Stable Relations and Abstract Interpretation of Higher-Order Programs

Benoît Montagu
Joint work with Thomas Jensen

chocola seminar — January 28, 2021

— Celtique Research Team

Frame Inference

Goal: Detect what a function call does **not** change

Frame Inference

Goal: Detect what a function call does **not** change

In an imperative setting:

- Which parts of the global state are *unchanged*?
- Or: find a bound on the *writes* that a function can perform

Frame Inference

Goal: Detect what a function call does **not** change

In an imperative setting:

- Which parts of the global state are *unchanged*?
- Or: find a bound on the *writes* that a function can perform
- Example: `writes` clauses in Why3

```
type t = { mutable left: int; mutable right: bool }
let incr_left (r: t) : unit


= r.left <- r.left + 1
```

Frame Inference

Goal: Detect what a function call does **not** change

In an imperative setting:

- Which parts of the global state are *unchanged*?
- Or: find a bound on the *writes* that a function can perform
- Example: `writes` clauses in Why3

```
type t = { mutable left: int; mutable right: bool }

let incr_left (r: t) : unit
writes { r.left }
ensures { (old r).right = r.right }
= r.left <- r.left + 1
```

Frame Inference

Goal: Detect what a function call does **not** change

In a functional setting:

- Which parts of the input and the output are identical?

Frame Inference

Goal: Detect what a function call does **not** change

In a functional setting:

- Which parts of the input and the output are identical?
- Example:

```
type u = { left: int; right: bool }

let incr_left_func (r: u) : u

= { left = r.left + 1; right = r.right }
```

Frame Inference

Goal: Detect what a function call does **not** change

In a functional setting:

- Which parts of the input and the output are identical?
- Example:

```
type u = { left: int; right: bool }

let incr_left_func (r: u) : u
ensures { r.right = result.right }
= { left = r.left + 1; right = r.right }
```

Frame Inference

Goal: Detect what a function call does **not** change

Frames are input-output relations

Frame Inference

Goal: Detect what a function call does **not** change

Frames are input-output relations

Frames are useful information:

- For optimising compilers:
  Common sub-expression elimination (CSE) across function calls
- For program verification:
  Properties on the unchanged parts of the state are preserved

Frame Inference

Goal: Detect what a function call does **not** change

Frames are input-output relations

Frames are useful information:

- For optimising compilers:
  Common sub-expression elimination (CSE) across function calls
- For program verification:
  Properties on the unchanged parts of the state are preserved

But frames can be large

- Tedious and hard to write down for large programs
- Proofs can be very time consuming...

Frame Inference

Goal: Detect what a function call does **not** change

Frames are input-output relations

Let's compute frames automatically!

Static Analysis for Frame Inference: Previous Work

- **Oana F. Andreescu Ph.D. thesis** (former student of Celtique)

Oana Fabiana Andreescu (May 2017). 'Static Analysis of Functional Programs with an Application to the Frame Problem in Deductive Verification'. Thèse de doctorat. Université Rennes 1

Static Analysis for Frame Inference: Previous Work

- **Oana F. Andreescu Ph.D. thesis** (former student of Celtique)

Oana Fabiana Andreescu (May 2017). 'Static Analysis of Functional Programs with an Application to the Frame Problem in Deductive Verification'. Thèse de doctorat. Université Rennes 1

- Correlation analysis for a first-order language

Oana F. Andreescu et al. (Jan. 2019). 'Inferring Frame Conditions with Static Correlation Analysis'. In: *Proc. ACM Program. Lang.* 3.POPL

Static Analysis for Frame Inference: Previous Work

- **Oana F. Andreescu Ph.D. thesis** (former student of Celtique)

Oana Fabiana Andreescu (May 2017). 'Static Analysis of Functional Programs with an Application to the Frame Problem in Deductive Verification'. Thèse de doctorat. Université Rennes 1

- Correlation analysis for a first-order language

Oana F. Andreescu et al. (Jan. 2019). 'Inferring Frame Conditions with Static Correlation Analysis'. In: *Proc. ACM Program. Lang.* 3.POPL

- **Correlation abstract domain:** binary relations over values of algebraic datatypes

Static Analysis for Frame Inference: Previous Work

- **Oana F. Andreescu Ph.D. thesis** (former student of Celtique)

Oana Fabiana Andreescu (May 2017). 'Static Analysis of Functional Programs with an Application to the Frame Problem in Deductive Verification'. Thèse de doctorat. Université Rennes 1

- Correlation analysis for a first-order language

Oana F. Andreescu et al. (Jan. 2019). 'Inferring Frame Conditions with Static Correlation Analysis'. In: *Proc. ACM Program. Lang.* 3.POPL

- **Correlation abstract domain:** binary relations over values of algebraic datatypes
- **Application to an industrial OS micro-kernel:**
  68 % of invariant preservation lemmas automatically proved

In This Talk

Correlation inference: Extension to a **higher-order** language
An interesting theoretical exercise… with practical applications!

☞ Analysis of libraries, of monadic code, *etc.*

 Benoît Montagu and Thomas P. Jensen (2020). ‘Stable Relations and Abstract Interpretation of Higher-order Programs’. In: *Proc. ACM Program. Lang.* 4.ICFP, 119:1–119:30

Long-term goal: integrate such an analysis in a proof assistant to make interactive program verification more lightweight

Contributions / Outline

We follow the abstract interpretation methodology:

1. Define the property of interest
   - In this work: Input-output relations for λ-terms

Contributions / Outline

We follow the abstract interpretation methodology:

1. Define the property of interest
   - In this work: Input-output relations for λ-terms
2. Define a **collecting semantics** for that property
   - A denotation of λ-terms as input-output relations
   - Soundness and completeness proof
   - In the paper: An equivalent definition as program logic

Contributions / Outline

We follow the abstract interpretation methodology:

1. Define the property of interest
   - In this work: Input-output relations for λ-terms
2. Define a **collecting semantics** for that property
   - A denotation of λ-terms as input-output relations
   - Soundness and completeness proof
   - In the paper: An equivalent definition as program logic
3. Abstraction: apply successive over-approximations
   - Independent attribute abstraction
   - Correlation domain extended with 1^{st}-class functions
   - Outcome: A surprisingly expressive modular analysis!

Contributions / Outline

We follow the abstract interpretation methodology:

1. Define the property of interest
   - In this work: Input-output relations for λ-terms
2. Define a **collecting semantics** for that property
   - A denotation of λ-terms as input-output relations
   - Soundness and completeness proof
   - In the paper: An equivalent definition as program logic
3. Abstraction: apply successive over-approximations
   - Independent attribute abstraction
   - Correlation domain extended with 1^{st}-class functions
   - Outcome: A surprisingly expressive modular analysis!

Results formalised in Coq 

Input-Output Relations

Untyped λ-Calculus with Pairs and Sums

Definition (Terms)

$$
\begin{array}{rcl}
t & ::= & \mathsf{x} \mid \lambda \mathsf{x}.\, t \mid t\, t \mid \mathsf{let}\, \mathsf{x} = t\, \mathsf{in}\, t \\
& \mid & () \mid (t, t) \mid \pi_1\, t \mid \pi_2\, t \\
& \mid & \mathsf{inj}_1\, t \mid \mathsf{inj}_2\, t \mid \mathsf{match}\ t\ \mathsf{with}\ \mathsf{inj}_1\, \mathsf{x}_1 \to t \mid \mathsf{inj}_2\, \mathsf{x}_2 \to t
\end{array}
$$

Untyped λ-Calculus with Pairs and Sums

Definition (Terms)

$$\begin{aligned} t \;::=\;& \mathsf{x} \mid \lambda \mathsf{x}.\, t \mid t\, t \mid \mathsf{let}\, \mathsf{x} = t\, \mathsf{in}\, t \\ \mid\;& () \mid (t, t) \mid \pi_1\, t \mid \pi_2\, t \\ \mid\;& \mathsf{inj}_1\, t \mid \mathsf{inj}_2\, t \mid \mathsf{match}\; t\; \mathsf{with}\; \mathsf{inj}_1\, \mathsf{x}_1 \to t \mid \mathsf{inj}_2\, \mathsf{x}_2 \to t \end{aligned}$$

Definition (Closed values)

$$\begin{aligned} v \in \mathcal{V} \;::=\;& \lambda \mathsf{x}.\, t \quad \text{where} \quad \mathsf{fv}\, t \subseteq \{\mathsf{x}\} \\ \mid\;& () \mid (v, v) \mid \mathsf{inj}_1\, v \mid \mathsf{inj}_2\, v \end{aligned}$$

Untyped λ-Calculus with Pairs and Sums

Definition (Terms)

$$
\begin{array}{rcl}
t & ::= & \mathsf{x} \mid \lambda \mathsf{x}.\, t \mid t\, t \mid \mathsf{let}\, \mathsf{x} = t\, \mathsf{in}\, t \\
& \mid & () \mid (t, t) \mid \pi_1\, t \mid \pi_2\, t \\
& \mid & \mathsf{inj}_1\, t \mid \mathsf{inj}_2\, t \mid \mathsf{match}\ t\ \mathsf{with}\ \mathsf{inj}_1\, \mathsf{x}_1 \to t \mid \mathsf{inj}_2\, \mathsf{x}_2 \to t
\end{array}
$$

Definition (Closed values)

$$
\begin{array}{rcl}
v \in \mathcal{V} & ::= & \lambda \mathsf{x}.\, t \quad \text{where} \quad \mathsf{fv}\, t \subseteq \{\mathsf{x}\} \\
& \mid & () \mid (v, v) \mid \mathsf{inj}_1\, v \mid \mathsf{inj}_2\, v
\end{array}
$$

Standard small-step call-by-value semantics: $t \rightsquigarrow t'$

$$
(\lambda \mathsf{x}.\, t)\, v \rightsquigarrow t\,[\mathsf{x} \leftarrow v] \qquad \mathsf{let}\, \mathsf{x} = v\, \mathsf{in}\, t \rightsquigarrow t\,[\mathsf{x} \leftarrow v] \qquad \pi_i\ (v_1, v_2) \rightsquigarrow v_i
$$

$$
\begin{array}{l}
\mathsf{match}\ (\mathsf{inj}_i\, v)\ \mathsf{with} \\
\mid \mathsf{inj}_1\, \mathsf{x}_1 \ \to\ t_1 \mid \mathsf{inj}_2\, \mathsf{x}_2 \ \to\ t_2
\end{array}
\rightsquigarrow t_i\,[\mathsf{x}_i \leftarrow v] \qquad (+\ \text{congruence rules})
$$

The Property of Interest: Input-Output Relations

Inputs of a program

Value-substitutions: $\sigma \in \Sigma_V ::= \bullet \mid \sigma, \mathsf{x} \mapsto v$

The Property of Interest: Input-Output Relations

Inputs of a program

Value-substitutions: $\sigma \in \Sigma_V ::= \bullet \mid \sigma, x \mapsto v$

Reachability semantics

$$\bigcup_{\sigma \in \mathcal{I}} \{v \mid t \cdot \sigma \rightsquigarrow^\star v\}$$

- Traditionally used in static analyses (including CFAs)
- *"Given some inputs, what might be the outputs?"*

The Property of Interest: Input-Output Relations

Inputs of a program

Value-substitutions: $\sigma \in \Sigma_V ::= \bullet \mid \sigma, x \mapsto v$

Input-output relational semantics

$$(\!|t|\!)_{\mathcal{I}} \triangleq \bigcup_{\sigma \in \mathcal{I}} \{(\sigma, v) \mid t \cdot \sigma \rightsquigarrow^{\star} v\}$$

- Used in *relational* static analyses
- *"Given some inputs, what might be the input/output pairs?"*
- I/O semantics is more precise than reachability semantics

The Property of Interest: Input-Output Relations

Inputs of a program

Value-substitutions: $\sigma \in \Sigma_V ::= \bullet \mid \sigma, \mathsf{x} \mapsto v$

Input-output relational semantics

$$(\!|t|\!)_{\mathcal{I}} \triangleq \bigcup_{\sigma \in \mathcal{I}} \{(\sigma, v) \mid t \cdot \sigma \rightsquigarrow^{\star} v\}$$

- Used in *relational* static analyses
- *"Given some inputs, what might be the input/output pairs?"*
- I/O semantics is more precise than reachability semantics

Example: $(\!|t_1|\!)_{\mathcal{I}_1} = \{(\sigma, 0) \mid \sigma(\mathsf{x}) < 0\} \cup \{(\sigma, v) \mid 0 \leq \sigma(\mathsf{x}) \leq v\}$

With reachability semantics, we could only express: $\{v \mid 0 \leq v\}$

Over-Approximations and Reduction

A general property: If $t_1 \rightsquigarrow t_2$, then $(\!|t_2|\!)_{\mathcal{I}} \subseteq (\!|t_1|\!)_{\mathcal{I}}$ (by def. of $(\!|\cdot|\!)$)

Over-Approximations and Reduction

A general property: If $t_1 \rightsquigarrow t_2$, then $(\!|t_2|\!)_{\mathcal{I}} \subseteq (\!|t_1|\!)_{\mathcal{I}}$ (by def. of $(\!|\cdot|\!)$)

Consequence: If $(\!|t_1|\!)_{\mathcal{I}} \subseteq \mathcal{R}$ and $t_1 \rightsquigarrow t_2$, then $(\!|t_2|\!)_{\mathcal{I}} \subseteq \mathcal{R}$

Over-Approximations and Reduction

A general property: If $t_1 \rightsquigarrow t_2$, then $(\!|t_2|\!)_{\mathcal{I}} \subseteq (\!|t_1|\!)_{\mathcal{I}}$ (by def. of $(\!|\cdot|\!)$)

Consequence: If $(\!|t_1|\!)_{\mathcal{I}} \subseteq \mathcal{R}$ and $t_1 \rightsquigarrow t_2$, then $(\!|t_2|\!)_{\mathcal{I}} \subseteq \mathcal{R}$

"*Being a sound approximation*" is preserved by reduction, independently of *how* the over-approximation is performed.

