The Independence Day of Witnessing the Correctness of Systems: From Topological Proofs and Beyond

BCS FACS (Formal Aspects of Computing Science)

Speaker
Claudio MENGHI

Date: 4th July 2023
Agenda

2017
From Model Checking to a Temporal Proof for Partial Models
International Conference on Software Engineering and Formal Methods (SEFM)
Bernasconi, Anna; Menghi, Claudio; Spoletini, Paola; Zuck, Lenore D; Ghezzi, Carlo

2020
Integrating Topological Proofs with Model Checking to Instrument Iterative Design
Fundamental Approaches to Software Engineering (FASE)
Menghi, Claudio; Rizzi, Alessandro Maria; Bernasconi, Anna

2021
TORPEDO: Witnessing Model Correctness with Topological Proofs
Formal Aspects of Computing (FAOC)
Menghi, Claudio; Rizzi, Alessandro Maria; Bernasconi, Anna; Spoletini, Paola

2023
Trace Diagnostics for Signal-based Temporal Properties
IEEE Transactions on Software Engineering (TSE), Boufaied, Chaima; Menghi, Claudio; Bianculli, Domenico; Briand, Lionel C
From Model Checking to a Temporal Proof for Partial Models

International Conference on Software Engineering and Formal Methods (SEFM)

2017

Bernasconi, Anna
Menghi, Claudio
Spoletini, Paola
Zuck, Lenore D
Ghezzi, Carlo
From Model Checking to a Temporal Proof for Partial Models

International Conference on Software Engineering and Formal Methods (SEFM)

Bernasconi, Anna
Menghi, Claudio
Spoletini, Paola
Zuck, Lenore D
Ghezzi, Carlo

Introduction
Genesis

Model Checking and Theorem Proving are two techniques proposed to help designers and developers in producing a software that is correct.

From model checking to a temporal proof.
Peled, Doron, and Lenore Zuck.
Genesis

Model Checking

M: model of the system
ϕ: property of interest

M ⊧ ϕ

yes

no + counterexample

Theorem Proving

M: model of the system
ϕ: property of interest

M ⊭ ϕ

yes + proof

no

From model checking to a temporal proof.
Peled, Doron, and Lenore Zuck.
Preliminaries

**Model Checking + Theorem Proving**

M: model of the system

\( \phi \): property of interest

\( M \models \phi \)

yes + **proof**

no + **counterexample**

*From model checking to a temporal proof.*

Peled, Doron, and Lenore Zuck.

Preliminaries

Model Checking + Theorem Proving

M: model of the system

\( \phi \): property of interest

\[ M \models \phi \]

yes + proof

no + counterexample

Assumption: the model M of the system is completely specified, i.e., it is a definitive model

From model checking to a temporal proof.

Peled, Doron, and Lenore Zuck.

Partial Models

However, in practice, models can be only partially specified or incomplete.
Partial Models (*Formal Methods*)

- A modal process logic
  Larsen, Kim G., and Bent Thomsen.
  Logic in Computer Science, 1988

- Model checking partial state spaces with 3-valued temporal logics
  G Bruns, P Godefroid
  Computer Aided Verification, 1999

- Multi-valued model checking via classical model checking.
  Gurfinkel, Arie, and Marsha Chechk.
  Lecture notes in computer science 2003

- Dealing with Incompleteness in Automata-Based Model Checking
  C Menghi, P Spoletini, C Ghezzi
  Formal Methods, 2016
Partial Models (Software Engineering)

- Managing design-time uncertainty
  Michalis Famelis· Marsha Chechik.

- Partial models: Towards modeling and reasoning with uncertainty
  M Famelis, R Salay, M Chechik
  Software Engineering (ICSE), 2012

- Synthesis of partial behavior models from properties and scenarios
  S Uchitel, G Brunet, M Chechik
  IEEE Transactions on Software Engineering, 2009
Partial Models (*Requirements Engineering*)

Running Example
Running Example
Running Example

- Red lights up infinitely often
  \[ \phi_1 = \Box \Diamond red. \]
- Green lights up infinitely often
  \[ \phi_2 = \Box \Diamond green. \]
- When the light is red, it will always be green
  \[ \phi_3 = \Box (\text{red} \rightarrow \Box \text{green}) \]
Problem Statement

Question
How to help designers in producing correct software with model checking and theorem providing results for partial models?

From model checking to a temporal proof.
Peled, Doron, and Lenore Zuck.
From Model Checking to a Temporal Proof for Partial Models

International Conference on Software Engineering and Formal Methods (SEFM)

Bernasconi, Anna  Menghi, Claudio  Spoletini, Paola  Zuck, Lenore D  Ghezzi, Carlo

Contribution
Contribution (THRIVE)

- THRIVE: THRee valued Integrated Verification framEwork for partial models.
Contribution (THRIVE)

- THRIVE: THRee valued Integrated Verification framEwork for partial models.

- When the light is red, it will always be green

\[ \phi_3 = \Box (\text{red} \rightarrow \Box \text{green}) \]
Contribution (THRIVE)

- THRIVE: THRee valued Integrated Verification framEwork for partial models.
Contribution (THRIVE)

- THRIVE: THRee valued Integrated Verification framEwork for partial models.

- Green lights up infinitely often

\[ \phi_2 = \square \Diamond \text{green}. \]
Contribution (THRIVE)

- THRIVE: THRee valued Integrated Verification framEwork for partial models.
Contribution (THRIVE)

- THRIVE: THRee valued Integrated Verification framEwork for partial models.

