Chaining Test Cases for Reactive System Testing

Peter Schrammell
peter.schrammel@cs.ox.ac.uk
(joint work with Tom Melham and Daniel Kroening)

University of Oxford

Group Seminar, June 2013
Reactive System Testing

Context:
- Safety critical embedded software
- Often modelled as synchronous reactive system
- Safety standards: tool support for systematic testing desirable

Problem:
- Often lengthy input sequences required to drive the system to a test goal
- Reset after each test case: serious problem in on-target testing

Goal:
- Find a test case chain: a single test case that covers a set of test goals and minimises overall test execution time
**Example: Generated C Code from Simulink**

```c
void init(t_state *s) {
    s->mode = OFF;
    s->speed = 0;
    s->enable = FALSE;
}

void compute(t_input *i, t_state *s) {
    mode = s->mode;
    switch (mode) {
        case ON: if (i->gas || i->brake) s->mode = DIS; break;
        case DIS:
            if ((s->speed == 2 && (i->dec || i->brake)) ||
                (s->speed == 0 && (i->acc || i->gas)))
                s->mode = ON;
            break;
        case OFF:
            if (s->speed == 0 && s->enable && (i->gas || i->acc)) ||
                s->speed == 1 && i->button ||
                s->speed == 2 && s->enable && (i->brake || i->dec))
                s->mode = ON;
            break;
    }
    if (i->button) s->enable = !s->enable;

    if ((i->gas || mode == ON && i->acc) && s->speed < 2) s->speed++;
    if ((i->brake || mode == ON && i->dec) && s->speed > 0) s->speed--;
}
```
Example
Example

\[ I = F \]

- \( \text{OFF,0,FALSE} \)
- \( \text{OFF,1,FALSE} \)
- \( \text{OFF,2,FALSE} \)
- \( \text{OFF,0,TRUE} \)
- \( \text{OFF,1,TRUE} \)
- \( \text{OFF,2,TRUE} \)
- \( \text{DIS,0,TRUE} \)
- \( \text{DIS,2,TRUE} \)

\( \text{button (p1)} \)
\( \text{button (p2)} \)
\( \text{button (p3)} \)
\( \text{button (p4)} \)

\( \text{gas} \lor \text{acc} \)
\( \text{brake} \lor \text{dec} \)

Schrammel (Oxford)

Test Chains

June 2013
Outline

1 Introduction

2 Preliminaries

3 Chaining Test Cases

4 Generalisations

5 Experimental Evaluation

6 Conclusions
Preliminaries

Program:
- State space $\Sigma$, input space $\Upsilon$
- Initial states $I \subseteq \Sigma$
- Transition relation $T \subseteq \Sigma \times \Upsilon \times \Sigma$

Bounded Model Checking:
Check the existence of a path $\langle s_0, s_1, \ldots, s_K \rangle$ of increasing length $K$ from $\phi$ to $\phi'$

$$
\phi(s_0) \land \bigwedge_{1 \leq k \leq K} T(s_{k-1}, i_{k-1}, s_k) \land \phi'(s_K)
$$

If SAT: satisfying assignment aka counterexample $(s_0, i_0, s_1, i_1, \ldots, s_{K-1}, i_{K-1}, s_K)$

Test case generation:
- $\phi = I$ and test goal $\phi'$
- Test case: input sequence $\langle i_0, \ldots, i_{K-1} \rangle$
Chaining Test Cases

Temporal logic safety specification:
- Set of properties, e.g., of type

\[ G(\text{mode} = \text{ON} \land \text{speed} = 1 \land \text{dec} \Rightarrow \text{X(speed} = 1)) \]

Test goals: set of assumptions

Approach
1. Abstraction: property reachability graph
2. Optimisation: shortest path
3. Concretisation: compute concrete test case
Abstraction: Property Reachability Graph

- **Nodes**: property assumptions $\varphi$
- **Edges**:
  - from $I$ to all $\varphi$s
  - from all $\varphi$s to $F$
  - pairwise links between $\varphi$s