☞ A general property in the abstract interpretation framework

☞ Enables modular soundness proofs

Stable Input-Output Relations

A central fact: defining more inputs than necessary is harmless

Stable Input-Output Relations

A central fact: defining more inputs than necessary is harmless

Definition (Extension ordering)

$$\sigma \sqsubseteq \sigma' \triangleq \operatorname{dom}\sigma \subseteq \operatorname{dom}\sigma' \wedge \forall x \in \operatorname{dom}\sigma, \sigma(x) = \sigma'(x)$$

Read: "σ' *is more defined than* σ" or: "σ' *extends* σ"

Stable Input-Output Relations

A central fact: defining more inputs than necessary is harmless

Definition (Extension ordering)

$$\sigma \sqsubseteq \sigma' \triangleq \mathrm{dom}\,\sigma \subseteq \mathrm{dom}\,\sigma' \wedge \forall x \in \mathrm{dom}\,\sigma, \sigma(x) = \sigma'(x)$$

Read: "σ' *is more defined than* σ" or: "σ' *extends* σ"

Lemma

If $\mathrm{fv}\,t \subseteq \mathrm{dom}\,\sigma$ *and* $\sigma \sqsubseteq \sigma'$, *then:* $t \cdot \sigma = t \cdot \sigma'$

Stable Input-Output Relations

A central fact: defining more inputs than necessary is harmless

Definition (Extension ordering)

$$\sigma \sqsubseteq \sigma' \triangleq \mathsf{dom}\,\sigma \subseteq \mathsf{dom}\,\sigma' \wedge \forall x \in \mathsf{dom}\,\sigma, \sigma(x) = \sigma'(x)$$

Read: "σ' *is more defined than* σ" or: "σ' *extends* σ"

Lemma

If $\mathsf{fv}\,t \subseteq \mathsf{dom}\,\sigma$ *and* $\sigma \sqsubseteq \sigma'$, *then:* $t \cdot \sigma = t \cdot \sigma'$

Corollary

If $\mathcal{I}$ *contains only substitutions whose domains contain* $\mathsf{fv}\,t$ *and if* $\mathcal{I}$ *is upper-closed under* $\sqsubseteq$*:*

$$\text{If } (\sigma, v) \in (\!|t|\!)_{\mathcal{I}} \text{ and } \sigma \sqsubseteq \sigma', \text{ then: } (\sigma', v) \in (\!|t|\!)_{\mathcal{I}}$$

Stable Input-Output Relations

A central fact: defining more inputs than necessary is harmless

Definition (Extension ordering)

$$\sigma \sqsubseteq \sigma' \triangleq \mathsf{dom}\,\sigma \subseteq \mathsf{dom}\,\sigma' \wedge \forall x \in \mathsf{dom}\,\sigma, \sigma(x) = \sigma'(x)$$

Read: "σ' *is more defined than* σ" or: "σ' *extends* σ"

Definition

A relation $\mathcal{R}$ is stable when for any substitution σ and value v:

$$(\sigma, v) \in \mathcal{R} \text{ implies } \forall \sigma',\ \sigma \sqsubseteq \sigma' \Longrightarrow (\sigma', v) \in \mathcal{R}$$

Stable Input-Output Relations

A central fact: defining more inputs than necessary is harmless

Definition (Extension ordering)

$$\sigma \sqsubseteq \sigma' \triangleq \mathsf{dom}\,\sigma \subseteq \mathsf{dom}\,\sigma' \wedge \forall x \in \mathsf{dom}\,\sigma, \sigma(x) = \sigma'(x)$$

Read: "σ' *is more defined than* σ" or: "σ' *extends* σ"

Definition

A relation $\mathcal{R}$ is stable when for any substitution σ and value v:

$$(\sigma, v) \in \mathcal{R} \text{ implies } \forall \sigma',\ \sigma \sqsubseteq \sigma' \Longrightarrow (\sigma', v) \in \mathcal{R}$$

- Stability is essential for the *soundness* of this work
- Stability internalises the property of *weakening*

Stable Input-Output Relations

A central fact: defining more inputs than necessary is harmless

Definition (Extension ordering)

$$\sigma \sqsubseteq \sigma' \triangleq \mathsf{dom}\,\sigma \subseteq \mathsf{dom}\,\sigma' \wedge \forall x \in \mathsf{dom}\,\sigma, \sigma(x) = \sigma'(x)$$

Read: "σ' *is more defined than* σ" or: "σ' *extends* σ"

Definition

A relation $\mathcal{R}$ is stable when for any substitution σ and value v:

$$(\sigma, v) \in \mathcal{R} \text{ implies } \forall \sigma',\ \sigma \sqsubseteq \sigma' \Longrightarrow (\sigma', v) \in \mathcal{R}$$

- Stability is essential for the *soundness* of this work
- Stability internalises the property of *weakening*
- Familiar to logicians: Kripke's *persistence*
- Familiar to semanticists: Monotone function $(\Sigma_v, \sqsubseteq) \to (\wp(\mathcal{V}), \subseteq)$

A Relational Collecting Semantics

A Denotation of Programs

A denotation of programs $[\![t]\!]_E$ that computes an input-output relation:

- The resulting relation is **stable**
- E is an environment that maps variables to **stable** relations

A Denotation of Programs

A denotation of programs $[\![t]\!]_E$ that computes an input-output relation:

- The resulting relation is **stable**
- E is an environment that maps variables to **stable** relations
- E denotes a ($\sqsubseteq$-upper closed) set of inputs $[\![E]\!]$

A Denotation of Programs

A denotation of programs $[\![t]\!]_E$ that computes an input-output relation:

- The resulting relation is **stable**
- E is an environment that maps variables to **stable** relations
- E denotes a ($\sqsubseteq$-upper closed) set of inputs $[\![E]\!]$
- The denotation $[\![t]\!]_E$ is defined in a **compositional** way

A Denotation of Programs

A denotation of programs $[\![t]\!]_E$ that computes an input-output relation:

- The resulting relation is **stable**
- E is an environment that maps variables to **stable** relations
- E denotes a ($\sqsubseteq$-upper closed) set of inputs $[\![E]\!]$
- The denotation $[\![t]\!]_E$ is defined in a **compositional** way

Theorem (Soundness)

$$(\!|t|\!)_{[\![E]\!]} \subseteq [\![t]\!]_E$$

i.e. if $\sigma \in [\![E]\!]$ and $t \cdot \sigma \rightsquigarrow^\star v$, then $(\sigma, v) \in [\![t]\!]_E$

A Denotation of Programs

A denotation of programs $[\![t]\!]_E$ that computes an input-output relation:

- The resulting relation is **stable**
- E is an environment that maps variables to **stable** relations
- E denotes a ($\sqsubseteq$-upper closed) set of inputs $[\![E]\!]$
- The denotation $[\![t]\!]_E$ is defined in a **compositional** way

Theorem (Soundness)

$$(\!|t|\!)_{[\![E]\!]} \subseteq [\![t]\!]_E$$

Theorem (Completeness)

$$(\!|t|\!)_{[\![E]\!]} \supseteq [\![t]\!]_E$$

i.e. if $\sigma \in [\![E]\!]$ and $t \cdot \sigma \rightsquigarrow^\star v$, then $(\sigma, v) \in [\![t]\!]_E$

Verified in Coq 

A Compositional Definition (Pairs)

A simple calculation for pairs of terms:

$$(\!|(t_1, t_2)|\!)_{\mathcal{I}} \quad = \quad \{(\sigma, v) \mid (t_1, t_2) \cdot \sigma \rightsquigarrow^{\star} v \ \wedge \ \sigma \in \mathcal{I}\}$$

A Compositional Definition (Pairs)

A simple calculation for pairs of terms:

$$\begin{aligned} (\!|(t_1, t_2)|\!)_{\mathcal{I}} &= \{(\sigma, v) \mid (t_1, t_2) \cdot \sigma \rightsquigarrow^\star v \ \wedge \ \sigma \in \mathcal{I}\} \\ &= \{(\sigma, (v_1, v_2)) \mid t_1 \cdot \sigma \rightsquigarrow^\star v_1 \ \wedge \ t_2 \cdot \sigma \rightsquigarrow^\star v_2 \ \wedge \ \sigma \in \mathcal{I}\} \end{aligned}$$

A Compositional Definition (Pairs)

A simple calculation for pairs of terms:

$$\begin{aligned}
(\!|(t_1, t_2)|\!)_{\mathcal{I}} &= \{(\sigma, v) \mid (t_1, t_2) \cdot \sigma \rightsquigarrow^\star v \ \wedge \ \sigma \in \mathcal{I}\} \\
&= \{(\sigma, (v_1, v_2)) \mid t_1 \cdot \sigma \rightsquigarrow^\star v_1 \ \wedge \ t_2 \cdot \sigma \rightsquigarrow^\star v_2 \ \wedge \ \sigma \in \mathcal{I}\} \\
&= \{(\sigma, (v_1, v_2)) \mid (\sigma, v_1) \in (\!|t_1|\!)_{\mathcal{I}} \ \wedge \ (\sigma, v_2) \in (\!|t_2|\!)_{\mathcal{I}}\}
\end{aligned}$$

A Compositional Definition (Pairs)

A simple calculation for pairs of terms:

$$\begin{aligned}
(\!|(t_1, t_2)|\!)_{\mathcal{I}} &= \{(\sigma, v) \mid (t_1, t_2) \cdot \sigma \rightsquigarrow^\star v \ \wedge \ \sigma \in \mathcal{I}\} \\
&= \{(\sigma, (v_1, v_2)) \mid t_1 \cdot \sigma \rightsquigarrow^\star v_1 \ \wedge \ t_2 \cdot \sigma \rightsquigarrow^\star v_2 \ \wedge \ \sigma \in \mathcal{I}\} \\
&= \{(\sigma, (v_1, v_2)) \mid (\sigma, v_1) \in (\!|t_1|\!)_{\mathcal{I}} \ \wedge \ (\sigma, v_2) \in (\!|t_2|\!)_{\mathcal{I}}\} \\
&= \mathsf{PAIR}^{\mathsf{R}}\,((\!|t_1|\!)_{\mathcal{I}}, (\!|t_2|\!)_{\mathcal{I}})
\end{aligned}$$

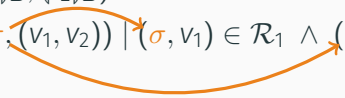
where $\mathsf{PAIR}^{\mathsf{R}}\,(\mathcal{R}_1, \mathcal{R}_2) \triangleq \{(\sigma, (v_1, v_2)) \mid (\sigma, v_1) \in \mathcal{R}_1 \ \wedge \ (\sigma, v_2) \in \mathcal{R}_2\}$

A Compositional Definition (Pairs)

A simple calculation for pairs of terms:

$$\begin{aligned}
(\!|(t_1, t_2)|\!)_{\mathcal{I}} &= \{(\sigma, v) \mid (t_1, t_2) \cdot \sigma \rightsquigarrow^\star v \ \wedge \ \sigma \in \mathcal{I}\} \\
&= \{(\sigma, (v_1, v_2)) \mid t_1 \cdot \sigma \rightsquigarrow^\star v_1 \ \wedge \ t_2 \cdot \sigma \rightsquigarrow^\star v_2 \ \wedge \ \sigma \in \mathcal{I}\} \\
&= \{(\sigma, (v_1, v_2)) \mid (\sigma, v_1) \in (\!|t_1|\!)_{\mathcal{I}} \ \wedge \ (\sigma, v_2) \in (\!|t_2|\!)_{\mathcal{I}}\} \\
&= \mathrm{PAIR}^{\mathrm{R}}\left((\!|t_1|\!)_{\mathcal{I}}, (\!|t_2|\!)_{\mathcal{I}}\right)
\end{aligned}$$

where $\mathrm{PAIR}^{\mathrm{R}}(\mathcal{R}_1, \mathcal{R}_2) \triangleq \{(\sigma, (v_1, v_2)) \mid (\sigma, v_1) \in \mathcal{R}_1 \ \wedge \ (\sigma, v_2) \in \mathcal{R}_2\}$

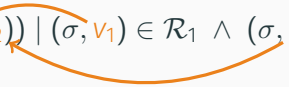


A Compositional Definition (Pairs)

A simple calculation for pairs of terms:

$$\begin{aligned}
(\!|(t_1, t_2)|\!)_{\mathcal{I}} &= \{(\sigma, v) \mid (t_1, t_2) \cdot \sigma \rightsquigarrow^\star v \ \wedge \ \sigma \in \mathcal{I}\} \\
&= \{(\sigma, (v_1, v_2)) \mid t_1 \cdot \sigma \rightsquigarrow^\star v_1 \ \wedge \ t_2 \cdot \sigma \rightsquigarrow^\star v_2 \ \wedge \ \sigma \in \mathcal{I}\} \\
&= \{(\sigma, (v_1, v_2)) \mid (\sigma, v_1) \in (\!|t_1|\!)_{\mathcal{I}} \ \wedge \ (\sigma, v_2) \in (\!|t_2|\!)_{\mathcal{I}}\} \\
&= \mathrm{PAIR}^{\mathrm{R}}\left((\!|t_1|\!)_{\mathcal{I}}, (\!|t_2|\!)_{\mathcal{I}}\right)
\end{aligned}$$

where $\mathrm{PAIR}^{\mathrm{R}}(\mathcal{R}_1, \mathcal{R}_2) \triangleq \{(\sigma, (v_1, v_2)) \mid (\sigma, v_1) \in \mathcal{R}_1 \ \wedge \ (\sigma, v_2) \in \mathcal{R}_2\}$



A Compositional Definition (Pairs)

A simple calculation for pairs of terms:

$$\begin{aligned}
(\!|(t_1, t_2)|\!)_{\mathcal{I}} &= \{(\sigma, v) \mid (t_1, t_2) \cdot \sigma \rightsquigarrow^\star v \ \wedge \ \sigma \in \mathcal{I}\} \\
&= \{(\sigma, (v_1, v_2)) \mid t_1 \cdot \sigma \rightsquigarrow^\star v_1 \ \wedge \ t_2 \cdot \sigma \rightsquigarrow^\star v_2 \ \wedge \ \sigma \in \mathcal{I}\} \\
&= \{(\sigma, (v_1, v_2)) \mid (\sigma, v_1) \in (\!|t_1|\!)_{\mathcal{I}} \ \wedge \ (\sigma, v_2) \in (\!|t_2|\!)_{\mathcal{I}}\} \\
&= \mathrm{PAIR}^{\mathrm{R}}\left((\!|t_1|\!)_{\mathcal{I}}, (\!|t_2|\!)_{\mathcal{I}}\right)
\end{aligned}$$

where $\mathrm{PAIR}^{\mathrm{R}}(\mathcal{R}_1, \mathcal{R}_2) \triangleq \{(\sigma, (v_1, v_2)) \mid (\sigma, v_1) \in \mathcal{R}_1 \ \wedge \ (\sigma, v_2) \in \mathcal{R}_2\}$