- Red lights up infinitely often
  \[ \phi_1 = \square \Diamond red. \]
From Model Checking to a Temporal Proof for Partial Models

International Conference on Software Engineering and Formal Methods (SEFM)

Bernasconi, Anna  Menghi, Claudio  Spoletini, Paola  Zuck, Lenore D  Ghezzi, Carlo

An Instance of THRIVE
An instance of THRIVE

• Model of the system:  
  Partial Kripke Structures (PKS)

• Property of interest:  
  Linear Time Temporal Logic (LTL)
An instance of THRIVE

- Two possible semantics of LTL over PKS can be considered
  - *Three-valued semantics*: it is based on information ordering $T > ? > \perp$
  - *Thorough semantics*: it is based on the notion of refinement

*Model checking partial state spaces with 3-valued temporal logics.*
Bruns, G., Godefroid, P.
CAV 1999

*Generalized model checking: reasoning about partial state spaces*
Bruns, G., Godefroid, P.
CONCUR 2000
An instance of THRIVE: Model checking

Two possible semantics of LTL over PKS can be considered

<table>
<thead>
<tr>
<th></th>
<th>Model checking</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Three-Valued</td>
<td>faster (it exploits two runs of classical model checkers)</td>
<td>Not &quot;correct&quot; when ? is returned</td>
</tr>
<tr>
<td>Thorough</td>
<td>slower (it requires more complex verification procedures)</td>
<td>Correct</td>
</tr>
</tbody>
</table>

*Model checking partial state spaces with 3-valued temporal logics.*
Bruns, G., Godefroid, P.
CAV 1999

*Generalized model checking: reasoning about partial state spaces*
Bruns, G., Godefroid, P.
CONCUR 2000
An instance of THRIVE: Model checking

Two possible semantics of LTL over PKS can be considered

<table>
<thead>
<tr>
<th></th>
<th>Model checking</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Three-Valued</td>
<td>faster (it exploits two runs of classical model checkers)</td>
<td>Not &quot;correct&quot; when ? is returned</td>
</tr>
<tr>
<td>Thorough</td>
<td>slower (it requires more complex verification procedures)</td>
<td>Correct</td>
</tr>
</tbody>
</table>

*Model checking partial state spaces with 3-valued temporal logics.*
Bruns, G., Godefroid, P.
CAV 1999

*Generalized model checking: reasoning about partial state spaces*
Bruns, G., Godefroid, P.
CONCUR 2000
An instance of THRIVE: Model checking

Two possible semantics of LTL over PKS can be considered

<table>
<thead>
<tr>
<th></th>
<th>Model checking</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Three-Valued</td>
<td>faster (it exploits two runs of classical model checkers)</td>
<td>Not &quot;correct&quot; when ? is returned</td>
</tr>
<tr>
<td>Thorough</td>
<td>slower (it requires more complex verification procedures)</td>
<td>Correct</td>
</tr>
</tbody>
</table>

Model checking partial state spaces with 3-valued temporal logics.
Bruns, G., Godefroid, P.
CAV 1999

Generalized model checking: reasoning about partial state spaces
Bruns, G., Godefroid, P.
CONCUR 2000
An instance of THRIVE: Model checking

The three-valued model checking can be solved as follows:

\[(M, s) \models \phi = \begin{cases} \top & \text{if } (M_{pes}, s) \models \phi \\ \bot & \text{if } (M_{opt}, s) \not\models \phi \\ ? & \text{otherwise} \end{cases}\]

*Model checking partial state spaces with 3-valued temporal logics.*
Bruns, G., Godefroid, P.
CAV 1999
An instance of THRIVE: Model checking

The three-valued model checking can be solved as follows

\[ [(M, s) \models \phi] = \begin{cases} 
\top & \text{if } (M_{pes}, s) \models \phi \\
\bot & \text{if } (M_{opt}, s) \not\models \phi \\
? & \text{otherwise}
\end{cases} \]

Model checking partial state spaces with 3-valued temporal logics.
Bruns, G., Godefroid, P.
CAV 1999
An instance of THRIVE: Model checking

The three-valued model checking can be solved as follows:

\[ [(M, s) \models \phi] = \begin{cases} \top & \text{if } (M_{pes}, s) \models \phi \\ \bot & \text{if } (M_{opt}, s) \not\models \phi \\ ? & \text{otherwise} \end{cases} \]

Model checking partial state spaces with 3-valued temporal logics.
Bruns, G., Godefroid, P.
CAV 1999
An instance of THRIVE: Model checking

The three-valued model checking can be solved as follows

\[ [(M, s) \models \phi] = \begin{cases} \top & \text{if } (M_{pes}, s) \models \phi \\ \bot & \text{if } (M_{opt}, s) \not\models \phi \\ ? & \text{otherwise} \end{cases} \]

If none of the previous condition holds

---

*Model checking partial state spaces with 3-valued temporal logics.*
Bruns, G., Godefroid, P.
CAV 1999
An instance of THRIVE: Theorem Proving

The *deductive verification* framework produces a **proof** which explains why \( M \models \phi \)
- it identifies **failed states**
- it applies a set of **deduction rules**
  - (successors, induction, conjunction rule)

*From model checking to a temporal proof.*
*Peled, Doron, and Lenore Zuck.*
*Proceedings of the 8th international SPIN workshop on Model checking of software. 2001.*
An instance of THRIVE
An instance of THRIVE: Running example

- When the light is red, it will always be green

\[ \phi_3 = \square(red \rightarrow \square green) \]

counterexample \((s_0, s_1)^w\)
An instance of THRIVE: Running example

- Green lights up infinitely often

\[ \phi_2 = \Box \Diamond \text{green}. \]

### Table

<table>
<thead>
<tr>
<th>Step 1</th>
<th>Step 2</th>
<th>Step 3</th>
<th>Step 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fail</td>
<td>Successors</td>
<td>Induction</td>
<td>Conjunction</td>
</tr>
<tr>
<td>( s_1, s_2 \in F(F_{opt}) ) ( s_2, s_1 \in F(F_{opt}) )</td>
<td>( s_0 \rightarrow { s_1, s_2 } ) ( s_1 \models g \lor \Box \Diamond g ) ( s_2 \models g \lor \Box \Diamond g )</td>
<td>( s_0 \rightarrow { s_1, s_2 } ) ( s_0 \rightarrow { s_1 } ) ( s_0 \rightarrow { s_2 } )</td>
<td>( s_0 \models \Box \Diamond g ) ( s_0 \models g \lor \Box \Diamond g ) ( \Box \Diamond g \land (g \lor \Box \Diamond g) \rightarrow \phi_2 )</td>
</tr>
</tbody>
</table>

\( (s_0, s_2) \omega \)

possible counterexample
An instance of THRIVE: Running example

- Red lights up infinitely often

$$\phi_1 = \Box \Diamond \text{red}.$$
An instance of THRIVE: Model checking

Two possible semantics of LTL over PKS can be considered

<table>
<thead>
<tr>
<th></th>
<th>Model checking</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Three-Valued</td>
<td>faster (it exploits two runs of classical model checkers)</td>
<td>Not correct when ? is returned</td>
</tr>
<tr>
<td>Thorough</td>
<td>slower (it requires more complex verification procedures)</td>
<td>Correct</td>
</tr>
</tbody>
</table>

Model checking partial state spaces with 3-valued temporal logics.
Bruns, G., Godefroid, P.
CAV 1999

Generalized model checking: reasoning about partial state spaces
Bruns, G., Godefroid, P.
CONCUR 2000
An instance of THRIVE: Correctness

• What about the **thorough** semantics?

  • In many practically interesting cases, the thorough semantics is **not more precise** than the three-valued*
  • If the LTL formula is *Self-minimizing* the result is correct**

* How thorough is thorough enough?
  Gurfinkel, A., Chechik, M.
  CHARME 2005

**Model checking vs. generalized model checking:
  semantic minimizations for temporal logics
  Godefroid, P., Huth, M.
  Logic in Computer Science, 2005
An instance of THRIVE: Correctness

• most of the **patterns** proposed in literature are expressed using self-minimising formulae *
• if satisfies some constraints (**sufficient conditions**) then it is self-minimizing **

---

* Model checking vs. generalized model checking: semantic minimizations for temporal logics.  
  Godefroid, P., Huth, M.  
  Logic in Computer Science

** Efficient patterns for model checking partial state spaces in CTL \ LTL  
  Antonik, A., Huth, M  
  Notes Theor. Comput. Sci
From Model Checking to a Temporal Proof for Partial Models

International Conference on Software Engineering and Formal Methods (SEFM)

2017

Preliminary Evaluation
Preliminary Evaluation

RQ: How effective is THRIVE w.r.t. incremental development?
Preliminary Evaluation

- we simulated the design of a critical software system*
- the system is used by physicians to check visual problems

Preliminary Evaluation

• We designed three properties that the system has to satisfy following well-known property patterns**
• We created an abstraction of the final model
• We checked how THRIVE supports incremental development

** M. B. Dwyer, G. S. Avrunin, and J. C. Corbett.
Property specification patterns for finite-state verification.
Preliminary Evaluation

For property $\psi_1$, THRIVE returns a definitive counterexample showing the reason for the violation.

The property is wrong.
Preliminary Evaluation

For property $\varphi_2$, THRIVE returns the $T$ value, since the property is satisfied.

The proof enabled us understanding the reason for the satisfaction.
For property $\psi_3$, THRIVE returns the value $?$ and
- a possible counterexample shows the violation for the pessimistic approximation
- The possible proof shows why the property of interest is satisfied on the optimistic approximation
From Model Checking to a Temporal Proof for Partial Models

International Conference on Software Engineering and Formal Methods (SEFM)

Lessons learned
Lessons learned

Creating new instances of THRIVE is **not easy**!

- **Choose/define** a *semantics* of formulae on partial models is not easy
- it **influences** the model checker and the theorem improving that can be used
Lessons learned

• The **selection** of the **model checkers** and the **theorem proving** to be combined must be done carefully to ensure the **correctness** of the obtained framework

• The **selected** model checker/theorem prover may be **changed** to be successfully combined
Conclusions
Conclusions and Future Work

• We propose THRIVE
• We show an instance of THRIVE that considers PKS and LTL
• We assess effectiveness on a simulated experiment
Conclusions and Future Work

Future Work: integrate THRIVE on top of existing theorem provers and model checkers
Integrating Topological Proofs with Model Checking to Instrument Iterative Design

Fundamental Approaches to Software Engineering (FASE)

2020

Menghi, Claudio

Rizzi, Alessandro Maria

Bernasconi, Anna
Introduction

Integrating Topological Proofs with Model Checking to Instrument Iterative Design

Fundamental Approaches to Software Engineering (FASE)

Menghi, Claudio
Rizzi, Alessandro Maria
Bernasconi, Anna

2020
Motivation

THRIVE: THRee valued Integrated Verification framEwork for partial models.

Model Checking + Theorem Proving

\[ M: \text{partial model} \]
\[ \varphi: \text{property} \]
\[ M \not\models \varphi \]

No (\(\perp\)) + counterexample

Yes (\(\top\)) + definitive proof

Maybe (?) + possible counterexample and proof

From model checking to a temporal proof for partial models
A Bernasconi, C Menghi, P Spoletini, LD Zuck, C Ghezzi
International Conference on Software Engineering and Formal Methods (SEFM), 2017
Motivation

THRIVE: THRee valued Integrated Verification framEwork for partial models.

Model Checking + Theorem Proving

$M$: partial model

$\varphi$: property

$M \models \varphi$

No ($\bot$) + counterexample

Yes ($\top$) + definitive proof

Maybe (?) + possible counterexample and proof
Motivation

Deductive proofs

- are usually difficult to understand

- their size significantly grows with the size of the model analysed
Motivation

How could we provide more effective support and guidance to engineers when properties of interest are satisfied or possibly satisfied?
Running Example

Integrating Topological Proofs with Model Checking to Instrument Iterative Design

Fundamental Approaches to Software Engineering (FASE)

Menghi, Claudio

Rizzi, Alessandro Maria

Bernasconi, Anna

2020
Vacuum-cleaner robot

<table>
<thead>
<tr>
<th>Textual Requirements</th>
<th>LTL formulae</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \phi_1 ): the robot is drawing dust (suck) only if it has reached the cleaning site.</td>
<td>( \phi_1 \equiv G(suck \rightarrow reached) )</td>
</tr>
<tr>
<td>( \phi_2 ): the robot must be turned on before it can move.</td>
<td>( \phi_2 \equiv G((\neg move) W on) )</td>
</tr>
<tr>
<td>( \phi_3 ): if the robot is on and stationary (( \neg move )), it must be drawing dust (suck).</td>
<td>( \phi_3 \equiv G(((\neg move) \land on) \rightarrow suck) )</td>
</tr>
<tr>
<td>( \phi_4 ): the robot must move before it is allowed to draw dust (suck).</td>
<td>( \phi_4 \equiv ((\neg suck) W(move \land (\neg suck))) )</td>
</tr>
</tbody>
</table>
### Vacuum-cleaner robot: Initial Design

![Diagram](image)

<table>
<thead>
<tr>
<th>Textual Requirements</th>
<th>LTL formulae</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \phi_1 ): the robot is drawing dust (\textit{suck}) only if it has reached the cleaning site.</td>
<td>( \phi_1 \equiv \mathcal{G}(\textit{suck} \rightarrow \text{reached}) )</td>
</tr>
<tr>
<td>( \phi_2 ): the robot must be turned \textit{on} before it can \textit{move}.</td>
<td>( \phi_2 \equiv \mathcal{G}((\neg \textit{move}) \mathcal{W} \textit{on}) )</td>
</tr>
<tr>
<td>( \phi_3 ): if the robot is \textit{on} and stationary ((-\textit{move}) ), it must be drawing dust (\textit{suck}).</td>
<td>( \phi_3 \equiv \mathcal{G}(((\neg \textit{move}) \land \textit{on}) \rightarrow \textit{suck}) )</td>
</tr>
<tr>
<td>( \phi_4 ): the robot must \textit{move} before it is allowed to draw dust (\textit{suck}).</td>
<td>( \phi_4 \equiv ((\neg \textit{suck}) \mathcal{W}(\textit{move} \land (\neg \textit{suck}))) )</td>
</tr>
</tbody>
</table>
Vacuum-cleaner robot: Initial Design

⊥: violated
⊤: satisfied
?: possibly satisfied

<table>
<thead>
<tr>
<th>Textual Requirements</th>
<th>LTL formulae</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\phi_1$: the robot is drawing dust ($suck$) only if it has reached the cleaning site.</td>
<td>$\phi_1 \equiv \mathcal{G}(suck \rightarrow reached)$</td>
</tr>
<tr>
<td>$\phi_2$: the robot must be turned $on$ before it can $move$.</td>
<td>$\phi_2 \equiv \mathcal{G}((\neg move) \mathcal{W} on)$</td>
</tr>
<tr>
<td>$\phi_3$: if the robot is $on$ and stationary ($\neg move$), it must be drawing dust ($suck$).</td>
<td>$\phi_3 \equiv \mathcal{G}(((\neg move) \land on) \rightarrow suck)$</td>
</tr>
<tr>
<td>$\phi_4$: the robot must $move$ before it is allowed to draw dust ($suck$).</td>
<td>$\phi_4 \equiv \mathcal{G}(\neg suck) \mathcal{W}(move \land (\neg suck))$</td>
</tr>
</tbody>
</table>
Vacuum-cleaner robot: Revision

- During a revision, an engineer can:
  - add/remove **states**
  - add/remove **transitions**
  - change the values of the **propositions**
Vacuum-cleaner robot: Revision
Integrating Topological Proofs with Model Checking to Instrument Iterative Design

Fundamental Approaches to Software Engineering (FASE)

2020

Menghi, Claudio
Rizzi, Alessandro Maria
Bernasconi, Anna
Topological Proofs

A topological proof is a slice of the model that witnesses property satisfaction.
Topological Proofs

A topological proof is a slice of the model that witnesses property satisfaction.

If the engineer does not modify elements of the models in the topological proof, then the revision will not violate the property.
TOrPEDO
TOrPEDO
TOrPEDO
TORPEDO

1. Initial design
   $\varphi: LTL$
   $M: PKS$

2. Analysis
   $\perp$ $\perp$-CE
   $\perp$ $\perp$-TP

3. Revision
   $M': PKS$
   $\text{inspire}$

4. Re-check
   $\text{inspire}$
   $\text{Correct design}$

   $\text{True, } M'$
   $\text{False, } M$
TOrPEDO
TOrPEDO
TOrPEDO
Topological Proofs

Integrating Topological Proofs with Model Checking to Instrument Iterative Design

Fundamental Approaches to Software Engineering (FASE)

2020

Menghi, Claudio
Rizzi, Alessandro Maria
Bernasconi, Anna
Topological Proofs

A topological proof is a slice of the model that witnesses property satisfaction.
Topological Proofs

Propositional Clause (TPP)

\( \langle \text{CLEANING}, \text{reached}, T \rangle \)
Propositional Clause (TPP)

\(\langle \text{CLEANING}, \text{reached}, \text{T} \rangle\)
Topological Proofs

Propositional Clause (TPP)

\[ \langle \text{CLEANING, reached, } T \rangle \]
Topological Proofs

Propositional Clause (TPP)

\[ \langle \text{CLEANING, reached, } \top \rangle \]
Topological Proofs

Propositional Clause (TPP)

⟨CLEANING, reached, ⊤⟩
Topological Proofs

Transitions-from-state Clause (TPT)

\langle \text{MOVING}, \{\text{MOVING}, \text{CLEANING}\} \rangle
Topological Proofs

Transitions-from-state Clause (TPT)

\( \langle \text{MOVING}, \{\text{MOVING, CLEANING}\} \rangle \)
Topological Proofs

Transitions-from-state Clause (TPT)

\langle \text{MOVING}, \{\text{MOVING, CLEANING}\} \rangle
Topological Proofs

Initial-states Clause (TPI)

\[ \langle \{ \text{OFF} \} \rangle \]
Topological Proofs

Initial-states Clause (TPI)

\[ \{OFF\} \]
Topological Proofs

\[ \text{TPP: } \langle \text{CLEANING, reached, } \top \rangle, \langle \text{OFF, suck, } \bot \rangle, \langle \text{IDLE, suck, } \bot \rangle, \langle \text{MOVING, suck, } ? \rangle \]

\[ \text{TPT: } \langle \text{OFF, } \{ \text{OFF, IDLE} \} \rangle, \langle \text{IDLE, } \{ \text{OFF, IDLE, MOVING} \} \rangle, \langle \text{MOVING, } \{ \text{MOVING, CLEANING} \} \rangle, \langle \text{CLEANING, } \{ \text{CLEANING, IDLE} \} \rangle \]

\[ \text{TPI: } \langle \{ \text{OFF} \} \rangle \]

\[ \phi_1: \text{the robot is drawing dust (suck) only if it has reached the cleaning site. } \quad \phi_1 \equiv G(suck \rightarrow reached) \]
Topological Proofs

• **Revision rules.** An engineer should not

  • add or remove transitions whose source state is in a transition included in the **TPT-clauses**;

  • change the value of propositions that are in a **TPP-clause**;

  • remove states that are in any **TPT, TPP, or TPI clause**;

  • change the initial states if they are in a **TPI-clause**.
Topological Proofs

If the engineer follows the revision rules, then the revision will not violate the property.
Vacuum-cleaner robot: Revision
Automated Support

Integrating Topological Proofs with Model Checking to Instrument Iterative Design

Fundamental Approaches to Software Engineering (FASE)

2020

Menghi, Claudio

Rizzi, Alessandro Maria

Bernasconi, Anna
Automated Support
Topological proof computation

Model

(P)KS

Negation of the Property

Sys2LTL

LTL: Clauses

GetUC (PLTL-MUP)

LTL: Conflicting Clauses

GetTP

Topological Proof
Topological proof computation

Model

(P)KS

LTL: Clauses

Sys2LTL

GetUC (PLTL-MUP)

LTL: Conflicting Clauses

GetTP

Topological Proof

Negation of the Property

LTL
Topological Proof Computation

* In our experiments we considered an extended version of PLTL-MUP, namely Hybrid, that improves the PLTL-MUP performances by combining it with TRP++UC.

Topological proof computation

Model

(P)KS

System to LTL (Sys2LTL)

LTL: Conflicting Clauses

GetUC (PLTL-MUP)

LTL: Clauses

GetTP

Topological Proof

Negation of the Property

LTL
Re-check
Re-check

The re-check verifies that the engineer did not:

- add or remove transitions whose source state is in a transition included in the TPT-clauses;
- change the value of propositions that are in a TPP-clause;
- remove states that are in any TPT, TPP, or TPI clause;
- change the initial states if they are in a TPI-clause.
Evaluation

Integrating Topological Proofs with Model Checking to Instrument Iterative Design

Fundamental Approaches to Software Engineering (FASE)

2020

Menghi, Claudio

Rizzi, Alessandro Maria

Bernasconi, Anna
Evaluation

• **RQ1**: How does the size of the proofs computed by the analysis component compares with the size of the original models?
RQ1: Size of The Topological Proofs

• We considered 60 model-requirement combinations
  • 12 models (PKS)
  • five properties per model
• We run TOrPEDO and computed the topological proofs
• We compared the size of the topological proof and the size of the model
RQ1: Size of The Topological Proofs

Topological proofs are approximately 60% smaller than the respective models.
Evaluation

• **RQ1**: How does the size of the proofs computed by the analysis component compares with the size of the original models?

• **RQ2**: How does the re-check component support the creation of model revisions?
RQ2: Support Provided by the Re-check Component

- We considered three models and five properties per model
- for each model we considered four revisions
- We run TOrPEDO and computed the topological proofs
- We computed the percentage of cases in which the re-check component confirmed that the revision was compliant with the topological proof
In 78% of the cases, the re-check component confirmed that the revision was compliant with the topological proof.
Evaluation

• **RQ1**: How does the size of the proofs computed by the analysis component compares with the size of the original models?

• **RQ2**: How does the re-check component support the creation of model revisions?

• **RQ3**: What is the **scalability** of TOrPEDO?
RQ3: Scalability of TOrPEDO

• To have a **ballpark estimation** of the scalability of TOrPEDO we
• assessed its performance on the models used in RQ1 and RQ2
• manually designed an additional model with 10 states and 5 atomic propositions and 26 transitions
RQ3: Scalability of TOrPEDO

For the models of RQ1 and RQ2, TOrPEDO required on average less than 10s to compute the topological proof. For the additional example, the topological proof was computed in 1m33s.
Conclusions

Integrating Topological Proofs with Model Checking to Instrument Iterative Design

Fundamental Approaches to Software Engineering (FASE)

2020

Menghi, Claudio
Rizzi, Alessandro Maria
Bernasconi, Anna
Conclusions and Future Work

• We proposed TOrPEDO, an integrated framework that supports the iterative model design

• We defined the novel notion of Topological Proofs

• We evaluated TOrPEDO by assessing the support provided by the analysis and re-check components and their scalability
Conclusions and Future Work

Our results show that

- **proofs** are 60% smaller than the original **models**
- revision can be verified **78% of the cases** by executing a simple syntactic check
- the scalability of existing tools **is not sufficient**
Conclusions and Future Work

**Future Work:** We need to develop a more efficient procedure to extract topological proofs.
TOrPEDO: Witnessing Model Correctness with Topological Proofs

Formal Aspects of Computing (FAOC)

Menghi, Claudio
Rizzi, Alessandro Maria
Bernasconi, Anna
Spoletini, Paola
Introduction
Problem Definition

In our previous work, we implemented TOrPEDO using

- **NuSMV** as a model checker, and
- **PLTL-MUP** to compute a minimal subset of unsatisfiable LTL formulae (from an unsatisfiable set of LTL formulae)

We will refer to this instance of TOrPEDO as TOrPEDO-MUP.
Topological Proof Computation

- Model
- (P)KS
- Negation of the Property

Sys2LTL

LTL: Clauses

GetUC (PLTL-MUP)

LTL: Conflicting Clauses

GetTP

Topological Proof
Problem

Can we reduce the computational cost required to compute topological proofs?
2021

TOrPEDO: Witnessing Model Correctness with Topological Proofs

Formal Aspects of Computing (FAOC)

Menghi, Claudio
Rizzi, Alessandro Maria
Bernasconi, Anna
Spoletini, Paola

Contribution
Contribution: TOrPEDO-SMT

We propose TOrPEDO-SMT

- converts LTL formulae into an SMT problem*

* Linear encodings of bounded LTL model checking
  Log Methods Comput Sci, 2,
  Episciences.org
Contribution: TOrPEDO-SMT

We propose TOrPEDO-SMT

• converts LTL formulae into an SMT problem*

• relies on Bit-Vectors**

---

* Efficient scalable verification of LTL specifications
  Baresi L, Kallehbasti MMP, Rossi M (2015)
  International conference on software engineering, pp 711–721. IEEE

** On how bit-vector logic can help verify LTL-based specifications.
  Pourhashem KMM, Rossi MG, Baresi L (2020)
  IEEE Trans Softw Eng, pp 1–1
Contribution: TOrPEDO-SMT
Contribution: TOrPEDO-SMT
Contribution: TOrPEDO-SMT

**LTL2PL**: converts LTL formulae into PL (Propositional Logic)
- Unrolls the LTL formula up to length $k$
Contribution: TOrPEDO-SMT

GetUC: computes the unsatisfiable core of a PL formula
- we employ the Z3 Theorem Prover
Contribution: TOrPEDO-SMT

PL2LTL: maps the conflicting propositional clauses to LTL
TOrPEDO: Witnessing Model Correctness with Topological Proofs

Formal Aspects of Computing (FAOC)

Evaluation

Menghi, Claudio
Rizzi, Alessandro Maria
Bernasconi, Anna
Spoletini, Paola
Evaluation

• **RQ3**: How *efficient* is TOrPEDO in *analyzing* models and how does TOrPEDO-SMT compare to TOrPEDO-MUP?
Comparison of Efficiency (RQ3): Benchmark

- We generated a set of random models
- The models have an increasing number of states (i.e., 10, 20, 30, and 40)
- The models are generated from the grade crossing semaphore example
- We considered two properties (satisfied and possibly satisfied)
Comparison of Efficiency (RQ3): Methodology

• We run TOrPEDO-MUP and TOrPEDO-SMT
• For TOrPEDO-SMT, we set 86 for the bound k*
• We set two hours as the timeout

* We selected this value since it ensures the correctness of the result, i.e., we set its value by considering to the size of the recurrence diameter (the longest initialised loop-free path in the state graph) and the size of the Büchi automaton representing the negation of the property

Clarke E, Kroening D, Ouaknine J, Strichman O (2005)
Computational challenges in bounded model checking.
Comparison of Efficiency (RQ3): Results

Fig. 4. Comparison of the efficiency of T0rPEDO-MUP and T0rPEDO-SMT. For the property $\phi_2$, T0rPEDO-MUP provided a result only for the model with 10 states in 2.1m
The answer to RQ3 is that, on the considered models,

- **TOrPEDO-SMT** can verify within the timeout models which are double in size compared to **TOrPEDO-MUP**
When both tools finished within the timeout, TOrPEDO-SMT is significantly faster than TOrPEDO-MUP. TOrPEDO-SMT required on average 1.4m, TOrPEDO-MUP required 15m.
Evaluation

- **RQ4**: How useful is TOrPEDO-SMT in supporting the designers in the model design on an example in the genomic domain?
Usefulness (RQ4): Benchmark Model

• We considered a (small) model from the genomic domain, related to Gene Regulatory Networks (GRNs).

• GRNs are collections of molecular regulators, interacting with each other.
Usefulness (RQ4): Benchmark Model

- The PKS represents the status of genes with propositions.
- The proposition is true if the gene is activated.
- states describes the status of the genes.
- The PKS consists of 64 states.
- Transitions encode how the status of the genes can change.
Usefulness (RQ4): Benchmark Model

• We considered two LTL properties from the literature discussed with domain experts
• We simulated an incremental model design with TOrPEDO
Usefulness (RQ4): Results

Table 9. LTL formulas checked on the 64 states (P)KS representing a sub-network of MAPK pathway.

<table>
<thead>
<tr>
<th>Property</th>
<th>Initial Model</th>
<th>Revision 1</th>
<th>Revision 2</th>
<th>Revision 3</th>
<th>Revision 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\phi_1$</td>
<td>ANALYSIS = $\top$</td>
<td>RE-CHECK = $\top$</td>
<td>RE-CHECK = $\top$</td>
<td>ANALYSIS = $\top$</td>
<td>ANALYSIS = $\top$</td>
</tr>
<tr>
<td>$\phi_2$</td>
<td>ANALYSIS = $?$</td>
<td>RE-CHECK = $?$</td>
<td>RE-CHECK = $?$</td>
<td>ANALYSIS = $\top$</td>
<td>ANALYSIS = $\top$</td>
</tr>
<tr>
<td>$\phi_3$</td>
<td>ANALYSIS = $\bot$</td>
<td>RE-CHECK = $\bot$</td>
<td>RE-CHECK = $\bot$</td>
<td>ANALYSIS = $\bot$</td>
<td>ANALYSIS = $\top$</td>
</tr>
</tbody>
</table>
Usefulness (RQ4): Results

Table 9. LTL formulas checked on the 64 states (P)KS representing a sub-network of MAPK pathway.

<table>
<thead>
<tr>
<th>Property</th>
<th>Initial Model</th>
<th>Revision 1</th>
<th>Revision 2</th>
<th>Revision 3</th>
<th>Revision 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\phi_1$</td>
<td>ANALYSIS = $\top$</td>
<td>RE-CHECK = $\top$</td>
<td>RE-CHECK = $\top$</td>
<td>ANALYSIS = $\top$</td>
<td>ANALYSIS = $\top$</td>
</tr>
<tr>
<td>$\phi_2$</td>
<td>ANALYSIS = $?$</td>
<td>RE-CHECK = $?$</td>
<td>RE-CHECK = $?$</td>
<td>ANALYSIS = $\top$</td>
<td>ANALYSIS = $\top$</td>
</tr>
<tr>
<td>$\phi_3$</td>
<td>ANALYSIS = $\bot$</td>
<td>RE-CHECK = $\bot$</td>
<td>RE-CHECK = $\bot$</td>
<td>ANALYSIS = $\bot$</td>
<td>ANALYSIS = $\top$</td>
</tr>
</tbody>
</table>

- We evaluated three properties on five models
Usefulness (RQ4): Results

Table 9. LTL formulas checked on the 64 states (P)KS representing a sub-network of MAPK pathway.

<table>
<thead>
<tr>
<th>Property</th>
<th>Initial Model</th>
<th>Revision 1</th>
<th>Revision 2</th>
<th>Revision 3</th>
<th>Revision 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\phi_1$</td>
<td>ANALYSIS = $\top$</td>
<td>RE-CHECK = $\top$</td>
<td>RE-CHECK = $\top$</td>
<td>ANALYSIS = $\top$</td>
<td>ANALYSIS = $\top$</td>
</tr>
<tr>
<td>$\phi_2$</td>
<td>ANALYSIS = ?</td>
<td>RE-CHECK = ?</td>
<td>RE-CHECK = ?</td>
<td>ANALYSIS = $\top$</td>
<td>ANALYSIS = $\top$</td>
</tr>
<tr>
<td>$\phi_3$</td>
<td>ANALYSIS = $\perp$</td>
<td>RE-CHECK = $\perp$</td>
<td>RE-CHECK = $\perp$</td>
<td>ANALYSIS = $\perp$</td>
<td>ANALYSIS = $\top$</td>
</tr>
</tbody>
</table>

- We evaluated three properties on five models
- We run the analysis three times
Usefulness (RQ4): Results

We evaluated three properties on five models.

We run the analysis three times and used the syntactic check twice.

Table 9. LTL formulas checked on the 64 states (P)KS representing a sub-network of MAPK pathway.

<table>
<thead>
<tr>
<th>Property</th>
<th>Initial Model</th>
<th>Revision 1</th>
<th>Revision 2</th>
<th>Revision 3</th>
<th>Revision 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\phi_1$</td>
<td>ANALYSIS = $\top$</td>
<td>RE-CHECK = $\top$</td>
<td>RE-CHECK = $\top$</td>
<td>ANALYSIS = $\top$</td>
<td>ANALYSIS = $\top$</td>
</tr>
<tr>
<td>$\phi_2$</td>
<td>ANALYSIS = ?</td>
<td>RE-CHECK = ?</td>
<td>RE-CHECK = ?</td>
<td>ANALYSIS = $\top$</td>
<td>ANALYSIS = $\top$</td>
</tr>
<tr>
<td>$\phi_3$</td>
<td>ANALYSIS = $\bot$</td>
<td>RE-CHECK = $\bot$</td>
<td>RE-CHECK = $\bot$</td>
<td>ANALYSIS = $\bot$</td>
<td>ANALYSIS = $\top$</td>
</tr>
</tbody>
</table>
Usefulness (RQ4): Results

We evaluated three properties on five models.
We run the analysis three times and used the syntactic check twice.
The topological proofs provide useful information.

<table>
<thead>
<tr>
<th>Property</th>
<th>Initial Model</th>
<th>Revision 1</th>
<th>Revision 2</th>
<th>Revision 3</th>
<th>Revision 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\phi_1$</td>
<td>ANALYSIS = $\top$</td>
<td>RE-CHECK = $\top$</td>
<td>RE-CHECK = $\top$</td>
<td>ANALYSIS = $\top$</td>
<td>ANALYSIS = $\top$</td>
</tr>
<tr>
<td>$\phi_2$</td>
<td>ANALYSIS = $?$</td>
<td>RE-CHECK = $?$</td>
<td>RE-CHECK = $?$</td>
<td>ANALYSIS = $\top$</td>
<td>ANALYSIS = $\top$</td>
</tr>
<tr>
<td>$\phi_3$</td>
<td>ANALYSIS = $\perp$</td>
<td>RE-CHECK = $\perp$</td>
<td>RE-CHECK = $\perp$</td>
<td>ANALYSIS = $\perp$</td>
<td>ANALYSIS = $\top$</td>
</tr>
</tbody>
</table>
The answer to RQ4 is that the topological proofs and counterexamples provided by TOrPEDO effectively supported the development of a (P)KS representing a gene regulatory network.
TOrPEDO: Witnessing Model Correctness with Topological Proofs

Formal Aspects of Computing (FAOC)

2021

Menghi, Claudio
Rizzi, Alessandro Maria
Bernasconi, Anna
Spoletini, Paola

Reflections
Correctness

The algorithm is **correct** if the LTL clauses are **contradicting**

- The correctness **depends** on the value of k
- If k is **higher than the completeness threshold**, the LTL clauses are **contradicting**

* Linear encodings of bounded LTL model checking
  Log Methods Comput Sci, 2, Episciences.org

* Linear completeness thresholds for bounded model checking.
  Computer aided verification, Springer

* Completeness and complexity of bounded model checking.
  International conference on verification, model checking, and abstract interpretation, Springer
Practical Guidelines

• Designers can
  • initially choose a value for \( k \) that is reasonably large
  • increase or decrease the value of \( k \) depending on
  • the efficiency of the analysis
  • the importance of the soundness
Why Faster

• TORPEDO-MUP is FSPACE complete, TORPEDO-SMT is NP-complete
Why Faster

• The Z3 Theorem Prover offers a mature technology;
  • an industry-strength tool,
  • awarded by ETAPS (Test of Time Award) and ACM SIGPLAN (Programming Languages Software Award)
Trace Diagnostics for Signal-based Temporal Properties

IEEE Transactions on Software Engineering (TSE)

Boufaied, Chaima
Menghi, Claudio
Bianculli, Domenico
Briand, Lionel C

2023
Trace Diagnostics

System

Execution Trace

System Requirements

properties

Trace Checking

Violated
Problem

How do we explain why a property is violated by a trace?
Contribution (TD-SB-TemPsy)

TD-SB-TemPsy: A trace-diagnostic approach for signal-based temporal properties.

• analyzes a trace and a property violated by the trace;
• provides an explanation for the property violation.
Contribution (TD-SB-TemPsy)

TD-SB-TemPsy relies on

- violation causes and
- diagnoses.
Contribution (TD-SB-TemPsy)

Violation cause: characterizes one of the possible behaviors of the system that may lead to the property violation.

Diagnoses: information associated with the property violation.
Violation cause: characterizes one of the possible behaviors of the system that may lead to the property violation.
Violation cause: characterizes one of the possible behaviors of the system that may lead to the property violation.

A violation cause should satisfy the following relation:
• if the violation cause holds, then the corresponding requirement should be violated
Topological Proofs and Violation Clauses: Parallelism

A topological proof is a slice of the model that witnesses property satisfaction.

A violation cause is a construct that if satisfied by a (slice) of the trace witnesses property violation.
Contribution (TD-SB-TemPsy)

The paper describes

- **TD-SB-TemPsy**, a trace-diagnostic approach for signal-based temporal properties expressed in SB-TemPsy-DSL,
- a methodology for defining violation causes and diagnoses, with formal guarantees of the soundness of the violation causes
Contribution (TD-SB-TemPsy)

The paper describes

• a catalog of 34 violation causes, each associated with one diagnosis,

• evaluates TD-SBTemPsy on two datasets, including one industrial case study.
TD-SB-TemPsy Evaluation

Evaluated with an Industrial Case Study

- 361 traces given by our industrial partner
- 98 requirements specified in SB-TemPsy-DSL
- Total: 35378 trace - property combinations
TD-SB-TemPsy yielded a diagnosis within a timeout of 1 minute for 83.66% of the combinations.
Reflections and Lessons Learned and Speculations

2017
From Model Checking to a Temporal Proof for Partial Models
International Conference on Software Engineering and Formal Methods (SEFM)
Bernasconi, Anna; Menghi, Claudio; Spoletini, Paola; Zuck, Lenore B; Ghezzi, Carlo

2020
Integrating Topological Proofs with Model Checking to Instrument Iterative Design
Fundamental Approaches to Software Engineering (FASE)
Menghi, Claudio; Rizzi, Alessandro Maria; Bernasconi, Anna

2021
TORPEDO: Witnessing Model Correctness with Topological Proofs
Formal Aspects of Computing (FAOC)
Menghi, Claudio; Rizzi, Alessandro Maria; Bernasconi, Anna; Spoletini, Paola

2023
Trace Diagnostics for Signal-based Temporal Properties
IEEE Transactions on Software Engineering (TSE), Boufaied, Chaima; Menghi, Claudio; Bianculli, Domenico; Briand, Lionel C
Reflections and Lessons Learned and Speculations

**Reflection 1**: There is a synergy between theory and practice

Automated Verification of Cyber-Physical Systems: From Theory to Practice Workshop on Software Reliability for Madrid Flight on Chip
https://flightonchip.es/workshop19/

Verification and Validation: from Theory to Practice and Back Again
November 6th, 2020
https://www.deib.polimi.it/eng/events/details/2111
Reflection 1: There is a synergy between theory and practice
Reflections and Lessons Learned and Speculations

**Reflection 1:** There is a synergy between theory and practice

Automated Verification of Cyber-Physical Systems: From Theory to Practice
Workshop on Software Reliability for Madrid Flight on Chip
https://flightonchip.es/workshop19/

Verification and Validation: from Theory to Practice and Back Again
November 6th, 2020
https://www.deib.polimi.it/eng/events/details/2111
Reflection 1: There is a synergy between theory and practice

Reflections and Lessons Learned and Speculations

Automated Verification of Cyber-Physical Systems: From Theory to Practice
Workshop on Software Reliability for Madrid Flight on Chip
https://flightonchip.es/workshop19/

Verification and Validation: from Theory to Practice and Back Again
November 6th, 2020
https://www.deib.polimi.it/eng/events/details/2111
Reflections and Lessons Learned and Speculations

**Reflection 1:** There is a synergy between **theory** and **practice**

Automated Verification of Cyber-Physical Systems: From Theory to Practice Workshop on Software Reliability for Madrid Flight on Chip
https://flightonchip.es/workshop19/

Verification and Validation: from Theory to Practice and Back Again
November 6th, 2020
https://www.deib.polimi.it/eng/events/details/2111
Reflections and Lessons Learned and Speculations

Reflection 2: The results are teamwork

Bernasconi, Anna
Rizzi, Alessandro Maria
Bianculli, Domenico
Boufaied, Chaima
Spoletini, Paola
Ghezzi, Carlo
Zuck, Lenore D
Briand, Lionel C
Reflections and Lessons Learned and Speculations

**Reflection 2:** The results are teamwork

Bernasconi, Anna  Rizzi, Alessandro Maria

They are first authors!
Reflections and Lessons Learned and Speculations

**Reflection 3**: Some of the reviewers significantly helped us in improving the papers.

VMCAI 2019: REVIEW 3 (Reject)

*For LTL formulae, the separated normal form [...] One can create an *equisatisfiable normalized formula*, but not an equivalent one. Why this should still work and how the reasons/understanding is explained using a non-equivalent formula is not discussed at all.*

---

It was indeed equivalent. Thanks a lot!
Reflections and Lessons Learned and Speculations

**Reflection 4**: Did we reach "The Independence Day of Witnessing the Correctness of Systems"?
Reflections and Lessons Learned and Speculations

**Reflection 4**: Did we reach ``The Independence Day of Witnessing the Correctness of Systems”?

Well, no, I think there is a lot of work that still to be done.
Reflections and Lessons Learned and Speculations

Variety of the modeling formalisms
Reflections and Lessons Learned and Speculations

Variety of the modeling formalisms

Variety of the Requirements Specification Languages
Reflections and Lessons Learned and Speculations

Variety of the modeling formalisms

Variety of the Requirements Specification Languages

Trade-off
Expressiveness and Performances
Reflections and Lessons Learned and Speculations

Variety of the modeling formalisms

Variety of the Requirements Specification Languages

Trade-off
Expressiveness and Performances

Developing Techniques that are Complete
Reflections and Lessons Learned and Speculations

- Variety of the modeling formalisms
- Variety of the Requirements Specification Languages
- Trade-off Expressiveness and Performances
- Usability for the End Users
- Developing Techniques that are Complete
Reflections and Lessons Learned and Speculations

Reaching
````The Independence Day of Witnessing
the Correctness of Systems””
is a journey, everyone is invited!

Enjoy the trip!
The Independence Day of Witnessing the Correctness of Systems: From Topological Proofs and Beyond

BCS FACS (Formal Aspects of Computing Science)