Reactive Synthesis

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

VTSA 2013

Nancy, France

24.09.2013

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

Synthesis, Part II: Advanced Topics

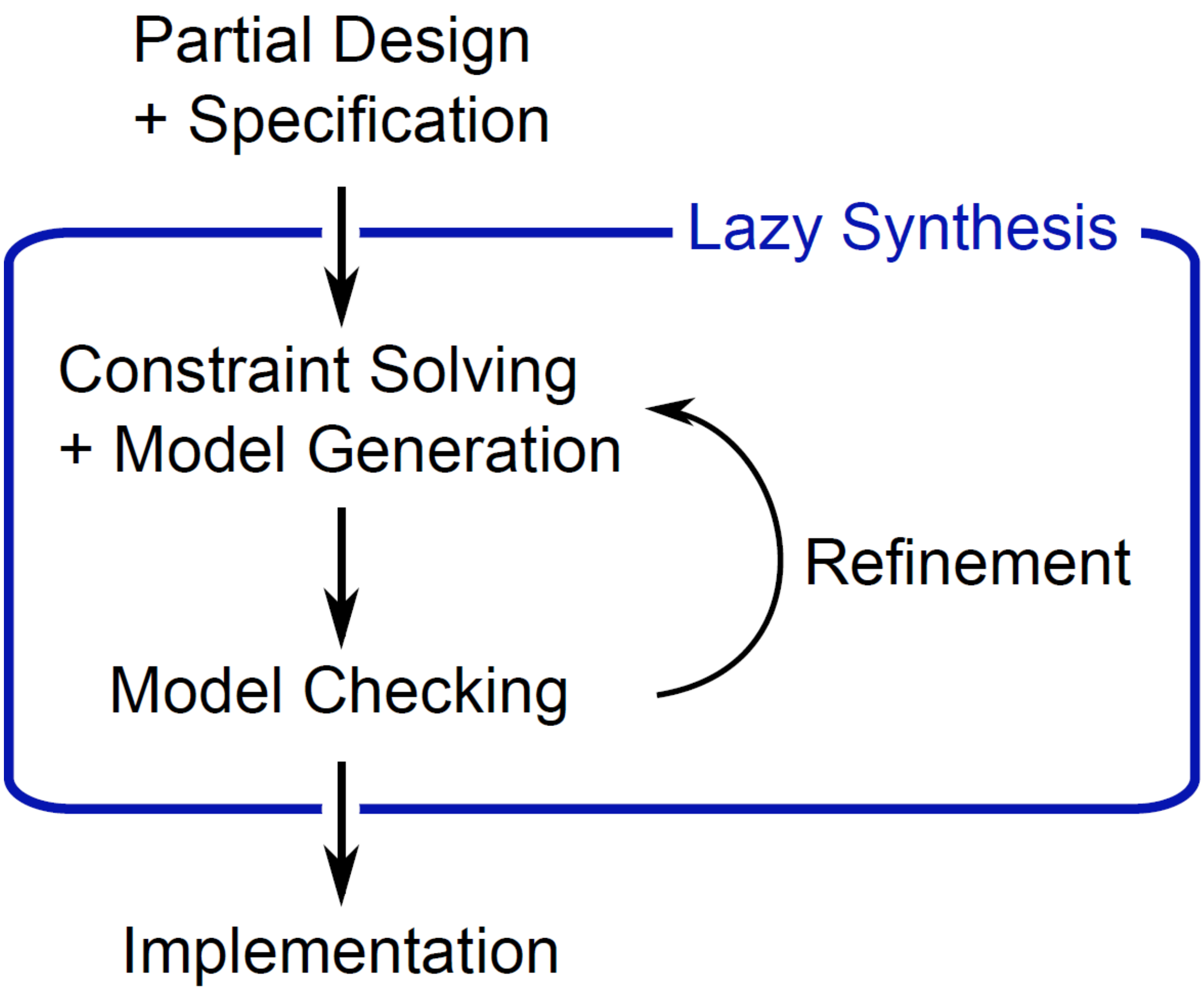
- **Lazy** Synthesis
- **Distributed** Synthesis
- **Parameterized** Synthesis
- **Quantitative** Specifications
  - **Robustness**

Lazy Synthesis

Lazy Synthesis [VMCAI12]

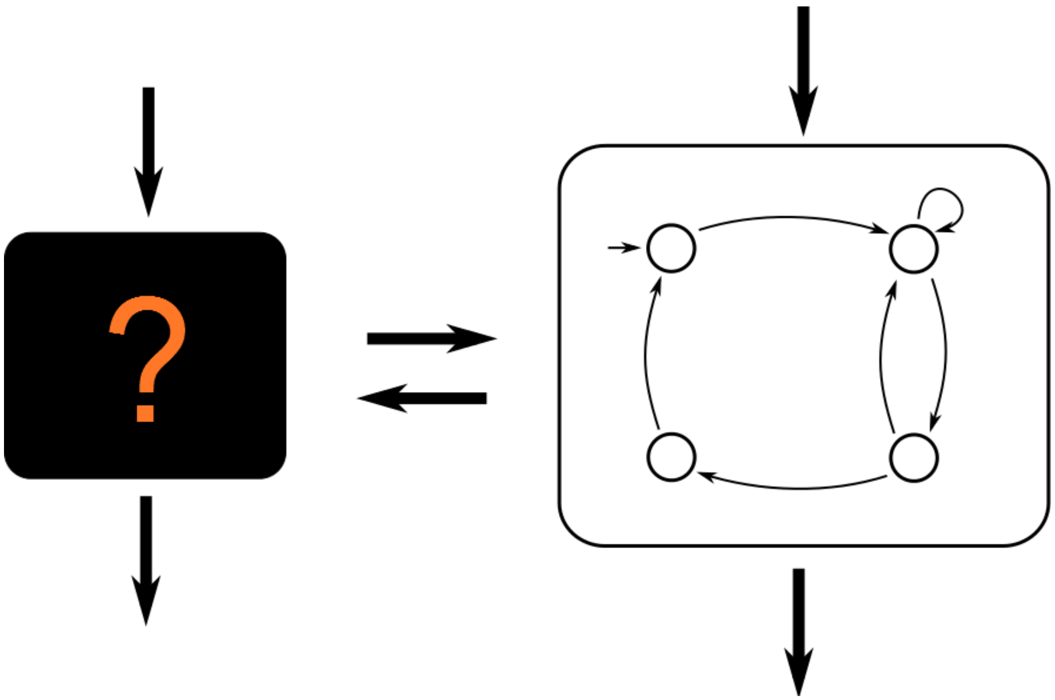
- Based on SMT-based Bounded Synthesis
- **Idea**: instead of full translation to SMT, use **lazy encoding** in abstraction refinement approach
- Integrates **model checking** approach to test candidate models and obtain counterexamples

Lazy Synthesis: Overview



SCOS Secure & Correct Systems

Partial Design



- Part of system already implemented
- Other part to be synthesized
- Interface of processes given

Lazy Synthesis: Overview

Outer Loop:

- Search for implementation of size n, increment n if unrealizability is proved

Synthesis Loop:

For a given bound n:

1. SOLVE: **check satisfiability** of constraints, obtain candidate implementation
2. CHECK: **model check** candidate and white-box with monitor automata
3. REFINE: if errors are reachable, **construct constraints** excluding error paths

Lazy Synthesis: Solve Phase

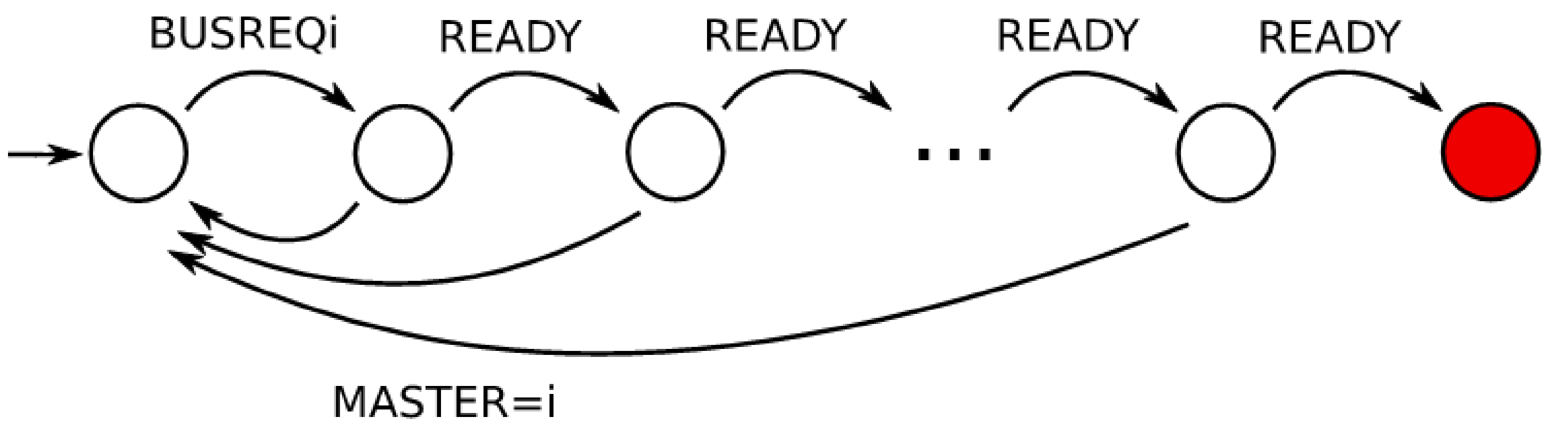
- **Transition relation** represented as function $trans\colon \mathbb{B}^{|I|} \times \mathbb{N} \to \mathbb{N}$,
- **Outputs** as functions of type $\mathbb{N} \to \mathbb{B}$
- **Initial constraints**: size constraint, initial state
- **More constraints** are added in subsequent calls
- Check satisfiability of constraints and **obtain model**

Lazy Synthesis: Check Phase

Translate assumptions & guarantees to **safety** automata

Assumption: $\mathbf{GF}\ READY$

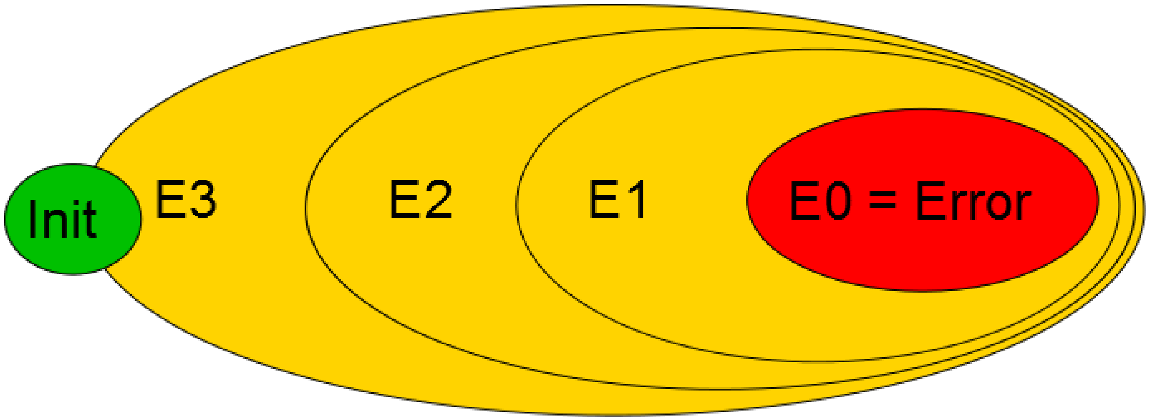
Guarantee: $\mathbf{G}\big(BUSREQi \rightarrow \mathbf{F}\ (MASTER = i)\big)$



Restriction to safety depends on size bound

Lazy Synthesis: Check Phase

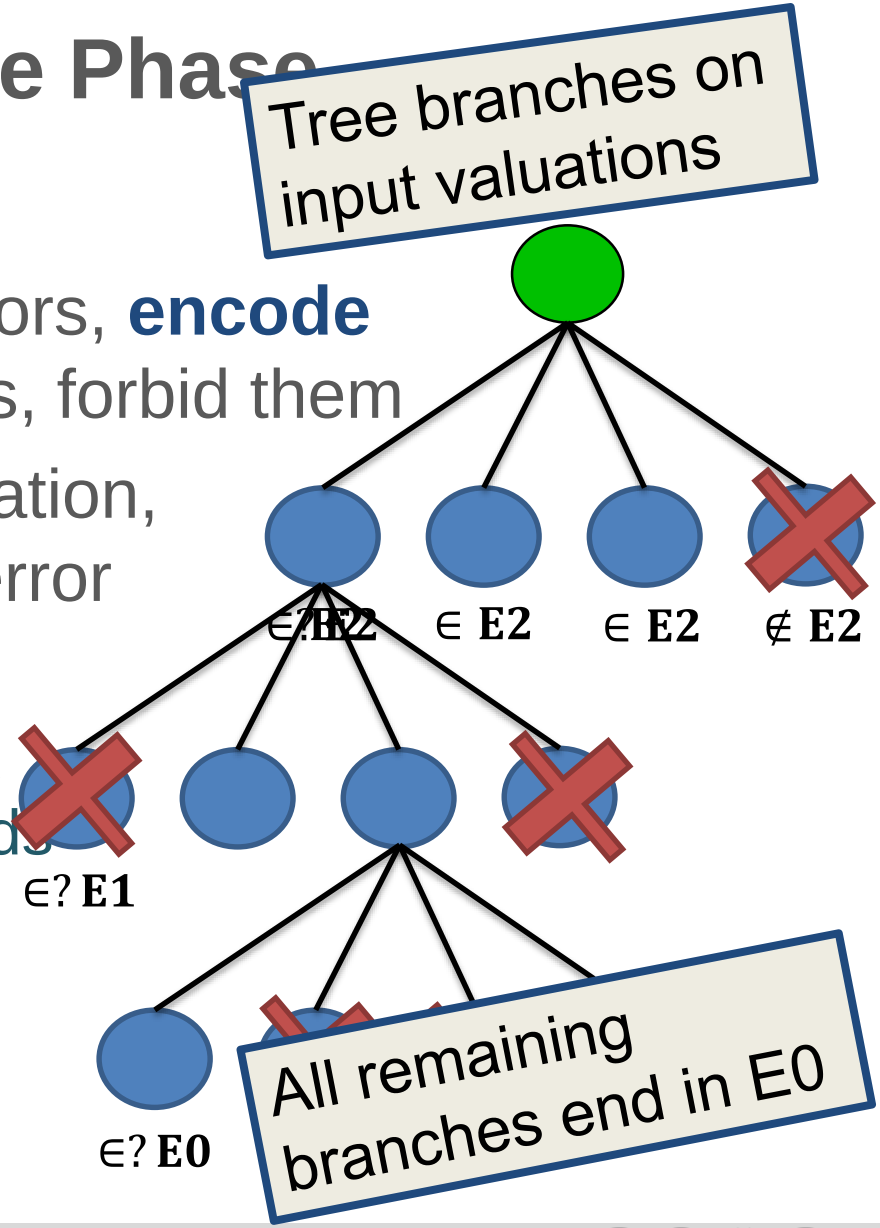
- **Model-check** candidate + white-box + automata

