Reactive Synthesis

Swen Jacobs <swen.jacobs@iaik.tugraz.at>

VTSA 2013

Nancy, France

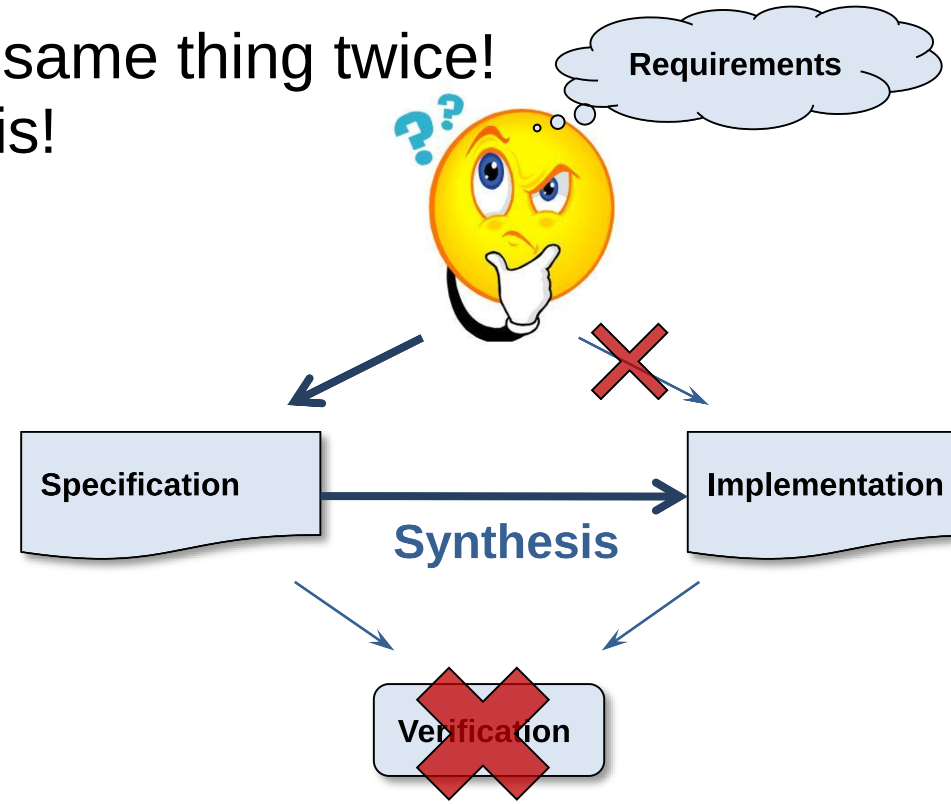
24.09.2013

Secure & Correct Systems

Property Synthesis
(You Will Never Code Again)

Construct Correct Systems Automatically

Don't do the same thing twice!
Use synthesis!



Motivation

- Coding is **hard**, want higher level of **abstraction:**
  Machine code ⇒ Assembly ⇒ C ⇒ Java ⇒ Ruby? ⇒ …
  Silicon ⇒ Gates ⇒ RTL ⇒ Transactions? ⇒ …
- Bugs are:
  - **very expensive**, especially in security critical applications and hardware
  - **hard to kill**: finding and fixing bugs takes 50%-80% of design time

Our Focus

- Reactive systems
  - Continuous interaction with environment
  - Correctness statements are temporal (temporal logic, automata)
  - Ex: Operating systems, web browsers, circuits, protocols
- Finite State
  - Prototypical finite state reactive system: circuit
- Not our focus: functions
  - One input, one output, non-termination is a bug
  - Correctness is input/output relation (Hoare logic)

Other Application Areas

- Program **repair**
- Program **sketching**
- Synthesis of **synchronization skeletons**
- …

Synthesis, Part I: Basics

- Synthesis as a **Game**
- **General**: LTL Synthesis
- **Time-Efficient**: GR(1) Synthesis
- **Application**: AMBA Bus Protocol
- **Space-Efficient**: Bounded/Safraless Approaches

Synthesis as a Game

Synthesis as a Game

Given

- **Input** and **output** signals
- Specification of the behavior

Determine

- **Realizability:** Is there a finite state system that realizes the specification?
- **Synthesis:** If system exists, **construct** it

Two player game

- **Environment**: determines inputs (not controllable)
- **System**: determines outputs (controllable)
- **Game**: finite state graph, infinite plays
- **Winning condition** for player **System**: formula φ

Games

Two player graph-based, turn-based games with infinitary winning conditions

- Antagonist controls I
- Protagonist controls O
- graph based:
  - Set of states Q
  - Initial state q_0
  - Transition function $\delta: Q \times I \times O \rightarrow Q$
- turn based:
  - Start from q_0
  - Antagonist selects i_k, protagonist selects o_k, proceed to $q_{k+1} = \delta(i_k, o_k)$
  - Ensuing play: $q_0\ i_0\ o_0\ q_1\ i_1\ o_1\ q_2$ ...
- **Winning condition**: objective over $\mathrm{F} \subseteq Q$
- **Strategy**: $Q \times I^* \rightarrow O$
- For every input sequence, strategy fixes a play
- **Winning strategy:** strategy such that all resulting plays fulfill φ

Winning Conditions

- **Reachability**: want to reach a state in $\mathrm{F} \subseteq Q$
- **Safety**: want to stay in $\mathrm{F} \subseteq Q$
- **Büchi**: want to visit $\mathrm{F} \subseteq Q$ infinitely often
- **Co-Büchi**: want to visit $\mathrm{F} \subseteq Q$ only finitely often
- others exist… (later)

Example

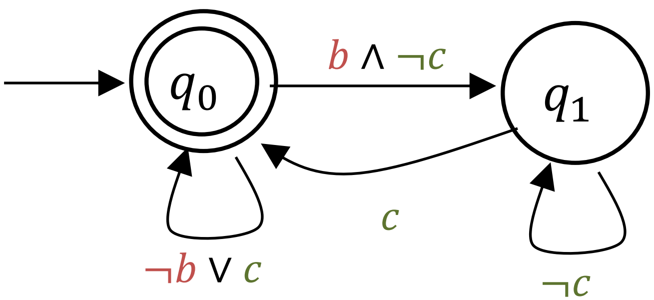
input button
output coffee
$\mathbf{G}(\mathit{button} \rightarrow \mathbf{F}\, \mathit{coffee})$

