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

\[
\begin{align*}
\text{nested expressions} & \quad \text{higher-order features} \\
\text{functional paradigm} & \\
\text{lazy evaluation} &
\end{align*}
\]

Operational mechanism:

\[\text{REWWRITING}\]

sub-expressions are rewritten according to program rules
Functional Logic Paradigm

+ nested expressions
+ higher-order features
+ lazy evaluation

+ logical variables
+ partial data structures
+ built-it search

Operational mechanism:

\[ \text{VARIABLE INSTANTIATION} + \text{REWRITING} = \text{NARROWING} \]

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

+ nested expressions
+ higher-order features
+ lazy evaluation
+ logical variables
+ partial data structures
+ built-it search

\[
\begin{align*}
\text{VARIABLE INSTANTIATION} &+ \text{REWRITING} = \text{NARROWING} \\
\text{variables are instantiated so that sub-expressions can be rewritten according to program rules}
\end{align*}
\]
equation solving & built-in search:

\[ 0 + x = x \quad 0 \leq y = \text{True} \]
\[ (S \, x) + y = S \, (x + y) \quad (S \, x) \leq 0 = \text{False} \]
\[ \text{double} \, x = x + x \quad (S \, x) \leq (S \, y) = x \leq y \]

the goal \((x + x) \leq 0\) where \(x\) free returns 2 solutions,
namely \(\{x \rightarrow 0\}\) True and \(\{x \rightarrow S \, x’\}\) False

non-deterministic operations:

overlapping rules are allowed \(\Rightarrow\) non-confluent programs

\[
\text{coin} = 0
\]
\[
\text{coin} = S \, 0
\]

\text{coin} returns 2 solutions, namely \(\{\}\) 0 and \(\{\}\) S 0
equation solving & built-in search:

\[
\begin{align*}
0 + x &= x & 0 &\leq y &= True \\
(S\ x) + y &= S\ (x + y) & (S\ x) &\leq 0 &= False \\
double\ x &= x + x & (S\ x) &\leq (S\ y) = x &\leq y
\end{align*}
\]

the goal \((x + x) \leq 0\) where \(x\) free returns 2 solutions, namely \(\{x \rightarrow 0\}\) True and \(\{x \rightarrow S\ x'\}\) False

non-deterministic operations:

overlapping rules are allowed \(\Rightarrow\) non-confluent programs

\[
\begin{align*}
\text{coin} &= 0 \\
\text{coin} &= S\ 0 \\
\text{coin} \text{ returns 2 solutions, namely } &\{\} \ 0 \text{ and } \{\} \ S\ 0
\end{align*}
\]
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:

\begin{align*}
\text{coin} &= 0 \\
\text{double } x &= x + x \\
\text{coin} &= S 
\end{align*}

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

\begin{align*}
\text{double } \text{coin} &\Rightarrow \text{coin} + \text{coin} \Rightarrow 0 + \text{coin} \Rightarrow \text{coin} \Rightarrow S 
\end{align*}

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

\begin{align*}
\text{double } \text{coin} &\Rightarrow \text{coin} + \text{coin} \Rightarrow 0 + 0 \Rightarrow 0 
\end{align*}

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

\[
\text{coin} = 0 \quad \text{double } x = x + x
\]

\[
\text{coin} = S\ 0
\]

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

\[
\text{double coin} \Rightarrow \text{coin} + \text{coin} \Rightarrow 0 + \text{coin} \Rightarrow \text{coin} \Rightarrow S\ 0
\]

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

\[
\text{double coin} \Rightarrow \text{coin} + \text{coin} \Rightarrow 0 + 0 \Rightarrow 0
\]
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:

\[
\begin{align*}
\text{coin} & = 0 \\
\text{coin} & = S \ 0 \\
\end{align*}
\]

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

\[
\begin{align*}
\text{double coin} & \Rightarrow \text{coin} + \text{coin} \Rightarrow 0 + \text{coin} \Rightarrow \text{coin} \Rightarrow S \ 0 \\
\end{align*}
\]

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

\[
\begin{align*}
\text{double coin} & \Rightarrow \text{coin} + \text{coin} \Rightarrow 0 + 0 \Rightarrow 0 \\
\end{align*}
\]
A subtle aspect of nondeterministic operations is their treatment if they are passed as arguments:

\[
\begin{align*}
\text{coin} &= 0 \\
\text{coin} &= S\ 0
\end{align*}
\]