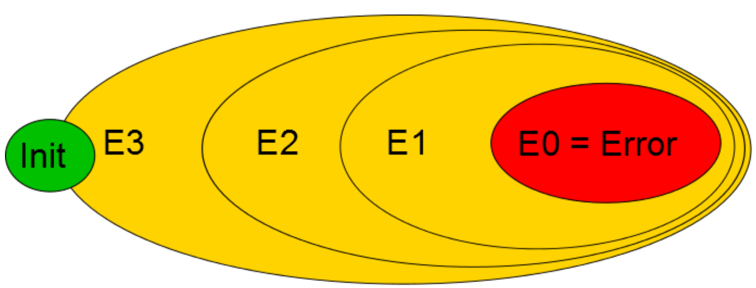


- If **errors** found, call Refine phase,
  otherwise candidate model **satisfies** full spec

Lazy Synthesis: Refine Phase

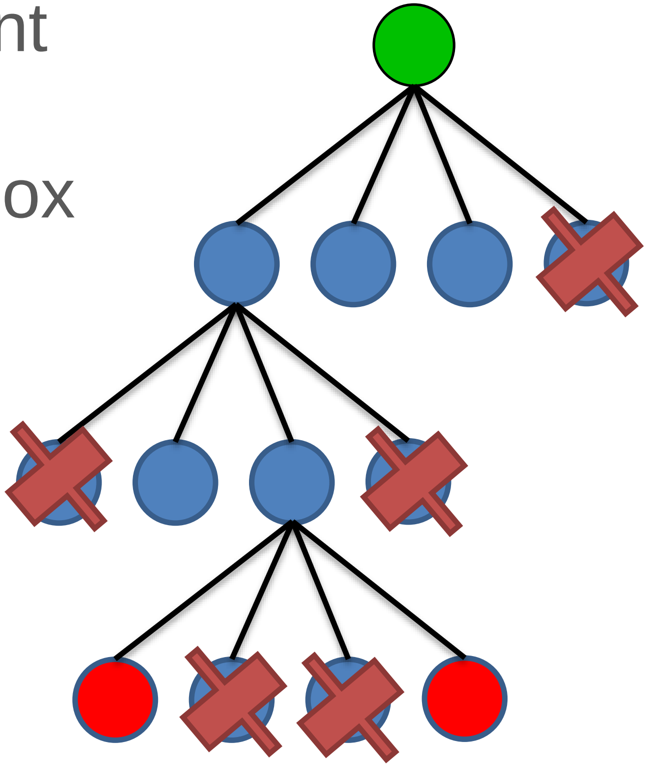
- If model checker finds errors, **encode** them into SMT constraints, forbid them
- In BDD-based implementation, we can obtain tree of all error paths of minimum length
  - this tree can be translated into a constraint that forbids all minimal errors





Lazy Synthesis: Refine Phase

- Error tree translated to constraint that **forbids all error paths**, restricted to interface of black-box
- For every path, the constraint expresses that **at least one** output needs to be different



Lazy Synthesis: Overview

Outer Loop:

- Search for implementation of size n, increment n if unrealizability is proved

Synthesis Loop:

For a given bound n:

1. SOLVE: **check satisfiability** of constraints, obtain candidate implementation
2. CHECK: **model check** candidate and white-box with monitor automata
3. REFINE: if errors are reachable, **construct constraints** excluding error paths

Lazy Synthesis: AMBA Case Study

Reconsider AMBA case study, with partial implementation for deterministic parts:

"The arbiter indicates which bus master is currently the highest priority [...] by asserting the appropriate GRANTi signal. When the current transfer completes, as indicated by READY HIGH, then [...] the arbiter will change the MASTER[3:0] signals to indicate the bus master number."

[AMBA Specification (Rev 2.0), ARM Ltd.]

Lazy Synthesis: AMBA Case Study

Other statements **translated to LTL**:

"The arbitration mechanism is used to ensure that only one master has access to the bus at any one time."

$$\forall i \neq j\colon \mathbf{G}\big(READY \rightarrow \neg(GRANTi \wedge GRANTj)\big)$$

Some statements modeled with **auxiliary variables**:

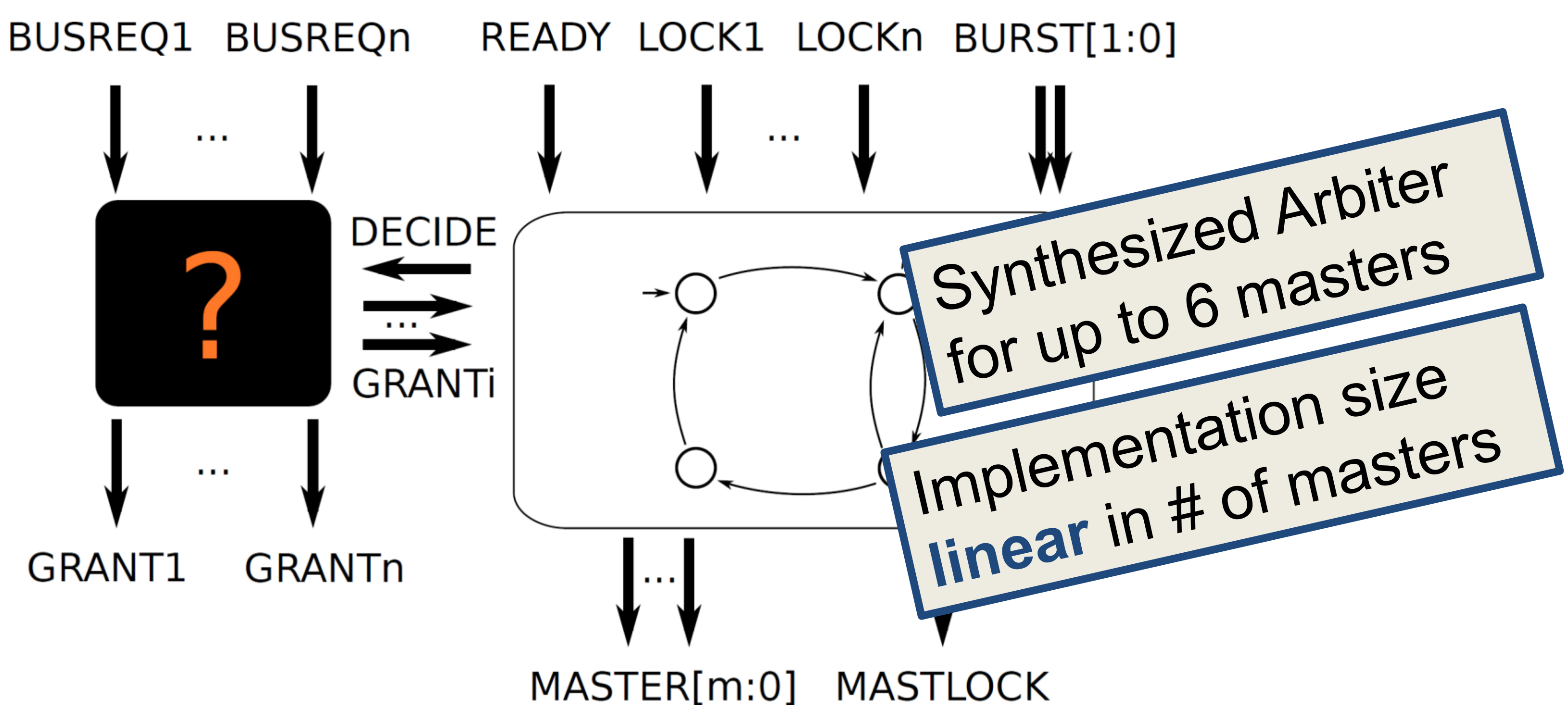
"Normally the arbiter will only grant a different bus master when a burst is completing."

$$\forall i\colon \mathbf{G}\big(\neg DECIDE \rightarrow (GRANTi \leftrightarrow \mathbf{X}\, GRANTi)\big)$$

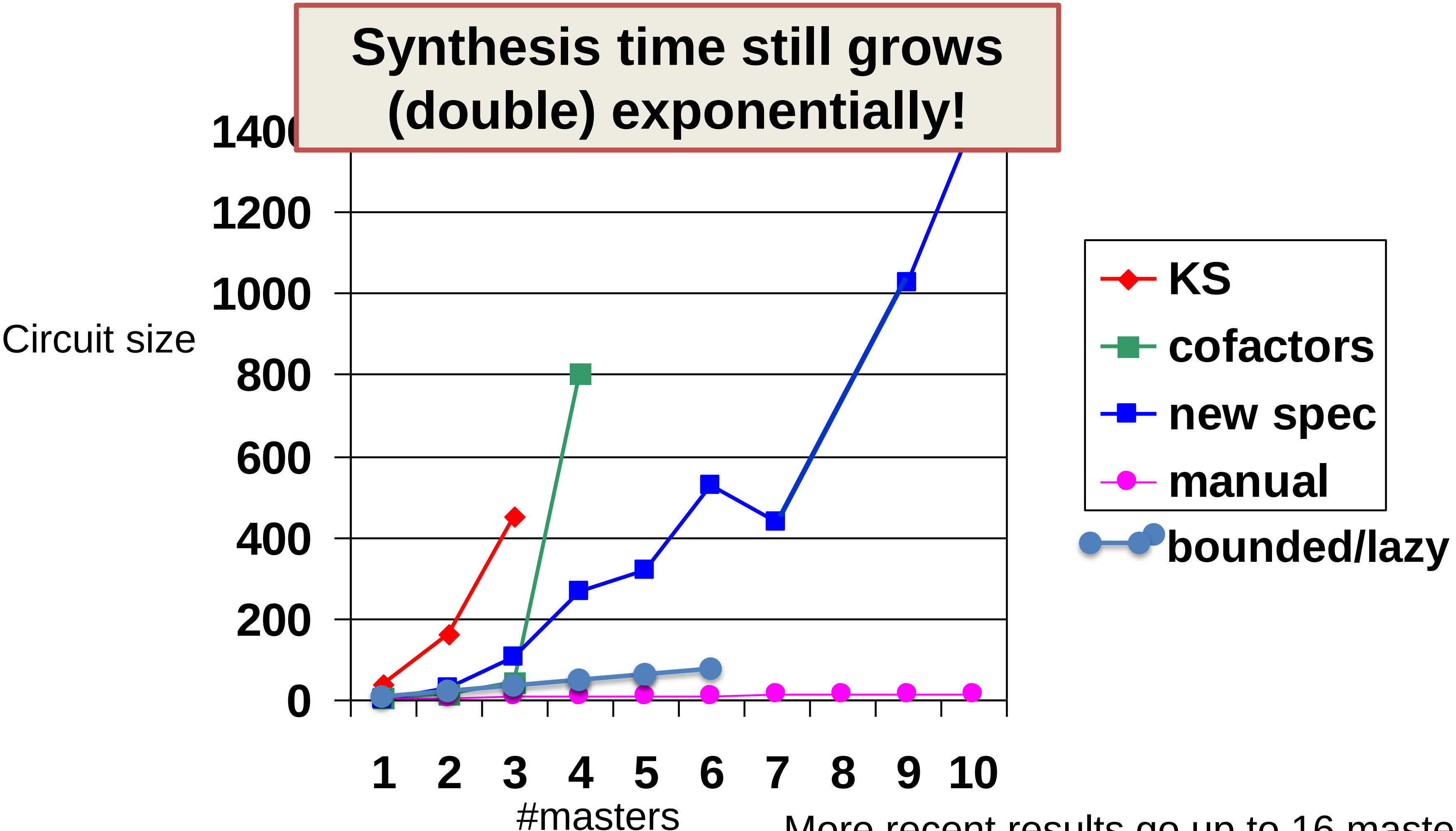
($DECIDE$ defined s.t. it is high when a burst completes)

