CIS 500

Software Foundations

Fall 2003

15 October

Plans

Where we've been:

- Inductive definitions
  - abstract syntax
  - inference rules
- Proofs by structural induction
- Operational semantics
- The lambda-calculus
- Typing rules and type soundness

Plans

Where we've been:

- Inductive definitions
  - abstract syntax
  - inference rules
- Proofs by structural induction
- Operational semantics
- The lambda-calculus
- Typing rules and type soundness

Where we're going:

- "Simple types" for the lambda-calculus
- Formalizing more features of real-world languages (records, datatypes, references, exceptions, etc.)
- Subtyping
- Objects

The Simply Typed Lambda-Calculus

Lambda-calculus with booleans

t	::=		terms
		x	variable
		λx.t	abstraction
		t t	application
		true	constant true
		false	constant false
		if t then t else t	conditional
v	::=		values
		λx.t	abstraction value
		true	true value
		false	false value

"Simple Types"

T ::=		types
	Bool	type of booleans
	T→T	types of functions

Typing rules

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

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

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

Typing rules

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

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

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

$$\frac{}{\texttt{x} : \texttt{T}} \qquad \text{(T-Var)}$$

Typing rules

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

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

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

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

Typing rules

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

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

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

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

Typing rules

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

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

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

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

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

Typing rules

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

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

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

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

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

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

Typing Derivations

What derivations justify the following typing statements?

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

Properties of $\lambda_{\rightarrow}$

As before, the fundamental property of the type system we have just defined is soundness with respect to the operational semantics.

Properties of $\lambda_{\rightarrow}$

As before, the fundamental property of the type system we have just defined is soundness with respect to the operational semantics.

1. Progress: A closed, well-typed term is not stuck

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

2. Preservation: Types are preserved by one-step evaluation

   If $\Gamma \vdash \mathtt{t} : \mathtt{T}$ and $\mathtt{t} \longrightarrow \mathtt{t}'$, then $\Gamma \vdash \mathtt{t}' : \mathtt{T}$.

Proving progress

Same steps as before...

Proving progress

Same steps as before...

- inversion lemma for typing relation
- canonical forms lemma
- progress theorem

Typing rules again (for reference)

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

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

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

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

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

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

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 $\Gamma \vdash \texttt{if } \texttt{t}_1 \texttt{ then } \texttt{t}_2 \texttt{ else } \texttt{t}_3 : \texttt{R}$, then $\Gamma \vdash \texttt{t}_1 : \texttt{Bool}$ and $\Gamma \vdash \texttt{t}_2, \texttt{t}_3 : \texttt{R}$.

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 $\Gamma \vdash \texttt{if } \texttt{t}_1 \texttt{ then } \texttt{t}_2 \texttt{ else } \texttt{t}_3 : \texttt{R}$, then $\Gamma \vdash \texttt{t}_1 : \texttt{Bool}$ and $\Gamma \vdash \texttt{t}_2, \texttt{t}_3 : \texttt{R}$.
4. If $\Gamma \vdash \texttt{x} : \texttt{R}$, then

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 $\Gamma \vdash \texttt{if } \texttt{t}_1 \texttt{ then } \texttt{t}_2 \texttt{ else } \texttt{t}_3 : \texttt{R}$, then $\Gamma \vdash \texttt{t}_1 : \texttt{Bool}$ and $\Gamma \vdash \texttt{t}_2, \texttt{t}_3 : \texttt{R}$.
4. If $\Gamma \vdash \texttt{x} : \texttt{R}$, then $\texttt{x:R} \in \Gamma$.

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 $\Gamma \vdash \texttt{if } \texttt{t}_1 \texttt{ then } \texttt{t}_2 \texttt{ else } \texttt{t}_3 : \texttt{R}$, then $\Gamma \vdash \texttt{t}_1 : \texttt{Bool}$ and $\Gamma \vdash \texttt{t}_2, \texttt{t}_3 : \texttt{R}$.
4. If $\Gamma \vdash \texttt{x} : \texttt{R}$, then $\texttt{x:R} \in \Gamma$.
5. If $\Gamma \vdash \lambda\texttt{x:T}_1\texttt{.t}_2 : \texttt{R}$, then

