Contracts Made Manifest

Michael Greenberg
Benjamin C. Pierce   Stephanie Weirich

University of Pennsylvania

2010-01-22 / POPL '10

First-order contracts

$\text{assert}(n > 0)$

First-order contracts

$\text{assert}(n > 0)$

$\text{sqrt} : \{x{:}\text{Float} \mid x \geq 0\} \mapsto \text{Float}$

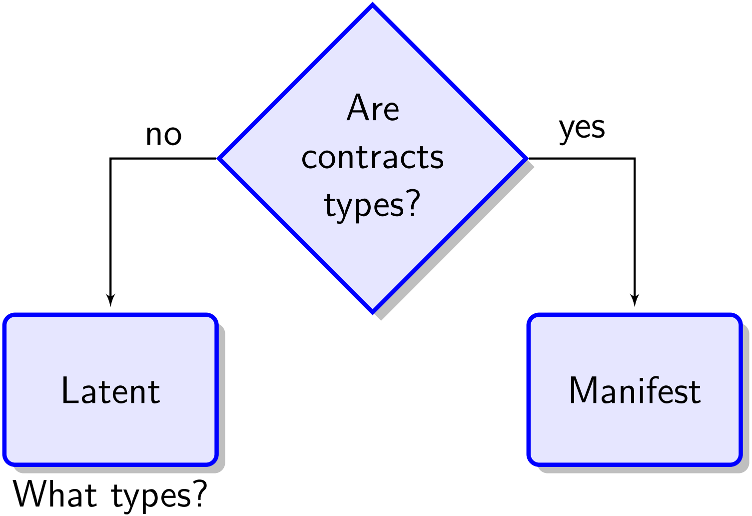
First-order contracts

$\text{assert}(n > 0)$

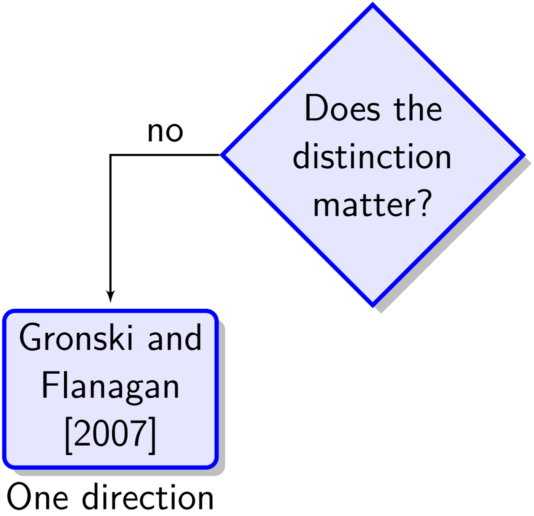
$\text{sqrt} : \{x{:}\text{Float} \mid x \geq 0\} \mapsto \text{Float}$

$\text{sqrt} : x{:}\{x{:}\text{Float} \mid x \geq 0\} \mapsto \{y{:}\text{Float} \mid \left|y^2 - x\right| < \epsilon\}$

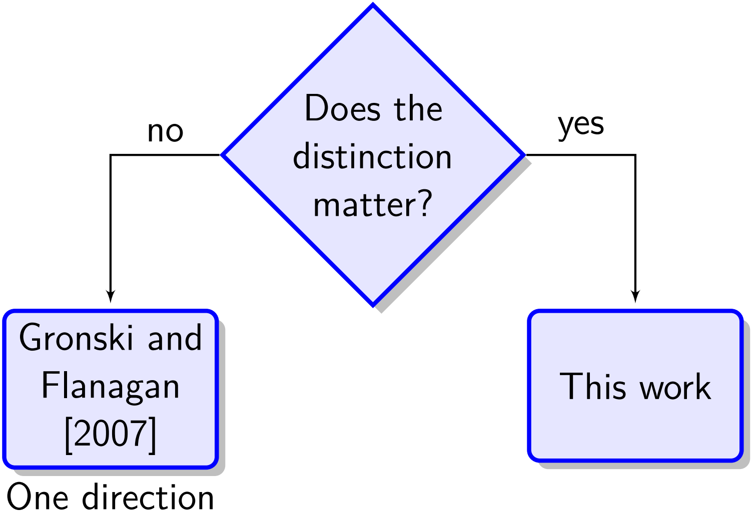
Contracts for the λ-calculus



Contracts for the λ-calculus



Contracts for the λ-calculus



Blame assignment

$$\begin{aligned} f &: \mathsf{Int} \to (\mathsf{Int} \to \mathsf{Int}) \to \mathsf{Int} \\ f &= \lambda n.\ \lambda g.\ (g\ n) \end{aligned}$$

