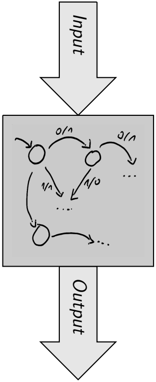
Synthesis from temporal logic

Guest Lecture
219C, Sanjit A. Seshia

26^{th} November 2012

Verification and synthesis of reactive systems

Verification:



$+$

$$\mathbf{G}(u = 0 \rightarrow \mathbf{X}(v = 1))$$

$\Rightarrow$

Satisfied

Not satisfied
(+ counter-example)

Synthesis:

$\mathbf{G}(u = 0 \rightarrow$
$\mathbf{X}(v = 1))$

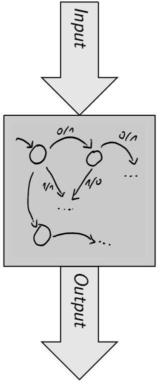
$+$

Input $= \{u, \ldots\}$
Output $= \{v, \ldots\}$

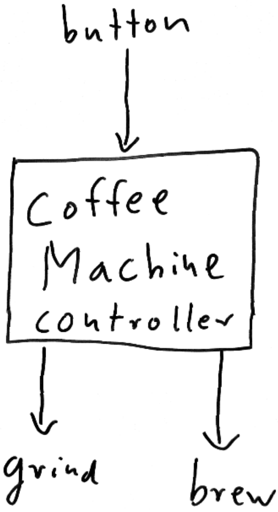
$\Rightarrow$

Realisable $\longrightarrow$

Not realisable

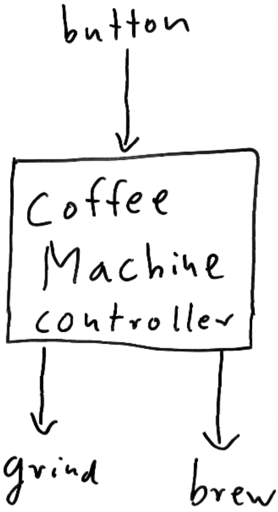


Synthesis of reactive systems - example



Atomic propositions

- $\mathsf{AP}_I = \{\textit{button}\}$
- $\mathsf{AP}_O = \{\textit{grind}, \textit{brew}\}$



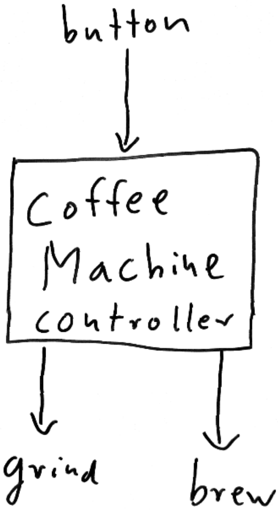
Atomic propositions

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

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

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

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.

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)

$$\begin{aligned}\mathbf{G}(\mathit{button} \quad \rightarrow \quad & (\mathit{grind} \;\wedge\; \mathbf{X}\,\mathit{grind} \;\wedge\; \mathbf{XX}\,(\mathit{brew} \wedge \neg \mathit{grind}) \\ \wedge \quad & \mathbf{XXX}\,(\mathit{brew} \wedge \neg \mathit{grind}) \;\wedge\; \mathbf{XXXX}\,(\mathit{brew} \wedge \neg \mathit{grind})))\end{aligned}$$

Synthesis of reactive systems - analyzing the example

Formal specification in linear-time temporal logic (LTL)

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

A surprise

The specification is *unrealisable*.

Example:

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

What do we want as a result?

The computed implementation should be...

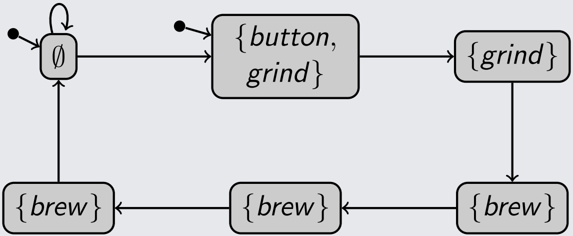
- ...finite-state,
- ...deterministic, and
- ...non-terminating and input-responsive

What do we want as a result?

