An Abstract Interpretation Framework for Semantics and Diagnosis of Functional Logic Programs

Giovanni Bacci

supervisor: Marco Comini

Dipartimento di Matematica e Informatica
University of Udine

15 March 2012 , Udine

Motivations

Context: **lazy** declarative languages

- Purely Functional (**Haskell**)
- Functional Logic (**Curry**, **TOY**)

Goal: efficacious semantic-based program manipulation tools

- Static Analysis
- Abstract Diagnosis
- Synthesis of Specifications

We need a semantics which is (at the same time)

- fully-abstract w.r.t. I/O observations
- goal-independent
- "condensed" (as concise as possible)

no such semantics in literature

Motivations

Context: **lazy** declarative languages

- Purely Functional (**Haskell**)
- Functional Logic (**Curry**, **TOY**)

Goal: efficacious semantic-based program manipulation tools

- Static Analysis
- Abstract Diagnosis
- Synthesis of Specifications

We need a semantics which is (at the same time)

- fully-abstract w.r.t. I/O observations
- goal-independent
- "condensed" (as concise as possible)

no such semantics in literature

Motivations

Context: **lazy** declarative languages

- Purely Functional (**Haskell**)
- Functional Logic (**Curry**, **TOY**)

Goal: efficacious semantic-based program manipulation tools

- Static Analysis
- Abstract Diagnosis
- Synthesis of Specifications

We need a semantics which is (at the same time)

- fully-abstract w.r.t. I/O observations
- goal-independent
- "condensed" (as concise as possible)

no such semantics in literature

Motivations

Context: **lazy** declarative languages

+ Purely Functional (**Haskell**)
+ Functional Logic (**Curry**, **TOY**)

Goal: efficacious semantic-based program manipulation tools

+ Static Analysis
+ Abstract Diagnosis
+ Synthesis of Specifications

We need a semantics which is (at the same time)

+ fully-abstract w.r.t. I/O observations
+ goal-independent
+ "condensed" (as concise as possible)

no such semantics in literature

Functional Logic Paradigm

+ nested expressions
+ higher-order features
+ lazy evaluation

$\Big\}$ **functional paradigm** $\Big\}$ **FLP**

Operational mechanism:

REWRITING

sub-expressions are rewritten according to program rules

Functional Logic Paradigm

$$\left.\begin{array}{l} \left.\begin{array}{l} + \text{ nested expressions} \\ + \text{ higher-order features} \\ + \text{ lazy evaluation} \end{array}\right\} \textbf{functional paradigm} \\ \\ \left.\begin{array}{l} + \text{ logical variables} \\ + \text{ partial data structures} \\ + \text{ built-it search} \end{array}\right\} \textbf{logic paradigm} \end{array}\right\} \textbf{FLP}$$

Operational mechanism:

$$\begin{array}{c}\text{VARIABLE} \\ \text{INSTANTIATION}\end{array} + \text{REWRITING} = \text{NARROWING}$$

variables are instantiated so that sub-expressions can be rewritten according to program rules

Functional Logic Paradigm

$$\left.\begin{array}{l}\text{+ nested expressions}\\ \text{+ higher-order features}\\ \text{+ lazy evaluation}\end{array}\right\}\textbf{functional paradigm}$$

$$\left.\begin{array}{l}\text{+ logical variables}\\ \text{+ partial data structures}\\ \text{+ built-it search}\end{array}\right\}\textbf{logic paradigm}$$

FO-FLP

Operational mechanism:

$$\begin{array}{c}\text{VARIABLE}\\ \text{INSTANTIATION}\end{array} + \text{REWRITING} = \text{NARROWING}$$

variables are instantiated so that sub-expressions can be rewritten according to program rules

equation solving & built-in search:

```
0 + x = x                      0 <= y = True
(S x) + y = S (x + y)          (S x) <= 0 = False
double x = x + x               (S x) <= (S y) = x <= y
```

the goal `(x + x) <= 0 where x free` returns 2 solutions, namely `{x -> 0} True` and `{x -> S x'} False`

non-deterministic operations:

overlapping rules are allowed $\Longrightarrow$ **non-confluent programs**

```
coin = 0
coin = S 0
```

`coin` returns 2 solutions, namely `{} 0` and `{} S 0`

equation solving & built-in search:

```
0 + x = x                         0 <= y = True
(S x) + y = S (x + y)             (S x) <= 0 = False
double x = x + x                  (S x) <= (S y) = x <= y
```

the goal `(x + x) <= 0 where x free` returns 2 solutions, namely `{x -> 0} True` and `{x -> S x'} False`

non-deterministic operations:

overlapping rules are allowed $\Longrightarrow$ **non-confluent programs**

```
coin = 0
coin = S 0
```

`coin` returns 2 solutions, namely `{} 0` and `{} S 0`

Nondeterminism & lazy evaluation

lazy evaluation

delays the evaluation of sub-expressions until it is not demanded

A subtle aspect of nondeterministic operations is their treatment if they are passed as arguments:

```
coin = 0                    double x = x + x
coin = S 0
```

need-time choice: the choice for the desired value of an operation is made when it is demanded

double coin ⇒ coin + coin ⇒ 0 + coin ⇒ coin ⇒ S 0

call-time choice the choice for the desired value of a operation is made at call time (not the evaluation)

double coin ⇒ coin + coin ⇒ 0 + 0 ⇒ 0

sharing

Nondeterminism & lazy evaluation

lazy evaluation

delays the evaluation of sub-expressions until it is not demanded

A subtle aspect of nondeterministic operations is their treatment if they are passed as arguments:

```
coin = 0                    double x = x + x
coin = S 0
```

need-time choice: the choice for the desired value of an operation is made when it is demanded

double coin ⇒ coin + coin ⇒ 0 + coin ⇒ coin ⇒ S 0

call-time choice the choice for the desired value of a operation is made at call time (not the evaluation)

double coin ⇒ coin + coin ⇒ 0 + 0 ⇒ 0

sharing

Nondeterminism & lazy evaluation

lazy evaluation

delays the evaluation of sub-expressions until it is not demanded

A subtle aspect of nondeterministic operations is their treatment if they are passed as arguments:

```
coin = 0                double x = x + x
coin = S 0
```

need-time choice: the choice for the desired value of an operation is made when it is demanded

$\underline{\texttt{double coin}} \Rightarrow \underline{\texttt{coin}}\ \texttt{+ coin} \Rightarrow \underline{\texttt{0 + coin}} \Rightarrow \underline{\texttt{coin}} \Rightarrow \texttt{S 0}$

call-time choice the choice for the desired value of a operation is made at call time (not the evaluation)

$\underline{\texttt{double coin}} \Rightarrow \underline{\texttt{coin}}\ \texttt{+}\ \underline{\texttt{coin}} \Rightarrow \underline{\texttt{0 + 0}} \Rightarrow \texttt{0}$

sharing

Nondeterminism & lazy evaluation

lazy evaluation

delays the evaluation of sub-expressions until it is not demanded

A subtle aspect of nondeterministic operations is their treatment if they are passed as arguments:

```
coin = 0                    double x = x + x
coin = S 0
```

need-time choice: the choice for the desired value of an operation is made when it is demanded

double coin ⇒ coin + coin ⇒ 0 + coin ⇒ coin ⇒ S 0

call-time choice the choice for the desired value of a operation is made at call time (not the evaluation)

double coin ⇒ coin + coin ⇒ 0 + 0 ⇒ 0

sharing

A semantics adequate w.r.t. an observation ϕ

Requirements:

- fix-point characterization (i.e., $\mathcal{F}[\![P]\!] := \mathit{lfp}\,\mathcal{P}[\![P]\!]$)
- goal-independent & "condensed"
- fully-abstract w.r.t. a behavioral observation ϕ

Full-abstraction (EAGER languages):

- $\mathcal{F}[\![P_1]\!] = \mathcal{F}[\![P_2]\!] \iff \mathcal{B}^{\phi}[\![P_1]\!] = \mathcal{B}^{\phi}[\![P_2]\!]$

Full-abstraction (LAZY languages):

- $\mathcal{F}[\![P_1]\!] = \mathcal{F}[\![P_2]\!] \iff \forall Q \in \mathbb{UP}^{\Sigma'}_{\Sigma}.\ \mathcal{B}^{\phi}[\![P_1 \cup Q]\!] = \mathcal{B}^{\phi}[\![P_2 \cup Q]\!]$