Lazy Synthesis: AMBA Case Study

- AMBA with partial implementation for deterministic parts
- crucial part synthesized: arbiter



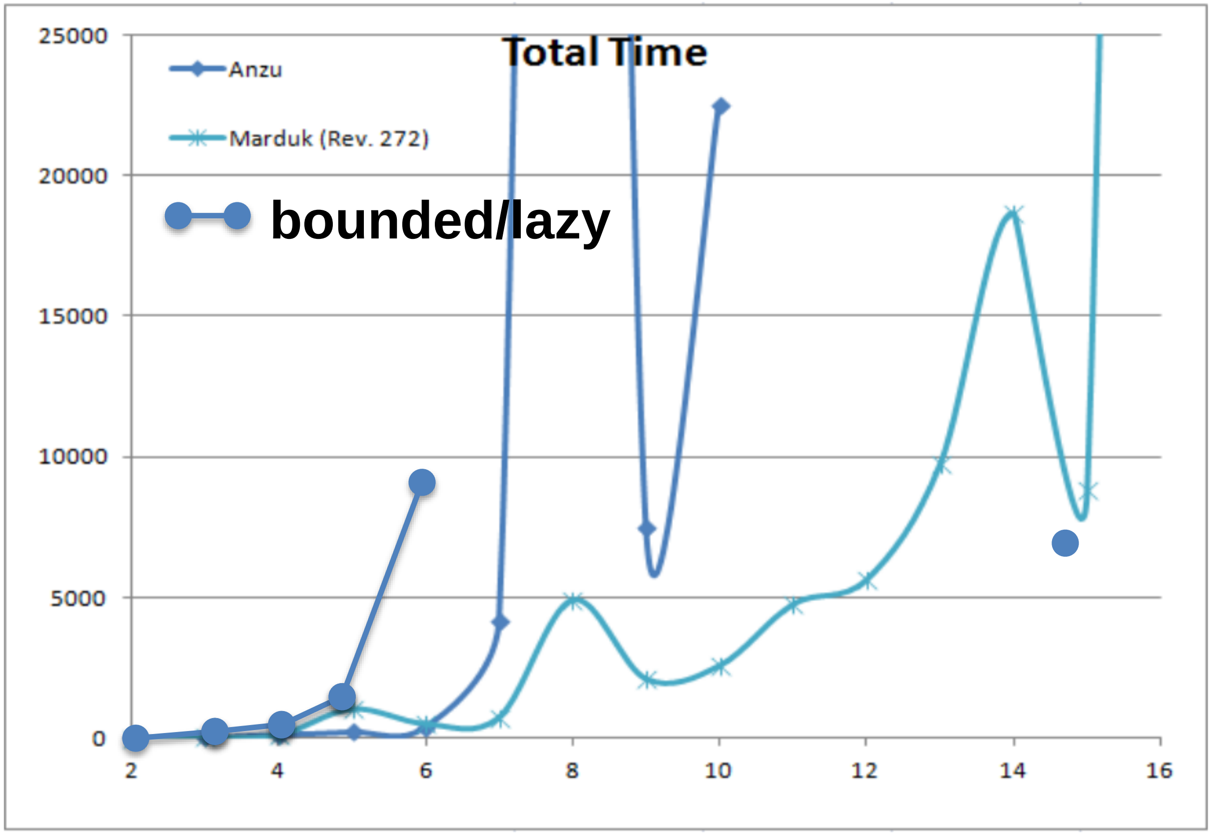
AMBA: Bounded size of implementations



More recent results go up to 16 masters

AMBA: Bounded size of implementations

Synthesis time still grows (double) exponentially!



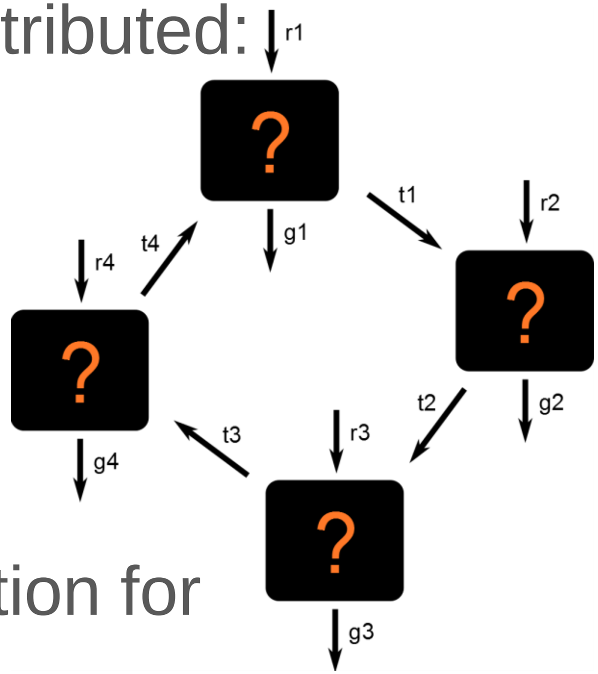
Lazy Synthesis: Challenges

- SMT solving incremental, but Model Checking restarted every time
  - deep integration of incremental model checking?
- interface and safety abstraction currently given by hand
  - automatically minimize interface?
  - automatic safety abstraction, or use liveness model checker?
- Parallelize?
- Extend to distributed case?

Distributed Synthesis

Why Distributed Synthesis?

- Many interesting systems are distributed:
  - multi-threaded programs
  - multi-core processors
  - communication protocols
  - distributed control
  - …
- Both a prerequisite and a motivation for parameterized synthesis



Distributed Synthesis

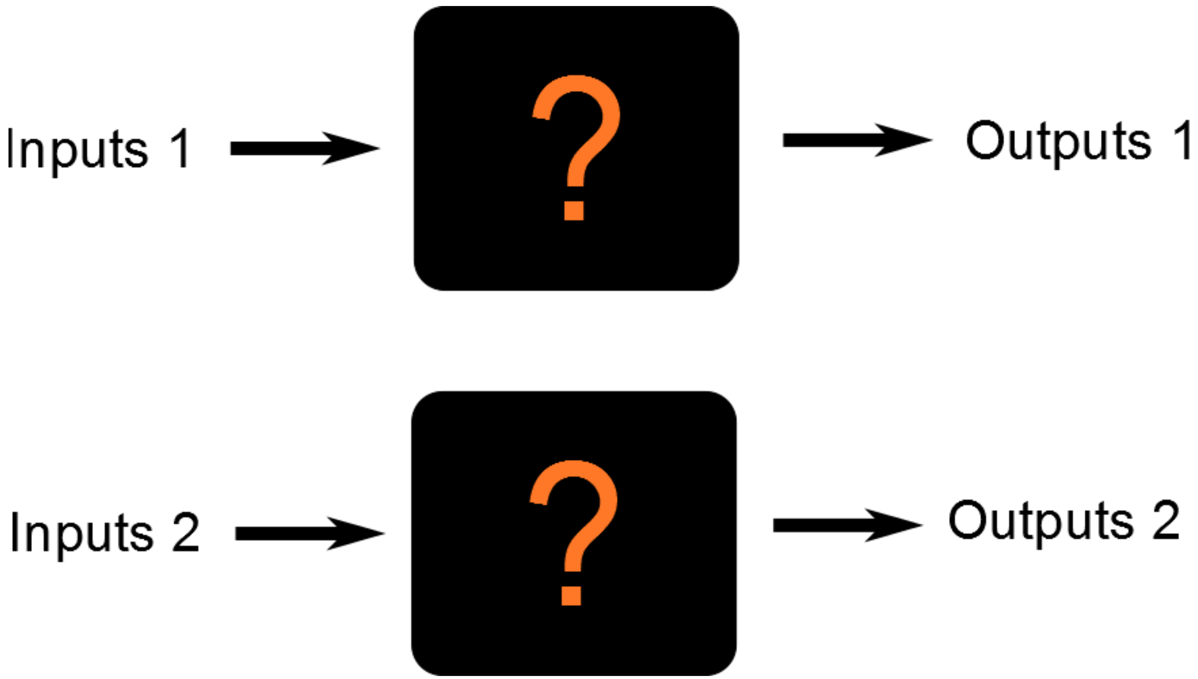
- Several processes, each decides about subset of outputs
- **Easy case**: all processes have full information; this reduces to standard synthesis problem
- **How so?**
  - Every process has all inputs, but only subset of outputs
  - In worst case, synthesize full system for all processes and throw away unnecessary outputs

Partial Information

- **Hard case**: every process only has limited information about environment (and other processes)
- **Very hard**, but decidable, for some architectures like pipelines

Partial Information

- **Undecidable** if there is an **information fork** [PnueliRosner90,FinkbeinerSchewe05]



Partial Information: Bounded Synthesis

Semi-decision procedure possible, e.g. based on bounded synthesis.

Model distributed systems by **projection functions** from a global state t to local state $d_i(t)$ of component i

Partial information then expressed by constraints of the form

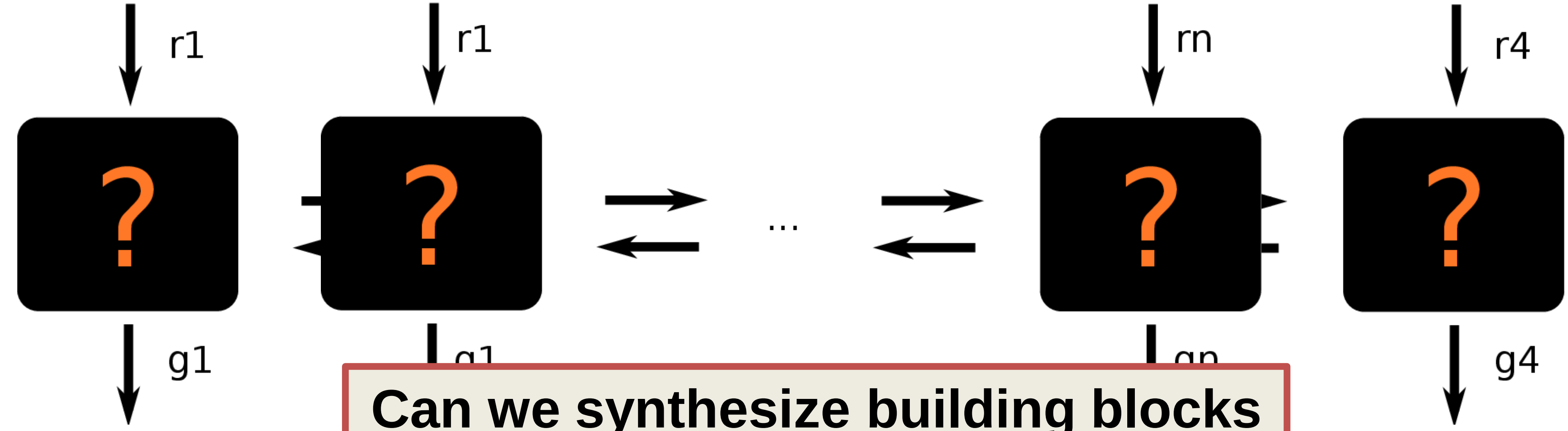
$$d_i(t) = d_i(t') \wedge (I \cap I_i) = (I' \cap I_i) \rightarrow d_i\big(\tau(t, I)\big) = d_i\big(\tau(t', I')\big)$$

(for every process i)

Parameterized Synthesis

Parameterized Synthesis
[TACAS12,VMCAI13]

- Many specifications are parametric in nature
- AMBA, communication protocols, etc.



Can we synthesize building blocks for arbitrary size systems?

Parameterized Synthesis

Building blocks:

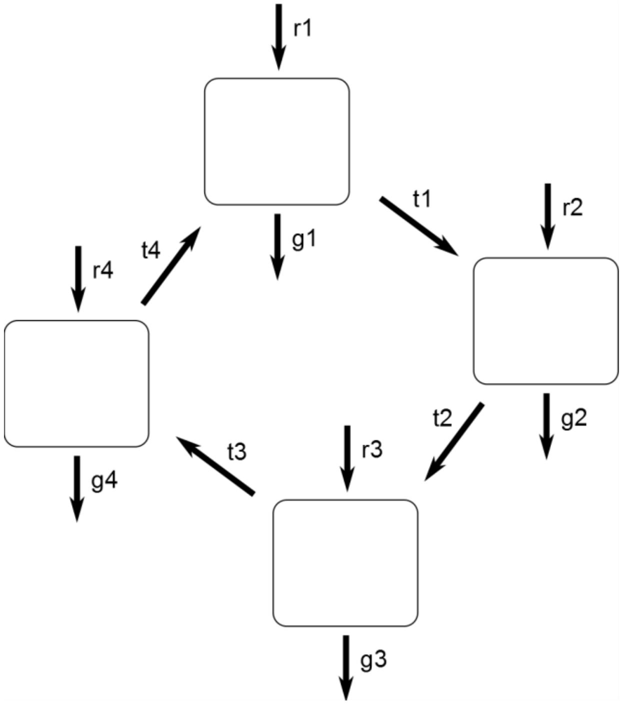
- Distributed synthesis
  - of uniform processes
- Decidability results for parameterized verification
  - particularly, cutoffs

Parameterized Verification

Parameterized verification is **decidable** for certain systems

Asynchronous System:
No global clock, a subset of processes are allowed to make a move in every global step (decided by external scheduler).

