CS611 Lecture 33 **Product and Sum Types** 19 November 2003

Scribe: Xin Qi Lecturer: Radu Rugina

1 Properties of Type Systems

1.1 Soundness

A type system is sound if the evaluation of any well-typed expression will not yield errors, i.e., will not get stuck:

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

The formula that expresses the fact that "the evaluation of e gets stuck" is:

$$\exists e' \,.\, e \to^* e' \ \wedge \ e' \notin \mathit{Value} \ \wedge \ \forall e'' \,.\, e' \not\to e''$$

The definition of soundness is the negation of this formula. As mentioned in the previous lecture, soundness follows from the Progress and Preservation properties.

1.2 Strong Normalization

A property stronger that soundness is Strong Normalization, which rules out the case of non-termination. A type system is strong normalizing if the evaluation of every well-typed expression yields a final value:

$$\vdash e : \tau \ \Rightarrow \ \exists v \in \mathit{Value} \,.\, e \to^* v$$

The following is a similar definition using denotational semantics:

$$(\rho \models \Gamma \wedge \Gamma \vdash e : \tau) \ \Rightarrow \ \mathcal{C}[\![e]\!]\rho \in \mathcal{T}[\![\tau]\!]$$

2 From $\lambda^{\to}$ to $\lambda^{\to\times+}$

Now we will further extend the simply-typed lambda calculus $\lambda^{\to}$ to a new calculus with pairs and case constructs $\lambda^{\to\times+}$.

$$\begin{array}{rcll} e & ::= & \ldots & \mid \langle e_0, e_1 \rangle \ \mid \ \mathsf{fst}\ e \ \mid \ \mathsf{snd}\ e \\ & & & \mid \ \mathsf{case}\ e_0\ \mathsf{of}\ (e_1 \mid e_2) \\ & & & \mid \ \mathsf{inl}\ e\ \mathsf{as}\ (\tau_1 + \tau_2) \\ & & & \mid \ \mathsf{inr}\ e\ \mathsf{as}\ (\tau_1 + \tau_2) \end{array}$$

Then we need to extend the definition of values:

$$v ::= p \ \mid \ \lambda x : \tau \,.\, e \ \mid \ \langle v_0, v_1 \rangle \ \mid \ \mathsf{inl}\ v\ \mathsf{as}\ (\tau_1 + \tau_2) \ \mid \ \mathsf{inr}\ v\ \mathsf{as}\ (\tau_1 + \tau_2)$$

and the definition of types:

$$\tau ::= \mathsf{int} \ \mid \ \mathsf{bool} \ \mid \ \mathsf{unit} \ \mid \ \tau_1 \to \tau_2 \ \mid \ \tau_1 \times \tau_2 \ \mid \ \tau_1 + \tau_2$$

We will define the formal semantics for this language, and then provide sound typing rules, which guarantee that the evaluation of each well-typed expression will not get stuck.

2.1 Operational Semantics

We consider the CBV semantics, with left-to-right evaluation of expressions. We will only give the axioms of $\lambda^{\to\times+}$ here and leave the remaining rules as exercises.

- β-reduction:

$$\frac{}{(\lambda x : \tau \,.\, e)e' \to e[x \to e']}$$

- Pairs:

$$\frac{}{\mathsf{fst}\ \langle v, v' \rangle \to v}$$

$$\frac{}{\mathsf{snd}\ \langle v, v' \rangle \to v'}$$

- Case constructs:

$$\frac{}{\mathsf{case}\ (\mathsf{inl}\ v\ \mathsf{as}\ (\tau_1 + \tau_2))\ \mathsf{of}\ (e_1 \mid e_2) \to e_1 v}$$

$$\frac{}{\mathsf{case}\ (\mathsf{inr}\ v\ \mathsf{as}\ (\tau_1 + \tau_2))\ \mathsf{of}\ (e_1 \mid e_2) \to e_2 v}$$

