Why Is This Type Different From All Other Types?

An "Existentialist" Perspective on Parametricity Research

Derek Dreyer

Max Planck Institute for Software Systems (MPI-SWS)
Kaiserslautern and Saarbrücken, Germany

Parametricity Workshop
Glasgow, UK
May 2, 2012

A kind invitation from Patty Johann...

Hi Derek,

I am wondering if you would be willing to kick off the workshop with a 75-minute talk describing the major trends in the area and their importance, from inception to the present. The idea would be to provide a contextual framework for the rest of the day's programme.

Hi Patty,

First, let me say that I'm very honored that you would ask me to give such a talk. In principle, I would be very happy to give it. However:

. . . I am terrified!

Hi Derek,

I am wondering if you would be willing to kick off the workshop with a 75-minute talk describing the major trends in the area and their importance, from inception to the present. The idea would be to provide a contextual framework for the rest of the day's programme.

Goals of this talk

To paint a high-level (but very incomplete) picture and provide some historical background

To ask some provocative/confused questions and incite the ire of the audience

To explain what excites me about parametricity and why I think we're in a "golden age"

Outline

1. What is parametricity?
2. Parametricity and effects
3. The golden age of parametricity research:
   An "existentialist" perspective

Outline

1. What is parametricity?
2. Parametricity and effects
3. The golden age of parametricity research:
   An “existentialist” perspective

Strachey's "parametric polymorphism" (1967)

Idea: Parametricity = Genericity

- Polymorphic function should behave "generically", *i.e.*, "run the same code" at any instantiation of its type
- Explained with a single example:

$$\mathsf{map} : \forall \alpha, \beta.\ (\alpha \to \beta) \to \alpha\,\mathsf{list} \to \beta\,\mathsf{list}$$

Supported by various languages, including Liskov's CLU and Girard-Reynolds' System F (early 1970s)

But not clear formally what parametricity-as-genericity is or what it buys you...

Reynolds' "relational parametricity" (1983)

Idea: Interpret types τ as logical relations $R\,[\![\tau]\!]\,\rho$

- Base types interpreted via identity relation
- Universal types $\forall\alpha.\tau$ interpreted by intersection over all relational interpretations of α (I'm being vague)

Abstraction Theorem: If $\Delta;\Gamma \vdash e : \tau$, then

$$\forall\rho \in \Delta \to \mathrm{Rel}.\ \forall(\gamma_1, \gamma_2) \in R\,[\![\Gamma]\!]\,\rho.\ ([\![e]\!]\,\gamma_1, [\![e]\!]\,\gamma_2) \in R\,[\![\tau]\!]\,\rho.$$

Upshot: Behavior of e must not be affected by change of data representation of type variables in Δ

- $[\![e]\!]\,\gamma_1 = [\![e]\!]\,\gamma_2$ if τ is a base type
- Ex.: switching between polar and cartesian coordinates

Reynolds' "relational parametricity" (1983)

Idea: Interpret types τ as logical relations $R \llbracket \tau \rrbracket \rho$

- Base types interpreted via identity relation

Question

But does Reynolds' relational parametricity capture Strachey's notion of genericity?

- $\llbracket e \rrbracket \gamma_1 = \llbracket e \rrbracket \gamma_2$ if τ is a base type
- Ex.: switching between polar and cartesian coordinates

A "criticism" of relational parametricity

From Abramsky and Jagadeesan (2005):

"Relational parametricity is a beautiful and important notion. However, in our view it is not the whole story. In particular:

- It is a "pointwise" notion, which gets at genericity indirectly, via a notion of uniformity applied to the family of instantiations of the program, rather than directly capturing the idea of a program written at the generic level, which necessarily cannot probe the structure of an instance.
- It is closely linked to strong extensionality principles, as shown e.g. in [ACC93, PA93], whereas the intuition of generic programs not probing the structure of instances is prima facie an intensional notion – a constraint on the behaviour of processes."

A “criticism” of relational parametricity

From Abramsky and Jagadeesan (2005):

“Relational parametricity is a beautiful and important notion.

My Answer

Reynolds’ relational parametricity
explains what you can DO with
Strachey’s “generic” notion of parametricity!

programs not probing the structure of instances is prima facie an intensional notion – a constraint on the behaviour of processes.”