LTL game

red moves first
green moves second

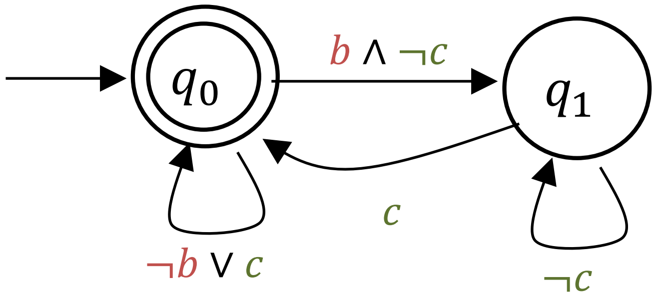
green's objective: visit q_0 infinitely often

Büchi game

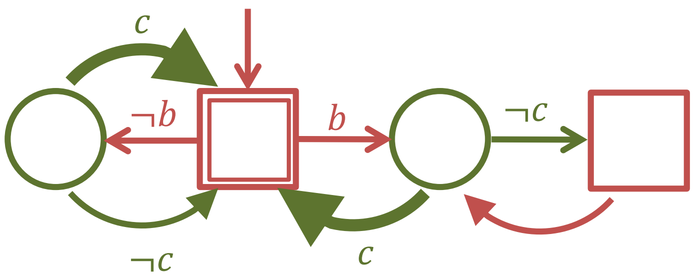


Possible strategy:
serve coffee iff automaton is in state q_1
In this case, LTL game reduces to Büchi game

Example: Alternative Representation

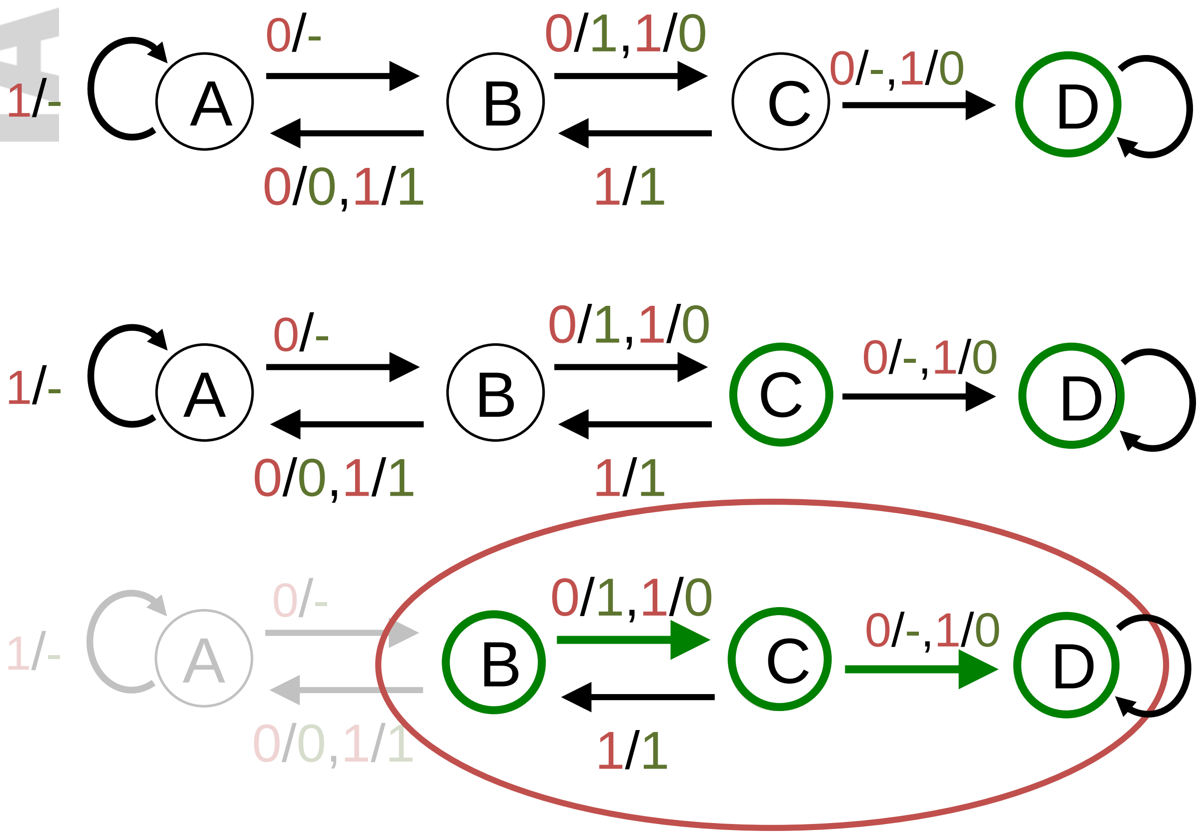


- compact
- looks like automaton
- order of moves (input, output) only implicit



- explicit order of moves
- need more states

Symbolic Computation: Fixpoints



Winning region

Label on edges:

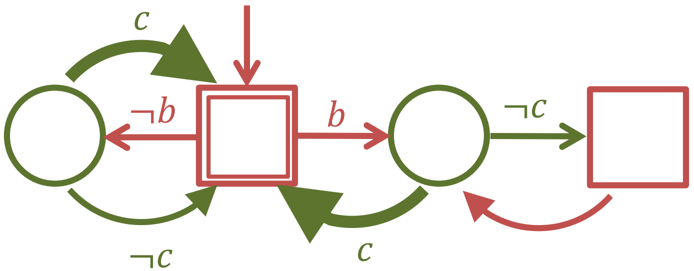
- Environment input
- System output

dash (–) means don‘t care

Find all states from which system can force visit to goal state (= winning region / attractor) + a strategy

SCOS Secure & Correct Systems

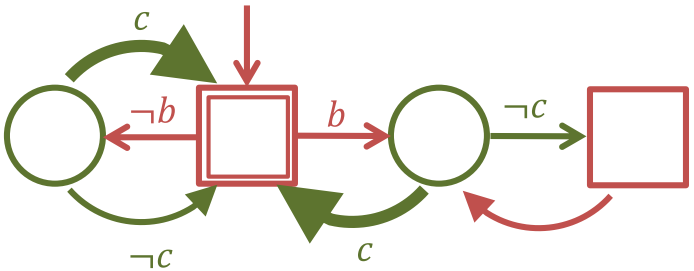
Computing Büchi Games



$\boldsymbol{Force^1(F)}$ = set of states from which system can force visit to S in one step

$$\boldsymbol{Force^1(F)} = \{\boldsymbol{q} \in \boldsymbol{Q} \mid \forall i \in I\ \exists o \in O\colon \delta(q, i, o) \in F\}$$

