Topic: Symbolic Synthesis

Lecture at the
1^{st} Summer School on Formal Methods for Cyber-Physical System

Rüdiger Ehlers, University of Bremen

September 2017

So what is left to be discussed?

Big problem

The games that we build for synthesis in bounded synthesis and GR(1) synthesis are still **huge**!

For example, if we have 10 input bits and 10 output bits in GR(1) synthesis, then we have up to 2^{40} transitions in the game $\rightarrow$ does not even fit into this computer's memory (it has 2^{33} bytes of RAM).

What can we do about this?

So what is left to be discussed?

Big problem

The games that we build for synthesis in bounded synthesis and GR(1) synthesis are still **huge**!

For example, if we have 10 input bits and 10 output bits in GR(1) synthesis, then we have up to 2^{40} transitions in the game $\rightarrow$ does not even fit into this computer's memory (it has 2^{33} bytes of RAM).

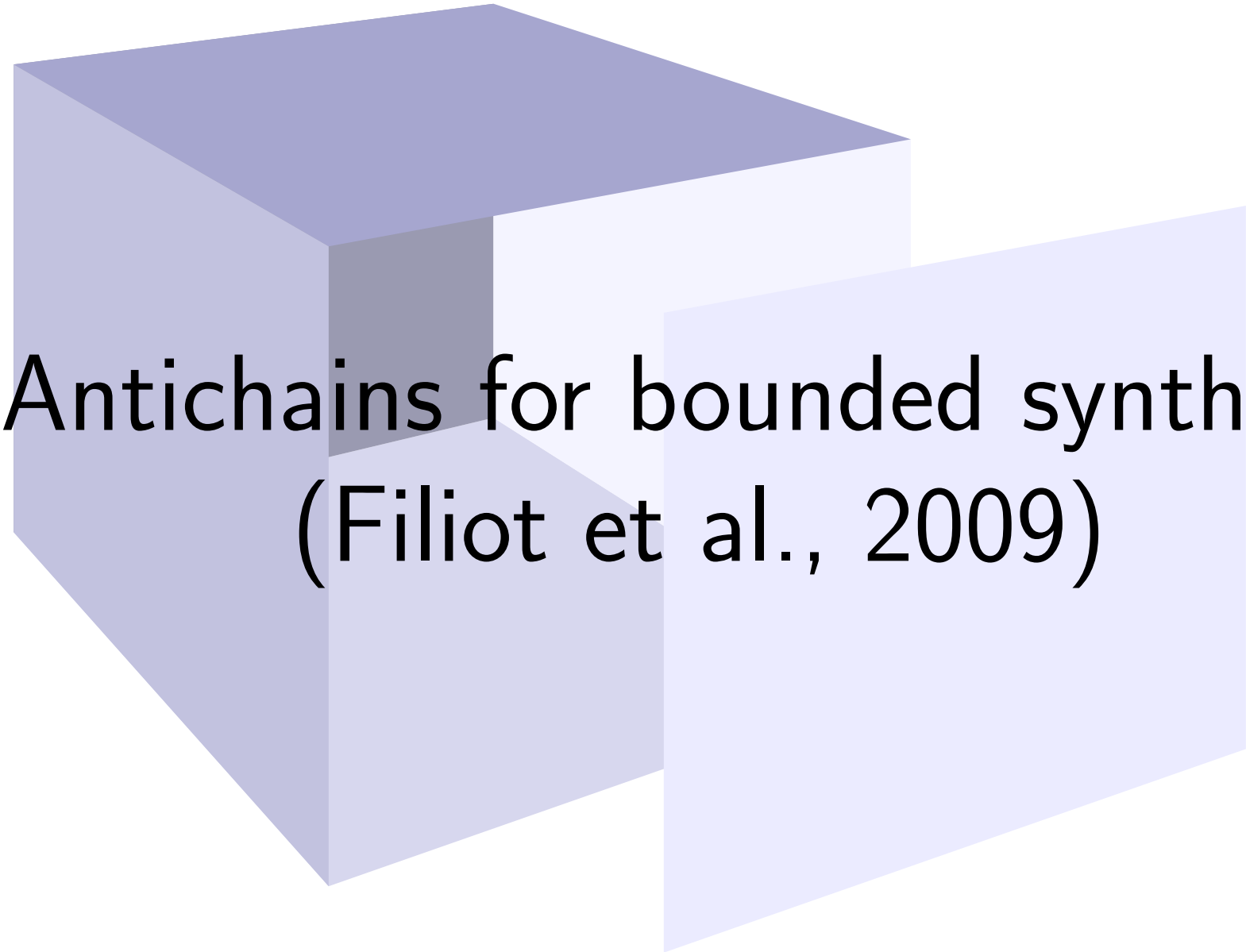
What can we do about this?

What would help?

What we need is a **symbolic way** to build and solve synthesis games.

Sub-topics in this lecture

- Antichains for bounded synthesis (Filiot et al., 2009)
- Binary decision diagrams for GR(1) and safety synthesis (Bloem et al., 2012)



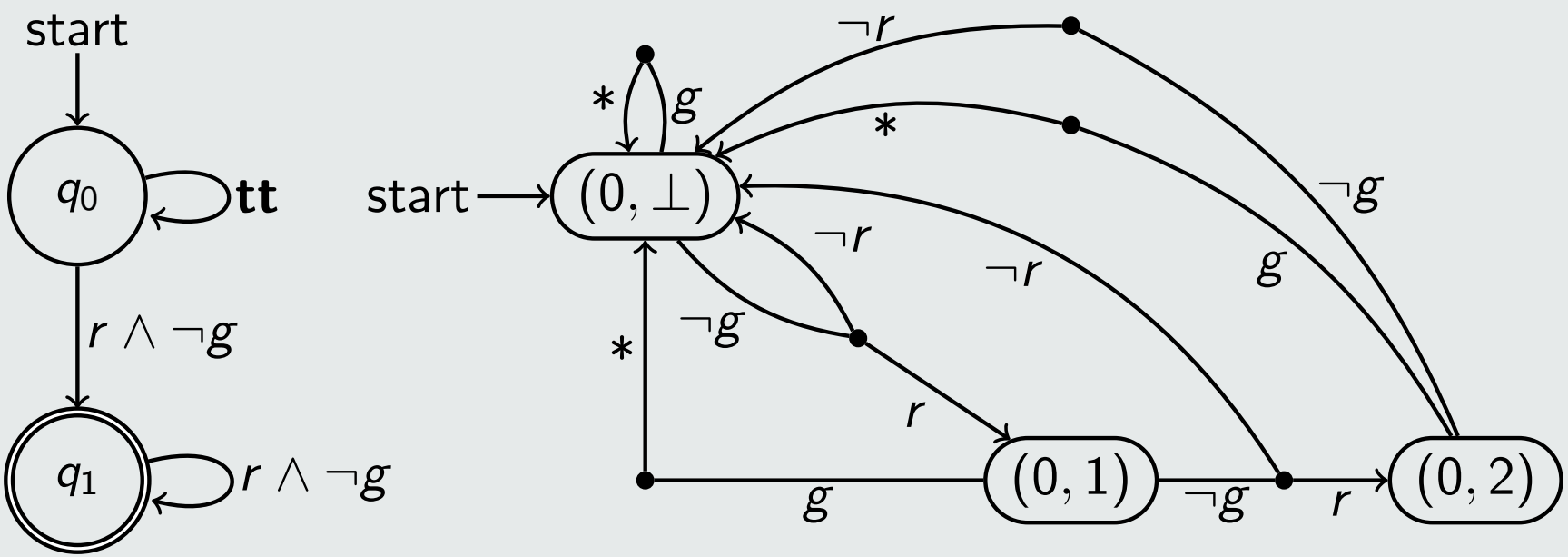
Antichains for bounded synthesis (Filiot et al., 2009)