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 $\Gamma \vdash \texttt{if } \texttt{t}_1 \texttt{ then } \texttt{t}_2 \texttt{ else } \texttt{t}_3 : \texttt{R}$, then $\Gamma \vdash \texttt{t}_1 : \texttt{Bool}$ and $\Gamma \vdash \texttt{t}_2, \texttt{t}_3 : \texttt{R}$.
4. If $\Gamma \vdash \texttt{x} : \texttt{R}$, then $\texttt{x}{:}\texttt{R} \in \Gamma$.
5. If $\Gamma \vdash \lambda\texttt{x}{:}\texttt{T}_1.\texttt{t}_2 : \texttt{R}$, then $\texttt{R} = \texttt{T}_1 \rightarrow \texttt{R}_2$ for some $\texttt{R}_2$ with $\Gamma, \texttt{x}{:}\texttt{T}_1 \vdash \texttt{t}_2 : \texttt{R}_2$.

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 $\Gamma \vdash \texttt{if } \texttt{t}_1 \texttt{ then } \texttt{t}_2 \texttt{ else } \texttt{t}_3 : \texttt{R}$, then $\Gamma \vdash \texttt{t}_1 : \texttt{Bool}$ and $\Gamma \vdash \texttt{t}_2, \texttt{t}_3 : \texttt{R}$.

4. If $\Gamma \vdash \texttt{x} : \texttt{R}$, then $\texttt{x:R} \in \Gamma$.

5. If $\Gamma \vdash \lambda \texttt{x:T}_1 \texttt{.t}_2 : \texttt{R}$, then $\texttt{R} = \texttt{T}_1 \rightarrow \texttt{R}_2$ for some $\texttt{R}_2$ with $\Gamma, \texttt{x:T}_1 \vdash \texttt{t}_2 : \texttt{R}_2$.

6. If $\Gamma \vdash \texttt{t}_1\ \texttt{t}_2 : \texttt{R}$, then

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 $\Gamma \vdash \texttt{if } \texttt{t}_1 \texttt{ then } \texttt{t}_2 \texttt{ else } \texttt{t}_3 : \texttt{R}$, then $\Gamma \vdash \texttt{t}_1 : \texttt{Bool}$ and $\Gamma \vdash \texttt{t}_2, \texttt{t}_3 : \texttt{R}$.
4. If $\Gamma \vdash \texttt{x} : \texttt{R}$, then $\texttt{x}{:}\texttt{R} \in \Gamma$.
5. If $\Gamma \vdash \lambda \texttt{x}{:}\texttt{T}_1.\texttt{t}_2 : \texttt{R}$, then $\texttt{R} = \texttt{T}_1 \to \texttt{R}_2$ for some $\texttt{R}_2$ with $\Gamma, \texttt{x}{:}\texttt{T}_1 \vdash \texttt{t}_2 : \texttt{R}_2$.
6. If $\Gamma \vdash \texttt{t}_1 \ \texttt{t}_2 : \texttt{R}$, then there is some type $\texttt{T}_{11}$ such that $\Gamma \vdash \texttt{t}_1 : \texttt{T}_{11} \to \texttt{R}$ and $\Gamma \vdash \texttt{t}_2 : \texttt{T}_{11}$.

Canonical Forms

Lemma:

Canonical Forms

Lemma:

1. If `v` is a value of type `Bool`, then

Canonical Forms

Lemma:

1. If `v` is a value of type `Bool`, then `v` is either `true` or `false`.

Canonical Forms

Lemma:

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

Canonical Forms

Lemma:

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` has the form $\lambda x{:}T_1.t_2$.

Progress

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: By induction

Progress

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: By induction on typing derivations.

Progress

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: By induction on typing derivations. The cases for boolean constants and conditions are the same as before. The variable case is trivial (because $\mathtt{t}$ is closed). The abstraction case is immediate, since abstractions are values.

Progress

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: By induction on typing derivations. The cases for boolean constants and conditions are the same as before. The variable case is trivial (because t is closed). The abstraction case is immediate, since abstractions are values.

Consider the case for application, where $t = t_1\ t_2$ with $\vdash t_1 : T_{11} \rightarrow T_{12}$ and $\vdash t_2 : T_{11}$.

Progress

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: By induction on typing derivations. The cases for boolean constants and conditions are the same as before. The variable case is trivial (because t is closed). The abstraction case is immediate, since abstractions are values.