Two kinds of applications of relational parametricity

1. “Universalist”

2. “Existentialist”

“Universalist” applications of relational parametricity

What can one say about all terms of a certain type?

Definability of types:

- Many types (*e.g.*, $\times$, $+$, $\exists$, μ, ν) can be Church-encoded in terms of $\forall$ and $\to$
- Can use parametricity to build simple, yet very expressive, “metalanguages” and type theories

Free theorems (Wadler, 1989):

- $\forall$-types say something interesting about their inhabitants, *e.g.*, $f : \forall\alpha.[\alpha] \to [\alpha]$ can only rearrange elements:

$$\forall g : \sigma \to \tau.\ \ (\mathtt{map}\ \ g) \circ f[\sigma] = f[\tau] \circ (\mathtt{map}\ \ g)$$

- Applicable to proving correctness of various program optimizations (*e.g.*, short-cut fusion)

“Existentialist” applications of relational parametricity

What can one say about particular terms of a certain type?

Representation independence (Mitchell, 1986):

- Can prove two ADTs contextually equivalent if there exists a simulation relation between their type representations that is preserved by their operations
- Essentially a special kind of universalist application: exploits a fact about all contexts of a certain type

Fundamentally a relational, extensional property, not an intensional one!

Can we talk about parametricity without talking about:

1. Semantics?
2. Syntax?

Plotkin and Abadi's logic for parametricity (1993)

Second-order logic with primitive notions of relations and equality

- Logical relations $R\,[\![\tau]\!]\,\rho$ definable in the logic

Parametricity axiom, which can be used to establish definability of types in a purely syntactic manner:

$$\forall \beta_1, \ldots, \beta_n.\ \forall x : (\forall \alpha.\tau[\alpha, \beta_1, \ldots, \beta_n]).$$
$$(x, x) \in R\,[\![\forall \alpha.\tau[\alpha, \beta_1, \ldots, \beta_n]]\!]\,\{\overline{\beta_j \mapsto \mathrm{eq}_{\beta_j}}\}$$

Demonstrates the semantics-independent expressive power of parametricity

Semantic models and relatives of parametricity

Reynolds built his logical relations over a naive, classical set-theoretic model of System F types that turned out not to exist!

Lots of work on models that do exist + semantic criteria for what being a "parametric model" means:

- Pitts' constructive set-theoretic model, Bainbridge et al.'s PER model, realizability models
- "Reflexive graph" models: parametric APL structures (Birkedal-Møgelberg), parametric limits (Dunphy-Reddy)

Semantic models and relatives of parametricity

Reynolds built his logical relations over a naive, classical set-theoretic model of System F types that

Question

What's the state of the art here?
(I don't know.)

Semantic models and relatives of parametricity

Reynolds built his logical relations over a naive, classical set-theoretic model of System F types that turned out not to exist!

Lots of work on models that do exist + semantic criteria for what being a "parametric model" means:

- Pitts' constructive set-theoretic model, Bainbridge et al.'s PER model, realizability models
- "Reflexive graph" models: parametric APL structures (Birkedal-Møgelberg), parametric limits (Dunphy-Reddy)

Semantic models and relatives of parametricity

Reynolds built his logical relations over a naive, classical set-theoretic model of System F types that turned out not to exist!

Lots of work on models that do exist + semantic criteria for what being a "parametric model" means:

- Pitts' constructive set-theoretic model, Bainbridge et al.'s PER model, realizability models
- "Reflexive graph" models: parametric APL structures (Birkedal-Møgelberg), parametric limits (Dunphy-Reddy)

Related notions of "uniformity":

- Naturality, dinaturality, genericity (Longo et al., 1993)
- Does parametricity subsume these?

Outline

1. What is parametricity?
2. Parametricity and effects
3. The golden age of parametricity research: An "existentialist" perspective

The overarching trend in parametricity research

From Voigtländer and Johann (2007):

"The ultimate goal of the line of research advanced in this paper is the development of tools for reasoning about parametricity properties of, and parametricity-based transformations on programs in, real programming languages rather than toy calculi."

