Topic: Bounded Synthesis

Lecture at the
1^{st} Summer School on Formal Methods for Cyber-Physical Systems

Rüdiger Ehlers, University of Bremen

September 2017

Main idea (well, my interpretation of it)

Starting point

We want to avoid

- the doubly-exponential specification automaton size blow-up *and*

Main idea (well, my interpretation of it)

Starting point

We want to avoid

- the doubly-exponential specification automaton size blow-up *and*
- the (practical) complexity of parity game solving

Main idea (well, my interpretation of it)

Starting point

We want to avoid

- the doubly-exponential specification automaton size blow-up *and*
- the (practical) complexity of parity game solving

What property do we want to exploit?

The fact that most specifications have *small* implementations.

Main idea (well, my interpretation of it)

Starting point

We want to avoid

- the doubly-exponential specification automaton size blow-up *and*
- the (practical) complexity of parity game solving

What property do we want to exploit?

The fact that most specifications have *small* implementations.

How can we exploit it?

We only build an *approximation* of the deterministic automaton that is just good enough to obtain a small implementation.

Classical (LTL) model checking (1)

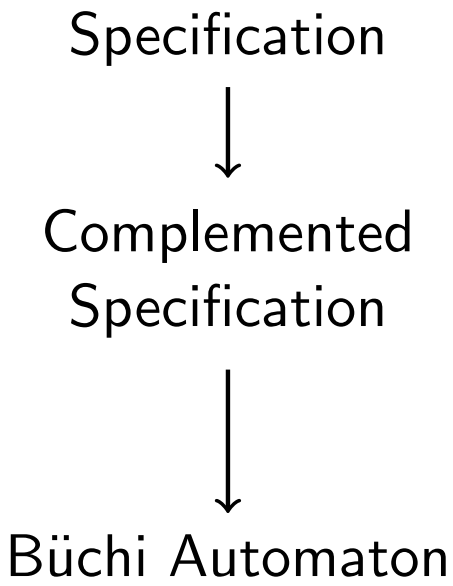
Specification

Classical (LTL) model checking (1)

Specification

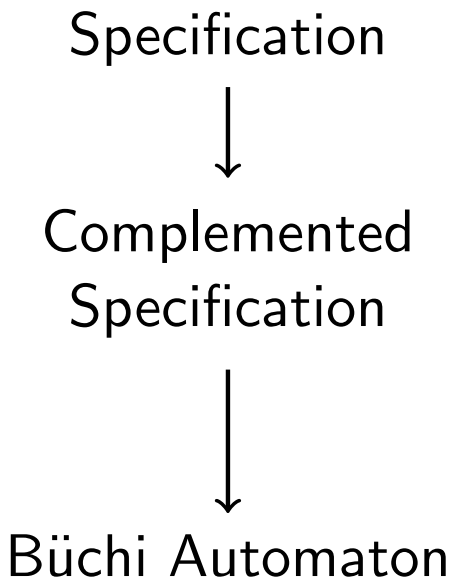
Complemented Specification

Classical (LTL) model checking (1)

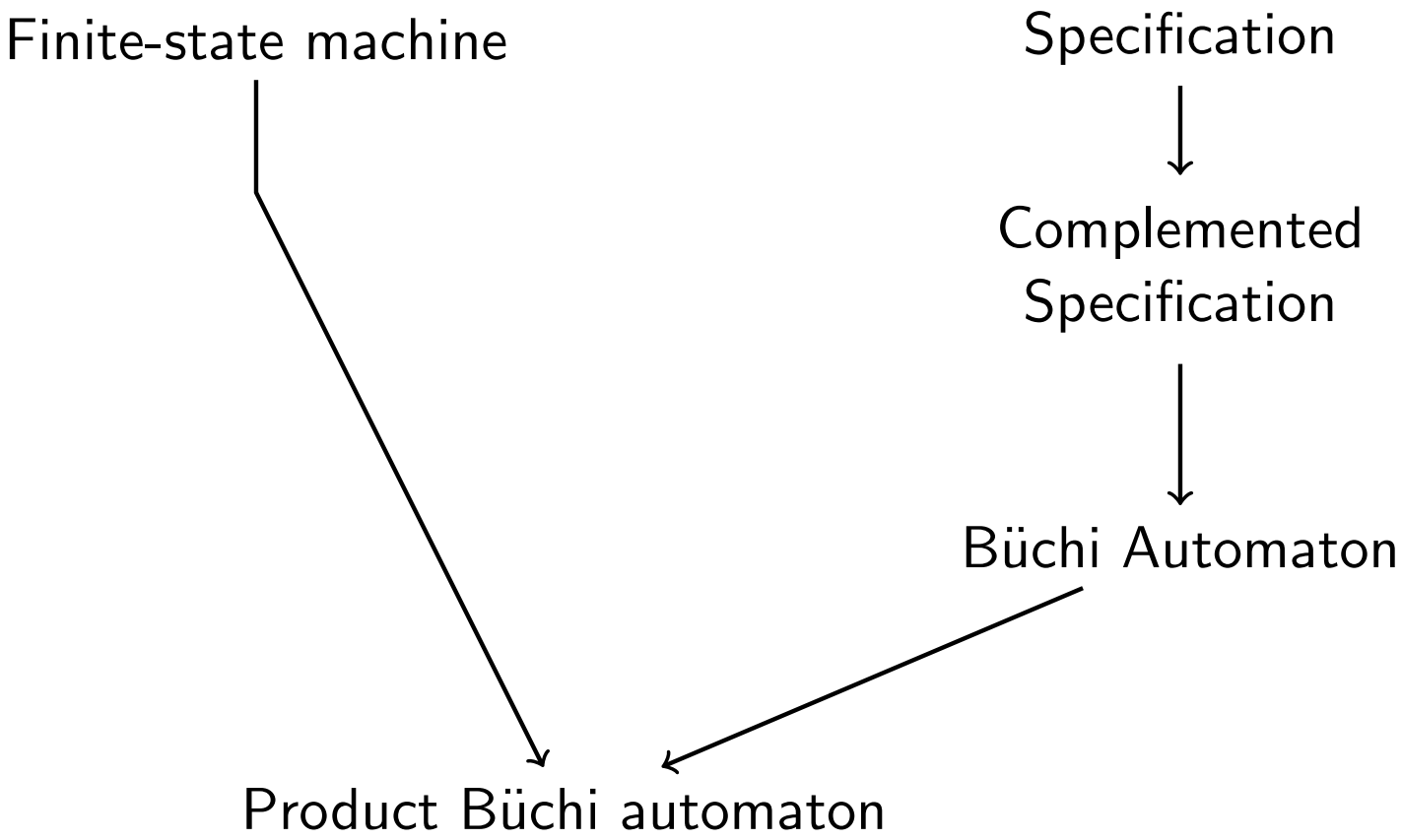


Classical (LTL) model checking (1)

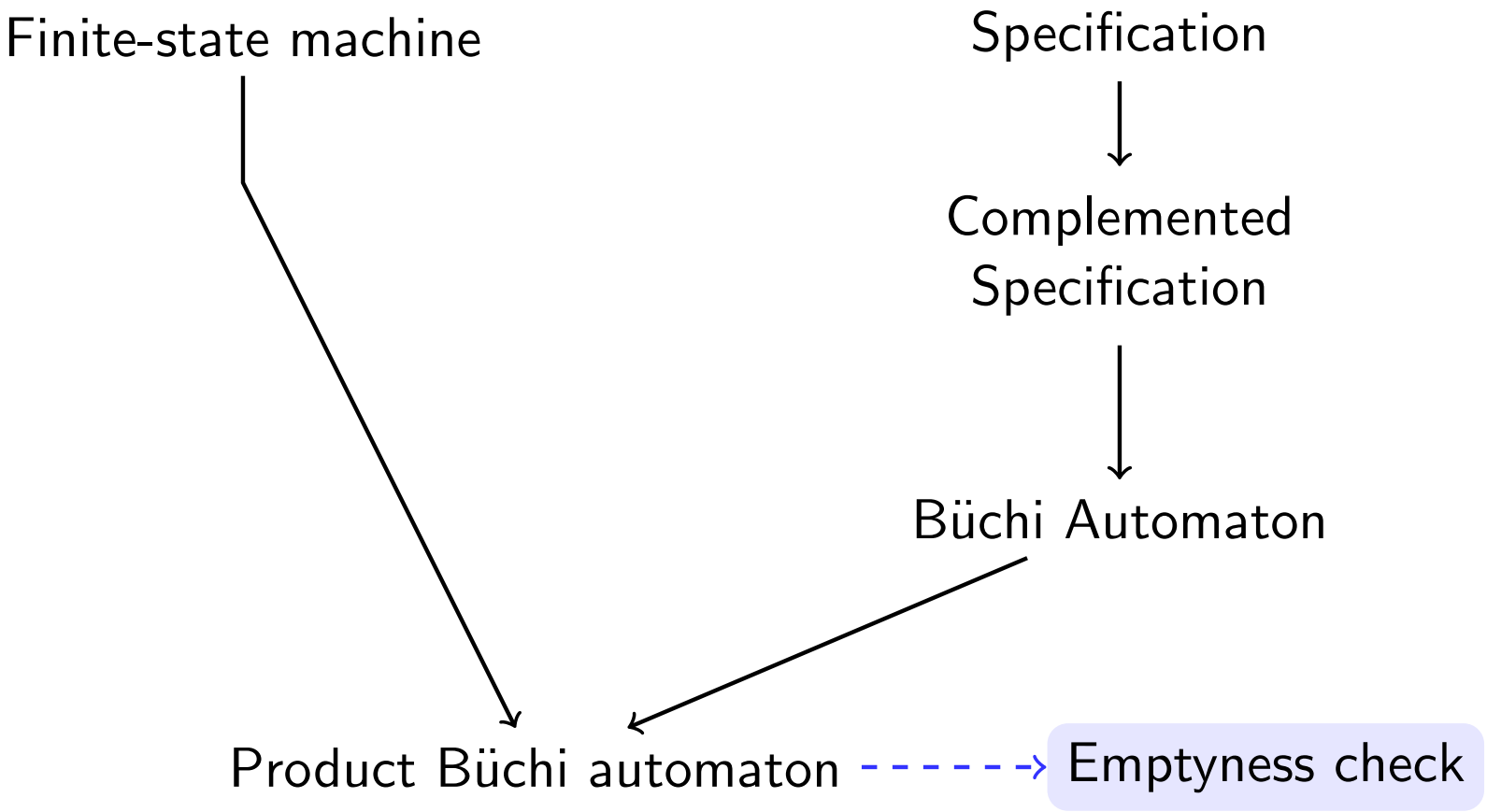
Finite-state machine



Classical (LTL) model checking (1)



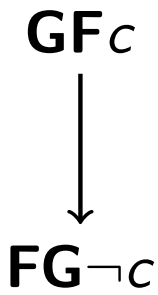
Classical (LTL) model checking (1)



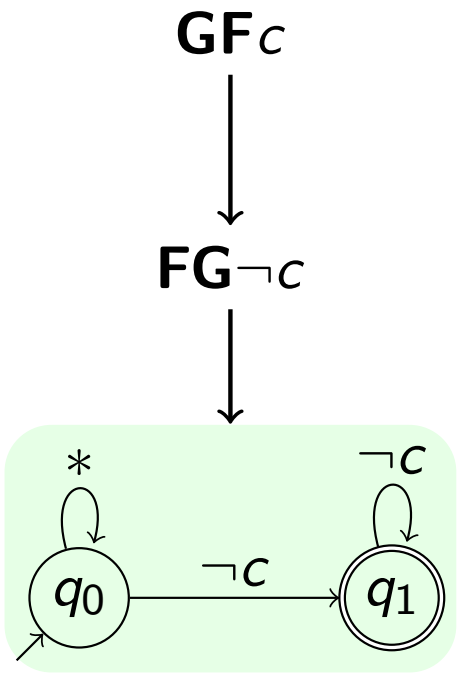
Classical (LTL) model checking (2)

$\mathbf{GF}c$

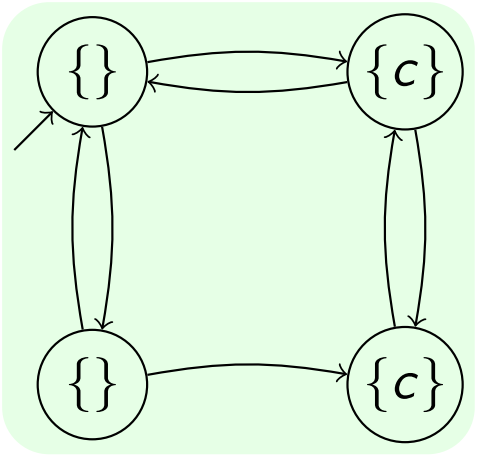
Classical (LTL) model checking (2)

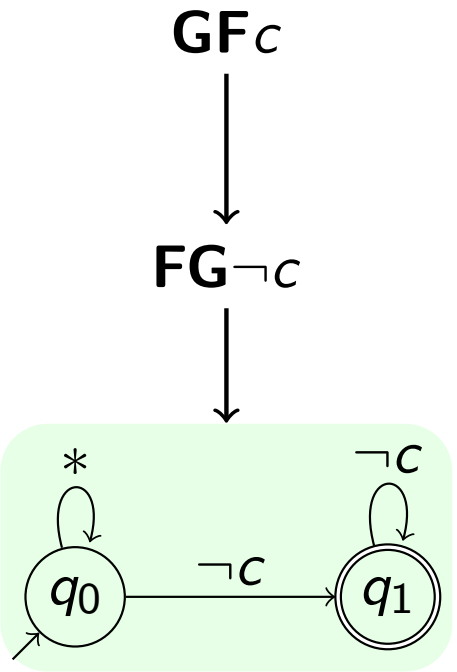


Classical (LTL) model checking (2)

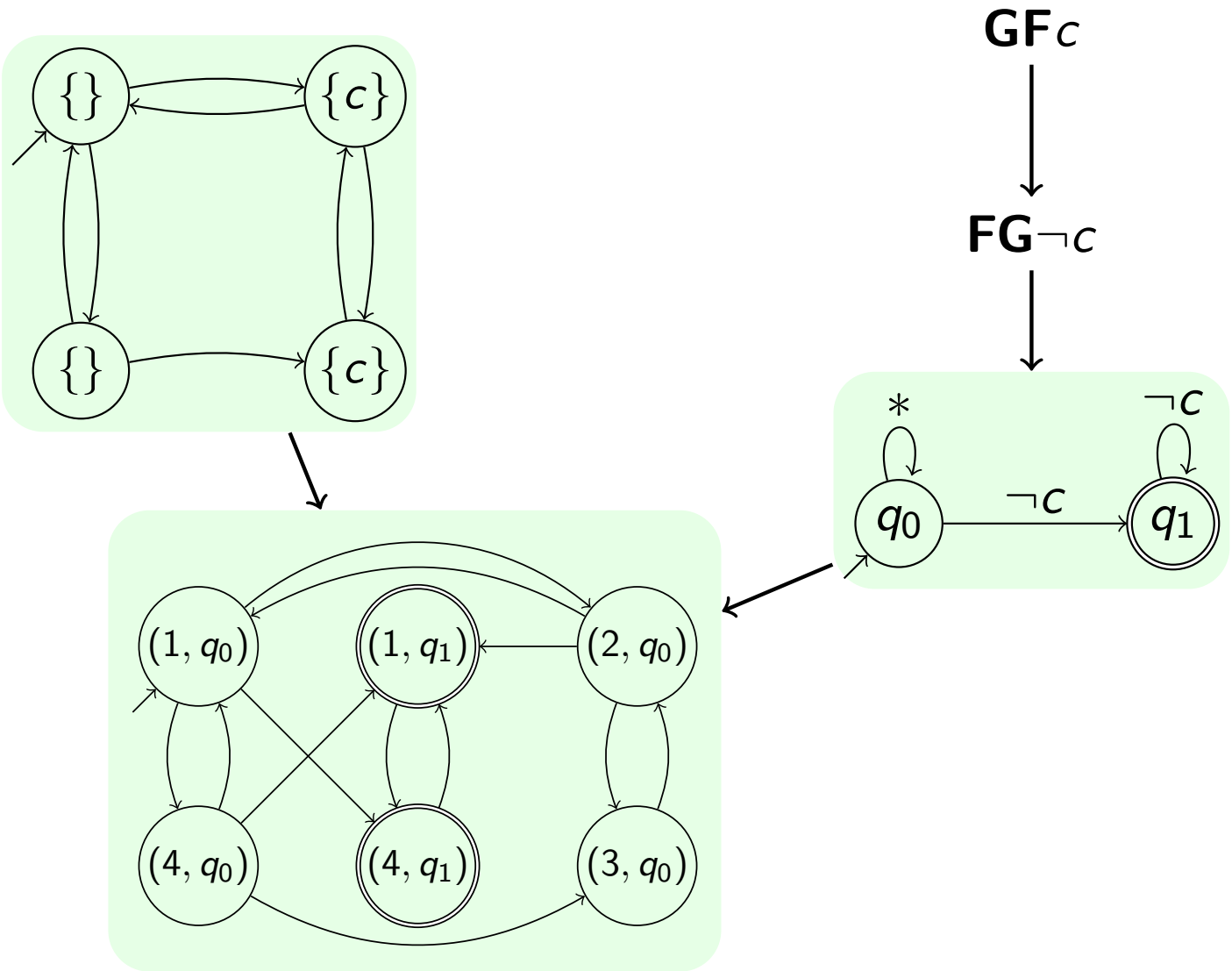


Classical (LTL) model checking (2)

