash e : T$ ⇐ under the assumption Γ, e has type T

$$
\begin{aligned}
\Gamma ::= \;& \emptyset \\
\mid\;& \Gamma, x : T
\end{aligned}
$$

⇐ type assumption for free variables

Operational semantics

$$\text{if true then } e_1 \text{ else } e_2 \longrightarrow e_1 \qquad \text{[E-IFTRUE]}$$

$$\text{if false then } e_1 \text{ else } e_2 \longrightarrow e_2 \qquad \text{[E-IFFALSE]}$$

$$\frac{e_1 \longrightarrow e_1'}{\text{if } e_1 \text{ then } e_2 \text{ else } e_3 \longrightarrow \text{if } e_1' \text{ then } e_2 \text{ else } e_3} \qquad \text{[E-IF]}$$

$$\frac{e_1 \longrightarrow e_1'}{e_1\ e_2 \longrightarrow e_1'\ e_2} \qquad \text{[E-APP1]}$$

$$\frac{e_2 \longrightarrow e_2'}{v\ e_2 \longrightarrow v\ e_2'} \qquad \text{[E-APP2]}$$

$$(\lambda x : T_{11} .\ e_2)\ v_1 \longrightarrow [x \mapsto v_1]\ e_2 \qquad \text{[E-APPABS]}$$

Typing rules:

$\text{true} : \text{Bool}$ [T-TRUE] $\quad \text{false} : \text{Bool}$ [T-FALSE]

$$\frac{e_1 : \text{Bool} \qquad e_2 : T \qquad e_3 : T}{\text{if } e_1 \text{ then } e_2 \text{ else } e_3 : T} \text{ [T-IF]}$$

$$\frac{x : T \in \Gamma}{\Gamma \vdash x : T} \text{ [T-VAR]}$$

$$\frac{\Gamma, x : T_1 \vdash e_2 : T_2}{\Gamma \vdash \lambda x : T_1 . e_2 : T_1 \to T_2} \text{ [T-ABS]}$$

$$\frac{\Gamma \vdash e_1 : T_1 \to T_2 , \qquad \Gamma \vdash e_2 : T_1}{\Gamma \vdash e_1\, e_2 : T_2} \text{ [T-APP]}$$

Typing rules:

$\text{true} : \text{Bool}$ [T-TRUE] $\text{false} : \text{Bool}$ [T-FALSE]

$$\frac{e_1 : \text{Bool} \qquad e_2 : T \qquad e_3 : T}{\text{if } e_1 \text{ then } e_2 \text{ else } e_3 : T} \; \text{[T-IF]}$$

$$\frac{x : T \in \Gamma}{\Gamma \vdash x : T} \; \text{[T-VAR]}$$

$$\frac{\Gamma, x : T_1 \vdash e_2 : T_2}{\Gamma \vdash \lambda x : T_1 . e_2 : T_1 \to T_2} \; \text{[T-ABS]}$$

$$\frac{\Gamma \vdash e_1 : T_1 \to T_2 , \qquad \Gamma \vdash e_2 : T_1}{\Gamma \vdash e_1 \, e_2 : T_2} \; \text{[T-APP]}$$

$$\frac{}{\vdash (\lambda x : \text{Bool} . x) \; \text{true} : \text{Bool}}$$

Typing rules:

$$\mathsf{true} : \mathsf{Bool}\ [\text{T-TRUE}] \qquad \mathsf{false} : \mathsf{Bool}\ [\text{T-FALSE}]$$

$$\frac{e_1 : \mathsf{Bool} \quad e_2 : T \quad e_3 : T}{\mathsf{if}\ e_1\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3 : T}\ [\text{T-IF}]$$

$$\frac{x : T \in \Gamma}{\Gamma \vdash x : T}\ [\text{T-VAR}]$$

$$\frac{\Gamma, x : T_1 \vdash e_2 : T_2}{\Gamma \vdash \lambda x : T_1 . e_2 : T_1 \to T_2}\ [\text{T-ABS}]$$

$$\frac{\Gamma \vdash e_1 : T_1 \to T_2 , \quad \Gamma \vdash e_2 : T_1}{\Gamma \vdash e_1\ e_2 : T_2}\ [\text{T-APP}]$$

$$\dfrac{\dfrac{\dfrac{x : \mathsf{Bool} \in x : \mathsf{Bool}}{x : \mathsf{Bool} \vdash x : \mathsf{Bool}}\ \text{T-VAR}}{\vdash \lambda x : \mathsf{Bool} . x : \mathsf{Bool} \to \mathsf{Bool}}\ \text{T-ABS} \qquad \dfrac{}{\vdash \mathsf{true} : \mathsf{Bool}}\ \text{T-TRUE}}{\vdash (\lambda x : \mathsf{Bool} . x)\ \mathsf{true} : \mathsf{Bool}}\ (\text{T-APP})$$

Exercise: Derive type derivation tree for:

$$f : \mathsf{Bool} \to \mathsf{Bool} \vdash \lambda x : \mathsf{Bool} .\ f\,(\mathsf{if}\ x\ \mathsf{then}\ \mathsf{true}\ \mathsf{else}\ x) : \mathsf{Bool} \to \mathsf{Bool}$$

Typing rules:

$true : Bool$ [T-TRUE] $false : Bool$ [T-FALSE]

$$\frac{e_1 : Bool \quad e_2 : T \quad e_3 : T}{\text{if } e_1 \text{ then } e_2 \text{ else } e_3 : T} \text{[T-IF]}$$

$$\frac{x : T \in \Gamma}{\Gamma \vdash x : T} \text{[T-VAR]}$$

$$\frac{\Gamma, x : T_1 \vdash e_2 : T_2}{\Gamma \vdash \lambda x : T_1 . e_2 : T_1 \to T_2} \text{[T-ABS]}$$

$$\frac{\Gamma \vdash e_1 : T_1 \to T_2 , \quad \Gamma \vdash e_2 : T_1}{\Gamma \vdash e_1\, e_2 : T_2} \text{[T-APP]}$$

