CPAchecker

„The Configurable Software-Verification Platform“

Marek Tomáštík

40. 3. 2015

Prerekvizity

Compile once, run anywhere

pro spuštění nástroje

- Java Runtime Environment verze 7 nebo vyšší

Prerekvizity

Compile once, run anywhere

pro spuštění nástroje

- Java Runtime Environment verze 7 nebo vyšší
  - Ubuntu: `apt-get install openjdk-7-jre`

Prerekvizity

Compile once, run anywhere

pro spuštění nástroje

- Java Runtime Environment verze 7 nebo vyšší
  - Ubuntu: `apt-get install openjdk-7-jre`

pro generování HTML reportů

- Python (snad) libovolné verze 2.x
- nástroj dot

Prerekvizity

Compile once, run anywhere

pro spuštění nástroje

- Java Runtime Environment verze 7 nebo vyšší
  - Ubuntu: `apt-get install openjdk-7-jre`

pro generování HTML reportů

- Python (snad) libovolné verze 2.x
- nástroj dot
  - Ubuntu: `apt-get install graphviz`

Představení

- http://cpachecker.sosy-lab.org/

Představení

- http://cpachecker.sosy-lab.org/
- „CPAchecker is a freely available software-verification framework, built on the concepts of Configurable Program Analysis (CPA). CPAchecker integrates most of the state-of-the-art technologies for software model checking, such as counterexample-guided abstraction refinement (CEGAR), lazy predicate abstraction, interpolation-based refinement, and large-block encoding.“[1]

[1] CPAchecker with Adjustable Predicate Analysis (Competition Contribution), 2012

Představení
Configurable Program Analysis

- „Automatic program verification requires a choice between precision and efficiency. The more precise a method, the fewer false positives it will produce, but also the more expensive it is, and thus applicable to fewer programs.“[2]

[2] Configurable Software Verification: Concretizing the Convergence of Model Checking and Program Analysis, 2007

Představení
Configurable Program Analysis

- „Automatic program verification requires a choice between precision and efficiency. The more precise a method, the fewer false positives it will produce, but also the more expensive it is, and thus applicable to fewer programs.“[2]
- „Historically, this trade-off was reflected in two major approaches to static verification: program analysis and model checking. (...) Program analyzers, by and large, still target the efficient computation of few simple facts about large programs; model checkers, by contrast, focus still on the removal of false alarms through ever more refined analyses of relatively small programs.“[2]

[2] Configurable Software Verification: Concretizing the Convergence of Model Checking and Program Analysis, 2007

Úspěchy

- 3× první místo, 1× druhé místo v soutěži Competition on Software Verification

Úspěchy

- 3× první místo, 1× druhé místo v soutěži Competition on Software Verification
  - vyvíjen v Software Systems Lab, University of Passau

Úspěchy

- 3× první místo, 1× druhé místo v soutěži Competition on Software Verification
  - vyvíjen v Software Systems Lab, University of Passau
  - „SV-COMP is sponsored by the University of Passau (Software Systems Lab), Germany and Microsoft Research, UK.“[3]

[3] sv-comp.sosy-lab.org/2015/

Úspěchy

- 3× první místo, 1× druhé místo v soutěži Competition on Software Verification
  - vyvíjen v Software Systems Lab, University of Passau
  - „SV-COMP is sponsored by the University of Passau (Software Systems Lab), Germany and Microsoft Research, UK.“[3]
- několik odhalených chyb v Linux kernelu

[3] sv-comp.sosy-lab.org/2015/

Úspěchy

- 3× první místo, 1× druhé místo v soutěži Competition on Software Verification
  - vyvíjen v Software Systems Lab, University of Passau
  - „SV-COMP is sponsored by the University of Passau (Software Systems Lab), Germany and Microsoft Research, UK.“[3]
- několik odhalených chyb v Linux kernelu
  - zdá se, že právě 7[4]