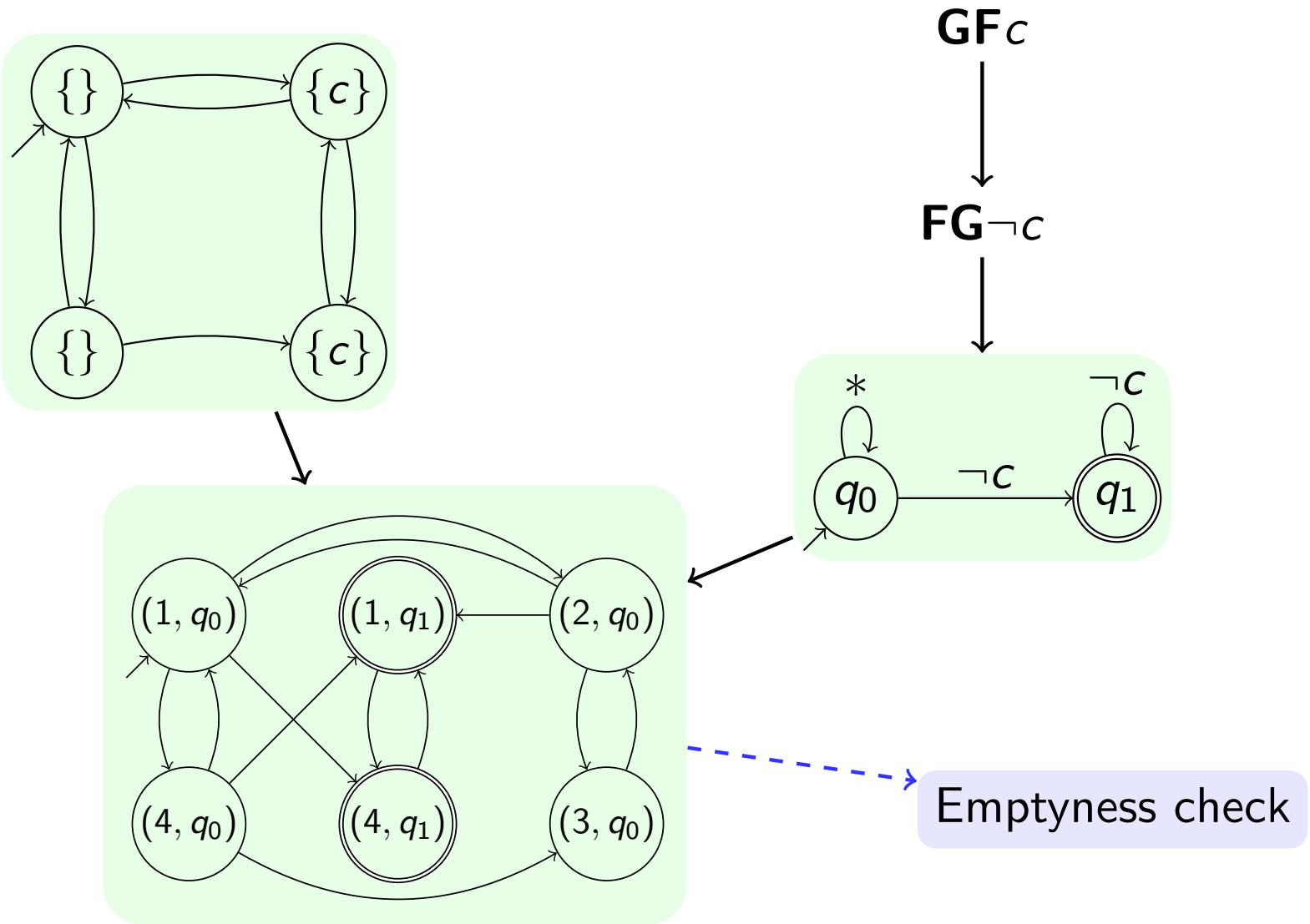




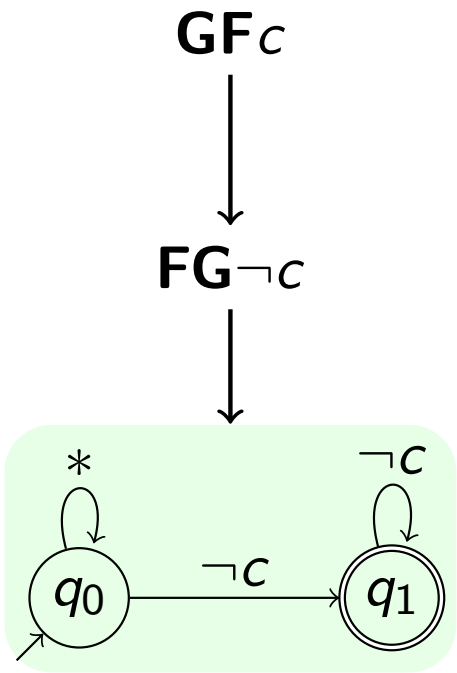
Classical (LTL) model checking (2)



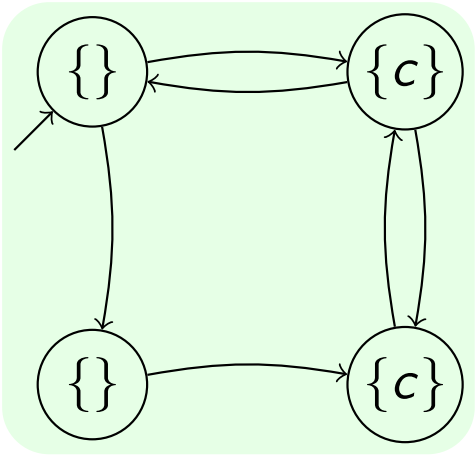
Classical (LTL) model checking (2)

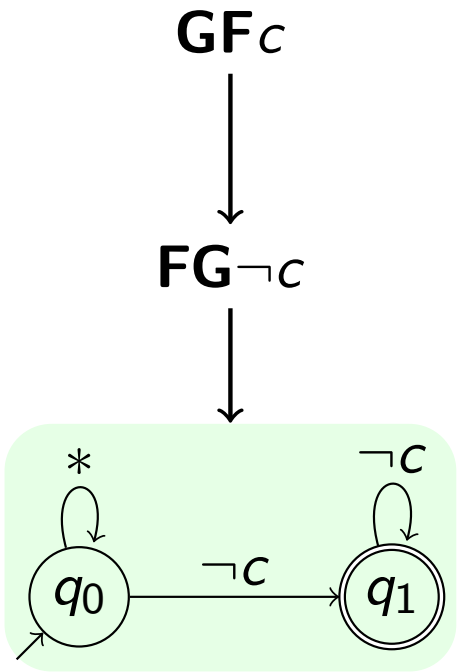


Classical (LTL) model checking (3)

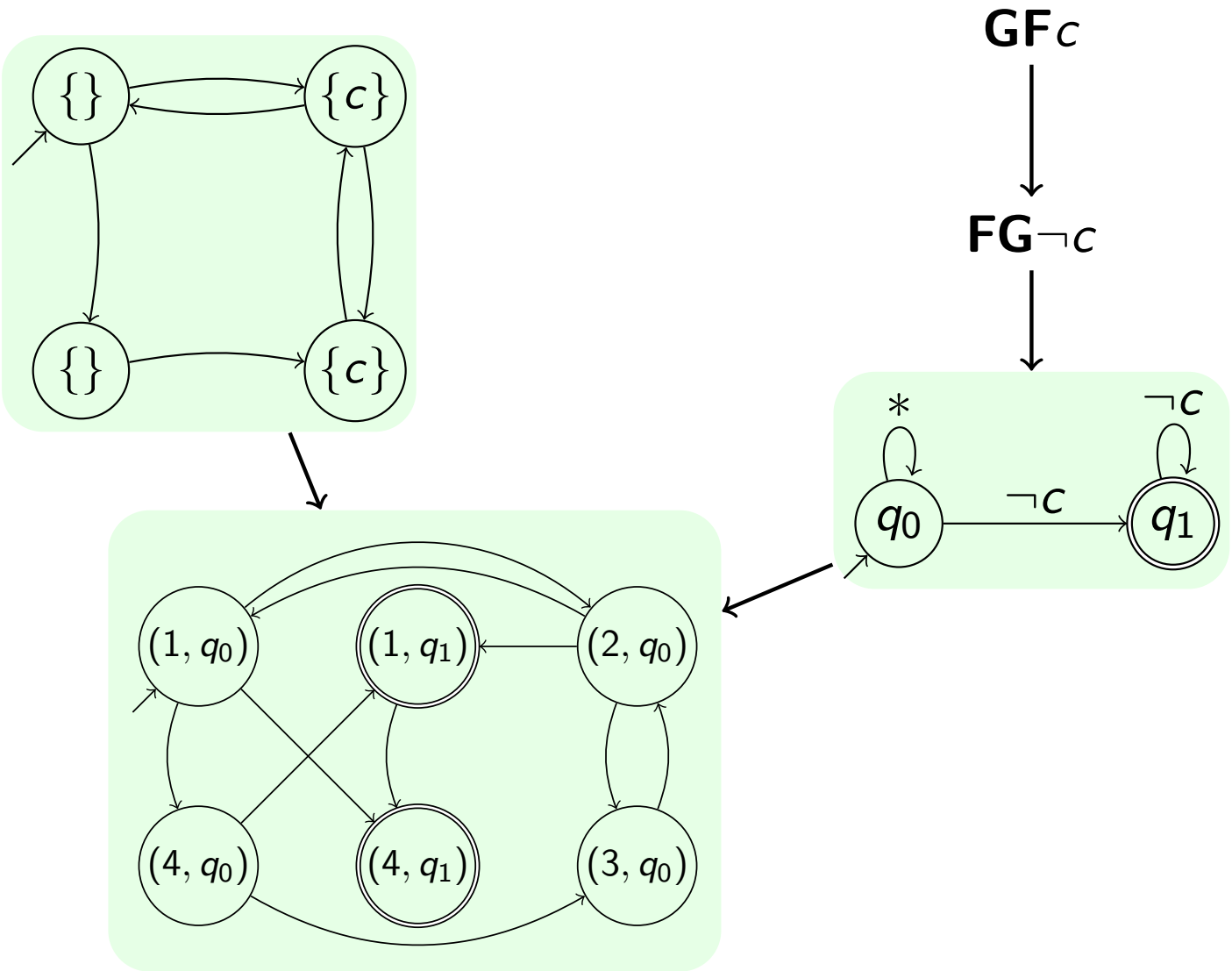


Classical (LTL) model checking (3)

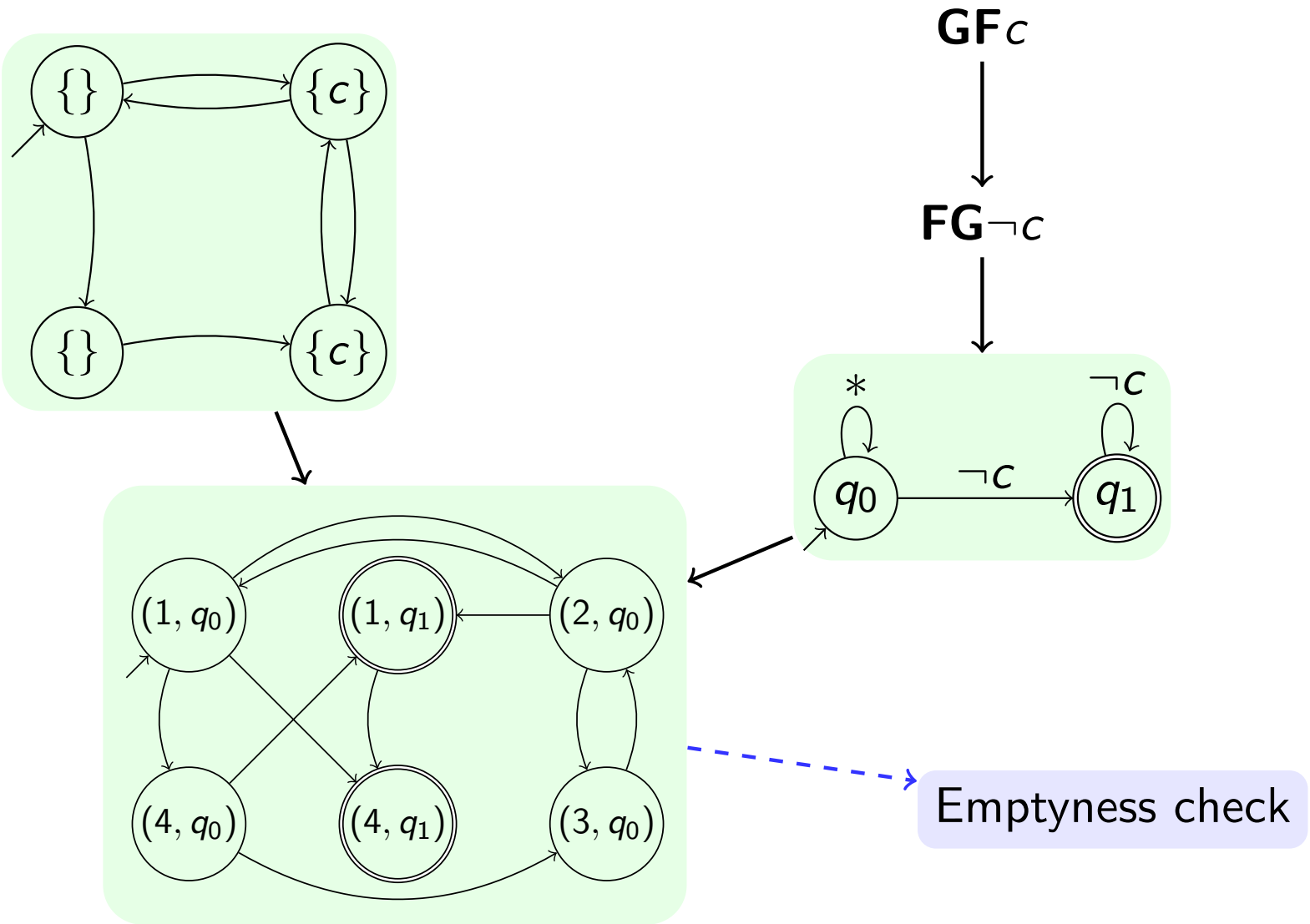




Classical (LTL) model checking (3)



Classical (LTL) model checking (3)



Property of correct implementations

Observations

- If the system satisfies the specification, then every run in the product automaton visits accepting states only *finitely* often.

Property of correct implementations

Observations

- If the system satisfies the specification, then every run in the product automaton visits accepting states only *finitely* often.
- For finite-state systems, there is then an upper bound b on the number of times an accepting state can be visited along *any* run of the automaton.

Property of correct implementations

Observations

- If the system satisfies the specification, then every run in the product automaton visits accepting states only *finitely* often.
- For finite-state systems, there is then an upper bound b on the number of times an accepting state can be visited along *any* run of the automaton.
- In **bounded synthesis** (Finkbeiner and Schewe, 2013), we exploit this fact...

Property of correct implementations

Observations

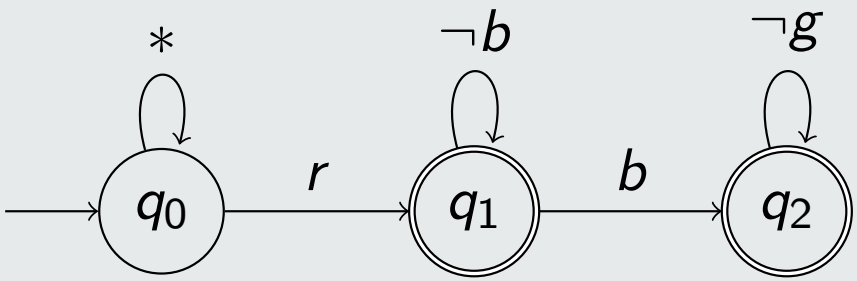
- If the system satisfies the specification, then every run in the product automaton visits accepting states only *finitely* often.
- For finite-state systems, there is then an upper bound b on the number of times an accepting state can be visited along *any* run of the automaton.
- In **bounded synthesis** (Finkbeiner and Schewe, 2013), we exploit this fact...
- ...while taking for granted that if there is an implementation for a specification, there is a finite-state implementation.