The overarching trend in parametricity research

Generalizing parametricity to handle richer languages supporting:

- Computational effects (recursion, mutable state, control operators, concurrency)
- Higher kinds, dependent types
- Units of measure
- Substructural types
- Dynamic type analysis
- ...

Generalizing parametricity to handle richer languages supporting:

- Computational effects (recursion, mutable state, control operators, concurrency)
- Higher kinds, dependent types
- Units of measure
- Substructural types
- Dynamic type analysis
- ...

Generalizing parametricity to handle effects

1. Definability of types in the presence of effects
2. Free theorems in the presence of effects
3. Representation independence and local state

Generalizing parametricity to handle effects

1. Definability of types in the presence of effects
2. Free theorems in the presence of effects
3. Representation independence and local state

System F + recursion/effects as a core metalanguage?

System F is very expressive, but it's total

Idea: Adding recursion/effects in the "right" way could enable it to serve as a metalanguage for the semantics of more realistic languages

- Encode rec. types $\Rightarrow$ Solve rec. domain equations in types

Problem: Even just parametricity + Y renders the type theory inconsistent (Huwig-Poigné, 1990)

- Need to restrict parametricity to only interpret abstract types with "admissible" (strict, chain-complete) relations

Plotkin's idea (1993): Use linearity to model strictness

Theory of PILL_Y/Lily worked out by Birkedal-Møgelberg-Petersen denotationally (2006) and Bierman-Pitts-Russo operationally (2000):

Smash product	$\tau \otimes \tau' \triangleq \forall \alpha. (\tau \multimap \tau' \multimap \alpha) \multimap \alpha$
Coalesced sum	$\tau \oplus \tau' \triangleq \forall \alpha. !(\tau \multimap \alpha) \multimap !(\tau' \multimap \alpha) \multimap \alpha$
Product	$\tau \times \tau' \triangleq \forall \alpha. ((\tau \multimap \alpha) \oplus (\tau' \multimap \alpha)) \multimap \alpha$
Separated sum	$\tau + \tau' \triangleq !\tau \oplus !\tau'$
Existential	$\exists\, \alpha. \tau(\alpha) \triangleq \forall \beta. (\forall \alpha. \tau(\alpha) \multimap \beta) \multimap \beta$
Truth values	$\mathsf{T} \triangleq \forall \alpha. !\alpha \multimap !\alpha \multimap \alpha$
Flat naturals	$\mathsf{N}_\perp \triangleq \forall \alpha. !\alpha \multimap !(!\alpha \multimap \alpha) \multimap \alpha$
Inductive	$\mu\alpha. \tau(\alpha) \triangleq \forall \alpha. !(\tau(\alpha) \multimap \alpha) \multimap \alpha \quad (\alpha \text{ +ve in } \tau(\alpha))$
Co-inductive	$\nu\alpha. \tau(\alpha) \triangleq \exists\, \alpha. !(\alpha \multimap \tau(\alpha)) \otimes \alpha \quad (\alpha \text{ +ve in } \tau(\alpha))$
Recursive	$\mathtt{rec}\, \alpha. \tau(\alpha, \alpha) \triangleq \nu\alpha. \tau(\mu\beta. \tau(\alpha, \beta), \alpha) \quad (\alpha \text{ +ve in } \tau(\alpha, \beta), \beta \text{ −ve in } \tau(\alpha, \beta))$

PE: Scaling to monadic effects

Møgelberg and Simpson (2007) define a type theory PE with linearity, polymorphism, and value/computation types à la Levy's CBPV

- Value/computation types needed, *e.g.*, to allow for effectful operations at polymorphic type

$$\text{choice} : \forall \underline{X}.\ \underline{X} \to \underline{X} \to \underline{X}$$

Again, many types are encodable, although PE does not handle recursion

PE: Scaling to monadic effects

Møgelberg and Simpson (2007) define a type theory

Question

Are these type theories (PILL_Y, PE) actually useful as metalanguages?

