By Irman FAQRIZAL, **Quentin NIVON** & Gwen SALAUN

Patching Liveness Bugs in Concurrent Programs

Introduction

Context

- Designing and developing distributed softwares nowadays is a **tedious and error-prone task** due to the **increasing complexity of the softwares.**
- We need ways of **detecting automatically** such bugs in concurrent programs.

Existing solutions

- **Model checking** is a well-known technique that can be used to **chase violations of temporal logic formulas** in such specifications.
- Model checkers return a **trace** that is a **counterexample** of the given property.
- However, **understanding and correcting** this bug can be a hard and painful task.

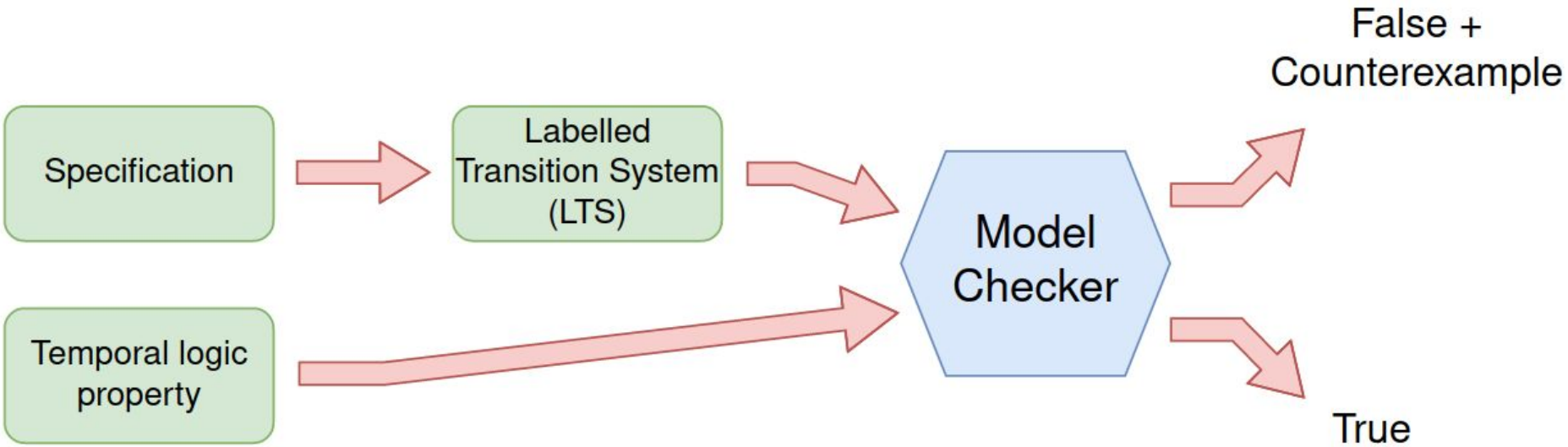
Introduction

Motivations

- **Detect all violations** of a liveness property and **propose a correction** of the specification.
- Propose a correction which is **as simple as possible**.
- Propose a correction that is **as minimal as possible** in terms of **number of changes** in the original specification.

