Cubex with List Comprehensions

Ross Tate

December 5, 2013

1 Lexing and Parsing

1.1 Core and Full Languages

$$\begin{array}{rrcl} \text{comprehension} & c & ::= & \varnothing \mid e, c \mid \textbf{if}\ (e)\ c \mid \textbf{for}\ (\nu_v\ \textbf{in}\ e)\ c \\ \text{expression} & e & ::= & \nu_v \mid \nu_{vc}\langle\tau,\ldots,\tau\rangle(e,\ldots,e) \mid e.\nu_v\langle\tau,\ldots,\tau\rangle(e,\ldots,e) \mid [c] \mid e +\!\!+ e \mid \textbf{true} \mid \textbf{false} \mid n \mid \texttt{"string"} \end{array}$$

Figure 1: Changes to the Cubex Core Language Grammar

The extension to the full language differs from the core language in a few ways:

- The symbol $\varnothing$ is not in the full language.
- An expression e when used in the full language where a comprehension is expected represents the comprehension $e, \varnothing$ in the core language.
- The expression [ ] in the full language represents the expression $[\varnothing]$ in the core language.

2 Validating

The following are the changed typing judgements and rules.

Judgement	Meaning	Figure
$\Psi \mid \Theta \mid \Delta \mid \Gamma \vdash c : \tau$	comprehension c generates values of type τ	2
$\Psi \mid \Theta \mid \Delta \mid \Gamma \vdash e : \tau$	expression e has type τ	3

$$\boxed{\Psi \mid \Theta \mid \Delta \mid \Gamma \vdash c : \tau}$$

$$\frac{}{\Psi \mid \Theta \mid \Delta \mid \Gamma \vdash \varnothing : \tau} \qquad \frac{\Psi \mid \Theta \mid \Delta \mid \Gamma \vdash e : \tau \quad \Psi \mid \Theta \mid \Delta \mid \Gamma \vdash c : \tau}{\Psi \mid \Theta \mid \Delta \mid \Gamma \vdash e, c : \tau}$$

$$\frac{\Psi \mid \Theta \mid \Delta \mid \Gamma \vdash e : \texttt{Boolean}\langle\rangle \quad \Psi \mid \Theta \mid \Delta \mid \Gamma \vdash c : \tau}{\Psi \mid \Theta \mid \Delta \mid \Gamma \vdash \textbf{if}\ (e)\ c : \tau} \qquad \frac{\Psi \mid \Theta \mid \Delta \mid \Gamma \vdash e : \texttt{Iterable}\langle\tau'\rangle \quad \Psi \mid \Theta \mid \Delta \mid \Gamma, \nu : \tau' \vdash c : \tau}{\Psi \mid \Theta \mid \Delta \mid \Gamma \vdash \textbf{for}\ (\nu\ \textbf{in}\ e)\ c : \tau}$$

Figure 2: Type Checking Comprehensions

$$\boxed{\Psi \mid \Theta \mid \Delta \mid \Gamma \vdash e : \tau}$$

$$\frac{\text{for all } i, \quad \Psi \mid \Theta \mid \Delta \mid \Gamma \vdash e_i : \tau}{\Psi \mid \Theta \mid \Delta \mid \Gamma \vdash [e_1, \ldots, e_n] : \texttt{Iterable}\langle\tau\rangle} \text{ becomes } \frac{\Psi \mid \Theta \mid \Delta \mid \Gamma \vdash c : \tau}{\Psi \mid \Theta \mid \Delta \mid \Gamma \vdash [c] : \texttt{Iterable}\langle\tau\rangle}$$

Figure 3: Type Checking Expressions

3 Semantics

Any expression $[c]$ should always terminate; that is, the elements of the iterable should be determined lazily. $\varnothing$ generates no values. e, c generates the value of e followed by the values generated by c. **if** (e) c generates no values if e evaluates to **false** and generates all the values generated by c if e evaluates to **true**. For each element v of iterable e, **for** $(\nu$ **in** $e)$ c generates the values generated by c with ν assigned to the value v; this is done lazily so that the comprehension generates values even if e is an infinite iterable.

If the body of a comprehension refers to a mutable variable, the comprehension should use the value of that variable at the point in time that the iterable is created.

4 Evaluation

The extension will be evaluated with the same process as PA4, except only testing stages 1 through 3. Note, though, that a lot of emphasis will be placed on testing the laziness of the generated iterables.