2.2 Denotational Semantics

The corresponding denotations for the expressions that manipulate pairs and case constructs are:

$$\begin{array}{rcl}
\mathcal{C}[\![\langle e_1, e_2 \rangle]\!]\rho & = & (\mathcal{C}[\![e_1]\!]\rho, \mathcal{C}[\![e_2]\!]\rho) \\
\mathcal{C}[\![\mathsf{fst}\ e]\!]\rho & = & \pi_1(\mathcal{C}[\![e]\!]\rho) \\
\mathcal{C}[\![\mathsf{snd}\ e]\!]\rho & = & \pi_2(\mathcal{C}[\![e]\!]\rho)
\end{array}$$

$$\begin{array}{rcl}
\mathcal{C}[\![\mathsf{inl}\ e\ \mathsf{as}\ (\tau_1 + \tau_2)]\!]\rho & = & in_1(\mathcal{C}[\![e]\!]\rho) \\
\mathcal{C}[\![\mathsf{inr}\ e\ \mathsf{as}\ (\tau_1 + \tau_2)]\!]\rho & = & in_2(\mathcal{C}[\![e]\!]\rho) \\
\mathcal{C}[\![\mathsf{case}\ e_0\ \mathsf{of}\ (e_1 \mid e_2)]\!]\rho & = & \mathsf{case}\ \mathcal{C}[\![e_0]\!]\rho\ \mathsf{of}\ \ in_1(x_1) \Rightarrow (\mathcal{C}[\![e_1]\!]\rho)x_1 \\
 & & \qquad\qquad\quad \mid \ \ in_2(x_2) \Rightarrow (\mathcal{C}[\![e_2]\!]\rho)x_2
\end{array}$$

In the above denotations, the signatures of the injection functions in_1 and in_2 are:

$$\begin{array}{rcl}
in_1 & : & \mathcal{T}[\![\tau_1]\!] \to \mathcal{T}[\![\tau_1 + \tau_2]\!] \\
in_2 & : & \mathcal{T}[\![\tau_2]\!] \to \mathcal{T}[\![\tau_1 + \tau_2]\!]
\end{array}$$

where of definitions of the function $\mathcal{T}[\![\cdot]\!]$ which maps types to their domain of values is extended as follows:

$$\begin{array}{rcl}
\mathcal{T}[\![\mathsf{int}]\!] & = & \mathbb{Z} \\
\mathcal{T}[\![\mathsf{bool}]\!] & = & \mathbb{T} \\
\mathcal{T}[\![\mathsf{unit}]\!] & = & \mathbb{U} \\
\mathcal{T}[\![\tau_1 \to \tau_2]\!] & = & \mathcal{T}[\![\tau_1]\!] \to \mathcal{T}[\![\tau_2]\!] \\
\mathcal{T}[\![\tau_1 \times \tau_2]\!] & = & \mathcal{T}[\![\tau_1]\!] \times \mathcal{T}[\![\tau_2]\!] \\
\mathcal{T}[\![\tau_1 + \tau_2]\!] & = & \mathcal{T}[\![\tau_1]\!] + \mathcal{T}[\![\tau_2]\!]
\end{array}$$

2.3 Typing Rules

The typing rules that statically enforce the error-free evaluation of expressions are as follows. These are the rules just for the new expressions; the typing rules for variables, abstractions, and applications are as before.

- Pairs:

$$\frac{\Gamma \vdash e_0 : \tau_0 \quad \Gamma \vdash e_1 : \tau_1}{\Gamma \vdash \langle e_0, e_1 \rangle : \tau_0 \times \tau_1}$$

$$\frac{\Gamma \vdash e : \tau_0 \times \tau_1}{\Gamma \vdash \mathsf{fst}\ e : \tau_0}$$

$$\frac{\Gamma \vdash e : \tau_0 \times \tau_1}{\Gamma \vdash \mathsf{snd}\ e : \tau_1}$$

