A Bottom-Up Semantics for FLP

Giovanni Bacci, Marco Comini

6th October 2009
Outline

- Main features of Functional Logic Programs
- Operational Semantics
- Bottom-up Semantics Operator (examples)
- Abstract Semantics (examples)
- Conclusions & Future Works
Functional Logic Languages (FLP)

Are defined over a bipartite signature $\Sigma := C \cup D$

- **Overlapping Rules**

  \[
  R_1 : x \rightarrow y \rightarrow x \\
  R_2 : x \rightarrow y \rightarrow y \\
  \]

**Example:**

\[
R_3 : \text{reach}(x) \rightarrow x \rightarrow \text{reach}(\text{adj}(x)) \\
\text{where } \forall i, k (i, j_k) \in E. \text{adj}(i) \rightarrow j_1 \rightarrow \ldots \rightarrow j_n
\]
**Logical variables**

\[ R_1 : \text{True} \&\& y \rightarrow y \]
\[ R_2 : \text{path}(x) \rightarrow [x] \]
\[ R_3 : \text{path}(x) \rightarrow \text{adj}(x, y) \&\& (x : \text{path}(y)) \]
where \( \forall (i, j) \in E. \text{adj}(i, j) \rightarrow \text{True} \)

Narrowing:
\[ t \leadsto_{\sigma} s \text{ iff } \exists \sigma. \sigma(t) \rightarrow s \]

\[ \text{path}(1) \xrightarrow{\epsilon} [1] \]
\[ \xrightarrow{\epsilon} \]

\[ \text{adj}(1, y) \&\& (1 : \text{path}(y)) \]
\[ \xrightarrow{\{y/3\}} \]
\[ \xrightarrow{\{y/2\}} \]
\[ \vdash \text{True} \&\& (1 : \text{path}(2)) \]
\[ \xrightarrow{\epsilon} (1 : \text{path}(2)) \xrightarrow{\epsilon} [1, 2] \]
**Needed Narrowing Strategy** [Antoy & Hanus]

**Example:**

\[ R_1 : 0 + y \rightarrow y \]
\[ R_2 : s(x) + y \rightarrow s(x + y) \]

... running a goal

\[ y + y \leq x \rightarrow 0 \leq x \rightarrow True \]
\[ (\emptyset, y/0, R_1) \]
\[ (\{1\}, y/s(y_1), R_2) \]
\[ s(y_1 + s(y_1)) \leq x \]

\[ (\emptyset, x/s(x), R_5) \]
\[ (\emptyset, x/0, R_1) \]
\[ y_1 + s(y_1) \leq x_1 \rightarrow False \]

\[ \langle y + y \leq x, \{x/0, y/s(y_1)\} \rangle, False \]

\[ R_3 : 0 \leq y \rightarrow True \]
\[ R_4 : s(x) \leq 0 \rightarrow False \]
\[ R_5 : s(x) \leq s(y) \rightarrow x \leq y \]

**Definitional Trees:**

- **Left Tree:**
  - \[ 0 + x_2 \]
  - \[ s(x_1) + x_2 \]
  - **Right Tree:**
  - \[ y_1 \leq y_2 \]
  - \[ 0 \leq y_2 \]
  - **Bottom:**
    - \[ s(y_3) \leq y_2 \]
    - **Bottom:**
      - \[ s(y_3) \leq 0 \]
      - **Bottom:**
        - \[ s(y_3) \leq s(y_4) \]
        - **Bottom:**
          - \[ s(y_3) \leq s(y_4) \]
          - **Bottom:**
            - \[ s(y_3) \leq s(y_4) \]
            - **Bottom:**
              - \[ s(y_3) \leq 0 \]
              - **Bottom:**
                - \[ s(y_3) \leq s(y_4) \]
      - **Bottom:**
        - \[ s(y_3) \leq 0 \]
        - **Bottom:**
          - \[ s(y_3) \leq s(y_4) \]
      - **Bottom:**
        - \[ s(y_3) \leq 0 \]
        - **Bottom:**
          - \[ s(y_3) \leq s(y_4) \]
Toward a compositional Semantics

**Definition (Compositionality)**

\[
[s(t_1, \ldots, t_n)]_P = \text{op}_s([t_1]_P, \ldots, [t_n]_P) \quad \text{where} \quad s \in \Sigma = C \cup D
\]

A natural notion of semantics:

\[
[t]_P := \{\langle t, \sigma, v \rangle \mid t \xrightarrow{\sigma}^* v, \ v \text{ is a value}\}
\]

\[
[P] := \bigcup \{[t]_P \mid t \text{ is an expression}\}
\]

since, the semantics of the previous program is:

\[
[P] = \{\langle x_1 + x_2, \{x_1/s^n(0)\}, s^n(x_2) \rangle \mid n \geq 0\} \cup
\{\langle x_1 \leq x_2, \{x_1/s^n(0), x_2/s^n(x_2')\}, True \rangle \mid n \geq 0\}
\]

\[
\langle x_1 \leq x_2, \{x_1/s^{n+1}(x_1'), x_2/s^n(0)\}, False \rangle \mid n \geq 0
\]

\[
[y + y]_P = \{\langle y + y, \{y/s^n(0)\}, s^{2n}(0) \rangle \mid n \geq 0\}
\]