A semantics adequate w.r.t. an observation ϕ

Requirements:

- fix-point characterization (i.e., $\mathcal{F}[\![P]\!] := \mathit{lfp}\,\mathcal{P}[\![P]\!]$)
- goal-independent & "condensed"
- fully-abstract w.r.t. a behavioral observation ϕ

Full-abstraction (EAGER languages):

- $\mathcal{F}[\![P_1]\!] = \mathcal{F}[\![P_2]\!] \iff \mathcal{B}^{\phi}[\![P_1]\!] = \mathcal{B}^{\phi}[\![P_2]\!]$

Full-abstraction (LAZY languages):

- $\mathcal{F}[\![P_1]\!] = \mathcal{F}[\![P_2]\!] \iff \forall Q \in \mathbb{UP}_{\Sigma}^{\Sigma'}.\ \mathcal{B}^{\phi}[\![P_1 \cup Q]\!] = \mathcal{B}^{\phi}[\![P_2 \cup Q]\!]$

using programs
can only define new operations

Computed Results Behavior

Computed result behavior of programs:

$$\mathcal{B}^{cr}[\![P]\!](e_0) := \left\{ (\sigma_1 \cdots \sigma_n)\restriction_{e_0} \cdot e_n \;\middle|\; e_0 \overset{\sigma_1}{\underset{p_1}{\Longrightarrow}} \cdots \overset{\sigma_n}{\underset{p_n}{\Longrightarrow}} e_n,\ e_n \in \mathcal{T}(\mathcal{C}, \mathcal{V}) \right\}$$

Problem: collecting computed results for every most general call leads to incorrect semantic denotations because of laziness

```
f x = S (g x)
g (S x) = 0
```

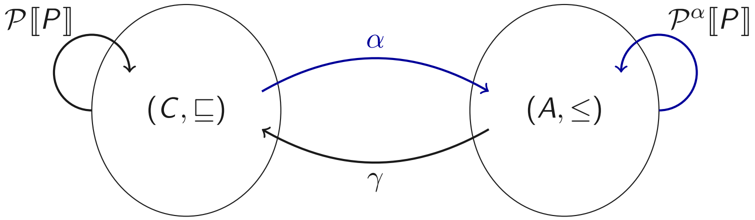
```
f (S x) = S 0
g (S x) = 0
```

$f(x)$ have the same computed results in both programs, namely

$$\mathcal{B}^{cr}[\![P]\!](f(x)) = \{\{x/s(x')\} \cdot s(0)\}$$

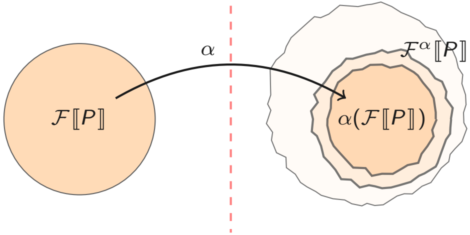
But for the goal $g(f(x))$ the former program computes $\varepsilon \cdot 0$ whereas the latter computes $\{x/s(x')\} \cdot 0$.

Systematic design of semantics by A.I. [Cousot 77]

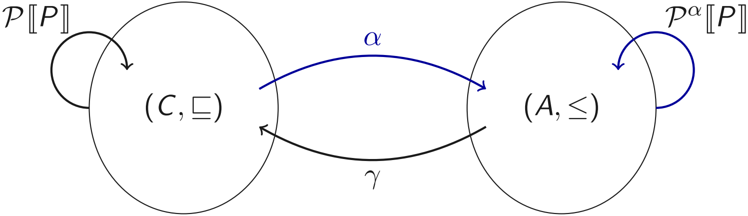


Results from the A.I. theory:

+ $\mathcal{P}^\alpha[\![P]\!] := \alpha \circ \mathcal{P}[\![P]\!] \circ \gamma$
+ $\mathcal{F}^\alpha[\![P]\!] := \mathit{lfp}\, \mathcal{P}^\alpha[\![P]\!]$
+ $\alpha(\mathcal{F}[\![P]\!]) \leq \mathcal{F}^\alpha[\![P]\!]$
+ α is precise $\implies$ $\mathcal{F}^\alpha$ is a standard semantics



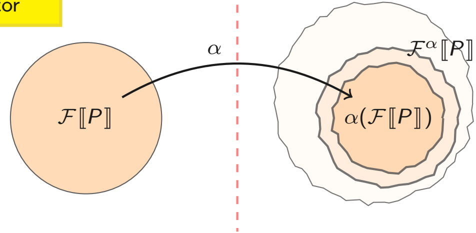
Systematic design of semantics by A.I. [Cousot 77]



Optimal abstract fix-point operator

Results from the A.I. theory:

- + $\mathcal{P}^\alpha[\![P]\!] := \alpha \circ \mathcal{P}[\![P]\!] \circ \gamma$
- + $\mathcal{F}^\alpha[\![P]\!] := \mathit{lfp}\, \mathcal{P}^\alpha[\![P]\!]$
- + $\alpha(\mathcal{F}[\![P]\!]) \leq \mathcal{F}^\alpha[\![P]\!]$
- + α is precise $\implies$ $\mathcal{F}^\alpha$ is a standard semantics



Development of a semantics adequate w.r.t. $\mathcal{B}^{cr}$

We started from a (very) concrete semantics modeling the small-step behavior ("trace semantics")

$$\mathcal{P}[\![P]\!] : \mathbb{WSST}^{\mathbb{MGC}} \to \mathbb{WSST}^{\mathbb{MGC}}$$
$$\mathcal{F}[\![P]\!] = \mathit{lfp}\, \mathcal{P}[\![P]\!]$$

Theorem

$$\mathcal{E}[\![e]\!]_{\mathcal{F}[\![P]\!]} = \mathcal{B}^{ss}[\![P]\!](e)$$

where $\mathcal{E}$ is the semantic evaluation function

Development of a semantics adequate w.r.t. $\mathcal{B}^{cr}$

We started from a (very) concrete semantics
modeling the small-step behavior ("trace semantics")

$$\mathcal{P}[\![P]\!] : \mathbb{WSST}^{\mathbb{MGC}} \to \mathbb{WSST}^{\mathbb{MGC}}$$
$$\mathcal{F}[\![P]\!] = \mathit{lfp}\,\mathcal{P}[\![P]\!]$$

most general
calls $f(\vec{x})$

Theorem

$$\mathcal{E}[\![e]\!]_{\mathcal{F}[\![P]\!]} = \mathcal{B}^{ss}[\![P]\!](e)$$

where $\mathcal{E}$ is the semantic evaluation function

Development of a semantics adequate w.r.t. $\mathcal{B}^{cr}$

We started from a (very) concrete semantics
modeling the small-step beha… ("… antics")

partial small-step
tree for $f(\vec{x})$

$$\mathcal{P}[\![P]\!] : \mathbb{WSST}^{\mathbb{MGC}} \to \mathbb{WSST}^{\mathbb{MGC}}$$
$$\mathcal{F}[\![P]\!] = \mathit{lfp}\, \mathcal{P}[\![P]\!]$$

Theorem

$$\mathcal{E}[\![e]\!]_{\mathcal{F}[\![P]\!]} = \mathcal{B}^{ss}[\![P]\!](e)$$

where $\mathcal{E}$ is the semantic evaluation function

Development of a semantics adequate w.r.t. $\mathcal{B}^{cr}$

We started from a (very) concrete semantics
modeling the small-step behavior ("trace semantics")

$$\mathcal{P}[\![P]\!] : \mathbb{WSST}^{\mathbb{MGC}} \to \mathbb{WSST}^{\mathbb{MGC}}$$
$$\mathcal{F}[\![P]\!] = \mathit{lfp}\, \mathcal{P}[\![P]\!]$$

Theorem

$$\mathcal{E}[\![e]\!]_{\mathcal{F}[\![P]\!]} = \mathcal{B}^{ss}[\![P]\!](e)$$

where $\mathcal{E}$ is the semantic evaluation function

Development of a semantics adequate w.r.t. $\mathcal{B}^{cr}$

...then, we proceed by successive abstractions

$$(\mathbb{WSST}, \sqsubseteq) \underset{\partial}{\overset{\partial^\gamma}{\leftrightarrows}} (\mathbb{ERT}, \preccurlyeq) \underset{\zeta}{\overset{\zeta^\gamma}{\leftrightarrows}} (\mathbb{WERS}, \hat{\precsim})$$

