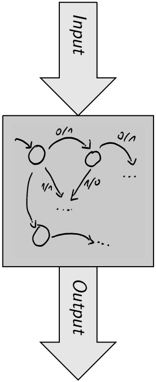
Model Checking and Games

Part VIII - Reactive Synthesis & Games

Rüdiger Ehlers, Clausthal University of Technology

September 2019

Verification:

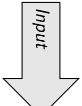


$+$ $\mathbf{G}(r \rightarrow \mathbf{X}g)$ $\Rightarrow$

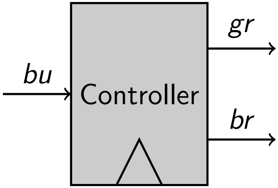
Satisfied

Not satisfied
(+ counter-example)

Synthesis:



Synthesis of reactive systems - example



Atomic propositions

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

A run of the system

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

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)

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

A surprise

The specification is *unrealizable*.

Example:

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

Fixing the specification

Idea

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

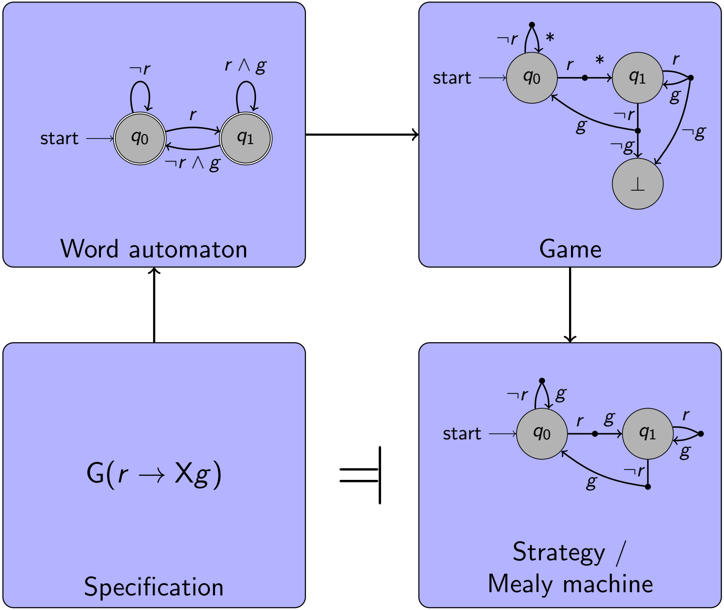
New formal specification in linear-time temporal logic (LTL)

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

Result

This one is realizable $\rightarrow$ Demo!

General synthesis workflow



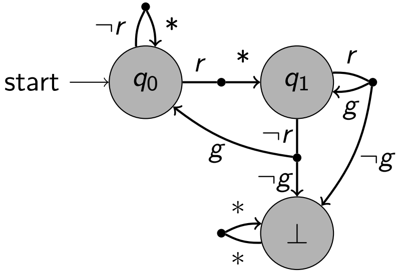
Games

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.

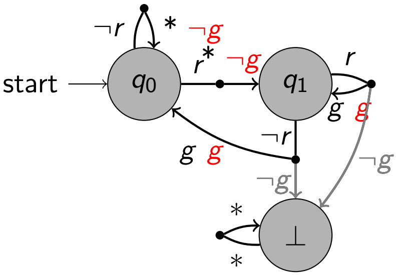


Games for synthesis

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.

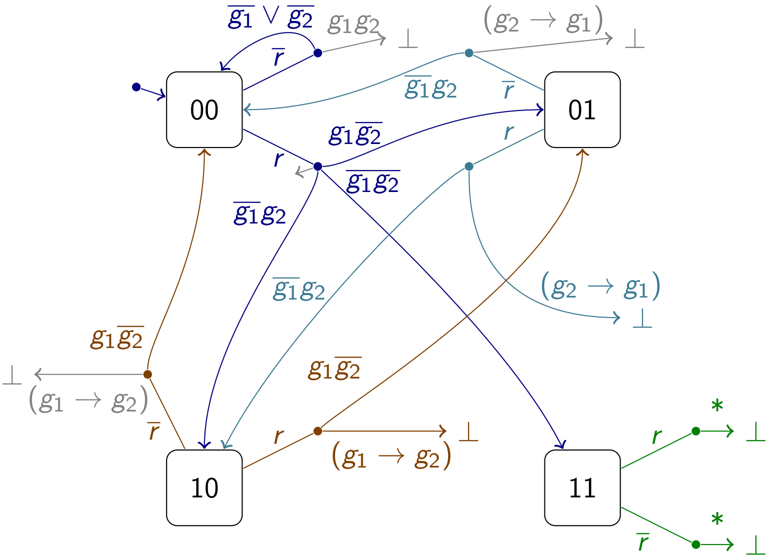


This is a Mealy Machine!

