Active Typechecking and Translation: A Safe Language-Internal Extension Mechanism
Appendix

Cyrus Omar and Jonathan Aldrich

Carnegie Mellon University, Pittsburgh, PA 15213, USA
{comar,aldrich}@cs.cmu.edu

A @λ

A.1 Helper Functions

The following helper functions are useful for working with lists of denotations:

$$
\begin{aligned}
\textbf{const} &:= (\lambda \mathbf{a}{:}\mathsf{list}[\mathsf{Den}].\lambda \mathbf{d}{:}\mathsf{Den}.\mathsf{fold}(\mathbf{a}; \mathbf{d}; \_, \_, \_.\mathsf{err})) \\
\textbf{pop} &:= (\lambda \mathbf{a}{:}\mathsf{list}[\mathsf{Den}].\lambda \mathbf{f}{:}\mathsf{ITm} \to \star \to \mathsf{list}[\mathsf{Den}] \to \mathsf{Den}. \\
&\qquad \mathsf{fold}(\mathbf{a}; \mathsf{err}; \mathbf{d}, \mathbf{b}, \_.\mathsf{case}\ \mathbf{d}\ \mathsf{of}\ [\![\mathbf{x}\ \mathsf{as}\ \mathbf{t}]\!] \Rightarrow \mathbf{f}\ \mathbf{x}\ \mathbf{t}\ \mathbf{b}\ \mathsf{ow}\ \mathsf{err})) \\
\textbf{pop\_final} &:= (\lambda \mathbf{a}{:}\mathsf{list}[\mathsf{Den}].\lambda \mathbf{f}{:}\mathsf{ITm} \to \star \to \mathsf{Den}. \\
&\qquad \mathsf{fold}(\mathbf{a}; \mathsf{err}; \mathbf{d}, \mathbf{b}, \_. \\
&\qquad \mathsf{fold}(\mathbf{b}; \mathsf{case}\ \mathbf{d}\ \mathsf{of}\ [\![\mathbf{x}\ \mathsf{as}\ \mathbf{t}]\!] \Rightarrow \mathbf{f}\ \mathbf{x}\ \mathbf{t}\ \mathsf{ow}\ \mathsf{err}; \_, \_, \_.\mathsf{err})))
\end{aligned}
$$

The following helper function is useful for checking the equivalence of two types before producing a denotation. If the two types are not equivalent, an error is returned.

$$\textbf{check\_type} := (\lambda \mathbf{t1}{:}\star.\lambda \mathbf{t2}{:}\star.\lambda \mathbf{d}{:}\mathsf{Den}.\mathsf{if}\ \mathbf{t1} \equiv_\star \mathbf{t2}\ \mathsf{then}\ \mathbf{d}\ \mathsf{else}\ \mathsf{err})$$

A.2 Definition of Arrow types

The definition of the Arrow type family is built into @λ as follows:

$$
\begin{aligned}
\Sigma_0 &:= \textsc{Arrow}[\star \times \star, \boldsymbol{ap}[1]] \\
\Phi_0 &:= \textsc{Arrow}[\theta_0, \mathbf{i}.\blacktriangledown(\mathsf{rep}(\mathsf{fst}(\mathbf{i})) \to \mathsf{rep}(\mathsf{snd}(\mathbf{i})))] \\
\theta_0 &:= \boldsymbol{ap}[1](\mathbf{i}, \mathbf{a}.\textbf{pop}\ \mathbf{a}\ \lambda \mathbf{x1}{:}\mathsf{Den}.\lambda \mathbf{t1}{:}\star.\lambda \mathbf{b}{:}\mathsf{list}[\mathsf{Den}]. \\
&\qquad \textbf{pop\_final}\ \mathbf{b}\ \lambda \mathbf{x2}{:}\mathsf{Den}.\lambda \mathbf{t2}{:}\star. \\
&\qquad \mathsf{case}\ \mathbf{t1}\ \mathsf{of}\ \textsc{Arrow}\langle \mathbf{j} \rangle \Rightarrow \\
&\qquad\quad \textbf{check\_type}\ \mathsf{fst}(\mathbf{j})\ \mathbf{t2}\ [\![\blacktriangledown(\blacktriangle(\mathbf{x1})\ \ \blacktriangle(\mathbf{x2}))\ \mathsf{as}\ \mathsf{snd}(\mathbf{j})]\!] \\
&\qquad \mathsf{ow}\ \mathsf{err}) \\
\Xi_0 &:= \textsc{Arrow}
\end{aligned}
$$

