Scalable Incremental Test-Case Generation from Large Behavior Models

Bernhard K. Aichernig, Dejan Ničković, and Stefan Tiran

TAP 2015, L'Aqulia, 2015-07-22

Motivation

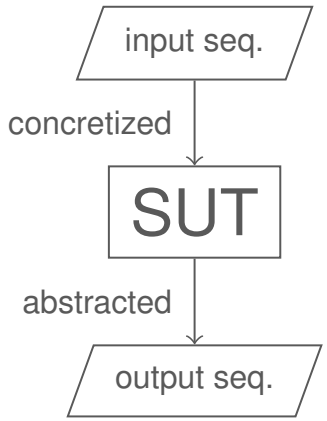
Source: Wikimedia: PARK interlocking, CC-BY-SA

Motivation

Source: Wikimedia: PARK interlocking, CC-BY-SA

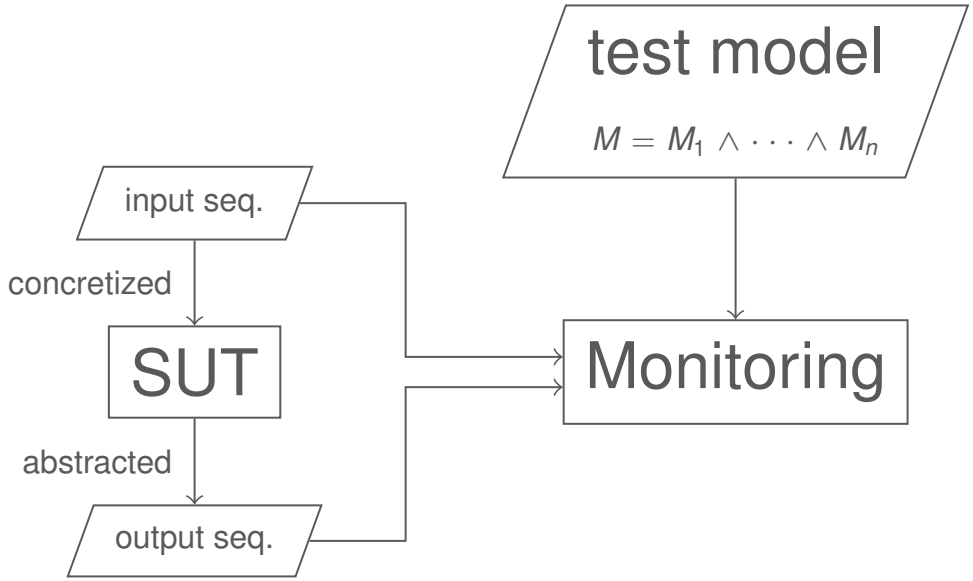
- Currently available technology does not scale
- Bottleneck: exploring the state-space of test models
- Partial models are feasible, but
  - complete test cases are needed
  - partial test might not be valid on complete system
- $\rightarrow$ extend test cases incrementally

Test Framework - Test Case Execution



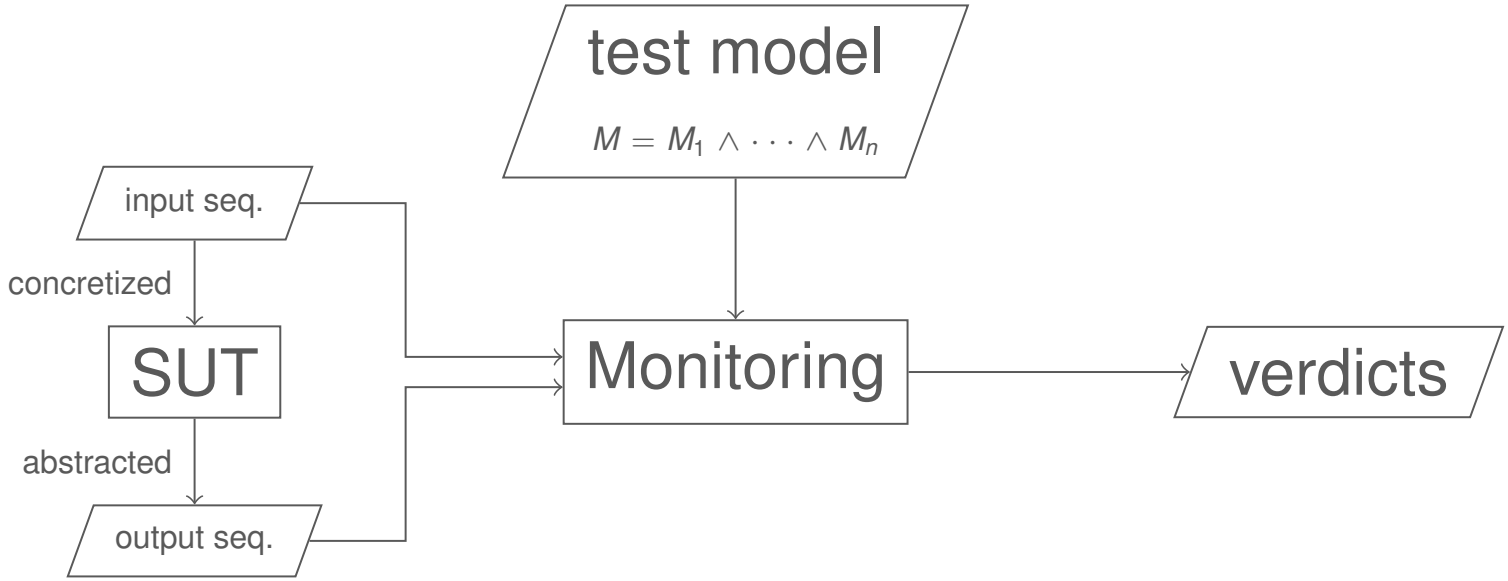
- Synchronous test model

Test Framework - Test Case Execution



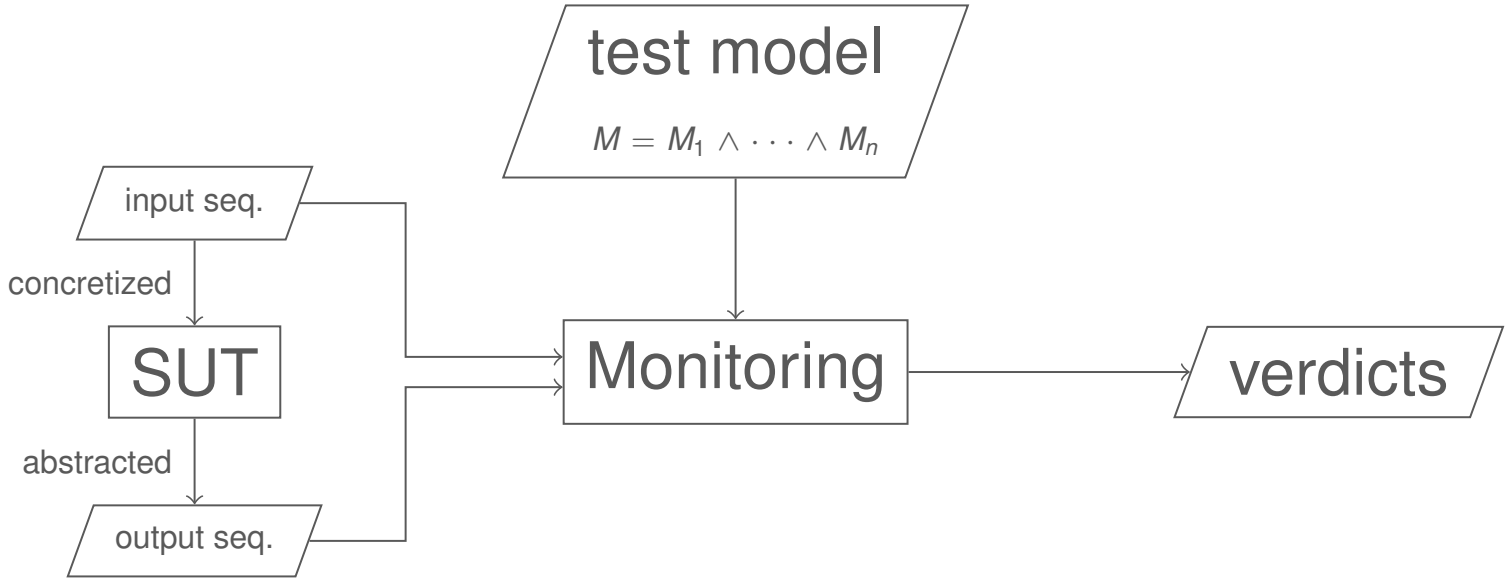
- Synchronous test model
- Test model consists of contracts (grouped by partial models $M_1 \ldots M_n$)
- Output monitor approach used as test oracle

