

# Faithful Translations between Polyvariant Flows and Polymorphic Types

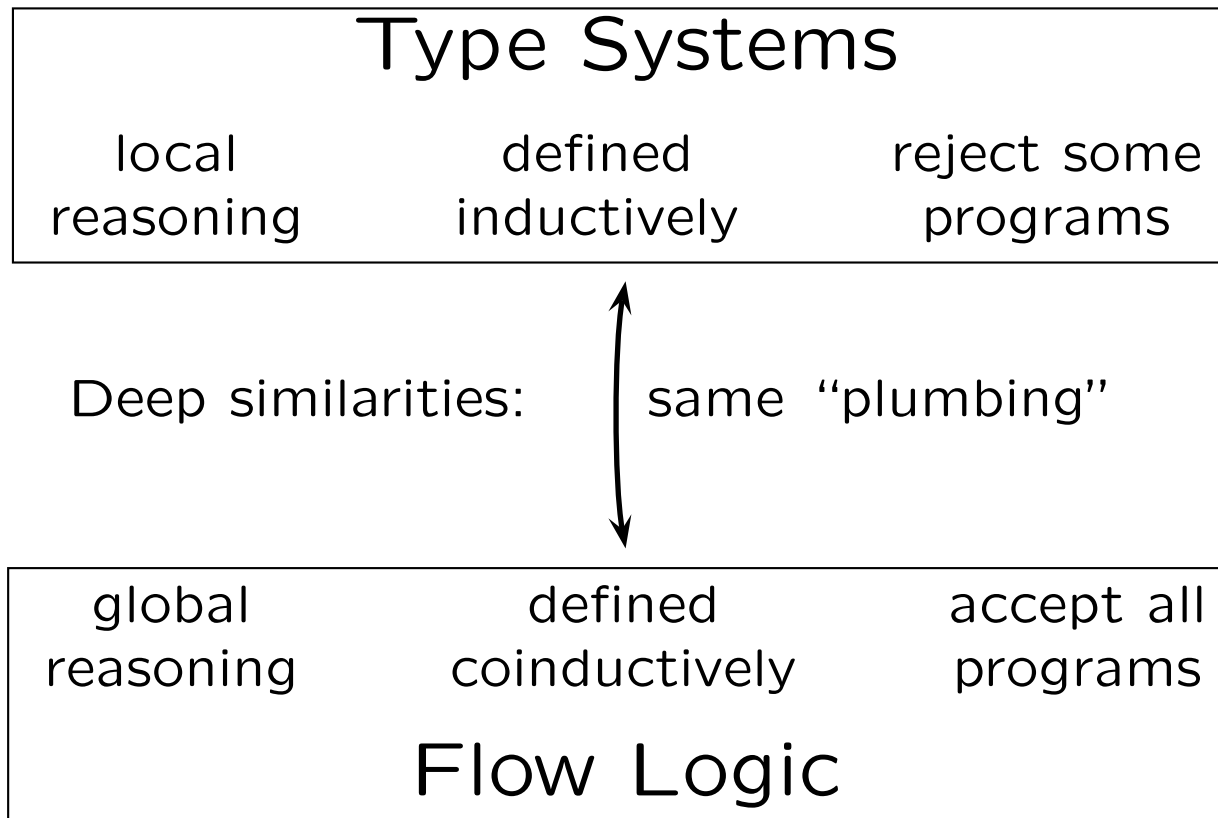
Torben Amtoft  
Boston University/Church Project