A.3 Deabstraction

$\boxed{\vdash_\Phi \sigma \not\downarrow \sigma'}$

$$\textsc{deabs-int}\ \frac{}{\vdash_\Phi \mathbb{Z} \not\downarrow \mathbb{Z}} \qquad \textsc{deabs-arrow}\ \frac{\vdash_\Phi \sigma_1 \not\downarrow \sigma_1' \qquad \vdash_\Phi \sigma_2 \not\downarrow \sigma_2'}{\vdash_\Phi \sigma_1 \to \sigma_2 \not\downarrow \sigma_1' \to \sigma_2'} \qquad \textsc{deabs-prod}\ \frac{\vdash_\Phi \sigma_1 \not\downarrow \sigma_1' \qquad \vdash_\Phi \sigma_2 \not\downarrow \sigma_2'}{\vdash_\Phi \sigma_1 \times \sigma_2 \not\downarrow \sigma_1' \times \sigma_2'}$$

$$\textsc{deabs-rep}\ \frac{\textsc{Fam}[\theta, \mathbf{i}.\tau_{\text{rep}}] \in \Phi \qquad [\tau_{\text{idx}}/\mathbf{i}]\tau \Downarrow \blacktriangledown(\sigma) \qquad \vdash_\Phi^{\Xi_0} \sigma \rightsquigarrow \sigma' \qquad \vdash_\Phi \sigma' \not\downarrow \sigma''}{\vdash_\Phi \mathsf{rep}(\textsc{Fam}\langle\tau_{\text{idx}}\rangle) \not\downarrow \sigma''}$$

$\boxed{\vdash_\Phi \gamma \not\downarrow \gamma'}$

$$\textsc{deabs-var}\ \frac{}{\vdash_\Phi x \not\downarrow x} \qquad \textsc{deabs-lam}\ \frac{\vdash_\Phi^{\Xi_0} \sigma \rightsquigarrow \sigma' \qquad \vdash_\Phi \sigma' \not\downarrow \sigma'' \qquad \vdash_\Phi \gamma \not\downarrow \gamma'}{\vdash_\Phi \lambda x{:}\sigma.\gamma \not\downarrow \lambda x{:}\sigma''.\gamma'} \qquad \textsc{deabs-fix}\ \frac{\vdash_\Phi^{\Xi_0} \sigma \rightsquigarrow \sigma' \qquad \vdash_\Phi \sigma' \not\downarrow \sigma'' \qquad \vdash_\Phi \gamma \not\downarrow \gamma'}{\vdash_\Phi \mathsf{fix}\ x{:}\sigma\ \mathsf{is}\ \gamma \not\downarrow \mathsf{fix}\ x{:}\sigma''\ \mathsf{is}\ \gamma'}$$

(omitted forms have trivially recursive rules)

$$\textsc{deabs-cancel}\ \frac{\vdash_\Phi \gamma \not\downarrow \gamma'}{\vdash_\Phi \triangle(\triangledown(\gamma)) \not\downarrow \gamma'} \qquad \textsc{deabs-trans}\ \frac{\vdash_\Phi \gamma \not\downarrow \gamma'}{\vdash_\Phi \mathsf{trans}([\![\triangledown(\gamma)\ \mathsf{as}\ \textsc{Fam}\langle\tau_{\text{idx}}\rangle]\!]) \not\downarrow \gamma'}$$

B Examples

B.1 Gödel's T

The implementation of Gödel's T in @λ encodes the following static and dynamic semantics.

