Chaining Test Cases for Reactive System Testing

Peter Schrammel, Tom Melham and Daniel Kroening
first.lastname@cs.ox.ac.uk

UNIVERSITY OF OXFORD

The 25th IFIP International Conference on Testing Software and Systems (ICTSS'13)

Nov 13-15, 2013, Istanbul, Turkey

Test Chains

Context:

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

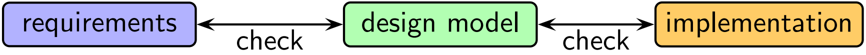
Problem:

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

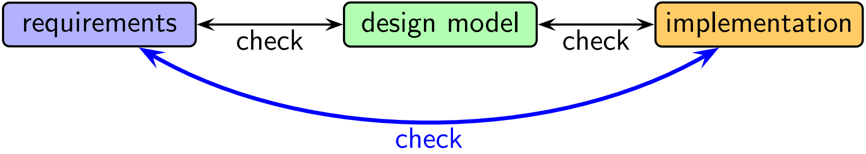
Goal:

- Find a test case chain: a single test case that covers a set of test goals and minimises overall test execution time

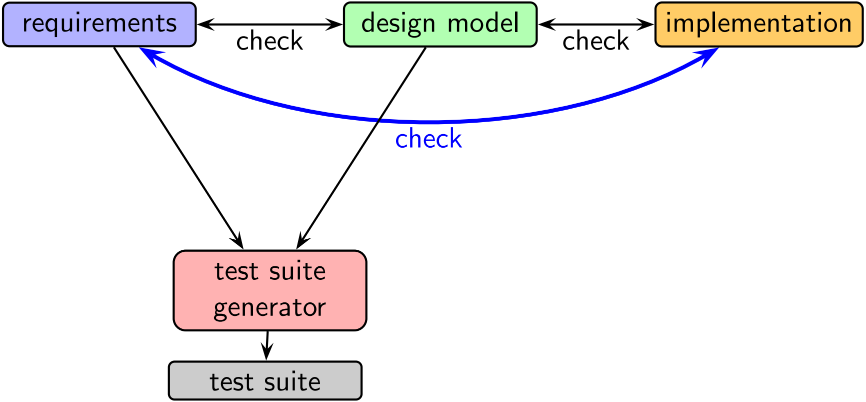