Test Framework - Test Case Execution



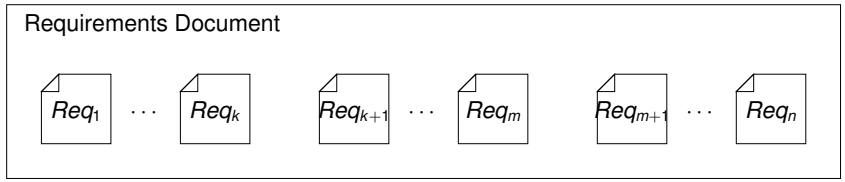
- Synchronous test model
- Test model consists of contracts (grouped by partial models $M_1 \ldots M_n$)
- Output monitor approach used as test oracle

Test Framework - Test Case Execution



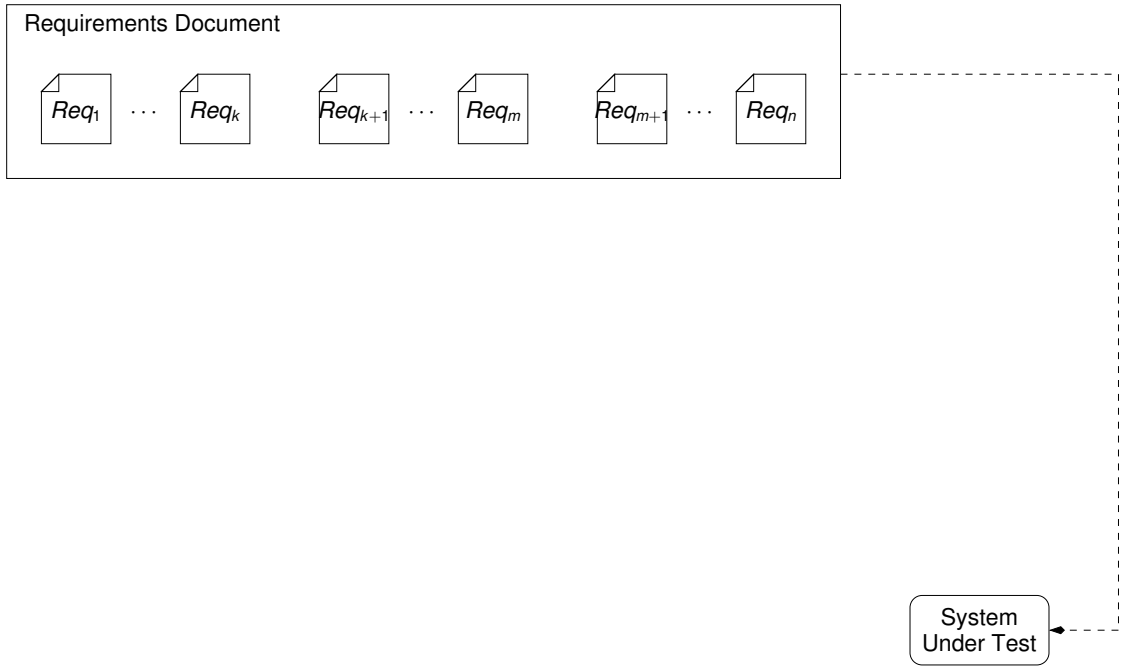
- Synchronous test model
- Test model consists of contracts (grouped by partial models $M_1 \ldots M_n$)
- Output monitor approach used as test oracle
- Test-case generation $\rightarrow$ finding input sequence

Test Framework - Modeling



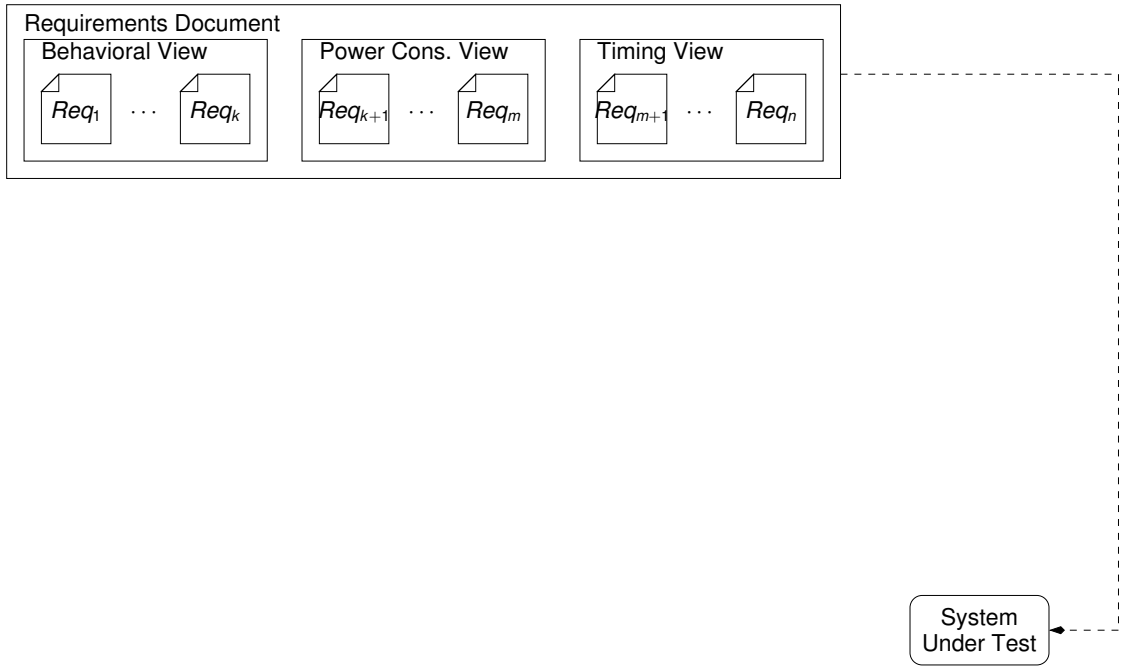
Bernhard K. Aichernig, Florian Lorber, Dejan Ničković, and Stefan Tiran. Require, Test and Trace IT. In FMICS'15, volume 9128 of LNCS, pages 113–127. Springer, 2015.

Test Framework - Modeling



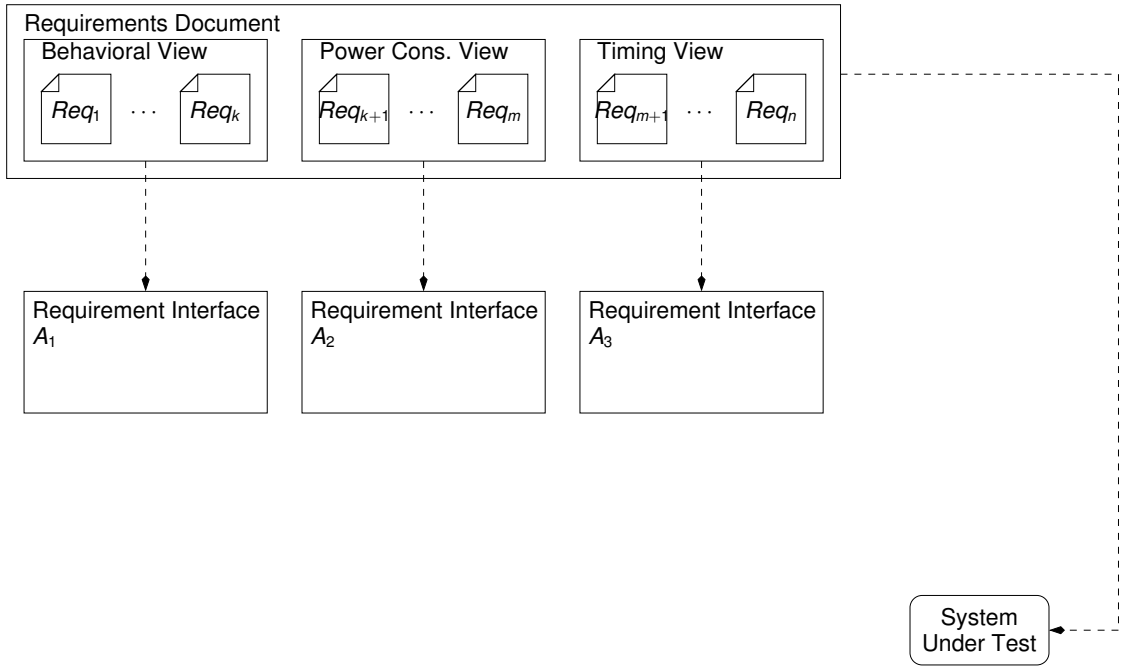
Bernhard K. Aichernig, Florian Lorber, Dejan Ničković, and Stefan Tiran. Require, Test and Trace IT. In FMICS'15, volume 9128 of LNCS, pages 113–127. Springer, 2015.

