Generalised Rabin(1) synthesis[1]

Rüdiger Ehlers

Saarland University, Reactive Systems Group

2nd Workshop on Games for Design, Verification and Synthesis – September 4, 2010

[1] This work was supported by the German Research Foundation (DFG) within the program "Performance Guarantees for Computer Systems" and the Transregional Collaborative Research Center "Automatic Verification and Analysis of Complex Systems" (SFB/TR 14 AVACS).

Overview of the talk

1. Synthesis of reactive systems - an overview
2. GR(1) - Generalised reactivity(1) synthesis
3. GRabin(1) - Generalised Rabin(1) synthesis
4. There's no room beyond GRabin(1)
5. What can we use GRabin(1) synthesis for?

Synthesis of reactive systems - overview

Problem description

Given ...

- a set of input atomic propositions AP_I,
- a set of output atomic propositions AP_O,
- a temporal logic formula ψ over $\mathsf{AP}_I \uplus \mathsf{AP}_O$

... does there exist a Mealy/Moore automaton reading AP_I and writing AP_O that satisfies ψ?

Properties of this problem

Church's problem is known to be 2EXPTIME-complete [PR89] for LTL specifications.

Incomplete overview of past synthesis approaches

Fast methods for subsets of LTL

- Safety synthesis
- Generalised reactivity(1) synthesis [PPS06]

Methods for full LTL

- Parity game based methods → Generalised parity game based methods [CHP07, SS09]
- Safraless synthesis [KV05] → Bounded synthesis [SF07]
- Rabin game based methods

GR(1) synthesis [PPS06] - Description

Form of the specification (f.t.p. of [KHB09])

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

with:

- a set A of assumptions $a_1, \ldots, a_n$
- a set G of guarantees $g_1, \ldots, g_m$

such that:

- all elements in $A \cup G$ are deterministic Büchi automata
- all elements in $A \cup G$ run over $2^{\mathsf{AP}_I \uplus \mathsf{AP}_O}$

GR(1) synthesis - Algorithm (f.t.p. of [BCG+10])

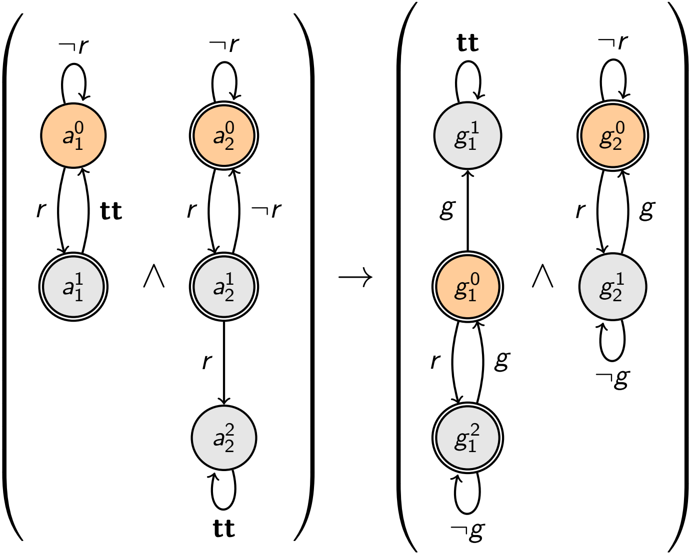
Explanation by example: a mutex

- Set of inputs: $\{r\}$ (a request)
- Set of outputs: $\{g\}$ (a grant)
- Assumptions:
  - $\mathsf{GF}\, r$
  - $\mathsf{G}(r \rightarrow \mathsf{X}\neg r)$
- Guarantees:
  - A grant is only issued after a request has been issued (since the last grant)
  - $\mathsf{GF}\, g$