[3] sv-comp.sosy-lab.org/2015/

[4] code.google.com/p/cpachecker/source/browse/trunk/doc/BugsFound.txt

Features
Teorie

Table 4. Technologies and features that the verification tools offer

Verifier	CEGAR	Predicate Abstraction	Symbolic Execution	Bounded Model Checking	k-Induction	Property-Directed Reachability	Explicit-Value Analysis	Numerical Interval Analysis	Shape Analysis	Separation Logic	Bit-Precise Analysis	ARG-Based Analysis	Lazy Abstraction	Interpolation	Automata-Based Analysis	Concurrency Support	Ranking Functions
APInfVE			✓				✓	✓			✓						✓
Blocco	✓	✓		✓													
Blast	✓	✓					✓					✓	✓	✓			
Cascade			✓	✓							✓						
Cbmc				✓	✓						✓					✓	
CPAchecker	✓	✓	✓	✓	✓		✓	✓	✓		✓	✓	✓	✓			

[5]

[5] Software Verification and Verifiable Witnesses (Report on SV-COMP 2015), 2015

Features
Praxe

- specifikace systému: program v jazyce C

Features
Praxe

- specifikace systému: program v jazyce C
  - „CPAchecker is able to parse and analyze a large subset of (GNU)C“[6]

[6] code.google.com/p/cpachecker/source/browse/trunk/README.txt

Features
Praxe

- specifikace systému: program v jazyce C
  - „CPAchecker is able to parse and analyze a large subset of (GNU)C“[6]
  - experimentální podpora pro Javu (nepodařilo se mi zprovoznit)

[6] code.google.com/p/cpachecker/source/browse/trunk/README.txt

Features
Praxe

- specifikace systému: program v jazyce C
  - „CPAchecker is able to parse and analyze a large subset of (GNU)C“[6]
  - experimentální podpora pro Javu (nepodařilo se mi zprovoznit)
- specifikace specifikace: několik (18) předdefinovaných

[6] code.google.com/p/cpachecker/source/browse/trunk/README.txt

Features
Praxe

- specifikace systému: program v jazyce C
  - „CPAchecker is able to parse and analyze a large subset of (GNU)C“[6]
  - experimentální podpora pro Javu (nepodařilo se mi zprovoznit)
- specifikace specifikace: několik (18) předdefinovaných
  - pouze reachability vlastnosti, vyjádřené jako jednoduché automaty (v nezdokumentovaném jazyce)

[6] code.google.com/p/cpachecker/source/browse/trunk/README.txt

Features
Praxe

- specifikace systému: program v jazyce C
  - „CPAchecker is able to parse and analyze a large subset of (GNU)C“[6]
  - experimentální podpora pro Javu (nepodařilo se mi zprovoznit)
- specifikace specifikace: několik (18) předdefinovaných
  - pouze reachability vlastnosti, vyjádřené jako jednoduché automaty (v nezdokumentovaném jazyce)
  - default: kontroluje asserty v kódu, dosažení labelu `ERROR` (`goto ERROR`), volání funkcí `abort()` a `exit()`

[6] code.google.com/p/cpachecker/source/browse/trunk/README.txt

Features
Praxe

- specifikace systému: program v jazyce C
  - „CPAchecker is able to parse and analyze a large subset of (GNU)C“[6]
  - experimentální podpora pro Javu (nepodařilo se mi zprovoznit)
- specifikace specifikace: několik (18) předdefinovaných
  - pouze reachability vlastnosti, vyjádřené jako jednoduché automaty (v nezdokumentovaném jazyce)
  - default: kontroluje asserty v kódu, dosažení labelu `ERROR` (`goto ERROR`), volání funkcí `abort()` a `exit()`
  - také možné ověřit memory safety (invalid free, invalid dereference, memory leak)

[6] code.google.com/p/cpachecker/source/browse/trunk/README.txt

Features
Nepodporováno

- nepodporuje verifikaci programů využívajících souběžnost

Features
Nepodporováno

- nepodporuje verifikaci programů využívajících souběžnost
- nepodporuje verifikaci programů využívajících rekurzi

Features
Nepodporováno

- nepodporuje verifikaci programů využívajících souběžnost
- nepodporuje verifikaci programů využívajících rekurzi
- není paralelní, umí využít jen jedno jádro

Použití
Co je to uživatel?

