Phantom Contracts for Better Linking

Or, why-oh-why can't we have cross-language type errors?

Daniel Patterson — dbp@dbpmail.net

https://dbp.io

Summary

- Increasingly, languages provide programmers with rich types to enforce invariants.
- But linking components from different languages relies on **unsafe** FFIs with bad errors.
- Since linking happens after compilation, the key question is: *how can we preserve source-level invariants through compilation?*
- We propose enriching existing compiler target languages with **phantom contracts**, which are optional code that *runs at type-checking time*.
- **Phantom contracts** allow compiler writers to flexibly encode static invariants that cannot be expressed in the type system of the target.

Sample Source Languages

Index$^\pm$

$$\begin{array}{lcl}
\tau & ::= & \eta \mid \forall\alpha.\tau \mid \tau \to \tau \\
\eta & ::= & \alpha \mid +0 \mid - \mid \; ? \mid \eta{+}\eta \mid \eta{*}\eta \\
e & ::= & n \mid x \mid e+e \mid e*e \mid \lambda x:\ \tau.e \mid e\,e \\
 & & \Lambda\alpha.e \mid e[\eta] \\
v & ::= & n \mid \lambda x:\ \tau.e \mid \Lambda\alpha.e
\end{array}$$

Index$^\pm$ is a language with an indexed type system that allows index computations and abstraction over the sign of integers.

$$\frac{n \geq 0}{H;\Gamma \vdash n : +0} \qquad \frac{n < 0}{H;\Gamma \vdash n : -} \qquad \frac{x:\tau \in \Gamma}{H;\Gamma \vdash x : \tau}$$

$$\frac{H;\Gamma \vdash e_1 : \eta_1 \qquad H;\Gamma \vdash e_2 : \eta_2}{H;\Gamma \vdash e_1 + e_2 : \Downarrow(\eta_1{+}\eta_2)}$$

$$\frac{H;\Gamma \vdash e_1 : \eta_1 \qquad H;\Gamma \vdash e_2 : \eta_2}{H;\Gamma \vdash e_1 * e_2 : \Downarrow(\eta_1{*}\eta_2)} \qquad \frac{H;\Gamma, x\ \tau_1 \vdash e : \tau_2}{H;\Gamma \vdash \lambda x.e : \tau_2 \to \tau_2}$$

$$\frac{H;\Gamma \vdash e : \tau_1 \to \tau_2 \qquad H;\Gamma \vdash e' : \tau_1}{H;\Gamma \vdash e\ e' : \tau_2} \qquad \frac{H,\alpha;\Gamma \vdash e : \tau}{H;\Gamma \vdash \Lambda\alpha.e : \forall\alpha.\tau}$$

$$\frac{H;\Gamma \vdash e : \forall\alpha.\tau}{H;\Gamma \vdash e[\eta] : \Downarrow(\tau[\eta/\alpha])}$$

$$\begin{array}{lcl}
\Downarrow(\eta_1+\eta_2) & = & +\Downarrow(\Downarrow(n_1),\Downarrow(n_2)) \\
\Downarrow(\eta_1*\eta_2) & = & *\Downarrow(\Downarrow(n_1),\Downarrow(n_2)) \\
\Downarrow(\eta) & = & \eta
\end{array}$$

$$\begin{array}{lcl}
+\Downarrow(+0,+0) & = & +0 \\
+\Downarrow(-,+0) & = & ? \\
+\Downarrow(+0,-) & = & ? \\
+\Downarrow(-,-) & = & - \\
+\Downarrow(?,_) & = & ? \\
+\Downarrow(_,?) & = & ? \\
+\Downarrow(\eta_1,\eta_2) & = & \eta_1{+}\eta_2
\end{array}
\qquad
\begin{array}{lcl}
*\Downarrow(+0,+0) & = & +0 \\
*\Downarrow(-,+0) & = & - \\
*\Downarrow(+0,-) & = & - \\
*\Downarrow(-,-) & = & +0 \\
*\Downarrow(?,_) & = & ? \\
*\Downarrow(_,?) & = & ? \\
\Downarrow(\eta_1,\eta_2) & = & \eta_1{}\eta_2
\end{array}$$

