Practical Reactive Synthesis

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

Rüdiger Ehlers, University of Bremen

September 2017

Topics of this lecture

Lectures:

1. From the Theory to the Practice of Reactive Synthesis – Overview of the Challenges
2. Bounded Synthesis
3. Generalized Reactivity(1) Synthesis
4. Symbolic Game Solving

Laboratory:

1. GR(1) Synthesis for (Discrete) CPS Control

Abstract

While the seminal works on reactive synthesis were already published in the 70's, only recently, researchers began to look at how synthesis can be made more practical. For most specification logics, synthesis has a high computational complexity. Hence, the key issue is to avoid this complexity, either by using a simpler specification logic, or by employing a synthesis approach that is able to make use of specification properties that allow an efficient synthesis process and that are commonly found in specifications from practice.

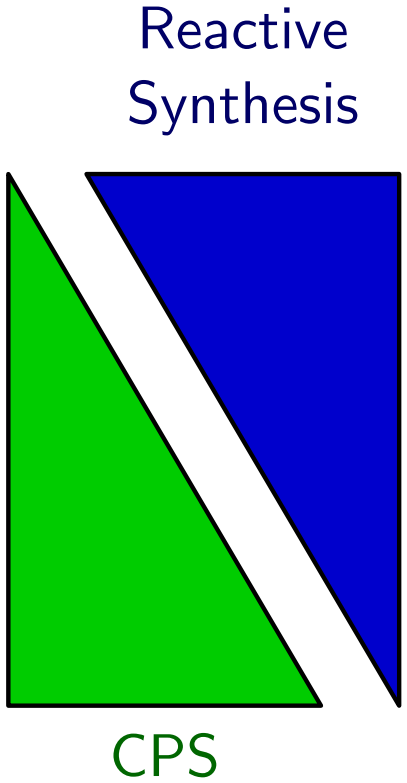
In this lecture, we will review such approaches and discuss how they can be applied in the presence of a plant model, as it is commonly the case in cyber-physical system control. A few practical examples will round off the lecture.

Categorization of the (technical) topics

Bounded Synthesis

Generalized Reactivity(1) Synthesis

Symbolic Game Solving

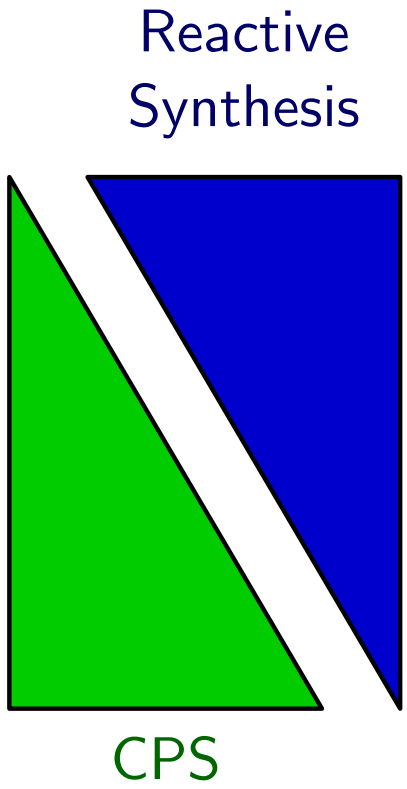


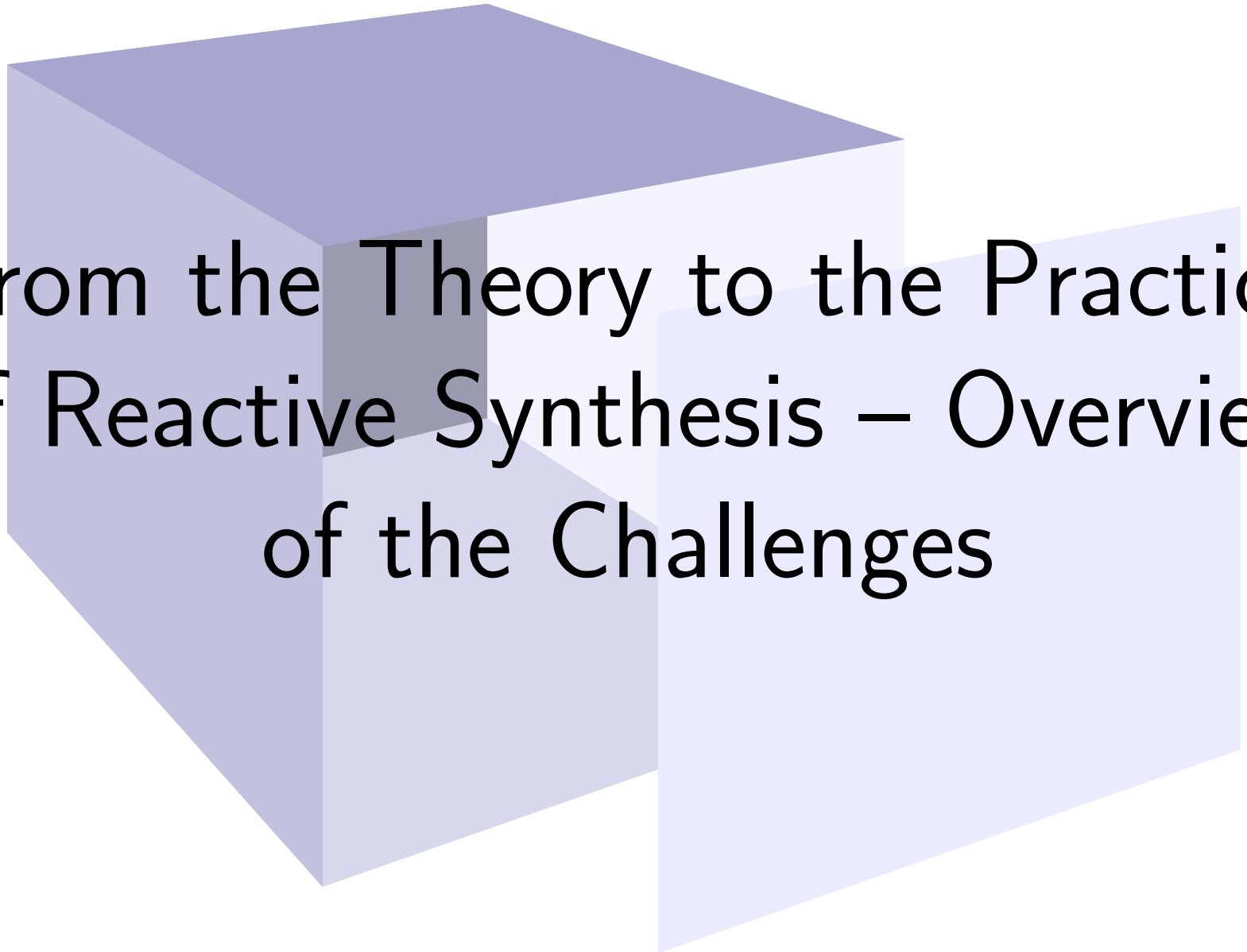
Categorization of the (technical) topics

Bounded Synthesis

Generalized Reactivity(1) Synthesis

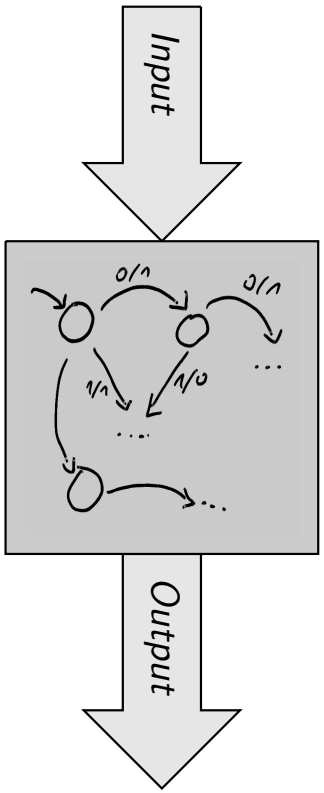
Symbolic Game Solving





From the Theory to the Practice of Reactive Synthesis – Overview of the Challenges

Verification:



$+ \quad \mathbf{G}(r \rightarrow \mathbf{X}g) \quad \Rightarrow$

Satisfied

Not satisfied
(+ counter-example)

Synthesis:

$$\mathbf{G}(r \rightarrow \mathbf{X}g)$$

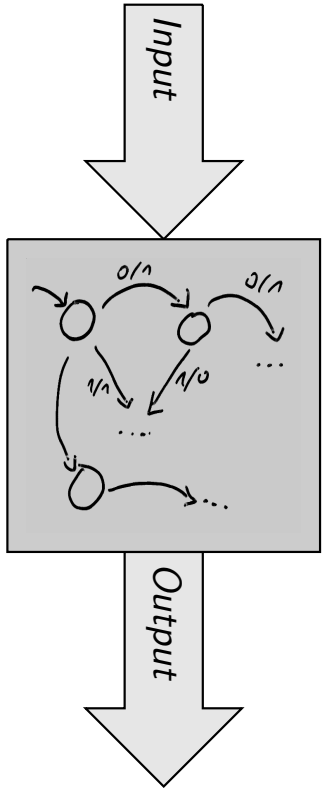
$$+$$

$$\text{Input} = \{r, \ldots\}$$
$$\text{Output} = \{g, \ldots\}$$

$\Rightarrow$

Realizable $\longrightarrow$

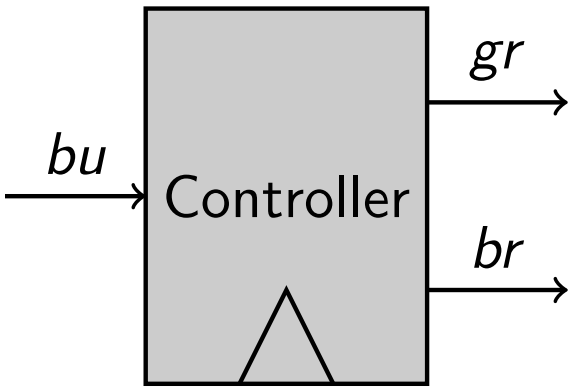
Not realizable



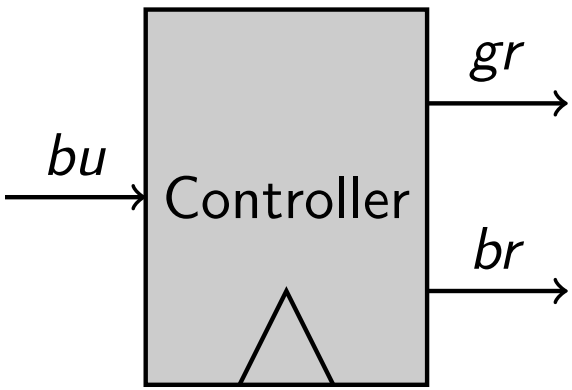
Synthesis of reactive systems - example

Atomic propositions

- $AP_I = \{button\}$
- $AP_O = \{grind, brew\}$



Synthesis of reactive systems - example



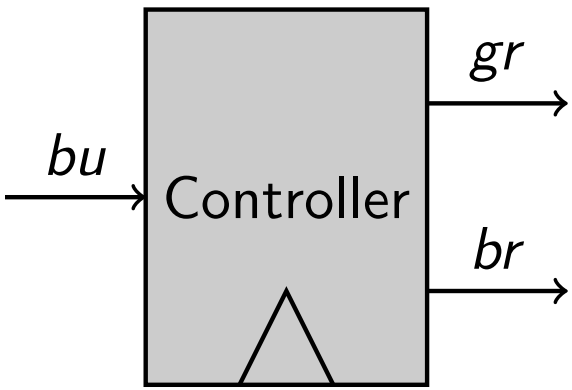
Atomic propositions

- $\mathrm{AP}_I = \{\mathit{button}\}$
- $\mathrm{AP}_O = \{\mathit{grind}, \mathit{brew}\}$

A run of the system

$$\rho = \begin{pmatrix}0\\0\\0\end{pmatrix}\begin{pmatrix}1\\1\\0\end{pmatrix}\begin{pmatrix}0\\1\\0\end{pmatrix}\begin{pmatrix}0\\0\\1\end{pmatrix}\begin{pmatrix}0\\0\\1\end{pmatrix}\begin{pmatrix}0\\0\\1\end{pmatrix}\begin{pmatrix}0\\0\\0\end{pmatrix}\ldots$$

Synthesis of reactive systems - example



Atomic propositions

- $\mathrm{AP}_I = \{\mathit{button}\}$
- $\mathrm{AP}_O = \{\mathit{grind}, \mathit{brew}\}$

A run of the system

$$\rho = \begin{pmatrix}0\\0\\0\end{pmatrix}\begin{pmatrix}1\\1\\0\end{pmatrix}\begin{pmatrix}0\\1\\0\end{pmatrix}\begin{pmatrix}0\\0\\1\end{pmatrix}\begin{pmatrix}0\\0\\1\end{pmatrix}\begin{pmatrix}0\\0\\1\end{pmatrix}\begin{pmatrix}0\\0\\0\end{pmatrix}\ldots$$

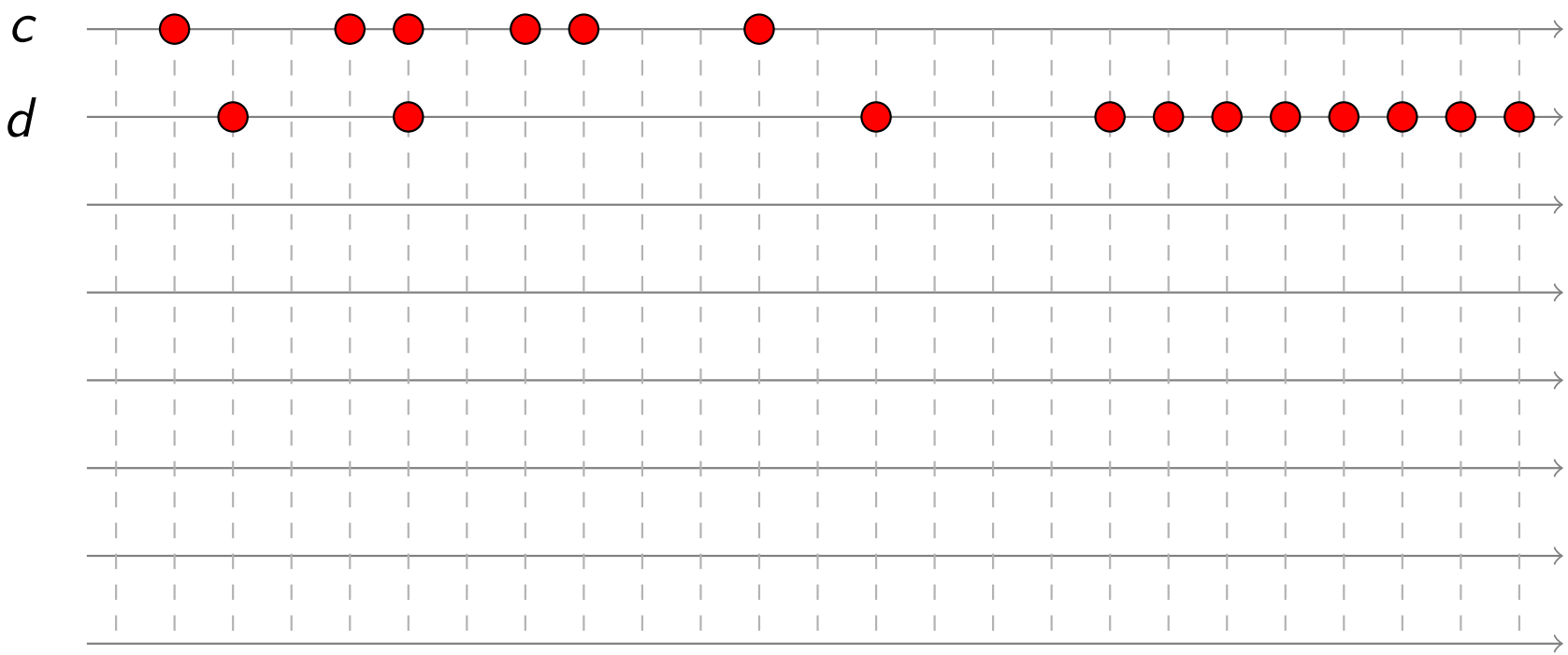
Specification

Whenever the user presses the button, the grinding unit should be activated for the next 2 steps. After that, the grinding module should be inactive while the brewing unit brews for the next 3 steps.

