CIS 500

Software Foundations

Fall 2004

6 October

Midterm 1 is next Wednesday

- Today's lecture will not be covered by the midterm.
- Next Monday, review class.
- Old exams and review questions on webpage.
- No recitation sections next week.
- New office hours next week, watch newsgroup for details.

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

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*

Typing rules

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

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

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

The Simply Typed Lambda-Calculus

"Simple Types"

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

Typing rules

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

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

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

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

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

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

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

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

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

Typing rules

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

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

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

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

Typing Derivations

What derivations justify the following typing statements?

- $\vdash$ `(λx:Bool.x) true : Bool`
- `f:Bool→Bool` $\vdash$ `f (if false then true else false) : Bool`
- `f:Bool→Bool` $\vdash$ `λx:Bool. f (if x then false else x) : Bool→Bool`

Properties of $\lambda_{\to}$

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

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

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

Typing rules

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

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

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

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

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

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

Properties of $\lambda_{\to}$

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

Proving progress

Same steps as before...

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

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 $\Gamma \vdash \texttt{t}_2, \texttt{t}_3 : \texttt{R}$.

Proving progress

Same steps as before...

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 $\Gamma \vdash \texttt{t}_2, \texttt{t}_3 : \texttt{R}$.
4. If $\Gamma \vdash \texttt{x} : \texttt{R}$, then $\texttt{x}:\texttt{R} \in \Gamma$.

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 $\texttt{x}:\texttt{R} \in \Gamma$.
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 $\Gamma, \texttt{x}:\texttt{T}_1 \vdash \texttt{t}_2 : \texttt{R}_2$.

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 $\texttt{x}:\texttt{R} \in \Gamma$.
5. If $\Gamma \vdash \lambda \texttt{x}:\texttt{T}_1.\texttt{t}_2 : \texttt{R}$, 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 $\Gamma \vdash \texttt{if } t_1 \texttt{ then } t_2 \texttt{ else } t_3 : \texttt{R}$, 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 $\texttt{x}:\texttt{R} \in \Gamma$.
5. If $\Gamma \vdash \lambda \texttt{x}:T_1.t_2 : \texttt{R}$, then $\texttt{R} = T_1 \to R_2$ 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 $\Gamma \vdash t_1 : T_{11} \to \texttt{R}$ and $\Gamma \vdash t_2 : T_{11}$.

Canonical Forms

Lemma:

1. If $\texttt{v}$ is a value of type Bool, 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 $\Gamma \vdash \texttt{if } t_1 \texttt{ then } t_2 \texttt{ else } t_3 : \texttt{R}$, 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 $\texttt{x}:\texttt{R} \in \Gamma$.
5. If $\Gamma \vdash \lambda \texttt{x}:T_1.t_2 : \texttt{R}$, then $\texttt{R} = T_1 \to R_2$ 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

Canonical Forms

Lemma:

Canonical Forms

Lemma:

1. If $\mathtt{v}$ is a value of type `Bool`, then $\mathtt{v}$ is either `true` or `false`.
2. If $\mathtt{v}$ is a value of type $\mathtt{T_1 \to T_2}$, then

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

Canonical Forms

Lemma:

1. If $\mathtt{v}$ is a value of type `Bool`, then $\mathtt{v}$ is either `true` or `false`.

Canonical Forms

Lemma:

1. If $\mathtt{v}$ is a value of type `Bool`, then $\mathtt{v}$ is either `true` or `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

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 $\mathtt{t}$ 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 $\vdash \mathtt{t}_1 : \mathtt{T}_{11} \rightarrow \mathtt{T}_{12}$ 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 $\mathtt{t}_2$.

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.

Consider the case for application, where $\mathtt{t} = \mathtt{t}_1\ \mathtt{t}_2$ with $\vdash \mathtt{t}_1 : \mathtt{T}_{11} \rightarrow \mathtt{T}_{12}$ and $\vdash \mathtt{t}_2 : \mathtt{T}_{11}$.

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?]

Case T-App: Given $\mathtt{t} = \mathtt{t}_1\ \mathtt{t}_2$

$\Gamma \vdash \mathtt{t}_1 : \mathtt{T}_{11} \to \mathtt{T}_{12}$

$\Gamma \vdash \mathtt{t}_2 : \mathtt{T}_{11}$

$\mathtt{T} = \mathtt{T}_{12}$

Show $\Gamma \vdash \mathtt{t}' : \mathtt{T}_{12}$

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

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

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

The "Substitution Lemma"

Lemma: Types are preserved under substitition.

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

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

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

Uh oh.

The "Substitution Lemma"

Lemma: Types are preserved under substitition.

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

Proof: ...

On to real programming languages...

