Observational Equivalence on low-level programs & Compositional Compiler Correctness

Chung-Kil Hur

joint work with Nick Benton

2 Dec 2009

@ PPS

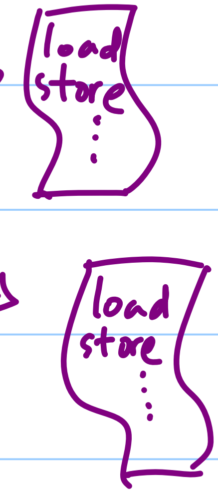
Overview

- difficulties in giving a good notion of observational equivalence
- step-indexed logical relation + biorthogonality
- Compositional Compiler Correctness
- limitations of step-indexing

Characteristics of low-level languages

- Untyped language
- Syntatic equality test

$Eq\ P_1\ P_2$



compare $\begin{cases} \text{true} & \text{if } P_1 = P_2 \\ \text{false} & \text{otherwise} \end{cases}$

Untyped λ-calculus with α-equality test

- $U\lambda\alpha$

$$
\begin{aligned}
t \; ::= \; & x \mid \lambda x.\, t \mid t\, t \mid \mathrm{rec}\; f\, x.\, t \\
\mid\; & \underline{n} \mid \mathrm{suc}\; t \mid \mathrm{pred}\; t \\
\mid\; & \mathrm{true} \mid \mathrm{false} \mid \mathrm{if}\; t\; \mathrm{then}\; t\; \mathrm{else}\; t \\
\mid\; & \mathrm{error} \\
\mid\; & =_\alpha
\end{aligned}
$$

- standard CBV left-to-right operational semantics

  value ::= terms in normal form

  N.B. $\lambda x.\, t$ is always a value.

- difficulties in giving a good notion of observational equivalence

- step-indexed logical relation + biorthogonality

- Compositional Compiler Correctness

- limitations of step-indexing

Try : Contextual equivalence

$$\boxed{t_1 \approx t_2 \quad \text{if } \forall C \quad C\,t_1 \downarrow \Leftrightarrow C\,t_2 \downarrow}$$

but

$$\lambda x.x \not\approx \lambda x.(\lambda y.y)\,x$$

because

$C := \underline{\lambda f.\ \text{if } f \approx_\alpha \lambda x.x \text{ then error else } (\text{rec } g\,x.\,g\,x)\ \underline{0}}$

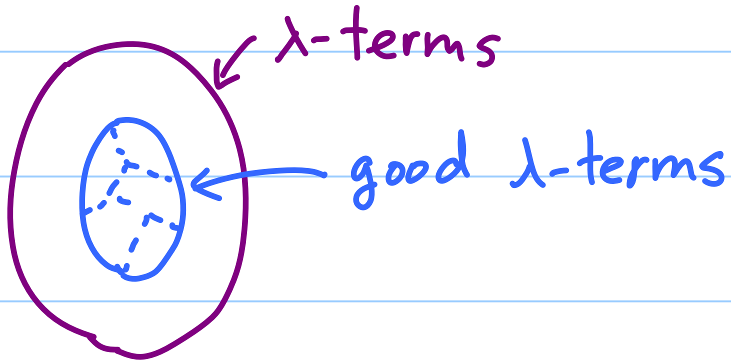
$C\,(\underline{\lambda x.x}) \downarrow$

$C\,(\underline{\lambda x.(\lambda y.y)\,x}) \uparrow$

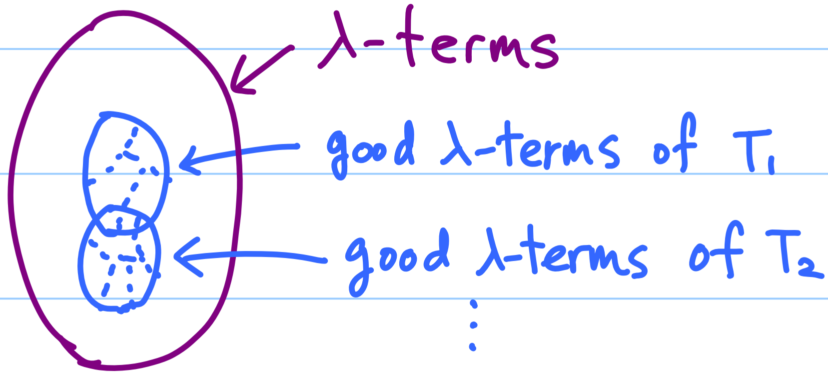
who's bad?