Test Framework - Modeling



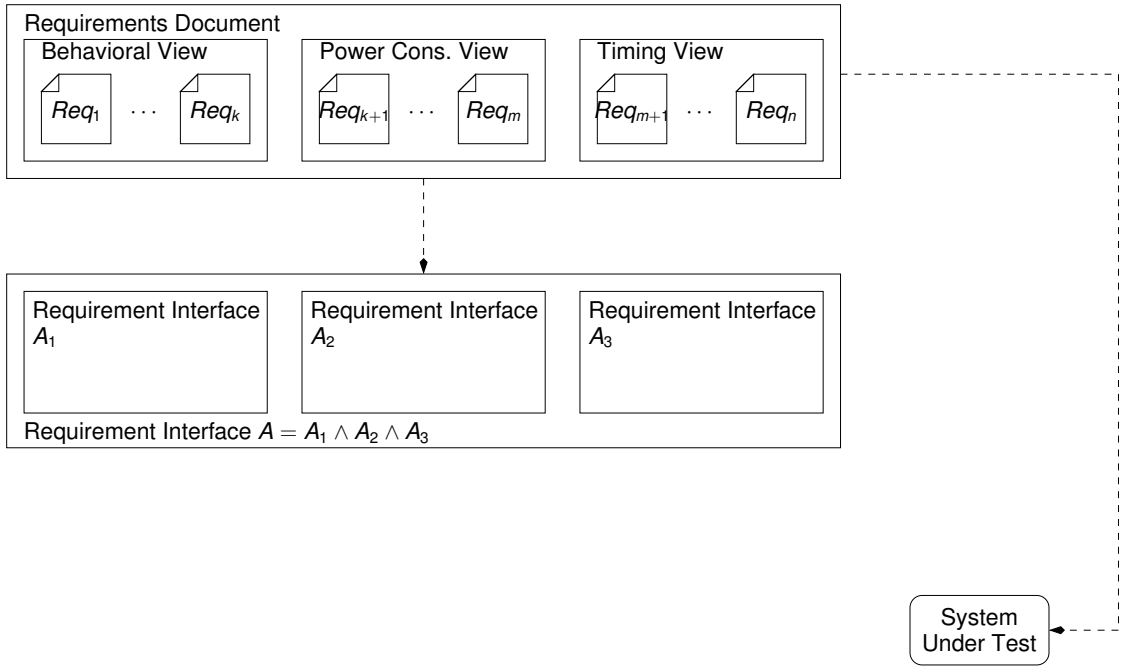
Bernhard K. Aichernig, Florian Lorber, Dejan Ničković, and Stefan Tiran. Require, Test and Trace IT. In FMICS'15, volume 9128 of LNCS, pages 113–127. Springer, 2015.

Test Framework - Modeling



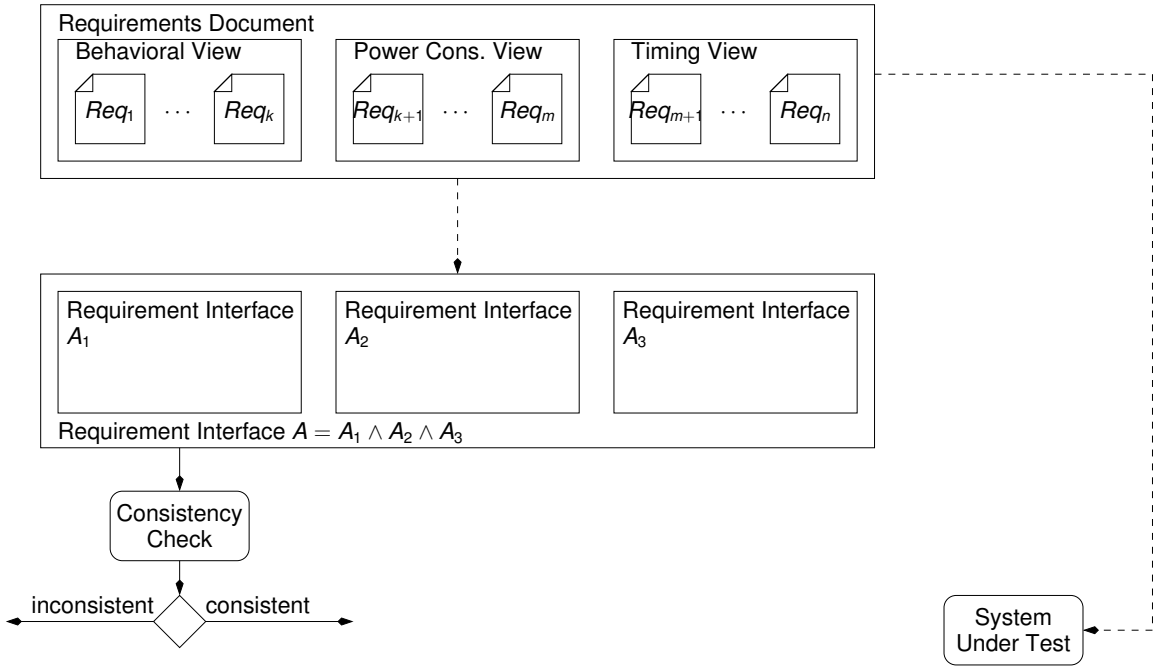
Bernhard K. Aichernig, Florian Lorber, Dejan Ničković, and Stefan Tiran. Require, Test and Trace IT. In FMICS'15, volume 9128 of LNCS, pages 113–127. Springer, 2015.

Test Framework - Modeling



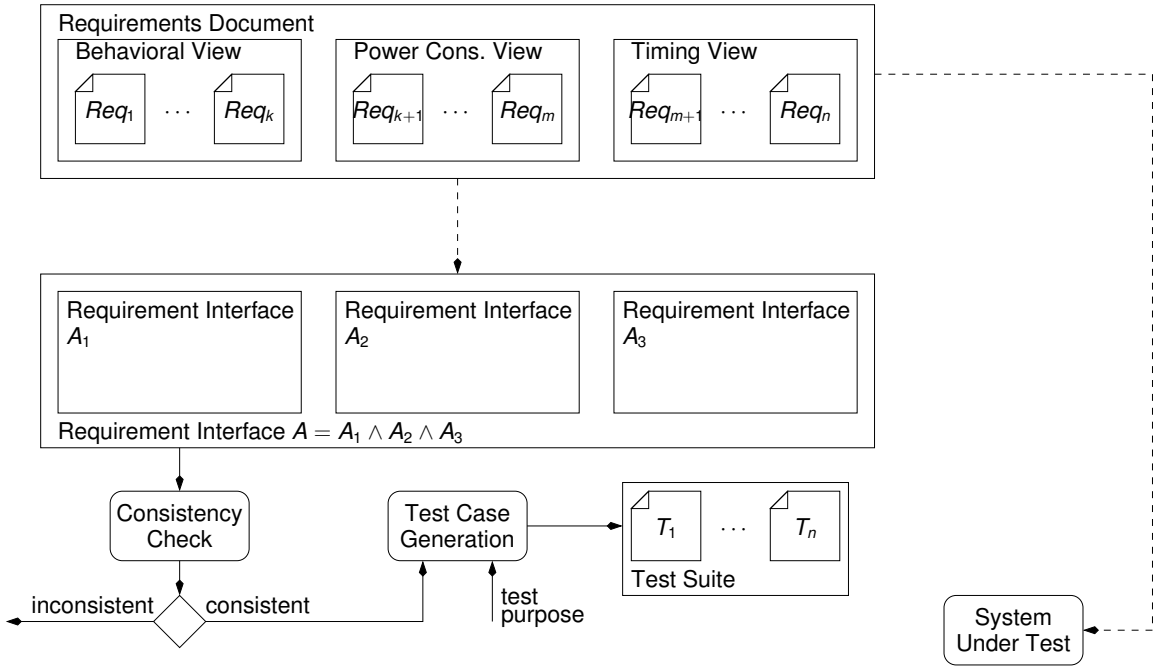
Bernhard K. Aichernig, Florian Lorber, Dejan Ničković, and Stefan Tiran. Require, Test and Trace IT. In FMICS'15, volume 9128 of LNCS, pages 113–127. Springer, 2015.

