Algorithmic Analysis of a Model Checking Partial Order Reduction Scheme Using Alloy

Stephen F. Siegel

Verified Software Laboratory
Department of Computer and Information Sciences
University of Delaware
Newark, DE, USA

Subodh Sharma's Class
Anywhere
December 14, 2020

Overview

- background: model checking

Overview

- background: model checking
- on-the-fly ample set POR
  - Peled, CAV 1994; FMSD 1996
    - *Combining partial order reductions with on-the-fly model-checking*
  - Holzmann & Peled, FORTE 1994
    - *An improvement in formal verification*

Overview

- background: model checking
- on-the-fly ample set POR
  - Peled, CAV 1994; FMSD 1996
    - *Combining partial order reductions with on-the-fly model-checking*
  - Holzmann & Peled, FORTE 1994
    - *An improvement in formal verification*
- $\sim$2009: gap in proof

Overview

- background: model checking
- on-the-fly ample set POR
  - Peled, CAV 1994; FMSD 1996
    - *Combining partial order reductions with on-the-fly model-checking*
  - Holzmann & Peled, FORTE 1994
    - *An improvement in formal verification*
- ∼2009: gap in proof

  - fix the proof? find a counterexample?

Overview

- background: model checking
- on-the-fly ample set POR
  - Peled, CAV 1994; FMSD 1996
    - *Combining partial order reductions with on-the-fly model-checking*
  - Holzmann & Peled, FORTE 1994
    - *An improvement in formal verification*
- ∼2009: gap in proof
  - fix the proof? find a counterexample?
- heard talk by Pamela Zave on using Alloy to find defect in Chord

Overview

- background: model checking
- on-the-fly ample set POR
  - Peled, CAV 1994; FMSD 1996
    - *Combining partial order reductions with on-the-fly model-checking*
  - Holzmann & Peled, FORTE 1994
    - *An improvement in formal verification*
- ∼2009: gap in proof

  - fix the proof? find a counterexample?
- heard talk by Pamela Zave on using Alloy to find defect in Chord
- used Alloy to model on-the-fly ample set POR

Overview

- background: model checking
- on-the-fly ample set POR
  - Peled, CAV 1994; FMSD 1996
    - *Combining partial order reductions with on-the-fly model-checking*
  - Holzmann & Peled, FORTE 1994
    - *An improvement in formal verification*
- ∼2009: gap in proof

  - fix the proof? find a counterexample?
- heard talk by Pamela Zave on using Alloy to find defect in Chord
- used Alloy to model on-the-fly ample set POR
- counterexample! (in theory and in Spin)

Overview

- background: model checking
- on-the-fly ample set POR
  - Peled, CAV 1994; FMSD 1996
    - *Combining partial order reductions with on-the-fly model-checking*
  - Holzmann & Peled, FORTE 1994
    - *An improvement in formal verification*
- ∼2009: gap in proof

  - fix the proof? find a counterexample?
- heard talk by Pamela Zave on using Alloy to find defect in Chord
- used Alloy to model on-the-fly ample set POR
- counterexample! (in theory and in Spin)
- a fix

Background: Model Checking

Background: Model Checking

- an automated approach to the verification of software and hardware systems
  - integrated circuits
  - network protocols
  - software: programs written in C, Java, machine language, ...

Background: Model Checking

- an automated approach to the verification of software and hardware systems
  - integrated circuits
  - network protocols
  - software: programs written in C, Java, machine language, ...
- particularly effective for concurrent systems
  - multithreaded programs, ...

Background: Model Checking

- an automated approach to the verification of software and hardware systems
  - integrated circuits
  - network protocols
  - software: programs written in C, Java, machine language, ...
- particularly effective for concurrent systems
  - multithreaded programs, ...
- many robust, widely-used model checking tools
  - SPIN — has its own input language, PROMELA, highly optimized
  - Divine (C++), CIVL (C+...), NuSMV, Java PathFinder (Java), ...

Using a Model Checker: 3 Tasks

Using a Model Checker: 3 Tasks

1. represent the system/program as a finite-state transition system

Using a Model Checker: 3 Tasks

1. represent the system/program as a finite-state transition system
2. specify correctness properties you wish to check
   - example "at most one thread is in the critical section at any time"
   - temporal logic, automata, ...

Using a Model Checker: 3 Tasks

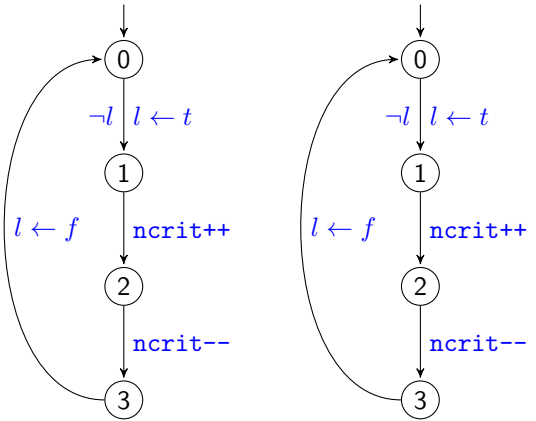
1. represent the system/program as a finite-state transition system
2. specify correctness properties you wish to check
   - example "at most one thread is in the critical section at any time"
   - temporal logic, automata, ...
3. apply automated algorithmic (graph, automata, ...) techniques to check that all executions of the program satisfy the properties
   - typically: explore the reachable states of the program
   - simultaneously updating a property automaton
   - report a violation as soon as an accepting cycle is detected

Example: mutual exclusion: Promela program

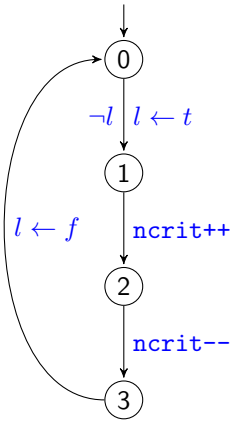
```
bool lock = false;
int ncrit = 0;

active [2] proctype thread() {
  do
  :: atomic { !lock -> lock = true }
     ncrit++;
     // ...this is the critical section...
     ncrit--;
     lock = false
  od
}
```

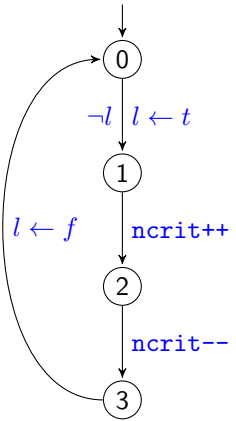
Example: mutual exclusion: state-transition system



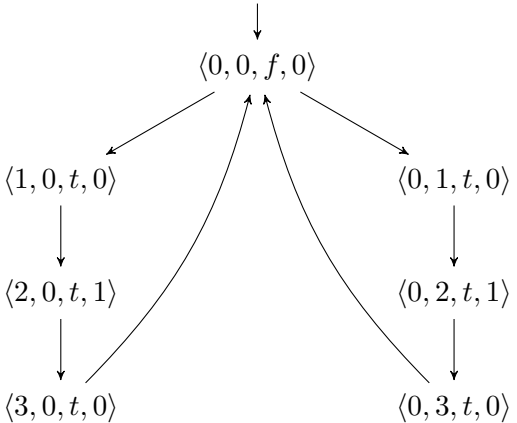
Example: mutual exclusion: state-transition system