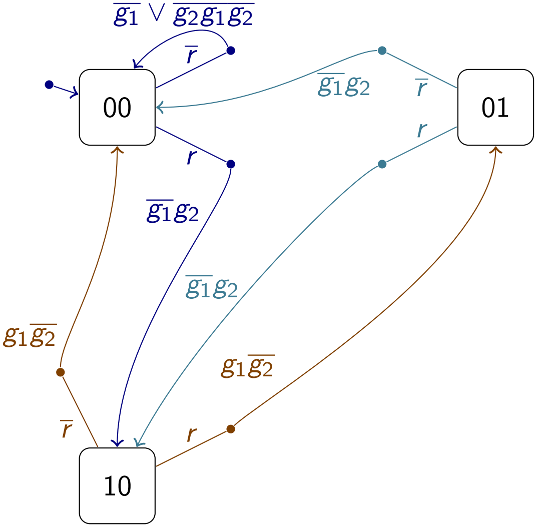
Strategies in synthesis games

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

A more complicated (safety) game

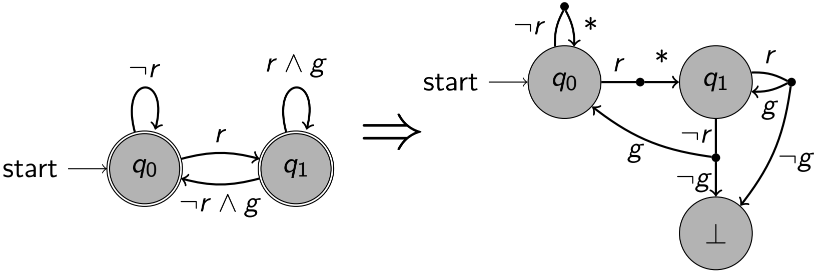


A more complicated (safety) game



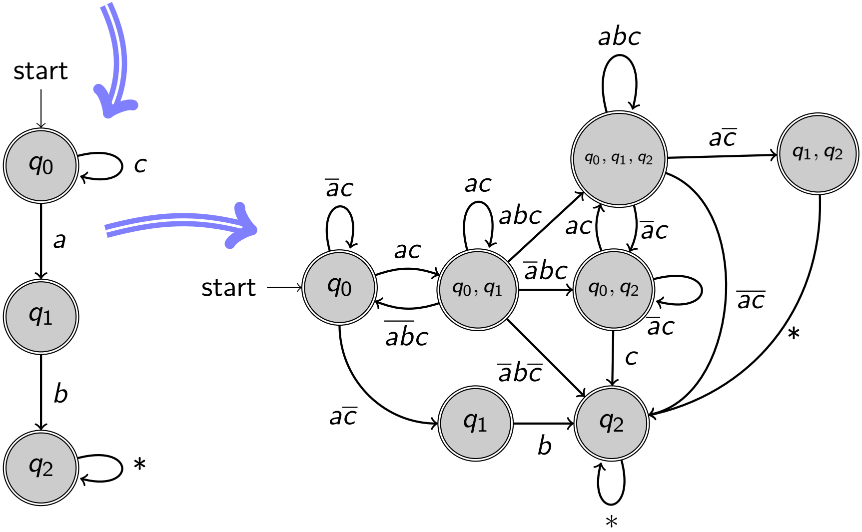
Ok, so how do we build a *synthesis game*?

Building safety games from deterministic safety automata

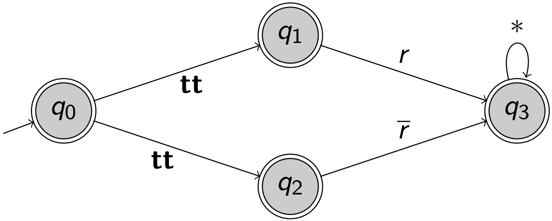


Building deterministic safety automata

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



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

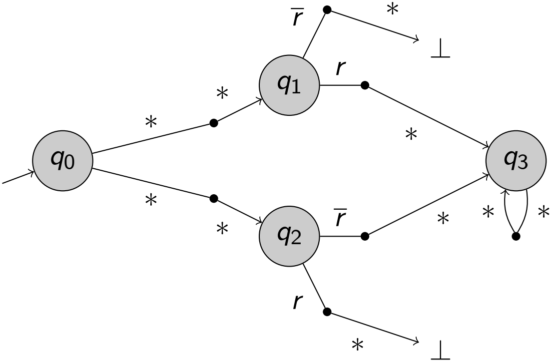


Specification: $\mathbf{X} r \vee \mathbf{X} \neg r$

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

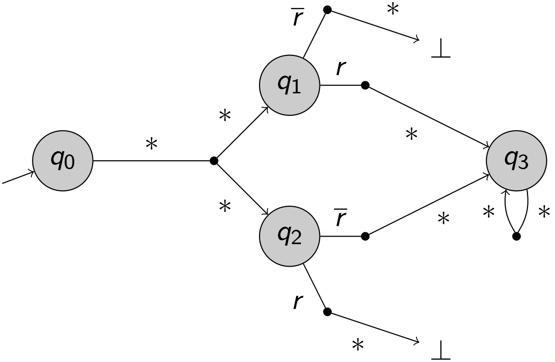
So why do we need determinization (2/3) ?

Case 1: The environment player resolves the nondeterminism



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

Case 2: The system player resolves the nondeterminism

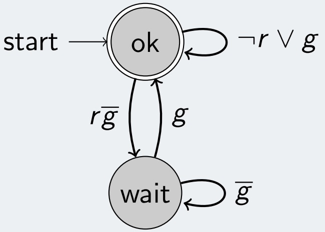


The non-safety, deterministic Büchi case

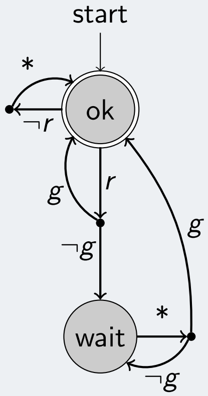
LTL Specification

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

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

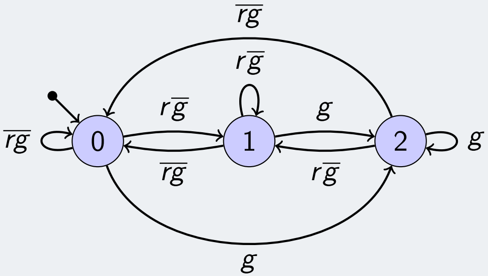
The automaton→game construction only works for *deterministic* automata