Preliminiaries: Linear Temporal Logic (LTL)

Basics

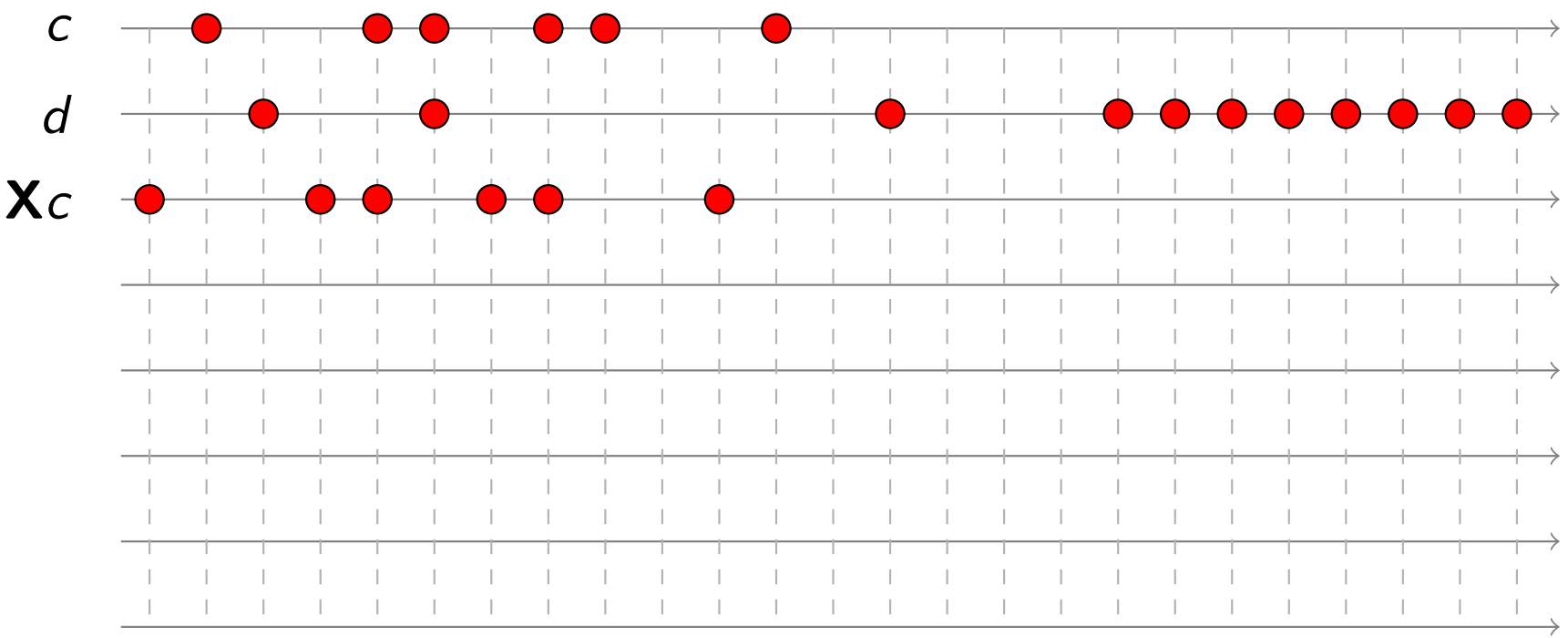
- LTL formulas are defined over some set of atomic propositions AP
- Grammar: $\varphi ::= p \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \varphi \mathcal{U} \varphi \mid \mathbf{G}\varphi \mid \mathbf{F}\varphi \mid \mathbf{X}\varphi$



Preliminiaries: Linear Temporal Logic (LTL)

Basics

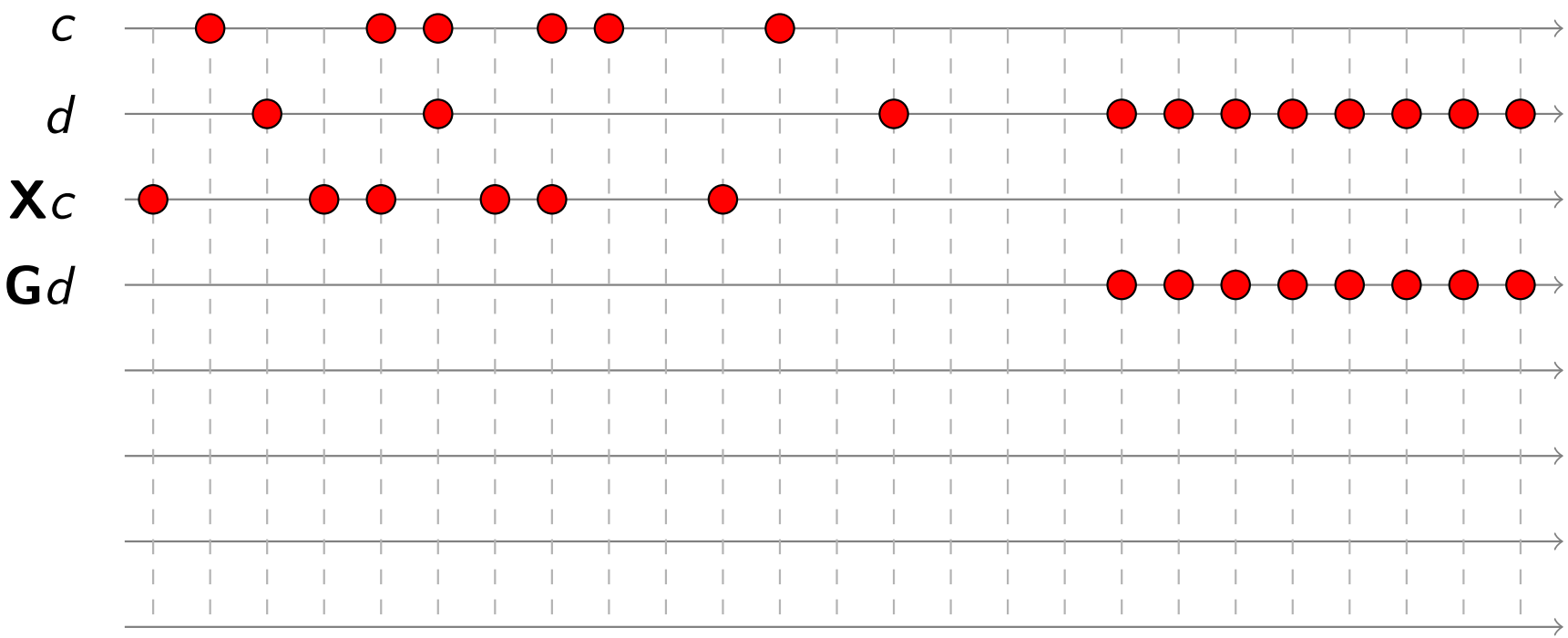
- LTL formulas are defined over some set of atomic propositions AP
- Grammar: $\varphi ::= p \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \varphi \mathcal{U} \varphi \mid \mathbf{G}\varphi \mid \mathbf{F}\varphi \mid \mathbf{X}\varphi$



Preliminiaries: Linear Temporal Logic (LTL)

Basics

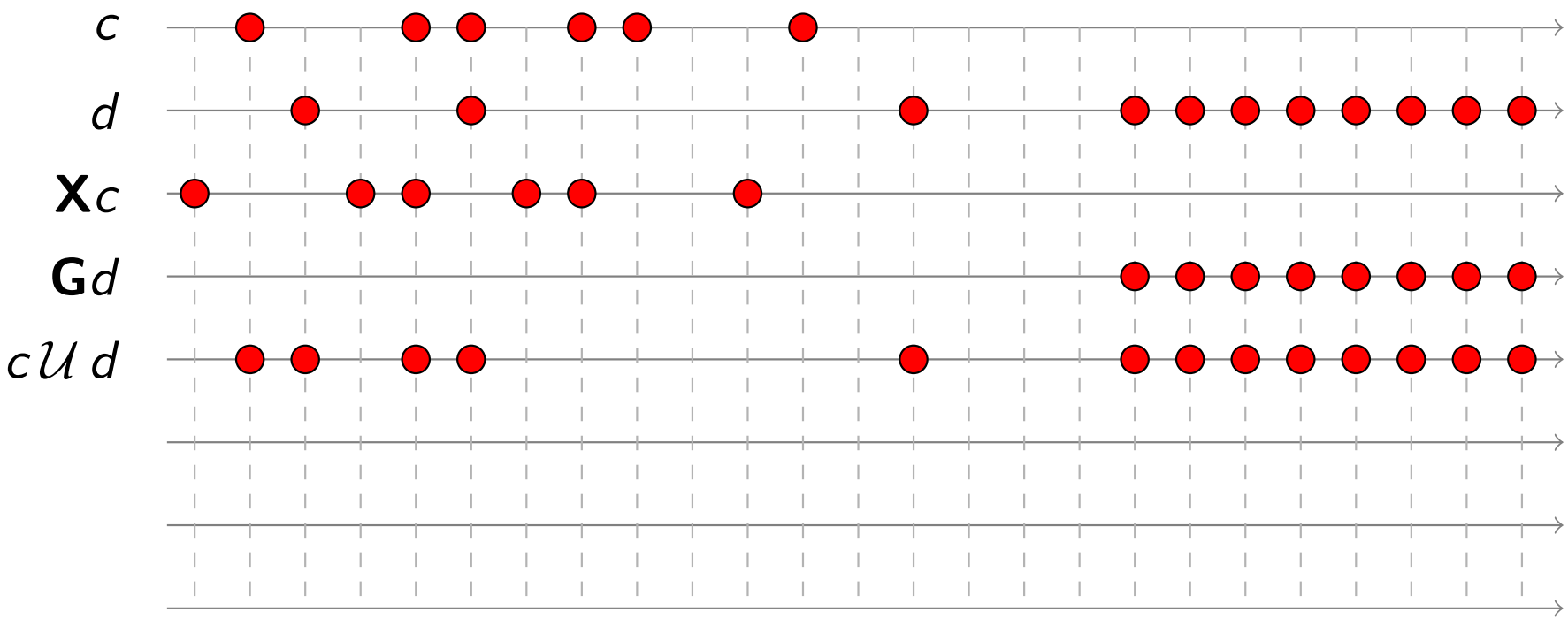
- LTL formulas are defined over some set of atomic propositions AP
- Grammar: $\varphi ::= p \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \varphi \mathcal{U} \varphi \mid \mathbf{G}\varphi \mid \mathbf{F}\varphi \mid \mathbf{X}\varphi$



Preliminiaries: Linear Temporal Logic (LTL)

Basics

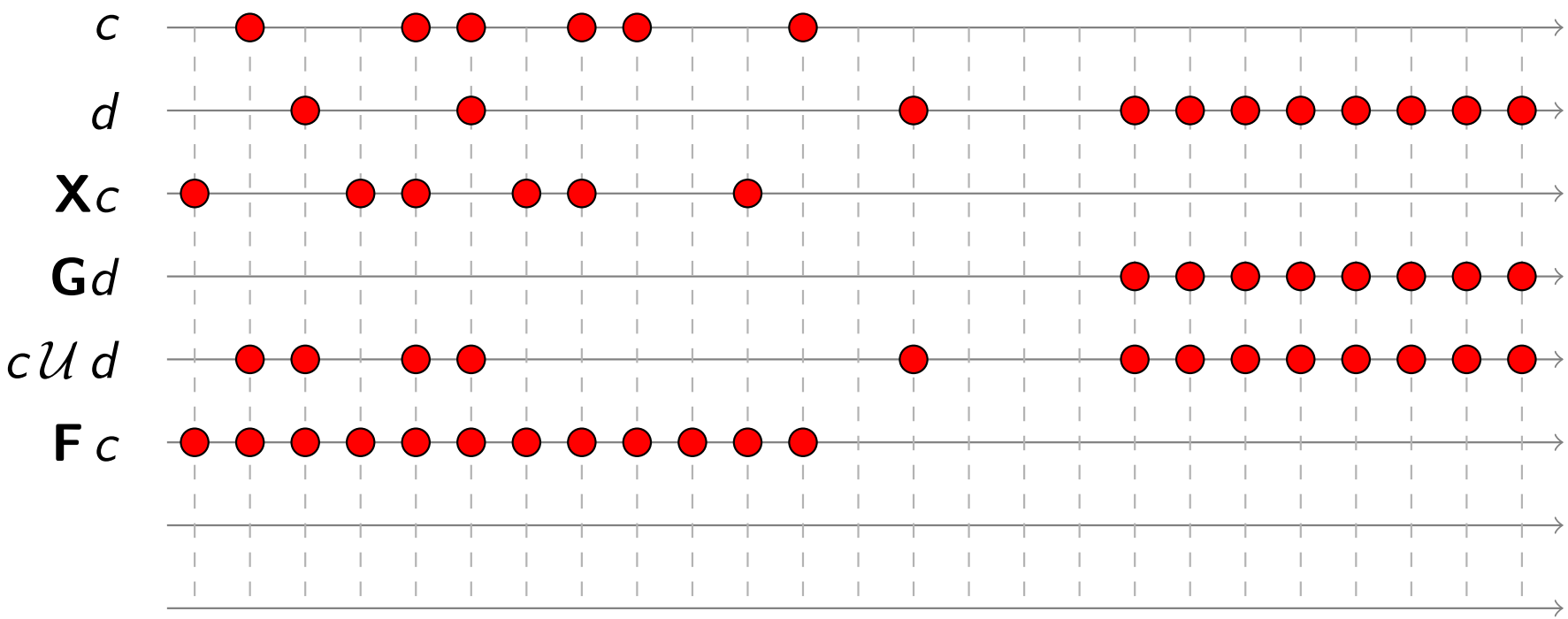
- LTL formulas are defined over some set of atomic propositions AP
- Grammar: $\varphi ::= p \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \varphi \mathcal{U} \varphi \mid \mathbf{G}\varphi \mid \mathbf{F}\varphi \mid \mathbf{X}\varphi$



Preliminiaries: Linear Temporal Logic (LTL)

Basics

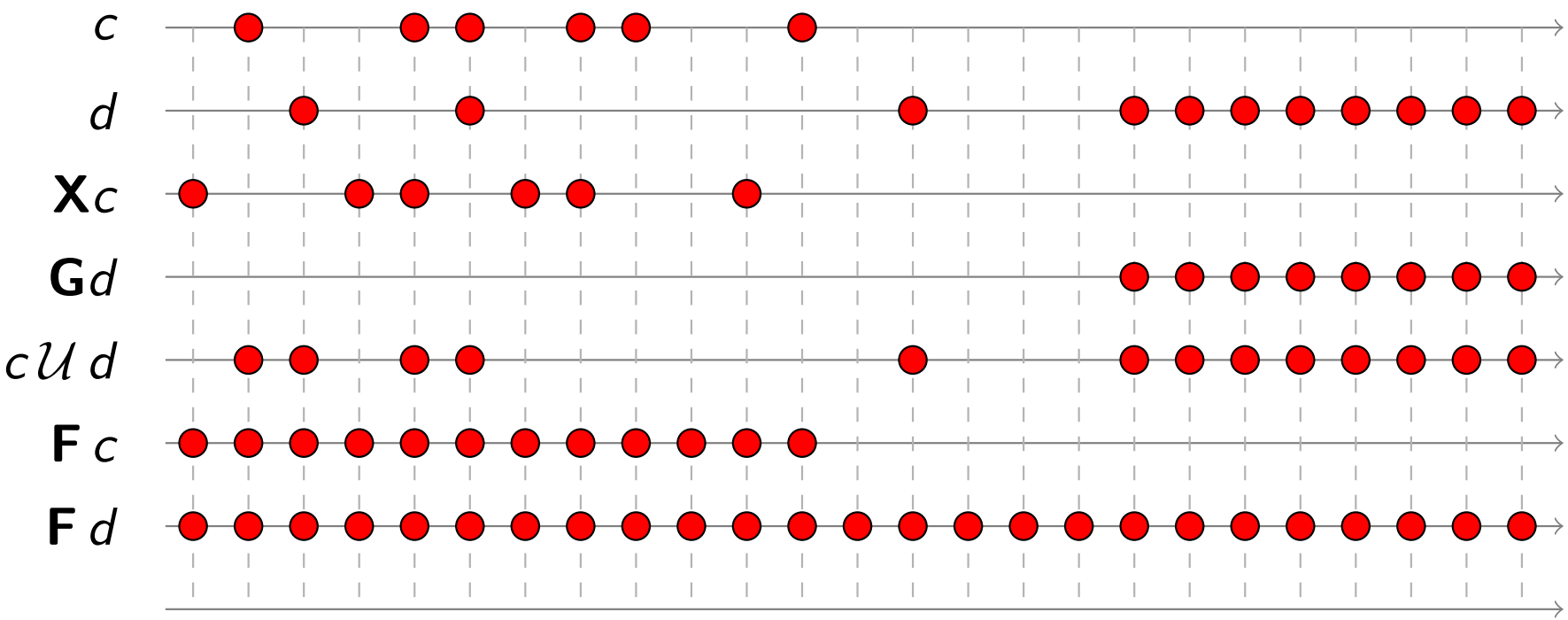
- LTL formulas are defined over some set of atomic propositions AP
- Grammar: $\varphi ::= p \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \varphi \mathcal{U} \varphi \mid \mathbf{G}\varphi \mid \mathbf{F}\varphi \mid \mathbf{X}\varphi$



Preliminiaries: Linear Temporal Logic (LTL)

Basics

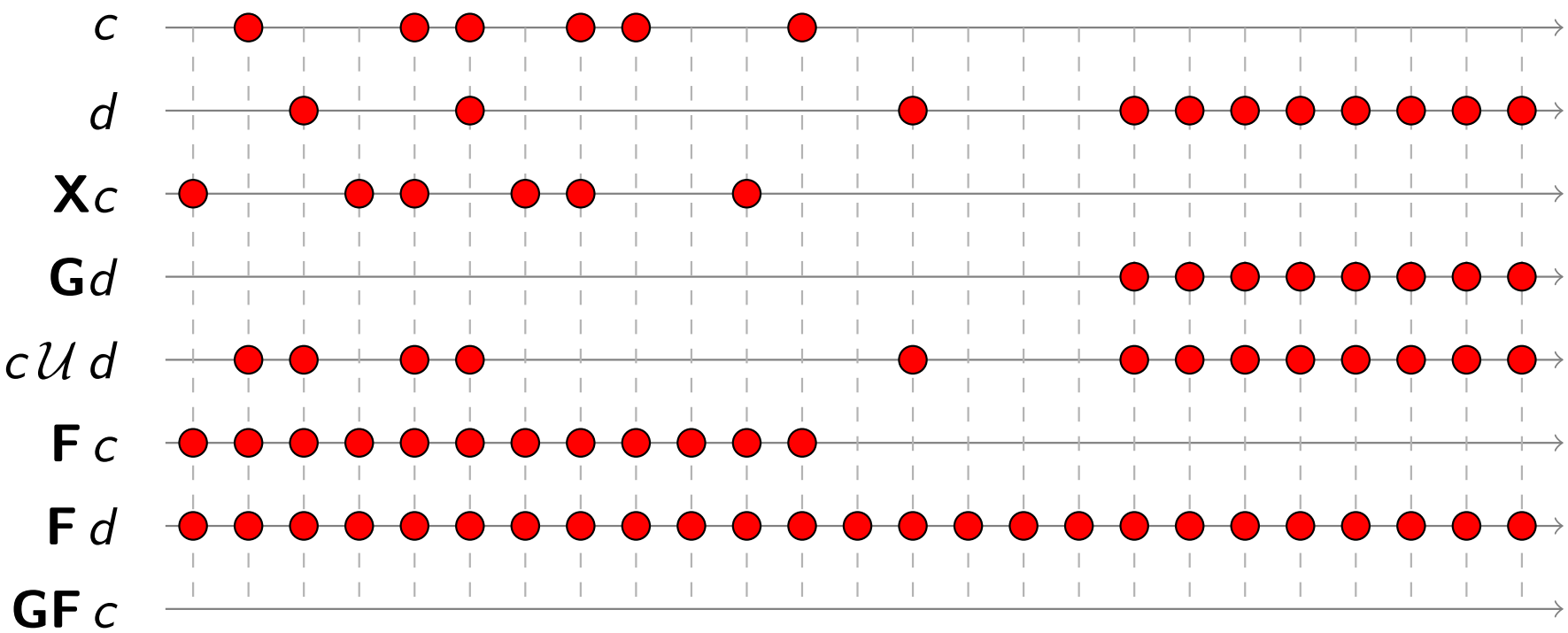
- LTL formulas are defined over some set of atomic propositions AP
- Grammar: $\varphi ::= p \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \varphi \mathcal{U} \varphi \mid \mathbf{G}\varphi \mid \mathbf{F}\varphi \mid \mathbf{X}\varphi$



Preliminiaries: Linear Temporal Logic (LTL)

Basics

- LTL formulas are defined over some set of atomic propositions AP
- Grammar: $\varphi ::= p \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \varphi \mathcal{U} \varphi \mid \mathbf{G}\varphi \mid \mathbf{F}\varphi \mid \mathbf{X}\varphi$



What do we want as a result?

