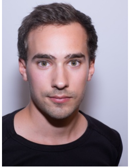
Program Transformations with CPAchecker

Thomas Lemberger

LMU Munich, Germany



CPAchecker

- CPAchecker is more than a verifier
- CPAchecker provides full program-analysis infrastructure

→ CFA frontend parses programs and creates a flexible program representation

→ CPA infrastructure is very flexible due to composite structure, and easy to extend

Outline

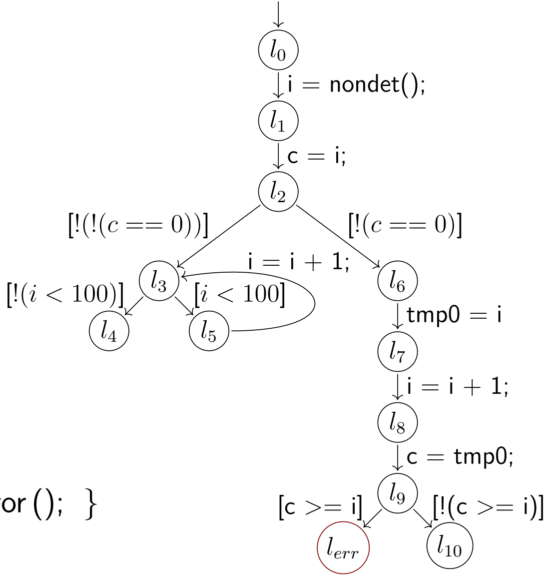
1. Program representation: CFA
2. Program state-space representation: ARG
3. Examples

Control-flow Automaton (CFA)

CFA $P = (L, l_0, G)$

```
int i = nondet();
int c = i;

if (c) {
  c = i++;
} else {
  while (i < 100) {
    i = i + 1;
  }
  return;
}
if (c >= i) { reach_error(); }
```

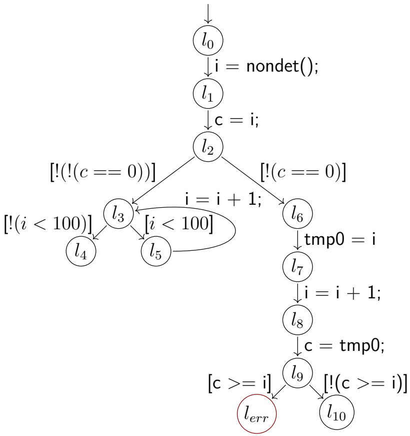


Transformations with CCfaTransformer

- `CFA substituteAstNodes(/* snip */ CFA, BiFunction<CFAEdge, CAstNode, CAstNode>)`
- `CFA createCfa(/* snip */ CFA, MutableGraph<CFANode, CFAEdge>, BiFunction<CFAEdge, CAstNode, CAstNode>)`

Exporting with CFAToCTranslator

Exports CFA to C.
Big challenge: Does not look like the original.




```c
int i;
i = nondet();
int c = i;
if (!(c == 0)) {
  int tmp0 = i;
  i = i + 1;
  c = tmp0;
  if (c >= i) {
    reach_error();
    goto label_2;
  } else {
  label_2:;
    goto label_1;
  }
} else {
label_0:;
  if (i < 100) {
    i = i + 1;
    goto label_0;
  } else {
    return;
  label_1:;
    abort();
  }
}
```

Example: Program Slicing

Given program P and a slicing criterion $\mathcal{C} \subseteq L$, compute a new program $slice(P, \mathcal{C})$ that is behaviorally equivalent regarding all program executions to program locations $l \in \mathcal{C}$.

Uses: Debugging, program abstraction

Example: Program Slicing

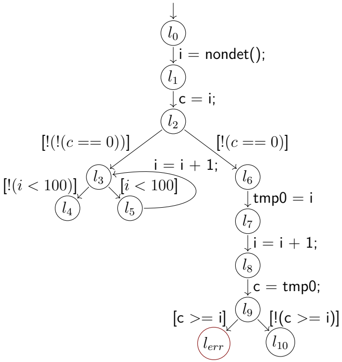
- Computed with control and use-def dependencies (`CSystemDependenceGraph`)[1]
    - Control dependency: Reaching location of interest is influenced by this control statement
    - Use-def dependency: Evaluation of statement of interest is influenced by this variable assignment
- **We** just replace program operations with `nop`
- Internally done by SlicingCPA

[1] S. Horwitz, T. W. Reps, and D. W. Binkley. "Interprocedural Slicing Using Dependence Graphs". In: *ACM Trans. Program. Lang. Syst.* 12.1 (1990), pp. 26–60. DOI: 10.1145/77606.77608

