Model Checking and Games

Part IX - Summary

Rüdiger Ehlers, Clausthal University of Technology

September 2019

Safety of systems

Driving question of this course

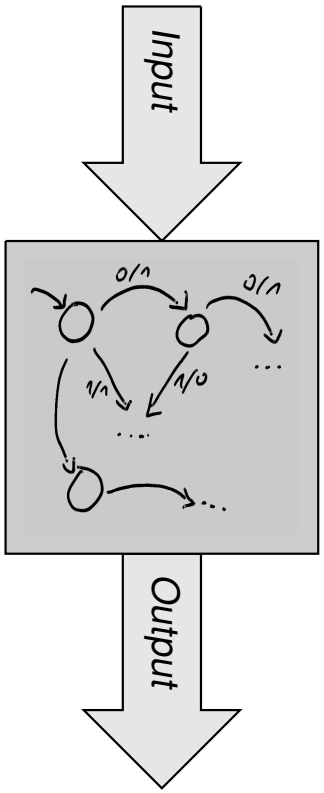
How can we ensure that a *reactive* system does what it is supposed to be doing?

Secondary driving question of this course

...And how can we automate what we do to achieve this to the greatest extent possible?

Verification and Synthesis

Verification:



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

Satisfied

Not satisfied
(+ counter-example)

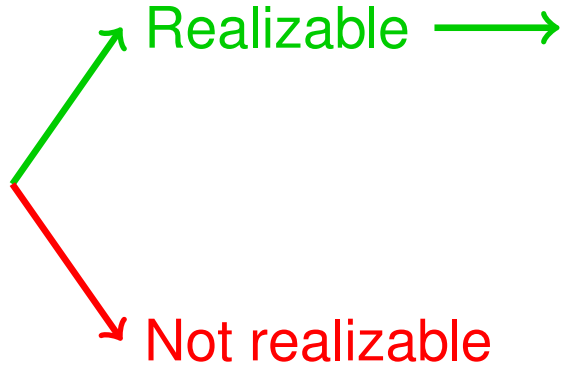
Verification and Synthesis

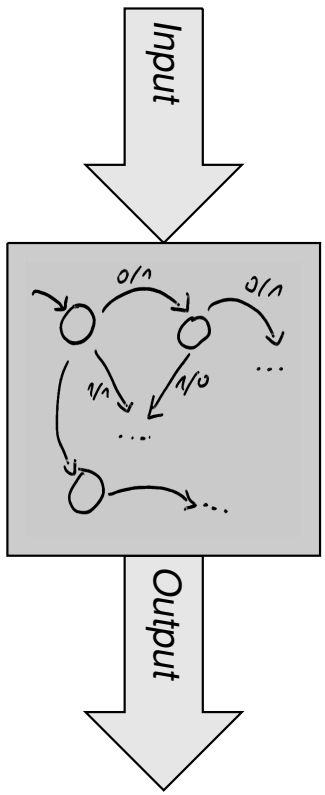
Synthesis:

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

$+$

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





Structure of this course

Basics

1. Modelling reactive systems
2. Modelling specifications
3. Verifying specifications against models

Tools

Spin

Symbolic verification

1. Binary decision diagrams / Algorithms for BDDs
2. Satisfiability solving & Bounded Model checking

Tools

~~(dd/CUDD)~~
~~Alloy~~

Reactive synthesis

1. Introduction to reactive synthesis

Tools

Slugs

Model checking vs. testing

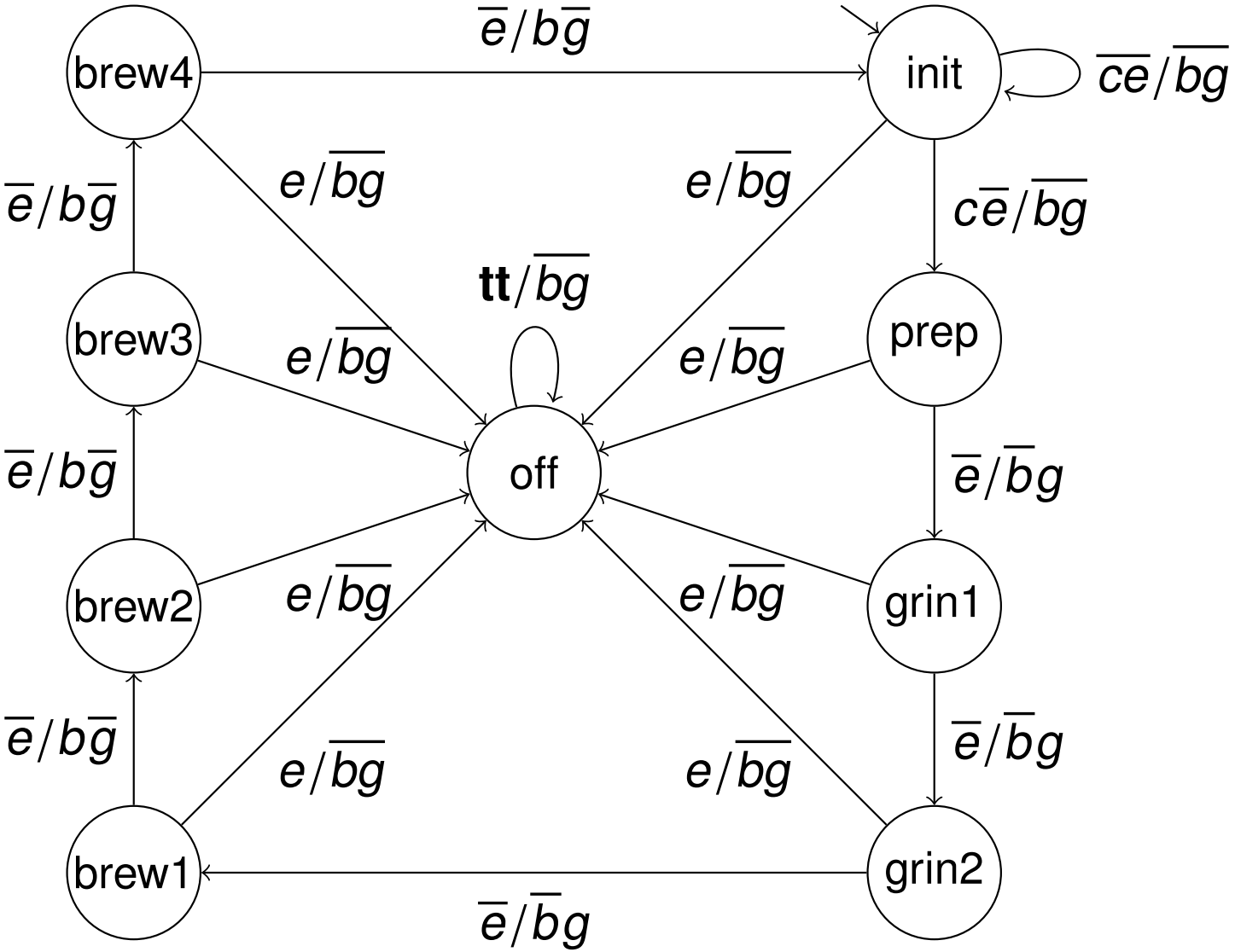
Advantages of testing

- It utilizes the system to test directly, no need to build a model
- Test cases are natural to write for programmers

Advantages of model checking

- Guarantees to find *all* bugs in the model

Example: Coffee machine



Kripke structures: *Labeled transition systems*

Definition

A *Kripke structure* is a tuple $(S, S_0, T, \mathrm{AP}, L)$ with:

- a set of states S,
- a set of initial states S_0, and
- a set of transitions $T \subseteq S \times S$.
- a set of *propositions* AP, and
- a labelling function $L : S \to 2^{\mathrm{AP}}$.

Comparison: LTS vs Mealy & Moore machines

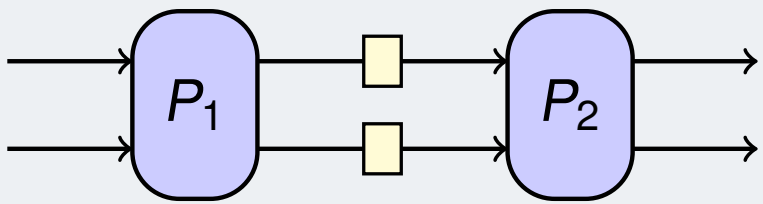
Comparison

Aspect	**Mealy / Moore Machines**	**Labeled Transition systems**
Input & Output	Clearly separated	No separation
Determinism	Yes	No
Time	Discrete time steps	No concrete notion of time
Ability to abstract from details	No	Yes
Non-binary alphabets	Yes	No

Synchronous communication between processes

Possibility 1: Synchronous composition in the same time step

Possibility 2: Synchronous composition with delay



Linear temporal logic

Basic properties

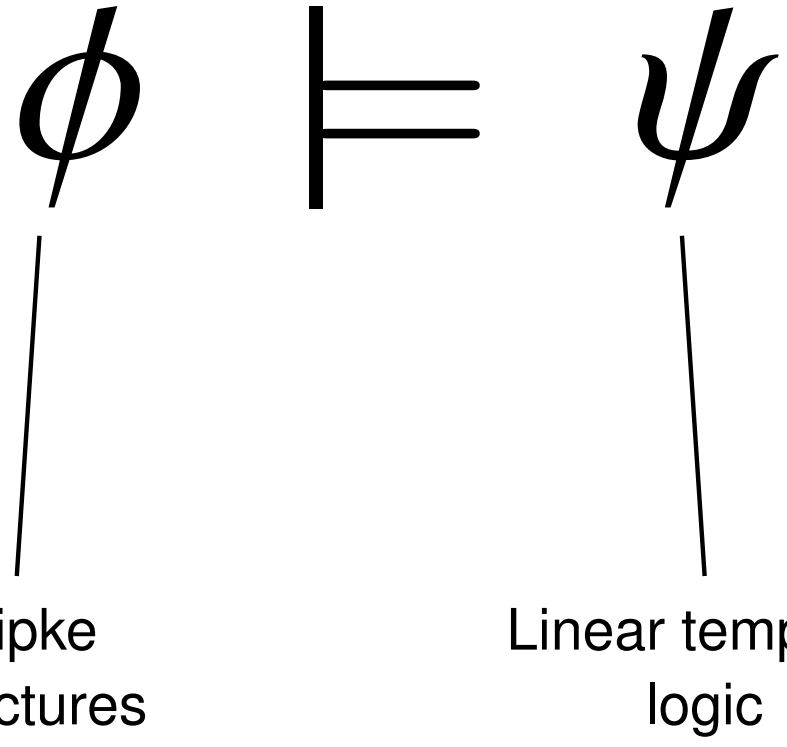
- Originally introduced by Pnueli (1977)
- It is a logic on *infinite words* over the alphabet 2^{AP} for some set of atomic propositions AP $\rightarrow$ can be used to reason about traces of a system

Use in verification

We can use LTL to express properties that we want to hold along *all* traces of a system. Examples:

- Every *green light* of a traffic light should eventually be followed by a *yellow light*.
- ~~For every direction *d*, the traffic light implementation should permit a future execution such that there is a green light for direction *d*.~~

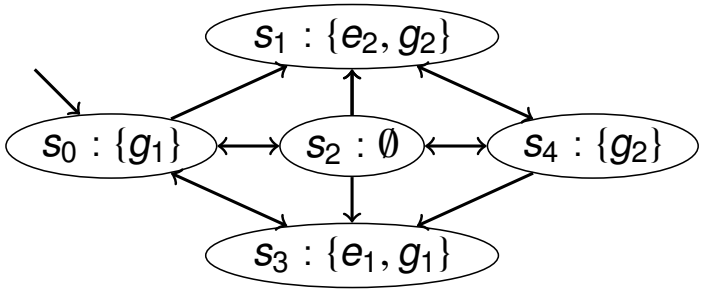
Did we define all important components?