The Unit type

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

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

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

Sequencing

t	::=	...	*terms*
		$\texttt{t}_1 \texttt{;} \texttt{t}_2$	

Derived forms

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

Equivalence of the two definitions

[board]

Sequencing

$\texttt{t} \quad ::= \quad \ldots$ *terms*

$\qquad \texttt{t}_1\texttt{;t}_2$

$$\frac{\texttt{t}_1 \longrightarrow \texttt{t}_1'}{\texttt{t}_1\texttt{;t}_2 \longrightarrow \texttt{t}_1'\texttt{;t}_2} \quad \text{(E-Seq)}$$

$$\texttt{unit;t}_2 \longrightarrow \texttt{t}_2 \quad \text{(E-SeqNext)}$$

$$\frac{\Gamma \vdash \texttt{t}_1 : \texttt{Unit} \qquad \Gamma \vdash \texttt{t}_2 : \texttt{T}_2}{\Gamma \vdash \texttt{t}_1\texttt{;t}_2 : \texttt{T}_2} \quad \text{(T-Seq)}$$

Sequencing as a derived form

$$\texttt{t}_1\texttt{;t}_2 \stackrel{\text{def}}{=} (\lambda\texttt{x:Unit.t}_2)\ \texttt{t}_1$$

$$\text{where } \texttt{x} \notin FV(\texttt{t}_2)$$

Ascription as a derived form

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

Pairs

$\texttt{t}$	$::=$	...	*terms*
		$\{\texttt{t},\texttt{t}\}$	*pair*
		$\texttt{t.1}$	*first projection*
		$\texttt{t.2}$	*second projection*
$\texttt{v}$	$::=$	...	*values*
		$\{\texttt{v},\texttt{v}\}$	*pair value*
$\texttt{T}$	$::=$	...	*types*
		$\texttt{T}_1 \times \texttt{T}_2$	*product type*

Ascription

New syntactic forms

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

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

$$\texttt{v}_1\ \texttt{as T} \longrightarrow \texttt{v}_1 \qquad \text{(E-Ascribe)}$$

$$\frac{\texttt{t}_1 \longrightarrow \texttt{t}_1'}{\texttt{t}_1\ \texttt{as T} \longrightarrow \texttt{t}_1'\ \texttt{as T}} \qquad \text{(E-Ascribe1)}$$

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

$$\frac{\Gamma \vdash \texttt{t}_1 : \texttt{T}}{\Gamma \vdash \texttt{t}_1\ \texttt{as T} : \texttt{T}} \qquad \text{(T-Ascribe)}$$

Let-bindings

New syntactic forms

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

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

$$\texttt{let x=v}_1\ \texttt{in t}_2 \longrightarrow [\texttt{x} \mapsto \texttt{v}_1]\texttt{t}_2 \qquad \text{(E-LetV)}$$

$$\frac{\texttt{t}_1 \longrightarrow \texttt{t}_1'}{\texttt{let x=t}_1\ \texttt{in t}_2 \longrightarrow \texttt{let x=t}_1'\ \texttt{in t}_2} \qquad \text{(E-Let)}$$

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

$$\frac{\Gamma \vdash \texttt{t}_1 : \texttt{T}_1 \qquad \Gamma, \texttt{x:T}_1 \vdash \texttt{t}_2 : \texttt{T}_2}{\Gamma \vdash \texttt{let x=t}_1\ \texttt{in t}_2 : \texttt{T}_2} \qquad \text{(T-Let)}$$

Typing rules for pairs

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

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

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

Evaluation rules for tuples

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

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

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

Evaluation rules for pairs

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

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

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

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

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

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

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*

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

Evaluation rules for records

$$\{l_i\texttt{=}v_i\ ^{i\in 1..n}\}.l_j \longrightarrow v_j \qquad \text{(E-ProjRcd)}$$

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

$$\frac{t_j \longrightarrow t_j'}{\begin{array}{l}\{l_i\texttt{=}v_i\ ^{i\in 1..j-1}, l_j\texttt{=}t_j, l_k\texttt{=}t_k\ ^{k\in j+1..n}\} \\ \longrightarrow \{l_i\texttt{=}v_i\ ^{i\in 1..j-1}, l_j\texttt{=}t_j', l_k\texttt{=}t_k\ ^{k\in j+1..n}\}\end{array}} \qquad \text{(E-Rcd)}$$

Typing rules for records

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

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

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.

What typing rules are introduction forms? What are elimination forms?

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 $\mathbf{P} \supset \mathbf{Q}$	type `P→Q`
proposition $\mathbf{P} \wedge \mathbf{Q}$	type `P × Q`
proof of proposition $\mathbf{P}$	term `t` of type `P`
proposition $\mathbf{P}$ is provable	type `P` is inhabited (by some term)

Propositions as Types

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

Typability

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

Cf. type reconstruction in OCaml.

Propositions as Types

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

Erasure

$$
\begin{array}{lcl}
\mathit{erase}(\mathtt{x}) & = & \mathtt{x} \\
\mathit{erase}(\lambda\mathtt{x}{:}\mathtt{T_1}.\ \mathtt{t_2}) & = & \lambda\mathtt{x}.\ \mathit{erase}(\mathtt{t_2}) \\
\mathit{erase}(\mathtt{t_1}\ \mathtt{t_2}) & = & \mathit{erase}(\mathtt{t_1})\ \mathit{erase}(\mathtt{t_2})
\end{array}
$$