



# Symbolic Bounded Synthesis

Rüdiger Ehlers

Saarland University, Reactive Systems Group

CAV 2010 – July 18, 2010

# 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  $\psi$  over  $AP_I \uplus AP_O$

... does there exist a Mealy/Moore automaton reading  $AP_I$  and outputting  $AP_O$  that satisfies  $\psi$ ?

## Properties of this problem

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

## Approaches

- Several approaches exist (e.g., generalized reactivity(1) synthesis [PPS06], “classical” parity game solving, etc.)
- Here, we are concerned with **bounded synthesis** [SF07], a Safrules approach for LTL synthesis [KV05].

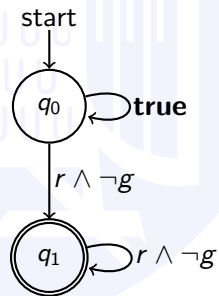
## Criteria for the evaluation of these approaches

- Expressivity
- Scalability
  - suitability for *typical* specifications
  - amenable to symbolic implementations

## Basic Approach

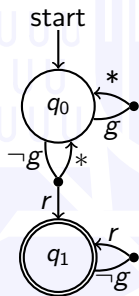
- 1 Convert  $\neg\psi$  to a non-deterministic Büchi word automaton  $\mathcal{A}$
- 2 Dualize  $\mathcal{A}$  to a universal co-Büchi word automaton (UCW)  $\mathcal{A}'$
- 3 Check the universal co-Büchi tree automaton (UCT) obtained from  $\mathcal{A}'$  for emptiness

Basic idea: **Universality makes the world simpler**



NBW for the negated specification /  
UCW for the specification



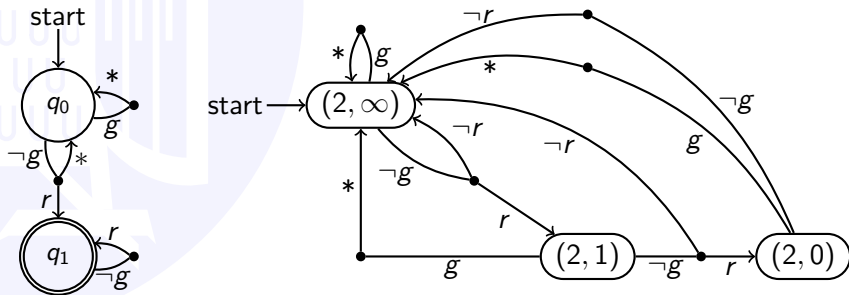


Corresponding UCT



## Central idea

- For every finite-state system satisfying  $\psi$ , there exists an upper bound on the number of visits to rejecting UCT states
- **Bound** that number!
- Then, synthesis can be done by solving a safety game.



# On the efficiency of the technique

## Properties of the game structure

- Number of states: roughly  $(b + 1)^{|Q|}$  – huge!
- Structure is amenable to symbolic implementations

## A symbolic approach from last year's CAV [FJR09]

Antichains can efficiently represent **frontier sets** during the game solving process.

Basic idea: sets of winning states are closed under counter increases, e.g., if state  $(2, 0)$  is winning for the system player, then so is state  $(2, 1)$ .

## Binary decision diagrams

Interestingly, they seem to be unconsidered so far. **In this work, we show how to solve the challenges of applying them in practice.**



## Points **for** BDDs

- Good for *tracking components that run in parallel*:
  - games/automata for the specification conjuncts
  - evolution of the counters

## Points **against** BDDs

- Counters in BDDs are evil! [Weg00, SL99, BMPY97, TV07]

The question raised and answered in this paper:

**How can we reduce the number of counters such that the BDD-approach to Safraless/bounded synthesis is feasible in practice?**

# Structure of the remainder of the talk

## The steps for reducing the number of counters

- Splitting the specification into safety/non-safety properties and composing them to a synthesis game
- Getting rid of some counters in the resulting synthesis game

## Experiments & Outlook

- Comparison of our prototype against Lily/Acacia

# Splitting a specification into safety and non-safety prop's

## The shape of a “typical specification”

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

## Decomposing the specification

- Assumptions  $a_1, \dots, a_n$
- Guarantees  $g_1, \dots, g_m$
- Both assumptions and guarantees typically contain safety formulas.