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

\[
\begin{align*}
\text{double } \text{coin} &\Rightarrow \text{coin} + \text{coin} &\Rightarrow 0 + \text{coin} &\Rightarrow \text{coin} &\Rightarrow S\ 0
\end{align*}
\]

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

\[
\begin{align*}
\text{double } \text{coin} &\Rightarrow \text{coin} + \text{coin} &\Rightarrow 0 + 0 &\Rightarrow 0
\end{align*}
\]

sharing
A semantics adequate w.r.t. an observation $\phi$

Requirements:

- fix-point characterization (i.e., $F[P] := \text{lfp } P[P]$)
- goal-independent & “condensed”
- fully-abstract w.r.t. a behavioral observation $\phi$

Full-abstraction (EAGER languages):

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

Full-abstraction (LAZY languages):

- $F[P_1] = F[P_2] \iff \forall Q \in \cup \mathbb{P}^2. B^{\phi}[P_1 \cup Q] = B^{\phi}[P_2 \cup Q]$
A semantics adequate w.r.t. an observation $\phi$

Requirements:

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

Full-abstraction (EAGER languages):

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

Full-abstraction (LAZY languages):

+ $\mathcal{F}[P_1] = \mathcal{F}[P_2] \iff \forall Q \in UP_{\Sigma}'. B^\phi[P_1 \cup Q] = B^\phi[P_2 \cup Q]$

using programs

can only define new operations
Computed Results Behavior

Computed result behavior of programs:

\[ B^{cr}[P](e_0) := \left\{ (\sigma_1 \cdots \sigma_n) \upharpoonright_{e_0} \cdot e_n \mid e_0 \xrightarrow{\sigma_1}{p_1} \cdots \xrightarrow{\sigma_n}{p_n} e_n, e_n \in \mathcal{T}(C, V) \right\} \]

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

\[
f(x) = S(g(x)) \quad f(Sx) = S0
\]

\[
g(Sx) = 0 \quad g(Sx) = 0
\]

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

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

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]

\[ \mathcal{P}[P] \]

\[ (\mathcal{C}, \sqsubseteq) \]

\[ \alpha \]

\[ (\mathcal{A}, \leq) \]

\[ \mathcal{P}^\alpha[P] \]

Results from the A.I. theory:

\[ \mathcal{P}^\alpha[P] := \alpha \circ \mathcal{P}[P] \circ \gamma \]

\[ \mathcal{F}^\alpha[P] := \text{lfp} \mathcal{P}^\alpha[P] \]

\[ \alpha(\mathcal{F}[P]) \leq \mathcal{F}^\alpha[P] \]

\[ \alpha \text{ is precise } \implies \mathcal{F}^\alpha \text{ is a standard semantics} \]
Results from the A.I. theory:

- $\mathcal{P}^\alpha[P] := \alpha \circ \mathcal{P}[P] \circ \gamma$
- $\mathcal{F}^\alpha[P] := \text{lfp } \mathcal{P}^\alpha[P]$
- $\alpha(\mathcal{F}[P]) \leq \mathcal{F}^\alpha[P]$
- $\alpha$ is precise $\implies \mathcal{F}^\alpha$ is a standard semantics
We started from a (very) concrete semantics modeling the small-step behavior ("trace semantics")

\[
P[P] : \text{WSST}^{\text{MGC}} \rightarrow \text{WSST}^{\text{MGC}}
\]

\[
\mathcal{F}[P] = \text{lfp} \ P[P]
\]

**Theorem**

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

where \( \mathcal{E} \) is the semantic evaluation function
We started from a (very) concrete semantics modeling the small-step behavior ("trace semantics")

\[
\mathcal{P}[P] : \text{WSST}^{\text{MGC}} \rightarrow \text{WSST}^{\text{MGC}}
\]

\[
\mathcal{F}[P] = \text{lfp} \mathcal{P}[P]
\]

**Theorem**

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

where \( \mathcal{E} \) is the semantic evaluation function
We started from a (very) concrete semantics modeling the small-step behavior (“trace semantics”)

\[ \mathcal{P}[P] : \text{WSST}^{\text{MGC}} \rightarrow \text{WSST}^{\text{MGC}} \]

\[ \mathcal{F}[P] = \text{lfp} \mathcal{P}[P] \]