Test Framework - Modeling



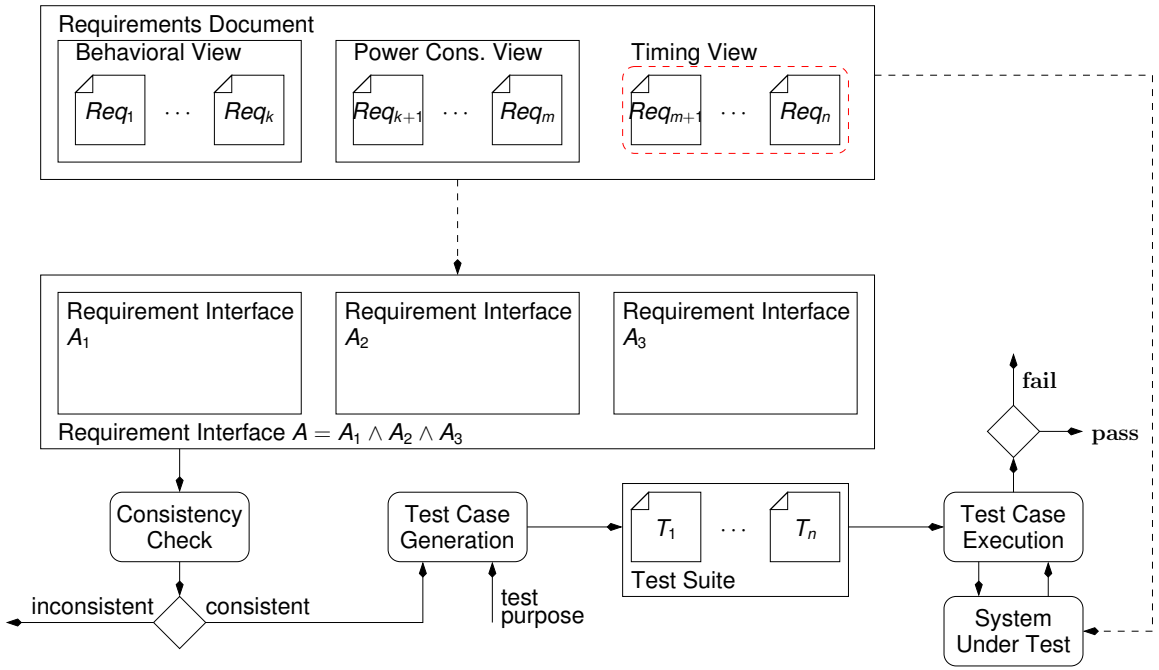
Bernhard K. Aichernig, Florian Lorber, Dejan Ničković, and Stefan Tiran. Require, Test and Trace IT. In FMICS'15, volume 9128 of LNCS, pages 113–127. Springer, 2015.

Test Framework - Modeling



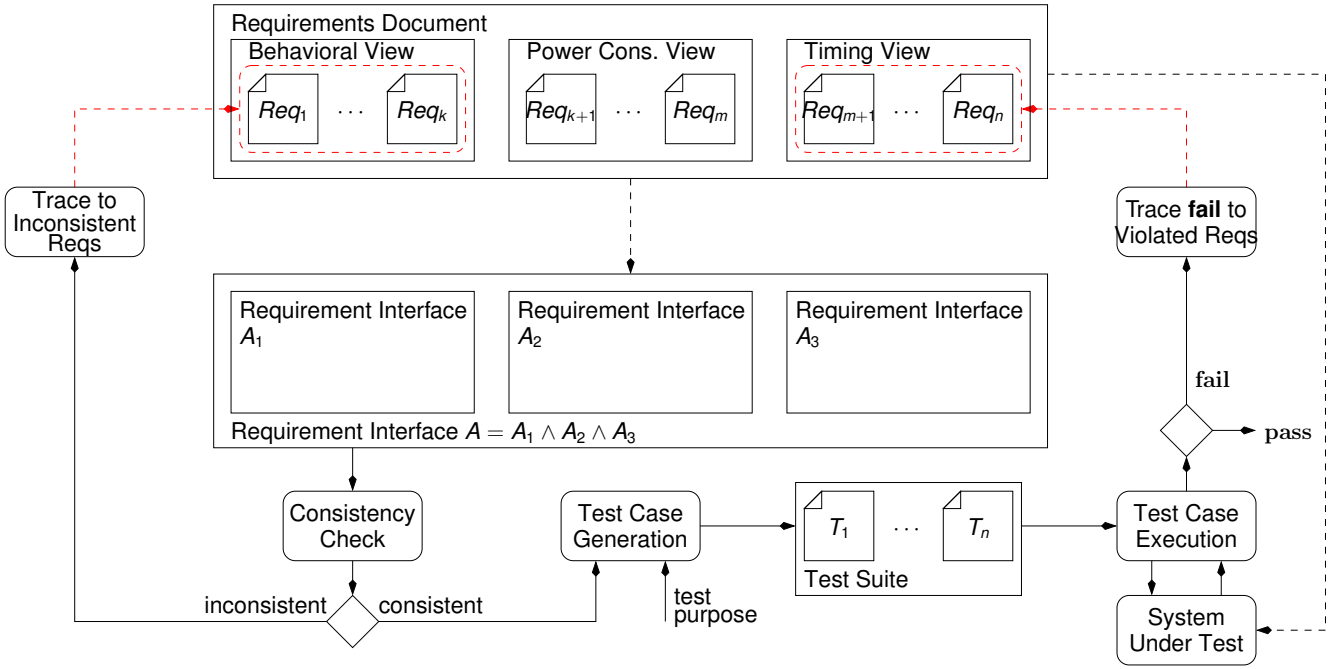
Bernhard K. Aichernig, Florian Lorber, Dejan Ničković, and Stefan Tiran. Require, Test and Trace IT. In FMICS'15, volume 9128 of LNCS, pages 113–127. Springer, 2015.

Test Framework - Modeling



Bernhard K. Aichernig, Florian Lorber, Dejan Ničković, and Stefan Tiran. Require, Test and Trace IT. In FMICS'15, volume 9128 of LNCS, pages 113–127. Springer, 2015.

Test Framework - Modeling



Bernhard K. Aichernig, Florian Lorber, Dejan Ničković, and Stefan Tiran. Require, Test and Trace IT. In FMICS'15, volume 9128 of LNCS, pages 113–127. Springer, 2015.

Example

{R1} enq triggers an enqueue operation when the buffer is not full.

```
{R1} assume enq' and not deq' and k < N  guarantee k' = k + 1
```

{R2} deq triggers a dequeue operation when the buffer is not empty.

```
{R2} assume not enq' and deq' and k > 0  guarantee k' = k - 1
```

{R3} E signals that the buffer is empty.

```
{R3} assume true guarantee k' = 0 <-> E'
```

{R4} F signals that the buffer is full.

```
{R4} assume true guarantee k' = N <-> F'
```

{R5} Simultaneous enq and deq (or their simultaneous absence), an enq on the full buffer or a deq on the empty buffer have no effect.

```
{R5} assume enq' = deq' or enq' and F or deq' and E guarantee k' = k
```

Bounded Reachability

- Transition relation ϕ

$$
\begin{aligned}
\phi^i = \quad & enq_{i+1} \wedge \neg deq_{i+1} \wedge k_i < N \rightarrow k_{i+1} = k_i + 1 \\
& \wedge \neg enq_{i+1} \wedge deq_{i+1} \wedge k_i > 0 \rightarrow k_{i+1} = k_i - 1 \\
& \wedge \top \rightarrow k_{i+1} = 0 \leftrightarrow E_{i+1} \\
& \wedge \top \rightarrow k_{i+1} = N \leftrightarrow F_{i+1} \\
& \wedge enq_{i+1} = deq_{i+1} \vee enq_{i+1} \wedge F_i \vee deq_{i+1} \wedge E_i \rightarrow k_{i+1} = k_i
\end{aligned}
$$

Bounded Reachability

- Transition relation ϕ

$$
\begin{aligned}
\phi^i = \quad & enq_{i+1} \wedge \neg deq_{i+1} \wedge k_i < N \rightarrow k_{i+1} = k_i + 1 \\
& \wedge \neg enq_{i+1} \wedge deq_{i+1} \wedge k_i > 0 \rightarrow k_{i+1} = k_i - 1 \\
& \wedge \top \rightarrow k_{i+1} = 0 \leftrightarrow E_{i+1} \\
& \wedge \top \rightarrow k_{i+1} = N \leftrightarrow F_{i+1} \\
& \wedge enq_{i+1} = deq_{i+1} \vee enq_{i+1} \wedge F_i \vee deq_{i+1} \wedge E_i \rightarrow k_{i+1} = k_i
\end{aligned}
$$

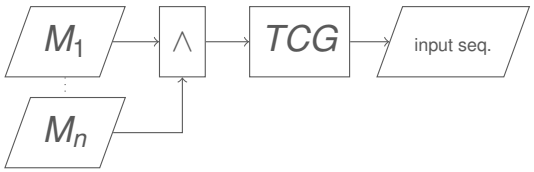
- unfold k times for k-bounded reachability analysis of property Π

$$\textbf{smt}(\ \phi^0 \wedge \ldots \wedge \phi^k \wedge \bigvee_{i \leq k} \Pi[X \backslash X^i])$$

Test-Case Generation

- In general: test suite according to coverage criterion
- Our approach: independent of test selection
- Assumption: test purpose (property) given from outside
- Goal: speed up reachability analysis
- Used technology: SMT solver Z3