Evolving Result Abstraction

$$(\mathbb{WSST}, \sqsubseteq) \underset{\partial}{\overset{\partial^\gamma}{\leftrightarrows}} (\mathbb{ERT}, \preccurlyeq) \underset{\zeta}{\overset{\zeta^\gamma}{\leftrightarrows}} (\mathbb{WERS}, \hat{\preccurlyeq})$$

We can observe differences in the computed results when evaluation introduces a new constructor

IDEA: combine together all intermediate small steps that do not introduce a new constructor

$$\begin{tikzcd}
f(x, g(y)) \arrow[r, Rightarrow, "\{x/A\}"] & f(A, g(y)) \arrow[r, Rightarrow, "\varepsilon"] & f(A, B) \arrow[d, mapsto, "\partial"] \arrow[r, Rightarrow, "\varepsilon"] & C(h(z)) \arrow[r, Rightarrow, "\{z/B\}"] & C(A) \\
& & \varepsilon \cdot \varrho \xrightarrow{\varrho} \{x/A\} \cdot C(\varrho_1) \xrightarrow{\varrho_1} \{x/A\} \cdot C(A) & &
\end{tikzcd}$$

Evolving Result Abstraction

$$(\mathbb{WSST}, \sqsubseteq) \underset{\partial}{\overset{\partial^{\gamma}}{\leftrightarrows}} (\mathbb{ERT}, \preccurlyeq) \underset{\zeta}{\overset{\zeta^{\gamma}}{\leftrightarrows}} (\mathbb{WERS}, \hat{\preccurlyeq})$$

We can observe differences in the computed results when evaluation introduces a new constructor

IDEA: combine together all intermediate small steps that do not introduce a new constructor

$$f(x, g(y)) \xRightarrow{\{x/A\}} f(A, g(y)) \overset{\varepsilon}{\Rightarrow} f(A, B) \overset{\varepsilon}{\Rightarrow} C(h(z)) \xRightarrow{\{z/B\}} C(A)$$

$$\downarrow \partial$$

$$\varepsilon \cdot \varrho \xrightarrow{\varrho} \{x/A\} \cdot C(\varrho_1) \xrightarrow{\varrho_1} \{x/A\} \cdot C(A)$$

Evolving Result Abstraction

$$(\mathbb{WSST}, \sqsubseteq) \underset{\partial}{\overset{\partial^{\gamma}}{\leftrightarrows}} (\mathbb{ERT}, \preceq) \underset{\zeta}{\overset{\zeta^{\gamma}}{\leftrightarrows}} (\mathbb{WERS}, \hat{\precsim})$$

We can observe differences in the computed results when evaluation introduces a new constructor

IDEA: combine together all intermediate small steps that do not introduce a new constructor

$$f(x, g(y)) \xRightarrow{\{x/A\}} f(A, g(y)) \overset{\varepsilon}{\Rightarrow} f(A, B) \overset{\varepsilon}{\Rightarrow} C(h(z)) \xRightarrow{\{z/B\}} C(A)$$

$$\Big\downarrow \partial$$

$$\varepsilon \cdot \varrho \xrightarrow{\varrho} \{x/A\} \cdot C(\varrho_1) \xrightarrow{\varrho_1} \{x/A\} \cdot C(A)$$

Evolving Result Abstraction

$$(\mathbb{WSST}, \sqsubseteq) \underset{\partial}{\overset{\partial^{\gamma}}{\leftrightarrows}} (\mathbb{ERT}, \preccurlyeq) \underset{\zeta}{\overset{\zeta^{\gamma}}{\leftrightarrows}} (\mathbb{WERS}, \hat{\precsim})$$

We can observe differences in the computed results when evaluation introduces a new constructor

IDEA: combine together all intermediate small steps that do not introduce a new constructor

$$f(x, g(y)) \xRightarrow{\{x/A\}} f(A, g(y)) \overset{\varepsilon}{\Rightarrow} f(A, B) \overset{\varepsilon}{\Rightarrow} C(h(z)) \xRightarrow{\{z/B\}} C(A)$$

$$\Big\downarrow \partial$$

$$\varepsilon \cdot \varrho \xrightarrow{\varrho} \{x/A\} \cdot C(\varrho_1) \xrightarrow{\varrho_1} \{x/A\} \cdot C(A)$$

Evolving Result semantics

$$(\mathbb{WSST}, \sqsubseteq) \underset{\partial}{\overset{\partial^\gamma}{\leftrightarrows}} (\mathbb{ERT}, \preccurlyeq) \underset{\zeta}{\overset{\zeta^\gamma}{\leftrightarrows}} (\mathbb{WERS}, \hat{\precsim})$$

The $\mathbb{ERT}$ (Evolving Result Trees) domain: $\mathbb{ERT} := \partial(\mathbb{WSST})$

$$\begin{tikzcd}
 & \{x/Z\} \cdot y & \\
\varepsilon \cdot \varrho \arrow[ur, "\varrho"] \arrow[dr, "\varrho"'] & & \{x/S(Z)\} \cdot S(y) \\
 & \{x/S(x_1)\} \cdot S(\varrho_1) \arrow[ur, "\varrho_1"] \arrow[dr, "\varrho_1"'] & \\
 & & \{x/S(S(x_2))\} \cdot S(S(\varrho_2))
\end{tikzcd}$$

infinite depth

Evolving Result semantics

$$(\mathbb{WSST}, \sqsubseteq) \underset{\partial}{\overset{\partial^\gamma}{\leftrightarrows}} (\mathbb{ERT}, \preccurlyeq) \underset{\zeta}{\overset{\zeta^\gamma}{\leftrightarrows}} (\mathbb{WERS}, \hat{\precsim})$$

The $\mathbb{ERT}$ (Evolving Result Trees) domain: $\mathbb{ERT} := \partial(\mathbb{WSST})$

$$\begin{array}{lll}
 & \xrightarrow{\varrho} & \{x/Z, y/S(y_1)\} \cdot True \\
 & \xrightarrow{\varrho} & \{x/S(Z), y/S(S(y_1))\} \cdot True \\
\varepsilon \cdot \varrho & & \vdots \\
 & \xrightarrow{\varrho} & \{x/S(S(x_1)), y/S(Z)\} \cdot False \\
 & \xrightarrow{\varrho} & \{x/S(x_1), y/Z\} \cdot False
\end{array}$$

infinite width

Evolving Result semantics

$$(\mathbb{WSST}, \sqsubseteq) \underset{\partial}{\overset{\partial^\gamma}{\leftrightarrows}} (\mathbb{ERT}, \preccurlyeq) \underset{\zeta}{\overset{\zeta^\gamma}{\leftrightarrows}} (\mathbb{WERS}, \hat{\preccurlyeq})$$

Induced optimal immediate consequence operator

$$\mathcal{P}^\partial[\![P]\!] : \mathbb{ERT}^{\mathbb{MGC}} \to \mathbb{ERT}^{\mathbb{MGC}}$$
$$\mathcal{P}^\partial[\![P]\!]_{\mathcal{I}^\partial} := (\partial \circ \mathcal{P}[\![P]\!] \circ \partial^\gamma)(\mathcal{I}^\partial)$$
$$= \lambda f(\overrightarrow{x_n}).\ \bigvee \left\{ \varepsilon \cdot \varrho \xrightarrow{\varrho} \mathcal{E}^\partial[\![r]\!]_{\mathcal{I}^\partial}^{\{\overrightarrow{x_n}/\overrightarrow{t_n}\}} \,\middle|\, f(t) \to r \in P \right\}$$

Evaluation function over $\mathbb{ERT}$

$$\mathcal{E}^\partial[\![x]\!]^\sigma_{\mathcal{I}^\partial} := \sigma \cdot x$$
$$\mathcal{E}^\partial[\![\varphi(\overrightarrow{t_n})]\!]^\sigma_{\mathcal{I}^\partial} := \mathcal{I}^\partial(\varphi(\overrightarrow{y_n}))[y_1/\,\mathcal{E}^\partial[\![t_1]\!]^\sigma_{\mathcal{I}^\partial}] \dots [y_n/\,\mathcal{E}^\partial[\![t_n]\!]^\sigma_{\mathcal{I}^\partial}]$$