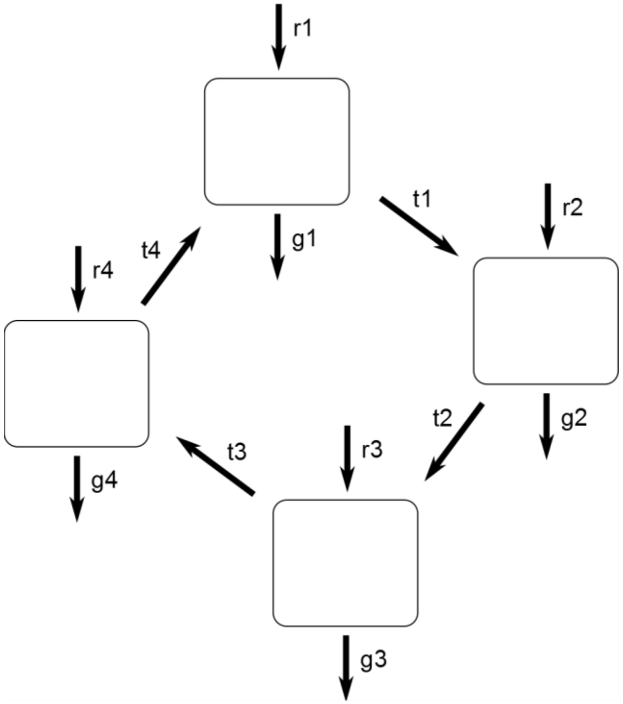
Token Ring:
Processes only communicate by passing single (value-less) token in ring architecture. Always exactly one process is scheduled, except for token passing steps.



Parameterized Verification

Parameterized verification is **decidable** for certain systems

Theorem [EmersonNamjoshi95]:
In **token rings** with fair token passing, a given process implementation satisfies parameterized specification φ in LTL\X **iff it satisfies φ in a ring of small size**:



Corollary: For **parameterized synthesis** in token rings, it is sufficient to synthesize a process implementation satisfying φ in a ring of size 2 – 5.

(Un)Decidability

Does decidability of parameterized verification make **synthesis decidable**?

No, since even for two uniform processes in a token ring, distributed synthesis is **undecidable**.

A **reduction result** from Clarke et al. [CTTV04] shows that parameterized synthesis for formulas $\forall i\colon \varphi(i)$ reduces to synthesis of one process, which is decidable.

Parameterized Synthesis: Procedure

1. **Use cutoff** to reduce parameterized synthesis problem to distributed synthesis problem
2. **Modified encoding** (from bounded synthesis) of realizability of specification with
   - uniform processes
   - in a token ring architecture
   - with fair scheduling and fair token-passing
3. **Solve** problem with SMT solver (for increasing bounds)

Modified Encoding

Bounded synthesis encoding with following **extensions**:

- synthesis of **uniform processes**:
  - add constraints that specify equivalence of local transitions
  - use same output labels for all processes
- **token-passing systems**:
  - add constraints ensuring correct token passing of exactly one token in the ring
- **fairness** of scheduling and token passing:
  - added directly to LTL specification

(First) Experiments

Can synthesize distributed arbiter in token ring of 4 processes with spec

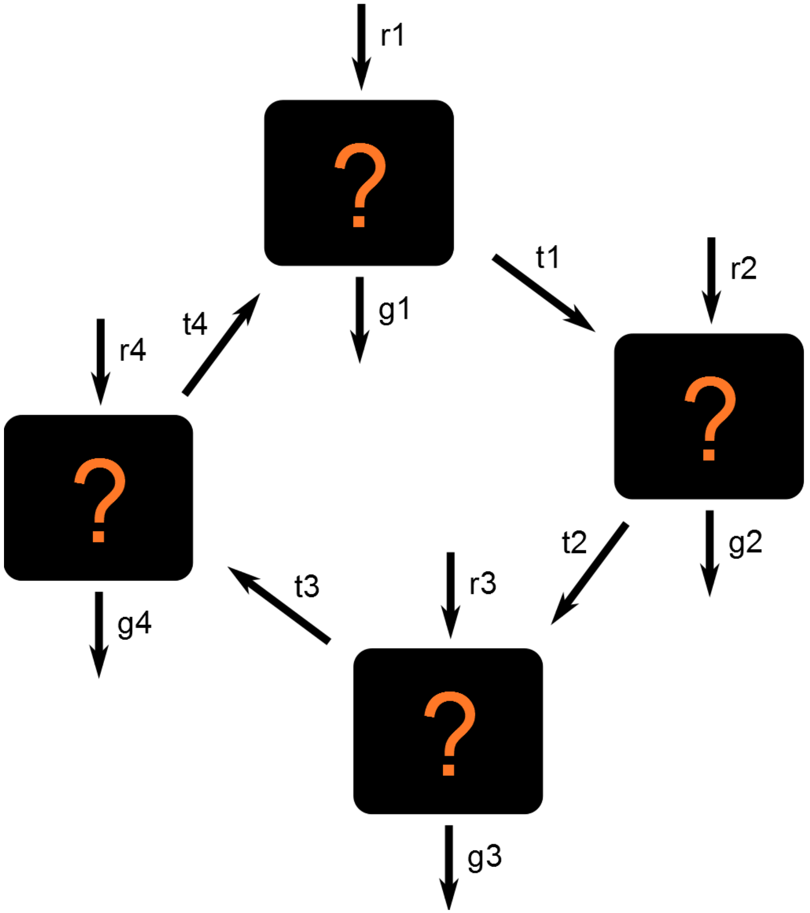
$\forall i: G(r_i \rightarrow F g_i)$
$\forall i \neq j: \neg(g_i \wedge g_j)$

This takes Z3 about 10 sec.

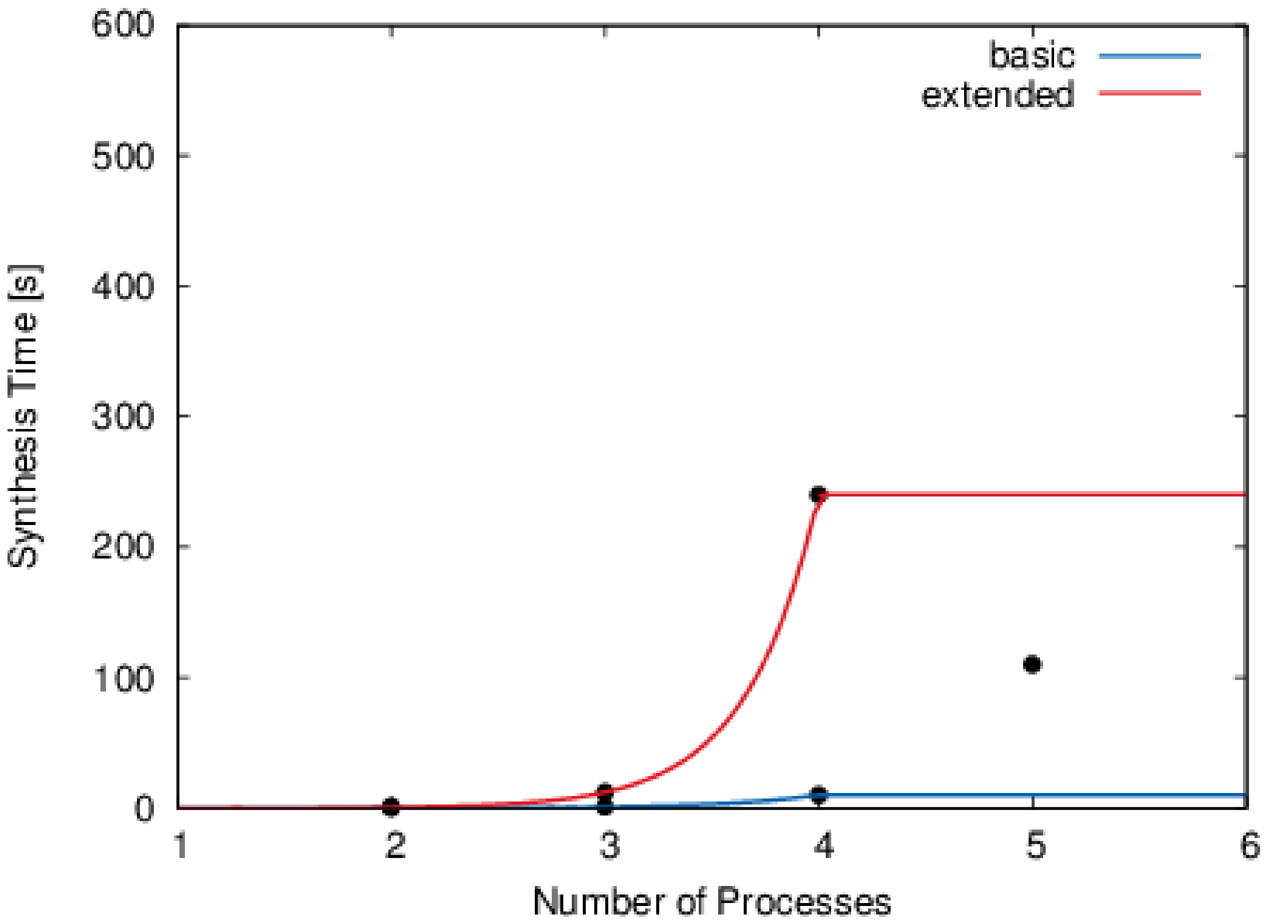
But: problem gets **hard** very fast.
For extended spec with

$\forall i: \neg g_i U r_i \wedge G(g_i \rightarrow \neg g_i U r_i)$,

needs about 240 sec.



Benefits of Parameterized Synthesis



Parameterized Synthesis: Optimizations [VMCAI13]

Modular Synthesis:

- Instead of one cutoff for whole system, use different cutoffs for conjuncts
  $\forall i: G(r_i \rightarrow F g_i)$ ⇒ cutoff 2
  $\forall i \neq j: G\neg(g_i \wedge g_j)$ ⇒ cutoff 4
  (before: one cutoff for whole formula)
- Encoded separately (with same uninterpreted functions), conjoined for solving
- large parts of specifications have small cutoffs (properties are local to the process)

Parameterized Synthesis: Optimizations

Table 2: Effect of general optimizations on solving time (in seconds). Timeout is 7200s.

	simple4	full2	full3	full4	pnueli2	pnueli3	pnueli4	pnueli5	pnueli6
bottom up	3	24	934	t/o	23	6737	t/o	t/o	t/o
strengthening	1	6	81	638	2	13	90	620	6375
modular	1	4	8	13	2	4	11	49	262

Size of SMT queries:

full4: 6MB → 0.6MB

pnueli4: 21MB → 4MB

Parameterized Synthesis: Optimizations

More optimizations:

- **local synthesis** for local properties $\forall i: \varphi(i)$
- **optimized annotations** (counters for SCCs)
- **bottom-up encoding** of global transition relation
- hard-coding token possession

Speed-up: $> 10^3$ on some examples

Parameterized Synthesis: Challenges

- Make approach applicable to more architectures
  - lots of parameterized verification results can potentially be lifted to synthesis
- Find out what is needed to synthesize industrial case studies, like AMBA, in parameterized way
  - theoretical extensions (synchronous, architecture)
  - additional optimizations

Quantitative Specifications

Specification Example: Arbiter

- Input: r0, r1
- Output: g0, g1

- Specification (in LTL):
- $G(r0 \rightarrow F\ g0)$
- $G(r1 \rightarrow F\ g1)$
- $G\ \neg(g0 \wedge g1)$

Any nasty arbiters that satisfy the spec?

Specification Example: Arbiter

- Input: r0, r1
- Output: g0, g1

- Specification (in LTL):
- $G(r0 \rightarrow F\ g0)$
- $G(r1 \rightarrow F\ g1)$
- $G\ \neg(g0 \wedge g1)$

- Unnecessary grants!
- Arbitrary time between request and grant!

A Different Arbiter (Safety)

- Input: r0, r1
- Output: g0, g1

Specification (in LTL):
Guarantees:

- $G(r0 \rightarrow g0)$
- $G(r1 \rightarrow g1)$
- $G\ \neg(g0 \wedge g1)$

Assumption:

- $G\ \neg(r0 \wedge r1)$

Any nasty arbiters that satisfy the spec?

A Different Arbiter (Safety)

- Input: r0, r1
- Output: g0, g1

Specification (in LTL):
Guarantees:

- $G(r0 \rightarrow g0)$
- $G(r1 \rightarrow g1)$
- $G \neg(g0 \wedge g1)$

Assumption:

- $G \neg(r0 \wedge r1)$

- What if two requests come simultaneously?
- Spec does not guarantee robustness!