BabyDill

$$\begin{array}{lcl}
\tau & ::= & \mathtt{int} \mid \tau \multimap \tau \mid !\tau \mid \tau\&\tau \mid \tau \otimes \tau \\
e & ::= & n \mid x \mid a \mid \lambda a : \tau.e \mid e\ e' \mid !e \\
 & & \mathtt{let}\ !x = e\ \mathtt{in}\ e' \mid \langle e, e' \rangle \mid e.1 \mid e.2 \\
 & & (e, e) \mid \mathtt{let}\ (a, a') = e\ \mathtt{in}\ e' \\
v & ::= & () \mid \lambda a : \tau.e \mid !e \mid \langle e, e' \rangle \mid (v, v')
\end{array}$$

BabyDill is a language with linear and unrestricted variables.

$$\frac{}{\Gamma; a:\tau \vdash a:\tau} \qquad \frac{x:\tau \in \Gamma}{\Gamma; \cdot \vdash x : \tau} \qquad \frac{}{\Gamma; \cdot \vdash n : \mathtt{int}}$$

$$\frac{\Gamma; \Delta, a:\tau_1 \vdash e : \tau_2}{\Gamma; \Delta \vdash \lambda a:\tau_1.e : \tau_1 \multimap \tau_2}$$

$$\frac{\Gamma;\Delta_1 \vdash e_1 : \tau_1 \multimap \tau_2 \qquad \Gamma;\Delta_2 \vdash e_2 : \tau_1 \qquad \Delta_3 \cong \Delta_1,\Delta_2}{\Gamma;\Delta_3 \vdash e_1\ e_2 : \tau_2}$$

$$\frac{\Gamma;\cdot \vdash e : \tau}{\Gamma;\cdot \vdash e : !\tau} \qquad \frac{\Gamma;\Delta \vdash e_1 : \tau_1 \qquad \Gamma;\Delta \vdash e_2 : \tau_2}{\Gamma;\Delta \vdash \langle e_1, e_2\rangle : \tau_1\&\tau_2}$$

$$\frac{\Gamma;\Delta \vdash e : \tau_1\&\tau_2}{\Gamma;\Delta \vdash e.1 : \tau_1} \qquad \frac{\Gamma;\Delta \vdash e : \tau_1\&\tau_2}{\Gamma;\Delta \vdash e.2 : \tau_2}$$

$$\frac{\Gamma;\Delta_1 \vdash e : !\tau \qquad \Gamma, x:\tau;\Delta_2 \vdash e' : \tau' \qquad \Delta_3 \cong \Delta_1,\Delta_2}{\Gamma;\Delta_3 \vdash \mathtt{let}\ !x = e\ \mathtt{in}\ e'}$$

$$\frac{\Gamma;\Delta_1 \vdash e_1 : \tau_1 \qquad \Gamma;\Delta_1 \vdash e_2 : \tau_2 \qquad \Delta_3 \cong \Delta_1,\Delta_2}{\Gamma;\Delta_3 \vdash (e_1, e_2) : \tau_1 \otimes \tau_2}$$

$$\frac{\Gamma;\Delta_1 \vdash e : \tau_1 \otimes \tau_2 \qquad \Gamma;\Delta_2, a:\tau_1, a':\tau_1 \vdash e' : \tau' \qquad \Delta_3 \cong \Delta_1,\Delta_2}{\Gamma;\Delta_3 \vdash \mathtt{let}\ (a, a') = e\ \mathtt{in}\ e' : \tau'}$$

Core Idea

- Terms $\hat{e}$ are paired with phantom contracts $\{\varphi\}$: *type-time operational code* to encode *static invariants*.
- Our BabyDill compiler encodes the number of variable uses left (≤ 1).

$$\begin{array}{rcl}
(a : \tau)^+ & \rightsquigarrow & a\{a := \delta^2(-, !a, 1)\} \\
(\lambda a : \tau_1.e)^+ & \rightsquigarrow & \lambda a : {\tau_1}^+.\mathtt{let_}\{a\} = 0\{\mathtt{ref}\ 1\}\ \mathtt{in} \\
 & & \mathtt{let_r}\{\} = e^+\ \mathtt{in} \\
 & & \mathtt{_r}\}\{\delta^1(\mathtt{assert}, \delta^2(\mathtt{sexp=}, !a, 0));\}
\end{array}$$

- Our Index$^\pm$ compiler encodes a representation of the source type.
- Safe interoperation of Index$^\pm$ & BabyDill means phantom contracts don't fail.