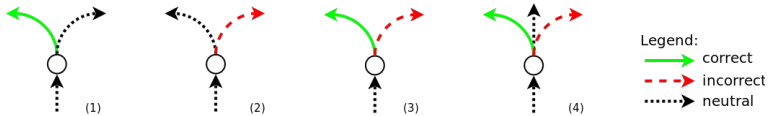
Plan

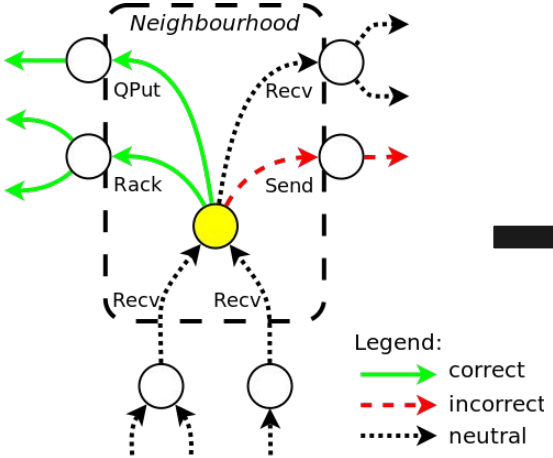
Preliminaries: Model Checking

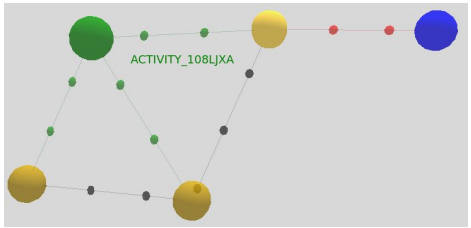


Preliminaries: Counterexample LTS

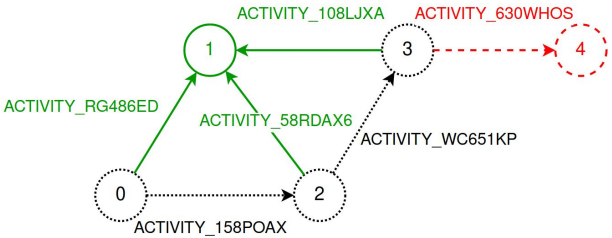
- Key idea 1: transitions categorisation (3 types)
  - correct transitions
  - incorrect transitions
  - neutral transitions
- Key idea 2: identification of faulty states
  - choices between transitions leading to **a correct or an incorrect part** of the model
  - Four kinds of faulty states:







CLTS output from the CLEAR tool



Clearer representation of the CLTS output

Preliminaries: Single Inevitable Execution Property

A **single inevitable execution property** is a temporal logic formula stating that **the action** belonging to the property should appear **at least once in every trace** of the specification.

This kind of property is **one of the most used liveness property** in practice.
[Avrunin, Corbett, Dwyer – ICSE'99]

It can be written as follows:

INEVITABLE("action") ⟷ mu X . (< true > true and [not "action"] X) ⟷ ◇action

Textual representation *MCL representation* *LTL representation*

Preliminaries: Patch

Given a **specification S** and a single inevitable execution **property P = INEVITABLE(“action”)**, **a patch** is defined as **a set of actions of S** before which the action of the inevitable property should be put in order to **correct the specification** (i.e., make it compliant with the property).

```
1  process main [START, RUN,
2  FINISH, LOG : any] is
3    START ;
4    par
5      select
6        LOG [] RUN
7      end select
8    ||
9      select
10       LOG [] FINISH
11     end select
12   end par
13 end process
```

INEVITABLE(“LOG”)

Possible patches:

- START
- RUN + FINISH
- …

Plan

1. Preliminaries
   - 1.1. Model checking
   - 1.2. Counterexample LTS
   - 1.3. Single Inevitable Execution Property
   - 1.4. Patch

2. **Solving approach**
   - **2.1. Global Representation**
   - **2.2. Transition Sets Computation**
   - **2.3. Patches Computation**

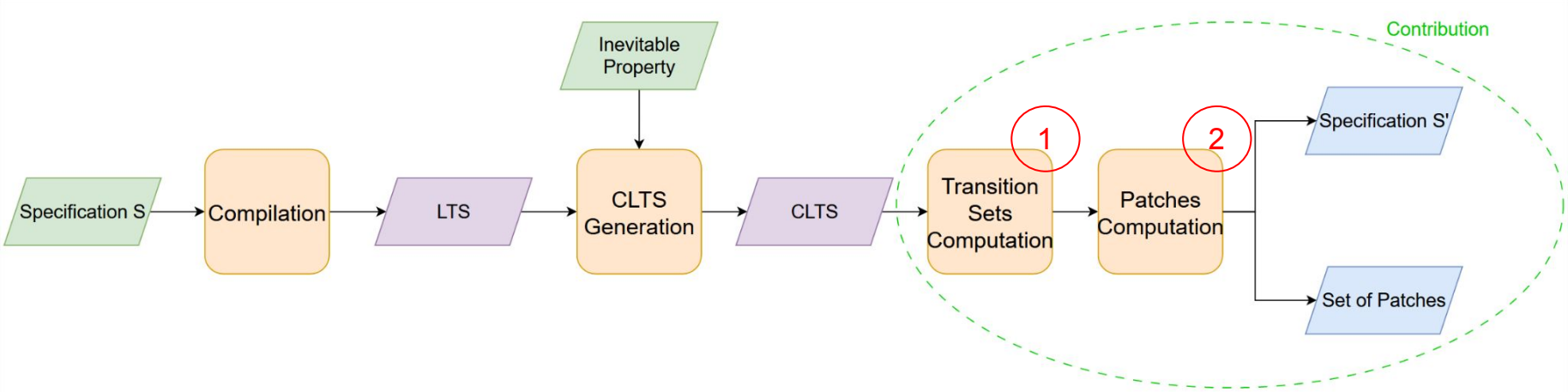
3. Implementation
   - 3.1. Performance study