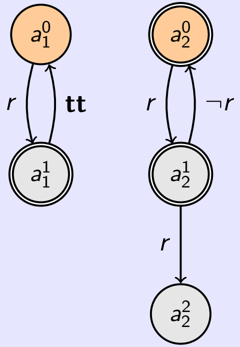
Basic idea of the algorithm

- Reduce the problem to solving a parity game with 3 colours : $0, 1, 2$
- The system player wins if the highest colour occurring infinitely often in the play is 0 or 2

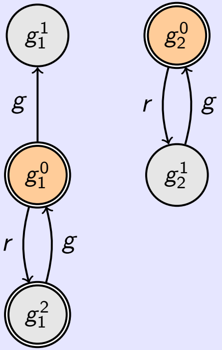
GR(1) synthesis - Algorithm (f.t.p. of [BCG+10])



GR(1) synthesis - Algorithm (f.t.p. of [BCG$^+$10])

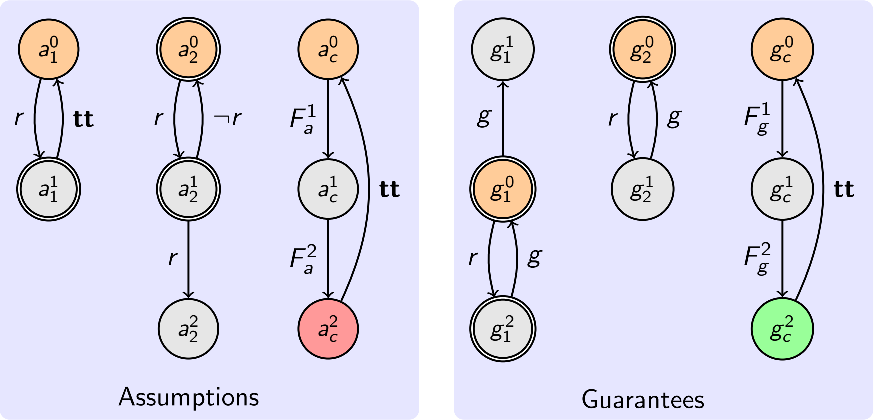


Assumptions



Guarantees

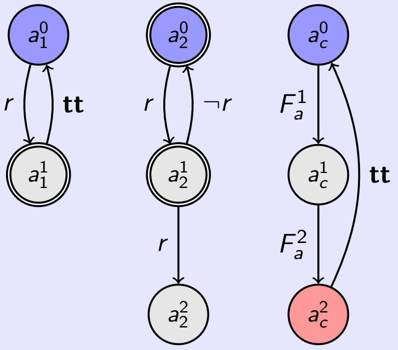
GR(1) synthesis - Algorithm (f.t.p. of [BCG+10])



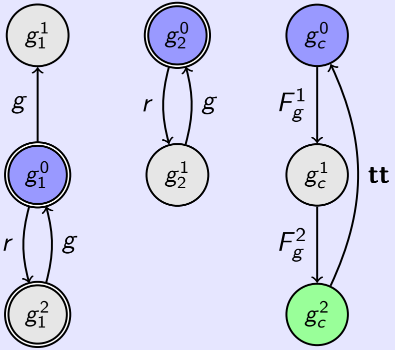
Main idea

Make a parity game over the product state space with 3 colours.

GR(1) synthesis - Algorithm (f.t.p. of [BCG+10])



Assumptions

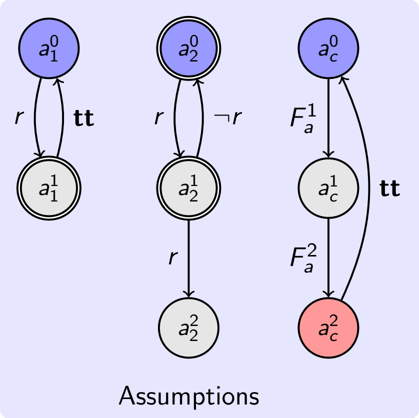


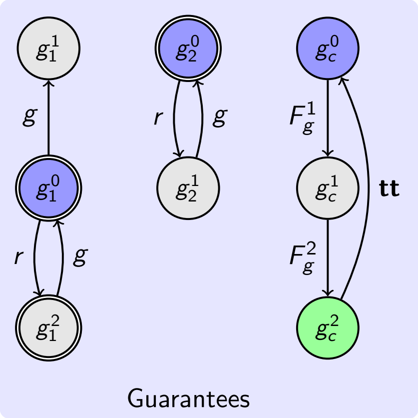
Guarantees

Example run

Request:			
Grant:			
Colour:			

