Test Case Generation for Concurrent Systems Using Event Structures

Konstantinos Athanasiou   Hernán Ponce de León   Stefan Schwoon

July 23th, TAP 2015

Motivation

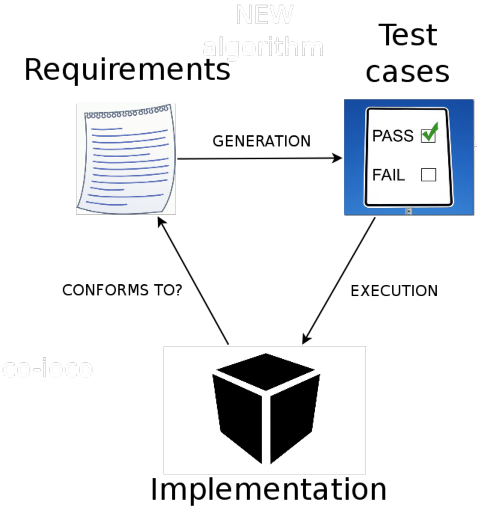
Concurrent and distributed systems are everywhere.

Testing is the most used technique to gain confidence in the correctness of the system.

- ✔ finds actual errors
- ✘ expensive, usually manual

Outline

Formal Black-box Testing



Requirements = *Formal specification*

⇒ *formal notion of conformance, test cases*

⇒ *automation of test case generation*

How to model and test requirement?

Sequential models

- *Formal model: Labeled Transition System*
- *Conformance:* **ioco** *(input-output conformance)*

Problem: State space explosion in concurrent systems

True concurrency models

- *Formal model: Petri nets, network of automata, event structures*
- *Conformance:* **co-ioco** *(concurrent* **ioco***)*

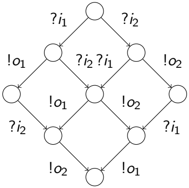
Test Cases

Sequential models

- *Complete test graph $\mathcal{G}$: all the tests for a purpose/criteria*
- *Test cases: partition (subgraphs) of $\mathcal{G}$*
- *Termination: finiteness of purpose/criteria*

$$\mathcal{S} = (i_1; o_1 \parallel i_2; o_2)$$

$$\mathcal{T} = \left\{ \begin{array}{c} ?i_1; !o_1; ?i_2; !o_2 \\ ?i_2; !o_2; ?i_1; !o_1 \\ ?i_1; ?i_2; (!o_1; !o_2 + !o_2; !o_1) \\ ?i_2; ?i_1; (!o_1; !o_2 + !o_2; !o_1) \end{array} \right\}$$

$\mathcal{G} =$ 

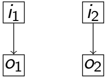
Test Cases

True concurrency models

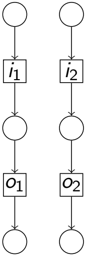
- *Complete test graph $\mathcal{U}$: finite unfolding of the specification*
- *Test cases: partition of $\mathcal{U}$*
- *Termination: cut-off events*

$$\mathcal{S} = (i_1; o_1 \parallel i_2; o_2)$$

$\mathcal{T} =$



$\mathcal{U} =$



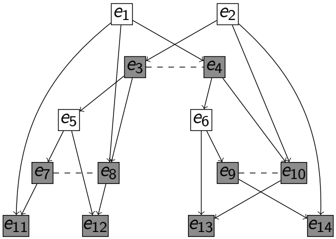
No choices between inputs due to concurrency: fewer test cases
No choice between outputs due to concurrency: smaller test cases

Outline

Definition (Input/Output Event Structure)

An IOES is 4-tuple $\mathcal{E} = (E, \leq, \#, \lambda)$ such that

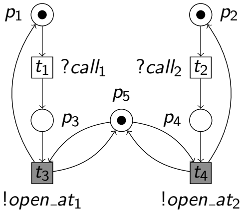
- E is a set of events,
- $\leq \subseteq E \times E$ is a partial order,
- $\# \subseteq E \times E$ is an irreflexive symmetric relation satisfying the property of *conflict heredity*, i.e. $\forall e, e', e'' \in E$: $e \# e' \wedge e' \leq e'' \Rightarrow e \# e''$,
- $\lambda : E \to (\mathcal{I}n \uplus \mathcal{O}ut)$ is a labeling mapping.