Exploiting boundedness – example

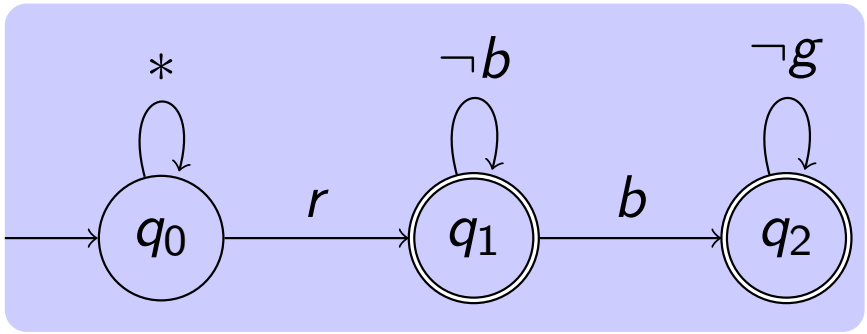
Example specification

$\mathbf{G}(r \rightarrow \mathbf{X}(\neg b \mathcal{U}(b \wedge \mathbf{XF}g)))$

Automaton for the complement of the specification

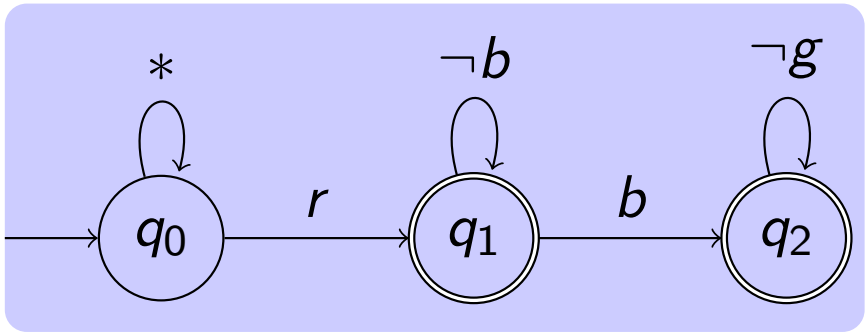


Let's see f.s. word if it is accepted by the automaton (1)



$$r\overline{g}\,\overline{b}$$
$$\vert$$
$$r\overline{g}\,\overline{b}$$
$$\vert$$
$$r\overline{g}\,b$$
$$\vert$$
$$r\overline{g}\,b$$
$$\downarrow$$

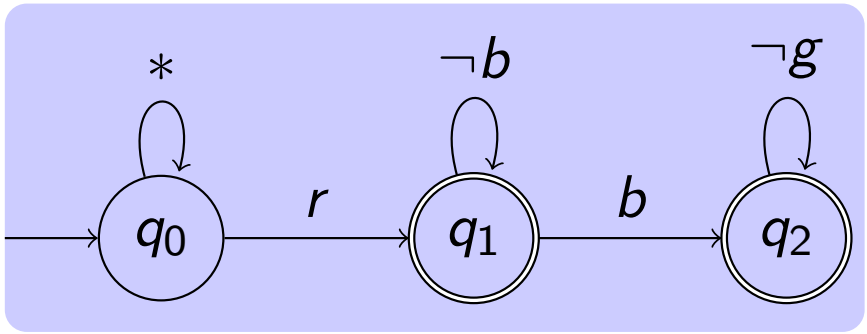
Let's see f.s. word if it is accepted by the automaton (1)



$$r\overline{g}\,\overline{b} \;|\; r\overline{g}\,\overline{b} \;|\; r\overline{g}\,b \;|\; r\overline{g}\,b \;|\; \downarrow$$

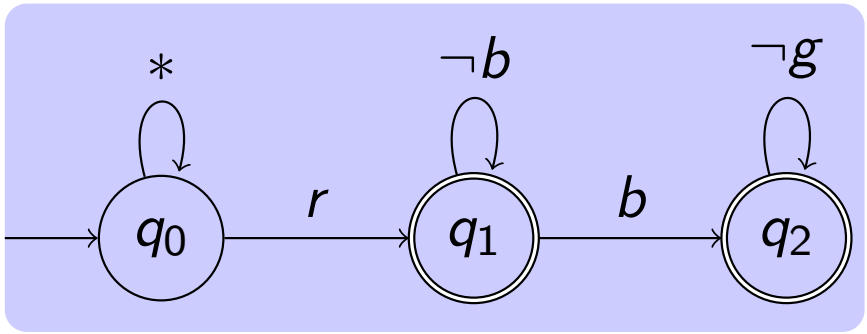
$$q_0 \;|\; q_0 \;|\; q_0 \;|\; q_0 \;|\; q_0 \;|\; \downarrow$$

Let's see f.s. word if it is accepted by the automaton (1)



$$
\begin{array}{c}
r\overline{g}\,\overline{b} \\
| \\
r\overline{g}\,\overline{b} \\
| \\
r\overline{g}\,b \\
| \\
r\overline{g}\,b \\
\downarrow
\end{array}
\qquad
\begin{array}{cc}
q_0 & q_0 \\
| & | \\
q_0 & q_1 \\
| & | \\
q_0 & q_1 \\
| & | \\
q_0 & q_2 \\
| & | \\
q_0 & q_2 \\
\downarrow & \downarrow
\end{array}
$$

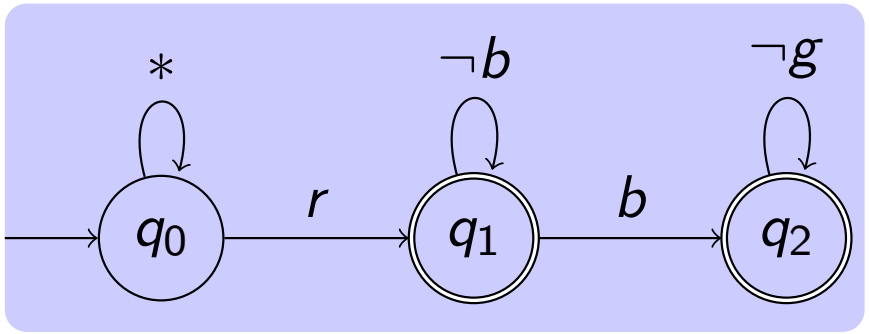
Let's see f.s. word if it is accepted by the automaton (1)



$$r\overline{g}\,\overline{b} \mid r\overline{g}\,\overline{b} \mid r\overline{g}\,b \mid r\overline{g}\,b \mid \downarrow$$

q_0	q_0	q_0
q_0	q_1	q_0
q_0	q_1	q_1
q_0	q_2	q_2
q_0	q_2	q_2
$\downarrow$	$\downarrow$	$\downarrow$

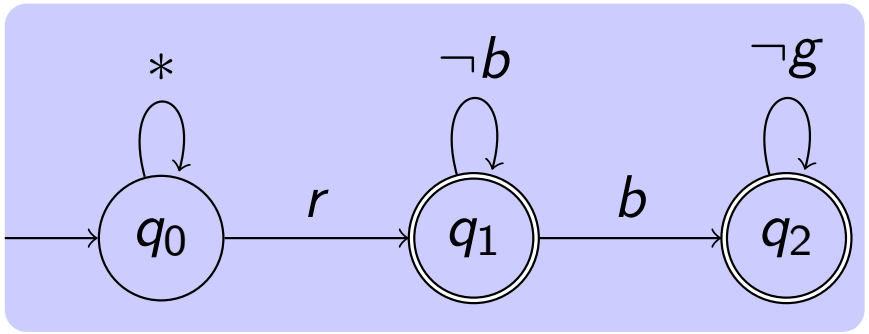
Let's see f.s. word if it is accepted by the automaton (1)



$$r\overline{g}\,\overline{b} \;|\; r\overline{g}\,\overline{b} \;|\; r\overline{g}\,b \;|\; r\overline{g}\,b \;|\; \downarrow$$

q_0	q_0	q_0	q_0	
q_0	q_1	q_0	q_0	
q_0	q_1	q_1	q_0	$\cdots$
q_0	q_2	q_2	q_1	
q_0	q_2	q_2	q_2	
$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	

Let's see f.s. word if it is accepted by the automaton (1)

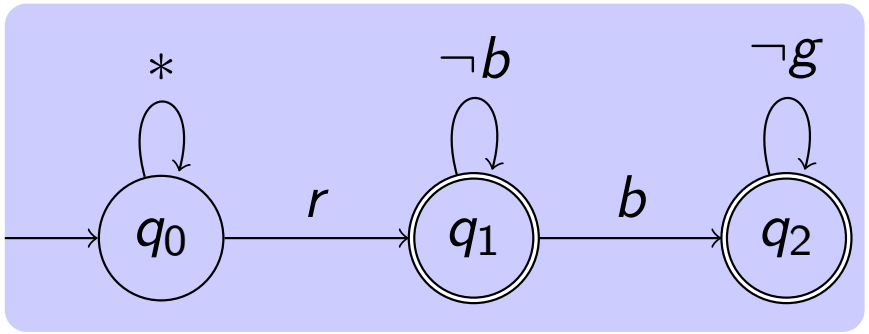


$r\overline{g}\,\overline{b}$
|
$r\overline{g}\,\overline{b}$
|
$r\overline{g}\,b$
|
$r\overline{g}\,b$
|
$r\overline{g}\,b$
↓

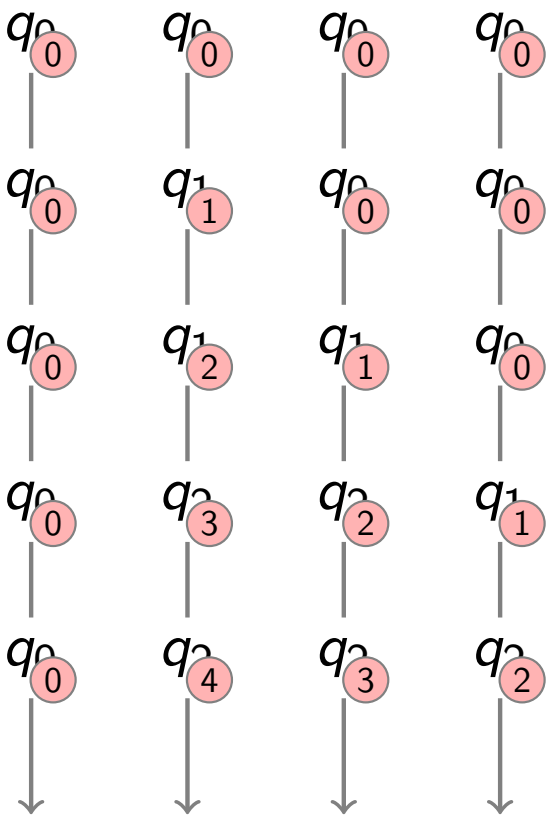
q_0	q_0	q_0	q_0
q_0	q_1	q_0	q_0
q_0	q_1	q_1	q_0
q_0	q_2	q_2	q_1
q_0	q_2	q_2	q_2
↓	↓	↓	↓

All of the runs have to reject the word in order for the word to be rejected.

Let's see f.s. word if it is accepted by the automaton (1)

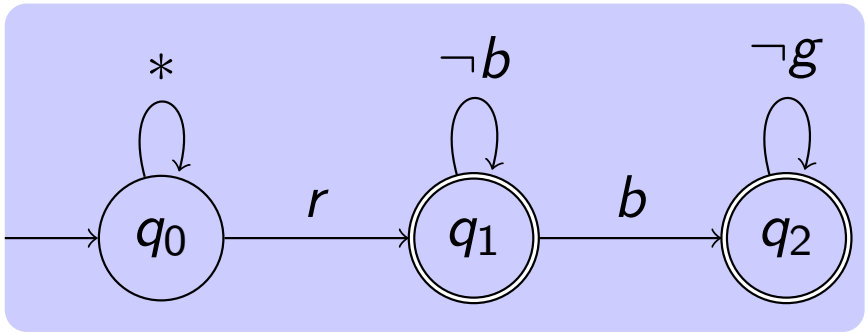


$r\overline{g}\overline{b}$
|
$r\overline{g}\overline{b}$
|
$r\overline{g}b$
|
$r\overline{g}b$
↓



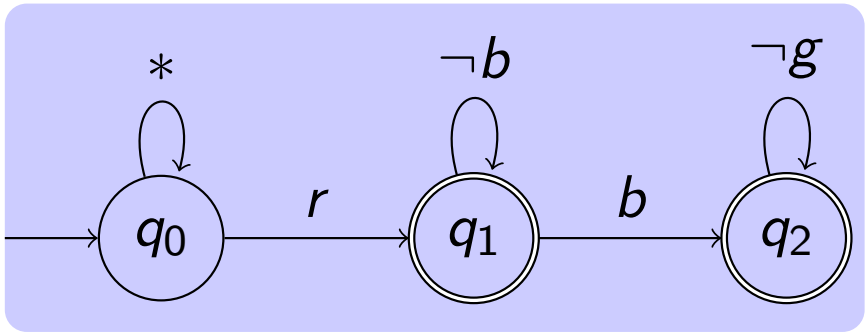
All of the runs have to reject the word in order for the word to be rejected.

Let's see f.s. word if it is accepted by the automaton (2)



$$r\overline{g}\,\overline{b}$$
$$\vert$$
$$r\overline{g}\,\overline{b}$$
$$\vert$$
$$r\overline{g}\,b$$
$$\vert$$
$$r\overline{g}\,b$$
$$\downarrow$$

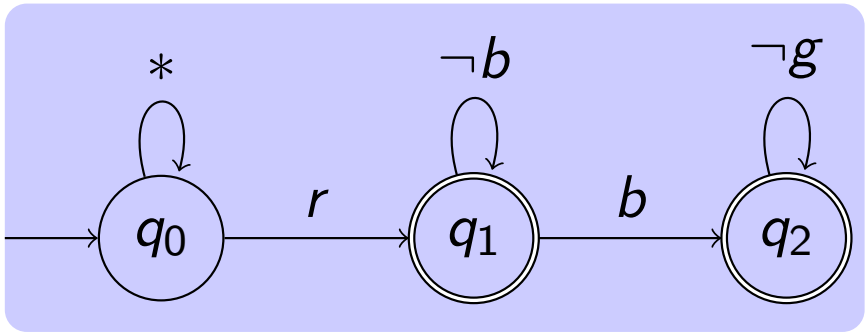
Let's see f.s. word if it is accepted by the automaton (2)



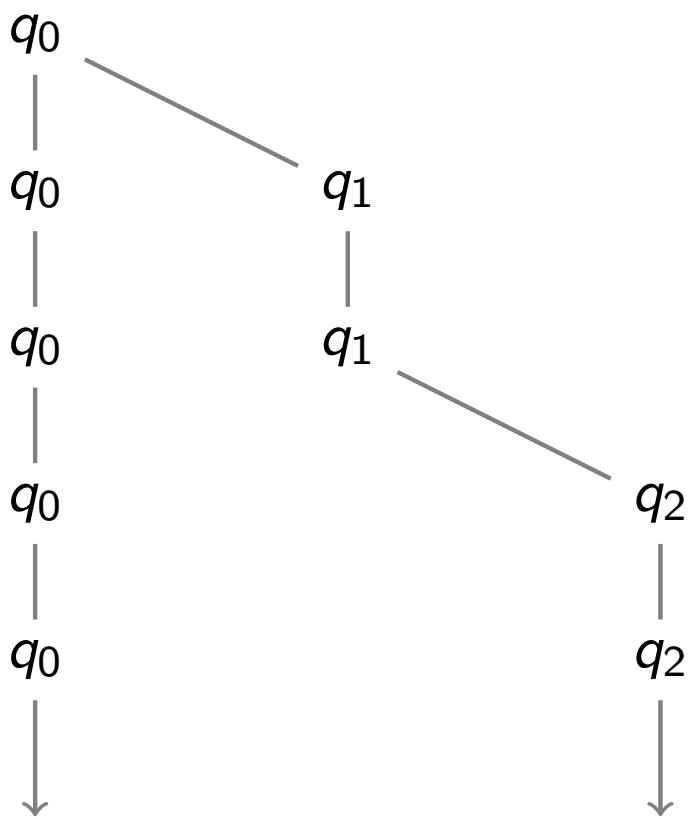
$$r\overline{g}\,\overline{b} \;|\; r\overline{g}\,\overline{b} \;|\; r\overline{g}\,b \;|\; r\overline{g}\,b \;|\; \ldots$$

$$q_0 \;|\; q_0 \;|\; q_0 \;|\; q_0 \;|\; q_0 \;|\; \ldots$$

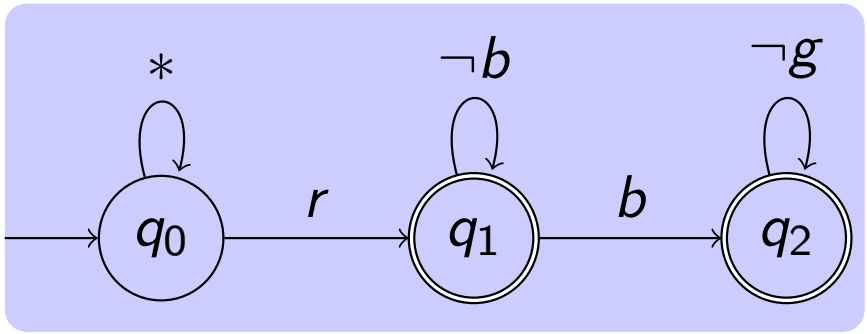
Let's see f.s. word if it is accepted by the automaton (2)



