CIS 500

Software Foundations

Fall 2002

16 October

Administrivia

- Exams will be graded over the weekend
- HW5??

Simply typed lambda-calculus with booleans

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

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

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

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

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

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

The Substitution Lemma

[board]

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 P must provide evidence for P.

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

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

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

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)

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
(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 $\mathtt{m}$ 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 $\mathsf{erase}(\mathtt{t}) = \mathtt{m}$ and $\Gamma \vdash \mathtt{t} : \mathtt{T}$.

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

$(\lambda f{:}S.\ \lambda g{:}T.\ f\ g)\ (\lambda x{:}B.\ x)$

is well typed.

The `Unit` type

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

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

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

Sequencing

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

Sequencing

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

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

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

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

Derived forms

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

Sequencing as a derived form

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

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

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

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 $\boxed{\Gamma \vdash \texttt{t : T}}$

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

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

Evaluation rules for pairs

$$\{\texttt{v}_1,\texttt{v}_2\}.1 \longrightarrow \texttt{v}_1 \qquad \text{(E-PairBeta1)}$$

$$\{\texttt{v}_1,\texttt{v}_2\}.2 \longrightarrow \texttt{v}_2 \qquad \text{(E-PairBeta2)}$$

$$\frac{\texttt{t}_1 \longrightarrow \texttt{t}_1'}{\texttt{t}_1.1 \longrightarrow \texttt{t}_1'.1} \qquad \text{(E-Proj1)}$$

$$\frac{\texttt{t}_1 \longrightarrow \texttt{t}_1'}{\texttt{t}_1.2 \longrightarrow \texttt{t}_1'.2} \qquad \text{(E-Proj2)}$$

$$\frac{\texttt{t}_1 \longrightarrow \texttt{t}_1'}{\{\texttt{t}_1,\texttt{t}_2\} \longrightarrow \{\texttt{t}_1',\texttt{t}_2\}} \qquad \text{(E-Pair1)}$$

$$\frac{\texttt{t}_2 \longrightarrow \texttt{t}_2'}{\{\texttt{v}_1,\texttt{t}_2\} \longrightarrow \{\texttt{v}_1,\texttt{t}_2'\}} \qquad \text{(E-Pair2)}$$

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

$\mathtt{t}$	::=	...	terms
		$\{\mathtt{l}_i\mathtt{=}\mathtt{t}_i\ ^{i\in 1..n}\}$	record
		$\mathtt{t.l}$	projection
$\mathtt{v}$	::=	...	values
		$\{\mathtt{l}_i\mathtt{=}\mathtt{v}_i\ ^{i\in 1..n}\}$	record value
$\mathtt{T}$	::=	...	types
		$\{\mathtt{l}_i\mathtt{:}\mathtt{T}_i\ ^{i\in 1..n}\}$	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

$$\frac{\text{for each } i \quad \Gamma \vdash \mathtt{t}_i : \mathtt{T}_i}{\Gamma \vdash \{\mathtt{l}_i\mathtt{=}\mathtt{t}_i\ ^{i\in 1..n}\} : \{\mathtt{l}_i\mathtt{:}\mathtt{T}_i\ ^{i\in 1..n}\}} \qquad \text{(T-Rcd)}$$

$$\frac{\Gamma \vdash \mathtt{t}_1 : \{\mathtt{l}_i\mathtt{:}\mathtt{T}_i\ ^{i\in 1..n}\}}{\Gamma \vdash \mathtt{t}_1\mathtt{.l}_j : \mathtt{T}_j} \qquad \text{(T-Proj)}$$