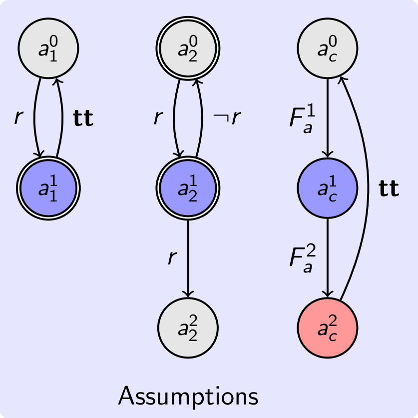
GR(1) synthesis - Algorithm (f.t.p. of [BCG+10])

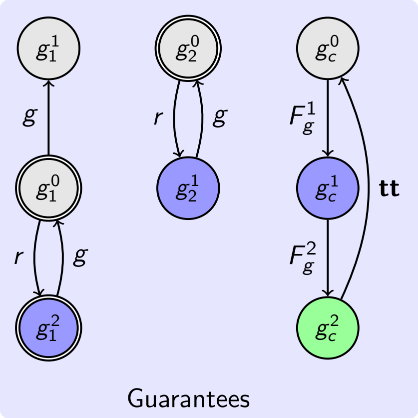




Example run

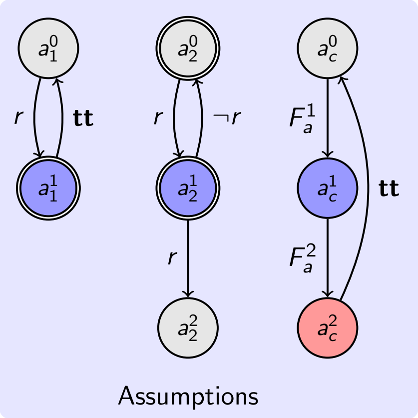
Request:	1		
Grant:			
Colour:			