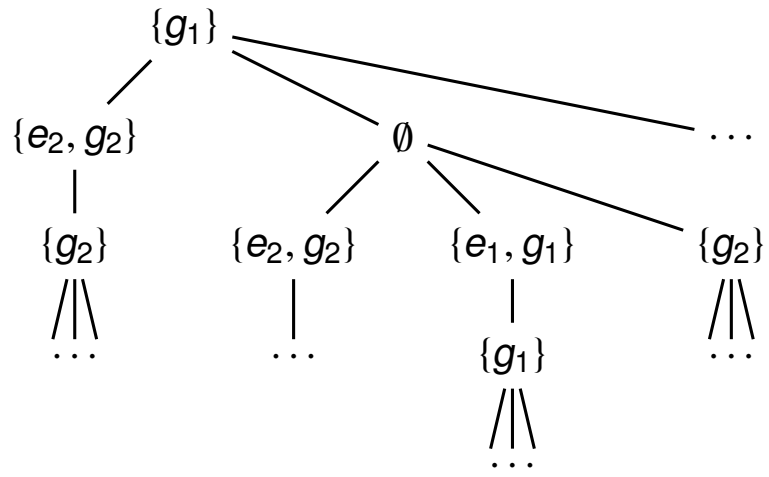


Observation

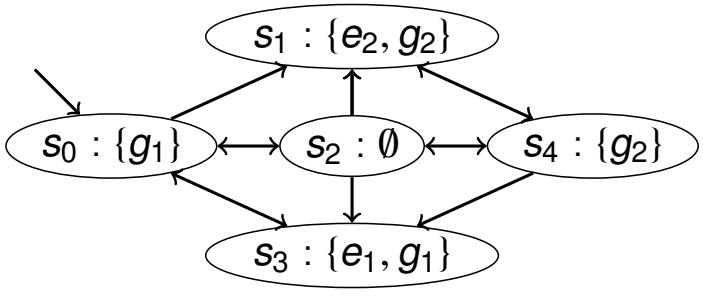
That does not fit! LTL is defined over words, not transition systems!

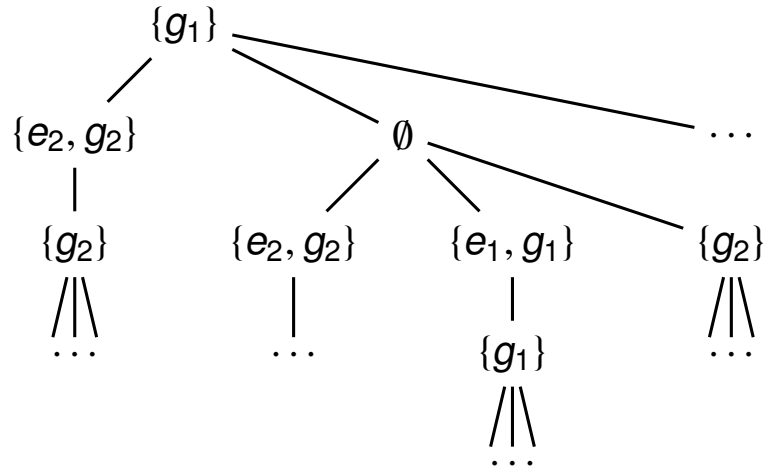
Unrolling a transition system to a tree (assuming a single initial state)





Unrolling a transition system to a tree (assuming a single initial state)





LTL interpretation

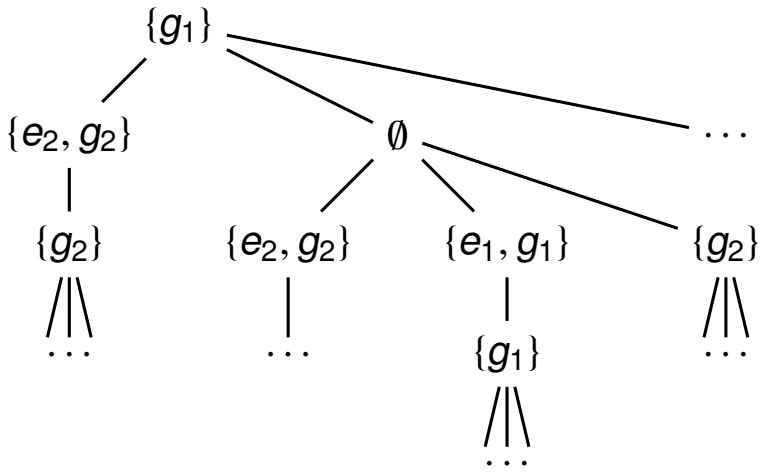
An LTL specification is checked along every *branch* in the computation tree!

Computation tree logic

Basic properties

- Originally introduced by Clarke and Emerson (1981)
- Is a logic over *trees with infinite branches*
- Extends propositional logic

Examples for CTL



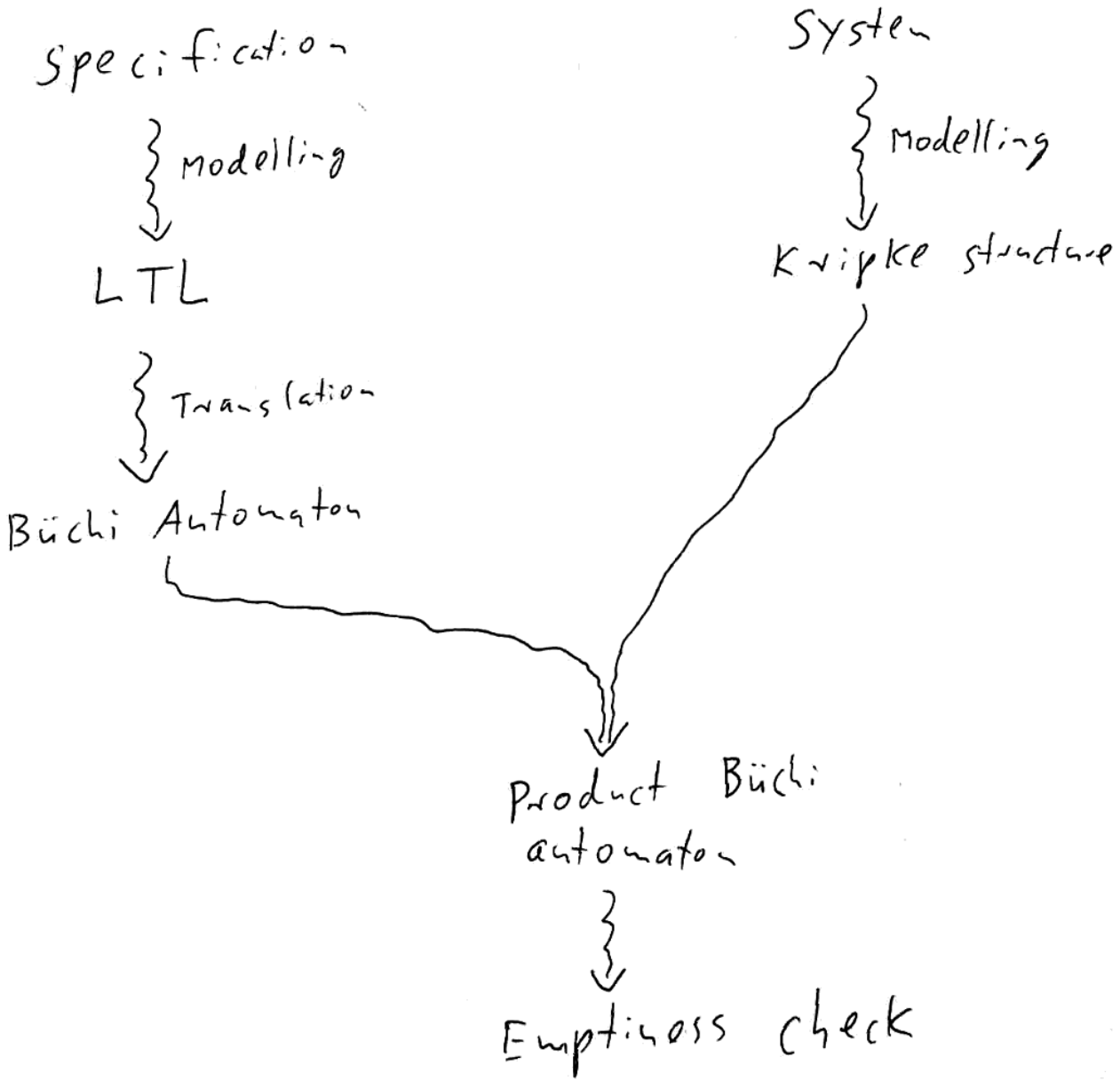
Property

There exists a trace on which when g_1 is true for the first time, g_1 can stay true together with e_2 immediately afterwards.

In CTL

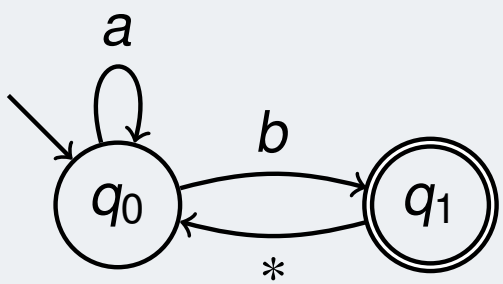
$$\mathbf{E}(\neg g_1 \,\mathcal{U}\, (g_1 \wedge \mathbf{EX}(g_1 \wedge e_2)))$$

Overview


Specification
modelling
LTL
Translation
Büchi Automaton
System
Modelling
Kripke structure
Product Büchi automaton
Emptiness check

Büchi automata by example

Example automaton 1 for $\Sigma = \{a, b\}$



An example *run* of the automaton for a word

$\rho =$	a	b	a	a	a	b	a	a	b	b	b	$\ldots$
$\pi =$	$q0$	$q0$	$q1$	$q0$	$q0$	$q0$	$q1$	$q0$	$q0$	$q1$	$q0$	$q1$ $\ldots$

Product construction – formal definition

Definition

Let $\mathcal{K} = (S, \text{AP}, S_0, T, L)$ be a Kripke structure and $\mathcal{A} = (Q, \Sigma, Q_0, \delta, F)$ be a Büchi automaton with $\Sigma = 2^{\text{AP}}$. We define the product Büchi automaton $\mathcal{B} = (Q', \Sigma, Q_0', \delta', F')$ with:

- $Q' = S \times Q$
- $Q_0' = S_0 \times Q_0$
- $\delta' = \{((q, s), x, (q', s')) \in Q' \times \Sigma \times Q' \mid (q, x, q') \in \mathcal{A}, (s, s') \in T, L(s) = x\}$
- $F = S \times F$

Proposition (without proof here)

$\mathcal{B}$ accepts exactly those words that accepted by $\mathcal{A}$ **and** are traces of $\mathcal{K}$ at the same time.

Checking Büchi automata language emptiness

A classical question in automata theory

Can we check an automaton for *language emptiness*?

Automata over finite words

We can test this by checking if there exists a reachable accepting state.

Automata over infinite words

We can test this by checking if there exists an accepting *lasso*.

A lasso consists of a *lasso handle* and a *lasso cycle*, where the lasso cycle contains at least one accepting state.

The double-DFS algorithm

Shared variables

```
stack = []
stackB = []
visited = set([])
marked = set([])
```

First DFS

```
def dfsA(node):
if node in visited:
  return
stack.push(node)
visited.add(node)
for all (v, v') ∈ E with v==node:
  dfsA(v')
if  v ∈ F:
  dfsB(v')
stack.pop()
```

Second DFS

```
def dfsB(node):
if node in marked:
  return
stackB.push(node)
marked.add(node)
for all (v,v') in E with v==node:
  if v' in stack:
    lasso found
  dfsB(v'):
stackB.pop()
```

Calling dfsa

```
for all v ∈ Q0:
  dfsA(v)
```

Translating from LTL to Büchi automata

Aim of the translation

Given an LTL formula, we want to translate an LTL formula to a Büchi automaton such that the language of the Büchi automaton is exactly the set of words satisfying the LTL formula.

Note that this is the last missing piece for a full model checking workflow!

Approach presented in the following

We do a multi-step translation:

LTL

$\Downarrow$

LTL in negation normal form

$\Downarrow$

Alternating Büchi automaton

$\Downarrow$

(Non-deterministic) Büchi automaton

LTL: Translation to NNF

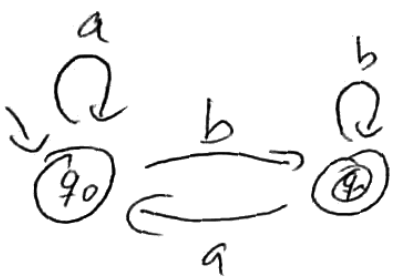
Equivalences

We utilize some equivalences (for all LTL subformulas ψ):

- $\neg \mathbf{F} \psi \equiv \mathbf{G} \neg \psi$
- $\neg \mathbf{G} \psi \equiv \mathbf{F} \neg \psi$
- $\neg \mathbf{X} \psi \equiv \mathbf{X} \neg \psi$
- $\neg (\psi \mathcal{U} \psi') \equiv \neg \psi \mathbf{R} \neg \psi'$
- $\neg (\psi \mathbf{R} \psi') \equiv \neg \psi \mathcal{U} \neg \psi'$
- $(\neg \neg \psi \equiv \psi)$

Branching modes

Det. Aut



One word

a b a b b a ...

One run

$q_0\ q_1 \ldots$

$w \in \mathcal{L}(A)$ if the run accepts

Non-det. Aut.

One word

a b a b b a ...

Many runs

$q_0 q_0 q_1 q_0 \ldots$

$q_0 q_0 q_0 q_0 \ldots$

$w \in \mathcal{L}(A)$ if <u>one</u> run accepts

Universal aut.

One word

a b a b b a ...

