Synthesizing Robust Systems

Roderick Bloem and Karin Greimel (TU-Graz)

Thomas Henzinger (EPFL and IST-Austria)

Barbara Jobstmann (CNRS/Verimag)

FMCAD 2009 in Austin, Texas

Motivation

When designing systems:

- environment might not be known, or cannot be precisely modeled, e.g., physical environment
- behaves differently from expected
  - program in libraries, might be used in other than the intended way
  - transmission and other errors
- environment model necessary

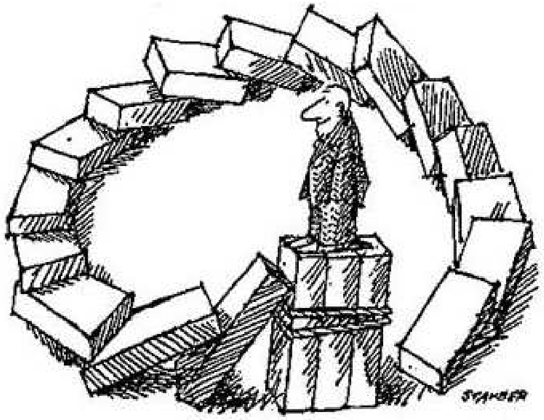
Motivation

A system should not only be

- **correct**, it should also
- behave '**reasonably**,' even in circumstances that were not anticipated in the requirements specification[. . .]

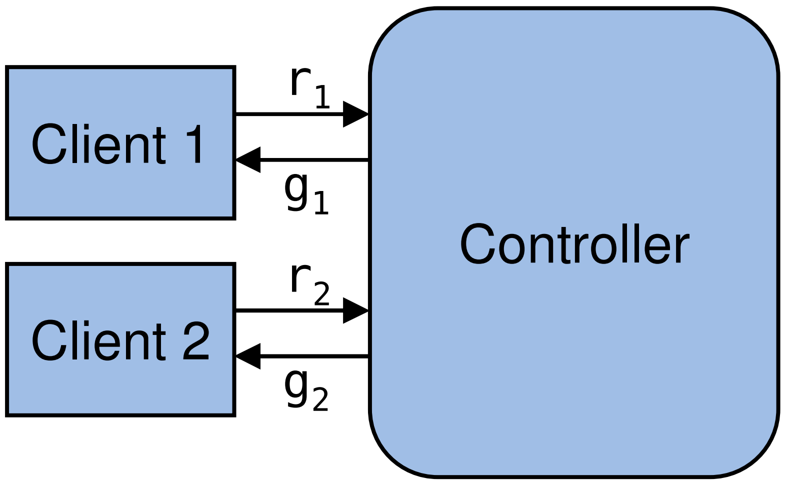
["Fundamentals of software engineering" by GJM]

In short, it should also be **robust.**



Simple Example

- Resource controller for two clients
  - Clients send requests using signals r_1, r_2
  - Controller grant resource using signals g_1, g_2

