Introduction to (untyped) lambda calculus

Syntax

$t ::=$

x	variable
$\lambda x.t$	abstraction
$t\ t$	application
true | false	boolean constants
if t then t else t	conditional
$\mathbb{I}$	integer constants
t op t	op $::= + \mid - \mid < \mid = \mid \ldots$

$v ::=$ values

$\lambda x.t$

true | false

$\mathbb{I}$

Semantics

Specified as a set of rewrite rules:

1. $(\lambda x.\, t_1)\ v_1 \longrightarrow [x \mapsto v_1]\, t_1$ [E-AppAbs]

- $[x \mapsto v_1]\, t_1$ replaces all occurrences of x in t_1 with v_1
- For simplicity we assume that each different 'λ' in the term uses a distinct variable name.

2. $$\frac{t_1 \longrightarrow t_1'}{t_1\ t_2 \longrightarrow t_1'\ t_2}$$ [E-App1]

Semantics – II

3. $$\frac{t_2 \longrightarrow t_2'}{v_1\ t_2 \longrightarrow v_1\ t_2'}$$ [E-App2]

4. if true then t_2 else t_3 $\longrightarrow$ t_2 [E-IfTrue]

5. if false then t_2 else t_3 $\longrightarrow$ t_3 [E-IfFalse]

6. $$\frac{t_1 \rightarrow t_1'}{\text{if } t_1 \text{ then } t_2 \text{ else } t_3 \longrightarrow \text{if } t_1' \text{ then } t_2 \text{ else } t_3}$$ [E-If]

7. $$\frac{t_1 \longrightarrow t_1'}{t_1 \text{ op } t_2 \longrightarrow t_1' \text{ op } t_2}$$ [E-Arith1]

Semantics - III

8. $$\frac{t_2 \to t_2'}{v_1 \text{ op } t_2 \to v_1 \text{ op } t_2'} \quad \text{[E-Arith2]}$$

9. $$\frac{[\![op]\!](v_1, v_2) = v_3}{v_1 \text{ op } v_2 \to v_3} \quad \text{[E-Arith3]}$$

[At each step, one of the rules should be applied at the <u>root</u> of the term.]

Examples

1. $(((\lambda x.\lambda y.x+y)\ 5)\ 6) \rightarrow ((\lambda y.5+y)\ 6) \rightarrow$
$(5+6) \rightarrow 11$

2. $(\lambda f.(f\ 6))((\lambda x.\lambda y.x+y)\ 5) \rightarrow$

$(\lambda f.(f\ 6))\ (\lambda y.5+y) \rightarrow$

$((\lambda y.5+y)\ 6) \rightarrow 5+6 \rightarrow 11$

3. $(\lambda x.x\ x)(\lambda x.x\ x) \rightarrow (\lambda x.x\ x)(\lambda x.x\ x)$
$\rightarrow \ldots$

[This term can keep reducing for ever, without ever reaching a normal form]

Examples - II

4.(a). $(\lambda y. z)(\underline{(\lambda x. x\ x)(\lambda x. x\ x)}) \xrightarrow{E\text{-}app2}$

$(\lambda y. z)(\underline{(\lambda x. x\ x)(\lambda x. x\ x)}) \xrightarrow{E\text{-}app2} \ldots$

4(b). If Rule E-AppAbs had allowed the argument to be not a value:

$\underline{(\lambda y. z)((\lambda x. x\ x)(\lambda x. x\ x))} \longrightarrow z$

[This shows that the set of rules chosen can influence termination.
However, rule E-AppAbs does not allow this reduction :( ]

5. $((\lambda x. (\lambda f. (f\ x))) 5)\ ((\lambda t. (\lambda z. z + t)) 1)$

Type Systems

What are type systems?

- An algorithmic technique to verify programs
  - An alternative to abstract interpretation
  - Normally applied to functional languages. Can be applied to imperative languages without arbitrary control flow.
- Operational semantics of underlying language needs to be specified using
  - Rewrite rules
  - A set of values (subset of normal forms)
    - Intuitively, values are meaningful normal forms.

Simply Typed Lambda Calculus

Types:

$$T ::= \text{Bool}$$
$$\quad\quad \text{Int}$$
$$\quad\quad T \to T$$

Type annotations:

$\lambda x.t,$ What is the type of x?
(Not clear. Need annotation.)

$\lambda x{:}type.\ t,$ 'type' is a member of the T language. E.g. Int, Bool, Int → Bool, (Int → Int) → Bool

Definitions

- Typing relation: an element of the domain
  Terms $\to$ Types (or, Environment $\times$ Terms $\to$ Types)
- Type system:
  - Underlying language & its operational semantics +
  - Domain of types (e.g., T in previous slides) +
  - Typing rules/constraints
- A term t is well-typed in a Type System if $\exists$ a typing relation R and a type T such that
  - $t : T \in R$
  - R satisfies typing rules

Definitions (contd.)