Definition

$$[\![(t_1, t_2)]\!]_E \ \triangleq \ \mathrm{PAIR}^{\mathrm{R}}\left([\![t_1]\!]_E, [\![t_2]\!]_E\right)$$

A Compositional Definition (Pairs)

A simple calculation for pairs of terms:

$$\begin{aligned}
(\!|(t_1, t_2)|\!)_{\mathcal{I}} &= \{(\sigma, v) \mid (t_1, t_2) \cdot \sigma \rightsquigarrow^\star v \ \wedge \ \sigma \in \mathcal{I}\} \\
&= \{(\sigma, (v_1, v_2)) \mid t_1 \cdot \sigma \rightsquigarrow^\star v_1 \ \wedge \ t_2 \cdot \sigma \rightsquigarrow^\star v_2 \ \wedge \ \sigma \in \mathcal{I}\} \\
&= \{(\sigma, (v_1, v_2)) \mid (\sigma, v_1) \in (\!|t_1|\!)_{\mathcal{I}} \ \wedge \ (\sigma, v_2) \in (\!|t_2|\!)_{\mathcal{I}}\} \\
&= \mathrm{PAIR}^{\mathrm{R}} \left((\!|t_1|\!)_{\mathcal{I}}, (\!|t_2|\!)_{\mathcal{I}} \right)
\end{aligned}$$

where $\mathrm{PAIR}^{\mathrm{R}} (\mathcal{R}_1, \mathcal{R}_2) \triangleq \{(\sigma, (v_1, v_2)) \mid (\sigma, v_1) \in \mathcal{R}_1 \ \wedge \ (\sigma, v_2) \in \mathcal{R}_2\}$

Definition

$$[\![(t_1, t_2)]\!]_E \quad \triangleq \quad \mathrm{PAIR}^{\mathrm{R}} \left([\![t_1]\!]_E , [\![t_2]\!]_E \right)$$

- Combinator used in the correlation abstract domain (POPL'19)
- Comes from the theory of allegories

Richard Bird and Oege de Moor (1996). 'The Algebra of Programming'. In: *NATO ASI DPD*, pp. 167–203

A Compositional Definition (Projections)

$$
\begin{aligned}
(\!|\pi_1\, t|\!)_{\mathcal{I}} &= (\!|t|\!)_{\mathcal{I}}; \{((v_1, v_2), v_1) \mid v_1, v_2 \in \mathcal{V}\} \\
(\!|\pi_2\, t|\!)_{\mathcal{I}} &= (\!|t|\!)_{\mathcal{I}}; \{((v_1, v_2), v_2) \mid v_1, v_2 \in \mathcal{V}\}
\end{aligned}
$$

A Compositional Definition (Projections)

$$
\begin{array}{rcccl}
(\!|\pi_1\, t|\!)_{\mathcal{I}} & = & (\!|t|\!)_{\mathcal{I}}; \{((v_1, v_2), v_1) \mid v_1, v_2 \in \mathcal{V}\} & = & (\!|t|\!)_{\mathcal{I}}; \mathsf{PAIR}^{\mathsf{L}}(\mathsf{EQ}, \top) \\
(\!|\pi_2\, t|\!)_{\mathcal{I}} & = & (\!|t|\!)_{\mathcal{I}}; \{((v_1, v_2), v_2) \mid v_1, v_2 \in \mathcal{V}\} & = & (\!|t|\!)_{\mathcal{I}}; \mathsf{PAIR}^{\mathsf{L}}(\top, \mathsf{EQ})
\end{array}
$$

$$
\begin{array}{rrcl}
\text{where:} & \mathsf{PAIR}^{\mathsf{L}}(\mathcal{R}_1, \mathcal{R}_2) & \triangleq & \{((v_1, v_2), v) \mid (v_1, v) \in \mathcal{R}_1 \wedge (v_2, v) \in \mathcal{R}_2\} \\
& \top & \triangleq & \mathcal{V} \times \mathcal{V} \\
& \mathsf{EQ} & \triangleq & \{(v, v) \mid v \in \mathcal{V}\}
\end{array}
$$

A Compositional Definition (Projections)

$$
\begin{array}{rcccl}
(\!|\pi_1\, t|\!)_{\mathcal{I}} & = & (\!|t|\!)_{\mathcal{I}}; \{((v_1, v_2), v_1) \mid v_1, v_2 \in \mathcal{V}\} & = & (\!|t|\!)_{\mathcal{I}}; \mathrm{PAIR}^{\mathrm{L}}(\mathrm{EQ}, \top) \\
(\!|\pi_2\, t|\!)_{\mathcal{I}} & = & (\!|t|\!)_{\mathcal{I}}; \{((v_1, v_2), v_2) \mid v_1, v_2 \in \mathcal{V}\} & = & (\!|t|\!)_{\mathcal{I}}; \mathrm{PAIR}^{\mathrm{L}}(\top, \mathrm{EQ})
\end{array}
$$

$$
\begin{array}{rcl}
\text{where: } \mathrm{PAIR}^{\mathrm{L}}(\mathcal{R}_1, \mathcal{R}_2) & \triangleq & \{((v_1, v_2), v) \mid (v_1, v) \in \mathcal{R}_1 \wedge (v_2, v) \in \mathcal{R}_2\} \\
\top & \triangleq & \mathcal{V} \times \mathcal{V} \\
\mathrm{EQ} & \triangleq & \{(v, v) \mid v \in \mathcal{V}\}
\end{array}
$$

A Compositional Definition (Projections)

$$\begin{array}{rcccl} (\!|\pi_1\, t|\!)_{\mathcal{I}} & = & (\!|t|\!)_{\mathcal{I}}; \{((v_1, v_2), v_1) \mid v_1, v_2 \in \mathcal{V}\} & = & (\!|t|\!)_{\mathcal{I}}; \mathrm{PAIR}^{\mathsf{L}}(\mathrm{EQ}, \top) \\ (\!|\pi_2\, t|\!)_{\mathcal{I}} & = & (\!|t|\!)_{\mathcal{I}}; \{((v_1, v_2), v_2) \mid v_1, v_2 \in \mathcal{V}\} & = & (\!|t|\!)_{\mathcal{I}}; \mathrm{PAIR}^{\mathsf{L}}(\top, \mathrm{EQ}) \end{array}$$

$$\begin{array}{rcl} \text{where: } \mathrm{PAIR}^{\mathsf{L}}(\mathcal{R}_1, \mathcal{R}_2) & \triangleq & \{((v_1, v_2), v) \mid (v_1, v) \in \mathcal{R}_1 \wedge (v_2, v) \in \mathcal{R}_2\} \\ \top & \triangleq & \mathcal{V} \times \mathcal{V} \\ \mathrm{EQ} & \triangleq & \{(v, v) \mid v \in \mathcal{V}\} \end{array}$$

A Compositional Definition (Projections)

$$
\begin{array}{rcl}
(\!|\pi_1\, t|\!)_{\mathcal{I}} & = & (\!|t|\!)_{\mathcal{I}}; \{((v_1, v_2), v_1) \mid v_1, v_2 \in \mathcal{V}\} \quad = \quad (\!|t|\!)_{\mathcal{I}}; \mathrm{PAIR}^{\mathrm{L}}(\mathrm{EQ}, \top) \\
(\!|\pi_2\, t|\!)_{\mathcal{I}} & = & (\!|t|\!)_{\mathcal{I}}; \{((v_1, v_2), v_2) \mid v_1, v_2 \in \mathcal{V}\} \quad = \quad (\!|t|\!)_{\mathcal{I}}; \mathrm{PAIR}^{\mathrm{L}}(\top, \mathrm{EQ})
\end{array}
$$

where:
$$
\begin{array}{rcl}
\mathrm{PAIR}^{\mathrm{L}}(\mathcal{R}_1, \mathcal{R}_2) & \triangleq & \{((v_1, v_2), v) \mid (v_1, v) \in \mathcal{R}_1 \wedge (v_2, v) \in \mathcal{R}_2\} \\
\top & \triangleq & \mathcal{V} \times \mathcal{V} \\
\mathrm{EQ} & \triangleq & \{(v, v) \mid v \in \mathcal{V}\}
\end{array}
$$

Definition

$$
\begin{array}{rcl}
[\![\pi_1\, t]\!]_E & \triangleq & [\![t]\!]_E ; \mathrm{PAIR}^{\mathrm{L}}(\mathrm{EQ}, \top) \\
[\![\pi_2\, t]\!]_E & \triangleq & [\![t]\!]_E ; \mathrm{PAIR}^{\mathrm{L}}(\top, \mathrm{EQ})
\end{array}
$$

Again, $\mathrm{PAIR}^{\mathrm{L}}(\mathcal{R}_1, \mathcal{R}_2)$ is used in correlation abstract domain, and originates from allegory theory.

Algebraic Laws for Relation Combinators

$$\mathrm{PAIR}^{\mathrm{R}}\left(\mathcal{R}_1 \cup \mathcal{R}_2, \mathcal{R}_3\right) = \mathrm{PAIR}^{\mathrm{R}}\left(\mathcal{R}_1, \mathcal{R}_3\right) \cup \mathrm{PAIR}^{\mathrm{R}}\left(\mathcal{R}_2, \mathcal{R}_3\right) \quad (1)$$

$$\mathrm{PAIR}^{\mathrm{R}}\left(\mathcal{R}_1, \mathcal{R}_2\right) \cup \mathrm{PAIR}^{\mathrm{R}}\left(\mathcal{R}_3, \mathcal{R}_4\right) \subseteq \mathrm{PAIR}^{\mathrm{R}}\left(\mathcal{R}_1 \cup \mathcal{R}_3, \mathcal{R}_2 \cup \mathcal{R}_4\right) \quad (2)$$

Algebraic Laws for Relation Combinators

$$\mathsf{PAIR}^{\mathsf{R}}\left(\mathcal{R}_1 \cup \mathcal{R}_2, \mathcal{R}_3\right) = \mathsf{PAIR}^{\mathsf{R}}\left(\mathcal{R}_1, \mathcal{R}_3\right) \cup \mathsf{PAIR}^{\mathsf{R}}\left(\mathcal{R}_2, \mathcal{R}_3\right) \quad (1)$$

$$\mathsf{PAIR}^{\mathsf{R}}\left(\mathcal{R}_1, \mathcal{R}_2\right) \cup \mathsf{PAIR}^{\mathsf{R}}\left(\mathcal{R}_3, \mathcal{R}_4\right) \subseteq \mathsf{PAIR}^{\mathsf{R}}\left(\mathcal{R}_1 \cup \mathcal{R}_3, \mathcal{R}_2 \cup \mathcal{R}_4\right) \quad (2)$$

$$\mathsf{PAIR}^{\mathsf{R}}\left(\mathcal{R}_1 \cap \mathcal{R}_2, \mathcal{R}_3\right) = \mathsf{PAIR}^{\mathsf{R}}\left(\mathcal{R}_1, \mathcal{R}_3\right) \cap \mathsf{PAIR}^{\mathsf{R}}\left(\mathcal{R}_2, \mathcal{R}_3\right) \quad (3)$$

$$\mathsf{PAIR}^{\mathsf{R}}\left(\mathcal{R}_1, \mathcal{R}_2\right) \cap \mathsf{PAIR}^{\mathsf{R}}\left(\mathcal{R}_3, \mathcal{R}_4\right) = \mathsf{PAIR}^{\mathsf{R}}\left(\mathcal{R}_1 \cap \mathcal{R}_3, \mathcal{R}_2 \cap \mathcal{R}_4\right) \quad (4)$$

Algebraic Laws for Relation Combinators

$$\mathrm{PAIR}^{\mathrm{R}}\,(\mathcal{R}_1 \cup \mathcal{R}_2, \mathcal{R}_3) = \mathrm{PAIR}^{\mathrm{R}}\,(\mathcal{R}_1, \mathcal{R}_3) \cup \mathrm{PAIR}^{\mathrm{R}}\,(\mathcal{R}_2, \mathcal{R}_3) \quad (1)$$

$$\mathrm{PAIR}^{\mathrm{R}}\,(\mathcal{R}_1, \mathcal{R}_2) \cup \mathrm{PAIR}^{\mathrm{R}}\,(\mathcal{R}_3, \mathcal{R}_4) \subseteq \mathrm{PAIR}^{\mathrm{R}}\,(\mathcal{R}_1 \cup \mathcal{R}_3, \mathcal{R}_2 \cup \mathcal{R}_4) \quad (2)$$

$$\mathrm{PAIR}^{\mathrm{R}}\,(\mathcal{R}_1 \cap \mathcal{R}_2, \mathcal{R}_3) = \mathrm{PAIR}^{\mathrm{R}}\,(\mathcal{R}_1, \mathcal{R}_3) \cap \mathrm{PAIR}^{\mathrm{R}}\,(\mathcal{R}_2, \mathcal{R}_3) \quad (3)$$

$$\mathrm{PAIR}^{\mathrm{R}}\,(\mathcal{R}_1, \mathcal{R}_2) \cap \mathrm{PAIR}^{\mathrm{R}}\,(\mathcal{R}_3, \mathcal{R}_4) = \mathrm{PAIR}^{\mathrm{R}}\,(\mathcal{R}_1 \cap \mathcal{R}_3, \mathcal{R}_2 \cap \mathcal{R}_4) \quad (4)$$

$$\mathrm{PAIR}^{\mathrm{R}}\,(\mathcal{R}_1, \mathcal{R}_2)\,;\,\mathrm{PAIR}^{\mathrm{L}}\,(\mathcal{R}_3, \mathcal{R}_4) = (\mathcal{R}_1; \mathcal{R}_3) \cap (\mathcal{R}_2; \mathcal{R}_4) \quad (5)$$

Algebraic Laws for Relation Combinators

$$\mathrm{PAIR}^{\mathrm{R}}(\mathcal{R}_1 \cup \mathcal{R}_2, \mathcal{R}_3) = \mathrm{PAIR}^{\mathrm{R}}(\mathcal{R}_1, \mathcal{R}_3) \cup \mathrm{PAIR}^{\mathrm{R}}(\mathcal{R}_2, \mathcal{R}_3) \quad (1)$$

$$\mathrm{PAIR}^{\mathrm{R}}(\mathcal{R}_1, \mathcal{R}_2) \cup \mathrm{PAIR}^{\mathrm{R}}(\mathcal{R}_3, \mathcal{R}_4) \subseteq \mathrm{PAIR}^{\mathrm{R}}(\mathcal{R}_1 \cup \mathcal{R}_3, \mathcal{R}_2 \cup \mathcal{R}_4) \quad (2)$$

$$\mathrm{PAIR}^{\mathrm{R}}(\mathcal{R}_1 \cap \mathcal{R}_2, \mathcal{R}_3) = \mathrm{PAIR}^{\mathrm{R}}(\mathcal{R}_1, \mathcal{R}_3) \cap \mathrm{PAIR}^{\mathrm{R}}(\mathcal{R}_2, \mathcal{R}_3) \quad (3)$$

$$\mathrm{PAIR}^{\mathrm{R}}(\mathcal{R}_1, \mathcal{R}_2) \cap \mathrm{PAIR}^{\mathrm{R}}(\mathcal{R}_3, \mathcal{R}_4) = \mathrm{PAIR}^{\mathrm{R}}(\mathcal{R}_1 \cap \mathcal{R}_3, \mathcal{R}_2 \cap \mathcal{R}_4) \quad (4)$$

$$\mathrm{PAIR}^{\mathrm{R}}(\mathcal{R}_1, \mathcal{R}_2) ; \mathrm{PAIR}^{\mathrm{L}}(\mathcal{R}_3, \mathcal{R}_4) = (\mathcal{R}_1; \mathcal{R}_3) \cap (\mathcal{R}_2; \mathcal{R}_4) \quad (5)$$

All those laws come from the theory of allegories.