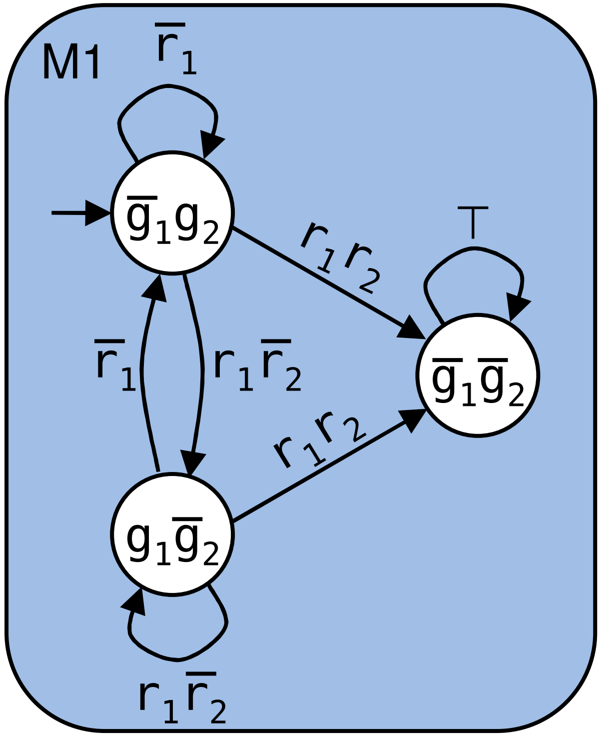


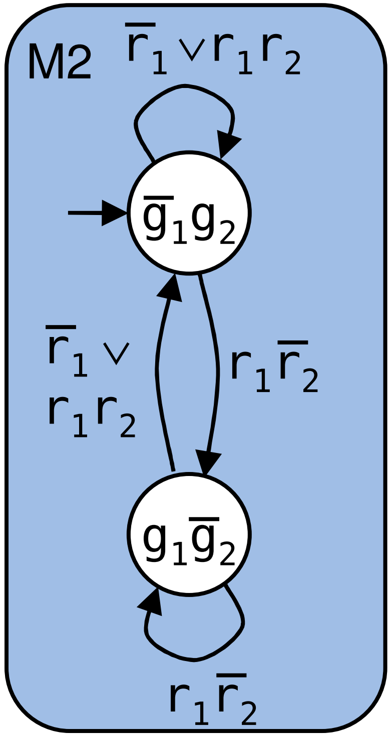
Specification

- Assumption A on clients
  - $\texttt{never}(r_1 \wedge r_2)$
- Guarantee G of controller
  - $\texttt{always}(r_1 \rightarrow \texttt{next}(g_1))$
  - $\texttt{always}(r_2 \rightarrow \texttt{next}(g_2))$
  - $\texttt{never}(g_1 \wedge g_2)$
- Specification $A \rightarrow G$