The computed implementation should be...

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

Kripke structures

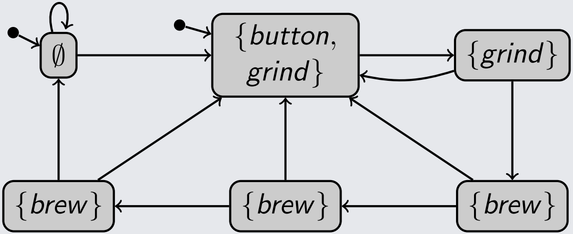


What do we want as a result?

The computed implementation should be...

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

Kripke structures

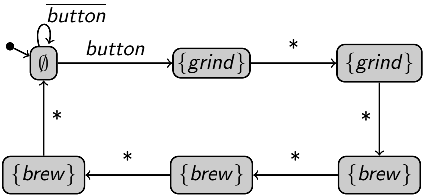


Adapted computation models

Moore machines

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

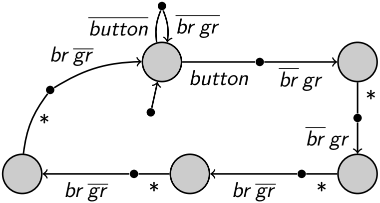
- Set of states S
- Input/output alphabets Σ^I/Σ^O
- Initial state s_0
- Transition function $\delta : S \times \Sigma^I \to S$
- State labeling: $L : S \to \Sigma^O$



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 $\delta : S \times \Sigma^I \to S \times \Sigma^O$

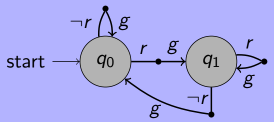


General synthesis workflow

$$G(r \rightarrow Xg)$$

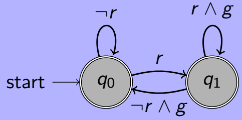
Specification

⫤



Mealy machine

General synthesis workflow

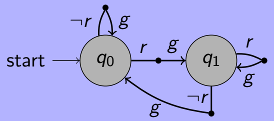


Word automaton

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

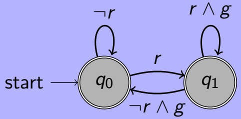
Specification

$$=\!|$$

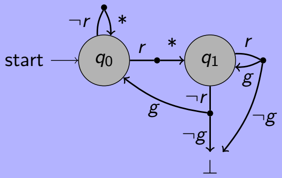


Mealy machine

General synthesis workflow



Word automaton

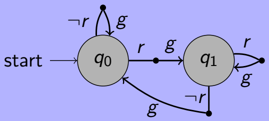


Game

$G(r \to Xg)$

Specification

$\Dashv$



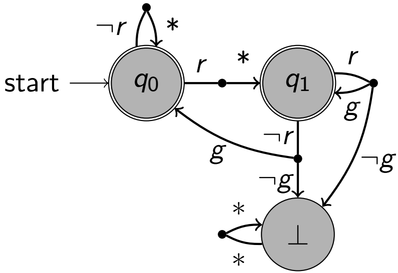
Strategy /
Mealy machine

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.

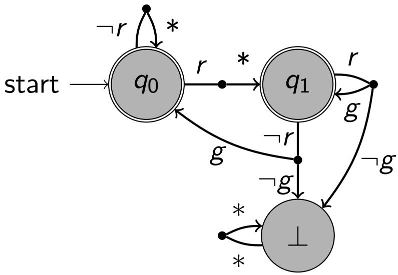


Games for synthesis

Strategies

One player is the **system player**, whereas the other player is the **environment player**.

