CS411 Notes 14 Disjoint Unions

A. Demers

9 Apr 2001

Here we add sum (disjoint union) types to our language. In lecture we relied on subtyping rules for sum constructs. Here we present an alternative in which sum types are provided with tagged "injection" constructors. This is more in the style of the recursive type rules, which will be presented in the next set of notes.

1 Syntax

Types

$$\tau \ ::= \ \mathbf{sum}(\ \ldots\ x_i : \tau_i\ \ldots\ )$$

Informally a **sum** type is a tagged union — each value is taken from a finite (multi-) set of constituent types, with an associated tag indicating the constituent type associated with the value. The constituent types of a **sum** are not constrained to be distinct. For example,

$$\mathbf{sum}(a : \mathbf{prod}(), b : \mathbf{prod}())$$

is a legal **sum** type.

Expressions

$$\begin{aligned} e \ ::= \ & \mathbf{inj}_\tau(\ x \sim e'\ ) \\ & \mid \mathbf{case}\ e_0\ (\ \ldots\ x_i\ \Rightarrow\ e_i\ \ldots\ ) \end{aligned}$$

The *injection* constructors $\mathbf{inj}_\tau$ convert a value to a **sum** type from one of its constituent types. Let

$$\tau \equiv \mathbf{sum}(a : \mathbf{prod}(), b : \mathbf{prod}(), c : \mathbf{int})$$

Then some expression of type τ are

$$\mathbf{inj}_\tau(a \sim \langle\rangle) \qquad \mathbf{inj}_\tau(b \sim \langle\rangle) \qquad \mathbf{inj}_\tau(c \sim 17)$$

Note the first two of these expressions are distinct — they inject the same value $\langle\rangle$ into the **sum** type, but with two different tags a and b.

A **case** expression analyzes a value of a **sum** type based on the value's tag. For example, for an expression e of the above type τ one could write

$$\mathbf{case}\ e\ (a \Rightarrow 0, b \Rightarrow 1, c \Rightarrow (c + 2))$$

Within the subexpression associated with a tag, the tag name itself is bound to the value inside e. Thus, if e is $\mathbf{inj}_\tau(c \sim 17)$, then the value of the above **case** expression is (17+2) or 19.

Values

$$v \ ::= \ \mathbf{inj}_\tau(\ x \sim v'\ )$$

As usual, whenever we add a new type we must extend our notion of a value (canonical term). In this case it is straightforward — the values of **sum** type are exactly the **inj** constructors applied to values of the constituent types.

2 Typing Rules

Here are the new typing rules for **sum** types.

$$\frac{\pi \vdash e : \tau'}{\pi \vdash \mathbf{inj}_\tau(\ x \sim e\ ) : \tau \quad \text{where } \tau \equiv \mathbf{sum}(\ \ldots\ x : \tau'\ \ldots\ )} \qquad \text{(T14.1)}$$

An **inj** constructor applied to a well typed expression of a constituent type is a well typed expression of the **sum** type.

$$\frac{\begin{array}{l}\pi \vdash e : \mathbf{sum}(\ \ldots\ x_i : \tau_i\ \ldots\ ) \\ \quad \ldots \\ (\pi \oplus \{x_i : \tau_i\}) \vdash e_i : \tau \\ \quad \ldots\end{array}}{\pi \vdash : \mathbf{case}\ e\ (\ \ldots x_i\ \Rightarrow\ e_i\ \ldots\ ) : \tau} \qquad \text{(T14.2)}$$

This rule analyzes an expression of a **sum** type according to the tag (hence the type) of the current value. Each case expression e_i must be well typed (and have the same type τ) when the tag identifier x_i is assumed to have the i^th constituent type.

3 Evaluation Rules

Finally, here are the evaluation rules.

$$\frac{\langle e, \phi, \sigma\rangle \to \langle v, \sigma'\rangle}{\langle \mathbf{inj}_\tau(\ x \sim e\ ), \phi, \sigma\rangle \to \langle \mathbf{inj}_\tau(\ x \sim v, \sigma'\rangle} \tag{E14.3}$$

An **inj** constructor applied to a well typed expression of a constituent type reduces to the value of the constituent expression tagged with the appropriate selector name.

$$\frac{\begin{array}{l}\langle e, \phi, \sigma\rangle \to \langle \mathbf{inj}_\tau(\ x_i \sim v_i, \sigma''\rangle \\ \langle e_i, (\phi \oplus \{x_i \sim v_i\}), \sigma''\end{array}}{\langle \mathbf{case}\ e\ (\ \ldots x_i\ \Rightarrow\ e_i\ \ldots\ ), \phi, \sigma\rangle \to \langle v_i, \sigma'\rangle} \tag{E14.4}$$

To reduce a **case** expression, first evaluate the object expression e (the result value will necessarily have the form $\mathbf{inj}_\tau(x_i \sim v_i)$); then select the arm of the **case** expression identified by x_i and evaluate it with x_i bound to the value v_i.

4 Soundness

The soundness proof outlined in Notes 12 can be extended to **sum** types without any surprises.