Franklyn Turbak  
Wellesley College/Church Project

March 31, 2000

ESOP, Berlin

## Program Analysis: Two Paradigms



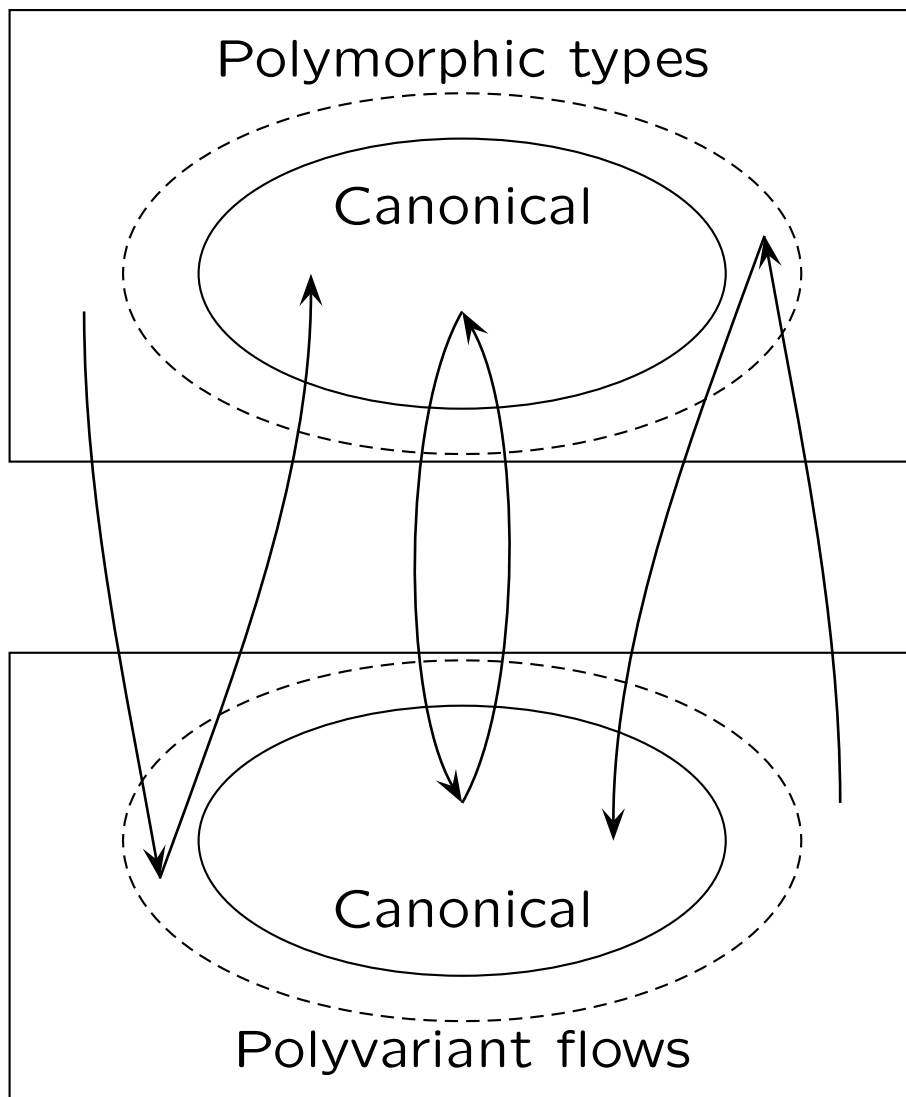
---

## Aim of Work

---

Understand connection between polyvariant flows and polymorphic types

Design systems enabling faithful translations:



---

## Contributions

---

- General framework for polyvariant flow analysis handling both
  - call-string based polyvariance
  - argument based polyvariance

extending Nielson & Nielson [POPL'97] with ideas from Palsberg & Pavlopoulou [POPL'98]

- Type system with labeled union and intersection types
  - generalises P&P
  - inspired by the CIL type system [Wells et.al, FASE'97]
- Subject reduction for flows and types
- First faithful translations (for polyvariance)
  - first type system corresponding to  $k$ -CFA ( $k \geq 1$ )
  - perspective: “let flows have it their way”

---

## Road map

---

- Summarize existing translations between flows and type and show how they lack faithfulness
- Sketch how to modify systems and translations to achieve faithfulness
- Sketch the flow and type frameworks used by our translations