Thread 0



Thread 1



States: $\langle \mathsf{pc}_0, \mathsf{pc}_1, l, \mathtt{ncrit} \rangle$

Example: mutual exclusion: property specification

Example: mutual exclusion: property specification

- an execution specifies a sequence of states
- a property specifies which state sequences are OK

Example: mutual exclusion: property specification

- an execution specifies a sequence of states
- a property specifies which state sequences are OK
- properties may be specified in natural language, using LTL formulas, or automata

Example: mutual exclusion: property specification

- an execution specifies a sequence of states
- a property specifies which state sequences are OK
- properties may be specified in natural language, using LTL formulas, or automata
- natural language: "`ncrit` is always at most 1"

Example: mutual exclusion: property specification

- an execution specifies a sequence of states
- a property specifies which state sequences are OK
- properties may be specified in natural language, using LTL formulas, or automata
- natural language: "`ncrit` is always at most 1"
- linear temporal logic (LTL): $\mathbf{G}(\texttt{ncrit} \leq 1)$

Example: mutual exclusion: property specification

- an execution specifies a sequence of states
- a property specifies which state sequences are OK
- properties may be specified in natural language, using LTL formulas, or automata
- natural language: "`ncrit` is always at most 1"
- linear temporal logic (LTL): $\mathbf{G}(\texttt{ncrit} \leq 1)$
  - $\mathbf{G}\ p =$ "henceforth p" ("from now on", "always")
  - $\mathbf{F}\ p =$ "eventually p"
  - $p\ \mathbf{U}\ q =$ "p until q"

Example: mutual exclusion: property specification

- an execution specifies a sequence of states
- a property specifies which state sequences are OK
- properties may be specified in natural language, using LTL formulas, or automata
- natural language: "`ncrit` is always at most 1"
- linear temporal logic (LTL): $\mathbf{G}(\texttt{ncrit} \leq 1)$
  - $\mathbf{G}\ p =$ "henceforth p" ("from now on", "always")
  - $\mathbf{F}\ p =$ "eventually p"
  - $p\ \mathbf{U}\ q =$ "p until q"
- Büchi automaton (negated) ...

Büchi automata

A Büchi automaton is a tuple

$$\mathcal{B} = \langle S, \Delta, \Sigma, \delta, F \rangle,$$

where

1. S is a finite set of *automaton states*
2. $\Delta \subseteq S$ is the set of *initial states*
3. Σ is a finite set called the *alphabet*
4. $\delta \subseteq S \times \Sigma \times S$ is the *transition relation*
5. $F \subseteq S$ is the set of *accepting states*.

Büchi automata

A Büchi automaton is a tuple

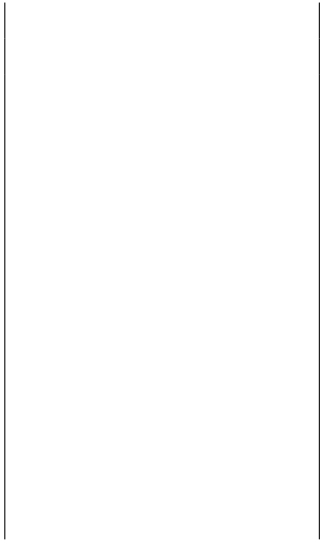
$$\mathcal{B} = \langle S, \Delta, \Sigma, \delta, F \rangle,$$

where

1. S is a finite set of *automaton states*
2. $\Delta \subseteq S$ is the set of *initial states*
3. Σ is a finite set called the *alphabet*
4. $\delta \subseteq S \times \Sigma \times S$ is the *transition relation*
5. $F \subseteq S$ is the set of *accepting states*.

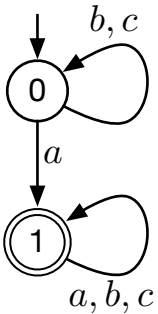
The language of $\mathcal{B}$, denoted $\mathcal{L}(B)$, is the set of all $\xi \in \Sigma^{\omega}$ generated by infinite paths in $\mathcal{B}$ that pass through an accepting state infinitely often.

Examples of Büchi automata, $\Sigma = \{a, b, c\}$



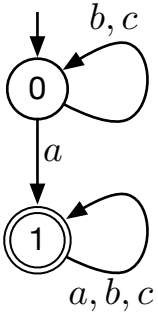
Examples of Büchi automata, $\Sigma = \{a, b, c\}$

"eventually a"

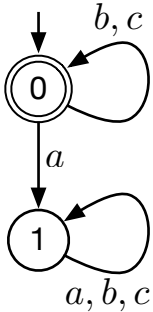


Examples of Büchi automata, $\Sigma = \{a, b, c\}$

"eventually a"

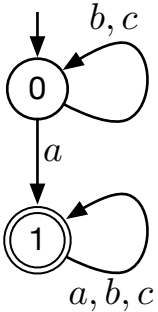


"never a"

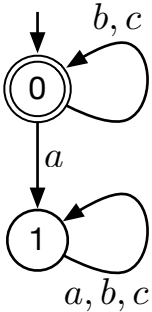


Examples of Büchi automata, $\Sigma = \{a, b, c\}$

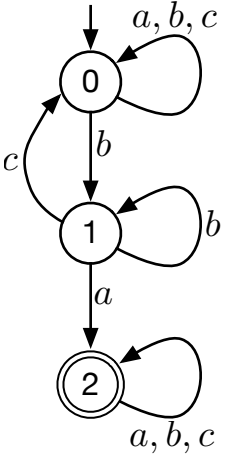
"eventually a"



"never a"



"after some b, a occurs before c"



Using Büchi automata to specify program properties

Using Büchi automata to specify program properties

- first, identify the set AP of atomic propositions
  - an atomic proposition is either true or false in each state
  - example: $p = (\texttt{ncrit} \leq 1)$

Using Büchi automata to specify program properties

- first, identify the set AP of atomic propositions
  - an atomic proposition is either true or false in each state
  - example: $p = (\texttt{ncrit} \leq 1)$
- the alphabet Σ of the Büchi Automaton is 2^{AP}
  - an element of Σ is a set of atomic propositions
  - interpretation: the set of propositions which are true in that state