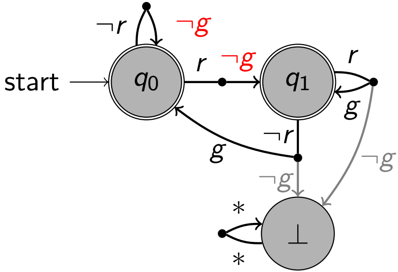
If player $p \in \{0,1\}$ has a **stategy** to win, then she can enforce to win a play by playing the strategy. We say that player p wins the game in such a case.



Games for synthesis

Strategies

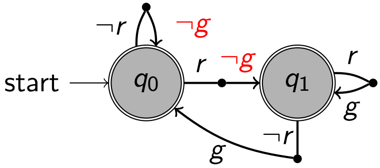
One player is the **system player**, whereas the other player is the **environment player**.
If player $p \in \{0, 1\}$ has a **stategy** to win, then she can enforce to win a play by playing the strategy. We say that player p wins the game in such a case.



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 a play by playing the strategy. We say that player p wins the game in such a case.

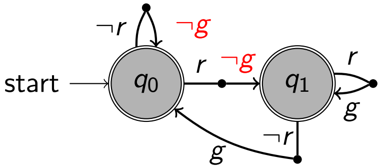


This is a Mealy Machine!

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 a play by playing the strategy. We say that player p wins the game in such a case.



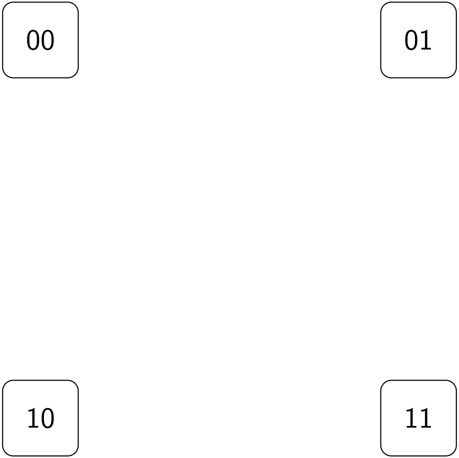
Strategies in Synthesis games

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

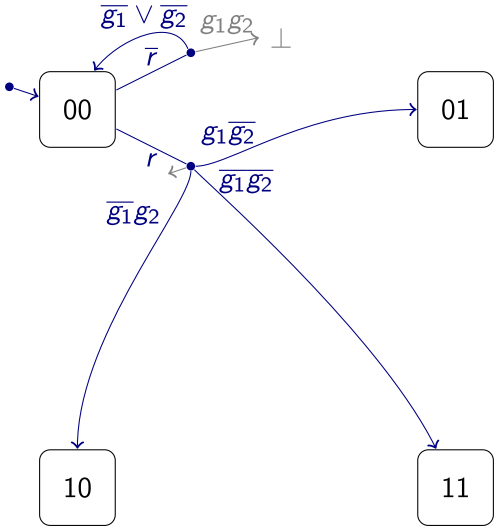
Main questions

1. How do we solve a game (determine the winner and a winning strategy for her)?
2. What winning condition do we need to use for general LTL?
3. How do we build games that correspond to specifications?