Ruling out Bad terms

- Untyped



- typed



Which type system?

- Simple types

$T ::= int \mid bool \mid T \to T$

What is a good equivalence?

Equivalence : $\{ [\![T]\!] \subseteq \lambda\text{-Tm} \times \lambda\text{Tm} \}_{T \in Ty}$

Notation. $t : T$ for $(t,t) \in [\![T]\!]$ $\quad t_1 \sim t_2 : T$ for $(t_1, t_2) \in [\![T]\!]$

- Adequacy
  - $t : T \Rightarrow t \not\rightsquigarrow \text{error}$ $\quad$ - $t \rightsquigarrow t' \wedge t' : T \Rightarrow t : T$
  - $t_1 \sim t_2 : B$ ← bool or int $\Rightarrow$ $t_1 \downarrow v \Leftrightarrow t_2 \downarrow v$
- Compositionality

  $t_1 \sim t_2 : S \to T \,,\; s_1 \sim s_2 : S \Rightarrow t_1 s_1 \sim t_2 s_2 : T$
- What else?

  Extensionality?? Reasoning principle?

Observation 1

Do you want to say $t_1 \sim t_2 : S \to T$ because

$\forall v_1 \sim v_2 : S,\ t_1 v_1 \sim t_2 v_2 : T$?

values

(B→B)→B→B B→B B

$F \overset{\text{def}}{=} \lambda f.\lambda g.\ \text{if } g \approx_\alpha \text{rec } h\, y.\, f\, h\, y \text{ then } \lambda x.\text{error else } g$

(B→B)→B→B B→B B→B B

$F' \overset{\text{def}}{=} \text{rec } h\, y.\ F\, h\, y$ $F'' \overset{\text{def}}{=} \text{rec } h\, y.\ F'\, h\, y$

f g x

- obs 1 : $\forall v,\ F''v \rightsquigarrow F'F''v \rightsquigarrow FF'F''v \rightsquigarrow \text{error}$ ∴ F'' is Bad!

f g

- obs 2 : $\forall v_1 \sim v_2 : B \to B,\ F'v_1 \rightsquigarrow FF'v_1 \rightsquigarrow v_1$ ∴ $F' \sim F' : (B \to B) \to B \to B$

  $F'v_2 \rightsquigarrow FF'v_2 \rightsquigarrow v_2$

(B→B)→B→B

- obs 3 : $H := \lambda f.\ \text{rec } h\, y.\ f\, h\, y$ ∴ H is Bad ???

  $H\, F'\, \text{true} \rightsquigarrow F''\, \text{true} \rightsquigarrow \text{error}$

observation 2

Do you want to say $t_1 \sim t_2 : S_1 \to \cdots \to S_n \to B$ because

$t_2 : S_1 \to \cdots \to S_n \to B$ and $\forall \underbrace{V_1 \cdots V_n}_{\text{all values including bad ones}}$

$$\begin{cases} t_1 V_1 \uparrow & \Leftrightarrow \quad t_2 V_1 \uparrow \\ t_1 V_1 V_2 \uparrow & \Leftrightarrow \quad t_2 V_1 V_2 \uparrow \\ \vdots & \qquad\quad \vdots \\ t_1 V_1 \cdots V_n \uparrow & \Leftrightarrow \quad t_2 V_1 \cdots V_n \uparrow \\ t_1 V_1 \cdots V_n \downarrow v & \Leftrightarrow \quad t_2 V_1 \cdots V_n \downarrow v \end{cases}$$

?

$F \stackrel{def}{=} \lambda f. \lambda g.$ if $g \approx_\alpha$ rec $h\, y. f\, h\, y$ then $\lambda x.$error else g

$F' \stackrel{def}{=}$ rec $h\, y. F\, h\, y$ $\qquad$ $F'' \stackrel{def}{=}$ rec $h\, y. F'\, h\, y$

$G \stackrel{def}{=} \lambda g^{:B \to B}. g$ $\Leftarrow$ should be $\lambda g. g : (B \to B) \to B \to B$