## Intuition for splitting the specification

Safety properties do not need counters.

# Splitting a simple conjunction

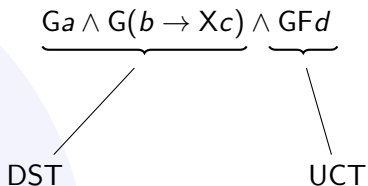
$$Ga \wedge G(b \rightarrow Xc) \wedge GFd$$



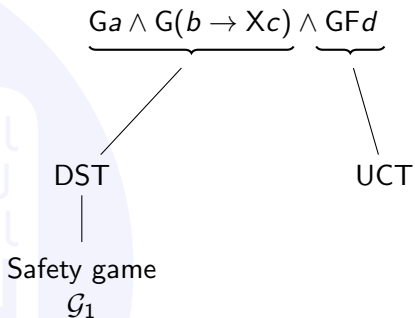
# Splitting a simple conjunction

$$\underbrace{Ga \wedge G(b \rightarrow Xc)}_{\text{safety}} \wedge \underbrace{GFd}_{\text{non-safety}}$$

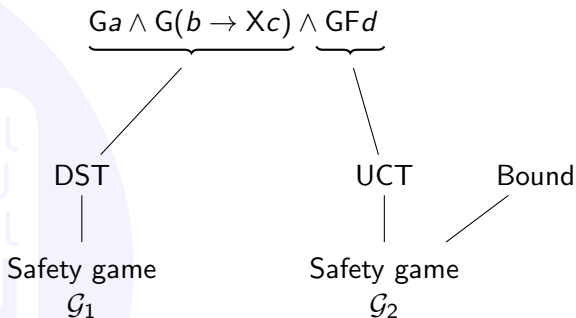
# Splitting a simple conjunction



# Splitting a simple conjunction

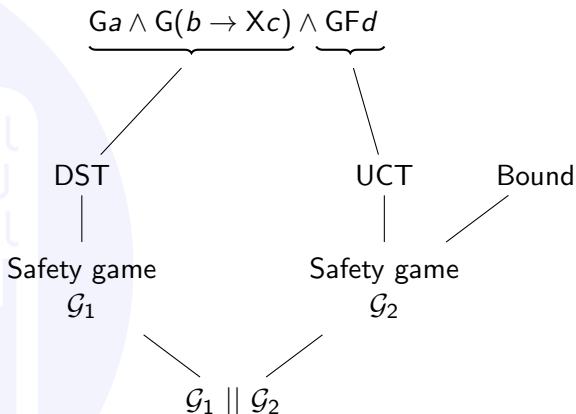


# Splitting a simple conjunction





# Splitting a simple conjunction



## Winning condition

The system player wins  $\mathcal{G}_1 \parallel \mathcal{G}_2$  iff she wins  $\mathcal{G}_1$  and  $\mathcal{G}_2$  at the same time.