The correlation domain (POPL'19) is a calculus of symbolic relations that exploits these laws.

Let-bindings

Let's write $E \vdash t : \mathcal{R}$ for $[\![t]\!]_E = \mathcal{R}$

We would like to mimic a typing rule for let-bindings:

$$\frac{E \vdash t_1 : \mathcal{R}_1 \qquad \mathsf{x} \notin \operatorname{dom} E \qquad E, \mathsf{x} : \mathcal{R}_1 \vdash t_2 : \mathcal{R}_2}{E \vdash \text{let } \mathsf{x} = t_1 \text{ in } t_2 : \quad \mathcal{R}?}$$

Let-bindings

Let's write $E \vdash t : \mathcal{R}$ for $[\![t]\!]_E = \mathcal{R}$

We would like to mimic a typing rule for let-bindings:

$$\frac{E \vdash t_1 : \mathcal{R}_1 \qquad \mathsf{x} \notin \operatorname{dom} E \qquad E, \mathsf{x} : \mathcal{R}_1 \vdash t_2 : \mathcal{R}_2}{E \vdash \operatorname{let} \mathsf{x} = t_1 \operatorname{in} t_2 : \quad \mathcal{R}?}$$

$$\sigma \in [\![E]\!] \quad (\operatorname{let} \mathsf{x} = t_1 \operatorname{in} t_2) \cdot \sigma \rightsquigarrow^\star v_2 \quad (\sigma, v_2) \in \mathcal{R}$$

Let-bindings

Let's write $E \vdash t : \mathcal{R}$ for $[\![t]\!]_E = \mathcal{R}$

We would like to mimic a typing rule for let-bindings:

$$\frac{\begin{array}{c} t_1 \cdot \sigma \rightsquigarrow^\star v_1 \\ \sigma \in [\![E]\!] \qquad (\sigma, v_1) \in \mathcal{R}_1 \end{array} \qquad E \vdash t_1 : \mathcal{R}_1 \qquad \mathsf{x} \notin \operatorname{dom} E \qquad E, \mathsf{x} : \mathcal{R}_1 \vdash t_2 : \mathcal{R}_2}{\begin{array}{c} E \vdash \operatorname{let} \mathsf{x} = t_1 \operatorname{in} t_2 : \quad \mathcal{R}? \\ \sigma \in [\![E]\!] \quad (\operatorname{let} \mathsf{x} = t_1 \operatorname{in} t_2) \cdot \sigma \rightsquigarrow^\star v_2 \quad (\sigma, v_2) \in \mathcal{R} \end{array}}$$

Let-bindings

Let's write $E \vdash t : \mathcal{R}$ for $[\![t]\!]_E = \mathcal{R}$

We would like to mimic a typing rule for let-bindings:

$$\frac{\begin{array}{ccccc} t_1 \cdot \sigma \rightsquigarrow^\star v_1 & & & & t_2 \cdot \sigma\,[\mathsf{x} \mapsto v_1] \rightsquigarrow^\star v_2 \\ \sigma \in [\![E]\!] \qquad (\sigma, v_1) \in \mathcal{R}_1 & \Longrightarrow & \sigma\,[\mathsf{x} \mapsto v_1] \in [\![E, \mathsf{x} : \mathcal{R}_1]\!] & & (\sigma\,[\mathsf{x} \mapsto v_1], v_2) \in \mathcal{R}_2 \\ E \vdash t_1 : \mathcal{R}_1 \qquad \mathsf{x} \notin \operatorname{dom} E & & & & E, \mathsf{x} : \mathcal{R}_1 \vdash t_2 : \mathcal{R}_2 \end{array}}{\begin{array}{c} E \vdash \mathsf{let}\,\mathsf{x} = t_1\,\mathsf{in}\,t_2 : \quad \mathcal{R}? \\ \sigma \in [\![E]\!] \quad (\mathsf{let}\,\mathsf{x} = t_1\,\mathsf{in}\,t_2) \cdot \sigma \rightsquigarrow^\star v_2 \quad (\sigma, v_2) \in \mathcal{R} \end{array}}$$

Let-bindings

Let's write $E \vdash t : \mathcal{R}$ for $[\![t]\!]_E = \mathcal{R}$

We would like to mimic a typing rule for let-bindings:

$$\frac{E \vdash t_1 : \mathcal{R}_1 \qquad \mathsf{x} \notin \operatorname{dom} E \qquad E, \mathsf{x} : \mathcal{R}_1 \vdash t_2 : \mathcal{R}_2}{E \vdash \operatorname{let} \mathsf{x} = t_1 \operatorname{in} t_2 : \quad \mathcal{R}?}$$

$$(\sigma, v_1) \in \mathcal{R}_1 \qquad (\sigma\,[\mathsf{x} \mapsto v_1], v_2) \in \mathcal{R}_2$$

$$(\sigma, v_2) \in \mathcal{R}$$

$$\mathcal{R} \quad = \quad \{(\sigma, v_2) \mid \exists v_1, (\sigma, v_1) \in \mathcal{R}_1 \wedge (\sigma\,[\mathsf{x} \mapsto v_1], v_2) \in \mathcal{R}_2\}$$

Let-bindings

Let's write $E \vdash t : \mathcal{R}$ for $[\![t]\!]_E = \mathcal{R}$

We would like to mimic a typing rule for let-bindings:

$$\frac{E \vdash t_1 : \mathcal{R}_1 \qquad \mathsf{x} \notin \mathsf{dom}\, E \qquad E, \mathsf{x} : \mathcal{R}_1 \vdash t_2 : \mathcal{R}_2}{E \vdash \mathsf{let}\, \mathsf{x} = t_1 \,\mathsf{in}\, t_2 : \quad \mathcal{R}?}$$

$(\sigma, v_1) \in \mathcal{R}_1$ $\qquad$ $(\sigma\,[\mathsf{x} \mapsto v_1], v_2) \in \mathcal{R}_2$

$$(\sigma, v_2) \in \mathcal{R}$$

$$\begin{aligned} \mathcal{R} &= \{(\sigma, v_2) \mid \exists v_1, (\sigma, v_1) \in \mathcal{R}_1 \wedge \\ &\qquad (\sigma\,[\mathsf{x} \mapsto v_1], v_2) \in \mathcal{R}_2\} \\ &\triangleq \mathsf{LET}\ \mathsf{x} \leftarrow \mathcal{R}_1\ \mathsf{IN}\ \mathcal{R}_2 \end{aligned}$$

Let-bindings

Let's write $E \vdash t : \mathcal{R}$ for $[\![t]\!]_E = \mathcal{R}$

We would like to mimic a typing rule for let-bindings:

$$\frac{E \vdash t_1 : \mathcal{R}_1 \qquad \mathsf{x} \notin \mathsf{dom}\, E \qquad E, \mathsf{x} : \mathcal{R}_1 \vdash t_2 : \mathcal{R}_2}{E \vdash \mathsf{let}\, \mathsf{x} = t_1 \,\mathsf{in}\, t_2 : \mathsf{LET}\ \mathsf{x} \leftarrow \mathcal{R}_1\ \mathsf{IN}\ \mathcal{R}_2}$$

$$\mathsf{LET}\ \mathsf{x} \leftarrow \mathcal{R}_1\ \mathsf{IN}\ \mathcal{R}_2 \triangleq \{(\sigma, v_2) \mid \exists v_1, (\sigma, v_1) \in \mathcal{R}_1 \wedge (\sigma\,[\mathsf{x} \mapsto v_1], v_2) \in \mathcal{R}_2\}$$

Let-bindings

Let's write $E \vdash t : \mathcal{R}$ for $[\![t]\!]_E = \mathcal{R}$

We would like to mimic a typing rule for let-bindings:

$$\frac{E \vdash t_1 : \mathcal{R}_1 \qquad \mathsf{x} \notin \operatorname{dom} E \qquad E, \mathsf{x} : \mathcal{R}_1 \vdash t_2 : \mathcal{R}_2}{E \vdash \operatorname{let} \mathsf{x} = t_1 \operatorname{in} t_2 : \mathsf{LET}\ \mathsf{x} \leftarrow \mathcal{R}_1\ \mathsf{IN}\ \mathcal{R}_2}$$

$$\mathsf{LET}\ \mathsf{x} \leftarrow \mathcal{R}_1\ \mathsf{IN}\ \mathcal{R}_2 \triangleq \{(\sigma, v_2) \mid \exists v_1, (\sigma, v_1) \in \mathcal{R}_1 \wedge (\sigma\,[\mathsf{x} \mapsto v_1], v_2) \in \mathcal{R}_2\}$$

Definition

$$[\![\operatorname{let} \mathsf{x} = t_1 \operatorname{in} t_2]\!]_E \triangleq \mathsf{LET}\ \mathsf{x} \leftarrow [\![t_1]\!]_E\ \mathsf{IN}\ [\![t_2]\!]_{E, \mathsf{x}:[\![t_1]\!]_E} \quad \text{for } \mathsf{x} \notin \operatorname{dom} E$$

Let-bindings

A necessary adjustment for α-equivalence:
Makes the definition independent from the choice of the name x

Murdoch Gabbay and Andrew M. Pitts (1999). 'A New Approach to Abstract Syntax Involving Binders'. In: *Proceedings. 14th Symposium on Logic in Computer Science (Cat. No. PR00158)*. IEEE Comput. Soc, pp. 214–224

$$\mathsf{LET}^{\mathcal{S}}\ \mathsf{x} \leftarrow \mathcal{R}_1\ \mathsf{IN}\ \mathcal{R}_2 \triangleq \bigcup_{\mathsf{y} \notin \mathcal{S}} [\mathsf{y} \leftrightarrow \mathsf{x}] \cdot \{(\sigma, v_2) \mid \exists v_1, (\sigma, v_1) \in \mathcal{R}_1 \wedge (\sigma\,[\mathsf{x} \mapsto v_1], v_2) \in \mathcal{R}_2\}$$

Definition

$$[\![\mathsf{let}\,\mathsf{x} = t_1\,\mathsf{in}\,t_2]\!]_E \triangleq \mathsf{LET}^{\mathsf{dom}\,E}\ \mathsf{x} \leftarrow [\![t_1]\!]_E\ \mathsf{IN}\ [\![t_2]\!]_{E,\mathsf{x}:[\![t_1]\!]_E} \quad \text{for } \mathsf{x} \notin \mathsf{dom}\,E$$

Variables and Stable Relations

The set of input substitutions $[\![E]\!]$ is defined such that:

$$\frac{\sigma \in [\![E]\!] \qquad (\sigma, v) \in \mathcal{R} \qquad \sigma \sqsubseteq \sigma\,[\mathsf{x} \mapsto v]}{\sigma\,[\mathsf{x} \mapsto v] \in [\![E, \mathsf{x} : \mathcal{R}]\!]}$$

Variables and Stable Relations

The set of input substitutions $[\![E]\!]$ is defined such that:

$$\frac{\sigma \in [\![E]\!] \qquad (\sigma, v) \in \mathcal{R} \qquad \sigma \sqsubseteq \sigma\,[\mathsf{x} \mapsto v]}{\sigma\,[\mathsf{x} \mapsto v] \in [\![E, \mathsf{x} : \mathcal{R}]\!]}$$

What is the denotation for variables? Is $[\![\mathsf{x}]\!]_E = E(\mathsf{x})$ sound?

Variables and Stable Relations

The set of input substitutions $[\![E]\!]$ is defined such that:

$$\frac{\sigma \in [\![E]\!] \qquad (\sigma, v) \in \mathcal{R} \qquad \sigma \sqsubseteq \sigma\,[\mathsf{x} \mapsto v]}{\sigma\,[\mathsf{x} \mapsto v] \in [\![E, \mathsf{x} : \mathcal{R}]\!]}$$

What is the denotation for variables? Is $[\![\mathsf{x}]\!]_E = E(\mathsf{x})$ sound?

Proof sketch of soundness:

- Assume $\sigma \in [\![E]\!]$. We want to show $(\sigma, \sigma(\mathsf{x})) \in E(\mathsf{x})$.