Incrementally build graph by reachability queries:
Abstraction: Property Reachability Graph

- Nodes: property assumptions $\varphi$
- Edges:
  - from $I$ to all $\varphi$s
  - from all $\varphi$s to $F$
  - pairwise links between $\varphi$s

Incrementally build graph by reachability queries: $K = 1$
Abstraction: Property Reachability Graph

- **Nodes**: property assumptions $\varphi$
- **Edges**:
  - from $I$ to all $\varphi$s
  - from all $\varphi$s to $F$
  - pairwise links between $\varphi$s

Incrementally build graph by reachability queries: $K = 2$
Existence of a Covering Path

Covering path: path that visits all nodes at least once.

There is a covering path from $I$ to $F$ iff

1. all nodes are reachable from $I$,
2. $F$ is reachable from all nodes, and
3. for all pairs of nodes $(\nu_1, \nu_2)$,
   (a) $\nu_2$ is reachable from $\nu_1$ or
   (b) $\nu_1$ is reachable from $\nu_2$.

Reachability can be decided in constant time on the transitive closure of the graph.
Existence of a Covering Path

**Covering path:** path that visits all nodes at least once.

There is a covering path from $I$ to $F$ iff

1. all nodes are reachable from $I$,
2. $F$ is reachable from all nodes, and
3. for all pairs of nodes $(v_1, v_2)$,
   - (a) $v_2$ is reachable from $v_1$ or
   - (b) $v_1$ is reachable from $v_2$.

Reachability can be decided in constant time on the transitive closure of the graph.
Existence of a Covering Path

**Covering path:** path that visits all nodes at least once.

There is a covering path from $I$ to $F$ iff

1. all nodes are reachable from $I$,
2. $F$ is reachable from all nodes, and
3. for all pairs of nodes $(v_1, v_2)$,
   - (a) $v_2$ is reachable from $v_1$ or
   - (b) $v_1$ is reachable from $v_2$.

Reachability can be decided in constant time on the transitive closure of the graph.
Optimisation: Shortest Path Computation

Our problem:
- Find a covering path from $I$ to $F$

Approach:
- Reduce to asymmetric travelling salesman problem (ATSP):
  - Tour that visits all nodes of a weighted directed graph exactly once
- Transitive closure

ATSP result: $\langle \varphi_2, \varphi_3, F, I, \varphi_4, \varphi_1 \rangle$
Shortest path: $\langle I, \varphi_4, \varphi_1, \varphi_2, \varphi_3, F \rangle$
Concretisation: Computing the Test Chain

\[ I \rightarrow^2 \varphi_4 \rightarrow^2 \varphi_1 \rightarrow^2 \varphi_2 \rightarrow^1 \varphi_3 \rightarrow^2 F \]

\[ I(s_0) \wedge T(s_0, i_0, s_1) \wedge T(s_1, i_1, s_2) \wedge \varphi_4(s_2, i_2) \]
\[ \wedge T(s_2, i_2, s_3) \wedge T(s_3, i_3, s_4) \wedge \varphi_1(s_4, i_4) \]
\[ \wedge T(s_4, i_4, s_5) \wedge T(s_5, i_5, s_6) \wedge \varphi_2(s_6, i_6) \]
\[ \wedge T(s_6, i_6, s_7) \wedge \varphi_3(s_7, i_7) \]
\[ \wedge T(s_7, i_7, s_8) \wedge T(s_8, i_8, s_9) \wedge F(s_9) \]

\[ \langle i_0, \ldots, i_8 \rangle = \langle \text{gas, acc, button, dec, dec, gas, dec, brake, button} \rangle \]
Concretisation: Computing the Test Chain

- \( I = F \)
- \( \text{OFF,0,FALSE} \)
- \( \text{OFF,1,FALSE} \)
- \( \text{OFF,2,FALSE} \)
- \( \text{OFF,0,TRUE} \)
- \( \text{OFF,1,TRUE} \)
- \( \text{OFF,2,TRUE} \)
- \( \text{DIS,0,TRUE} \)
- \( \text{DIS,2,TRUE} \)