What's left to do here?

not handle recursion

1. Definability of types in the presence of effects
2. Free theorems in the presence of effects
3. Representation independence and local state

Pitts-Stark's $\top\top$-closure and operational logical relations

Pitts-Stark (1998) propose a simpler alternative to "admissibility" and PILL_Y for operational models:

- $\top\top$-closure (aka $\bot\bot$-closure, biorthogonality)

Useful for several reasons:

- Ensures that the LR is admissible in the domain-theoretic sense (and thus, closure under fixed-points)
- Ensures completeness w.r.t. contextual equivalence
- Works even for lang's with "context-sensitive" semantics

Key results about free theorems in the presence of effects

Pitts (2000):

- Studied PolyPCF, a lazy language with recursion
- Proved various extensionality principles, as well as definability of list and $\exists$ types

Johann (2002):

- Proved correctness of various free-theorem-based optimizations like short-cut fusion in a setting like Pitts's

Key results about free theorems in the presence of effects

Johann-Voigtländer (2004, 2007):

- Proved correctness of restrictions of the above optimizations in the presence of "seq"
- Influential partly due to its surprising (negative) results

Johann-Simpson-Voigtländer (2010):

- Generic framework for $\top\top$-closed relations in the presence of arbitrary Plotkin-Power-style "algebraic effects"
- Proved extensionality principles and definability of monadic type $T(\tau) \approx \forall\alpha.(\tau \to \alpha) \to \alpha$

Key results about free theorems in the presence of effects

Johann-Voigtländer (2004, 2007):

- Proved correctness of restrictions of the above

Question

Do the JV free-theorem restrictions invalidate common cases where short-cut fusion is useful?

monadic type $T(\tau) \approx \forall\alpha.(\tau \to \alpha) \to \alpha$

The "identity extension" lemma

Apparently important lemma whose importance confuses me:

$$R\,[\![\tau]\!]\,(\alpha \mapsto \mathrm{eq}_\sigma) = \mathrm{eq}_{\tau[\sigma/\alpha]}$$

Seems necessary to prove parametricity in denotational settings

But not needed in operational settings

- Falls out as a consequence of $\top\top$-closure, but not when step-indexing is used!
- Seems relevant in proving certain definability results and free theorems (*e.g.*, short-cut fusion) but not others

The "identity extension" lemma

Apparently important lemma whose importance confuses me:

Question

What the hell is going on here?

- Seems relevant in proving certain definability results and free theorems (*e.g.*, short-cut fusion) but not others

Generalizing parametricity to handle effects

1. Definability of types in the presence of effects

2. Free theorems in the presence of effects

3. Representation independence and local state

Representation independence for local state

Reasoning about local state much like reasoning about abstract types

- Should be able to change internal data representation without affecting clients

Some major differences:

- State and the invariants on it may "change shape" as the program is executed
- State has a "temporal" component in that it can undergo irreversible changes

Denotational models of Algol, focused on "invariants"

Reynolds-Oles (1981-82):

- Functor-category model (a kind of Kripke model), but fairly weak reasoning principles

Meyer-Sieber (1988):

- Shows how to support reasoning about invariants on a range of interesting (second-order) examples:

$$\begin{array}{l} \textbf{begin} \\ \quad \textbf{integer}\ x; \\ \quad \textbf{procedure}\ Add2; \\ \quad\quad \textbf{begin}\ x := x + 2\ \textbf{end} \\ \quad x := 0; P(Add2); \\ \quad \textbf{if}\ x\ \textbf{mod}\ 2 = 0\ \textbf{then}\ diverge \\ \textbf{end} \end{array} \quad \approx \quad diverge$$

Denotational models of representation independence

O'Hearn-Tennent (1993):

- The first approach to really support reasoning about representation independence:

begin
  integer x;
  integer procedure Val;
    $Val := x$;
  procedure Inc;
    begin $x := x + 1$ **end**
  $x := 0; P(Inc, Val)$;
end
	$\approx$	**begin**
		  integer x;
		  integer procedure Val;
		    $Val := -x$;
		  procedure Dec;
		    begin $x := x - 1$ **end**
		  $x := 0; P(Dec, Val)$;
		end

- Reduces rep. ind. in Algol to rep. ind. in System F by a polymorphic store-passing interpretation
- Sieber (1992) provides an alternative approach, also based on logical relations, that I don't know the details of

Denotational accounts of irreversible state change

O'Hearn-Reynolds (1995):

