Assignment 5: Universal, Existential, and Recursive Types
Model Solution

15-312 Foundations of Programming Languages
Kevin Watkins ⟨kw@cmu.edu⟩

April 28, 2005

Please refer to the assignment itself for the full description and statement of each problem.

§ 1. Encodings in System F **A 1.1.** [5 pts]

$$\begin{aligned}
\mathit{mkpair} &= \Lambda\beta_1.\,\Lambda\beta_2.\,\lambda x_1:\beta_1.\,\lambda x_2:\beta_2.\,\Lambda\alpha.\,\lambda y:(\beta_1\to\beta_2\to\alpha).\,y\;x_1\;x_2\\
\mathit{fst} &= \Lambda\beta_1.\,\Lambda\beta_2.\,\lambda p:\beta_1\times\beta_2.\,p\,[\beta_1]\,(\lambda x_1:\beta_1.\,\lambda x_2:\beta_2.\,x_1)\\
\mathit{snd} &= \Lambda\beta_1.\,\Lambda\beta_2.\,\lambda p:\beta_1\times\beta_2.\,p\,[\beta_2]\,(\lambda x_1:\beta_1.\,\lambda x_2:\beta_2.\,x_2)
\end{aligned}$$

A 1.2. [5 pts] Let $\mathbf{unit} = \forall\alpha.\,\alpha\to\alpha$, which has the single System F value $\Lambda\alpha.\,\lambda x:\alpha.\,\alpha$. (In System F we usually allow evaluation under a lambda.)

§ 2. Encoding Existential Types **A 1.3.** [5 pts]

$$\begin{aligned}
\ulcorner\mathbf{pack}\;(\tau',e)\;\mathbf{as}\;\exists\alpha.\,\tau\urcorner &= \Lambda\beta.\,\lambda y:(\forall\alpha.\,\tau\to\beta).\,y\,[\tau']\,\ulcorner e\urcorner\\
\ulcorner\mathbf{unpack}_{\tau'}\;(\alpha,x)=e_1\;\mathbf{in}\;e_2\urcorner &= \ulcorner e_1\urcorner\,\tau'\,(\Lambda\alpha.\,\lambda x:\tau.\ulcorner e_2\urcorner)
\end{aligned}$$

§ 3. A Mystery Encoding **A 1.4.** [5 pts] It's the MinML type constructor for sums. The encodings are

$$\begin{aligned}
\ulcorner\mathbf{in}_{1\,\tau_1+\tau_2}\;e\urcorner &= \Lambda\alpha.\,\lambda y_1:(\tau_1\to\alpha).\,\lambda y_2:(\tau_2\to\alpha).\,y_1\,\ulcorner e\urcorner\\
\ulcorner\mathbf{in}_{2\,\tau_1+\tau_2}\;e\urcorner &= \Lambda\alpha.\,\lambda y_1:(\tau_1\to\alpha).\,\lambda y_2:(\tau_2\to\alpha).\,y_2\,\ulcorner e\urcorner\\
\ulcorner\mathbf{case}_\tau\;e\;\mathbf{of}\;\mathbf{in}_1\,x_1\Rightarrow e_1\mid\mathbf{in}_2\,x_2\Rightarrow e_2\urcorner &= \ulcorner e\urcorner\,[\tau]\,(\lambda x_1:\tau_1.\ulcorner e_1\urcorner)\,(\lambda x_2:\tau_2.\ulcorner e_2\urcorner)
\end{aligned}$$

A 1.5. [extra credit] **foo** is the void type 0; τ **bar** is the type τ **cont**. (The type τ **bar** isn't particularly useful in pure System F, though, because the **letcc** construct isn't available.)