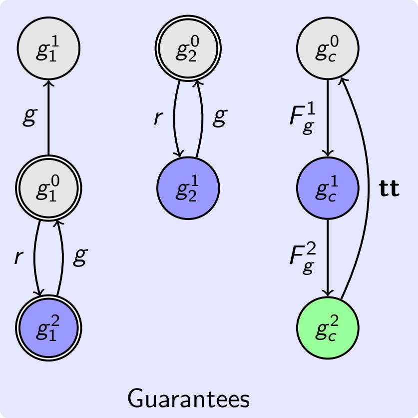




Example run

Request:	1		
Grant:	0		
Colour:	0		

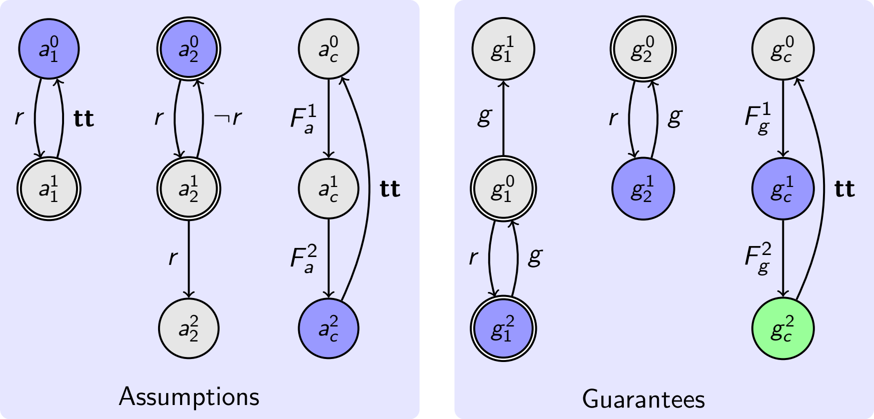




Example run

Request:	1	0
Grant:	0	
Colour:	0	

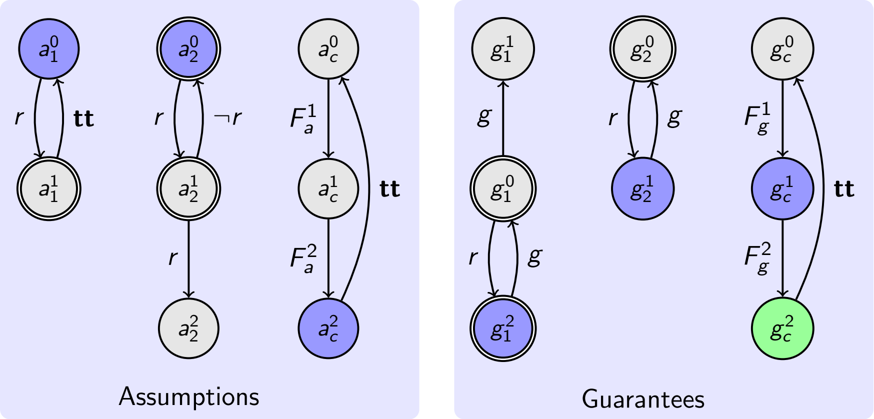
GR(1) synthesis - Algorithm (f.t.p. of [BCG+10])



Example run

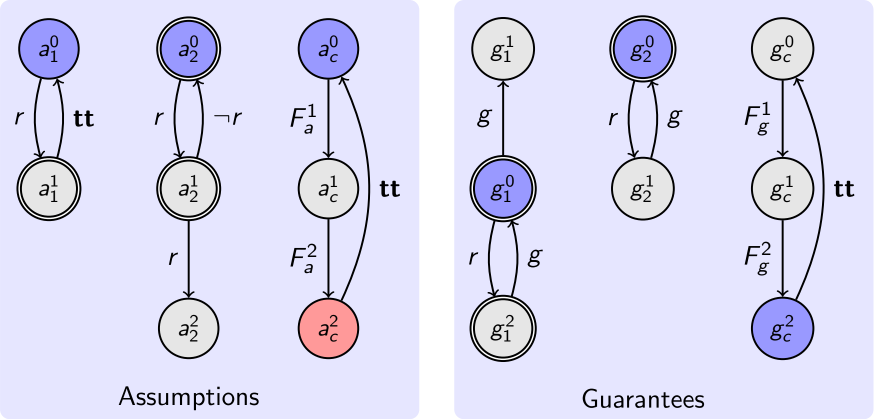
Request:	1	0
Grant:	0	0
Colour:	0	1

GR(1) synthesis - Algorithm (f.t.p. of [BCG+10])