Two Correct Controllers

Specification: A→G



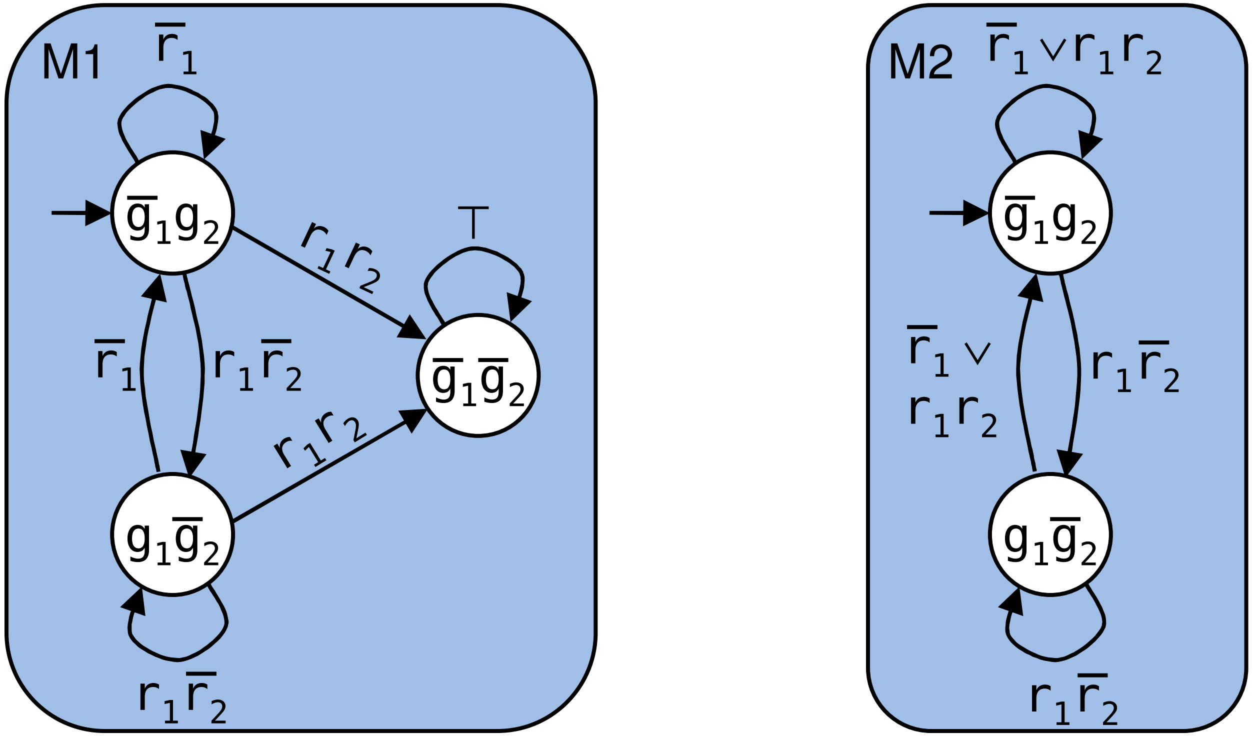


Input trace: $(r_1 r_2)(\overline{r}_1 r_2)^\omega$ $\quad$ $(r_1 r_2)(\overline{r}_1 r_2)^\omega$

Output trace: $(\overline{g}_1 g_2)(\overline{g}_1 \overline{g}_2)^\omega$ $\quad$ $(\overline{g}_1 g_2)(\overline{g}_1 g_2)^\omega$

Two Correct Controllers

Does not recover from an error! | **Does recover from an error!**



Input trace: $(r_1 r_2)(\overline{r}_1 r_2)^\omega$ | $(r_1 r_2)(\overline{r}_1 r_2)^\omega$

Output trace: $(\overline{g}_1 g_2)(\overline{g}_1\overline{g}_2)^\omega$ | $(\overline{g}_1 g_2)(\overline{g}_1 g_2)^\omega$

Outline

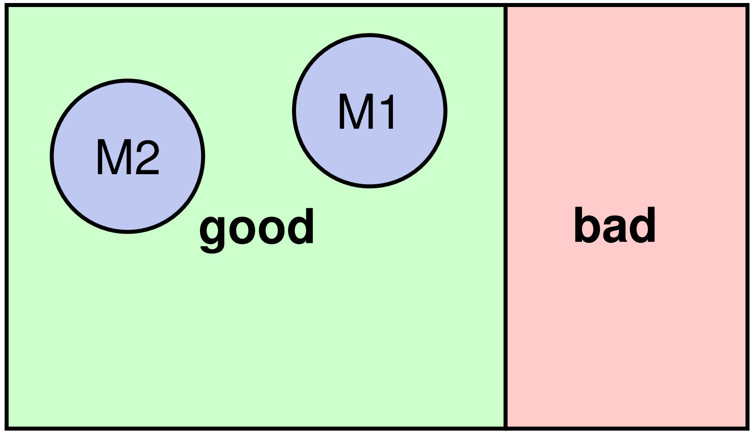
- Definition
  - Error specifications
  - Robustness (our proposal)
- Verify Robustness
- Synthesize Robust Systems
- Conclusion

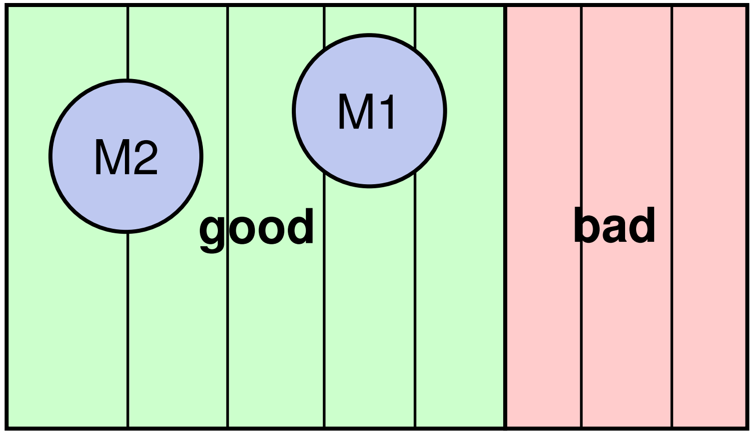
Setting

- Reactive finite-state system M (signals I/O)
  - Alphabet $\Sigma = 2^{I \cup O}$ (evaluations of $I \cup O$)
  - Behavior of M is a sequence $\sigma \in \Sigma^\omega$
  - Set of all behavior: L(M)
- Specification $\varphi = A \rightarrow G$
  - A is assumption on environment
  - G is guarantee of system
  - A and G are safety specifications over Σ
  - σ satisfies or does not satisfy φ