Statics

$$\textsc{var}\ \frac{}{\Gamma, x : \tau \vdash x : \tau} \qquad \textsc{arrow-intro}\ \frac{\Gamma, x : \tau \vdash e : \tau'}{\Gamma \vdash \lambda x{:}\tau.e : \tau \to \tau'} \qquad \textsc{arrow-elim}\ \frac{\Gamma \vdash e_1 : \tau \to \tau' \qquad \Gamma \vdash e_2 : \tau}{\Gamma \vdash e_1\ e_2 : \tau'}$$

$$\textsc{nat-intro-z}\ \frac{}{\Gamma \vdash \mathsf{z} : \mathtt{nat}} \qquad \textsc{nat-intro-s}\ \frac{\Gamma \vdash e : \mathtt{nat}}{\Gamma \vdash \mathsf{s}(e) : \mathtt{nat}} \qquad \textsc{nat-elim}\ \frac{\Gamma \vdash e_1 : \mathtt{nat} \qquad \Gamma \vdash e_2 : \tau \qquad \Gamma \vdash e_3 : \mathtt{nat} \to \tau \to \tau}{\Gamma \vdash \mathsf{rec}(e_1; e_2; e_3) : \tau}$$

Dynamics

$$\textsc{lam-val}\ \frac{}{\lambda x{:}\tau.e\ \mathtt{val}} \qquad \textsc{ap-step-l}\ \frac{e_1 \mapsto e_1'}{e_1\ e_2 \mapsto e_1'\ e_2} \qquad \textsc{ap-step-r}\ \frac{e_2 \mapsto e_2'}{\lambda x{:}\tau.e\ e_2 \mapsto \lambda x{:}\tau.e\ e_2'} \qquad \textsc{ap-subst}\ \frac{e_2\ \mathtt{val}}{\lambda x{:}\tau.e\ e_2 \mapsto [e_2/x]e}$$

$$\textsc{z-val}\ \frac{}{\mathsf{z}\ \mathtt{val}} \qquad \textsc{s-step}\ \frac{e \mapsto e'}{\mathsf{s}(e) \mapsto \mathsf{s}(e')} \qquad \textsc{s-val}\ \frac{e\ \mathtt{val}}{\mathsf{s}(e)\ \mathtt{val}} \qquad \textsc{rec-step}\ \frac{e_1 \mapsto e_1'}{\mathsf{rec}(e_1; e_2; e_3) \mapsto \mathsf{rec}(e_1'; e_2; e_3)}$$

$$\textsc{rec-z}\ \frac{}{\mathsf{rec}(\mathsf{z}; e_2; e_3) \mapsto e_2} \qquad \textsc{rec-s}\ \frac{}{\mathsf{rec}(\mathsf{s}(e); e_2; e_3) \mapsto e_3\ e\ \mathsf{rec}(e; e_2; e_3)}$$

B.2 n-tuples

The static and dynamic semantics of n-tuples (that is, n-ary products) shown below can be encoded in @λ. Notice that this judgemental description of n-tuples relies on elipses frequently. This correspond to folds over the list of types that an n-tuple is indexed by, and underscores the need for a more concrete functional language for extensions, rather than a simple declarative scheme which is only completely precise and decidable when complex side conditions, elipses and search schemes are not necessary.