- obs : $F' \sim G : (B \to B) \to B \to B$ $\qquad$ Still F' is good H is bad

$F' V_1 \rightsquigarrow \begin{cases} \lambda x.\text{error} & \text{if } V_1 \approx_\alpha F'' \\ V_1 & \text{otherwise} \end{cases}$ $\qquad$ $G V_1 \rightsquigarrow \begin{cases} F'' & \text{if } V_1 \approx_\alpha F'' \\ V_1 & \text{otherwise} \end{cases}$

$F' V_1 V_2 \rightsquigarrow \begin{cases} \text{error} & \text{if } V_1 \approx_\alpha F'' \\ V_1 V_2 & \text{otherwise} \end{cases}$ $\qquad$ $G S_1 S_2 \rightsquigarrow \begin{cases} \text{error} & \text{if } V_1 \approx_\alpha F'' \\ V_1 V_2 & \text{otherwise} \end{cases}$

Lessons from the observations

applicative tests + extensional$_{(?)}$ observation $(\Uparrow, \Downarrow)$

is NOT enough.

Two possibilities to deal with this problem

① more intensional$_{(?)}$ observation ⟸ has some limitations due to intensionality

step-indexing $(\Downarrow_k, \Uparrow_k)$

② more tests ⟸ ideal approach, but how?

impose conditions like $\underline{t : (A \to B) \to (A \to B)}$ only when $\underline{\mathrm{rec}\ hy.\, thy : A \to B}$?

is $\underline{\mathrm{rec}}$ the only recursion? how about Y-combinator?

any term R s.t. $R\, V_1 V_2 \rightsquigarrow V_1 (R\, V_1) V_2$?

any term R s.t. $R\, V_1 V_2 \rightsquigarrow V_1 (\lambda x.\, R\, V_1 x)\, V_2$?

- difficulties in giving a good notion of observational equivalence

- step-indexed logical relation + biorthogonality

- Compositional Compiler Correctness

- limitations of step-indexing

Review : step-indexed logical relation

closed values, closed terms, open terms, open terms

we define $- \lhd_k^v = : T$, $- \lhd_k^c = : T$, $- \lhd_k^o = : T$, $- \sim = : T$

restriction, restriction

by induction on T

① from $- \lhd_k^v = : T$ to $- \lhd_k^c = : T$

$t_1 \lhd_k^c t_2 : T$ <u>iff</u> $\forall j<k\ \forall v_1,\ t_1 \Downarrow^j v_1 \Rightarrow \exists v_2,\ t_2 \Downarrow v_2 \wedge v_1 \lhd_{k-j}^v v_2 : T$

② $- \lhd_k^v = : B$

$\mathrm{true} \lhd_k^v \mathrm{true} : \mathrm{bool}$, $\mathrm{false} \lhd_k^v \mathrm{false} : \mathrm{bool}$, $\underline{n} \lhd_k^v \underline{n} : \mathrm{int}$

③ $- \lhd_k^v = : S \to T$

$f_1 \lhd_k^v f_2 : S \to T$ <u>iff</u> $\forall j<k\ \forall s_1 \lhd_j^v s_2 : S$, $f_1 s_1 \lhd_j^c f_2 s_2 : T$

④ $- \lhd_k^o = : T$ $\quad c_1 \lhd_k^o c_2 : T$ <u>iff</u> $\forall j<k\ \forall \rho_1 \lhd_j^e \rho_2$, $c_1\{\rho_1\} \lhd_j^c c_2\{\rho_2\} : T$

④ $t_1 \sim t_2 : T$ <u>iff</u> $\forall k\ \ t_1 \lhd_k^o t_2 : T \wedge t_2 \lhd_k^o t_1 : T$

✱ only applicative test , but observation $\Downarrow_k$

Revisit the example

$F \stackrel{\text{def}}{=} \lambda f.\lambda g.\ \text{if } g \approx_\alpha \text{rec } h\, y. f\, h\, y \text{ then } \lambda x.\text{error else } g$

$F' \stackrel{\text{def}}{=} \text{rec } h\, y. F\, h\, y \qquad F'' \stackrel{\text{def}}{=} \text{rec } h\, y. F'\, h\, y$

$G \stackrel{\text{def}}{=} \lambda g. g$

Recall that F' and G observationally behaves the same in any applicative test.