4. Conclusion

Solving approach: Global representation

The solving approach is a **2-step approach**:

- ➢ First, we **compute several transition sets** needed for the computation of patches. (1)
- ➢ Then, we **compute the set of patches** correcting the specification. (2)



Solving approach: Transition Sets Computation – Definition

The purpose of this step is to **compute three transition sets** that we will use to compute the patches of the specification.
These three sets are the set of:

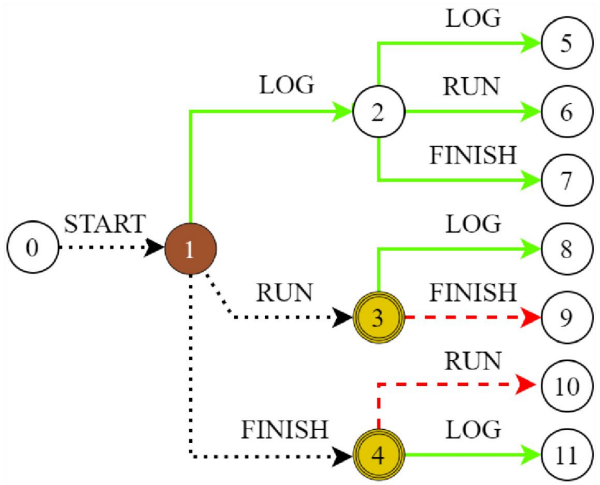
- **All Incorrect Transitions (AIT)**, which contains **all the incorrect transitions** of the CLTS.
- **First Incorrect Transitions (FIT)**, which is a subset of AIT containing **all the incorrect transitions outgoing from faulty states**.
- **Exclusively Incorrect Transitions (EIT)**, which is a subset of FIT containing all the incorrect transitions outgoing from faulty states for which **there exists no other transition with the same label that is either correct or neutral** in the CLTS.

Finally, we recall that **EIT ⊆ FIT ⊆ AIT**.

Solving approach: Transition Sets Computation - Example

```
1  process main [START, RUN,
2  FINISH, LOG : any] is
3    START ;
4    par
5      select
6        LOG [] RUN
7      end select
8    ||
9      select
10       LOG [] FINISH
11     end select
12   end par
13 end process
```

INEVITABLE(“LOG”)



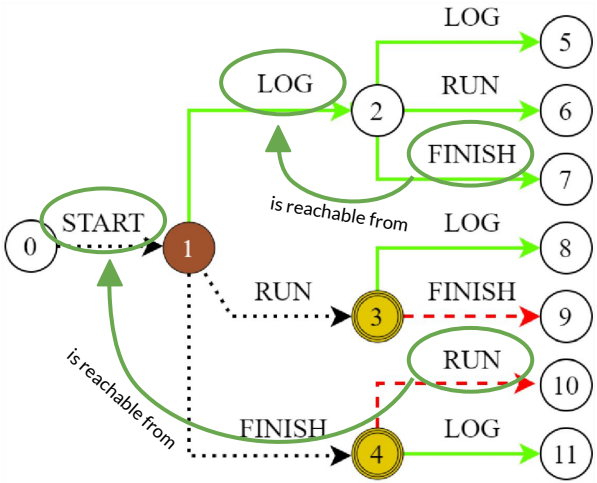
AIT = {FINISH, RUN}

FIT = {FINISH, RUN}

EIT = {}

Solving approach: Computation of Patches – Intuition

The intuition of this step is built on top of the notion of **reachability between transitions**.
A transition is said to be reachable from another transition as long as there exists **several states** (or transitions) **connecting these transitions**.