✔ $(\lambda a{:}\mathtt{int}\&\mathtt{int}.\ a.1)^+((2 * 3)^+, (4)^+)$

✘ $(\lambda x{:}{+0}.\ x + 2)^+(2)^+$

- Compiler writers can provide safe FFIs to help satisfy phantom contracts.

✔ $(\lambda x{:}{+0}.\ x + 2)^+\mathtt{not\_neg}(2)^+$

$\mathtt{not\_neg}\ x = \mathtt{if}(x < 0)\ \mathtt{fail}\ \mathtt{else}\ x\{\langle\!\langle +0 \rangle\!\rangle\}$

- Or extend their compilers once common encodings become established.

$(n : \mathtt{int})^+ \rightsquigarrow n\{\langle\!\langle +0 \rangle\!\rangle\}\ \mathtt{if}\ n >= 0$

✔ $(\lambda x{:}{+0}.\ x + 2)^+(2)^+$

Sample Target Language

Phantom

$$\begin{array}{lcl}
\tau & ::= & \mathtt{int} \mid \tau \to \tau' \mid \tau \times \tau' \mid \mu\alpha.\tau \mid \alpha \\
e & ::= & \hat{e}\{\varphi\} \\
\hat{e} & ::= & x \mid n \mid e + e \mid e * e \mid \lambda x{:}\tau.e \mid e\ e' \\
 & & (e, e) \mid \mathtt{fst}\ e \mid \mathtt{snd}\ e \mid \mathtt{fold}\ e \\
 & & \mathtt{unfold}\ e \mid \mathtt{let}\ x\{\nu\}{:}\tau = e\ \mathtt{in}\ e' \\
v & ::= & n \mid \mathtt{fold}\ v \mid \lambda x{:}\tau.e \mid (v, v) \\
\varphi & ::= & \ell \mid \mathtt{sexp} \mid \mathtt{ref}\ \mathtt{sexp} \mid \varphi := \varphi \mid !\varphi \mid \nu \\
 & & \mathtt{match}\ \varphi \mid n\nu_1.\varphi_1 \mid s\nu_2.\varphi_2 \mid b\nu_3.\varphi_3 \mid (\nu_4, \nu_5).\varphi_4 \\
 & & \lambda\nu{:}\varphi\tau.\varphi \mid \varphi\ \varphi' \mid \varphi;\varphi' \mid \delta^n(\mathtt{op}^n, \varphi_1, \ldots \varphi_n) \\
\mathtt{sexp} & ::= & n \mid s \mid \mathtt{true} \mid \mathtt{false} \mid (\mathtt{sexp}, \mathtt{sexp}') \\
\mathtt{op}^1 & ::= & \mathtt{assert} \mid \mathtt{not} \mid \mathtt{length} \mid \ldots \\
\mathtt{op}^2 & ::= & \mathtt{sexp=} \mid \mathtt{append} \mid + \mid * \mid \ldots \\
\varphi\tau & ::= & \mathtt{sexp} \mid \mathtt{ref}\ \mathtt{sexp} \mid \varphi\tau \to \varphi\tau
\end{array}$$

$$\frac{\Gamma;S \vdash \hat{e} : \tau;S' \qquad \Gamma \vdash (S', \varphi) \Downarrow (S'', \varphi')}{\Gamma;S \vdash \hat{e}\{\varphi\} : \tau;S''} \qquad \frac{x : \tau \in \Gamma}{\Gamma;S \vdash x : \tau;S}$$

$$\frac{}{\Gamma;S \vdash n : \mathtt{int};S} \qquad \frac{\Gamma;S \vdash e_1 : \mathtt{int};S' \qquad \Gamma;S' \vdash e_2;S'' : \mathtt{int}}{\Gamma;S \vdash e_1 + e_2 : \mathtt{int};S''}$$