- one can show $G \sim G : (B \to B) \to B \to B$
- now we see $F' \not\sim F' : (B \to B) \to B \to B$

step-indexing can distinguish G and F' !!!

takes one more step

$F'' \lhd^{v}_{6} F'' : B \to B$ because $F''v \to \underline{F'F''}v \overset{4}{\rightsquigarrow} (\lambda x.\text{error})v \to \text{error}$ (takes 6 steps)

$F'F'' \not\lhd^{c}_{6} F'F'' : B \to B$ because $\underline{F'}F'' \downarrow_4 \lambda x.\text{error}$ but $\lambda x.\text{error} \not\lhd^{v}_{2} \lambda x.\text{error} : B \to B$

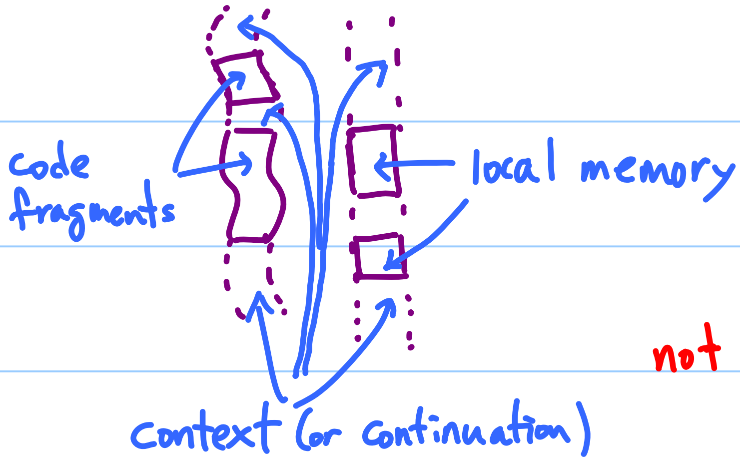
as $\text{error} \not\lhd^{v}_{1} \text{error} : B$

- Furthermore, rec, Y-combinator, ... can be shown to be good!!

structure of realistic machine languages

step-indexed logical relation works well for $u\lambda_\alpha$

but how about for assembly languages?



computation (or term) = code frags + loc. mem.

configuration = computation + context

runnable

not runnable itself (it may interact with its contexts
e.g. by calling memalloc,
even it may optimize
its context at runtime)

$\boxed{Val \subseteq Comp}$

- Conceptually ① $-[=] : Context \times Comp \to Config$

  ② we can only make observations on Config.

Realizability & biorthogonality

$$\underset{\substack{\text{Contextualizing}\\ \text{by biorthogonality}}}{L_0} \lhd \overset{\text{make observations on terms}}{\underset{\substack{\text{specification}\\ \text{denotational semantics, sufficiently expressive high-level languages} \dots}}{H_i}} , \quad L_0 \rhd H_i \rightsquigarrow L_0 \sim H_i \rightsquigarrow L_0 \sim L_0$$

Intuitively, $\ell \sim H$ captures that ℓ realizes H.

$\ell_1 \sim \ell_2$ captures that both ℓ_1 and ℓ_2 realize some H.

Basic ideas : $\lhd_k$

Idea

$\ell \lhd^{C}_{k} H : T$ iff $\forall j<k\ \forall v,\ \ell \downarrow_j v \Rightarrow \exists V,\ H \downarrow V \wedge v \lhd^{V}_{k-j} V : T$

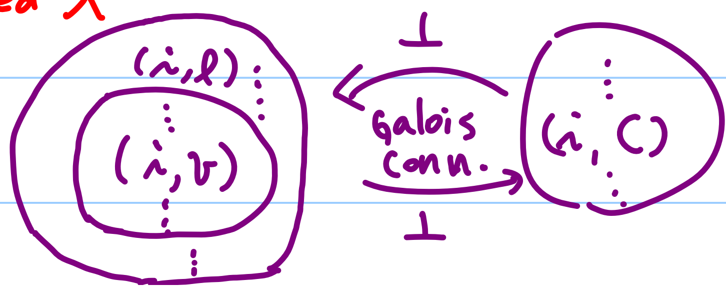
iff $\begin{cases} H\uparrow \Rightarrow \ell \uparrow_k \xrightarrow{\text{contextualizing}} \forall C,\ C[\ell]\uparrow_k \\ H\downarrow V \Rightarrow \forall j<k\ \forall v,\ \ell\downarrow_j v \Rightarrow v \lhd^{V}_{k-j} V : T \end{cases}$

contextualizing $\rightarrow$ $\forall C,\ (\underline{\forall i<k\ \forall v,\ v \lhd^{V}_{i} V : T \Rightarrow C[v]\uparrow_i}) \Rightarrow C[\ell]\uparrow_k$