Flashback: Bounded synthesis

Safety game solving

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

Example



Bounded synthesis: Observations

Game structure

In bounded synthesis, game states are tuples of the form $\{\bot, 0, \ldots, b\}^n$.
Higher values of these counters *restrict* the capabilities of the system.

Bounded synthesis: Observations

Game structure

In bounded synthesis, game states are tuples of the form $\{\bot, 0, \ldots, b\}^n$. Higher values of these counters *restrict* the capabilities of the system.

Example

We know that the set of winning strategies from state $(2, 5, 3, 1, \bot)$ is smaller than or equal to the ones from state $(2, 4, 2, 1, \bot)$, no matter how the specification automaton looks like.

Bounded synthesis: Observations

Game structure

In bounded synthesis, game states are tuples of the form $\{\bot, 0, \ldots, b\}^n$. Higher values of these counters *restrict* the capabilities of the system.

Example

We know that the set of winning strategies from state $(2, 5, 3, 1, \bot)$ is smaller than or equal to the ones from state $(2, 4, 2, 1, \bot)$, no matter how the specification automaton looks like.

Definition

We define a partial order over the counter tuples such that we have $(c_1, \ldots, c_n) \leq (c'_1, \ldots, c'_n)$ if and only if for all $i \in \{1, \ldots, n\}$, we have $c_i \leq c'_i$ *or* $c_i = \bot$.

Antichains-based bounded synthesis: idea

Idea

When evaluating $W = \nu X.X \wedge \mathrm{EnfPre}(X)$, we

1. build the game graph on-the-fly
2. store W in *symbolic* form by listing only the largest elements (w.r.t. the order $\leq$)

Antichains-based bounded synthesis: idea

Idea

When evaluating $W = \nu X.X \wedge \mathrm{EnfPre}(X)$, we

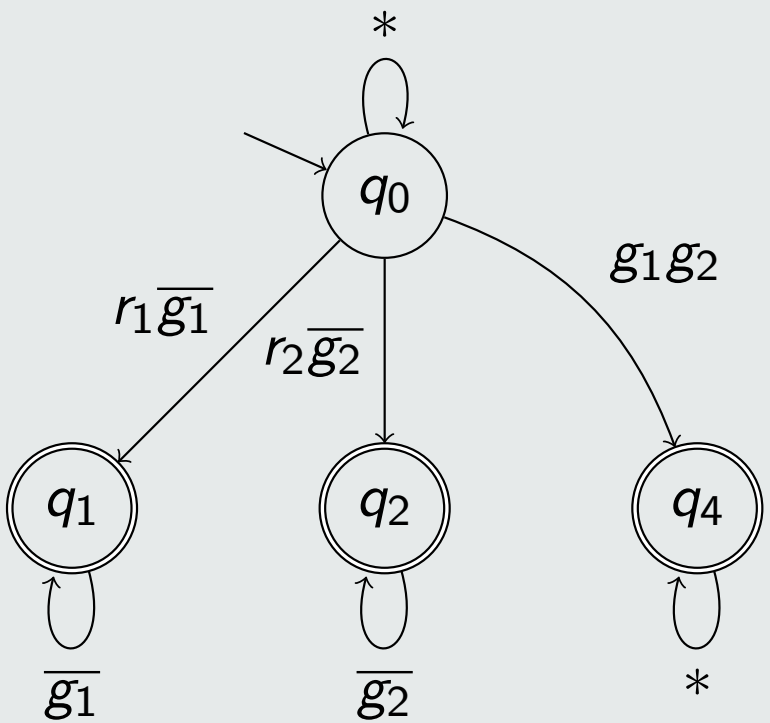
1. build the game graph on-the-fly
2. store W in *symbolic* form by listing only the largest elements (w.r.t. the order $\leq$)

Example

If during evaluation of the fixed point formula for $n = 3$ and $b = 3$, we have that W is represented by $\{(\bot, 2, 1), (\bot, 0, 2)\}$, then all states $\leq$ any of these are contained in W.

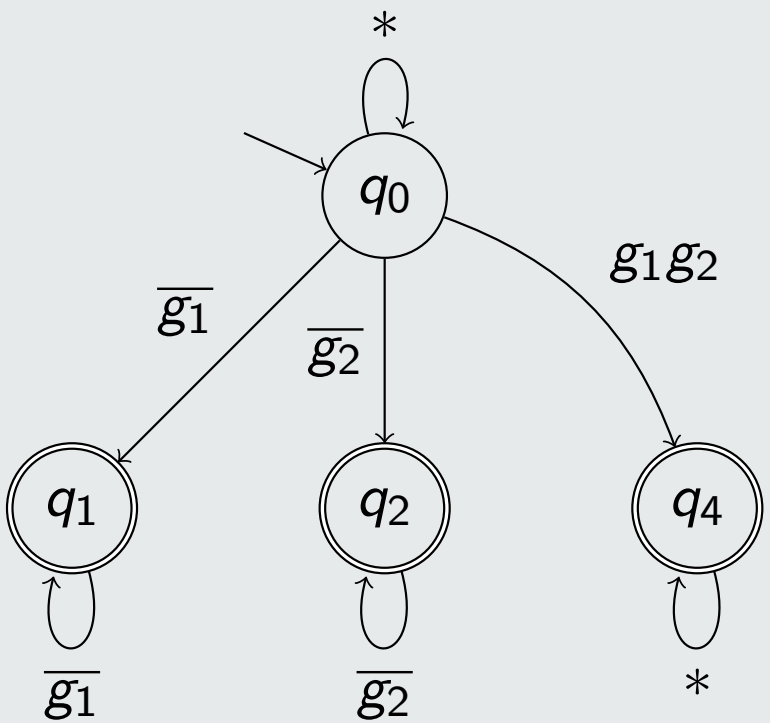
A more complete example (1)