Statics

$$n\text{-TUPLE-INTRO}\ \frac{\Gamma \vdash e_1 : \tau_1 \qquad \cdots \qquad \Gamma \vdash e_n : \tau_n}{\Gamma \vdash (e_1, \ldots, e_n) : (\tau_1, \ldots, \tau_n)}\ (n \geq 0) \qquad n\text{-TUPLE-ELIM}\ \frac{\Gamma \vdash e : (\tau_1, \ldots, \tau_n)}{\Gamma \vdash \mathsf{pr}[i](e) : \tau_i}\ (1 \leq i \leq n)$$

Dynamics

$$n\text{-TUPLE-STEP}\ \frac{e_1\ \mathtt{val} \qquad \cdots \qquad e_{i-1}\ \mathtt{val} \qquad e_i \mapsto e_i'}{(e_1, \ldots, e_i, \ldots, e_n) \mapsto (e_1, \ldots, e_i', \ldots, e_n)} \qquad n\text{-TUPLE-VAL}\ \frac{e_1\ \mathtt{val} \qquad \cdots \qquad e_n\ \mathtt{val}}{(e_1, \ldots, e_n)\ \mathtt{val}}$$

$$\text{PROJ-}i\text{-STEP}\ \frac{e \mapsto e'}{\mathsf{pr}[i](e) \mapsto \mathsf{pr}[i](e')}\ (1 \leq i \leq n) \qquad \text{PROJ-}i\text{-OF-}n\ \frac{(e_1, \ldots, e_n)\ \mathtt{val}}{\mathsf{pr}[i]((e_1, \ldots, e_n)) \mapsto e_i}\ (1 \leq i \leq n)$$

Implementation in @λ n-tuples are implemented differently depending on n. For $n = 0$ (that is, the unit value), the integer 0 is used. For $n = 1$, the tuple only contains 1 element so it is represented without any additional run-time adornment. For $n > 1$,

nested pairs are used.