Many runs

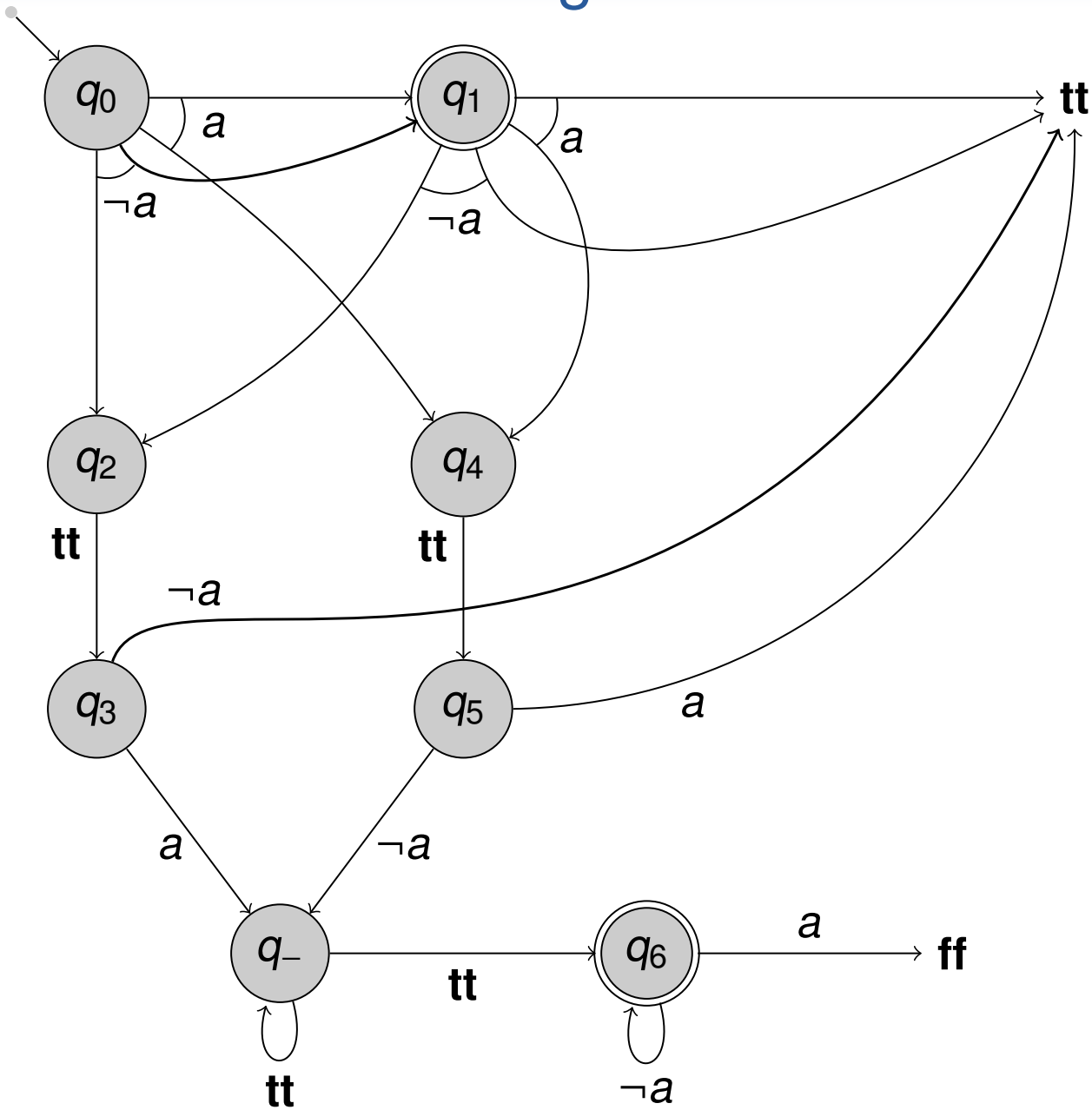
$q_0\ q_0 q_0 q_0 \ldots$

$q_0\ q_1$

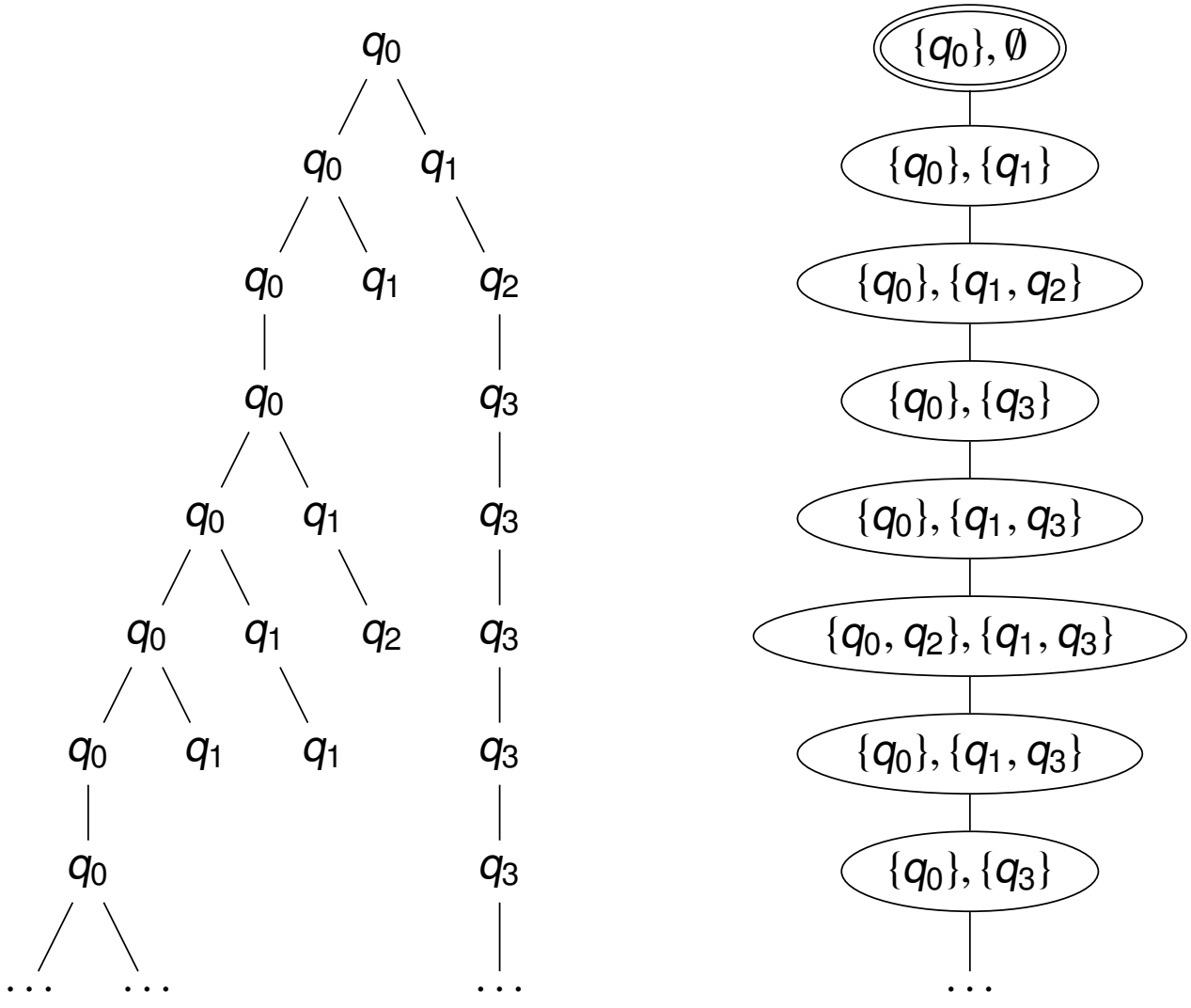
$q_0\ q_0 q_0 q_1 \ldots$

$w \in \mathcal{L}(A)$ if <u>all</u> infinite runs accept.

An alternating automaton

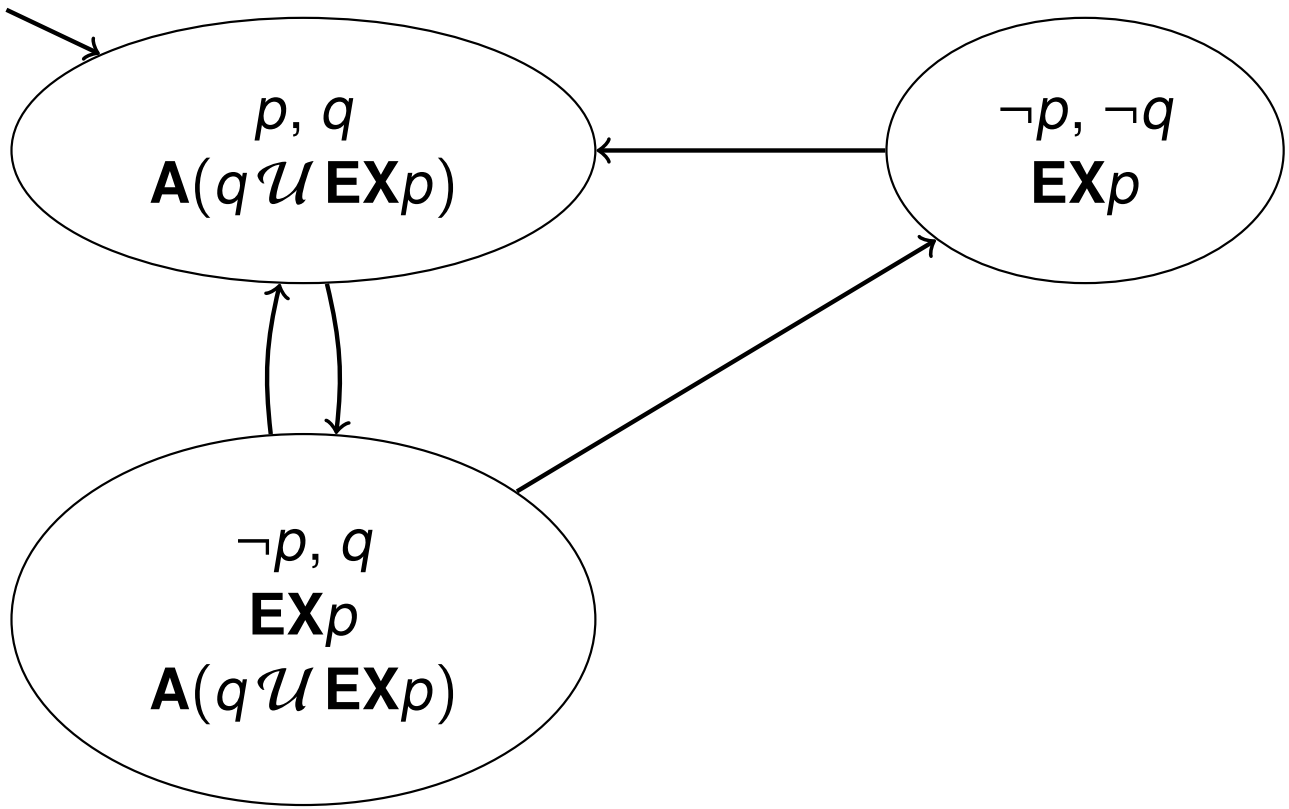


The Miyano-Hayashi Construction



CTL Example using a simple Kripke structure

CTL formula of interest: $\mathbf{A}(q \,\mathcal{U}\, \mathbf{EX}p)$



Labeling states with CTL subformulas (excerpt)

$\mathbf{EF}\psi$

Perform the following steps:

- Label every state satisfying ψ with $\mathbf{EF}\psi$.
- Label every state with a successor state labeled by $\mathbf{EF}\psi$ by $\mathbf{EF}\psi$ as well.
- Repeat the previous step until no more states can be labeled with $\mathbf{EF}\psi$.

$\mathbf{AG}\psi$

Let us use the fact that $\mathbf{AG}\psi \equiv \mathbf{EF}\neg\psi$. Using this fact, we can execute the following approach:

- Initially, label every state satisfying ψ with $\mathbf{AG}\psi$.
- Remove the $\mathbf{AG}\psi$ label of every state with a successor state not labelled by $\mathbf{AG}\psi$.
- Repeat the previous step until no more state labels can be removed.

Modal μ-calculus (short version!)