The computed implementation should be...

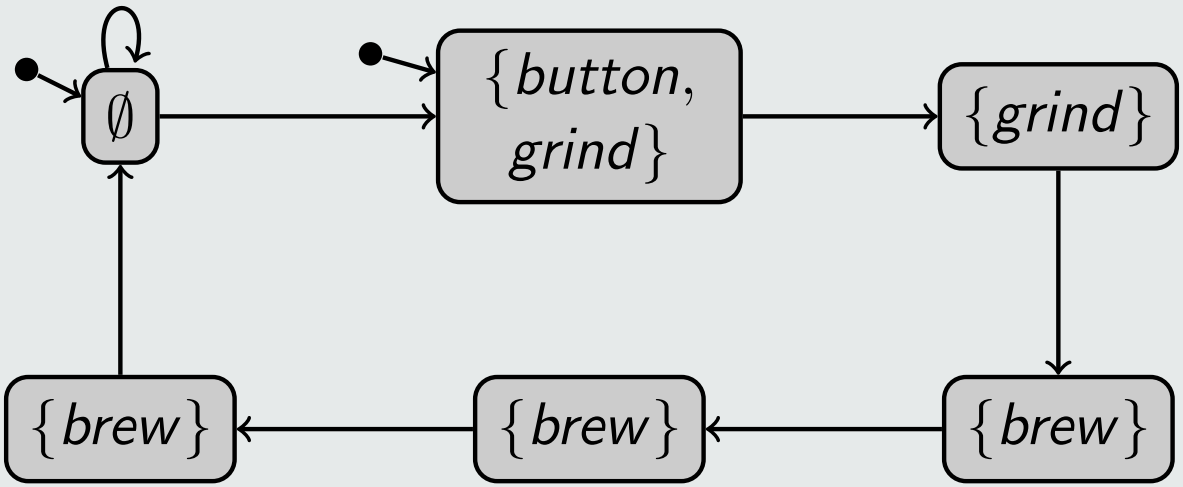
- ...finite-state,
- ...deterministic, and
- ...non-terminating and input-responsive

What do we want as a result?

The computed implementation should be...

- ...finite-state,
- ...deterministic, and
- ...non-terminating and input-responsive

Kripke structures

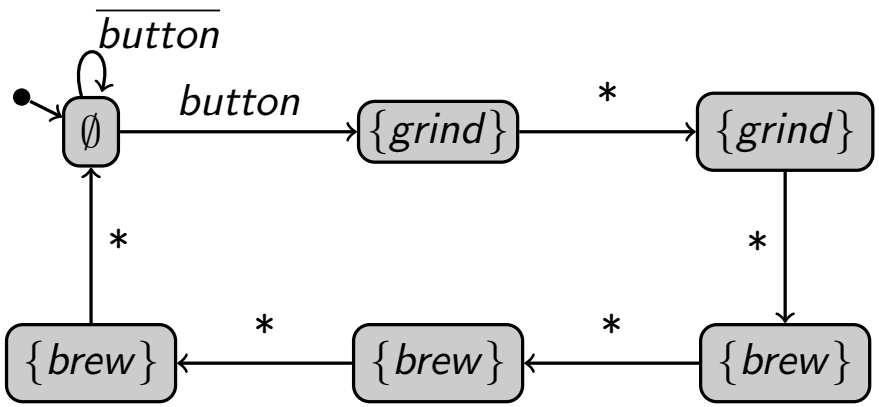


Computation models

Moore machines

$\mathcal{M} = (S, \Sigma^I, \Sigma^O, s_0, \delta, L)$ with:

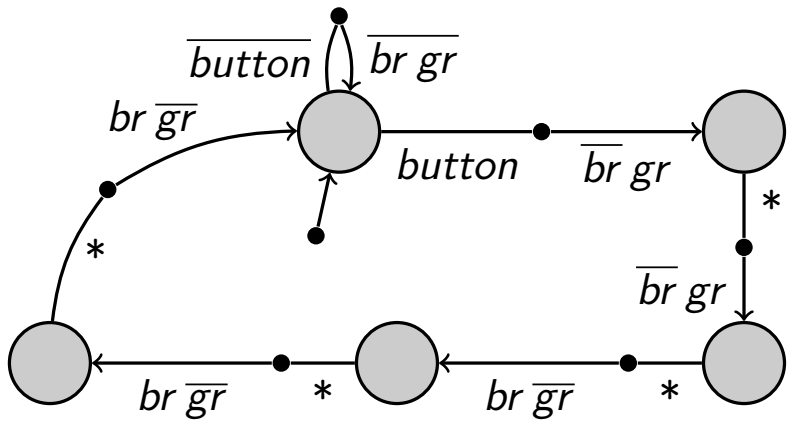
- Set of states S
- Input/output alphabets Σ^I / Σ^O
- Initial state s_0
- Transition function $\delta : S \times \Sigma^I \to S$
- State labeling: $L : S \to \Sigma^O$



Mealy machines

$\mathcal{M} = (S, \Sigma^I, \Sigma^O, s_0, \delta)$ with:

- Set of states S
- Input/output alphabets Σ^I / Σ^O
- Initial state s_0
- Transition function $\delta : S \times \Sigma^I \to S \times \Sigma^O$



Synthesis of reactive systems - formalizing the example

Informal specification

Whenever the user presses the button, the grinding unit should be activated for the next 2 steps. After that, the grinding module should be inactive while the brewing unit brews for the next 3 steps.

Synthesis of reactive systems - formalizing the example

Informal specification

Whenever the user presses the button, the grinding unit should be activated for the next 2 steps. After that, the grinding module should be inactive while the brewing unit brews for the next 3 steps.

Formal specification in linear-time temporal logic (LTL)

$$\mathbf{G}(\mathit{button} \rightarrow (\mathit{grind} \wedge \mathbf{X}\,\mathit{grind} \wedge \mathbf{XX}(\mathit{brew} \wedge \neg\mathit{grind}) \wedge \mathbf{XXX}(\mathit{brew} \wedge \neg\mathit{grind}) \wedge \mathbf{XXXX}(\mathit{brew} \wedge \neg\mathit{grind})))$$

Synthesis of reactive systems - analyzing the example

Formal specification in linear-time temporal logic (LTL)

$$\mathbf{G}(\mathit{button} \rightarrow (\mathit{grind} \wedge \mathbf{X}\,\mathit{grind} \wedge \mathbf{XX}\,(\mathit{brew} \wedge \neg \mathit{grind}) \wedge \mathbf{XXX}\,(\mathit{brew} \wedge \neg \mathit{grind}) \wedge \mathbf{XXXX}\,(\mathit{brew} \wedge \neg \mathit{grind})))$$

A surprise

The specification is *unrealizable*.

Example:

$$\rho = \begin{pmatrix}0\\0\\0\end{pmatrix}\begin{pmatrix}1\\1\\0\end{pmatrix}\begin{pmatrix}1\\1\\0\end{pmatrix}\begin{pmatrix}0\\???\\1\end{pmatrix}\ldots$$

Fixing the specification

Idea

Let us rewrite the specification such that button presses are only considered if the machine did not do anything previously

New formal specification in linear-time temporal logic (LTL)

$$\mathbf{G}((\neg grind \wedge \neg brew) \rightarrow \mathbf{X}(button \rightarrow (grind \ \wedge \ \mathbf{X}\, grind \ \wedge \ \mathbf{XX}\,(brew \wedge \neg grind) \wedge \mathbf{XXX}\,(brew \wedge \neg grind) \ \wedge \ \mathbf{XXXX}\,(brew \wedge \neg grind))))$$

Fixing the specification

Idea

Let us rewrite the specification such that button presses are only considered if the machine did not do anything previously

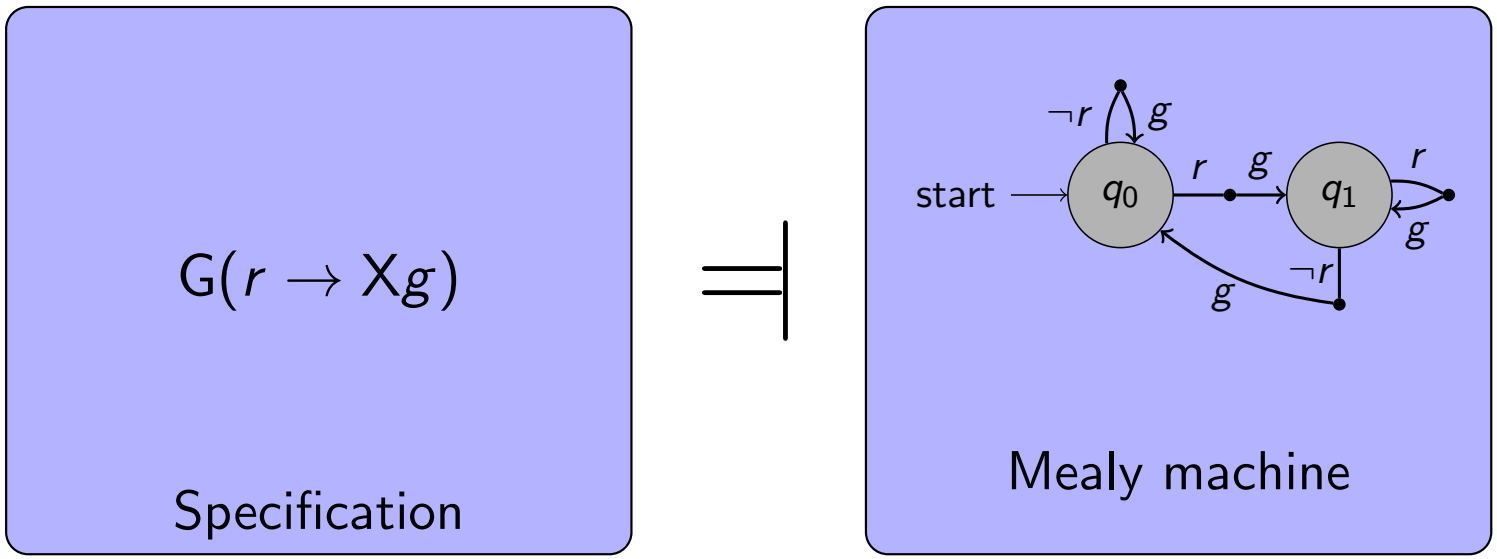
New formal specification in linear-time temporal logic (LTL)

$$\mathbf{G}((\neg grind \wedge \neg brew) \rightarrow \mathbf{X}(button \rightarrow (grind \ \wedge \ \mathbf{X}\, grind \ \wedge \ \mathbf{XX}\,(brew \wedge \neg grind) \wedge \mathbf{XXX}\,(brew \wedge \neg grind) \ \wedge \ \mathbf{XXXX}\,(brew \wedge \neg grind))))$$

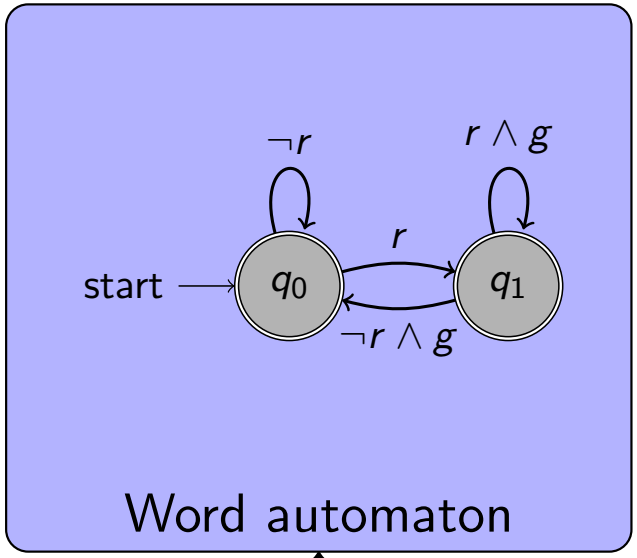
Result

This one is realizable $\rightarrow$ Demo!