## Types: Local Reasoning

I stands for the integer type

$$\frac{\frac{\frac{[f:I \rightarrow I] \vdash f : I \rightarrow I \quad [f:I \rightarrow I] \vdash 3 : I}{[f:I \rightarrow I] \vdash f @ 3 : I}}{[] \vdash \lambda f.f @ 3 : (I \rightarrow I) \rightarrow I} \quad \frac{[x:I] \vdash x : I}{[] \vdash \lambda x.x : I \rightarrow I}}{[] \vdash (\lambda f.f @ 3) @ (\lambda x.x) : I}$$

## Flows: Global Reasoning

$\rho$ : Variables  $\rightarrow \mathcal{P}(\text{FlowVal})$        $\mathcal{C}$ : Program point  $\rightarrow \mathcal{P}(\text{FlowVal})$

Flow analysis (least 0-CFA) for  $P = (\lambda f.f @ 3) @ (\lambda x.x)$

$$\begin{aligned} \{\lambda x.x\} &= \mathcal{C}(\lambda x.x) = \rho(f) = \mathcal{C}(f) \\ \{\lambda f.f @ 3\} &= \mathcal{C}(\lambda f.f @ 3) \\ \{\text{Int}\} &= \mathcal{C}(3) = \rho(x) = \mathcal{C}(x) = \mathcal{C}(f @ 3) = \mathcal{C}(P) \end{aligned}$$

Correctness criteria include:

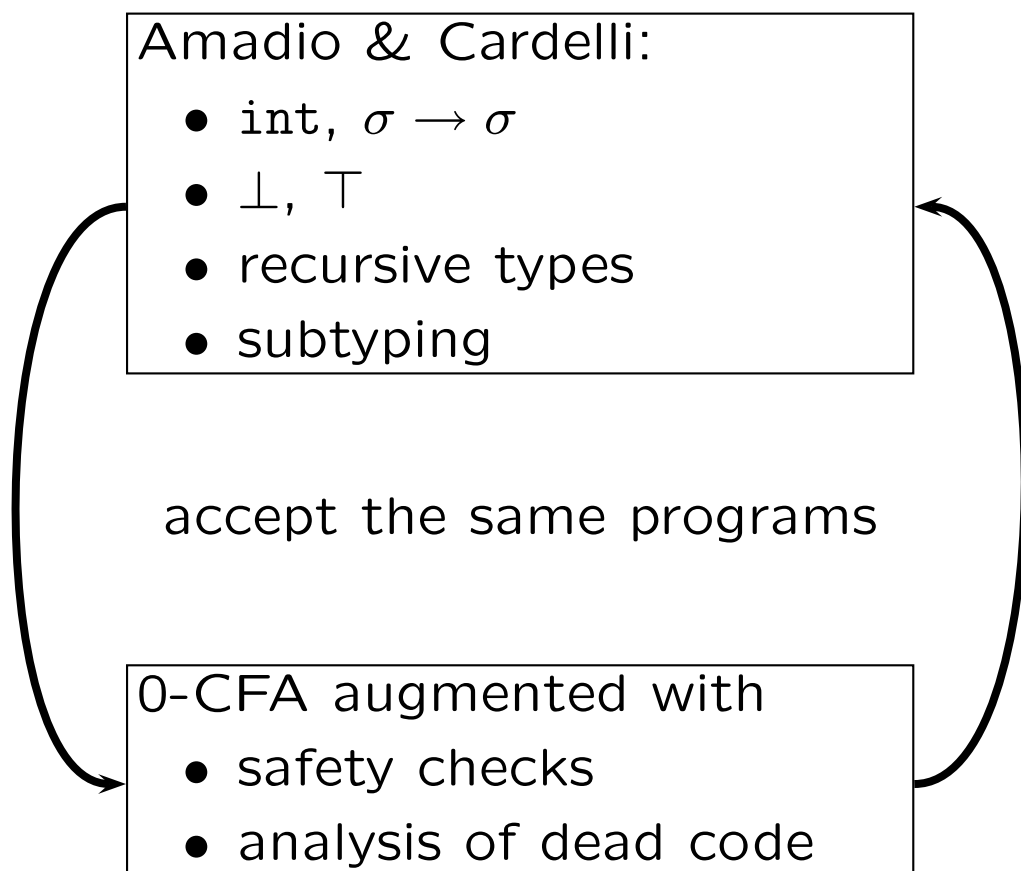
- $\lambda x.x \in \mathcal{C}(f)$  implies  $\mathcal{C}(3) \subseteq \rho(x)$