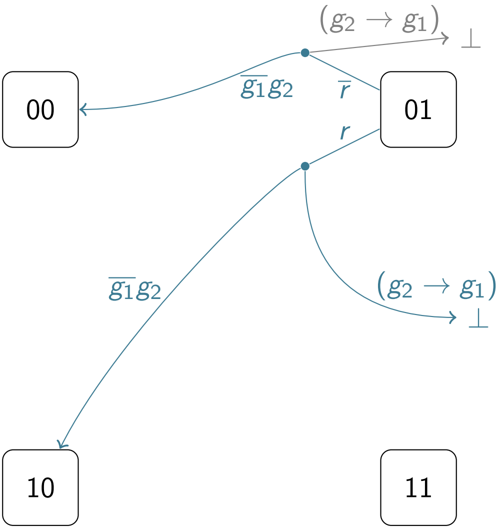
A more complicated (safety) game



A more complicated (safety) game

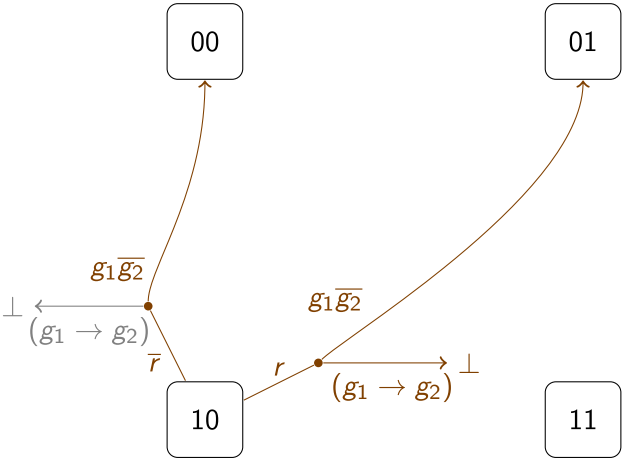


A more complicated (safety) game



3

A more complicated (safety) game

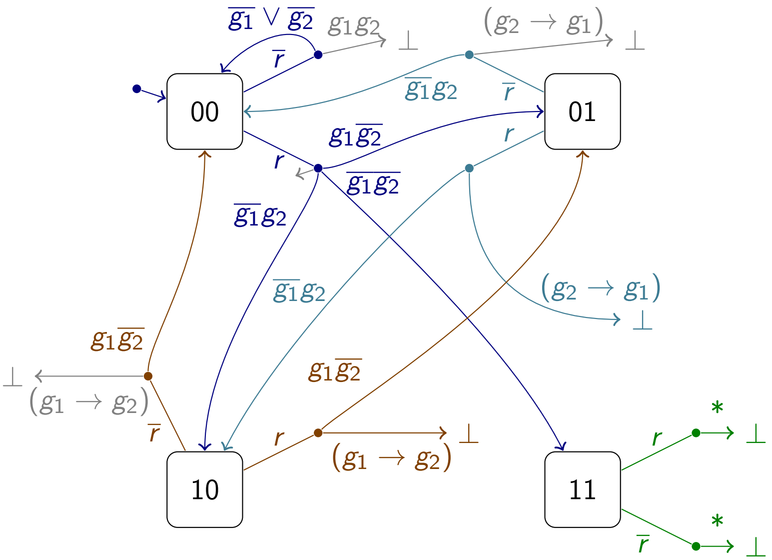


3

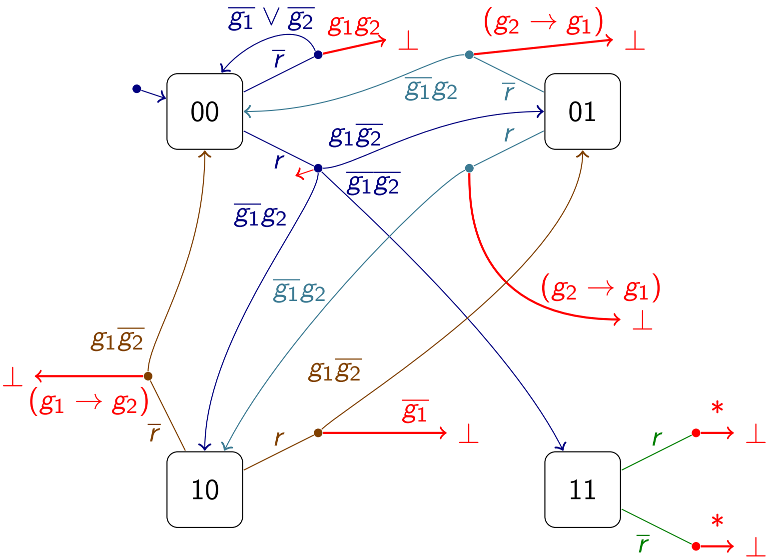
A more complicated (safety) game

3

A more complicated (safety) game

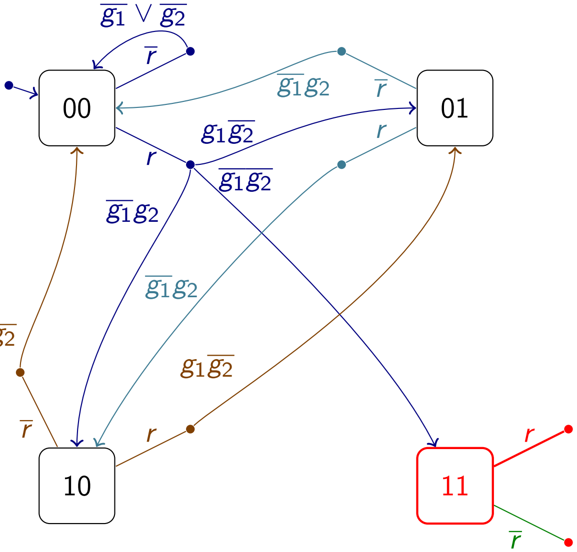


A more complicated (safety) game

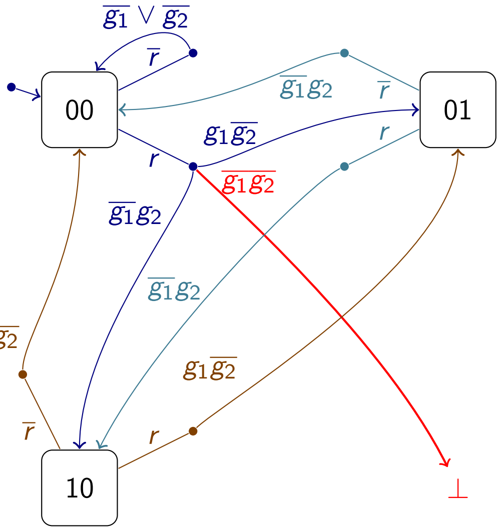


3

A more complicated (safety) game