Error Specification (1)

- Error function d: $\Sigma^\omega \cup \Sigma^* \rightarrow N \cup \{\infty\}$
  - maps all behaviors/prefixes to number or ∞
  - Idea: count errors (violations of specification), **higher value means more errors**
  - Quantitative specification



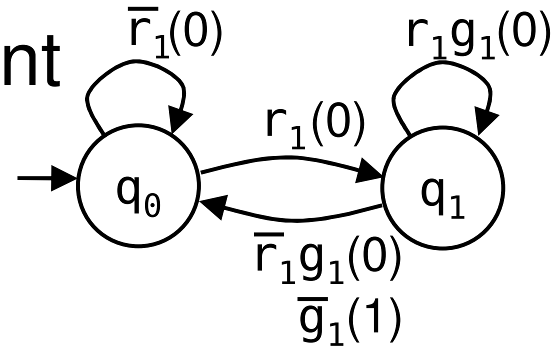


Error Specification (2)

- Error function d: $\Sigma^\omega \cup \Sigma^* \rightarrow N \cup \{\infty\}$
- Det. Automaton A with weight function w mapping edges to weights

  $d(\sigma)$ = sum of weights of run over σ

- Error specification is pair (d_e,d_s)
  - d_e…error function for environment
  - d_s…error function for systems

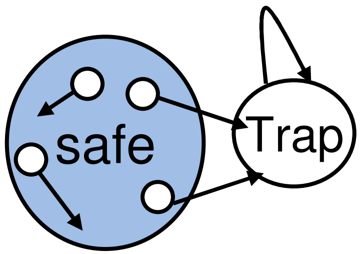


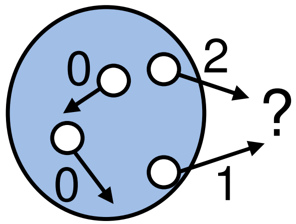
From Spec to Error Function

- In general, “a design choice”
- We show: one way of going from spec to error function

Safety Spec → Automaton → Automaton + weight fct → Error Function

$\varphi \subseteq \Sigma^\omega$

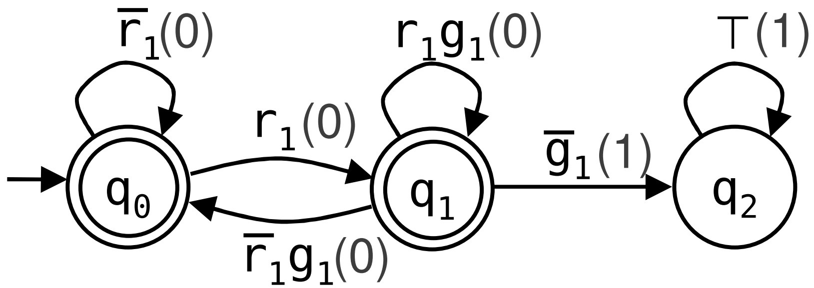




$d: \Sigma^{*/\omega} \to N$

From Spec to Error Function

Recall, example: $\mathtt{always}(r_1 \rightarrow \mathtt{next}(g_1)) \wedge \ldots$



Safe region=$\{q_0,q_1\}$

Good properties of this error function:

- If behavior σ is error-free, then $d(\sigma)=0$
- If behavior σ has errors $d(\sigma)>0$

Bad properties:

- Does not distinguish between single and multiple errors (it's a trap)

