Systematic Predicate Abstraction using Deep Learning

A Preprint

Name [*]
Department of
University
Address
ccc@c.cc

Name [*]
Department of
University
Address
ccc@c.cc

Name [*]
Department of
University
Address
ccc@c.cc

Abstract

Systematic Predicate Abstraction using Deep Learning

***K*eywords** First keyword · Second keyword · More

1 Introduction

Systematic Predicate Abstraction using Deep Learning

2 Background

2.1 Abstraction Based Model Checking

2.2 CEGAR

2.3 Abstract Interpolation

3 Problem Overview

4 Data Collection

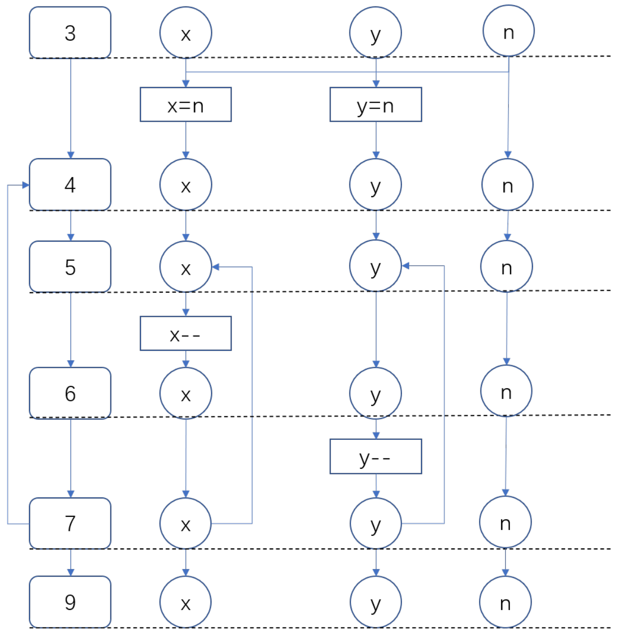
For a single instance, the inputs are a list of templates and a program in the form of text stream and graph. The output is a list of templates with labels.

The strategy A (fixed time out) to extract the training data (a list of templates with labels) is that for one program, we run Eldarica with an abstraction heuristic (e.g. use option -abstract:manual for .c files and -abstract for .smt2 files). If Eldarica can solve the program within 60 seconds. We mark this program as solvable. In Eldarica, we have an initial list of templates. We delete these templates one by one to see if the program can be solved with the remaining templates. If the program is still solvable within the timeout, then that deleted template is critical, we mark it as useful and label it as 1. By doing so iteratively, we can label the initial list of templates. This labelled list of templates will be used in the training process.

1. chc-comp19-benchmarks: 38/1216 .smt2 files (programs) need templates to solve the program within in 60 seconds.
2. sv-comp-smt-benchmarks: 7/6814 .smt2 files need templates.

[*]All contributions are considered equal.

```
extern int n;
void main(){
    int x,y;
    assume(x==n,y==n);
    while(x!=0){
        x--;
        y--;
    }
    assert(y==0);
}
```



(a) C program for horn graph example

(b) Horn graph example

3. sv-comp-c-benchmarks: 31/545 .c files need templates.

Only 76 training programs in total.

A alternative strategy B (variable timeout): First, confirm the solvability (i.e. the program can be solved within 60 seconds with the abstraction heuristic). Second, record the solving time with and without the abstraction heuristic. If Eldarica takes less solving time with abstraction heuristic, pass this solving time as timeout to Eldarica to label the templates used in the program.

Strategy C: keep Strategy A's extracting process. Collect more programs from much larger benchmarks, and use these three benchmarks only for testing.

5 Training Model

5.1 Program Embedding

5.1.1 Text Level

5.1.2 Graph Level

5.1.3 Graph Representation

```
extern int n;
void main(){
    int x,y;
    assume(x==n,y==n);
    while(x!=0){
        x--;
        y--;
    }
    assert(y==0);
}
```

6 Experiments

6.1 Work flow

7 Related work

Program synthesis:

Logical reasoning.

Machine learning methods:

Encode inputs to fixed vector, then decode this vector to target language.

Guide formal method's search process by Training a statistical model to rank some options existed in search space and try them first.

8 Discussion and Future work

References