CS611 Lecture 29 — Soundness of Typing Rules — 7 November 2001

Scribe: Stephen Chong and Prakash Linga — Lecturer: Andrew Myers

1 Recapitulation of Static Semantics of $\lambda^{\rightarrow}$

$$\begin{array}{rcl} \vdash e : \tau & \text{means} & \text{"}e\text{ is a well-formed program with type }\tau\text{"} \\ \Gamma \vdash e : \tau & \text{means} & \text{"}e\text{ is a well-formed program with type }\tau\text{ in typing context }\Gamma\text{"} \\ \vdash e : \tau & \overset{\text{def}}{\Longleftrightarrow} & \emptyset \vdash e : \tau \end{array}$$

1.1 Recasting the Typing Context Γ

In the last lecture, we regarded Γ as being a partial function from *Var* to *Type*: $\Gamma \in \mathit{Var} \rightharpoonup \mathit{Type}$.

This time, we will regard Γ as a syntactic object, defined by the formation rules:

$$\Gamma ::= \emptyset \mid \Gamma, x\!:\!\tau$$

or, alternatively, as being defined by the following axiom and inference rule:

$$\frac{}{\vdash \emptyset} \qquad\qquad \frac{\vdash \Gamma}{\vdash \Gamma, x\!:\!\tau}$$

where $\vdash \Gamma$ means "Γ is a well-formed context." For notational convenience, a typing context $\emptyset, x_1 : \tau_1, \ldots, x_n\!:\!\tau_n$ can be written as $x_1\!:\!\tau_1, \ldots, x_n\!:\!\tau_n$.

Order within a typing context Γ is not important since we want them to be isomorphic to partial functions. We formalize the unimportance of order by using an equivalence relation $\approx$ on typing contexts, and the following inference rule that allows us to substitute equivalent contexts when constructing a typing derivation:

$$\frac{\Gamma \vdash e : \tau \quad \vdash \Gamma \approx \Gamma'}{\Gamma' \vdash e\!:\!\tau} \text{ [ctxt-subst]}$$

The equivalence relation $\approx$ on typing contexts is defined by the following rules, plus the usual rules that ensure the relation is reflexive, symmetric, and transitive:

$$\frac{}{\vdash \Gamma, x\!:\!\tau, x\!:\!\tau' \approx \Gamma, x\!:\!\tau'} \text{ [update]}$$

$$\frac{}{\vdash \Gamma, x\!:\!\tau, x'\!:\!\tau' \approx \Gamma, x'\!:\!\tau', x\!:\!\tau} \ (x \neq x') \text{ [exchange]}$$

$$\frac{\vdash \Gamma \approx \Gamma'}{\vdash \Gamma, x\!:\!\tau \approx \Gamma', x\!:\!\tau} \text{ [extend]}$$

Treating the typing context as a syntactic object, instead of as a partial function, gives us more flexibility in defining language semantics. For example, we could ensure that no variable could be declared in the scope of another variable with the same name. Making Γ purely syntactic also makes our proofs a little more mechanical, which is good.

1.2 Static Semantics

Using the typing contexts just defined, we can express the static semantics similarly to the previous lecture; but with explicit well formedness requirements on the typing context.

$$\frac{\vdash \Gamma}{\Gamma, x\!:\!\tau \vdash x : \tau}\ [\text{var}]$$

$$\frac{\Gamma \vdash e_0 : \tau \to \tau' \quad \Gamma \vdash e_1 : \tau \quad \vdash \Gamma}{\Gamma \vdash e_0\ e_1 : \tau'}\ [\text{app}]$$

$$\frac{\Gamma, x\!:\!\tau \vdash e : \tau' \quad \vdash \Gamma}{\Gamma \vdash (\lambda x\!:\!\tau.\, e) : \tau \to \tau'}\ [\text{abs}]$$