Safety includes:

- $\text{Int} \notin \rho(f)$

— Correspondence in Monovariant Case —

Equivalence result by  
Palsberg & O'Keefe [POPL'95]



## — Correspondence in Polyvariant Case —

Equivalence result by  
Palsberg & Pavlopoulou [POPL'98]

Type system with finitary polymorphism:

- $\text{int}, \sigma \rightarrow \sigma$
- recursive types
- subtyping
- union types (model multiple sources)
- intersection (model multiple sinks)

translations back and forth

Flow analysis with finitary polyvariance

- argument-based (not call-string)
- safety checks
- analysis also of dead code

## Monovariant Round Trip

$$\begin{array}{c}
 [f:I \rightarrow I] \vdash f : I \rightarrow I \dots \vdash 3 : I \\
 \hline
 [f:I \rightarrow I] \vdash f @ 3 : I \qquad [x:I] \vdash x : I \\
 \hline
 [] \vdash \lambda f.f @ 3 : (I \rightarrow I) \rightarrow I \qquad [] \vdash \lambda x.x : I \rightarrow I \\
 \hline
 [] \vdash P = (\lambda f.f @ 3) @ (\lambda x.x) : I
 \end{array}$$

$$\begin{aligned}
 \mathcal{F}(I) &= \{\text{Int}\} \\
 \mathcal{F}(I \rightarrow I) &= \{\lambda x.x\} \\
 \mathcal{F}((I \rightarrow I) \rightarrow I) &= \{\lambda f.f @ 3\}
 \end{aligned}$$

$$\begin{aligned}
 \mathcal{T}(\{\text{Int}\}) &= I \\
 \mathcal{T}(\{\lambda x.x\}) &= \mathcal{T}(\rho(x)) \rightarrow \mathcal{T}(\mathcal{C}(x)) = I \rightarrow I \\
 \mathcal{T}(\{\lambda f.f @ 3\}) &= \mathcal{T}(\rho(f)) \rightarrow \mathcal{T}(\mathcal{C}(f @ 3)) \\
 &= (I \rightarrow I) \rightarrow I
 \end{aligned}$$

$$\begin{aligned}
 \{\lambda x.x\} &= \rho(f) = \mathcal{C}(f) = \mathcal{C}(\lambda x.x) \\
 \{\lambda f.f @ 3\} &= \mathcal{C}(\lambda f.f @ 3) \\
 \{\text{Int}\} &= \mathcal{C}(3) = \rho(x) = \mathcal{C}(x) = \mathcal{C}(f @ 3) = \mathcal{C}(P)
 \end{aligned}$$

## Round Trip Losing Flow Information

$$\begin{array}{c}
 \frac{}{[]} \vdash \lambda x. \text{succ } x : \mathbb{I} \rightarrow \mathbb{I} \quad \frac{}{[]} \vdash \lambda y. y : \mathbb{I} \rightarrow \mathbb{I} \dots \\
 \hline
 \frac{}{[]} \vdash P = (\lambda x. \text{succ } x) @ ((\lambda y. y) @ 3) : \mathbb{I}
 \end{array}$$

$$T(\{\lambda y. y\}) = \mathbb{I} \rightarrow \mathbb{I}$$

$$= T(\rho(y)) \rightarrow T(\mathcal{C}(y)) \quad \mathcal{F}(\mathbb{I} \rightarrow \mathbb{I}) = \{\lambda x. \text{succ } x, \lambda y. y\}$$