Transition rules:

- \( \text{OFF,0,FALSE} \rightarrow \text{OFF,1,FALSE} \) via \( \text{button} \) (\( \text{p}_1 \))
- \( \text{OFF,1,FALSE} \rightarrow \text{DIS,0,TRUE} \) via \( \text{button} \) (\( \text{p}_2 \))
- \( \text{OFF,0,TRUE} \rightarrow \text{ON,1,TRUE} \) via \( \text{button} \) (\( \text{p}_3 \))
- \( \text{DIS,2,TRUE} \rightarrow \text{OFF,2,TRUE} \) via \( \text{button} \) (\( \text{p}_4 \))

Symbols:

- \( \text{gas} \lor \text{acc} \)
- \( \text{brake} \lor \text{dec} \)

Note: The diagram shows the transitions and conditions for each state. The arrows represent the conditions under which a transition occurs.
Optimality

The test case chain is minimal if

1. the program and the properties admit a test chain,
2. all property assumptions are singleton sets, and
3. the test chain visits each property once in the $K$-reachability graph.

Reachability diameter $d = \text{length of maximum, shortest path between any two states}$

There is a $K \leq d$ such that, under the preconditions (1) and (2), the test chain is minimal.
Optimality

The test case chain is minimal if

(1) the program and the properties admit a test chain,
(2) all property assumptions are singleton sets, and
(3) the test chain visits each property once in the $K$-reachability graph.

Reachability diameter $d = \text{length of maximum, shortest path between any two states}$

There is a $K \leq d$ such that, under the preconditions (1) and (2), the test chain is minimal.
Generalisations

Multi-state property assumptions
- Concretisation may fail $\rightarrow$ chain repair

Drop all preconditions except existence of a single chain
- Completeness lost $\rightarrow$ abstraction refinement

Multiple chains
- Find a partition of properties
Multi-State Property Assumptions

Broken chain:
- Path \( \langle I, \varphi_1, \varphi_2 \rangle \) not feasible in a single step, but requires two steps.

Chain repair:
- Systematically increase edge weights of failed subpath
- Minimality lost

Completeness:
- The chain repair succeeds if the given path admits a chain in the concrete program.
- If for each property assumption the states are strongly connected and there exists a single test chain then we will find it.
Multi-State Property Assumptions

Broken chain:
- Path \( \langle I, \varphi_1, \varphi_2 \rangle \) not feasible in a single step, but requires two steps.

Chain repair:
- Systematically increase edge weights of failed subpath
- Minimality lost

Completeness:
- The chain repair succeeds if the given path admits a chain in the concrete program.
- If for each property assumption the states are strongly connected and there exists a single test chain then we will find it.
Multi-State Property Assumptions

Broken chain:
- Path $\langle I, \varphi_1, \varphi_2 \rangle$ not feasible in a single step, but requires two steps.

Chain repair:
- Systematically increase edge weights of failed subpath
- Minimality lost

Completeness:
- The chain repair succeeds if the given path admits a chain in the concrete program.
- If for each property assumption the states are strongly connected and there exists a single test chain then we will find it.
Ensuring Completeness

\[ I \xrightarrow{0} \varphi_1 \xrightarrow{1} \varphi_2 \xrightarrow{2} F \]
Ensuring Completeness

Abstraction refinement:
Ensuring Completeness

Abstraction refinement: Find any path
Ensuring Completeness

Abstraction refinement: Optimise with TSP solver
Ensuring Completeness

Abstraction refinement: Optimise with TSP solver
Multiple Chains

No single chain if

- the $N$-reachability property graph has no chain, or
- the fully refined property graph has no chain.

Necessary condition for existence of multiple chains:

(1) Each property reachable from $I$ and
(2) $F$ reachable from each property.