Specifications

- Claim: traditional specs have their drawbacks
- Goal: introduce new specification language to state properties like
  - ASAP
  - As little as possible
  - Robustness
  - …

Boolean View – Black & White

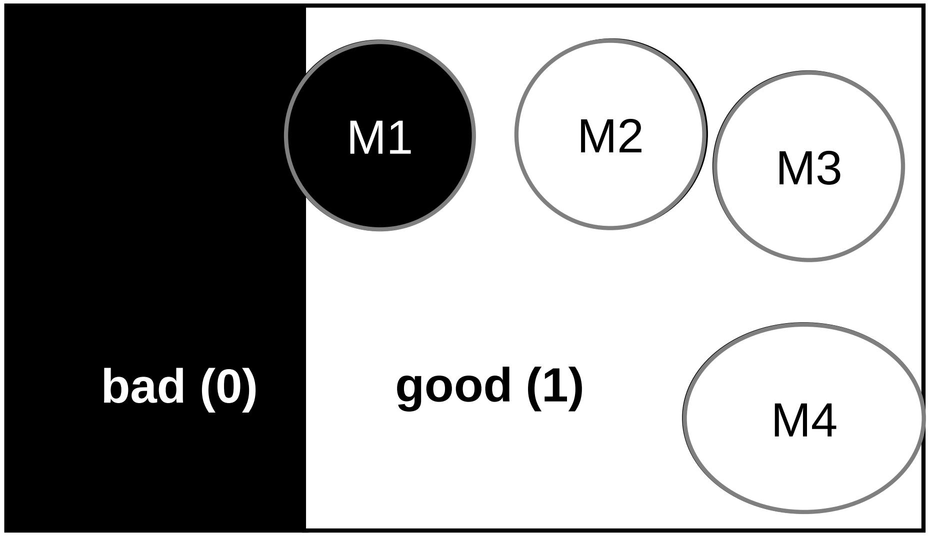
Language is function mapping words to {0,1}

System is a set of words

A good system has only good words

But: some systems are *better* than others! Now what?

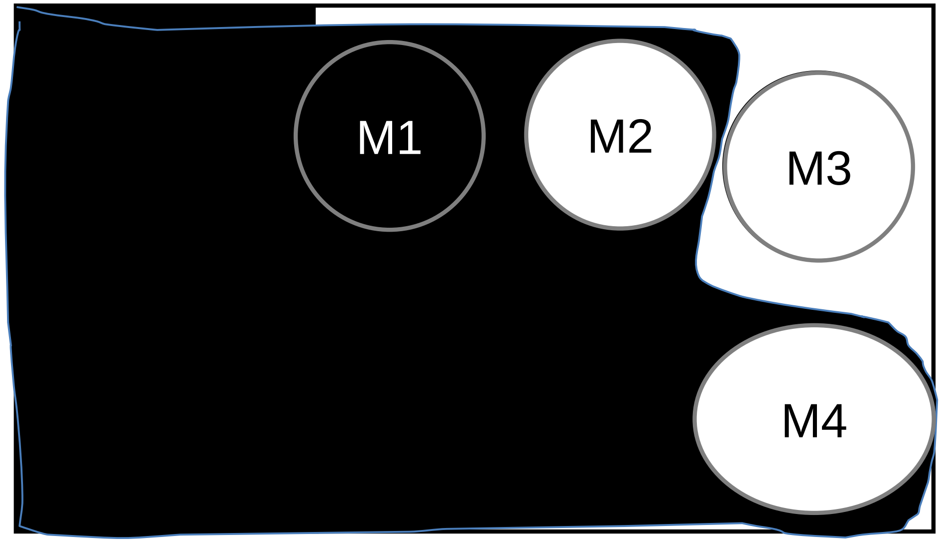
Set of all words



Boolean View – Black & White

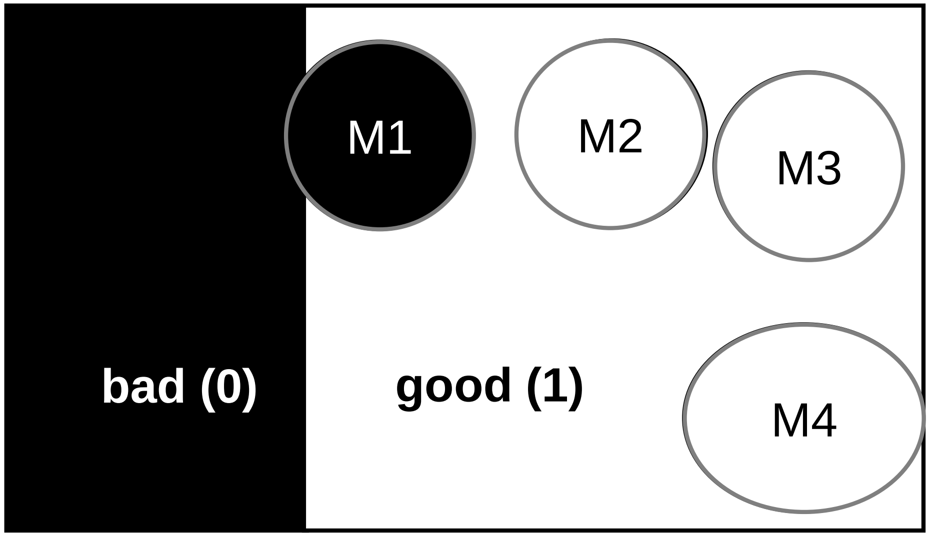
Updating the spec may be hard

- Properties may be hard to find
- You may loose abstraction
- Spec may become long & unreadable



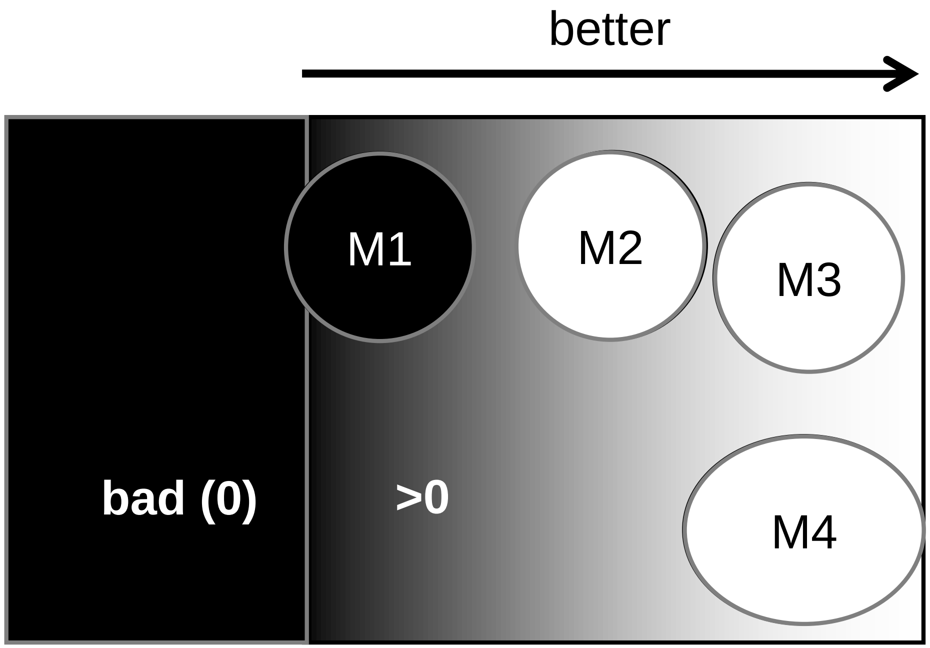
Revisit Basic Assumption

Language is function mapping words to {0,1}



Quantitative view – Grey scale

Language is function mapping words to $\mathbb{R}$



Questions

Design Questions:

- How do we assign a value to a word?
- Given L: $\Sigma^\omega \to \mathbb{R}$, what is the value of a system?

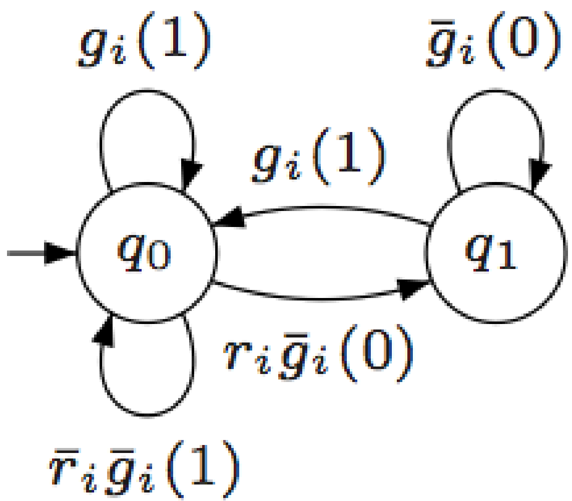
Technical Questions

- How do we verify that the value of a system is OK?
- How do we synthesize an optimal system?

Value of a Word

- Idea: reward good events
- Use deterministic automata with weights on edges
  - $A: \Sigma^\omega \to N^\omega$
- Summarize weights of a word. Options:
  - $L_A(w) = \min(A(w))$
  - $L_A(w) = \max(A(w))$
  - $L_A(w) = \mathrm{meanvalue}(A(w))$
- Mean value gives you mean payoff automata

Example: Quick Grants



$w_1 = (rg \quad \bar{r}\bar{g} \quad rg)^\omega$	$(111)^\omega$	$value(w_1) = 1$
$w_2 = (r\bar{g} \quad \bar{r}\bar{g} \quad \bar{r}g)^\omega$	$(001)^\omega$	$value(w_2) = \frac{1}{3}$
$w_3 = (r\bar{g} \quad r\bar{g} \quad r\bar{g})^\omega$	$(000)^\omega$	$value(w_3) = 0$

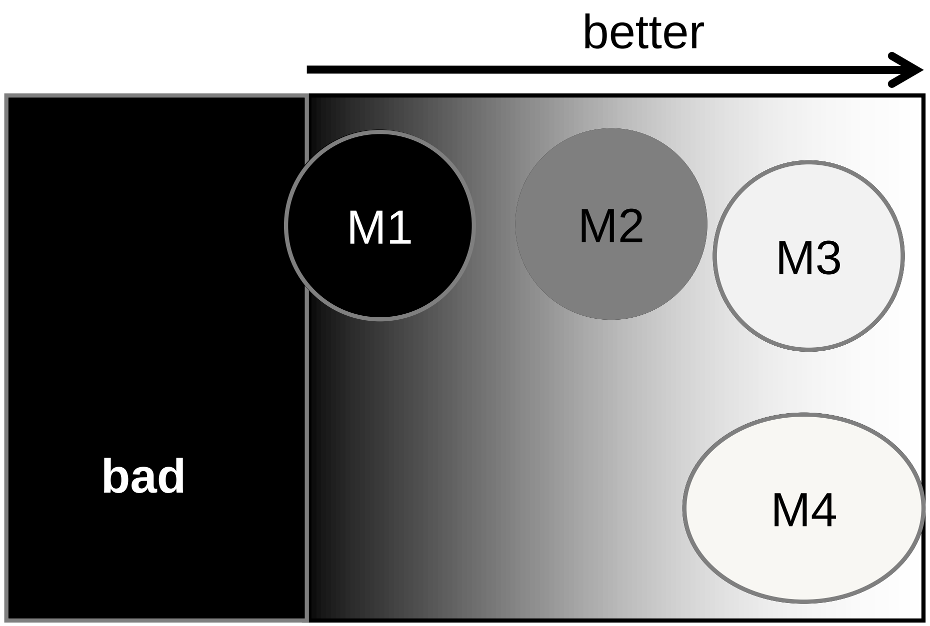
Value determined by mean-payoff automaton

Value of a System

What is the value of a system?

- The value of the worst word
- The value of an average word
- The value of the best word

Worst-case analysis is natural extension of Boolean case



Questions

Design Questions:

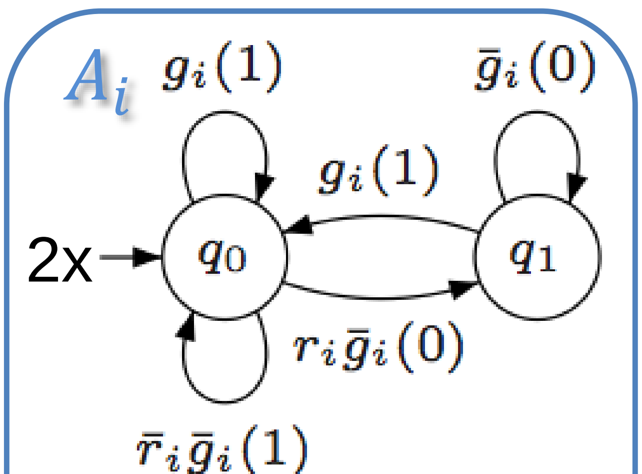
- How do we assign a value to a word?
- Given L: $\Sigma^\omega \rightarrow \mathbf{R}$, what is the value of a system?

Technical Questions

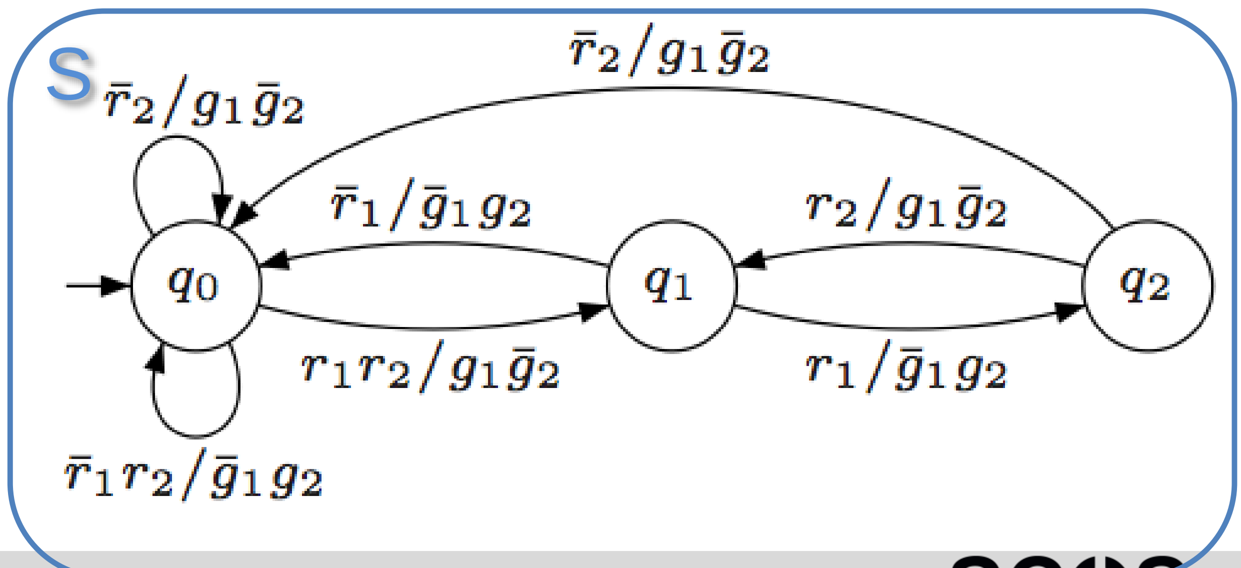
- How do we verify that the value of a system is OK?
- How do we synthesize an optimal system?