Syntax

Modal μ-calculus is an extension of propositional logic. For some given set of variable symbols $\mathcal{V}$, formulas in modal μ-calculus over some set of variables with defined values V and some set of atomic proposition AP is are defined as follows (for $p \in \mathrm{AP}$ and $x \in \mathcal{V} \setminus V$):

$$\begin{aligned}\psi(V,\mathrm{AP}) :== \top \mid \bot \mid p \mid x \mid \Box\psi(V,\mathrm{AP}) \mid \Diamond\psi(V,\mathrm{AP}) \\ \mid \psi(V,\mathrm{AP}) \cup \psi(V,\mathrm{AP}) \mid \psi(V,\mathrm{AP}) \cap \psi(V,\mathrm{AP}) \\ \mid \mu x.\psi(V \cup \{x\},\mathrm{AP}) \mid \nu x.\psi(V \cup \{x\},\mathrm{AP})\end{aligned}$$

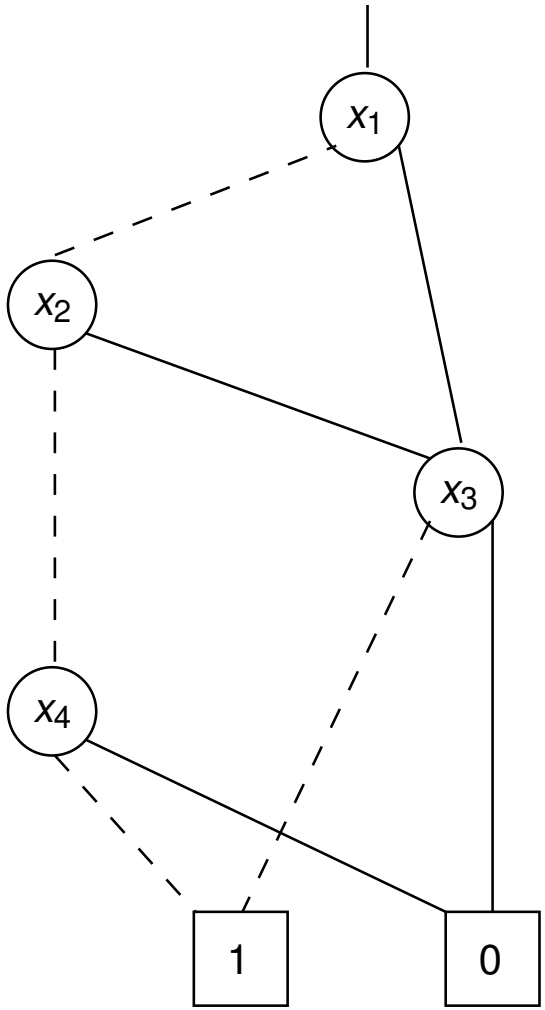
A more thorough formalization of CTL operators

Using fixed point equation

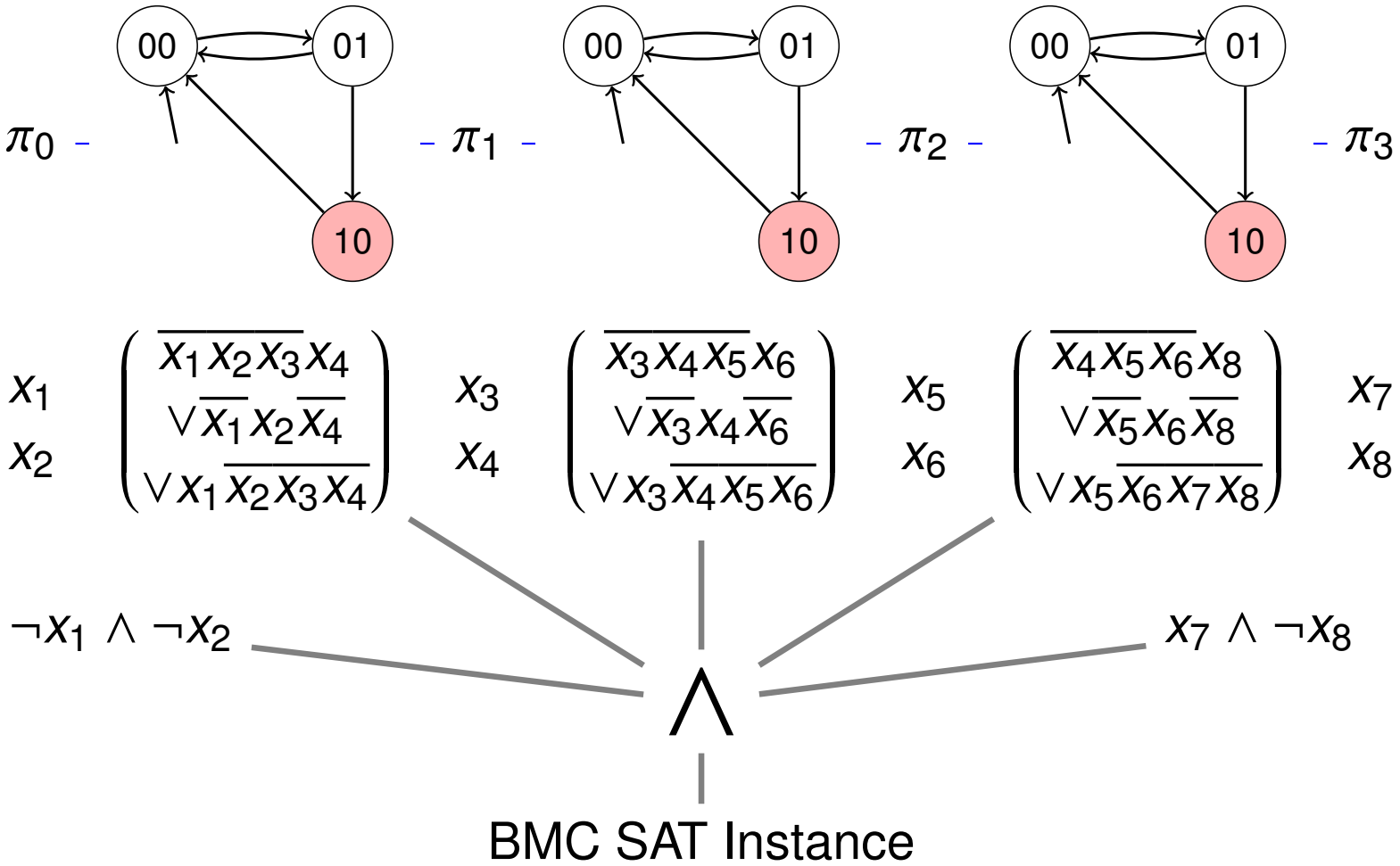
We can formalize these rules as follows:

$\mathbf{AX}\psi$	$\Box\psi$
$\mathbf{EX}\psi$	$\Diamond\psi$
$\mathbf{AG}\psi$	$\nu X.\psi \cap \Box X$
$\mathbf{EG}\psi$	$\nu X.\psi \cap \Diamond X$
$\mathbf{AF}\psi$	$\mu X.\psi \cup \Box X$
$\mathbf{EF}\psi$	$\mu X.\psi \cup \Diamond X$
$\mathbf{A}(\psi\,\mathcal{U}\,\psi')$	$\mu X.\psi' \cup (\psi \cap \Box X)$
$\mathbf{E}(\psi\,\mathcal{U}\,\psi')$	$\mu X.\psi' \cup (\psi \cap \Diamond X)$
$\mathbf{A}(\psi\,\mathbf{R}\,\psi')$	$\nu X.\psi' \cap (\psi \cup \Box X)$
$\mathbf{E}(\psi\,\mathbf{R}\,\psi')$	$\nu X.\psi' \cap (\psi \cup \Diamond X)$

Reduced Ordered Binary Decision Diagrams



SAT solving for bounded model checking

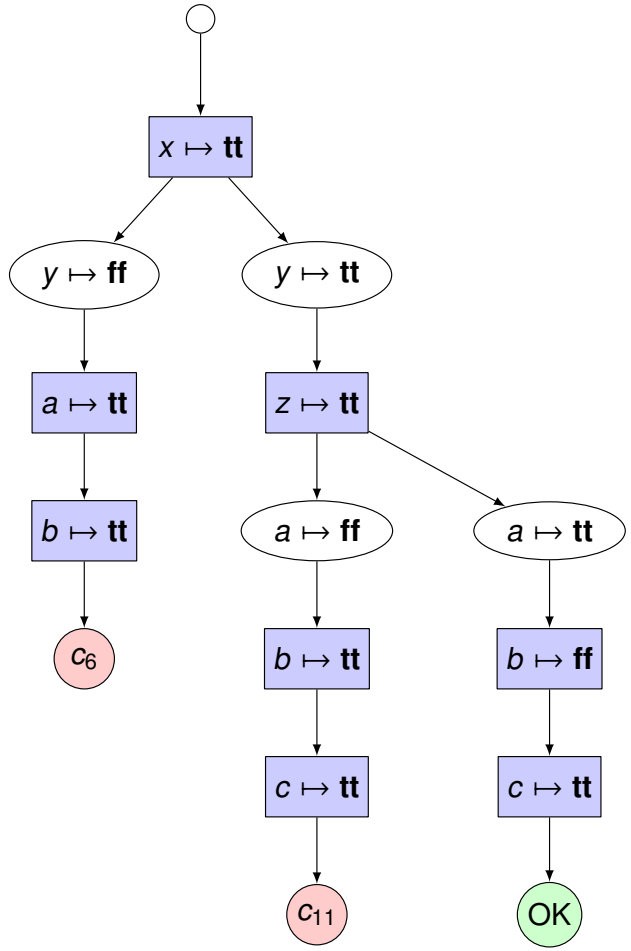


A first example: Sudoku

1				7				
		9	3		8	2		
		2			9		8	
	8					4	5	2
			6					
		1					3	
	4			1		3	7	
		6	7			9		
	1							

Unit propagation

$$\psi(x,y,z,a,b,c) = \underbrace{x}_{c_1} \wedge \underbrace{(\neg x \vee y \vee z)}_{c_2} \wedge \underbrace{(\neg y \vee z)}_{c_3}$$
$$\wedge \underbrace{(y \vee a)}_{c_4} \wedge \underbrace{(y \vee b)}_{c_5} \wedge \underbrace{(\neg a \vee \neg b)}_{c_6}$$
$$\wedge \underbrace{(\neg y \vee a \vee b)}_{c_7} \wedge \underbrace{(\neg a \vee b \vee c)}_{c_8}$$
$$\wedge \underbrace{(a \vee b \vee \neg c)}_{c_9} \wedge \underbrace{(a \vee \neg b \vee c)}_{c_{10}}$$
$$\wedge \underbrace{(a \vee \neg b \vee \neg c)}_{c_{11}}$$

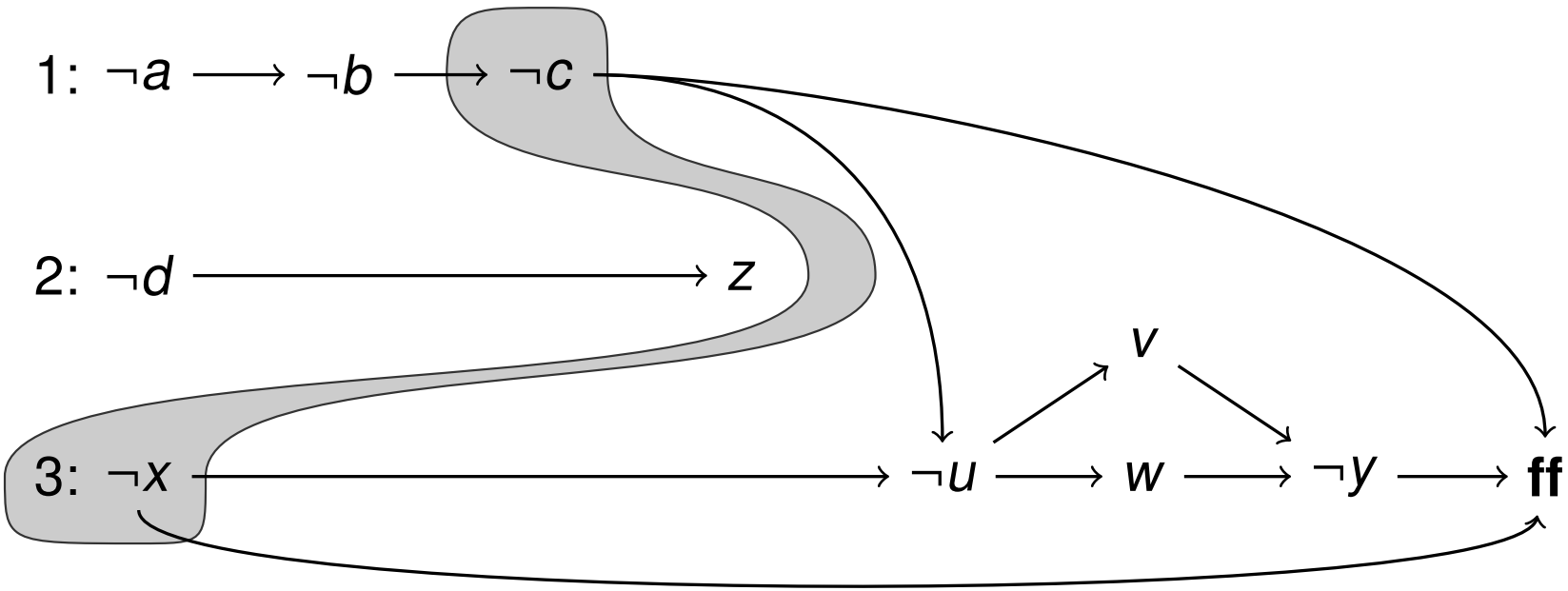


The complete DPLL algorithm

Algorithm 2 The complete DPLL algorithm with *unit propagation* and with the *pure literal rule*. The parameter $\mathcal{V}$ represents the set of variables, ψ is the CNF formula, and $A : \mathcal{V} \rightharpoonup \mathbb{B}$ is the current partial assignment.

```
1:  function SEARCH(𝒱,ψ,A)
2:      for all assignments v_k = b_k implied by (ψ, A) by unit propagation do
3:          A := A ∪ {v_k ↦ b_k}
4:      end for
5:      for all assignments v_k = b_k implied by (ψ, A) by pure literal elimination do
6:          A := A ∪ {v_k ↦ b_k}
7:      end for
8:      if some clause in ψ is falsified by A then
9:          return ∅
10:     end if
11:     if |A| = |𝒱| then
12:         return A
13:     end if
14:     Pick a variable v that is not yet in the domain of A
15:     A' ← SEARCH(𝒱,ψ,A ∪ {v ↦ false})
16:     if A' ≠ ∅ then return A'
17:     A' ← SEARCH(𝒱,ψ,A ∪ {v ↦ true})
18:     if A' ≠ ∅ then return A'
19:     return ∅
20: end function
```