Example specification (original version)



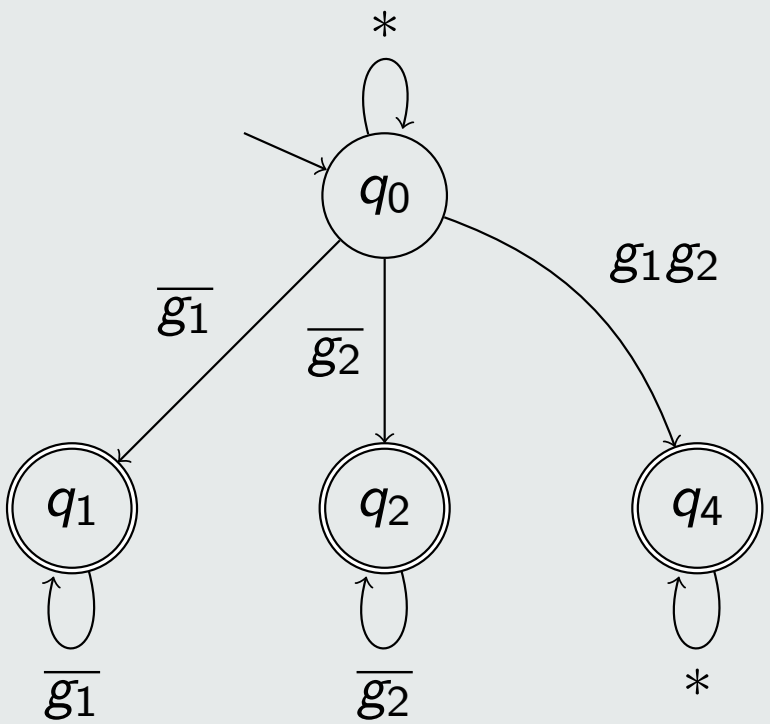
A more complete example (1)

Example specification (simple version)

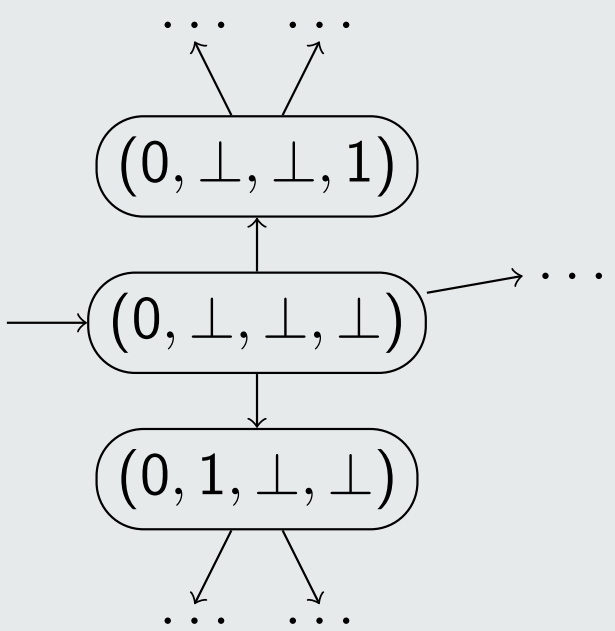


A more complete example (1)

Example specification (simple version)

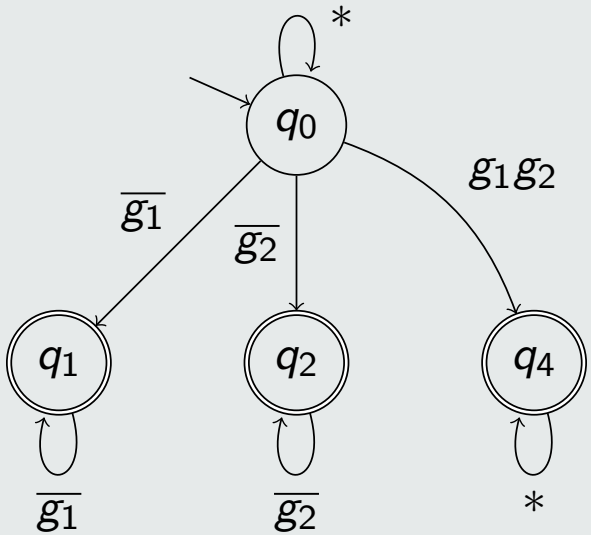


Game excerpt



A more complete example (2)

Example specification (simple version)

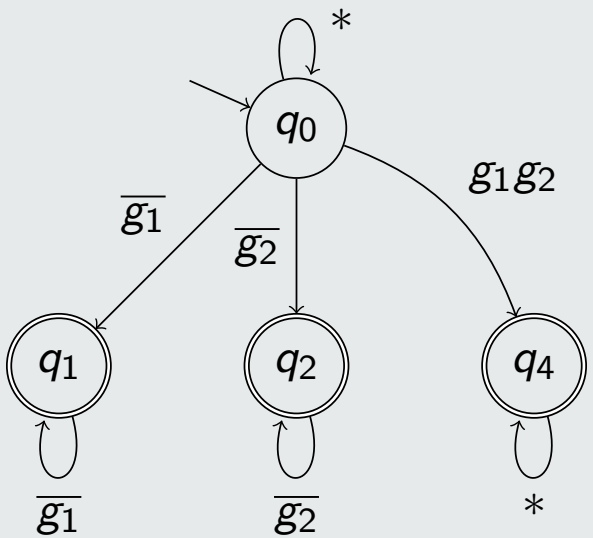


Evolution of the set of winning positions (for $b = 3$)

$W_0 =$ everything $\leq$ one of $\{(3, 3, 3, 3)\}$

A more complete example (2)

Example specification (simple version)

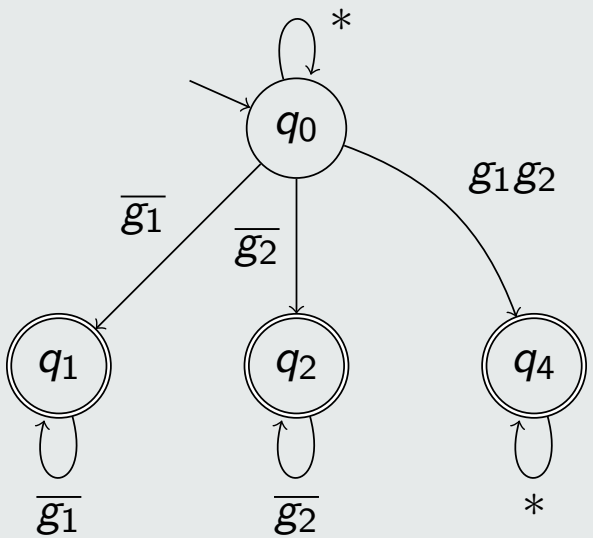


Evolution of the set of winning positions (for $b = 3$)

