CIS 500

Software Foundations

Fall 2005

7 November

Announcements

- Midterm II is one week from Wednesday (November 16).
  - It will cover TAPL chapters 8-14 (except 12).
  - Recitations this week will be review for midterm.
  - No in class review.
- Homework 6 due today.
- Homework 7 out today, due November 14.

References

Another example

```
BoolArray = Ref (Nat→Bool);

newarray = λ_:Unit. ref (λn:Nat.false);
           : Unit → BoolArray

lookup = λa:BoolArray. λn:Nat. (!a) n;
       : BoolArray → Nat → Bool

update = λa:BoolArray. λm:Nat. λv:Bool.
           let oldf = !a in
           a := (λn:Nat. if equal m n then v else oldf n);
       : BoolArray → Nat → Bool → Unit


let a = newarray () in
print (lookup a 3);
update a 3 true;
lookup a 3
```

Syntax

`t` ::=		*terms*
	`unit`	*unit constant*
	`x`	*variable*
	`λx:T.t`	*abstraction*
	`t t`	*application*
	`ref t`	*reference creation*
	`!t`	*dereference*
	`t:=t`	*assignment*
	`l`	*store location*
`v` ::=		*values*
	`unit`	*unit constant*
	`λx:T.t`	*abstraction value*
	`l`	*store location*

Evaluation

An assignment $t_1 := t_2$ first evaluates t_1 and t_2 until they become values...

$$\frac{t_1 \mid \mu \longrightarrow t_1' \mid \mu'}{t_1 := t_2 \mid \mu \longrightarrow t_1' := t_2 \mid \mu'} \qquad \text{(E-Assign1)}$$

$$\frac{t_2 \mid \mu \longrightarrow t_2' \mid \mu'}{v_1 := t_2 \mid \mu \longrightarrow v_1 := t_2' \mid \mu'} \qquad \text{(E-Assign2)}$$

... and then returns unit and updates the store:

$$l := v_2 \mid \mu \longrightarrow \texttt{unit} \mid [l \mapsto v_2]\mu \qquad \text{(E-Assign)}$$

A term of the form $\texttt{ref}\ \texttt{t}_1$ first evaluates inside $\texttt{t}_1$ until it becomes a value...

$$\frac{\texttt{t}_1 \mid \mu \longrightarrow \texttt{t}_1' \mid \mu'}{\texttt{ref}\ \texttt{t}_1 \mid \mu \longrightarrow \texttt{ref}\ \texttt{t}_1' \mid \mu'} \quad \text{(E-Ref)}$$

... and then chooses (allocates) a fresh location $\mathbf{l}$, augments the store with a binding from $\mathbf{l}$ to $\texttt{v}_1$, and returns $\mathbf{l}$:

$$\frac{\mathbf{l} \notin dom(\mu)}{\texttt{ref}\ \texttt{v}_1 \mid \mu \longrightarrow \mathbf{l} \mid (\mu, \mathbf{l} \mapsto \texttt{v}_1)} \quad \text{(E-RefV)}$$

A term $!t_1$ first evaluates in t_1 until it becomes a value...

$$\frac{t_1 \mid \mu \longrightarrow t_1' \mid \mu'}{!t_1 \mid \mu \longrightarrow !t_1' \mid \mu'} \qquad \text{(E-Deref)}$$

... and then looks up this value (which must be a location, if the original term was well typed) and returns its contents in the current store:

$$\frac{\mu(l) = v}{!l \mid \mu \longrightarrow v \mid \mu} \qquad \text{(E-DerefLoc)}$$

Evaluation rules for function abstraction and application are augmented with stores, but don't do anything with them directly.

$$\frac{\mathtt{t}_1 \mid \mu \longrightarrow \mathtt{t}_1' \mid \mu'}{\mathtt{t}_1\ \mathtt{t}_2 \mid \mu \longrightarrow \mathtt{t}_1'\ \mathtt{t}_2 \mid \mu'} \qquad \text{(E-App1)}$$

$$\frac{\mathtt{t}_2 \mid \mu \longrightarrow \mathtt{t}_2' \mid \mu'}{\mathtt{v}_1\ \mathtt{t}_2 \mid \mu \longrightarrow \mathtt{v}_1\ \mathtt{t}_2' \mid \mu'} \qquad \text{(E-App2)}$$

$$(\lambda \mathtt{x}{:}\mathtt{T}_{11}.\mathtt{t}_{12})\ \mathtt{v}_2 \mid \mu \longrightarrow [\mathtt{x} \mapsto \mathtt{v}_2]\mathtt{t}_{12} \mid \mu \qquad \text{(E-AppAbs)}$$

Store Typings

Typing Locations

Q: What is the type of a location?