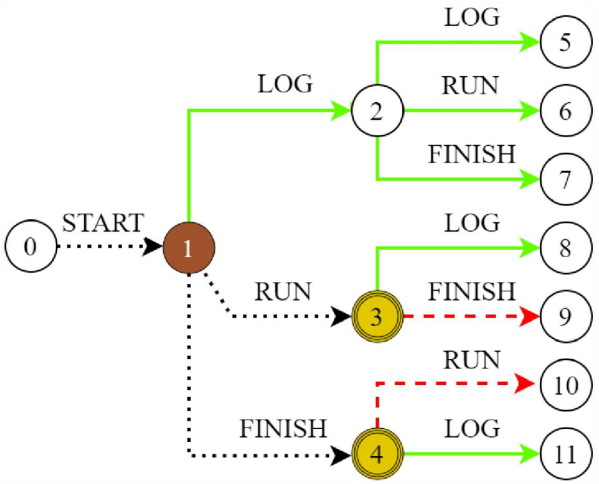
- "FINISH" is reachable from "LOG"
- "RUN" is reachable from "START"
- …

Intuition: **patching only some well-chosen actions** should correct the specification.

Solving approach: Computation of Patches – Impact

In this work, we differentiate two types of patches: patches that **impact correct traces of the specification**, and patches that do not.

A patch is said to impact correct traces of the specification if after **applying this patch** to the specification, **some of its correct traces no longer exist**.



INEVITABLE("LOG") {START}

Trace 0 START 1 LOG 2 LOG 5 no longer exists!

Solving approach: Computation of Patches – Optimality

In both cases, we want to compute the **optimal patches**.

In this work, the notion of **optimality** is defined by the **size of the patches** returned and the **impact** of these patches **on correct traces** of the specification.

The **size** of a patch corresponds to the **number of actions that it contains**, and the **impact** of a patch corresponds to the **number of actions belonging to correct traces impacted** by the patch.

The **smaller** the **patch** the more optimised it is, and the **smaller** its **impact** the more optimised it is.

Solving approach: Computation of Patches – First Step

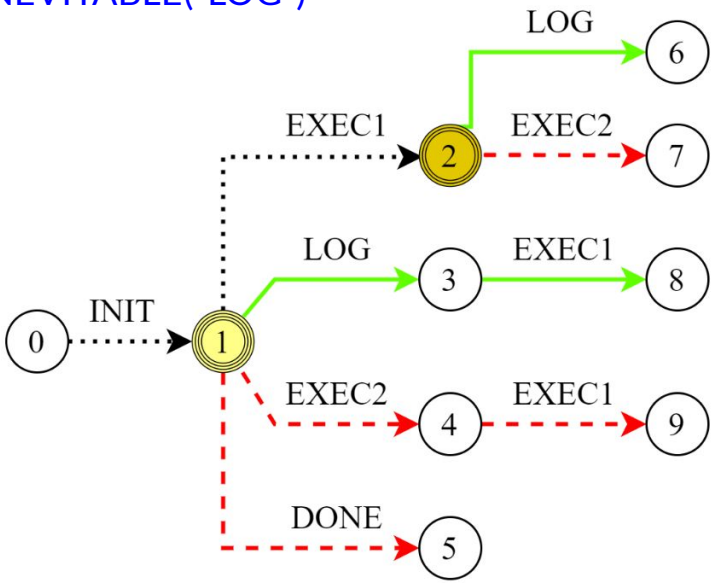
In a first step of the computation, we check whether the **patches** will necessarily **impact** some **correct traces** of the specification **or not**.
In order to **avoid impacting correct traces**, at least **one** of the two following conditions **must hold**:

Condition 1: If **all the first incorrect transitions are exclusively incorrect**, we can patch the specification without impact on correct traces.

Condition 2: If **all the incorrect transitions are reachable from exclusively incorrect transitions**, we can patch the specification without impact on correct traces.

