Coinductive Functional Logic Programming

Ronald de Haan

TU Dresden

February 18, 2011
Outline

1. Background
   - Coinductive Logic Programming
   - Functional Logic Programming

2. Future work
   - Coinductive Functional Logic Programming
Outline

1 Background
   • Coinductive Logic Programming
     • Functional Logic Programming

2 Future work
   • Coinductive Functional Logic Programming
Intuitions behind CLP

- Logic programming works with inductively defined (finite) terms
- Coinductive logic programming computes with infinite (cyclic) terms
- The idea: use coinduction to finitely reason about infinite things
A coinductive logic program is a definite program with maximal co-Herbrand model declarative semantics.

- greatest fixed point interpretation

Operational semantics: co-SLD-resolution

- regular SLD-resolution steps
- “coinductive hypothesis” step:
  - if a goal Q is called, and Q unifies with a call made earlier, then Q succeeds
  - you can think of this as a circularity check

Operational semantics is sound and complete w.r.t. the declarative semantics [1]
Example

- Coinductive logic program

\[ \mathcal{P} = \{ \text{bit}(0). \]
\[ \quad \text{bit}(1). \]
\[ \quad \text{bitstream}([\text{H}|\text{T}]) :\!- \text{bit}(\text{H}), \text{bitstream}(\text{T}). \} \]

- The following query succeeds

\[ ?- X = [1,0,1,1|X], \text{bitstream}(X). \]
Advantages

- Declarative programming with infinite terms
  - Graphs with cycles
  - Automata over infinite strings
  - Bisimilarity
  - Infinite processes

- Different approaches to other areas possible
  - e.g., predicate answer set programming with a top-down approach [2]
Outline

1 Background
   - Coinductive Logic Programming
   - Functional Logic Programming

2 Future work
   - Coinductive Functional Logic Programming
Intuitions behind FLP

- Combining functional and logic programming
  - efficient execution principles of functional programming
  - flexibility of logic programming
- In the following, we look at (the principles behind) the language Curry in particular
Formal machinery

- Needed narrowing [3]
  - narrowing is “rewriting with binding”
  - every step consists of binding and reducing
  - this happens in a way that is optimal
    - no unnecessary steps,
    - shortest derivations,
    - minimal set of computed solutions
  - used narrowing strategy can be computed from a program

- Sound and complete for a large class of rewrite systems
Example

- Rewrite rules:
  
  \[ 0 \leq x \rightarrow true \]
  \[ (S \ x) \leq 0 \rightarrow false \]
  \[ (S \ x) \leq (S \ y) \rightarrow x \leq y \]

- One possible needed narrowing derivation (not the only one!):
  
  \[(S \ x) \leq (S \ (S \ y)) \leadsto_{r_3,\epsilon} x \leq (S \ y) \leadsto_{r_1,\{x \mapsto 0\}} true\]
Advantages

- (More) efficient execution of programs with free variables
- Can be (fairly) easily extended with
  - conditional rewrite rules
  - constraints
  - concurrency
- Integration of declarative paradigms
Outline

1 Background
   - Coinductive Logic Programming
   - Functional Logic Programming

2 Future work
   - Coinductive Functional Logic Programming
Intuitions

- The (current) idea: extend the needed narrowing approach with a circularity check, similar to the extension of LP to coinductive LP
- This would result in a bisimilarity relation, instead of (or next to) an equality relation
Formal machinery

- Proof theoretic framework for circular coinduction [4]
- ...

Suggestions?
Questions or suggestions?

- All suggestions are most welcome
- Don’t hesitate to contact me
  (Ronald.de_Haan@mailbox.tu-dresden.de)
References