- „There is no manual for CPAchecker yet, other than the open source code ;)“[7]

[7] CPAchecker Users Google Group, 2015

Použití
Co je to uživatel?

- „There is no manual for CPAchecker yet, other than the open source code ;)“[7]

[7] CPAchecker Users Google Group, 2015

Použití
Co je to uživatel?

- „There is no manual for CPAchecker yet, other than the open source code ;)“[7]
- „Tutorial coming soon“[8]

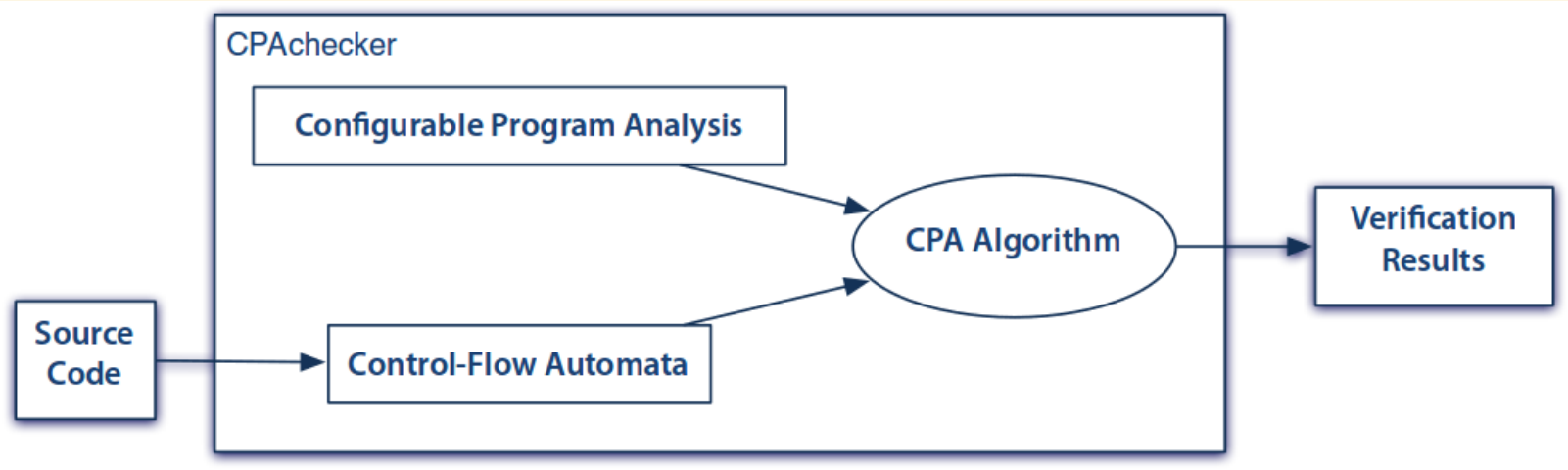
[7] CPAchecker Users Google Group, 2015
[8] cpachecker.sosy-lab.org/doc.php

Použití
Základní konfigurace

- „The usual command line for running CPAchecker is to specify a configuration file (either with "-config FILE" or "-CONFIGFILE") and a program file.“[9]

[9] cpachecker.googlecode.com/svn/trunk/doc/Configuration.txt

Použití
Flowchart



[10]

[10] Configurable Program Analysis + CPAchecker, 2013

Použití
Základní konfigurace

- „The usual command line for running CPAchecker is to specify a configuration file (either with "-config FILE" or "-CONFIGFILE") and a program file.“[9]
- configuration file specifikuje parametry verifikace, hlavně abstraktní doménu