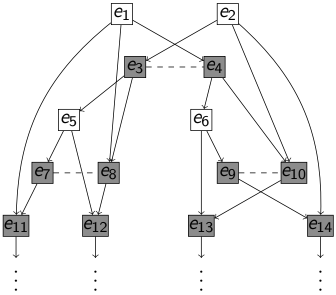


- e_1, e_2 can happen concurrently;
- e_8 needs e_1 and e_3;
- if e_3 happens, e_4 cannot;
- if e_8 happens, e_4 cannot.

Unfoldings

Complete behavior of a system $\approx$ (infinite) unfolding event structure.

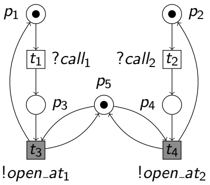




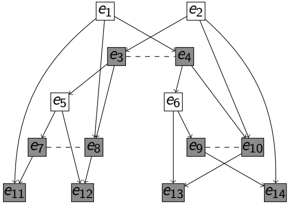
Testing should finish:

- all-states coverage
- all-transitions coverage
- all-loops coverage

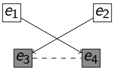
Coverage ≈ finite prefix of the unfolding **[PHL15]**. Detect cut-off events and stop unravelling there.



Specification



All-states

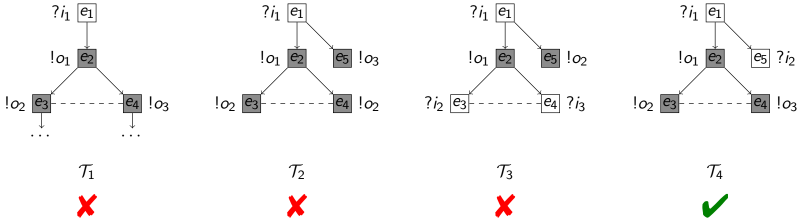


All-transitions

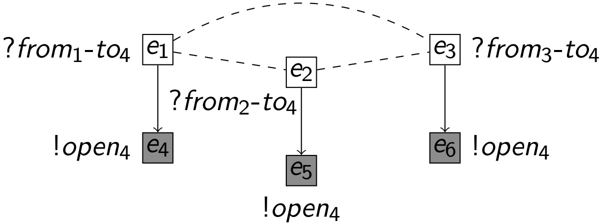
A tester should know the next input to propose and should finish.

Definition

A *global test case* is a finite, deterministic event structure $\mathcal{T} = (E, \leq, \#, \lambda)$ such that $(E^{\mathcal{I}n} \times E) \cap \#^i = \emptyset$.



Add events in order as far as they do not introduce conflicts between inputs:



If e_1 is added then e_2 and e_5 cannot be added (similar to e_3, e_6).

Once e_1 is added, the order of e_2, e_3 is irrelevant!

Preserve causality:

$$\forall e, f \in E : \bigwedge_{f \leq^i e} \varphi_e \Rightarrow \varphi_f$$

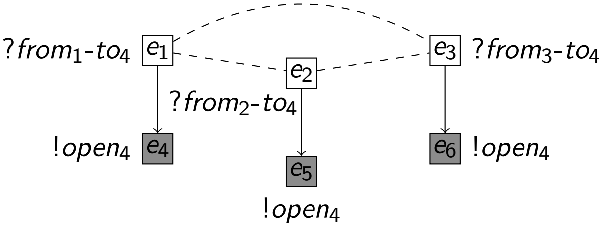
No direct conflicts for inputs:

$$\forall e \in E, f \in E^{\mathcal{I}n} : \bigwedge_{f \#^i e} \neg\varphi_e \vee \neg\varphi_f$$

All events are covered:

$$\forall e, f \in E, g \in E^{\mathcal{I}n} : \neg\varphi_e \Rightarrow \left(\bigvee_{f \leq^i e} \neg\varphi_f \vee \bigvee_{g \#^i e} \varphi_g \right)$$

Example



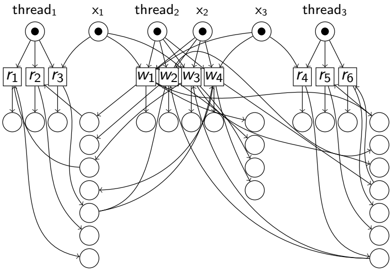
$$\mathrm{AMO}(\varphi_{e_1}, \varphi_{e_2}, \varphi_{e_3}) \wedge (\varphi_{e_1} \vee \varphi_{e_2} \vee \varphi_{e_3}) \wedge (\varphi_{e_1} \Leftrightarrow \varphi_{e_4})$$

$$\wedge(\varphi_{e_2} \Leftrightarrow \varphi_{e_5}) \wedge (\varphi_{e_3} \Leftrightarrow \varphi_{e_6})$$

