lp2cpp: A Tool For Compiling Stratified Logic Programs

B. Cuteri, A. F. De Rosis, F. Ricca
Department of Mathematics and Computer Science, University of Calabria, Italy
{cuteri,ricca}@mat.unical.it alessandrof.derosis@gmail.com

AI*IA 2017

UNIVERSITÀ DELLA CALABRIA

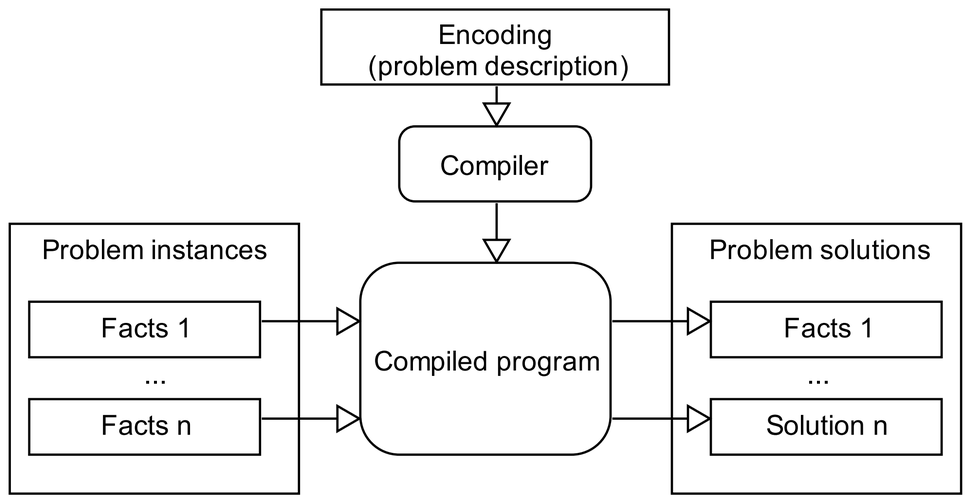
ABSTRACT

The evaluation of logic programs is traditionally implemented in monolithic systems that are general-purpose in the sense that they are able to process an entire class of programs. In this work, we follow a different approach; we present a compilation procedure that is able to generate a problem-specific executable implementation of a given (non-ground) logic program. Our implementation follows a bottom-up evaluation strategy. Moreover, we implemented such procedure into a C++ tool and we present an experimental analysis that shows the performance benefits that can be obtained by a compilation-based approach.

CONTEXT AND CONTRIBUTION

- **Datalog**
  - A simple, yet flexible logic language
- **Datalog Systems**
  - General-purpose approach
    - process an entire class of programs
  - Compilation-based approach
    - compile the program in a specialized procedure
- **A new compiler for stratified Datalog programs**
- **Experiment with well-know benchmarks**

COMPILER ARCHITECTURE



- Program parsing and pre-processing → compiler
- Dependency Graph construction → compiler
- Detect SCCs and identify program modules → compiler
- Generate a C++ program → compiler
- Evaluate modules by respecting dependencies → compiled program