Using Büchi automata to specify program properties

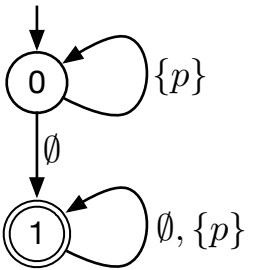
- first, identify the set AP of atomic propositions
  - an atomic proposition is either true or false in each state
  - example: $p = (\texttt{ncrit} \leq 1)$
- the alphabet Σ of the Büchi Automaton is 2^{AP}
  - an element of Σ is a set of atomic propositions
  - interpretation: the set of propositions which are true in that state
- example: $\mathsf{AP} = \{p\}$
  - $\Sigma = \{\{p\}, \emptyset\}$
  - $\{p\}$: p is true
  - $\emptyset$: p is false

Using Büchi automata to specify program properties

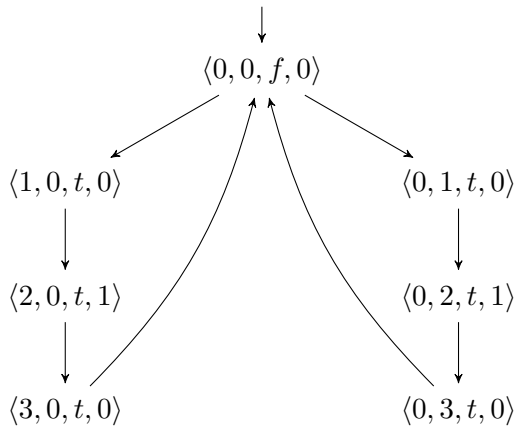
- first, identify the set AP of atomic propositions
  - an atomic proposition is either true or false in each state
  - example: $p = (\texttt{ncrit} \leq 1)$
- the alphabet Σ of the Büchi Automaton is 2^{AP}
  - an element of Σ is a set of atomic propositions
  - interpretation: the set of propositions which are true in that state
- example: $\mathsf{AP} = \{p\}$
  - $\Sigma = \{\{p\}, \emptyset\}$
  - $\{p\}$: p is true
  - $\emptyset$: p is false
- each execution yields a trace, which is an infinite word in Σ

Using Büchi automata to specify program properties

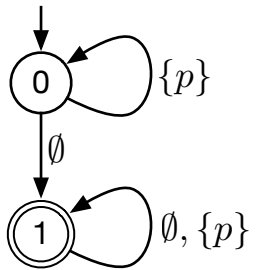
- first, identify the set AP of atomic propositions
  - an atomic proposition is either true or false in each state
  - example: $p = (\texttt{ncrit} \leq 1)$
- the alphabet Σ of the Büchi Automaton is 2^{AP}
  - an element of Σ is a set of atomic propositions
  - interpretation: the set of propositions which are true in that state
- example: AP $= \{p\}$
  - $\Sigma = \{\{p\}, \emptyset\}$
  - $\{p\}$: p is true
  - $\emptyset$: p is false
- each execution yields a trace, which is an infinite word in Σ
- the BA specifies which infinite words violate the property
- example: a trace violating "always $\texttt{ncrit} \leq 1$":



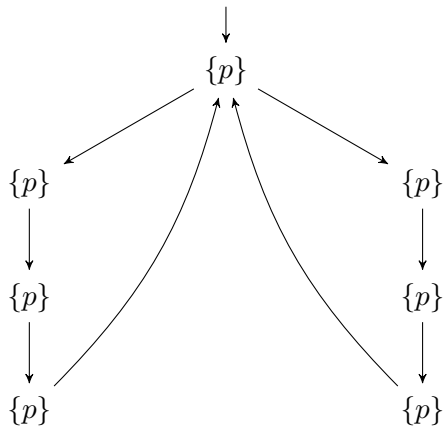
Automated technique: compute the sycnchronous product

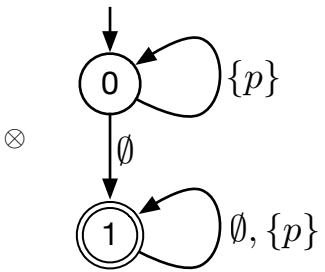


$\otimes$

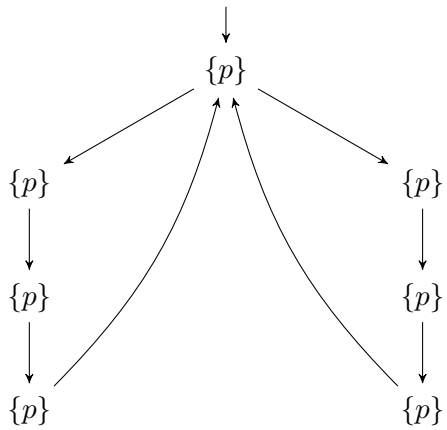


Automated technique: compute the sycnchronous product

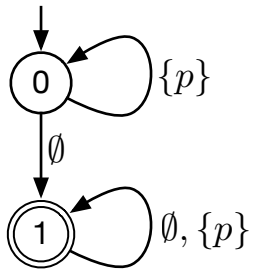




Automated technique: compute the sycnchronous product



$\otimes$



- in the synchronous product, no accepting cycle is reached – try it in SPIN!

Background: Partial Order Reduction

Background: Partial Order Reduction

- POR is a model checking optimization
  - POR can dramatically reduce the number of states that need to be explored in order to verify certain properties
  - the yes/no result should always be the same whether or not POR is used

Background: Partial Order Reduction

- POR is a model checking optimization
  - POR can dramatically reduce the number of states that need to be explored in order to verify certain properties
  - the yes/no result should always be the same whether or not POR is used
- POR can only be applied when the property is stutter invariant
  - let $\mathcal{L} \subset \Sigma^\omega$ (a set of infinite words over Σ)
  - $\mathcal{L}$ is stutter invariant if for all $a_1, a_2, \ldots \in \Sigma$, $i_1, i_2, \ldots \geq 1$:

$$a_1 a_2 \cdots \in \mathcal{L} \;\Leftrightarrow\; a_1^{i_1} a_2^{i_2} \cdots \in \mathcal{L}$$

Background: Partial Order Reduction

- POR is a model checking optimization
  - POR can dramatically reduce the number of states that need to be explored in order to verify certain properties
  - the yes/no result should always be the same whether or not POR is used
- POR can only be applied when the property is stutter invariant
  - let $\mathcal{L} \subset \Sigma^\omega$ (a set of infinite words over Σ)
  - $\mathcal{L}$ is stutter invariant if for all $a_1, a_2, \ldots \in \Sigma$, $i_1, i_2, \ldots \geq 1$:

$$a_1 a_2 \cdots \in \mathcal{L} \;\Leftrightarrow\; a_1^{i_1} a_2^{i_2} \cdots \in \mathcal{L}$$

- hence the property is invariant under repeating an entry (stuttering)
- and invariant under removing stutters of an entry (but leave at least one occurrence)

Background: Partial Order Reduction

- POR is a model checking optimization
  - POR can dramatically reduce the number of states that need to be explored in order to verify certain properties
  - the yes/no result should always be the same whether or not POR is used
- POR can only be applied when the property is stutter invariant
  - let $\mathcal{L} \subset \Sigma^\omega$ (a set of infinite words over Σ)
  - $\mathcal{L}$ is stutter invariant if for all $a_1, a_2, \ldots \in \Sigma$, $i_1, i_2, \ldots \geq 1$:

$$a_1 a_2 \cdots \in \mathcal{L} \;\Leftrightarrow\; a_1^{i_1} a_2^{i_2} \cdots \in \mathcal{L}$$

