

Polymorphism

- Outline
 - *Simple types*
 - *Logic*
 - *System F (Second-order lambda calculus, Girard 1972)*



Simply-typed lambda-calculus

- Types are “simple,” meaning not polymorphic
- We start with a set of base types, and a type of functions

$$\begin{array}{l} t \quad ::= \quad \bullet \quad \quad \quad (\text{or } \mathbb{N}, \mathbb{Z}, \mathbb{B}, \dots) \\ \quad \quad | \quad t \rightarrow t \quad (\text{functions}) \end{array}$$



Programs with minimal typing info

- The programs contain just enough type information
- Concise types
 - *Add a type constraint to function parameters*

$$e ::= v \mid e_1 \ e_2 \mid \lambda x:t.e$$


Evaluation

- Ignore the types

$$(\lambda x:t.e_1) e_2 \rightarrow_{\beta} e_1[e_2/x]$$

Define multi-step reduction \rightarrow_{β}^* in the normal way (**not** symmetric).



Typing rules

- Type judgments are defined by inference rules (just like we used for specifying natural semantics)
- We need three rules
 - *Variables*
 - *Applications*
 - *Abstractions (functions)*



Typing rules

$$\frac{}{\Gamma, x:t \vdash x : t} \text{ var}$$

$$\frac{\Gamma \vdash e_1 : (s \rightarrow t) \quad \Gamma \vdash e_2 : s}{\Gamma \vdash (e_1 e_2) : t} \text{ app}$$

$$\frac{\Gamma, x:s \vdash t}{\Gamma \vdash (\lambda x:s.e) : (s \rightarrow t)} \text{ abs}$$



K combinator

K combinator:

$$\frac{\frac{\frac{}{x:s, y:t \vdash x:s} \quad 3}{x:s \vdash (\lambda y:t.x) : (t \rightarrow s)} \quad 2}{\vdash (\lambda x:s.\lambda y:t.x) : (s \rightarrow t \rightarrow s)} \quad 1$$



Type safety

- Well-typed programs do not get “stuck”
- Four parts:
 - *All well-typed programs are closed*
 - *Substitution lemma*
 - *Preservation*
 - *Progress*



All well-typed programs are closed

If $\Gamma \vdash e \in t$ and x does not appear in Γ , then x is not free in e .

var if e is a var, trivial.

abs if $e = \lambda y:t_1.e'$, then $t = t_1 \rightarrow t_2$, and

$$\frac{\Gamma, y:t_1 \vdash e' \in t_2}{\Gamma \vdash (\lambda y:t_1.e') \in (t_1 \rightarrow t_2)} \text{ abs}$$

If $x \neq y$ then then x is not free in e' by induction.

app if $e = e_1 e_2$, then $\Gamma \vdash e_1 \in t' \rightarrow t$ and $\Gamma \vdash e_2 \in t'$, for some t' . By induction x is not free in e_1 or e_2 .



Substitution

If $\Gamma, x:s, \Delta \vdash e_1 \in t$, and $\Gamma, \Delta \vdash e_2 \in s$, then $\Gamma, \Delta \vdash e_1[e_2/x] \in t$.

var if e is a var, trivial.

app if $e = e_3 e_4$, result follows by induction.

abs if $e = \lambda y:u.e_3$, then $t = u \rightarrow v$, and $\Gamma, x:s, \Delta, y:u \vdash e_3 \in v$, so $\Gamma, \Delta, y:u \vdash e_3[e_2/x]:v$ by induction.



Subject-reduction (preservation)

Theorem (Preservation) If $e_1 \rightarrow_{\beta}^* e_2$ and $\Gamma \vdash e_1 \in t$ then $\Gamma \vdash e_2 \in t$.

Proof

- Substitution is the key lemma
- Reduction
 - $(\lambda x:t.e_1) e_2 \rightarrow_{\beta} e_1[e_2/x]$
 - ignores types
 - can be used in any context



Subject-reduction proof

- Use induction to reduce to a single β step.
- Show if $\Gamma \vdash (\lambda x:t_1.e_1) e_2 \in t_2$, then $\Gamma \vdash e_1[e_2/x] \in t_2$.
- By typing rules, $\Gamma \vdash (\lambda x:t_1.e_1) \in (t_1 \rightarrow t_2)$, and $\Gamma \vdash e_2 \in t_1$.
- Then $\Gamma, x:t_1 \vdash e_1 \in t_2$,
- So $\Gamma \vdash e_1[e_2/x] \in t_2$ by substitution lemma.



Progress

Theorem (Progress) If $e : t$ and e is not a value (it is not in normal form), then there is an e' such that $e \rightarrow_{\beta} e'$.