Solution

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

Parity automata by example (1)

Automaton

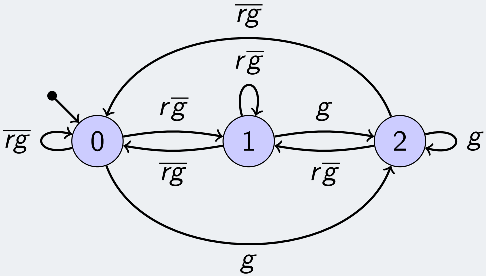


Acceptance condition

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

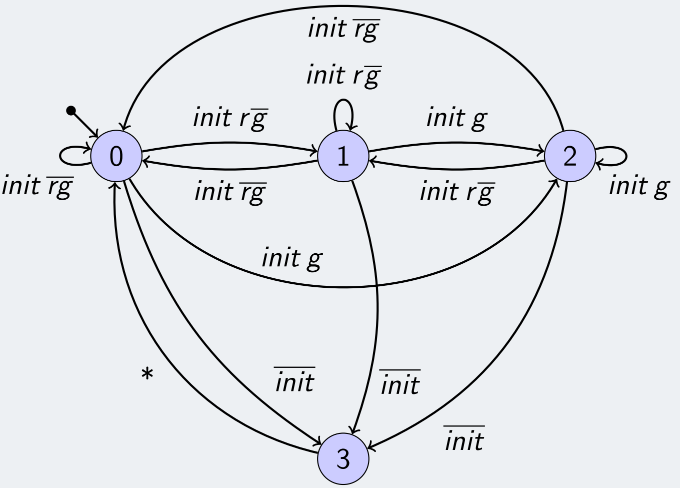


Encoded LTL property

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

Parity automata by example (2)

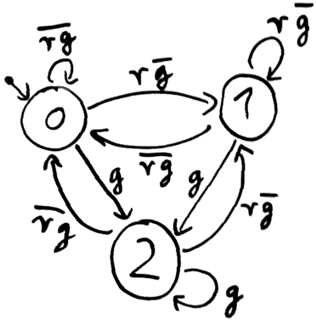
Automaton

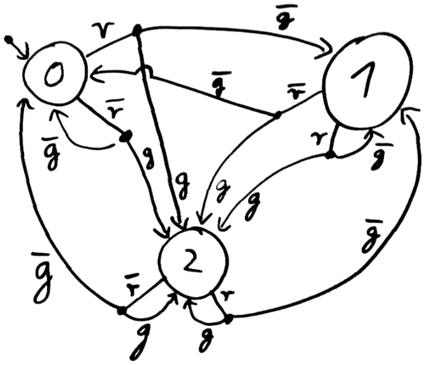


Spec'

