Counterexample-guided synthesis of controller logic from execution traces and temporal formulas

Daniil Chivilikhin, Igor Buzhinsky, Vladimir Ulyantsev, Andrey Stankevich, Anatoly Shalyto, Valeriy Vyatkin

IEEE ETFA 2018, Turin, Italy
5 September 2018
Program synthesis

- Derive implementation from examples/specification
  - From seminal work [A. Church, 1963]

- Motivation
  - Fundamental in computer science
  - Automation of software engineering
    - Reverse engineering
Reverse engineering of control software

- Rights limitations
- Changing standards
- Data loss

- Maintenance
- Reconfiguration
- Optimization
- Add new functionality
Addressed problem

Construct a model of a black-box logic controller using:

1. Execution traces derived from the controller
2. Temporal specification
3. Formal model of the plant
Target language: IEC 61499 basic function blocks
General reverse engineering scheme

**Preparation**

1. Tests gen
2. Tests
3. Temporal properties

**Simulation**

- Plant model

**Synthesis**

1. Execution traces
2. Model inference
3. Model
## Execution traces

<table>
<thead>
<tr>
<th>Input event</th>
<th>Input vars values</th>
<th>Output event</th>
<th>Output vars values</th>
</tr>
</thead>
</table>

\[
t_1 = \left[ \langle A[x_1 = 1, x_2 = 0], B[z_1 = 1, z_2 = 0] \rangle; \right.
\]
\[
\langle A[x_1 = 1, x_2 = 1], B[z_1 = 1, z_2 = 1] \rangle; \right.
\]
\[
\left. \langle A[x_1 = 0, x_2 = 0], B[z_1 = 0, z_2 = 1] \rangle \right]
\]
Temporal formulas

- Linear temporal logic (LTL)
- Allows expressing properties of programs referring to sequences of events/actions
- Temporal operators
  - \( \textbf{G} f \) – f holds for all states starting from the current one
  - \( \textbf{X} f \) – f holds for the next state
  - \( \textbf{F} f \) – f holds for some consequent state
  - \( G(x_1 \rightarrow F(z_2)) \) – if \( x_1 \) then eventually \( z_2 \)
Existing approaches

1. Metaheuristic: [Chivilikhin et al. / INDIN’16]
   - Rather slow
   - Incomplete

2. LTL synthesis [Faymonville et al. / TACAS’17]
   - Does not support execution traces
   - Large state machines

3. SAT-based incremental counterexample elimination [Ulyantsev et al. / STTT’18]
   - Uses a very simple state machine model
   - Insufficient for large # of input/output variables
Proposed approach: translation to Constraint Satisfaction Problem

1. Minimize #states, #transitions and the guard conditions of the state machine
2. Use closed-loop model checking for getting counterexamples

https://srlabs.de/bites/minisat-intro/
Proposed approach scheme

1. **# states N**
2. **# transitions R**
3. **Translation function**
4. **Constraints on variables V with domains D**
5. **CSP solver**
6. **No solution (UNSAT)**
7. **Automaton**
8. **Verification**
9. **LTL ok**
10. **Counter-example**
11. **Done.**

- **Positive traces**
- **Negative traces**
State machine model: transitions

\[ x_3 \land \neg x_5 \]

\[ \neg x_1 \land x_2 \]

\[ x_4 \]

<table>
<thead>
<tr>
<th>Source/destination</th>
<th>Input variable</th>
<th>Literal sign</th>
</tr>
</thead>
<tbody>
<tr>
<td>( r ) ( S_r )</td>
<td>( d_r )</td>
<td>( g_{*,1} )</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>
State machine model: states

State 1
\( z_1 := \text{FALSE}; \)
\( z_2 := \text{FALSE}; \)
\( z_3 := \text{FALSE}; \)

\[ \neg x_1 \land x_2 \]

State 2
\( z_1 := \text{TRUE}; \)
\( z_3 := \text{FALSE}; \)

<table>
<thead>
<tr>
<th>State</th>
<th>( z_1 )</th>
<th>( z_2 )</th>
<th>( z_3 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
</tr>
<tr>
<td>2</td>
<td>TRUE</td>
<td>*</td>
<td>FALSE</td>
</tr>
</tbody>
</table>
Positive traces tree

\[ t_1 = \left[ \langle A[x_1 = 1, x_2 = 0], B[z_1 = 1, z_2 = 0]\rangle; \right. \]
\[ \langle A[x_1 = 1, x_2 = 1], B[z_1 = 1, z_2 = 1]\rangle; \]
\[ \left. \langle A[x_1 = 0, x_2 = 0], B[z_1 = 0, z_2 = 1]\rangle \right] \]

\[ t_2 = \left[ \langle A[x_1 = 0, x_2 = 1], B[z_1 = 0, z_2 = 1]\rangle; \right. \]
\[ \langle A[x_1 = 0, x_2 = 0], B[z_1 = 1, z_2 = 1]\rangle; \]
\[ \left. \langle A[x_1 = 1, x_2 = 1], B[z_1 = 1, z_2 = 0]\rangle \right] . \]
Counterexamples & negative tree