Solving approach: Computation of Patches – First Step (Example)

INEVITABLE("LOG")



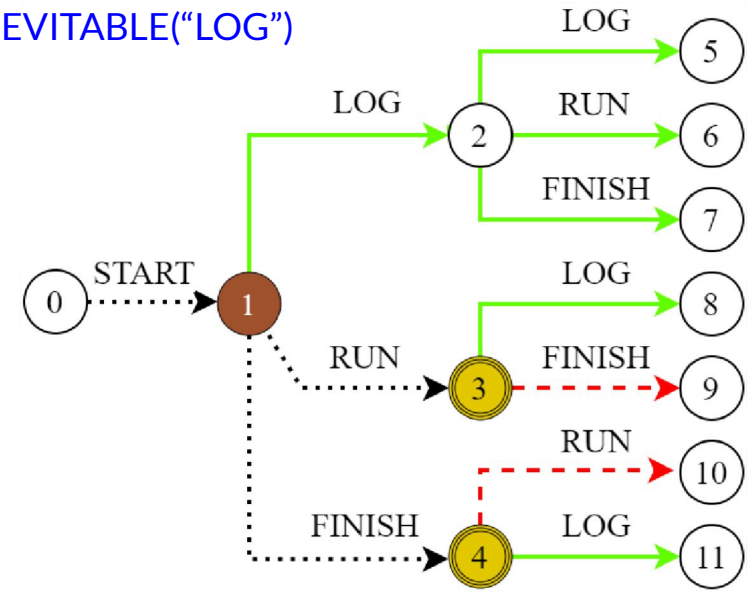
AIT = {EXEC1, EXEC2, RUN}

FIT = {EXEC2, DONE}

EIT = {EXEC2, DONE}

=

INEVITABLE("LOG")



AIT = {FINISH, RUN}

FIT = {FINISH, RUN}

EIT = {}

Solving approach: Computation of Patches – Second Step

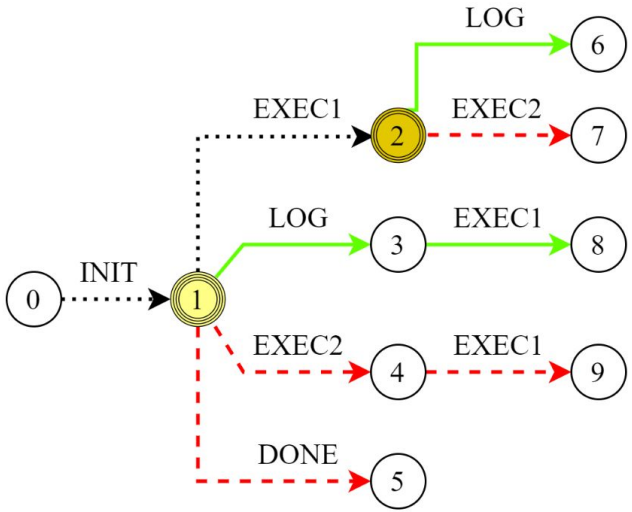
First scenario: Patches **have no impact** on correct traces.

Optimality criterion: **Size** of the patches.

Intuition of the algorithm:

1. Compute **all combinations** of EIT of **size 1.**
2. For each combination, verify if the corresponding transitions can reach all incorrect transitions of the CLTS (i.e., they can correct the specification).
3. If the combination corrects the specification, add it to the list of patches.
4. If a patch was found in step 3, stop iterating and return the patches found. Otherwise, restart an iteration with combinations of size 2 (and so on).

Solving approach: Computation of Patches – Example



Itr.	Current set of actions	Reachable incorrect transitions	Incorrect transitions	Property is satisfied
1	{EXEC2}	{EXEC1, EXEC2}	{EXEC1, EXEC2, DONE}	*false*
	{DONE}	{DONE}	{EXEC1, EXEC2, DONE}	*false*
2	{EXEC2, DONE}	{EXEC1, EXEC2, DONE}	{EXEC1, EXEC2, DONE}	*true*