[9] cpachecker.googlecode.com/svn/trunk/doc/Configuration.txt

Použití
Základní konfigurace

- „The usual command line for running CPAchecker is to specify a configuration file (either with "-config FILE" or "-CONFIGFILE") and a program file.“[9]
- configuration file specifikuje parametry verifikace, hlavně abstraktní doménu
  - zahrnuje i specifikaci, která se má ověřovat

[9] cpachecker.googlecode.com/svn/trunk/doc/Configuration.txt

Použití
Základní konfigurace

- „The usual command line for running CPAchecker is to specify a configuration file (either with "-config FILE" or "-CONFIGFILE") and a program file.“[9]
- configuration file specifikuje parametry verifikace, hlavně abstraktní doménu
  - zahrnuje i specifikaci, která se má ověřovat
  - několik (116) předdefinovaných

[9] cpachecker.googlecode.com/svn/trunk/doc/Configuration.txt

Použití
Configuration files – jak si vybrat?

- „For general purpose verification tasks (outside the competition), we recommend the configuration `-predicateAnalysis`.“[11]

[11] CPAchecker with Adjustable Predicate Analysis (Competition Contribution), 2012

Použití

Configuration files – jak si vybrat?

- „For general purpose verification tasks (outside the competition), we recommend the configuration `-predicateAnalysis`.“[11]
- „I assume you tried valueAnalysis. Maybe run it with `-predicateAnalysis`, or `-predicateAnalysis-bitprecise` for a predicate analysis or even a bit-precise predicate analysis. Then `-octagonAnalysis` is also worth a try, so is `-bmc-induction`, or `-sv-comp15`, the latter being a serialization of many different approaches.“[12]

[11] CPAchecker with Adjustable Predicate Analysis (Competition Contribution), 2012

[12] CPAchecker Users Google Group, 2015

Použití

Configuration files – jak si vybrat?

- „It really depends on the type of programs you have, there is not a single best configuration. Trying a combination like `-sv-comp15` might be a good idea. Other than that, you have to try what works best for you.“[13]

[13] CPAchecker Users Google Group, 2015

Použití

Configuration files – jak si vybrat?

- „It really depends on the type of programs you have, there is not a single best configuration. Trying a combination like `-sv-comp15` might be a good idea. Other than that, you have to try what works best for you.“[13]
- “(...) value analysis (...) is in general not able to track values of variables that have a non-deterministic (or unknown) value. Instead, it is only able to track concrete values. (...) For this program, as simple as it may be, you will fail with the value analysis, because you need some sort of symbolic analysis.“[14]

[13] CPAchecker Users Google Group, 2015
[14] CPAchecker Users Google Group, 2014

Použití

Configuration files – výběr

`-valueAnalysis` (dříve `-explicitAnalysis`)

This configuration file uses value analysis of integer variables in a model-checking configuration. Bitprecise predicate analysis is used to cross-check counterexamples. This configuration makes use of a CEGAR approach, by only tracking variables that are found, via interpolation, to be relevant to the error.

Použití

Configuration files – výběr

`-valueAnalysis` (dříve `-explicitAnalysis`)

This configuration file uses value analysis of integer variables in a model-checking configuration. Bitprecise predicate analysis is used to cross-check counterexamples. This configuration makes use of a CEGAR approach, by only tracking variables that are found, via interpolation, to be relevant to the error.

`-predicateAnalysis`

This configuration file uses the Adjustable-Block Encoding CPA for predicate analysis with CEGAR as described in "Predicate Abstraction with Adjustable-Block Encoding" (Beyer et al.). It is configured for abstractions at loop heads, similar to LBE but with function inlining.

Použití

Configuration files – výběr

`-bmc-induction`

This configuration file enables Bounded Model Checking and uses induction for proving safety (EXPERIMENTAL).

Použití

Configuration files – výběr

`-bmc-induction`

This configuration file enables Bounded Model Checking and uses induction for proving safety (EXPERIMENTAL).