$(k,\ell) \in \{(i,v) \mid \exists V, H\downarrow V \wedge v \lhd^{V}_{i} V : T\}^{\perp\perp}$

Def $\ell \lhd^{C}_{k} H : T$ iff $\forall C,\ (\forall V,\ H\downarrow V \Rightarrow \quad '' \quad) \Rightarrow C[\ell]\uparrow_k$

in mind $\quad u\lambda_\alpha \quad$ simply typed λ



Step-indexing + Biorthogonality : $\lhd_k$

$$\lhd_k^V \rightsquigarrow \lhd_k^C \rightsquigarrow \lhd_k^O \rightsquigarrow \lll$$

- $\lhd_k^V \rightsquigarrow \lhd_k^C$: $\ell \lhd_k^C H : T$ iff $(k, \ell) \in \{(i, v) \mid \exists V, H \Downarrow V \wedge v \lhd_i^V V : T\}^{\perp\perp}$
- $\lhd_k^V : \mathbb{B}$: $\mathtt{true} \lhd_k^V \mathtt{TRUE} : \mathrm{bool}$, $\mathtt{false} \lhd_k^V \mathtt{FALSE} : \mathrm{bool}$, $\underline{n} \lhd_k^V \underline{n} : \mathrm{int}$
- $\lhd_k^V : S \to T$: $f \lhd_k^V F : S \to T$ iff $\forall j < k\ \forall v \lhd_j^V V : S,\ \mathrm{app}(f, v_i) \lhd_k^C F V : T$

  ↑ calling convention in low

  this may vary depending on structure of low
- $\lhd_k^O : T$: $\ell \lhd_k^O H : T$ iff $\forall j < k\ \forall \rho \lhd_j^e \psi,\ \ell\{\rho\} \lhd_k^C H\{\psi\}$
- $\lll : T$: $\ell \lll H$ iff $\exists H', H' \sqsubseteq H \wedge \forall k\ \ell \lhd_k^O H'$

  Contextual approximation in High

step-indexing & Biorthogonality : $\rhd$

- standard approach

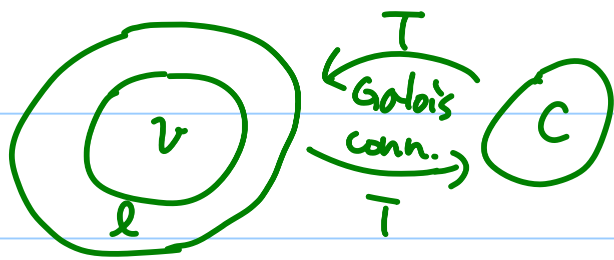
step-indexing on High

$\boxed{\text{idea}}$ $\ell \rhd^{c}_{k} H : T$ iff $\forall j<k\ \forall V,\ H \downarrow_j V \Rightarrow \exists v,\ \ell \downarrow v \wedge v \rhd^{v}_{k-j} V : T$

Contextualizing

$\forall C,\ (\forall v \rhd^{v}_{k-j} V{:}T,\ C[v]\downarrow) \Rightarrow C[\ell]\downarrow$

$\boxed{\text{Def}}$ $\ell \rhd^{c}_{k} H : T$ iff $\forall j<k\ \forall V,\ H \downarrow_j V \Rightarrow \ell \in \{ v \mid v \rhd^{v}_{k-j} V : T \}^{\top\top}$



Step-indexing & biorthogonality : $\triangleright$

- Our approach (what if High is denotational semantics?)

$$\ell \triangleright^c H : T \quad \underline{\text{iff}} \quad \forall V,\ H \Downarrow V \Rightarrow \ell \in \{ v \mid v \triangleright^v V : T \}^{\top\top}$$

↳ o.K for both den. & op. sem (standard log. rel. + biorthogonality)

↳ doesn't work for rec due to $=_\alpha$ (c.f. Pitt's TT closure)

Then

① Den: chain lub

$$\ell \ggg H : T \quad \underline{\text{iff}} \quad \underbrace{\exists \{H_i\}_{i \in \mathbb{N}} \text{ s.t. } \lim H_i \geq H}_{} \wedge \forall i,\ \ell \triangleright H_i : T$$

Q: operational notion of lim?