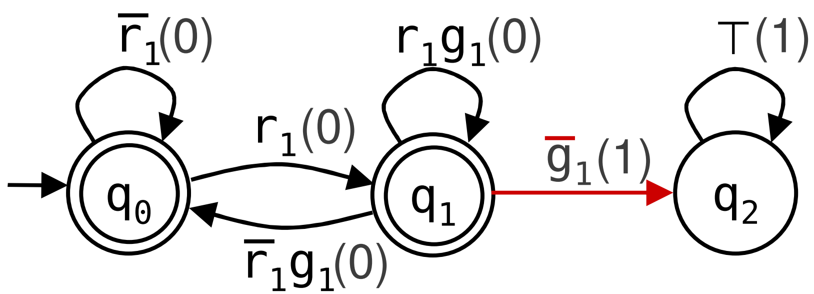
Example

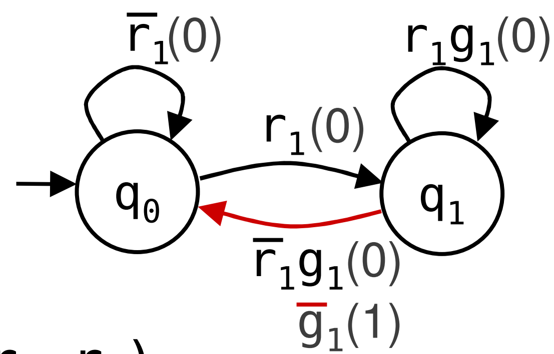
What to do with traps?

(a) Reset $\rightarrow$ go to initial states (restart property)

(b) Skip $\rightarrow$ self loop (ignore input)

(c) Follow closest correct letter in alphabet





A: never$(r_1 \wedge r_2)$	$d_e = \#(r_1 \wedge r_2)$
G_1: always$(r_1 \rightarrow$ next$(g_1))$	$d_{s1} = \#(r_1 \wedge$ next$(\overline{g}_1))$
G_2: always$(r_2 \rightarrow$ next$(g_2))$	$d_{s2} = \#(r_2 \wedge$ next$(\overline{g}_2))$
G_3: never$(g_1 \wedge g_2)$	$d_{s3} = \#(g_1 \wedge g_2)$

$d_s = d_{s1} + d_{s2} + d_{s3}$

Robustness

- System M is **robust** wrt error spec (d_e,d_s) if forall $\sigma \in L(M)$:

$$d_e(\sigma) \neq \infty \text{ implies } d_s(\sigma) \neq \infty$$

- System M is **k-robust** wrt (d_e,d_s) if exists $c \in N$ s.t. forall $\sigma \in prefix(L(M))$

$$d_s(\sigma) \leq k \cdot d_e(\sigma) + c$$

- k-robustness classifies robust systems wrt infinite behavior

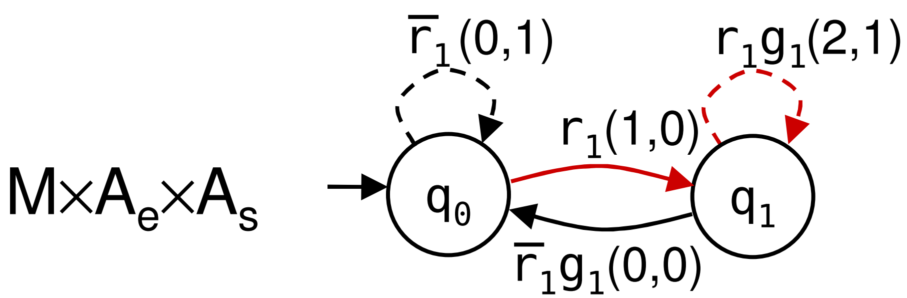
Verifying Robustness (1)

- Given system M and automata A_e and A_s for d_e and d_s, resp., check if M is robust
  - Compute product $M \times A_e \times A_s$ with two weight functions w_s and w_e
  - Then, M is robust if forall $\sigma \in L(M \times A_e \times A_s)$ sum of w_e over $\sigma \neq \infty$ implies sum of $w_s \neq \infty$.
  - Equivalently, if infinitely often $w_s > 0$, then infinitely often $w_e > 0$ (Streett)