Typing Locations

Q: What is the type of a location?

A: It depends on the store!

E.g., in the store $(l_1 \mapsto \texttt{unit}, l_2 \mapsto \texttt{unit})$, the term $!l_2$ has type `Unit`.

But in the store $(l_1 \mapsto \texttt{unit}, l_2 \mapsto \lambda x{:}\texttt{Unit}.x)$, the term $!l_2$ has type $\texttt{Unit}\to\texttt{Unit}$.

Typing Locations — first try

Roughly:

$$\frac{\Gamma \vdash \mu(l) \ : \mathtt{T_1}}{\Gamma \vdash l : \mathtt{Ref}\ \mathtt{T_1}}$$

Typing Locations — first try

Roughly:

$$\frac{\Gamma \vdash \mu(l) : T_1}{\Gamma \vdash l : \texttt{Ref } T_1}$$

More precisely:

$$\frac{\Gamma \mid \mu \vdash \mu(l) : T_1}{\Gamma \mid \mu \vdash l : \texttt{Ref } T_1}$$

I.e., typing is now a four-place relation (between contexts, stores, terms, and types).

Problem

However, this rule is not completely satisfactory. For one thing, it can make typing derivations very large!

E.g., if

$$
\begin{aligned}
(\mu = \; & l_1 \mapsto \lambda \texttt{x:Nat. 999}, \\
& l_2 \mapsto \lambda \texttt{x:Nat. } !l_1 \; (!l_1 \; \texttt{x}), \\
& l_3 \mapsto \lambda \texttt{x:Nat. } !l_2 \; (!l_2 \; \texttt{x}), \\
& l_4 \mapsto \lambda \texttt{x:Nat. } !l_3 \; (!l_3 \; \texttt{x}), \\
& l_5 \mapsto \lambda \texttt{x:Nat. } !l_4 \; (!l_4 \; \texttt{x})),
\end{aligned}
$$

then how big is the typing derivation for $!l_5$?

Problem!

But wait... it gets worse. Suppose

$$\begin{aligned}(\mu = {} & \mathtt{l_1} \mapsto \lambda\mathtt{x{:}Nat.\ !l_2\ x},\\ & \mathtt{l_2} \mapsto \lambda\mathtt{x{:}Nat.\ !l_1\ x}),\end{aligned}$$

Now how big is the typing derivation for $\mathtt{!l_2}$?

Store Typings

Observation: The typing rules we have chosen for references guarantee that a given location in the store is always used to hold values of the same type.

These intended types can be collected into a store typing — a partial function from locations to types.

E.g., for

$$\begin{aligned}\boldsymbol{\mu} = (\mathbf{l}_1 &\mapsto \lambda\texttt{x:Nat. 999},\\ \mathbf{l}_2 &\mapsto \lambda\texttt{x:Nat. !}\mathbf{l}_1\texttt{ (!}\mathbf{l}_1\texttt{ x)},\\ \mathbf{l}_3 &\mapsto \lambda\texttt{x:Nat. !}\mathbf{l}_2\texttt{ (!}\mathbf{l}_2\texttt{ x)},\\ \mathbf{l}_4 &\mapsto \lambda\texttt{x:Nat. !}\mathbf{l}_3\texttt{ (!}\mathbf{l}_3\texttt{ x)},\\ \mathbf{l}_5 &\mapsto \lambda\texttt{x:Nat. !}\mathbf{l}_4\texttt{ (!}\mathbf{l}_4\texttt{ x)}),\end{aligned}$$

A reasonable store typing would be

$$\begin{aligned}\boldsymbol{\Sigma} = (\mathbf{l}_1 &\mapsto \texttt{Nat}\to\texttt{Nat},\\ \mathbf{l}_2 &\mapsto \texttt{Nat}\to\texttt{Nat},\\ \mathbf{l}_3 &\mapsto \texttt{Nat}\to\texttt{Nat},\\ \mathbf{l}_4 &\mapsto \texttt{Nat}\to\texttt{Nat},\\ \mathbf{l}_5 &\mapsto \texttt{Nat}\to\texttt{Nat})\end{aligned}$$

Now, suppose we are given a store typing Σ describing the store μ in which we intend to evaluate some term t. Then we can use Σ to look up the types of locations in t instead of calculating them from the values in μ.

$$\frac{\Sigma(l) = T_1}{\Gamma \mid \Sigma \vdash l : \texttt{Ref}\ T_1} \qquad \text{(T-Loc)}$$

I.e., typing is now a four-place relation between between contexts, store typings, terms, and types.