Model-Based Testing



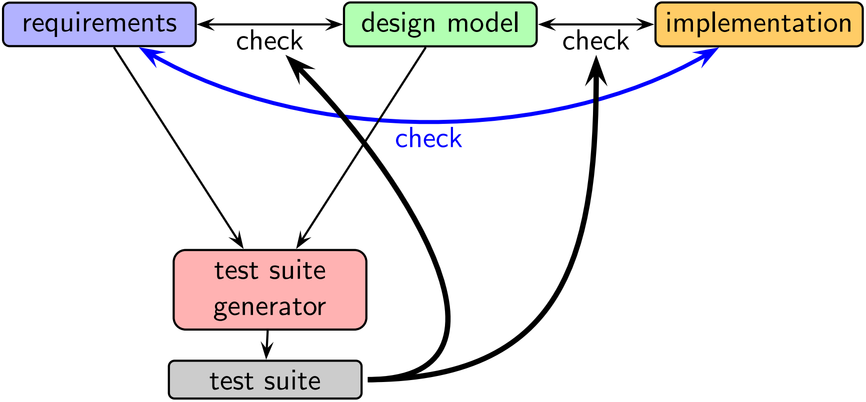
Model-Based Testing



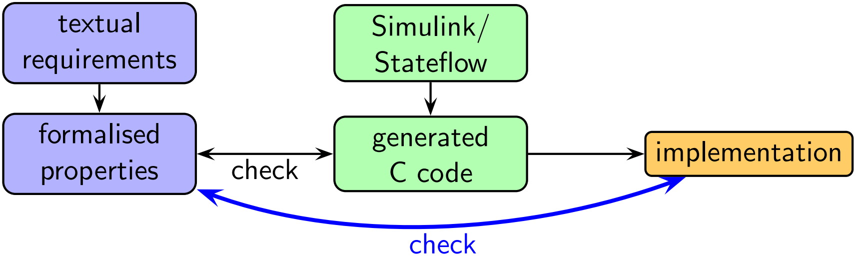
Model-Based Testing



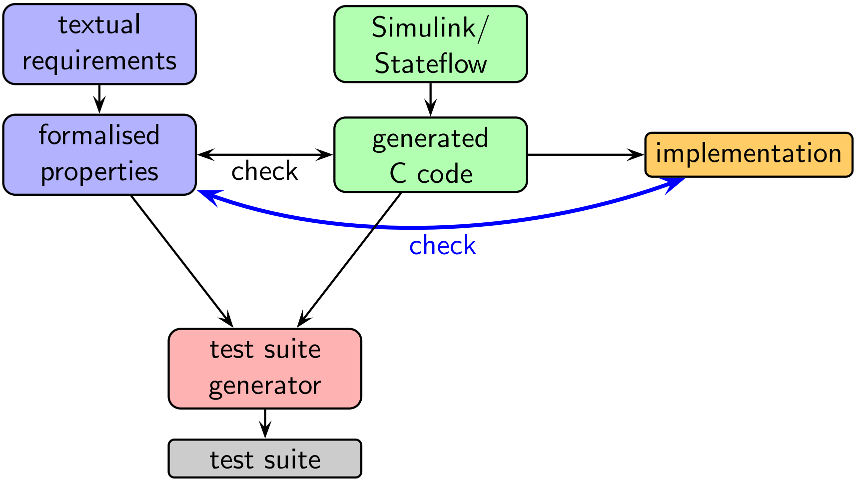
Model-Based Testing



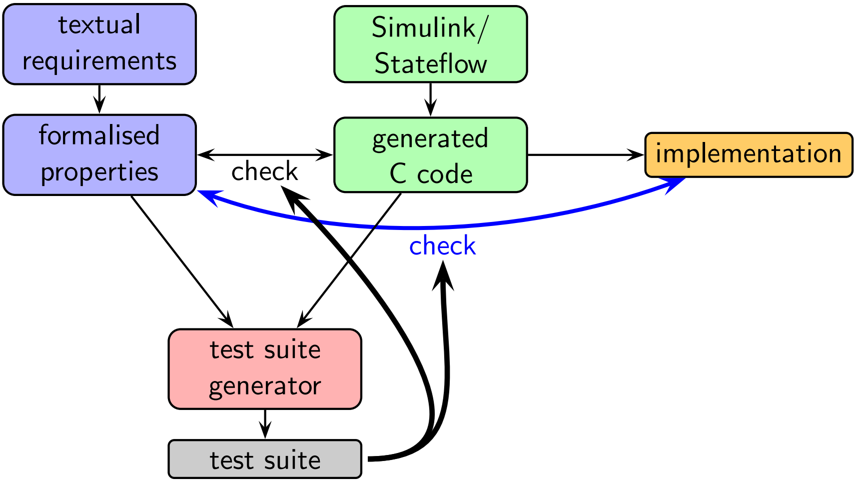
Model-Based Testing



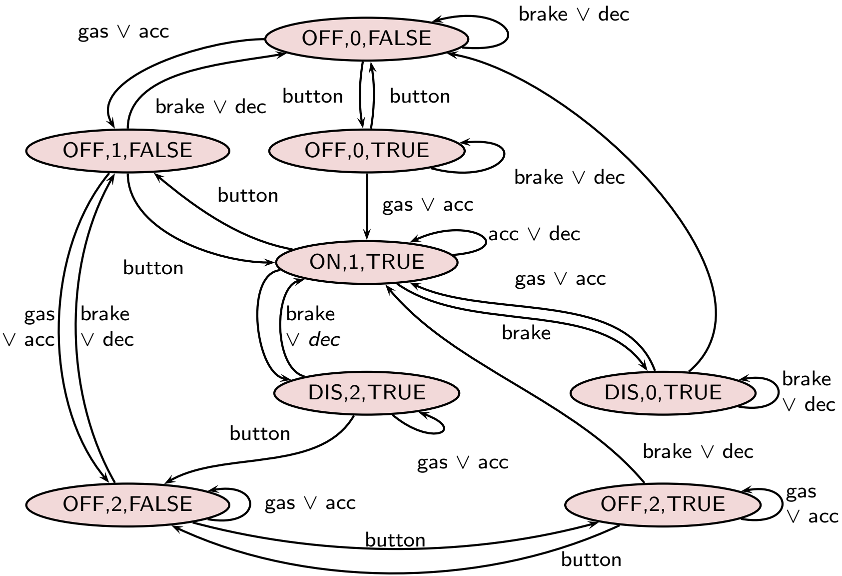
Model-Based Testing



Model-Based Testing



Example: Cruise Control



Example: Generated C Code from SIMULINK

```c
void init(state_t *s) {
  s->mode = OFF;
  s->speed = 0;
  s->enable = FALSE;
}
void compute(io_t *i, state_t *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: Generated C Code from SIMULINK