If we give f the contract

$$\mathsf{Int} \mapsto \big(\{x{:}\mathsf{Int} \mid x > 0\} \mapsto \{y{:}\mathsf{Int} \mid y > 0\}\big) \mapsto \mathsf{Int}$$

How does $(f\ 0)\ \lambda x.\ 1$ evaluate?

Blame assignment

$$\begin{array}{rcl} f & : & \mathsf{Int} \to (\mathsf{Int} \to \mathsf{Int}) \to \mathsf{Int} \\ f & = & \lambda n.\ \lambda g.\ (g\ n) \end{array}$$

If we give f the contract

$$\mathsf{Int} \mapsto \big(\{x{:}\mathsf{Int} \mid x > 0\} \mapsto \{y{:}\mathsf{Int} \mid y > 0\}\big) \mapsto \mathsf{Int}$$

How does $(f\ 0)\ \lambda x.\ 1$ evaluate?

What about $(f\ 1)\ \lambda x.0$?

What about $(f\ 0)\ \lambda x.0$?

Latent contracts

According to Findler and Felleisen [2002]

$$
\begin{array}{rcll}
c & ::= & \{x{:}B \mid t\} & \text{base contracts} \\
  & \mid & c_1 \mapsto c_2 & \text{function contracts}
\end{array}
$$

Latent contracts

According to Findler and Felleisen [2002]

$$
\begin{array}{lcll}
c & ::= & \{x{:}B \mid t\} & \text{base contracts} \\
  & \mid & c_1 \mapsto c_2 & \text{function contracts} \\
\\
t & ::= & \ldots & \\
  & \mid & \langle c \rangle^{l,l'} & \text{contract obligations} \\
  & \mid & \Uparrow l & \text{blame}
\end{array}
$$

Latent contracts

According to Findler and Felleisen [2002]

$$
\begin{array}{rcll}
c & ::= & \{x{:}B \mid t\} & \text{base contracts} \\
  & \mid & c_1 \mapsto c_2 & \text{function contracts}
\end{array}
$$

$$
\begin{array}{rcll}
t & ::= & \ldots & \\
  & \mid & \langle c \rangle^{l,l'} & \text{contract obligations} \\
  & \mid & \Uparrow l & \text{blame}
\end{array}
$$

Can't in general *decide* whether a function is, e.g. $\mathtt{Pos} \mapsto \mathtt{Pos}$

Instead, defer checking to runtime

Check that argument, result satisfy contracts

Higher-order contracts

Let $\texttt{Pos}$ mean $\{x{:}\mathsf{Int} \mid x > 0\}$

$$\begin{aligned}
\langle\texttt{Pos}\rangle^{l,l'}\,1 &\longrightarrow^* 1\\
\langle\texttt{Pos}\rangle^{l,l'}\,0 &\longrightarrow^* \Uparrow l
\end{aligned}$$

$$\begin{aligned}
(\langle\texttt{Pos} \mapsto \texttt{Pos}\rangle^{l_{\mathsf{fun}},l_{\mathsf{arg}}}\,\lambda x{:}\mathsf{Int}.\ x)\,1 &\longrightarrow^* 1\\
(\langle\texttt{Pos} \mapsto \texttt{Pos}\rangle^{l_{\mathsf{fun}},l_{\mathsf{arg}}}\,\lambda x{:}\mathsf{Int}.\ x)\,0 &\longrightarrow^* \Uparrow l_{\mathsf{arg}}\\
(\langle\texttt{Pos} \mapsto \texttt{Pos}\rangle^{l_{\mathsf{fun}},l_{\mathsf{arg}}}\,\lambda x{:}\mathsf{Int}.\ x - 1)\,1 &\longrightarrow^* \Uparrow l_{\mathsf{fun}}
\end{aligned}$$