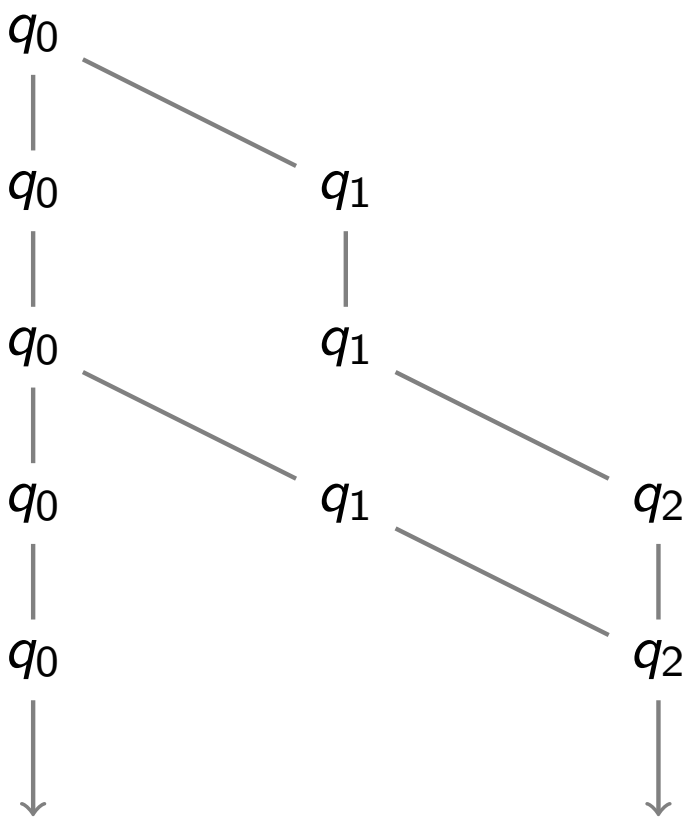
$$r\overline{g}\,\overline{b}$$
$$|$$
$$r\overline{g}\,\overline{b}$$
$$|$$
$$r\overline{g}\,b$$
$$|$$
$$r\overline{g}\,b$$
$$\downarrow$$



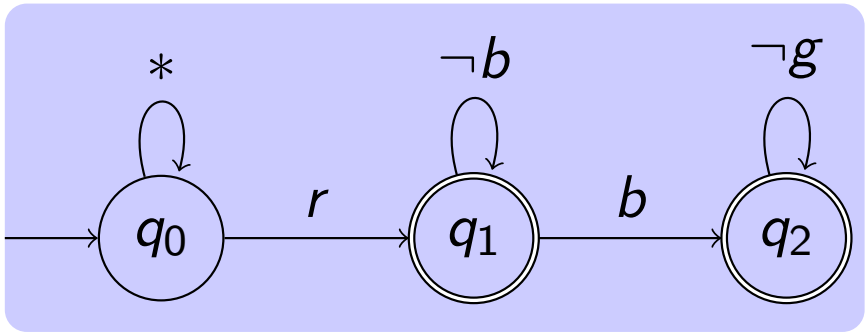
Let's see f.s. word if it is accepted by the automaton (2)



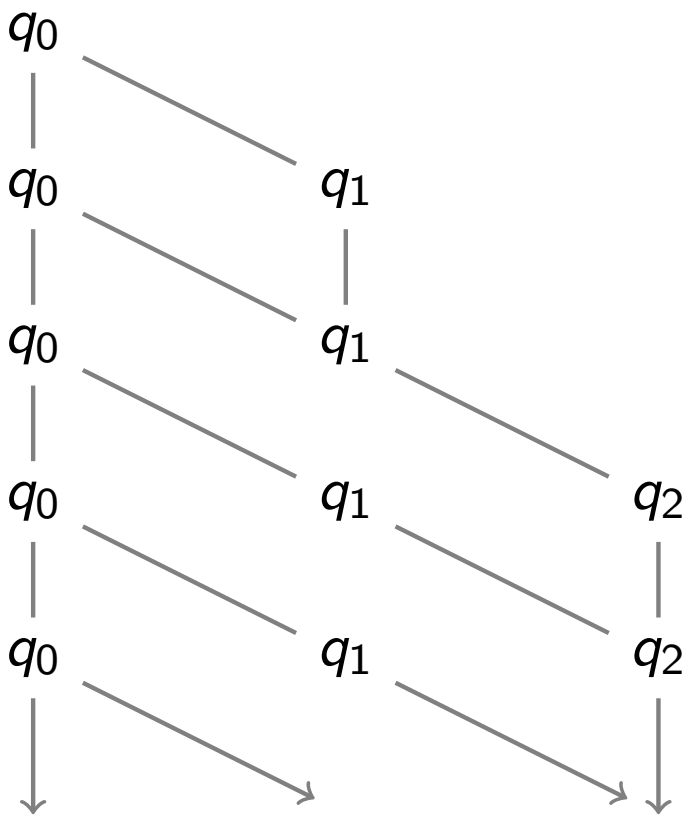
$r\overline{g}\,\overline{b}$
|
$r\overline{g}\,\overline{b}$
|
$r\overline{g}\,b$
|
$r\overline{g}\,b$
|
$\downarrow$



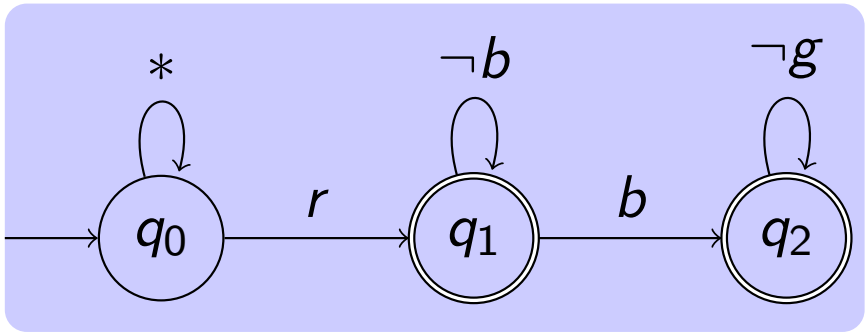
Let's see f.s. word if it is accepted by the automaton (2)



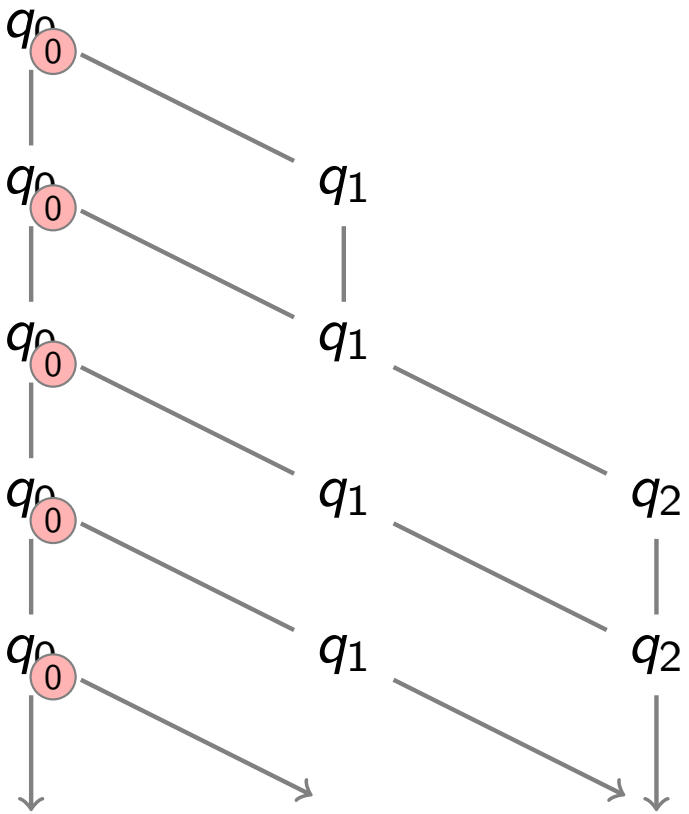
$$r\overline{g}\,\overline{b}$$
$$|$$
$$r\overline{g}\,\overline{b}$$
$$|$$
$$r\overline{g}\,b$$
$$|$$
$$r\overline{g}\,b$$
$$\downarrow$$



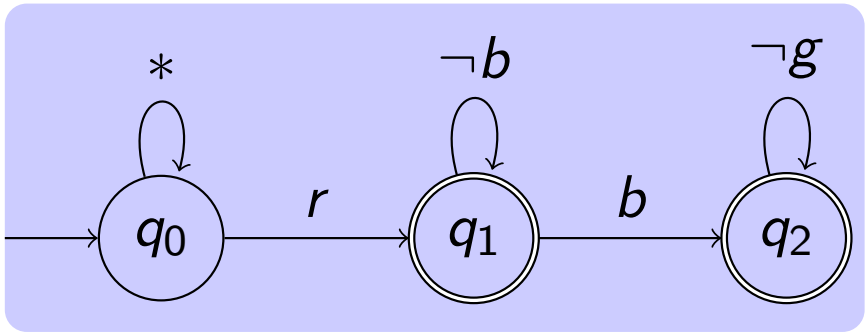
Let's see f.s. word if it is accepted by the automaton (2)



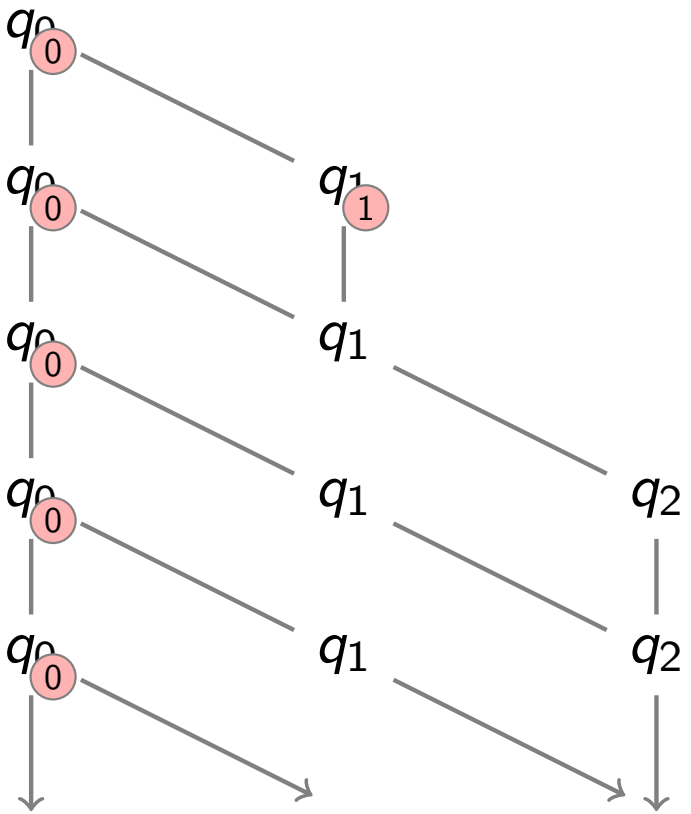
$r\overline{g}\,\overline{b}$
|
$r\overline{g}\,\overline{b}$
|
$r\overline{g}\,b$
|
$r\overline{g}\,b$
|
$\downarrow$



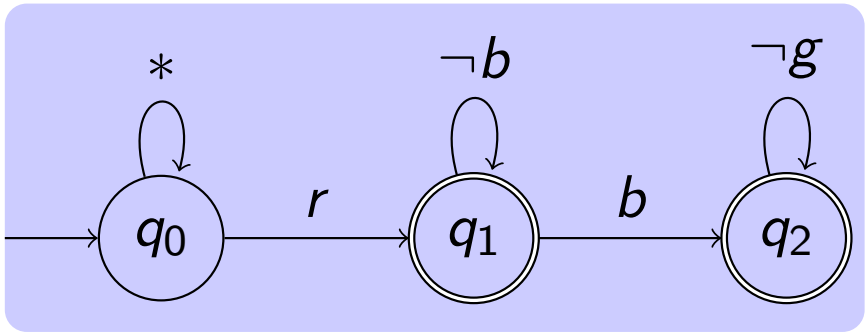
Let's see f.s. word if it is accepted by the automaton (2)



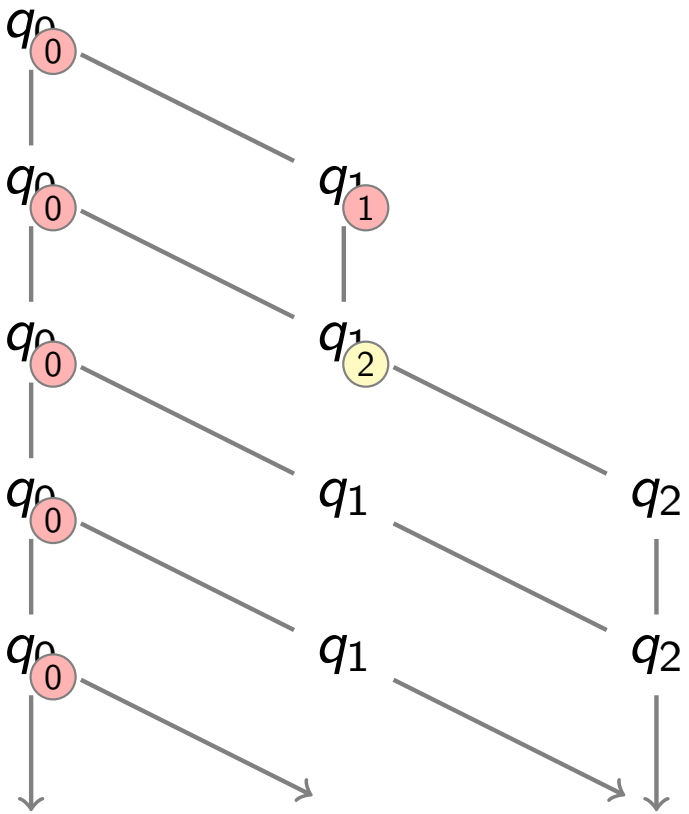
$r\overline{g}\,\overline{b}$
|
$r\overline{g}\,\overline{b}$
|
$r\overline{g}\,b$
|
$r\overline{g}\,b$
|
$\downarrow$



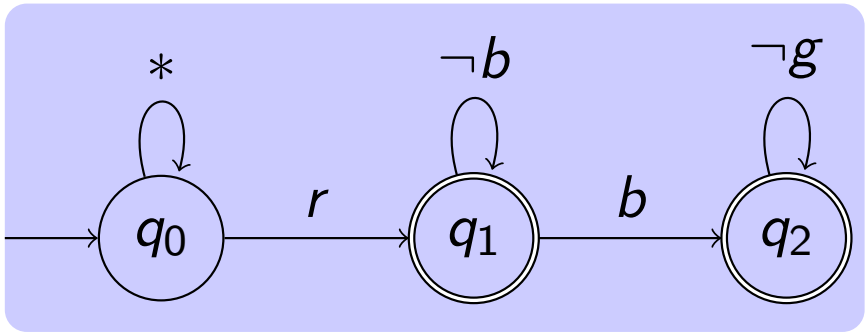
Let's see f.s. word if it is accepted by the automaton (2)



$$r\overline{g}\,\overline{b}$$
$$|$$
$$r\overline{g}\,\overline{b}$$
$$|$$
$$r\overline{g}\,b$$
$$|$$
$$r\overline{g}\,b$$
$$\downarrow$$



Let's see f.s. word if it is accepted by the automaton (2)