$$
\begin{array}{ll}
\mathsf{family}\ \textsc{Ntuple}[\mathsf{list}[\star]] \sim \mathbf{i}. & (1)\\
\quad \mathsf{fold}(\mathbf{i}; \blacktriangledown(\mathbb{Z}); \mathbf{s}, \mathbf{j}, \mathbf{r}.\mathsf{fold}(\mathbf{j}; \blacktriangledown(\mathsf{rep}(\mathbf{s})); \_, \_, \_.\blacktriangledown(\mathsf{rep}(\mathbf{s}) \times \blacktriangle(\mathbf{r})))) \ \{ & (2)\\
\quad \boldsymbol{new}[1](\_, \mathbf{a}.\mathsf{fold}(\mathbf{a}; [\![\triangledown(0)\ \mathsf{as}\ \textsc{Ntuple}\langle[]_\star\rangle]\!]; \mathbf{d}, \mathbf{b}, \mathbf{r}. & (3)\\
\quad\quad \mathsf{case}\ \mathbf{d}\ \mathsf{of}\ [\![\mathbf{dx}\ \mathsf{as}\ \mathbf{dt}]\!] \Rightarrow \mathsf{case}\ \mathbf{r}\ \mathsf{of}\ [\![\mathbf{rx}\ \mathsf{as}\ \mathbf{rt}]\!] \Rightarrow & (4)\\
\quad\quad \mathsf{case}\ \mathbf{rt}\ \mathsf{of}\ \textsc{Ntuple}\langle\mathbf{i}\rangle \Rightarrow \mathsf{fold}(\mathbf{b}; [\![\mathbf{dx}\ \mathsf{as}\ \textsc{Ntuple}\langle\mathbf{dt} :: \mathbf{i}\rangle]\!]; \_, \_, \_. & (5)\\
\quad\quad\quad [\![\triangledown((\triangle(\mathbf{dx}), \triangle(\mathbf{rx})))\ \mathsf{as}\ \textsc{Ntuple}\langle\mathbf{dt} :: \mathbf{i}\rangle]\!])\ \mathsf{ow\ err\ ow\ err\ ow\ err})); & (6)\\
\quad \boldsymbol{pr}[\mathbb{Z}](\mathbf{i}, \mathbf{a}.\mathbf{pop\_final\ a}\ \lambda\mathbf{x}{:}\mathsf{Den}.\lambda\mathbf{nt}{:}\star.\mathsf{case}\ \mathbf{nt}\ \mathsf{of}\ \textsc{Ntuple}\langle\mathbf{nl}\rangle \Rightarrow & (7)\\
\quad\quad \mathsf{fold}(\mathbf{nl}; \mathsf{err}; \mathbf{t1}, \mathbf{j}, \_. & (8)\\
\quad\quad \mathsf{fold}(\mathbf{j}; \mathsf{if}\ \mathbf{i} \equiv_{\mathbb{Z}} 0\ \mathsf{then}\ [\![\mathbf{x}\ \mathsf{as}\ \mathbf{t1}]\!]\ \mathsf{else\ err}; \_, \_, \_. & (9)\\
\quad\quad\quad \mathsf{fst}(\mathbf{foldl\ nl}\ ([\![\mathbf{x}\ \mathsf{as}\ \mathbf{nt}]\!], 0)\ \lambda\mathbf{r}{:}\mathsf{Den} \times \mathbb{Z}.\lambda\mathbf{t}{:}\star.\lambda\mathbf{ts}{:}\mathsf{list}[\star]. & (10)\\
\quad\quad\quad\quad \mathsf{if}\ \mathbf{i} \equiv_{\mathbb{Z}} \mathsf{snd}(\mathbf{r})\ \mathsf{then}\ \mathbf{r}\ \mathsf{else\ case}\ \mathsf{fst}(\mathbf{r})\ \mathsf{of}\ [\![\mathbf{rx}\ \mathsf{as}\ \_]\!] \Rightarrow & (11)\\
\quad\quad\quad\quad\quad \mathsf{if}\ \mathbf{i} \equiv_{\mathbb{Z}} \mathsf{snd}(\mathbf{r}) + 1\ \mathsf{then} & (12)\\
\quad\quad\quad\quad\quad\quad \mathsf{if}\ \mathbf{ts} \equiv_{\mathsf{list}[\star]} []_\star\ \mathsf{then}\ ([\![\mathbf{rx}\ \mathsf{as}\ \mathbf{t}]\!], \mathbf{i}) & (13)\\
\quad\quad\quad\quad\quad\quad\ \mathsf{else}\ ([\![\triangledown(\mathsf{fst}(\triangle(\mathbf{rx})))\ \mathsf{as}\ \mathbf{t}]\!], \mathbf{i}) & (14)\\
\quad\quad\quad\quad\quad \mathsf{else}\ ([\![\triangledown(\mathsf{snd}(\triangle(\mathbf{rx})))\ \mathsf{as}\ \mathbf{t}]\!], \mathsf{snd}(\mathbf{r}) + 1)\ \mathsf{ow\ err}))\ \mathsf{ow\ err}) & (15)\\
\} & (16)
\end{array}
$$

This definition uses a left fold function, **foldl**, defined in terms of the right fold operator built into the type-level language in the usual way, as follows:

$$
\begin{array}{l}
\mathbf{foldl} := \lambda\mathbf{l}{:}\mathsf{list}[\star].\lambda\mathbf{b}{:}\mathsf{Den} \times \mathbb{Z}.\lambda\mathbf{f}{:}(\mathsf{Den} \times \mathbb{Z}) \to \star \to \mathsf{list}[\star] \to (\mathsf{Den} \times \mathbb{Z}).\\
\qquad \mathsf{fold}(\mathbf{l}; \lambda\mathbf{x}{:}\mathsf{Den} \times \mathbb{Z}.\mathbf{x}; \mathbf{t}, \mathbf{ts}, \mathbf{r}.\lambda\mathbf{x}{:}\mathsf{Den} \times \mathbb{Z}.\mathbf{r}\ (\mathbf{f\ x\ t\ ts}))\ \mathbf{b}
\end{array}
$$