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
$\mathbb{N}$	integer constants
t op t	op ::= $+ \mid - \mid < \mid = \mid \cdots$

$v ::=$

$\lambda x.t$	values
true | false	
$\mathbb{N}$	

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. $$\text{if true then } t_2 \text{ else } t_3 \longrightarrow t_2$$ [E-IfTrue]

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

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

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

Semantics - III

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

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

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

App → Abs
App 1
App 2

If → If T
If F
If

Arith → Arith 1
Arith 2
Arith 3

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). $(\lambda y.z)((\lambda x.x\ x)(\lambda x.x\ x)) \rightarrow$

$(\lambda y.z)((\lambda x.x\ x)(\lambda x.x\ x)) \rightarrow \ldots$

[There are two candidate redexes in the term above. A redex is a subterm that can be reduced. In each step we use underlining to indicate the redex that we select for reducing.]

4(b). $(\lambda y.z)((\lambda x.x\ x)(\lambda x.x\ x)) \rightarrow z$

[This shows that the choice of the redex can influence termination!
However, rule E-AppAbs does not allow this reduction :( ]

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

$((\lambda x(\lambda f(f\ x)))5)\ ((\lambda t(\lambda z.\ z+t))1)$ → E-App1 + E-AppAbs

$\Rightarrow (\lambda f(f\ 5))\ ((\lambda t(\lambda z.\ z+t))1)$ → E App 2 + E App Abs

$\Rightarrow (\lambda f(f\ 5))\ (\lambda z\ z+1)$ → EApp Abs

$\Rightarrow (\lambda z.\ z+1)5$ → EApp Abs

$\Rightarrow 5+1$ → EArith3

$\Rightarrow 6$

$\Rightarrow$

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:

$T ::= Bool$

Int

$T \to T$

Type annotations:

$\lambda x.t_1$ What is the type of x? (Not clear. Need annotation.)

$\lambda x:type.\ t_1$ 'type' is a member of the T language. E.g. Int, Bool, $Int \to Bool$, $(Int \to Int) \to Bool$

Definitions

- Typing relation: an element of the domain Terms → Types (or, Environment × Terms → Types)

- Type system:
  - t → Underlying language & its operational semantics +
  - T → Domain of types (e.g., T in previous slides) +
  - Typing rules/constraints

- A term t is well-typed in a Type System if
  - ∃ a typing relation R and a type T such that
    - t : T ∈ R R(t) = T
    - 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...~~

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

$$\frac{}{\texttt{true} : \texttt{Bool}} \quad \text{(T-True)}$$

$$\frac{}{\texttt{false} : \texttt{Bool}} \quad \text{(T-False)}$$

$$\frac{t_1 : \texttt{Bool} \qquad t_2 : T \qquad t_3 : T}{\texttt{if } t_1 \texttt{ then } t_2 \texttt{ else } t_3 : T} \quad \text{(T-If)}$$

$$\frac{???}{\lambda x{:}T_1.t_2 : T_1 \to T_2} \quad \text{(T-Abs)}$$

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

$$\Gamma \vdash \texttt{true} : \texttt{Bool} \qquad \text{(T-True)}$$

$$\Gamma \vdash \texttt{false} : \texttt{Bool} \qquad \text{(T-False)}$$

$$\frac{\Gamma \vdash t_1 : \texttt{Bool} \quad \Gamma \vdash t_2 : T \quad \Gamma \vdash t_3 : T}{\Gamma \vdash \texttt{if } t_1 \texttt{ then } t_2 \texttt{ else } t_3 : T} \qquad \text{(T-If)}$$

$$\frac{\Gamma, x{:}T_1 \vdash t_2 : T_2}{\Gamma \vdash \lambda x{:}T_1.t_2 : T_1 \to T_2} \qquad \text{(T-Abs)}$$

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

$$\frac{\Gamma \vdash t_1 : T_{11} \to T_{12} \quad \Gamma \vdash t_2 : T_{11}}{\Gamma \vdash t_1\ t_2 : T_{12}} \qquad \text{(T-App)}$$

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

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

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

Uniqueness and canonical forms

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

- Canonical Forms:

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

Progress

- Progress theorem:

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

- Proof is by induction on the typing derivation

height of the

- 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 typing derivation

height of the

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 = Type check (Γ, t1);
      T2 = Type check (Γ, t2);
      if (∃T4.(T1 = T2 → T4))
        return T4
      else
        return NO;

    Case "if t1 Then t2 else t3":
      Ti = Type check (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.

$$\lambda x.\lambda y.((\lambda z.(x\,(z+1)))((\lambda w.w+1)\,y))$$

1. Show the term above with suitable type annotations given to all variables introduced in λ-abstractions, such that the term becomes well-typed. Assume that all primitive values in this problem are integers.

$TC(\emptyset, \lambda x.\lambda y.(\ \ldots\ ))$

$TC(\{x:Int\}, \lambda y.(\ \ldots\ ))$

$TC(\{x:Int, y:Int\}, (\lambda z.(\ \ldots\ ))\ ((\lambda w.w+1)\,y))$

$TC(\{x,y:Int\}, \lambda z.(x(z+1)))$ $\quad TC(\{x,y:Int\}, ((\lambda w.w+1)\,y))$

$TC(\{x,y,z:Int\}, (x\,(z+1)))$ $\quad (x(z+1)) : ?$ Ill Typed

$x : Int$ $\quad z+1 : Int$

$TC(\{x,y,z:Int\}, x)$ $\quad TC(\{x,y,z:Int\}, z+1)$