Function contracts

$$
\begin{array}{rcl}
(\langle \texttt{Pos} \mapsto \texttt{Pos} \rangle^{l_{\mathsf{fun}}, l_{\mathsf{arg}}}\, \lambda x{:}\mathsf{Int}.\ x)\, 0 & \longrightarrow & \\
\langle \texttt{Pos} \rangle^{l_{\mathsf{fun}}, l_{\mathsf{arg}}}\, ((\lambda x{:}\mathsf{Int}.\ x)\, (\langle \texttt{Pos} \rangle^{l_{\mathsf{arg}}, l_{\mathsf{fun}}}\, 0)) & \longrightarrow^{*} & \\
\langle \texttt{Pos} \rangle^{l_{\mathsf{fun}}, l_{\mathsf{arg}}}\, ((\lambda x{:}\mathsf{Int}.\ x)\, \Uparrow l_{\mathsf{arg}}) & \longrightarrow^{*} & \Uparrow l_{\mathsf{arg}}
\end{array}
$$

Function contracts

$$
\begin{array}{rll}
(\langle \mathtt{Pos} \mapsto \mathtt{Pos} \rangle^{l_{\mathsf{fun}}, l_{\mathsf{arg}}}\, \lambda x{:}\mathsf{Int}.\ x)\, 0 & \longrightarrow & \\
\langle \mathtt{Pos} \rangle^{l_{\mathsf{fun}}, l_{\mathsf{arg}}}\, ((\lambda x{:}\mathsf{Int}.\ x)\, (\langle \mathtt{Pos} \rangle^{l_{\mathsf{arg}}, l_{\mathsf{fun}}}\, 0)) & \longrightarrow^{*} & \\
\langle \mathtt{Pos} \rangle^{l_{\mathsf{fun}}, l_{\mathsf{arg}}}\, ((\lambda x{:}\mathsf{Int}.\ x)\, \Uparrow l_{\mathsf{arg}}) & \longrightarrow^{*} & \Uparrow l_{\mathsf{arg}}
\end{array}
$$

$$
\begin{array}{rll}
(\langle \mathtt{Pos} \mapsto \mathtt{Pos} \rangle^{l_{\mathsf{fun}}, l_{\mathsf{arg}}}\, \lambda x{:}\mathsf{Int}.\ x - 1)\, 1 & \longrightarrow & \\
\langle \mathtt{Pos} \rangle^{l_{\mathsf{fun}}, l_{\mathsf{arg}}}\, ((\lambda x{:}\mathsf{Int}.\ x - 1)\, (\langle \mathtt{Pos} \rangle^{l_{\mathsf{arg}}, l_{\mathsf{fun}}}\, 1)) & \longrightarrow^{*} & \\
\langle \mathtt{Pos} \rangle^{l_{\mathsf{fun}}, l_{\mathsf{arg}}}\, ((\lambda x{:}\mathsf{Int}.\ x - 1)\, 1) & \longrightarrow^{*} & \\
\langle \mathtt{Pos} \rangle^{l_{\mathsf{fun}}, l_{\mathsf{arg}}}\, 0 & \longrightarrow^{*} & \Uparrow l_{\mathsf{fun}}
\end{array}
$$

Function contracts

$$
\begin{array}{rll}
(\langle \texttt{Pos} \mapsto \texttt{Pos} \rangle^{l_{\text{fun}}, l_{\text{arg}}}\ \lambda x{:}\text{Int}.\ x)\ 0 & \longrightarrow & \\
\langle \texttt{Pos} \rangle^{l_{\text{fun}}, l_{\text{arg}}}\ ((\lambda x{:}\text{Int}.\ x)\ (\langle \texttt{Pos} \rangle^{l_{\text{arg}}, l_{\text{fun}}}\ 0)) & \longrightarrow^{*} & \\
\langle \texttt{Pos} \rangle^{l_{\text{fun}}, l_{\text{arg}}}\ ((\lambda x{:}\text{Int}.\ x)\ \Uparrow l_{\text{arg}}) & \longrightarrow^{*} & \Uparrow l_{\text{arg}}
\end{array}
$$

$$
\begin{array}{rll}
(\langle \texttt{Pos} \mapsto \texttt{Pos} \rangle^{l_{\text{fun}}, l_{\text{arg}}}\ \lambda x{:}\text{Int}.\ x - 1)\ 1 & \longrightarrow & \\
\langle \texttt{Pos} \rangle^{l_{\text{fun}}, l_{\text{arg}}}\ ((\lambda x{:}\text{Int}.\ x - 1)\ (\langle \texttt{Pos} \rangle^{l_{\text{arg}}, l_{\text{fun}}}\ 1)) & \longrightarrow^{*} & \\
\langle \texttt{Pos} \rangle^{l_{\text{fun}}, l_{\text{arg}}}\ ((\lambda x{:}\text{Int}.\ x - 1)\ 1) & \longrightarrow^{*} & \\
\langle \texttt{Pos} \rangle^{l_{\text{fun}}, l_{\text{arg}}}\ 0 & \longrightarrow^{*} & \Uparrow l_{\text{fun}}
\end{array}
$$