Theorem

$$\mathcal{F}^\partial[\![P]\!] = \partial(\mathcal{F}[\![P]\!])$$

Evolving Result semantics

$$(\mathbb{WSST}, \sqsubseteq) \underset{\partial}{\overset{\partial^\gamma}{\leftrightarrows}} (\mathbb{ERT}, \preccurlyeq) \underset{\zeta}{\overset{\zeta^\gamma}{\leftrightarrows}} (\mathbb{WERS}, \hat{\preccurlyeq})$$

Theorem (correctness)

$$\mathcal{F}^\partial[\![P_1]\!] = \mathcal{F}^\partial[\![P_2]\!] \implies \forall Q \in \mathbb{UP}_{\Sigma}^{\Sigma'}.\ \mathcal{B}^{cr}[\![P_1 \cup Q]\!] = \mathcal{B}^{cr}[\![P_2 \cup Q]\!]$$

The converse implication doesn't hold

Counterexample

Consider the programs P_1 and P_2

```
f x = A x                    f x = id (A (id x))
```

$\mathcal{F}^\partial[\![P_1]\!](f(x)) = \varepsilon \cdot \varrho \xrightarrow{\varrho} \varepsilon \cdot A(x)$ whereas
$\mathcal{F}^\partial[\![P_2]\!](f(x)) = \varepsilon \cdot \varrho \xrightarrow{\varrho} \varepsilon \cdot A(\varrho_1) \xrightarrow{\varrho_1} \varepsilon \cdot A(x)$.

Only when a substitution changes there is a visible effect in the behavior

Evolving Result semantics

$$(\mathbb{WSST}, \sqsubseteq) \underset{\partial}{\overset{\partial^\gamma}{\leftrightarrows}} (\mathbb{ERT}, \preccurlyeq) \underset{\zeta}{\overset{\zeta^\gamma}{\leftrightarrows}} (\mathbb{WERS}, \hat{\precsim})$$

Theorem (correctness)

$$\mathcal{F}^\partial[\![P_1]\!] = \mathcal{F}^\partial[\![P_2]\!] \implies \forall Q \in \mathbb{UP}_\Sigma^{\Sigma'}.\ \mathcal{B}^{cr}[\![P_1 \cup Q]\!] = \mathcal{B}^{cr}[\![P_2 \cup Q]\!]$$

The converse implication doesn't hold

Counterexample

Consider the programs P_1 and P_2

```
f x = A x                    f x = id (A (id x))
```

$\mathcal{F}^\partial[\![P_1]\!](f(x)) = \varepsilon \cdot \varrho \xrightarrow{\varrho} \varepsilon \cdot A(x)$ whereas
$\mathcal{F}^\partial[\![P_2]\!](f(x)) = \varepsilon \cdot \varrho \xrightarrow{\varrho} \varepsilon \cdot A(\varrho_1) \xrightarrow{\varrho_1} \varepsilon \cdot A(x)$.

Only when a substitution changes there is a visible effect in the behavior

Weakly Evolving Abstraction

$$(\mathbb{WSST}, \sqsubseteq) \underset{\partial}{\overset{\partial^{\gamma}}{\leftrightarrows}} (\mathbb{ERT}, \preccurlyeq) \underset{\zeta}{\overset{\zeta^{\gamma}}{\leftrightarrows}} (\mathbb{WERS}, \hat{\precsim})$$

IDEA: combine together all partial computed results that refer to the same substitution and lead to the same partial result

concise representation: we denotes with $\sigma \centerdot s_1 – s_2$ the set of partial computed results $\sigma \cdot s$ where $s_1 \lesssim s \lesssim s_2$.

$$\begin{tikzcd}
\varepsilon \cdot \varrho \xrightarrow{\varrho} \varepsilon \cdot A(x) \arrow[dr, mapsto, "\zeta"'] & & \varepsilon \cdot \varrho_1 \xrightarrow{\varrho_1} \varepsilon \cdot A(\varrho_2) \xrightarrow{\varrho_2} \varepsilon \cdot A(x) \arrow[dl, mapsto, "\zeta"] \\
& \varepsilon \centerdot \varrho – A(x) &
\end{tikzcd}$$

Weakly Evolving Abstraction

$$(\mathbb{WSST}, \sqsubseteq) \underset{\partial}{\overset{\partial^{\gamma}}{\leftrightarrows}} (\mathbb{ERT}, \preccurlyeq) \underset{\zeta}{\overset{\zeta^{\gamma}}{\leftrightarrows}} (\mathbb{WERS}, \hat{\lesssim})$$

IDEA: combine together all partial computed results that refer to the same substitution and lead to the same partial result

concise representation: we denotes with $\sigma \,\mathbf{.}\, s_1{-}s_2$ the set of partial computed results $\sigma \cdot s$ where $s_1 \lesssim s \lesssim s_2$.

interval

$$\begin{tikzcd}
\varepsilon \cdot \varrho \arrow[r, "\varrho"] & \varepsilon \cdot A(x) \arrow[drr, mapsto, "\zeta"] & & \varepsilon \cdot \varrho_1 \arrow[r, "\varrho_1"] & \varepsilon \cdot A(\varrho_2) \arrow[dl, mapsto, "\zeta"'] \arrow[r, "\varrho_2"] & \varepsilon \cdot A(x) \\
& & & \varepsilon \,\mathbf{.}\, \varrho{-}A(x) & &
\end{tikzcd}$$

Weakly Evolving semantics

$$(\mathbb{WSST}, \sqsubseteq) \underset{\partial}{\overset{\partial^\gamma}{\leftrightarrows}} (\mathbb{ERT}, \preccurlyeq) \underset{\zeta}{\overset{\zeta^\gamma}{\leftrightarrows}} (\mathbb{WERS}, \hat{\precsim})$$

Induced immediate consequence operator

$$\mathcal{P}^\nu[\![P]\!] : \mathbb{WERS}^{\mathbb{MGC}} \to \mathbb{WERS}^{\mathbb{MGC}}$$

$$\mathcal{P}^\nu[\![P]\!]_{\mathcal{I}^\nu} = \lambda f(\overrightarrow{x_n}).\ \hat{\Upsilon}\left\{\mathcal{E}^\nu[\![r]\!]_{\mathcal{I}^\nu}^{\{\overrightarrow{x_n}/\overrightarrow{t_n}\}} \,\middle|\, f(t) \to r \in P\right\}$$

Evaluation function over $\mathbb{WERS}$

$$\mathcal{E}^\nu[\![x]\!]_{\mathcal{I}^\partial}^\sigma := \sigma \centerdot \varrho{-}x$$

$$\mathcal{E}^\nu[\![\varphi(\overrightarrow{t_n})]\!]_{\mathcal{I}^\nu}^\sigma := \mathcal{I}^\nu(\varphi(\overrightarrow{y_n}))[y_1/\,\mathcal{E}^\nu[\![t_1]\!]_{\mathcal{I}^\nu}^\sigma] \ldots [y_n/\,\mathcal{E}^\nu[\![t_n]\!]_{\mathcal{I}^\nu}^\sigma]$$

Theorem (full-abstraction)

- $\nu(\mathcal{E}[\![e]\!]_{\mathcal{I}}) = \mathcal{E}^\nu[\![e]\!]_{\nu(\mathcal{I})}$
- $\forall P\ \mathcal{F}^\nu[\![P]\!] = \nu(\mathcal{F}[\![P]\!])$
- $\mathcal{F}^\nu[\![P_1]\!] = \mathcal{F}^\nu[\![P_2]\!] \iff \forall Q \in \mathbb{UP}_{\Sigma}^{\Sigma'}.\ \mathcal{B}^{cr}[\![P_1 \cup Q]\!] = \mathcal{B}^{cr}[\![P_2 \cup Q]\!]$

what about Haskell?