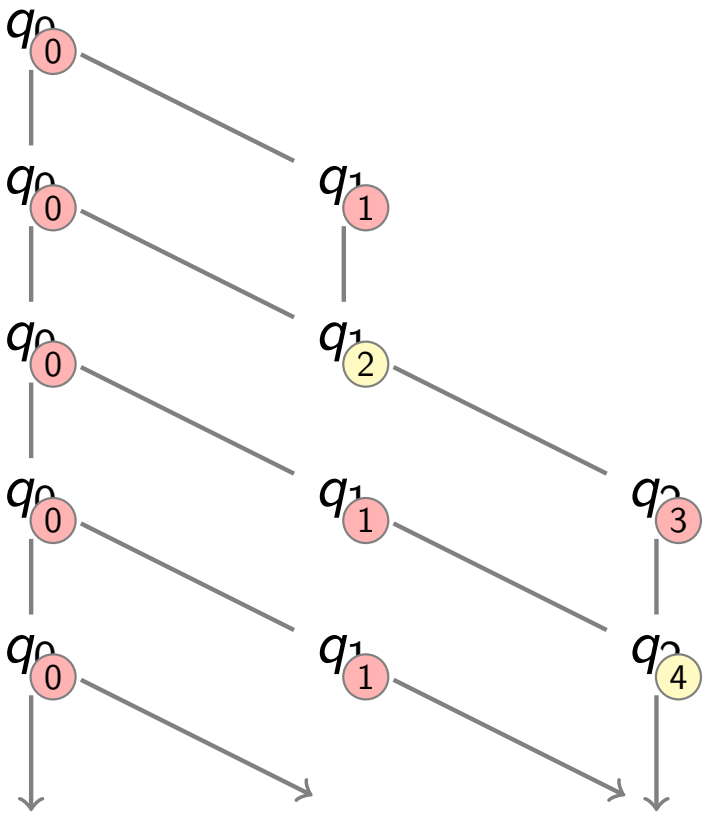


$r\overline{g}\,\overline{b}$

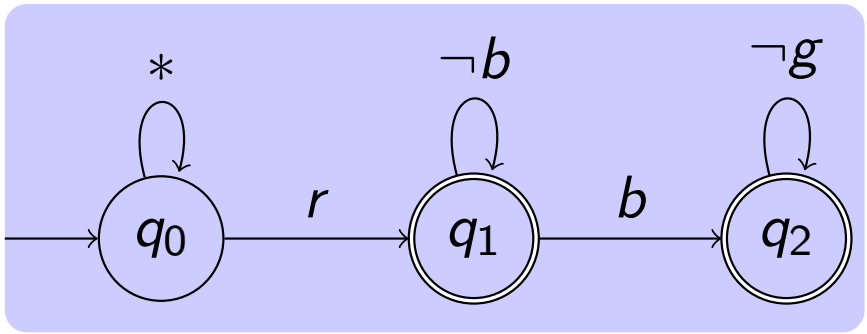
$r\overline{g}\,\overline{b}$

$r\overline{g}\,b$

$r\overline{g}\,b$

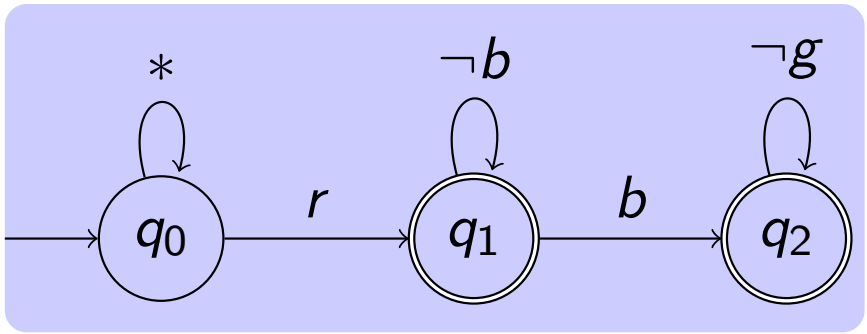


Let's see f.s. word if it is accepted by the automaton (3)



$$r\overline{g}\,\overline{b}$$
$$|$$
$$r\overline{g}\,\overline{b}$$
$$|$$
$$r\overline{g}\,b$$
$$|$$
$$r\overline{g}\,b$$
$$\downarrow$$

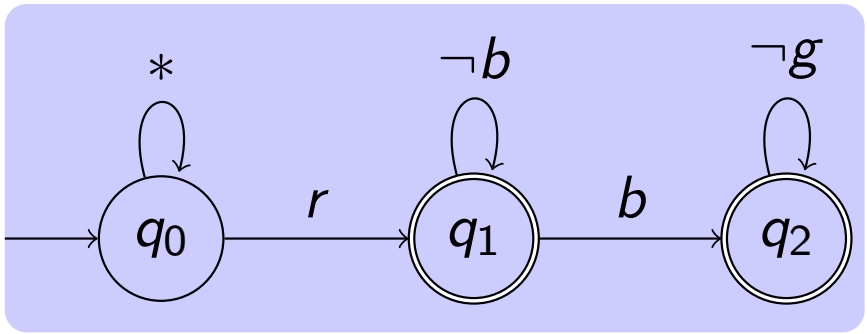
Let's see f.s. word if it is accepted by the automaton (3)



$r\overline{g}\,\overline{b}$
|
$r\overline{g}\,\overline{b}$
|
$r\overline{g}\,b$
|
$r\overline{g}\,b$
|
$\downarrow$

$(0, \bot, \bot)$
|
$(0, 1, \bot)$
|
$(0, 2, \bot)$
|
$(0, 1, 3)$
|
$(0, 1, 4)$
|
$\downarrow$

Let's see f.s. word if it is accepted by the automaton (3)



$r\overline{g}\,\overline{b}$
|
$r\overline{g}\,\overline{b}$
|
$r\overline{g}\,b$
|
$r\overline{g}\,b$
|
$\downarrow$

$(0, \bot, \bot)$
|
$(0, 1, \bot)$
|
$(0, 2, \bot)$
|
$(0, 1, 3)$
|
$(0, 1, 4)$
|
$\downarrow$

The counters progress
deterministically
(for fixed input and output)

Using bounds for synthesis

Idea

If we define a finite bound $b \in \mathbb{N}$, then there are only finitely many counter tuples $(c_1, \ldots, c_n) \in (\{0, \ldots, b, \bot\})^n$ for $n = |Q|$.

Approach

- Translate the (negation of the) specification to a non-det. Büchi automaton
- Translate the Büchi automaton and a bound $b \in \mathbb{N}$ to a deterministic safety automaton that rejects a word if and only if any of the counters exceed $b \in \mathbb{N}$.
- Translate the safety automaton to a game and solve it as usual.

Getting rid of the double negation

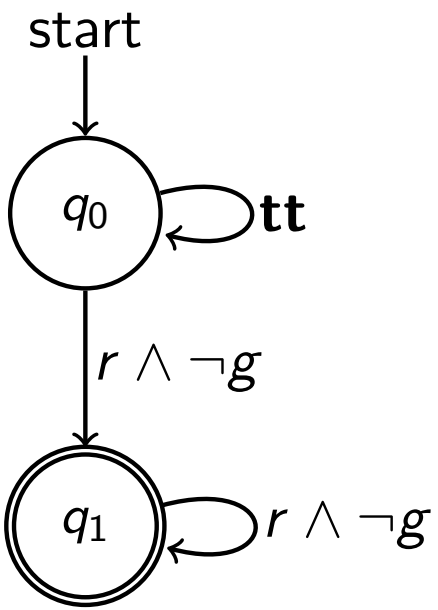
So far

The complementation of the specification is represented as a non-deterministic Büchi word (NBW) automaton.

To simplify thinking...

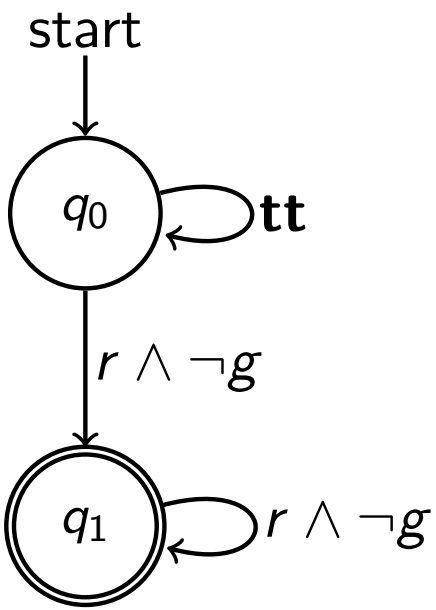
We henceforth say that the specification is represented as a *universal co-Büchi word automata (UCW)*.

Integrated example



NBW for the negated specification / UCW for the specification

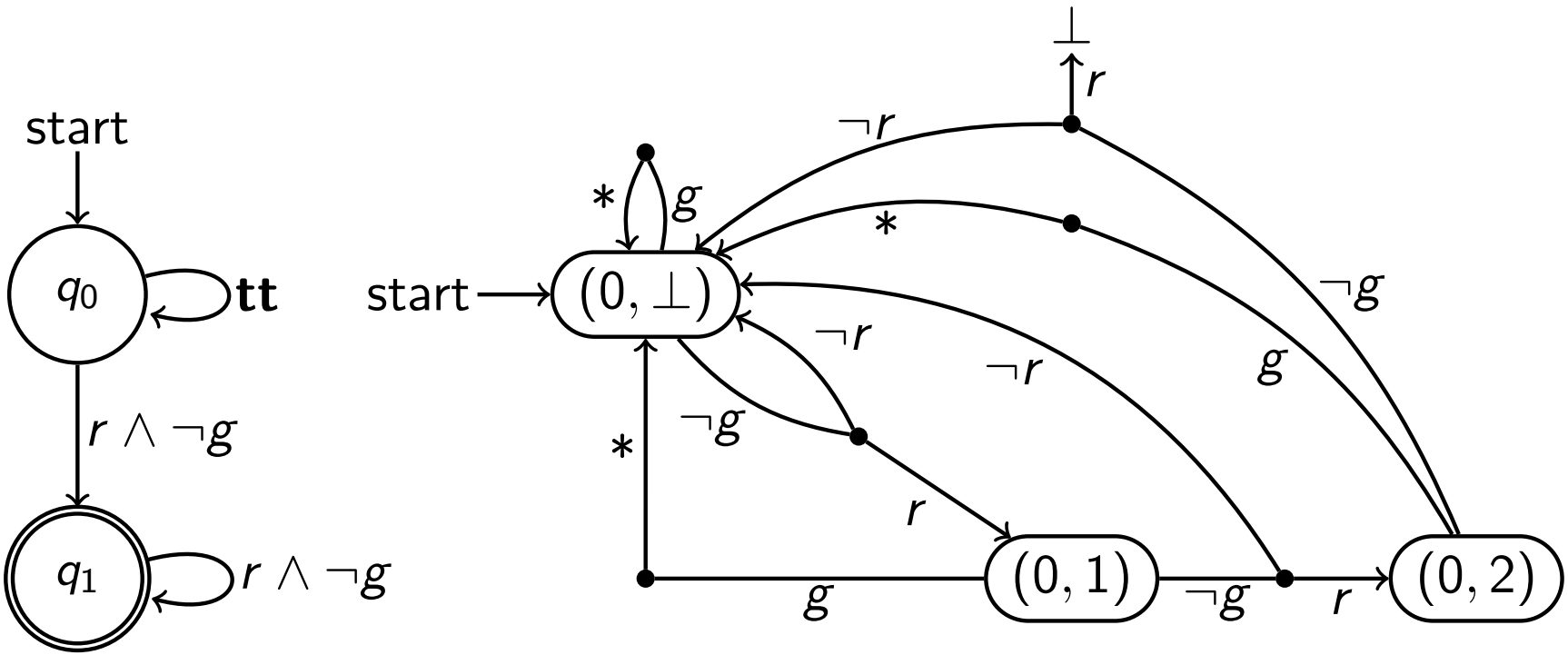
Integrated example



Integrated example

Note

- The bound in this example is 2



Now did that make sense?

The size of the automata/games

The size of the generated automata is $(b+2)^n$ for $n = |Q|$, which is ok for small b, but can still be huge! $\rightarrow$ bottleneck

Now did that make sense?

The size of the automata/games

The size of the generated automata is $(b+2)^n$ for $n = |Q|$, which is ok for small b, but can still be huge! $\rightarrow$ bottleneck

What if the resulting game is losing for the system?

...then the bound may be too small or the specification is *unrealizable*. By using a bound of $b = 2^{O(n \cdot \log n)}$, the game becomes *complete*.

Now did that make sense?

The size of the automata/games

The size of the generated automata is $(b+2)^n$ for $n = |Q|$, which is ok for small b, but can still be huge! $\rightarrow$ bottleneck

What if the resulting game is losing for the system?

...then the bound may be too small or the specification is *unrealizable*. By using a bound of $b = 2^{O(n \cdot \log n)}$, the game becomes *complete*.

Is this problem avoidable (in theory)?

No, as the overall approach would be *incomplete* if it had a complexity smaller than 2EXPTIME

What about unrealizable specifications?

Problem

Going all the way to the worst-case bound is infeasible

Solution

We solve two synthesis problems in parallel. One for the system, and one for the environment. Oftentimes, one of them can be solved with a small bound.

Efficiency in Bounded Synthesis

Remaining main problem

The games that we build are huge!

Efficiency in Bounded Synthesis

Remaining main problem

The games that we build are huge!

Some possible solutions

- Symbolic game solving (which we discuss later)
- Compositional synthesis
- SMT-based bounded synthesis

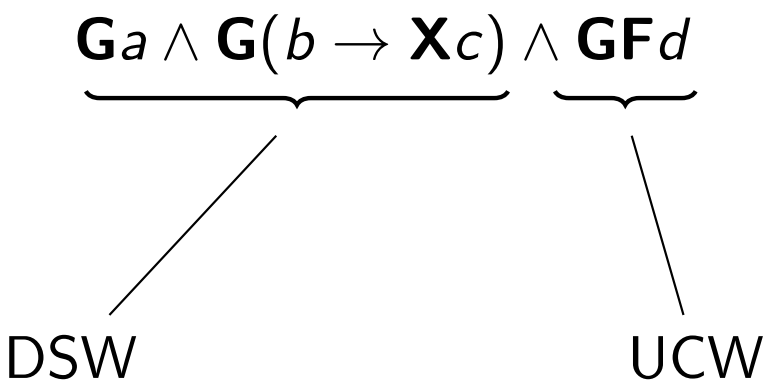
Splitting a simple conjunction

$$\mathbf{G}a \wedge \mathbf{G}(b \rightarrow \mathbf{X}c) \wedge \mathbf{GF}d$$

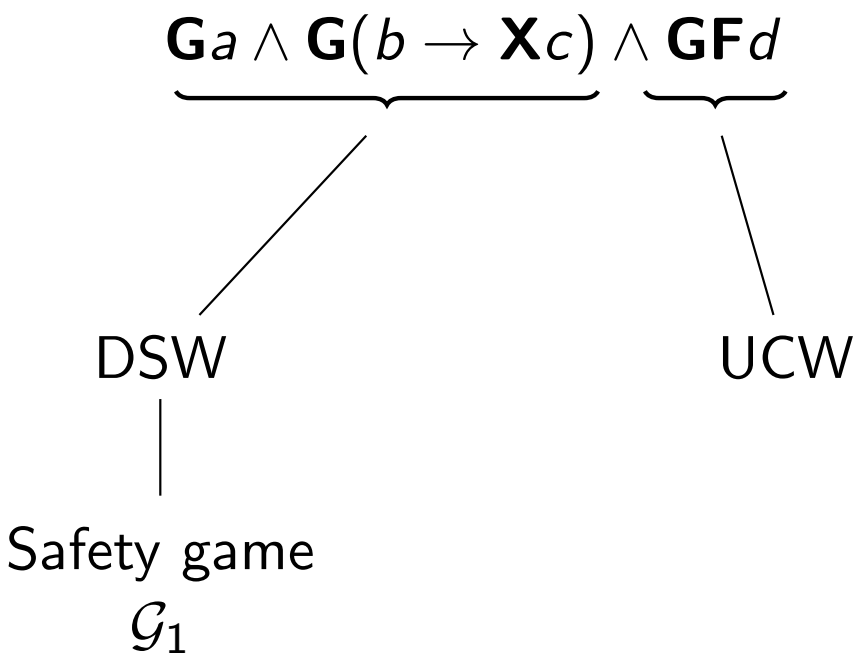
Splitting a simple conjunction

$$\underbrace{\mathbf{G}a \wedge \mathbf{G}(b \rightarrow \mathbf{X}c)}_{\text{safety}} \wedge \underbrace{\mathbf{GF}d}_{\text{non-safety}}$$