Consider the case for application, where $t = t_1\ t_2$ with $\vdash t_1 : T_{11} \to T_{12}$ and $\vdash t_2 : T_{11}$. By the induction hypothesis, either t_1 is a value or else it can make a step of evaluation, and likewise t_2.

Progress

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: By induction on typing derivations. The cases for boolean constants and conditions are the same as before. The variable case is trivial (because $\mathtt{t}$ is closed). The abstraction case is immediate, since abstractions are values.

Consider the case for application, where $\mathtt{t} = \mathtt{t}_1\ \mathtt{t}_2$ with $\vdash \mathtt{t}_1 : \mathtt{T}_{11} \to \mathtt{T}_{12}$ and $\vdash \mathtt{t}_2 : \mathtt{T}_{11}$. By the induction hypothesis, either $\mathtt{t}_1$ is a value or else it can make a step of evaluation, and likewise $\mathtt{t}_2$. If $\mathtt{t}_1$ can take a step, then rule E-App1 applies to $\mathtt{t}$. If $\mathtt{t}_1$ is a value and $\mathtt{t}_2$ can take a step, then rule E-App2 applies. Finally, if both $\mathtt{t}_1$ and $\mathtt{t}_2$ are values, then the canonical forms lemma tells us that $\mathtt{t}_1$ has the form $\lambda\mathtt{x}{:}\mathtt{T}_{11}.\mathtt{t}_{12}$, and so rule E-AppAbs applies to $\mathtt{t}$.

Proving Preservation

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

Proof: By induction

Proving Preservation

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

Proof: By induction on typing derivations.

[Which case is the hard one?]

Proving Preservation

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

Proof: By induction on typing derivations.

[Which case is the hard one?]

Case T-App: Given $t = t_1 \; t_2$

$\Gamma \vdash t_1 : T_{11} \to T_{12}$

$\Gamma \vdash t_2 : T_{11}$

$T = T_{12}$

Show $\Gamma \vdash t' : T_{12}$

Proving Preservation

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

Proof: By induction on typing derivations.

[Which case is the hard one?]

Case T-App: Given

$$t = t_1\ t_2$$

$$\Gamma \vdash t_1 : T_{11} \to T_{12}$$

$$\Gamma \vdash t_2 : T_{11}$$

$$T = T_{12}$$

Show

$$\Gamma \vdash t' : T_{12}$$

By the inversion lemma for evaluation, there are three subcases...

Proving Preservation

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

Proof: By induction on typing derivations.

[Which case is the hard one?]

Case T-App: Given $t = t_1\ t_2$

$\Gamma \vdash t_1 : T_{11} \to T_{12}$

$\Gamma \vdash t_2 : T_{11}$

$T = T_{12}$

Show $\Gamma \vdash t' : T_{12}$

By the inversion lemma for evaluation, there are three subcases...

Subcase: $t_1 = \lambda x{:}T_{11}.\ t_{12}$

t_2 a value v_2

$t' = [x \mapsto v_2]t_{12}$

Proving Preservation

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

Proof: By induction on typing derivations.

[Which case is the hard one?]

Case T-App: Given $t = t_1\ t_2$

$\Gamma \vdash t_1 : T_{11} \to T_{12}$

$\Gamma \vdash t_2 : T_{11}$

$T = T_{12}$

Show $\Gamma \vdash t' : T_{12}$

By the inversion lemma for evaluation, there are three subcases...

Subcase: $t_1 = \lambda x{:}T_{11}.\ t_{12}$

t_2 a value v_2

$t' = [x \mapsto v_2]t_{12}$

Uh oh.

The "Substitution Lemma"

Lemma: Types are preserved under substititution.

If $\Gamma, \mathtt{x}{:}\mathtt{S} \vdash \mathtt{t} : \mathtt{T}$ and $\Gamma \vdash \mathtt{s} : \mathtt{S}$, then $\Gamma \vdash [\mathtt{x} \mapsto \mathtt{s}]\mathtt{t} : \mathtt{T}$.

The "Substitution Lemma"

Lemma: Types are preserved under substitition.

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

Proof: ...