Example: Program Slicing

- Computed with control and use-def dependencies (`CSystemDependenceGraph`)[1]
  - Control dependency: Reaching location of interest is influenced by this control statement
  - Use-def dependency: Evaluation of statement of interest is influenced by this variable assignment
- **We** just replace program operations with `nop`
- Internally done by SlicingCPA

$$\mathcal{C} = \{l_{err}\}$$

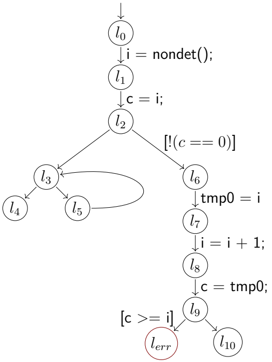


[1] S. Horwitz, T. W. Reps, and D. W. Binkley. "Interprocedural Slicing Using Dependence Graphs". In: *ACM Trans. Program. Lang. Syst.* 12.1 (1990), pp. 26–60. DOI: 10.1145/77606.77608

Example: Program Slicing

- Computed with control and use-def dependencies (`CSystemDependenceGraph`)[1]
  - Control dependency: Reaching location of interest is influenced by this control statement
  - Use-def dependency: Evaluation of statement of interest is influenced by this variable assignment
- **We** just replace program operations with `nop`
- Internally done by SlicingCPA

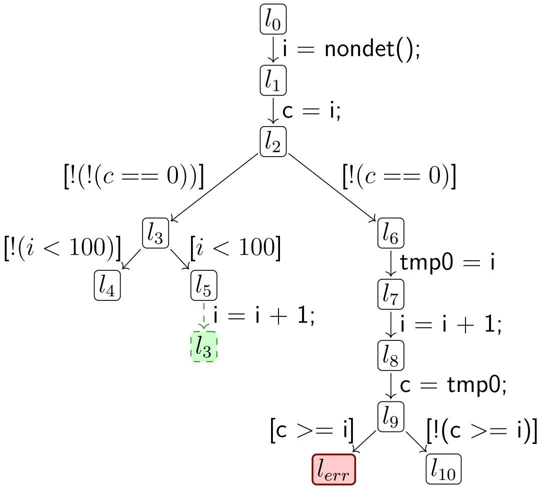
$$\mathcal{C} = \{l_{err}\}$$



[1] S. Horwitz, T. W. Reps, and D. W. Binkley. "Interprocedural Slicing Using Dependence Graphs". In: *ACM Trans. Program. Lang. Syst.* 12.1 (1990), pp. 26–60. DOI: 10.1145/77606.77608

Abstract Reachability Graph (ARG)

Represents the explored state space as parent-child relation between abstract program states.



Exporting with ARGToCTranslator

Translates state space of ARG to C program.
Similar structure as CFAToCTranslator.

Any abstract domain can be used to compute ARG (here as *configurable program analysis*)
$\Rightarrow$ very flexible tool.

Configurable Program Analysis (CPA)

CPA $\mathbb{D} = (D, \rightsquigarrow, \mathsf{merge}, \mathsf{stop})$ [2] with:

- Abstract domain $D = (C, \mathcal{E}, [\![\cdot]\!])$
- Transfer relation $\rightsquigarrow$
- Operator merge
- Operator stop

is used in fix-point algorithm to compute abstract state space.

[2] D. Beyer, S. Gulwani, and D. Schmidt. "Combining Model Checking and Data-Flow Analysis". In: *Handbook of Model Checking*. Springer, 2018, pp. 493–540. DOI: 10.1007/978-3-319-10575-8_16

Example: LocationCPA

CPA $\mathbb{L} = (D_{\mathbb{L}}, \rightsquigarrow_{\mathbb{L}}, \mathsf{merge}^{sep}, \mathsf{stop}^{sep})$ can be used to compute all reachable program locations:

$$l \overset{(l,op,l')}{\rightsquigarrow}_{\mathbb{L}} l'$$

When starting with l_0, all program locations from the program entry are computed.
stop^{sep} makes the algorithm explore each program location only once.

Example: ControlAutomatonCPA

Introduced as protocol analysis[3].

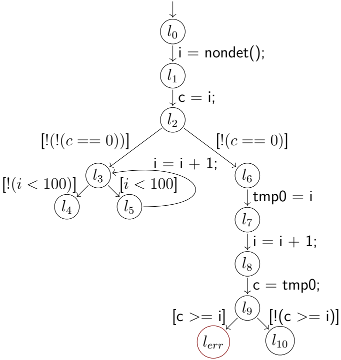
Non-deterministic, finite automaton $A = (Q, \Sigma, \delta, q_0, F)$ for CFA $P = (L, l_0, G)$ with:

- States Q
- Alphabet $\Sigma \subseteq 2^G \times \Phi$ (source-code guards and state-space guards)
- Transition relation $\delta \subseteq Q \times \Sigma \times Q$
- Initial state q_0
- Accepting states F

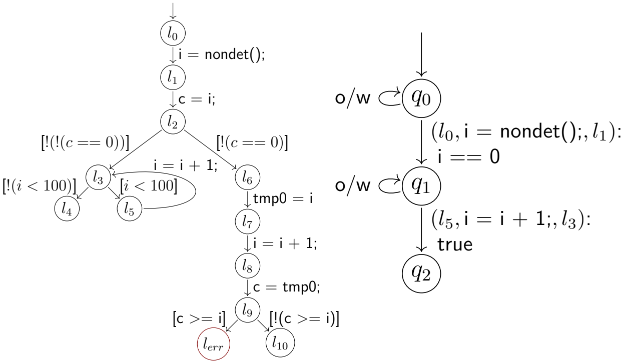
ControlAutomatonCPA tracks the currently possible states of the automaton during program analysis and restricts the state space through source-code and state-space guards.