General synthesis workflow



General synthesis workflow

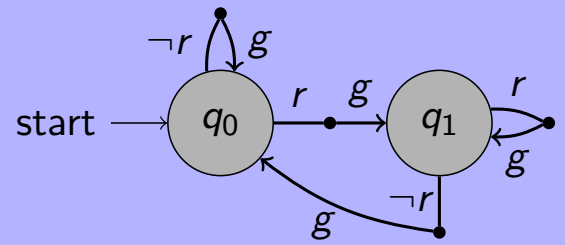


Word automaton

$\mathsf{G}(r \rightarrow \mathsf{X}g)$

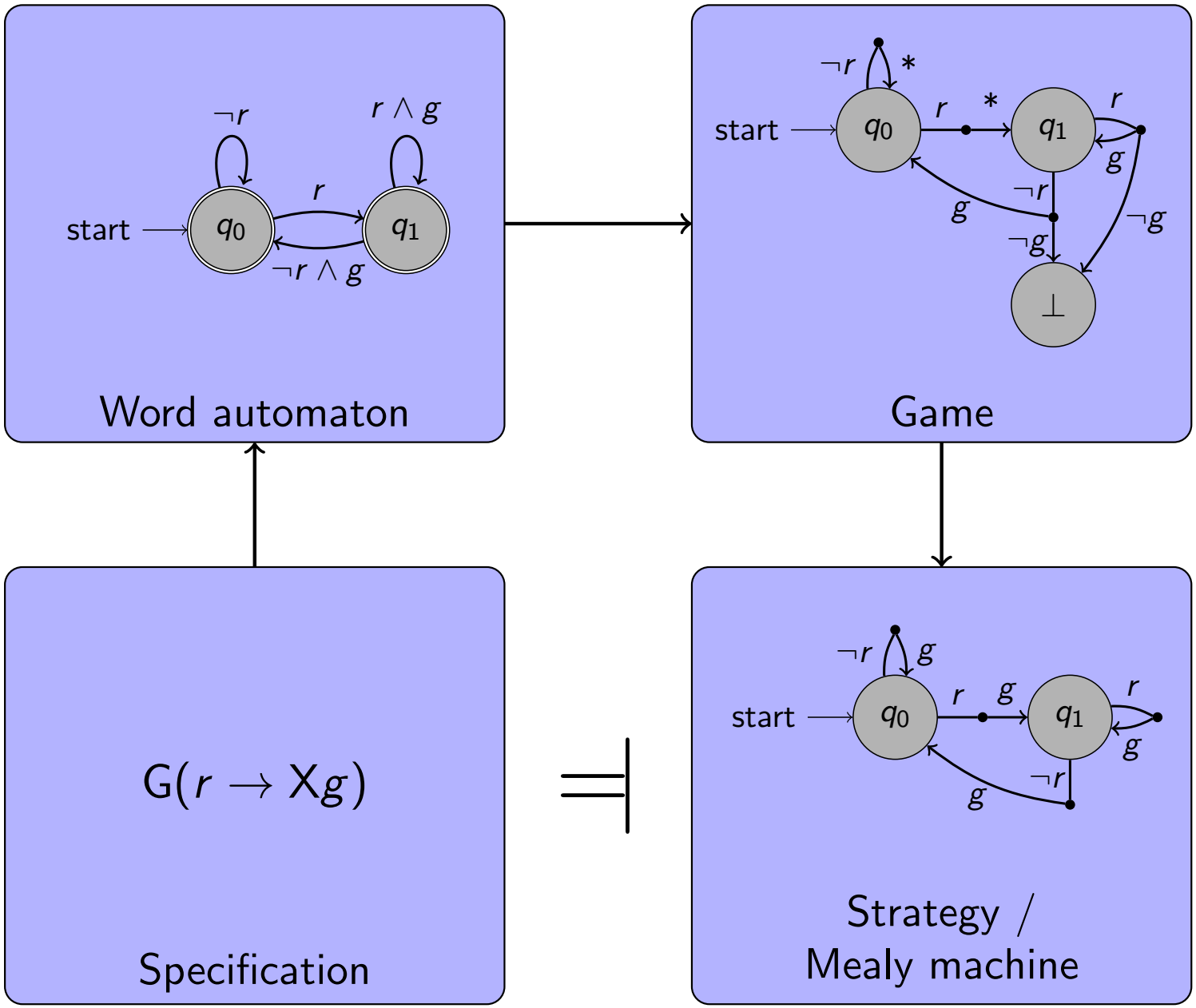
Specification

$\models$



Mealy machine

General synthesis workflow

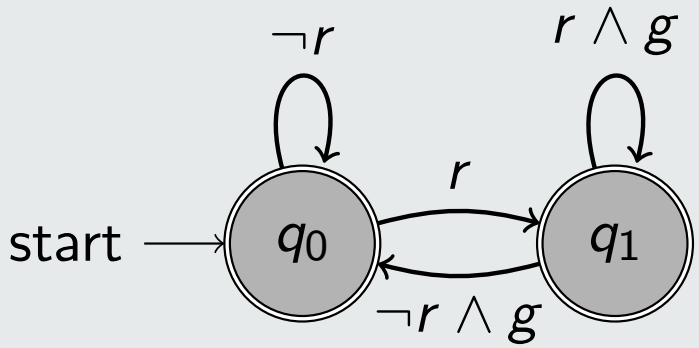


(Deterministic word) automata

Idea

Automata capture languages on a technical level

Example automaton

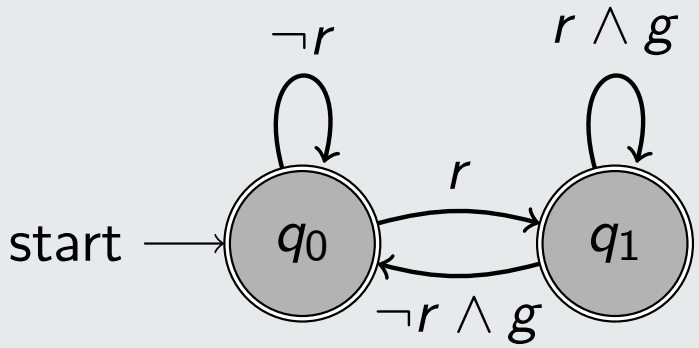


(Deterministic word) automata

Idea

Automata capture languages on a technical level

Example automaton



A word and the run induced by the automaton

$$\rho = \begin{pmatrix}0\\0\end{pmatrix} \begin{pmatrix}1\\0\end{pmatrix} \begin{pmatrix}1\\1\end{pmatrix} \begin{pmatrix}0\\1\end{pmatrix} \begin{pmatrix}0\\0\end{pmatrix} \cdots$$

$$\pi = q_0 - q_0 - q_1 - q_1 - q_0 - q_0 - \cdots$$

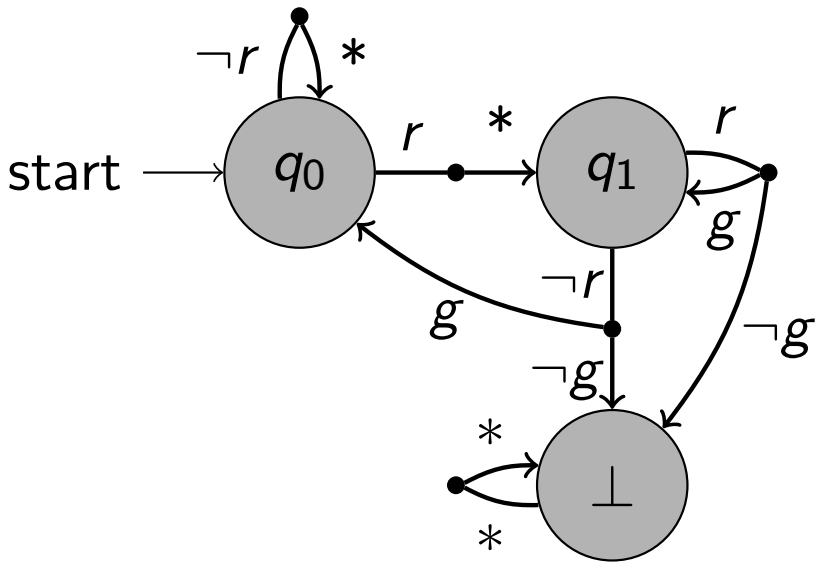
Games

Definition

Every player in a (two-player) game $\mathcal{G} = (V_0, V_1, \Sigma_0, \Sigma_1, E_0, E_1, v_0, \mathcal{F})$ has:

- Positions
- Actions
- Transitions
- A goal

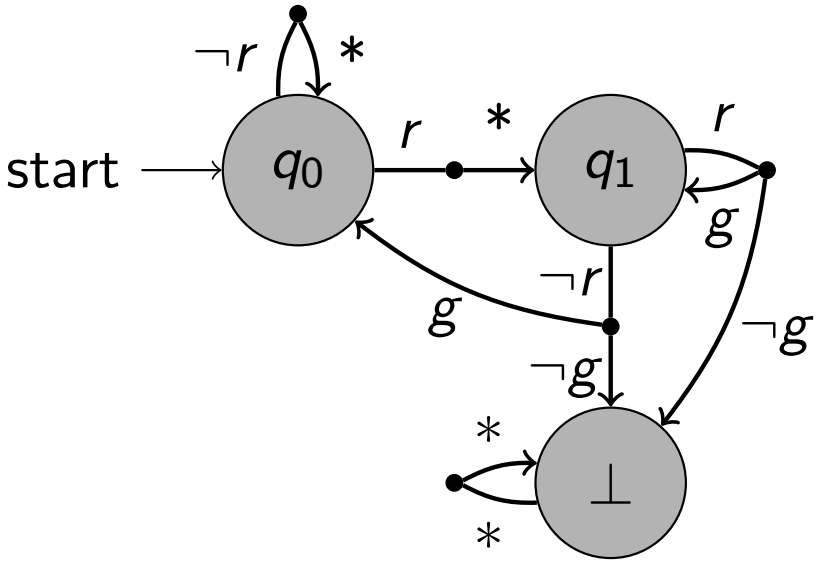
Additionally, there is some initial position.



Games for synthesis

Strategies

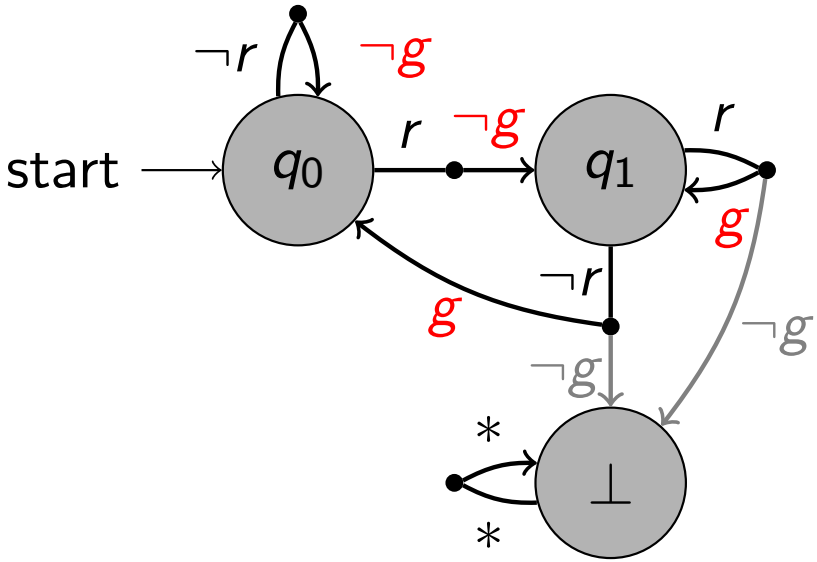
One player is the **system player**, whereas the other player is the **environment player**.
If player $p \in \{0, 1\}$ has a **stategy** to win, then she can enforce to win by playing the strategy. We say that player p **wins the game** in such a case.



Games for synthesis

Strategies

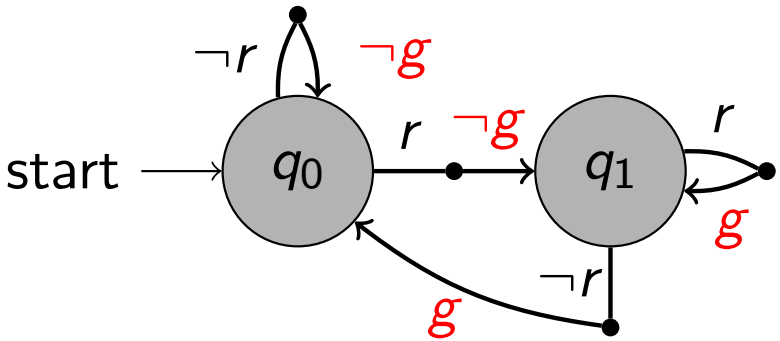
One player is the **system player**, whereas the other player is the **environment player**.
If player $p \in \{0, 1\}$ has a **stategy** to win, then she can enforce to win by playing the strategy. We say that player p **wins the game** in such a case.



Games for synthesis

Strategies

One player is the **system player**, whereas the other player is the **environment player**.
If player $p \in \{0, 1\}$ has a **stategy** to win, then she can enforce to win by playing the strategy. We say that player p **wins the game** in such a case.

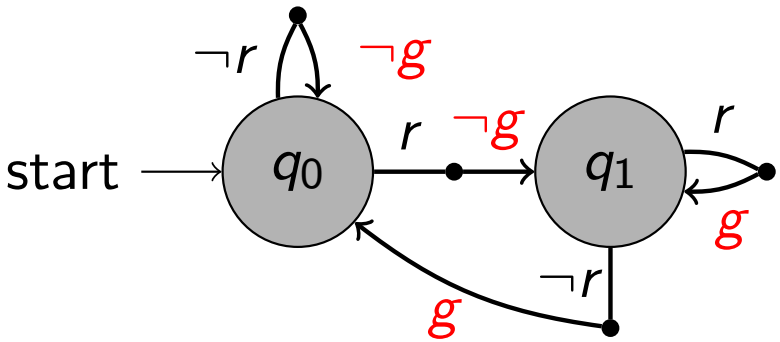


This is a Mealy Machine!

Games for synthesis

Strategies

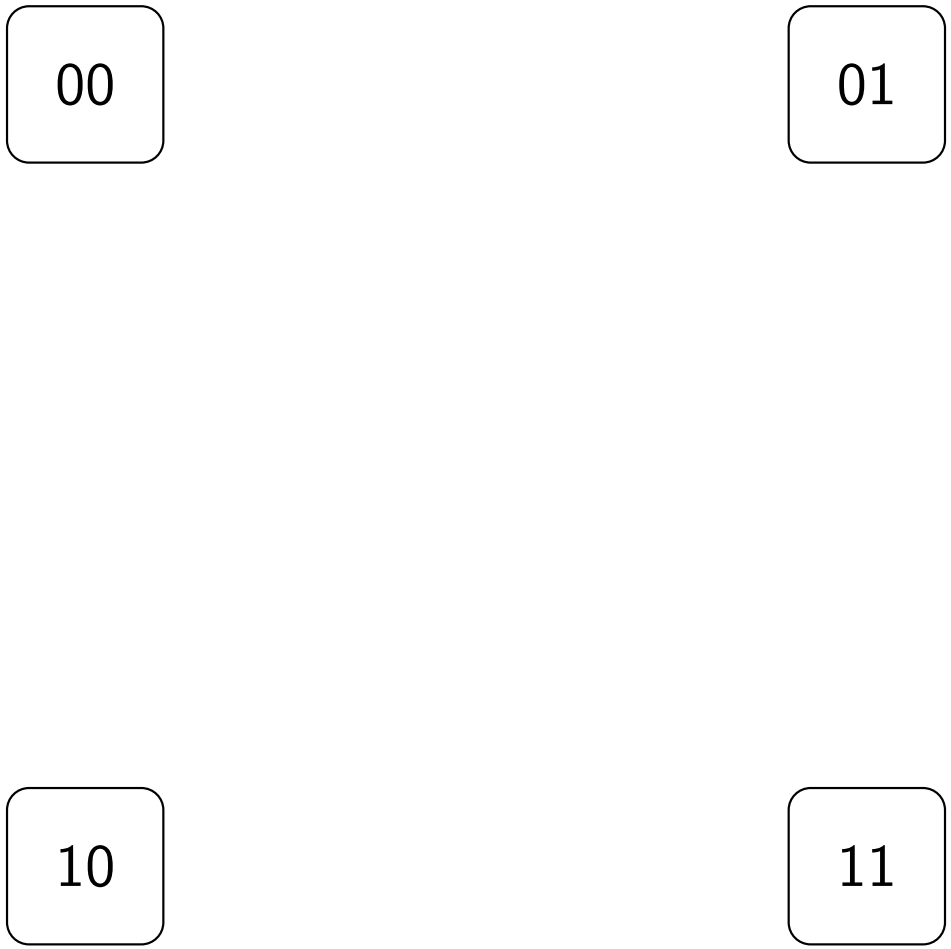
One player is the **system player**, whereas the other player is the **environment player**.
If player $p \in \{0, 1\}$ has a **stategy** to win, then she can enforce to win by playing the strategy. We say that player p **wins the game** in such a case.



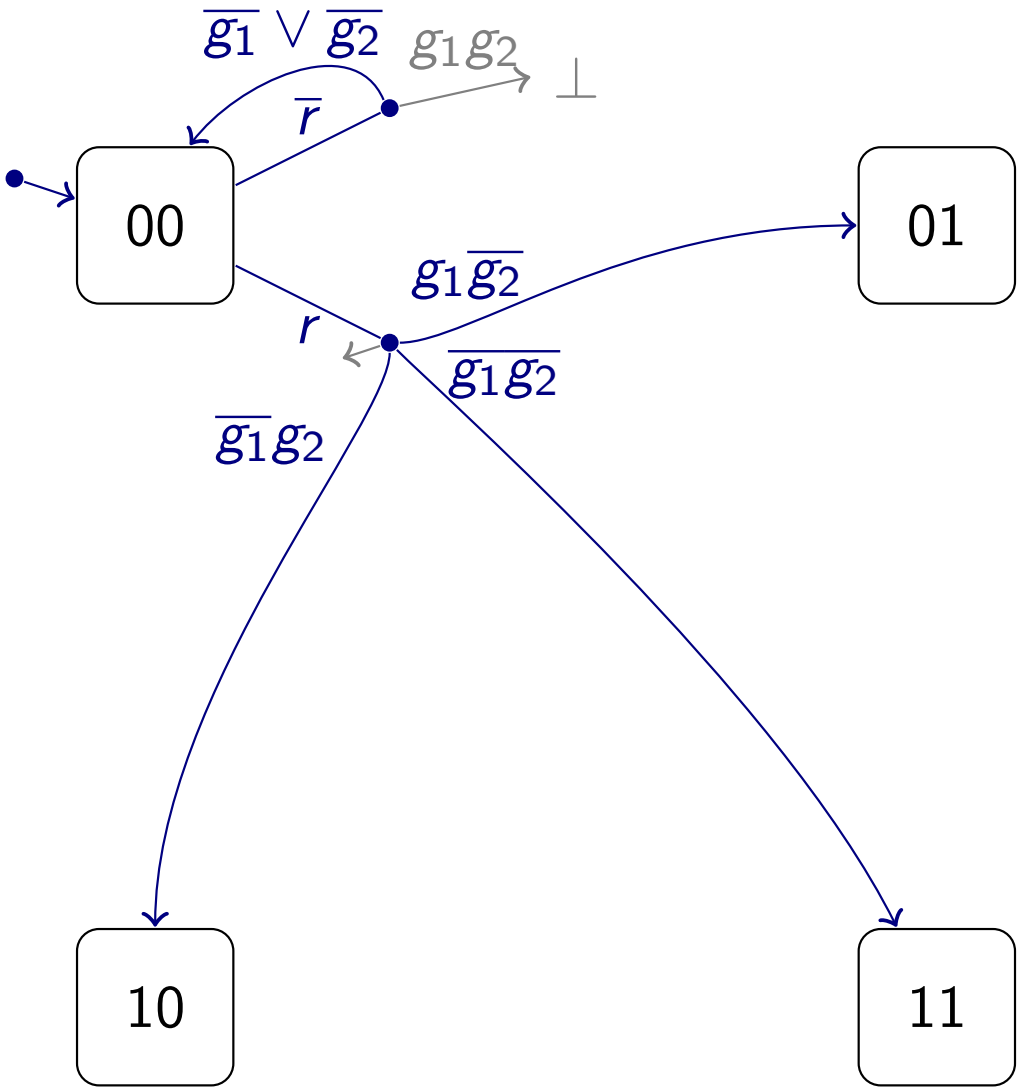
Strategies in synthesis games

In games that correspond to a specification, winning strategies for the system player represent Mealy (or Moore) machines that satisfy the specification.