Verifying Robustness (2)

– Two sets E_s, E_e of edges: one with $w_s>0$ and on with $w_e>0$

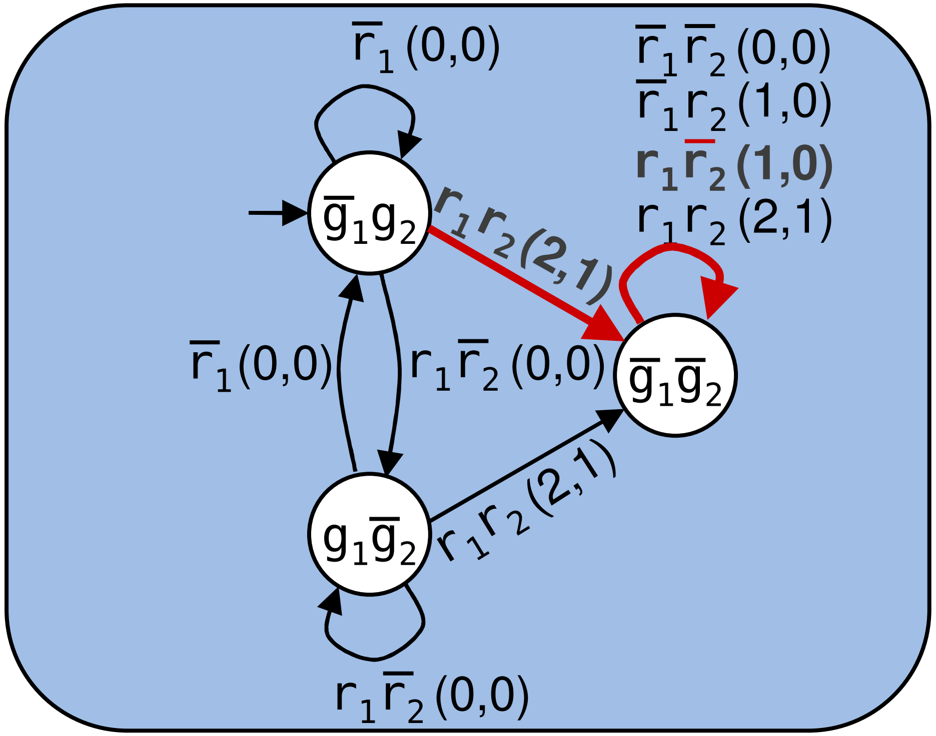
– Streett cond. with pair (E_s,E_e), linear in #edges



Example

$M \times A_e \times A_s$

	(s,e)
$d_e = \#(r_1 \wedge r_2)$	(0,1)
$d_{s1} = \#(r_1 \wedge next(g_1))$	(1,0)
$d_{s2} = \#(r_2 \wedge next(\overline{g}_2))$	(1,0)
$d_{s3} = \#(g_1 \wedge g_2)$	(1,0)