Variables and Stable Relations

The set of input substitutions $[\![E]\!]$ is defined such that:

$$\frac{\sigma \in [\![E]\!] \qquad (\sigma, v) \in \mathcal{R} \qquad \sigma \sqsubseteq \sigma\,[\mathsf{x} \mapsto v]}{\sigma\,[\mathsf{x} \mapsto v] \in [\![E, \mathsf{x} : \mathcal{R}]\!]}$$

What is the denotation for variables? Is $[\![\mathsf{x}]\!]_E = E(\mathsf{x})$ sound?

Proof sketch of soundness:

- Assume $\sigma \in [\![E]\!]$. We want to show $(\sigma, \sigma(\mathsf{x})) \in E(\mathsf{x})$.
- We have $E = E_1, \mathsf{x} : E(\mathsf{x}), E_2$

Variables and Stable Relations

The set of input substitutions $[\![E]\!]$ is defined such that:

$$\frac{\sigma \in [\![E]\!] \qquad (\sigma, v) \in \mathcal{R} \qquad \sigma \sqsubseteq \sigma\,[\mathsf{x} \mapsto v]}{\sigma\,[\mathsf{x} \mapsto v] \in [\![E, \mathsf{x} : \mathcal{R}]\!]}$$

What is the denotation for variables? Is $[\![\mathsf{x}]\!]_E = E(\mathsf{x})$ sound?

Proof sketch of soundness:

- Assume $\sigma \in [\![E]\!]$. We want to show $(\sigma, \sigma(\mathsf{x})) \in E(\mathsf{x})$.
- We have $E = E_1, \mathsf{x} : E(\mathsf{x}), E_2$
- We have $(\sigma_1, \sigma_1(\mathsf{x})) \in E(\mathsf{x})$ for some $\sigma_1 \in [\![E_1]\!]$ such that $\sigma_1 \sqsubseteq \sigma$

Variables and Stable Relations

The set of input substitutions $[\![E]\!]$ is defined such that:

$$\frac{\sigma \in [\![E]\!] \qquad (\sigma, v) \in \mathcal{R} \qquad \sigma \sqsubseteq \sigma\,[\mathsf{x} \mapsto v]}{\sigma\,[\mathsf{x} \mapsto v] \in [\![E, \mathsf{x} : \mathcal{R}]\!]}$$

What is the denotation for variables? Is $[\![\mathsf{x}]\!]_E = E(\mathsf{x})$ sound?

Proof sketch of soundness:

- Assume $\sigma \in [\![E]\!]$. We want to show $(\sigma, \sigma(\mathsf{x})) \in E(\mathsf{x})$.
- We have $E = E_1, \mathsf{x} : E(\mathsf{x}), E_2$
- We have $(\sigma_1, \sigma_1(\mathsf{x})) \in E(\mathsf{x})$ for some $\sigma_1 \in [\![E_1]\!]$ such that $\sigma_1 \sqsubseteq \sigma$
- Since $\sigma(\mathsf{x}) = \sigma_1(\mathsf{x})$, we have $(\sigma_1, \sigma(\mathsf{x})) \in E(\mathsf{x})$

Variables and Stable Relations

The set of input substitutions $[\![E]\!]$ is defined such that:

$$\frac{\sigma \in [\![E]\!] \qquad (\sigma, v) \in \mathcal{R} \qquad \sigma \sqsubseteq \sigma\,[\mathsf{x} \mapsto v]}{\sigma\,[\mathsf{x} \mapsto v] \in [\![E, \mathsf{x} : \mathcal{R}]\!]}$$

What is the denotation for variables? Is $[\![\mathsf{x}]\!]_E = E(\mathsf{x})$ sound?

Proof sketch of soundness:

- Assume $\sigma \in [\![E]\!]$. We want to show $(\sigma, \sigma(\mathsf{x})) \in E(\mathsf{x})$.
- We have $E = E_1, \mathsf{x} : E(\mathsf{x}), E_2$
- We have $(\sigma_1, \sigma_1(\mathsf{x})) \in E(\mathsf{x})$ for some $\sigma_1 \in [\![E_1]\!]$ such that $\sigma_1 \sqsubseteq \sigma$
- Since $\sigma(\mathsf{x}) = \sigma_1(\mathsf{x})$, we have $(\sigma_1, \sigma(\mathsf{x})) \in E(\mathsf{x})$
- Because $E(\mathsf{x})$ is stable, we get $(\sigma, \sigma(\mathsf{x})) \in E(\mathsf{x})$

Variables and Stable Relations

The set of input substitutions $[\![E]\!]$ is defined such that:

$$\frac{\sigma \in [\![E]\!] \qquad (\sigma, v) \in \mathcal{R} \qquad \sigma \sqsubseteq \sigma\,[\mathsf{x} \mapsto v]}{}$$

Wha

Proc

- We have $(\sigma_1, \sigma_1(\mathsf{x})) \in E(\mathsf{x})$ for some $\sigma_1 \in [\![E_1]\!]$ such that $\sigma_1 \sqsubseteq \sigma$
- Since $\sigma(\mathsf{x}) = \sigma_1(\mathsf{x})$, we have $(\sigma_1, \sigma(\mathsf{x})) \in E(\mathsf{x})$
- Because $E(\mathsf{x})$ is stable, we get $(\sigma, \sigma(\mathsf{x})) \in E(\mathsf{x})$

Stability is essential for soundness!

If E contains only stable relations,
then $[\![t]\!]_E$ is a stable relation too.

Variables and Completeness

The rule for variables "$[\![\mathsf{x}]\!]_E = E(\mathsf{x})$" is very imprecise. For example:

$$[\![\mathsf{x}]\!]_{\mathsf{x}:\top,\mathsf{y}:\{(\sigma,v)|\sigma(\mathsf{x})<0\}} = \top = \Sigma_V \times \mathcal{V}$$

We have lost two pieces of information:

- The admissible inputs σ are such that $\sigma(\mathsf{x}) < 0$
- The returned value is the value that was assigned to x

Variables and Completeness

The rule for variables "$[\![\mathsf{x}]\!]_E = E(\mathsf{x})$" is very imprecise. For example:

$$[\![\mathsf{x}]\!]_{\mathsf{x}:\top,\mathsf{y}:\{(\sigma,v)|\sigma(\mathsf{x})<0\}} = \top = \Sigma_V \times \mathcal{V}$$

We have lost two pieces of information:

- The admissible inputs σ are such that $\sigma(\mathsf{x}) < 0$
- The returned value is the value that was assigned to x

Definition

$$[\![\mathsf{x}]\!]_E \triangleq \text{GATHER}(E) \cap \text{SELF}(\mathsf{x}) \qquad (\subseteq E(\mathsf{x}))$$

Variables and Completeness

The rule for variables "$[\![\mathsf{x}]\!]_E = E(\mathsf{x})$" is very imprecise. For example:

$$[\![\mathsf{x}]\!]_{\mathsf{x}:\top,\mathsf{y}:\{(\sigma,v)\mid\sigma(\mathsf{x})<0\}} = \top = \Sigma_v \times \mathcal{V}$$

We have lost two pieces of information:

- The admissible inputs σ are such that $\sigma(\mathsf{x}) < 0$
- The returned value is the value that was assigned to x

Definition

$$[\![\mathsf{x}]\!]_E \triangleq \mathrm{GATHER}(E) \cap \mathrm{SELF}(\mathsf{x}) \qquad (\subseteq E(\mathsf{x}))$$

$\bigcap_{\mathsf{y}\in\mathsf{dom}\,E}\{(\sigma, v) \mid (\sigma, \sigma(\mathsf{y})) \in E(\mathsf{y}))\}$
"These are the inputs accepted by E"

Variables and Completeness

The rule for variables "$[\![\mathsf{x}]\!]_E = E(\mathsf{x})$" is very imprecise. For example:

$$[\![\mathsf{x}]\!]_{\mathsf{x}:\top,\mathsf{y}:\{(\sigma,v)\mid\sigma(\mathsf{x})<0\}} = \top = \Sigma_V \times \mathcal{V}$$

We have lost two pieces of information:

- The admissible inputs σ are such that $\sigma(\mathsf{x}) < 0$
- The returned value is the value that was assigned to x

Definition

$$[\![\mathsf{x}]\!]_E \triangleq \mathrm{GATHER}(E) \cap \mathrm{SELF}(\mathsf{x}) \qquad (\subseteq E(\mathsf{x}))$$

$\bigcap_{\mathsf{y}\in\mathsf{dom}\,E}\{(\sigma, v) \mid (\sigma, \sigma(\mathsf{y})) \in E(\mathsf{y}))\}$
"These are the inputs accepted by E"

$\{(\sigma, v) \mid v = \sigma(\mathsf{x})\}$
"This is the value given to x"

Denotation for Functions

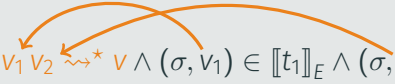
Definition

$$\begin{aligned}\llbracket t_1\, t_2 \rrbracket_E &= \mathrm{APP}(\llbracket t_1 \rrbracket_E, \llbracket t_2 \rrbracket_E) \\ &= \{(\sigma, v) \mid \exists v_1, \exists v_2, v_1\, v_2 \rightsquigarrow^\star v \wedge (\sigma, v_1) \in \llbracket t_1 \rrbracket_E \wedge (\sigma, v_2) \in \llbracket t_2 \rrbracket_E\}\end{aligned}$$

Denotation for Functions

Definition

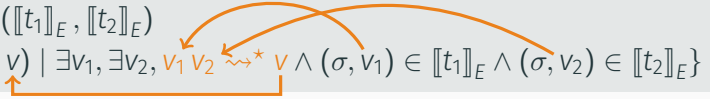
$$[\![t_1\, t_2]\!]_E = \mathrm{APP}([\![t_1]\!]_E, [\![t_2]\!]_E)$$
$$= \{(\sigma, v) \mid \exists v_1, \exists v_2, v_1\, v_2 \rightsquigarrow^\star v \wedge (\sigma, v_1) \in [\![t_1]\!]_E \wedge (\sigma, v_2) \in [\![t_2]\!]_E\}$$



Denotation for Functions

Definition

$$\begin{aligned}[\![t_1\, t_2]\!]_E &= \mathrm{APP}([\![t_1]\!]_E, [\![t_2]\!]_E) \\ &= \{(\sigma, v) \mid \exists v_1, \exists v_2, v_1\, v_2 \rightsquigarrow^\star v \wedge (\sigma, v_1) \in [\![t_1]\!]_E \wedge (\sigma, v_2) \in [\![t_2]\!]_E\}\end{aligned}$$



Denotation for Functions

Definition

$$\begin{aligned}[\![t_1\, t_2]\!]_E &= \mathrm{APP}([\![t_1]\!]_E, [\![t_2]\!]_E) \\ &= \{(\sigma, v) \mid \exists v_1, \exists v_2, v_1\, v_2 \rightsquigarrow^\star v \wedge (\sigma, v_1) \in [\![t_1]\!]_E \wedge (\sigma, v_2) \in [\![t_2]\!]_E\}\end{aligned}$$

Definition

$$[\![\lambda \mathrm{x}.\, t]\!]_E = \mathrm{GATHER}(E) \cap \mathrm{SELF}(\lambda \mathrm{x}.\, t)$$

$\{(\sigma, v) \mid v = \lambda \mathrm{x}.\, (t \cdot \sigma)\}$

Denotation for Functions

Definition

$$\begin{aligned}\llbracket t_1\, t_2 \rrbracket_E &= \mathrm{APP}(\llbracket t_1 \rrbracket_E, \llbracket t_2 \rrbracket_E) \\ &= \{(\sigma, v) \mid \exists v_1, \exists v_2, v_1\, v_2 \rightsquigarrow^\star v \wedge (\sigma, v_1) \in \llbracket t_1 \rrbracket_E \wedge (\sigma, v_2) \in \llbracket t_2 \rrbracket_E\}\end{aligned}$$

Definition

$$\begin{aligned}\llbracket \lambda \mathsf{x}.\, t \rrbracket_E &= \mathrm{GATHER}(E) \cap \mathrm{SELF}(\lambda \mathsf{x}.\, t) \\ &\subseteq (\mathsf{x} : \mathcal{R}) \to^{\mathrm{dom}\, E} \llbracket t \rrbracket_{E, \mathsf{x}:\mathcal{R}}\end{aligned}$$

Denotation for Functions

Definition

$$\begin{aligned}\llbracket t_1\, t_2 \rrbracket_E &= \mathrm{APP}(\llbracket t_1 \rrbracket_E, \llbracket t_2 \rrbracket_E) \\ &= \{(\sigma, v) \mid \exists v_1, \exists v_2, v_1\, v_2 \rightsquigarrow^\star v \wedge (\sigma, v_1) \in \llbracket t_1 \rrbracket_E \wedge (\sigma, v_2) \in \llbracket t_2 \rrbracket_E\}\end{aligned}$$

Definition

$$\begin{aligned}\llbracket \lambda \mathsf{x}.\, t \rrbracket_E &= \mathrm{GATHER}(E) \cap \mathrm{SELF}(\lambda \mathsf{x}.\, t) \\ &\subseteq (\mathsf{x} : \mathcal{R}) \to^{\mathrm{dom}\, E} \llbracket t \rrbracket_{E, \mathsf{x}:\mathcal{R}} \\ &= \left\{ (\sigma, v) \;\middle|\; \begin{array}{l} \forall v_1, v_2,\ v\, v_1 \rightsquigarrow^\star v_2 \Rightarrow \\ \forall \sigma',\ \sigma \sqsubseteq \sigma' \Rightarrow \\ \quad (\sigma', v_1) \in \mathcal{R} \Rightarrow (\sigma' [\mathsf{x} \mapsto v_1], v_2) \in \llbracket t \rrbracket_{E, \mathsf{x}:\mathcal{R}} \end{array} \right\}\end{aligned}$$

(nominal adjustments omitted)

Denotation for Functions

Definition

$$\begin{aligned}\llbracket t_1\, t_2 \rrbracket_E &= \mathrm{APP}(\llbracket t_1 \rrbracket_E, \llbracket t_2 \rrbracket_E) \\ &= \{(\sigma, v) \mid \exists v_1, \exists v_2, v_1\, v_2 \rightsquigarrow^\star v \wedge (\sigma, v_1) \in \llbracket t_1 \rrbracket_E \wedge (\sigma, v_2) \in \llbracket t_2 \rrbracket_E\}\end{aligned}$$