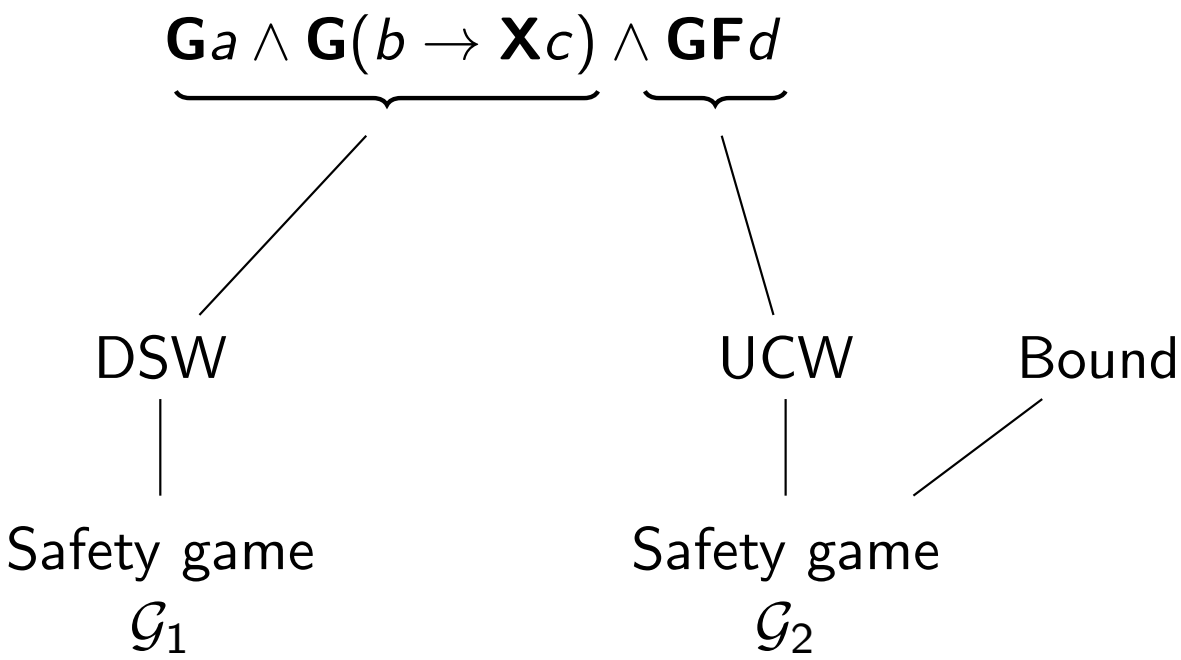
Splitting a simple conjunction



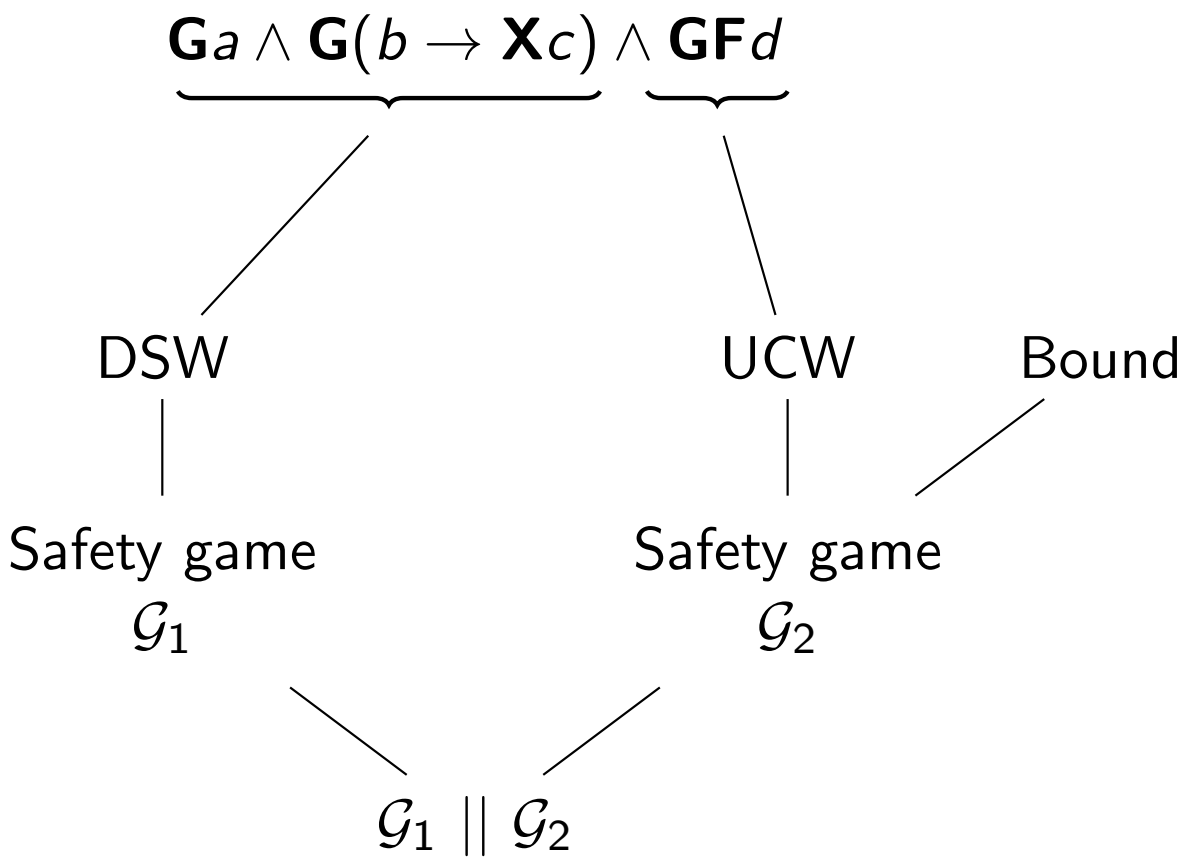
Splitting a simple conjunction



Splitting a simple conjunction



Splitting a simple conjunction



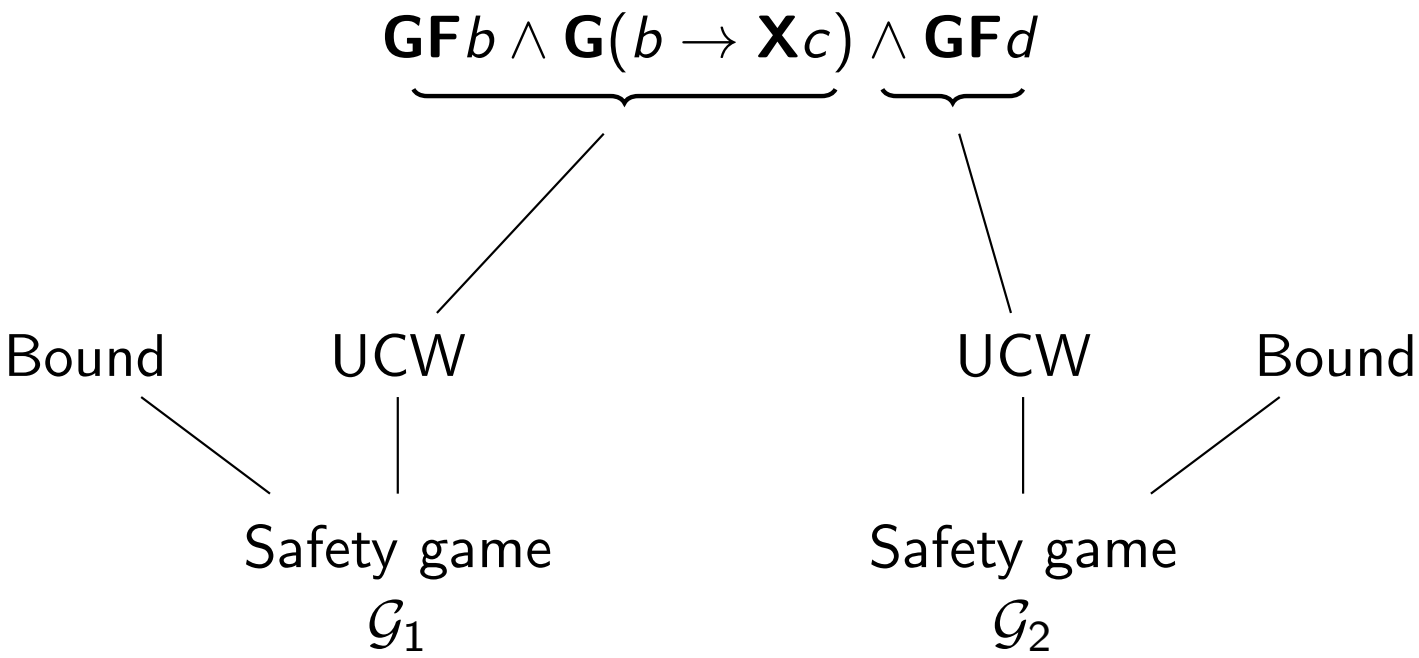
Winning condition

The system player wins $\mathcal{G}_1 \,||\, \mathcal{G}_2$ iff she wins $\mathcal{G}_1$ and $\mathcal{G}_2$ at the same time.

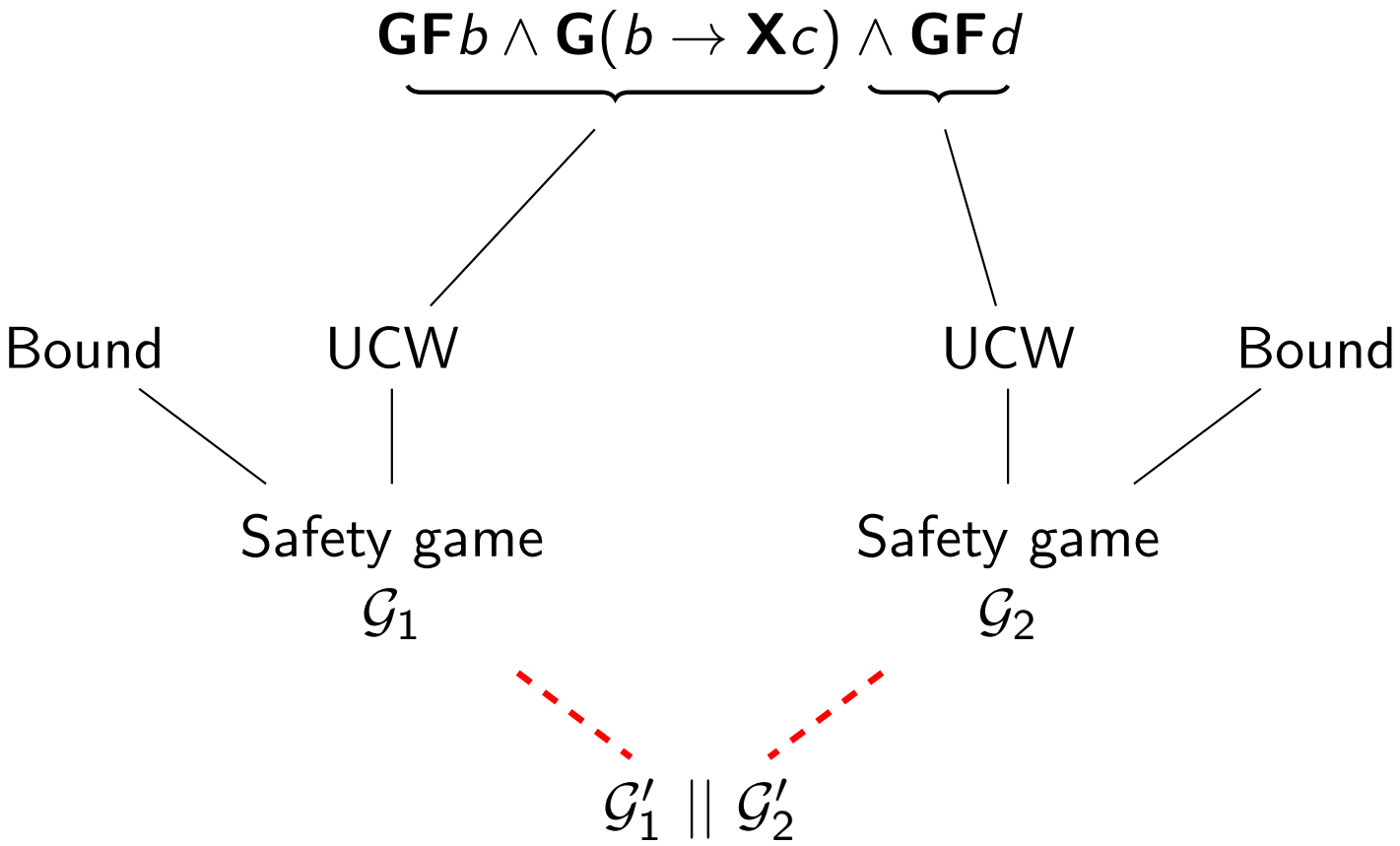
The Acacia+ approach (Filiot et al., 2010)

$$\mathbf{GF}b \wedge \mathbf{G}(b \rightarrow \mathbf{X}c) \wedge \mathbf{GF}d$$

The Acacia+ approach (Filiot et al., 2010)



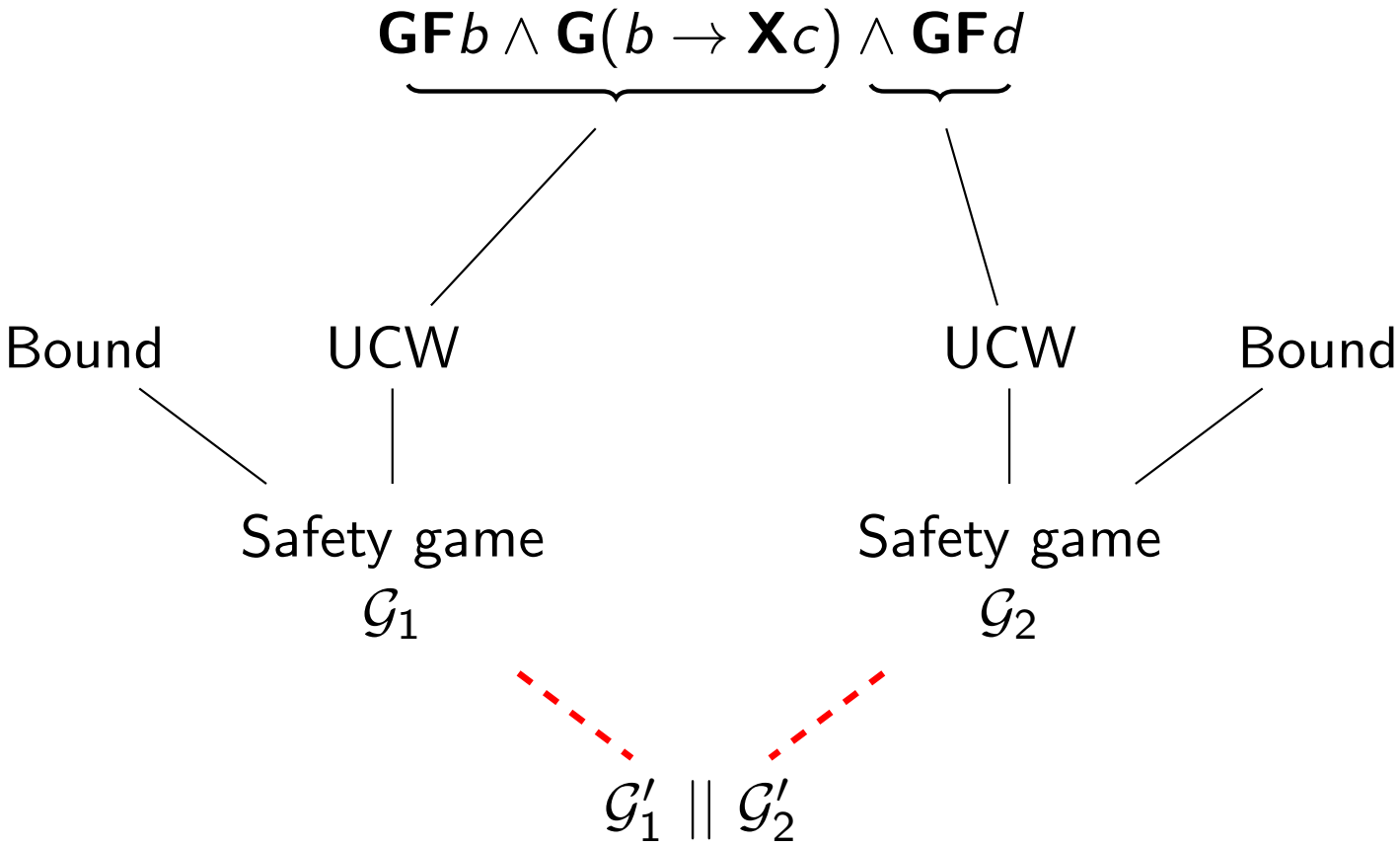
The Acacia+ approach (Filiot et al., 2010)



Main advantages

- LTL-to-UCW translation is more efficient
- Safety games can be pruned before composing

The Acacia+ approach (Filiot et al., 2010)



Main disadvantages

- Does not work for proving unrealizability
- Safety games can still be large
- Which bound to increase if the product game is lost by the system player?