- Similar to O'Hearn-Tennent, but interprets Algol into a polymorphic linear type system in order to track irreversibility of state change

$$\begin{array}{l} \textbf{begin} \\ \quad \textbf{integer}\ x; \\ \quad \textbf{procedure}\ \mathit{Inc}; \\ \qquad \textbf{begin}\ x := x + 1\ \textbf{end} \\ \quad x := 0; P(\mathit{Inc}); \\ \quad \textbf{if}\ x > 0\ \textbf{then}\ \mathit{diverge} \\ \textbf{end} \end{array} \quad \approx \quad P(\mathit{diverge})$$

O'Hearn-Reddy (1995):

- A completely different approach to locality and irreversibility based on placing invariants on the observable actions on local state

Operational models of local first-order state

Pitts (1997):

- Operational possible-worlds model of Idealized Algol (IA), inspired by O'Hearn-Reynolds and prior work
- Provides a more direct method of proving all previous results, including reasoning about irreversibility

Pitts-Stark (1998):

- Models a simply-typed ML-like language with `int ref`'s, but does not support reasoning about irreversibility
- Major difficulty involves the fact that variables may escape their scope ($\top\top$-closure is introduced to deal with this)
- One of my all-time favorite papers

Operational models of local first-order state

Pitts (1997):

- Operational possible-worlds model of Idealized Algol (IA),

Question

Are there any examples of Algol equivalences that become inequivalences when ported to ML?

their scope ($\top\top$-closure is introduced to deal with this)

- One of my all-time favorite papers

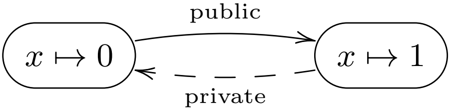
Operational model of local higher-order state $+\ \mu, \forall, \exists$

$$
\begin{aligned}
\tau &= \exists\alpha.\,\exists\beta.\,(\mathsf{unit} \to \alpha) \times (\mathsf{unit} \to \beta) \times (\alpha \times \beta \to \mathsf{bool}) \\
e_1 &= \mathsf{let}\ x = \mathsf{ref}\ 0\ \mathsf{in} \\
&\quad\ \mathsf{pack}\ \langle \mathsf{int}, \mathsf{pack}\ \langle \mathsf{int}, \lambda\_.\ x := !x + 1; !x, \\
&\qquad\qquad\qquad\qquad\quad \lambda\_.\ x := !x + 1; !x, \\
&\qquad\qquad\qquad\qquad\quad \lambda p.\ p.1 = p.2 \rangle\rangle \\
e_2 &= \mathsf{pack}\ \langle \mathsf{unit}, \mathsf{pack}\ \langle \mathsf{unit}, \lambda\_.\ \langle\rangle, \\
&\qquad\qquad\qquad\qquad\quad \lambda\_.\ \langle\rangle, \\
&\qquad\qquad\qquad\qquad\quad \lambda\_.\ \mathsf{false} \rangle\rangle
\end{aligned}
$$

Ahmed-Dreyer-Rossberg (2009):

- Building on Pitts-Stark (1998) and Ahmed (2004, 2006), step-indexing used to model higher-order state $+\ \mu, \forall, \exists$
- Key idea: Irreversibility of state change modeled through state transition systems (STS's)
- Especially useful for modeling "generative" ADTs that grow over time in accordance with changes to local state

The local state of the art

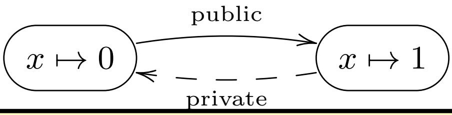


$$
\begin{aligned}
\tau &= (\mathsf{unit} \to \mathsf{unit}) \to \mathsf{int} \\
e_1 &= \lambda f.\,(f\ \langle\rangle; f\ \langle\rangle; 1) \\
e_2 &= \mathsf{let}\ x = \mathsf{ref}\ 0\ \mathsf{in}\ \lambda f.\,(x := 0; f\ \langle\rangle; x := 1; f\ \langle\rangle; !x)
\end{aligned}
$$

Dreyer-Neis-Birkedal (2010):

- Show that a $\top\top$-closure of the ADR model is sound in the presence of call/cc, but some extensions to it are not
- Give a framework for understanding the impact of higher-order state, call/cc and exceptions on STS-style reasoning about local state

The local state of the art



Question

"Full abstraction" in operational models seems to be a rather uninformative property.