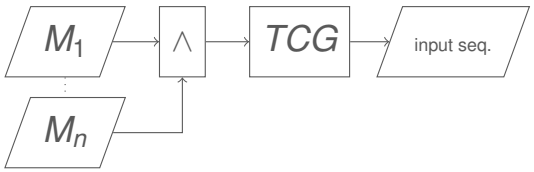
Monolithic vs. Incremental Approach



monolithic

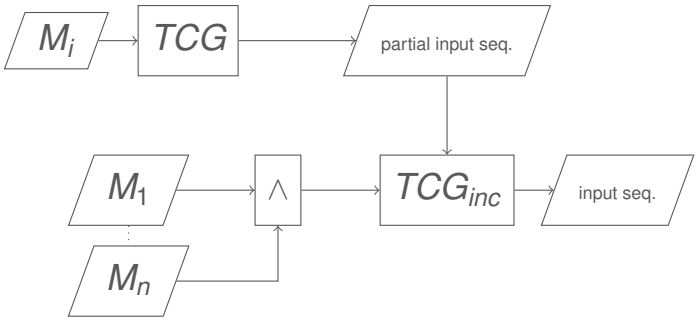
- Check reachability of purpose Π on compound model $M_1 \wedge \cdots \wedge M_n$

Monolithic vs. Incremental Approach



monolithic

- Check reachability of purpose Π on compound model $M_1 \wedge \cdots \wedge M_n$



incremental

- Check reachability of purpose Π on partial model M_i first $\rightarrow$ partial input sequence
- Use partial input sequence as additional constraint

Wheel Loader Case Study - Introduction

- Electronic Control Unit of a Wheel Loader
- Input
  - Deflection from joystick
  - Faults reported by joystick
- Output
  - Electric current for electromagnets controlling valves
  - Status graphics for TFT monitor
- Model split into three view-points
  - Error handling (19 contracts)
  - Configuration of error handling (1 contract)
  - Positive behavior (115 contracts)

Wheel Loader Case Study - Results

Test goal: reach “STOP” state, run-times in seconds

Configuration							
K	M	N	Depth	*vars*	$t_{partial}$	$t_{incremental}$	$t_{monolithic}$
4	3	5	41	861	2.0	3.6	16.8
4	3	6	45	945	2.0	3.4	47.7
4	4	5	46	966	1.6	2.8	31.9
4	3	7	49	1029	2.5	4.6	139.1
4	4	6	50	1050	2.1	3.9	45.3
4	5	5	51	1071	3.4	4.9	47.5
4	3	8	53	1113	3.3	4.8	52.2
4	4	7	54	1134	2.9	4.6	53.7
4	5	6	55	1155	3.0	4.8	252.4
4	4	8	58	1218	4.2	7.3	135.4
4	5	7	59	1239	6.1	7.8	33,810.4

Wheel Loader Case Study - Results

Test goal: reach “STOP” state, run-times in seconds

Configuration							
K	M	N	Depth	*vars*	$t_{partial}$	$t_{incremental}$	$t_{monolithic}$
4	3	5	41	861	2.0	3.6	16.8
4	3	6	45	945	2.0	3.4	47.7
4	4	5	46	966	1.6	2.8	31.9
4	3	7	49	1029	2.5	4.6	139.1
4	4	6	50	1050	2.1	3.9	45.3
4	5	5	51	1071	3.4	4.9	47.5
4	3	8	53	1113	3.3	4.8	52.2
4	4	7	54	1134	2.9	4.6	53.7
4	5	6	55	1155	3.0	4.8	252.4
4	4	8	58	1218	4.2	7.3	135.4
4	5	7	59	1239	6.1	7.8	33,810.4

- 9.4 hours

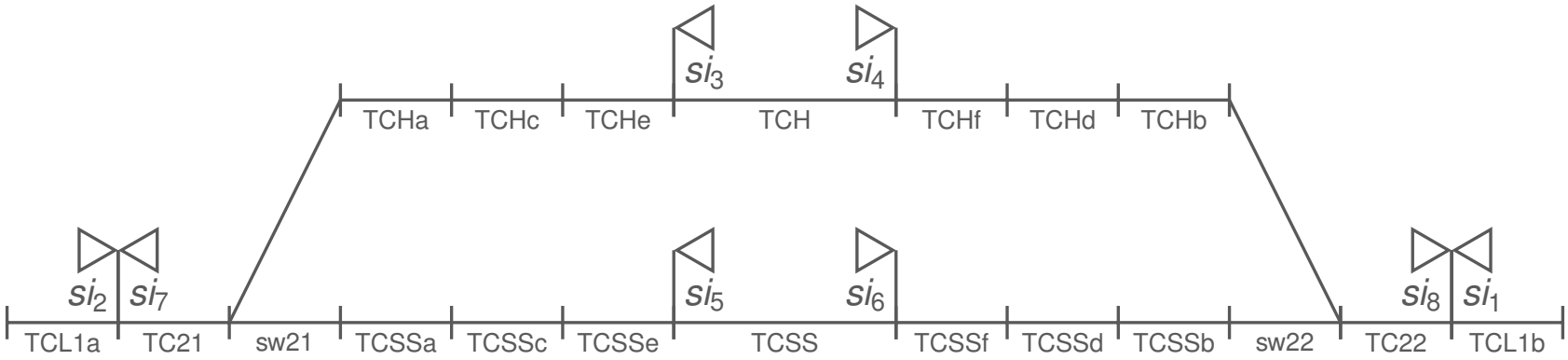
Wheel Loader Case Study - Results

Test goal: reach “STOP” state, run-times in seconds

Configuration							
K	M	N	Depth	*vars*	$t_{partial}$	$t_{incremental}$	$t_{monolithic}$
4	3	5	41	861	2.0	3.6	16.8
4	3	6	45	945	2.0	3.4	47.7
4	4	5	46	966	1.6	2.8	31.9
4	3	7	49	1029	2.5	4.6	139.1
4	4	6	50	1050	2.1	3.9	45.3
4	5	5	51	1071	3.4	4.9	47.5
4	3	8	53	1113	3.3	4.8	52.2
4	4	7	54	1134	2.9	4.6	53.7
4	5	6	55	1155	3.0	4.8	252.4
4	4	8	58	1218	4.2	7.3	135.4
4	5	7	59	1239	6.1	7.8	33,810.4

- 9.4 hours $\rightarrow$ 7.8 seconds

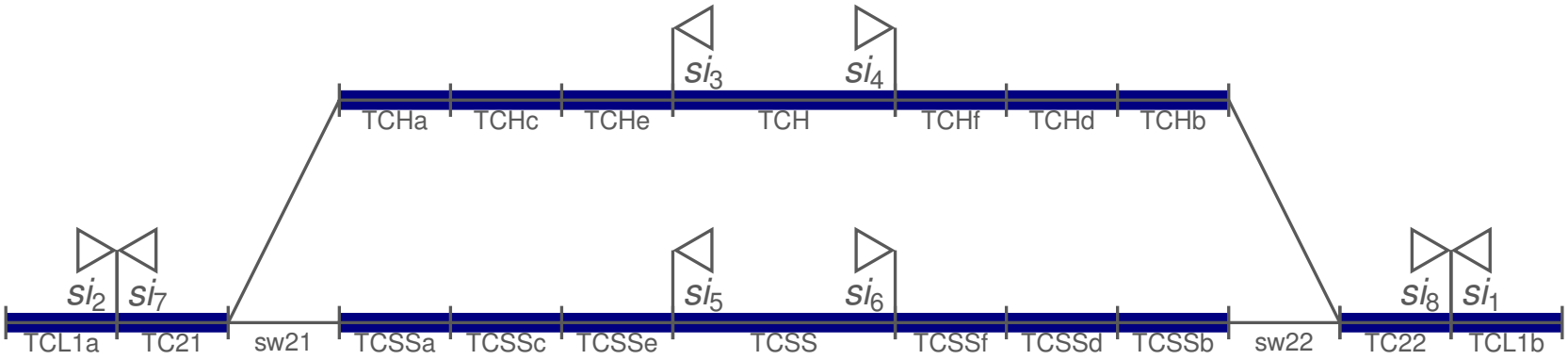
Interlocking System - A Simple Meeting Station



- Model has an object-oriented nature
- Classes with many instances (tracks, switches, signals, train routes)
- Each class has specific rules which describe behavior