Proof If e is not a value, it must either:

- Contain a redex, but then it can be reduced.
- Have weak head normal form $\lambda e_1 e_2 \cdots e_n$, where e_1, \dots, e_n are values. Not possible, because this program is not closed.



Normalization

- Strong-normalization
 - *Every reduction sequence terminates*
 - *We won't prove this*
- Intuition: weak normalization
 - *Some reduction sequence terminates*
 - *Define a well-order on types*
 - *The "order" of a type is the number of arrows in the type*
 - *The "order" of a program is the number of types it contains with maximal order*
 - *To evaluate a program*
 - *Pick a redex where the function type has maximal order*
 - *Reducing it will decrease its order*



Polymorphic lambda calculus

- Simply-typed lambda calculus is too weak
- Outline for the next few parts
 - *First-order logic*
 - *Curry-Howard isomorphism*
 - *Polymorphic lambda calculus*
 - *Type safety*
 - *Type inference*
- After that
 - *Imperative languages (tentative)*
 - *Axiomatic semantics*



Logic

- Logic, languages, and types are inextricably linked
- We'll need to develop some logical tools
 - *So we can reason about programs*
 - *To serve as a design guide*
 - *To define algorithms like type inference*
 - *Because propositions and types (and proofs and programs) are really the same thing*



Defining logics

- A logic is defined in three parts
 - *Syntax*
 - *Define what is “true”*
 - *Define derivation procedures*



Defining the syntax

Start with a countable set of propositional letters P, Q, R, \dots
Define the propositions inductively:

- \top (true) is a proposition
- \perp (false) is a proposition
- Any propositional letter is a proposition
- If A is a proposition, so is $\neg A$ (negation)
- If A and B are propositions, so are
 - $A \wedge B$ (conjunction)
 - $A \vee B$ (disjunction)
 - $A \Rightarrow B$ (implication)



Standard syntax definition

Propositions:

$$\begin{array}{l} e ::= \top \mid \perp \\ \quad | P, Q, R, \dots \\ \quad | \neg e \\ \quad | e \wedge e \\ \quad | e \vee e \\ \quad | e \Rightarrow e \end{array}$$



Semantics

- The semantics of constants: $\top = 1, \perp = 0$
- The semantics of a propositional letter is its truth value.
- The semantics of a compound proposition is determined by truth tables.

A	B	$A \vee B$
0	0	0
0	1	1
1	0	1
1	1	1

A	B	$A \wedge B$
0	0	0
0	1	0
1	0	0
1	1	1

A	B	$A \wedge B$
0	0	1
0	1	1
1	0	0
1	1	1



Extending the truth assignment

A	B	C	$A \wedge B$	$(A \wedge B) \Rightarrow C$
0	0	0	0	1
0	0	1	0	1
0	1	0	0	1
0	1	1	0	1
1	0	0	0	1
1	0	1	0	1
1	1	0	1	0
1	1	1	1	1



Some definitions

- A *truth assignment* \mathcal{A} assigns a value to each propositional letter A a unique truth value $\mathcal{A}(A) \in \{0, 1\}$
- A *truth valuation* \mathcal{V} assigns a truth value to each proposition (it can be constructed from a \mathcal{A} by following the truth tables).
- A proposition α is *satisfiable* if there is a truth valuation $\mathcal{V}(\alpha) = 1$
- A proposition α is *true* (a tautology) if it is true in all valuations.



A proof (Pierce's Law)

A	B	$A \Rightarrow B$	$(A \Rightarrow B) \Rightarrow A$	$((A \Rightarrow B) \Rightarrow A) \Rightarrow A$
0	0	1	0	1
0	1	1	0	1
1	0	0	1	1
1	1	1	1	1



Derivations

- Truth tables are hard to use
- We want a mechanical method for proving propositions
- Use sequents and truth judgments



Sequents

- A *sequent* has the form $\Gamma \vdash \Delta$
- Δ is a list of propositions $\alpha_1, \dots, \alpha_n$
- Γ is a *context* containing a list of propositions β_1, \dots, β_n
- We can extend valuations to sequents, to get the following semantics:
 - A sequent $\beta_1, \dots, \beta_n \vdash \alpha_1, \dots, \alpha_n$ is true if some α_i is true whenever β_1, \dots, β_n are all true.