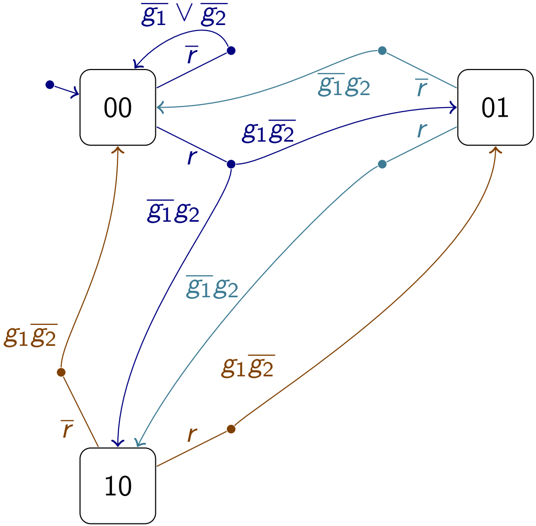


A more complicated (safety) game

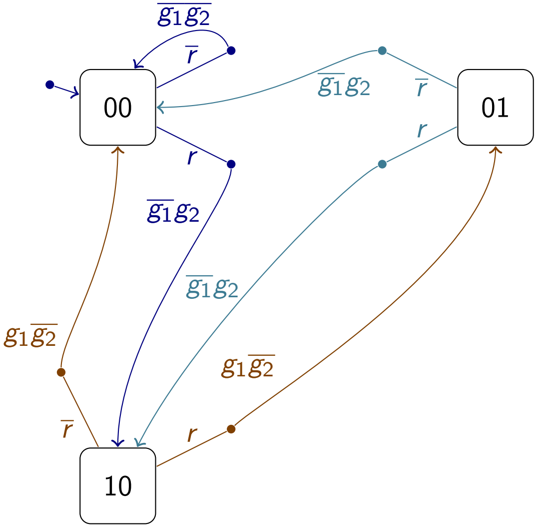


3

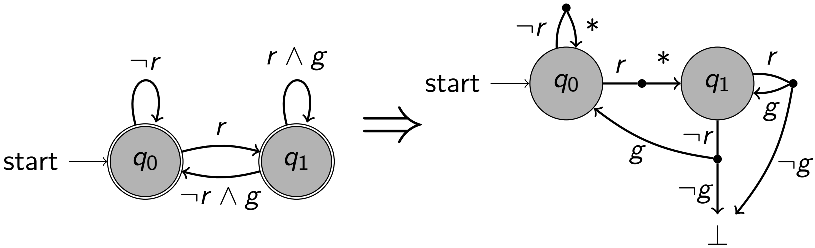
A more complicated (safety) game



A more complicated (safety) game

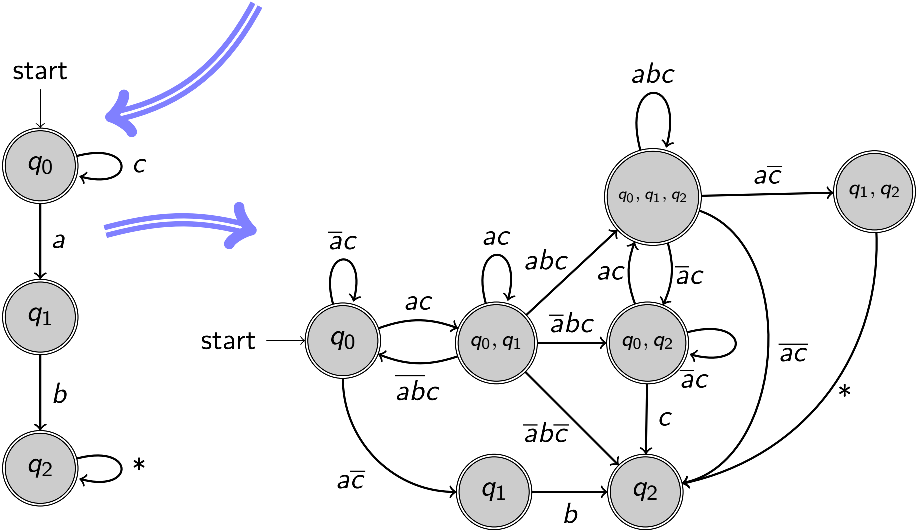


Building safety games from deterministic safety automata



Building deterministic safety automata

$\psi = (a \wedge \mathbf{X} b)\, \mathbf{R}\, c, \quad \mathrm{AP} = \{a, b, c\}$



From safety to general LTL

To get from safety to the general LTL case, we need to ...