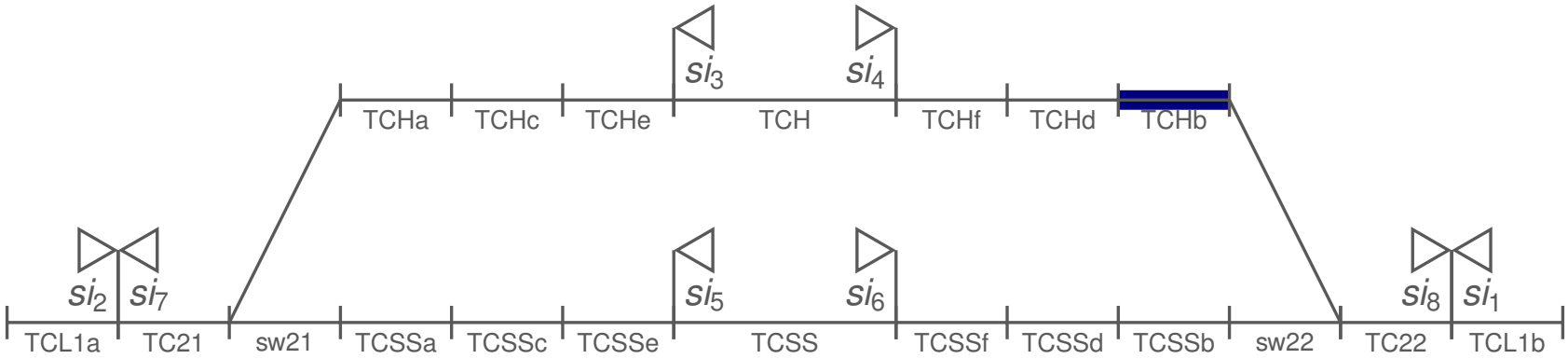
Bernhard K. Aichernig, Dejan Ničković, and Stefan Tiran

Interlocking System - Modeling



- Instances of class “straight track”
- Variables for each instance
  - Input: `occupied` $\in \{$*true*, *false*$\}$
  - Output: `usage` $\in \{$*Unused*, *NotYetFree*, *Free*, *Occupied*, *HadBeenOccupied*, *Dissolved*$\}$

Interlocking System - Modeling



1 {R243, R241} **assume** $\ldots \wedge \neg TCHb\_occupied' \wedge TCHb\_usage = NotYetFree$
2 **guarantee** $TCHb\_usage' = Free$
3 {R243, R241} **assume** $\ldots \wedge \neg TCHb\_occupied' \wedge TCHb\_usage = Free$
4 **guarantee** $TCHb\_usage' = Free$
5 {R243, R241} **assume** $\ldots \wedge TCHb\_occupied' \wedge TCHb\_usage = Free$
6 **guarantee** $TCHb\_usage' = Occupied$
7 ...

Interlocking System - Modeling

Class	Input	Output	Hidden
Track	occupied	usage	-
Switch	occupied	usage, position, locked, interlocked	-
Signal	-	stop, locked	-
Train Route	-	-	state
Global	command	not_permitted, cancel_log	-

- $occupied, locked, interlocked, stop \in \{true, false\}$
- $position \in \{Unknown, Left, Right\}$
- $state \in \{Idle, Admiss.Check, SetUp, SignalClearing, Supervision\}$
- $command \in \{request(tr13), cancel(tr13), takeback(tr13), change(sw22), \ldots\}$
- $not\_permitted, cancel\_log \in \{tr13, tr15, sw22, \ldots\}$

Interlocking System - Modeling (cont)

- Parametrized contracts with abstract variables
- Concrete variables are substituted when actual test model is generated
- This enables
  - arbitrary track layouts
  - partial models that only contain certain objects

Parametrized Contract (example)

IL:RULE:161
A train route shall not be admissible, if any element in the selected route except the start element and the goal element is used in another train route.

Parametrized Contract (example)

IL:RULE:161
A train route shall not be admissible, if any element in the selected route except the start element and the goal element is used in another train route.

1 $\{\text{R161}\}$ **assume** $command' = request(tr) \wedge isInState(tr, Admiss.\ Check)$
2 $\wedge refuseCondition(tr)^1$
3 **guarantee** $setState(tr, Idle) \wedge not\_permitted' = tr$

Parametrized Contract (example)

IL:RULE:161
A train route shall not be admissible, if any element in the selected route except the start element and the goal element is used in another train route.

$$\{\text{R161}\}\ \textbf{assume}\ \ command' = request(tr) \wedge isInState(tr, Admiss.\ Check) \wedge refuseCondition(tr)^1 \ \textbf{guarantee}\ \ setState(tr, Idle) \wedge not\_permitted' = tr$$

$$\{\text{R161}\}\ \textbf{assume}\ \ command' = request(tr\_13) \wedge tr\_13\_state = AdmissibilityCheck \wedge$$

$$\textbf{guarantee}\ \ tr13\_state' = Idle \wedge not\_permitted' = tr\_13$$

Bernhard K. Aichernig, Dejan Ničković, and Stefan Tiran

Parametrized Contract (example)

IL:RULE:161
A train route shall not be admissible, if any element in the selected route except the start element and the goal element is used in another train route.

$\{\text{R161}\}$ **assume** $command' = request(tr) \wedge isInState(tr, Admiss.\ Check)$
$\wedge refuseCondition(tr)^1$
guarantee $setState(tr, Idle) \wedge not\_permitted' = tr$

$refuseCondition(tr)^1 \quad =_{def} \quad exists(((tracks(tr) \cup switches(tr)) \backslash goal(tr)),$
$usageNotEquals(\text{“}elm\text{“}, Unused), \text{“}elm\text{“})$

$\{\text{R161}\}$ **assume** $command' = request(tr\_13)$
$\wedge tr\_13\_state = AdmissibilityCheck \wedge$

$(TC22\_usage \neq Unused \vee sw22\_usage \neq Unused$
$\vee TCHd\_usage \neq Unused \vee TCHb\_usage \neq Unused$

guarantee $tr13\_state' = Idle \wedge not\_permitted' = tr\_13$

Partial Input Vector (example)

Test purpose: set-up and dissolve train route tr_13

step	command	TC22_occ	sw22_occ	TCHb_occ	TCHd_occ	TCHf_occ	TCH_occ
1	request(tr_13)	false	false	false	false	false	false
2	request(tr_13)	false	false	false	false	false	false
3	request(tr_13)	false	false	false	false	false	false
4	None	true	false	false	false	false	false
5	None	false	true	false	false	false	false
6	None	false	false	true	false	false	false
7	None	false	false	false	true	false	false
8	None	false	false	false	false	true	false
9	None	false	false	false	false	false	true

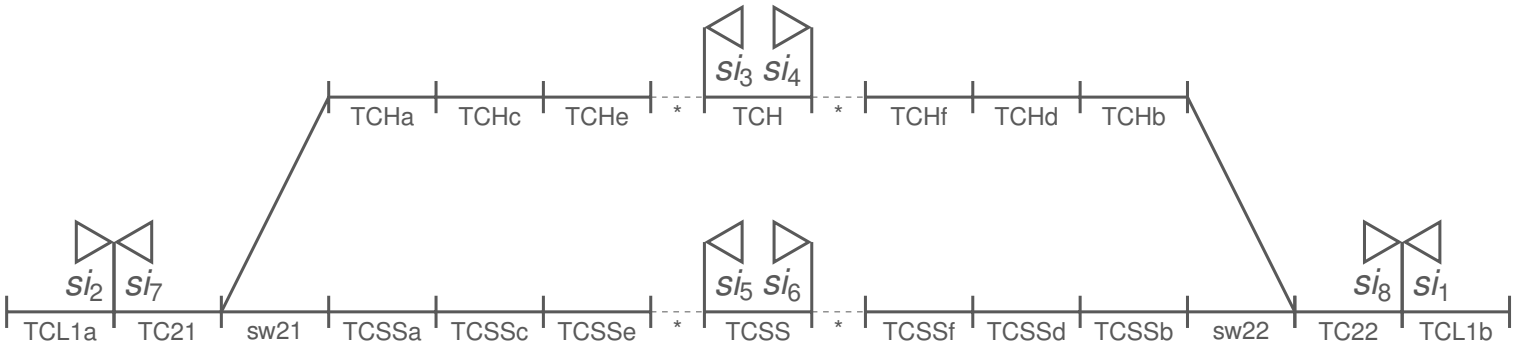
Complete Input Vector (example)