Example run

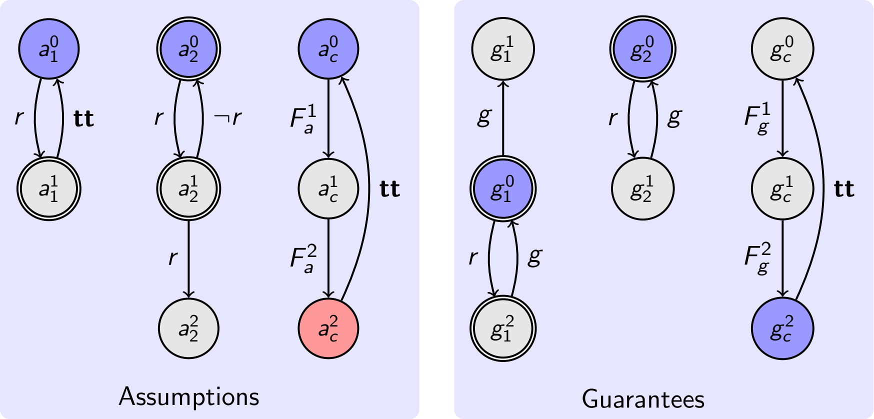
Request:	1	0	0
Grant:	0	0	
Colour:	0	1	



Example run

Request:	1	0	0
Grant:	0	0	1
Colour:	0	1	2

GR(1) synthesis - Algorithm (f.t.p. of [BCG+10])



Example run

Request:	1	0	0
Grant:	0	0	1
Colour:	0	1	2

...

GR(1) synthesis - summary

The highest colour occurring infinitely often

- Colour 0 - Neither the assumptions nor the guarantees are fulfilled
- Colour 1 - The assumptions are fulfilled but not the guarantees
- Colour 2 - The guarantees are fulfilled

The nice properties of this approach

- The game arena for the parity game is the parallel composition of the Büchi automata and some polynomially sized control structure
- The parity game we obtain has a constant number of colours

$\Rightarrow$ Amenable to symbolic implementations, as confirmed by two case studies [BGJ$^+$07a, BGJ$^+$07b]

On extending GR(1)

Main question of this work

How far can we push the expressivity of GR(1) without losing its nice properties?:

- The game arena for the parity game is the parallel composition of the Büchi automata and some polynomially sized control structure
- We have a constant number of colours, independent of the number of assumptions and guarantees

Intuition on why we want this

- Many properties in practice cannot be expressed yet, e.g., FG(*ready*)

Answer to the question raised

A little bit further, but that's it then (assuming P$\neq$NP)

Generalised Rabin(1) synthesis

Form of the specification

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

with:

- a set A of assumptions $a_1, \ldots, a_n$
- a set G of guarantees $g_1, \ldots, g_m$

such that:

- all elements in $A \cup G$ are deterministic **Rabin automata with one acceptance pair**
- all elements in $A \cup G$ run over $2^{\mathsf{AP}_I \uplus \mathsf{AP}_O}$

Rabin automata

Formal definition

A **one-pair** det. Rabin automaton is a tuple $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ with:

- a state set Q
- an alphabet Σ (here, $\Sigma = 2^{\mathsf{AP}_I \uplus \mathsf{AP}_O}$)
- a transition relation $Q \times \Sigma \rightarrow Q$
- an initial state $q_0 \in Q$
- an acceptance component $F = (F_1, F_2) \in 2^Q \times 2^Q$

We say that $\mathcal{A}$ accepts a run $\pi = \pi_0\pi_1\ldots$ if $\pi_0 = q_0$ and $\inf(\pi) \cap F_1 = \emptyset$ and $\inf(\pi) \cap F_2 \neq \emptyset$

A special property of one-pair Rabin automata

A run is accepted by a one-pair Rabin automaton $(Q, \Sigma, \delta, q_0, (F_1, F_2))$ iff it is accepted by the co-Büchi automaton $(Q, \Sigma, \delta, q_0, F_1)$ and the Büchi automaton $(Q, \Sigma, \delta, q_0, F_2)$.