- Typing Algorithm
  - Given a term t and an environment (which gives types to all free variables in the term)
  - Either returns "Ill Typed", or
  - Assigns types to t (and to all subterms of t), such that
    - The types assigned to t & its subterms constitute a typing relation R such that
      - R satisfies the typing rules
      - (I.e., algorithm is sound)

Definitions (contd.)

- Soundness of typing rules:
  If a term t is well typed there exists no finite rewrite sequence that takes t to a stuck state (a normal form that's not a value)

Typing rules

~~Dealing with free variables...~~

$\ldots -2, -1, 0, 1, 2 \ldots : Int$ (T-Int)

$$\texttt{true} : \texttt{Bool} \qquad \text{(T-TRUE)}$$

$$\texttt{false} : \texttt{Bool} \qquad \text{(T-FALSE)}$$

$$\frac{\texttt{t}_1 : \texttt{Bool} \qquad \texttt{t}_2 : \texttt{T} \qquad \texttt{t}_3 : \texttt{T}}{\texttt{if } \texttt{t}_1 \texttt{ then } \texttt{t}_2 \texttt{ else } \texttt{t}_3 : \texttt{T}} \qquad \text{(T-IF)}$$

$$\frac{???}{\lambda \texttt{x}:\texttt{T}_1 . \texttt{t}_2 : \texttt{T}_1 \to \texttt{T}_2} \qquad \text{(T-ABS)}$$

Cannot be simply $t_2 : T_2$, because x occurs free inside t_2. t_2's type cannot be checked unless some assumption is made on the type of x.

~~... by introducing context information.~~

Therefore, we introduce environments into typing rules.

The context Γ (Gamma) is a (finite) mapping of variables to types

$$\Gamma \vdash \mathtt{true} : \mathtt{Bool} \qquad \text{(T-TRUE)}$$

$$\Gamma \vdash \mathtt{false} : \mathtt{Bool} \qquad \text{(T-FALSE)}$$

$$\frac{\Gamma \vdash \mathtt{t}_1 : \mathtt{Bool} \qquad \Gamma \vdash \mathtt{t}_2 : \mathtt{T} \qquad \Gamma \vdash \mathtt{t}_3 : \mathtt{T}}{\Gamma \vdash \mathtt{if}\ \mathtt{t}_1\ \mathtt{then}\ \mathtt{t}_2\ \mathtt{else}\ \mathtt{t}_3 : \mathtt{T}} \qquad \text{(T-IF)}$$

$$\frac{\Gamma, \mathtt{x}{:}\mathtt{T}_1 \vdash \mathtt{t}_2 : \mathtt{T}_2}{\Gamma \vdash \lambda \mathtt{x}{:}\mathtt{T}_1.\mathtt{t}_2 : \mathtt{T}_1 \to \mathtt{T}_2} \qquad \text{(T-ABS)}$$

$$\frac{\mathtt{x}{:}\mathtt{T} \in \Gamma}{\Gamma \vdash \mathtt{x} : \mathtt{T}} \qquad \text{(T-VAR)}$$

$$\frac{\Gamma \vdash \mathtt{t}_1 : \mathtt{T}_{11} \to \mathtt{T}_{12} \qquad \Gamma \vdash \mathtt{t}_2 : \mathtt{T}_{11}}{\Gamma \vdash \mathtt{t}_1\ \mathtt{t}_2 : \mathtt{T}_{12}} \qquad \text{(T-APP)}$$

Using a derivation tree to prove that a term is well-typed

$$
\dfrac{
\dfrac{
\dfrac{
\dfrac{x:B\to B \in \{x:B\to B,\ y:B\}}{\{x:B\to B,\ y:B\}\vdash x:B\to B}\,[\text{T-Var}]
\qquad
\dfrac{y:B \in \{x:B\to B,\ y:B\}}{\{x:B\to B,\ y:B\}\vdash y:B}\,[\text{T-Var}]
}{\{x:B\to B\},\ y:B \vdash (x\ y):B}\,[\text{T-App}]
}{\phi,\ x:B\to B \vdash (\lambda y:B.(x\ y)) : B\to B}\,[\text{T-Abs}]
}{\phi \vdash (\lambda x:B\to B.\lambda y:B.\ (x\ y)) : (B\to B)\to(B\to B)}\,[\text{T-Abs}]
$$

Updated defininitions

A typing relation is now a partial function   Environment x Terms ⇀ Types

We say $\Gamma \vdash t : T$ if there exists typing relation R that respects the rules and such that $R(\Gamma,t) = T$

We say $t : T$ iff $\varnothing \vdash t : T$

We say $\Gamma \vdash t$ is well typed if there exists a type T such that $\Gamma \vdash t : T$. We say t is well typed if $\varnothing \vdash t$ is well typed.