- hence the property is invariant under repeating an entry (stuttering)
- and invariant under removing stutters of an entry (but leave at least one occurrence)
- all properties we have seen so far are stutter invariant

Background: Partial Order Reduction

- POR is a model checking optimization
  - POR can dramatically reduce the number of states that need to be explored in order to verify certain properties
  - the yes/no result should always be the same whether or not POR is used
- POR can only be applied when the property is stutter invariant
  - let $\mathcal{L} \subset \Sigma^\omega$ (a set of infinite words over Σ)
  - $\mathcal{L}$ is stutter invariant if for all $a_1, a_2, \ldots \in \Sigma$, $i_1, i_2, \ldots \geq 1$:

$$a_1 a_2 \cdots \in \mathcal{L} \;\Leftrightarrow\; a_1^{i_1} a_2^{i_2} \cdots \in \mathcal{L}$$

- hence the property is invariant under repeating an entry (stuttering)
- and invariant under removing stutters of an entry (but leave at least one occurrence)
- all properties we have seen so far are stutter invariant
- all properties expressible with LTL operators **F**, **G**, **U** are stutter invariant

Background: Partial Order Reduction

- POR is a model checking optimization
  - POR can dramatically reduce the number of states that need to be explored in order to verify certain properties
  - the yes/no result should always be the same whether or not POR is used
- POR can only be applied when the property is stutter invariant
  - let $\mathcal{L} \subset \Sigma^\omega$ (a set of infinite words over Σ)
  - $\mathcal{L}$ is stutter invariant if for all $a_1, a_2, \ldots \in \Sigma$, $i_1, i_2, \ldots \geq 1$:

$$a_1 a_2 \cdots \in \mathcal{L} \;\Leftrightarrow\; a_1^{i_1} a_2^{i_2} \cdots \in \mathcal{L}$$

- hence the property is invariant under repeating an entry (stuttering)
- and invariant under removing stutters of an entry (but leave at least one occurrence)
- all properties we have seen so far are stutter invariant
- all properties expressible with LTL operators **F**, **G**, **U** are stutter invariant
- almost any property you would ever want to express is stutter invariant

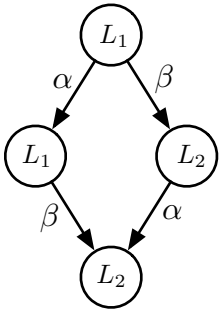
Partial Order Reduction: intuition

Partial Order Reduction: intuition

- in a parallel program, an action α on local variables commutes with all actions β from other threads
  - the order of the two actions doesn't matter
  - you end up in the same state either way
  - if the local action α can not change the value of any atomic proposition...
  - ...you only need to explore one of the two orders!

Partial Order Reduction: intuition

- in a parallel program, an action α on local variables commutes with all actions β from other threads
  - the order of the two actions doesn't matter
  - you end up in the same state either way
  - if the local action α can not change the value of any atomic proposition...
  - ...you only need to explore one of the two orders!



- one path yields trace $\cdots L_1 L_1 L_2 \cdots$
- other path yields trace $\cdots L_1 L_2 L_2 \cdots$
- stutter equivalent!
- either both paths satisfy the property or both violate

Definitions: Programs

Definitions: Programs

A program consists of

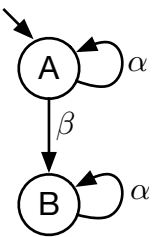
- a set Q of states
- an initial state $\iota \in Q$
- a set T of operations
  - each $\alpha \in T$ is a (deterministic) function α: $\mathrm{en}_\alpha \to Q$, for some $\mathrm{en}_\alpha \subseteq Q$

Definitions: Programs

A program consists of

- a set Q of states
- an initial state $\iota \in Q$
- a set T of operations
  - each $\alpha \in T$ is a (deterministic) function $\alpha\colon \mathsf{en}_\alpha \to Q$, for some $\mathsf{en}_\alpha \subseteq Q$

Example:



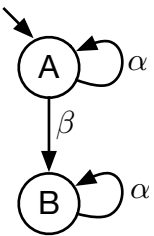
$$Q = \{A, B\}$$
$$\iota = A$$
$$T = \{\alpha, \beta\}$$
$$\mathsf{en}_\alpha = \{A, B\}$$
$$\mathsf{en}_\beta = \{A\}$$

Definitions: Programs

A program consists of

- a set Q of states
- an initial state $\iota \in Q$
- a set T of operations
  - each $\alpha \in T$ is a (deterministic) function $\alpha\colon \mathsf{en}_\alpha \to Q$, for some $\mathsf{en}_\alpha \subseteq Q$

Example:



$$
\begin{aligned}
Q &= \{A, B\} \\
\iota &= A \\
T &= \{\alpha, \beta\} \\
\mathsf{en}_\alpha &= \{A, B\} \\
\mathsf{en}_\beta &= \{A\}
\end{aligned}
$$

- for $q \in Q$, define $\mathsf{en}(q) = \{\alpha \in T \mid q \in \mathsf{en}_\alpha\}$

Program Traces

- let AP be a set of atomic propositions
- an interpretation is a function $L\colon Q \to \Sigma = 2^{\mathsf{AP}}$

Program Traces

- let AP be a set of atomic propositions
- an interpretation is a function $L\colon Q \to \Sigma = 2^{\mathsf{AP}}$
- $\mathcal{L}(P) \subseteq \Sigma^{\omega}$ is the set of infinite traces

$$L(q_0)L(q_1)\cdots$$

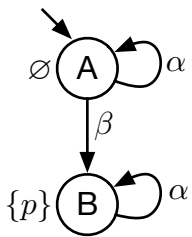
where $\iota = q_0 \to q_1 \to \cdots$ is an infinite initial path in P

Program Traces

- let AP be a set of atomic propositions
- an interpretation is a function $L\colon Q \to \Sigma = 2^{\mathsf{AP}}$
- $\mathcal{L}(P) \subseteq \Sigma^{\omega}$ is the set of infinite traces

$$L(q_0)L(q_1)\cdots$$

where $\iota = q_0 \to q_1 \to \cdots$ is an infinite initial path in P



$$\mathsf{AP} = \{p\}$$
$$L(A) = \varnothing$$
$$L(B) = \{p\}$$
$$\varnothing\varnothing\varnothing\{p\}^{\omega} \in \mathcal{L}(P)$$