Derivations

- There are two kinds of inference rules
 - *Introduction* rules operate on the right of the turnstile
 - *Elimination* rules operate on the left of the turnstile
- The base axiom

$$\frac{}{\Gamma_1, \alpha, \Gamma_2 \vdash \Delta_1, \alpha, \Delta_2} \text{ axiom}$$



Introduction rules, part I

$$\frac{}{\Gamma \vdash \Delta_1, \top, \Delta_2} \text{ true intro}$$

$$\frac{\Gamma \vdash \Delta_1, \alpha, \Delta_2 \quad \Gamma \vdash \Delta_1, \beta, \Delta_2}{\Gamma \vdash \Delta_1, \alpha \wedge \beta, \Delta_2} \text{ and intro}$$

$$\frac{\Gamma \vdash \Delta_1, \alpha, \beta, \Delta_2}{\Gamma \vdash \Delta_1, \alpha \vee \beta, \Delta_2} \text{ or intro}$$



Introduction rules, part II

$$\frac{\Gamma, \alpha \vdash \Delta_1, \beta, \Delta_2}{\Gamma \vdash \Delta_1, \alpha \Rightarrow \beta, \Delta_2} \text{ implies intro}$$

$$\frac{\Gamma, \alpha \vdash \Delta_1, \Delta_2}{\Gamma \vdash \Delta_1, \neg \alpha, \Delta_2} \text{ not intro}$$



Elimination rules

$$\frac{}{\Gamma_1, \perp, \Gamma_2 \vdash \Delta} \text{ false elim}$$

$$\frac{\Gamma_1, \alpha, \beta, \Gamma_2 \vdash \Delta}{\Gamma_1, \alpha \wedge \beta, \Gamma_2 \vdash \Delta} \text{ and elim}$$

$$\frac{\Gamma_1, \alpha, \Gamma_2 \vdash \Delta \quad \Gamma_1, \beta, \Gamma_2 \vdash \Delta}{\Gamma_1, \alpha \vee \beta, \Gamma_2 \vdash \Delta} \text{ or elim}$$

$$\frac{\Gamma_1, \beta, \Gamma_2 \vdash \Delta \quad \Gamma_1, \Gamma_2 \vdash \alpha, \Delta}{\Gamma_1, \alpha \Rightarrow \beta, \Gamma_2 \vdash \Delta} \text{ implies elim}$$



Rule table

$$\overline{\Gamma \vdash \Delta_1, \top, \Delta_2}$$

$$\overline{\Gamma_1, \perp, \Gamma_2 \vdash \Delta}$$

$$\frac{\Gamma \vdash \Delta_1, \alpha, \Delta_2 \quad \Gamma \vdash \Delta_1, \beta, \Delta_2}{\Gamma \vdash \Delta_1, \alpha \wedge \beta, \Delta_2}$$

$$\frac{\Gamma_1, \alpha, \beta, \Gamma_2 \vdash \Delta}{\Gamma_1, \alpha \wedge \beta, \Gamma_2 \vdash \Delta}$$

$$\frac{\Gamma \vdash \Delta_1, \alpha, \beta, \Delta_2}{\Gamma \vdash \Delta_1, \alpha \vee \beta, \Delta_2}$$

$$\frac{\Gamma_1, \alpha, \Gamma_2 \vdash \Delta \quad \Gamma_1, \beta, \Gamma_2 \vdash \Delta}{\Gamma_1, \alpha \vee \beta, \Gamma_2 \vdash \Delta}$$

$$\frac{\Gamma, \alpha \vdash \Delta_1, \beta, \Delta_2}{\Gamma \vdash \Delta_1, \alpha \Rightarrow \beta, \Delta_2}$$

$$\frac{\Gamma_1, \beta, \Gamma_2 \vdash \Delta \quad \Gamma_1, \Gamma_2 \vdash \alpha, \Delta}{\Gamma_1, \alpha \Rightarrow \beta, \Gamma_2 \vdash \Delta}$$



Proving the law of excluded middle

- Every proposition is either true or false

$$\frac{\frac{\overline{A \vdash A}}{\vdash A, \neg A}}{\vdash A \vee \neg A}$$

3
2
1



Pierce's law

$$\frac{\frac{\frac{\frac{\overline{A \vdash B, A} \quad 4}{\vdash A \Rightarrow B, A} \quad 3}{(A \Rightarrow B) \Rightarrow A \vdash A} \quad 2}{\vdash ((A \Rightarrow B) \Rightarrow A) \Rightarrow A} \quad 1}{\vdash ((A \Rightarrow B) \Rightarrow A) \Rightarrow A} \quad 1$$



Currying

$$\frac{\frac{\frac{\frac{\frac{\overline{A, B \vdash A} \quad 5}{A, B \vdash A \wedge B} \quad 4}{(A \wedge B) \Rightarrow C, A, B \vdash C} \quad 2}{(A \wedge B) \Rightarrow C \vdash A \Rightarrow B \Rightarrow C} \quad 1}{\vdash ((A \wedge B) \Rightarrow C) \Rightarrow (A \Rightarrow (B \Rightarrow C))} \quad 6}{\frac{\overline{A, B \vdash B} \quad 6}{C, A, B \vdash C} \quad 3} \quad 6$$