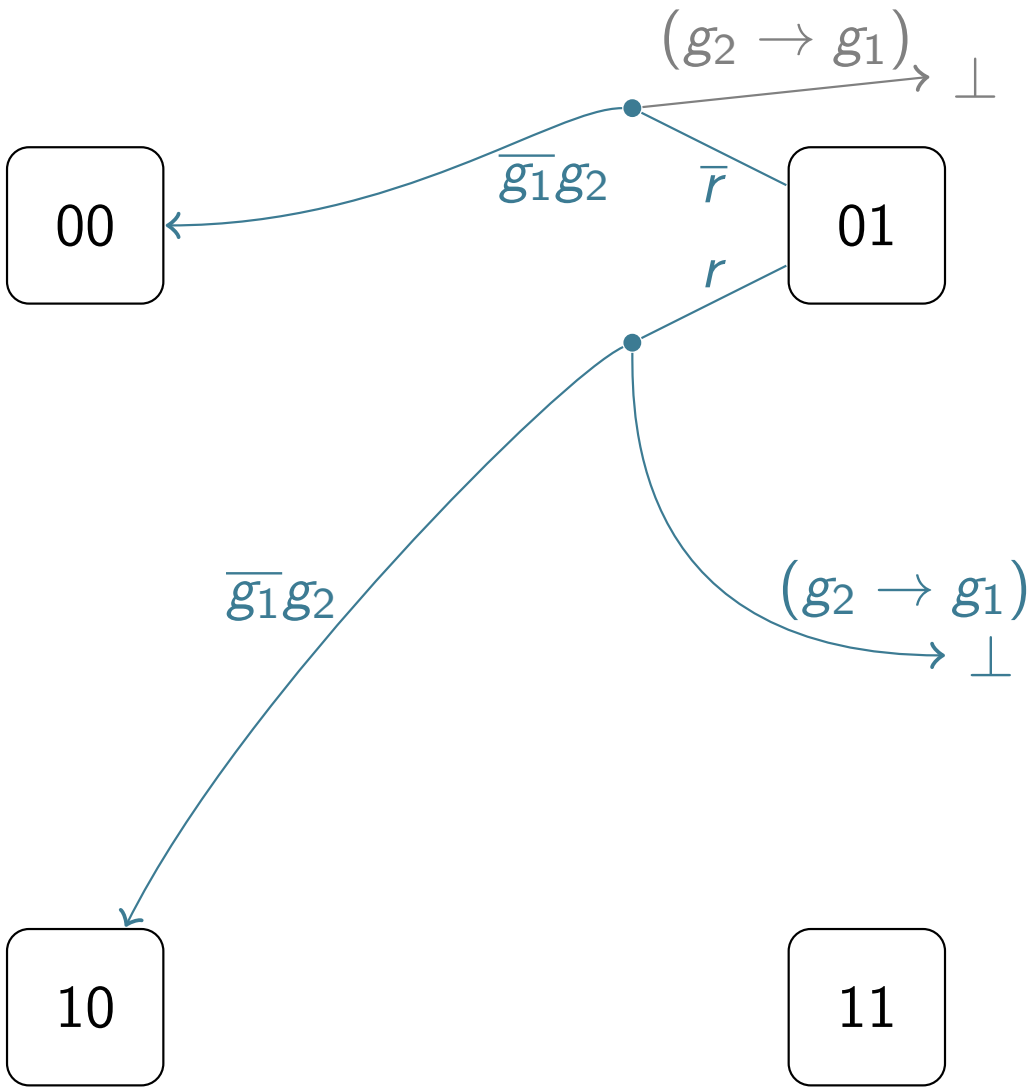
A more complicated (safety) game



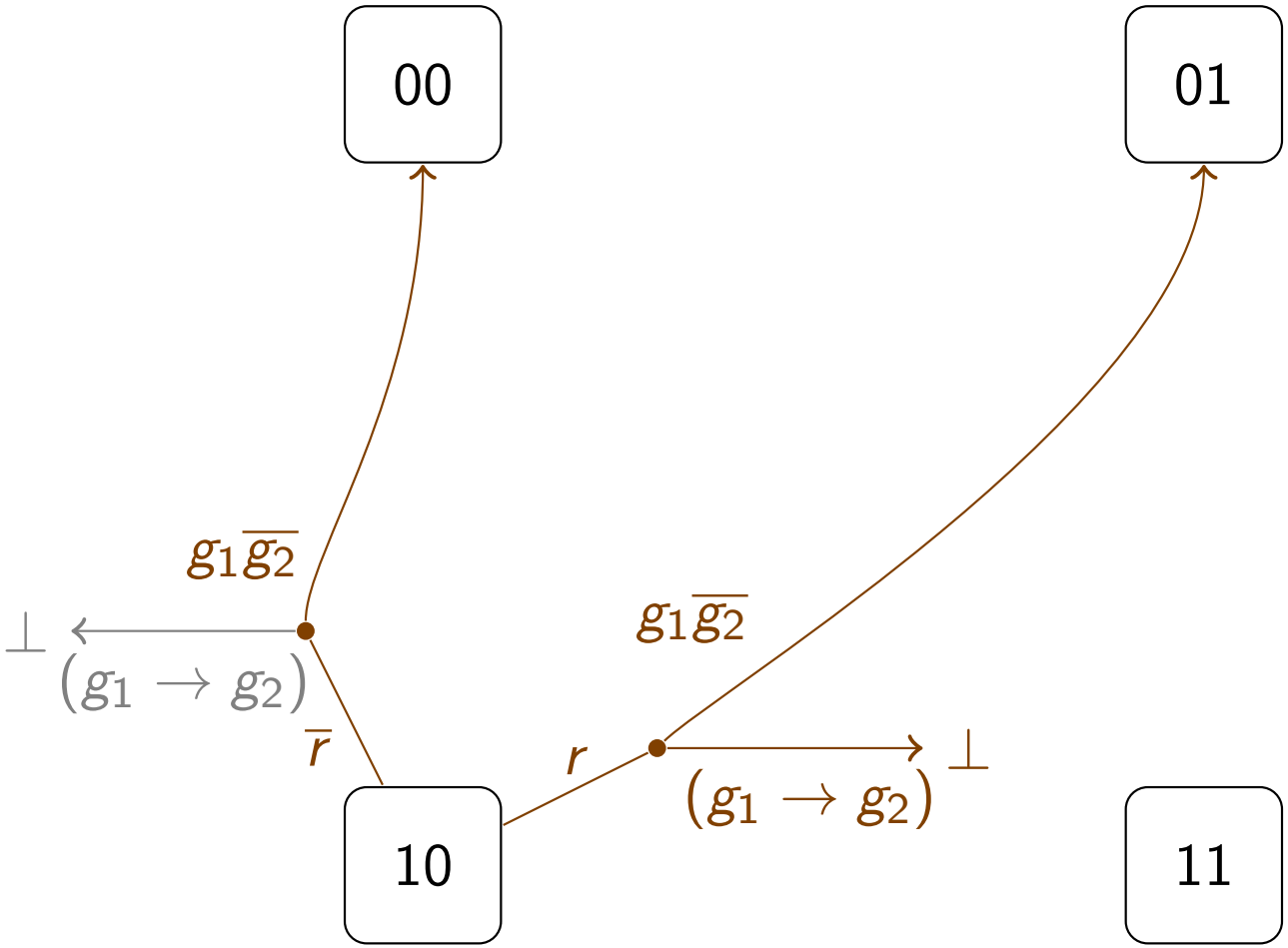
A more complicated (safety) game



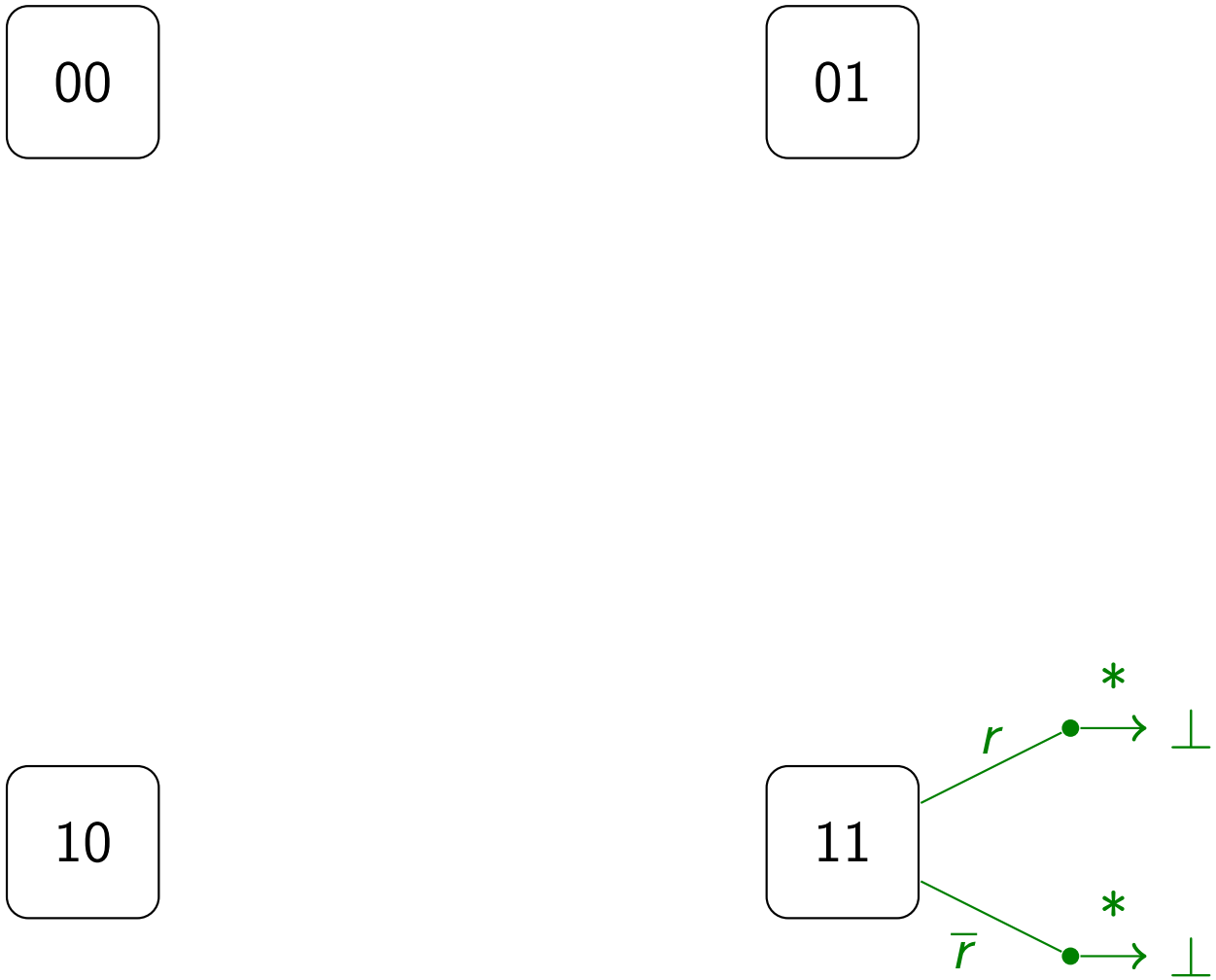
A more complicated (safety) game



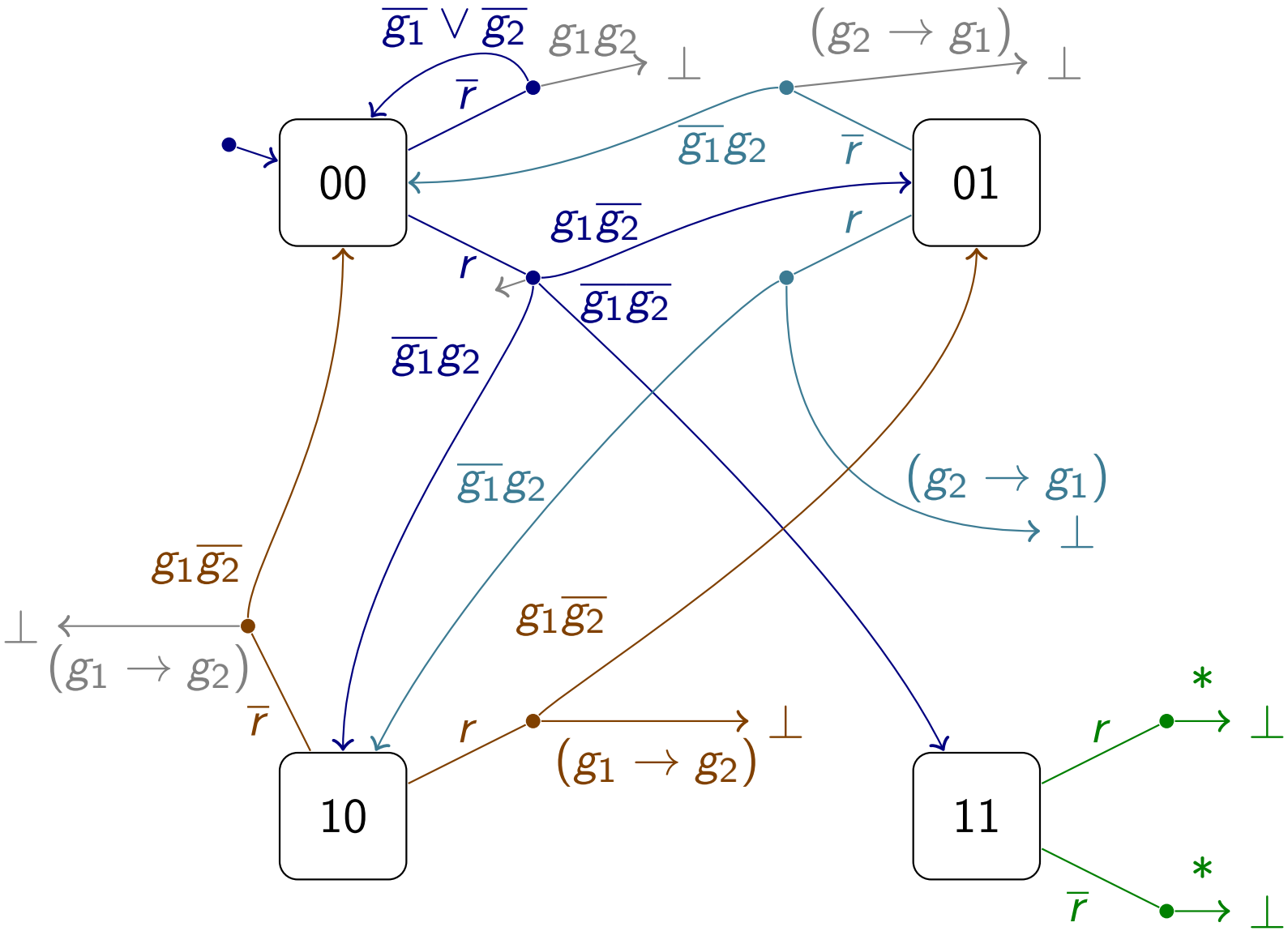
A more complicated (safety) game



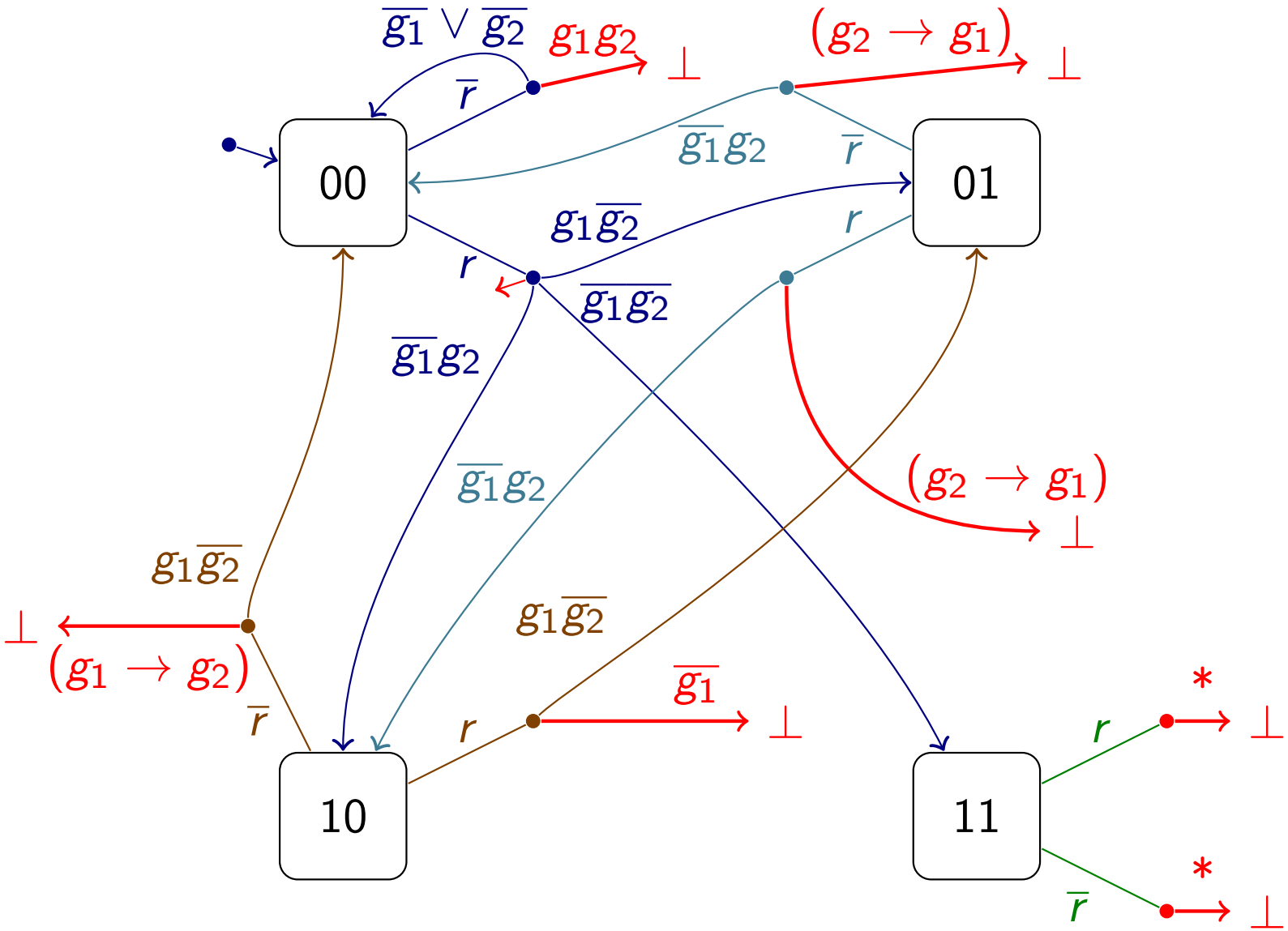
A more complicated (safety) game



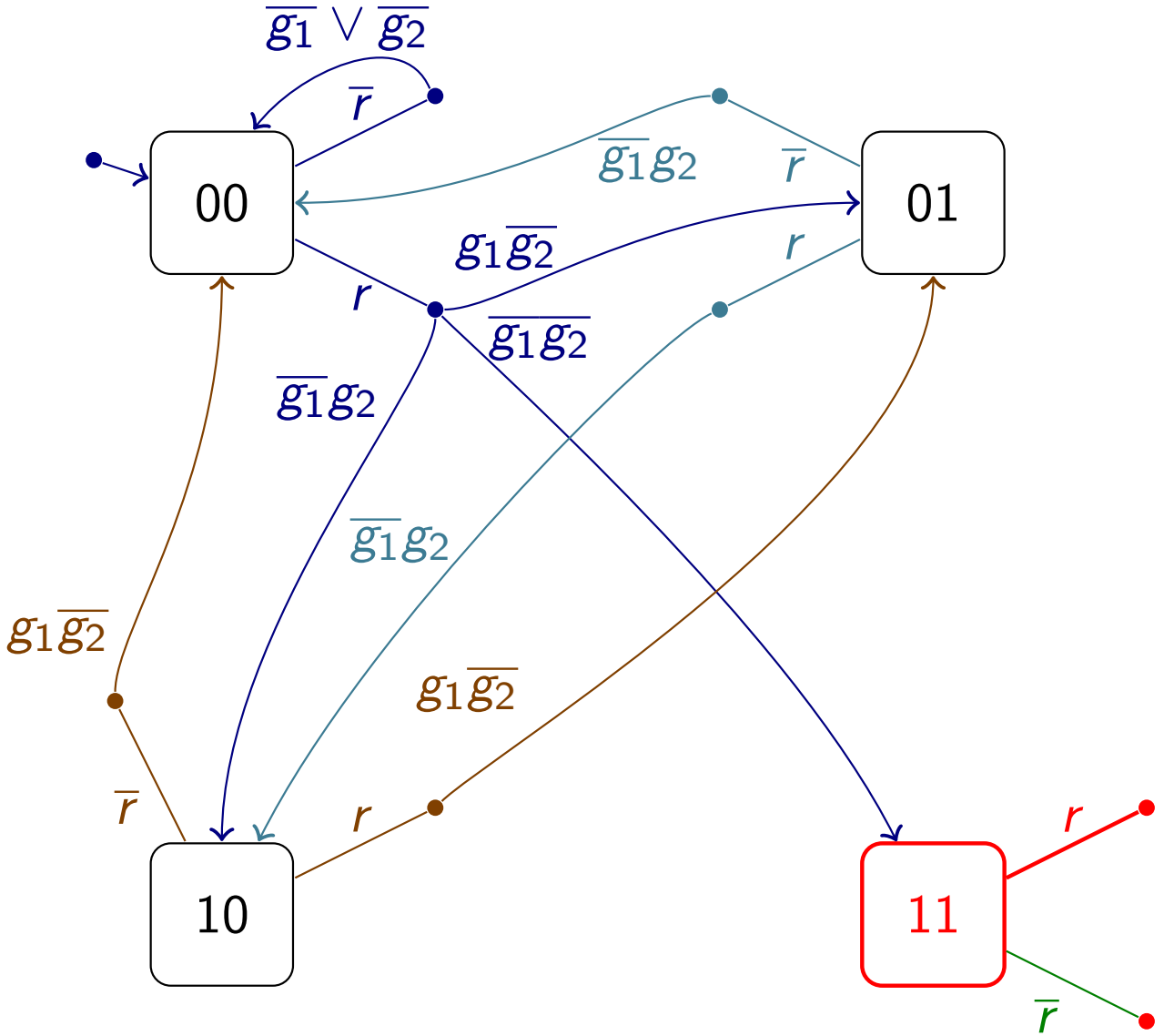
A more complicated (safety) game



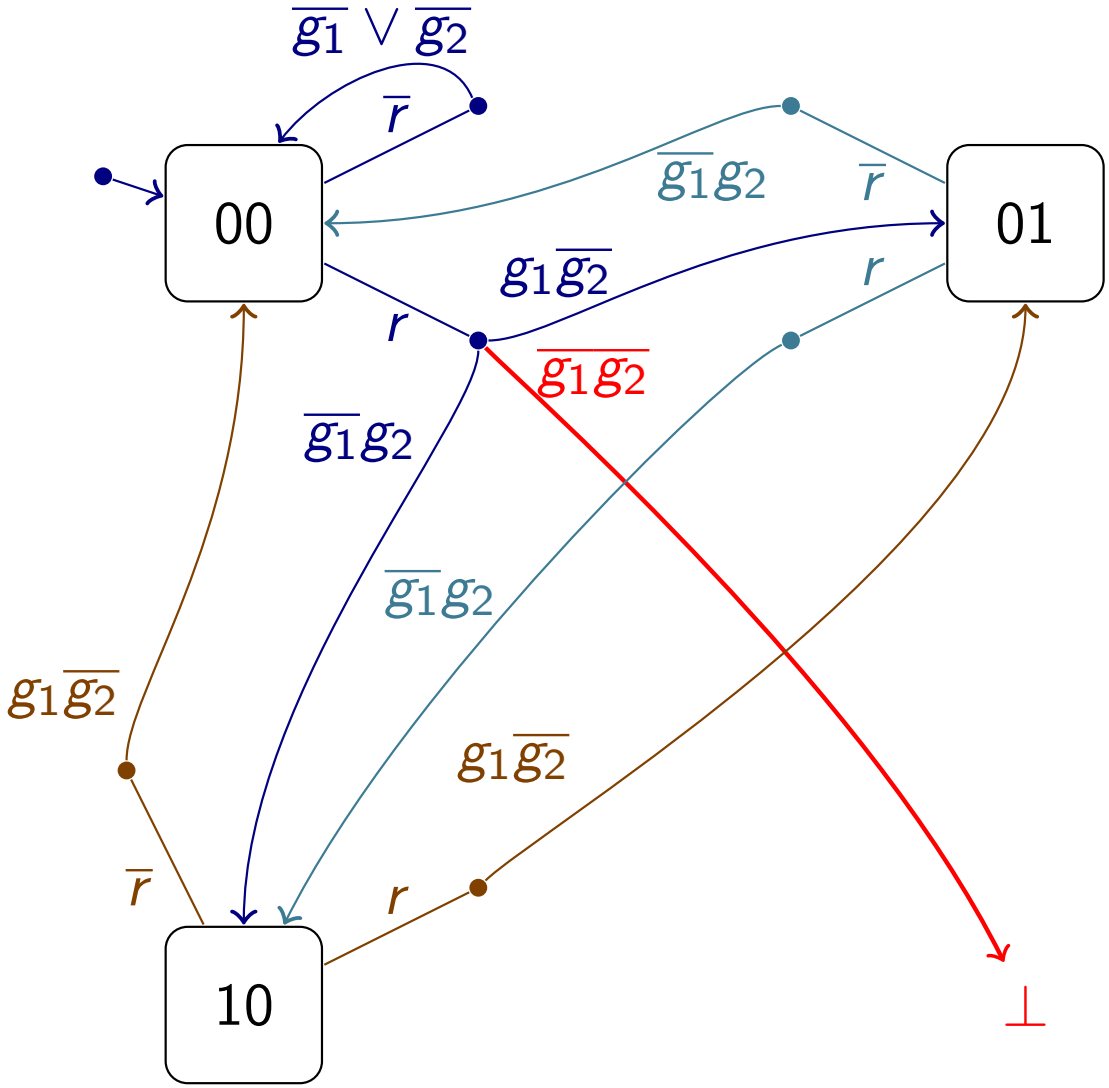
A more complicated (safety) game



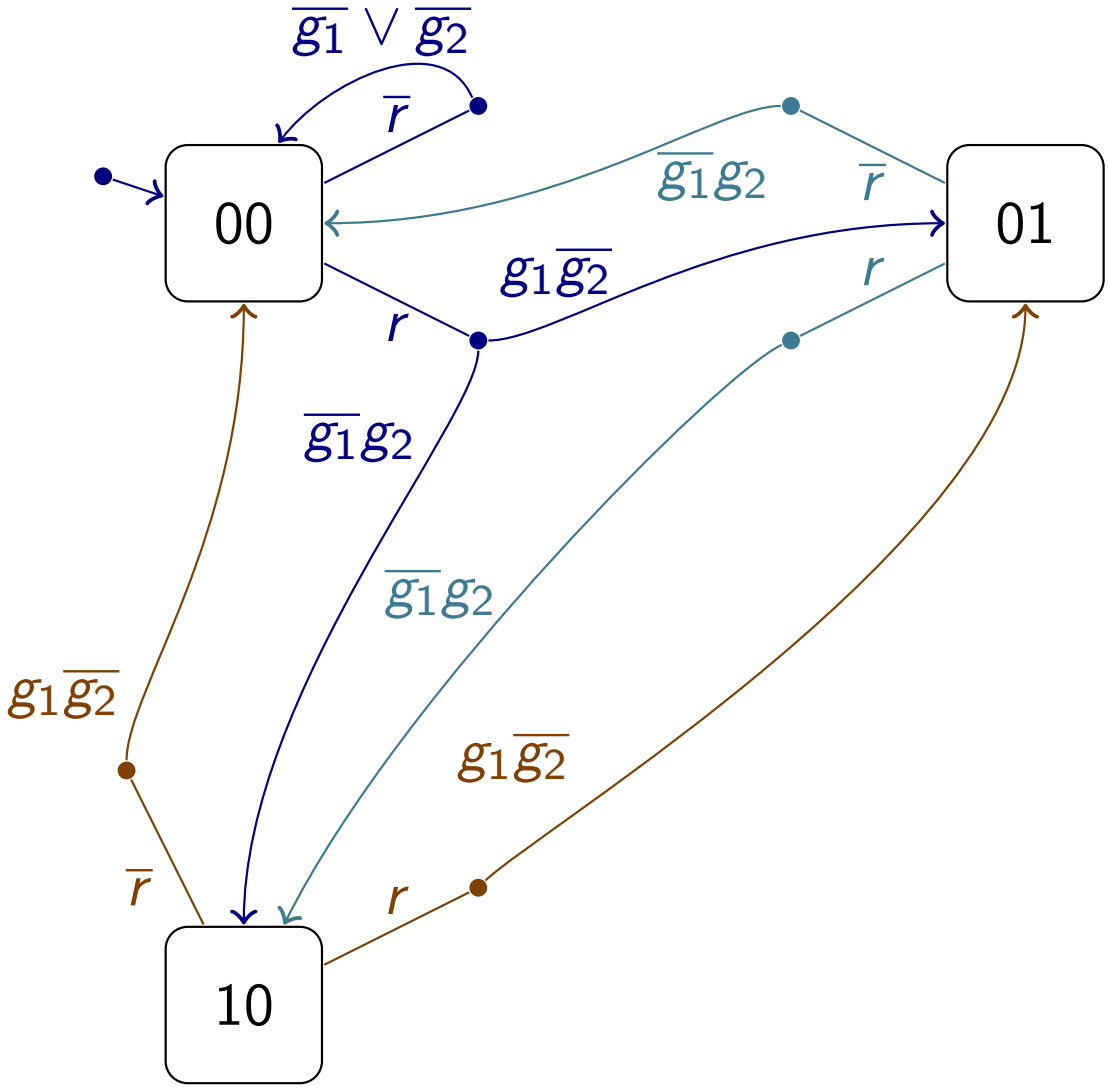
A more complicated (safety) game



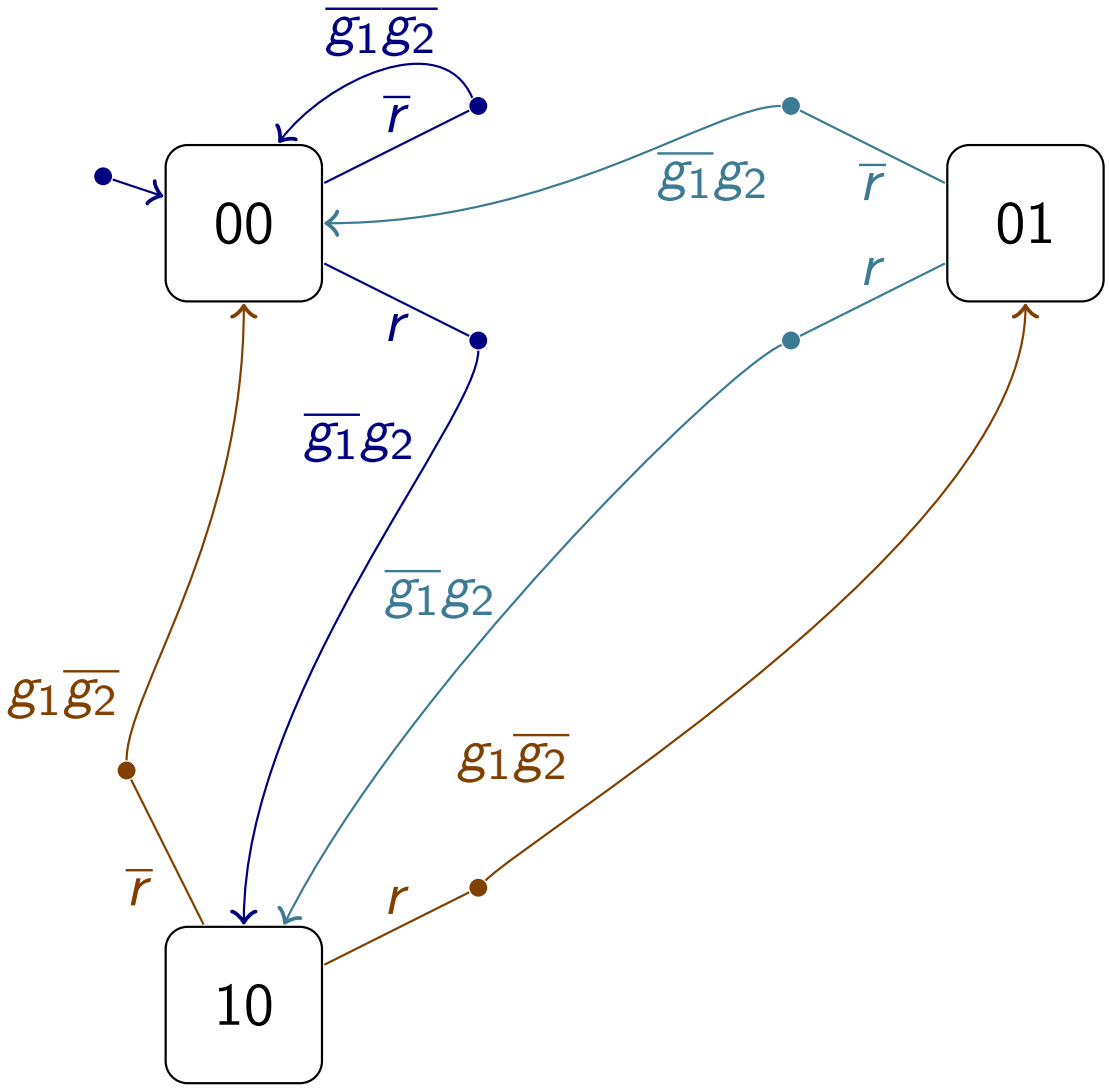
A more complicated (safety) game



A more complicated (safety) game



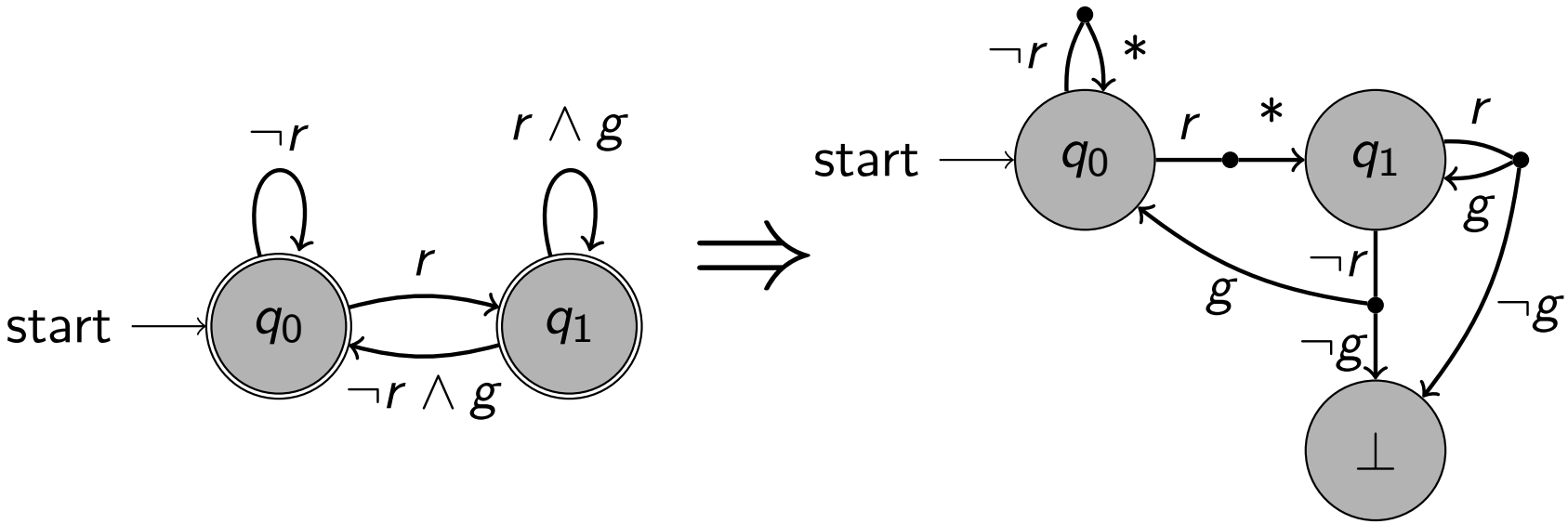
A more complicated (safety) game



Main question