`-sv-comp15`

This configuration file uses a sequential combination of three different analyses, namely a value analysis, a bounded model checking analysis with k-induction, and a predicate analysis, in a total of six different configurations.

Použití

Configuration files – výběr

`-bmc-induction`

This configuration file enables Bounded Model Checking and uses induction for proving safety (EXPERIMENTAL).

`-sv-comp15`

This configuration file uses a sequential combination of three different analyses, namely a value analysis, a bounded model checking analysis with k-induction, and a predicate analysis, in a total of six different configurations.

`-sv-comp15--memorysafety`

This configuration file uses a combination of value analysis and symbolic memory graphs to verify memory safety properties.

Použití
Další parametry

`-preprocess`

potřeba pokud vstupní program obsahuje direktivy preprocesoru

Použití
Další parametry

`-preprocess`

potřeba pokud vstupní program obsahuje direktivy preprocesoru

`-heap <MEM>`

limit paměti pro Java heap, default: `1200M`

Použití
Další parametry

`-preprocess`

potřeba pokud vstupní program obsahuje direktivy preprocesoru

`-heap <MEM>`

limit paměti pro Java heap, default: `1200M`

`-entryfunction <FUNC>`

entry function pro verifikaci, default: `main`

Použití
Další parametry

`-preprocess`

potřeba pokud vstupní program obsahuje direktivy preprocesoru

`-heap <MEM>`

limit paměti pro Java heap, default: `1200M`

`-entryfunction <FUNC>`

entry function pro verifikaci, default: `main`

`-32, -64`

verifikace pro 32bitovou nebo 64bitovou architekturu, default: `-32`

Příklady

Hello, world!

verifikace

```
scripts/cpa.sh -predicateAnalysis
doc/examples/example.c
```

Příklady
Hello, world!

verifikace

```
scripts/cpa.sh -predicateAnalysis
doc/examples/example.c
```

- vytvoří několik souborů v adresáři `./output`

Příklady
Hello, world!

verifikace

```
scripts/cpa.sh -predicateAnalysis
doc/examples/example.c
```

- vytvoří několik souborů v adresáři `./output`

vygenerování reportu

```
scripts/report-generator.py
```

Příklady

Hello, world!

verifikace

```
scripts/cpa.sh -predicateAnalysis
doc/examples/example.c
```

- vytvoří několik souborů v adresáři `./output`

vygenerování reportu

```
scripts/report-generator.py
```

- vytvoří ze souborů v `./output` HTML report

Příklady

Explicitně vs. symbolicky a ARG

symbolická analýza, explicitní analýza

```
scripts/cpa.sh [-predicateAnalysis |
-valueAnalysis] doc/examples/example.c &&
scripts/report-generator.py
```

Příklady
Explicitně vs. symbolicky a ARG

symbolická analýza, explicitní analýza

```
scripts/cpa.sh [-predicateAnalysis |
-valueAnalysis] doc/examples/example.c &&
scripts/report-generator.py
```

- ARG (Abstract Reachability Graph) – graf cest v programu, které byly ověřeny při verifikaci

Příklady
Explicitně vs. symbolicky a ARG

symbolická analýza, explicitní analýza

```
scripts/cpa.sh [-predicateAnalysis |
-valueAnalysis] doc/examples/example.c &&
scripts/report-generator.py
```

- ARG (Abstract Reachability Graph) – graf cest v programu, které byly ověřeny při verifikaci
  - pokud nebyla nalezena chyba, slouží jako certifikát výsledku

Příklady
Explicitně vs. symbolicky a ARG

symbolická analýza, explicitní analýza

```
scripts/cpa.sh [-predicateAnalysis |
-valueAnalysis] doc/examples/example.c &&
scripts/report-generator.py
```

- ARG (Abstract Reachability Graph) – graf cest v programu, které byly ověřeny při verifikaci
  - pokud nebyla nalezena chyba, slouží jako certifikát výsledku
  - pokud byla nalezena chyba, je v něm vyznačen eror path