A cut through an implication graph



DPLL+Clause learning = CDCL

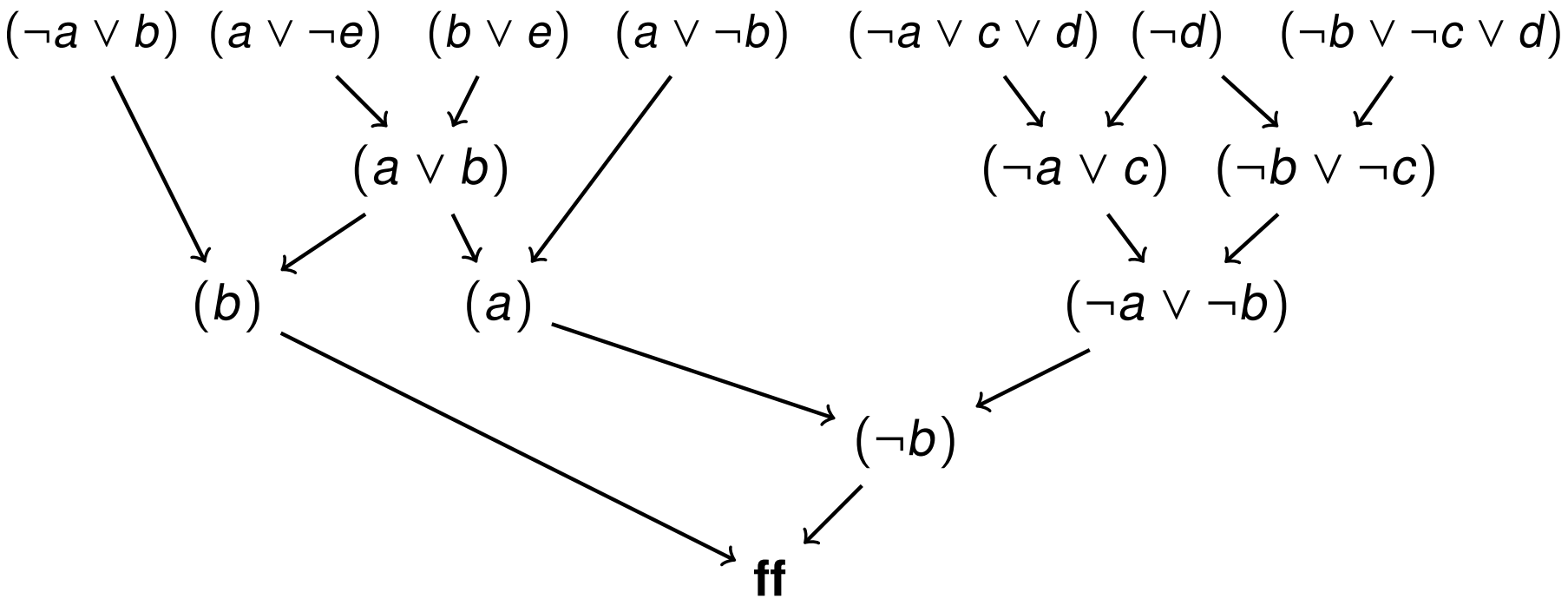
Core techniques in CDCL solvers

- Backtracking
- Unit propagation
- Pure literal elimination
- Conflict-driven clause learning

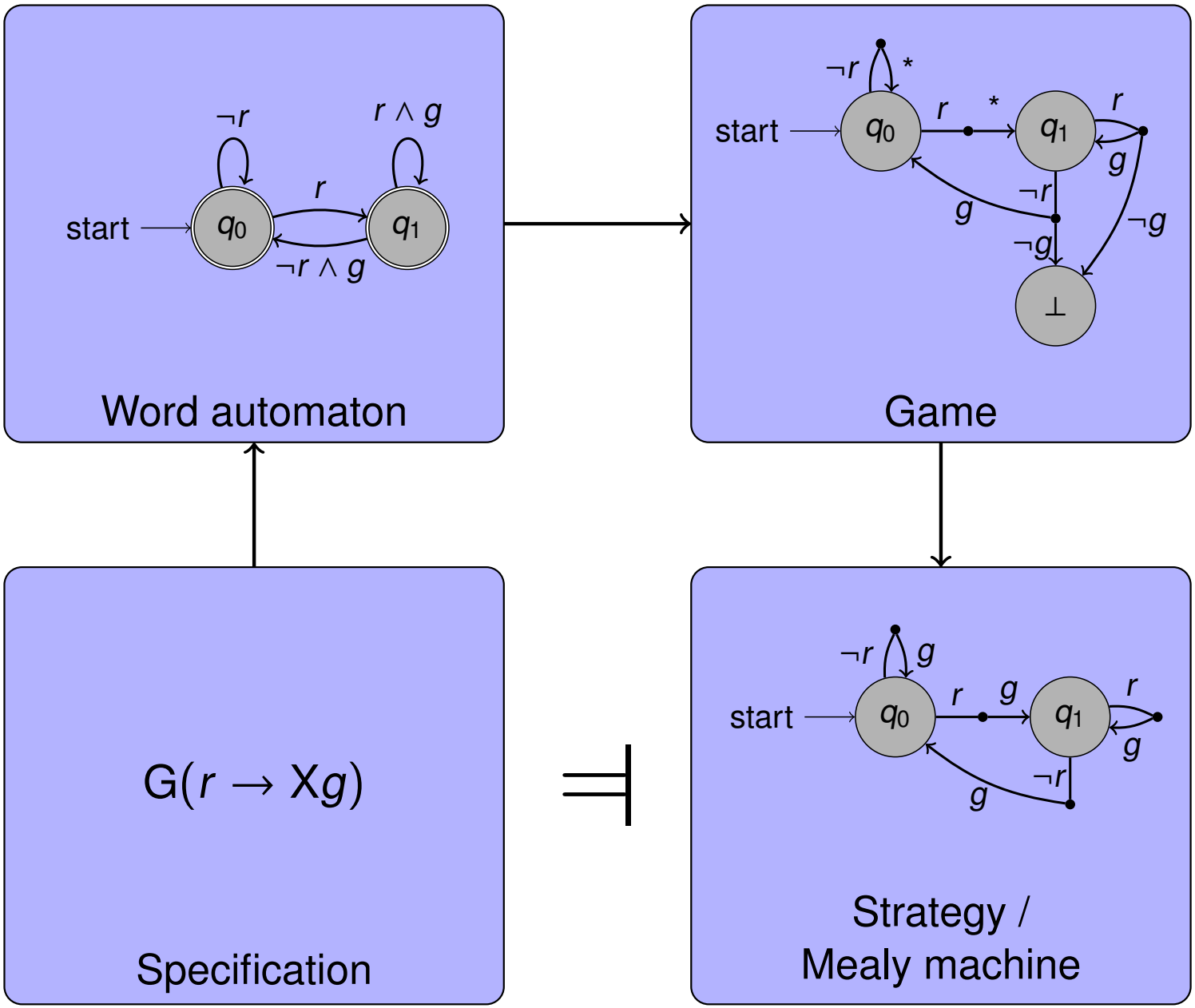
Main publications for CDCL were made between 1997 and 2004.[a]

[a] Handbook of Satisfiability, page 119, first paragraph of Section 3.6.3.3

Analyzing the Pidgeon Hole Principle SAT instances (3)



General synthesis workflow



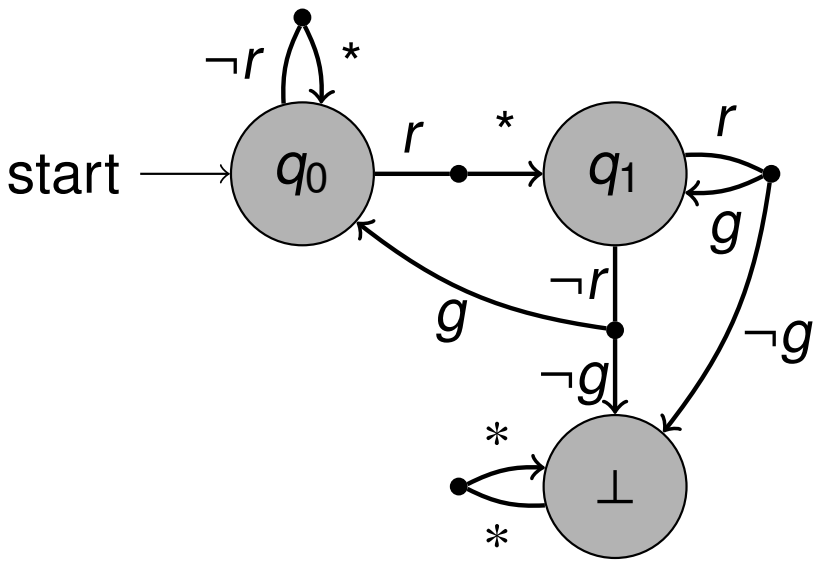
Games

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

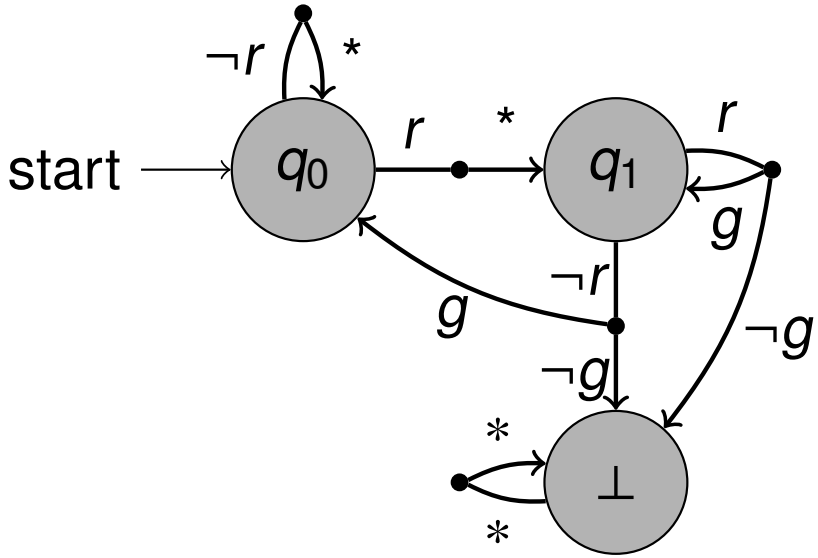
Additionally, there is some initial position.



Games for synthesis

Strategies

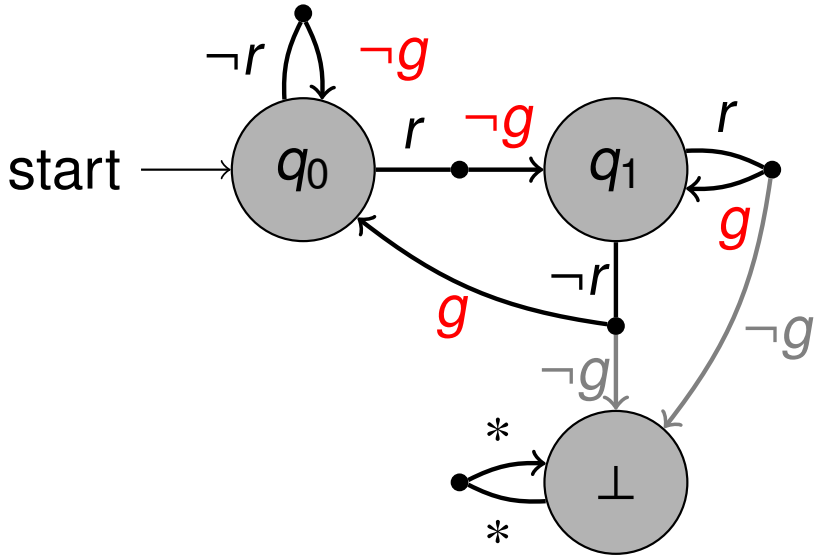
One player is the **system player**, whereas the other player is the **environment player**.
If player $p \in \{0, 1\}$ has a **stategy** to win, then she can enforce to win by playing the strategy. We say that player p **wins the game** in such a case.