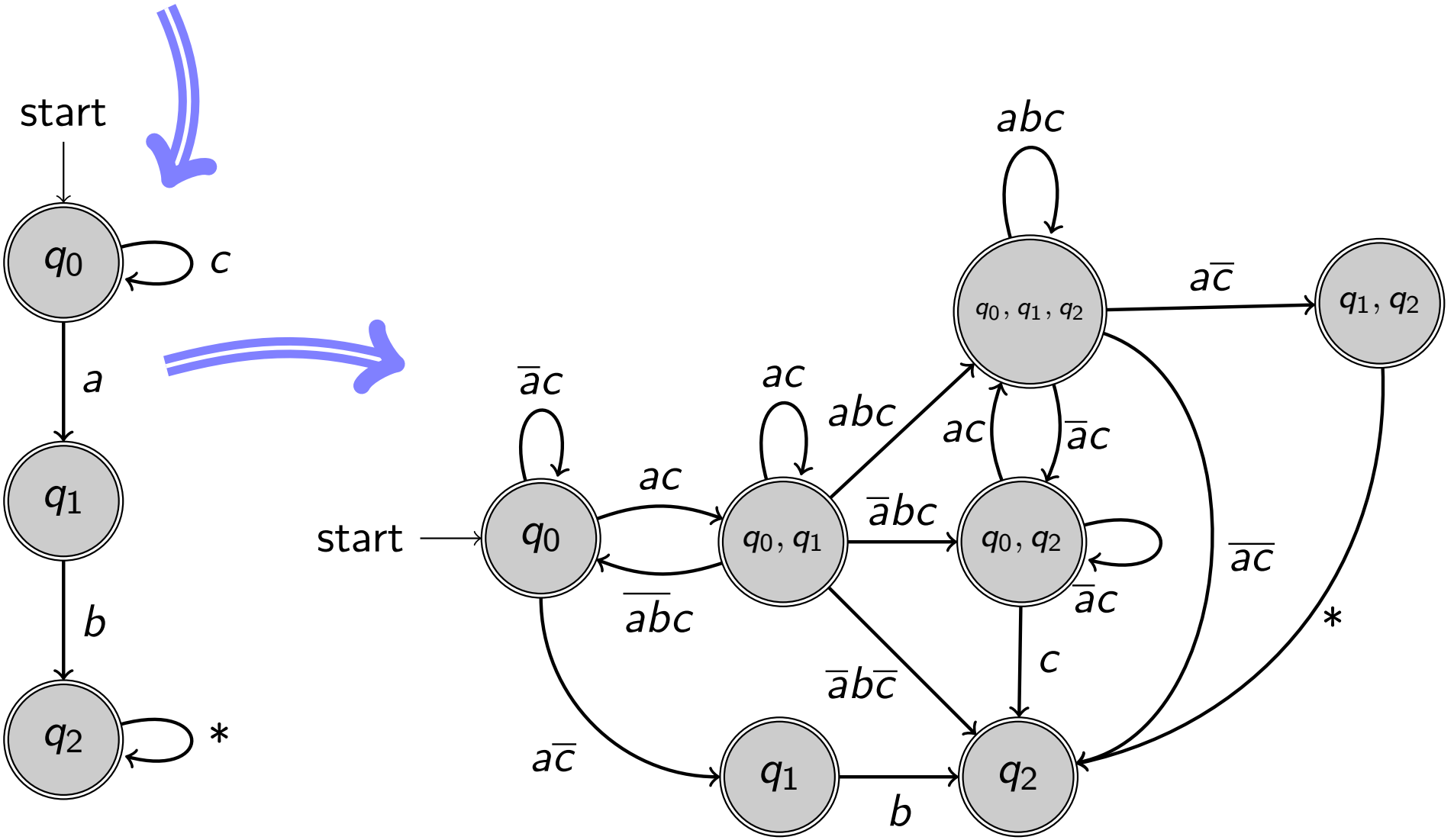
Ok, so how do we build a *synthesis game*?

Building safety games from deterministic safety automata

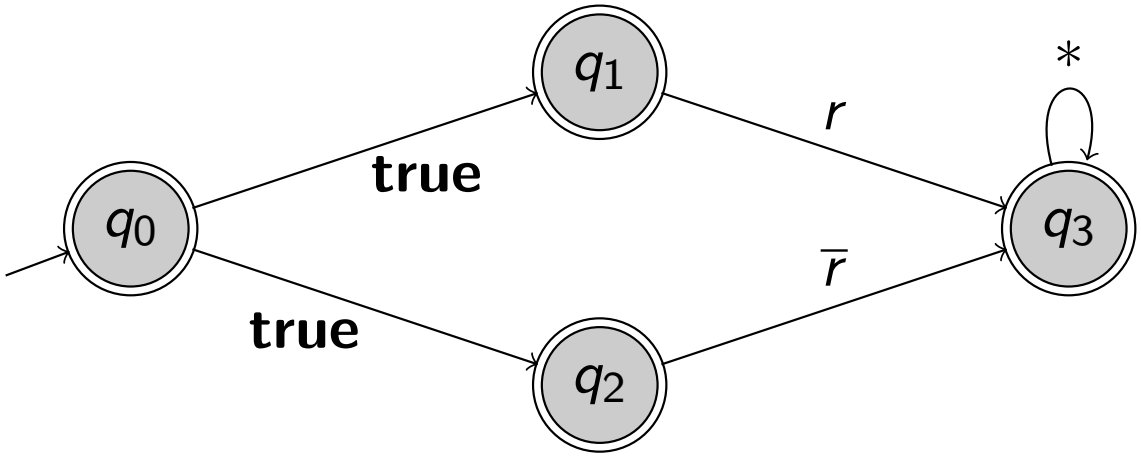


Building deterministic safety automata

$\psi = c\,\mathcal{U}\,(a \wedge \mathbf{X}b) \vee \mathbf{G}c$



So why do we need determinization (1/3) ?

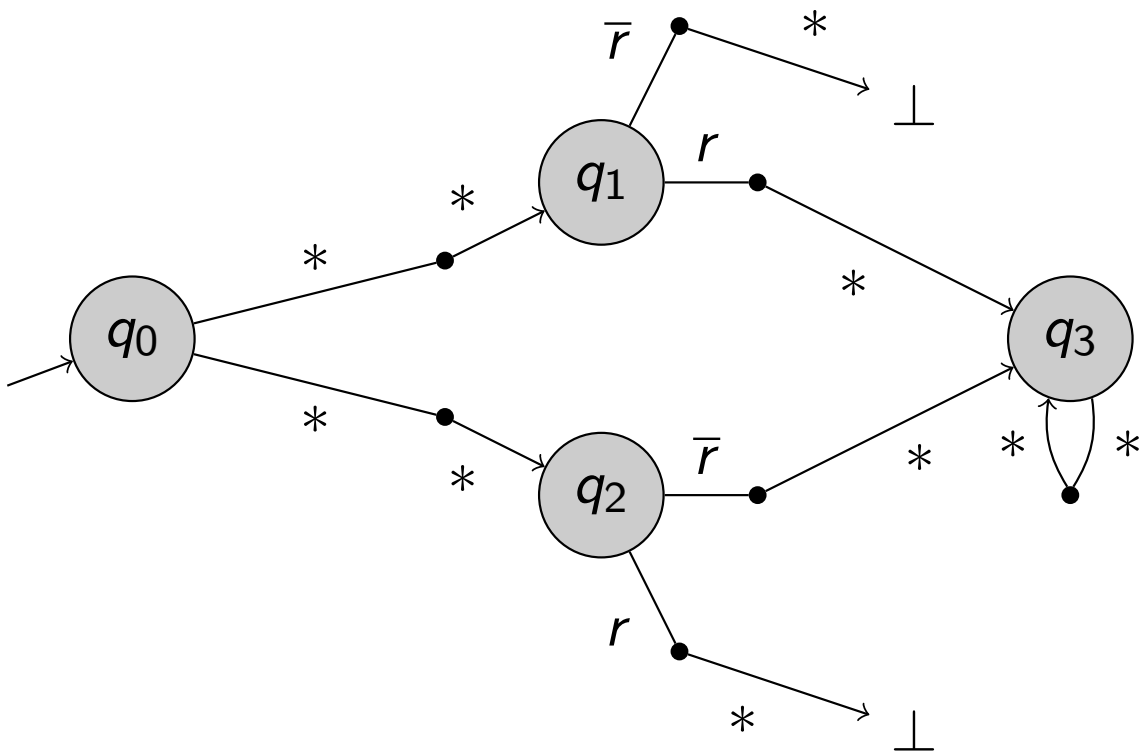


Specification: $\mathbf{X}r \vee \mathbf{X}\neg r$

$\mathrm{AP}_I = \{r\}$, $\mathrm{AP}_O = \{g\}$

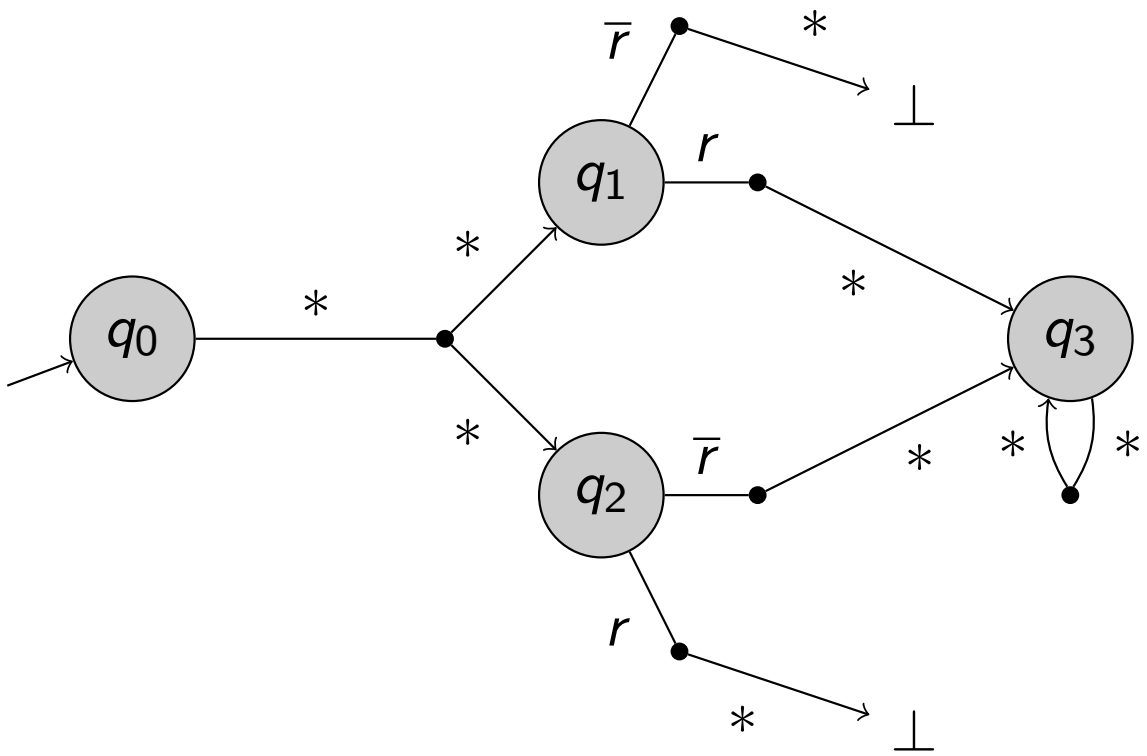
So why do we need determinization (2/3) ?

Case 1: The environment player resolves the nondeterminism



So why do we need determinization (2/3) ?

Case 2: The system player resolves the nondeterminism

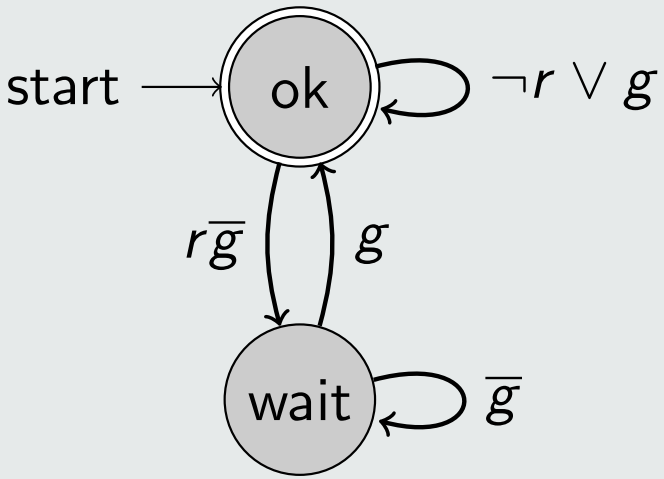


The non-safety, deterministic Büchi case

