A Web Frontend For Visualization Of Computation Steps And Their Results In CPAchecker

Sonja Münchow

SoSy-Lab LMU München

July 15, 2020

Overview

1. Introduction
2. Implementation
   - CPA
   - web frontend
3. Results
   - live demo
   - evaluation
4. Summary

Introduction

Visualization

1. Motivation
2. Goal

- Visualize relevant computation steps
- Deeper understanding of the CPA algorithm

Introduction

Visualization

1. Motivation
2. Goal

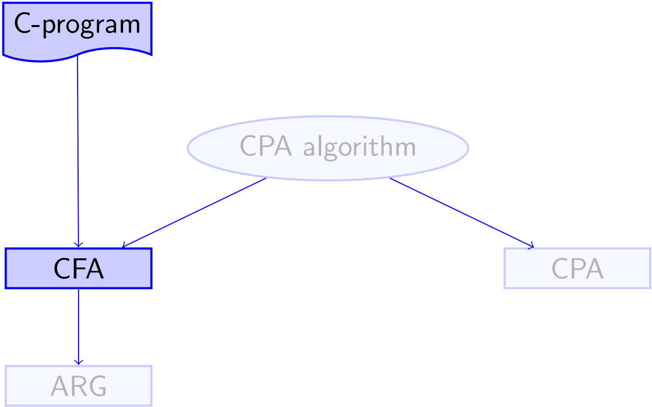
Provide a new web frontend design and implementation for visualization of computation steps and their results

What is CPAchecker?

CPAchecker

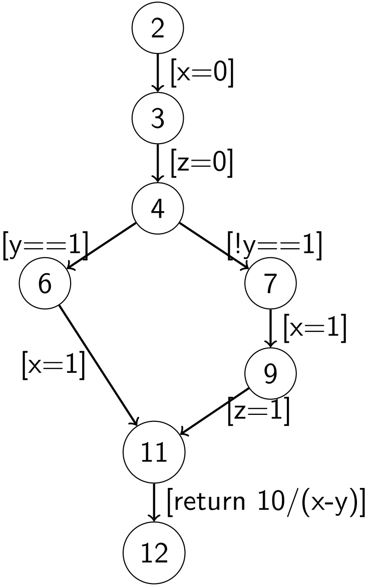
- A software verification tool based on the Configurable Program Analysis (CPA) concepts
- Combines model checking and dataflow analysis
- Results can be viewed interactively in form of a HTML document

What is CPAchecker?

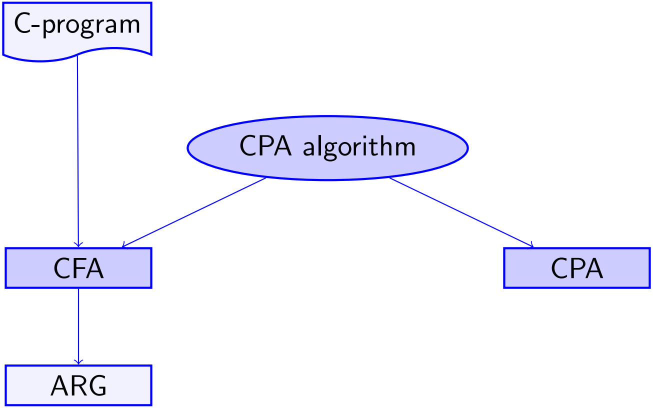


C-pogram and corresponding CFA

```
1
2  int main(int y) {
3   int x = 0;
4   int z = 0;
5
6   if (y== 1){
7    x=1;
8   }else{
9    x=1;
10   z=1;
11  }
12  return 10/ (x-y);
13 }
```



What is CPAchecker?