$$\frac{\Gamma;S \vdash e_1 : \mathtt{int};S' \qquad \Gamma;S' \vdash e_2;S'' : \mathtt{int}}{\Gamma;S \vdash e_1 * e_2 : \mathtt{int};S''}$$

$$\frac{\Gamma, x : \tau;S \vdash e : \tau';S'}{\Gamma;S \vdash \lambda x{:}\tau.e : \tau \to \tau';S'} \qquad \frac{\Gamma;S \vdash e : \tau \to \tau';S' \qquad \Gamma;S' \vdash e' : \tau;S''}{\Gamma;S \vdash e\ e' : \tau';S''}$$

$$\frac{\Gamma;S \vdash e_1 : \tau_1;S' \qquad \Gamma;S' \vdash e_2;S'' : \tau_2}{\Gamma;S \vdash (e_1, e_2) : \tau_1 \times \tau_2;S''} \qquad \frac{\Gamma;S \vdash e : \tau_1 \times \tau_2;S'}{\Gamma;S \vdash \mathtt{fst}\ e : \tau_1;S'}$$

$$\frac{\Gamma;S \vdash e : \tau_1 \times \tau_2;S'}{\Gamma;S \vdash \mathtt{snd}\ e : \tau_2;S'} \qquad \frac{\Gamma;S \vdash e : \tau[\mu\alpha.\tau/\alpha];S'}{\Gamma;S \vdash \mathtt{fold}\ e : \mu\alpha.\tau;S'}$$

$$\frac{\Gamma;S \vdash e : \mu\alpha.\tau;S'}{\Gamma;S \vdash \mathtt{unfold}\ e : \tau[\mu\alpha.\tau/\alpha];S'}$$

$$\frac{\Gamma;S_1 \vdash \hat{e}_1 : \tau;S_2 \qquad \Gamma \vdash (S_2, \varphi) \Downarrow (S_3, \varphi') \qquad \Gamma, x{:}\tau, \nu{:}\varphi';S_3 \vdash e_2{:}\tau_2;S_4}{\Gamma;S_1 \vdash \mathtt{let}\ x\{\nu\}{:}\tau = \hat{e}_1\{\varphi\}\ \mathtt{in}\ e_2 : \tau_2;S_4}$$

$$\frac{\Gamma \vdash (S_1, \varphi_1) \Downarrow (S_2, n_1) \qquad \Gamma \vdash (S_2, \varphi_2) \Downarrow (S_3, n_2) \qquad n_3 = n_1 - n_2}{\Gamma \vdash (S_1, \delta^2(-, \varphi_1, \varphi_2)) \Downarrow (S_3, n_3)}$$

Sample Compilers

Index$^\pm$ ⇝ Phantom

$$\begin{array}{rcl}
(n : +0)^+ & \rightsquigarrow & n\{\langle\!\langle +0 \rangle\!\rangle\} \\
(n : -)^+ & \rightsquigarrow & n\{\langle\!\langle - \rangle\!\rangle\} \\
(e_1 + e_2 : \eta)^+ & \rightsquigarrow & ({e_1}^+ + {e_2}^+)\{\langle\!\langle \eta \rangle\!\rangle\} \\
(e_1 * e_2 : \eta)^+ & \rightsquigarrow & ({e_1}^+ * {e_2}^+)\{\langle\!\langle \eta \rangle\!\rangle\} \\
(x : \tau)^+ & = & x\{\delta^1(\mathtt{assert}, \delta^2(\mathtt{sexp=}, \langle\!\langle \tau \rangle\!\rangle, \nu_x)); \langle\!\langle \tau \rangle\!\rangle\} \\
(\lambda x{:}\tau.e : \tau \to \tau')^+ & = & (\lambda x{:}\tau^+.\mathtt{let}\ _\{\nu_x\} : \mathtt{int} = 0\{\langle\!\langle \tau \rangle\!\rangle\}\ \mathtt{in}\ e^+) \\
 & & \{\langle\!\langle \tau \to \tau' \rangle\!\rangle\} \\
(e\ e' : \tau))^+ & \rightsquigarrow & \mathtt{let}\ f\{\nu_f\} : (\tau' \to \tau)^+ = e^+\ \mathtt{in} \\
 & & \mathtt{let}\ a\{\nu_a\} : \tau'^+ = e'^+\ \mathtt{in} \\
 & & (f\ a)\{\mathtt{match}\ \nu_f \\
 & & \mid(\text{'}\mapsto\text{'}, (\nu_{\tau'}, \nu_\tau)). \\
 & & \delta^1(\mathtt{assert}, \delta^2(\mathtt{sexp=}, \nu_{\tau'}, \nu_a)); \langle\!\langle \tau \rangle\!\rangle\} \\
(\Lambda\alpha.e : \forall\alpha.\tau)^+ & \rightsquigarrow & e^+\{\langle\!\langle \forall\alpha.\tau \rangle\!\rangle\} \\
(e[\eta] : \tau[\eta/\alpha])^+ & \rightsquigarrow & e^+\{\langle\!\langle \tau[\eta/\alpha] \rangle\!\rangle\}
\end{array}$$