```
void init(state_t *s) {
  s->mode = OFF;
  s->speed = 0;
  s->enable = FALSE;
}
void compute(io_t *i, state_t *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==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--;
}
```

Formalised properties:

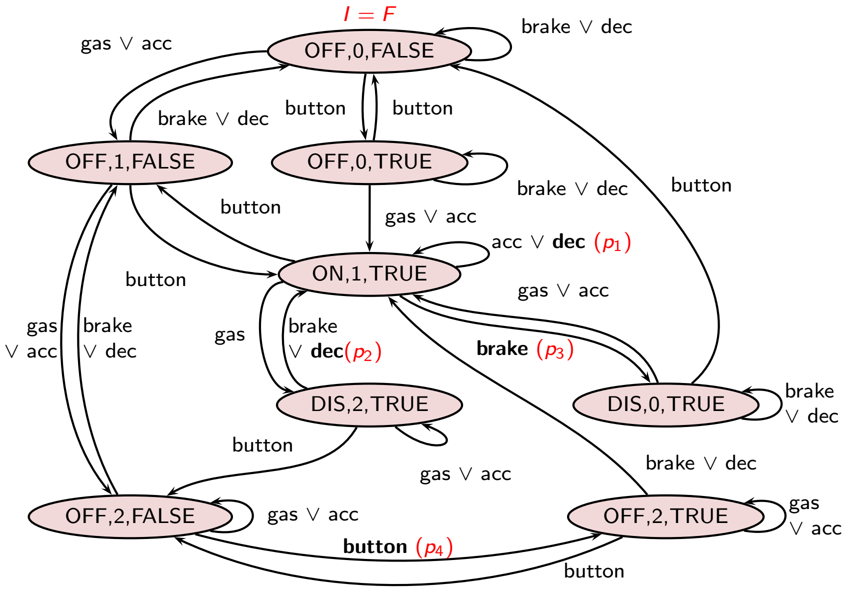
p_1: $\mathbf{G}\big(\mathit{mode} = \mathit{ON} \wedge \mathit{speed} = 1 \wedge \mathit{dec} \Rightarrow \mathbf{X}(\mathit{speed} = 1)\big)$

p_2: $\mathbf{G}\big(\mathit{mode} = \mathit{DIS} \wedge \mathit{speed} = 2 \wedge \mathit{dec} \Rightarrow \mathbf{X}(\mathit{mode} = \mathit{ON})\big)$

p_3: $\mathbf{G}\big(\mathit{mode} = \mathit{ON} \wedge \mathit{brake} \Rightarrow \mathbf{X}(\mathit{mode} = \mathit{DIS})\big)$

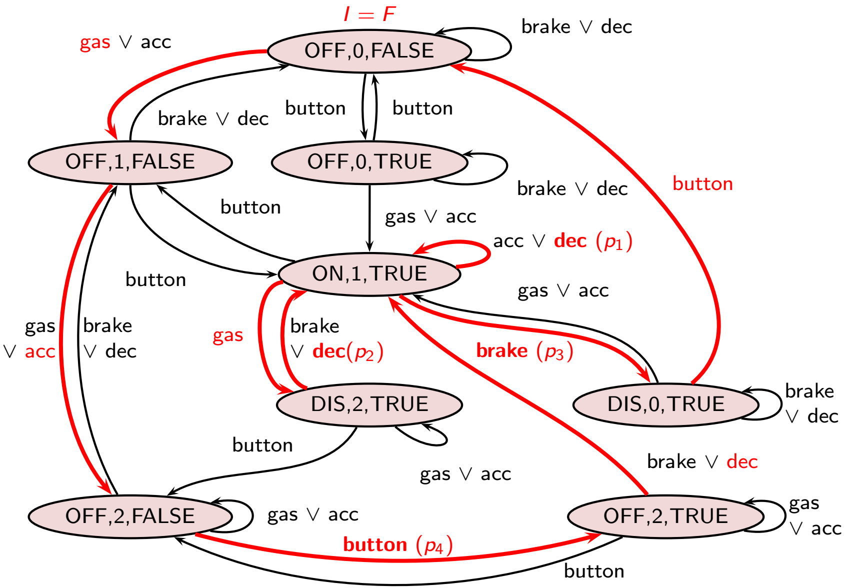
p_4: $\mathbf{G}\big(\mathit{mode} = \mathit{OFF} \wedge \mathit{speed} = 2 \wedge \neg\mathit{enable} \wedge \mathit{button} \Rightarrow \mathbf{X}\ \mathit{enable}\big)$

Example



$I = F$
OFF,0,FALSE
brake ∨ dec
gas ∨ acc
button
button
brake ∨ dec
OFF,1,FALSE
OFF,0,TRUE
brake ∨ dec
button
button
gas ∨ acc
acc ∨ **dec** (p_1)
ON,1,TRUE
button
gas ∨ acc
gas ∨ acc
brake ∨ dec
gas
brake ∨ **dec**(p_2)
brake (p_3)
DIS,2,TRUE
DIS,0,TRUE
brake ∨ dec
button
gas ∨ acc
brake ∨ dec
OFF,2,FALSE
gas ∨ acc
OFF,2,TRUE
gas ∨ acc
button (p_4)
button

Example



I = F
OFF,0,FALSE
brake ∨ dec
gas ∨ acc
brake ∨ dec
button
button
OFF,1,FALSE
OFF,0,TRUE
brake ∨ dec
button
button
gas ∨ acc
acc ∨ dec (p1)
ON,1,TRUE
button
gas ∨ acc
gas
∨ acc
brake
∨ dec
gas
brake
∨ dec(p2)
brake (p3)
DIS,2,TRUE
DIS,0,TRUE
brake
∨ dec
button
brake ∨ dec
gas ∨ acc
OFF,2,FALSE
gas ∨ acc
OFF,2,TRUE
gas
∨ acc
button (p4)
button

Preliminaries

Program:

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

Preliminaries

Program:

- State space Σ, input space Υ
- 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 ϕ to ϕ'

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

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

Preliminaries

Program:

- State space Σ, input space Υ
- 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 ϕ to ϕ'

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

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 ϕ'
- Test case: input sequence $\langle i_0, \ldots, i_{K-1} \rangle$, expected outcome