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

where \( \mathcal{E} \) is the semantic evaluation function.
We started from a (very) concrete semantics modeling the small-step behavior ("trace semantics")

\[ \mathcal{P}[P] : WSST_{\text{MGC}} \rightarrow WSST_{\text{MGC}} \]

\[ \mathcal{F}[P] = \text{lfp } \mathcal{P}[P] \]

**Theorem**

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

where \( \mathcal{E} \) is the semantic evaluation function
...then, we proceed by successive abstractions

\[ (\text{WSST}, \sqsubseteq) \xleftarrow{\partial^\gamma} (\text{ERT}, \preceq) \xrightarrow{\zeta^\gamma} (\text{WERS}, \prec) \]
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.

\[
\begin{align*}
f(x, g(y)) & \xrightarrow{\{x/A\}} f(A, g(y)) \xrightarrow{\xi} f(A, B) \xrightarrow{\xi} C(h(z)) \xrightarrow{\{z/B\}} C(A) \\
\end{align*}
\]
Evolving Result Abstraction

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

\[
\begin{align*}
  f(x, g(y)) & \xrightarrow{\{x/A\}} f(A, g(y)) \xrightarrow{\varepsilon} f(A, B) \xrightarrow{\varepsilon} C(h(z)) \xrightarrow{\{z/B\}} C(A)
\end{align*}
\]
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

\[
\begin{align*}
\varepsilon \cdot \varrho & \xrightarrow{\varrho} \{x/A\} \cdot C(\varrho_1) \xrightarrow{\varrho_1} \{x/A\} \cdot C(A) \\
\end{align*}
\]
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

\[
\begin{align*}
  f(x, g(y)) \xrightarrow{\{x/A\}} f(A, g(y)) \xrightarrow{\xi} f(A, B) \xrightarrow{\xi} C(h(z)) \xrightarrow{\{z/B\}} C(A)
\end{align*}
\]
The **ERT** (Evolving Result Trees) domain: \( \text{ERT} := \partial(\text{WSST}) \)

\[
\varepsilon \cdot \varrho \xrightarrow{\varrho} \{x/Z\} \cdot y \\
\varepsilon \cdot \varrho \xrightarrow{\varrho} \{x/S(x_1)\} \cdot S(\varrho_1) \\
\varepsilon \cdot \varrho \xrightarrow{\varrho_1} \{x/S(Z)\} \cdot S(y) \\
\varepsilon \cdot \varrho \xrightarrow{\varrho_1} \{x/S(S(x_2))\} \cdot S(S(\varrho_2))
\]

infinite depth
Evolving Result semantics

\[(\text{WSST}, \sqsubseteq) \xleftarrow{\partial^{\gamma}} (\text{ERT}, \lhd) \xrightarrow{\zeta^{\gamma}} (\text{WERS}, \hat{\zeta})\]

The **ERT** (Evolving Result Trees) domain: \(\text{ERT} := \partial(\text{WSST})\)

```
\[\varepsilon \cdot \emptyset \rightarrow \{x/\emptyset, y/S(y_1)\} \cdot \text{True}\]
\[\emptyset \rightarrow \{x/S(\emptyset), y/S(S(y_1))\} \cdot \text{True}\]
\[\vdots\]
\[\emptyset \rightarrow \{x/S(S(x_1)), y/S(\emptyset)\} \cdot \text{False}\]
\[\emptyset \rightarrow \{x/S(x_1), y/\emptyset\} \cdot \text{False}\]
```

infinite width
Evolving Result semantics

\[(\text{WSST}, \sqsubseteq) \xleftrightarrow{\partial^\gamma} (\text{ERT}, \preceq) \xleftrightarrow{\zeta^\gamma} (\text{WERS}, \approx)\]

Induced optimal immediate consequence operator

\[\mathcal{P}^\partial[P]: \text{ERT}^{\text{MGC}} \rightarrow \text{ERT}^{\text{MGC}}\]

\[\mathcal{P}^\partial[P]_{\mathcal{I}^\partial} := (\partial \circ \mathcal{P}[P] \circ \partial^\gamma)(\mathcal{I}^\partial)\]

\[= \lambda f(\overrightarrow{x_n}). \bigg\{ \varepsilon \cdot \varrho \xrightarrow{\varrho} \mathcal{E}^\partial[r]_{\mathcal{I}^\partial}\{\overrightarrow{x_n}/\overrightarrow{t_n}\} \bigg| f(t) \rightarrow r \in P \bigg\}\]