Computing Büchi Games

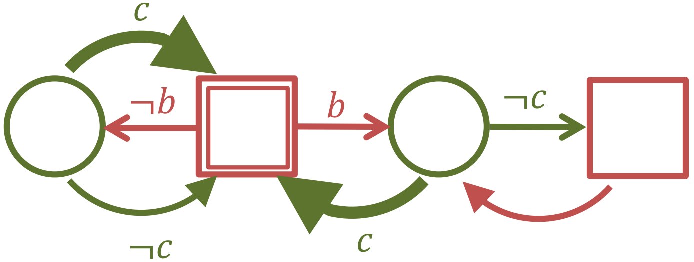


$\boldsymbol{Force^1(F)}$ = set of states from which system can force visit to F in one step

$\boldsymbol{Force^*(F)}$ = set of states from which system can force visit to F in any number of steps (least fixpoint of applying $Force^1$ to F)

$\boldsymbol{Recur(F)}$ = set of states from which system can repeatedly force visit to F in any number of steps (nested fixpoint operation)

Computing Büchi Games



Winning region is $\boldsymbol{Force}^*(\boldsymbol{F})$ for reachability game, $\boldsymbol{Recur}(\boldsymbol{F})$ for Büchi game.

(Safety defined with dual $Force$ operator for environment)

For reachability, safety and Büchi games, **memoryless** strategies are sufficient, i.e., strategies $Q \times I \rightarrow O$

SCOS Secure & Correct Systems

FourSteps to Synthesis

1. Specify
   - LTL, Büchi automata,…
2. Obtain a game
3. Solve the game
4. Construct circuit

LTL Synthesis

LTL Synthesis

LTL Synthesis [PnueliRosner89]

1. Specify
   - Formula φ in LTL
2. Obtain a game
   - Convert φ to nondeterministic Büchi automaton A **(exponential blowup)**
   - Convert A to **deterministic** Rabin or Parity automaton (=game) **(exponential blowup)**
3. Solve the game
   - parity games can be solved in polynomial time
4. Construct Circuit

LTL Synthesis is **hard**

SCOS Secure & Correct Systems

Arbiter: From LTL to Büchi

1. **Specify**
2. Obtain a game
3. Solve the game
4. Construct circuit

Input: r_1, r_2 (requests)

Output: g_1, g_2 (grants)

Specification:

$$\mathbf{G}(r_1 \rightarrow \mathbf{F}\, g_1)$$
$$\mathbf{G}(r_2 \rightarrow \mathbf{F}\, g_2)$$
$$\mathbf{G}\neg(g_1 \wedge g_2)$$

Obtaining a game

- From LTL to Büchi automata
  - Not in detail in this tutorial – see [VardiWolper86]
- From Büchi automata to games
  - Non-determinism is bad
  - Advanced acceptance conditions

Arbiter: From LTL to Büchi

1. Specify
2. **Obtain a game**
3. Solve the game
4. Construct circuit

Input: r_1, r_2 (requests)

Output: g_1, g_2 (grants)

Specification:

$\mathbf{G}(r_1 \rightarrow \mathbf{F}\, g_1)$

$\mathbf{G}(r_2 \rightarrow \mathbf{F}\, g_2)$

$\mathbf{G}\neg(g_1 \land g_2)$

Blackboard

Nondeterminism is bad

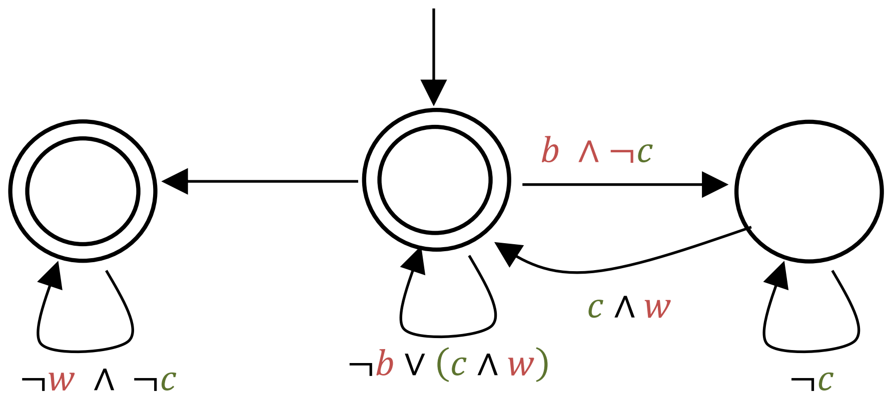
Need to determinize.

```
input button, water
output coffee
```

LTL game

$$\big(\mathbf{GF}(water) \rightarrow \mathbf{G}(button \rightarrow \mathbf{F}\, coffee)\big) \wedge \mathbf{G}(\neg water \rightarrow \neg coffee)$$

won?



Note: not complete!

Büchi game

won?

No winning strategy because of nondeterminism, even though LTL game is won

Advanced Acceptance Conditions

- **Rabin**: defined by $\{(E_1, F_1), \ldots, (E_n, F_n)\}$, with $E_i, F_i \subseteq Q$. System wins if there **exists an *i*** such that E_i is visited finitely often **and** F_i is visited infinitely often.
- **Streett**: like Rabin, but System wins if **for all *i***, if F_i is visited infinitely often, **then** E_i must be visited infinitely often. (negation of Rabin)
- **Parity**: every state is assigned a **priority** from $\mathbb{N}$. System wins if **minimum** priority of all states visited infinitely often is **even**.

LTL Synthesis

1. Specify
   - Formula φ in LTL, size n
2. Obtain a game
   - Convert φ to a nondeterministic Büchi Automaton A, size 2^n
   - Determinize A to a deterministic Parity automaton (=game), size 2^{2^n}
3. Solve the parity game, time 2^{2^n}

Will not consider this approach in detail.
It is complex and not very scalable.