- Case constructs:

$$\frac{\Gamma \vdash e : \tau_1}{\Gamma \vdash \mathsf{inl}\ e\ \mathsf{as}\ (\tau_1 + \tau_2) : \tau_1 + \tau_2}$$

$$\frac{\Gamma \vdash e : \tau_2}{\Gamma \vdash \mathsf{inr}\ e\ \mathsf{as}\ (\tau_1 + \tau_2) : \tau_1 + \tau_2}$$

$$\frac{\Gamma \vdash e_0 : \tau_1 + \tau_2 \quad \Gamma \vdash e_1 : \tau_1 \to \tau_3 \quad \Gamma \vdash e_2 : \tau_2 \to \tau_3}{\Gamma \vdash \mathsf{case}\ e_0\ \mathsf{of}\ (e_1 \mid e_2) : \tau_3}$$

The resulting type system is sound. We leave the proofs for Progress and Preservation as an exercise. One can also show that the new type system is also strongly normalizing.

2.4 Remark

Considering the injection constructs inr and inl, we can actually infer the type $\tau_1 + \tau_2$ for these expressions from their inclosing case construct. Then why do we need to put the type annotation $\tau_1 + \tau_2$ explicitly in inl/inr constructs? The answer is that, without these annotations, we could have different types for the same expression. These annotations guarantee the uniqueness of types.

Consider the following example with two case constructs:

$$\mathsf{case}\ (\mathsf{inl}\ \mathsf{true})\ \mathsf{of}\ ((\lambda x : \mathsf{bool} \,.\, x) \mid (\lambda x : \mathsf{int} \,.\, \mathsf{true}))$$
$$\mathsf{case}\ (\mathsf{inl}\ \mathsf{true})\ \mathsf{of}\ ((\lambda x : \mathsf{bool} \,.\, x) \mid (\lambda x : \mathsf{bool} \,.\, \mathsf{false}))$$

Then, in the first case, "inl true" has the type bool + int, and in the second case, it has the type bool + bool. So the same expression "inl true" might have different types in different contexts. With type annotations, we have different expressions: "inl true as (bool+int)" and "inl true as (bool+bool)".

3 Recursion

We next add the rec construct for recursive functions:

$$e ::= \ldots \quad | \quad \mathsf{rec}\ y : \tau \to \tau'(\lambda x : \tau.\, e)$$

The typing rule for recursive abstractions is:

$$\frac{\Gamma[x \mapsto \tau][y \mapsto \tau \to \tau'] \vdash e : \tau'}{\Gamma \vdash \mathsf{rec}\ y : \tau \to \tau'(\lambda x : \tau \,.\, e) : \tau \to \tau'}$$

In the presence of recursion, we may have well-typed expressions whose evaluation does not terminate. For instance, take a recursive function that calls itself with the same parameters. Then, the application of this function to an arbitrary argument will not terminate. Therefore, adding this construct breaks the strong normalization property.

The denotation for the recursive function is:

$$\mathcal{C}[\![\mathsf{rec}\ y : \tau \to \tau'(\lambda x : \tau.\, e)]\!]\rho = \mathit{fix}(\lambda f \in \mathcal{T}[\![\tau]\!] \to \mathcal{T}[\![\tau']\!]_\bot .\, \lambda v \in \mathcal{T}[\![\tau]\!].\, \mathcal{C}[\![e]\!]\rho[x \mapsto v][y \mapsto f])$$

Note that the range of f contains $\bot$. The reason for this is that the Fixed Point Theorem requires that the domain over which we take fixed points must have a bottom. In this case, the domain is $\mathcal{T}[\![\tau]\!] \to \mathcal{T}[\![\tau']\!]$; to add bottom to this function domain, we must add bottom to its range.

Also note that $\bot$ indicates indicates the possibility of non-termination, which is in agreement with the fact that the strong normalization property no longer holds.

With denotational semantics, the soundness property can be expressed as:

$$\rho \models \Gamma \wedge \Gamma \vdash e : \tau \Rightarrow \mathcal{C}[\![e]\!]\rho \in \mathcal{T}[\![\tau]\!]_\bot$$