Final typing rules

$$\frac{\Sigma(l) = T_1}{\Gamma \mid \Sigma \vdash l : \texttt{Ref}\ T_1} \qquad \text{(T-Loc)}$$

$$\frac{\Gamma \mid \Sigma \vdash t_1 : T_1}{\Gamma \mid \Sigma \vdash \texttt{ref}\ t_1 : \texttt{Ref}\ T_1} \qquad \text{(T-Ref)}$$

$$\frac{\Gamma \mid \Sigma \vdash t_1 : \texttt{Ref}\ T_{11}}{\Gamma \mid \Sigma \vdash\ !t_1 : T_{11}} \qquad \text{(T-Deref)}$$

$$\frac{\Gamma \mid \Sigma \vdash t_1 : \texttt{Ref}\ T_{11} \qquad \Gamma \mid \Sigma \vdash t_2 : T_{11}}{\Gamma \mid \Sigma \vdash t_1 := t_2 : \texttt{Unit}} \qquad \text{(T-Assign)}$$

Q: Where do these store typings come from?

Q: Where do these store typings come from?

A: When we first typecheck a program, there will be no explicit locations, so we can use an empty store typing.

So, when a new location is created during evaluation,

$$\frac{\mathtt{l} \notin dom(\mu)}{\mathtt{ref}\ \mathtt{v_1} \mid \mu \longrightarrow \mathtt{l} \mid (\mu, \mathtt{l} \mapsto \mathtt{v_1})} \qquad \text{(E-RefV)}$$

we can observe the type of $\mathtt{v_1}$ and extend the "current store typing" appropriately.

Proving type safety

Stating the presevation theorem is a little trickier now. What is wrong with this statement of preservation?

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

Proving type safety

Stating the presevation theorem is a little trickier now. What is wrong with this statement of preservation?

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

We need to talk about how stores can be typed! There is no connection between Σ and μ.

Store typing

A store μ is said to be well-typed with respect to a typing context Γ and a store typing Σ, written $\Gamma \mid \Sigma \vdash \mu$, if

$$dom(\mu) = dom(\Sigma) \text{ and } \Gamma \mid \Sigma \vdash \mu(l) : \Sigma(l) \text{ for every } l \in dom(\mu)$$

Preservation theorem, second try

What is wrong with this statement of the preservation theorem?

If $\Gamma \mid \Sigma \vdash \mathtt{t} : \mathtt{T}$ and $\Gamma \mid \Sigma \vdash \mu$ and $\mathtt{t} \mid \mu \longrightarrow \mathtt{t}' \mid \mu'$ then

$$\Gamma \mid \Sigma \vdash \mathtt{t}' : \mathtt{T}$$

Preservation theorem

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

$$\text{for some } \Sigma' \supseteq \Sigma, \Gamma \mid \Sigma' \vdash \mathtt{t}' : \mathtt{T}$$

New lemmas for preservation

Substitution for stores:

If $\Gamma \mid \Sigma \vdash \mu$ and $\Sigma(l) = \mathtt{T}$ and $\Gamma \mid \Sigma \vdash \mathtt{v} : \mathtt{T}$ then

$$\Gamma \mid \Sigma \vdash [l \mapsto v]\mu$$

New lemmas for preservation

Substitution for stores:

If $\Gamma \mid \Sigma \vdash \mu$ and $\Sigma(l) = \mathtt{T}$ and $\Gamma \mid \Sigma \vdash \mathtt{v} : \mathtt{T}$ then

$$\Gamma \mid \Sigma \vdash [l \mapsto v]\mu$$

Weakening for stores:

If $\Gamma \mid \Sigma \vdash \mathtt{t} : \mathtt{T}$ and $\Sigma' \supseteq \Sigma$, then

$$\Gamma \mid \Sigma' \vdash \mathtt{t} : \mathtt{T}$$

Progress theorem

Suppose that $\emptyset \mid \Sigma \vdash \mathtt{t} : \mathtt{T}$ then either

1. $\mathtt{t}$ is a value, or else
2. for any store μ such that $\emptyset \mid \Sigma \vdash \mu$, there is some $\mathtt{t}'$ and store μ' with $\mathtt{t} \mid \mu \longrightarrow \mathtt{t}' \mid \mu'$.

Progress theorem

Suppose that $\emptyset \mid \Sigma \vdash t : T$ then either

1. t is a value, or else
2. for any store μ such that $\emptyset \mid \Sigma \vdash \mu$, there is some t' and store μ' with $t \mid \mu \longrightarrow t' \mid \mu'$.

Why isn't Σ required to be empty?

Safety

If $\emptyset \mid \emptyset \vdash \mathtt{t} : \mathtt{T}$ and $\mathtt{t} \mid \emptyset \longrightarrow^{*} \mathtt{t'} \mid \mu$ and $\mathtt{t'} \mid \mu \not\longrightarrow$ then $\mathtt{t}$ is a value.