Generalised Rabin(1) synthesis - simplified

Form of the specification

$$\begin{aligned} & (A_1 \wedge A_2 \wedge \ldots \wedge A_{|A|} \wedge B_1 \wedge B_2 \wedge \ldots \wedge B_{|B|}) \\ \rightarrow \quad & (C_1 \wedge C_2 \wedge \ldots \wedge C_{|C|} \wedge D_1 \wedge D_2 \wedge \ldots \wedge D_{|D|}) \end{aligned}$$

with:

- a set A of Büchi assumptions $A_1, \ldots, A_{|A|}$
- a set B of co-Büchi assumptions $B_1, \ldots, B_{|B|}$
- a set C of Büchi guarantees $C_1, \ldots, C_{|C|}$
- a set D of co-Büchi guarantees $D_1, \ldots, D_{|D|}$

such that:

- all elements in $A \cup C$ are deterministic Büchi automata
- all elements in $B \cup D$ are deterministic co-Büchi automata
- all elements in $A \cup B \cup C \cup D$ run over $2^{\mathsf{AP}_I \uplus \mathsf{AP}_O}$

GRabin(1) synthesis - solution idea

(Still flawed) reduction to a parity game

- State space is the product of the automata state spaces and the two Büchi assumption- and guarantee-checker automata A_C and G_C
- 5 colours:
  - Colour 4 - Some co-Büchi assumption is violated
  - Colour 3 - Some co-Büchi guarantee is violated
  - Colour 2 - The Büchi guarantee automata have "recently" all visited their accepting states
  - Colour 1 - The Büchi assumption automata have "recently" all visited their accepting states
  - Colour 0 - none of the above

GRabin(1) synthesis - solution idea analysis

Maximal colour occurring inf. often on a play

	co-B. Ass.	co-B. Ass. B. Ass.	B. Ass.	
co-B. Gua.	0	1	4	4
co-B. Gua. B. Gua.	2	2	4	4
B. Gua.	3	3	4	4
	3	3	4	4

GRabin(1) synthesis - solution idea analysis

Maximal colour occurring inf. often on a play

	co-B. Ass.	co-B. Ass. B. Ass.	B. Ass.	
co-B. Gua.	0	1	4	4
co-B. Gua. B. Gua.	2	2	4	4
B. Gua.	3	3	4	4
	3	3	4	4

A problem

Currently we lose on too many plays!

The solution

Add storage bit to the game tracking whether the counter automaton for A has "recently" completed a cycle. Only use colour 3 if this was the case (and reset the bit in this case).

GRabin(1) synthesis - solution idea analysis

Maximal colour occurring inf. often on a play (fixed)

		co-B. Ass.	co-B. Ass. B. Ass.	B. Ass.	
co-B. Gua.		0	1	4	4
co-B. Gua.	B. Gua.	2	2	4	4
	B. Gua.	2	3	4	4
		0	3	4	4

A problem

Currently we lose on too many plays!

The solution

Add storage bit to the game tracking whether the counter automaton for *A* has "recently" completed a cycle. Only use colour 3 if this was the case (and reset the bit in this case).

Can we generalise even further?

Answer: No!

Generalised Streett(1) synthesis cannot work in the way described:

- Game arena is the product of the individual automata and a polynomially sized control structure (in the number of assumptions and guarantees)
- Parity game with a constant number of colours

Reason

Generalised parity game solving is NP-hard for certain cases

Generalised parity games [CHP07]

The conjunctive version

- A game graph with k parity functions is given
- Player 0 needs to win for all of these functions at the same time

A hard case

For parity functions with the co-domain $\{0, 1, 2\}$, solving (the conjunctive version of) generalised parity games is NP-hard

A reduction of the hard case to generalised Streett(1) games

Convert such a game with k parity functions to the specification $(\mathbf{tt}) \rightarrow (G_1 \wedge \ldots \wedge G_k)$ for one-pair Streett guarantees G_i

The reduction (continued)

Examining the specification

$(\mathbf{tt}) \rightarrow (G_1 \wedge \ldots \wedge G_k)$ has a special property: $G_1, \ldots, G_k$ have the same transition structure.

Consequence

If the product game arena was the product of the individual automata and a polynomially sized control structure (in the number of assumptions and guarantees) and the game had a constant number of colours, we could solve an NP-hard problem in polynomial time.