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

$GFr \to GFg$





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

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)"
```

→ 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

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

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

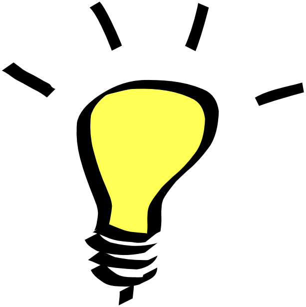
How can we get around the doubly-exponential synthesis complexity **in practice**?

But how can we do this?

Answer

By exploiting some properties of the problem such as:

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



Generalized Reactivity(1) Synthesis

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

What are we willing to trade?

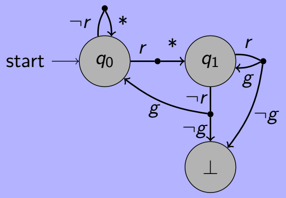
...the full expressivity of LTL!

What do we want?

A reduction in time complexity from doubly-exponential to singly exponential!

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

Direct translation, exponential blow-up

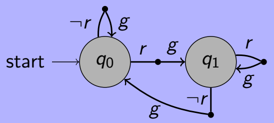


Game

$$G(r \rightarrow Xg)$$

Specification

$\models$



Strategy / Mealy machine

GR(1) – What should be supported?

Computation model

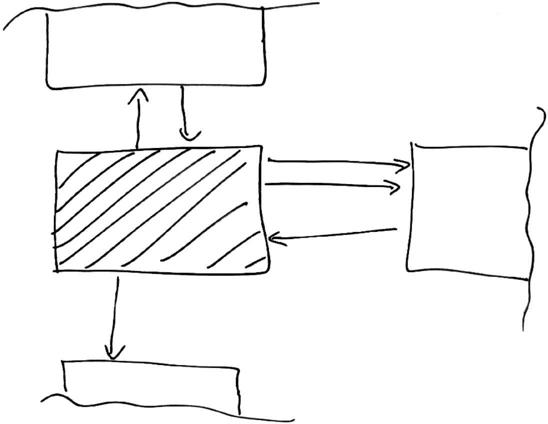
We choose a Mealy-type computation model

Focus

A specification consists of *assumptions* and *guarantees*, each of which are either

- initialization properties,
- basic safety properies, or
- basic liveness properties.

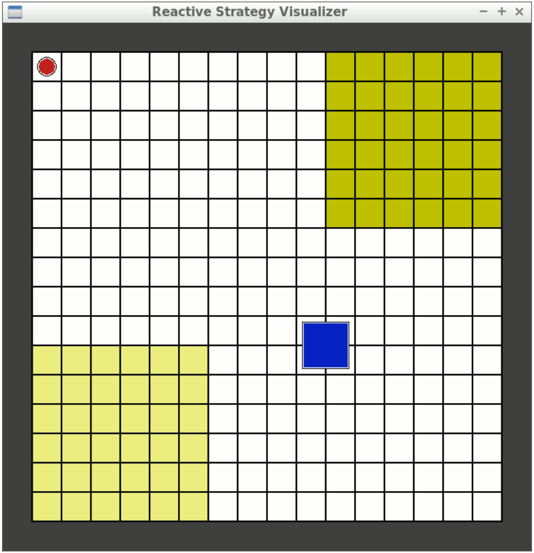
Assumptions and guarantees in specifications



Specification shape

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

Demo – Assumptions & Guarantees



GR(1) – Overall specification shape

Specification shape

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

GR(1) – Overall specification shape

Specification shape

$$\left(\varphi_i^a \wedge \varphi_s^a \wedge \varphi_l^a\right) \rightarrow \left(\varphi_i^g \wedge \varphi_s^g \wedge \varphi_l^g\right)$$

GR(1) – Overall specification shape

Specification shape

$$\left(\underbrace{\varphi_i^a}_{\substack{\text{initialization} \\ \text{assumptions}}} \wedge \underbrace{\varphi_s^a}_{\substack{\text{safety} \\ \text{assumptions}}} \wedge \underbrace{\varphi_l^a}_{\substack{\text{liveness} \\ \text{assumptions}}} \right) \rightarrow (\varphi_i^a \wedge \varphi_s^a \wedge \varphi_l^a)$$

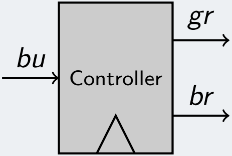
GR(1) – Overall specification shape

Specification shape

$$\left(\varphi_i^g \wedge \varphi_s^g \wedge \varphi_l^g\right) \rightarrow \left(\underbrace{\varphi_i^g}_{\substack{\text{initialization}\\ \text{guarantees}}} \wedge \underbrace{\varphi_s^g}_{\substack{\text{safety}\\ \text{guarantees}}} \wedge \underbrace{\varphi_l^g}_{\substack{\text{liveness}\\ \text{guarantees}}} \right)$$

Specification parts: Initialization assumptions

Controller shape – Coffee machine example



Here, $\mathrm{AP}_I = \{bu\}$, $\mathrm{AP}_O = \{gr, br\}$

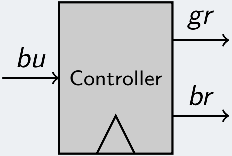
Initialization assumptions

These are properties without a temporal operator, only over AP_I.
Example:

- $\neg bu$

Specification parts: Safety assumptions

Controller shape – Coffee machine example



Here, $\mathsf{AP}_I = \{bu\}$, $\mathsf{AP}_I = \{gr, br\}$

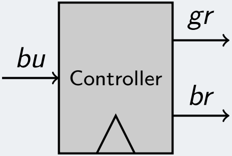
Safety assumptions

These are properties of the form $\mathbf{G}(\psi)$ where ψ is a Boolean formula over $\mathsf{AP}_I \cup \mathsf{AP}_O \cup \{\mathbf{X}\, y \mid y \in \mathsf{AP}_I\}$. Examples:

- $\mathbf{G}(bu \rightarrow \mathbf{X}\neg bu)$
- $\mathbf{G}((gr \vee br) \rightarrow \mathbf{X}\neg bu)$

Specification parts: Liveness assumptions

Controller shape – Coffee machine example



Here, $\mathrm{AP}_I = \{bu\}$, $\mathrm{AP}_I = \{gr, br\}$

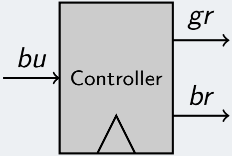
Liveness assumptions

These are properties of the form $\mathbf{GF}(\psi)$ where ψ is a Boolean formula over $\mathrm{AP}_I \cup \mathrm{AP}_O \cup \{\mathbf{X}\, y \mid y \in \mathrm{AP}_I \cup \mathrm{AP}_O\}$. Examples:

- $\mathbf{GF}(bu)$
- $\mathbf{GF}(\neg br \wedge \neg gr \wedge \mathbf{X}\, bu)$

Specification parts: Initialization guarantees

Controller shape – Coffee machine example



Here, $\mathrm{AP}_I = \{bu\}$, $\mathrm{AP}_I = \{gr, br\}$

Initialization guarantees

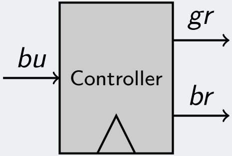
These are properties without a temporal operator, only over $\mathrm{AP}_I \cup \mathrm{AP}_O$.

Example:

- $\neg gr \wedge \neg br$
- $\neg bu \rightarrow (\neg gr \wedge \neg br)$

Specification parts: Safety guarantees

Controller shape – Coffee machine example



Here, $\mathrm{AP}_I = \{bu\}$, $\mathrm{AP}_I = \{gr, br\}$

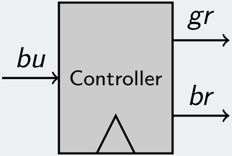
Safety guarantees

These are properties of the form $\mathbf{G}(\psi)$ where ψ is a Boolean formula over $\mathrm{AP}_I \cup \mathrm{AP}_O \cup \{\mathbf{X}\, y \mid y \in \mathrm{AP}_I \cup \mathrm{AP}_O\}$. Examples:

- $\mathbf{G}(gr \rightarrow \mathbf{X}\neg gr)$
- $\mathbf{G}(gr \wedge \mathbf{X} bu \rightarrow \mathbf{X} gr)$

Specification parts: Liveness guarantees

Controller shape – Coffee machine example



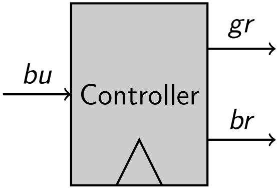
Here, $\mathrm{AP}_I = \{bu\}$, $\mathrm{AP}_I = \{gr, br\}$

Liveness guarantees

These are properties of the form $\mathbf{GF}(\psi)$ where ψ is a Boolean formula over $\mathrm{AP}_I \cup \mathrm{AP}_O \cup \{\mathbf{X}\, y \mid y \in \mathrm{AP}_I \cup \mathrm{AP}_O\}$. Examples:

- $\mathbf{GF}(gr \wedge \mathbf{X}\, br)$
- $\mathbf{GF}(bu \vee br)$

GR(1) (Mealy) execution semantics step-by-step



Atomic propositions

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

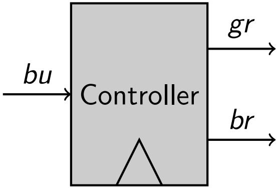
A run of the system

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

Step 1

The environment selects values for AP_I that satisfy the environment initialization assumptions

GR(1) (Mealy) execution semantics step-by-step



Atomic propositions

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

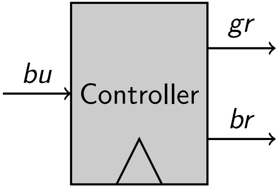
A run of the system

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

Step 2

The system selects values for AP_O such that the first element of ρ satisfies all initialization guarantees

GR(1) (Mealy) execution semantics step-by-step



Atomic propositions

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

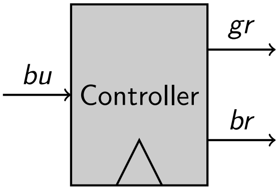
A run of the system

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

Step $2 \cdot i + 1$

The environment selects values for AP_I such that the last element of ρ and the new values for AP_I satisfy the environment safety assumptions

GR(1) (Mealy) execution semantics step-by-step



Atomic propositions

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

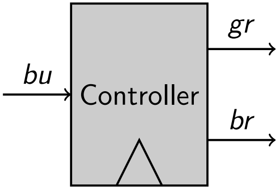
A run of the system

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

Step $2 \cdot i + 2$

The system selects values for AP_O such that the last element of ρ and the new values for AP_I and AP_O satisfy the system safety guarantees

GR(1) (Mealy) execution semantics step-by-step



Atomic propositions

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

A run of the system

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

And so on...

This process continues ad infinitum.

GR(1) Semantics – Who wins the game?

Finitary winning

If at some point, one of the players does not stick to the rules of the game, then the player doing so first **loses** the game.

Otherwise: Infinitary winning

If both players play according to the rules, then the system player wins if and only if the winning condition

$$\varphi_l^a \rightarrow \varphi_l^g$$

is fulfilled.

Let's explore the semantics by example (1)

GR(1) synthesis tool used

slugs – Live web-based version available at
http://webslugs.ruediger-ehlers.de

Specification