Properties

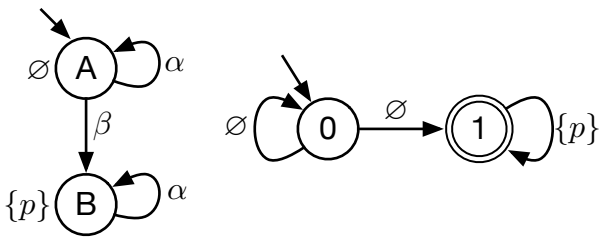
- a property is a subset of Σ^{ω}

Properties

- a property is a subset of Σ^ω
- a Büchi automaton $\mathcal{B}$ with alphabet Σ specifies the property $\Sigma^\omega \setminus \mathcal{L}(\mathcal{B})$
  - the program violates the property if $\mathcal{L}(P) \cap \mathcal{L}(\mathcal{B}) \neq \varnothing$

Properties

- a property is a subset of Σ^ω
- a Büchi automaton $\mathcal{B}$ with alphabet Σ specifies the property $\Sigma^\omega \setminus \mathcal{L}(\mathcal{B})$
  - the program violates the property if $\mathcal{L}(P) \cap \mathcal{L}(\mathcal{B}) \neq \varnothing$



- violation: $\varnothing\varnothing\varnothing\{p\}^\omega \in \mathcal{L}(P) \cap \mathcal{L}(\mathcal{B})$

Product automaton

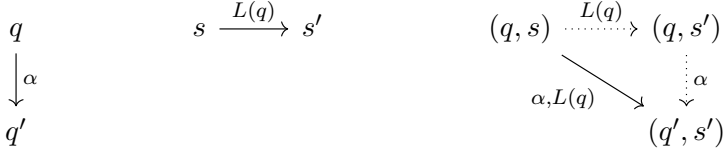
Given program $P = (Q, T, \iota)$ and Büchi automaton $\mathcal{B}$ with states S:
The product automaton $P \otimes \mathcal{B}$ is the Büchi automaton

- alphabet: $T \times \Sigma$, states: $Q \times S$, (q, s) accepting $\Leftrightarrow$ s accepting

Product automaton

Given program $P = (Q, T, \iota)$ and Büchi automaton $\mathcal{B}$ with states S:
The product automaton $P \otimes \mathcal{B}$ is the Büchi automaton

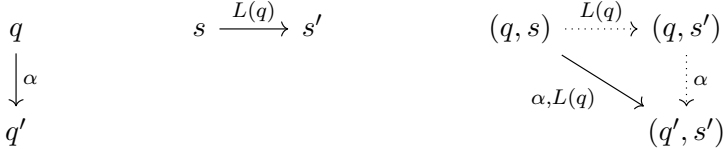
- alphabet: $T \times \Sigma$, states: $Q \times S$, (q, s) accepting $\Leftrightarrow$ s accepting
- transitions:



Product automaton

Given program $P = (Q, T, \iota)$ and Büchi automaton $\mathcal{B}$ with states S:
The product automaton $P \otimes \mathcal{B}$ is the Büchi automaton

- alphabet: $T \times \Sigma$, states: $Q \times S$, (q, s) accepting $\Leftrightarrow$ s accepting
- transitions:

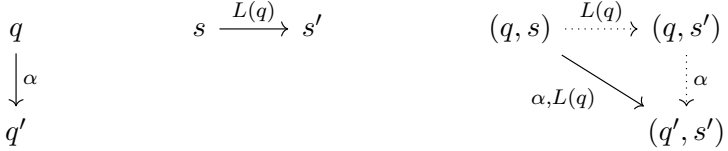


$$\mathcal{L}(P \otimes \mathcal{B}) = \mathcal{L}(P) \cap \mathcal{L}(\mathcal{B})$$

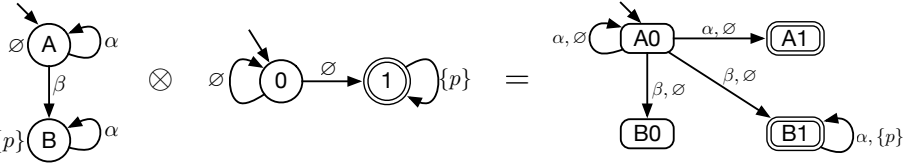
Product automaton

Given program $P = (Q, T, \iota)$ and Büchi automaton $\mathcal{B}$ with states S:
The product automaton $P \otimes \mathcal{B}$ is the Büchi automaton

- alphabet: $T \times \Sigma$, states: $Q \times S$, (q, s) accepting $\Leftrightarrow$ s accepting
- transitions:



$$\mathcal{L}(P \otimes \mathcal{B}) = \mathcal{L}(P) \cap \mathcal{L}(\mathcal{B})$$



Partial Order Reduction

- $\mathcal{A} = P \otimes \mathcal{B}$
- explore a sub-automaton $\mathcal{A}' \subseteq \mathcal{A}$ such that $\mathcal{L}(\mathcal{A}') = \varnothing \Leftrightarrow \mathcal{L}(A) = \varnothing$

Partial Order Reduction

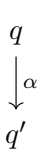
- $\mathcal{A} = P \otimes \mathcal{B}$
- explore a sub-automaton $\mathcal{A}' \subseteq \mathcal{A}$ such that $\mathcal{L}(\mathcal{A}') = \varnothing \Leftrightarrow \mathcal{L}(A) = \varnothing$
- "offline" POR: $\mathcal{A}' = P' \otimes \mathcal{B}$. "on-the-fly" POR: not necessarily

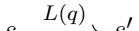
Partial Order Reduction

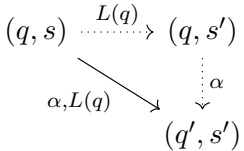
- $\mathcal{A} = P \otimes \mathcal{B}$
- explore a sub-automaton $\mathcal{A}' \subseteq \mathcal{A}$ such that $\mathcal{L}(\mathcal{A}') = \varnothing \Leftrightarrow \mathcal{L}(A) = \varnothing$
- "offline" POR: $\mathcal{A}' = P' \otimes \mathcal{B}$. "on-the-fly" POR: not necessarily
- on-the-fly ample set POR
  - given $(q, s) \in Q \times S$, specify $\mathsf{amp}(q, s) \subseteq \mathsf{en}(q)$

Partial Order Reduction

- $\mathcal{A} = P \otimes \mathcal{B}$
- explore a sub-automaton $\mathcal{A}' \subseteq \mathcal{A}$ such that $\mathcal{L}(\mathcal{A}') = \varnothing \Leftrightarrow \mathcal{L}(A) = \varnothing$
- "offline" POR: $\mathcal{A}' = P' \otimes \mathcal{B}$. "on-the-fly" POR: not necessarily
- on-the-fly ample set POR
  - given $(q, s) \in Q \times S$, specify $\mathsf{amp}(q, s) \subseteq \mathsf{en}(q)$
  - define $\mathcal{A}'$:







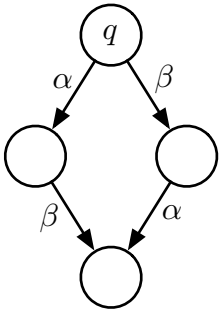
$$\alpha \in \mathsf{amp}(q, s')$$

Dependence and visibility

Definition

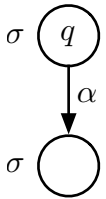
Two operations $\alpha, \beta \in T$ are independent if for all $q \in \mathsf{en}_\alpha \cap \mathsf{en}_\beta$:

1. $\alpha(q) \in \mathsf{en}_\beta$
2. $\beta(q) \in \mathsf{en}_\alpha$ and
3. $\alpha(\beta(q)) = \beta(\alpha(q))$.



