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
Visualization

1 Motivation
2 Goal

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

CPA algorithm

CFA

ARG

CPA
```c
int main(int y) {
    int x = 0;
    int z = 0;
    if (y == 1) {
        x = 1;
    } else {
        x = 1;
        z = 1;
    }
    return 10 / (x - y);
}
```
What is CPAchecker?

C-program → CPA algorithm → CFA → CPA → ARG
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

\[ \text{ARG} \]

\[ R = (N, i, G_{\text{ARG}}) \]

Set of Nodes \( N \): reachable abstract states

Set of Edges \( G_{\text{ARG}} \): program operations of corresponding CFA
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
What is a logically demanding context in regards to CPAchecker?
Merge Operation

- Combining of two abstract states
The Problem

CFA

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

result of merge: $pc=11, x=1$
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 \( a \)

PrecisionAdjustment,
TransferRelation,
Merge, Stop

ARGState \( a' \)

CollectorState \( e' \)

ARGState \( a' \)

wrap
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**

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

if $a_c \in E_{ARG} : \exists a'_c \in E_{ARG} : a_c \leadsto a'_c$
Collector CPA Definitions

CollectorState e

\[ e = (a_c, H_{\text{merge}}) = (a_c, (a_c, a'_c, a''_c)) \]

H merge

Each tuple \( H_{\text{merge}} = (a_c, a'_c, a''_c) \) describes the merge operation \( \text{merge}_{\text{ARG}}(a'_c, a''_c) = a_c \)

Collector Merge Operator

\[ \text{merge}_c((a'_c, H'_{\text{merge}}), (a''_c, H''_{\text{merge}})) = (\text{merge}_{\text{ARG}}(a'_c, a''_c), H_{\text{merge}}) = (a_c, H_{\text{merge}}) \]
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

Collector CPA

JSON object

JS → CSS → HTML

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
Evaluation

- Evaluation concept
  - Survey
  - How well do the participants know CPAchecker
  - Task 1: operability, ideas for improvements
  - Task 2: typical student exercise
  - General ratings
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
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
Dirk Beyer and Sumit Gulwani and David A. Schmidt (2018)
Combining Model Checking and Data-Flow Analysis

Dirk Beyer and Thomas Lemberger (2017)
Software Verification: Testing vs. Model Checking - A Comparative Evaluation of the State of the Art

Dirk Beyer and M. Erkan Keremoglu (2011)
CPAchecker: A Tool for Configurable Software Verification
Thank you!