6.6. POLYMORPHIC TYPE SYSTEMS AND TYPED $\mu$SCHEME

Type checking

This book does not provide a type checker for Typed $\mu$Scheme; its implementation is left as Exercise 14. Type checking requires an expression or definition, a type environment, and a kind environment. Calling $\text{elabdef}(t, \Gamma, \Delta)$ should return a pair $(\Gamma', s)$, where $(t, \Gamma) \to \Gamma'$ and $s$ is a string that represents the type of the thing defined.

(type checking for Typed $\mu$Scheme [prototype] 283a) =

| exception LeftAsExercise of string |
| fun typeof _ = raise LeftAsExercise "typeof" |
| fun elabdef _ = raise LeftAsExercise "elabdef" |

6.6.7 The rest of an interpreter for Typed $\mu$Scheme

Evaluation

Here is an appropriate place to dispose of the evaluation rules for Typed $\mu$Scheme. The rules for expressions are exactly the same as the rules for $\mu$Scheme; at run time, the types have no effect whatever. We require new rules for type abstraction and application, but the evaluator behaves exactly as if these constructs aren’t there.

\[
\langle e, \rho, \sigma \rangle \Downarrow \langle v, \sigma' \rangle
\]

(TYAPPLY)

\[
\langle \text{TYAPPLY}(e, \tau_1, \ldots, \tau_n), \rho, \sigma \rangle \Downarrow \langle v, \sigma' \rangle
\]

(TYAPPLY)

\[
\langle e, \rho, \sigma \rangle \Downarrow \langle v, \sigma' \rangle
\]

(TYLAMBDA)

\[
\langle \text{TYLAMBDA}(\alpha_1, \ldots, \alpha_n, e), \rho, \sigma \rangle \Downarrow \langle v, \sigma' \rangle
\]

(TYLAMBDA)

Most of the evaluator for Typed $\mu$Scheme is just like the evaluator for $\mu$Scheme in Chapter 5. The code for the two new cases acts as if $\text{TYAPPLY}$ and $\text{TYLAMBDA}$ aren’t there.

(alternatives for ev for $\text{TYAPPLY}$ and $\text{TYLAMBDA}$ 283b) =

| ev (\text{TYAPPLY} (e, _)) = ev e |
| ev (\text{TYLAMBDA} (_, e)) = ev e |

The rest of the evaluator appears in Appendix G.

The rules for definitions are slightly different from those in $\mu$Scheme; as described in Exercise 37 in Chapter 3, $\text{VAL}$ must always create a new binding. Otherwise, we could subvert the type system by using $\text{VAL}$ to change the type of an existing value. The $\text{VAL}$ rule must use the old environment; $\text{VAL-REC}$ uses the new one.

\[
\ell \notin \text{dom} \sigma
\]

\[
\langle e, \rho, \sigma \rangle \Downarrow \langle v, \sigma' \rangle
\]

(VAL)

\[
\langle \text{VAL}(x, e), \rho, \sigma \rangle \to \langle \rho\{x \mapsto \ell\}, \sigma'\{\ell \mapsto v\} \rangle
\]

(VAL)

\[
\ell \notin \text{dom} \sigma
\]

\[
\langle e, \rho\{x \mapsto \ell\}, \sigma\{\ell \mapsto \text{unspecified}\} \rangle \Downarrow \langle v, \sigma' \rangle
\]

(VAL-REC)

\[
\langle \text{VAL-REC}(x, \tau, e), \rho, \sigma \rangle \to \langle \rho\{x \mapsto \ell\}, \sigma'\{\ell \mapsto v\} \rangle
\]

(VAL-REC)

The code that implements these rules is in Appendix G.