infinitely often $w_s > 0$
but only finitely often $w_e > 0$

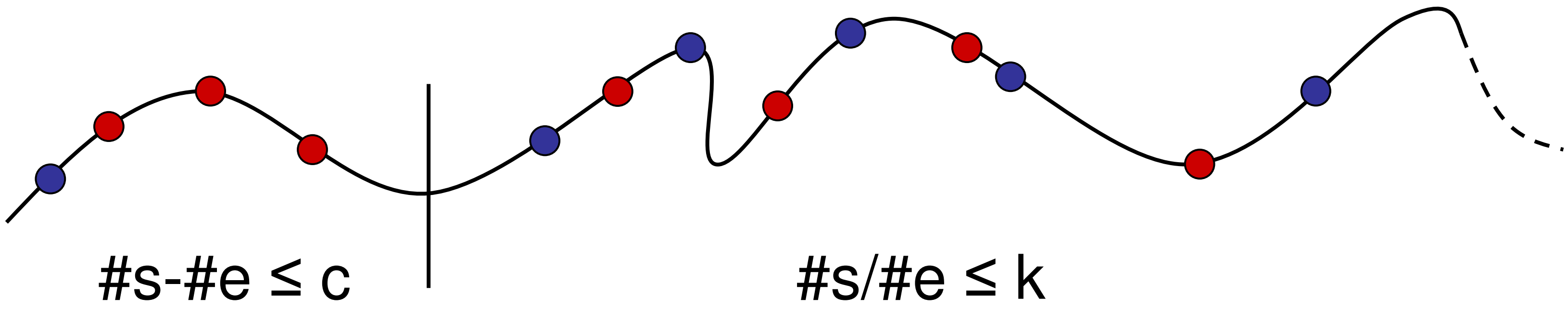
Non-robust

Verifying K-Robustness (1)

- Recall, exists $c \in N$, forall $\sigma \in$ prefix(L(M))

$$d_s(\sigma) \leq k \cdot d_e(\sigma) + c$$

- **0-Robust** = finitely many errors $d_s(\sigma) \leq c$
- **k-Robust** = average ratio between #system errors and #environment errors is $\leq k$



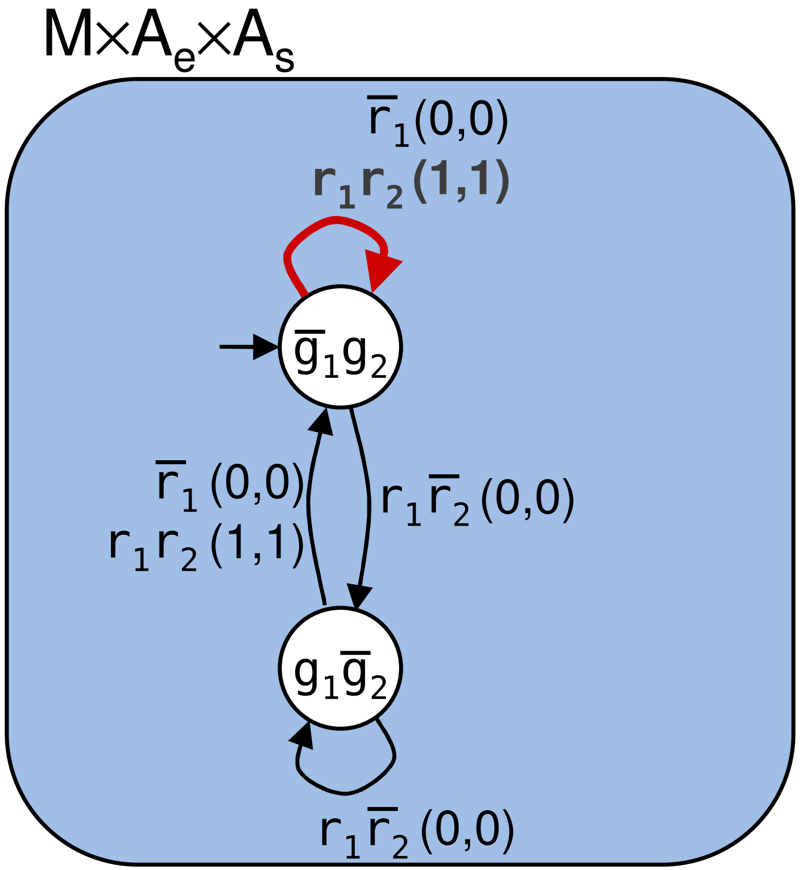
Verifying K-Robustness (2)

- Given robust system M, automata A_e and A_s, check if M is k-robust.
  - Compute product $M \times A_e \times A_s$ with two weight functions w_s and w_e
  - M is k-Robust if forall runs $q_0 q_1 q_2 \ldots$

$$\lim_{m\to\infty} \liminf_{l\to\infty} \frac{\mathrm{sum}_{i=m..l}(w_s(q_i,q_{i+1}))}{1+\mathrm{sum}_{i=m..l}(w_e(q_i,q_{i+1}))} \leq k$$