Function contract obligations

$$
(\langle c_1 \mapsto c_2 \rangle^{l,l'}\ v_1)\ v_2 \longrightarrow \langle c_2 \rangle^{l,l'}\ (v_1\ (\langle c_1 \rangle^{l',l}\ v_2))
$$

Dependency

Nondependent

$$(\langle c_1 \mapsto c_2 \rangle^{l,l'}\, v_1)\, v_2 \longrightarrow \langle c_2 \rangle^{l,l'}\, (v_1\, (\langle c_1 \rangle^{l',l}\, v_2))$$

Dependent

$$\langle c_2\{x := v_2\} \rangle^{l,l'}\, (v_1\, (\langle c_1 \rangle^{l',l}\, v_2))$$

lax

$$(\langle x{:}c_1 \mapsto c_2 \rangle^{l,l'}\, v_1)\, v_2$$

picky

$$\langle c_2\{x := \langle c_1 \rangle^{l',l}\, v_2\} \rangle^{l,l'}\, (v_1\, (\langle c_1 \rangle^{l',l}\, v_2))$$

Dependency

$$f\ n \;=\; \langle g{:}(\texttt{Pos} \mapsto \texttt{Pos}) \mapsto \{z{:}\mathsf{Int} \mid z = g\,0\}\rangle^{l_f,l_g}$$
$$(\lambda g{:}(\mathsf{Int} \to \mathsf{Int}).\ g\ n)$$

$$(f\ 1)\ \lambda x{:}\mathsf{Int}.\ 1 \qquad \longrightarrow$$
$$(\langle g{:}(\texttt{Pos} \mapsto \texttt{Pos}) \mapsto \{z{:}\mathsf{Int} \mid z = g\,0\}\rangle^{l_f,l_g} \qquad g := ?$$
$$(\lambda g{:}(\mathsf{Int} \to \mathsf{Int}).\ g\ 1))\ \lambda x{:}\mathsf{Int}.\ 1 \qquad \longrightarrow$$

Dependency

$$f\ n \ = \ \langle g{:}(\mathtt{Pos} \mapsto \mathtt{Pos}) \mapsto \{z{:}\mathsf{Int} \mid z = g\,0\}\rangle^{l_f, l_g} \ (\lambda g{:}(\mathsf{Int} \to \mathsf{Int}).\ g\ n)$$

$$(f\ 1)\ \lambda x{:}\mathsf{Int}.\ 1 \longrightarrow$$

$$(\langle g{:}(\mathtt{Pos} \mapsto \mathtt{Pos}) \mapsto \{z{:}\mathsf{Int} \mid z = g\,0\}\rangle^{l_f, l_g} \quad g := ?$$

$$(\lambda g{:}(\mathsf{Int} \to \mathsf{Int}).\ g\,1))\ \lambda x{:}\mathsf{Int}.\ 1 \longrightarrow$$

$$\langle \{z{:}\mathsf{Int} \mid z = (\lambda x{:}\mathsf{Int}.\ 1)\,0\}\rangle^{l_f, l_g} \quad \mathsf{lax}$$

$$((\lambda g{:}\mathsf{Int} \to \mathsf{Int}.\ g\,1)\,(\langle \mathtt{Pos} \mapsto \mathtt{Pos}\rangle^{l_g, l_f}\ \lambda x{:}\mathsf{Int}.\ 1)) \longrightarrow^* 1$$

