Program Transformations with CPAchecker

Thomas Lemberger

LMU Munich, Germany
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 $P = (L, l_0, G)$

```c
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>)
Exports CFA to C.
Big challenge: Does not look like the original.

```
int i;
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 $\text{slice}(P, \mathcal{C})$ 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 ($\text{CSystemDependenceGraph}$)\(^1\)
  - 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

---

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 \texttt{nop}

- Internally done by SlicingCPA

---

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 \texttt{nop}
- Internally done by SlicingCPA

\[ C = \{ l_{err} \} \]

\[ l_0 \]
\[ l_1 \]
\[ l_2 \]
\[ (\neg (c == 0)) \]
\[ i < 100 \]
\[ i = i + 1 \]
\[ (\neg (i < 100)) \]
\[ (\neg (c == 0)) \]
\[ tmp_0 = i \]
\[ i = i + 1 \]
\[ c = tmp_0 \]
\[ (c >= i) \]
\[ (\neg (c >= i)) \]

\[ l_{err} \]

\[ l_{10} \]

---

Abstract Reachability Graph (ARG)

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

\[ i = \text{nondet}(); \]
\[ c = i; \]
\[ !(i < 100) \]
\[ i = i + 1; \]
\[ c = \text{tmp0}; \]
\[ c \geq i \]
\[ !(c \geq i) \]
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) ⇒ very flexible tool.
Configurable Program Analysis (CPA)

CPA $\mathbb{D} = (D, \rightsquigarrow, \text{merge, stop})^2$ with:

- Abstract domain $D = (C, E, \llbracket \cdot \rrbracket)$
- Transfer relation $\rightsquigarrow$
- Operator merge
- Operator stop

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

---

Example: Location CPA

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

$$l^{(l, op, l')} \xrightarrow{\sim} \mathbb{L} l'$$

When starting with $l_0$, all program locations from the program entry are computed.

$\text{stop}^{sep}$ 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.

Example: ControlAutomatonCPA

\[
\begin{align*}
    l_0 & \xrightarrow{} i = \text{nondet}() \\
    l_1 & \xrightarrow{} c = i \\
    l_2 & \xrightarrow{} c = i \\
    l_3 & \xrightarrow{} i = i + 1 \\
    l_4 & \xrightarrow{} i < 100 \\
    l_5 & \xrightarrow{} i = i + 1 \\
    l_6 & \xrightarrow{} \text{tmp0} = i \\
    l_7 & \xrightarrow{} \text{tmp0} = i \\
    l_8 & \xrightarrow{} c = \text{tmp0} \\
    l_9 & \xrightarrow{} c = \text{tmp0} \\
    l_{\text{err}} & \xrightarrow{} \text{err} \\
    l_{10} & \xrightarrow{} \text{err}
\end{align*}
\]
Example: ControlAutomatonCPA

\[
\begin{align*}
\text{l}_{0} & \quad \text{i} = \text{nondet}(); \\
\text{l}_{1} & \quad \text{c} = \text{i}; \\
\text{l}_{2} & \quad [\neg (\neg (\text{c} == 0))], [\neg (\text{c} == 0)], [\neg (\text{i} < 100)], [\neg (\text{i} < 100)] \\
\text{l}_{3} & \quad \text{i} = \text{i} + 1; \\
\text{l}_{4} & \quad [\neg (\text{i} < 100)], [\neg (\text{i} < 100)] \\
\text{l}_{5} & \quad \text{l}_{6} \quad \text{tmp}0 = \text{i} \\
\text{l}_{6} & \quad \text{l}_{7} \quad \text{i} = \text{i} + 1; \\
\text{l}_{7} & \quad \text{l}_{8} \quad \text{i} = \text{i} + 1; \\
\text{l}_{8} & \quad \text{l}_{9} \quad \text{c} = \text{tmp}0; \\
\text{l}_{9} & \quad \text{l}_{10} \quad [\text{c} >= \text{i}], [\neg (\text{c} >= \text{i})] \\
\text{l}_{10} & \quad \text{l}_{11} \quad \text{err} \\
\end{align*}
\]

\text{o/w} \quad q_{0} \quad (\text{l}_{0}, \text{i} = \text{nondet}();, \text{l}_{1}): \\
\quad \text{i} == 0 \quad \text{o/w} \quad q_{1} \quad (\text{l}_{5}, \text{i} = \text{i} + 1;, \text{l}_{3}): \\
\quad \text{true}
Example: ControlAutomatonCPA

\begin{align*}
l_0 & \quad \text{i = nondet();} \\
l_1 & \quad c = i; \\
l_2 & \quad [!((c == 0))], [!(c == 0)] \\
l_3 & \quad [!(i < 100)] \\
l_4 & \quad [i < 100] \\
l_5 & \quad l_6 \\
l_6 & \quad \text{tmp0 = i} \\
l_7 & \quad l_8 \\
l_8 & \quad c = \text{tmp0}; \\
l_9 & \quad [c >= i] \quad [!(c >= i)] \\
l_{err} & \\
l_{10} & \\
\end{align*}

\begin{align*}
q_0 & \quad (l_0, i = \text{nondet()};, l_1): \\
q_1 & \quad i == 0 \\
q_2 & \quad (l_5, i = i + 1;, l_3): \text{true} \\
\end{align*}
Example: ControlAutomatonCPA

\begin{itemize}
\item \((l_0, \emptyset, q_0)\)
\item \((l_1, \{i = 0\}, q_1)\)
\item \((l_2, \{i = 0, c = 0\}, q_1)\)
\item \([!(c == 0))\]
\item \((l_3, \{\ldots\}, q_1)\)
\item \([i < 100]\)
\item \((l_5, \{\ldots\}, q_1)\)
\item \(i = i + 1;\)
\item \((l_3, \{i = 1, c = 0\}, q_2)\)
\end{itemize}

\begin{verbatim}
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;
    }
}
\end{verbatim}

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

```
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\textsuperscript{4} and difference verification\textsuperscript{5}.


Other examples for program transformations in CPAchecker:

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

---

Future Work

In export, stay as close to original program as possible.
Thank you!