... scale up from the safety winning condition to something more complex

Upcoming:

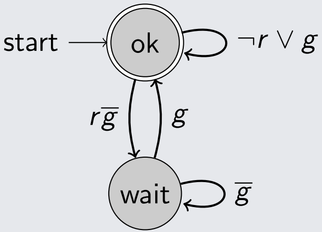
- Büchi games
- Parity games

Example deterministic Büchi automaton and game

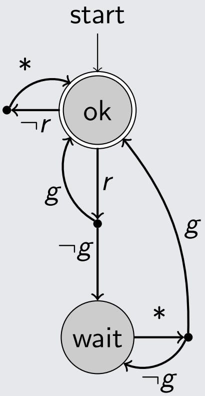
LTL Specification

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

Büchi automaton



Büchi game

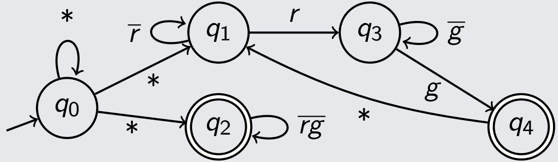


On using non-deterministic Büchi automata

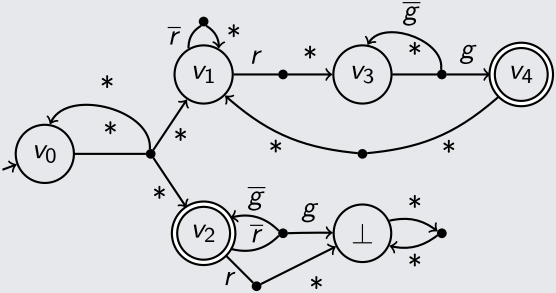
LTL

$(\mathbf{GF}r \wedge \mathbf{GF}g) \vee$
$(\mathbf{FG}\neg r \wedge \mathbf{FG}\neg g)$