Games for synthesis

Strategies

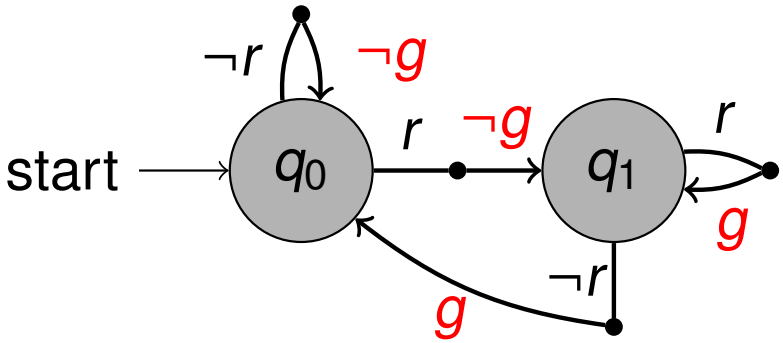
One player is the **system player**, whereas the other player is the **environment player**.
If player $p \in \{0, 1\}$ has a **stategy** to win, then she can enforce to win by playing the strategy. We say that player p **wins the game** in such a case.



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.

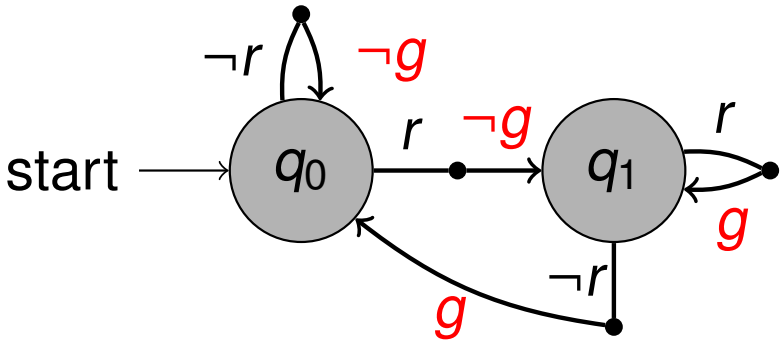


This is a Mealy Machine!

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.

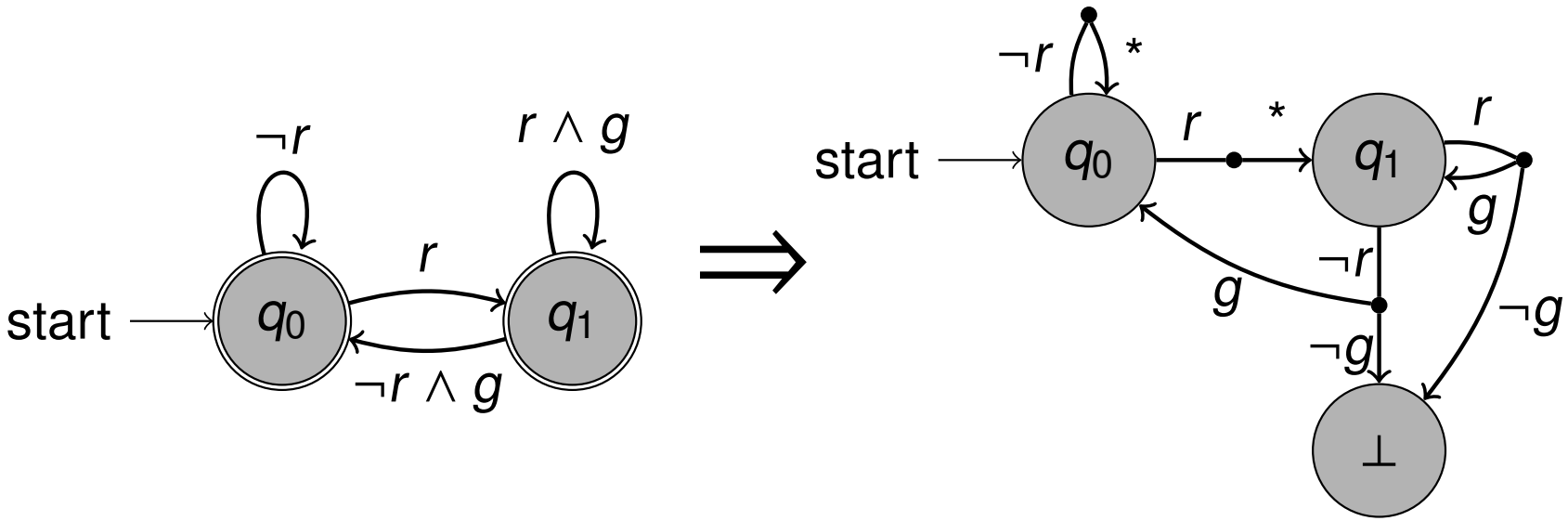


This is a Mealy Machine!

Strategies in synthesis games

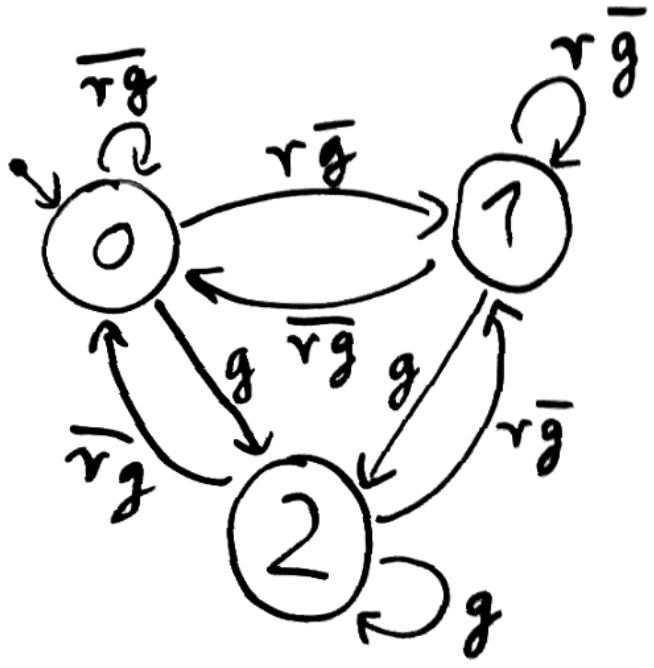
In games that correspond to a specification, winning strategies for the system player represent Mealy (or Moore) machines that satisfy the specification.

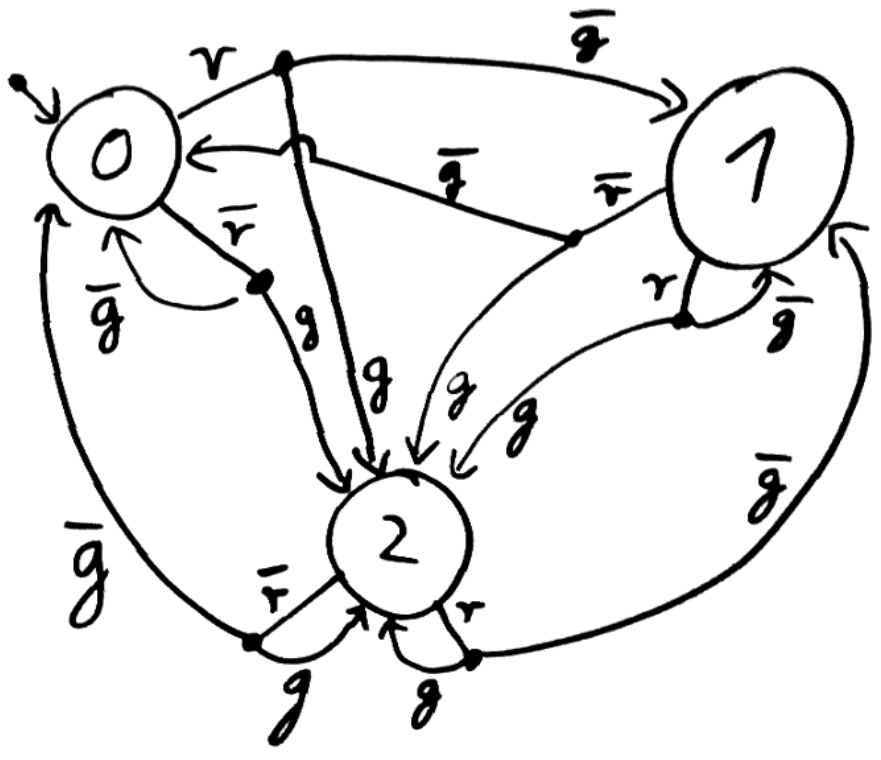
Building safety games from deterministic safety automata