§ 4. Simulation Theorem **A 1.6.** [5 pts] This is broken in so many ways it's not even funny. First of all, each MinML step gets simulated by many System F steps. For example, the MinML step $(v_1, v_2).\,1 \mapsto v_1$ is simulated by the sequence

$$\begin{array}{ll} & (\Lambda\alpha.\,\lambda y:(\tau_1\to\tau_2\to\alpha).\,y\;v_1\;v_2)\;[\tau_1]\;(\lambda x_1:\tau_1.\,\lambda x_2:\tau_2.\,x_1) \\ \mapsto & (\lambda y:(\tau_1\to\tau_2\to\tau_1).\,y\;v_1\;v_2)\;(\lambda x_1:\tau_1.\,\lambda x_2:\tau_2.\,x_1) \\ \mapsto & (\lambda x_1:\tau_1.\,\lambda x_2:\tau_2.\,x_1)\;v_1\;v_2 \\ \mapsto & (\lambda x_2:\tau_2.\,v_1)\;v_2 \\ \mapsto & v_1 \end{array}$$

Beyond that, we have the problem that MinML evaluates the components of a pair eagerly, while the System F encoding of a pair is a value (because it involves lambdas, which are values). So the encoding is really for lazy pairs, not eager pairs. Since in pure System F, every term evaluates to a value in finitely many steps, this isn't a big deal for System F itself. But it does break the simulation theorem as it's stated, because the order of evaluation will be totally different between pure MinML and pure System F.

A 2.1. [5 pts] Let $bits = \mu\alpha.\,1+\alpha+\alpha$. I'll use ternary sums (see Assignment 7). Then we define

$$\begin{array}{rcl} empty &=& \mathbf{roll}\,(\mathbf{in}_1\,()) \\ zero &=& \lambda x:bits.\,\mathbf{roll}\,(\mathbf{in}_2\,x) \\ one &=& \lambda x:bits.\,\mathbf{roll}\,(\mathbf{in}_3\,x) \\ bitcase &=& \lambda x:bits.\,\lambda y_1:(1\to\tau).\,\lambda y_2:(bits\to\tau).\,\lambda y_3:(bits\to\tau). \\ && \quad \mathbf{case\ unroll}\,x\;\mathbf{of\ in}_1\,\_\Rightarrow y_1\,()\mid \mathbf{in}_2\,x_2\Rightarrow y_2\,x_2\mid \mathbf{in}_3\,x_3\Rightarrow y_3\,x_3 \end{array}$$

A 2.2. [5 pts] One tedious but simple solution is

$$\begin{array}{rcl} or &=& \mathbf{fun}\;or(x:bits):bits.\,\lambda y:bits. \\ && \quad bitcase\,x \\ && \qquad (\lambda\_.\,bitcase\,y\,(\lambda\_.\,empty)\,(\lambda\_.\,\mathbf{fail})\,(\lambda\_.\,\mathbf{fail})) \\ && \qquad (\lambda x'.\,bitcase\,y\,(\lambda\_.\,\mathbf{fail})\,(\lambda y'.\,zero\,(or\,x'\,y'))\,(\lambda y'.\,one\,(or\,x'\,y'))) \\ && \qquad (\lambda x'.\,bitcase\,y\,(\lambda\_.\,\mathbf{fail})\,(\lambda y'.\,one\,(or\,x'\,y'))\,(\lambda y'.\,one\,(or\,x'\,y'))) \end{array}$$

A 2.3. [5 pts] Let

$$\begin{array}{rcl} BITS &=& \exists\alpha.\,\alpha\times(\alpha\to\alpha)\times(\alpha\to\alpha)\times \\ && \quad (\forall\beta.\,\alpha\to(1\to\beta)\to(\alpha\to\beta)\to(\alpha\to\beta)\to\beta) \\ Bits &=& \mathbf{pack}\,(bits,(empty,zero,one,\Lambda\beta.\,bitcase))\;\mathbf{as}\;BITS \end{array}$$

with *bits*, *empty*, *zero*, *one*, and *bitcase* as above.

A 2.4. [10 pts] Use

$$\mathbf{unpack}\,(bits,x)=Bits\;\mathbf{in}\;\{x.1,x.2,x.3,x.4\,[bits]/empty,zero,one,bitcase\}\,or$$

with *or* as above.