- By a simple program transformation (Cnv) an Haskell program is transformed into a Curry semantic-equivalent version

Theorem (Adequacy of Cnv)

Given P an Haskell program and e_0 ground expression.

$$e_0 \overset{p_1}{\Longrightarrow} \dots \overset{p_n}{\Longrightarrow} e_n \textit{ using } P \iff e_0 \underset{\varepsilon}{\overset{p_1}{\Longrightarrow}} \dots \underset{\varepsilon}{\overset{p_n}{\Longrightarrow}} e_n \textit{ using } Cnv(P)$$

- . . . all results apply to Haskell as well

APPLICATIONS

Abstraction Framework:

- consider a **true** abstraction α
- $(\mathbb{WERS}, \hat{\precsim}) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} (\mathbb{A}, \leq)$
- abstract semantics $\mathcal{F}^{\alpha}$ **can be effectively computed**

Proposed case studies: *depth*(k) and $\mathcal{POS}$

Applications:

- Static Analysis
- Abstract Debugging
- Automatic Synthesis of algebraic Specifications

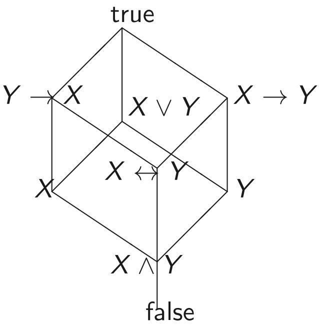
Application: Groundness Dependencies Analysis

first proposal in literature

Domain: $(\mathcal{POS}, \leq)$ set of positive formulas ordered by implication

Abstraction:
Collects $\mathcal{POS}$ abstractions of (final) computed results only

+ $\Gamma_\varrho(S) := \bigvee\{\Gamma(\sigma\{\varrho/v\}) \mid \sigma \,\mathbf{.}\, t{-}v \in S,\ v \in \mathbb{T}(\mathcal{C}, \mathcal{V})\}$ (WERS)
+ $\Gamma(\vartheta) := \bigwedge_{y/t \in \vartheta}(y \leftrightarrow (\bigwedge_{x \in var(t)} x))$ (substitutions)



Examples:

+ $x + y \triangleright \varrho \mapsto x \wedge (\varrho \leftrightarrow y)$
  first argument ground, and result ground iff second argument ground
+ $x \leq y \triangleright \varrho \mapsto \varrho \wedge (x \vee y)$
  result ground, and at least one argument ground

Induced optimal immediate consequence operator

$$\begin{aligned}\mathcal{P}^{gr}[\![P]\!] &:= \alpha_\Gamma \circ \mathcal{P}^\nu \circ \gamma_\Gamma \\ &= \lambda f(\overrightarrow{x_n}) \rhd \varrho. \bigvee_{f(\overrightarrow{t_n}) \to r \in P} (\Gamma(\{\overrightarrow{x_n}/\overrightarrow{t_n}\}) \wedge \mathcal{E}^{gr}[\![r \rhd \varrho]\!]_{\mathcal{I}^{gr}})|_{\overrightarrow{x_n},\varrho}\end{aligned}$$

Evaluation function over $\mathbb{GR}$

$$\mathcal{E}^{gr}[\![x \rhd \varrho]\!]_{\mathcal{I}^{gr}} := \varrho \leftrightarrow x$$

$$\mathcal{E}^{gr}[\![\varphi(\overrightarrow{t_n}) \rhd \varrho]\!]_{\mathcal{I}^{gr}} := \mathcal{I}^{gr}(\varphi(\overrightarrow{\varrho_n}) \rhd \varrho) \wedge \bigwedge_{i=1}^{n} \Phi_i \qquad \overrightarrow{\varrho_n} \text{ fresh}$$

where

$$\Phi_i := \begin{cases} \mathcal{E}^{gr}[\![t_i \rhd \varrho_i]\!]_{\mathcal{I}^{gr}} & \text{if } \mathcal{I}^{gr}(\varphi(\overrightarrow{\varrho_n}) \rhd \varrho) \leq (\varrho \to \varrho_i) \text{ or} \\ & \quad t_i \in \mathcal{T}(\mathcal{C}, \mathcal{V}) \\ \mathit{true} & \text{otherwise} \end{cases}$$

Analysis Example

Program:

```
[] ++ ys     = ys
(x:xs) ++ ys = x : (xs++ys)
```

Analysis session:

$$\mathcal{P}^{gr}[\![P]\!]\!\uparrow\!0 = \left\{ xs +\!\!+\ ys \triangleright \varrho \mapsto \mathit{false} \right.$$

$$\mathcal{P}^{gr}[\![P]\!]\!\uparrow\!1 = \left\{ xs +\!\!+\ ys \triangleright \varrho \mapsto xs \wedge (\varrho \leftrightarrow ys) \right.$$

$$\mathcal{P}^{gr}[\![P]\!]\!\uparrow\!2 = \left\{ xs +\!\!+\ ys \triangleright \varrho \mapsto \varrho \leftrightarrow (xs \wedge ys) \right.$$

$$\mathcal{P}^{gr}[\![P]\!]\!\uparrow\!3 = \mathcal{P}^{gr}[\![P]\!]\!\uparrow\!2 = \mathcal{P}^{gr}[\![P]\!]\!\uparrow\!\omega$$

. . . running tool

Analysis Example

Program:

```
[] ++ ys = ys
(x:xs) ++ ys = x : (xs++ys)
```

Analysis session:

$$\mathcal{P}^{gr}[\![P]\!]\uparrow 0 = \left\{ xs +\!\!+\, ys \triangleright \varrho \mapsto \mathit{false} \right.$$

$$\mathcal{P}^{gr}[\![P]\!]\uparrow 1 = \left\{ xs +\!\!+\, ys \triangleright \varrho \mapsto xs \wedge (\varrho \leftrightarrow ys) \right.$$

$$\mathcal{P}^{gr}[\![P]\!]\uparrow 2 = \left\{ xs +\!\!+\, ys \triangleright \varrho \mapsto \varrho \leftrightarrow (xs \wedge ys) \right.$$

$$\mathcal{P}^{gr}[\![P]\!]\uparrow 3 = \mathcal{P}^{gr}[\![P]\!]\uparrow 2 = \mathcal{P}^{gr}[\![P]\!]$$

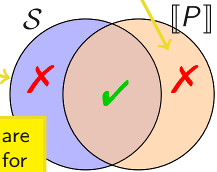
the result of ++ is ground
if and only if both its
argument are ground

. . . running tool

Application: Abstract Diagnosis

incompleteness symptom

incorrectness symptom



Automatic Debugging

Input: program P + specification $\mathcal{S}$

Goal: automatically locate bugs in P

in general it is **undecidable**

rules which are responsible for symptoms ✗

How to deal with this problem?

+ Declarative Debugging $\Rightarrow$ partial inspection of the symptomatic *computation tree*
+ **Abstract Diagnosis** $\Rightarrow$ use a correct approximation of the semantics which is finitely representable

Application: Abstract Diagnosis

incompleteness symptom

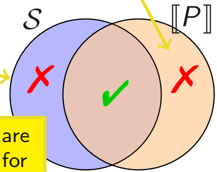
incorrectness symptom

Automatic Debugging

Input: program P + specification $\mathcal{S}$

Goal: automatically locate bugs in P

in general it is **undecidable**



rules which are responsible for symptoms ✗

How to deal with this problem?

+ Declarative Debugging

+ **Abstract Diagnosis**

There are some cons:

+ symptom driven
+ semi-automatic
+ can't ensure that a property holds for P

Application: Abstract Diagnosis

incompleteness symptom

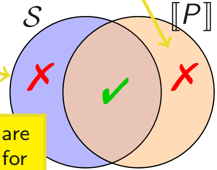
incorrectness symptom

Automatic Debugging

Input: program P + specification $\mathcal{S}$

Goal: automatically locate bugs in P

in general it is **undecidable**



rules which are responsible for symptoms ✗

How to deal with this problem?