Chaining Test Cases

Temporal logic safety specification:

- Set of properties, *e.g.*, of type

$$\mathbf{G}\big(\underbrace{\mathit{mode} = \mathit{ON} \wedge \mathit{speed} = 1 \wedge \mathit{dec}}_{\text{assumption } \varphi} \Rightarrow \mathbf{X}(\mathit{speed} = 1)\big)$$

Chaining Test Cases

Temporal logic safety specification:

- Set of properties, *e.g.*, of type

$$\mathbf{G}\big(\underbrace{mode = ON \wedge speed = 1 \wedge dec}_{\text{assumption } \varphi} \Rightarrow \mathbf{X}(speed = 1)\big)$$

Test goals: set of assumptions φ (finite paths)

Chaining Test Cases

Temporal logic safety specification:

- Set of properties, *e.g.*, of type

$$\mathbf{G}\big(\underbrace{mode = ON \wedge speed = 1 \wedge dec}_{\text{assumption } \varphi} \Rightarrow \mathbf{X}(speed = 1)\big)$$

Test goals: set of assumptions φ (finite paths)

Test chain: from initial states I via all φs to final states F

Chaining Test Cases

Temporal logic safety specification:

- Set of properties, *e.g.*, of type

$$\mathbf{G}\Big(\underbrace{mode = ON \land speed = 1 \land dec}_{\text{assumption } \varphi} \Rightarrow \mathbf{X}(speed = 1)\Big)$$

Test goals: set of assumptions φ (finite paths)

Test chain: from initial states I via all φs to final states F

Approach

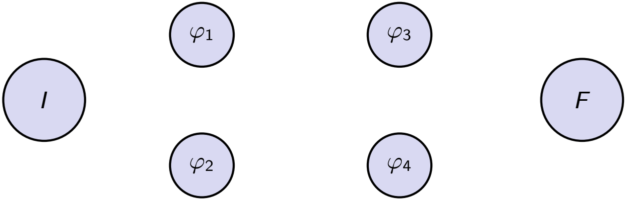
1. Abstraction: property reachability graph
2. Optimisation: shortest path
3. Concretisation: compute concrete test case

Abstraction: Property Reachability Graph

Weighted, directed graph:

- Nodes: test goals φ
- Edges:
  - from I to all φs
  - from all φs to F
  - pairwise links between φs
- Edge weights: number of execution steps

Incrementally build graph by reachability queries:

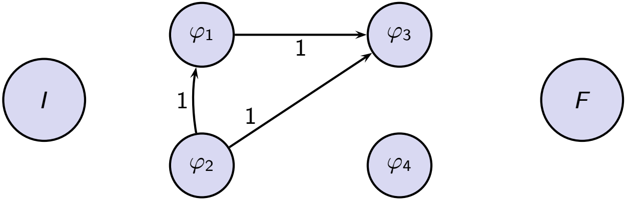


Abstraction: Property Reachability Graph

Weighted, directed graph:

- Nodes: test goals φ
- Edges:
  - from I to all φs
  - from all φs to F
  - pairwise links between φs
- Edge weights: number of execution steps

Incrementally build graph by reachability queries: $K = 1$

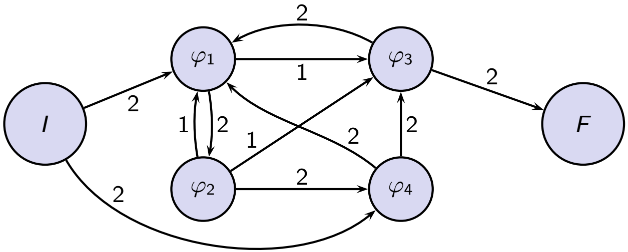


Abstraction: Property Reachability Graph

Weighted, directed graph:

- Nodes: test goals φ
- Edges:
  - from I to all φs
  - from all φs to F
  - pairwise links between φs