Lemma [Canonical forms]:

1. If v is a value of type bool then v is either true or false
2. If v is a value of type $T_1 \to T_2$, then v is of the form $\lambda x : T_1 . e$

Typing rules:

$\text{true} : \text{Bool}$ [T-TRUE] $\quad \text{false} : \text{Bool}$ [T-FALSE]

$$\frac{e_1 : \text{Bool} \quad e_2 : T \quad e_3 : T}{\text{if } e_1 \text{ then } e_2 \text{ else } e_3 : T} \text{ [T-IF]}$$

$$\frac{x : T \in \Gamma}{\Gamma \vdash x : T} \text{ [T-VAR]}$$

$$\frac{\Gamma, x : T_1 \vdash e_2 : T_2}{\Gamma \vdash \lambda x : T_1 . e_2 : T_1 \to T_2} \text{ [T-ABS]}$$

$$\frac{\Gamma \vdash e_1 : T_1 \to T_2 , \quad \Gamma \vdash e_2 : T_1}{\Gamma \vdash e_1 \, e_2 : T_2} \text{ [T-APP]}$$

Thm [Progress]

Suppose e is a closed, well-typed term.

ie.. $\vdash e : T$

Then either e is a value

or $\exists e'$ s.t.

$e \to e'$

Proof:

Induction on typing derivation of $e : T$

Typing rules:

$true : Bool$ [T-TRUE] $false : Bool$ [T-FALSE]

$$\frac{e_1 : Bool \quad e_2 : T \quad e_3 : T}{\text{if } e_1 \text{ then } e_2 \text{ else } e_3 : T} \text{[T-IF]}$$

$$\frac{x : T \in \Gamma}{\Gamma \vdash x : T} \text{[T-VAR]}$$

$$\frac{\Gamma, x : T_1 \vdash e_2 : T_2}{\Gamma \vdash \lambda x : T_1 . e_2 : T_1 \to T_2} \text{[T-ABS]}$$

$$\frac{\Gamma \vdash e_1 : T_1 \to T_2 , \quad \Gamma \vdash e_2 : T_1}{\Gamma \vdash e_1\, e_2 : T_2} \text{[T-APP]}$$

Thm [Progress]

Suppose e is a closed, well-typed term.

ie.. $\vdash e : T$

Then either e is a value

or $\exists e'$ s.t.

$$e \to e'$$

Proof:

Induction on typing derivation of $e : T$

Typing rules:

$true : Bool$ [T-TRUE] $false : Bool$ [T-FALSE]

$$\frac{e_1 : Bool \quad e_2 : T \quad e_3 : T}{\text{if } e_1 \text{ then } e_2 \text{ else } e_3 : T} \text{[T-IF]}$$

$$\frac{x : T \in \Gamma}{\Gamma \vdash x : T} \text{[T-VAR]}$$

$$\frac{\Gamma, x : T_1 \vdash e_2 : T_2}{\Gamma \vdash \lambda x : T_1 . e_2 : T_1 \to T_2} \text{[T-ABS]}$$

$$\frac{\Gamma \vdash e_1 : T_1 \to T_2 , \quad \Gamma \vdash e_2 : T_1}{\Gamma \vdash e_1 \, e_2 : T_2} \text{[T-APP]}$$

Thm [Preservation under substitution]

if $\Gamma, x : T' \vdash e : T$

and $\Gamma \vdash s : T'$

then

$$\Gamma \vdash [x \mapsto s] e : T$$

Proof:

By induction on derivation of

$$\Gamma, x : T' \vdash e : T$$

Typing rules:

$\mathsf{true} : \mathsf{Bool}$ [T-TRUE] $\mathsf{false} : \mathsf{Bool}$ [T-FALSE]

$$\frac{e_1 : \mathsf{Bool} \quad e_2 : T \quad e_3 : T}{\mathsf{if}\ e_1\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3 : T} \text{[T-IF]}$$

$$\frac{x : T \in \Gamma}{\Gamma \vdash x : T} \text{[T-VAR]}$$

$$\frac{\Gamma, x : T_1 \vdash e_2 : T_2}{\Gamma \vdash \lambda x : T_1 . e_2 : T_1 \to T_2} \text{[T-ABS]}$$

$$\frac{\Gamma \vdash e_1 : T_1 \to T_2, \quad \Gamma \vdash e_2 : T_1}{\Gamma \vdash e_1\ e_2 : T_2} \text{[T-APP]}$$

Thm [Preservation]

if $\Gamma \vdash t : T$ and $t \to t'$,

then $\Gamma \vdash t' : T$

Proof

By Induction on derivation of $\Gamma \vdash t : T$

Well-typing : Sufficient but not necessary

$e : T \Rightarrow$ "e will never get stuck"

Converse is not true

Well-typing : Sufficient but not necessary

$e : T \Rightarrow$ "e will never get stuck"

Converse is not true

eq1: if (true) then (succ 0) else false

eq2: if (false) then $(((\lambda x . x)\ y)\ (\lambda z . z))$
else
$\lambda x . x$

Curry-Howard Correspondence

or Curry-Howard Isomorphism

Correspondence between proofs in <u>constructive</u> logics and type derivations

Logic	type derivations
propositions	types
proposition $P \Rightarrow Q$	type $P \to Q$
proposition $P \wedge Q$	type $(P \times Q)$ [not covered]
proof of proposition P	term t of type P
proposition P is provable	type P is <u>inhabited</u> (by some term)

Next time...

Type inference ...