2 Denotational Semantics

$$\begin{array}{rcl}
\mathcal{T}[\![1]\!] & = & \mathbb{U} \\
\mathcal{T}[\![\mathsf{int}]\!] & = & \mathbb{Z} \\
\mathcal{T}[\![\mathsf{bool}]\!] & = & \mathbb{B} \\
\mathcal{T}[\![\tau_0 \to \tau_1]\!] & = & \mathcal{T}[\![\tau_0]\!] \to \mathcal{T}[\![\tau_1]\!]
\end{array}$$

$$\mathcal{C}[\![\Gamma \vdash e : \tau]\!]\rho \quad \in \quad \mathcal{T}[\![\tau]\!] \text{ if } \rho \models \Gamma$$

where

$$\rho \models \Gamma \overset{\text{def}}{\Longleftrightarrow} \forall x, \tau\ .\ \Gamma \vdash x : \tau \Rightarrow \rho(x) \in \mathcal{T}[\![\tau]\!]$$

The denotational semantics are defined by induction on typing derivations:

$$\begin{array}{rcl}
\mathcal{C}[\![\Gamma \vdash \#\mathsf{u} : 1]\!]\rho & = & u \\
\mathcal{C}[\![\Gamma \vdash \#\mathsf{f} : \mathsf{bool}]\!]\rho & = & \mathit{true} \\
\mathcal{C}[\![\Gamma \vdash \#\mathsf{t} : \mathsf{bool}]\!]\rho & = & \mathit{false} \\
\mathcal{C}[\![\Gamma \vdash n : \mathsf{int}]\!]\rho & = & n \\
\mathcal{C}[\![\Gamma \vdash x : \tau]\!]\rho & = & \rho(x) \\
\mathcal{C}[\![\Gamma \vdash e_0\ e_1 : \tau']\!]\rho & = & (\mathcal{C}[\![\Gamma \vdash e_0 : \tau \to \tau']\!]\rho)(\mathcal{C}[\![\Gamma \vdash e_1 : \tau]\!]\rho) \\
\mathcal{C}[\![\Gamma \vdash (\lambda x\!:\!\tau.\, e) : \tau \to \tau']\!]\rho & = & \lambda v \in \mathcal{T}[\![\tau]\!]\,.\,\mathcal{C}[\![\Gamma, x\!:\!\tau \vdash e : \tau']\!]\rho[x \mapsto v]
\end{array}$$

It is necessary to show that $\rho \models \Gamma \Rightarrow \rho[x \mapsto v] \models \Gamma, x\!:\!\tau$ for the last rule to be correct. This is left as an exercise for the reader.

2.1 Example of denotational semantics

$$\frac{\overline{x\!:\!\mathsf{int} \vdash x : \mathsf{int}}}{\emptyset \vdash (\lambda x : \mathsf{int}.\, x) : \mathsf{int} \to \mathsf{int}}$$

$$\begin{array}{rcl}
\mathcal{C}[\![\emptyset \vdash (\lambda x\!:\!\mathsf{int}.\, x) : \mathsf{int} \to \mathsf{int}]\!]\emptyset & = & \lambda v \in \mathcal{T}[\![\mathsf{int}]\!].\, \mathcal{C}[\![x\!:\!\mathsf{int} \vdash x : \mathsf{int}]\!]\{x \mapsto v\} \\
 & = & \lambda v \in \mathbb{Z}.\, v
\end{array}$$

2.2 A Type for $\lambda x.\,(x\ x)$?

Suppose that x is of type τ, and the term itself is of type $\tau \to \tau'$. Consider the derivation tree for this term:

$$\dfrac{\dfrac{x\!:\!\tau \vdash x : \tau'' \to \tau' \quad x\!:\!\tau \vdash x : \tau''}{x\!:\!\tau \vdash (x\ x) : \tau'}}{\emptyset \vdash (\lambda x\!:\!\tau.\,(x\ x)) : \tau \to \tau'}$$

From this derivation, we can see that τ'' must be equal to τ. That is, we need a τ such that $\tau \equiv \tau \to \tau'$. However, this is not possible, so we cannot assign a type to this term.