An application: synthesis of robust systems

Example specification - A processing machine (base version)

$$\left(\mathsf{GF}\left(\begin{array}{c}\text{part}\\\text{incoming}\end{array}\right) \wedge \mathsf{G}\left(\begin{array}{c}\text{no over-}\\\text{sized parts}\end{array}\right) \wedge \ldots\right)$$

$$\rightarrow \left(\mathsf{GF}\left(\begin{array}{c}\text{part}\\\text{processed}\end{array}\right) \wedge \mathsf{G}\left(\begin{array}{c}\text{no}\\\text{jam}\end{array}\right) \wedge \ldots\right)$$

Example specification - A processing machine (robust)

$$\left(\mathsf{GF}\left(\begin{array}{c}\text{part}\\\text{incoming}\end{array}\right) \wedge \mathsf{FG}\left(\begin{array}{c}\text{no over-}\\\text{sized parts}\end{array}\right) \wedge \ldots\right)$$

$$\rightarrow \left(\mathsf{GF}\left(\begin{array}{c}\text{part}\\\text{processed}\end{array}\right) \wedge \mathsf{FG}\left(\begin{array}{c}\text{no}\\\text{jam}\end{array}\right) \wedge \ldots\right)$$

Summary

Generalised Rabin(1) synthesis is ...

- a symbolically implementable synthesis method
- in some sense the best we can get (in this line of research)
- a practically relevant fragment of LTL

Important questions beyond the scope of this talk

- How to get from logic to Rabin(1) automata
- How to encode these automata into BDDs
- How to solve the resulting games efficiently

References I

Roderick Bloem, Krishnendu Chatterjee, Karin Greimel, Thomas A. Henzinger, and Barbara Jobstmann.
Robustness in the presence of liveness.
In Tayssir Touili, Byron Cook, and Paul Jackson, editors, *CAV*, volume 6174 of *Lecture Notes in Computer Science*, pages 410–424. Springer, 2010.

Roderick Bloem, Stefan Galler, Barbara Jobstmann, Nir Piterman, Amir Pnueli, and Martin Weiglhofer.
Interactive presentation: Automatic hardware synthesis from specifications: a case study.
In Rudy Lauwereins and Jan Madsen, editors, *DATE*, pages 1188–1193. ACM, 2007.

Roderick Bloem, Stefan Galler, Barbara Jobstmann, Nir Piterman, Amir Pnueli, and Martin Weiglhofer.
Specify, compile, run: Hardware from PSL.
Electr. Notes Theor. Comput. Sci., 190(4):3–16, 2007.

Krishnendu Chatterjee, Thomas A. Henzinger, and Nir Piterman.
Generalized parity games.
In Helmut Seidl, editor, *FoSSaCS*, volume 4423 of *Lecture Notes in Computer Science*, pages 153–167. Springer, 2007.

Robert Könighofer, Georg Hofferek, and Roderick Bloem.
Debugging formal specifications using simple counterstrategies.
In *FMCAD*, pages 152–159. IEEE, 2009.

Orna Kupferman and Moshe Y. Vardi.
Safraless decision procedures.
In *FOCS*, pages 531–542. IEEE, 2005.

Nir Piterman, Amir Pnueli, and Yaniv Sa'ar.
Synthesis of reactive(1) designs.
In E. Allen Emerson and Kedar S. Namjoshi, editors, *VMCAI*, volume 3855 of *LNCS*, pages 364–380. Springer, 2006.

References II

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 *LNCS*, pages 652–671. Springer, 1989.

Sven Schewe and Bernd Finkbeiner.
Bounded synthesis.
In Kedar S. Namjoshi, Tomohiro Yoneda, Teruo Higashino, and Yoshio Okamura, editors, *ATVA*, volume 4762 of *LNCS*, pages 474–488. Springer, 2007.

Saqib Sohail and Fabio Somenzi.
Safety first: A two-stage algorithm for LTL games.
In *FMCAD*, pages 77–84. IEEE Computer Society, 2009.