It seems to imply something stronger in denotational models, but why?

- Give a framework for understanding the impact of higher-order state, call/cc and exceptions on STS-style reasoning about local state

Bisimulations for ML-like languages

Environmental bisimulations

- Coinductively-defined sets of relations, quite similar to the ADR model in terms of expressive power
- Sumii-Pierce (2004, 2005), Koutavas-Wand (2006), Sangiorgi-Kobayashi-Sumii (2007), Sumii (2009)

Normal form (or open) bisimulations

- Elegant treatment of higher-order functions, can be combined with env. bisim. to model local H-O state
- Lassen-Levy (2007, 2008), Støvring-Lassen (2007)

Parametric bisimulations

- Synthesis of ideas from the above techniques, as well as Dreyer-Neis-Birkedal logical relations
- Hur-Dreyer-Neis-Vafeiadis (2012)

Outline

1. What is parametricity?
2. Parametricity and effects
3. The golden age of parametricity research:
   An "existentialist" perspective

My "existentialist" view

Looking at concrete applications of parametricity is really useful!

- Examples help to convey intuitions and uncover deficiencies in existing models

Some of the most influential papers are in large part influential thanks to emphasis on concrete examples

- Meyer-Sieber (1988), Wadler (1989), Kennedy (1997), Pitts-Stark (1998), Johann-Voigtländer (2004), . . .

Denotational models provide amazing insights, but operational models offer a lower barrier to entry

- Enabled someone like me to get involved in the field and start working out examples quickly without learning a huge body of mathematics first

We're in a golden age of parametricity research!

We've spent 30 years building the foundations of the house of parametricity, let's live in it!

- Now that we've adapted parametricity to more realistic languages, let's start deploying it in a broader range of "real" applications besides free theorems and ctx. equiv.

This is win-win

- New apps will expose further holes in our foundations, just as concrete examples have done in the past
- For reasoning about large systems, abstraction is key, and parametricity is the only game in town

Application #1: Compositional compiler correctness

Goal: compositional equivalences between programs
in different languages (Benton et al.)

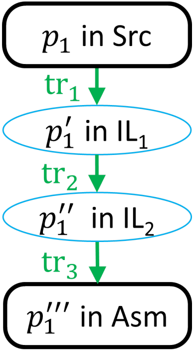
- *e.g.*, compositional compiler correctness

Application #1: Compositional compiler correctness

Goal: compositional equivalences between programs in different languages (Benton et al.)

- *e.g.*, compositional compiler correctness

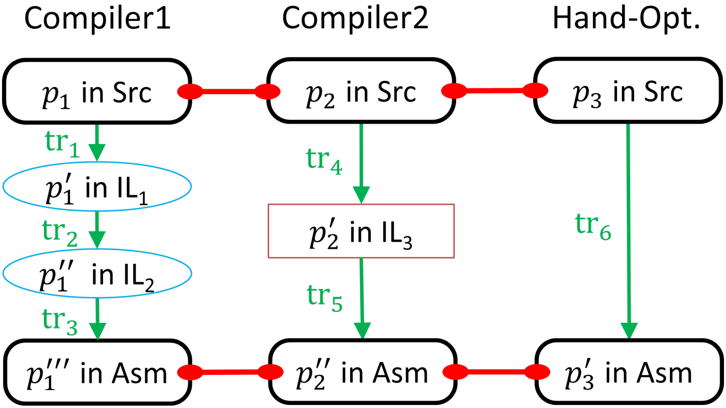
Compiler



Application #1: Compositional compiler correctness

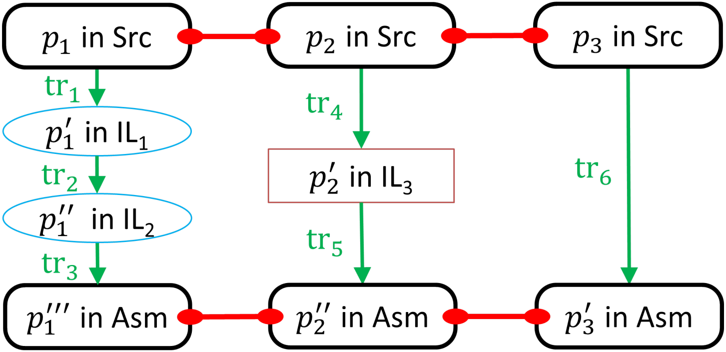
Goal: compositional equivalences between programs in different languages (Benton et al.)