Evaluation function over ERT

\[\mathcal{E}^\partial[x]_{\mathcal{I}^\partial}^{\sigma} := \sigma \cdot x\]

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

Theorem

\[\mathcal{F}^\partial[P] = \partial(\mathcal{F}[P])\]
Evolving Result semantics

$$(\text{WSST}, \sqsubseteq) \xrightarrow{\partial^\gamma} (\text{ERT}, \preceq) \xrightarrow{\zeta^\gamma} (\text{WERS}, \simeq)$$

**Theorem (correctness)**

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

The converse implication doesn’t hold

**Counterexample**

Consider the programs $P_1$ and $P_2$

\[
\begin{align*}
f \ x &= A \ x \\
f \ x &= \text{id} \ (A \ (\text{id} \ x))
\end{align*}
\]

\[
\begin{align*}
\mathcal{F}^\partial[P_1](f(x)) &= \varepsilon \cdot \varrho \xrightarrow{\varrho} \varepsilon \cdot A(x) \quad \text{whereas} \\
\mathcal{F}^\partial[P_2](f(x)) &= \varepsilon \cdot \varrho \xrightarrow{\varrho_1} \varepsilon \cdot A(\varrho_1) \xrightarrow{\varrho_1} \varepsilon \cdot A(x).
\end{align*}
\]

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

\[(\text{WSST}, \sqsubseteq) \xleftrightarrow{\partial^\gamma} (\text{ERT}, \preceq) \xleftrightarrow{\zeta^\gamma} (\text{WERS}, \preceq)\]

**Theorem (correctness)**

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

The converse implication doesn’t hold

**Counterexample**

Consider the programs \(P_1\) and \(P_2\)

\[
\begin{align*}
f \ x &= A \ x \\
f \ x &= \text{id} \ (A \ (\text{id} \ x))
\end{align*}
\]

\[
\begin{align*}
\mathcal{F}^{\partial}[P_1](f(x)) &= \varepsilon \cdot \varrho \xrightarrow{\varrho} \varepsilon \cdot A(x) \text{ whereas} \\
\mathcal{F}^{\partial}[P_2](f(x)) &= \varepsilon \cdot \varrho \xrightarrow{\varrho_1} \varepsilon \cdot A(\varrho_1) \xrightarrow{\varrho_1} \varepsilon \cdot A(x).
\end{align*}
\]

**Only when a substitution changes there is a visible effect in the behavior**
**IDEA:** combine together all partial computed results that refer to the same substitution and lead to the same partial result

**conceise representation:** we denotes with $\sigma \cdot s_1 \sim s_2$ the set of partial computed results $\sigma \cdot s$ where $s_1 \preceq s \preceq s_2$. 

\[
\begin{align*}
\varepsilon \cdot \varrho &\xrightarrow{\varrho} \varepsilon \cdot A(x) \\
\varepsilon \cdot \varrho_1 &\xrightarrow{\varrho_1} \varepsilon \cdot A(\varrho_2) \xrightarrow{\varrho_2} \varepsilon \cdot A(x)
\end{align*}
\]
**Weakly Evolving Abstraction**

\[(\text{WSST}, \sqsubseteq) \leftrightarrow_{\frac{\partial}{\partial}} (\text{ERT}, \preceq) \leftrightarrow_{\frac{\zeta}{\zeta}} (\text{WERS}, \hat{\preceq})\]

**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 \( \sigma \cdot s_1 \preceq s \preceq s_2 \) the set of partial computed results \( \sigma \cdot s \) where \( s_1 \preceq s \preceq s_2 \).

\[
\varepsilon \cdot \varrho \xrightarrow{\varrho} \varepsilon \cdot A(x) \quad \varepsilon \cdot \varrho_1 \xrightarrow{\varrho_1} \varepsilon \cdot A(\varrho_2) \xrightarrow{\varrho_2} \varepsilon \cdot A(x)
\]
Weakly Evolving semantics

\( (WSST, \sqsubseteq) \xleftrightarrow{\partial^\gamma} (ERT, \preceq) \xleftrightarrow{\zeta^\gamma} (WERS, \hat{<}) \)