We have lost some expressive power, although later in the course we will see constructions that allow us to assign a type to this expression.

3 Soundness from the Operational Perspective

We will now look at the soundness of the typing rules in the operational perspective. What this means is,

$$\textit{Typing rules are sound} \iff \textit{no well formed program gets stuck.}$$

Note that we use well-formed and well-typed equivalently for this language. To be more precise,

$$\vdash e : \tau \ \wedge\ e \longmapsto^* e' \Rightarrow (e' \in \mathit{Value} \vee \exists e''.e' \longmapsto e'')$$

To show this we will use a 2-step approach. We will show the following two lemmas:

- *Preservation*: As we evaluate a program its type is preserved at each step.

$$\vdash e : \tau \ \wedge\ e \longmapsto e' \Rightarrow \ \vdash e' : \tau$$

- *Progress*: Every program is either a value or can be stepped into another program (and using *preservation* lemma this will be of the same type!)

$$\vdash e : \tau \Rightarrow e = v \in \mathit{Value} \vee \exists e''.e \longmapsto e''$$

Note that with these two lemmas soundness easily follows. We use induction on the number of steps taken in $e \longmapsto^* e'$ to show that e' must have the same type as e; the *progress* lemma can then be applied to e! We will now set out to prove these lemmas.

3.1 Proof of *Preservation* lemma

Assuming $e : \tau \wedge e \longmapsto e'$ we need to show that $\vdash e' : \tau$.

We will do this by well-founded induction on typing derivations. (Note that the typing derivations are finite and therefore the relation of subderivation is well-founded.) The property we are trying to show on typing derivations is:

$$P(\vdash e : \tau) \iff \forall e'.e \longmapsto e' \Rightarrow \vdash e' : \tau$$

We define the following for convenience:

$$e_0 \equiv (\lambda x\!:\!\tau.\ e_1)\ v \qquad e_0' \equiv e_1\{v/x\}$$

Then we have $e \equiv C[e_0]$ and $e' \equiv C[e_0']$. Note that because the typing context is $\emptyset$, the [ctxt-subst] rule cannot be useful in proving $\vdash e : \tau$. So we can assume that the proof uses one of the other rules.

Now consider the possible forms of e. There are three cases to consider-

- Case 1: $C = C' \ e_2$.

  Then $e = C[e_0] = C'[e_0] \ e_2$. Using β-reduction $C'[e_0] \longmapsto C'[e_0']$. Using the rule for application, for some τ_2,

$$\frac{\vdash C'[e_0] : \tau_2 \to \tau \quad \vdash e_2 : \tau_2}{\vdash C[e_0] : \tau}$$

  Now $\vdash C'[e_0] : \tau_2 \to \tau$ has a smaller derivation than $\vdash e : \tau$ and therefore using the induction hypothesis implies $\vdash C'[e_0'] : \tau_2 \to \tau$. So we can use:

$$\frac{\vdash C'[e_0'] : \tau_2 \to \tau \quad \vdash e_2 : \tau_2}{\vdash C[e_0'] : \tau}$$

  Therefore $P(\vdash e : \tau)$ holds in this case.

- Case 2: $C = v \ C'$

  Similar to Case 1.

- Case 3: $C = [\,]$

  Now $e \equiv e_0$ and $e' \equiv e_0'$. The typing derivation of $\vdash e : \tau$ must look like this:

$$\frac{\dfrac{x : \tau' \vdash e_2 : \tau}{\vdash (\lambda x : \tau'.\, e_2) : \tau' \to \tau} \quad \vdash v : \tau'}{\vdash (\lambda x : \tau'.\, e_2) \ v : \tau}$$

  We need to show that $\vdash e_2\{v/x\} : \tau$ using $x : \tau' \vdash e_2 : \tau$ and $\vdash v : \tau'$. This follows as a special case of the *Substitution* lemma which we will prove below.