$W_0 =$ everything $\leq$ one of $\{(3, 3, 3, 3)\}$
$W_1 =$ everything $\leq$ one of $\{(2, 3, 3, 2)\}$

A more complete example (2)

Example specification (simple version)

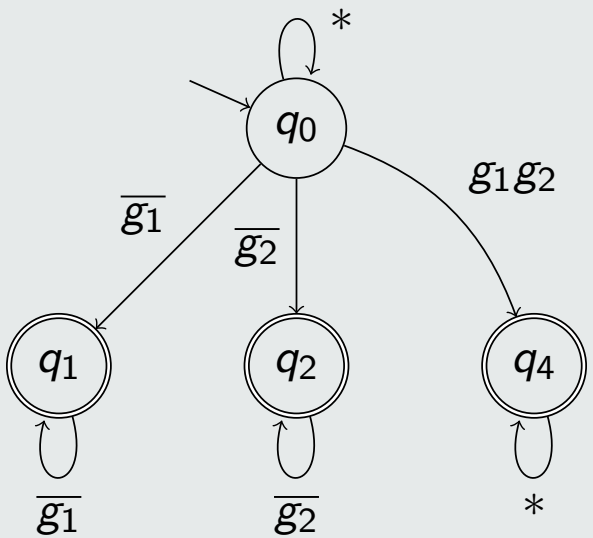


Evolution of the set of winning positions (for $b = 3$)

$W_1 =$ everything $\leq$ one of $\{(2, 3, 3, 2)\}$
$W_2 =$ everything $\leq$ one of $\{(2, 3, 2, 1), (1, 3, 3, 1), (2, 2, 3, 1)\}$

A more complete example (2)

Example specification (simple version)

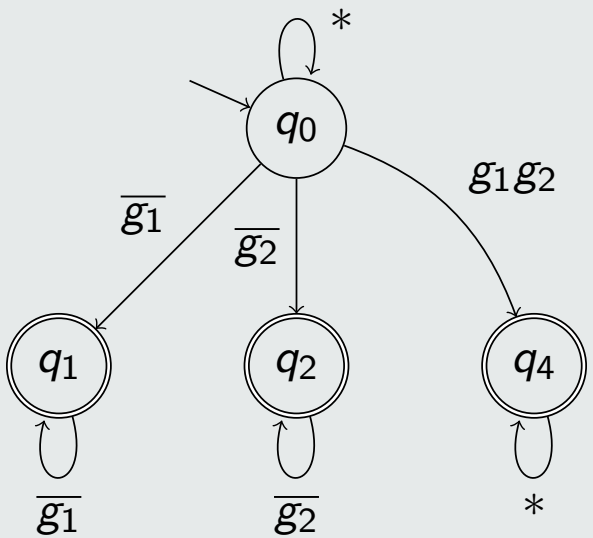


Evolution of the set of winning positions (for $b = 3$)

$W_2 =$ everything $\leq$ one of $\{(2,3,2,1),(1,3,3,1),(2,2,3,1)\}$
$W_3 =$ everything $\leq$ one of $\{(2,2,3,0),(0,3,3,0),(2,3,2,0)\}$

A more complete example (2)

Example specification (simple version)

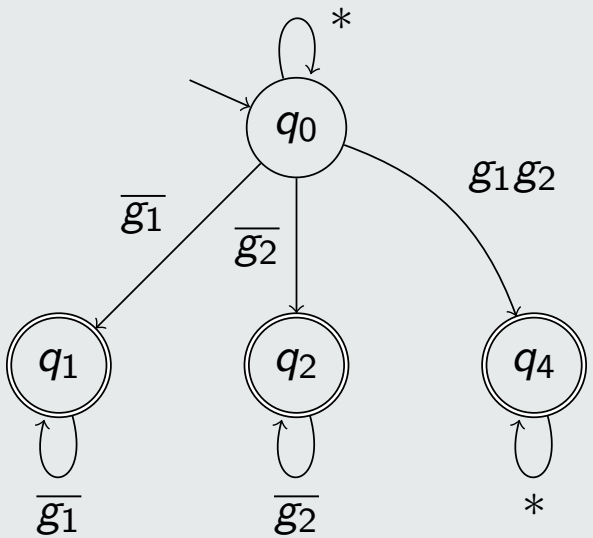


Evolution of the set of winning positions (for $b = 3$)

$W_3 =$ everything $\leq$ one of $\{(2, 2, 3, 0), (0, 3, 3, 0), (2, 3, 2, 0)\}$
$W_4 =$ everything $\leq$ one of $\{(2, 2, 3, \bot), (2, 3, 2, \bot)\}$

A more complete example (2)

Example specification (simple version)



Evolution of the set of winning positions (for $b = 3$)

$W_4 =$ everything $\leq$ one of $\{(2, 2, 3, \bot), (2, 3, 2, \bot)\}$

$W_5 = W_4$

Some notes

Sizes of the representations of W_i

The game had $5^4 = 25 \cdot 25 = 625$ states, but the representation of W was always small.