Compute System Value

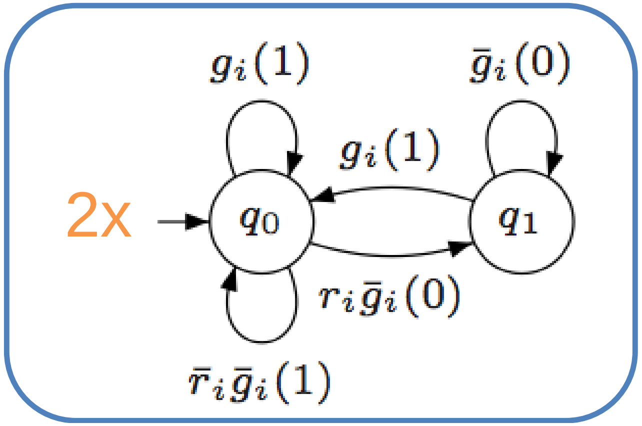
- Given a mean-payoff automaton A and a reactive system S, compute value(S)

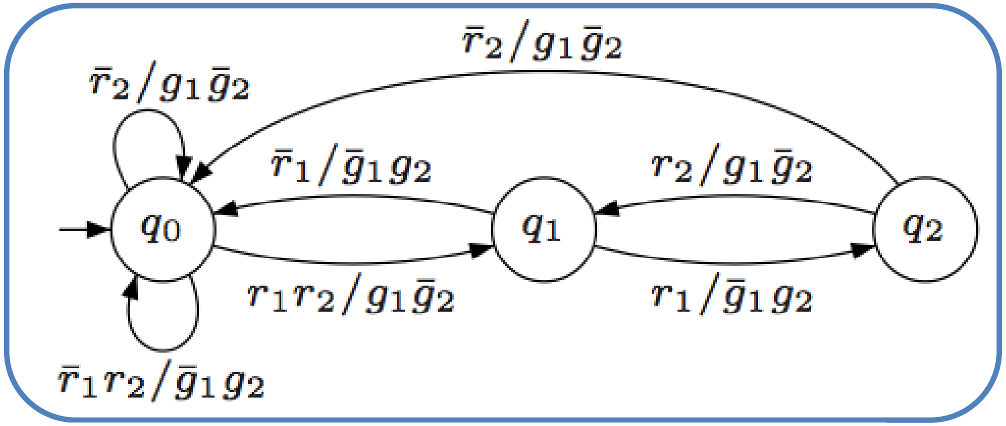


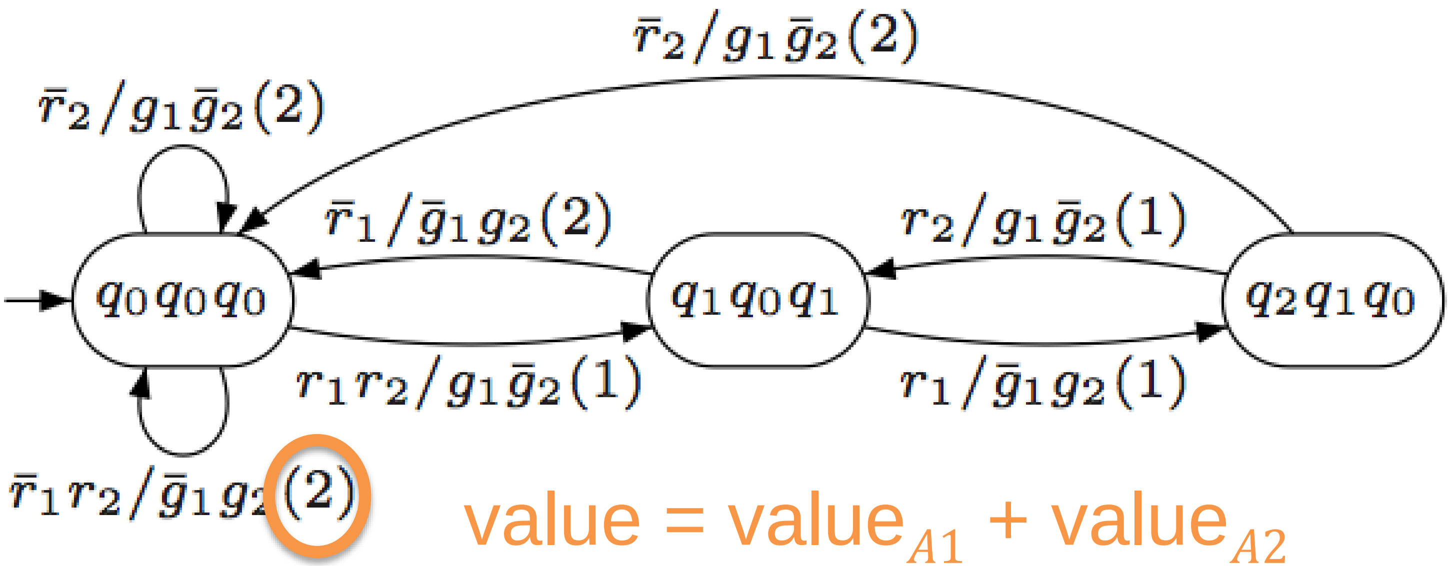
value = value$_{A1}$ + value$_{A2}$



Specification × System

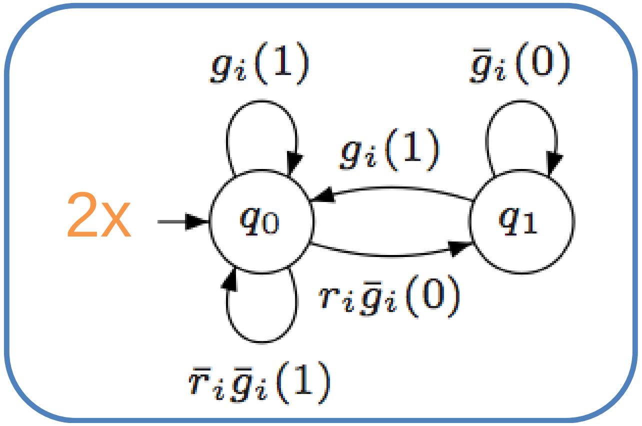


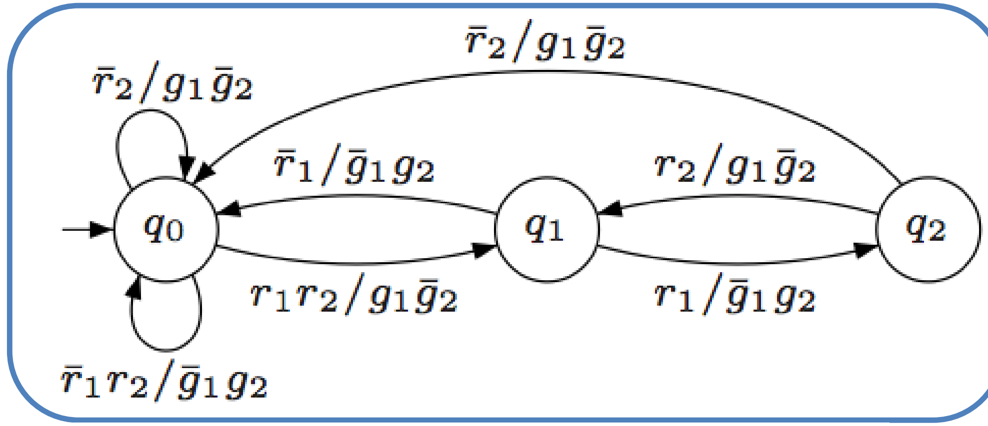


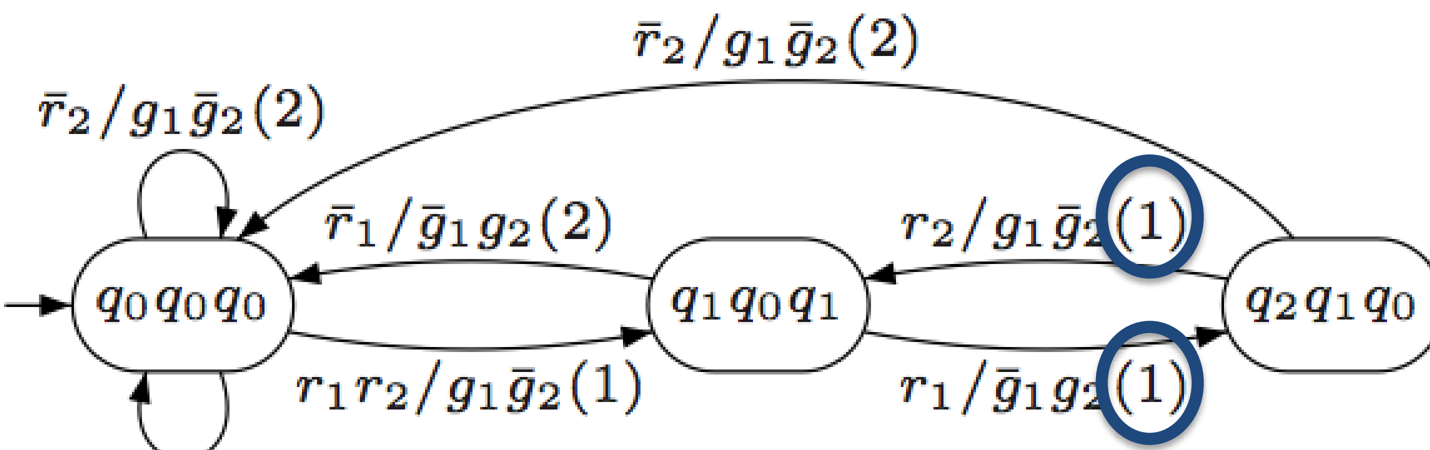


value = value$_{A1}$ + value$_{A2}$

Specification × System







Worst mean-payoff = payoff in minimum mean-payoff cycle

How to Construct Optimal System?