LTL Synthesis – Alternative Approaches

Synthesis problem can also be solved by

- decomposing φ, simplifying each part, then composing [SohailSomenzi09, MorgensternSchneider10] (not in this tutorial)
- Limiting size of solution, incrementally increasing bound [ScheweFinkbeiner07,FiliotJinRaskin11, Ehlers12] (Later!)
- Considering efficiently decidable fragments (**Now!**)

GR(1) Synthesis

Avoiding Complexity: GR(1) Games

LTL Synthesis [PnueliRosner89]

1. Specify
   - Formula φ in Linear Temporal Logic
2. Obtain a game
   - Convert φ to a nondeterministic Büchi Automaton A (exponential blowup)
   - Determinize A to a deterministic Rabin or Parity automaton (=game) (exponential blowup)
3. Solve the game
   - equals solving a parity game, can be done in polynomial time
4. Construct Circuit

GR(1) Synthesis [PitermanPnueliSa'ar06]

1. Specify
   - Sets of deterministic Büchi automata, for environment and system
2. Specification = game
   - no work
3. Solve the game
   - A GR(1) game
4. Construct Circuit

SCOS
Secure & Correct Systems

Avoiding Complexity: GR(1) Specs

1. Specification:
   - Set of m deterministic Büchi automata for assumptions: A_1 ... A_m
   - Set of n deterministic Büchi automata for guarantees: G_1 ... G_n
     Both encoded symbolically
2. Specification = Game
3. Solve the game
4. Determine circuit from winning strategy (…)

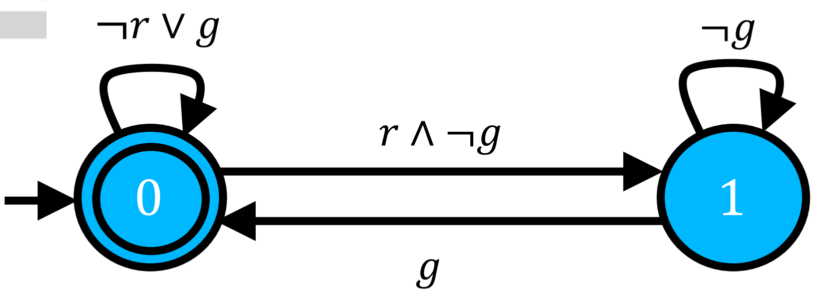
Advantages of this setting:

- We do not need one automaton for full spec
- We do not need to determinize
- Symbolic formulation

But: not all LTL properties can be expressed this way(!)

Obtaining a GR(1) Specification

Example: G(r → F g)



Symbolic:

Introduce x as variable for state space

initial $i_A = \neg x$

transition relation T_A =

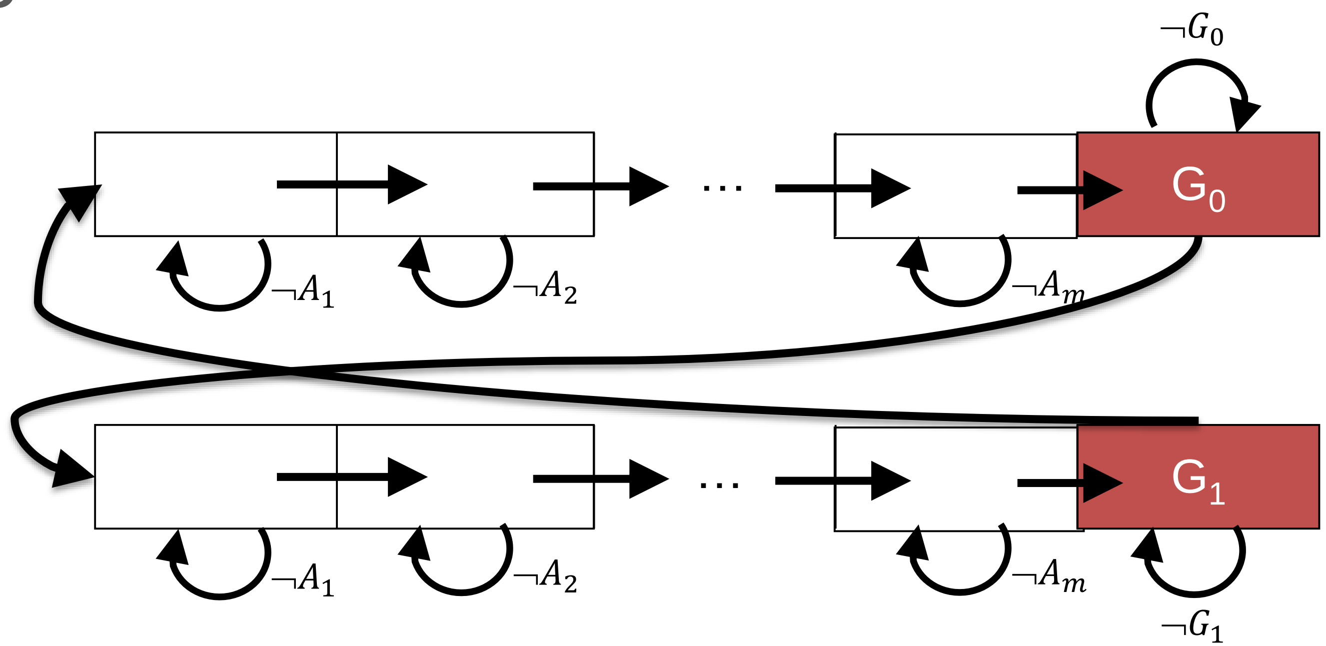
$$\neg x \wedge (\neg r \vee g) \rightarrow \neg x'$$
$$\neg x \wedge r \wedge \neg g \rightarrow x'$$
$$x \wedge \neg g \rightarrow x'$$
$$x \wedge g \rightarrow \neg x'$$

fairness F_A: $\mathbf{GF}\, \neg state'$

Note: symbolic automaton is a (passive/monitor) circuit

Computing a GR1 Game

Gen. Reactivity(1):
$\bigwedge_i (\mathbf{GF}\ A_i) \rightarrow \bigwedge_j (\mathbf{GF}\ G_j)$



To solve: compute nested fixpoints of states from which system can force visit to G_j if environment satisfies assumptions A_i

Direct symbolic implementation. Complexity: $O(|Q|^2 \cdot |T| \cdot m \cdot n)$
[KestenPitermanPnueli05,PitermanPnueliSa'ar06]

Alternative: Reduce GR(1) to Streett

GR(1) $A_1 \wedge \ldots \wedge A_m \to G_1 \wedge \ldots \wedge G_n$

counting construction
blowup: O(m)

reduction

counting construction
blowup:O(n)

1-pair Streett $A \to G$

Note: counting construction on G introduces memory of size n

Solve using Jurdzinki's algorithm in $O(|Q| \cdot |T|)$ time

[d'AlfaroFaella09]

better because $m, n << Q$

	[PPS06]	Streett reduction algorithm
time	$O(\lvert Q\rvert^2 \cdot \lvert T\rvert \cdot m \cdot n)$	$O(\lvert Q\rvert \cdot \lvert T\rvert \cdot (m \cdot n)^2)$