$$T(\{\lambda x. \text{succ } x\}) = \mathbb{I} \rightarrow \mathbb{I}$$

$$\begin{array}{l}
 \{\text{Int}\} = \rho(x) = \mathcal{C}(x) = \mathcal{C}(\text{succ } x) \\
 \quad = \rho(y) = \mathcal{C}(y) = \mathcal{C}(3) \\
 \quad = \mathcal{C}((\lambda y. y) @ 3) = \mathcal{C}(P) \\
 \{\lambda x. \text{succ } x\} = \mathcal{C}(\lambda x. \text{succ } x) \\
 \{\lambda y. y\} = \mathcal{C}(\lambda y. y)
 \end{array}$$

$$\begin{array}{l}
 \{\text{Int}\} = \dots \text{ as before} \\
 \{\lambda x. \text{succ } x, \lambda y. y\} \\
 \quad = \mathcal{C}'(\lambda x. \text{succ } x) \\
 \quad = \mathcal{C}'(\lambda y. y)
 \end{array}$$



## Roundtrip (Polyvariant) Introducing Infinite Types

$$I2I = I \rightarrow I \quad u = \wedge\{I2I, I2I \rightarrow I2I\}$$

$$\begin{array}{c}
 [f:u] \vdash f : I2I \rightarrow I2I \quad [f:u] \vdash f : I2I \\
 \hline
 [f:u] \vdash f \circ f : I2I \quad \dots \\
 \hline
 [f:u] \vdash (f \circ f) \circ 7 : I \quad [x:I] \vdash^1 x : I \\
 \hline
 [f:u] \vdash (f \circ f) \circ 7 : I \quad [x:I2I] \vdash^2 x : I2I \\
 \hline
 [] \vdash \lambda f.(f \circ f) \circ 7 : u \rightarrow I \quad [] \vdash \lambda x.x : u \\
 \hline
 [] \vdash P = (\lambda f.(f \circ f) \circ 7) \circ (\lambda x.x) : I
 \end{array}$$

$$\begin{array}{l}
 \{\lambda x.x\} = \mathcal{F}(I2I) \\
 = \mathcal{F}(I2I \rightarrow I2I)
 \end{array}
 \quad
 \begin{array}{l}
 \boxed{\mathcal{T}(\{\lambda x.x\})} = \wedge\{\mathcal{T}(\rho(x, 1)) \rightarrow \mathcal{T}(\mathcal{C}(x, [x:1])), \\
 \mathcal{T}(\rho(x, 2)) \rightarrow \mathcal{T}(\mathcal{C}(x, [x:2]))\} \\
 = \wedge\{I \rightarrow I, \boxed{\mathcal{T}(\{\lambda x.x\})} \rightarrow \boxed{\mathcal{T}(\{\lambda x.x\})}\}
 \end{array}$$

$$\boxed{\rho(x, 1) = \mathcal{C}(x, [x:1]) = \{\text{Int}\}} \quad \boxed{\rho(x, 2) = \mathcal{C}(x, [x:2]) = \{\lambda x.x\}}$$

1 and 2 “mementa” denoting sink(s)

## Roundtrip Preserving Type Finiteness

$$\begin{array}{c}
 [f:u] \vdash f : I2I \rightarrow I2I \quad [f:u] \vdash f : I2I \\
 \hline
 [f:u] \vdash f \circ f : I2I \quad \dots \\
 \hline
 [f:u] \vdash (f \circ f) \circ \tau : I \quad \dots \\
 \hline
 [x:I] \vdash^1 x : I \\
 [x:I2I] \vdash^2 x : I2I \\
 \hline
 [] \vdash \lambda f.(f \circ f) \circ \tau : u \rightarrow I \quad [] \vdash \lambda x.x : u \\
 \hline
 [] \vdash P = (\lambda f.(f \circ f) \circ \tau) \circ (\lambda x.x) : I
 \end{array}$$