Definition

$$\begin{aligned}\llbracket \lambda \mathsf{x}.\, t \rrbracket_E &= \mathrm{GATHER}(E) \cap \mathrm{SELF}(\lambda \mathsf{x}.\, t) \\ &\subseteq (\mathsf{x} : \mathcal{R}) \to^{\mathrm{dom}\, E} \llbracket t \rrbracket_{E, \mathsf{x}:\mathcal{R}} \\ &= \left\{ (\sigma, v) \;\middle|\; \begin{array}{l} \forall v_1, v_2,\ v\, v_1 \rightsquigarrow^\star v_2 \Rightarrow \\ \forall \sigma',\ \sigma \sqsubseteq \sigma' \Rightarrow \\ \quad (\sigma', v_1) \in \mathcal{R} \Rightarrow (\sigma'\,[\mathsf{x} \mapsto v_1], v_2) \in \llbracket t \rrbracket_{E, \mathsf{x}:\mathcal{R}} \end{array} \right\}\end{aligned}$$

(nominal adjustments omitted)

Denotation for Functions

Definition

$$\begin{aligned}\llbracket t_1\, t_2 \rrbracket_E &= \mathrm{APP}(\llbracket t_1 \rrbracket_E, \llbracket t_2 \rrbracket_E) \\ &= \{(\sigma, v) \mid \exists v_1, \exists v_2, v_1\, v_2 \rightsquigarrow^\star v \wedge (\sigma, v_1) \in \llbracket t_1 \rrbracket_E \wedge (\sigma, v_2) \in \llbracket t_2 \rrbracket_E\}\end{aligned}$$

Definition

$$\begin{aligned}\llbracket \lambda \mathsf{x}.\, t \rrbracket_E &= \mathrm{GATHER}(E) \cap \mathrm{SELF}(\lambda \mathsf{x}.\, t) \\ &\subseteq (\mathsf{x} : \mathcal{R}) \to^{\mathrm{dom}\, E} \llbracket t \rrbracket_{E, \mathsf{x}:\mathcal{R}} \\ &= \left\{ (\sigma, v) \;\middle|\; \begin{array}{l} \forall v_1, v_2,\ v\, v_1 \rightsquigarrow^\star v_2 \Rightarrow \\ \forall \sigma',\ \sigma \sqsubseteq \sigma' \Rightarrow \\ \quad (\sigma', v_1) \in \mathcal{R} \Rightarrow (\sigma'[\mathsf{x} \mapsto v_1], v_2) \in \llbracket t \rrbracket_{E, \mathsf{x}:\mathcal{R}} \end{array} \right\}\end{aligned}$$

(nominal adjustments omitted)

Denotation for Functions

Definition

$$\begin{aligned}\llbracket t_1\, t_2 \rrbracket_E &= \mathrm{APP}(\llbracket t_1 \rrbracket_E, \llbracket t_2 \rrbracket_E) \\ &= \{(\sigma, v) \mid \exists v_1, \exists v_2, v_1\, v_2 \rightsquigarrow^\star v \wedge (\sigma, v_1) \in \llbracket t_1 \rrbracket_E \wedge (\sigma, v_2) \in \llbracket t_2 \rrbracket_E\}\end{aligned}$$

Definition

$$\begin{aligned}\llbracket \lambda \mathsf{x}.\, t \rrbracket_E &= \mathrm{GATHER}(E) \cap \mathrm{SELF}(\lambda \mathsf{x}.\, t) \\ &\subseteq (\mathsf{x} : \mathcal{R}) \to^{\mathrm{dom}\, E} \llbracket t \rrbracket_{E, \mathsf{x}:\mathcal{R}} \\ &= \left\{ (\sigma, v) \;\middle|\; \begin{array}{l} \forall v_1, v_2,\ v\, v_1 \rightsquigarrow^\star v_2 \Rightarrow \\ \forall \sigma',\ \sigma \sqsubseteq \sigma' \Rightarrow \\ \quad (\sigma', v_1) \in \mathcal{R} \Rightarrow (\sigma'\,[\mathsf{x} \mapsto v_1], v_2) \in \llbracket t \rrbracket_{E, \mathsf{x}:\mathcal{R}} \end{array} \right\}\end{aligned}$$

(nominal adjustments omitted)

Denotation for Functions

Definition

$$\begin{aligned}\llbracket t_1\, t_2 \rrbracket_E &= \mathrm{APP}(\llbracket t_1 \rrbracket_E, \llbracket t_2 \rrbracket_E) \\ &= \{(\sigma, v) \mid \exists v_1, \exists v_2, v_1\, v_2 \rightsquigarrow^\star v \wedge (\sigma, v_1) \in \llbracket t_1 \rrbracket_E \wedge (\sigma, v_2) \in \llbracket t_2 \rrbracket_E\}\end{aligned}$$

Definition

$$\begin{aligned}\llbracket \lambda \mathsf{x}.\, t \rrbracket_E &= \mathrm{GATHER}(E) \cap \mathrm{SELF}(\lambda \mathsf{x}.\, t) \\ &\subseteq (\mathsf{x} : \mathcal{R}) \to^{\mathrm{dom}\, E} \llbracket t \rrbracket_{E, \mathsf{x}:\mathcal{R}} \\ &= \left\{ (\sigma, v) \;\middle|\; \begin{array}{l} \forall v_1, v_2,\ v\, v_1 \rightsquigarrow^\star v_2 \Rightarrow \\ \forall \sigma',\ \sigma \sqsubseteq \sigma' \Rightarrow \\ \quad (\sigma', v_1) \in \mathcal{R} \Rightarrow (\sigma'\,[\mathsf{x} \mapsto v_1], v_2) \in \llbracket t \rrbracket_{E, \mathsf{x}:\mathcal{R}} \end{array} \right\}\end{aligned}$$

Makes the relation stable!

(nominal adjustments omitted)

Denotation for Functions

Definition

$$\begin{aligned}\llbracket t_1\, t_2 \rrbracket_E &= \mathrm{APP}(\llbracket t_1 \rrbracket_E, \llbracket t_2 \rrbracket_E) \\ &= \{(\sigma, v) \mid \exists v_1, \exists v_2, v_1\, v_2 \rightsquigarrow^\star v \wedge (\sigma, v_1) \in \llbracket t_1 \rrbracket_E \wedge (\sigma, v_2) \in \llbracket t_2 \rrbracket_E\}\end{aligned}$$

Definition

$$\begin{aligned}\llbracket \lambda \mathsf{x}.\, t \rrbracket_E &= \mathrm{GATHER}(E) \cap \mathrm{SELF}(\lambda \mathsf{x}.\, t) \\ &\subseteq (\mathsf{x} : \mathcal{R}) \to^{\mathrm{dom}\, E} \llbracket t \rrbracket_{E, \mathsf{x}:\mathcal{R}} \\ &= \left\{ (\sigma, v) \;\middle|\; \begin{array}{l} \forall v_1, v_2,\ v\, v_1 \rightsquigarrow^\star v_2 \Rightarrow \\ \forall \sigma',\ \sigma \sqsubseteq \sigma' \Rightarrow \\ \quad (\sigma', v_1) \in \mathcal{R} \Rightarrow (\sigma' [\mathsf{x} \mapsto v_1], v_2) \in \llbracket t \rrbracket_{E, \mathsf{x}:\mathcal{R}} \end{array} \right\}\end{aligned}$$

(nominal adjustments omitted)

The rule enables a modular analysis of functions!

Collecting Semantics: Summary

$$[\![\mathsf{x}]\!]_E = \mathrm{SELF}(\mathsf{x}) \cap \mathrm{GATHER}(E) \quad \text{if } \mathsf{x} \in \mathrm{dom}\, E$$

$$[\![\mathsf{let}\, \mathsf{x} = t_1 \,\mathsf{in}\, t_2]\!]_E = \mathrm{LET}^{\mathrm{dom}\, E}\ \mathsf{x} \leftarrow [\![t_1]\!]_E\ \ \mathsf{IN}\ \ [\![t_2]\!]_{E,\mathsf{x}:[\![t_1]\!]_E} \text{ with } \mathsf{x} \notin \mathrm{dom}\, E$$

$$[\![\lambda \mathsf{x}.\, t]\!]_E = ((\mathsf{x} : \top) \to^{\mathrm{dom}\, E} [\![t]\!]_{E,\mathsf{x}:\top}) \cap \mathrm{SELF}(\lambda \mathsf{x}.\, t) \cap \mathrm{GATHER}(E)$$

$$[\![t_1\, t_2]\!]_E = \mathrm{APP}([\![t_1]\!]_E, [\![t_2]\!]_E)$$

$$[\![()]\!]_E = \mathrm{UNIT}^{\mathsf{R}} \cap \mathrm{GATHER}(E)$$

$$[\![(t_1, t_2)]\!]_E = \mathrm{PAIR}^{\mathsf{R}}\,([\![t_1]\!]_E, [\![t_2]\!]_E)$$

$$[\![\pi_1\, t]\!]_E = [\![t]\!]_E\,;\mathrm{PAIR}^{\mathsf{L}}\,(\mathrm{EQ}, \top_{\mathsf{v}}) \qquad [\![\pi_2\, t]\!]_E = [\![t]\!]_E\,;\mathrm{PAIR}^{\mathsf{L}}\,(\top_{\mathsf{v}}, \mathrm{EQ})$$

$$[\![\mathrm{inj}_1\, t]\!]_E = \mathrm{SUM}^{\mathsf{R}}\,([\![t]\!]_E, \bot) \qquad [\![\mathrm{inj}_2\, t]\!]_E = \mathrm{SUM}^{\mathsf{R}}\,(\bot, [\![t]\!]_E)$$

$$\left[\!\!\left[\begin{array}{l}\mathsf{match}\ t\ \mathsf{with}\\ \mid\ \mathrm{inj}_1\, \mathsf{x}_1\ \to\ t_1\\ \mid\ \mathrm{inj}_2\, \mathsf{x}_2\ \to\ t_2\end{array}\right]\!\!\right]_E = \begin{array}{l}\mathrm{MATCH}^{\mathrm{dom}\, E}\ [\![t]\!]_E\ \ \mathsf{WITH}\\ \mid\ \mathsf{x}_1 \leftarrow [\![t_1]\!]_{E,\mathsf{x}_1:[\![t]\!]_E;\mathrm{SUM}^{\mathsf{L}}(\mathrm{EQ},\bot)} \text{ with } \mathsf{x}_1, \mathsf{x}_2 \notin \mathrm{dom}\, E\\ \mid\ \mathsf{x}_2 \leftarrow [\![t_2]\!]_{E,\mathsf{x}_2:[\![t]\!]_E;\mathrm{SUM}^{\mathsf{L}}(\bot,\mathrm{EQ})}\end{array}$$

Collecting Semantics: Summary

$$[\![\mathsf{x}]\!]_E = \mathrm{SELF}(\mathsf{x}) \cap \mathrm{GATHER}(E) \quad \text{if } \mathsf{x} \in \mathrm{dom}\, E$$

$$[\![\mathrm{let}\, \mathsf{x} = t_1 \,\mathrm{in}\, t_2]\!]_E = \mathrm{LET}^{\mathrm{dom}\, E}\ \mathsf{x} \leftarrow [\![t_1]\!]_E\ \ \mathrm{IN}\ \ [\![t_2]\!]_{E,\mathsf{x}:[\![t_1]\!]_E} \ \text{with } \mathsf{x} \notin \mathrm{dom}\, E$$

$$[\![\lambda \mathsf{x}.\, t]\!]_E = ((\mathsf{x} : \top) \rightarrow^{\mathrm{dom}\, E} [\![t]\!]_{E,\mathsf{x}:\top}) \cap \mathrm{SELF}(\lambda \mathsf{x}.\, t) \cap \mathrm{GATHER}(E)$$

$$[\![t_1\, t_2]\!]_E = \mathrm{APP}([\![t_1]\!]_E, [\![t_2]\!]_E)$$

$$[\![()]\!]_E = \mathrm{UNIT}^{\mathrm{R}} \cap \mathrm{GATHER}(E)$$

$$[\![(t_1, t_2)]\!]_E = \mathrm{PAIR}^{\mathrm{R}}\,([\![t_1]\!]_E, [\![t_2]\!]_E)$$

$$[\![\pi_1\, t]\!]_E = [\![t]\!]_E\,;\mathrm{PAIR}^{\mathrm{L}}\,(\mathrm{EQ}, \top_{\mathrm{v}}) \qquad [\![\pi_2\, t]\!]_E = [\![t]\!]_E\,;\mathrm{PAIR}^{\mathrm{L}}\,(\top_{\mathrm{v}}, \mathrm{EQ})$$

$$[\![\mathrm{inj}_1\, t]\!]_E = \mathrm{SUM}^{\mathrm{R}}\,([\![t]\!]_E, \bot) \qquad [\![\mathrm{inj}_2\, t]\!]_E = \mathrm{SUM}^{\mathrm{R}}\,(\bot, [\![t]\!]_E)$$

$$\left[\!\!\left[\begin{array}{l}\mathrm{match}\ t\ \mathrm{with}\\ \mid\ \mathrm{inj}_1\, \mathsf{x}_1\ \rightarrow\ t_1\\ \mid\ \mathrm{inj}_2\, \mathsf{x}_2\ \rightarrow\ t_2\end{array}\right]\!\!\right]_E = \begin{array}{l}\mathrm{MATCH}^{\mathrm{dom}\, E}\ [\![t]\!]_E\ \ \mathrm{WITH}\\ \mid\ \mathsf{x}_1 \leftarrow [\![t_1]\!]_{E,\mathsf{x}_1:[\![t]\!]_E;\mathrm{SUM}^{\mathrm{L}}(\mathrm{EQ},\bot)} \ \text{with } \mathsf{x}_1, \mathsf{x}_2 \notin \mathrm{dom}\, E\\ \mid\ \mathsf{x}_2 \leftarrow [\![t_2]\!]_{E,\mathsf{x}_2:[\![t]\!]_E;\mathrm{SUM}^{\mathrm{L}}(\bot,\mathrm{EQ})}\end{array}$$