```
[INPUT]
bu

[OUTPUT]
br
gr

[ENV_INIT]

[SYS_INIT]
gr <-> bu
! br
```

Let's explore the semantics by example (2)

Added specification parts

```
[SYS_TRANS]
br' <-> gr
gr' -> bu'

[ENV_TRANS]
bu' -> !gr & !br
```

Observation

The system **can** now make coffee, but does not *have* to.

Let's explore the semantics by example (3)

Added specification parts

```
[SYS_LIVENESS]
br
```

Observation

Since the system cannot enforce a button press, it now loses

Let's explore the semantics by example (4)

Added specification parts

[ENV_LIVENESS]
bu

Observation

Now everything works as expected!

Beware the semantics of GR(1) – Part I

Note

There is a discrepancy between the presentation of a GR(1) problem in the form

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

and the step-by-step execution explained above.

Example (1)

$$(\mathbf{GF}r \wedge \mathbf{G}\neg r) \to (\mathbf{G}g \wedge \mathbf{G}\neg g)$$

Beware the semantics of GR(1) – Part II

Note

There is a discrepancy between the presentation of a GR(1) problem in the form

$$\left(\bigwedge \mathsf{Assumptions}\right) \to \left(\bigwedge \mathsf{Guarantees}\right)$$

and the step-by-step execution explained above.

Example (2)

$$(\mathbf{G}r \wedge \mathbf{G}\neg r) \to (\mathbf{GX}g \wedge \mathbf{GX}\neg g)$$

Syntactic Extension to GR(1) – Counters

Using counters

To simplify working with cyber-physical systems, we will syntactically extend the set of GR(1) specifications by *counter variables*, which are actually binary-encoded into the atomic propositions.

Note

In LTL, this does not make sense:

$$\mathbf{G}(\mathit{counter} \leq \mathbf{X}(\mathit{counter}) + 7)$$

But some synthesis tools such as `slugs` and `TuLiP` (Wongpiromsarn et al., 2011) support this anyway.

Syntactic Extension to GR(1) – Counters in Slugs

Version with counters