# Splitting an assumptions $\rightarrow$ guarantees specification (1/2)

$$AP_I = \{a, b\}, AP_O = \{c, d\}$$

$$(Ga \wedge GFb) \rightarrow (Gc \wedge GFd)$$

# Splitting an assumptions $\rightarrow$ guarantees specification (1/2)

$$AP_I = \{a, b\}, AP_O = \{c, d\}$$

$$\underbrace{(Ga \wedge GFb)}_{\text{DST}} \rightarrow (Gc \wedge GFd)$$

DST

# Splitting an assumptions $\rightarrow$ guarantees specification (1/2)

$$AP_I = \{a, b\}, AP_O = \{c, d\}$$

$$\underbrace{(Ga \wedge GFb)} \rightarrow (Gc \wedge GFd)$$

DST

Safety game

$\mathcal{G}_1$

# Splitting an assumptions $\rightarrow$ guarantees specification (1/2)

$$AP_I = \{a, b\}, AP_O = \{c, d\}$$

$$(\underbrace{Ga \wedge GFb}) \rightarrow (\underbrace{Gc \wedge GFd})$$

DST

DST

Safety game

$\mathcal{G}_1$

# Splitting an assumptions $\rightarrow$ guarantees specification (1/2)

$$AP_I = \{a, b\}, AP_O = \{c, d, \text{safe}\}$$

$$\underbrace{(Ga \wedge GFb)} \rightarrow \underbrace{(Gc \wedge GFd)}$$

DST

Safety game  
 $\mathcal{G}_1$

DST

Safety game  $\mathcal{G}_2$ ,  
won if *safe* always  
represents whether the  
I/O so far is still  
accepted by the DST

# Splitting an assumptions $\rightarrow$ guarantees specification (1/2)

$$AP_I = \{a, b\}, AP_O = \{c, d, \text{safe}\}$$

$$\underbrace{(Ga \wedge GFb)} \rightarrow \underbrace{(Gc \wedge GFd)}$$

$$(GFb) \rightarrow (G\text{safe} \wedge GFd)$$

DST

Safety game  
 $\mathcal{G}_1$

DST

Safety game  $\mathcal{G}_2$ ,  
won if *safe* always  
represents whether the  
I/O so far is still  
accepted by the DST

# Splitting an assumptions $\rightarrow$ guarantees specification (1/2)

$$AP_I = \{a, b\}, AP_O = \{c, d, \text{safe}\}$$

$$\underbrace{(Ga \wedge GFb)} \rightarrow \underbrace{(Gc \wedge GFd)}$$

$$(GFb) \rightarrow (G\text{safe} \wedge GFd)$$

DST

Safety game  $\mathcal{G}_1$

DST

Safety game  $\mathcal{G}_2$ ,  
won if *safe* always  
represents whether the  
I/O so far is still  
accepted by the DST

UCT

Bound

Safety game  $\mathcal{G}_3$



# Splitting an assumptions $\rightarrow$ guarantees specification (1/2)

$$AP_I = \{a, b\}, AP_O = \{c, d, \text{safe}\}$$

$$\underbrace{(Ga \wedge GFb)} \rightarrow \underbrace{(Gc \wedge GFd)}$$

$$(GFb) \rightarrow (G\text{safe} \wedge GFd)$$

DST

DST

UCT

Bound

Safety game  
 $\mathcal{G}_1$

Safety game  $\mathcal{G}_2$ ,  
won if *safe* always  
represents whether the  
I/O so far is still  
accepted by the DST

Safety game  
 $\mathcal{G}_3$

$$\mathcal{G}_1 \parallel \mathcal{G}_2 \parallel \mathcal{G}_3$$

## Splitting an assumptions $\rightarrow$ guarantees specification (2/2)

### Winning condition

The system player wins  $\mathcal{G}_1 \parallel \mathcal{G}_2 \parallel \mathcal{G}_3$  iff she loses  $\mathcal{G}_1$  or she wins  $\mathcal{G}_2$  and  $\mathcal{G}_3$  at the same time.