- Edge weights: number of execution steps

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

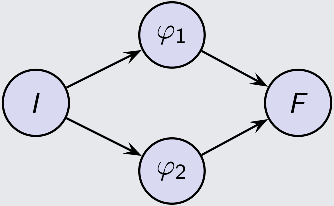
(1) all nodes are reachable from I,

(2) F is reachable from all nodes, and

(3) for all pairs of nodes (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

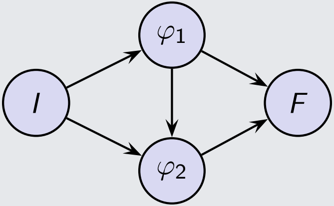
(1) all nodes are reachable from I,

(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.

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

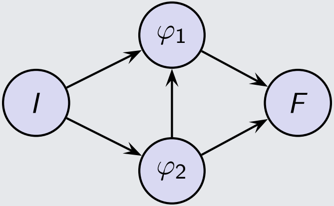
(1) all nodes are reachable from I,

(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.

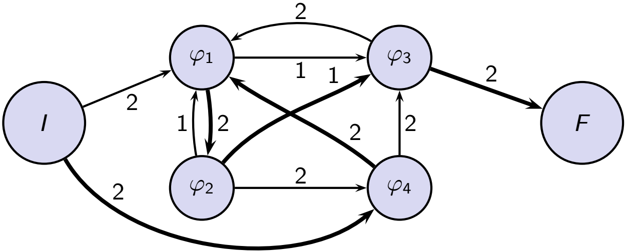


Reachability can be decided in constant time on the transitive closure of the graph.

Optimisation: Shortest Path Computation

Find a covering path from I to F:

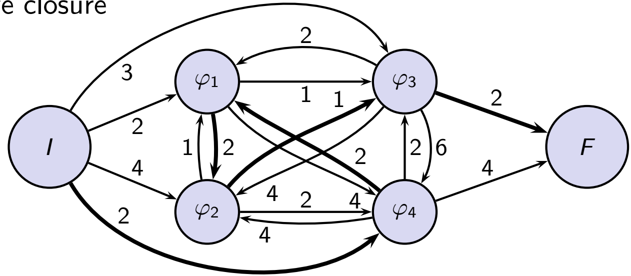
- Reduce to asymmetric travelling salesman problem (ATSP):
    - Tour that visits all nodes of a weighted directed graph exactly once
- Transitive closure



Optimisation: Shortest Path Computation

Find a covering path from I to F:

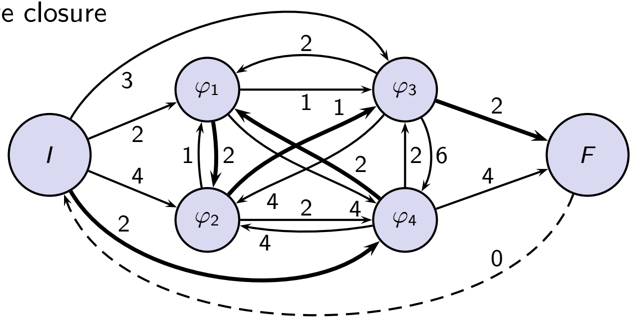
- Reduce to asymmetric travelling salesman problem (ATSP):
  - Tour that visits all nodes of a weighted directed graph exactly once
- Transitive closure



Optimisation: Shortest Path Computation

Find a covering path from I to F:

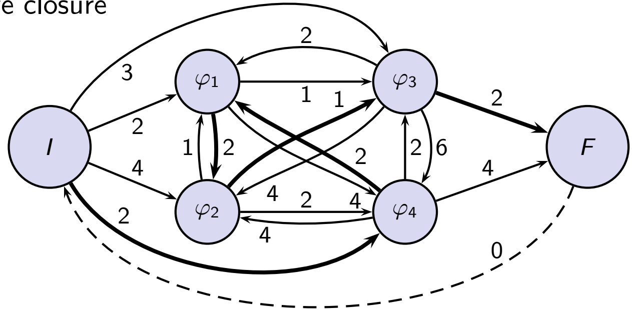
- Reduce to asymmetric travelling salesman problem (ATSP):
  - Tour that visits all nodes of a weighted directed graph exactly once
- Transitive closure



Optimisation: Shortest Path Computation

Find a covering path from I to F:

- 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: $\langle I, \varphi_4, \varphi_1, \varphi_2, \varphi_3, F \rangle$

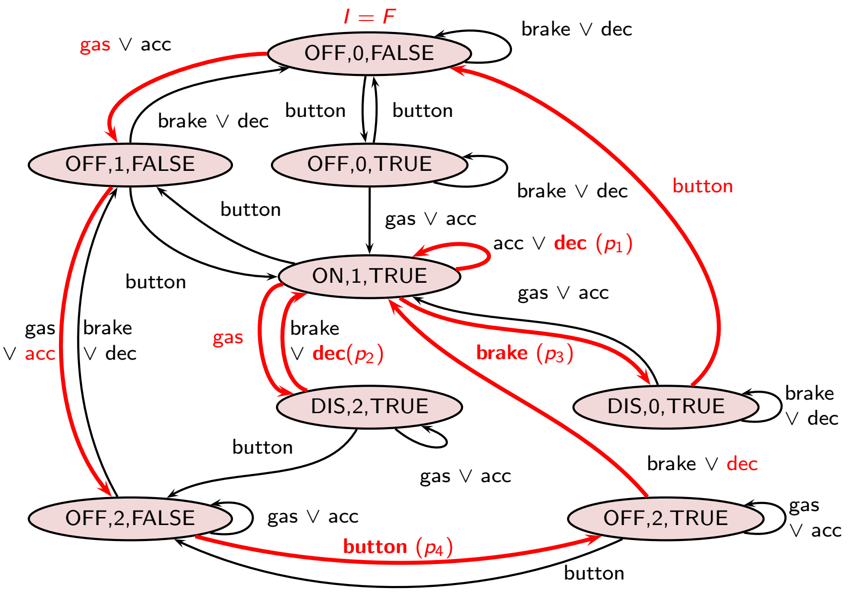
Concretisation: Computing the Test Chain

$$I \xrightarrow{2} \varphi_4 \xrightarrow{2} \varphi_1 \xrightarrow{2} \varphi_2 \xrightarrow{1} \varphi_3 \xrightarrow{2} F$$

$$\begin{aligned}
&& I(s_0) \\
\wedge T(s_0, i_0, s_1) \wedge T(s_1, i_1, s_2) &\wedge& \varphi_4(s_2, i_2) \\
\wedge T(s_2, i_2, s_3) \wedge T(s_3, i_3, s_4) &\wedge& \varphi_1(s_4, i_4) \\
\wedge T(s_4, i_4, s_5) \wedge T(s_5, i_5, s_6) &\wedge& \varphi_2(s_6, i_6) \\