LTL formula

\[ G(pp1 \rightarrow F(vp1)) \]

NuSMV model checker

Counterexample

Negative tree
Main negative tree constraints (1)

Propagate color $c_2$ to node $v$ if:

1. Vertex $u$ has color $c_1$
2. Automaton has corresponding transition
3. State $c_2$ correctly transforms output variables

$z_u = \langle 0011 \rangle$

$x_1 = 1$
$x_2 = 0$
$x_3 = 1$

$z_v = \langle 1111 \rangle$

$a_{c_2} = \langle 11 \ast \ast \rangle$

$\langle 11 \ast \ast \rangle \cdot \langle 0011 \rangle = \langle 1111 \rangle$
Main negative tree constraints (2)

- Forbid counterexamples, same as in [Ulyantsev et al., 2018]
- Colors of nodes connected with a back-edge must be different: $n(u) \neq n(v)$
Main algorithm

**Algorithm 1:** Iterative FB model inference

**Data:** positive traces tree $T^+$, temporal properties $f$

$T^- \leftarrow \emptyset$  // negative traces tree

$N \leftarrow 2$, $R \leftarrow 1$

**while** True **do**

**A** $\leftarrow$ findModel($T^+, T^-, N, R$)

**if** $A = \emptyset$  // solver returned UNSAT **then**

**if** $R \leq 2N + 1$ **then** $R \leftarrow R + 1$

**else** $N \leftarrow N + 1$

**else**

$C \leftarrow$ verify($A$, $f$)

**if** $C = \emptyset$ **then** **return** $A$

**else** $T^-$.add($C$)
Case study: Pick-and-Place manipulator

Goal – generate controller model from
1. Small set of traces
2. LTL specification
LTL specification

<table>
<thead>
<tr>
<th>Property</th>
<th>Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\varphi_1$</td>
<td>$G(\neg(c1\text{Extend} \land c1\text{Retract}))$</td>
</tr>
<tr>
<td>$\varphi_2$</td>
<td>$G(\neg(c2\text{Extend} \land c2\text{Retract}))$</td>
</tr>
<tr>
<td>$\varphi_3$</td>
<td>$G(\neg(vacuum_on \land vacuum_off))$</td>
</tr>
<tr>
<td>$\varphi_4$</td>
<td>$G(\neg vc\text{Home} \land \neg vc\text{End} \rightarrow c1\text{Home} \lor c1\text{End})$</td>
</tr>
<tr>
<td>$\varphi_5$</td>
<td>$G(\neg c1\text{Home} \land \neg c1\text{End} \rightarrow vc\text{Home} \lor vc\text{End})$</td>
</tr>
<tr>
<td>$\varphi_6$</td>
<td>$G(pp1 \rightarrow F(vp1))$</td>
</tr>
<tr>
<td>$\varphi_7$</td>
<td>$G(lifted \rightarrow F(dropped))$</td>
</tr>
<tr>
<td>$\varphi_8$</td>
<td>$G(all_home \land \neg pp1 \land \neg lifted \rightarrow X(\neg c1\text{Extend} \land \neg c2\text{Extend} \land \neg vc\text{Extend}))$</td>
</tr>
</tbody>
</table>

WP is always eventually lifted from slider 1
Positive traces generation

All tests with length = 1, 2, 3

#1: 1
#2: 2
...
#12: 1, 1, 1
...
#39: 3, 3, 3

Execution trace

REQ [0, 1, 0, 1, 0, 0, 0, 1, 0], CNF [1, 0, 1, 0, 1, 0, 0]
## Experimental results

<table>
<thead>
<tr>
<th></th>
<th>Set 4–39</th>
<th>Set 1–3</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Cover” property $G(pp1 \rightarrow F(vp1))$</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td><strong>Time</strong></td>
<td>376 s (92 – 858)</td>
<td>8h – 18h</td>
</tr>
<tr>
<td><strong># iterations</strong></td>
<td>11 (10-15)</td>
<td>461 – 649</td>
</tr>
<tr>
<td><strong>Negative tree size</strong></td>
<td>54 (42 – 84)</td>
<td>4067 – 5079</td>
</tr>
</tbody>
</table>
• Results were validated by closed-loop simulation of generated controllers in NxtStudio
Conclusion & Future work

• Developed a counterexample-guided approach for synthesizing controllers from behavior examples and temporal logic specification
• Demonstrated its viability on the example of Pick-and-Place manipulator IEC 61499 controller synthesis

Future work
• Use automatically generated plant models [Buzhinsky et al. / TII’17; Ovsianikova et al., ETFA’18]
• Use SPIN instead of NuSMV [Buzhinsky et al. / IECON’17]
• Use incremental SAT-solver instead of CSP-solver
• New case studies
Thank you for your attention!