Test purpose: set-up and dissolve train route tr_13

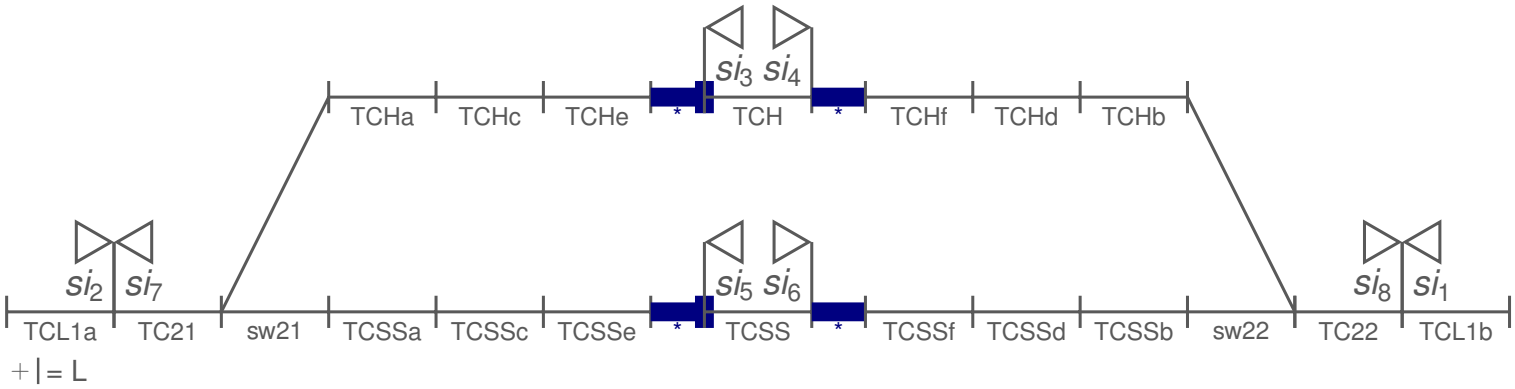
step	command	TC22_occ	sw22_occ	TCHb_occ	TCHd_occ	TCHf_occ	TCH_occ	TCHe_occ	TCHc_occ	TCHa_occ	sw21_occ
1	request(tr_13)	false	false	false	false	false	false	false	false	false	false
2	request(tr_13)	false	false	false	false	false	false	false	false	false	false
3	request(tr_13)	false	false	false	false	false	false	false	false	false	false
4	None	true	false	false	false	false	false	false	false	false	false
5	None	false	true	false	false	false	false	false	false	false	false
6	None	false	false	true	false	false	false	false	false	false	false
7	None	false	false	false	true	false	false	false	false	false	false
8	None	false	false	false	false	true	false	false	false	false	false
9	None	false	false	false	false	false	true	false	false	false	false

TC21_occ	TCL1a_occ	TCSSa_occ	TCSSc_occ	TCSSe_occ	TCSS_occ	TCSSf_occ	TCSSd_occ	TCSSb_occ	TCL1b_occ
false	false	false	false	false	false	false	false	false	false
false	false	false	false	false	false	false	false	false	false
false	false	false	false	false	false	false	false	false	false
false	false	false	false	false	false	false	false	false	false
false	false	false	false	false	false	false	false	false	false
false	false	false	false	false	false	false	false	false	false
false	false	false	false	false	false	false	false	false	false
false	false	false	false	false	false	false	false	false	false
false	false	false	false	false	false	false	false	false	false

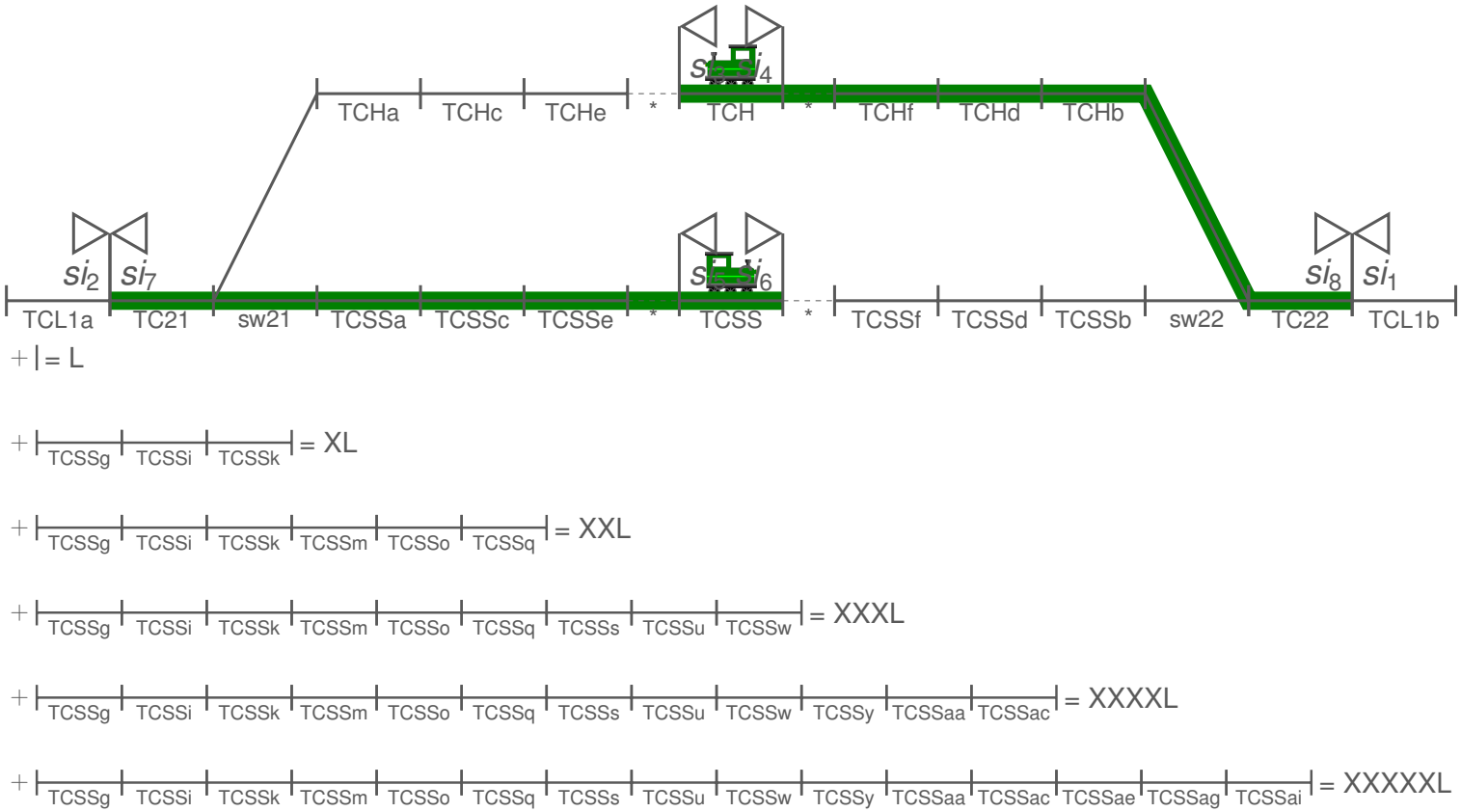
Interlocking System - Experiment Design



Interlocking System - Experiment Design



Interlocking System - Experiment Design



- Common test goal: set-up and dissolve two train routes (trains stand on TCH and TCSS resp)

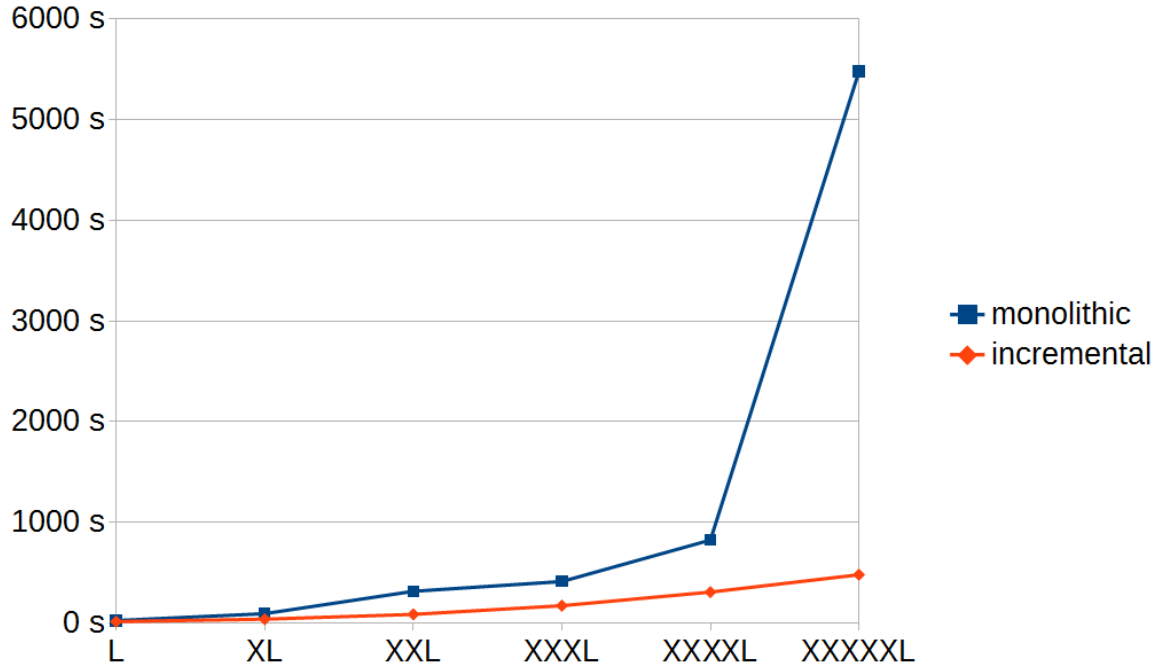
Interlocking System - Results

Model size	# $vars_{part}$	# $vars_{full}$	t_{part}	t_{inc}	t_{mono}
L	720	1360	1.6	15.6	25.7
XL	1083	2071	4.3	39.3	94.2
XXL	1518	2926	10.4	87.3	316.2
XXXL	2025	3925	24.0	173.3	413.2
XXXXL	2604	5068	36.1	308.3	825.0
XXXXXL	3255	6355	63.1	480.8	5476.1