+ Declarative Debugging
+ **Abstract Diagnosis**

does not suffer

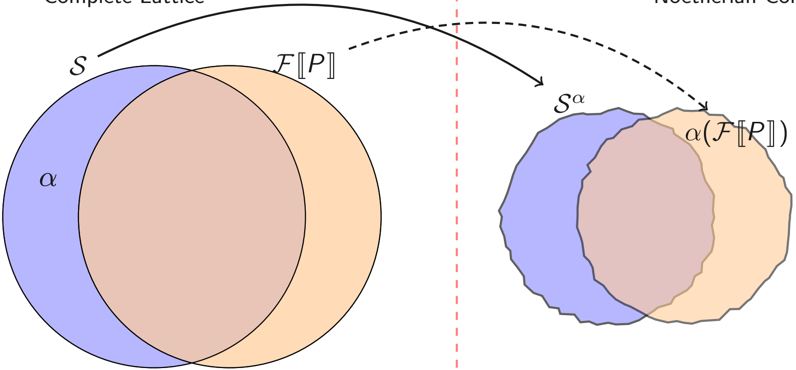
There are some cons:

+ symptom driven
+ semi-automatic
+ can't ensure that a property holds for P

The main idea

$(\mathbb{C}, \sqsubseteq, \sqcup, \sqcap, \bot_{\mathbb{C}}, \top_{\mathbb{C}})$
Complete Lattice

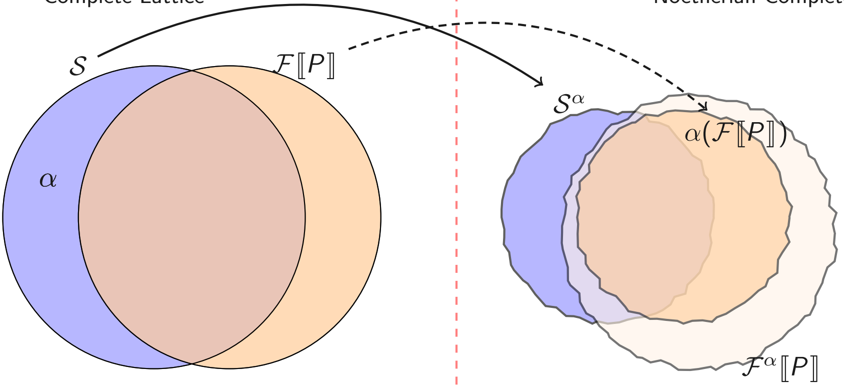
$(\mathbb{A}, \leq, \vee, \wedge, \bot_{\mathbb{A}}, \top_{\mathbb{A}})$
Noetherian Complete Lattice



The main idea

$(\mathbb{C}, \sqsubseteq, \sqcup, \sqcap, \bot_{\mathbb{C}}, \top_{\mathbb{C}})$
Complete Lattice

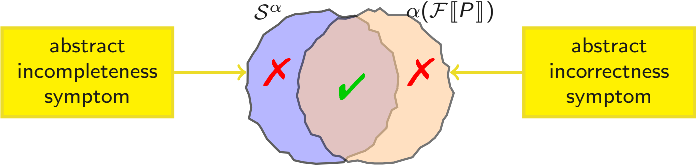
$(\mathbb{A}, \leq, \vee, \wedge, \bot_{\mathbb{A}}, \top_{\mathbb{A}})$
Noetherian Complete Lattice



Abstract Bugs & Symptoms (Abstract Diagnosis)

Let P be a program and α a property

+ (abstract) **partially correct** w.r.t. $\mathcal{S}^\alpha$: $\alpha(\mathcal{F}[\![P]\!]) \leq \mathcal{S}^\alpha$
+ (abstract) **complete** w.r.t. $\mathcal{S}^\alpha$: $\mathcal{S}^\alpha \leq \alpha(\mathcal{F}[\![P]\!])$



Problem: **interference** between incorrectness and uncovered errors **can be symptomless**

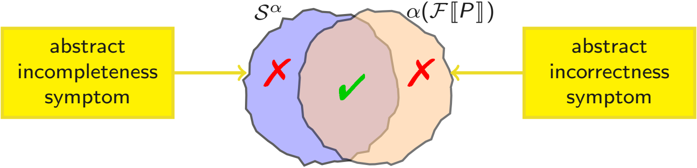
$\Downarrow$

Declarative Diagnosis **cannot** reveal all errors **simultaneously**

Abstract Bugs & Symptoms

Let P be a program and α a property

- (abstract) **partially correct** w.r.t. $\mathcal{S}^\alpha$: $\alpha(\mathcal{F}[\![P]\!]) \leq \mathcal{S}^\alpha$
- (abstract) **complete** w.r.t. $\mathcal{S}^\alpha$: $\mathcal{S}^\alpha \leq \alpha(\mathcal{F}[\![P]\!])$



Problem: **interference** between incorrectness and uncovered errors **can be symptomless**

$\Downarrow$

Declarative Diagnosis **cannot** reveal all errors **simultaneously**

Abstract Diagnosis Framework

Based on **abstract version of Park's Induction Principle**:

$$\mathcal{P}^{\alpha}[\![P]\!]_{\mathcal{S}^{\alpha}} \overset{?}{\leq} \mathcal{S}^{\alpha}$$

+ $e \leq \mathcal{P}^{\alpha}[\![\{l \to r\}]\!]_{\mathcal{S}^{\alpha}}$ and $e \not\leq \mathcal{S}^{\alpha}$ (abstractly incorrect rule)

+ $e \wedge \mathcal{P}^{\alpha}[\![P]\!]_{\mathcal{S}^{\alpha}} = \bot_{\mathbb{A}}$ and $e \leq \mathcal{S}^{\alpha}$ (abstractly uncovered elem.)

Abstract Diagnosis Framework

Based on **abstract version of Park's Induction Principle**:

$$\mathcal{P}^\alpha[\![P]\!]_{\mathcal{S}^\alpha} \overset{?}{\leq} \mathcal{S}^\alpha$$

using $\mathcal{S}^\alpha$,
$l \to r$
produces $e\ldots$

+ $e \leq \mathcal{P}^\alpha[\![\{l \to r\}]\!]_{\mathcal{S}^\alpha}$ and $e \not\leq \mathcal{S}^\alpha$ (abstractly incorrect rule)

+ $e \wedge \mathcal{P}^\alpha[\![P]\!]_{\mathcal{S}^\alpha} = \bot_{\mathbb{A}}$ and $e \leq \mathcal{S}^\alpha$ (abstractly uncovered elem.)

Abstract Diagnosis Framework

Based on **abstract version of Park's Induction Principle**:

$$\mathcal{P}^\alpha[\![P]\!]_{\mathcal{S}^\alpha} \stackrel{?}{\leq} \mathcal{S}^\alpha$$

using $\mathcal{S}^\alpha$, $l \to r$ produces e...

...but e was not expected by $\mathcal{S}^\alpha$

+ $e \leq \mathcal{P}^\alpha[\![\{l \to r\}]\!]_{\mathcal{S}^\alpha}$ and $e \not\leq \mathcal{S}^\alpha$ (abstractly incorrect rule)

+ $e \wedge \mathcal{P}^\alpha[\![P]\!]_{\mathcal{S}^\alpha} = \perp_{\mathbb{A}}$ and $e \leq \mathcal{S}^\alpha$ (abstractly uncovered elem.)

Abstract Diagnosis Framework

Based on **abstract version of Park's Induction Principle**:

$$\mathcal{P}^{\alpha}[\![P]\!]_{\mathcal{S}^{\alpha}} \stackrel{?}{\leq} \mathcal{S}^{\alpha}$$

using $\mathcal{S}^{\alpha}$, $l \rightarrow r$ produces e...

...but e was not expected by $\mathcal{S}^{\alpha}$

+ $e \leq \mathcal{P}^{\alpha}[\![\{l \rightarrow r\}]\!]_{\mathcal{S}^{\alpha}}$ and $e \not\leq \mathcal{S}^{\alpha}$ (abstractly incorrect rule)

using $\mathcal{S}^{\alpha}$, P can't produce e...

+ $e \wedge \mathcal{P}^{\alpha}[\![P]\!]_{\mathcal{S}^{\alpha}} = \bot_{\mathbb{A}}$ and $e \leq \mathcal{S}^{\alpha}$ (abstractly uncovered elem.)

Abstract Diagnosis Framework

Based on **abstract version of Park's Induction Principle**:

$$\mathcal{P}^\alpha[\![P]\!]_{\mathcal{S}^\alpha} \overset{?}{\leq} \mathcal{S}^\alpha$$

using $\mathcal{S}^\alpha$, $l \to r$ produces e...