\wedge T(s_6, i_6, s_7) &\wedge& \varphi_3(s_7, i_7) \\
\wedge T(s_7, i_7, s_8) \wedge T(s_8, i_8, s_9) &\wedge& F(s_9)
\end{aligned}$$

$$\langle i_0, \ldots, i_8 \rangle = \langle \mathit{gas},\ \mathit{acc},\ \mathit{button},\ \mathit{dec},\ \mathit{dec},\ \mathit{gas},\ \mathit{dec},\ \mathit{brake},\ \mathit{button} \rangle$$

Concretisation: Computing the Test Chain



Optimality

The test case chain is *minimal* if

(1) the program and the properties admit a test chain,

(2) all test goals are singleton sets, and

(3) the test chain visits each property once in the K-reachability graph.

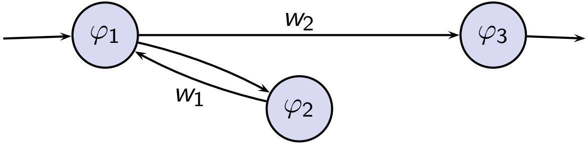
Optimality

The test case chain is *minimal* if

(1) the program and the properties admit a test chain,

(2) all test goals are singleton sets, and

(3) the test chain visits each property once in the K-reachability graph.



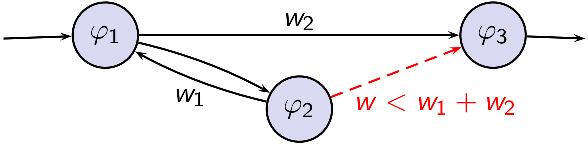
Optimality

The test case chain is *minimal* if

(1) the program and the properties admit a test chain,

(2) all test goals are singleton sets, and

(3) the test chain visits each property once in the K-reachability graph.



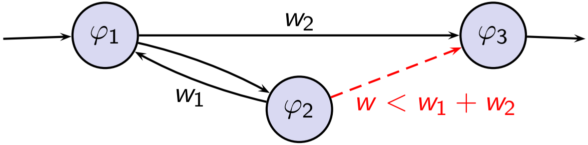
Optimality

The test case chain is *minimal* if

(1) the program and the properties admit a test chain,

(2) all test goals are singleton sets, and

(3) the test chain visits each property once in the K-reachability graph.



Reachability diameter d = 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.

In practice, fix a bound K and obtain minimised chain.

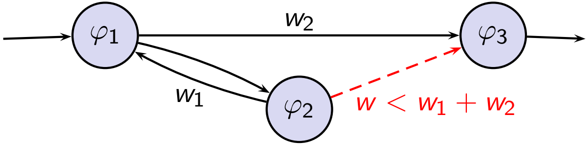
Optimality

The test case chain is *minimal* if

(1) the program and the properties admit a test chain,

(2) all test goals are singleton sets, and

(3) the test chain visits each property once in the K-reachability graph.

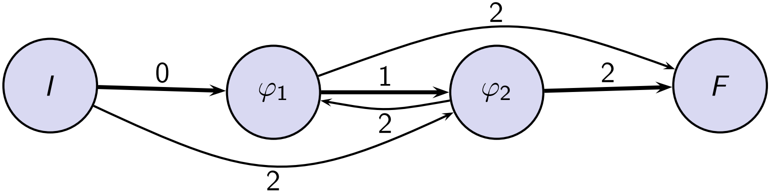


Reachability diameter d = length of maximum, shortest path between any two states

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

In practice, fix a bound K and obtain minimised chain.

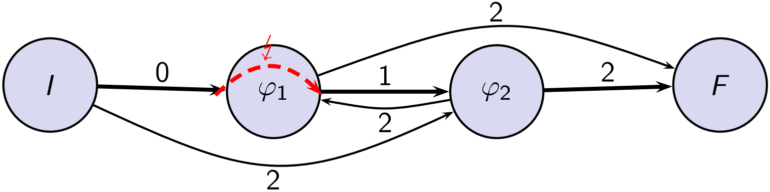
Multi-State Test Goals



p_1 : $\mathbf{G}\big(mode = OFF \wedge \neg enable \wedge button \Rightarrow \mathbf{X}\ enable\big)$

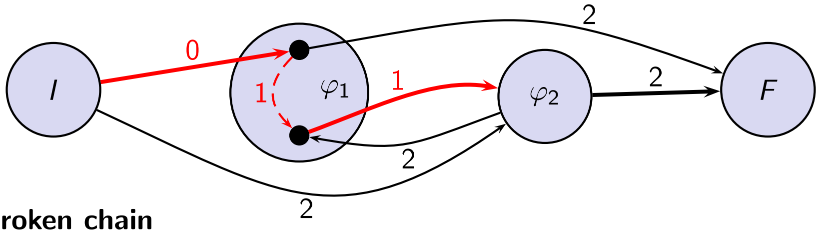