Dependency

$$f\ n \ = \ \langle g{:}(\mathtt{Pos} \mapsto \mathtt{Pos}) \mapsto \{z{:}\mathsf{Int} \mid z = g\,0\}\rangle^{l_f,l_g}$$
$$(\lambda g{:}(\mathsf{Int} \to \mathsf{Int}).\ g\ n)$$

$$(f\ 1)\ \lambda x{:}\mathsf{Int}.\ 1 \quad \longrightarrow$$
$$(\langle g{:}(\mathtt{Pos} \mapsto \mathtt{Pos}) \mapsto \{z{:}\mathsf{Int} \mid z = g\,0\}\rangle^{l_f,l_g} \quad g := ?$$
$$(\lambda g{:}(\mathsf{Int} \to \mathsf{Int}).\ g\,1))\ \lambda x{:}\mathsf{Int}.\ 1 \quad \longrightarrow$$

$$\langle\{z{:}\mathsf{Int} \mid z = (\lambda x{:}\mathsf{Int}.\ 1)\,0\}\rangle^{l_f,l_g} \quad \text{lax}$$
$$((\lambda g{:}\mathsf{Int} \to \mathsf{Int}.\ g\,1)\,(\langle \mathtt{Pos} \mapsto \mathtt{Pos}\rangle^{l_g,l_f}\ \lambda x{:}\mathsf{Int}.\ 1)) \quad \longrightarrow^{*}\ 1$$

$$\langle\{z{:}\mathsf{Int} \mid z = (\langle \mathtt{Pos} \mapsto \mathtt{Pos}\rangle^{l_g,l_f}\ \lambda x{:}\mathsf{Int}.\ 1)\,0\}\rangle^{l_f,l_g} \quad \text{picky}$$
$$((\lambda g{:}\mathsf{Int} \to \mathsf{Int}.\ g\,1)\,(\langle \mathtt{Pos} \mapsto \mathtt{Pos}\rangle^{l_g,l_f}\ \lambda x{:}\mathsf{Int}.\ 1)) \quad \longrightarrow^{*}\ \Uparrow l_f$$

Abusive contracts

An abusive contract

$$g{:}(\mathrm{Pos} \mapsto \mathrm{Pos}) \mapsto \{z{:}\mathsf{Int} \mid z = g\,0\}$$

Picky checking detects abusive contracts
Lax checking doesn't

Only higher-order contracts can be abusive

Contracts, made manifest

Based on Flanagan [2006]

Contracts = Types

$$
\begin{array}{rcll}
S & ::= & \{x{:}B \mid s\} & \text{refinements of base type} \\
  & \mid & x{:}S_1 \rightarrow S_2 & \text{function contracts}
\end{array}
$$

Contracts, made manifest

Based on Flanagan [2006]

Contracts = Types

$$S ::= \{x{:}B \mid s\} \qquad \text{refinements of base type}$$
$$\mid \quad x{:}S_1 \to S_2 \qquad \text{function contracts}$$

$$s ::= \ldots$$
$$\mid \quad \langle S_1 \Rightarrow S_2 \rangle^l \qquad \text{casts}$$
$$\mid \quad \Uparrow l \qquad \text{blame}$$

Contracts, made manifest

Based on Flanagan [2006]

Contracts = Types

Unfold function casts contravariantly; semi-picky

Choice forced by the type system

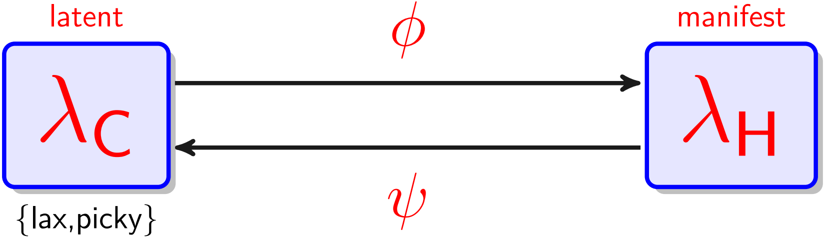
Complicated metatheory

Particularly in dependent case

Our question

Does the distinction between latent and manifest matter?

Our Work



Comparing latent calculi

Latent calculi

	FF02	BM06	HJL06	GF07 λ_C	our λ_C
dependency	✓ lax	✓ $\perp$	✓ picky	$\times$	✓ either
eval order	CBV	CBV	lazy	CBV	CBV
blame	$\Uparrow l$	$\Uparrow l$ or $\perp$	$\Uparrow l$	$\Uparrow l$	$\Uparrow l$
checking	if	active	if	$\bigcirc$	active
typing	✓	n/a	✓	✓	✓
arb. con.	✓	✓	✓	✓	✓

Legend

dependency	Dependent function contracts?	FF02	Findler and Felleisen [2002]
blame	How are failures indicated?	BM06	Blume and McAllester [2006]
checking	How are refinements checked?	HJL06	Hinze, Jeuring, and Löh [2006]
typing	Type system well-defined?	GF07	Gronski and Flanagan [2007]
arb. con.	Arbitrary user-defined contracts?		