Some notes

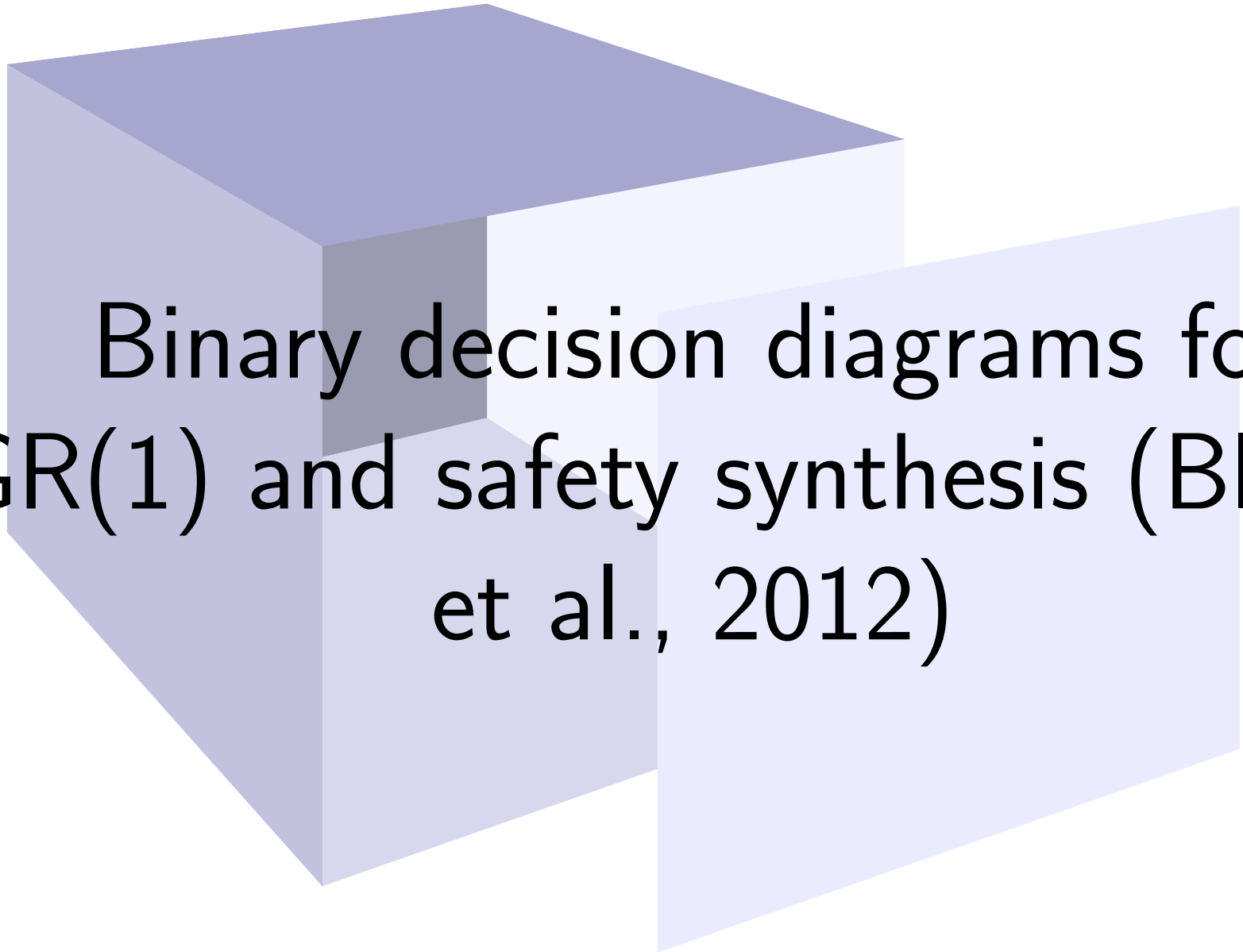
Sizes of the representations of W_i

The game had $5^4 = 25 \cdot 25 = 625$ states, but the representation of W was always small.

Implementation

The tool Acacia+ (Bohy et al., 2012) implements this approach. The representation of W is called an *antichain*.

The final synthesized implementation does never have more states than the number of elements in the representation of W.



Binary decision diagrams for GR(1) and safety synthesis (Bloem et al., 2012)

Problem with GR(1) synthesis

As always, scalability

For example, if we have 10 input propositions and 10 output propositions, then there can be up to 2^{40} edges in the game.
$\rightarrow$ too large for memory

Idea

Represent the allowed transitions for the environment and the system as *characteristic Boolean functions* and use a suitable data structure for Boolean functions.

Boolean functions

Formal definition

Given a set of atomic propositions $\mathcal{X}$, a function f is a boolean function if f maps elements from $(\mathcal{X} \to \mathbb{B})$ to $\mathbb{B}$.

Note

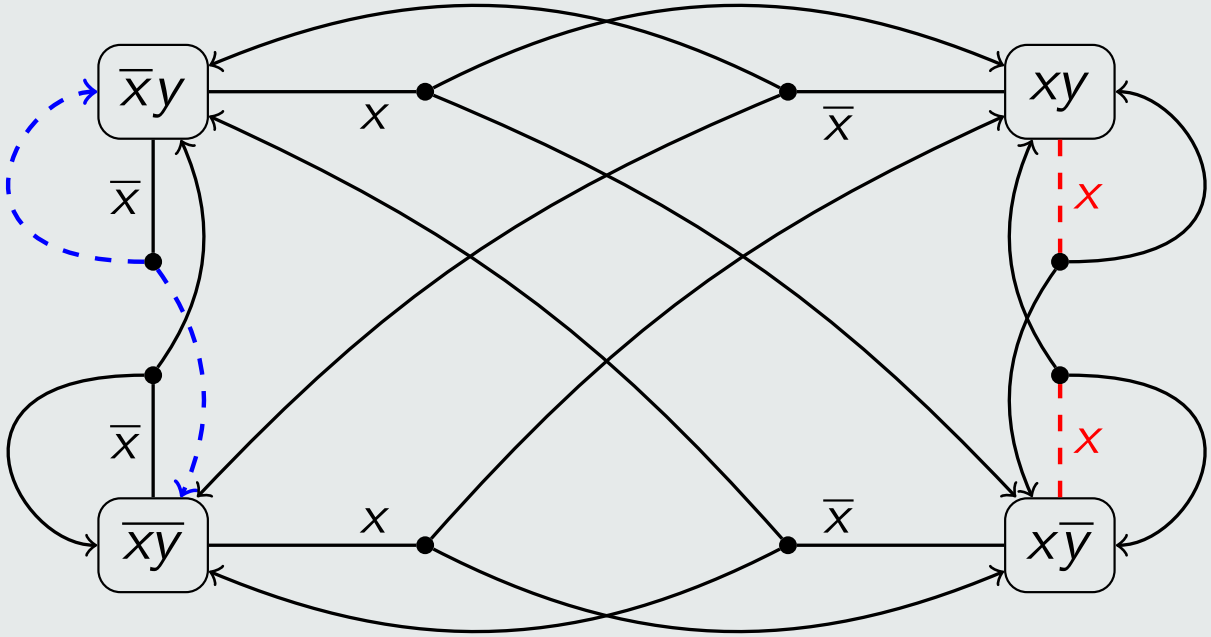
We treat $2^{\mathcal{X}}$ and $(\mathcal{X} \to \mathbb{B})$ interchangeably in the following.