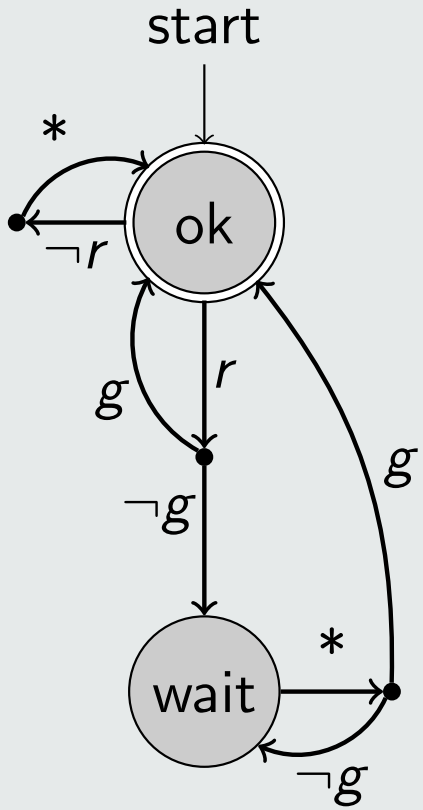
LTL Specification

$\mathbf{G}(r \rightarrow \mathbf{F}g)$

Büchi automaton



Büchi game



Deterministic vs. non-deterministic Büchi automata

Properties of Büchi automata

- For every LTL formula, there exists a non-deterministic Büchi automaton
- For some LTL formulas, there do not exist deterministic Büchi automata

Problem

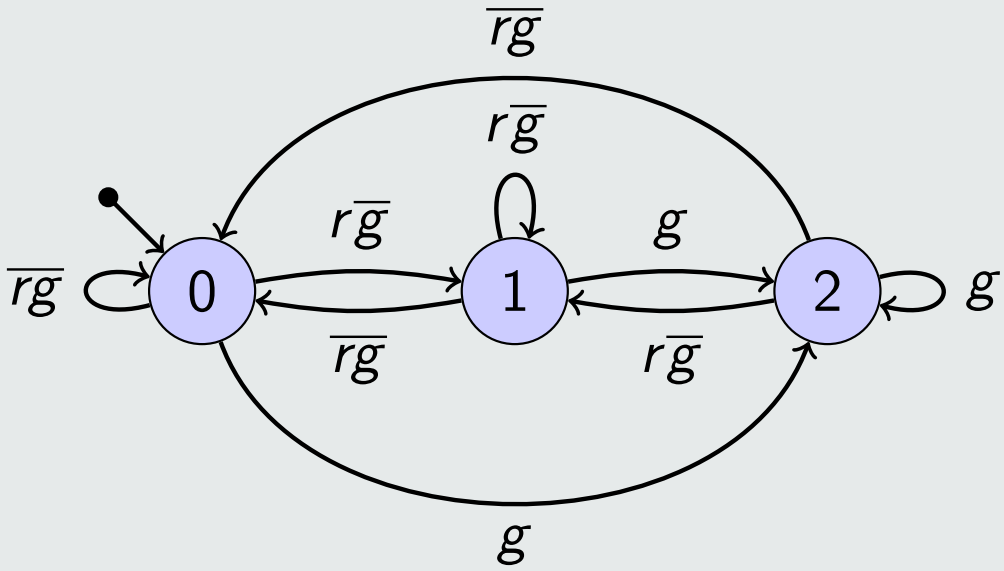
The automaton→game construction only works for *deterministic* automata

Solution

Use a richer automaton model/game winning condition: *parity automata*

Parity automata by example (1)

Automaton

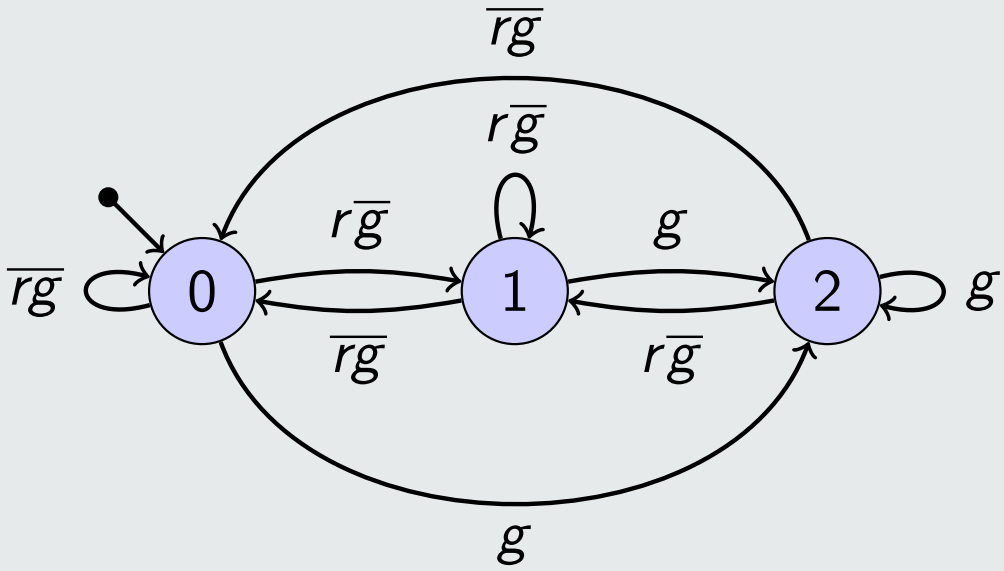


Acceptance condition

A deterministic parity word automaton accepts a word if the *highest color* visited infinitely often along the run of the automaton is *even*.