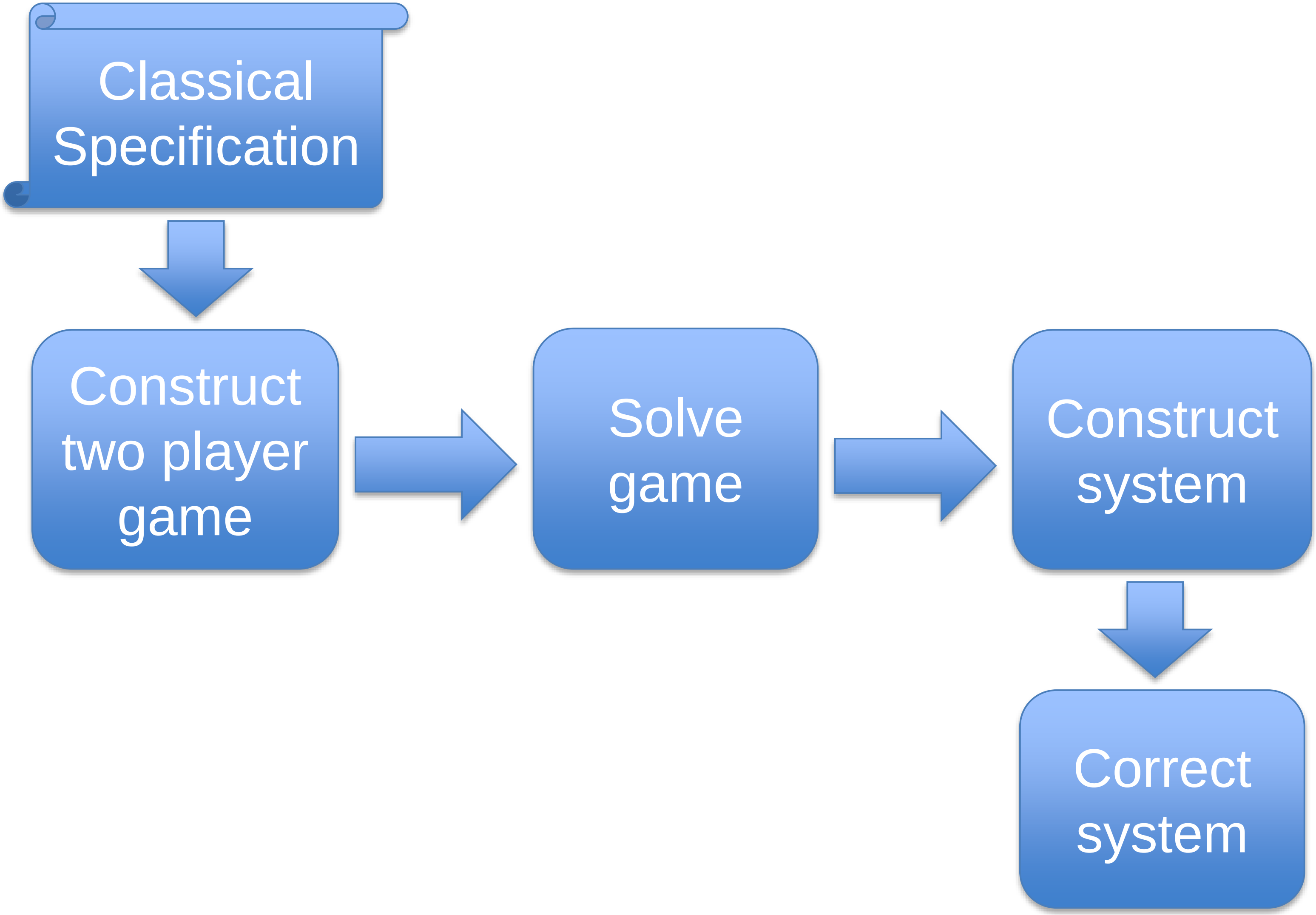
Given

- A classical specification φ
- A quantitative specification ψ

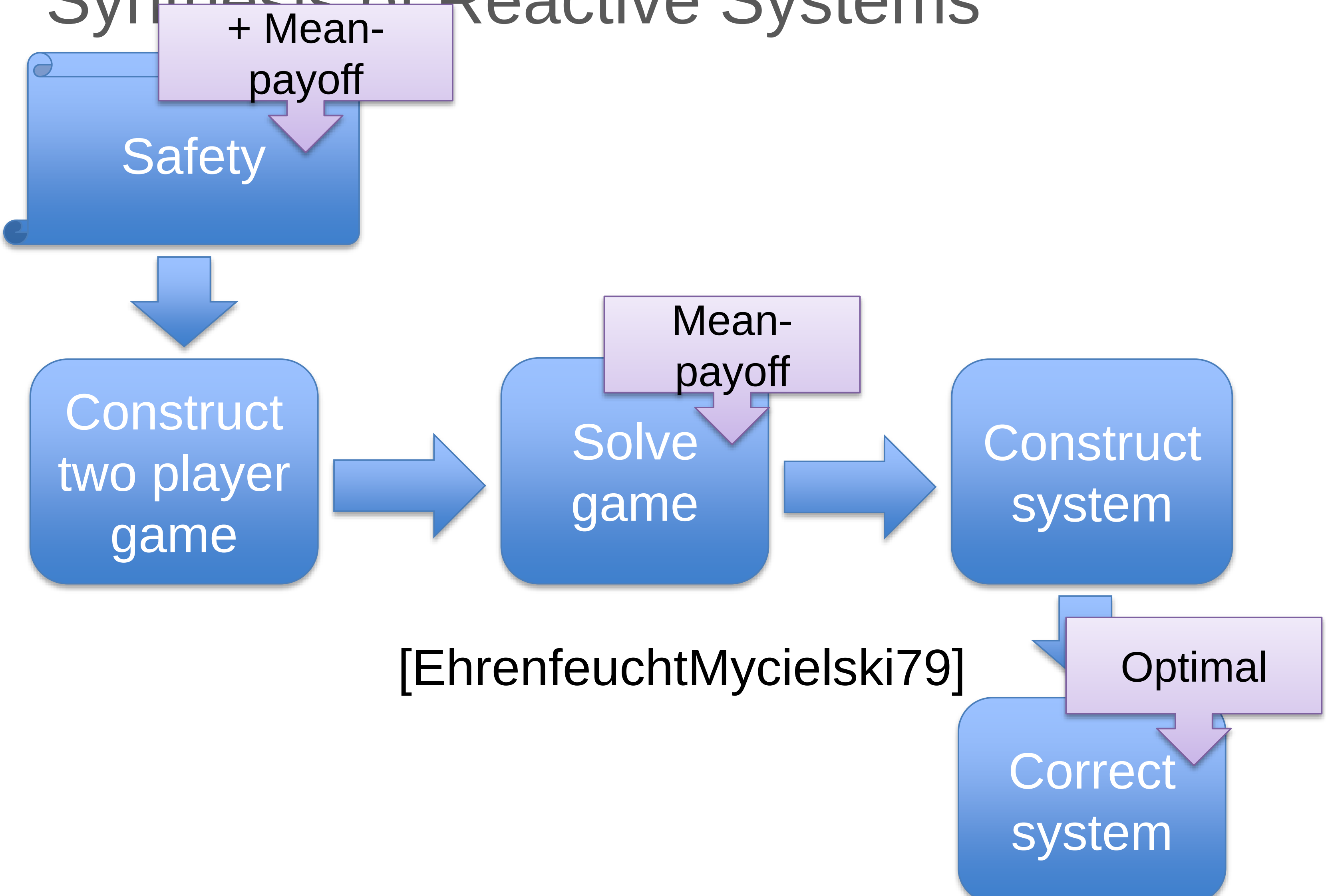
Construct a reactive system S that

- satisfies φ and
- optimizes ψ.

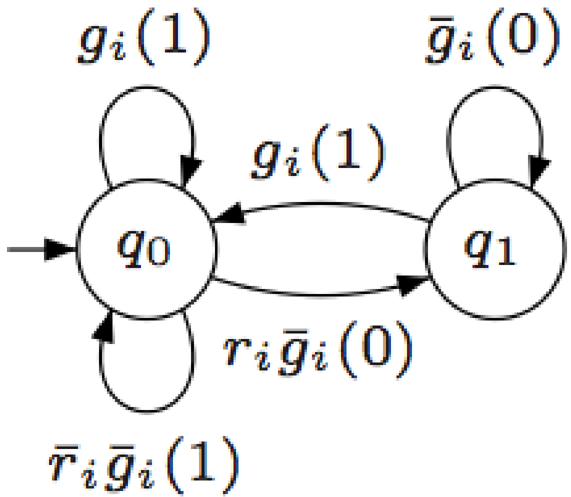
Synthesis of Reactive Systems



Synthesis of Reactive Systems



Example: Quick Grants



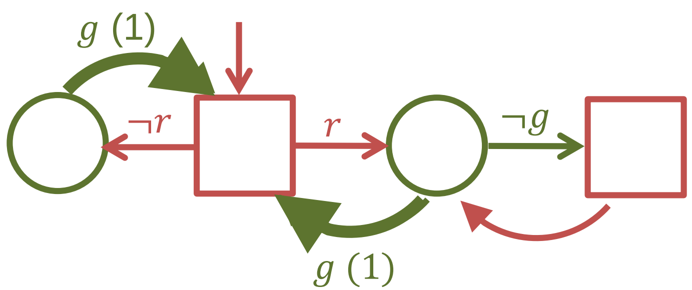
turn into game.

Example: Quick Grants

Mean payoff game:

- Circle maximizes, square minimizes.
- Unmarked edges have value 0

value? strategy?



game
strategy

Drawbacks of Worst Case Analysis?

- Input: r0, r1
- Output: g0, g1

Specification (in LTL):

Guarantees:

- $G(r0 \rightarrow g0)$
- $G(r1 \rightarrow g1)$
- $G \neg(g0 \wedge g1)$
- **minimize #grants**

Assumption:

- $G \neg(r0 \wedge r1)$

Suppose payoff 1 when no grant is given

Worst case value?

Optimal implementation?

Drawbacks of Worst Case Analysis?

- Input: r0, r1
- Output: g0, g1

Specification (in LTL):

Guarantees:

- $G(r0 \rightarrow g0)$
- $G(r1 \rightarrow g1)$
- $G \neg(g0 \wedge g1)$
- **minimize #grants**

Assumption:

- $G \neg(r0 \wedge r1)$

Worst case: grant in every tick – payoff 0

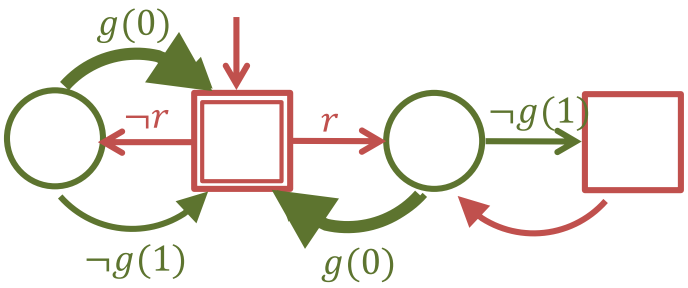
Thus, behavior when no requests arrive is irrelevant!

Arbiter that behaves best in worst case

$\neq$

best arbiter!

Drawbacks of Worst-Case Analysis



G(r → g)
mininize #g

value?
worst-case optimal: 0

optimal strategy?

An optimal, but undesirable strategy!

What to do?

Admissibility

- Strategy σ **dominates** strategy σ' if
  $\forall$antagonist strategies ρ, payoff(σ, ρ) $\geq$ payoff(σ', ρ)
  $\exists$antagonist strategy ρ, payoff(σ, ρ) > payoff(σ', ρ)
- Strategy σ' is **admissible** if there is no σ such that σ dominates σ'
- Careful: theorems from Boolean games break.
  - e.g. admissible strategy may not be winning
- Not all mean payoff games have finite admissible optimal strategies!

Case II: Liveness

- Liveness spec stated as parity automata
- Solve Mean-payoff parity game [ChatterjeeHenzingerJurdzinski05]
- Lexicographic version for multiple objectives [BloemChatterjeeHenzingerJobstmann09]

Robustness

(An Application of Quantitative Specs)

Robustness

A robust system behaves “reasonably” even in circumstances that were not anticipated in the requirements specification. [GhezziJazayeriMandrioli91]

Questions

- How do you specify robustness?
- How do you check robustness or construct robust systems?

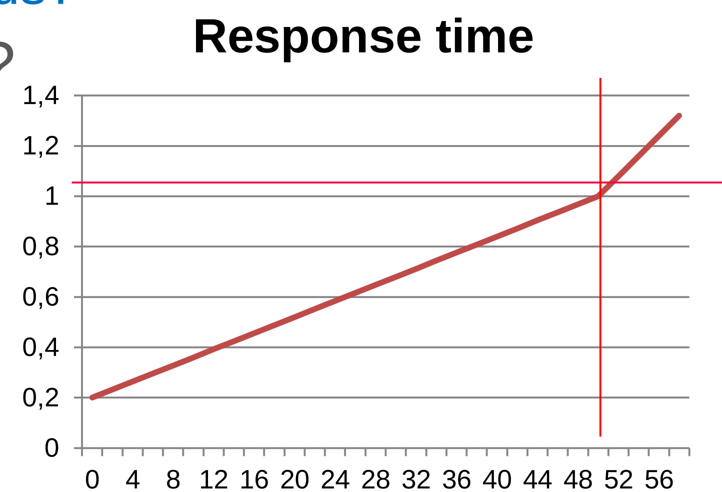
Very little attention in formal methods

Example: Air Traffic Control

The air traffic control system must track up to 50 planes.
(In that case,) response time must be at most 1 second.
[Davis90]

- What happens when plane 51 arrives?
  - System crashes?
  - Airplane 51 is ignored?
  - Response time goes up to 1.2 seconds?
- What about airplane 52? 53? 99?

You want **graceful degradation!**
But: digital systems have no natural notion of continuity!



Example: Arbiter

- Input: r0, r1
- Output: g0, g1

Specification (in LTL):

Guarantees G:

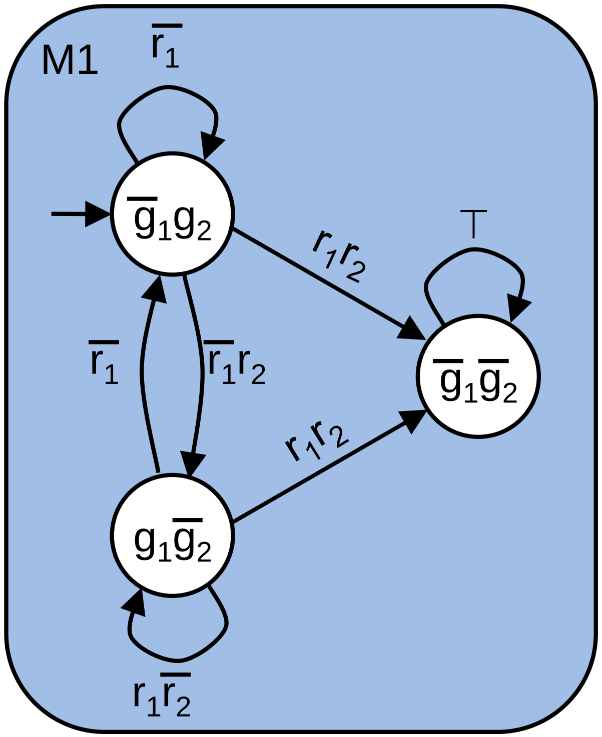
- $G(r0 \rightarrow X\ g0)$
- $G(r1 \rightarrow X\ g1)$
- $G\ \neg(g0 \wedge g1)$

Assumption A:

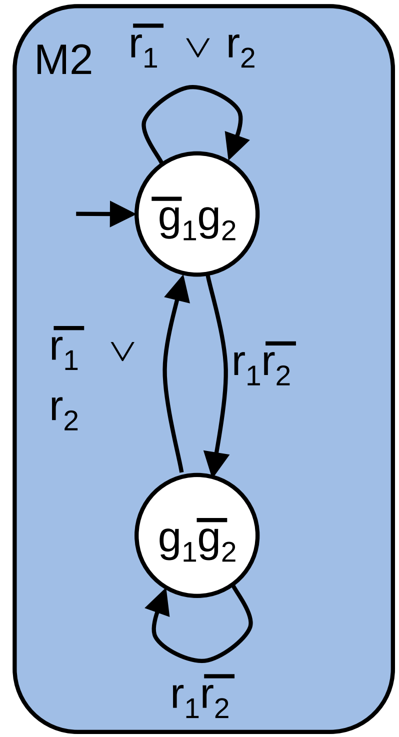
- $G\ \neg(r0 \wedge r1)$

Two Correct Controllers

Does not recover from an error!



Does recover from an error!



Verification does not distinguish between two systems
Synthesis may give you either system

What May Go Wrong?

- System errors
  - Soft errors (transient)
  - Permanent faults
- Environment errors
  - Operator error
  - Transmission line error
  - Implementation error

We focus on environment errors

What is Reasonable?

Typical proposals:

- System behavior unchanged [FeySuelflowDrechsler]
- System behaves according to original spec [SeshiaLiMitra]

too strict

- System recovers to safe state [self-stabilization, Dijkstra]
- System recovers to safe state quickly [Baarir et al.]

too lax?

What is Reasonable?

Claim: User should decide what is reasonable

For arbiter:

When two requests come

- drop one?
- drop both?
- grant both?

$g1 = G(r1 \rightarrow X\ g1) \wedge G(r2 \rightarrow X\ g2)$

$g2 = G\ \neg(g1 \wedge g2)$

$a\ \ = G\ \neg(r1 \wedge r2)$

Spec: $a \rightarrow g1 \wedge g2$

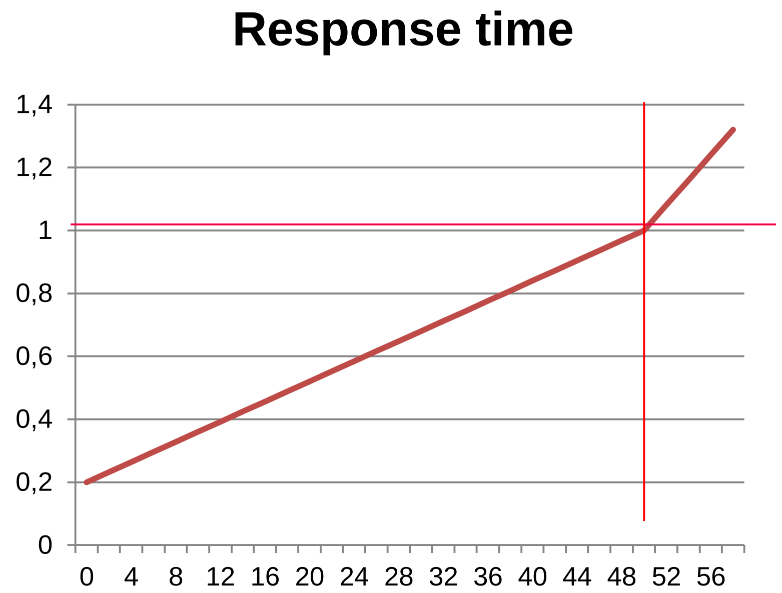
How do we state what is preferable?

Stating what is Preferable

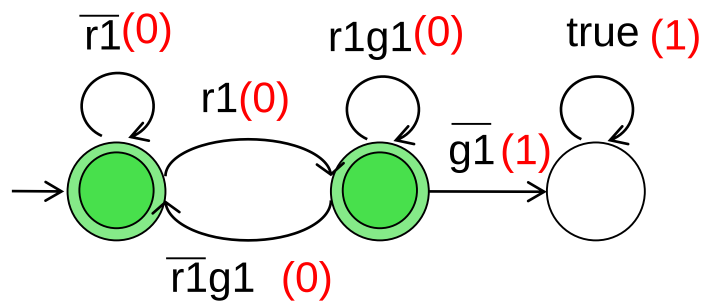
Case by case analysis of wrong behavior?

- bothersome!
- impossible?

planes	response time (s)
≤50	1
51	1.1
52	1.2
53	1.3
…	…



Proposal: Error Functions as Automata



G(r1 → X g1)

Error measure d is sum of weights on edges

Good properties of this error function:

- Behavior σ is error-free: d(σ)=0
- Behavior σ has errors: d(σ)>0

Bad property:

- Does not distinguish between single and multiple errors

r1	0	1	1	1	1	1	...
r2	0	0	0	0	0	0	...
g1	0	0	1	1	1	1	...
g2	1	1	0	0	0	0	...

Environment error: 0

System error: 0

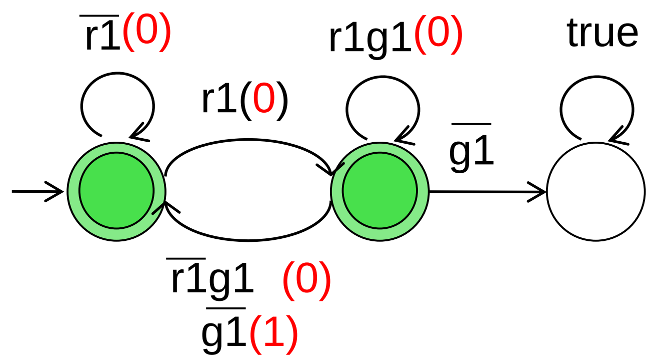
r1	0	1	1	1	1	1	...
r2	0	1	0	0	0	0	...
g1	0	0	0	1	1	1	...
g2	1	1	1	0	0	0	...

Environment error: 1

System error: ∞

SCOS
Secure & Correct Systems

A Better Error Function



similar for other properties

r1	0	1	1	1	1	1	...
r2	0	0	0	0	0	0	...
g1	0	0	1	1	1	1	…
g2	1	1	0	0	0	0	…

Environment error: 0
System error: 0

r1	0	1	1	1	1	1	...
r2	0	1	0	0	0	0	...
g1	0	0	0	1	1	1	…
g2	1	1	1	0	0	0	…

Environment error: 1
System error: 1

Error Specifications

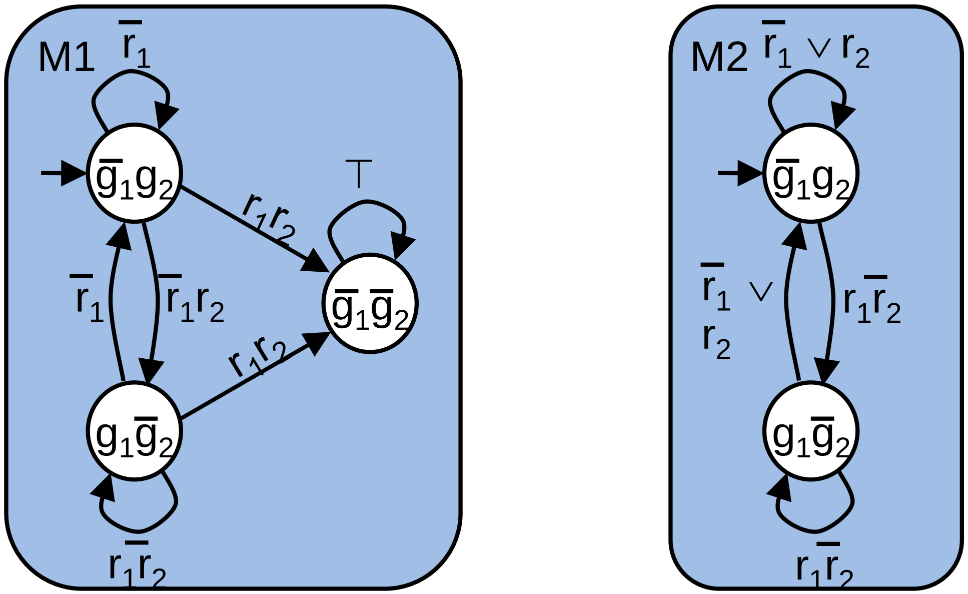
- Specs have the form A $\rightarrow$ G
- Error specs consist of an error automaton for the environment and one for the system
  - For each word: an error value for environment and for system
- Specify
  - How you interpret incorrect input?
  - How to continue with output
- Typical choices for input:
  - ignore input
  - reset
  - treat like similar input

Robustness

Robustness = recovery from error

- We call a system **robust** if
  - Finite environment error implies finite system error

Cf. two arbiters



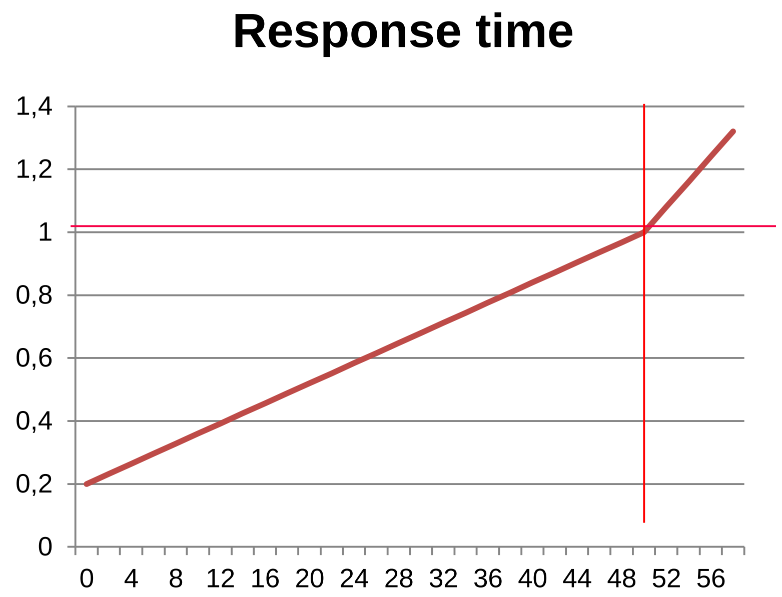
Refining the Idea – Quantitative Specs

Spec is of the form A $\rightarrow$ G

- A are assumptions on environment
- G are guarantees of system

Idea: take ratio of system errors to environment errors

- Airplanes: ratio of excess planes to excess response time
- Arbiter: ratio of double requests to missed requests

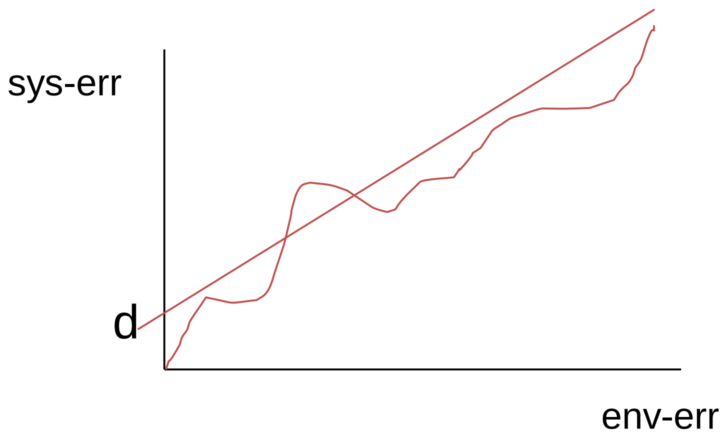


Ratios

System is **k-robust** if

For every environment error, there are at most k system errors (in the limit)

$\exists d$: sys-err = k · env-err + d



Robustness – Wrap-up

Questions:

- how to specify robustness (graceful degradation)
- how to check robustness
- how to synthesize robust systems

One solution:

- User defines costs for “non-standard” behavio
  - Value of a words: mean payoff automaton
  - Value of a system: minimium value of its words
  - Combining values: addition or lexicographic
- Robustness means that system can only make finitely many errors if the system does
- k-robustness means that the ration between system faults and environment faults is at most k.

One more slide, bear with me :-)

Concluding - Synthesis

- Synthesis: Applying game theory to real problems
- Solving games
- Constructing efficient strategies/implementations
- Distributed and parameterized cases
- Specification
  - influences complexity, expressibility, ease of use
  - Quantitative measures may help

Thanks for your interest and patience.

Bibliography

[VMCAI12] B. Finkbeiner, S. Jacobs: Lazy Synthesis. VMCAI 12.

[PnueliRosner90] A. Pnueli, R. Rosner: Distributed Reactive Systems are Hard to Synthesize. FOCS 90.

[FinkbeinerSchewe05] B. Finkbeiner, S. Schewe: Uniform Distributed Synthesis. LICS 05.

[TACAS12] S. Jacobs, R. Bloem: Parameterized Synthesis. TACAS 12.

[VMCAI13] A. Khalimov, S. Jacobs, R. Bloem: Towards Efficient Parameterized Synthesis. VMCAI 13.

[EmersonNamjoshi95] E. Emerson, K. Namjoshi: Reasoning about Rings. POPL 95.

[EhrenfeuchtMycielski79] A. Ehrenfeucht, J. Mycielski: Positional Strategies for Mean Payoff Games. IJGT 79.

[ChatterjeeHenzingerJurdzinski05] K. Chatterjee, T. Henzinger, M. Jurdzinski: Mean-Payoff Parity Games. LICS 05.

[BloemChatterjeeHenzingerJobstmann09] R. Bloem, K. Chatterjee, T. Henzinger, B. Jobstmann: Better quality in synthesis through quantitative objectives. CAV 09.

[GhezziJazayeriMandrioli91] C. Ghezzi, M. Jazayeri, D. Mandrioli: Software qualities and principles.