Definition

An operation $\alpha \in T$ is invisible if, for all $q \in \mathsf{en}_\alpha$: $L(q) = L(\alpha(q))$.



Conditions on the ample sets

Conditions on the ample sets

C0 For all $q \in Q$, $s \in S$: $\mathsf{en}(q) \neq \emptyset \implies \mathsf{amp}(q, s) \neq \emptyset$

Conditions on the ample sets

C0 For all $q \in Q$, $s \in S$: $\mathsf{en}(q) \neq \emptyset \implies \mathsf{amp}(q, s) \neq \emptyset$

C1 For all $q \in Q$, $s \in S$, and paths in P starting from q:

- no operation dependent on an operation in $\mathsf{amp}(q, s)$ can occur before some operation in $\mathsf{amp}(q, s)$ occurs

Conditions on the ample sets

C0 For all $q \in Q$, $s \in S$: $\mathsf{en}(q) \neq \emptyset \implies \mathsf{amp}(q, s) \neq \emptyset$

C1 For all $q \in Q$, $s \in S$, and paths in P starting from q:

- no operation dependent on an operation in $\mathsf{amp}(q, s)$ can occur before some operation in $\mathsf{amp}(q, s)$ occurs

C2 For all $q \in Q$, $s \in S$:

- if $\mathsf{amp}(q, s) \neq \mathsf{en}(q)$ then $\forall \alpha \in \mathsf{amp}(q, s)$, α is invisible

Conditions on the ample sets

C0 For all $q \in Q$, $s \in S$: $\mathsf{en}(q) \neq \emptyset \implies \mathsf{amp}(q, s) \neq \emptyset$

C1 For all $q \in Q$, $s \in S$, and paths in P starting from q:

- no operation dependent on an operation in $\mathsf{amp}(q, s)$ can occur before some operation in $\mathsf{amp}(q, s)$ occurs

C2 For all $q \in Q$, $s \in S$:

- if $\mathsf{amp}(q, s) \neq \mathsf{en}(q)$ then $\forall \alpha \in \mathsf{amp}(q, s)$, α is invisible

C3 [weaker version] In any cycle in $\mathcal{A}'$:

- there is a transition from $\langle q, s \rangle$ to $\langle q', s' \rangle$ for which $\mathsf{amp}(q, s') = \mathsf{en}(q)$

Conditions on the ample sets

C0 For all $q \in Q$, $s \in S$: $\mathsf{en}(q) \neq \emptyset \implies \mathsf{amp}(q, s) \neq \emptyset$

C1 For all $q \in Q$, $s \in S$, and paths in P starting from q:

- no operation dependent on an operation in $\mathsf{amp}(q, s)$ can occur before some operation in $\mathsf{amp}(q, s)$ occurs

C2 For all $q \in Q$, $s \in S$:

- if $\mathsf{amp}(q, s) \neq \mathsf{en}(q)$ then $\forall \alpha \in \mathsf{amp}(q, s)$, α is invisible

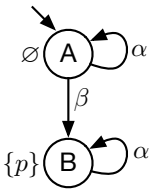
C3 [weaker version] In any cycle in $\mathcal{A}'$:

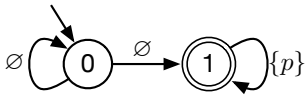
- there is a transition from $\langle q, s \rangle$ to $\langle q', s' \rangle$ for which $\mathsf{amp}(q, s') = \mathsf{en}(q)$

Combined Theorem:

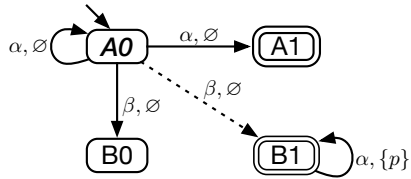
If $\mathcal{L}(\mathcal{B})$ is stutter-invariant and **C0**–**C3** hold, then $\mathcal{L}(A') = \varnothing \Leftrightarrow \mathcal{L}(A) = \varnothing$.

Counterexample to combined theorem





amp	0	1
A	$\{\alpha, \beta\}$	$\{\alpha\}$
B	$\{\alpha\}$	$\{\alpha\}$



C0 For all $q \in Q$, $s \in S$: $\text{en}(q) \neq \emptyset \implies \text{amp}(q, s) \neq \emptyset$

C1 For all $q \in Q$, $s \in S$, and paths in P starting from q:

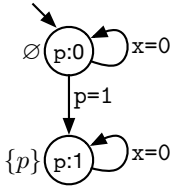
- no operation dependent on an operation in $\text{amp}(q, s)$ can occur before some operation in $\text{amp}(q, s)$ occurs

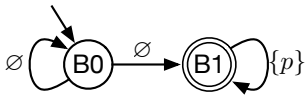
C2 For all $q \in Q$, $s \in S$: if $\text{amp}(q, s) \neq \text{en}(q)$ then $\forall \alpha \in \text{amp}(q, s)$, α is invisible

C3 In any cycle in $\mathcal{A}'$: there is a transition from $\langle q, s \rangle$ to $\langle q', s' \rangle$ for which $\text{amp}(q, s') = \text{en}(q)$

Spin

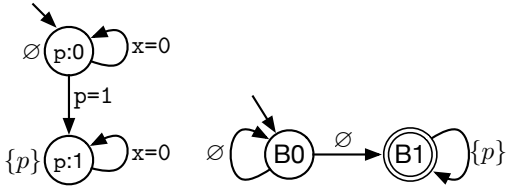
```
bit p = 0;
active proctype p0() { p=1 }
active proctype p1() { bit x=0; do :: x=0 od }
never {
  B0: do :: !p :: !p -> break od
  accept_B1: do :: p od
}
```





Spin

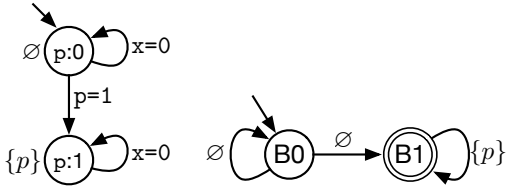
```
bit p = 0;
active proctype p0() { p=1 }
active proctype p1() { bit x=0; do :: x=0 od }
never {
  B0: do :: !p :: !p -> break od
  accept_B1: do :: p od
}
```




```
$ spin -a test1.pml; cc -o pan -DNOREDUCE pan.c; ./pan -a -n
pan:1: acceptance cycle (at depth 2)
...
```