Practical bounded synthesis

Example

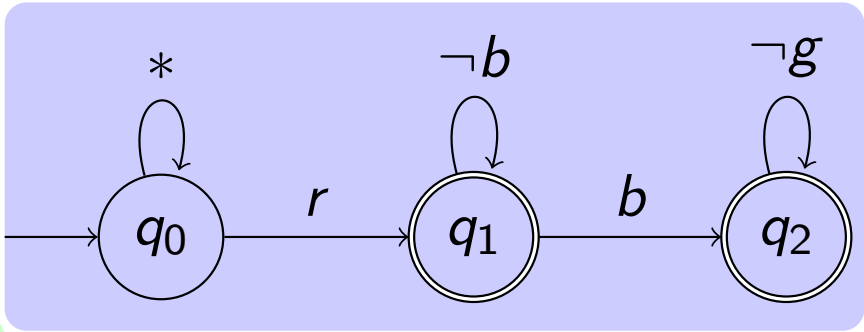
- We use `Acacia+`
- Online version: `http://lit2.ulb.ac.be/acaciaplus/onlinetest/LTLSynthesis/`
- Specification: `G F g && G ( g` $\rightarrow$ `r )`
- Input bit is r, output bit is g

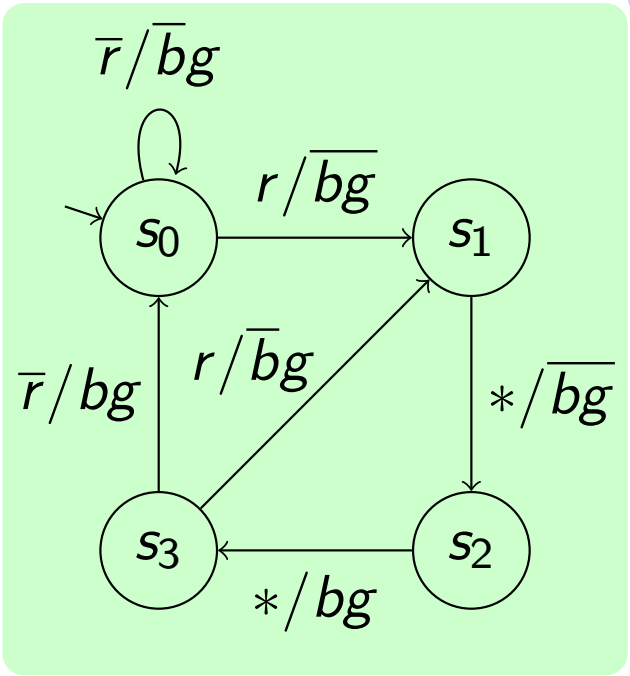
SMT-based bounded synthesis (Finkbeiner and Schewe, 2013)

Main question

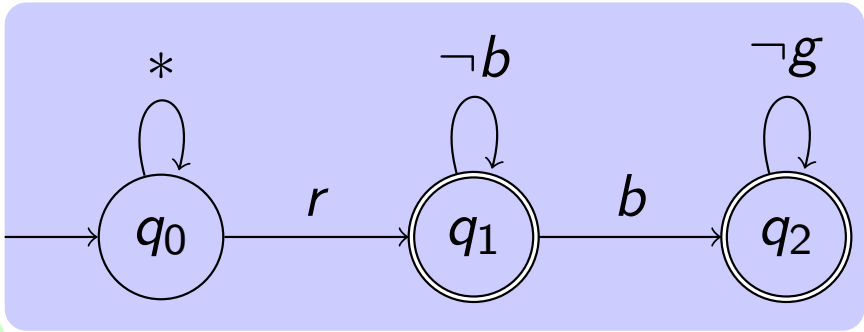
Can we find a way to perform *bounded synthesis* without actually building the synthesis game?

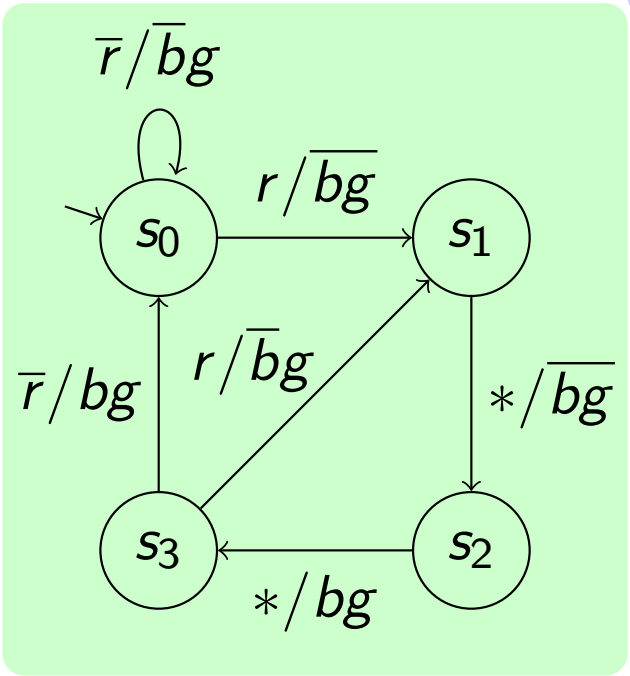
SMT-Based Bounded Synthesis – Starting observation





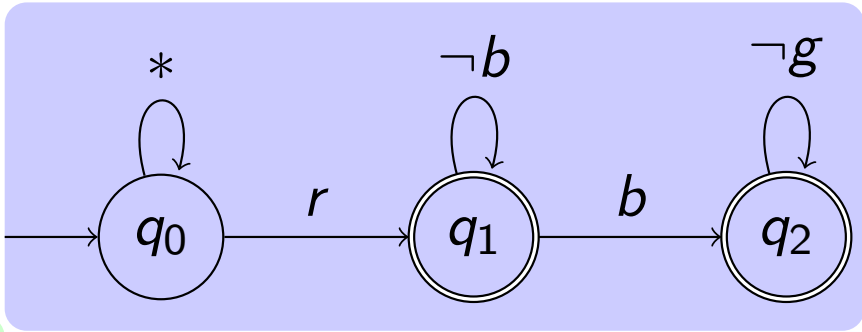
SMT-Based Bounded Synthesis – Starting observation

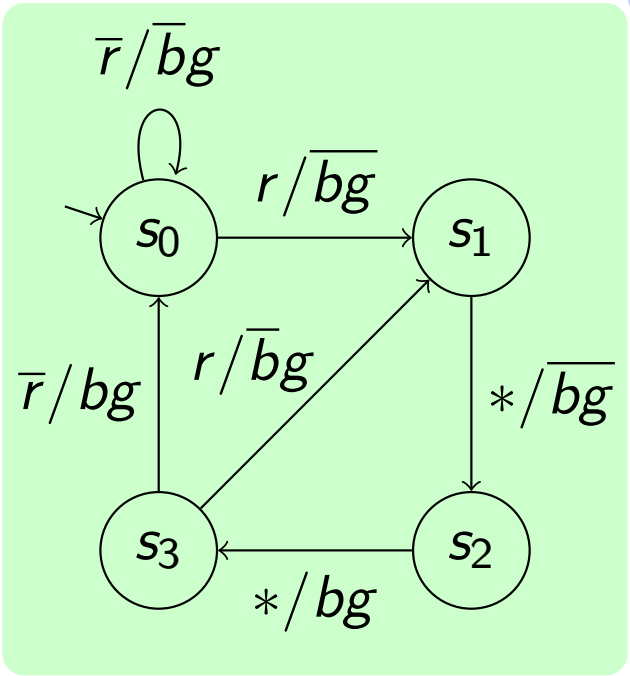




The FSM is correct with a bound of 4, i.e., rejecting states are visited at most 4 times along any run of the product.

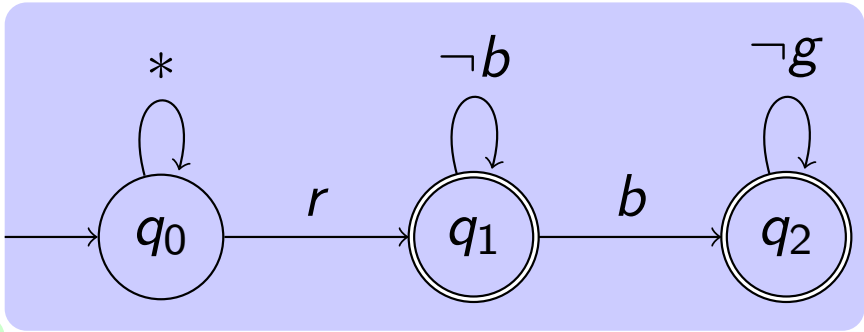
SMT-Based Bounded Synthesis – Starting observation

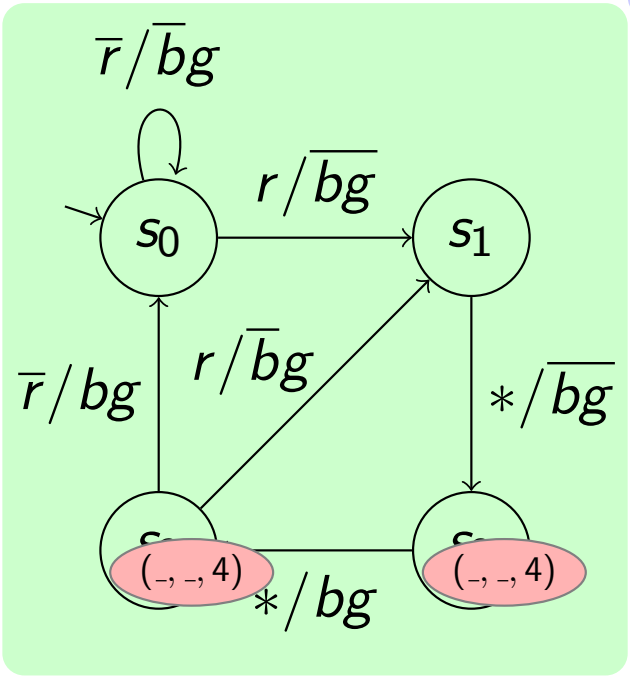




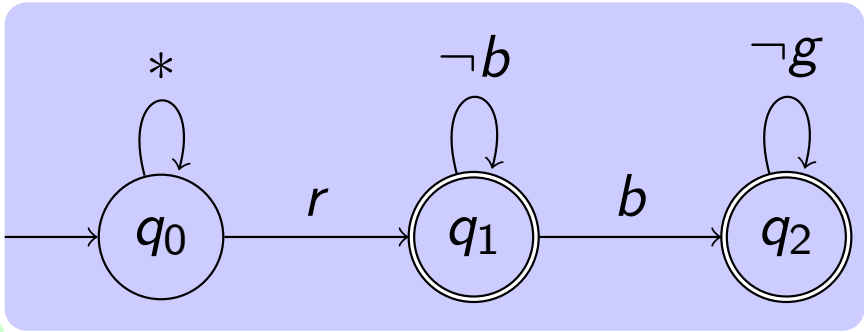
Assuming 4 is the tightest bound, we can label the states of the FSM by for which Büchi automaton state an incoming run with 4 visits to rejecting states so far can be found.

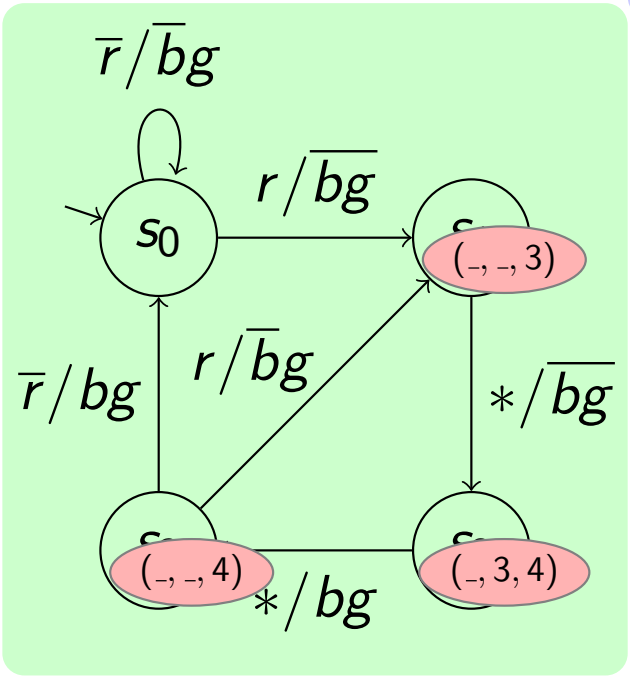
SMT-Based Bounded Synthesis – Starting observation



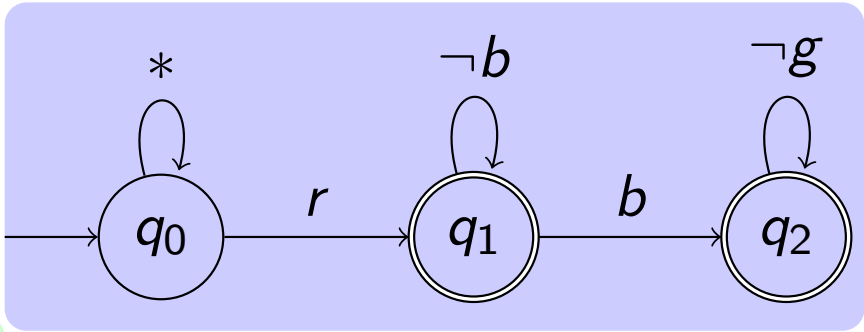


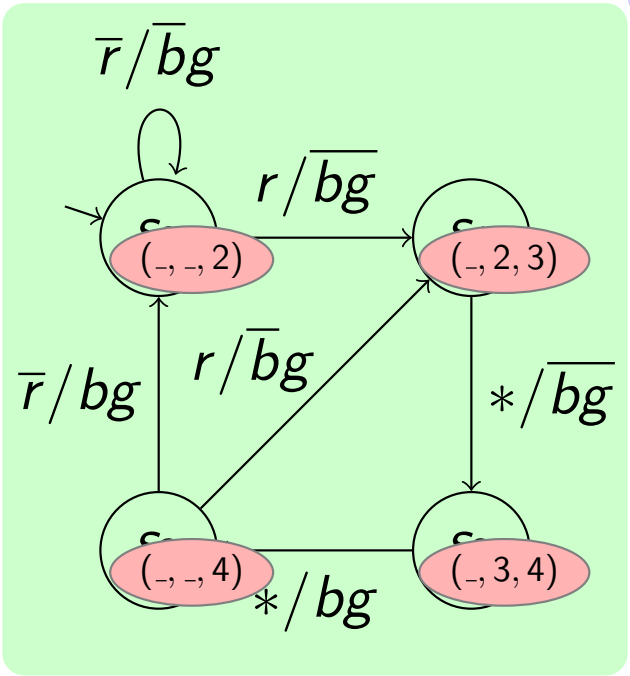
SMT-Based Bounded Synthesis – Starting observation



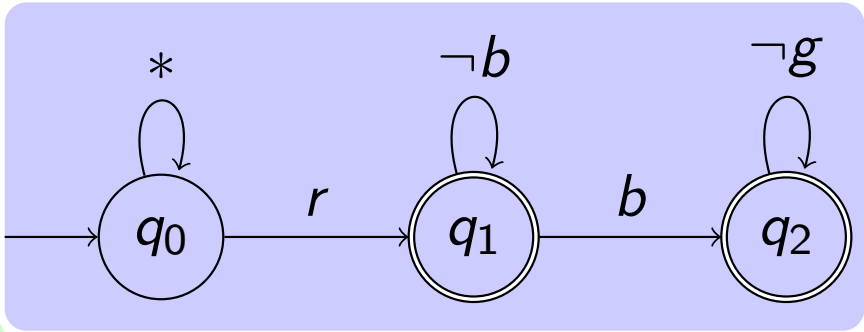


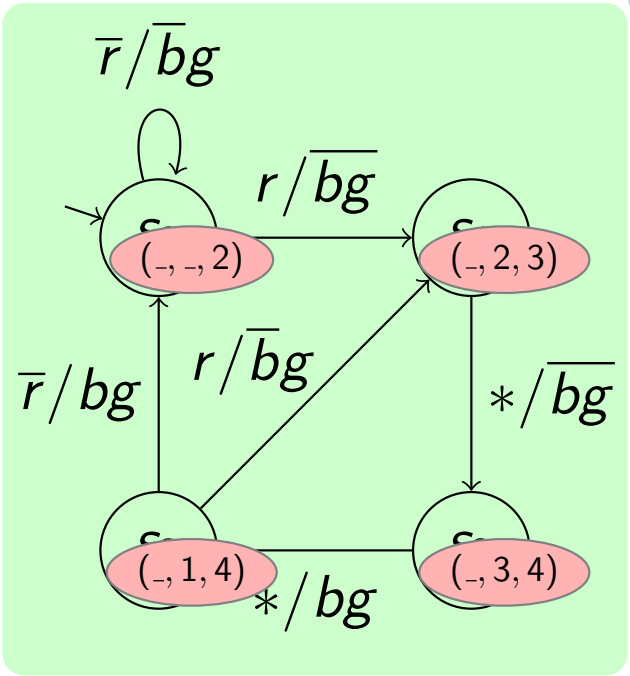
SMT-Based Bounded Synthesis – Starting observation