```
[INPUT]
a:0...15

[OUTPUT]
b

[ENV_INIT]
a >= 3

[ENV_TRANS]
a' <= a + 8

...
```

Version without counters

```
[INPUT]
a@0.0.15
a@1
a@2
a@3

[OUTPUT]
b

[ENV_INIT]
a@0.0.15 & a@1 | a@2 | a@3

[ENV_TRANS]
...
```

Slugs – Example with counters

```
[INPUT]
u

[OUTPUT]
c:0...10

[SYS_INIT]
c = 0

[SYS_TRANS]
u' -> c'=c+1
```

How GR(1) synthesis works

Step 1: Building a synthesis game

Basic idea

The state space of the game is $2^{\mathsf{AP}_I \cup \mathsf{AP}_O}$.

- The initialization assumptions and guarantees are used to define the set of initial states of the game.
- The safety assumptions and guarantees are used to define the transition structure of the game
- The liveness assumptions and guarantees are used to define the winning condition of the game.

Central property of the game

$\rightarrow$ size is exponential in $|\mathsf{AP}_I \cup \mathsf{AP}_O|$.

Example

Specification

$$(\mathbf{GF}x \wedge \mathbf{G}(\neg x \vee \neg \mathbf{X}\, x)) \rightarrow (\mathbf{G}((\neg x \wedge y) \rightarrow \mathbf{X}\, x) \wedge \mathbf{GFX}y \wedge (x \leftrightarrow y))$$

Breaking the specification into pieces

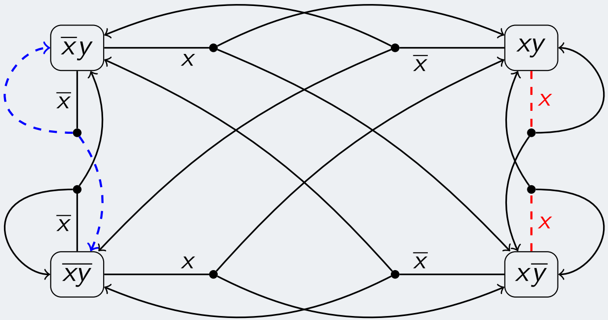
- Initialization assumptions: none
- Safety assumptions: $\mathbf{G}(\neg x \vee \neg \mathbf{X}\, x)$
- Liveness assumptions: $\mathbf{GF}x$
- Initialization guarantees: $(x \leftrightarrow y)$
- Safety guarantees: $\mathbf{G}((\neg x \wedge y) \rightarrow \mathbf{X}\, x)$
- Liveness guarantees: $\mathbf{GFX}y$

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



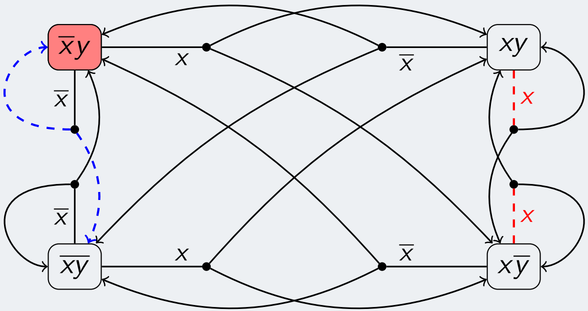
Solving GR(1) games – Step 1: Safety game solving

Process

Compute the largest set of (winning) game states W such that:

- a state is removed from W if the environment has a (legal) successor state such that all successors are non-winning (or the transition is illegal)

Game



Solving GR(1) games – Step 1: Safety game solving

Process

Compute the largest set of (winning) game states W such that:

- a state is removed from W if the environment has a (legal) successor state such that all successors are non-winning (or the transition is illegal)

Towards modelling this as a μ-calculus formula

We search for the largest $W \subseteq 2^{\mathsf{AP}_I \cup \mathsf{AP}_O}$ such that:

$$W = \mathsf{EnfPre}(W),$$

where for every state set $X \subseteq 2^{\mathsf{AP}_I \cup \mathsf{AP}_O}$, we have that $\mathsf{EnfPre}(X)$ contains all $x \in 2^{\mathsf{AP}_I \cup \mathsf{AP}_O}$ such that the system player can enforce that after one move of each player, the play is in a state in X.

Solving GR(1) games – Step 1: Safety game solving

Towards modelling this as a μ-calculus formula

To obtain set W, we can compute (for finite-sized games):

$$W_0 = 2^{\mathsf{AP}_I \cup \mathsf{AP}_O}$$

followed by

$$W_1 = \mathsf{EnfPre}(W_0),$$

$$W_2 = \mathsf{EnfPre}(W_1)$$

and so on, until we reach a *fixpoint*.

Modelling as a μ-calculus formula

$$W = \nu X.\mathsf{EnfPre}(X)$$

Solving GR(1) games – Step 2: Reachability game solving

The next step

Now we need to take the winning condition of the GR(1) game into consideration. For our example GR(1) game, this is:

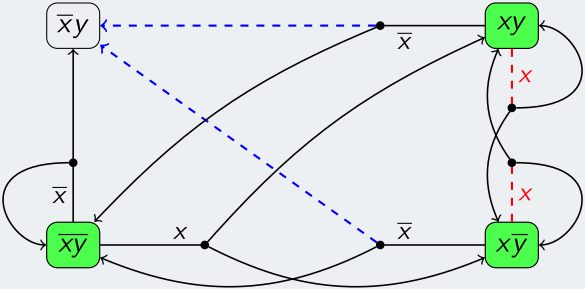
$$(\mathbf{GF}x) \rightarrow (\mathbf{GFX}y)$$