Induced immediate consequence operator

\[ \mathcal{P}^\nu[P] : WERS^{MGC} \rightarrow WERS^{MGC} \]

\[ \mathcal{P}^\nu[P] I^\nu = \lambda f(\overrightarrow{x}_n). \bigwedge \left\{ E^\nu[r] \{ \overrightarrow{x}_n/t_n \} \mid f(t) \rightarrow r \in P \right\} \]

Evaluation function over \( WERS \)

\[ E^\nu[x]_{I^\emptyset} := \sigma \cdot \varphi - x \]

\[ E^\nu[\varphi(t_n)]_{I^\emptyset} := I^\nu(\varphi(\overrightarrow{y}_n))[y_1/E^\nu[t_1]_{I^\emptyset}] \ldots [y_n/E^\nu[t_n]_{I^\emptyset}] \]

Theorem (full-abstraction)

1. \( \nu(E[e]_I) = E^\nu[e]_{\nu(I)} \)
2. \( \forall P \ F^\nu[P] = \nu(F[P]) \)
3. \( F^\nu[P_1] = F^\nu[P_2] \iff \forall Q \in \mathbb{UP}^{\Sigma'} \cup P_1 \cup Q \cup B^{cr}[P_1 \cup Q] = B^{cr}[P_2 \cup Q] \)
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.

$$
\begin{align*}
    e_0 \xrightarrow{p_1} \ldots \xrightarrow{p_n} e_n \text{ using } P & \iff e_0 \xrightarrow{\varepsilon} \ldots \xrightarrow{\varepsilon} e_n \text{ using } Cnv(P)
\end{align*}
$$

... all results apply to Haskell as well.
Abstraction Framework:

+ consider a **true** abstraction $\alpha$
+ $(\text{WERS}, \lesssim) \xrightarrow{\gamma} (\mathcal{A}, \leq)$
+ abstract semantics $\mathcal{F}_\alpha$ can be effectively computed

**Proposed case studies:** $\text{depth}(k)$ 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

\[
\begin{align*}
\Gamma_{\varrho}(S) & := \bigvee \{ \Gamma(\sigma\{\varrho/\nu\}) \mid \sigma \cdot t - \nu \in S, \ \nu \in \mathbb{T}(\mathcal{C}, \mathcal{V}) \} \quad \text{(WERS)} \\
\Gamma(\vartheta) & := \bigwedge_{y/t \in \vartheta} (y \leftrightarrow (\bigwedge_{x \in \text{var}(t)} x)) \quad \text{(substitutions)}
\end{align*}
\]

Examples:

\[
\begin{align*}
& x + y \triangleright \varrho \iff x \land (\varrho \leftrightarrow y) \\
& \quad \text{first argument ground, and result ground iff second argument ground} \\
& x \leq y \triangleright \varrho \iff \varrho \land (x \lor y) \\
& \quad \text{result ground, and at least one argument ground}
\end{align*}
\]
Abstract semantic functions

Induced optimal immediate consequence operator

\[ \mathcal{P}^{gr}[P] := \alpha_\Gamma \circ \mathcal{P}^\nu \circ \gamma_\Gamma \]

\[ = \lambda f(\vec{x}_n) \triangleright_\nu \phi \cdot \bigvee_{f(\vec{t}_n) \rightarrow r \in P} (\Gamma(\{\vec{x}_n / \vec{t}_n\}) \land \mathcal{E}^{gr}[r \triangleright_\nu \phi]_{\mathcal{I}^{gr}})(\vec{x}_n, \phi) \]

Evaluation function over \( \mathcal{G} \)

\[ \mathcal{E}^{gr}[x \triangleright_\nu \phi]_{\mathcal{I}^{gr}} := \phi \leftrightarrow x \]

\[ \mathcal{E}^{gr}[\phi(\vec{t}_n) \triangleright_\nu \phi]_{\mathcal{I}^{gr}} := \mathcal{I}^{gr}(\phi(\vec{q}_n) \triangleright_\nu \phi) \land \bigwedge_{i=1}^n \Phi_i \quad \vec{q}_n \text{ fresh} \]

where

\[ \Phi_i := \begin{cases} \mathcal{E}^{gr}[t_i \triangleright_\nu \phi_i]_{\mathcal{I}^{gr}} & \text{if } \mathcal{I}^{gr}(\phi(\vec{q}_n) \triangleright_\nu \phi) \leq (\phi \rightarrow \phi_i) \text{ or } t_i \in \mathcal{T}(\mathcal{C}, \mathcal{V}) \\ true & \text{otherwise} \end{cases} \]
Program:

\[
\begin{align*}
[] ++ ys & = ys \\
(x:xs) ++ ys & = x : (xs ++ ys)
\end{align*}
\]