$$u = \wedge\{I2I, I2I \rightarrow I2I\}$$

$$\begin{aligned}
 \mathcal{T}(\{(\lambda x.x, \{1\})\}) &= I2I \\
 &= \mathcal{T}(\rho(x, 1)) \rightarrow \mathcal{T}(\mathcal{C}(x, [x:1]))
 \end{aligned}$$

$$\begin{aligned}
 \mathcal{F}(I2I) &= \{(\lambda x.x, \{1\})\} \\
 \mathcal{F}(I2I \rightarrow I2I) &= \{(\lambda x.x, \{2\})\} \\
 \mathcal{F}(u) &= \{(\lambda x.x, \{1, 2\})\}
 \end{aligned}$$

$$\begin{aligned}
 \mathcal{T}(\{(\lambda x.x, \{2\})\}) &= I2I \rightarrow I2I \\
 &= \mathcal{T}(\rho(x, 2)) \rightarrow \mathcal{T}(\mathcal{C}(x, [x:2])) \\
 \mathcal{T}(\{(\lambda x.x, \{1, 2\})\}) &= \wedge\{I2I, I2I \rightarrow I2I\} = u
 \end{aligned}$$

$$\rho(x, 1) = \mathcal{C}(x, [x:1]) = \{\text{Int}\}$$

$$\rho(x, 2) = \mathcal{C}(x, [x:2]) = \mathcal{F}(I2I) = \{(\lambda x.x, \{1\})\}$$

We equip flow values with sinks

---

## Road map

---

- Summarize existing translations between flows and type and show how they lack faithfulness
- Sketch how to modify systems and translations to achieve faithfulness
- Sketch the flow and type frameworks used by our translations

## Polymorphic Type System

$u ::= \bigvee_{i \in I} \{q_i : t_i\}$ $I$ finite $q_i$ disjoint	tagged union • model multiple sources
$q \in \text{U-tag}$	track sources
$t ::= I$   $\bigwedge_{i \in I} \{K_i : u_i \rightarrow u'_i\}$ $I$ finite $K_i$ disjoint $\neq \emptyset$	tagged intersections • model multiple sinks
$K ::= \{k_1 \dots k_n\}$ $k \in \text{I-tag}$	track sinks

### Infinite types

- regular and finitely branching
- model flow cycles [Heintze, SAS'95]
- no explicit syntax

## Subtyping

Coinductive definition

Untagged rule

Tagged rule

$\forall i \in I \{t_i\} \leq_u \forall j \in J \{t'_j\}$ <p style="text-align: center;">iff</p> $\forall i \in I. \exists j \in J. t_i \leq_t t'_j$	$\forall i \in I \{q_i : t_i\} \leq_u \forall j \in J \{q'_j : t'_j\}$ <p style="text-align: center;">iff</p> $\forall i \in I. \exists j \in J. q_i = q'_j \text{ and } t_i \leq_t t'_j$
$\wedge \{u_1 \rightarrow u'_1, u_2 \rightarrow u'_2\}$ $\leq_t \wedge \{u_1 \rightarrow u'_1\}$	$\wedge \{K_1 : u_1 \rightarrow u'_1, K_2 : u_2 \rightarrow u'_2\}$ $\leq_t \wedge \{K_1 : u_1 \rightarrow u'_1\}$
$\wedge \{u_1 \rightarrow u, u_2 \rightarrow u\}$ $\leq_t \wedge \{(v\{u_1, u_2\}) \rightarrow u\}$	$\wedge \{K_1 : u_1 \rightarrow u, K_2 : u_2 \rightarrow u\}$ $\leq_t \wedge \{K_1 \cup K_2 : v\{u_1, u_2\} \rightarrow u\}$