EXPERIMENTS AND RESULTS

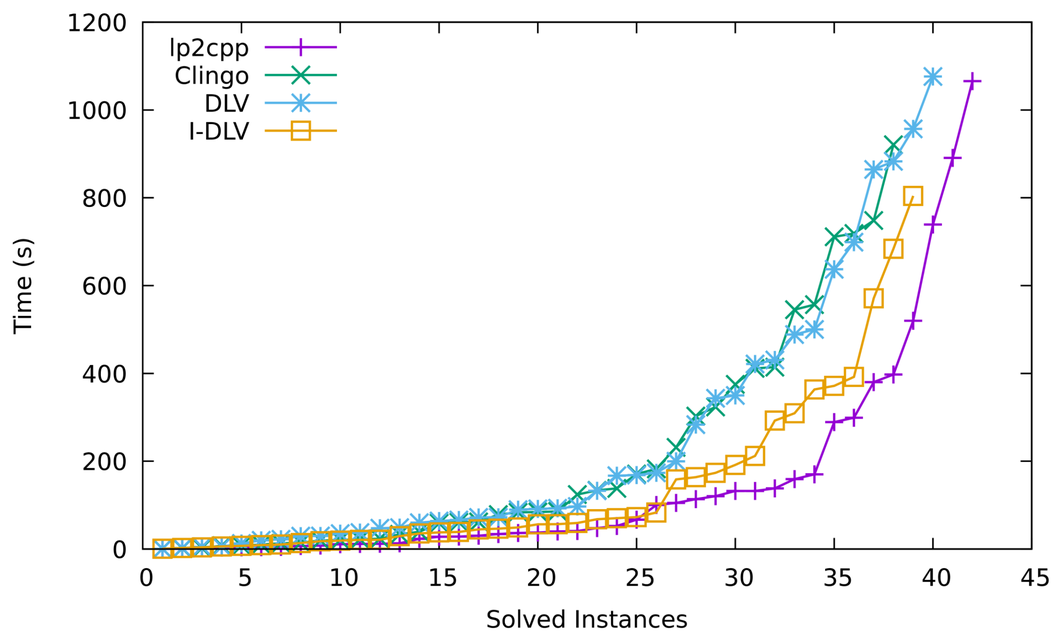


Figure 1: Overall performance: Cactus plot of the execution times reported for OpenRuleBench

		lp2cpp		Clingo		DLV		I-DLV	
Benchmark	**Inst.**	**Time**	**Sol.**	**Time**	**Sol.**	**Time**	**Sol.**	**Time**	**Sol.**
LargeJoins	20	**319**	**18**	677	15	578	16	637	15
Recursion	20	**337**	**19**	510	18	458	19	356	19
Stratified negation	5	**72**	**5**	195	5	237	5	76	5
Totals	45	**300**	**42**	545	38	484	40	444	39

Table 1: Average execution times in seconds and number of solved instances grouped by solvers and domains, best performance outlined in bold face.

- **The compiled approach performs well in all domains and in less time**

EXAMPLE OF COMPILED PROGRAM

Encoding

reaches(X,Y) :- edge(X,Y). (1)

reaches(X,Y) :- edge(X,Z), reaches(Z,Y). (2)

noReach(Y) :- vertex(Y), not reaches(2,Y). (3)

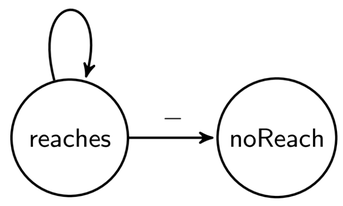


Figure 2: Dependency Graph of the reachability program example

Algorithm 1 Reachability compiled program

```
1:  //Predicate Sets declarations
2:  PredicateSet vertex
3:  PredicateSet edge
4:  PredicateSet reaches
5:  PredicateSet noreaches
6:
7:  //Auxiliary maps declarations
8:  AuxiliaryMap edge_1
9:
10: ...Facts reading procedure...
11:
12: //Evaluation of rule(1)
13: for all tuple ∈ edge do
14:     head ← ⟨tuple[0], tuple[1]⟩
15:     reaches.add(head)
16: end for
17:
18: //Evaluation of rule(2)
19: current_index_tuple_reaches ← 0
20: while current_index_tuple_reaches ≠ reaches.size() do
21:     while current_index_tuple_reaches ≠ reaches.size() do
22:         tuple ← reaches.at(current_index_tuple_reaches)
23:         for all X ∈ edge_1.at(tuple[0]) do
24:             head ← ⟨X, tuple[1]⟩
25:             reaches.push_back(head)
26:         end for
27:         current_index_tuple_reaches ← current_index_tuple_reaches + 1
28:     end while
29: end while
30:
31: //Evaluation of rule(3)
32: for all tuple ∈ vertex do
33:     if not reaches.contains(⟨2, tuple[0]⟩) then
34:         head ← ⟨tuple[0]⟩
35:         noreaches.add(head)
36:     end if
37: end for
38:
39: //Print model
40: vertex.print()
41: edge.print()
42: reaches.print()
43: noreaches.print()
```

CONCLUSIONS AND FUTURE WORKS

- **Specialized C++ implementation for a given program**
  - repetitive operations are omitted
  - ah-hoc data structures are generated
- **Experimental results are encouraging**
  - can be faster than monolithic implementations
- **We plan to extend the complier to the grounding of ASP programs**

ADDITIONAL DETAILS

- **The tool is available at:**

Acknowledgment: The paper has been partially supported by the Italian Ministry for Economic Development (MISE) under project "PIUCultura – Paradigmi Innovativi per l'Utilizzo della Cultura" (n. F/020016/01- 02/X27), under project "Smarter Solutions in the Big Data World (S2BDW)" (n. F/050389/01-03/X32) funded within the call "HORIZON2020" PON I&C 2014-2020.