$$\begin{array}{rcl}
\langle\!\langle \forall\alpha.\tau \rangle\!\rangle & = & (\text{'}\forall\text{'}, (\alpha, \langle\!\langle \tau \rangle\!\rangle)) \\
\langle\!\langle \tau \to \tau' \rangle\!\rangle & = & (\text{'}\mapsto\text{'}, (\langle\!\langle \tau \rangle\!\rangle, \langle\!\langle \tau' \rangle\!\rangle)) \\
\langle\!\langle \alpha \rangle\!\rangle & = & \text{'}\alpha\text{'} \\
\langle\!\langle +0 \rangle\!\rangle & = & \text{'}+0\text{'} \\
\langle\!\langle - \rangle\!\rangle & = & \text{'}-\text{'} \\
\langle\!\langle ? \rangle\!\rangle & = & \text{'}?\text{'} \\
\langle\!\langle \eta_1 + \eta_2 \rangle\!\rangle & = & (\text{'}+\text{'}, (\langle\!\langle \eta_1 \rangle\!\rangle, \langle\!\langle \eta_2 \rangle\!\rangle)) \\
\langle\!\langle \eta_1 * \eta_2 \rangle\!\rangle & = & (\text{'}*\text{'}, (\langle\!\langle \eta_1 \rangle\!\rangle, \langle\!\langle \eta_2 \rangle\!\rangle))
\end{array}
\qquad
\begin{array}{rcl}
(\forall\alpha.\mathtt{tau})^+ & = & \tau^+ \\
(\tau \to \tau')^+ & = & \tau^+ \to \tau'^+ \\
\eta^+ & = & \mathtt{int}
\end{array}$$