② op : $\{H_i\}_{i \in \mathbb{N}}$ s.t. $\forall i$ $H_i \blacktriangleright_i H$

standard step-indexed logical relation in High

- $\sim$: $\quad \ell \sim H : T \quad \underline{\text{iff}} \quad \ell \lll H : T \ \wedge\ \ell \ggg H : T$

Results (formalized & verified in Coq)

1. Den. Sem.

simple types

SECD with Eq $\sim$ $[\![T]\!] \in \omega Cpo : T$ [ICFP 09]

2. Op. sem.

simple types + $\forall$ + $\exists$

SECD with Eq $\sim$ SysF with vec & $\exists$-type $: T$ [submitted]

properties

① adequacy :

$$\ell \sim H : B \Rightarrow \big(H \downarrow v \text{ iff } \mathrm{contextualize}(\ell \downarrow v) \big)$$

② Compositionality :

$$f \sim F : S \to T \ \wedge \ a \sim A : S \Rightarrow \mathrm{app}(f, a) \sim FA : T$$

- difficulties in giving a good notion of observational equivalence
- step-indexed logical relation + biorthogonality
- Compositional Compiler Correctness
- limitations of step-indexing

Compositional Compiler Correctness (formalized in Coq)

toy compiler

$(\!|-|\!) \;:\; \mathrm{SysF} + \mathrm{rec} + \exists \longrightarrow \mathrm{SECD} + \mathrm{Eq}$

doing tailcall optimization

Theorem

$\forall t : T \quad (\!| t |\!) \sim t : T$

Compositionality

$(\!| t |\!) : S_1 \to S_2 \to T$

$(\!| t' |\!) : S_1$

$\ell : S_2$

↑ written by hand

if $(\!| t |\!) \sim t$

$(\!| t' |\!) \sim t'$

$\ell \sim t''$

then $\mathrm{app}(\mathrm{app}((\!| t |\!), (\!| t' |\!)), \ell) \sim t\,t'\,t''$

Example: Fixpoint Combinator (formalized in Coq)

$\mathrm{Fix} := \Lambda X. \Lambda Y. \lambda F : (X \to Y) \to X \to Y.\ \mathrm{Rec}\ f\, x.\ F\, f\, x$

$Y :=$ SECD code directly encoding

$\lambda F. \lambda x. (\lambda y. F(\lambda z. y\, y\, z))(\lambda y. F(\lambda z. y\, y\, z))\, x$

Theorem

$Y \sim \mathrm{Fix} : \forall X \forall Y\, ((X \to Y) \to X \to Y) \to X \to Y$

Example : Polymorphic List Module (formalized in Coq)

$$\mathrm{SigPolList} := \forall X.\ \exists LX.\ LX \times (X \times LX \to LX) \times (LX \to \mathrm{Option}\,(X \times LX))$$

- Implementation in Source

$$\mathrm{List}\ \tau := \forall Y.\ Y \times (\tau \times Y \to Y) \to Y$$

$$\mathrm{nil}_\tau := \cdots \quad : \mathrm{List}\ \tau$$

$$\mathrm{cons}_\tau := \cdots \quad : \tau \times \mathrm{List}\ \tau \to \mathrm{List}\ \tau$$

$$\mathrm{split}_\tau := \cdots \quad : \mathrm{List}\ \tau \to \mathrm{option}\,(\tau \times \mathrm{List}\ \tau)$$

very inefficient (split is in $\Theta(n)$), but best implementation due to lack of recursive types.

$$\mathrm{LST} := \Lambda X.\ \mathrm{Pack}\,\{\mathrm{List}\ X,\ (\mathrm{nil}_X, \mathrm{cons}_X, \mathrm{split}_X)\} \quad : \mathrm{SigPolList}$$

- Heavy-optimized implementation in SECD

using IsNum command

$$\mathrm{encode}\ vs = \begin{cases} 0 & \text{if } vs = [\,] \\ 2^n \times 3^m & \text{if } vs = \underline{n} :: tl \ \wedge\ \mathrm{encode}\ tl = \underline{m} \\ \mathrm{PR}(hd, \mathrm{encode}\ tl) & \text{otherwise } (vs = hd :: tl) \end{cases}$$

$$\mathrm{NIL} = \underset{\sim}{0} \quad \mathrm{CONS} = [\mathrm{Push}\,C \cdots] \quad \mathrm{SPLIT} = [\mathrm{Push}\,C \cdots]$$