- # vars determines size of SMT problem
- Run-times in seconds
- Model size XXXXXL: monolithic: $> 1h$, incremental: $\sim 8min$

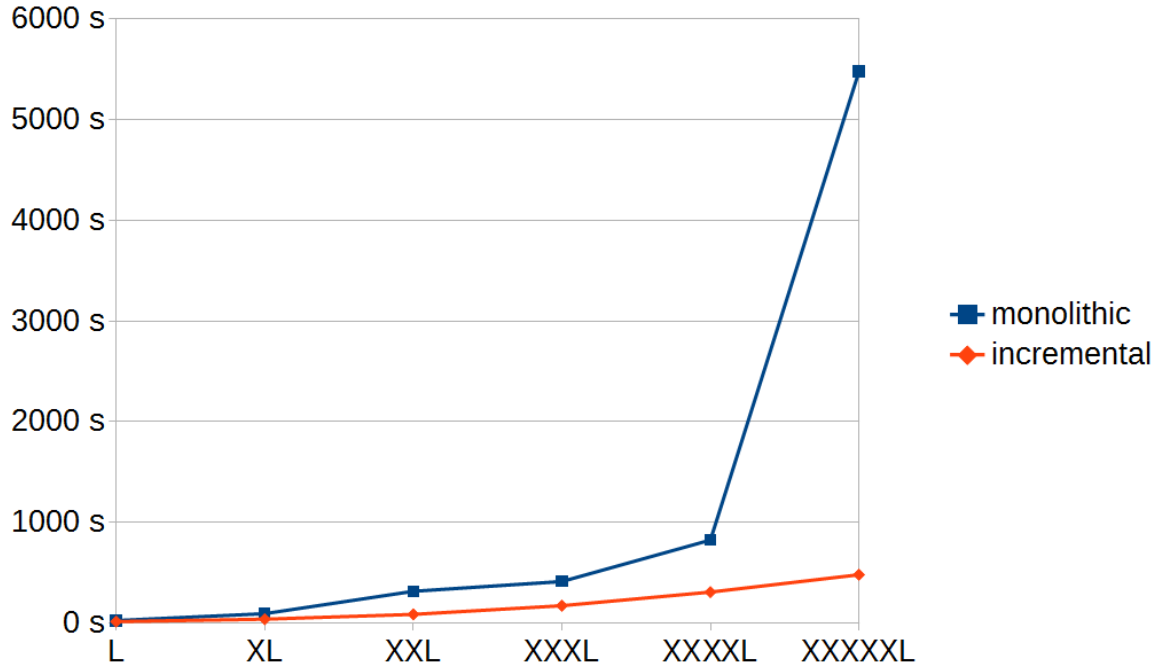
Interlocking System - Runtimes (cont)

We believe this approach scales up



Interlocking System - Runtimes (cont)

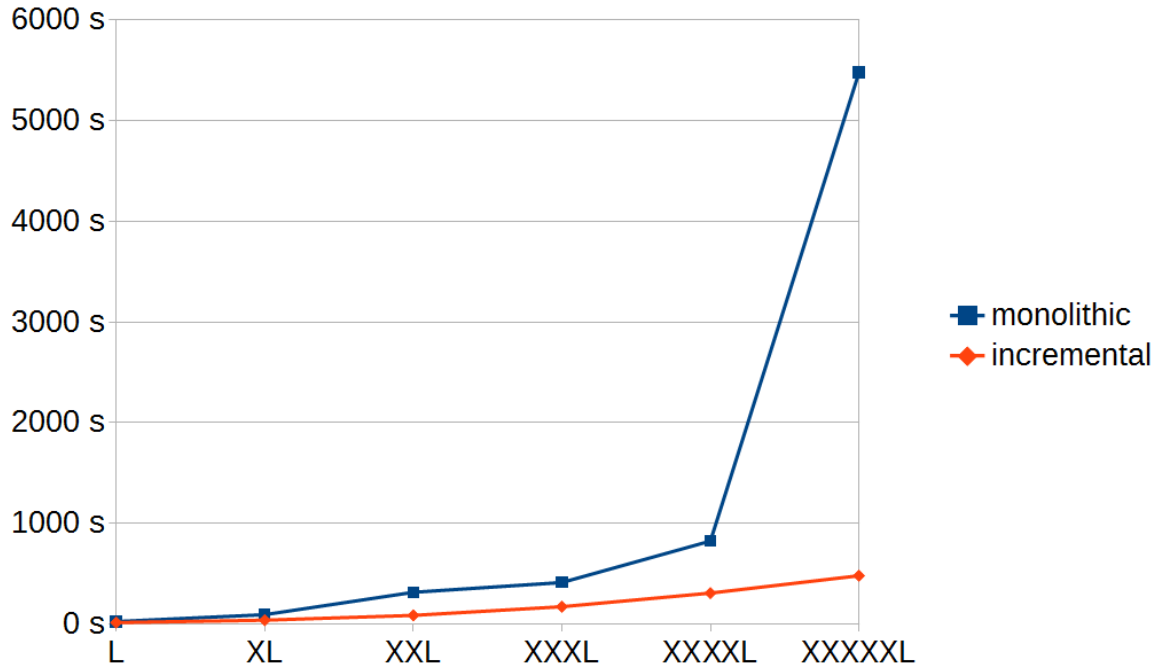
We believe this approach scales up



✓ synchronous

Interlocking System - Runtimes (cont)

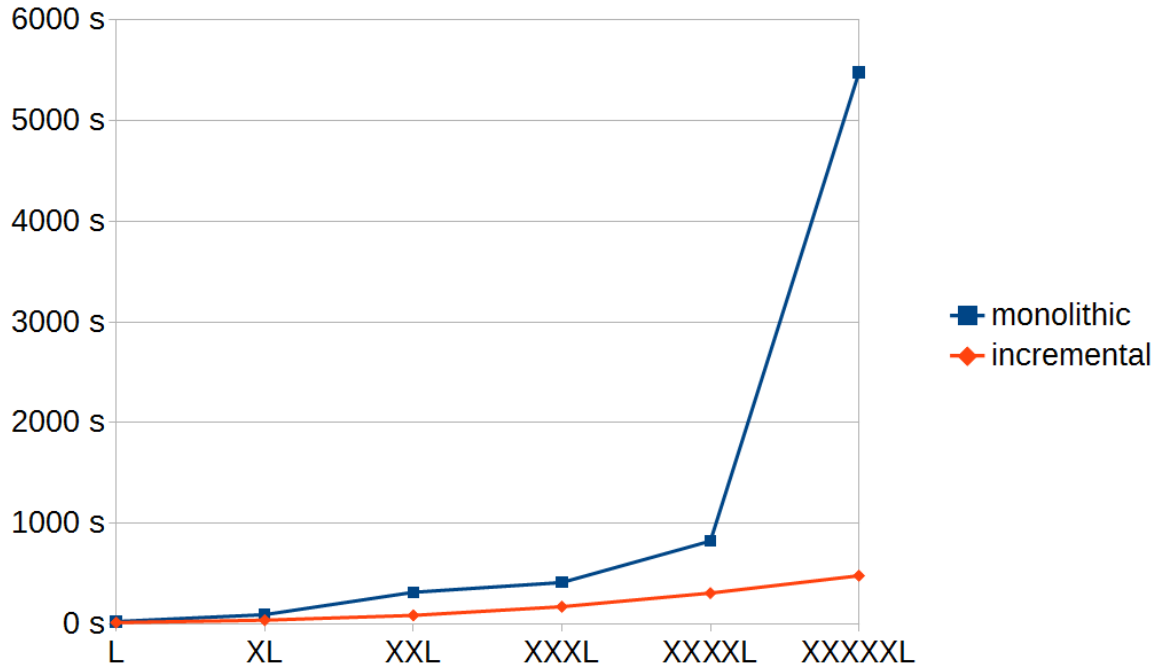
We believe this approach scales up



✓ synchronous ✓ incremental

Interlocking System - Runtimes (cont)

We believe this approach scales up



✓ synchronous ✓ incremental → auto-partitioning