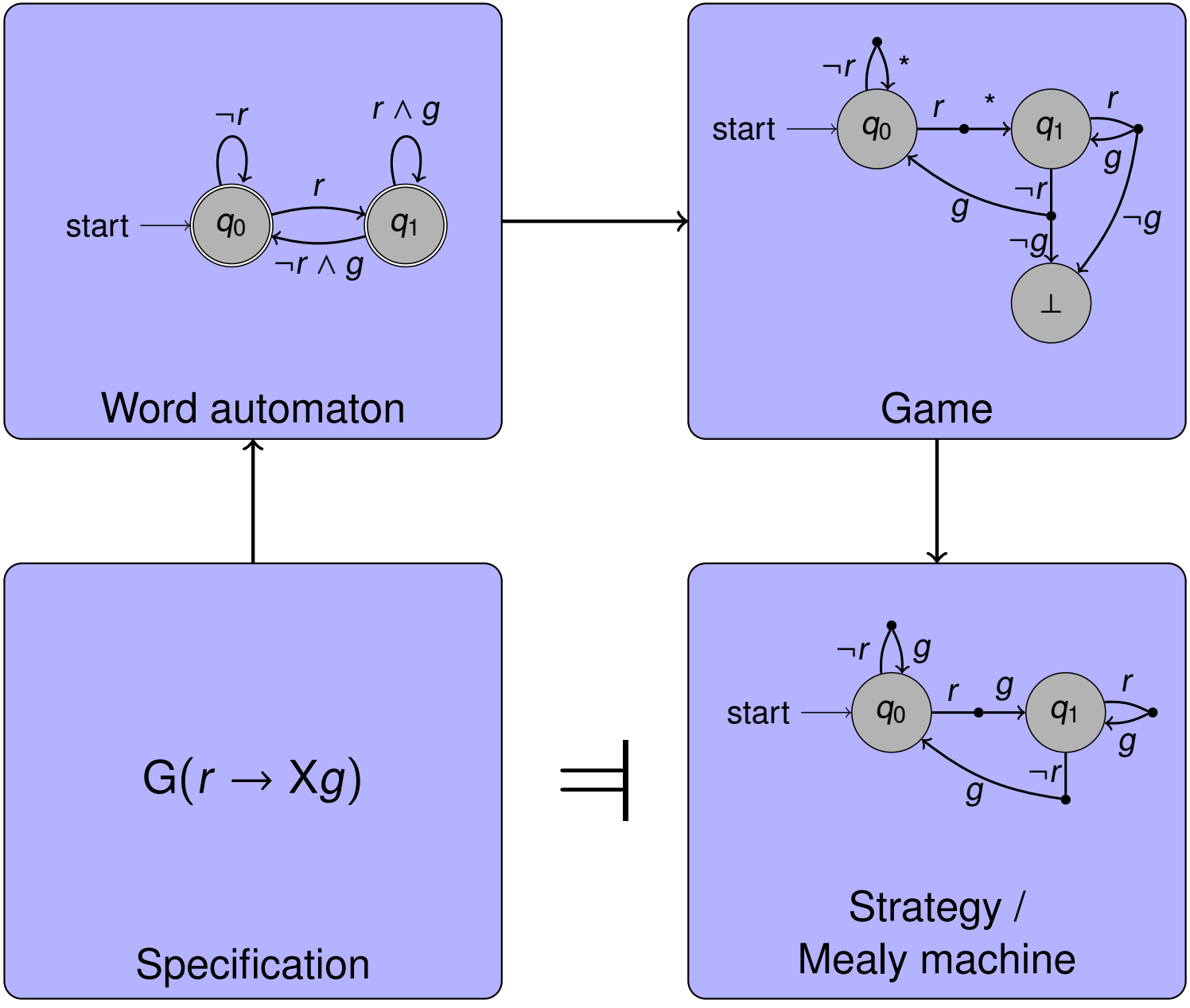
Parity automata and games

$GFr \rightarrow GFg$

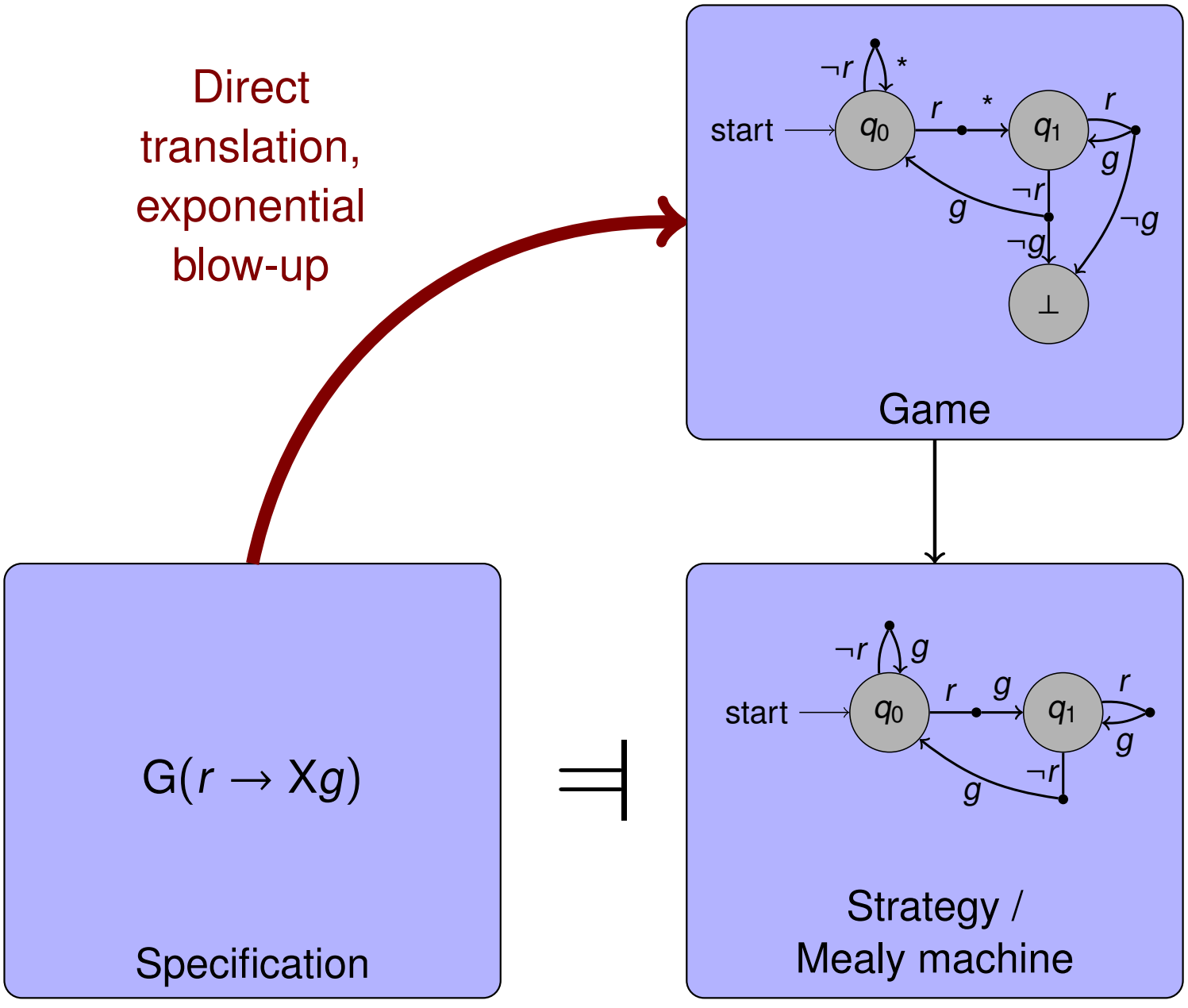




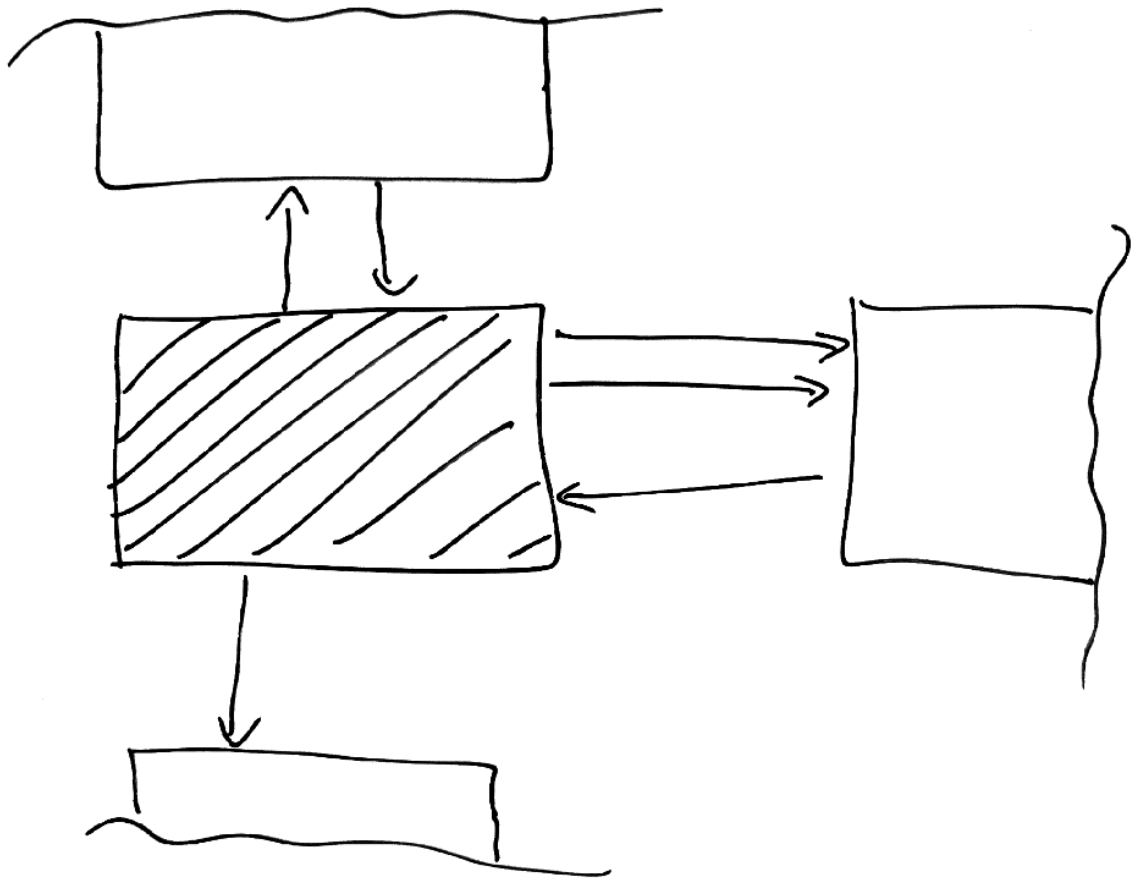
GR(1) Synthesis (Bloem et al., 2012) – Main idea (2)



GR(1) Synthesis (Bloem et al., 2012) – Main idea (2)



Assumptions and guarantees in specifications



Specification shape

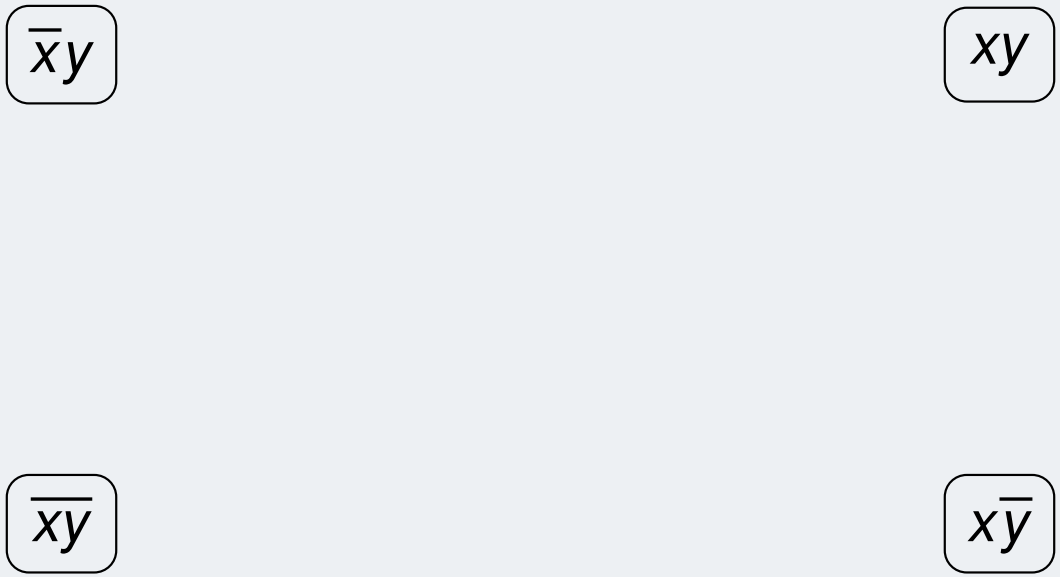
$$\left(\bigwedge \text{Assumptions}\right) \rightarrow \left(\bigwedge \text{Guarantees}\right)$$

Building the game

Relevant specification parts for building the game

- Safety assumptions: $\mathbf{G}(\neg x \vee \neg \mathbf{X}\, x)$
- Safety guarantees: $\mathbf{G}((\neg x \wedge y) \rightarrow \mathbf{X}\, x)$

Game

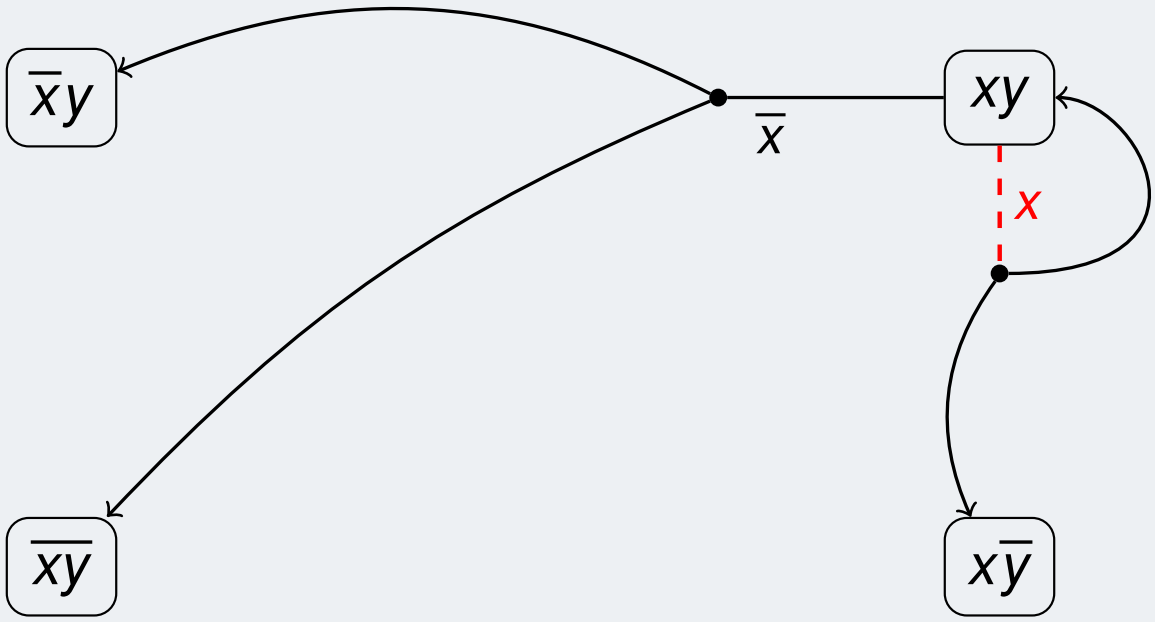


Building the game

Relevant specification parts for building the game

- Safety assumptions: $\mathbf{G}(\neg x \vee \neg \mathbf{X}\, x)$
- Safety guarantees: $\mathbf{G}((\neg x \wedge y) \rightarrow \mathbf{X}\, x)$

