Towards Ontology-Based Program Analysis

Yue Zhao, Chunhua Liao*, Xipeng Shen

Department of Computer Science, NCSU, Center of Applied Scientific Computing, LLNL*

Abstract

Declarative program analysis has shown the promise to improve the productivity in the development of program analysis. However it is subject to some major limitations in supporting cooperations among analysis tools, guiding program optimizations. It often requires much effort for repeated program preprocessing.

We advocate the integration of ontology into declarative program analysis to standardize the definitions of concepts and representation of the knowledge in the domain.

We develop a prototype framework named PATO for conducting program analysis upon ontology-based program representation. Experiments confirm the potential of ontology for complementing existing declarative program analysis.

Introduction

Declarative program analysis has been proposed to overcome the productivity issues of traditional imperative approach. We inspect the three important limitations of current declarative program analysis

- Cooperations - due to differences in ad-hoc representation of programs and relations.
- Optimizations - less effort is done on the optimization.
- Preprocessing - repeated preprocessing is done for every newly developed program analysis.

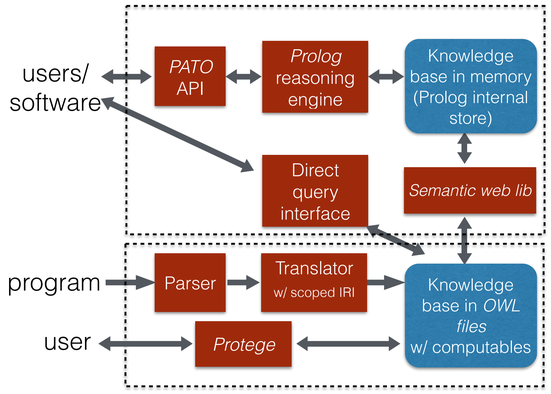
Ontology is a formal, explicit description of a domain's knowledge, including

- concepts (or classes),
- properties of each concept (or relations) and
- individuals (or instances).

It is often represented in standard triple format {subject, property, object} and has a popular standardized format Web Ontology Language (OWL) with a rich tools and ecosystem.

Overview

We propose to use ontology as a generic way to represent and utilize program analysis knowledge.



Challenges and Solutions

- *Ontology design* To create a vocabulary in the domain, particularly C program analysis in this work, for use. We use a language standard-oriented design and continuous enrichment.
- *Knowledge generation* We build a compiler-based ontology converter to parse the source code and generate its ontology representation. The compile used is the ROSE source-to-source compiler for its faithful representation of source code details.
- *Knowledge utilization* The declarative program analysis should be both efficient in performance and general in handling different analysis and computations. We use SWI-Prolog for its versatility and interoperability with C/C++.

Experience

Our experience of using PATO for program analysis to examine the feasibility of using a single ontology to support multiple different program analyses efficiently.

Canonical Loop Analysis

Canonical loop analysis (CLA) checks whether a loop is in a predefined canonical form (defined in OpenMP specification).

Productivity: 190 LOC of Prolog vs 380 LOC C++/ROSE.

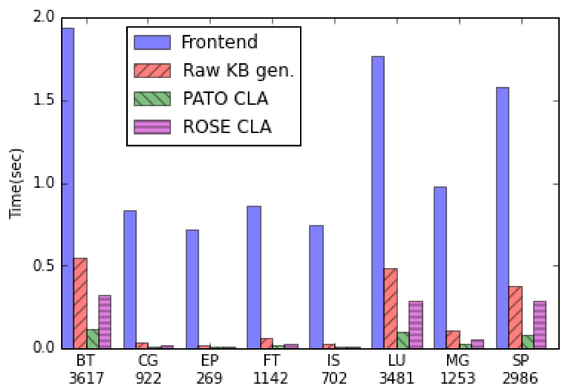


Figure 1: Time comparison of different implementation.

Pointer Analysis

We implement Andersen's pointer analysis in PATO.

1. Extract base constraints from the program – done on the ontology representation through pattern matching rules.
2. Propagate the points-to relations by solving the graph transitive closure problem in Prolog.

```
andersenPtr :-
 select(WorkList, V),
 propLoad(V); propStore(V);
 propFieldLoad(V); propFieldStore(V);
 propEdge(V),
 andersenPtr. % itratively
```

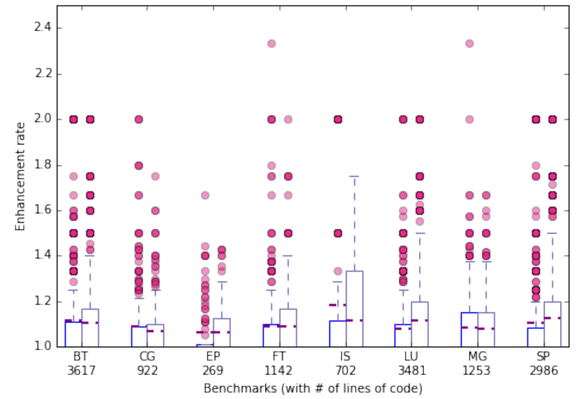
Productivity: implementation has < 500 lines, ≈300 lines for the preprocessing step.

Performance: NPB benchmarks, less than 2 seconds each; bzip2 (7K LOC), 2.1 sec; gzip (8.6K LOC), 3 sec; oggenc (58K LOC) 36 sec.

Facilitating Compiler Cooperations

Liveness analysis is conservative. By converting the analysis results into ontology, we can combine results by different compilers. Let A and B stand for two Liveness analyses and R_A and R_B be their Liveout set for a given basic block.

$$enhancementRate(A) = \frac{|R_A|}{|R_A \cap R_B|}$$



Combined analysis results improve the average precision of liveness analysis by over 10% and 20% for Clang and ROSE respectively.

Conclusion

This work demonstrates the promise of ontology for overcoming the three major limitations of current declarative program analysis

- A single ontology is able to support multiple different program analysis efficiently.
- Knowledge from different sources can be linked with a uniform format.
- Ontology facilitates cooperations among different compilers or program analysis tools.

Acknowledgements

This material is based upon work supported by DOE Early Career Award (DE-SC0013700), and the National Science Foundation (NSF) under Grants No. 1455404, 1455733 (CAREER), and 1525609. This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.