BabyDill ⇝ Phantom

$$\begin{array}{rcl}
(x : \tau)^+ & \rightsquigarrow & x\{\} \\
(a : \tau)^+ & \rightsquigarrow & a\{a := \delta^2(-, !a, 1)\} \\
(n : \mathtt{int})^+ & \rightsquigarrow & n\{\} \\
(\lambda a : \tau_1.e)^+ & \rightsquigarrow & \lambda a : {\tau_1}^+.\mathtt{let_}\{a\} = 0\{\mathtt{ref}\ 1\}\ \mathtt{in} \\
 & & \mathtt{let_r}\{\} = e^+\ \mathtt{in} \\
 & & \mathtt{_r}\{\delta^1(\mathtt{assert}, \delta^2(\mathtt{sexp=}, !a, 0));\} \\
(e\ e')^+ & \rightsquigarrow & e^+\ e'^+\{\} \\
(!e)^+ & \rightsquigarrow & e^+\{\} \\
(\mathtt{let}\ !x = e{:}\tau\ \mathtt{in}\ e')^+ & \rightsquigarrow & \mathtt{let}\ x\{\} : \tau^+ = e^+\ \mathtt{in}\ e'^+\{\} \\
(\Gamma; a_1 : \tau_1 \ldots \vdash & \rightsquigarrow & \mathtt{let}\ v_1\{\} = {e_1}^+\ \mathtt{in} \\
\langle e_1, e_2 \rangle : \tau_1\&\tau_2)^+ & & \mathtt{let_}\{\} = 0\{\delta^1(\mathtt{assert}, \delta^2(\mathtt{sexp=}, !a_1, 0)) \\
 & & \ldots; a_1 := 1 \ldots\}\ \mathtt{in}\ (v_1, {e_2}^+)\{\} \\
(e.1)^+ & \rightsquigarrow & \mathtt{fst}\ e^+\{\} \\
(e.2)^+ & \rightsquigarrow & \mathtt{snd}\ e^+\{\} \\
((e_1, e_2))^+ & \rightsquigarrow & ({e_1}^+, {e_2}^+)\{\} \\
(\mathtt{let}\ (a, a') = e & \rightsquigarrow & \mathtt{let}\ t\{\}{:}(\tau_1 \otimes \tau_2)^+ = e^+\ \mathtt{in} \\
\mathtt{in}\ e')^+ & & \mathtt{let}\ a\{a\}{:}{\tau_1}^+ = \mathtt{fst}\ t\{\mathtt{ref}\ 1\}\ \mathtt{in} \\
 & & \mathtt{let}\ a'\{a'\}{:}{\tau_2}^+ = \mathtt{snd}\ t\{\mathtt{ref}\ 1\} \\
 & & \mathtt{in}\ e'^+\{\delta^1(\mathtt{assert}, \delta^2(\mathtt{sexp=}, !a, 0)); \\
 & & \quad \delta^1(\mathtt{assert}, \delta^2(\mathtt{sexp=}, !a', 0))\}
\end{array}$$

$$\begin{array}{rcl}
\mathtt{int}^+ & = & \mathtt{int} \\
(!\tau)^+ & = & \tau^+ \\
(\tau_1 \otimes \tau_2)^+ & = & {\tau_2}^+ \times {\tau_2}^+
\end{array}
\qquad
\begin{array}{rcl}
(\tau \multimap \tau')^+ & = & \tau^+ \to \tau'^+ \\
(\tau_1\&\tau_2)^+ & = & {\tau_2}^+ \times {\tau_2}^+
\end{array}$$

Discussion

- Particular phantom contract encodings are *only* given meaning by compilers, since $\{\varphi\}$ can be attached to *any* term $\hat{e}$.
- But given a source language A, we can define:

$$\mathcal{E}_A[\![\tau]\!] = \{\hat{e}\{\varphi\} \mid \text{acts as } \tau \text{, with } A\text{-compiler encoding}\}$$

- Then two languages A and B are *safe-to-link* if:

$$\forall e_A, e_B.\ e_A^+ \bowtie e_B^+ \text{ links w/o error} \implies e_B^+ \in \bigcup_\tau \mathcal{E}_A[\![\tau]\!]$$

i.e., linkable B output is expressible as A behavior.

- But could also design a specialized T and $\mathcal{E}_T[\![\tau]\!]$ as a rich low-level interface.
- **Phantom contracts** enable this design process without needing to change the target language.

$$\frac{\nu{:}\varphi \in \Gamma}{\Gamma \vdash (S, \nu) \Downarrow (S, \varphi)} \qquad \frac{\Gamma \vdash (S, \varphi) \Downarrow (S', \mathtt{true})}{\Gamma \vdash (S, \delta^1(\mathtt{assert}, \varphi)) \Downarrow (S', \mathtt{true})}$$

$$\frac{\text{fresh } \ell}{\Gamma \vdash (S, \mathtt{ref}\ \varphi) \Downarrow (S[\ell \mapsto \varphi], \ell)} \qquad \frac{\ell \in S}{\Gamma \vdash (S, \ell := \varphi) \Downarrow (S[\ell \mapsto \varphi], 0)}$$

$$\frac{\Gamma \vdash (S, \varphi_1) \Downarrow (S', s) \qquad \Gamma, \nu{:}s \vdash (S', \varphi_2) \Downarrow (S'', \varphi_2')}{\Gamma \vdash (S, \mathtt{match}\ \varphi_1 \mid s\nu.\varphi_2) \Downarrow (S'', \varphi_2)} \qquad \frac{S[\ell] = \varphi}{\Gamma \vdash (S, !\ell) \Downarrow (S, \varphi)}$$