Spin

```
bit p = 0;
active proctype p0() { p=1 }
active proctype p1() { bit x=0; do :: x=0 od }
never {
  B0: do :: !p :: !p -> break od
  accept_B1: do :: p od
}
```



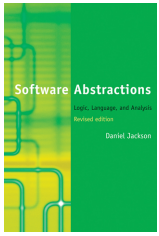

```
$ spin -a test1.pml; cc -o pan -DNOREDUCE pan.c; ./pan -a -n
pan:1: acceptance cycle (at depth 2)
...
```

```
$ spin -a test1.pml; cc -o pan pan.c; ./pan -a -n
warning: for p.o. reduction to be valid the never claim must be stutter-invariant
(never claims generated from LTL formulae are stutter-invariant)
+ Partial Order Reduction
State-vector 36 byte, depth reached 2, errors: 0
        3 states, stored
        2 states, matched
        5 transitions (= stored+matched)
```

Finding Counterexamples to POR Claims

Alloy

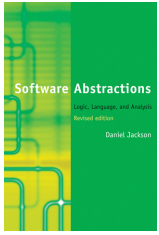
- specification language based on relational/first order logic
- a model specifies "signatures" (sets), relations, constraints



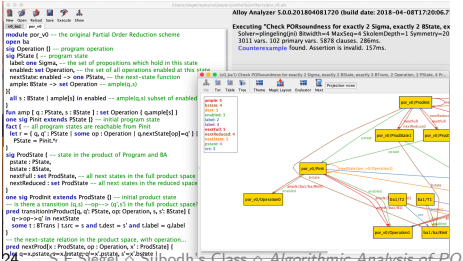
Finding Counterexamples to POR Claims

Alloy

- specification language based on relational/first order logic
- a model specifies "signatures" (sets), relations, constraints



The Alloy Analyzer



given

- an Alloy model
- an assertion
- an upper bound on the size of each signature

produces either

- "assertion may hold"
  - it holds within the bounds, or
- "assertion is violated"

Finding Counterexamples to POR Claims, cont.

- constructed Alloy model of on-the-fly ample-set POR
- 71 lines of Alloy code (reproduced in paper in entirety)

Finding Counterexamples to POR Claims, cont.

- constructed Alloy model of on-the-fly ample-set POR
- 71 lines of Alloy code (reproduced in paper in entirety)
- modeled in complete generality
  - program, operation, independence, invisiblity, Büchi automaton, ample set, product automaton, language emptiness
- asserted $\mathcal{L}(A') = \varnothing \Leftrightarrow \mathcal{L}(A) = \varnothing$

Finding Counterexamples to POR Claims, cont.

- constructed Alloy model of on-the-fly ample-set POR
- 71 lines of Alloy code (reproduced in paper in entirety)
- modeled in complete generality
  - program, operation, independence, invisiblity, Büchi automaton, ample set, product automaton, language emptiness
- asserted $\mathcal{L}(A') = \varnothing \Leftrightarrow \mathcal{L}(A) = \varnothing$
- could not find a way to model stutter-invariance in Alloy
  - resorted to specifying concrete Buchi automata for most experiments

Finding Counterexamples to POR Claims, cont.

- constructed Alloy model of on-the-fly ample-set POR
- 71 lines of Alloy code (reproduced in paper in entirety)
- modeled in complete generality
  - program, operation, independence, invisiblity, Büchi automaton, ample set, product automaton, language emptiness
- asserted $\mathcal{L}(A') = \varnothing \Leftrightarrow \mathcal{L}(A) = \varnothing$
- could not find a way to model stutter-invariance in Alloy
  - resorted to specifying concrete Buchi automata for most experiments
- counterexample shown earlier
  - I specified the Büchi automaton
  - Alloy found the (minimal) program and ample sets

Finding Counterexamples to POR Claims, cont.

- constructed Alloy model of on-the-fly ample-set POR
- 71 lines of Alloy code (reproduced in paper in entirety)
- modeled in complete generality
  - program, operation, independence, invisiblity, Büchi automaton, ample set, product automaton, language emptiness
- asserted $\mathcal{L}(A') = \varnothing \Leftrightarrow \mathcal{L}(A) = \varnothing$
- could not find a way to model stutter-invariance in Alloy
  - resorted to specifying concrete Buchi automata for most experiments
- counterexample shown earlier
  - I specified the Büchi automaton
  - Alloy found the (minimal) program and ample sets

```
fact C1 {
  all q: PState, s: BState | let A=amp[q,s] |
    let r = { q1, q2: PState | some op: Operation-A |
              q1->op->q2 in nextState } |
      all q': q.*r, op1: q'.enabled-A, op2: A | independent[op1, op2]
}
```

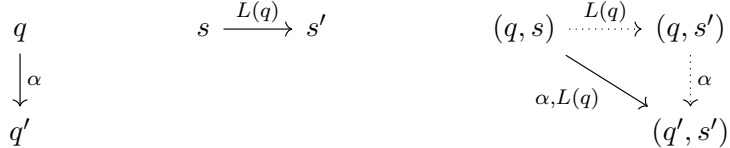
Variations on the Combined Theorem

- make the ample set a function of the source product state
  - not of the intermediate state

Variations on the Combined Theorem

- make the ample set a function of the source product state
  - not of the intermediate state

Define $\mathcal{A}'$:

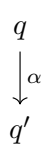


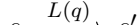
$$\alpha \in \mathsf{amp}(q, s)$$

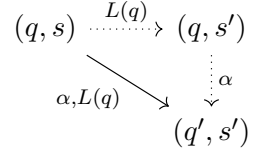
Variations on the Combined Theorem

- make the ample set a function of the source product state
  - not of the intermediate state

Define $\mathcal{A}'$:







$$\alpha \in \mathsf{amp}(q, s)$$

C0–C2: (same)

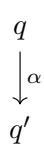
C3: In any cycle in $\mathcal{A}'$:

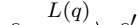
- there is a state $\langle q, s \rangle$ for which $\mathsf{amp}(q, s) = \mathsf{en}(q)$

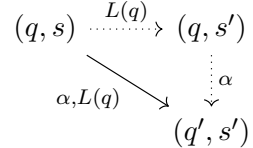
Variations on the Combined Theorem

- make the ample set a function of the source product state
  - not of the intermediate state

Define $\mathcal{A}'$:







$$\alpha \in \mathsf{amp}(q, s)$$

C0–C2: (same)

C3: In any cycle in $\mathcal{A}'$:

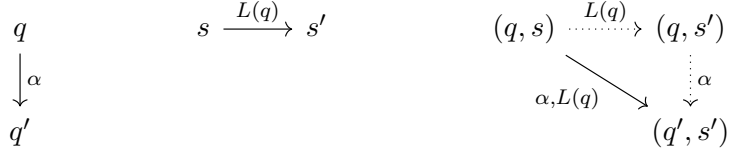
- there is a state $\langle q, s \rangle$ for which $\mathsf{amp}(q, s) = \mathsf{en}(q)$