AIT = {EXEC1, EXEC2, RUN}

FIT = {EXEC2, DONE}

EIT = {EXEC2, DONE}

Patches: {{EXEC2, DONE}}

Solving approach: Computation of Patches – Second Step

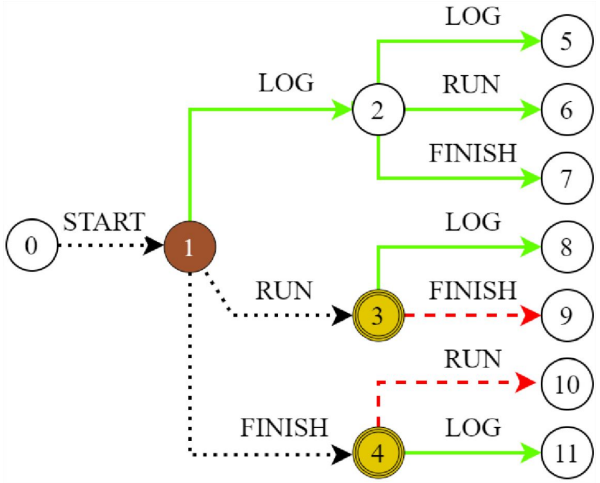
First scenario: Patches **have an impact** on correct traces.

Optimality criterion: **Impact** on correct traces.

Intuition of the algorithm:

1. Compute **all combinations** of FIT of **size 1.**
2. For each combination, verify if the corresponding transitions can reach all incorrect transitions of the CLTS (i.e., they can correct the specification).
3. If the combination corrects the specification, verifies if its impact is smaller than or equal to the previous patches found.
4. If yes, add it to the list of patches.
5. Start a new iteration until having computed all combinations of FIT, regardless of their size.

Solving approach: Computation of Patches – Example



Itr.	Patch	Reachable incorrect transitions from patch	Reachable correct transitions from patch	All Incorrect Transitions	Specification patched	Number of correct actions impacted
1	{RUN}	{RUN, FINISH}	{RUN, LOG}	{RUN, FINISH}	*true*	2
	{FINISH}	{RUN, FINISH}	{FINISH, LOG}	{RUN, FINISH}	*true*	2
2	{RUN, FINISH}	{RUN, FINISH}	{RUN, FINISH, LOG}	{RUN, FINISH}	*true*	3

AIT = {FINISH, RUN}

FIT = {FINISH, RUN}

EIT = {}

Patches: {{RUN}, {FINISH}}

Plan

1. Preliminaries
   - 1.1. Model checking
   - 1.2. Counterexample LTS
   - 1.3. Single Inevitable Execution Property
   - 1.4. Patch
2. Solving approach
   - 2.1. Global Representation
   - 2.2. Transition Sets Computation
   - 2.3. Patches Computation
3. **Implementation**
   - **3.1. Performance study**
4. Conclusion

Implementation

Prototype

The approach detailed in this presentation has been **fully implemented and tested** on dozens of LNT specifications. It consists of approximately 2,000 lines of Python code.

Performance study

Specification	Spec. Size (loc)	CLTS Size (States / Trans.)	Size of T_{fit}	Size of the Patches	Algo.	CLTS Gen. Time (ms)	Patches Comp. Time	Global Exec. Time
1. Backward [9]	50	69/78	6	5	2	0.91	1.52ms	3.62ms
2. Iteration [4]	23	112/159	1	1	3	0.75	0.09ms	2.21ms
3. Muller [16]	43	204/518	3	1	3	3.11	2.18ms	7.47ms
4. Omprace [20]	21	306/808	1	1	2	4.49	0.52ms	8.95ms
5. Interleaving [4]	42	501/1k	1	1	2	10.2	0.67ms	16.2ms
6. IoT [15]	507	1k/4,3k	46	1	2	27.1	24.3ms	93.5ms
7. Ifttt [8]	215	1k/2,9k	1	1	3	26.6	0.90ms	35.3ms
8. Causality [4]	116	1,7k/5,4k	2	1	3	31.0	4.83ms	52.1ms
9. BRP [11]	329	3,4k/5,4k	7	1	3	34.8	26.1ms	90.8ms
10. Station [3]	97	3,8k/14k	1	1	2	216	3.25ms	261ms
11. Fail. Man. [17]	1,9k	11k/21k	20	1	3	192	2,756s	2,756s

Very short!
(300ms at most)

Rather long (~45m)

Plan

1. Preliminaries
   - 1.1. Model checking
   - 1.2. Counterexample LTS
   - 1.3. Single Inevitable Execution Property
   - 1.4. Patch
2. Solving approach
   - 2.1. Global Representation
   - 2.2. Transition Sets Computation
   - 2.3. Patches Computation
3. Implementation
   - 3.1. Performance study
4. **Conclusion**

Conclusion

In this work, we proposed a way for **automatically finding and correcting bugs** related to **single inevitable execution properties** in labelled transitions systems.