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 exist no deterministic Büchi automata

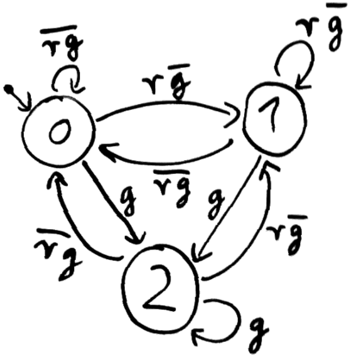
Problem

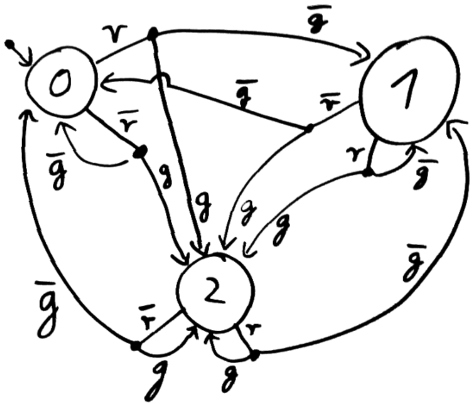
The automaton→game construction only works for *deterministic* automata

Solution

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

$$GFr \rightarrow GFg$$





Building parity automata

Classical approach

- Construct a non-deterministic Büchi automaton from the LTL specification
- Translate the Büchi automaton to a deterministic parity automaton (Piterman, 2006)

$\rightarrow$ doubly-exponential blow-up!

More practical approach

Many LTL subsets allow building the deterministic parity automaton directly (see, e.g., Bloem et al., 2012, where this is done implicitly)

How to solve games, the general case

Safety: $W_{sys} = \nu X.(V_{env} \cap \overline{\bot} \cap \Box X) \cup (V_{sys} \cap \overline{\bot} \cap \Diamond X)$

Büchi:

Parity:

How to solve games, the general case

Safety: $W_{sys} = \nu X.(V_{env} \cap \overline{\bot} \cap \Box X) \cup (V_{sys} \cap \overline{\bot} \cap \Diamond X)$

Büchi: $W_{sys} = \nu X.\mu Y.(V_{env} \cap \overline{F} \cap \Box Y) \cup (V_{sys} \cap \overline{F} \cap \Diamond Y) \cup (V_{env} \cap F \cap \Box X) \cup (V_{sys} \cap F \cap \Diamond X)$

Parity:

How to solve games, the general case

Safety: $W_{sys} = \nu X.(V_{env} \cap \overline{\bot} \cap \Box X) \cup (V_{sys} \cap \overline{\bot} \cap \Diamond X)$

Büchi:
$$W_{sys} = \nu X.\mu Y.(V_{env} \cap \overline{F} \cap \Box Y) \cup (V_{sys} \cap \overline{F} \cap \Diamond Y) \\ \cup (V_{env} \cap F \cap \Box X) \cup (V_{sys} \cap F \cap \Diamond X)$$

Parity:
$$W_{sys} = \gamma X_{c-1}. \ldots . \nu X_2.\mu X_1.\nu X_0. \\ \bigcup_{i \in \{c-1,\ldots,0\}} (V_{env} \cap C_i \cap \Box X_i) \cup (V_{sys} \cap C_i \cap \Diamond X_i)$$