### The role of *safe*

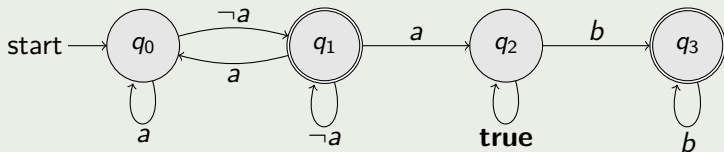
The AP *safe* connects the non-safety and safety guarantee parts. This is important for soundness. Example:

$$(Ga \wedge GF\neg a) \rightarrow (Gc \wedge G\neg c)$$

# Getting rid of additional counters in $\mathcal{G}_3$

## Example specification/Example UCT

$FGa \wedge G((\neg a \wedge Xa) \rightarrow XXGF\neg b)$



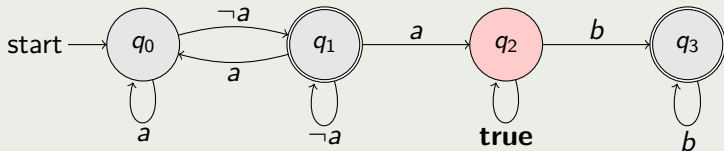
## States of type $(*, \infty, *, \infty)$ in the safety game for $b = 3$

$(3, \infty, \infty, \infty)$	$(2, \infty, \infty, \infty)$	$(1, \infty, \infty, \infty)$	$(0, \infty, \infty, \infty)$
	$(2, \infty, 2, \infty)$	$(1, \infty, 2, \infty)$	$(0, \infty, 2, \infty)$
	$(2, \infty, 1, \infty)$	$(1, \infty, 1, \infty)$	$(0, \infty, 1, \infty)$
	$(2, \infty, 0, \infty)$	$(1, \infty, 0, \infty)$	$(0, \infty, 0, \infty)$

# Getting rid of additional counters in $\mathcal{G}_3$

## Example specification/Example UCT

$$FGa \wedge G((\neg a \wedge Xa) \rightarrow XXGF\neg b)$$



## States of type $(*, \infty, *, \infty)$ in the safety game for $b = 3$

$(3, \infty, \infty, \infty)$	$(2, \infty, \infty, \infty)$	$(1, \infty, \infty, \infty)$	$(0, \infty, \infty, \infty)$
$(2, \infty, 3, \infty)$	$(1, \infty, 3, \infty)$	$(0, \infty, 3, \infty)$	

## A prototype implementation of the BDD-based approach

- Tools/Libraries used:
  - The cudd BDD library
  - The lt12ba LTL→Büchi converter
  - For verifying the results: NuSMV
- Written in C++
- Available at <http://react.cs.uni-saarland.de/tools/unbeast>

## General workflow

- Read the specification and solve the synthesis problem for increasing bounds until the game is winning.
- Also run the tool with negated specification and swapped input/output at the same time (to detect unrealisability)

## The 23 examples from Lily

Speed comparison on an AMD Opteron 2.6 Ghz computer (2 GB of memory available, 1h time limit):

- Lily: 54.35 seconds
- Acacia: 53.71 seconds
- Unbeast: 19.41 seconds

## Performance comparison (2/3)

The scalable example from the Acacia paper

<b># Clients:</b>	1	2	3	4	5	6	7
<b>Unbeast:</b>	0.3 s	0.7 s	0.6 s	1.9 s	0.9 s	4.6 s	3.0 s
<b>Acacia:</b>	0.9 s	2.0 s	4.0 s	9.8 s	47.3 s	506.5 s	m/o

<b># Clients:</b>	10	14	15	20	21	22
<b>Unbeast:</b>	651.5 s	491.0 s	t/o	1909.0 s	t/o	t/o
<b>Acacia:</b>	m/o	m/o	m/o	m/o	m/o	m/o