Preservation

Homework: Complete the proof of preservation

Discussion

Intro vs. elim forms

An introduction form for a given type gives us a way of constructing elements of this type.

An elimination form for a type gives us a way of using elements of this type.

The Curry-Howard Correspondence

In constructive logics, a proof of $\mathbf{P}$ must provide evidence for $\mathbf{P}$.

- "law of the excluded middle" — $\mathbf{P} \vee \neg\mathbf{P}$ — not recognized.

A proof of $\mathbf{P} \wedge \mathbf{Q}$ is a pair of evidence for $\mathbf{P}$ and evidence for $\mathbf{Q}$.

A proof of $\mathbf{P} \supset \mathbf{Q}$ is a procedure for transforming evidence for $\mathbf{P}$ into evidence for $\mathbf{Q}$.

Propositions as Types

Logic	Programming languages
propositions	types
proposition $P \supset Q$	type `P→Q`
proposition $P \wedge Q$	type `P` $\times$ `Q`
proof of proposition P	term `t` of type `P`
proposition P is provable	type `P` is inhabited (by some term)

Propositions as Types

Logic	Programming languages
propositions	types
proposition $P \supset Q$	type $P \to Q$
proposition $P \wedge Q$	type $P \times Q$
proof of proposition P	term t of type P
proposition P is provable	type P is inhabited (by some term)
	evaluation

Propositions as Types

Logic	Programming languages
propositions	types
proposition $P \supset Q$	type $\mathtt{P} \rightarrow \mathtt{Q}$
proposition $P \wedge Q$	type $\mathtt{P} \times \mathtt{Q}$
proof of proposition P	term $\mathtt{t}$ of type $\mathtt{P}$
proposition P is provable	type $\mathtt{P}$ is inhabited (by some term)
proof simplification	evaluation
(a.k.a. “cut elimination”)	

Erasure

$$\begin{aligned}
\mathsf{erase}(\mathtt{x}) &= \mathtt{x} \\
\mathsf{erase}(\lambda\mathtt{x}{:}\mathtt{T_1}.\ \mathtt{t_2}) &= \lambda\mathtt{x}.\ \mathsf{erase}(\mathtt{t_2}) \\
\mathsf{erase}(\mathtt{t_1}\ \mathtt{t_2}) &= \mathsf{erase}(\mathtt{t_1})\ \mathsf{erase}(\mathtt{t_2})
\end{aligned}$$

Typability

An untyped λ-term m is said to be **typable** if there is some term t in the simply typed lambda-calculus, some type T, and some context Γ such that $\mathsf{erase}(t) = m$ and $\Gamma \vdash t : T$.

Cf. **type reconstruction** in OCaml.

On to real programming languages...

Base types

Up to now, we've formulated "base types" (e.g. `Nat`) by adding them to the syntax of types, extending the syntax of terms with associated constants (`zero`) and operators (`succ`, etc.) and adding appropriate typing and evaluation rules. We can do this for as many base types as we like.

For more theoretical discussions (as opposed to programming) we can often ignore the term-level inhabitants of base types, and just treat these types as uninterpreted constants.

E.g., suppose `B` and `C` are some base types. Then we can ask (without knowing anything more about `B` or `C`) whether there are any types `S` and `T` such that the term

```
(λf:S. λg:T. f g) (λx:B. x)
```

is well typed.

The Unit type

t	::=	...	terms
		unit	constant unit
v	::=	...	values
		unit	constant *unit*
T	::=	...	types
		Unit	unit type

New typing rules $\boxed{\Gamma \vdash t : T}$

$$\Gamma \vdash \texttt{unit} : \texttt{Unit} \qquad \text{(T-UNIT)}$$

Sequencing

t	::=	...	terms
		$t_1;t_2$	

Sequencing

$\mathtt{t} \quad ::= \quad \ldots$ terms

$\mathtt{t_1 ; t_2}$

$$\frac{\mathtt{t_1} \longrightarrow \mathtt{t_1'}}{\mathtt{t_1 ; t_2} \longrightarrow \mathtt{t_1' ; t_2}} \qquad \text{(E-SEQ)}$$

$$\mathtt{unit;t_2} \longrightarrow \mathtt{t_2} \qquad \text{(E-SEQNEXT)}$$

$$\frac{\Gamma \vdash \mathtt{t_1} : \mathtt{Unit} \qquad \Gamma \vdash \mathtt{t_2} : \mathtt{T_2}}{\Gamma \vdash \mathtt{t_1 ; t_2} : \mathtt{T_2}} \qquad \text{(T-SEQ)}$$