Parity automata by example (1)

Automaton

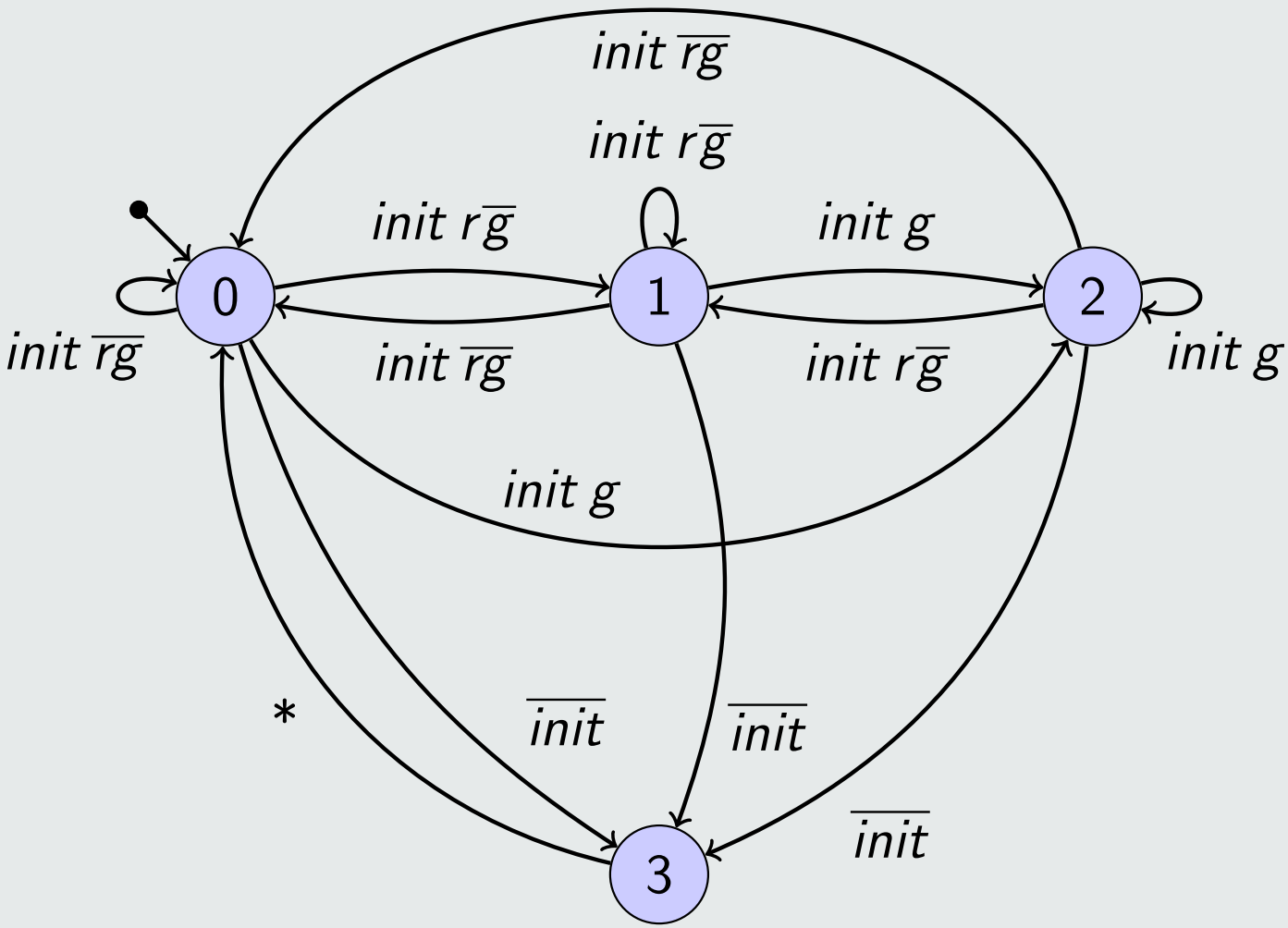


Encoded LTL property

$$\mathbf{GF}r \rightarrow \mathbf{GF}g$$

Parity automata by example (2)

Automaton

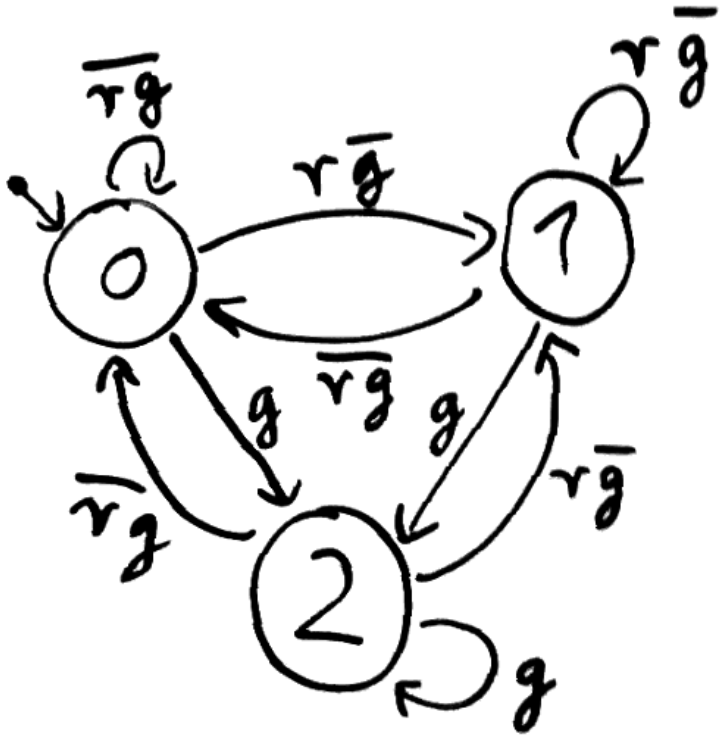


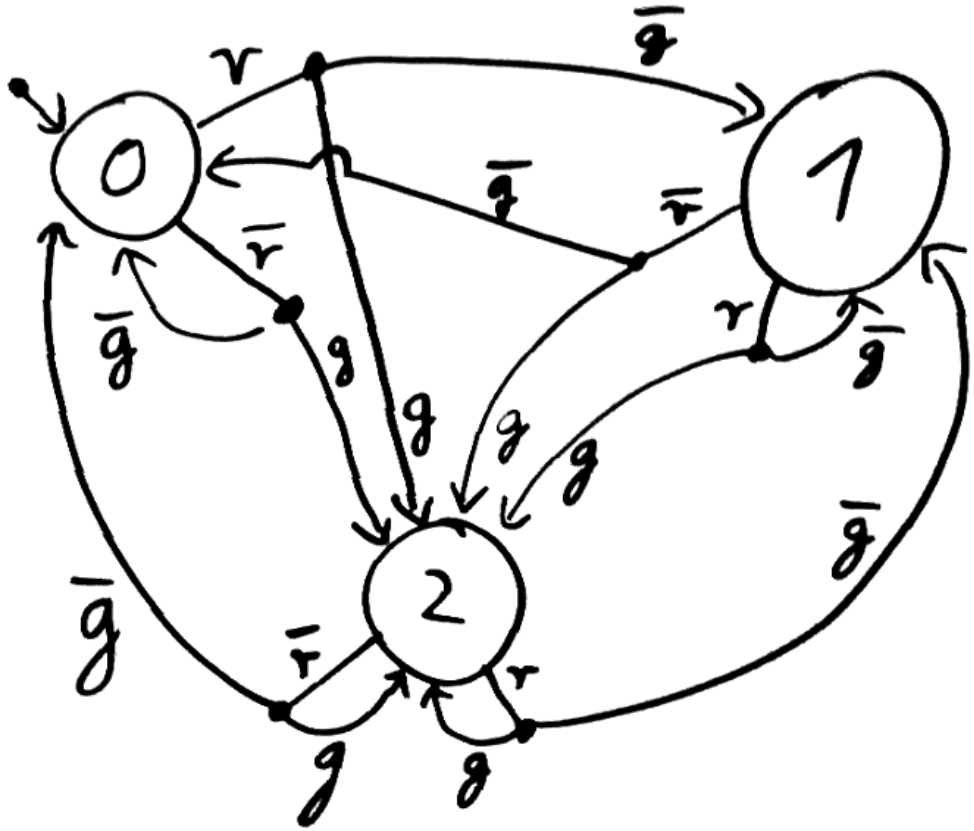
Spec'

$(\mathbf{GF}r \rightarrow \mathbf{GF}g) \wedge \mathbf{FG}\, init$

Parity automata and games

$$GF r \rightarrow GF g$$





Complexity considerations – Safety & full LTL

Overall complexity (time and space)

- Specification $\rightarrow$ non-deterministic automaton: **exponential**
- Non-deterministic aut. $\rightarrow$ deterministic aut.: **exponential**
- Building and solving the game: **polynomial-time for simple specification classes**, does not add an exponent for full LTL.

$\rightarrow$ Overall: doubly-exponential

Complexity considerations – Safety & full LTL

Overall complexity (time and space)

- Specification → non-deterministic automaton: **exponential**
- Non-deterministic aut. → deterministic aut.: **exponential**
- Building and solving the game: **polynomial-time for simple specification classes**, does not add an exponent for full LTL.

→ Overall: doubly-exponential

Can we do better?

Not for linear temporal logic: **2EXPTIME-complete** (Pnueli and Rosner, 1989)

The classical synthesis construction in practice

Nowadays...

..we have pretty good LTL-to-parity tools that work with *reasonably-sized* specifications. Using them, only parity game solving remains.

But what is *reasonably-sized*? – Positive example

```
./ltl2dpa --state-acceptance "G(process1 -> (process1 U (spitout1 U ready1))) &
(F G calib1 | G F fail1) & G(calib1 -> ! process1) & (G F ready1 -> G F calib1)"
```

$\rightarrow$ 35 states, 7 colors, 1.617s computation time

Tool used in this example: `owl` (Esparza et al., 2017), `https://www7.in.tum.de/~sickert/projects/owl/`

The classical synthesis construction in practice

Nowadays...

..we have pretty good LTL-to-parity tools that work with *reasonably-sized* specifications. Using them, only parity game solving remains.

But what is *reasonably-sized*? – Positive example

```
./ltl2dpa --state-acceptance "G(process1 -> (process1 U (spitout1 U ready1))) &
(F G calib1 | G F fail1) & G(calib1 -> ! process1) & (G F ready1 -> G F calib1)"
```

→ 35 states, 7 colors, 1.617s computation time

But what is *reasonably-sized*? – Half-negative example

```
./ltl2dpa --state-acceptance "G(process1 -> (process1 U (spitout1 U ready1))) &
(F G calib1 | G F fail1) & G(calib1 -> ! process1) & (G F ready1 -> G F calib1)
& G(process2 -> (process2 U (spitout2 U ready2))) & (F G calib2 | G F fail2) &
G(calib2 -> ! process2) & (G F ready2 -> G F calib2) & G(ready1 -> X process2)"
```

→ 54021 states, 19 colors (1 GB automaton file!), after 9m24.260s using 4.3 GB of RAM

What happens if we have a parity automaton?

Final synthesis step: Parity game solving

Complexity of some algorithms:

- $\approx O(n^c)$ (McNaughton, 1993; Zielonka, 1998)
- $\approx O(cmn^{\lceil c/2 \rceil})$ (Jurdzinski, 2000)
- $\approx O(n^{\sqrt{n}})$ (Jurdzinski et al., 2008)
- $\approx O(cmn^{\lceil c/3 \rceil})$ (Schewe, 2017)

Observation

Parity game solving can easily be a bottleneck

The central question of practical reactive synthesis

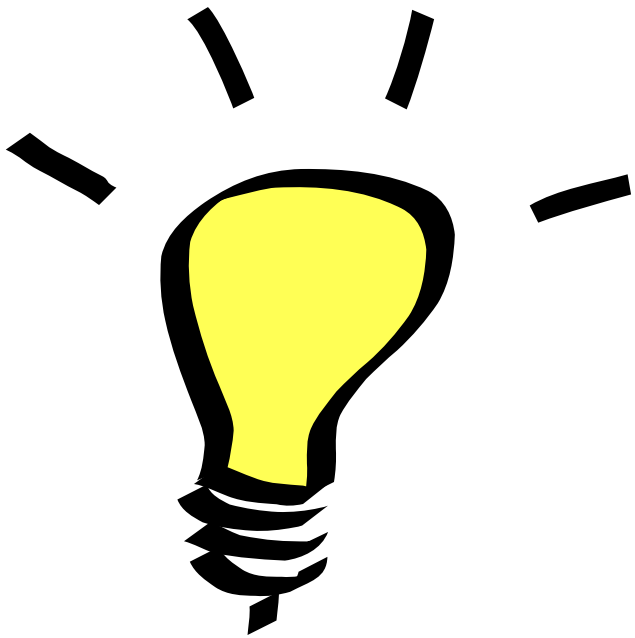
How can we get around the doubly-exponential synthesis complexity **in practice**?

But how can we do this?

Answer

By exploiting some properties of the problem such as:

- A small *synthesized implementation*
- A simple structure of the *specification*
- The *regularity* of the computed synthesis games

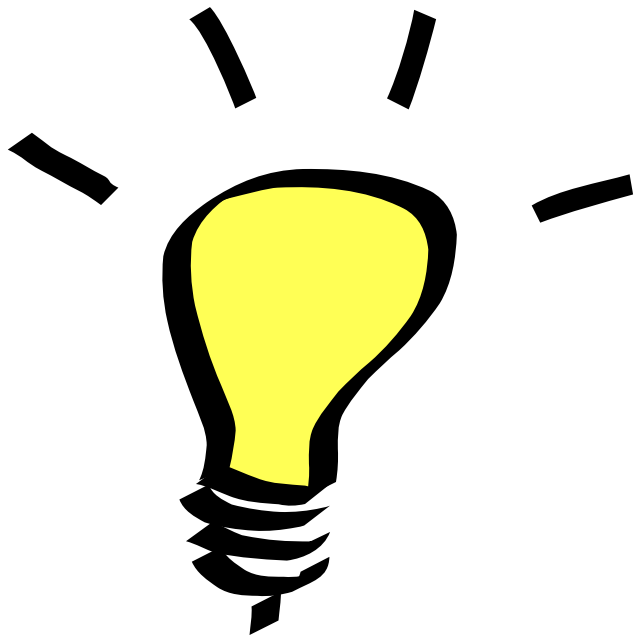


But how can we do this?

Answer

By exploiting some properties of the problem such as:

- A small *synthesized implementation*
  $\rightarrow$ **Bounded Synthesis**
- A simple structure of the *specification*
  $\rightarrow$ **GR(1) Synthesis**
- The *regularity* of the computed synthesis games
  $\rightarrow$ **Symbolic Synthesis**



References I

Javier Esparza, Jan Kretínský, Jean-François Raskin, and Salomon Sickert. From LTL and limit-deterministic büchi automata to deterministic parity automata. In *Tools and Algorithms for the Construction and Analysis of Systems - 23rd International Conference, TACAS 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings, Part I*, pages 426–442, 2017. doi: 10.1007/978-3-662-54577-5_25. URL https://doi.org/10.1007/978-3-662-54577-5_25.

Marcin Jurdzinski. Small progress measures for solving parity games. In *STACS 2000, 17th Annual Symposium on Theoretical Aspects of Computer Science, Lille, France, February 2000, Proceedings*, pages 290–301, 2000. doi: 10.1007/3-540-46541-3_24. URL https://doi.org/10.1007/3-540-46541-3_24.

Marcin Jurdzinski, Mike Paterson, and Uri Zwick. A deterministic subexponential algorithm for solving parity games. *SIAM J. Comput.*, 38(4):1519–1532, 2008. doi: 10.1137/070686652. URL https://doi.org/10.1137/070686652.

Robert McNaughton. Infinite games played on finite graphs. *Ann. Pure Appl. Logic*, 65(2): 149–184, 1993. doi: 10.1016/0168-0072(93)90036-D. URL https://doi.org/10.1016/0168-0072(93)90036-D.

Amir Pnueli and Roni Rosner. On the synthesis of an asynchronous reactive module. In Giorgio Ausiello, Mariangiola Dezani-Ciancaglini, and Simona Ronchi Della Rocca, editors, *ICALP*, volume 372 of *Lecture Notes in Computer Science*, pages 652–671. Springer, 1989. ISBN 3-540-51371-X.

References II

Sven Schewe. Solving parity games in big steps. *J. Comput. Syst. Sci.*, 84:243–262, 2017. doi: 10.1016/j.jcss.2016.10.002. URL https://doi.org/10.1016/j.jcss.2016.10.002.

Wieslaw Zielonka. Infinite games on finitely coloured graphs with applications to automata on infinite trees. *Theor. Comput. Sci.*, 200(1-2):135–183, 1998. doi: 10.1016/S0304-3975(98)00009-7. URL https://doi.org/10.1016/S0304-3975(98)00009-7.