[3] D. Beyer et al. "Witness Validation and Stepwise Testification across Software Verifiers". In: *Proc. FSE*. ACM, 2015, pp. 721–733. DOI: 10.1145/2786805.2786867

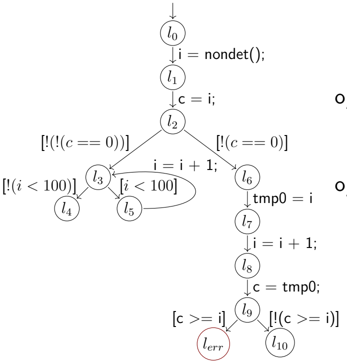
Example: ControlAutomatonCPA

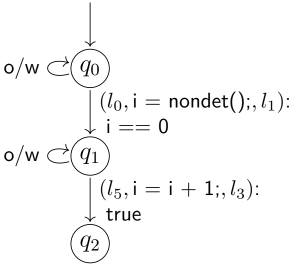


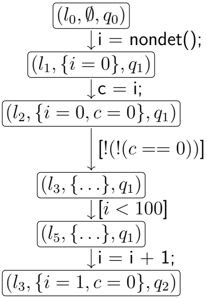
Example: ControlAutomatonCPA



Example: ControlAutomatonCPA







Example: ControlAutomatonCPA

$$\boxed{(l_0, \emptyset, q_0)}$$
$$\downarrow \text{i = nondet();}$$
$$\boxed{(l_1, \{i = 0\}, q_1)}$$
$$\downarrow \text{c = i;}$$
$$\boxed{(l_2, \{i = 0, c = 0\}, q_1)}$$
$$\downarrow [!(!(c == 0))]$$
$$\boxed{(l_3, \{\ldots\}, q_1)}$$
$$\downarrow [i < 100]$$
$$\boxed{(l_5, \{\ldots\}, q_1)}$$
$$\downarrow \text{i = i + 1;}$$
$$\boxed{(l_3, \{i = 1, c = 0\}, q_2)}$$

```
int i;
i = nondet();
int c = i;
if (!(c == 0)) {
  label_0:;
  abort();
} else {
  if (i < 100) {
    i = i + 1;
    goto label_0;
  } else {
    goto label_0;
  }
}
```

Example: ControlAutomatonCPA

$$\begin{tikzcd}
\boxed{(l_0, \emptyset, q_0)} \arrow[d, "\text{i = nondet();}"] \\
\boxed{(l_1, \{i = 0\}, q_1)} \arrow[d, "\text{c = i;}"] \\
\boxed{(l_2, \{i = 0, c = 0\}, q_1)} \arrow[d, "{[!(!(c == 0))]}"] \\
\boxed{(l_3, \{\ldots\}, q_1)} \arrow[d, "{[i < 100]}"] \\
\boxed{(l_5, \{\ldots\}, q_1)} \arrow[d, "\text{i = i + 1;}"] \\
\boxed{(l_3, \{i = 1, c = 0\}, q_2)}
\end{tikzcd}$$

```
int i;
i = nondet();
int c = i;
if (!(c == 0)) {
  label_0:;
  abort();
} else {
  if (i < 100) {
    i = i + 1;
    goto label_0;
  } else {
    goto label_0;
  }
}
```

- ARGToCTranslator can differentiate between if and else-conditions
- Information from abstract states is not used (yet)

Example: ControlAutomatonCPA

This concept is used for residual-program generation [4] and difference verification [5].

[4] D. Beyer et al. "Reducer-Based Construction of Conditional Verifiers". In: *Proc. ICSE*. ACM, 2018, pp. 1182–1193. DOI: 10.1145/3180155.3180259

[5] D. Beyer, M.-C. Jakobs, and T. Lemberger. "Difference Verification with Conditions". In: *Proc. SEFM*. LNCS 12310. Springer, 2020, pp. 133–154. DOI: 10.1007/978-3-030-58768-0_8

Other examples

Other examples for program transformations in CPAchecker:

- Abstraction of loops over arrays through CFA transformation
- MetaVal[6]
- Adding test-goal labels for coverage measurement (LabelAdder)
- Program repair with CFA mutations?

[6] D. Beyer and M. Spiessl. "MetaVal: Witness Validation via Verification". In: *Proc. CAV*. LNCS 12225. Springer, 2020, pp. 165–177. DOI: 10.1007/978-3-030-53291-8_10

Future Work

In export, stay as close to original program as possible.

Thank you!