Avoiding Complexity: GR(1) Games

LTL Synthesis [PnueliRosner89]

1. Specify
   - Formula φ in Linear Temporal Logic
2. Obtain a game
   - Convert φ to a nondeterministic Büchi Automaton A (exponential blowup)
   - Determinize A to a deterministic Rabin or Parity automaton (=game) (exponential blowup)
3. Solve the game
   - equals solving a parity game, can be done in polynomial time
4. Construct Circuit

GR(1) Synthesis [PitermanPnueliSa'ar06]

1. Specify
   - Sets of deterministic Büchi automata, for environment and system
2. Specification = game
   - no work
3. Solve the game
   - A GR(1) game
4. **Construct Circuit**

Selecting One Implementation

Specification = Set of sequential circuits

↓ GR(1) Synthesis
(fix memory elements)

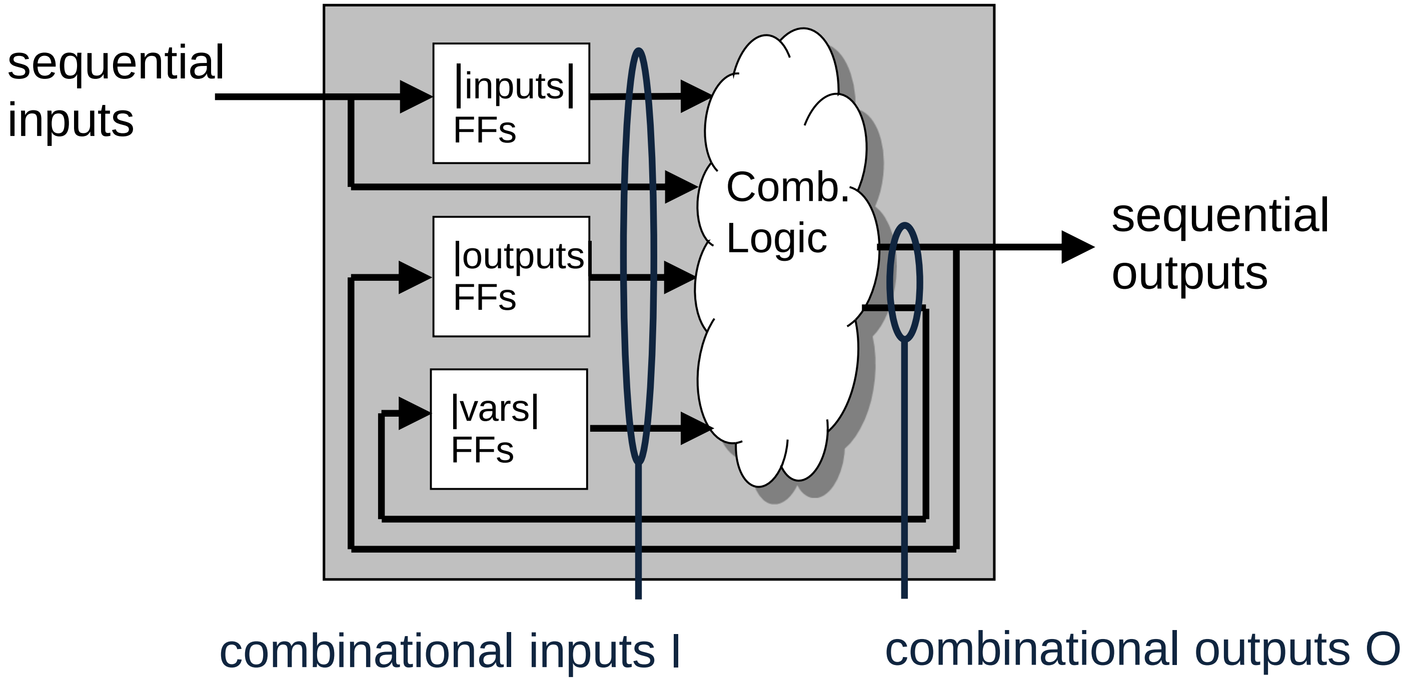
Strategy = Set of combinational circuits

↓ Construction of circuit

One combinational circuit

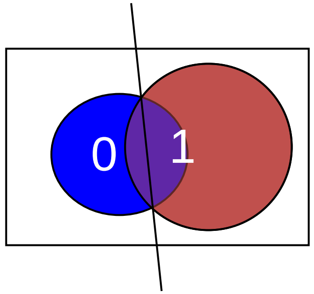
Less freedom
Fewer circuits
More complexity

Constructing Circuit



- Spec is given in terms of sequential inputs and outputs
- Flipflops keep track of state of specification automata (state space of game)
- Strategy is relation between combinational inputs and combinational outputs: $R \subseteq I \times O$
- A circuit is a function $f: I \rightarrow O$

From BDD to Circuit



Relation Solving

Given: Strategy R: I x O

Find: function f: I $\rightarrow$ O such that

if f(i) = o then

(i,o) $\in$ R or $\neg\exists$o. (i,o) $\in$ R

Multiple possibilities lead to wildly different sizes in circuits

Strategy Minimization/Determinization

Challenges:

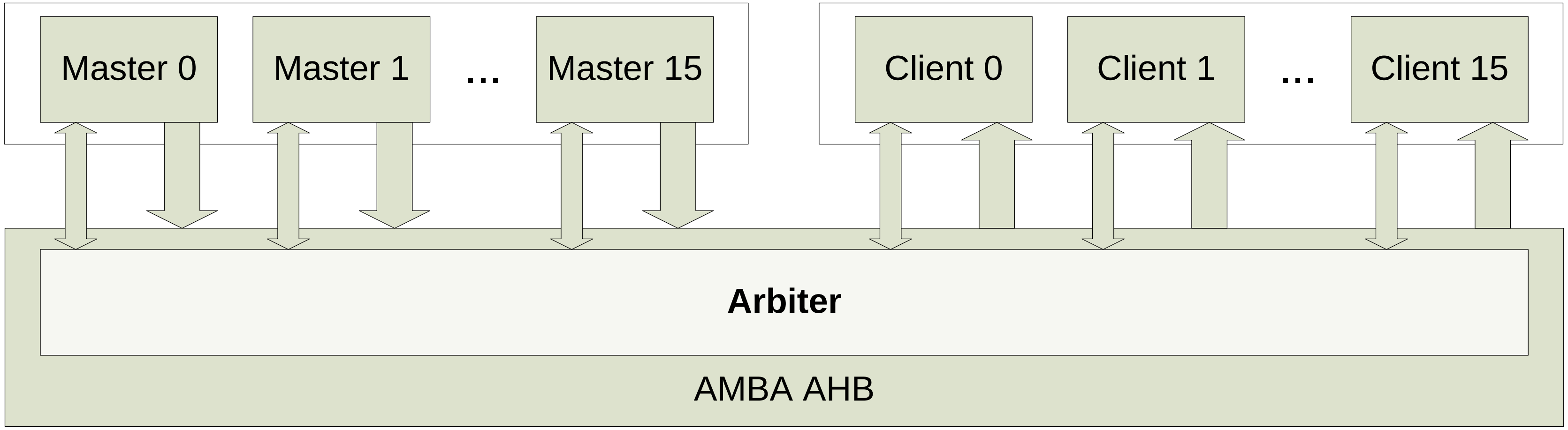
- Find simple function (small number of gates)
- Strategy relations are huge
  - Encoded symbolically (e.g. BDD)
  - ➢ Symbolic algorithms
- Efficiency

Different approaches based on BDD manipulation and/or learning.

(Synthesizing)
The AMBA Bus Protocol

AMBA Bus

- Industrial standard
- ARM's AMBA AHB bus
  - High performance on-chip bus
  - Data, address, and control signals (pipelined)
  - Arbiter part of bus (determines control signals)
  - Up to 16 masters and 16 clients



AMBA Bus

- Master initiates transfer. Signals:
  - HBUSREQi - Master i wants the bus
  - HLOCKi - Master i wants an uninterruptible access
  - HBURST - This access has length 1/4/incr
  - address & data lines
- The arbiter decides access
  - HGRANTi - Next transfer for master i
  - HMASTER[..] - Currently active master
  - HMASTLOCK - Current access is uninterruptible
- The clients synchronize the transfer
  - HREADY - Ready for next transfer
- Sequence for master
  - Ask; wait for grant; wait for hready; state transfer type & start transfer

AMBA Arbiter

- Specification
- 3 Assumptions, 12 Guarantees.
- Example:

"When a locked unspecified length burst starts, new access does not start until current master (i) releases bus by lowering HBUSREQi."

$\bigwedge_i$ G(HMASTLOCK $\wedge$ HBURST=INCR $\wedge$ HMASTER=i $\wedge$ START $\rightarrow$

X($\neg$START U $\neg$HBUSREQi))

Can completely be specified in GR(1)

Synthesis successful for $\leq$ 4 Masters

Formulation of Spec Matters

Assumption that master must eventually release locked bus

$\bigwedge$i: G((HMASTLOCK $\wedge$ HBURST=INCR $\wedge$ HMASTER=i) $\rightarrow$ F $\neg$HBUSREQ[i])

can also be written as

G((HMASTLOCK $\wedge$ HBURST=INCR) $\rightarrow$ F $\neg$HBUSREQ[HMASTER])

(We know that bus master does not change)

Now, instead of *n* automata with *n* fairness constraints, we have one!

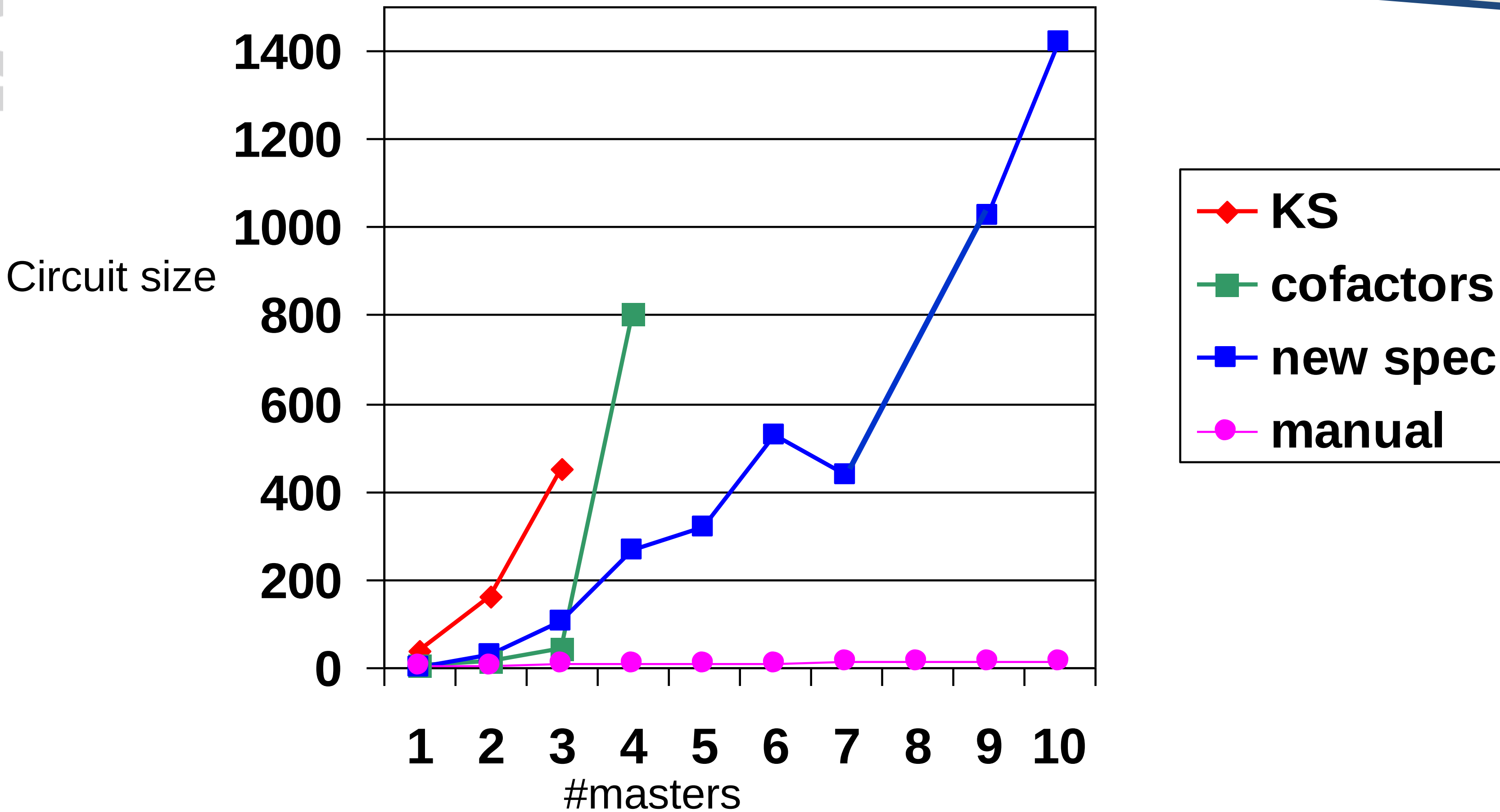
Spec can be simplified

Synthesis successful for $\leq$ 10 Masters

SCOS Secure & Correct Systems

New Spec

More recent results go up to 16 masters



AMBA Case Study: Results

- Expressibility of GR(1) is sufficient
- Deciding realizability is fast
- Specification is short and easy to understand
- Synthesis works!