...but e was not expected by $\mathcal{S}^\alpha$

+ $e \leq \mathcal{P}^\alpha[\![\{l \to r\}]\!]_{\mathcal{S}^\alpha}$ and $e \not\leq \mathcal{S}^\alpha$ (abstractly incorrect rule)

using $\mathcal{S}^\alpha$, P can't produce e...

...but e was expected by $\mathcal{S}^\alpha$

+ $e \wedge \mathcal{P}^\alpha[\![P]\!]_{\mathcal{S}^\alpha} = \bot_{\mathbb{A}}$ and $e \leq \mathcal{S}^\alpha$ (abstractly uncovered elem.)

Abstract Diagnosis Framework

Based on **abstract version of Park's Induction Principle**:

$$\mathcal{P}^\alpha[\![P]\!]_{\mathcal{S}^\alpha} \overset{?}{\leq} \mathcal{S}^\alpha$$

using $\mathcal{S}^\alpha$, $l \to r$ produces e...

...but e was not expected by $\mathcal{S}^\alpha$

+ $e \leq \mathcal{P}^\alpha[\![\{l \to r\}]\!]_{\mathcal{S}^\alpha}$ and $e \not\leq \mathcal{S}^\alpha$ (abstractly incorrect rule)

using $\mathcal{S}^\alpha$, P can't produce e...

...but e was expected by $\mathcal{S}^\alpha$

+ $e \wedge \mathcal{P}^\alpha[\![P]\!]_{\mathcal{S}^\alpha} = \perp_{\mathbb{A}}$ and $e \leq \mathcal{S}^\alpha$ (abstractly uncovered elem.)

Pros:
+ Static test (requires just one $\mathcal{P}^\alpha[\![P]\!]$ step on $\mathcal{S}^\alpha$)
+ reveal all abstract errors regardless of symptoms interference

Cons:
+ imprecision of α can lead to **false positives**:

Case study: *depth*(k)

Program: R: `from n = n : from n`

Specification: with $\kappa = 3$

$$\mathcal{S}^{\kappa} := \left\{ \mathit{from}(n) \mapsto \{ \varepsilon \centerdot \varrho \text{–} n{:}S(\hat{x}_1) : \hat{x}_2 : \hat{x}_3 \} \right.$$

We detect that rule R is abstractly incorrect since

$$\mathcal{P}^{\kappa}[\![\{R\}]\!]_{\mathcal{S}^{\kappa}} = \left\{ \mathit{from}(n) \mapsto \{ \varepsilon \centerdot \varrho \text{–} n : n : \hat{x}_1 : \hat{x}_2 \} \right. \quad \not\leq \mathcal{S}^{\kappa}$$

Application: Automatic Synthesis of algebraic property-oriented Specifications

Goal:

Automatically infer a set of equations of the form $e_1 = e_2$ relating program calls to their behavior

in general it is **undecidable**

- Black-Box approach ⇒ works only by running the executable on a (automatically generated) set of tests from which the specification is inferred.
  - ✔ no restrictions on the considered language
  - ✘ **cannot guarantee the correctness** of the results
- **Glass-Box approach** ⇒ assumes that the source code of the program is available.
  - ✘ language-dependent
  - ✔ the inference can be semantic-based ⇒ the inferred equations **can be correct**

What specifications? (Automatic Synthesis of Specifications)

Program:

```
not True = False
not False = True
or True _ = True
or False x = x
```

```
and True x = x
and False _ = False
imp False x = True
imp True x = x
```

what kind of expression one would expect?

the **lazy** nature of the language makes this aspect not so trivial . . .

Contextual Equiv. states that two terms have the same computed results for any context $C[]$

$$\texttt{or x y} =_C \texttt{imp (not x) y}$$
$$\texttt{not (not (not x))} =_C \texttt{not x}$$

Computed-result Equiv. states that two terms have the same computed results

Ground Equiv. states that two terms have the same outcome for every ground instance.

Equations Kinds

Contextual Equiv. states that two terms have the same computed results for any context $C[\,]$

$$e_1 =_C e_2 \iff \mathcal{E}^{\nu}[\![e_1]\!]_{\mathcal{F}^{\nu}[\![P]\!]} = \mathcal{E}^{\nu}[\![e_2]\!]_{\mathcal{F}^{\nu}[\![P]\!]}$$

Computed-result Equiv. states that two terms have the same computed results

$\texttt{or (not x) y} =_{CR} \texttt{imp x y}$

$\texttt{not (and x y)} =_{CR} \texttt{imp x (not y)}$

Ground Equiv. states that two terms have the same outcome for every ground instance.

Contextual Equiv. states that two terms have the same computed results for any context $C[\,]$

$$e_1 =_C e_2 \iff \mathcal{E}^{\nu}[\![e_1]\!]_{\mathcal{F}^{\nu}[\![P]\!]} = \mathcal{E}^{\nu}[\![e_2]\!]_{\mathcal{F}^{\nu}[\![P]\!]}$$

Computed-result Equiv. states that two terms have the same computed results

$$e_1 =_{CR} e_2 \iff cr(\mathcal{E}^{\nu}[\![e_1]\!]_{\mathcal{F}^{\nu}[\![P]\!]}) = cr(\mathcal{E}^{\nu}[\![e_2]\!]_{\mathcal{F}^{\nu}[\![P]\!]})$$

Ground Equiv. states that two terms have the same outcome for every ground instance.

$\texttt{x} =_G \texttt{not (not x)}$

$\texttt{and x (and y z)} =_G \texttt{and (and x y) z}$

$\texttt{not (or x y)} =_G \texttt{and (not x) (not y)}$

Contextual Equiv. states that two terms have the same computed results for any context $C[\,]$

$$e_1 =_C e_2 \iff \mathcal{E}^{\nu}[\![e_1]\!]_{\mathcal{F}^{\nu}[\![P]\!]} = \mathcal{E}^{\nu}[\![e_2]\!]_{\mathcal{F}^{\nu}[\![P]\!]}$$

Computed-result Equiv. states that two terms have the same computed results

$$e_1 =_{CR} e_2 \iff cr(\mathcal{E}^{\nu}[\![e_1]\!]_{\mathcal{F}^{\nu}[\![P]\!]}) = cr(\mathcal{E}^{\nu}[\![e_2]\!]_{\mathcal{F}^{\nu}[\![P]\!]})$$

Ground Equiv. states that two terms have the same outcome for every ground instance.

$$e_1 =_G e_2 \iff g(cr(\mathcal{E}^{\nu}[\![e_1]\!]_{\mathcal{F}^{\nu}[\![P]\!]})) = g(cr(\mathcal{E}^{\nu}[\![e_2]\!]_{\mathcal{F}^{\nu}[\![P]\!]}))$$

Contextual Equiv. states that two terms have the same computed results for any context $C[\,]$

$$e_1 =_C e_2 \iff \mathcal{E}^\nu[\![e_1]\!]_{\mathcal{F}^\nu[\![P]\!]} = \mathcal{E}^\nu[\![e_2]\!]_{\mathcal{F}^\nu[\![P]\!]}$$

Computed-result Equiv. states that two terms have the same computed results

$$e_1 =_{CR} e_2 \iff cr(\mathcal{E}^\nu[\![e_1]\!]_{\mathcal{F}^\nu[\![P]\!]}) = cr(\mathcal{E}^\nu[\![e_2]\!]_{\mathcal{F}^\nu[\![P]\!]})$$

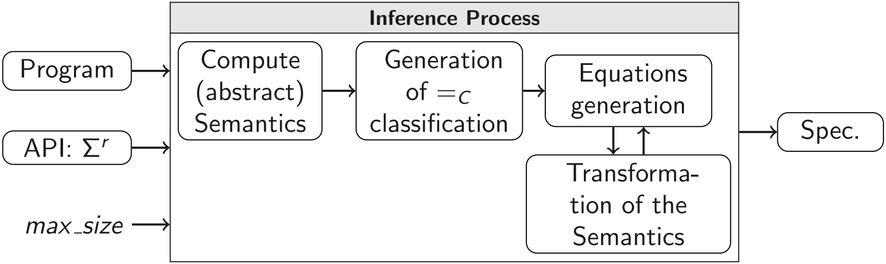
Ground Equiv. states that two terms have the same outcome for every ground instance.