Example

$$f(x, y, z) = (x \wedge y) \vee (\neg x \wedge z)$$

Encoding example – System transitions

The game

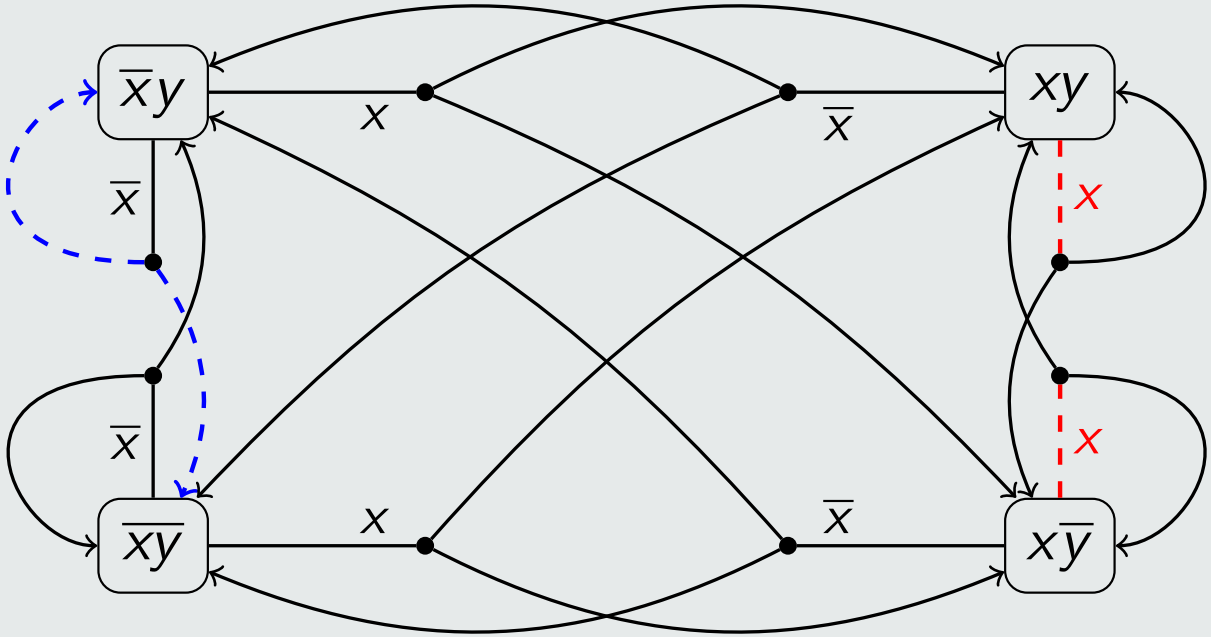


Representation as Boolean formula over $\mathcal{V} = \{x, y, x', y'\}$

$f =$ **false**

Encoding example – System transitions

The game

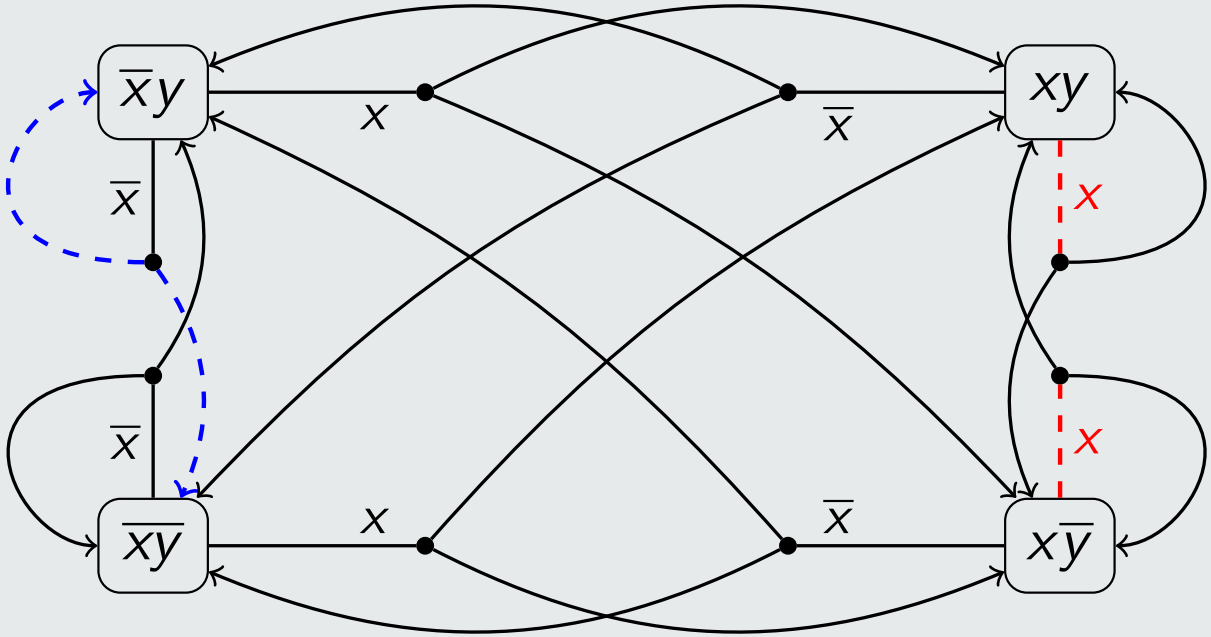


Representation as Boolean formula over $\mathcal{V} = \{x, y, x', y'\}$

$f = \overline{xyx'y'}$

Encoding example – System transitions

The game

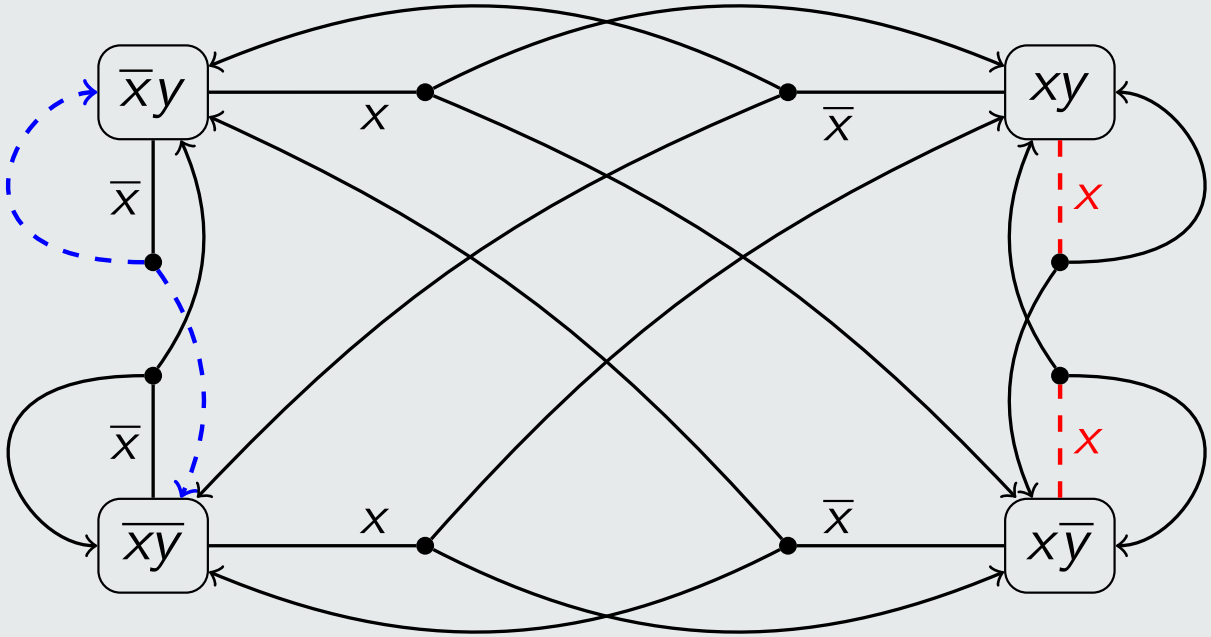


Representation as Boolean formula over $\mathcal{V} = \{x, y, x', y'\}$

$f = \overline{xyx'y'} \vee \overline{xy}x'y'$

Encoding example – System transitions

The game

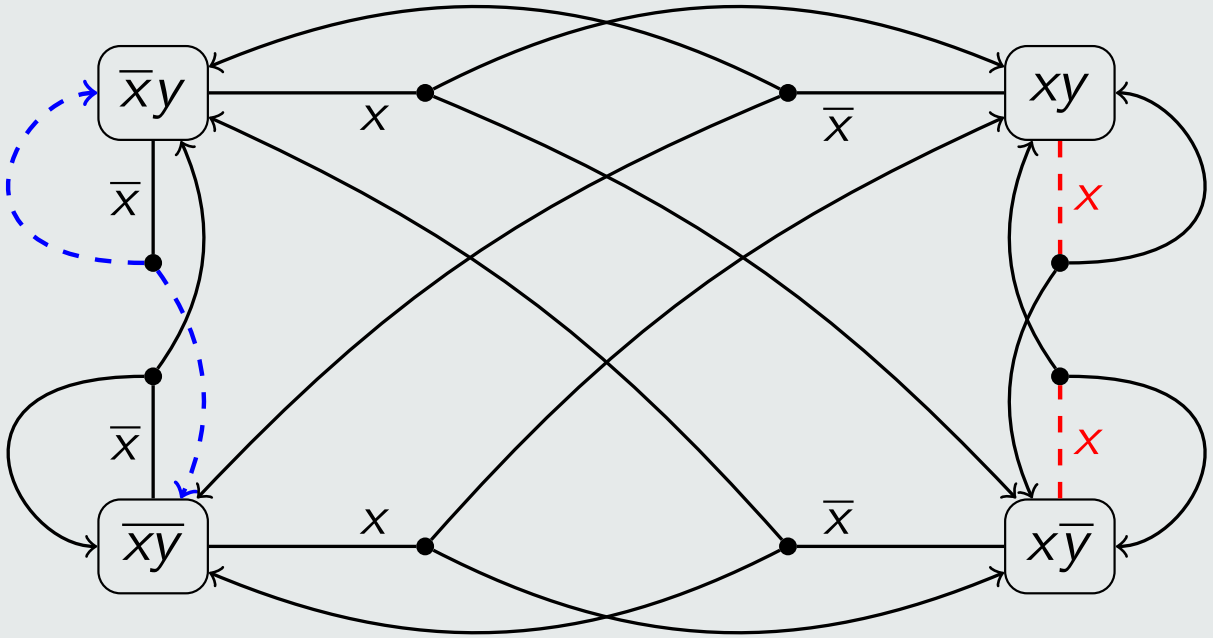


Representation as Boolean formula over $\mathcal{V} = \{x, y, x', y'\}$

$f = \overline{xyx'y'} \vee \overline{xy}x'y' \vee \ldots$

Encoding example – System transitions

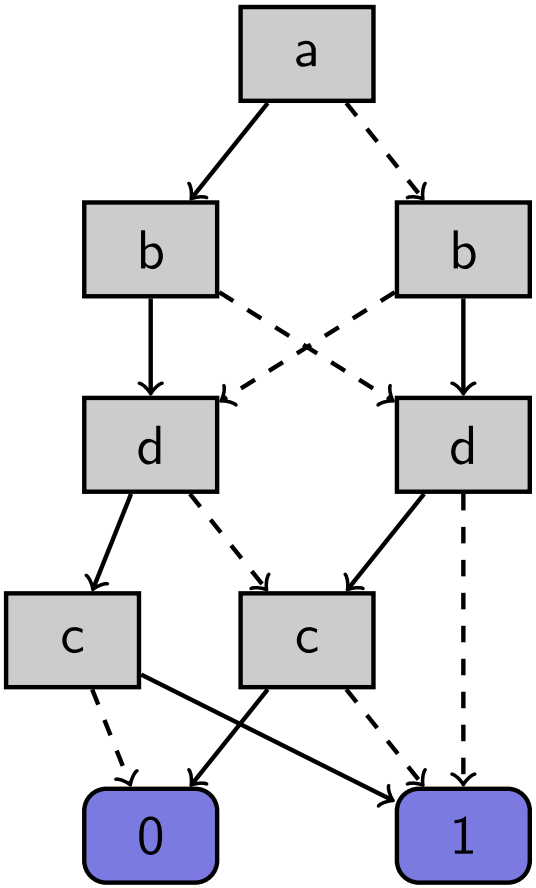
The game



Representation as Boolean formula over $\mathcal{V} = \{x, y, x', y'\}$

$f = \neg(\overline{x}y\overline{x'})$

Binary decision diagrams (BDDs)



Basic ideas