Collecting Semantics: Summary

$$[\![\lambda \mathsf{x}.\, t]\!]_E = ((\mathsf{x} : \top) \to^{\mathsf{dom}\, E} [\![t]\!]_{E,\mathsf{x}:\top}) \cap \mathrm{SELF}(\lambda \mathsf{x}.\, t) \cap \mathrm{GATHER}(E)$$

Collecting Semantics: Summary

Information on the context (environment of the closure)

$$[\![\lambda x.\, t]\!]_E = ((x : \top) \to^{\mathrm{dom}\, E} [\![t]\!]_{E,x:\top}) \cap \mathrm{SELF}(\lambda x.\, t) \cap \mathrm{GATHER}(E)$$

Collecting Semantics: Summary

Information on the context
(environment of the closure)

$$[\![\lambda \mathsf{x}.\, t]\!]_E = ((\mathsf{x} : \top) \to^{\mathsf{dom}\, E} [\![t]\!]_{E,\mathsf{x}:\top}) \cap \mathrm{SELF}(\lambda \mathsf{x}.\, t) \cap \mathrm{GATHER}(E)$$

This is the *exact* code
☞ Whole program analysis
(code re-analyzed at every call site)

Collecting Semantics: Summary

Information on the context
(environment of the closure)

$$[\![\lambda x.\, t]\!]_E = ((x : \top) \to^{\mathsf{dom}\, E} [\![t]\!]_{E,x:\top}) \cap \mathrm{SELF}(\lambda x.\, t) \cap \mathrm{GATHER}(E)$$

This is the *exact* code
☞ Whole program analysis
(code re-analyzed at every call site)

Extensional behaviour
☞ Summary for modular analysis
(code analyzed only once)

Collecting Semantics: Summary

Information on the context
(environment of the closure)

$$[\![\lambda x.\, t]\!]_E = ((x : \top) \to^{\mathsf{dom}\, E} [\![t]\!]_{E,x:\top}) \cap \mathrm{SELF}(\lambda x.\, t) \cap \mathrm{GATHER}(E)$$

This is the *exact* code
☞ Whole program analysis
(code re-analyzed at every call site)

Extensional behaviour
☞ Summary for modular analysis
(code analyzed only once)

☞ Modular context-sensitive analysis!

Abstracting the Collecting Semantics

Abstractions Steps

$[\![t]\!]_E$ is a sound and complete collecting semantics:

It can serve as a safe and precise starting point for abstraction

Abstractions Steps

$[\![t]\!]_E$ is a sound and complete collecting semantics:

It can serve as a safe and precise starting point for abstraction

Abstraction strategy to obtain an effective analyser:

$[\![t]\!]_E$: Input-output relations $\mathcal{R}$

How are the inputs σ related to v, globally?

Abstractions Steps

$[\![t]\!]_E$ is a sound and complete collecting semantics:

It can serve as a safe and precise starting point for abstraction

Abstraction strategy to obtain an effective analyser:

$[\![t]\!]_E$: Input-output relations $\mathcal{R}$

Pointwise relations $\{\overline{\mathsf{x} \mapsto \mathcal{R}_v}\}$

How are the inputs σ related to v, globally?

How is each input $\sigma(\mathsf{x})$ related to v, independently?

Abstractions Steps

$[\![t]\!]_E$ is a sound and complete collecting semantics:
It can serve as a safe and precise starting point for abstraction

Abstraction strategy to obtain an effective analyser:

$[\![t]\!]_E$: Input-output relations $\mathcal{R}$

Pointwise relations $\{\overline{\mathsf{x} \mapsto \mathcal{R}_v}\}$

Pointwise correlations $\{\overline{\mathsf{x} \mapsto \mathcal{C}}\}$

How are the inputs σ related to v, globally?

How is each input $\sigma(\mathsf{x})$ related to v, independently?

Abstract domain to represent binary relations on higher-order values
☞ Function summaries!

Abstractions Steps

$[\![t]\!]_E$ is a sound and complete collecting semantics:
It can serve as a safe and precise starting point for abstraction

Abstraction strategy to obtain an effective analyser:

$[\![t]\!]_E$: Input-output relations $\mathcal{R}$

Pointwise relations $\{\overline{\mathsf{x} \mapsto \mathcal{R}_v}\}$

Pointwise correlations $\{\overline{\mathsf{x} \mapsto \mathcal{C}}\}$

A surprisingly precise analysis!

How are the inputs σ related to v, globally?

How is each input $\sigma(\mathsf{x})$ related to v, independently?

Abstract domain to represent binary relations on higher-order values
☞ Function summaries!

Abstraction Steps on an Example

```
let combine x y = (π1 x, π2 y)
```

Abstraction Steps on an Example

`let combine x y =` $(\pi_1\,\mathsf{x}, \pi_2\,\mathsf{y})$

For the body of **`combine`**, we have the collecting semantics:

$$[\![(\pi_1\,\mathsf{x}, \pi_2\,\mathsf{y})]\!]_{\mathsf{x}:\top,\mathsf{y}:\top} = \left\{ (\sigma, v) \;\middle|\; \begin{array}{l} v = (v_1, v_2') \wedge \\ \sigma(\mathsf{x}) = (v_1, v_2) \wedge \sigma(\mathsf{y}) = (v_1', v_2') \end{array} \right\}$$

Abstraction Steps on an Example

`let combine x y =` $(\pi_1\,\mathsf{x}, \pi_2\,\mathsf{y})$

For the body of `combine`, we have the collecting semantics:

$$[\![(\pi_1\,\mathsf{x}, \pi_2\,\mathsf{y})]\!]_{\mathsf{x}:\top,\mathsf{y}:\top} = \left\{ (\sigma, v) \;\middle|\; \begin{array}{l} v = (v_1, v_2') \wedge \\ \sigma(\mathsf{x}) = (v_1, v_2) \wedge \sigma(\mathsf{y}) = (v_1', v_2') \end{array} \right\}$$

After *pointwise* abstraction, we get:

$$\left\{ \begin{array}{lcl} \mathsf{x} & \mapsto & \{(v_\mathsf{x}, v) \mid v_\mathsf{x} = (v_1, v_2) \wedge v = (v_1, w)\} \\ \mathsf{y} & \mapsto & \{(v_\mathsf{y}, v) \mid v_\mathsf{y} = (v_1', v_2') \wedge v = (w, v_2')\} \end{array} \right\}$$

Abstraction Steps on an Example

`let combine x y =` $(\pi_1\,\mathsf{x}, \pi_2\,\mathsf{y})$

For the body of `combine`, we have the collecting semantics:

$$[\![(\pi_1\,\mathsf{x}, \pi_2\,\mathsf{y})]\!]_{\mathsf{x}:\top,\mathsf{y}:\top} = \left\{ (\sigma, v) \;\middle|\; \begin{array}{l} v = (v_1, v_2') \wedge \\ \sigma(\mathsf{x}) = (v_1, v_2) \wedge \sigma(\mathsf{y}) = (v_1', v_2') \end{array} \right\}$$

Relates the input $\sigma(\mathsf{x})$ with the result

After *pointwise* abstraction, we get:

$$\left\{ \begin{array}{lcl} \mathsf{x} & \mapsto & \{(v_\mathsf{x}, v) \mid v_\mathsf{x} = (v_1, v_2) \wedge v = (v_1, w)\} \\ \mathsf{y} & \mapsto & \{(v_\mathsf{y}, v) \mid v_\mathsf{y} = (v_1', v_2') \wedge v = (w, v_2')\} \end{array} \right\}$$

Abstraction Steps on an Example

`let combine x y =` $(\pi_1\, \mathsf{x}, \pi_2\, \mathsf{y})$

For the body of `combine`, we have the collecting semantics:

$$[\![(\pi_1\, \mathsf{x}, \pi_2\, \mathsf{y})]\!]_{\mathsf{x}:\top,\mathsf{y}:\top} = \left\{ (\sigma, v) \;\middle|\; \begin{array}{l} v = (v_1, v_2') \wedge \\ \sigma(\mathsf{x}) = (v_1, v_2) \wedge \sigma(\mathsf{y}) = (v_1', v_2') \end{array} \right\}$$

After *pointwise* abstraction, we get:

$$\left\{ \begin{array}{lcl} \mathsf{x} & \mapsto & \{(v_{\mathsf{x}}, v) \mid v_{\mathsf{x}} = (v_1, v_2) \wedge v = (v_1, w)\} \\ \mathsf{y} & \mapsto & \{(v_{\mathsf{y}}, v) \mid v_{\mathsf{y}} = (v_1', v_2') \wedge v = (w, v_2')\} \end{array} \right\}$$

Relates the input $\sigma(\mathsf{x})$ with the result

Relates the input $\sigma(\mathsf{y})$ with the result

Abstraction Steps on an Example

`let combine x y =` $(\pi_1\,\mathsf{x}, \pi_2\,\mathsf{y})$

For the body of `combine`, we have the collecting semantics:

$$[\![(\pi_1\,\mathsf{x}, \pi_2\,\mathsf{y})]\!]_{\mathsf{x}:\top,\mathsf{y}:\top} = \left\{ (\sigma, v) \;\middle|\; \begin{array}{l} v = (v_1, v_2') \wedge \\ \sigma(\mathsf{x}) = (v_1, v_2) \wedge \sigma(\mathsf{y}) = (v_1', v_2') \end{array} \right\}$$

After *pointwise* abstraction, we get:

$$\left\{ \begin{array}{lcl} \mathsf{x} & \mapsto & \{(v_\mathsf{x}, v) \mid v_\mathsf{x} = (v_1, v_2) \wedge v = (v_1, w)\} \\ \mathsf{y} & \mapsto & \{(v_\mathsf{y}, v) \mid v_\mathsf{y} = (v_1', v_2') \wedge v = (w, v_2')\} \end{array} \right\}$$

After *correlation* abstraction, we get:

$$\left\{ \begin{array}{lcl} \mathsf{x} & \mapsto & \mathrm{PAIR}^{\mathrm{R}}\left(\mathrm{PAIR}^{\mathrm{L}}\left(\mathrm{EQ}, \top\right), \top\right) \\ \mathsf{y} & \mapsto & \mathrm{PAIR}^{\mathrm{R}}\left(\top, \mathrm{PAIR}^{\mathrm{L}}\left(\top, \mathrm{EQ}\right)\right) \end{array} \right\}$$

A Modular Analysis

- OCaml implementation of a **bottom-up**, **modular** analysis
- **Functions analysed only once**: summaries $\approx$ extensional behaviour
- The analysis was not designed to compete with existing CFAs
  Modular, but still surprisingly precise!

A Modular Analysis

- OCaml implementation of a **bottom-up**, **modular** analysis
- **Functions analysed only once**: summaries $\approx$ extensional behaviour
- The analysis was not designed to compete with existing CFAs
  Modular, but still surprisingly precise!

A classic CFA example: 0-CFA is not precise, 1-CFA is precise

$$
\begin{array}{l}
\text{let } \mathsf{f} = \lambda \mathsf{x}.\, \mathsf{x} \text{ in} \\
\text{let } \mathsf{y}_1 = \mathsf{f}\,(\mathsf{inj}_1\,()) \text{ in} \\
\text{let } \mathsf{y}_2 = \mathsf{f}\,(\mathsf{inj}_2\,()) \text{ in} \\
\mathsf{y}_1
\end{array}
$$

A Modular Analysis

- OCaml implementation of a **bottom-up**, **modular** analysis
- **Functions analysed only once**: summaries $\approx$ extensional behaviour
- The analysis was not designed to compete with existing CFAs
  Modular, but still surprisingly precise!

A classic CFA example: 0-CFA is not precise, 1-CFA is precise

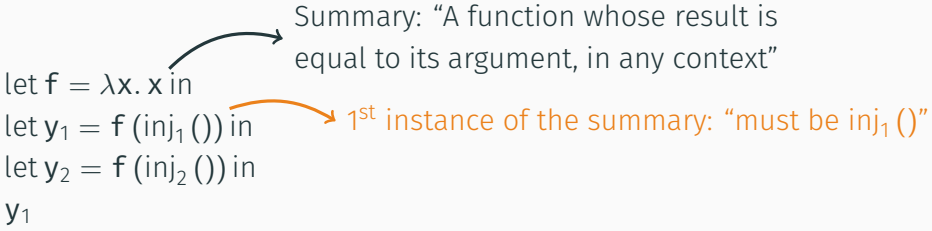
Summary: "A function whose result is equal to its argument, in any context"

$$
\begin{array}{l}
\text{let } \mathsf{f} = \lambda \mathsf{x}.\, \mathsf{x} \text{ in} \\
\text{let } \mathsf{y}_1 = \mathsf{f}\,(\mathsf{inj}_1\,()) \text{ in} \\
\text{let } \mathsf{y}_2 = \mathsf{f}\,(\mathsf{inj}_2\,()) \text{ in} \\
\mathsf{y}_1
\end{array}
$$

A Modular Analysis

- OCaml implementation of a **bottom-up**, **modular** analysis
- **Functions analysed only once**: summaries $\approx$ extensional behaviour
- The analysis was not designed to compete with existing CFAs
  Modular, but still surprisingly precise!

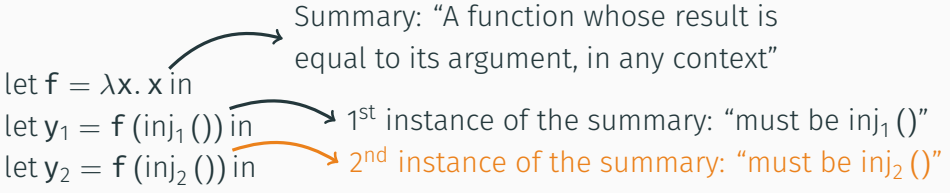
A classic CFA example: 0-CFA is not precise, 1-CFA is precise



A Modular Analysis

- OCaml implementation of a **bottom-up**, **modular** analysis
- **Functions analysed only once**: summaries $\approx$ extensional behaviour
- The analysis was not designed to compete with existing CFAs
  Modular, but still surprisingly precise!