Analysis session:

\[
\begin{align*}
P^{gr}[P] \uparrow 0 & = \begin{cases} 
xs ++ ys \triangleright \varrho & \mapsto false \\
\end{cases} \\
P^{gr}[P] \uparrow 1 & = \begin{cases} 
xs ++ ys \triangleright \varrho & \mapsto xs \land (\varrho \leftrightarrow ys) \\
\end{cases} \\
P^{gr}[P] \uparrow 2 & = \begin{cases} 
xs ++ ys \triangleright \varrho & \mapsto \varrho \leftrightarrow (xs \land ys) \\
\end{cases} \\
P^{gr}[P] \uparrow 3 & = P^{gr}[P] \uparrow 2 = P^{gr}[P] \uparrow \omega
\end{align*}
\]

...running tool
Program:

\( [\ ] \,+\,\,ys = ys \)

\( (x:xs) \,+\,\,ys = x : (xs++ys) \)

Analysis session:

\[ \mathcal{P}^{gr} [P] \uparrow 0 = \begin{cases} \text{false} \end{cases} \]

\[ \mathcal{P}^{gr} [P] \uparrow 1 = \begin{cases} xs \,+\,\,ys \downarrow \,\,\varrho \,\,\mapsto \,\,xs \wedge (\varrho \leftrightarrow ys) \end{cases} \]

\[ \mathcal{P}^{gr} [P] \uparrow 2 = \begin{cases} \varrho \,\,\mapsto \,\,(xs \wedge ys) \end{cases} \]

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

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

...running tool
Application: Abstract Diagnosis

Automatic Debugging

**Input:** program $P$ + specification $S$

**Goal:** automatically locate bugs in $P$

in general it is **undecidable**

How to deal with this problem?

+ **Declarative Debugging**  $\implies$ partial inspection of the symptomatic *computation tree*

+ **Abstract Diagnosis**  $\implies$ use a correct approximation of the semantics which is finitely representable
Application: Abstract Diagnosis

Automatic Debugging

**Input:** program $P$ + specification $S$

**Goal:** automatically locate bugs in $P$

in general it is **undecidable**

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

Automatic Debugging

**Input:** program $P$ + specification $S$

**Goal:** automatically locate bugs in $P$

In general it is **undecidable**

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 $P$
The main idea

(Abstract Diagnosis)

\((\mathbb{C}, \subseteq, \cup, \cap, \bot_{\mathbb{C}}, \top_{\mathbb{C}})\)
Complete Lattice

\((\mathbb{A}, \leq, \lor, \land, \bot_{\mathbb{A}}, \top_{\mathbb{A}})\)
Noetherian Complete Lattice

\(S\)

\(\mathcal{F}[P]\)

\(\alpha\)

\(S^\alpha\)

\(\alpha(\mathcal{F}[P])\)
The main idea

\((\mathcal{C}, \subseteq, \sqcup, \sqcap, \bot_{\mathcal{C}}, \top_{\mathcal{C}})\)  
Completeness Lattice

\((\mathbb{A}, \leq, \lor, \land, \bot_{\mathbb{A}}, \top_{\mathbb{A}})\)  
Noetherian Complete Lattice

\(\mathcal{F}[\mathbb{P}]\)

\(\mathcal{S}\)

\(\alpha\)

\(\mathcal{F}^{\alpha}[\mathbb{P}]\)
Let $P$ be a program and $\alpha$ a property

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

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

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

\[\Downarrow\]

Declarative Diagnosis **cannot** reveal all errors **simultaneously**
Let $P$ be a program and $\alpha$ a property

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

**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:

\[ \mathcal{P}^\alpha[P]S^\alpha \leq S^\alpha \]

- \( e \leq \mathcal{P}^\alpha[\{l \rightarrow r\}]S^\alpha \) and \( e \not\in S^\alpha \) (abstractly incorrect rule)