Coming up

Let us have a look at from which states in the game the system player can enforce that eventually a transition is taken along which $\mathbf{X}y$ is satisfied.

Eventually taking a transition satisfying $\mathbf{X}y$

The game (modified!)



Solving GR(1) games – Step 2: Reachability game solving

New μ-calculus equation for eventually taking goal transition ψ

$$\mu X . X \cup \mathsf{EnfPre}(X' \cup \psi)$$

...using the extension of EnfPre to range over transition instead of states, where a dash indicates a state reached after a transition.

Solving GR(1) games – Step 3: Environment goals

Next step

Now the system only needs to reach the next goal under the assumption that the environment fulfils its liveness assumptions:

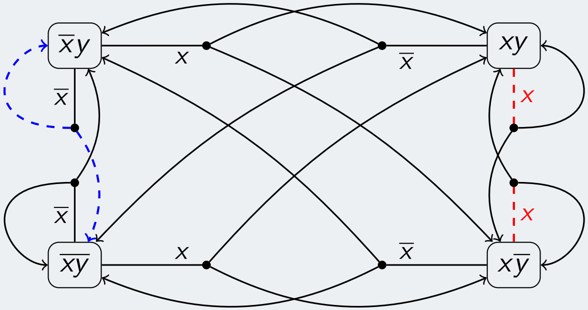
$$(\mathbf{GF}x) \rightarrow (\mathbf{FX}y)$$

Idea

The system now only needs to make progress towards its *goal* whenever the environment reaches one of its *goals*.

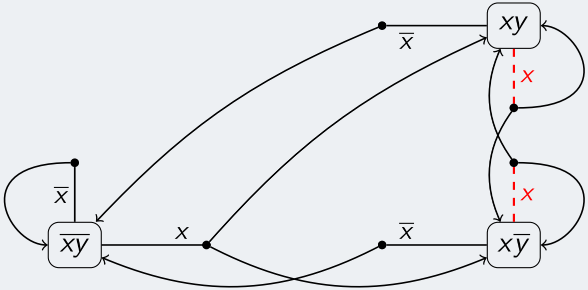
Working on $(\mathbf{GF}x) \rightarrow (\mathbf{FX}y)$

The game (with the losing states)



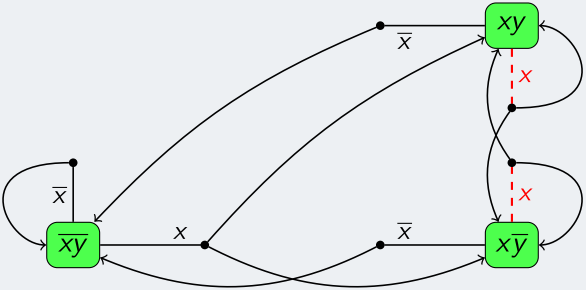
Working on $(\mathbf{GF}x) \to (\mathbf{FX}y)$

The game (without the losing states)



Working on $(\mathbf{GF}x) \rightarrow (\mathbf{FX}y)$

The game (without the losing states)



Solving GR(1) games – Step 3: Environment goals

New μ-calculus formula, first step

Idea: In every step of the system's strategy execution, the strategy either (1) waits for the environment to reach a goal or (2) moves closer towards its own goal:

$$\mu Y.\mathsf{EnfPre}(\psi^g \cup Y') \cup \nu X.\mathsf{EnfPre}((X' \cap \neg\psi^a) \cup Y')$$

Small problem

Which of the two cases above holds may not be under the control of the system. Alternative formula:

$$\mu Y.\nu X.\mathsf{EnfPre}(\psi^g \cup Y' \cup (X' \cap \neg\psi^a))$$

GR(1) Synthesis – Plugging things together

What is still missing

- The system goals need to be reached infinitely often
- Support for multiple environment goals and system goals

Completion of the formula

$$\mu Y.\nu X.\mathsf{EnfPre}(\psi^g \cup Y' \cup (X' \cap \neg\psi^a))$$

$$\Downarrow$$

$$\nu Z.\mu Y.\nu X.\mathsf{EnfPre}(Z' \cap \psi^g \cup Y' \cup (X' \cap \neg\psi^a))$$

$$\Downarrow$$

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

The final GR(1) fixpoint

$$\nu Z. \bigcap_{j \in \{1,\ldots,n\}} \mu Y. \bigcup_{i \in \{1,\ldots,m\}} \nu X. \,\mathrm{EnfPre}\Big((Z' \cap \psi^g_j) \cup Y' \cup (\neg\psi^a_i \cap X') \Big)$$

Reaching the next system goal

Getting closer to the next system goal

Waiting for some environment goal

Give semantics to X, Y, and Z

How do the synthesized strategies look like?

The equation

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

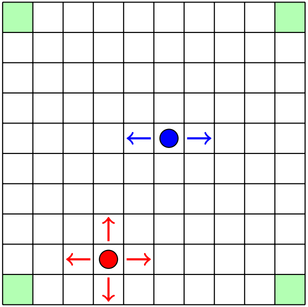
Strategy extraction

For every liveness guarantee no. $i \in \{1, \ldots, n\}$, the transitions computed during the computation of the νY prefix points while Z and X are fully evaluated represent the set of transitions getting closer to system goal i.

So the final strategy...

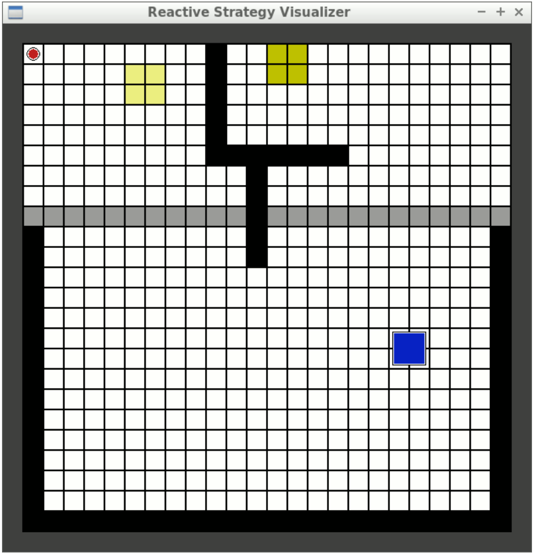
...performs the tasks in a round-robin fashion.

Toggling through the goals – A simple CPS example