Additional necessary condition for single chains:

for all pairs of vertices $(v_1, v_2)$,

(a) $v_2$ is reachable from $v_1$ or
(b) $v_1$ is reachable from $v_2$.

$\implies$ **Partition** by graph colouring
Generalisations

Multiple Chains

Necessary condition for single chains: for all pairs of vertices \((v_1, v_2)\),

(a) \(v_2\) is reachable from \(v_1\) or

(b) \(v_1\) is reachable from \(v_2\).
Multiple Chains

Necessary condition for single chains: for all pairs of vertices \((v_1, v_2)\),

(a) \(v_2\) is reachable from \(v_1\) or

(b) \(v_1\) is reachable from \(v_2\).

Pairs of vertices violating this condition: \((1, 2), (3, 4), (1, 4)\)
Multiple Chains

Necessary condition for single chains:
for all pairs of vertices \((v_1, v_2)\),
(a) \(v_2\) is reachable from \(v_1\) or
(b) \(v_1\) is reachable from \(v_2\).

Pairs of vertices violating this condition: \((1, 2), (3, 4), (1, 4)\)
Partition: \(\{1, 3\} \{2, 4\}\)
Multiple Chains

Necessary condition for single chains:
for all pairs of vertices \((v_1, v_2)\),

(a) \(v_2\) is reachable from \(v_1\) or

(b) \(v_1\) is reachable from \(v_2\).

Pairs of vertices violating this condition: \((1, 2), (3, 4), (1, 4)\)
Partition: \(\{1, 3\} \{2, 4\}\)
Implementation

![Diagram showing the implementation process]

- Reactive system model
- C code generator
- Static analyser
- ChainCover
- Test suite
Implementation

Properties specified as C functions:

```c
void p_1(t_input* i, t_state* s) {
    _CPROVER_assume(s->mode==ON && s->speed==1 && i->dec);
    compute(i,s);
    assert(s->speed==1);
}
```

BMC engine of C\textsc{Bmc}

- Property reachability graph construction like \texttt{--all-claims}
- Chain repair by concrete chaining

L\textsc{Kh} travelling salesman problem solver
Benchmarks and Comparison

Benchmarks

- Cruise control model
- Window controller
- Car alarm system
- Elevator model
- Robot arm model

Comparison with

- **FShell**: a BMC-based test generator with test suite minimisation
- Random case generator with test suite minimisation
- **KLEE**: a test case generator based on symbolic execution
Results: Test Case Length

- **KLEE**
- **RANDOMTEST**
- **FSHELL**
- **CHAINCOVER**
Results: Test Case Generator Runtime

- □ KLEE
- ◊ RANDOMTest
- * FSHELL
- △ CHAINCover

Accumulated runtimes vs. Number of benchmarks
Related Work

Test case generation with model checkers

Reactive system testing

- **Random testing**: mostly state of practice
- Random testing with **symbolic and concrete execution** to guide exhaustive path enumeration: DART (Godefroid et al.), CUTE (Sen and Agha), KLEE (Cadar et al.)
- **Scenario-based testing**: test specifications to guide test case generation towards a particular functionality, static analysis and random test case generation, LUTESS, LURETTE, LUTIN (Du Bousquet, Raymond, Jahier et al.)
- **Model-based testing**: exhaustive test case generation from labelled transition system-based specification models TGV (Jard and Jéron), TORX (Tretmans)

Minimal checking sequences (Hierons, Ural, Duale, Petrenko et al.)

- Conformance testing of a specification model against a black box implementation
- Minimal path, genetic algorithms to minimise feasible path, random input generation to trigger the path
Summary and Prospects

Summary

- Test chain for reactive systems
  - Test goals from requirements, specification model, code coverage criteria
  - Equally applicable to model-based and structural coverage-based testing
- Minimal test chain for single-state property assumptions, otherwise heuristics
- Experimental evaluation

Future work

- Acceleration to handle deep loops
- Better exploit incremental BMC and SAT
- Optimality in general case