- \( e \land \mathcal{P}^\alpha[P]S^\alpha = \bot \) and \( e \leq S^\alpha \) (abstractly uncovered elem.)
Abstract Diagnosis Framework

Based on abstract version of Park’s Induction Principle:

\[ P^\alpha [P] S^\alpha \leq S^\alpha \]

using \( S^\alpha \),
\[ l \rightarrow r \]
produces \( e \)…

+ \( e \leq P^\alpha [\{ l \rightarrow r \}] S^\alpha \) and \( e \notin S^\alpha \) (abstractly incorrect rule)

+ \( e \land P^\alpha [P] S^\alpha = \bot \) and \( e \leq S^\alpha \) (abstractly uncovered elem.)
Abstract Diagnosis Framework

Based on abstract version of Park’s Induction Principle:

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

+ \( e \leq \mathcal{P}^\alpha [\{l \rightarrow r\}]_{S^\alpha} \) and \( e \notin S^\alpha \) (abstractly incorrect rule)

+ \( e \wedge \mathcal{P}^\alpha [P]_{S^\alpha} = \bot \wedge \) and \( e \leq S^\alpha \) (abstractly uncovered elem.)

using \( S^\alpha \),

\( l \rightarrow r \)

produces \( e \). . .

. . . but \( e \) was not expected by \( S^\alpha \)
Abstract Diagnosis Framework

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

\[ P^\alpha[P]_{S^\alpha} \leq S^\alpha \]

- using \( S^\alpha \), \( l \rightarrow r \) produces \( e \)...

- \( e \leq P^\alpha[l \rightarrow r]_{S^\alpha} \) and \( e \not\in S^\alpha \) (abstractly incorrect rule)

- but \( e \) was not expected by \( S^\alpha \)

- using \( S^\alpha \), \( P \) can’t produce \( e \)...

- \( e \wedge P^\alpha[P]_{S^\alpha} = \bot_A \) and \( e \leq S^\alpha \) (abstractly uncovered elem.)
Abstract Diagnosis Framework

Based on abstract version of Park’s Induction Principle:

\[ \mathcal{P}_\alpha[P] \leq S_\alpha \]

using \( S_\alpha \), \( l \to r \) produces \( e \)...

\[ + \ e \leq \mathcal{P}_\alpha[l \to r] S_\alpha \text{ and } e \nsubseteq S_\alpha \] (abstractly incorrect rule)

\[ \text{using } S_\alpha, \ P \text{ can’t produce } e \ldots \]

\[ + \ e \land \mathcal{P}_\alpha[P] S_\alpha = \bot_A \text{ and } e \leq S_\alpha \] (abstractly uncovered elem.)

\[ \ldots \text{but } e \text{ was not expected by } S_\alpha \]

\[ \ldots \text{but } e \text{ was expected by } S_\alpha \]
Abstract Diagnosis Framework

Based on abstract version of Park’s Induction Principle:

\[ P^\alpha \llbracket P \rrbracket_{S^\alpha} \leq S^\alpha \]

...but \( e \) was not expected by \( S^\alpha \)

\[ e \leq P^\alpha \llbracket \{ l \rightarrow r \} \rrbracket_{S^\alpha} \text{ and } e \notin S^\alpha \]

(abstractly incorrect rule)

\[ e \land P^\alpha \llbracket P \rrbracket_{S^\alpha} = \bot_A \text{ and } e \leq S^\alpha \]

(abstractly uncovered elem.)

Pros:  
+ Static test (requires just one \( P^\alpha \llbracket P \rrbracket \) step on \( S^\alpha \))
+ reveal all abstract errors regardless of symptoms interference

Cons:  
+ imprecision of \( \alpha \) can lead to false positives:
Case study: $\textit{depth}(k)$

**Program:** $R: \text{from } n = n : \text{from } n$

**Specification:** with $\kappa = 3$

$$S^\kappa := \left\{ \text{from}(n) \mapsto \{ \varepsilon \cdot \varrho - n : S(\hat{x}_1) : \hat{x}_2 : \hat{x}_3 \} \right\}$$

We detect that rule $R$ is abstractly incorrect since

$$\mathcal{P}^\kappa \left[ \{ R \} \right]_{S^\kappa} = \left\{ \text{from}(n) \mapsto \{ \varepsilon \cdot \varrho - n : n : \hat{x}_1 : \hat{x}_2 \} \right\} \nsubseteq S^\kappa$$
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 $\Rightarrow$ 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 $\Rightarrow$ assumes that the source code of the program is available.
  - ✗ language-dependent
  - ✓ the inference can be semantic-based $\Rightarrow$ the inferred equations can be correct
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 $C[\ ]$

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

**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.
**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

or

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

**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.

$$x =_G \lnot(\lnot x)$$
and $x$ (and $y$ $z) =_G$ and ($and x$ $y$) $z$

not (or $x$ $y) =_G$ 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.