ARG CPA

ARG CPA

- Defined to track reachable abstract states and build the Abstract Reachability Graph (ARG) using predecessor-successor relations of two abstract states

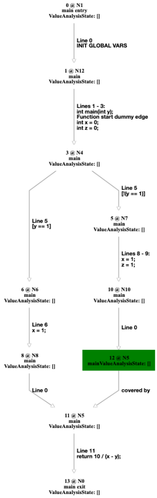
Abstract Reachability Graph

ARG

$R = (N, i, G_{ARG})$

Set of Nodes N: reachable abstract states

Set of Edges G_{ARG}: program operations of corresponding CFA



Introduction

- Final results as Abstract Reachability Graph (ARG)
- Intermediate states are missing in the final ARG

Visualization

- Process of translating abstract or logically difficult to formulate contexts into visible form

Introduction

What is a logically demanding context in regards to CPAchecker?

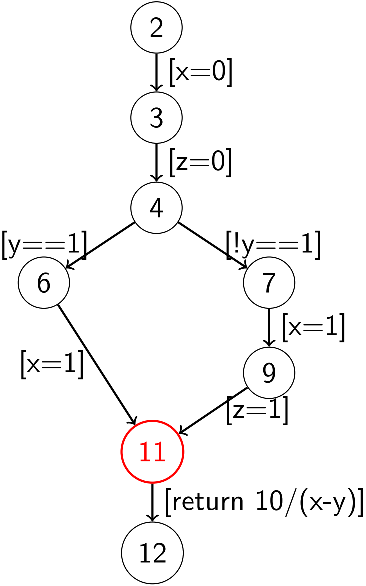
Introduction

Merge Operation

- Combining of two abstract states

The Problem

CFA



can be represented by the predicate $pc = 11 \wedge ((x = 1 \wedge y = 1 \wedge z = 0) \vee (x = 1 \wedge y \neq 1 \wedge z = 1))$

result of merge: pc=11,x=1

The Idea

Step-by-Step creation of ARG in the correct chronological order

How is this realized?

- Build a wrapper CPA around ARG CPA to collect missing states
- Visualize the step by step ARG

How is this realized?

Implementation

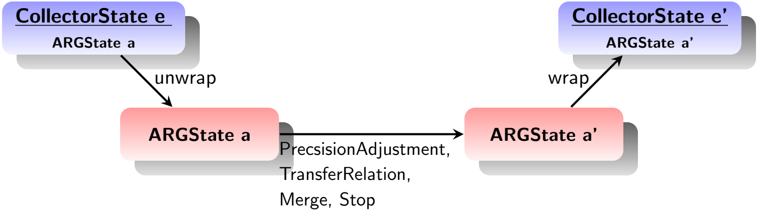
CPA

- Abstract Domain
- Abstract Element
- Transfer Relation
- Merge Operator
- Stop Operator
- Precision Adjustment

Web Frontend

- JavaScript
- CSS
- HTML

Unwrapping of CollectorState for CPA operations



CollectorState e

ARGState a

unwrap

ARGState a

PrecsisionAdjustment, TransferRelation, Merge, Stop

ARGState a'

wrap

CollectorState e'

ARGState a'

Collector CPA Requirements

The CPA interface offers the possibility to define program analyses.

- Abstract Domain
- Abstract Element
- Transfer Relation
- Merge Operator
- Stop Operator
- Precision Adjustment

Collector CPA Definitions

CollectorState e

Each $e \in E_c$ is defined as tuple $e = (a_{c,} H_{merge})$

Collector Transfer Relation

$e = (a_c, H_{merge}) \rightsquigarrow (a'_c, H_{\{\}}) = e'$

if $a_c \in E_{ARG} : \exists a'_c \in E_{ARG} : a_c \rightsquigarrow a'_c$

Collector CPA Definitions

CollectorState e

$e = (a_{c,} H_{merge}) = (a_c, (a_c, a'_c, a''_c))$

H merge

Each tuple $H_{merge} = (a_c, a'_c, a''_c)$ describes the merge operation $merge_{ARG}(a'_c, a''_c) = a_c$