# Performance comparison (3/3) - The load balancer

Tool	Setting / # Clients	2	3	4	5	6	7	8	9
Unbeast	1	0.6	0.6	0.2	1.3	0.2	0.3	0.2	0.3
Acacia		0.3	0.4	0.6	0.9	1.5	2.7	5.3	12.1
Unbeast	1 $\wedge$ 2	0.4	0.3	0.6	0.6	0.7	0.6	0.6	0.7
Acacia		0.3	0.3	0.4	0.4	0.6	0.9	1.6	3.1
Unbeast	1 $\wedge$ 2 $\wedge$ 3	0.5	0.5	0.5	0.5	0.7	1.0	6.9	73.9
Acacia		19.2	475.6	t/o	t/o	t/o	m/o	m/o	t/o
Unbeast	1 $\wedge$ 2 $\wedge$ 4	0.3	0.4	0.9	65.5	104.6	990.3	t/o	t/o
Acacia		0.6	1.3	8.7	277.9	m/o	m/o	m/o	t/o
Unbeast	1 $\wedge$ 2 $\wedge$ 4 $\wedge$ 5	0.2	0.7	t/o	t/o	t/o	t/o	t/o	t/o
Acacia		163.4	t/o	t/o	t/o	m/o	m/o	m/o	t/o
Unbeast	6 $\rightarrow$ 1 $\wedge$ 2 $\wedge$ 4 $\wedge$ 5	0.2	0.7	3244.1	t/o	t/o	t/o	t/o	t/o
Acacia		175.3	t/o	t/o	t/o	m/o	m/o	t/o	t/o
Unbeast	6 $\wedge$ 7 $\rightarrow$ 1 $\wedge$ 2 $\wedge$ 4 $\wedge$ 5	0.5	1.1	t/o	t/o	t/o	t/o	t/o	t/o
Acacia		190.7	m/o	t/o	t/o	t/o	t/o	t/o	t/o
Unbeast	6 $\wedge$ 7 $\rightarrow$ 1 $\wedge$ 2 $\wedge$ 5 $\wedge$ 8	0.3	0.6	2.4	20.7	368.6	t/o	t/o	t/o
Acacia		7.5	69.0	357.4	m/o	t/o	t/o	t/o	t/o
Unbeast	6 $\wedge$ 7 $\rightarrow$ 1 $\wedge$ 2 $\wedge$ 5 $\wedge$ 8 $\wedge$ 9	0.3	0.2	0.3	1.0	16.8	449.1	t/o	t/o
Acacia		48.8	2133.5	t/o	m/o	t/o	t/o	t/o	t/o
Unbeast	6 $\wedge$ 7 $\wedge$ 10 $\rightarrow$ 1 $\wedge$ 2 $\wedge$ 5 $\wedge$ 8 $\wedge$ 9	0.4	0.8	118.7	t/o	t/o	t/o	t/o	t/o
Acacia		26.9	295.8	m/o	t/o	t/o	t/o	t/o	t/o



## The contributions of this paper

- Showing that BDDs have potential for synthesis from full LTL
- Providing optimisation techniques for this case
- Describing a new scalable benchmark for synthesis from LTL specifications

## Details of the paper left out

- Efficient encoding of safety specifications into games
- Extracting winning strategies from the game in a symbolic way
- Dealing with unrealisability checking
- Counter encoding in BDDs
- Swapping input and output for shorter specifications
- Putting labels onto the edges of the (co-)Büchi automata

# References I



Marius Bozga, Oded Maler, Amir Pnueli, and Sergio Yovine.  
Some progress in the symbolic verification of timed automata.  
In Orna Grumberg, editor, *CAV*, volume 1254 of *LNCS*, pages 179–190. Springer, 1997.



Emmanuel Filiot, Naiyong Jin, and Jean-François Raskin.  
An antichain algorithm for LTL realizability.  
In *CAV*, volume 5643 of *LNCS*, pages 263–277. Springer, 2009.



Orna Kupferman and Moshe Y. Vardi.  
Safriless 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.



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.



K. Schneider and G. Logothetis.  
Abstraction of systems with counters for symbolic model checking.  
In M. Mutz and N. Lange, editors, *Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen*, pages 31–40, Braunschweig, Germany, 1999. Shaker.



D. Tabakov and M. Vardi.  
Model checking Büchi specifications.  
In *LATA*, 2007.



Ingo Wegener.

*Branching Programs and Binary Decision Diagrams.*  
SIAM, 2000.