- They represent boolean functions as directed acyclic graphs from a root to 0 and 1 nodes.
- From every node, there is a **false**- and a **true** successor.
- BDDs are normally **reduced** $\rightarrow$ no two nodes have the same successors and no node has the same node as its two successors
- All commonly used Boolean operations can be efficiently performed on BDDs!

Operations available for BDDs

Example: Disjunction

- Given BDDs for f and f' over $\mathcal{X}$, we can compute a BDD for $(f \vee f')$ such that for every $v \in 2^{\mathcal{X}}$, we have $(f \vee f')(v) = \textbf{true}$ if and only if $f(v) = \textbf{true}$ or $f'(v) = \textbf{true}$.

Operations available for BDDs

Example: Disjunction

- Given BDDs for f and f' over $\mathcal{X}$, we can compute a BDD for $(f \vee f')$ such that for every $v \in 2^{\mathcal{X}}$, we have $(f \vee f')(v) = \textbf{true}$ if and only if $f(v) = \textbf{true}$ or $f'(v) = \textbf{true}$.

Example: Complementation

- Given a BDD for f over $\mathcal{X}$, we can compute a BDD for $(\neg f)$ such that for every $v \in 2^{\mathcal{X}}$, we have $(\neg f)(v) = \textbf{true}$ if and only if $f(v) = \textbf{false}$.