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

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

(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:

A set of equations $e_1 =\{C,CR,G\} e_2$ where $e_1, e_2 \in \mathcal{T}(\Sigma^r, \mathcal{V})$
**Classification** = “a set of pairs of the form \( \langle S, \{ e_1, \ldots, e_n \} \rangle \)”
Inference Process

Program → Compute (abstract) Semantics → Generation of \( =_c \) classification → Equations generation → Transformation of the Semantics → Spec.

- \( \text{compute } \mathcal{F}^\alpha[P] \)
- \( S_f(\vec{x}_n) := \mathcal{F}^\alpha[P](f(\vec{x}_n)) \) for every \( f \in \Sigma^r \)
- \( C_0 := \{ \langle S_f(\vec{x}_n), \{ f(\vec{x}_n) \} \rangle \mid f \in \Sigma^r \} \)
**Inference Process**

**Program**

**API: \( \Sigma^r \)**

**max_size**

---

**Inference Process**

- **Compute (abstract) Semantics**
- **Generation of \( =_C \) classification**
- **Equations generation**
- **Transformation of the Semantics**
- **Spec.**

---

**Inference Process**

**Program**

**API: \( \Sigma^r \)**

**max_size**

---

**Inference Process**

1. **iterate** `max_size` times
   1. take \( f \in \Sigma^r \) and \( \langle S_1, E_1 \rangle, \ldots, \langle S_k, E_k \rangle \in \mathcal{C}_h \)
   2. compute \( S = S_{f(x_n)}[x_1/S_1] \ldots [x_n/S_n] \)
   3. update \( \mathcal{C}_h \) inserting \( \langle S, \{ f(e_n) \} \rangle \) where \( e_i = \text{min } E_i \)

2. \( \mathcal{C} := \mathcal{C}_{max_size} \) 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

Program

API: $\Sigma^r$

Compute (abstract) Semantics

Generation of $=_{C}$ classification

Equations generation

Transformation of the Semantics

Spec.

Inference Process

Program

API: $\Sigma^r$

Compute (abstract) Semantics

Generation of $=_{C}$ classification

Equations generation

Transformation of the Semantics

Spec.

associated to every $f(\overrightarrow{e_n})$

for any $\overrightarrow{e_n} \in E_1 \times \cdots \times E_n$

+ iterate $\text{max\_size}$

+ take $f \in \Sigma^r$ and $\langle S_1, E_1 \rangle, \ldots, \langle S_k, E_k \rangle \in C_h$

+ compute $S = S_f(\overrightarrow{x_n})[x_1/S_1] \cdots[x_n/S_n]$

+ update $C_h$ inserting $\langle S, \{f(\overrightarrow{e_n})\} \rangle$ where $e_i = \min E_i$

+ $C := C_{\text{max\_size}}$ and print the equations $e_1 =_{C} \cdots =_{C} e_n$ for each $\langle S, \{e_1, \ldots, e_n\} \rangle \in C$. 

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

( Automatic Synthesis of Specifications )

Inference Process

<table>
<thead>
<tr>
<th>Compute (abstract) Semantics</th>
<th>Generation of $=_{c}$ classification</th>
<th>Equations generation</th>
<th>Transformation of the Semantics</th>
</tr>
</thead>
</table>

- Program
- API: $\Sigma^r$
- $max_{size}$

**Compute $=_{cr}$ equations**
- $C_{CR} = \text{gather}(\langle cr(S), \{min\ E\} \rangle | \langle S, E \rangle \in C)$
- Print the induced $=_{cr}$-equations

**Compute $=_{g}$ equations**
- $C_{G} = \text{gather}(\langle g(S), \{min\ E\} \rangle | \langle S, E \rangle \in C_{CR})$
- 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
Collaborations:

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


Technical Reports:

