Counterexample searches in Spot

Clément Gillard

LRDE
EPITA Research and Development Laboratory

Seminar – 2019-01-16

2019-01-27

Counterexample searches in Spot

Counterexample searches in Spot

Clément Gillard

LRDE
EPITA Research and Development Laboratory

Seminar – 2019-01-16

ω-automata Model Checking 101

$$M \models \varphi$$

ω-automata Model Checking 101

$M \models \varphi$

ω-automata Model Checking 101

$$M \models \varphi$$

$$\iff \quad L(A_M) \subseteq L\left(A_\varphi\right)$$

ω-automata Model Checking 101

$M \models \varphi$

$\iff L(A_M) \subseteq L\left(A_\varphi\right)$

ω-automata Model Checking 101

$$
\begin{aligned}
& M \models \varphi \\
\iff \quad & L\left(A_M\right) \subseteq L\left(A_\varphi\right) \\
\iff \quad & L\left(A_M\right) \cap L\left(\overline{A_\varphi}\right) = \emptyset
\end{aligned}
$$

ω-automata Model Checking 101

$$
\begin{aligned}
& M \models \varphi \\
\iff \quad & L\left(A_M\right) \subseteq L\left(A_\varphi\right) \\
\iff \quad & L\left(A_M\right) \cap L\left(\overline{A_\varphi}\right) = \emptyset
\end{aligned}
$$

ω-automata Model Checking 101

$$
\begin{aligned}
& M \models \varphi \\
\iff \quad & L(A_M) \subseteq L\left(A_\varphi\right) \\
\iff \quad & L(A_M) \cap L\left(\overline{A_\varphi}\right) = \emptyset \\
\iff \quad & L(A_M) \cap L\left(A_{\neg\varphi}\right) = \emptyset
\end{aligned}
$$

ω-automata Model Checking 101

$$
\begin{aligned}
& M \models \varphi \\
\iff \quad & L(A_M) \subseteq L\left(A_\varphi\right) \\
\iff \quad & L(A_M) \cap L\left(\overline{A_\varphi}\right) = \emptyset \\
\iff \quad & L(A_M) \cap L\left(A_{\neg\varphi}\right) = \emptyset
\end{aligned}
$$

ω-automata Model Checking 101

$$
\begin{aligned}
& M \models \varphi \\
\iff\quad & L(A_M) \subseteq L\left(A_\varphi\right) \\
\iff\quad & L(A_M) \cap L\left(\overline{A_\varphi}\right) = \emptyset \\
\iff\quad & L(A_M) \cap L\left(A_{\neg\varphi}\right) = \emptyset \\
\iff\quad & L\left(A_M \otimes A_{\neg\varphi}\right) = \emptyset
\end{aligned}
$$

ω-automata Model Checking 101

$M \models \varphi$
$\iff L(A_M) \subseteq L\left(A_\varphi\right)$
$\iff L(A_M) \cap L\left(\overline{A_\varphi}\right) = \emptyset$
$\iff L(A_M) \cap L\left(A_{\neg\varphi}\right) = \emptyset$
$\iff L\left(A_M \otimes A_{\neg\varphi}\right) = \emptyset$

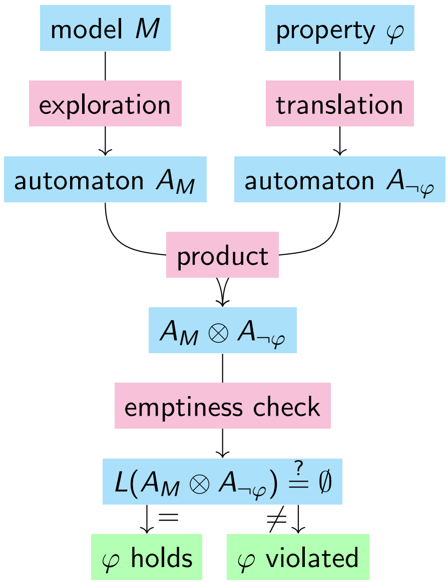
ω-automata Model Checking 101

$$
\begin{aligned}
& M \models \varphi \\
\iff \quad & L\left(A_M\right) \subseteq L\left(A_\varphi\right) \\
\iff \quad & L\left(A_M\right) \cap L\left(\overline{A_\varphi}\right) = \emptyset \\
\iff \quad & L\left(A_M\right) \cap L\left(A_{\neg\varphi}\right) = \emptyset \\
\iff \quad & {\color{blue} L\left(A_M \otimes A_{\neg\varphi}\right) = \emptyset}
\end{aligned}
$$

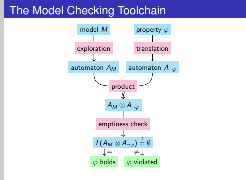
ω-automata Model Checking 101

$$
\begin{aligned}
& M \models \varphi \\
\iff \quad & L\left(A_M\right) \subseteq L\left(A_\varphi\right) \\
\iff \quad & L\left(A_M\right) \cap L\left(\overline{A_\varphi}\right) = \emptyset \\
\iff \quad & L\left(A_M\right) \cap L\left(A_{\neg\varphi}\right) = \emptyset \\
\iff \quad & L\left(A_M \otimes A_{\neg\varphi}\right) = \emptyset
\end{aligned}
$$

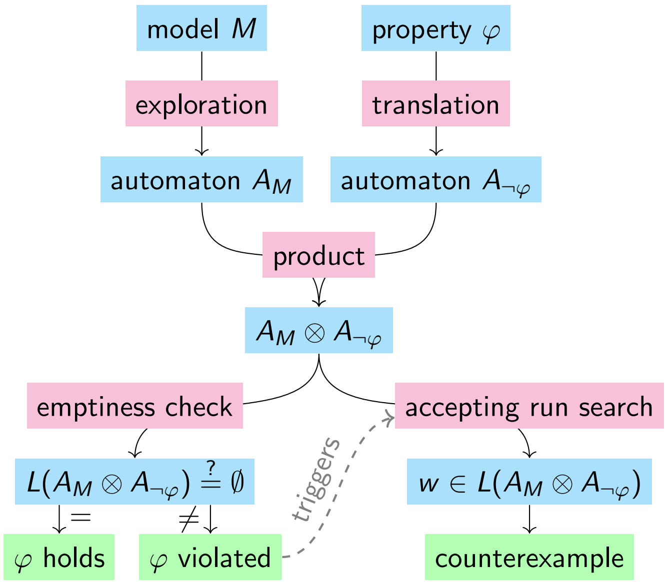
The Model Checking Toolchain

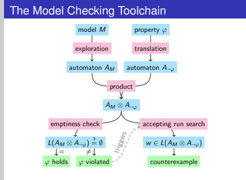


2019-01-27



The Model Checking Toolchain





ω-automata Equivalence

$$A_1 \equiv A_2$$

ω-automata Equivalence

$A_1 \equiv A_2$

ω-automata Equivalence

$$
\begin{aligned}
& A_1 \equiv A_2 \\
\iff \quad & L(A_1) = L(A_2)
\end{aligned}
$$

ω-automata Equivalence

$A_1 \equiv A_2$
$\iff L(A_1) = L(A_2)$

ω-automata Equivalence

$$
\begin{aligned}
& A_1 \equiv A_2 \\
\iff \quad & L(A_1) = L(A_2) \\
\iff \quad & L(A_1) \subseteq L(A_2) \quad \textit{and} \quad L(A_1) \supseteq L(A_2)
\end{aligned}
$$

ω-automata Equivalence

$A_1 \equiv A_2$
$\iff L(A_1) = L(A_2)$
$\iff L(A_1) \subseteq L(A_2)$ *and* $L(A_1) \supseteq L(A_2)$

ω-automata Equivalence

$$
\begin{aligned}
& A_1 \equiv A_2 \\
\iff \quad & L(A_1) = L(A_2) \\
\iff \quad & L(A_1) \subseteq L(A_2) \qquad \textit{and} \quad L(A_1) \supseteq L(A_2) \\
\iff \quad & L(A_1) \cap L\left(\overline{A_2}\right) = \emptyset \quad \textit{and} \quad L\left(\overline{A_1}\right) \cap L(A_2) = \emptyset
\end{aligned}
$$

ω-automata Equivalence

$$
\begin{aligned}
& A_1 \equiv A_2 \\
\iff \quad & L(A_1) = L(A_2) \\
\iff \quad & L(A_1) \subseteq L(A_2) \qquad \textit{and} \quad L(A_1) \supseteq L(A_2) \\
\iff \quad & L(A_1) \cap L\left(\overline{A_2}\right) = \emptyset \quad \textit{and} \quad L\left(\overline{A_1}\right) \cap L(A_2) = \emptyset
\end{aligned}
$$

ω-automata Equivalence

$$
\begin{aligned}
& A_1 \equiv A_2 \\
\iff \quad & L(A_1) = L(A_2) \\
\iff \quad & L(A_1) \subseteq L(A_2) \quad \textit{and} \quad L(A_1) \supseteq L(A_2) \\
\iff \quad & L(A_1) \cap L\left(\overline{A_2}\right) = \emptyset \quad \textit{and} \quad L\left(\overline{A_1}\right) \cap L(A_2) = \emptyset \\
\iff \quad & L\left(A_1 \otimes \overline{A_2}\right) = \emptyset \quad \textit{and} \quad L\left(\overline{A_1} \otimes A_2\right) = \emptyset
\end{aligned}
$$

ω-automata Equivalence

$$
\begin{aligned}
& A_1 \equiv A_2 \\
\iff \quad & L(A_1) = L(A_2) \\
\iff \quad & L(A_1) \subseteq L(A_2) \quad \textit{and} \quad L(A_1) \supseteq L(A_2) \\
\iff \quad & L(A_1) \cap L\left(\overline{A_2}\right) = \emptyset \quad \textit{and} \quad L\left(\overline{A_1}\right) \cap L(A_2) = \emptyset \\
\iff \quad & L\left(A_1 \otimes \overline{A_2}\right) = \emptyset \quad \textit{and} \quad L\left(\overline{A_1} \otimes A_2\right) = \emptyset
\end{aligned}
$$

ω-automata Equivalence

$$
\begin{array}{lllll}
 & A_1 \equiv A_2 & & & \\
\iff & L(A_1) = L(A_2) & & & \\
\iff & L(A_1) \subseteq L(A_2) & \textit{and} & L(A_1) \supseteq L(A_2) \\
\iff & L(A_1) \cap L\left(\overline{A_2}\right) = \emptyset & \textit{and} & L\left(\overline{A_1}\right) \cap L(A_2) = \emptyset \\
\iff & \color{blue}{L\left(A_1 \otimes \overline{A_2}\right) = \emptyset} & \textit{and} & \color{blue}{L\left(\overline{A_1} \otimes A_2\right) = \emptyset}
\end{array}
$$

ω-automata Equivalence

$$
\begin{array}{lllll}
 & A_1 \equiv A_2 & & & \\
\iff & L(A_1) = L(A_2) & & & \\
\iff & L(A_1) \subseteq L(A_2) & \textit{and} & L(A_1) \supseteq L(A_2) \\
\iff & L(A_1) \cap L\left(\overline{A_2}\right) = \emptyset & \textit{and} & L\left(\overline{A_1}\right) \cap L(A_2) = \emptyset \\
\iff & L\left(A_1 \otimes \overline{A_2}\right) = \emptyset & \textit{and} & L\left(\overline{A_1} \otimes A_2\right) = \emptyset
\end{array}
$$

ω-automata Equivalence

$$
\begin{aligned}
& A_1 \equiv A_2 \\
\iff\quad & L(A_1) = L(A_2) \\
\iff\quad & L(A_1) \subseteq L(A_2) \quad \textit{and} \quad L(A_1) \supseteq L(A_2) \\
\iff\quad & L(A_1) \cap L\left(\overline{A_2}\right) = \emptyset \quad \textit{and} \quad L\left(\overline{A_1}\right) \cap L(A_2) = \emptyset \\
\iff\quad & L\left(A_1 \otimes \overline{A_2}\right) = \emptyset \quad \textit{and} \quad L\left(\overline{A_1} \otimes A_2\right) = \emptyset
\end{aligned}
$$

A counterexample helps us debug.

ω-automata Equivalence

$A_1 \equiv A_2 \iff L(A_1) = L(A_2) \iff L(A_1) \subseteq L(A_2)$ and $L(A_1) \supseteq L(A_2) \iff L(A_1) \cap L(\overline{A_2}) = \emptyset$ and $L(\overline{A_1}) \cap L(A_2) = \emptyset \iff L(A_1 \otimes \overline{A_2}) = \emptyset$ and $L(\overline{A_1} \otimes A_2) = \emptyset$

A counterexample helps us debug.

Spot 2

- Duret-Lutz et al., “Spot 2.0 — a framework for LTL and ω-automata manipulation”, ATVA’16 [2]
- C++ library, Python interface
- algorithms and bridges for model checking
- executables for testing and comparing tools: `ltlcross`, `autcross`

Spot 2

- Duret-Lutz et al., “Spot 2.0 — a framework for LTL and ω-automata manipulation”, ATVA’16 [2]
- C++ library, Python interface
- algorithms and bridges for model checking
- executables for testing and comparing tools: `ltlcross`, `autcross`

Multiple algorithms

Name	Type of automata	Emptiness check	Accepting run search
Couvreur New:	TGBA – on-the-fly – explicit	 ✔ ✔	 ✔ ✔
Product EC&CE:	TGBA – on-the-fly – explicit	 ✔ ✔	 ✔ ✔
Generic EC:	Any – explicit	 ✔	

Multiple algorithms

Multiple algorithms

Name	Type of automata	Emptiness check	Accepting run search	
Couvreur New:	TGBA – on-the-fly – explicit	 ✔ ✔	 ✔ ✔	
Product EC&CE:	TGBA – on-the-fly – explicit	 ✔ (2017) ✔ (2017)	 ✔ (2018) ✔ (2018)	 (benches 2019)
Generic EC:	Any – explicit	 ✔	 ✔ (2019)	

Multiple algorithms

Counterexample searches in Spot

1. Introduction
2. Generalities on accepting run searches
3. Product accepting run search
4. Generic accepting run search
5. Conclusion

Counterexample searches in Spot
Introduction
Generalities on accepting run searches
Product accepting run search
Generic accepting run search
Conclusion

Generalities on accepting run searches

1 Introduction

2 Generalities on accepting run searches

3 Product accepting run search

4 Generic accepting run search

5 Conclusion

└Generalities on accepting run searches

Generalities on accepting run searches

Introduction
Generalities on accepting run searches
Product accepting run search
Generic accepting run search
Conclusion

Papers on efficient counterexample search

- Gastin, Moro, and Zeitoun, “Minimization of counterexamples in SPIN”, SPIN’04 [5]
- Hansen and Kervinen, “Minimal Counterexamples in $O(n \log n)$ Memory and $O(n^2)$ Time”, ACSD’06 [8]
- Gastin and Moro, “Minimal Counterexample Generation for SPIN”, MCS’07 [4]
- Hansen and Geldenhuys, “Cheap and Small Counterexamples”, SEFM’08 [7]

Papers on efficient counterexample search

- Gastin, Moro, and Zeitoun, “Minimization of counterexamples in SPIN”, SPIN’04 [5]
- Hansen and Kervinen, “Minimal Counterexamples in $O(n \log n)$ Memory and $O(n^2)$ Time”, ACSD’06 [8]
- Gastin and Moro, “Minimal Counterexample Generation for SPIN”, MCS’07 [4]
- Hansen and Geldenhuys, “Cheap and Small Counterexamples”, SEFM’08 [7]

Papers on efficient counterexample search

- Gastin, Moro, and Zeitoun, “Minimization of counterexamples in SPIN”, SPIN’04 [5]
  ✘ no knowledge search
- Hansen and Kervinen, “Minimal Counterexamples in $O(n \log n)$ Memory and $O(n^2)$ Time”, ACSD’06 [8]
  ✘ no knowledge search
- Gastin and Moro, “Minimal Counterexample Generation for SPIN”, MCS’07 [4]
  ✔ path reconstruction...
- Hansen and Geldenhuys, “Cheap and Small Counterexamples”, SEFM’08 [7]
  ✘ no knowledge search

Papers on efficient counterexample search

- Gastin, Moro, and Zeitoun, “Minimization of counterexamples in SPIN”, SPIN’04 [5]
  ✘ no knowledge search
- Hansen and Kervinen, “Minimal Counterexamples in $O(n \log n)$ Memory and $O(n^2)$ Time”, ACSD’06 [8]
  ✘ no knowledge search
- Gastin and Moro, “Minimal Counterexample Generation for SPIN”, MCS’07 [4]
  ✔ path reconstruction...
- Hansen and Geldenhuys, “Cheap and Small Counterexamples”, SEFM’08 [7]
  ✘ no knowledge search

Papers on efficient counterexample search

- Gastin, Moro, and Zeitoun, "Minimization of counterexamples in SPIN", SPIN'04 [5]
  ✘ no knowledge search
- Hansen and Kervinen, "Minimal Counterexamples in $O(n \log n)$ Memory and $O(n^2)$ Time", ACSD'06 [8]
  ✘ no knowledge search
- Gastin and Moro, "Minimal Counterexample Generation for SPIN", MCS'07 [4]
  ✔✘ path reconstruction... for bitstate hashing
- Hansen and Geldenhuys, "Cheap and Small Counterexamples", SEFM'08 [7]
  ✘ no knowledge search

Papers on efficient counterexample search

- Gastin, Moro, and Zeitoun, "Minimization of counterexamples in SPIN", SPIN'04 [5]
  ✘ no knowledge search
- Hansen and Kervinen, "Minimal Counterexamples in $O(n \log n)$ Memory and $O(n^2)$ Time", ACSD'06 [8]
  ✘ no knowledge search
- Gastin and Moro, "Minimal Counterexample Generation for SPIN", MCS'07 [4]
  ✔✘ path reconstruction... for bitstate hashing
- Hansen and Geldenhuys, "Cheap and Small Counterexamples", SEFM'08 [7]
  ✘ no knowledge search

Quote

From “Minimal Counterexample Generation for SPIN” [4]:

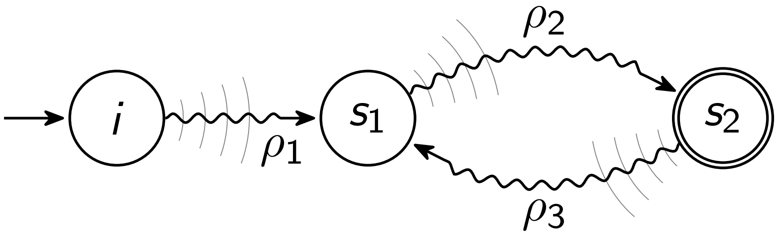
> *“If states are stored in an hash table as usual, one can recover the trace of the counter-example using a BFS algorithm [. . .]. It then suffices to apply this BFS from the initial state i to s_1 to generate ρ_1, then to apply it from s_1 to s_2 to generate ρ_2 and finally to apply it once more from s_2 to s_1 to generate ρ_3.”*
>
> Gastin and Moro

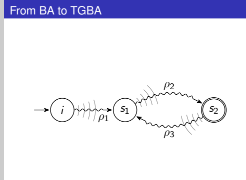
Quote

From “Minimal Counterexample Generation for SPIN” [4]:

“If states are stored in an hash table as usual, one can recover the trace of the counter-example using a BFS algorithm [. . .]. It then suffices to apply this BFS from the initial state i to s_1 to generate ρ_1, then to apply it from s_1 to s_2 to generate ρ_2 and finally to apply it once more from s_2 to s_1 to generate ρ_3.” Gastin and Moro

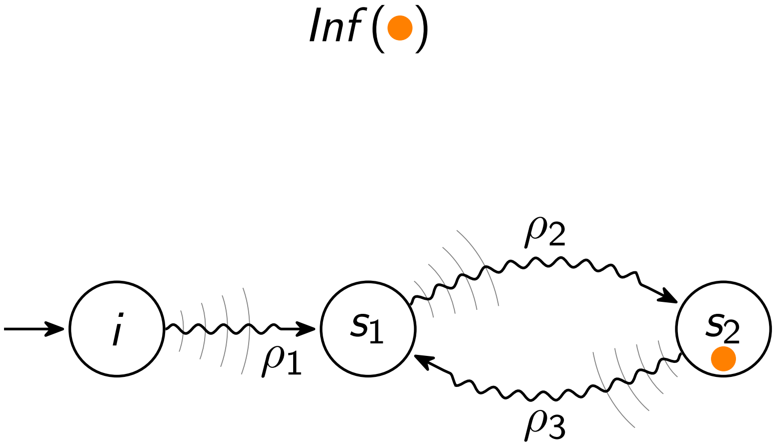
From BA to TGBA



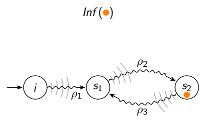


1. acceptance sets, acceptance marks, acceptance condition

From BA to TGBA



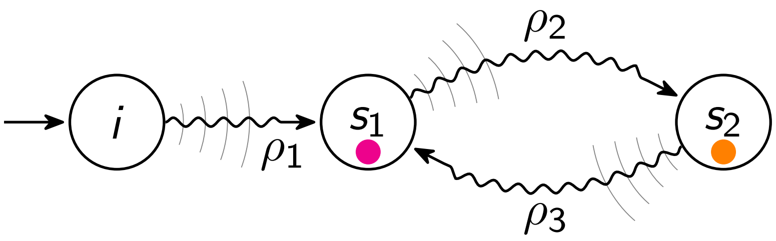
From BA to TGBA



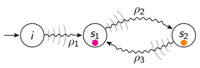
1. more accepting sets

From BA to TGBA

$$Inf(\textcolor{magenta}{\bullet}) \wedge Inf(\textcolor{orange}{\bullet})\ldots$$



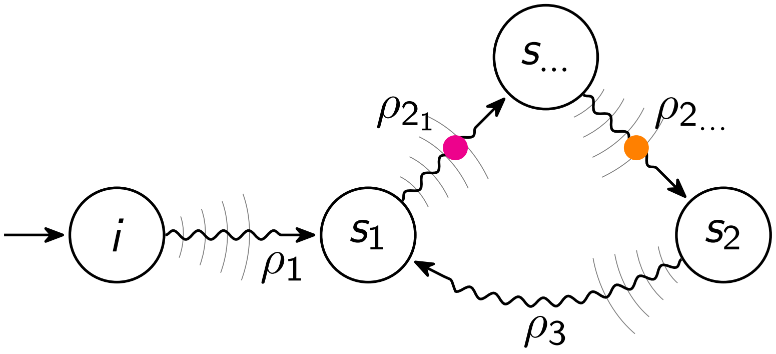
From BA to TGBA
$Inf(\bullet) \wedge Inf(\bullet)\ldots$

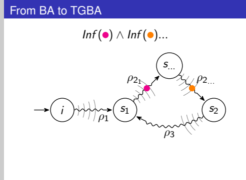


1. accepting marks on transitions

From BA to TGBA

$$Inf(\bullet) \wedge Inf(\bullet)\ldots$$





1. s_2 becomes the last state to see, ρ_2 becomes multiple BFS

Product accepting run search

1 Introduction

2 Generalities on accepting run searches

3 Product accepting run search

4 Generic accepting run search

5 Conclusion

└Product accepting run search

Product accepting run search

Introduction
Generalities on accepting run searches
Product accepting run search
Generic accepting run search
Conclusion

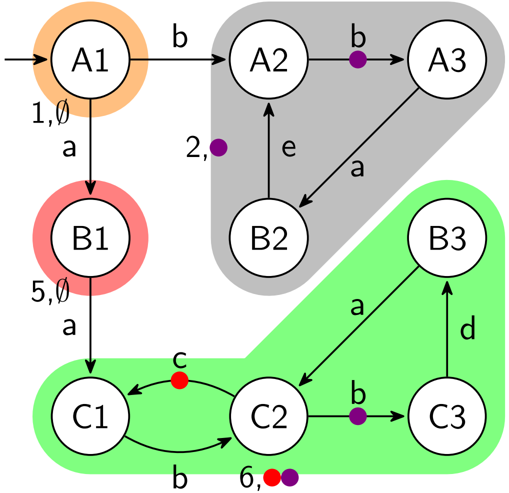
The Product Emptiness Check

- Gillard, Two-automaton emptiness check in Spot, 2017 [6]
- Algorithm by Couvreur, “On-the-fly Verification of Temporal Logic”, FM’99 [1]
- Removes a limitation of building the product by simulating it with custom data structures
- Generalised Büchi acceptance condition only

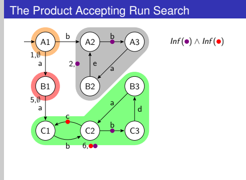
The Product Emptiness Check

- Gillard, Two-automaton emptiness check in Spot, 2017 [6]
- Algorithm by Couvreur, “On-the-fly Verification of Temporal Logic”, FM’99 [1]
- Removes a limitation of building the product by simulating it with custom data structures
- Generalised Büchi acceptance condition only

The Product Accepting Run Search

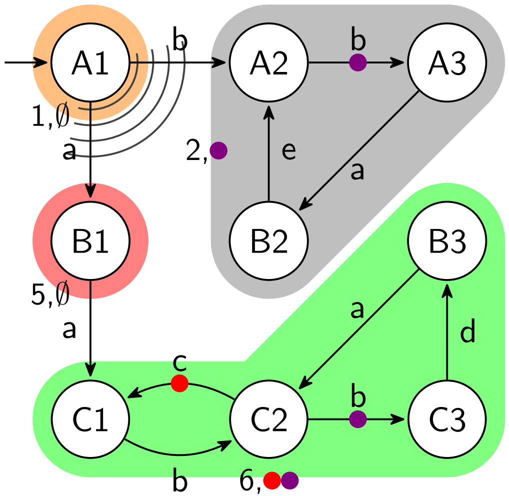


$Inf(\bullet) \wedge Inf(\bullet)$



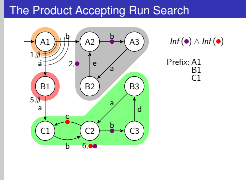
1. States of the SCC
2. Marks of the SCC

The Product Accepting Run Search



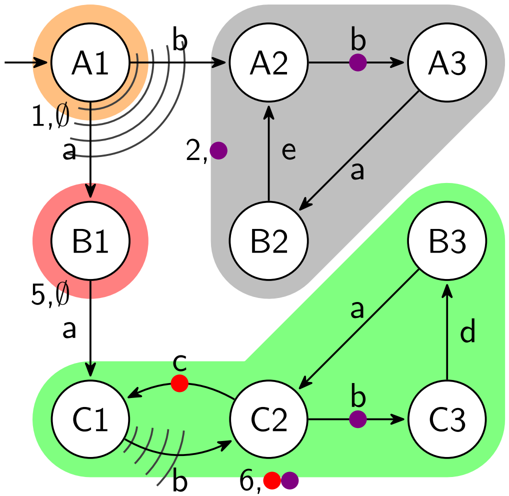
$Inf(\bullet) \wedge Inf(\bullet)$

Prefix: A1
B1
C1



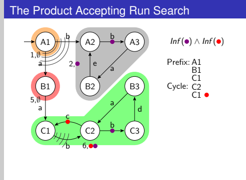
1. States of the SCC
2. Marks of the SCC

The Product Accepting Run Search



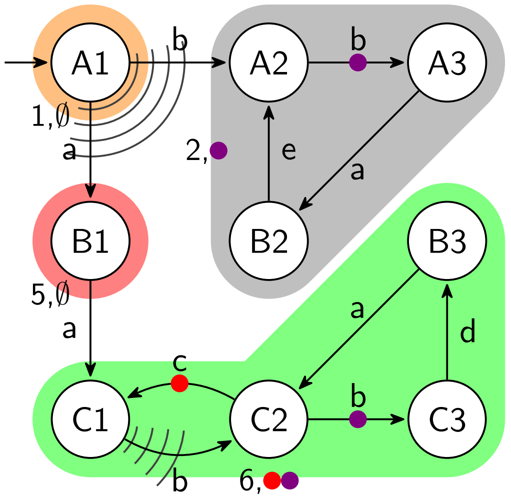
$Inf(\bullet) \wedge Inf(\bullet)$

Prefix: A1
B1
C1
Cycle: C2
C1



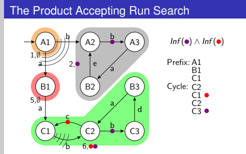
1. States of the SCC
2. Marks of the SCC

The Product Accepting Run Search



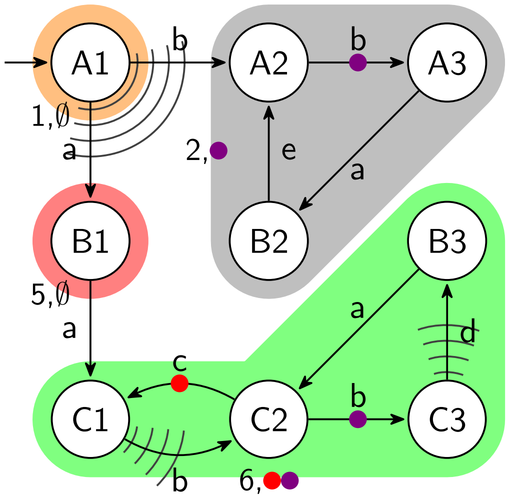
Inf(●) ∧ *Inf*(●)

Prefix: A1
B1
C1
Cycle: C2
C1 ●
C2
C3 ●



1. States of the SCC
2. Marks of the SCC

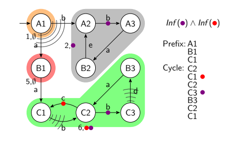
The Product Accepting Run Search



$Inf(\bullet) \wedge Inf(\bullet)$

Prefix: A1
B1
C1
Cycle: C2
C1 •
C2
C3 •
B3
C2
C1

The Product Accepting Run Search

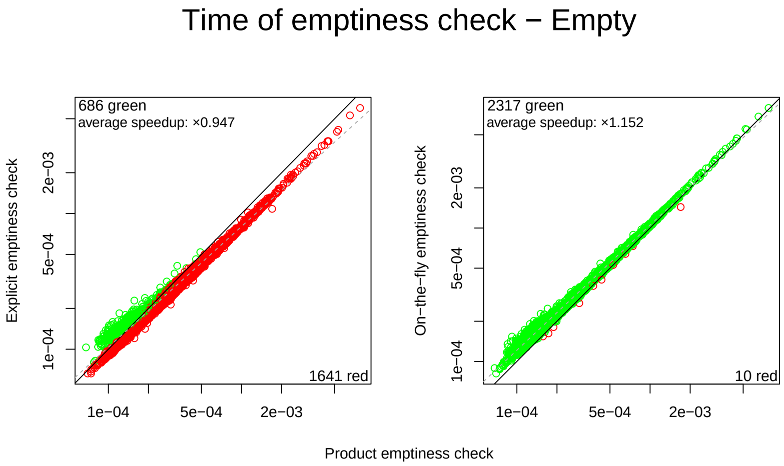


1. States of the SCC
2. Marks of the SCC

- Knowledge:
  - map: *state* of $A_1 \times$ *state* of $A_2 \to$ *order*
  - "*order*" of s_1
  - marks seen in the accepting SCC
- Implementation:
  - Reimplementation of the `bfs_steps` algorithm
  - BFS from i to s_1 $\to \rho_1$
  - repeated BFSs to cross off marks, until s_2 is reached $\to \rho_2$
  - BFS from s_2 to s_1 $\to \rho_3$

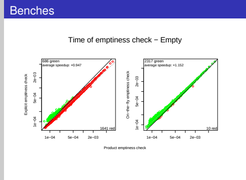
The Product Accepting Run Search
- Knowledge:
  - map: *state* of $A_1 \times$ *state* of $A_2 \to$ *order*
  - "*order*" of s_1
  - marks seen in the accepting SCC
- Implementation:
  - Reimplementation of the `bfs_steps` algorithm
  - BFS from i to s_1 $\to \rho_1$
  - repeated BFSs to cross off marks, until s_2 is reached $\to \rho_2$
  - BFS from s_2 to s_1 $\to \rho_3$

Benches

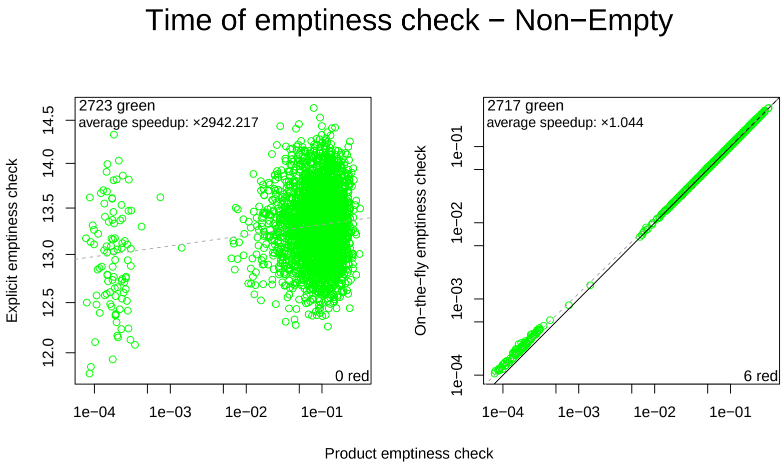


2019-01-27

Counterexample searches in Spot
└Product accepting run search
  └Benches
    └Benches

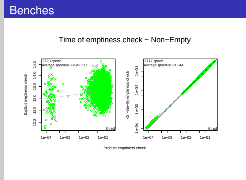


Benches

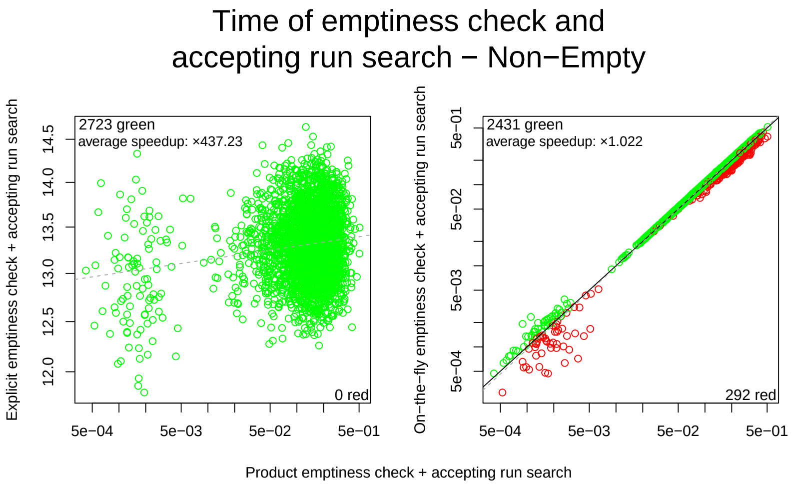


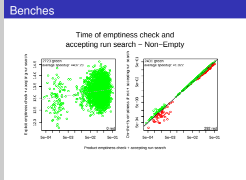
2019-01-27

Counterexample searches in Spot
└Product accepting run search
└Benches
└Benches



Benches


Time of emptiness check and
accepting run search − Non−Empty
2723 green
average speedup: ×437.23
0 red
Explicit emptiness check + accepting run search
12.0
12.5
13.0
13.5
14.0
14.5
2431 green
average speedup: ×1.022
292 red
On−the−fly emptiness check + accepting run search
5e−04
5e−03
5e−02
5e−01
5e−04
5e−03
5e−02
5e−01
5e−04
5e−03
5e−02
5e−01
Product emptiness check + accepting run search