Game

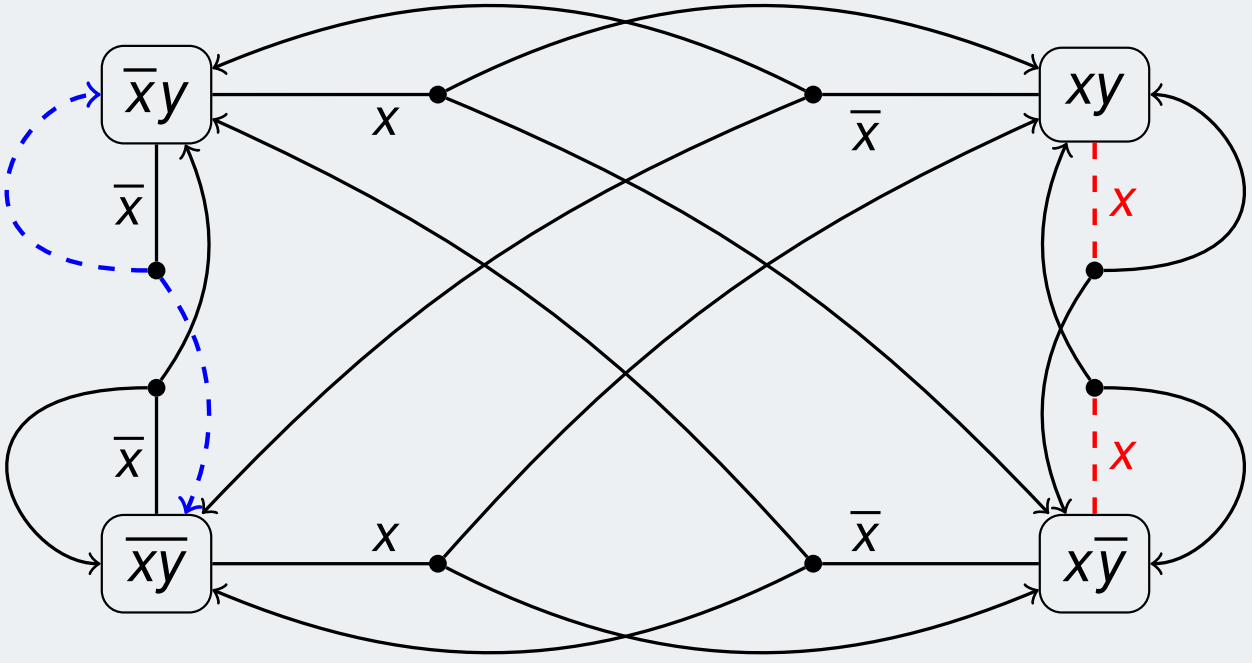


Building the game

Relevant specification parts for building the game

- Safety assumptions: $\mathbf{G}(\neg x \vee \neg \mathbf{X}\, x)$
- Safety guarantees: $\mathbf{G}((\neg x \wedge y) \rightarrow \mathbf{X}\, x)$

Game



The final GR(1) fixpoint

$$\nu Z.\ \bigcap_{j\in\{1,\ldots,n\}}\ \mu Y.\ \bigcup_{i\in\{1,\ldots,m\}}\ \nu X.\ \mathsf{EnfPre}\Big(\ (Z' \cap \psi_j^g)\ \cup\ Y'\ \cup\ (\neg\psi_i^a \cap X')\ \Big)$$

The final GR(1) fixpoint

$$\nu Z.\ \bigcap_{j\in\{1,\ldots,n\}} \mu Y.\ \bigcup_{i\in\{1,\ldots,m\}} \nu X.\ \mathsf{EnfPre}\Big((Z' \cap \psi_j^g) \cup Y' \cup (\neg\psi_i^a \cap X') \Big)$$

Reaching the next system goal

The final GR(1) fixpoint

$$\nu Z.\ \bigcap_{j\in\{1,\ldots,n\}}\ \mu Y.\ \bigcup_{i\in\{1,\ldots,m\}}\ \nu X.\ \mathsf{EnfPre}\Big(\big(Z' \cap \psi_j^g\big) \cup Y' \cup \big(\neg\psi_i^a \cap X'\big)\Big)$$

Reaching the next system goal

Getting closer to the next system goal

The final GR(1) fixpoint

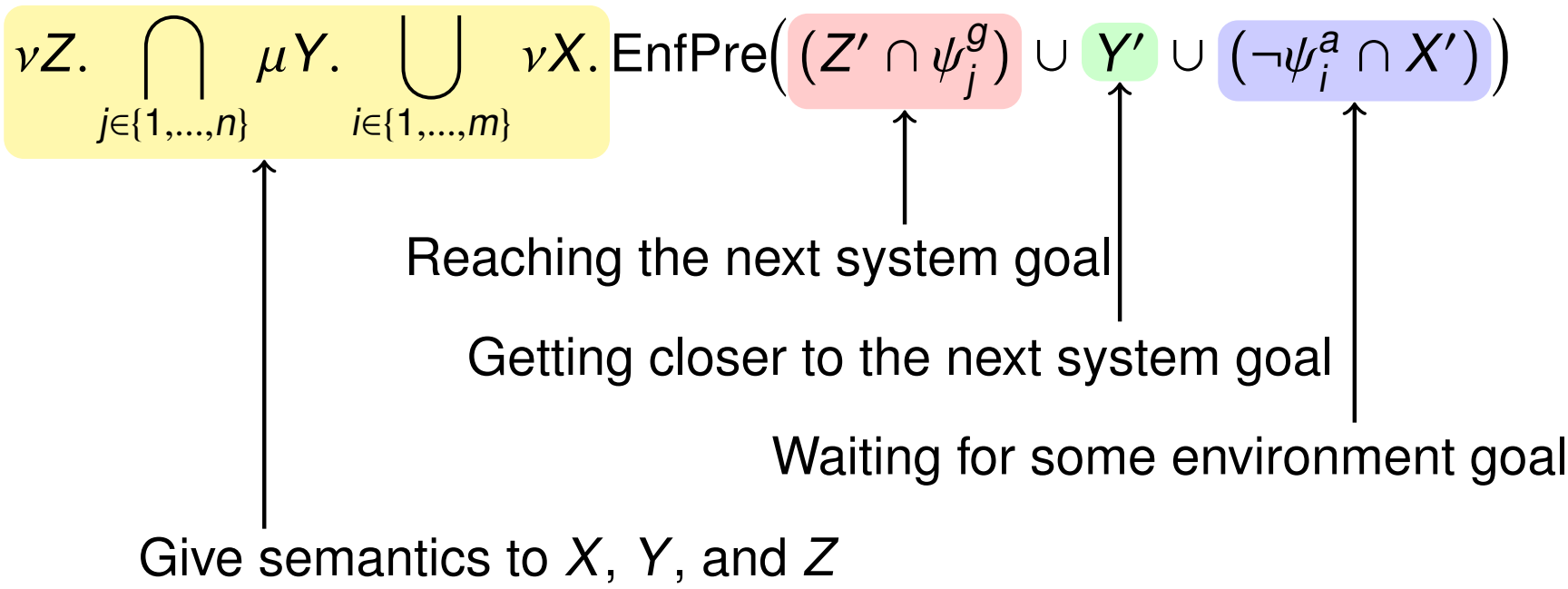
$$\nu Z.\ \bigcap_{j\in\{1,\ldots,n\}}\ \mu Y.\ \bigcup_{i\in\{1,\ldots,m\}}\ \nu X.\ \mathsf{EnfPre}\Big(\big(Z' \cap \psi_j^g\big) \cup Y' \cup \big(\neg\psi_i^a \cap X'\big)\Big)$$

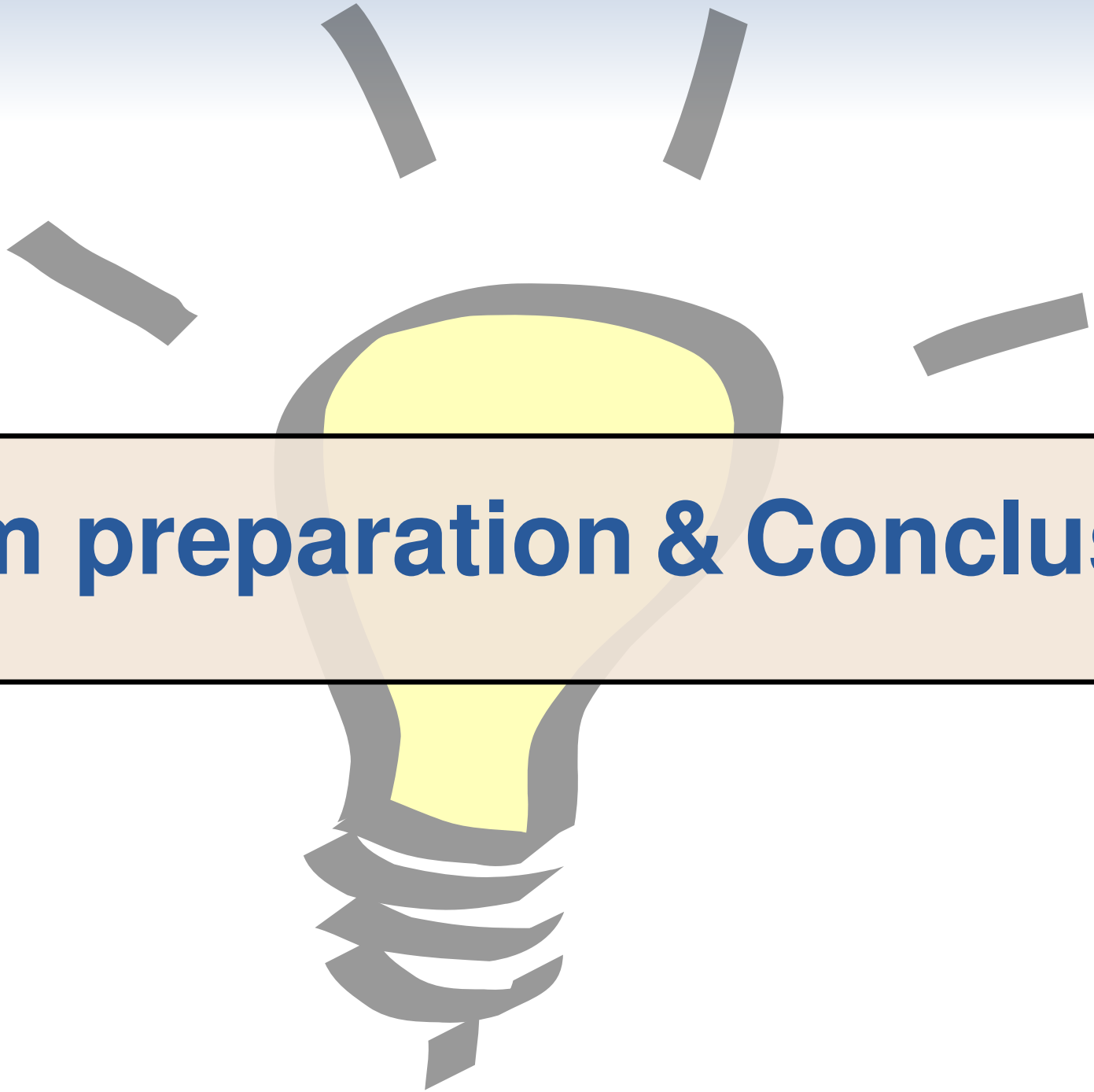
Reaching the next system goal

Getting closer to the next system goal

Waiting for some environment goal

The final GR(1) fixpoint





Exam preparation & Conclusion

Some hints on typical oral exam questions

SOME answers that you should be able to give

- What was the course about? What were the main questions? Why did we deal with "Model Checking and Games" at all?
- What were the main topics of the course?
- For each of the main topics:
  - Why did we deal with them?
  - What were the main ideas?
  - How can the main ideas be applied to verify actual systems?
  - What were the main constructions/main algorithms/main ideas?
  - What were the main ideas of the algorithms/constructions?
  - Demonstrate that you know roughly how the constructions/algorithms work.

Thanks for participating in this course!

Appointments

If you want to discuss some lecture material as part of your exam preparation, you can book an appointment for the lecturer's office hours at:

`https://appointments.ruediger-ehlers.de`

References I

Roderick Bloem, Barbara Jobstmann, Nir Piterman, Amir Pnueli, and Yaniv Sa'ar. Synthesis of reactive(1) designs. *J. Comput. Syst. Sci.*, 78(3):911–938, 2012.

Edmund M. Clarke and E. Allen Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In Dexter Kozen, editor, *Logics of Programs, Workshop, Yorktown Heights, New York, USA, May 1981*, volume 131 of *Lecture Notes in Computer Science*, pages 52–71. Springer, 1981. doi: 10.1007/BFb0025774. URL https://doi.org/10.1007/BFb0025774.

Amir Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA, 31 October - 1 November 1977*, pages 46–57, 1977. doi: 10.1109/SFCS.1977.32. URL https://doi.org/10.1109/SFCS.1977.32.