CIS 500

Software Foundations

Fall 2005

26 October, 2005

Simply-Typed Lambda Calculus

Last time

Last time we talked about proving type soundness for the simply-typed lambda calculus with booleans. This involved showing:

- Canonical Forms
- Inversion Lemmas
- Progress Theorem
- Preservation Theorem

Preservation

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

We got to the point where we needed a substitution lemma to show the preservation theorem:

> If `x:S` $\vdash$ `t : T` and $\vdash$ `s : S`, then $\vdash$ [`x` $\mapsto$ `s`]`t : T`.

Typing rules

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

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

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

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

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

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

Strengthened induction hypothesis

Lemma: Types are preserved under substitition.

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

Lemmas about the context

Lemma [Permutation]: If $\Gamma \vdash \mathtt{t} : \mathtt{T}$ and Δ is a permutation of Γ then $\Delta \vdash \mathtt{t} : \mathtt{T}$.

Lemma [Weakening]: If $\Gamma \vdash \mathtt{t} : \mathtt{T}$ and $\mathtt{x} \notin \mathit{dom}(\Gamma)$ then $\Gamma, \mathtt{x} : \mathtt{S} \vdash \mathtt{t} : \mathtt{T}$.

Important points about type soundness for STLC

- Typing rules need a context to deal with variables.
- Progress lemma must have an empty context to be true.
- Preservation lemma need not (see TAPL).
- Application case of preservation lemma needs substitution lemma.
- Substitution lemma must be strenthened to non-empty contexts.
- Substitution lemma cannot be proved by induction on typing derivation.

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*
		$t_1 ; t_2$	

Sequencing

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

$\quad\quad\quad \mathtt{t}_1\,;\mathtt{t}_2$

$$\frac{\mathtt{t}_1 \longrightarrow \mathtt{t}_1'}{\mathtt{t}_1\,;\mathtt{t}_2 \longrightarrow \mathtt{t}_1'\,;\mathtt{t}_2} \qquad \text{(E-Seq)}$$

$$\mathtt{unit};\mathtt{t}_2 \longrightarrow \mathtt{t}_2 \qquad \text{(E-SeqNext)}$$

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

Derived forms

One way to extend a programming language is through derived forms.

A derived form is a new syntax, operational semantics $\mathtt{t} \longrightarrow_{\mathtt{E}} \mathtt{t'}$ and type system $\Gamma \vdash^{\mathtt{E}} \mathtt{t} : \mathtt{T}$ for a language such that there is a function e that translates the new language into the old language with the following properties:

- $\mathtt{t} \longrightarrow_{\mathtt{E}} \mathtt{t'}$ iff $e(\mathtt{t}) \longrightarrow e(\mathtt{t'})$.
- $\Gamma \vdash \mathtt{t} : \mathtt{T}$ iff $\Gamma \vdash e(\mathtt{t}) : \mathtt{T}$.

Derived forms are also called syntactic sugar.

Sequencing as a derived form

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

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

Equivalence of the two definitions

[board]

Let-bindings

New syntactic forms

`t`	::=	...	*terms*
		`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 $\boxed{\Gamma \vdash \texttt{t} : \texttt{T}}$

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

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

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

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

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

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

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

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

Typing rules for pairs

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

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

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

Tuples

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

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

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

Records

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

Evaluation rules for records

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

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

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

Typing rules for records

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

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

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?

Erasure

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

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.