- minor changes to the Alloy model

Variations on the Combined Theorem

- make the ample set a function of the source product state
  - not of the intermediate state

Define $\mathcal{A}'$:



$$\alpha \in \mathsf{amp}(q, s)$$

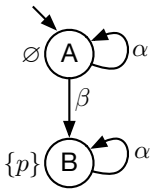
C0–C2: (same)

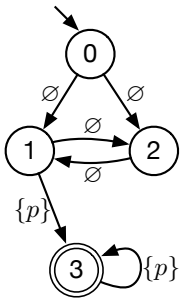
C3: In any cycle in $\mathcal{A}'$:

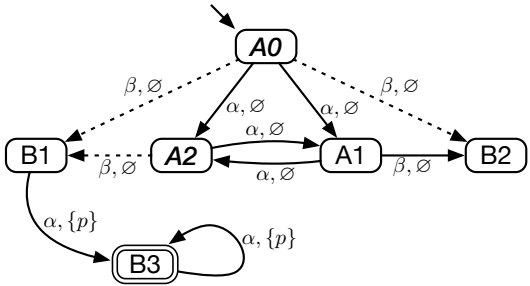
- there is a state $\langle q, s \rangle$ for which $\mathsf{amp}(q, s) = \mathsf{en}(q)$

- minor changes to the Alloy model
- Alloy Analyzer: previous Büchi automaton now OK!

New counterexample







amp	0	1	2	3
A	$\{\alpha\}$	$\{\alpha, \beta\}$	$\{\alpha\}$	$\{\alpha, \beta\}$
B	$\{\alpha\}$	$\{\alpha\}$	$\{\alpha\}$	$\{\alpha\}$

Note: only $A0$ and $A2$ are not fully enabled

Is there a fix?

Is there a fix?

- define a "stutter-invariant normal form" for Büchi automata
- every BA with a SI language can be transformed into SI normal form
- similar ideas in literature
  - Etessami 1999 (Muller automata)
  - Holzmann & Kupferman 1997 (Büchi transition system)

Is there a fix?

- define a "stutter-invariant normal form" for Büchi automata
- every BA with a SI language can be transformed into SI normal form
- similar ideas in literature
  - Etessami 1999 (Muller automata)
  - Holzmann & Kupferman 1997 (Büchi transition system)

Consequence of normal form:

1. If $s_1 \xrightarrow{a} s_2 \xrightarrow{b} s_3$ is a path in $\mathcal{B}$, then for some $s_2' \in S$, $s_1 \xrightarrow{a} s_2 \xrightarrow{a} s_2' \xrightarrow{b} s_3$ is a path in $\mathcal{B}$.
2. If $s_1 \xrightarrow{a} s_2 \xrightarrow{a} s_3 \xrightarrow{b} s_4$ is a path in $\mathcal{B}$, then $s_1 \xrightarrow{a} s_2 \xrightarrow{b} s_4$ is a path in $\mathcal{B}$. Moreover, if s_3 is accepting, then s_2 is accepting.

Is there a fix?

- define a "stutter-invariant normal form" for Büchi automata
- every BA with a SI language can be transformed into SI normal form
- similar ideas in literature
  - Etessami 1999 (Muller automata)
  - Holzmann & Kupferman 1997 (Büchi transition system)

Consequence of normal form:

1. If $s_1 \xrightarrow{a} s_2 \xrightarrow{b} s_3$ is a path in $\mathcal{B}$, then for some $s_2' \in S$, $s_1 \xrightarrow{a} s_2 \xrightarrow{a} s_2' \xrightarrow{b} s_3$ is a path in $\mathcal{B}$.
2. If $s_1 \xrightarrow{a} s_2 \xrightarrow{a} s_3 \xrightarrow{b} s_4$ is a path in $\mathcal{B}$, then $s_1 \xrightarrow{a} s_2 \xrightarrow{b} s_4$ is a path in $\mathcal{B}$. Moreover, if s_3 is accepting, then s_2 is accepting.

- this is all you need to make on-the-fly POR work

A Fix: Stutter-Invariant Normal Form

Theorem

If $\mathcal{B}$ is in SI normal form and C0–C3 holds, then $\mathcal{L}(A') = \varnothing \Leftrightarrow \mathcal{L}(A) = \varnothing$.

- a (LaTeX) proof is in the paper
- proof goes through for original and variant schemes
- verified for Alloy for small scope [thanks to reviewers]
  - ≤ 3 symbols in alphabet Σ
  - ≤ 4 states in the Büchi automaton
  - ≤ 6 transitions in Büchi automaton
  - ≤ 3 operations in program
  - ≤ 4 program states
  - ≤ 16 product states
  - time: 2,265 seconds

Conclusions

1. a theorem combining on-the-fly model checking and POR is incorrect
   - a theoretical counterexample
   - for same reason, Spin can declare an erroneous program correct

Conclusions

1. a theorem combining on-the-fly model checking and POR is incorrect
   - a theoretical counterexample
   - for same reason, Spin can declare an erroneous program correct
2. but can be fixed by restricting to certain normalized Büchi automata
   - what other classes of Büchi automata are OK?

Conclusions

1. a theorem combining on-the-fly model checking and POR is incorrect
   - a theoretical counterexample
   - for same reason, Spin can declare an erroneous program correct
2. but can be fixed by restricting to certain normalized Büchi automata
   - what other classes of Büchi automata are OK?
3. Alloy is an effective tool for checking POR algorithms
   - models can be constructed with little effort
   - can find counterexamples quickly
   - can verify correctness within small scopes

Conclusions

1. a theorem combining on-the-fly model checking and POR is incorrect
   - a theoretical counterexample
   - for same reason, Spin can declare an erroneous program correct
2. but can be fixed by restricting to certain normalized Büchi automata
   - what other classes of Büchi automata are OK?
3. Alloy is an effective tool for checking POR algorithms
   - models can be constructed with little effort
   - can find counterexamples quickly
   - can verify correctness within small scopes
4. but even better would be mechanized proofs of correctness
   - see Brunner & Lammich, 2018
     - Formal verification of an executable LTL model checker with partial order reduction, J. Automated Reasoning

Conclusions

1. a theorem combining on-the-fly model checking and POR is incorrect
   - a theoretical counterexample
   - for same reason, Spin can declare an erroneous program correct
2. but can be fixed by restricting to certain normalized Büchi automata
   - what other classes of Büchi automata are OK?
3. Alloy is an effective tool for checking POR algorithms
   - models can be constructed with little effort
   - can find counterexamples quickly
   - can verify correctness within small scopes
4. but even better would be mechanized proofs of correctness
   - see Brunner & Lammich, 2018
     - Formal verification of an executable LTL model checker with partial order reduction, J. Automated Reasoning
5. perhaps the best approach
   - use Alloy to prototype (quick iterations)
   - once Alloy says everything is OK, invest in a full mechanized proof