```
[INPUT]
o:0...10

[OUTPUT]
x:0...10
y:0...10

[SYS_INIT]
x=0
y=0

[ENV_INIT]
o=0

[SYS_TRANS]
x'=x | y'=y
y'<=y+1
y'+1>=y
x'<=x+1
x'+1>=x

[SYS_LIVENESS]
x'=0 & y'=0
x'=10 & y'=0
x'=10 & y'=10
x'=0 & y'=10

[SYS_TRANS]
y'!=5 | o'!=y' & o'+1!=y' & o'!=y'+1

[ENV_LIVENESS]
o'=0
o'=5
o'=10

[ENV_TRANS]
o'<=o+1
o'+1>=o
```

Another CPS example with a discrete abstraction



Some general notes on the practice of GR(1) synthesis

Notes

- Most GR(1) synthesis tools do not allow $\mathbf{X}$ in the liveness assumptions and guarantees
  $\rightarrow$ No big deal, we can use additional helper variables

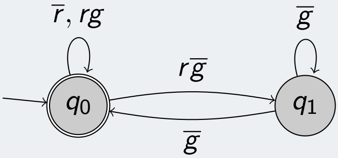
Example

$$\mathbf{GF}(y \wedge \mathbf{X}y)$$
$$\Downarrow$$
$$\mathbf{GF}(y \wedge p) \wedge \mathbf{G}(\mathbf{X}p \leftrightarrow y) \wedge \neg p$$

with the additional output proposition p

Encoding deterministic Büchi automata

Example deterministic Büchi automata for $\mathbf{G}(b \rightarrow \mathbf{F}g)$



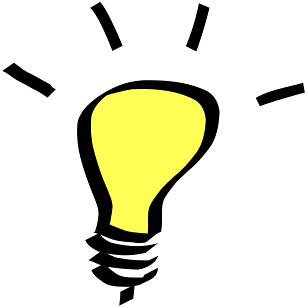
Slugs specification code (interpreting $\mathbf{G}(b \rightarrow \mathbf{F}g)$ as a guarantee)

```
[OUTPUT]
s

[SYS_INIT]
! s

[SYS_TRANS]
s' <-> ((! g & s) | r & ! g)

[SYS_LIVENESS]
! s'
```

GR(1) Synthesis – Conclusion

Summary

- Fast (exponential time) synthesis for simple specification classes
- Approach can be compressed to a single fixed point equation
  $\rightarrow$ allows extensions (e.g., Dathathri et al., 2017; Ehlers, 2011; Wolff et al., 2013, ...)
- Useful for CPS if an environment abstraction is available.

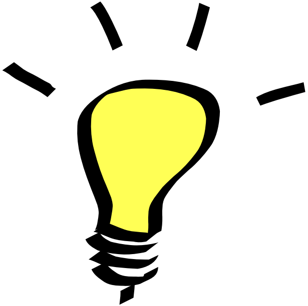


Reactive Synthesis – Summary

Reactive Synthesis – Conclusion

Summary

- A more advanced approach to building correct-by-construction systems
- Main approach: Translate the synthesis problem to a game
- Main difficulty: The sizes of the game
- Generalized Reactivity(1) Synthesis as a way to build smaller games



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.

Sumanth Dathathri, Scott C. Livingston, and Richard M. Murray. Enhancing tolerance to unexpected jumps in GR(1) games. In *Proceedings of the 8th International Conference on Cyber-Physical Systems, ICCPS 2017, Pittsburgh, Pennsylvania, USA, April 18-20, 2017*, pages 37–47, 2017. doi: 10.1145/3055004.3055014. URL http://doi.acm.org/10.1145/3055004.3055014.

Rüdiger Ehlers. Generalized rabin(1) synthesis with applications to robust system synthesis. In *NASA Formal Methods - Third International Symposium, NFM 2011, Pasadena, CA, USA, April 18-20, 2011. Proceedings*, pages 101–115, 2011. doi: 10.1007/978-3-642-20398-5_9. URL https://doi.org/10.1007/978-3-642-20398-5_9.

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.

Eric M. Wolff, Ufuk Topcu, and Richard M. Murray. Efficient reactive controller synthesis for a fragment of linear temporal logic. In *2013 IEEE International Conference on Robotics and Automation, Karlsruhe, Germany, May 6-10, 2013*, pages 5033–5040, 2013. doi: 10.1109/ICRA.2013.6631296. URL https://doi.org/10.1109/ICRA.2013.6631296.

Tichakorn Wongpiromsarn, Ufuk Topcu, Necmiye Ozay, Huan Xu, and Richard M. Murray. Tulip: a software toolbox for receding horizon temporal logic planning. In *Proceedings of the 14th ACM International Conference on Hybrid Systems: Computation and Control, HSCC 2011, Chicago, IL, USA, April 12-14, 2011*, pages 313–314, 2011. doi: 10.1145/1967701.1967747. URL http://doi.acm.org/10.1145/1967701.1967747.

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.