Benches
Time of emptiness check and
accepting run search − Non−Empty

Generic accepting run search

1 Introduction

2 Generalities on accepting run searches

3 Product accepting run search

4 Generic accepting run search

5 Conclusion

└Generic accepting run search

Generic accepting run search

Introduction
Generalities on accepting run searches
Product accepting run search
Generic accepting run search
Conclusion

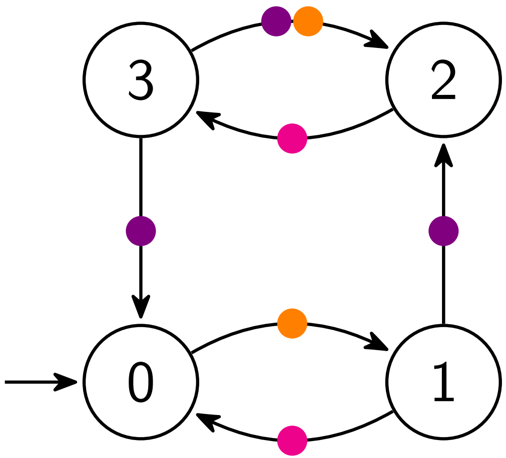
The Generic Emptiness Check

- New recursive algorithm
- Works with any acceptance condition
- NP-complete
  - Emerson and Lei, “Modalities for Model Checking: Branching Time Logic Strikes Back”, SCP’87 [3]

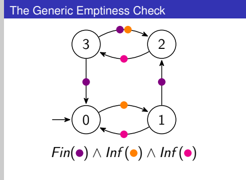
The Generic Emptiness Check

- New recursive algorithm
- Works with any acceptance condition
- NP-complete
  - Emerson and Lei, “Modalities for Model Checking: Branching Time Logic Strikes Back”, SCP’87 [3]

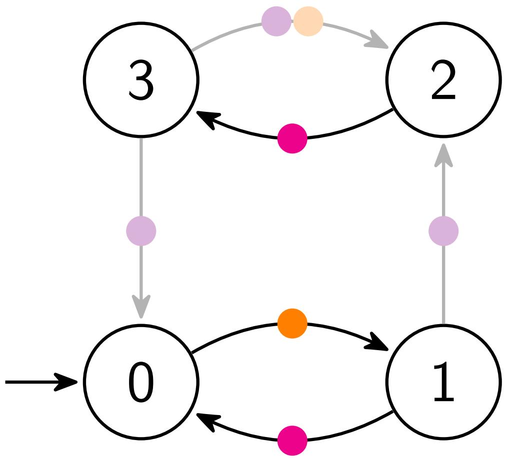
The Generic Emptiness Check



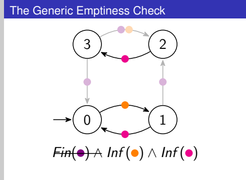
$$Fin(\bullet) \land Inf(\bullet) \land Inf(\bullet)$$



The Generic Emptiness Check



$\cancel{Fin(\bullet) \land}\ Inf(\bullet) \land Inf(\bullet)$



The Generic Accepting Run Search

- Knowledge:
  - states of the accepting SCC
  - marks seen in the accepting SCC
  - cut-out acceptance condition
- Implementation:
  - At the deepest recursion level, in `scc_info`
  - Repeat what was done for the product accepting run search, what has to be cut out has been already, right?
  - We can use `bfs_steps` as is

The Generic Accepting Run Search

- Knowledge:
  - states of the accepting SCC
  - marks seen in the accepting SCC
  - cut-out acceptance condition
- Implementation:
  - At the deepest recursion level, in `scc_info`
  - Repeat what was done for the product accepting run search, what has to be cut out has been already, right?
  - We can use `bfs_steps` as is

We had assumed *Fin*-less

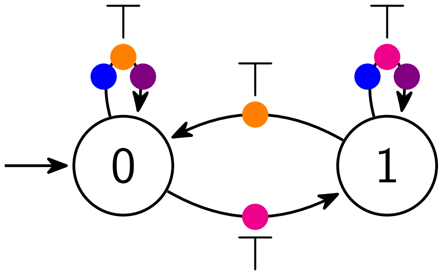
The Generic Accepting Run Search

- Knowledge:
  - states of the accepting SCC
  - marks seen in the accepting SCC
  - cut-out acceptance condition
- Implementation:
  - At the deepest recursion level, in `scc_info`
  - Repeat what was done for the product accepting run search, what has to be cut out has been already, right? ✘ Wrong!
  - We can use `bfs_steps` as is

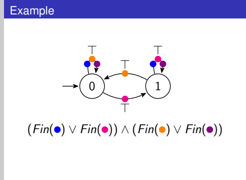
The Generic Accepting Run Search
- Knowledge:
  - states of the accepting SCC
  - marks seen in the accepting SCC
  - cut-out acceptance condition
- Implementation:
  - At the deepest recursion level, in scc_info
  - Repeat what was done for the product accepting run search, what has to be cut out has been already, right? ✘ Wrong!
  - We can use bfs_steps as is

We had assumed *Fin*-less

Example



$$(Fin({\color{blue}\bullet}) \vee Fin({\color{magenta}\bullet})) \wedge (Fin({\color{orange}\bullet}) \vee Fin({\color{purple}\bullet}))$$