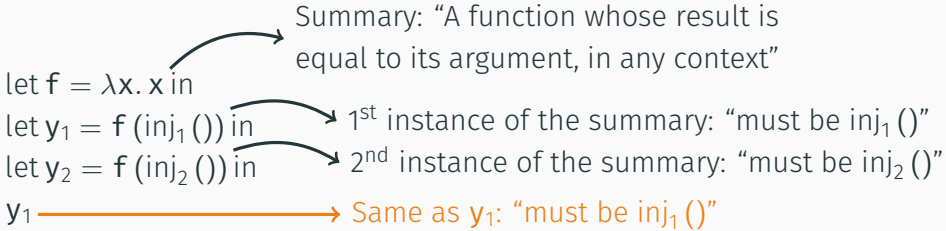
A classic CFA example: 0-CFA is not precise, 1-CFA is precise



A Modular Analysis

- OCaml implementation of a **bottom-up**, **modular** analysis
- **Functions analysed only once**: summaries $\approx$ extensional behaviour
- The analysis was not designed to compete with existing CFAs
  Modular, but still surprisingly precise!

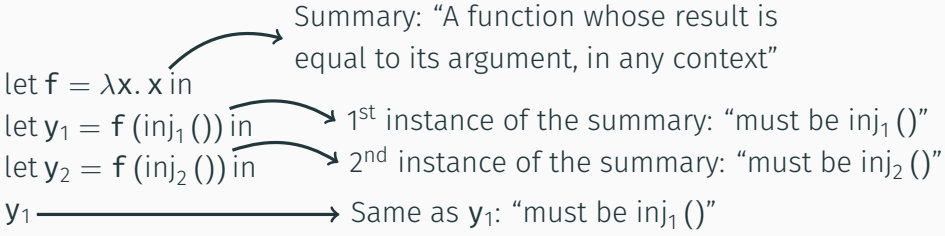
A classic CFA example: 0-CFA is not precise, 1-CFA is precise



A Modular Analysis

- OCaml implementation of a **bottom-up**, **modular** analysis
- **Functions analysed only once**: summaries ≈ extensional behaviour
- The analysis was not designed to compete with existing CFAs
  Modular, but still surprisingly precise!

A classic CFA example: 0-CFA is not precise, 1-CFA is precise



For this example: On par with 1-CFA!

Limitations of the Analysis

- Pointwise abstraction loses a lot of information
- In particular the abstraction of APP is not precise enough:
  Assuming $\mathsf{f} : \top, \mathsf{x} : \top$, the application $\mathsf{f}\ \mathsf{x}$ is abstracted by $\top$

Limitations of the Analysis

- Pointwise abstraction loses a lot of information
- In particular the abstraction of APP is not precise enough:
  Assuming $\mathsf{f} : \top, \mathsf{x} : \top$, the application $\mathsf{f}\ \mathsf{x}$ is abstracted by $\top$
- **The issue is the the application of unknown functions:**
  The function summaries lack precision in such cases

Limitations of the Analysis

- Pointwise abstraction loses a lot of information
- In particular the abstraction of APP is not precise enough:
  Assuming $\mathsf{f} : \top, \mathsf{x} : \top$, the application $\mathsf{f}\ \mathsf{x}$ is abstracted by $\top$
- **The issue is the the application of unknown functions:**
  The function summaries lack precision in such cases
- Remark: CFA gets more precise results by finding out
  which functions are actually applied

Limitations of the Analysis

- Pointwise abstraction loses a lot of information
- In particular the abstraction of APP is not precise enough:
  Assuming $\mathsf{f} : \top, \mathsf{x} : \top$, the application $\mathsf{f}\ \mathsf{x}$ is abstracted by $\top$
- **The issue is the the application of unknown functions:**
  The function summaries lack precision in such cases
- Remark: CFA gets more precise results by finding out
  which functions are actually applied
- A lot of room for improvement!

Conclusion

Conclusion and Future Work

Solid foundations for relational analysis for higher-order programs

- Sound and complete input-output relational semantics
- An example analysis obtained by successive abstractions
- Mechanised in Coq: artefact available!

`https://people.irisa.fr/Benoit.Montagu/papers/icfp2020_artifact.tar.bz2`

Conclusion and Future Work

Solid foundations for relational analysis for higher-order programs

- Sound and complete input-output relational semantics
- An example analysis obtained by successive abstractions
- Mechanised in Coq: artefact available!

`https://people.irisa.fr/Benoit.Montagu/papers/icfp2020_artifact.tar.bz2`

Future work:

- Long-term: integrate with proof assistants
- Support more language features
- Improve precision (e.g. with numeric abstract domains)
- Towards relational CFA

Pointwise Relations

$$\mathcal{M} \quad ::= \quad \bot^\natural \quad | \quad \{\overline{\mathsf{x} \mapsto \mathcal{R}_\mathsf{x}}^{\mathsf{x} \in \mathcal{S}}, \mathsf{this} \mapsto \mathcal{R}_\mathsf{this}\}$$

$$\gamma_\mathcal{S}(\bot^\natural) = \bot$$

$$\gamma_\mathcal{S}(\{\overline{\mathsf{x} \mapsto \mathcal{R}_\mathsf{x}}^{\mathsf{x} \in \mathcal{S}'}, \mathsf{this} \mapsto \mathcal{R}_\mathsf{this}\}) = \left\{ (\sigma, v) \;\middle|\; \begin{array}{l} \sigma \in \Sigma_v \wedge v \in \mathcal{V} \wedge \mathcal{S} \subseteq \mathsf{dom}\,\sigma \wedge \mathcal{S} = \mathcal{S}' \\ \wedge\, (\forall \mathsf{x} \in \mathcal{S}, (\sigma(\mathsf{x}), v) \in \mathcal{R}_\mathsf{x}) \\ \wedge\, (\forall v' \in \mathcal{V}, (v', v) \in \mathcal{R}_\mathsf{this}) \end{array} \right\}$$

$$\alpha_\mathcal{S}(\mathcal{R}) = \text{if } \mathcal{R} = \bot \text{ then } \bot^\natural \text{ else } \{\overline{\mathsf{x} \mapsto \mathcal{R}_\mathsf{x}}^{\mathsf{x} \in \mathcal{S}}, \mathsf{this} \mapsto \mathcal{R}_\mathsf{this}\}$$

$$\text{where } \mathcal{R}_\mathsf{x} = \bigcup_{\sigma \in \Sigma_v} \left\{ (\sigma(\mathsf{x}), v) \;\middle|\; \mathcal{S} \subseteq \mathsf{dom}\,\sigma \wedge (\sigma, v) \in \mathcal{R} \right\}$$

$$\text{and } \mathcal{R}_\mathsf{this} = \bigcup_{\sigma \in \Sigma_v} \left\{ (v', v) \;\middle|\; \mathcal{S} \subseteq \mathsf{dom}\,\sigma \wedge v' \in \mathcal{V} \wedge (\sigma, v) \in \mathcal{R} \right\}$$

Correlation Abstract Domain

$$
\begin{array}{rcl}
S & ::= & \mathsf{L} \mid \mathsf{R} \\
\\
\mathcal{C} & ::= & \bot^\sharp \mid \mathsf{EQ}^\sharp \mid \top^\sharp \\
& \mid & \mathsf{PAIR}^{\sharp S}(\mathcal{C}, \mathcal{C}) \\
& \mid & \mathsf{SUM}^{\sharp S}(\mathcal{C}, \mathcal{C}) \\
& \mid & \mathsf{FUN}^{\sharp S}(\mathcal{C}, \mathcal{C}, \mathcal{C})
\end{array}
$$

Correlation Abstract Domain

A typed concretization for correlations:

$$\gamma^\sharp_{\tau_1\otimes\tau_2}(\bot^\sharp) = \bot \qquad \gamma^\sharp_{\tau\otimes\tau}(\mathsf{EQ}^\sharp) = \mathsf{EQ}\cap\mathcal{V}_\tau\times\mathcal{V}_\tau \qquad \gamma^\sharp_{\tau_1\otimes\tau_2}(\top^\sharp) = \mathcal{V}_{\tau_1}\times\mathcal{V}_{\tau_2}$$

$$\begin{aligned}
\gamma^\sharp_{(\tau_1\times\tau_1')\otimes\tau_2}(\mathsf{PAIR}^{\sharp\mathsf{L}}(\mathcal{C},\mathcal{C}')) &= \mathsf{PAIR}^{\mathsf{L}}\left(\gamma^\sharp_{\tau_1\otimes\tau_2}(\mathcal{C}),\gamma^\sharp_{\tau_1'\otimes\tau_2}(\mathcal{C}')\right)\\
\gamma^\sharp_{\tau_1\otimes(\tau_2\times\tau_2')}(\mathsf{PAIR}^{\sharp\mathsf{R}}(\mathcal{C},\mathcal{C}')) &= \mathsf{PAIR}^{\mathsf{R}}\left(\gamma^\sharp_{\tau_1\otimes\tau_2}(\mathcal{C}),\gamma^\sharp_{\tau_1\otimes\tau_2'}(\mathcal{C}')\right)\\
\gamma^\sharp_{(\tau_1+\tau_1')\otimes\tau_2}(\mathsf{SUM}^{\sharp\mathsf{L}}(\mathcal{C},\mathcal{C}')) &= \mathsf{SUM}^{\mathsf{L}}\left(\gamma^\sharp_{\tau_1\otimes\tau_2}(\mathcal{C}),\gamma^\sharp_{\tau_1'\otimes\tau_2}(\mathcal{C}')\right)\\
\gamma^\sharp_{\tau_1\otimes(\tau_2+\tau_2')}(\mathsf{SUM}^{\sharp\mathsf{R}}(\mathcal{C},\mathcal{C}')) &= \mathsf{SUM}^{\mathsf{R}}\left(\gamma^\sharp_{\tau_1\otimes\tau_2}(\mathcal{C}),\gamma^\sharp_{\tau_1\otimes\tau_2'}(\mathcal{C}')\right)\\
\gamma^\sharp_{(\tau_1\to\tau_1')\otimes\tau_2}(\mathsf{FUN}^{\sharp\mathsf{L}}(\mathcal{C}_\mathsf{a},\mathcal{C}_\mathsf{i},\mathcal{C}_\mathsf{o})) &= \mathsf{FUN}^{\mathsf{L}}_{\mathsf{v}}\left(\gamma^\sharp_{\tau_1\otimes\tau_2}(\mathcal{C}_\mathsf{a}),\gamma^\sharp_{\tau_1\otimes\tau_1'}(\mathcal{C}_\mathsf{i}),\gamma^\sharp_{\tau_1'\otimes\tau_2}(\mathcal{C}_\mathsf{o})\right)\\
&\quad\cap(\mathcal{V}_{\tau_1\to\tau_1'}\times\mathcal{V}_{\tau_2})\\
\gamma^\sharp_{\tau_1\otimes(\tau_2\to\tau_2')}(\mathsf{FUN}^{\sharp\mathsf{R}}(\mathcal{C}_\mathsf{a},\mathcal{C}_\mathsf{i},\mathcal{C}_\mathsf{o})) &= \mathsf{FUN}^{\mathsf{R}}_{\mathsf{v}}\left(\gamma^\sharp_{\tau_1\otimes\tau_2}(\mathcal{C}_\mathsf{a}),\gamma^\sharp_{\tau_2\otimes\tau_2'}(\mathcal{C}_\mathsf{i}),\gamma^\sharp_{\tau_1\otimes\tau_2'}(\mathcal{C}_\mathsf{o})\right)\\
&\quad\cap(\mathcal{V}_{\tau_1}\times\mathcal{V}_{\tau_2\to\tau_2'})\\
\gamma^\sharp_{\tau_1\otimes\tau_2}(\mathcal{C}) &= \bot \text{ in all other cases}
\end{aligned}$$

More Relational Combinators for Binary Relations on Values

$$\mathrm{APP}_V(\mathcal{R}_1, \mathcal{R}_2) = \left\{ (v, v') \;\middle|\; \begin{array}{r} \exists v_1, v_2, (v, v_1) \in \mathcal{R}_1 \\ \wedge\, (v, v_2) \in \mathcal{R}_2 \;\wedge\; v_1\, v_2 \rightsquigarrow^\star v' \end{array} \right\}$$

$$\mathrm{FUN}_v^{\mathrm{R}}(\mathcal{R}_{\mathrm{arg}}, \mathcal{R}_{\mathrm{inner}}, \mathcal{R}_{\mathrm{outer}}) = \left\{ (v, v') \;\middle|\; \begin{array}{r} \forall v_1, v_2, v'\, v_1 \rightsquigarrow^\star v_2 \Rightarrow (v, v_1) \in \mathcal{R}_{\mathrm{arg}} \Rightarrow \\ (v_1, v_2) \in \mathcal{R}_{\mathrm{inner}} \wedge (v, v_2) \in \mathcal{R}_{\mathrm{outer}} \end{array} \right\}$$

Recursive Functions

$$t ::= \ldots \mid \mu \mathsf{f}.\, \lambda \mathsf{x}.\, t \qquad\qquad v ::= \ldots \mid \mu \mathsf{f}.\, \lambda \mathsf{x}.\, t$$

$$(\mu \mathsf{f}.\, \lambda \mathsf{x}.\, t)\, v \rightsquigarrow (\lambda \mathsf{x}.\, t)\, [\mathsf{f} \leftarrow \mu \mathsf{f}.\, t]\, v$$

Interpretation of fixpoints:

$$[\![\mu \mathsf{f}.\, \lambda \mathsf{x}.\, t]\!]_E = \mathcal{R}_{\text{base}} \cap \mathsf{lfp}\, f$$

where

$$\mathcal{R}_{\text{base}} = \mathsf{GATHER}(E) \cap \mathsf{SELF}(\mu \mathsf{f}.\, \lambda \mathsf{x}.\, t)$$

and where

$$\begin{aligned} f(\mathcal{R}) = \;& \mathcal{R}_{\text{base}} \cup \mathsf{FUN}^{\mathsf{dom}\, E}(\top, \lambda v.\mathsf{APP}(\mathcal{R}, \mathsf{SELF}(v))) \\ & \cup \mathsf{LET}^{\mathsf{dom}\, E}\ \mathsf{f} \leftarrow \mathcal{R}\ \mathsf{IN}\ (\!|\lambda \mathsf{x}.\, t|\!)_{E, \mathsf{f}:\mathcal{R}} \end{aligned}$$