$$e_1 =_G e_2 \iff g(cr(\mathcal{E}^\nu[\![e_1]\!]_{\mathcal{F}^\nu[\![P]\!]})) = g(cr(\mathcal{E}^\nu[\![e_2]\!]_{\mathcal{F}^\nu[\![P]\!]}))$$

Specification in AbsSpec: $=_C \subseteq =_{CR} \subseteq =_G$

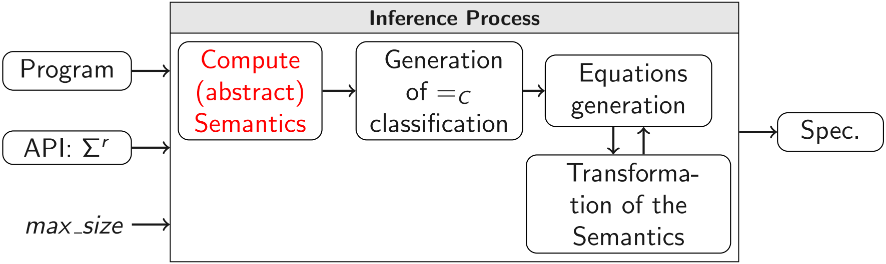
A set of equations $e_1 =_{\{C,CR,G\}} e_2$ where $e_1, e_2 \in \mathcal{T}(\Sigma^r, \mathcal{V})$

Inference Process (Automatic Synthesis of Specifications)

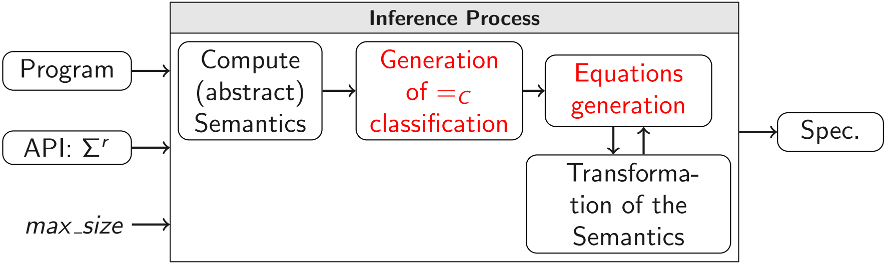


Classification = "a set of pairs of the form $\langle S, \{e_1, \ldots, e_n\}\rangle$"

Inference Process (Automatic Synthesis of Specifications)

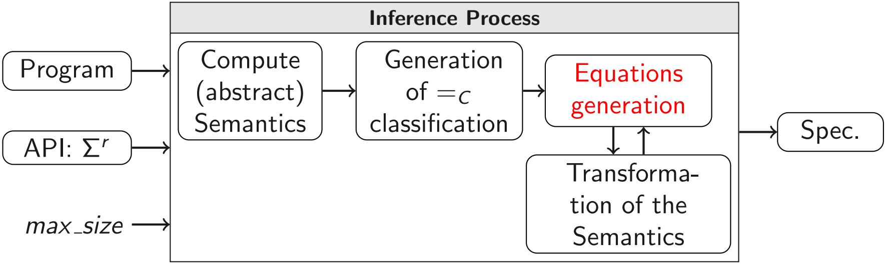


- compute $\mathcal{F}^\alpha[\![P]\!]$
- $S_{f(\overrightarrow{x_n})} := \mathcal{F}^\alpha[\![P]\!](f(\overrightarrow{x_n}))$ for every $f \in \Sigma^r$
- $\mathcal{C}_0 := \{\langle S_{f(\overrightarrow{x_n})}, \{f(\overrightarrow{x_n})\}\rangle \mid f \in \Sigma^r\}$



- iterate *max_size* times
  - take $f \in \Sigma^r$ and $\langle S_1, E_1\rangle, \ldots, \langle S_k, E_k\rangle \in \mathcal{C}_h$
  - compute $S = S_{f(\overrightarrow{x_n})}[x_1/S_1]\ldots[x_n/S_n]$
  - update $\mathcal{C}_h$ inserting $\langle S, \{f(\overrightarrow{e_n})\}\rangle$ where $e_i = min\, E_i$
- $\mathcal{C} := \mathcal{C}_{max\_size}$ and print the equations $e_1 =_C \cdots =_C e_n$ for each $\langle S, \{e_1, \ldots, e_n\}\rangle \in \mathcal{C}$.

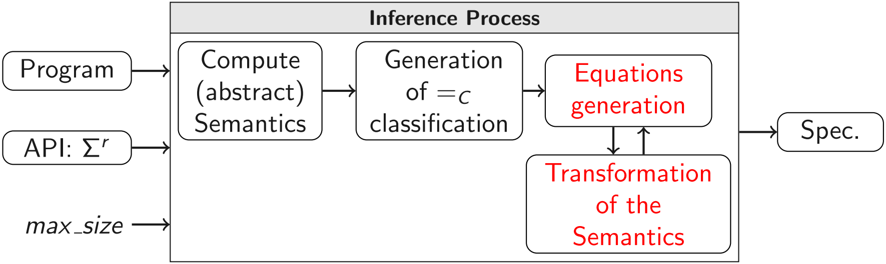
Inference Process (Automatic Synthesis of Specifications)



associated to every $f(\overrightarrow{e_n})$
for any $\overrightarrow{e_n} \in E_1 \times \cdots E_n$

- iterate max_s
  - take $f \in \Sigma^r$ and $\langle S_1, E_1\rangle, \ldots, \langle S_k, E_k\rangle \in \mathcal{C}_h$
  - compute $S = S_{f(\overrightarrow{x_n})}[x_1/S_1] \ldots [x_n/S_n]$
  - update $\mathcal{C}_h$ inserting $\langle S, \{f(\overrightarrow{e_n})\}\rangle$ where $e_i = min\, E_i$
- $\mathcal{C} := \mathcal{C}_{max\_size}$ and print the equations $e_1 =_C \cdots =_C e_n$ for each $\langle S, \{e_1, \ldots, e_n\}\rangle \in \mathcal{C}$.

Inference Process (Automatic Synthesis of Specifications)



- Compute $=_{CR}$ equations
  - $\mathcal{C}_{CR} = gather(\{\langle cr(S), \{min\, E\}\rangle \mid \langle S, E\rangle \in \mathcal{C}\})$
  - print the induced $=_{CR}$-equations
- Compute $=_G$ equations
  - $\mathcal{C}_G = gather(\{\langle g(S), \{min\, E\}\rangle \mid \langle S, E\rangle \in \mathcal{C}_{CR}\})$
  - print the induced $=_G$-equations

Discussion on the results

+ Summary

- Fix-point semantic characterization:
  - ✔ models the typical features of F/FL languages
  - ✘ does not handle H.O. and Residuation
  - ✔ goal-independent & "condensed"
  - ✔ fully-abstract w.r.t. computed result behavior
- Applications
  - Static Analysis
  - Abstract Debugging
  - Automatic Synthesis of Specifications

+ Future work

- applying this techniques on more interesting abstract domains
- extend our results to Higher-Order and Residuation

Research activity

Collaborations:

+ I've been invited for CHR working-week in Ulm (Germany)
+ Collaborated with the ELP group at Universidad Politcnica de Valencia (Spain)

Publications

G. Bacci and M. Comini. *Abstract Diagnosis of First Order Functional Logic Programs*. In M. Alpuente, editor, *Logic-based Program Synthesis and Transformation, 20th International Symposium*, volume 6564 of LNCS, 215–233, Berlin, 2011. Springer-Verlag.

G. Bacci, M. Comini, M. A. Feliú, and A. Villanueva. *Automatic Synthesis of Specifications for Curry Programs*. In *Proc. of Logic-based Program Synthesis and Transformation, 21th International Symposium*.

Technical Reports:

G. Bacci and M. Comini. *A Compact Goal-Independent Bottom-Up Fixpoint Modeling of the Behaviour of First Order Curry*. Technical Report DIMI-UD/06/2010/RR

G. Bacci and M. Comini. *A Fully-Abstract Compact Goal-Independent Bottom-Up Fixpoint Modeling of the Behaviour of First Order Curry*. Technical Report DIMI-UD/02/2012/RR,

G. Bacci and M. Comini. *Abstract Diagnosis of First Order Functional Logic Programs*. Technical Report DIMI-UD/03/2012/RR, (journal version)