Challenges: Specification

- Informal specs often ambiguous (AMBA spec is)
  - you also have this problem when writing Verilog code
- Is specifying really easier than coding?
  - GR(1) is a very special case, interesting things may not be (easily) expressible

Challenges: Size

- Circuits are **LARGE**, size depends on parameter (# masters)
  - Much bigger increase than necessary (see manual implementation)
- Smarter circuit generation needed
- Size depends strongly on formulation of specification

Bounded (Safraless) Approaches

Reactive Systems, More Formally

3 views on synthesis:

- synthesize a **strategy** for a game – depends on game graph
- synthesize a **circuit** – special form, good for bit-level symbolic reasoning
- synthesize a **labelled transition system** – this is close to the "automata" point of view

Of course, with right definitions, all 3 are equivalent

Labelled Transition Systems

A **labelled transition system** (**LTS**) S with inputs I and outputs O is a tuple (T, t_0, τ, o) with

- T a set of states
- t_0 an initial state
- $\tau: T \times \mathbb{B}^I \rightarrow T$ a transition function
- $o\colon T \rightarrow \mathbb{B}^O$ a (state) labelling function

Bounded (Safraless) Approaches

Avoid determinisation step by alternative approach:

1. reduce synthesis problem to emptiness check of **universal coBüchi tree automaton**

Universal Co-Büchi Tree Automaton (UCT)

Universal: takes all possible transitions at once, i.e., can be in multiple states at the same time

Co-Büchi: no state in F may be visited inf. often

Tree Automaton: reads trees instead of words

Space of executions of an LTS is a tree: branches labeled with inputs, nodes with outputs.

A UCT can directly read a system and accept/reject it.

SCOS Secure & Correct Systems

Bounded (Safraless) Approaches

Avoid determinisation step by alternative approach:

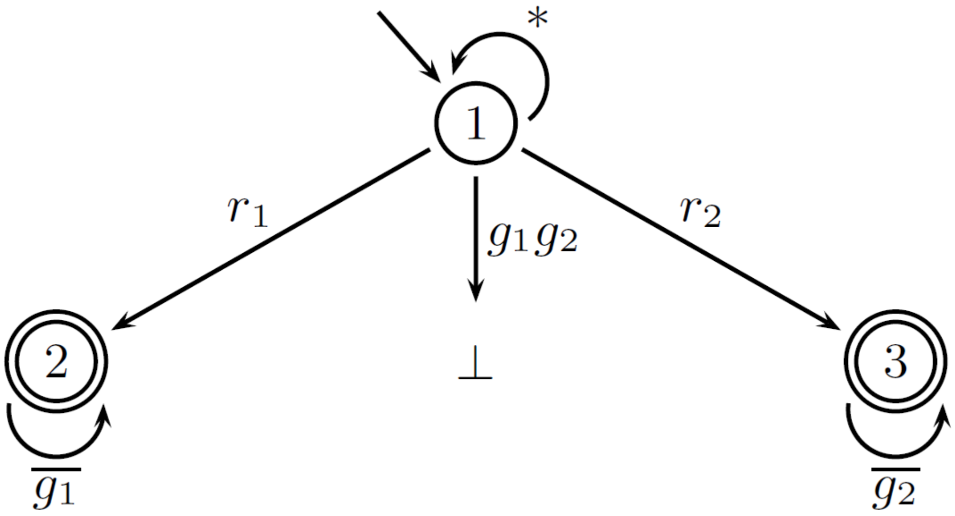
1. reduce synthesis problem to emptiness check of universal coBüchi tree automaton
2. reduce emptiness check to checking acceptance of trees/systems of **bounded size**.

For bounded size, problem can be encoded as decidable SMT constraints [ScheweFinkbeiner07] (alternative: [FiliotJinRaskin11])

Bounded Synthesis [ScheweFinkbeiner07]

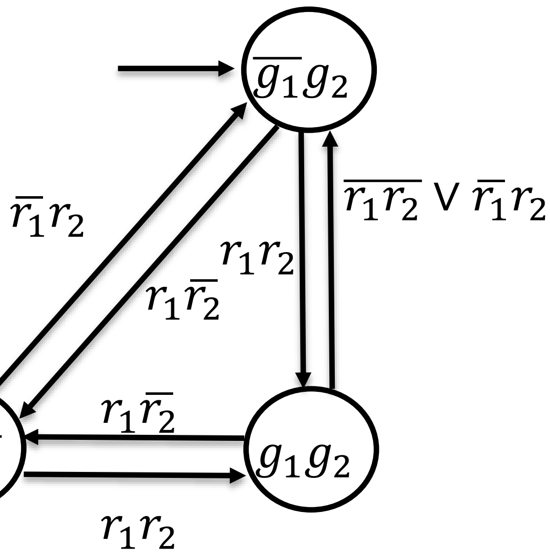
1. **Translate** LTL specification into UCT
2. **Generate SMT constraints** equivalent to realizability of spec (in system of size k)
3. **Solve constraints** for increasing k, obtain system (if one exists)