Solutions: $\{e_1, e_4\}, \{e_2, e_5\}, \{e_3, e_6\}$

From multithreaded programs to unfoldings **[KSH12, KH14]**

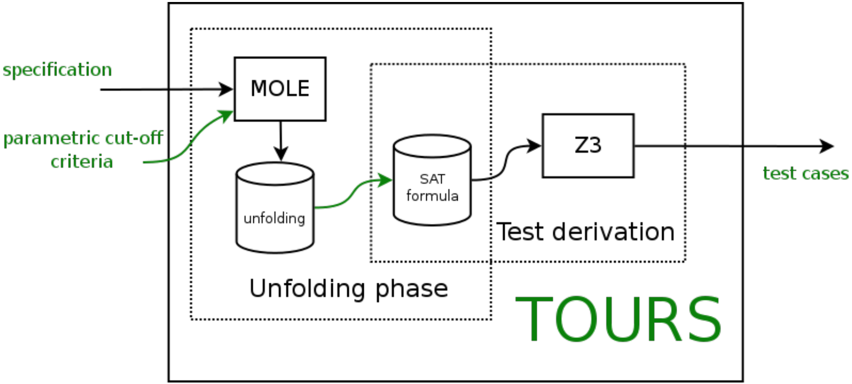
```
Global variable:
int x;

Thread 1:          Thread 2:       Thread 3:
local b = x;       x = 5           local c = x;
```

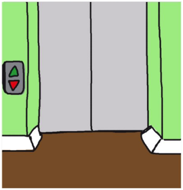


Minimal test suites for multithreaded programs **[PSK+15]** based on an SMT encoding.

Outline

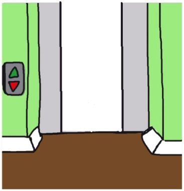
TOURS


specification
parametric cut-off
criteria
MOLE
unfolding
Unfolding phase
SAT
formula
Z3
Test derivation
test cases
TOURS



⇐ concurrent ?calls

concurrent or sequential !opens ⇒



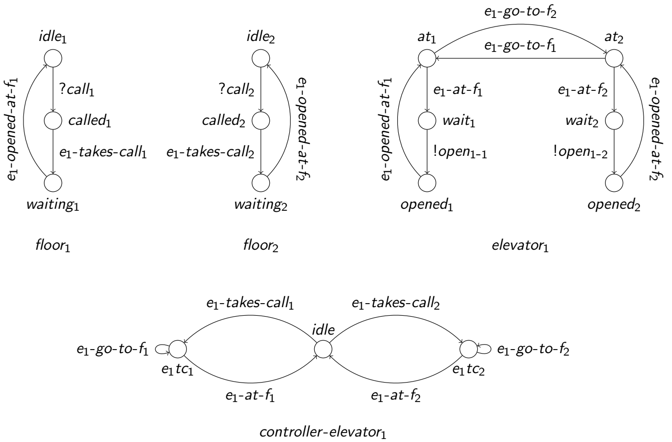
The set up:

- MOLE original cut-off criterion
- interleaving semantics: reachability graph as the input

Compare:

- size of the prefix vs size of the test graph
- number of test cases

Example (2)



Network of automata $\approx$ Petri net

Some Results

Floors	Elevators	Prefix	co-ioco tests	Test graph	ioco tests
2	1	11	1	95	14
2	2	29	1	3929	✗$_{SAT}$
3	1	43	1	2299	✗$_{SAT}$
3	2	220	1	3911179	✗$_{SAT}$
3	3	1231	1	✗$_{unf}$	
4	1	219	1	✗$_{unf}$	
4	2	1853	1	✗$_{unf}$	
4	3	17033	1	✗$_{unf}$	
4	4	140873	✗$_{SAT}$	✗$_{unf}$	

✗$_{unf}$: no solution from the unfolding after 24hs

✗$_{SAT}$: no solution from the SAT solver after 24hs

Smaller and fewer test cases with partial order semantics

Conclusions

- Event structures produce less and smaller test cases than sequential models.
- Automatic test case generation with unfoldings and SAT.
- Minimality can be encode in SMT.

Future work:

- From event structures to executable test cases.
- Validation of the approach on real-life examples.

Thanks!