Involves choice

Tags are witnesses

Last two rules special cases of complex rule

## Typing Rules

$$\boxed{\frac{}{A \vdash x : u} \text{ [var]}} \text{ if } A(x) \leq_u u \text{ (subtyping inlined)}$$

$$\boxed{\frac{\forall k \in K : A[x:u_k] \vdash e : u'_k}{A \vdash \lambda x.e : u} \text{ [fun]}(q:t)}$$

if  $\bigvee\{q : t\} \leq_u u$  with  $t = \bigwedge_{k \in K} \{\{k\} : u_k \rightarrow u'_k\}$

$q$  enables us to trace back the source

$K$  may be empty (for dead code),  
letting flows have it their way

$$\boxed{\frac{A \vdash e_1 : u_1 \quad A \vdash e_2 : u_2}{A \vdash e_1 \odot e_2 : u} \text{ [app]}^w}$$

if  $u_1 = \bigvee_{q \in Q} \{q : \bigwedge\{w(q) : u_2 \rightarrow u\}\}$

For a given source  $q$ ,  $[\text{app}]^w$  is a sink of the abstraction labeled  $q$  analyzed at the mementa(s) in  $w(q)$

## Polyvariant Derivation

$$u = \bigvee \{x : \bigwedge \{ \{1\} : I \rightarrow I, \{2\} : u' \rightarrow u' \} \}$$

$$u' = \bigvee \{x : \bigwedge \{ \{1\} : I \rightarrow I \} \} \quad u'' = \bigvee \{x : \bigwedge \{ \{2\} : u' \rightarrow u' \} \}$$

$$u_0 = \bigvee \{f : \bigwedge \{ \{0\} : u \rightarrow I \} \}$$

$[f : u] \vdash f : u''$	$[f : u] \vdash f : u'$	
$[f : u] \vdash f \odot f : u'$		$w_2$
$[f : u] \vdash (f \odot f) \odot 7 : I$		$\dots$
$[\ ] \vdash \lambda f. (f \odot f) \odot 7 : u_0$		$w_1$
$[\ ] \vdash P = (\lambda f. (f \odot f) \odot 7) \odot (\lambda x. x) : I$		$[x : I] \vdash^1 x : I$
		$[x : u'] \vdash^2 x : u'$
		$[\ ] \vdash \lambda x. x : u$

Here  $w_1(x) = \{1\}$  and  $w_2(x) = \{2\}$

---

## The Flow Logic

---

Flow value:  $((\lambda x.e, me), M)$

Specification (coinductive):

$(\mathcal{C}, \rho) \models^{me} e$  denotes that

$(\mathcal{C}, \rho)$  is a correct analysis of  $e$  wrt.  $me$

We extend N&N to model P&P's cover:

$(\mathcal{C}, \rho) \models^{me} e_1 @ e_2$  iff

$\dots \forall ((\lambda x.e_0, me_0), M_0) \in \mathcal{C}(e_1, me).$

$\exists M \subseteq M_0 \dots \mathcal{C}(e_2, me) \subseteq \bigcup_{m \in M} \rho(x, m)$

Expressive power:

- Models monovariance (0-CFA)  
when mementa universe =  $\{\bullet\}$
- Models call-string based analysis ( $k$ -CFA)  
where mementa are strings (length  $\leq k$ )  
of application site labels
- Models argument-based analyses  
(encodes P&P) where mementa  
correspond to sets of flow values
- Does not model polymorphic splitting

## Translations and Roundtrips

**Theorem** Subject reduction (flows & types)

**Theorem** (proof by coinduction): valid and *uniform* typing translates into valid and safe flow analysis

**Theorem:** valid and safe flow analysis translates into valid and uniform typing

**Theorem:** the roundtrips are faithful, and

- the canonical flow analyses are those where everything is “reachable”
- the canonical type derivations are those which are “consistent”

---

## Conclusion

---

We have achieved isomorphism (for canonical items) by

- augmenting types with source tags and
- augmenting flows with sink tags

What is this work good for?

- Can switch between perspectives (cf. polar/rectangular)
- Can encode flow information in a typed intermediate language for compilation (e.g., CIL)