3.2 *Substitution* lemma

$$\Gamma, x : \tau' \vdash e : \tau \ \wedge \ \vdash v : \tau' \Rightarrow \Gamma \vdash e\{v/x\} : \tau$$

We will prove this by induction on the typing derivation of e.

- Case 1: $e = b$. In this case the result trivially holds.
- Case 2
  - Case 2a: $e = x$. Then $e\{v/x\} = v$ and $\tau' = \tau$ and hence the result holds in this case.
  - Case 2b: $e = y \neq x$. In this case $e\{v/x\} = e$ and therefore the result trivially holds.
- Case 3: $e = e_0 \ e_1$. Using the definition of substitution, $e\{v/x\} = e_0\{v/x\} \ e_1\{v/x\}$.

  Using the derivation of $\Gamma, x : \tau' \vdash e : \tau$ we have $\Gamma, x : \tau' \vdash e_0 : \tau_1 \to \tau$ and $\Gamma, x : \tau' \vdash e_1 : \tau_1$:

$$\frac{\Gamma, x : \tau' \vdash e_0 : \tau_1 \to \tau \quad \Gamma, x : \tau' \vdash e_1 : \tau_1}{\Gamma, x : \tau' \vdash e : \tau}$$

  Therefore we can use the induction hypothesis to obtain $\Gamma \vdash e_0\{v/x\} : \tau_1 \to \tau$ and $\Gamma \vdash e_1\{v/x\} : \tau_1$. This therefore implies $\Gamma \vdash e\{v/x\} : \tau$ via the application rule.

- Case 4
  - Case 4a: $e \equiv \lambda x : \tau'' \ e_2$.

    Then $e\{v/x\} = e$ and therefore the result holds trivially.

– Case 4b: $e \equiv \lambda y\!:\!\tau''\ e_2$.

  Note that $y \notin FV[v]$, so $e\{v/x\} = \lambda y : \tau''.\, e_2\{v/x\}$ and the typing of e looks like the following, where $\tau = \tau'' \rightarrow \tau_2$:

$$\frac{\Gamma, x\!:\!\tau', y\!:\!\tau'' \vdash e_2\!:\!\tau_2}{\Gamma, x\!:\!\tau', \vdash (\lambda y\!:\!\tau''.\, e_2)\!:\!\tau'' \rightarrow \tau_2}$$

  Use [exchange] and [app] to derive $\Gamma, y\!:\!\tau'', x\!:\!\tau' \vdash e_2\!:\!\tau_2$ and then use the inductive hypothesis to obtain $\Gamma, y\!:\!\tau'' \vdash e_2\{v/x\}\!:\!\tau_2$. By [abs], therefore, $\Gamma \vdash (\lambda y\!:\!\tau''.\, e_2\{v/x\})\!:\!\tau$ and we are done.

The careful reader will notice that we did not consider typing derivations whose final step is [ctxt-subst]. In order to handle such derivations, we can generalize our substitution lemma slightly, to the following:

$$\vdash \Gamma \approx \Gamma', x\!:\!\tau' \ \wedge\ \Gamma \vdash e : \tau \ \wedge\ \vdash v : \tau' \Rightarrow \Gamma' \vdash e\{v/x\} : \tau$$

This change has no real effect on the proof just given, except that case 2 becomes more complicated. Two lemmas make this case trivial once more, by capturing some essential properties of typing contexts:

$$\Gamma \vdash x : \tau \ \wedge\ \Gamma \vdash x : \tau' \ \Rightarrow\ \tau \equiv \tau'$$

$$\vdash \Gamma \approx \Gamma', x\!:\!\tau' \ \wedge\ \Gamma \vdash y : \tau \ \wedge\ x \not\equiv y \ \Rightarrow\ \Gamma' \vdash y : \tau'$$

The proof of these lemmas is left as an exercise.

3.3 *Progress* Lemma

This will be covered in the next lecture!