p_2 : $\mathbf{G}\big(mode = ON \wedge brake \Rightarrow \mathbf{X}(mode = DIS)\big)$

Multi-State Test Goals



Broken chain

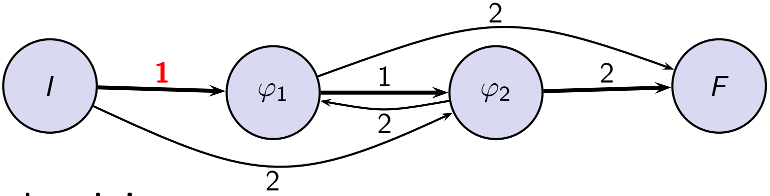
Multi-State Test Goals



Broken chain

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

Multi-State Test Goals



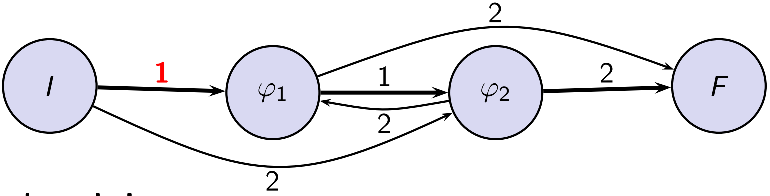
Broken chain

- 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

Multi-State Test Goals



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

- Succeeds if path admits chain in concrete program
- If for each test goal the states are strongly connected

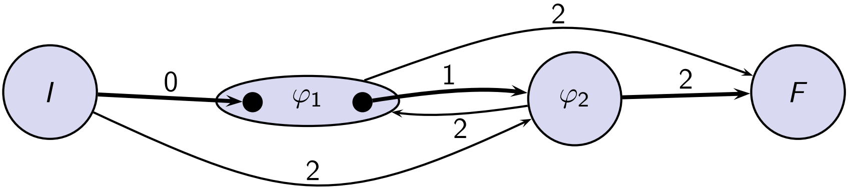
In practice: many systems are (almost) strongly connected.

General Case

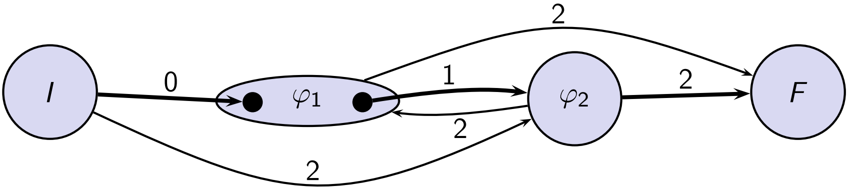
Completeness

- Not strongly connected systems:
  - Abstraction refinement

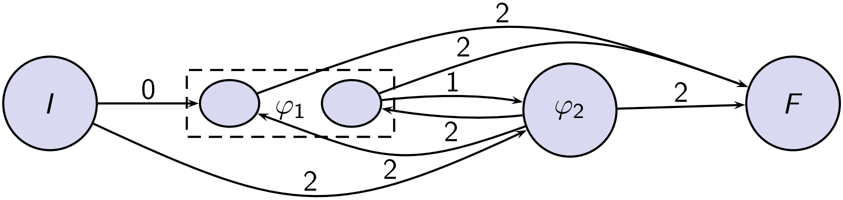
Abstraction Refinement



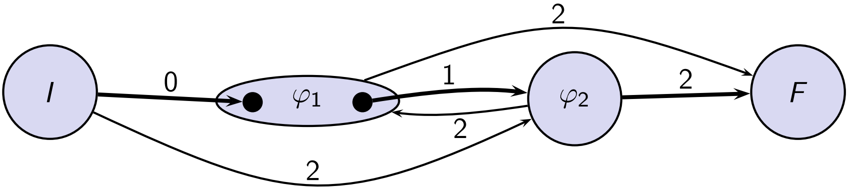
Abstraction Refinement



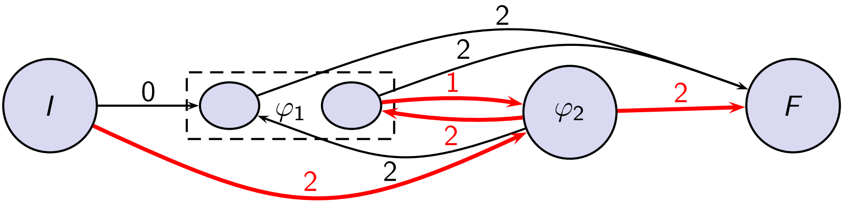
Abstraction refinement:



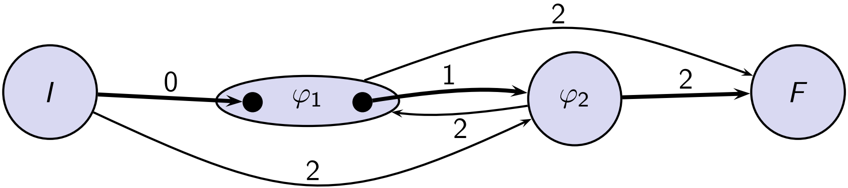
Abstraction Refinement



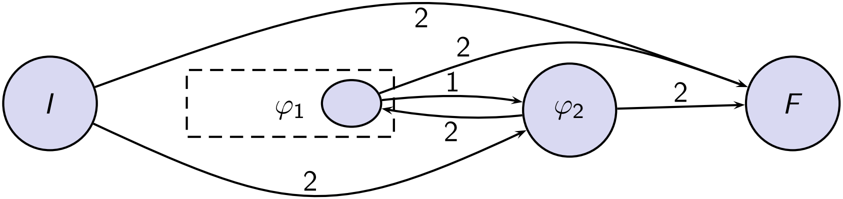
Abstraction refinement: Find any path



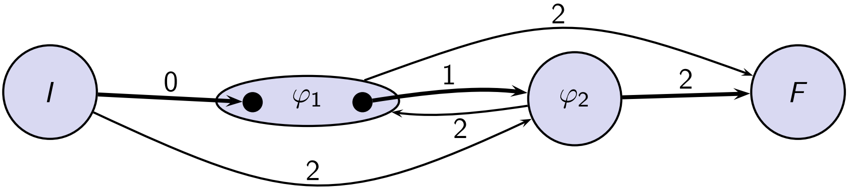
Abstraction Refinement



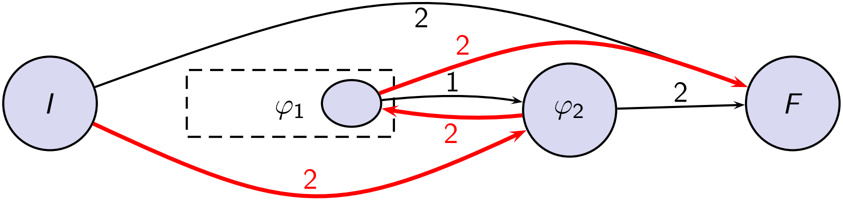
Abstraction refinement: Optimise with TSP solver



Abstraction Refinement



Abstraction refinement: Optimise with TSP solver



General Case

Completeness

- Not strongly connected systems:
  - Abstraction refinement

General Case

Completeness

- Not strongly connected systems:
  - Abstraction refinement
  - More general solver than TSP solver, e.g. ASP solver

General Case

Completeness

- Not strongly connected systems:
  - Abstraction refinement
  - More general solver than TSP solver, e.g. ASP solver
- Multiple chains :
  - Partitioning by graph colouring

General Case

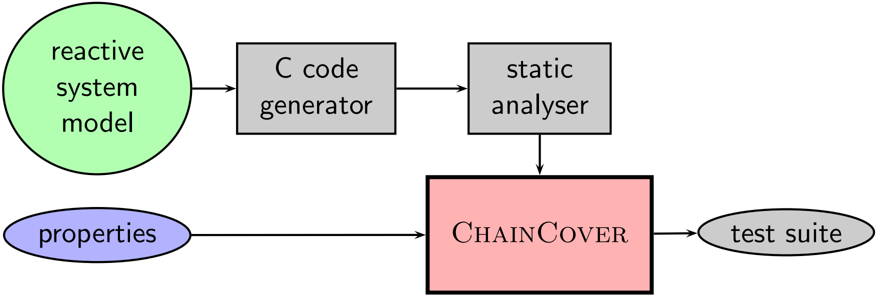
Completeness

- Not strongly connected systems:
  - Abstraction refinement
  - More general solver than TSP solver, e.g. ASP solver
- Multiple chains :
  - Partitioning by graph colouring

Optimality

- Would require to optimise over concrete system
- In practice, minimised rather than minimal solutions relevant

Implementation



Implementation

Properties specified as C functions:

```
void p_1(io_t* i, state_t* s) {
  __CPROVER_assume(s->mode==ON && s->speed==1 && i->dec);
  compute(i,s);
  assert(s->speed==1);
}
```

Woven into program during test case generation.

BMC engine of CBMC

- Property reachability graph construction
- Exploits incremental SAT solving
- Chain repair by concrete chaining

LKH travelling salesman problem solver
CLINGO answer set programming solver

Benchmarks and Comparison

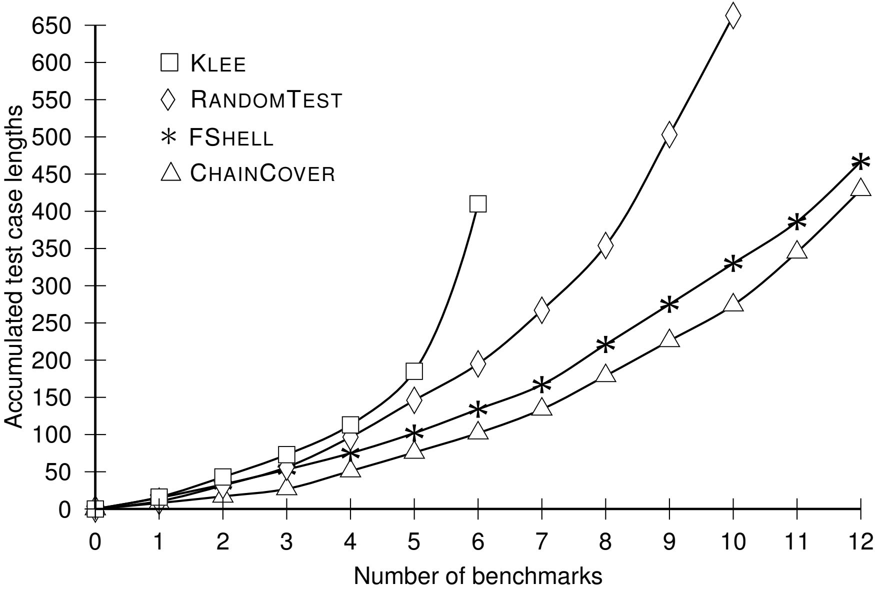
Benchmarks

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

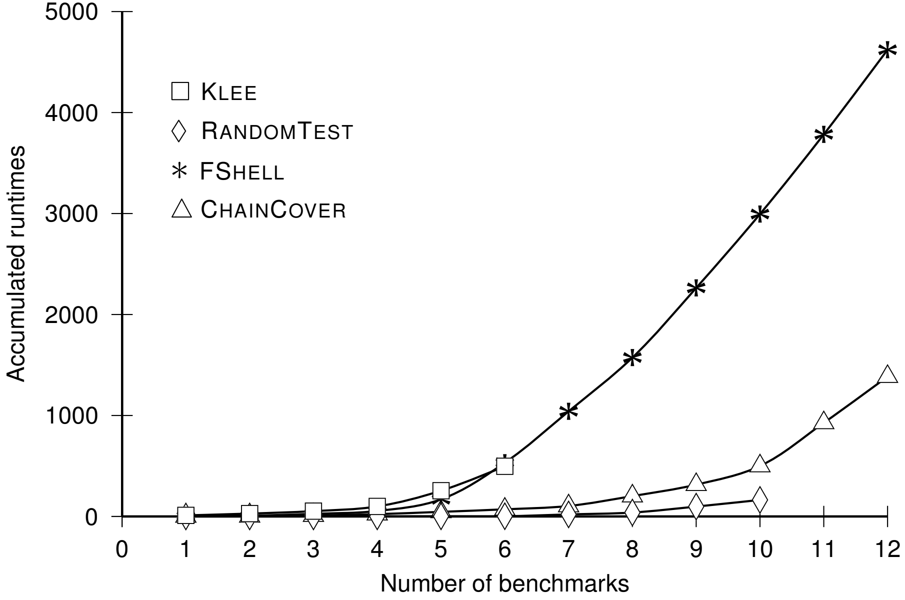
Comparison with

- 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



Results: Test Case Generator Runtime



Summary and Current Work

Summary

- Test chain for reactive systems
  - Test goals from requirements, specification model, code coverage criteria
- Minimal test chain for single-state test goals, otherwise heuristics
- Experimental evaluation
- Application: on-target testing, acceptance testing

Current work

- Integrate acceleration to handle deep loops
- Test chains for code coverage criteria, e.g. MC/DC

Further questions

- Incremental test chain generation
  - In the case of model modifications
  - When test execution gets stuck due to a failed test goal

Download me!

http://www.cprover.org/chaincover