We also provided an **automated tool** implementing **the proposed solution**, for which we **assessed the scalability** through a performance study.

Short-term improvements:

- Manage non-fixed action names (such as “!” or “?” in LNT).
- Try to reduce the complexity of the algorithm (avoid computing all combinations)

Mid-term improvements:

- Extend the approach to (at least) nested inevitable properties.

Questions

Solving approach: Computation of Patches – Main Algo

In a first step of the computation, we check whether the patches will necessarily impact some correct traces of the specification or not.

Algorithm 1 Algorithm for Computing Patches

Inputs: $T_{ait}, T_{fit}, T_{eit}$
Output: P (Set of computed patches)

C1 C2

1: **if** $(T_{fit} = T_{eit}) \vee (\bigcup_{\forall t \in T_{eit}} \Delta_I(t.l) = T_{ait})$ **then**
2: P $\leftarrow$ PatchIncorrectOnly(T_{eit}, T_{ait})
3: **else**
4: P $\leftarrow$ PatchIncorrectAndCorrect(T_{fit}, T_{ait})
5: **end if**
6: **return** P

C1: If **all the first incorrect transitions are exclusively incorrect**, we can patch the specification without impact on correct traces

C2: If **all the incorrect transitions are reachable from exclusively incorrect transitions**, we can patch the specification without impact on correct traces

Solving approach: Computation of Patches – Second Step

First scenario: Patches **have no impact** on correct traces
Optimality criterion: Size of the patches

Algorithm 2 PatchIncorrectOnly

Inputs: T_{ait}, T_{eit}
Output: P (Set of computed patches)

1: $PF \leftarrow$ False ; $PS \leftarrow 1$; $P \leftarrow \emptyset$
2: **while** $PF =$ False $\wedge PS \leq |\Phi(T_{eit})|$ **do**
3: $PP = \text{FINDALLCOMBINATIONS}(PS, \Phi(T_{eit}))$
4: **for all** $pp \in PP$ **do**
5: **if** $\bigcup_{\forall l \in pp} \Delta_I(l) = T_{ait}$ **then**
6: $PF \leftarrow$ True
7: $P \leftarrow P \cup \{pp\}$
8: **end if**
9: **end for**
10: $PS \leftarrow PS + 1$
11: **end while**
12: **return** P

iterate until finding a patch

find all combinations of size PS of EIT

for each combination, verifies if it corrects the specification

if yes, add the combination to the list of patches and prevent the next iteration

Solving approach: Computation of Patches – Impact

Second scenario: Patches **have impact** on correct traces
Optimality criterion: Impact on correct traces

Algorithm 3 PatchIncorrectAndCorrect

Inputs: T_{ait}, T_{fit}
Output: P (Set of computed patches)

1: $PF \leftarrow$ False ; $PS \leftarrow 1$; $P \leftarrow \emptyset$; $TI \leftarrow \infty$
2: **while** $PS \leq |\Phi(T_{fit})|$ **do**
3: $PP = \text{FINDALLCOMBINATIONS}(PS, \Phi(T_{fit}))$
4: **for all** $pp \in PP$ **do**
5: **if** $\bigcup_{\forall l \in pp} \Delta_I(l) = T_{ait}$ **then**
6: $CAI \leftarrow |\Phi(\bigcup_{\forall l \in pp} \Delta_C(l))|$
7: **if** $CAI < TI$ **then**
8: $P \leftarrow \{pp\}$
9: $TI \leftarrow CAI$
10: **else if** $CAI = TI$ **then**
11: $P \leftarrow P \cup \{pp\}$
12: **end if**
13: **end if**
14: **end for**
15: $PS \leftarrow PS + 1$
16: **end while**
17: **return** P

iterate over all possible combinations

find all combinations of size PS of FIT

for each combination, verifies if it corrects the specification

if yes, verifies if it has at most the same impact than previous patches found