Properties

- The two key properties are:
  - Progress:

    A closed, well-typed term is not stuck

    If $\vdash t : T$, *then either* t *is a value or else* $t \longrightarrow t'$ *for some* t'.

  - Preservation:

    If $\Gamma \vdash t : T$ *and* $t \longrightarrow t'$, *then* $\Gamma \vdash t' : T$.

These two properties imply soundness of the type system.

- To prove them, we proceed in a similar way as for expressions

Inversion Lemma

1. If $\Gamma \vdash \mathtt{true} : \mathtt{R}$, then $\mathtt{R} = \mathtt{Bool}$.
2. If $\Gamma \vdash \mathtt{false} : \mathtt{R}$, then $\mathtt{R} = \mathtt{Bool}$.
3. If $\Gamma \vdash \mathtt{if}\ \mathtt{t_1}\ \mathtt{then}\ \mathtt{t_2}\ \mathtt{else}\ \mathtt{t_3} : \mathtt{R}$, then $\Gamma \vdash \mathtt{t_1} : \mathtt{Bool}$ and $\Gamma \vdash \mathtt{t_2}, \mathtt{t_3} : \mathtt{R}$.
4. If $\Gamma \vdash \mathtt{x} : \mathtt{R}$, then $\mathtt{x}{:}\mathtt{R} \in \Gamma$.
5. If $\Gamma \vdash \lambda \mathtt{x}{:}\mathtt{T_1}.\mathtt{t_2} : \mathtt{R}$, then $\mathtt{R} = \mathtt{T_1} \rightarrow \mathtt{R_2}$ for some $\mathtt{R_2}$ with $\Gamma, \mathtt{x}{:}\mathtt{T_1} \vdash \mathtt{t_2} : \mathtt{R_2}$.
6. If $\Gamma \vdash \mathtt{t_1}\ \mathtt{t_2} : \mathtt{R}$, then there is some type $\mathtt{T_{11}}$ such that $\Gamma \vdash \mathtt{t_1} : \mathtt{T_{11}} \rightarrow \mathtt{R}$ and $\Gamma \vdash \mathtt{t_2} : \mathtt{T_{11}}$.

Uniqueness and canonical forms

- Uniqueness:
  - In a given context Γ, if a term is typable, then it is only in one way

    This means there is a unique maximal typing relation R that satisfies all the rules.
- Canonical Forms:

1. If $\mathtt{v}$ is a value of type $\mathtt{Bool}$, then $\mathtt{v}$ is either $\mathtt{true}$ or $\mathtt{false}$.
2. If $\mathtt{v}$ is a value of type $\mathtt{T_1 \to T_2}$, then $\mathtt{v}$ has the form $\lambda\mathtt{x:T_1.t_2}$.

Progress

- Progress theorem:

> *Theorem:* Suppose $\mathtt{t}$ is a closed, well-typed term (that is, $\vdash \mathtt{t} : \mathtt{T}$ for some $\mathtt{T}$). Then either $\mathtt{t}$ is a value or else there is some $\mathtt{t}'$ with $\mathtt{t} \longrightarrow \mathtt{t}'$.

- Proof is by induction on the height of the typing derivation
- Note: if the term is not closed, progress can fail

Preservation

- Substitution Lemma:

Lemma: Types are preserved under substitition.

That is, if $\Gamma, x{:}S \vdash t : T$ and $\Gamma \vdash s : S$, then $\Gamma \vdash [x \mapsto s]t : T$.

- Preservation Theorem:

Theorem: If $\Gamma \vdash t : T$ and $t \longrightarrow t'$, then $\Gamma \vdash t' : T$.

- Proof by induction on height of the typing derivation

Algorithm

```
TypeCheck(Γ, t){
  switch (t){
   case v:
    if (∃T.((v:T) ∈ Γ))
        return T
    else
        return NO;

   Case λx:T.e:
    T' = TypeCheck((Γ, x:T), e);
    if (T' = NO)
        return NO;
    else
        return T→T';

   Case true: Case false:
    return Bool;

   Case t1 t2:
    T1 = TypeCheck(Γ, t1);
    T2 = TypeCheck(Γ, t2);
    if (T1 is of the form T2→T4 for some T4)
       return T4
    else
       return NO;

   Case "if t1 then t2 else t3":
    Ti = TypeCheck(Γ, ti), i = 1,2,3
    if (T1 = Bool and T2 = T3 and T2 ≠ NO)
       return T2
    else
       return NO;
  }
}
```

Soundness of Algorithm

Theorem: If TypeCheck (Γ, t) returns a type T (other than 'NO'), then there exists a proof tree rooted at $\Gamma \vdash t : T$.

In other words, T is a valid type for t under the environment Γ.