..but \( y + y \leq x \xrightarrow{\sigma}^* \{y/s(y_1), x/0\} \) \( False \)
A Fix-point Operator over Narrowing Trees

Let \( T_P : \mathbb{C} \rightarrow \mathbb{C} \) where \( \mathbb{C} := (PP \rightarrow NT)/\sim \)

\[ T_P(\mathcal{I}) := \lambda f(\vec{x}). \sum_{f(\vec{t}) \rightarrow r \in P} \langle f(\vec{x}); \{ [], \{ \vec{x}/\vec{t} \}, \mathcal{E}[[r]] \mathcal{I} \} \rangle \]

where \( \mathcal{E}[[\cdot]] \) is the evaluation function:

\[ \mathcal{E}[[x]] \mathcal{I} := (x; \emptyset) \]
\[ \mathcal{E}[[c(t_1, \ldots, t_n)]] \mathcal{I} := (c(x_1, \ldots, x_n); \emptyset) \bullet \Phi[x/\mathcal{E}[[t_i]] \mathcal{I}]_{i=1,\ldots,n} \]
\[ \mathcal{E}[[f(t_1, \ldots, t_n)]] \mathcal{I} := \mathcal{I}(f(x_1, \ldots, x_n)) \bullet \Phi[x/\mathcal{E}[[t_i]] \mathcal{I}]_{i=1,\ldots,n} \]

Theorem (Soundness & Completeness)

- \( \text{lfp}(T_P)(f(\vec{x})) =_{\mathbb{C}} \{ d \mid d \text{ is a derivation starting from } f(\vec{x}) \} /\sim \)
- \( \mathcal{E}[[t]]_{\text{lfp}(T_P)} =_{\mathbb{C}} \{ d \mid d \text{ is a derivation starting from } t \} /\sim \)
Toward a more abstract Semantics

The previous semantics is very precise ... even too much
We are able to distinguish too many programs:

$$\begin{align*}
&f \rightarrow g \\
g \rightarrow 0 & \quad \left[ P_1 \right] = \\
g \rightarrow 1
\end{align*}$$

$$\begin{align*}
&f \rightarrow 0 \\
f \rightarrow 1 & \\
g \rightarrow 0 \\
g \rightarrow 1
\end{align*}$$
The Context Tree Abstraction

Functions behaves as a black box

Inner steps are compressed

We just remember how the outermost context grows

Γ(\overline{x}) := \lambda f(\overline{x}). \Gamma(f(\overline{x}))
The Abstract Fix-point Operator

Using standard techniques from Abstract Interpretation*, we defined it just by means of \( \Gamma \) and \( T_P \):

\[
T_P^\alpha : A \rightarrow A \text{ where } A := (PP \rightarrow CT) / \sim
\]

\[
T_P^\alpha (I^\alpha) = \lambda f(\vec{x}). (\exists; S)
\]

where

\[
S = \{ \langle [], \{\vec{x}/t\}, \tilde{E} [r]_{I^\alpha} \rangle \mid f(\vec{x}) \rightarrow r \in P, \ r \text{ is } C \text{ rooted} \} \\
\{ \langle [], \{\vec{x}/t\} \circ \sigma, T \rangle \mid f(\vec{x}) \rightarrow r \in P, \ r \text{ is } D \text{ rooted} \}
\]

\[
\langle p, \sigma, T \rangle \in \text{steps}(\tilde{E} [r]_{I^\alpha})
\]

and \( \tilde{E} [\cdot]_{I^\alpha} \) is the abstract evaluation function.

**Lemma (Our abstraction is precise)**

\[
\forall I. \forall t. \alpha (E[t]_I) = \tilde{E} [t]_{\alpha(I)}
\]

*It is able to characterize the computed answer behavior*
Conclusions

- we characterize FLP semantics using a bottom-up construction
- the semantics is **goal-independent**
- we found a more abstract semantics which precisely describe the computed answer behavior
- compositionality gives us the possibility to
  - define incremental and modular analysis tools
  - ... as well as verification tools.
Future works on CHR

Gabrielli & Meo defined a fully abstract fix-point semantics for CHR and they concluded saying:

\[\text{ [...] it would be desirable to introduce in the semantics the minimum amount of information needed to obtain compositionality, while preserving correctness. In other words, it would be desirable to obtain a fully abstract semantics for data sufficient answers. [...]}\]

[Gabrielli & Meo]