Derived forms

- Syntatic sugar
- Internal language vs. external (surface) language

Sequencing as a derived form

$$\mathtt{t_1;t_2} \stackrel{\text{def}}{=} (\lambda\mathtt{x}:\mathtt{Unit}.\mathtt{t_2})\ \mathtt{t_1}$$

$$\text{where } \mathtt{x} \notin FV(\mathtt{t_2})$$

Equivalence of the two definitions

[board]

Ascription

New syntactic forms

$\mathtt{t}$	::=	...	terms
		$\mathtt{t\ as\ T}$	ascription

New evaluation rules $\boxed{\mathtt{t} \longrightarrow \mathtt{t}'}$

$$\mathtt{v_1\ as\ T} \longrightarrow \mathtt{v_1} \qquad \text{(E-Ascribe)}$$

$$\frac{\mathtt{t_1} \longrightarrow \mathtt{t_1'}}{\mathtt{t_1\ as\ T} \longrightarrow \mathtt{t_1'\ as\ T}} \qquad \text{(E-Ascribe1)}$$

New typing rules $\boxed{\Gamma \vdash \mathtt{t} : \mathtt{T}}$

$$\frac{\Gamma \vdash \mathtt{t_1} : \mathtt{T}}{\Gamma \vdash \mathtt{t_1\ as\ T} : \mathtt{T}} \qquad \text{(T-Ascribe)}$$

Ascription as a derived form

$$\texttt{t as T} \stackrel{\text{def}}{=} (\lambda\texttt{x:T. x) t}$$

Let-bindings

New syntactic forms

$\mathtt{t}$	::=	...	terms
		$\mathtt{let\ x{=}t\ in\ t}$	let binding

New evaluation rules $\boxed{\mathtt{t} \longrightarrow \mathtt{t}'}$

$$\mathtt{let\ x{=}v_1\ in\ t_2} \longrightarrow [\mathtt{x} \mapsto \mathtt{v_1}]\mathtt{t_2} \qquad \text{(E-LetV)}$$

$$\frac{\mathtt{t_1} \longrightarrow \mathtt{t_1'}}{\mathtt{let\ x{=}t_1\ in\ t_2} \longrightarrow \mathtt{let\ x{=}t_1'\ in\ t_2}} \qquad \text{(E-Let)}$$

New typing rules $\boxed{\Gamma \vdash \mathtt{t} : \mathtt{T}}$

$$\frac{\Gamma \vdash \mathtt{t_1} : \mathtt{T_1} \qquad \Gamma, \mathtt{x{:}T_1} \vdash \mathtt{t_2} : \mathtt{T_2}}{\Gamma \vdash \mathtt{let\ x{=}t_1\ in\ t_2} : \mathtt{T_2}} \qquad \text{(T-Let)}$$

Pairs

t	::=	...	terms
		{t,t}	pair
		t.1	first projection
		t.2	second projection
v	::=	...	values
		{v,v}	pair value
T	::=	...	types
		$T_1 \times T_2$	product type

Evaluation rules for pairs

$$\{v_1, v_2\}.1 \longrightarrow v_1 \quad \text{(E-PairBeta1)}$$

$$\{v_1, v_2\}.2 \longrightarrow v_2 \quad \text{(E-PairBeta2)}$$

$$\frac{t_1 \longrightarrow t_1'}{t_1.1 \longrightarrow t_1'.1} \quad \text{(E-Proj1)}$$

$$\frac{t_1 \longrightarrow t_1'}{t_1.2 \longrightarrow t_1'.2} \quad \text{(E-Proj2)}$$

$$\frac{t_1 \longrightarrow t_1'}{\{t_1, t_2\} \longrightarrow \{t_1', t_2\}} \quad \text{(E-Pair1)}$$

$$\frac{t_2 \longrightarrow t_2'}{\{v_1, t_2\} \longrightarrow \{v_1, t_2'\}} \quad \text{(E-Pair2)}$$

Typing rules for pairs

$$\frac{\Gamma \vdash t_1 : T_1 \qquad \Gamma \vdash t_2 : T_2}{\Gamma \vdash \{t_1, t_2\} : T_1 \times T_2} \qquad \text{(T-Pair)}$$

$$\frac{\Gamma \vdash t_1 : T_{11} \times T_{12}}{\Gamma \vdash t_1.1 : T_{11}} \qquad \text{(T-Proj1)}$$

$$\frac{\Gamma \vdash t_1 : T_{11} \times T_{12}}{\Gamma \vdash t_1.2 : T_{12}} \qquad \text{(T-Proj2)}$$

Tuples

t	::=	...	terms
		$\{t_i\ ^{i\in 1..n}\}$	tuple
		$t.i$	projection
v	::=	...	values
		$\{v_i\ ^{i\in 1..n}\}$	tuple value
T	::=	...	types
		$\{T_i\ ^{i\in 1..n}\}$	tuple type

Evaluation rules for tuples

$$\{v_i\ ^{i\in 1..n}\}.j \longrightarrow v_j \qquad \text{(E-ProjTuple)}$$

$$\frac{t_1 \longrightarrow t_1'}{t_1.i \longrightarrow t_1'.i} \qquad \text{(E-Proj)}$$

$$\frac{t_j \longrightarrow t_j'}{\{v_i\ ^{i\in 1..j-1}, t_j, t_k\ ^{k\in j+1..n}\} \longrightarrow \{v_i\ ^{i\in 1..j-1}, t_j', t_k\ ^{k\in j+1..n}\}} \qquad \text{(E-Tuple)}$$

Typing rules for tuples

$$\frac{\text{for each } i \quad \Gamma \vdash t_i : T_i}{\Gamma \vdash \{t_i\ {}^{i \in 1..n}\} : \{T_i\ {}^{i \in 1..n}\}} \qquad \text{(T-Tuple)}$$

$$\frac{\Gamma \vdash t_1 : \{T_i\ {}^{i \in 1..n}\}}{\Gamma \vdash t_1.j : T_j} \qquad \text{(T-Proj)}$$

Records

t	::=	...	terms
		$\{l_i{=}t_i\ ^{i\in 1..n}\}$	record
		$t.l$	projection
v	::=	...	values
		$\{l_i{=}v_i\ ^{i\in 1..n}\}$	record value
T	::=	...	types
		$\{l_i{:}T_i\ ^{i\in 1..n}\}$	type of records

Evaluation rules for records

$$\{\mathtt{l}_i\mathtt{=}\mathtt{v}_i\ ^{i\in 1..n}\}\mathtt{.l}_j \longrightarrow \mathtt{v}_j \qquad \text{(E-ProjRcd)}$$

$$\frac{\mathtt{t}_1 \longrightarrow \mathtt{t}_1'}{\mathtt{t}_1\mathtt{.l} \longrightarrow \mathtt{t}_1'\mathtt{.l}} \qquad \text{(E-Proj)}$$

$$\frac{\mathtt{t}_j \longrightarrow \mathtt{t}_j'}{\begin{array}{l}\{\mathtt{l}_i\mathtt{=}\mathtt{v}_i\ ^{i\in 1..j-1}\mathtt{,l}_j\mathtt{=}\mathtt{t}_j\mathtt{,l}_k\mathtt{=}\mathtt{t}_k\ ^{k\in j+1..n}\} \\ \longrightarrow \{\mathtt{l}_i\mathtt{=}\mathtt{v}_i\ ^{i\in 1..j-1}\mathtt{,l}_j\mathtt{=}\mathtt{t}_j'\mathtt{,l}_k\mathtt{=}\mathtt{t}_k\ ^{k\in j+1..n}\}\end{array}} \qquad \text{(E-Rcd)}$$

Typing rules for records

$$\frac{\text{for each } i \quad \Gamma \vdash t_i : T_i}{\Gamma \vdash \{l_i = t_i \ ^{i \in 1..n}\} : \{l_i : T_i \ ^{i \in 1..n}\}} \qquad \text{(T-Rcd)}$$

$$\frac{\Gamma \vdash t_1 : \{l_i : T_i \ ^{i \in 1..n}\}}{\Gamma \vdash t_1.l_j : T_j} \qquad \text{(T-Proj)}$$