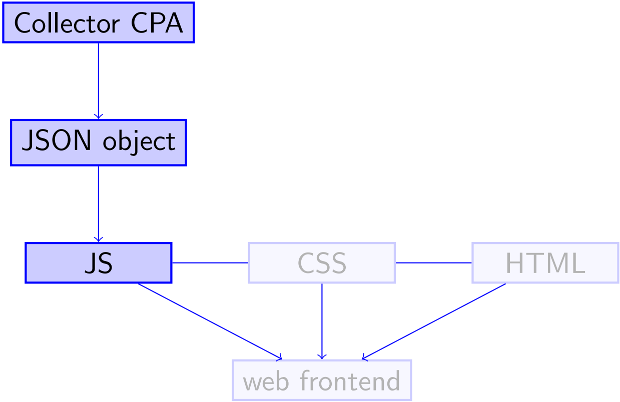
Collector Merge Operator

$\mathrm{merge}_{\mathrm{c}}((a'_c, H'_{merge}), (a''_c, H''_{merge})) = (\mathrm{merge}_{\mathrm{ARG}}(a'_c, a''_c), H_{merge}) = (a_c, H_{merge})$

Collector CPA - Web Frontend

Graph data

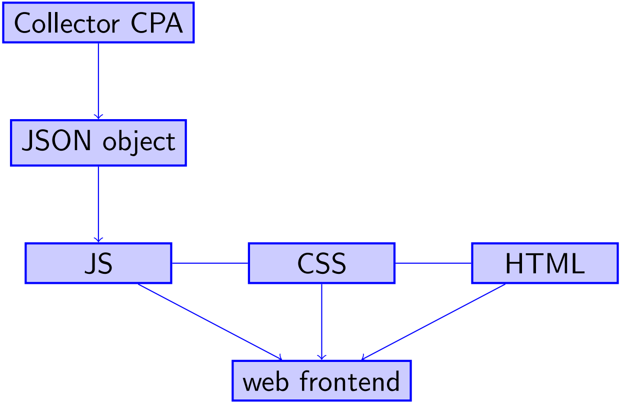
- Transform proven as reachable state to a JSON object



Web Frontend - Visualize the data

Separation of Concerns principle

- HTML framework
- CSS style
- JS



Web Frontend

Javascript

- Graph data in JSON format
- Graph building (Dagre D3)
- Evolving and manipulating the graph interactively (D3 libraries, jQuery)

Web Frontend

- Results as live demo

Results

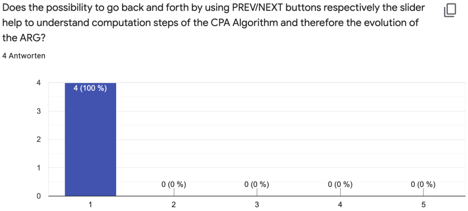
Evaluation

- Evaluation concept
  - Survey
    - How well do the participants know CPAchecker
    - Task 1: operability, ideas for improvements
    - Task 2: typical student exercise
    - General ratings

Results

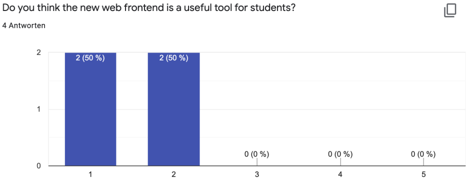
Evaluation results

- Features work to full satisfaction
- Back and forth function and Coloring help to understand computation steps
- Beneficial tool for students and general user



Does the possibility to go back and forth by using PREV/NEXT buttons respectively the slider help to understand computation steps of the CPA Algorithm and therefore the evolution of the ARG?

4 Antworten



Do you think the new web frontend is a useful tool for students?

4 Antworten

scale 1 = Yes; 5 =No

Summary

- Design and Implementation of the new web frontend achieves a better comprehension of the calculation steps of CPAchecker
- Beneficial tool for teaching and general user
- Good basis for further improvements, supports and integration

References

- Dirk Beyer and Sumit Gulwani and David A. Schmidt (2018)
  Combining Model Checking and Data-Flow Analysis
  Handbook of Model Checking Springer 2018, 493–540.

- Dirk Beyer and Thomas Lemberger (2017)
  Software Verification: Testing vs. Model Checking - A Comparative Evaluation of the State of the Art
  Lecture Notes in Computer Science Springer 2017, 99–114.

- Dirk Beyer and M. Erkan Keremoglu (2011)
  CPAchecker: A Tool for Configurable Software Verification
  Lecture Notes in Computer Science Springer 2011, 184–190.

Thank you!