Methods in Structures

Luca Cardelli

There is a slight technical problem in adding methods to dependent structures. The “self” parameter of a method in a binding needs to know the final signature, since the method may want to refer to methods to its right through self. So the usual left-to-right checking of structures and signatures does not quite work. I don’t think we want to make the entire binding recursive. So here is a left-to-right solution that accounts for mutual method access through self.

A structure \( \{B\} \) contains a dependent binding \( B \), and a signature \( \{D\} \) contains a dependent declaration \( D \). We keep track at all times of the final declaration, via the judgment \( \Gamma \vdash B : D D' \), meaning that \( B \) has declaration \( D \), but must still be “integrated” with \( D' \). Eventually we prove \( \Gamma \vdash B : D() \) and we are done.

See [Harper Lillibridge 1993] for the notation and the other necessary rules. Here \( \xi \) is the label-stripping function, and \( \xi \) is a vector of variables/labels, where each \( \xi \) is a type or term variable/label.

Bindings: \( B \)
Structures: \( \{B\} \)
Declarations: \( D \)
Signatures: \( \{D\} \)

\[
\begin{align*}
\Gamma \vdash A :: K & \quad \text{type } A \text{ has kind } K \\
\Gamma \vdash a : A & \quad \text{term } a \text{ has type } A \\
\Gamma \vdash B : D D' & \quad \text{binding } B \text{ has declaration } D, \text{ pending } D' \\
\Gamma \vdash \{B\} : \{D\} & \quad \text{structure } \{B\} \text{ has signature } \{D\} \\
\end{align*}
\]

\( x \div A \) method \( x \) has result type \( A \) (used in contexts and declarations)

(Empty binding)
\[
\begin{align*}
\Gamma \vdash & D \\
\Gamma \vdash & () : () D
\end{align*}
\]

(Type binding)
\[
\begin{align*}
\Gamma \vdash B : D \triangleright X :: K, D' & \quad \Gamma, \xi \vdash A :: K X \notin \text{dom}(\Gamma, \xi) \\
\Gamma \vdash & B, b \triangleright X :: K = A : D, b \triangleright X :: K D'
\end{align*}
\]

(Term binding)
\[
\begin{align*}
\Gamma \vdash B : D \triangleright b : x :: A, D' & \quad \Gamma, \xi \vdash a : A x \notin \text{dom}(\Gamma, \xi) \\
\Gamma \vdash & B, b \triangleright x :: A = a : D, b \triangleright x :: A D'
\end{align*}
\]
The substitution $A\{\overline{y,\xi}\}$ in the (Method binding) rule needs some explanations. At first I wrote:

$$
\begin{align*}
\Gamma \vdash B : D \mid b \triangleright x : A, D' & \quad \Gamma, \mathcal{D}, y : S \vdash a : (A\{\overline{y,\xi}\}) \quad x, y \notin \text{dom}(\Gamma, \mathcal{D}) \quad \xi \notin \text{dom}(\mathcal{D}) \\
\Gamma \vdash B, b \triangleright x : A \{\overline{\xi}\} = \zeta(y : S)a : D, b \triangleright x : A \{\overline{\xi}\} / D'
\end{align*}
$$

Where the restriction $y \notin A$ is similar to the usual restriction for the dot notation in function result types. But, according to (Method binding 0), the following does not typecheck (I am abbreviating $\xi \triangleright \xi$ as $\xi$):

$$
\{X::\text{Type} = \text{Int}, \ z \triangleright X = \zeta(y : \{X::\text{Type}, z \triangleright X\}) \ y, z\} : \{X::\text{Type} = \text{Int}, \ z \triangleright X\}
$$

However, we know that the current $D$ in (Method binding 0) is a prefix of the final $S$, which is the type of $y$. Hence $y.X$ is really the same as $X$ in the current context, for any $X$ declared in $D$. This is what (Method binding) is saying, and the example above is then typeable.

By the way, for a similar situation [Harper Lillibridge 1993] uses the following rules:

$$
\begin{align*}
\Gamma, x : A \vdash a : A' \\
\Gamma \vdash \lambda(x : A)a : \Pi(x : A)A'
\end{align*}
$$

$$
\begin{align*}
\Gamma \vdash a' : \Pi(x : A)A' & \quad \Gamma \vdash a : A' \quad x \notin A' \\
\Gamma \vdash a'(a) : A'
\end{align*}
$$

I don’t quite understand why the side condition is placed on elimination, and not on introduction. Without subsumption, if $x$ occurs in $A'$ then $\lambda(x : A)a$ is unusable, and we are only delaying the error messages. Subsumption can eliminate occurrences of $x$ in $A'$, but is this really useful?