Comparing manifest calculi

Manifest calculi

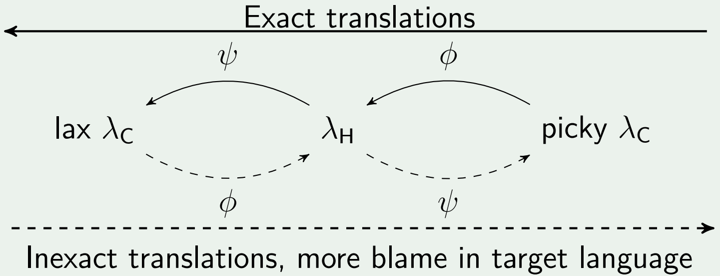
	OTMW04	F06	GF07 λ_H	KF09	WF09	our λ_H
dependency	$\checkmark$	$\checkmark$	$\times$	$\checkmark$	$\times$	$\checkmark$
eval order	CBV	NDCBN	CBV	full β	CBV	CBV
blame	$\Uparrow$	stuck	$\Uparrow l$	stuck	$\Uparrow l$	$\Uparrow l$
checking	if	$\bigcirc$	$\bigcirc$	active	active	active
typing	$\checkmark$	$\times$	$\times$	$\checkmark$	$\checkmark$	$\checkmark$
arb. con.	$\times$	$\checkmark$	$\checkmark$	$\checkmark$	$\checkmark$	$\checkmark$

Legend

dependency	Dependent function contracts?	OTMW04	Ou, Tan, Mandelbaum, and Walker [2004]
blame	How are failures indicated?	F06	Flanagan [2006]
checking	How are refinements checked?	GF07	Gronski and Flanagan [2007]
typing	Type system well-defined?	KF09	Knowles and Flanagan [2009]
arb. con.	Arbitrary user-defined contracts?	WF09	Wadler and Findler [2009]

Our answer

The axis of blame



Inexactitude due to treatment of abusive contracts

Correspondence

Nondependent	Dependent	
	First-order	Higher-order

Correspondence

Nondependent	Dependent	
	First-order	Higher-order
Exact!		

No lax/picky distinction in λ_C

Correspondence

Nondependent	Dependent	
	First-order	Higher-order
Exact!	Exact!	

No abusive contracts

Correspondence

Nondependent	Dependent	
	First-order	Higher-order
Exact!	Exact!	Inexact

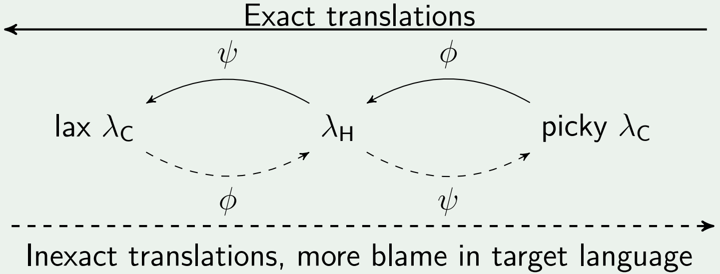
Due to abusive contracts...

Exactitude

in the higher-order dependent case

Can *add* checks to be pickier

The axis of blame



Can't *remove* checks to be laxer

None of the languages inter-translate *exactly*

Conclusion

Lax λ_{C}, λ_{H}, and picky λ_{C} are all subtly different

Not entirely clear which is the "right" one

Conclusion

Lax λ_C, λ_H, and picky λ_C are all subtly different

Not entirely clear which is the "right" one

	Latent	Manifest
Implemented		
Language	✓	×
Library	✓	N/A
Extensible	✓	×
Intuitive (to Michael Greenberg)		
Op. Beh.	✓	×
Meaning	✓	✓
Blame	?	?

Outlook

What is the surface language?
Different for latent and manifest?

How does blame compare in the two approaches?

What does a high-performance implementation of manifest contracts look like?