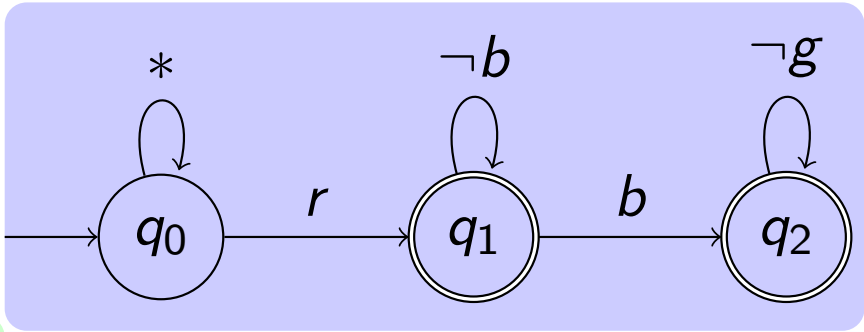


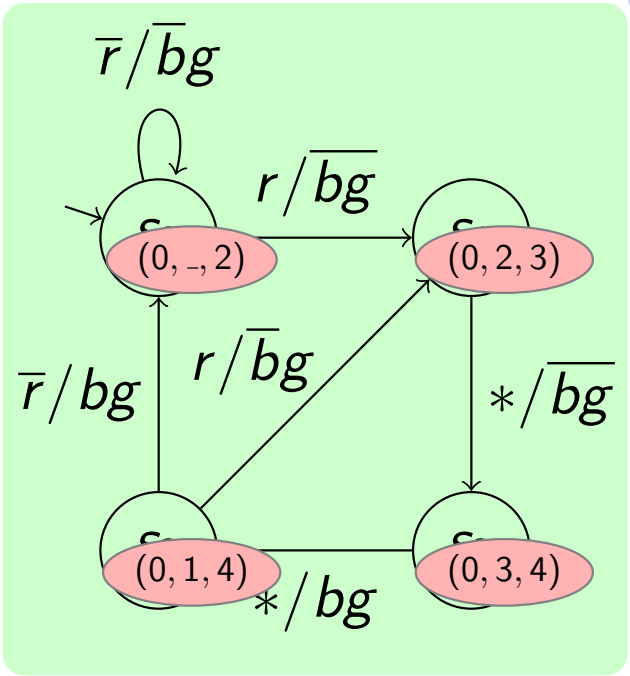
SMT-Based Bounded Synthesis – Starting observation



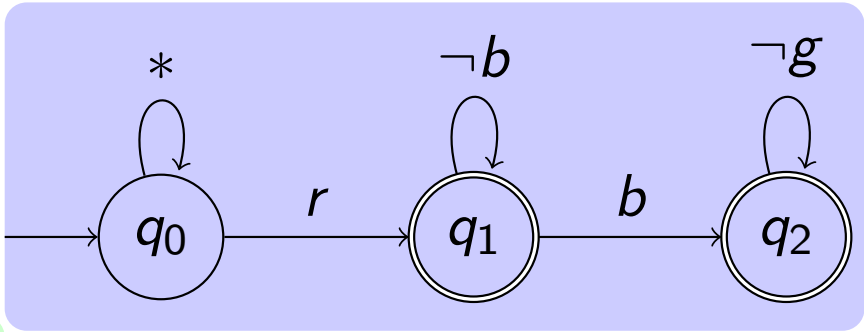


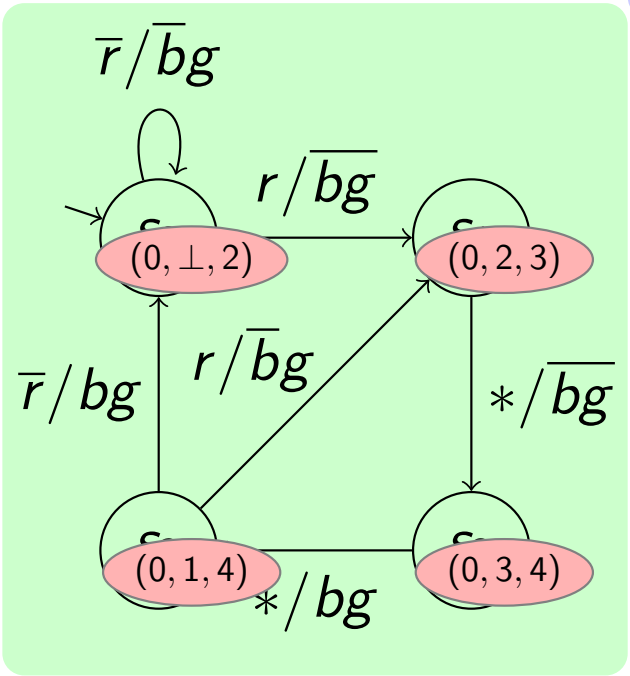
SMT-Based Bounded Synthesis – Starting observation





SMT-Based Bounded Synthesis – Starting observation





Main observation used in SMT-based bounded synthesis

Solution labeling

If and only if a finite-state machine is accepted by a UCW, we can label the states by counters that represent the maximum number of visits to rejecting states in the product automaton along any run, and these counters have an upper bound.

Main observation used in SMT-based bounded synthesis

Solution labeling

If and only if a finite-state machine is accepted by a UCW, we can label the states by counters that represent the maximum number of visits to rejecting states in the product automaton along any run, and these counters have an upper bound.

New idea

We can ask an *SMT solver* to search for a FSM (with $c \in \mathbb{N}$ many states) **together** with such a solution labelling. The resulting SMT instance is satisfiable *if and only if* there exists a correct implementation with c many states.

SMT in a nutshell

Input

- Variable declarations (e.g., real-value variables, integer variables, Boolean variables)
- Constraints (Boolean constraints, linear constraints, Boolean combinations of linear constraints, ...)

SMT in a nutshell

Input

- Variable declarations (e.g., real-value variables, integer variables, Boolean variables)
- Constraints (Boolean constraints, linear constraints, Boolean combinations of linear constraints, ...)

Output

Either:

- A valuation to all variables that satisfies all constraints
- A notification that no such variable valuation exists

SMT Example

$$(x \geq 5 \vee y \geq 10) \wedge (x - y \leq -1 \vee x \geq 9) \wedge (y - x \leq 4 \vee x \leq 3 \vee b)$$

SMT Example

$$(x \geq 5 \vee y \geq 10) \wedge (x - y \leq -1 \vee x \geq 9) \wedge (y - x \leq 4 \vee x \leq 3 \vee b)$$

$$\rightarrow \text{Satisfiable: } x = 9,\ y = 9,\ b = \textbf{false}$$

SMT-based bounded synthesis (for Moore machines)

Starting point

UVW $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ where $\Sigma = \Sigma_I \times \Sigma_O$, and a state number bound $c \in \mathbb{N}$.

SMT-based bounded synthesis (for Moore machines)

Starting point

UVW $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ where $\Sigma = \Sigma_I \times \Sigma_O$, and a state number bound $c \in \mathbb{N}$.

First set of variables

- $u_{(i,x,j)}$ represent for $i, j \in \{1, \ldots, c\}$ and $x \in \Sigma_I$ whether the FSM transitions from state no. i to state no. j when reading an x
- $v_{(i,y)}$ represent for $i \in \{1, \ldots, c\}$ whether state no. i is labeled by $y \in \Sigma_O$.

SMT-based bounded synthesis (for Moore machines)

Starting point

UVW $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ where $\Sigma = \Sigma_I \times \Sigma_O$, and a state number bound $c \in \mathbb{N}$.

First set of variables

- $u_{(i,x,j)}$ represent for $i, j \in \{1, \ldots, c\}$ and $x \in \Sigma_I$ whether the FSM transitions from state no. i to state no. j when reading an x
- $v_{(i,y)}$ represent for $i \in \{1, \ldots, c\}$ whether state no. i is labeled by $y \in \Sigma_O$.

First set of constraints

$$\bigwedge_{i,\in\{1,\ldots,c\},x\in\Sigma_I} \bigvee_{j\in\{1,\ldots,c\}} u_{(i,x,j)}$$

$$\bigwedge_{i,j,j'\in\{1,\ldots,c\},j\neq j',x\in\Sigma_I} (\neg u_{i,x,j} \vee \neg u_{i,x,j'})$$

SMT-based bounded synthesis (for Moore machines)

Second set of constraints

$$\bigwedge_{i,\in\{1,\ldots,c\}} \bigvee_{y\in\Sigma_O} v_{(i,x)}$$

$$\bigwedge_{i\in\{1,\ldots,c\},y,y'\in\Sigma_O,y\neq y'} (\neg v_{i,y} \vee \neg v_{i,y'})$$

SMT-based bounded synthesis (for Moore machines)

Second set of variables

- $d_{(i,q)} \in \{-1, 0, 1, \ldots\}$ represent the counter value for visits to rejecting states for $i \in \{1, \ldots, c\}$ and $q \in Q$. The value of -1 represents the $\perp$ special case.

SMT-based bounded synthesis (for Moore machines)

Second set of variables

- $d_{(i,q)} \in \{-1, 0, 1, \ldots\}$ represent the counter value for visits to rejecting states for $i \in \{1, \ldots, c\}$ and $q \in Q$. The value of -1 represents the $\perp$ special case.

Sane initial marking

- $d_{0,q_0} \geq 0$

SMT-based bounded synthesis (for Moore machines)

Second set of variables

- $d_{(i,q)} \in \{-1, 0, 1, \ldots\}$ represent the counter value for visits to rejecting states for $i \in \{1, \ldots, c\}$ and $q \in Q$. The value of -1 represents the $\bot$ special case.

Sane initial marking

- $d_{0,q_0} \geq 0$

Correct successor marking

$$\forall_{i,j \in \{1,\ldots,c\}, \forall x \in \Sigma_I, q \in Q, y \in \Sigma_O, q' \in \delta(q,(x,y))} (u_{i,x,j} \wedge v_{i,y} \rightarrow d_{j,q'} \geq d_{i,q} + \mathbb{1}_F(q'))$$

SMT-based bounded synthesis – Demo

https://www.react.uni-saarland.de/tools/online/BoSy/

Bounded synthesis – Discussion

Basic ideas

- We don't want to determinize the specification automaton
- We are fine with bounding some property of the solution:
  - Number of visits to rejecting automaton states along any execution of the solution
  - Number of states of the implementation automaton

Bounded synthesis – Discussion

Basic ideas

- We don't want to determinize the specification automaton
- We are fine with bounding some property of the solution:
  - Number of visits to rejecting automaton states along any execution of the solution
  - Number of states of the implementation automaton

Advantages

- Simple!
- State-of-the-art in terms of efficiency (for full LTL)
- The non-SMT variant can be combined with a *plant model* of the plant to be controlled
- The SMT variant can leverage the power of modern SMT solvers (and can be used for *distributed synthesis*)

Bounded synthesis – Discussion

Disadvantages

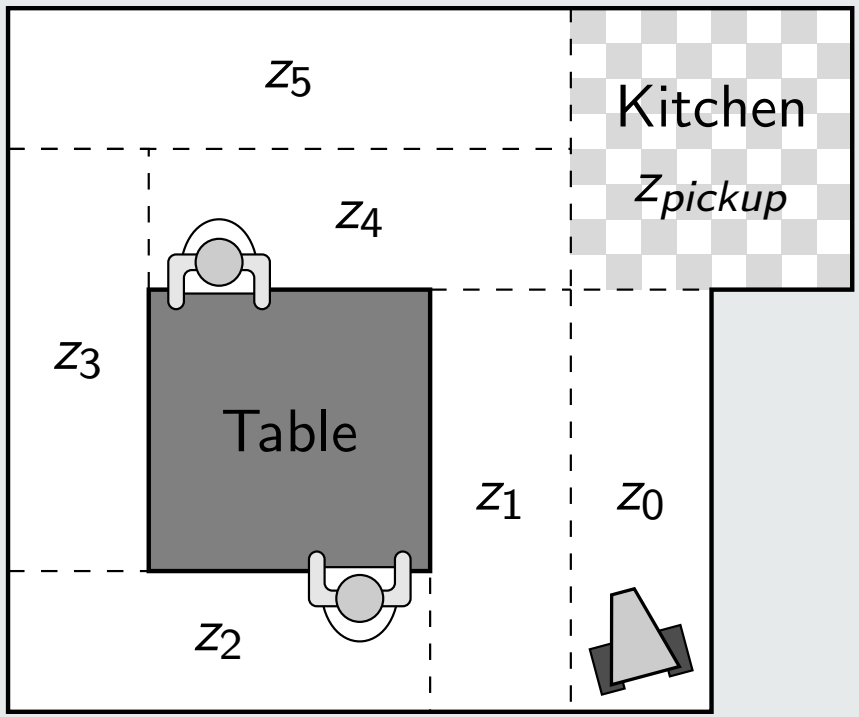
- Selected bound may be too small
- Detecting unrealizability requires running multiple synthesis processes in parallel
- Still less efficient than GR(1) synthesis (coming up next)!

Bounded synthesis and CPS

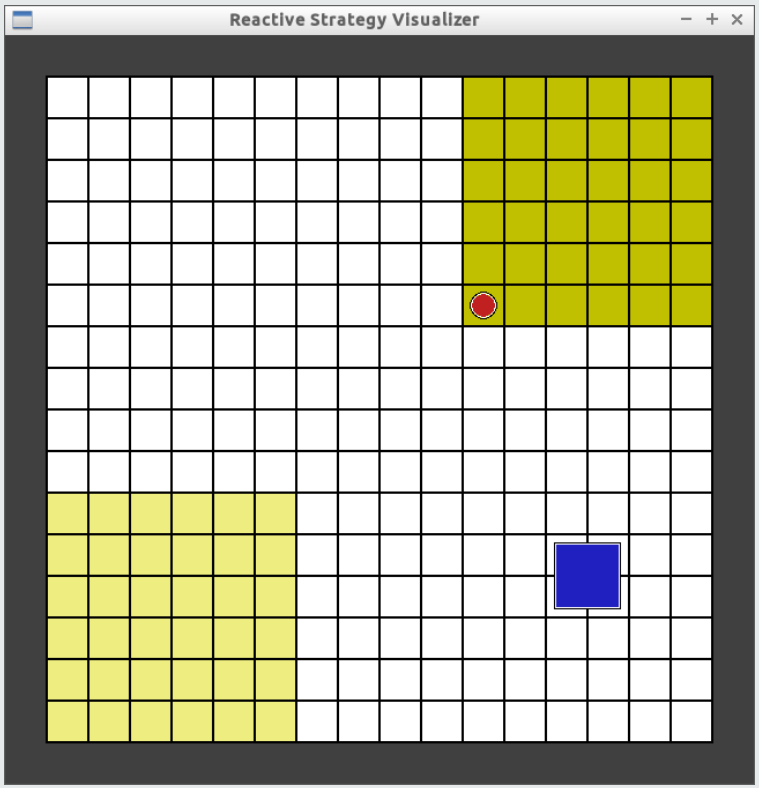
What about CPS?

- Good for CPS if we want to enforce **complex temporal properties**, but have a simple environment abstraction

Good



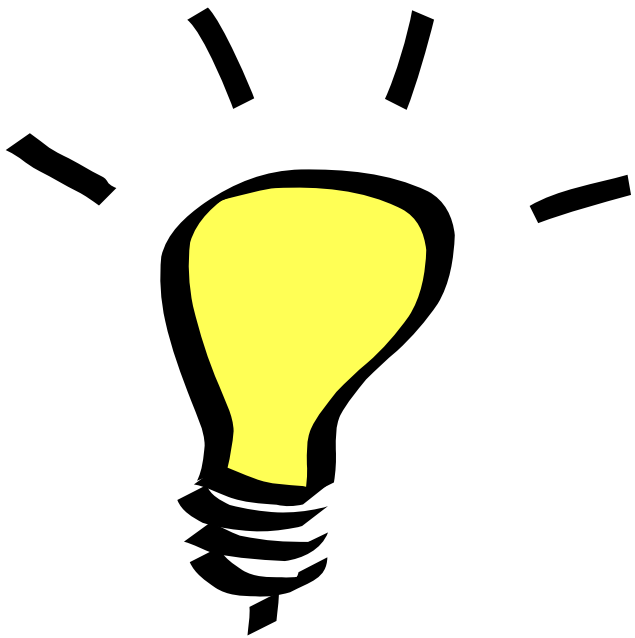
Bad



Bounded synthesis – Conclusion

Summary

- A state-of-the-art synthesis approach
- Circumvents determinization by the introduction of some type of *bound*
- Two variants are available (also as tools): one with safety game solving and one based on SMT solving
- Useful for CPS? That depends on the application.



References I

Emmanuel Filiot, Naiyong Jin, and Jean-François Raskin. Compositional algorithms for LTL synthesis. In *Automated Technology for Verification and Analysis - 8th International Symposium, ATVA 2010, Singapore, September 21-24, 2010. Proceedings*, pages 112–127, 2010. doi: 10.1007/978-3-642-15643-4_10. URL `https://doi.org/10.1007/978-3-642-15643-4_10`.

Bernd Finkbeiner and Sven Schewe. Bounded synthesis. *STTT*, 15(5-6):519–539, 2013. doi: 10.1007/s10009-012-0228-z. URL `https://doi.org/10.1007/s10009-012-0228-z`.