Conclusion

1. Introduction
2. Generalities on accepting run searches
3. Product accepting run search
4. Generic accepting run search
5. Conclusion

2019-01-27

Counterexample searches in Spot
└Conclusion

└Conclusion

Introduction
Generalities on accepting run searches
Product accepting run search
Generic accepting run search
Conclusion

Conclusion

- Emptiness checks are useful in many ways
- Multiple ways to do them, with tradeoffs on efficiency
- Go together with accepting run searches, which completes their usage
- Different knowledge can be extracted from different algorithms

Conclusion

- Emptiness checks are useful in many ways
- Multiple ways to do them, with tradeoffs on efficiency
- Go together with accepting run searches, which completes their usage
- Different knowledge can be extracted from different algorithms

Conclusion

- Fix the generic accepting run search
  - Custom BFS? More versatile `bfs_steps`?
  - Bench
- Integrate the product emptiness check to Spot
- Better automata for benching

Conclusion

- Fix the generic accepting run search
  - Custom BFS? More versatile bfs_steps?
  - Bench
- Integrate the product emptiness check to Spot
- Better automata for benching

Questions?

Thank you for your attention!

Any imprecision? Would you like me to go into more details?

Questions?

Thank you for your attention!

Any imprecision? Would you like me to go into more details?

Bibliography I

- Jean-Michel Couvreur. "On-the-fly Verification of Temporal Logic". In: Proceedings of the World Congress on Formal Methods in the Development of Computing Systems (FM'99). Ed. by Jeannette M. Wing, Jim Woodcock, and Jim Davies. Vol. 1708. Lecture Notes in Computer Science. Toulouse, France: Springer-Verlag, Sept. 1999, pp. 253–271. ISBN: 3-540-66587-0.
- Alexandre Duret-Lutz et al. "Spot 2.0 — a framework for LTL and ω-automata manipulation". In: Proceedings of the 14th International Symposium on Automated Technology for Verification and Analysis (ATVA'16). Vol. 9938. Lecture Notes in Computer Science. Springer, Oct. 2016, pp. 122–129.

Bibliography I

Jean-Michel Couvreur. "On-the-fly Verification of Temporal Logic". In: Proceedings of the World Congress on Formal Methods in the Development of Computing Systems (FM'99). Ed. by Jeannette M. Wing, Jim Woodcock, and Jim Davies. Vol. 1708. Lecture Notes in Computer Science. Toulouse, France: Springer-Verlag, Sept. 1999, pp. 253–271. ISBN: 3-540-66587-0.

Alexandre Duret-Lutz et al. "Spot 2.0 — a framework for LTL and ω-automata manipulation". In: Proceedings of the 14th International Symposium on Automated Technology for Verification and Analysis (ATVA'16). Vol. 9938. Lecture Notes in Computer Science. Springer, Oct. 2016, pp. 122–129.

Bibliography II

- E. Allen Emerson and Chin-Laung Lei. “Modalities for Model Checking: Branching Time Logic Strikes Back”. In: Science of Computer Programming 8.3 (June 1987), pp. 275–306.
- Paul Gastin and Pierre Moro. “Minimal Counterexample Generation for SPIN”. en. In: Model Checking Software. Berlin, Heidelberg, 2007. DOI: 10.1007/978-3-540-73370-6_4.
- Paul Gastin, Pierre Moro, and Marc Zeitoun. “Minimization of counterexamples in SPIN”. In: Proceedings of the 11th International SPIN Workshop on Model Checking of Software (SPIN’04). Ed. by S. Graf and L. Mounier. Vol. 2989. Lecture Notes in Computer Science. Apr. 2004, pp. 92–108.

Bibliography II

- E. Allen Emerson and Chin-Laung Lei. “Modalities for Model Checking: Branching Time Logic Strikes Back”. In: Science of Computer Programming 8.3 (June 1987), pp. 275–306.
- Paul Gastin and Pierre Moro. “Minimal Counterexample Generation for SPIN”. en. In: Model Checking Software. Berlin, Heidelberg, 2007. DOI: 10.1007/978-3-540-73370-6_4.
- Paul Gastin, Pierre Moro, and Marc Zeitoun. “Minimization of counterexamples in SPIN”. In: Proceedings of the 11th International SPIN Workshop on Model Checking of Software (SPIN’04). Ed. by S. Graf and L. Mounier. Vol. 2989. Lecture Notes in Computer Science. Apr. 2004, pp. 92–108.

Bibliography III

- Clément Gillard. Two-automaton emptiness check in Spot. Tech. rep. 1706. EPITA Research and Development Laboratory (LRDE), 2017.
- Henri Hansen and Jaco Geldenhuys. "Cheap and Small Counterexamples". en. In: 2008 Sixth IEEE International Conference on Software Engineering and Formal Methods. Cape Town, South Africa, Nov. 2008. DOI: 10.1109/SEFM.2008.18.
- Henri Hansen and Antti Kervinen. "Minimal Counterexamples in $O(n \log n)$ Memory and $O(n^2)$ Time". In: Sixth International Conference on Application of Concurrency to System Design (ACSD'06). Turku, Finland, 2006. DOI: 10.1109/ACSD.2006.11.

Bibliography III
- Clément Gillard. Two-automaton emptiness check in Spot. Tech. rep. 1706. EPITA Research and Development Laboratory (LRDE), 2017.
- Henri Hansen and Jaco Geldenhuys. "Cheap and Small Counterexamples". en. In: 2008 Sixth IEEE International Conference on Software Engineering and Formal Methods. Cape Town, South Africa, Nov. 2008. DOI: 10.1109/SEFM.2008.18.
- Henri Hansen and Antti Kervinen. "Minimal Counterexamples in $O(n \log n)$ Memory and $O(n^2)$ Time". In: Sixth International Conference on Application of Concurrency to System Design (ACSD'06). Turku, Finland, 2006. DOI: 10.1109/ACSD.2006.11.