- *e.g.*, compositional compiler correctness



Application #1: Compositional compiler correctness

- Horizontal compositionality is preservation of equivalence under linking of modules.
- Vertical compositionality is transitive composition of equivalence proofs.



Parametric bisimulations to the rescue! (POPL'12 – Joint work with Hur, Neis, Vafeiadis)

Logical relations are not transitively composable

- Especially step-indexed Kripke logical relations
- Hur et al. [ICFP09, POPL11] only studied one-pass compilers

Bisimulations do not scale (in an obvious way) to inter-language reasoning

- Due to their use of "syntactic" devices for H-O functions

Parametric bisimulations remove these limitations

- "Relational" treatment of H-O fcns (like logical relations)
- Supports transitive composition of proofs (like bisim's)

Application #2: Making substructural types more flexible

Combination of existential + substructural types

- Allows for precise control over invariants on private state
- Example: interface of a memory allocator whose internal invariant depends on the set of allocated locations

$$
\begin{array}{rcl}
 & & \exists A : \mathsf{LocSet} \to \mathsf{Type}. \\
\mathsf{init\_cap} & : & A(\emptyset) \\
\otimes\ \mathsf{malloc} & : & !\forall L : \mathsf{LocSet}.\ A(L) \multimap \\
 & & \quad \exists X : \mathsf{Loc}.\ \mathsf{ptr}\ X \otimes \mathsf{cap}\ X\ 1 \otimes A(L \uplus \{X\}) \\
\otimes\ \mathsf{free} & : & !\forall L : \mathsf{LocSet}.\ \forall X : \mathsf{Loc}. \\
 & & \quad \mathsf{ptr}\ X \otimes \mathsf{cap}\ X\ 1 \otimes A(L \uplus \{X\}) \multimap A(L)
\end{array}
$$

Problem: Interface pollution for clients

- A client must thread the "capability" $A(L)$ through its interface to guard against interference from other clients

Superficially substructural types (Submitted – Joint work with Krishnaswami, Turon, Garg)

We propose a new sharing rule:

- Enables $A(L)$ to be split into "fictionally disjoint" pieces, so clients can be oblivious to one another's existence

$$\begin{array}{lcl} \text{split} & : & \forall L_1, L_2 : \text{LocSet}.\ A(L_1 \uplus L_2) \multimap A(L_1) \otimes A(L_2) \\ \text{join} & : & \forall L_1, L_2 : \text{LocSet}.\ A(L_1) \otimes A(L_2) \multimap A(L_1 \uplus L_2) \end{array}$$

This can be done for any commutative monoid!

- Each ADT can pick whatever monoid is best
- Builds on Birkedal et al.'s work on separation logic
- Soundness of the rule proven using a novel variant of Dreyer-Neis-Birkedal possible-worlds model, with the STS's replaced by monoids

Application #3: Log. relations for fine-grained concurrency (Joint work with Turon, Thamsborg, Ahmed, Birkedal)

Verification of fine-grained concurrent algorithms

- People have focused on linearizability (Herlihy-Wing, '90), but what client really cares about is contextual refinement

```
class TreiberStack[A]
  def push (a : A)  =  ...
  def tryPop ()  =  ...
```

$\preceq$

```
class StackSpec[A]
  private val s  =  new SeqStack[A]
  def push (a : A)  =  atomic {s.push(a); }
  def tryPop ()  =  atomic {s.tryPop(); }
```

We're adapting STS-based logical relations to verify these contextual refinement properties directly

- This is work in progress, but already we can see that new and interesting extensions of existing models are required

Parametricity is our secret weapon.
Let's put it to work!