Bounded Synthesis: Construct UCT

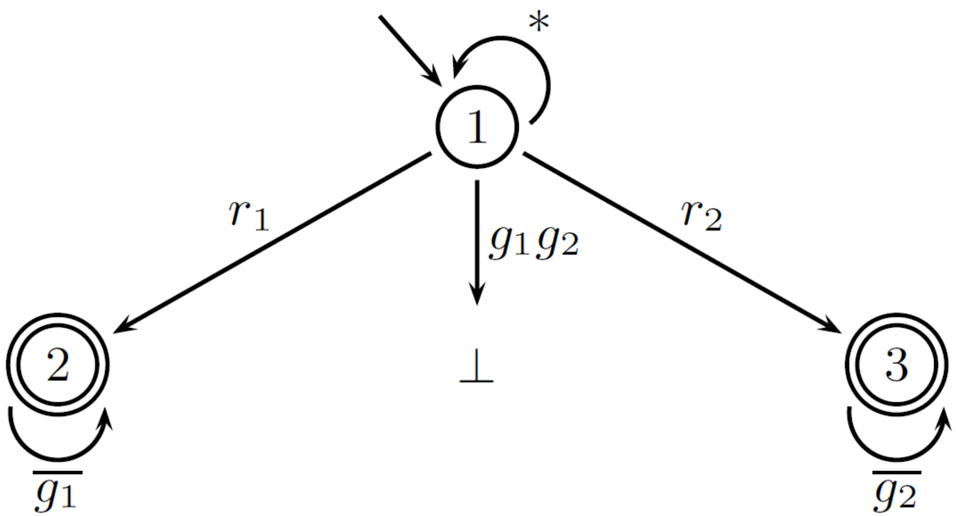
Specification	Automaton
$\bigwedge_{1 \leq i \leq 2} \mathbf{G}(r_i \rightarrow \mathbf{F} g_i)$ $\mathbf{G}\neg(g_1 \wedge g_2)$	

Bounded Synthesis: Acceptance of UCT

System	Automaton



(implicit self-loops in remaining cases)

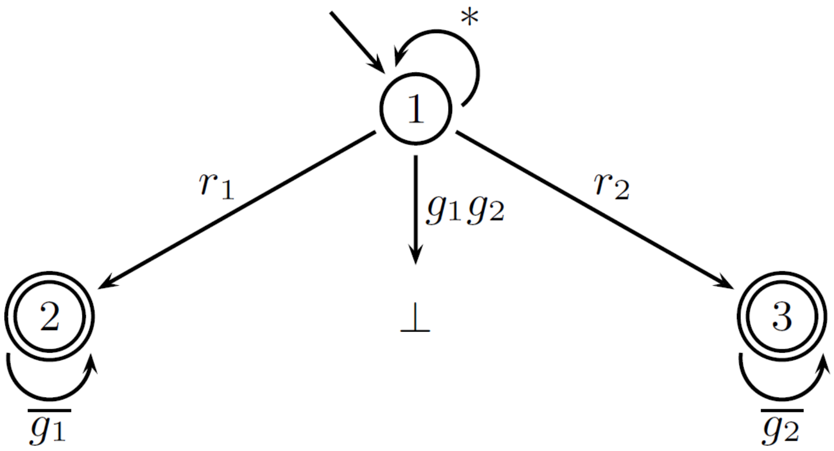


Bounded Synthesis: SMT Constraints

Idea: Annotate states of system with

- **predicates** $\lambda_q^{\mathbb{B}}: T \rightarrow \mathbb{B}$, representing reachable states of the automaton, i.e.,
  $\lambda_q^{\mathbb{B}}(t)$ is true if partial run of system that ends in t can lead to automaton state that includes q
- **counting functions** $\lambda_q^{\#}: T \rightarrow \mathbb{N}$, representing maximum number of visits to rejecting states in any partial run of the system that ends in t

Bounded Synthesis: Annotations

Annotation	Automaton
$\lambda_1^{\mathbb{B}}(t_0)$ $\lambda_1^{\#}(t_0) = 0$ $\forall I\ \forall t$: $\lambda_1^{\mathbb{B}}(t)$ $\rightarrow \lambda_1^{\mathbb{B}}\big(\tau(t,I)\big) \wedge \lambda_1^{\#}\big(\tau(t,I)\big) \geq \lambda_1^{\#}(t)$ $\forall I\ \forall t$: $\lambda_1^{\mathbb{B}}(t) \wedge r_1 \in I$ $\rightarrow \lambda_2^{\mathbb{B}}\big(\tau(t,I)\big) \wedge \lambda_2^{\#}\big(\tau(t,I)\big) > \lambda_1^{\#}(t)$ … …	

For given system S and UCT A, satisfying annotation exists iff A accepts S.

Bounded Synthesis: Solving

- For given system, such SMT constraints are **decidable and solved automatically**
- If we let transition function and output function of system be **unknown/uninterpreted**, we can use SMT solver for synthesis
- In this case, need to **restrict size** of system (s.t. quantifiers can be finitely instantiated)
- Very mature SMT solvers can be used out-of-the-box

Bounded Synthesis: Wrap-up

Bounded synthesis

- solves the synthesis problem by **smart encoding** into SMT constraints
- finds the **smallest implementation** (wrt. # states in LTS, or other metrics)
- **does not scale** very well (without additional optimizations)

How to make this work for bigger systems?

End of Synthesis, Part I: Basics

- Synthesis as a **Game**
- **General**: LTL Synthesis
- **Time-Efficient**: GR(1) Synthesis
- **Application**: AMBA Bus Protocol
- **Space-Efficient**: Bounded/Safraless Approaches

Bibliography

[PnueliRosner89] A. Pnueli, R. Rosner: On the Synthesis of a Reactive Module. POPL 89.

[SohailSomenzi09] S. Sohail, F. Somenzi: Safety First: A Two-Stage Algorithm for LTL Games. FMCAD 09.

[MorgensternSchneider10] A. Morgenstern, K. Schneider: Exploiting the Temporal Logic Hierarchy and the Non-Confluence Property for Efficient LTL Synthesis. GANDALF 10.

[ScheweFinkbeiner07] S. Schewe, B. Finkbeiner: Bounded Synthesis. ATVA 07.

[FiliotJinRaskin11] E. Filiot, N. Jin, J.F. Raskin: Antichains and compositional algorithms for LTL synthesis. FMSD 11.

[Ehlers12] R. Ehlers: Symbolic Bounded Synthesis. FMSD 12.

[VardiWolper86] M. Vardi, P. Wolper: Automata-theoretic techniques for modal logics of programs. JCSS 86.

[KestenPitermanPnueli05] Y. Kesten, N. Piterman, A. Pnueli: Bridging the Gap between Fair Simulation and Trace Inclusion. I&C 05.

[PitermanPnueliSa’ar06] N. Piterman, A. Pnueli, Y. Sa'ar: Synthesis of Reactive(1) Designs. VMCAI 06.