- Proposition : $(\mathrm{NIL} +\!\!+ \mathrm{CONS} +\!\!+ [\mathrm{MkPair}] +\!\!+ \mathrm{SPLIT} +\!\!+ [\mathrm{MkPair}], \mathrm{nil}) \sim \mathrm{LST} : \mathrm{SigPolList}$

Example : optimizing iteration (formalized in Coq)

- SysF term appn : (int→int)→int→int→int $\stackrel{def}{=}$ appn f n v = $f^n v$
- hand-optimized implementation of appn

appnoptcode = [pushC ...]

$\lambda f.\lambda n.\lambda v.$ if Eq(f, ⦇ $\lambda x.x$ ⦈) syntactic Eq test
then v
else ⦇ appn ⦈ f n v

Theorem

(appnoptcode, nil) ~ appn : (int→int)→int→int→int

- difficulties in giving a good notion of observational equivalence
- step-indexed logical relation + biorthogonality
- Compositional Compiler Correctness
- limitations of step-indexing

Limitations of step-indexing

We proved

$$\text{appnoptcode} \sim \text{appn} : (\text{int} \to \text{int}) \to \text{int} \to \text{int} \to \text{int}$$

But

$$\lambda f. \lambda n. \lambda v.\ \text{if } Eq(f, id_{100}) \text{ then } v \text{ else } f^n v$$

$$\not\sim \text{appn} : (\text{int} \to \text{int}) \to \text{int} \to \text{int} \to \text{int}$$

for $id_{100} := \lambda x.\ \text{donothing } 100 ;\ x$

Step-indexing rules out all bad programs usin Eq,

but also many good programs using Eq.

Limitations of step-indexing : example

Recall $-\triangleleft_k =: \mathcal{T}$ between $u\lambda_\alpha$

$\mathrm{id}_{100} := \lambda x.\,(\lambda x.\,(\lambda x. \cdots \overset{100\text{ times}}{\downarrow} (\lambda x.\,x)\,x)\,x) \cdots)\,x$

$\mathrm{applopt} := \lambda f.\ \mathrm{if}\ f =_\alpha \mathrm{id}_{100}\ \mathrm{then}\ \lambda x.\,x\ \mathrm{else}\ \lambda x.\,f\,x$

$\mathrm{appl} := \lambda f.\,\lambda x.\,f\,x$

Claim: $\mathrm{applopt}\ \cancel{\triangleleft}^{v}_{51}\ \mathrm{appl} : (\mathrm{int} \to \mathrm{int}) \to \mathrm{int} \to \mathrm{int}$

① $\mathrm{id}_{100} \triangleleft^{v}_{50} \mathrm{rec}\ f\,x.\,f\,x$ because $\forall v\ \ \mathrm{id}_{100}\,v \uparrow_{50}$

② $\mathrm{applopt} \cdot \mathrm{id}_{100} \downarrow_2 \lambda x.\,x$, $\mathrm{appopt} \cdot (\mathrm{rec}\ f\,x.\,f\,x) \downarrow \lambda x.\,(\mathrm{rec}\ f\,x.\,f\,x)\,x$

but $\lambda x.\,x\ \cancel{\triangleleft}^{v}_{48}\ \lambda x.\,(\mathrm{rec}\ f\,x.\,f\,x)\,x$ because $(\lambda x.\,x)\,\underline{0} \downarrow_1 \underline{0}$

, but $(\mathrm{rec}\ f\,x.\,f\,x)\,\underline{0} \uparrow$

Discussion & future work

- Discussion
  - 12000 lines in Coq ← Strongly typed representation with JMeq
  - first compiler correctness result for Polymorphic Language!
- Future work
  - recursive types
  - effects (references, exceptions, input & output, ...)
  - realistic assembly language
  - more extensional equivalence & realization without using step-indexes
- Related work

  Recent draft of Adam Chlipala (Oct 2009) proposes Syntactic Compositional Compiler Correctness.

  problem with polymorphism (parametricity)

  ↳ now computational adequacy + compositionality

  ↳ simple, easy to implement, but far from extensionality.

Thank you !!