How to solve games, the general case

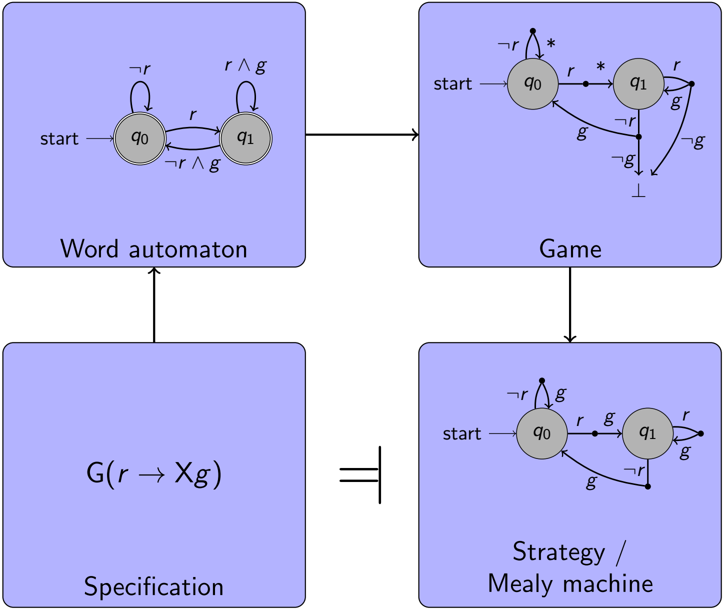
Safety: $W_{sys} = \nu X.(V_{env} \cap \overline{\bot} \cap \Box X) \cup (V_{sys} \cap \overline{\bot} \cap \Diamond X)$

Büchi: $W_{sys} = \nu X.\mu Y.(V_{env} \cap \overline{F} \cap \Box Y) \cup (V_{sys} \cap \overline{F} \cap \Diamond Y)$
$\cup (V_{env} \cap F \cap \Box X) \cup (V_{sys} \cap F \cap \Diamond X)$

Parity: $W_{sys} = \gamma X_{c-1}.\ldots.\nu X_2.\mu X_1.\nu X_0.$
$\bigcup_{i \in \{c-1,\ldots,0\}} (V_{env} \cap C_i \cap \Box X_i) \cup (V_{sys} \cap C_i \cap \Diamond X_i)$

All games have positional winning strategies.
All complexities are polynomial (for some constant c)

Conceptual summary



Efficiently implementing reactive synthesis

Symbolic data structures

- Binary decision diagrams
- Safety games: list of worst-case positions $\rightarrow$ antichains

Targeting specific specification classes - Example 1/2

Specification form:

$$(a_1 \wedge a_2 \wedge \ldots \wedge a_n) \rightarrow (g_1 \wedge g_2 \wedge \ldots \wedge g_m)$$

for which every a_i and g_i is of one of the following forms:

(1) ψ

(2) $\mathsf{G}(\psi_1 \rightarrow \mathsf{X}(\psi_2))$

(3) $\mathsf{GF}(\psi)$

Efficiently implementing reactive synthesis

Symbolic data structures

- Binary decision diagrams
- Safety games: list of worst-case positions $\rightarrow$ antichains

Targeting specific specification classes - Example 2/2

Specification form:

$$(a_1 \wedge a_2 \wedge \ldots \wedge a_n) \rightarrow (g_1 \wedge g_2 \wedge \ldots \wedge g_m)$$

for which most of $a_1, \ldots, g_m$ are *safety* properties.

Reactive Synthesis - summary

Main concepts

- Mealy/Moore machines
- Deterministic word automata
- Games with ω-regular winning condition

Main difficulties

- Complexity! It is 2EXPTIME for LTL specifications
- Complicated constructions (e.g., Büchi$\rightarrow$parity)

Main applications

- Automatic system construction
- Fast prototyping
- Specification debugging

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.

Nir Piterman. From nondeterministic Buchi and Streett automata to deterministic parity automata. In *LICS*, pages 255–264. IEEE Computer Society, 2006.