Operations available for BDDs

Example: Disjunction

- Given BDDs for f and f' over $\mathcal{X}$, we can compute a BDD for $(f \vee f')$ such that for every $v \in 2^{\mathcal{X}}$, we have $(f \vee f')(v) = \textbf{true}$ if and only if $f(v) = \textbf{true}$ or $f'(v) = \textbf{true}$.

Example: Complementation

- Given a BDD for f over $\mathcal{X}$, we can compute a BDD for $(\neg f)$ such that for every $v \in 2^{\mathcal{X}}$, we have $(\neg f)(v) = \textbf{true}$ if and only if $f(v) = \textbf{false}$.

Example: Existential abstraction

- Given a BDD for f over $\mathcal{X}$ and a set of variables $X \subseteq \mathcal{X}$, we can compute a BDD for $(\exists X.f)$ such that for every $v \in 2^{\mathcal{X}}$, we have $(\exists X.f)(v) = \textbf{true}$ if and only if there exists an $x \subseteq X$ such that $f((v \setminus X) \cup x) = \textbf{true}$.

Example: Working with BDDs

```
#include "BF.h"
#include "bddDump.h"

int main() {

    // Allocate BDD data structure
    BFManager mgr;

    // Allocate BDD variables
    BF x = mgr.newVariable();
    BF y = mgr.newVariable();
    BF z = mgr.newVariable();
    BF a = mgr.newVariable();

    // Build some non-trivial BDDs
    BF f = (!x & !y) | (y & z) | (z & a) | (x & y);
    BF g = f.ExistAbstractSingleVar(a);

    // Prepare drawing the BDD
    std::vector<BF> allVars{x,y,z,a};
    std::vector<std::string> allVarNames{"x","y","z","a"};
    auto drawingInfoContainer =
           TrivialVariableInfoContainer(mgr,allVarNames,allVars);

    // Draw f and g
    BF_newDumpDot(drawingInfoContainer,f,NULL,"f.dot");
    BF_newDumpDot(drawingInfoContainer,g,NULL,"g.dot");
}
```

Game solving with BDDs

Safety games

Original fixpoint formula:

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

Game solving with BDDs

Safety games

Original fixpoint formula:

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

Safety games

Assuming the pre/post input and output variables are named $\mathcal{I}/\mathcal{I}'$ and $\mathcal{O}/\mathcal{O}'$, respectively, we need to evaluate

$$W = \nu X.X \wedge \forall \mathcal{I}'.(T_e \wedge \exists \mathcal{O}'.(T_s \wedge X[post/pre])),$$

where T_e is a BDD for the allowed environment player transitions and T_s represents the safety player transitions.

Simple parity game solving

Fixedpoint equation by Walukiewicz (1996)

For the special case of strict alternation between the two players and only environment player positions having colors other than 0:

$$\sigma X_n, \ldots, \mu X_1 . \nu X_0 . \mathrm{EnfPre}\left(\bigcup_{i \in \{0,\ldots,n\}} (C_i \cap X_i) \right)$$

Simple parity game solving

Fixedpoint equation by Walukiewicz (1996)

For the special case of strict alternation between the two players and only environment player positions having colors other than 0:

$$\sigma X_n, \ldots, \mu X_1 . \nu X_0 . \mathrm{EnfPre}\left(\bigcup_{i \in \{0,\ldots,n\}} (C_i \cap X_i) \right)$$

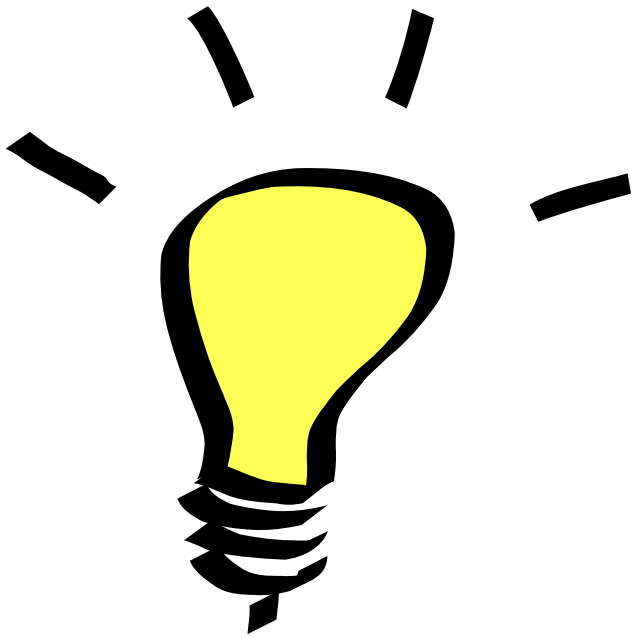
Coming up next...

...a demo with SCOTS (Rungger and Zamani, 2016)!

Symbolic Synthesis – Conclusion

Summary

- By making use of the regularity of the games dealt with in synthesis, we gain efficiency
- BDDs can kind-of always be applied, but for bounded synthesis, there is a specialized data structure available
- Symbolic methods can be used in combination with CPS abstractions.



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.

Aaron Bohy, Véronique Bruyère, Emmanuel Filiot, Naiyong Jin, and Jean-François Raskin. Acacia+, a tool for LTL synthesis. In *Computer Aided Verification - 24th International Conference, CAV 2012, Berkeley, CA, USA, July 7-13, 2012 Proceedings*, pages 652–657, 2012. doi: 10.1007/978-3-642-31424-7_45. URL https://doi.org/10.1007/978-3-642-31424-7_45.

Emmanuel Filiot, Naiyong Jin, and Jean-François Raskin. An antichain algorithm for LTL realizability. In *Computer Aided Verification, 21st International Conference, CAV 2009, Grenoble, France, June 26 - July 2, 2009. Proceedings*, pages 263–277, 2009. doi: 10.1007/978-3-642-02658-4_22. URL https://doi.org/10.1007/978-3-642-02658-4_22.

Matthias Rungger and Majid Zamani. SCOTS: A tool for the synthesis of symbolic controllers. In *Proceedings of the 19th International Conference on Hybrid Systems: Computation and Control, HSCC 2016, Vienna, Austria, April 12-14, 2016*, pages 99–104, 2016. doi: 10.1145/2883817.2883834. URL http://doi.acm.org/10.1145/2883817.2883834.

Igor Walukiewicz. Monadic second order logic on tree-like structures. In *STACS 96, 13th Annual Symposium on Theoretical Aspects of Computer Science, Grenoble, France, February 22-24, 1996, Proceedings*, pages 401–413, 1996. doi: 10.1007/3-540-60922-9_33. URL https://doi.org/10.1007/3-540-60922-9_33.