  - True if maximum simple cycle ratio $\leq k$, computable in $O(\#states^2 \cdot \#edges)$

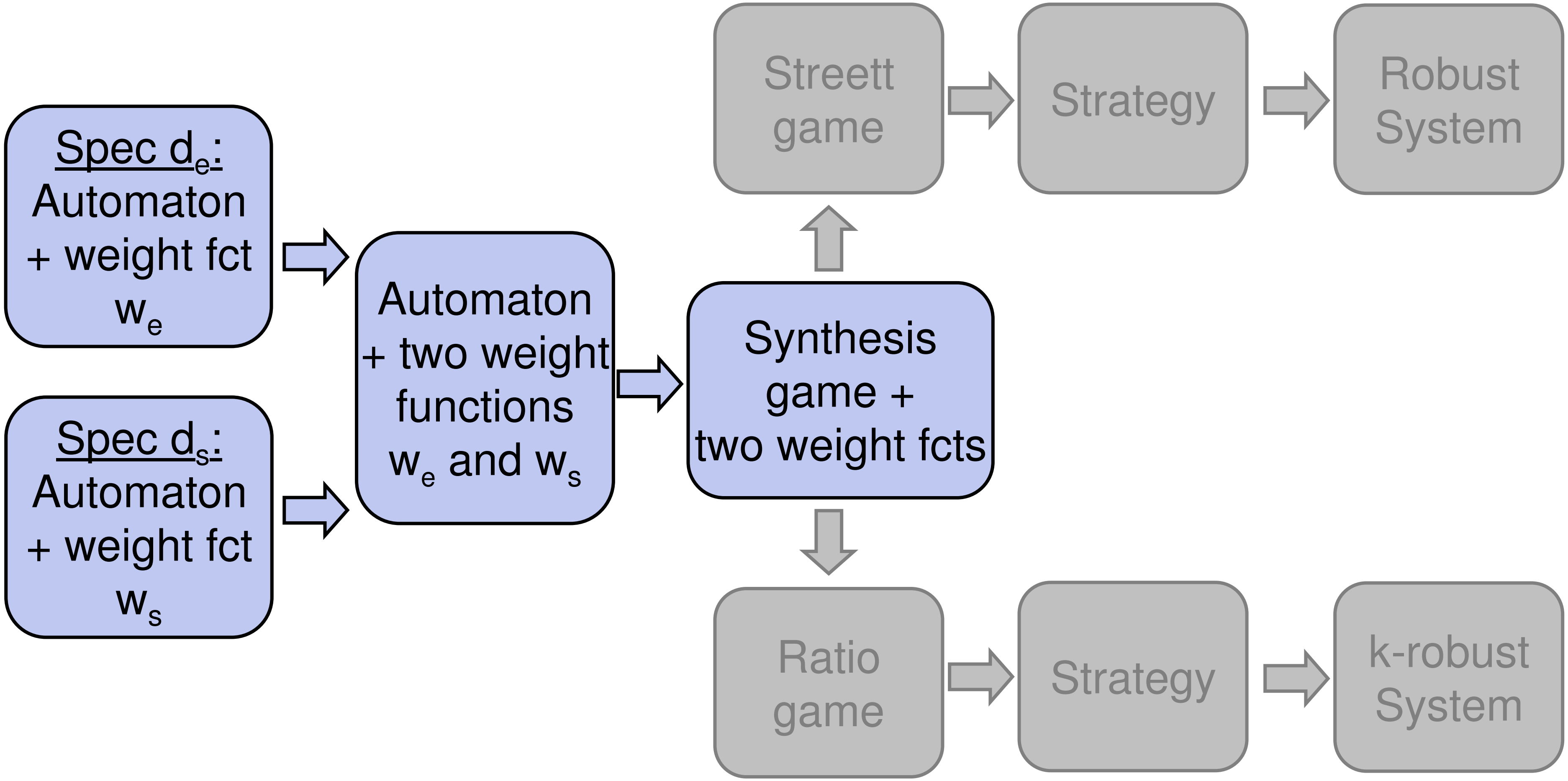
Example



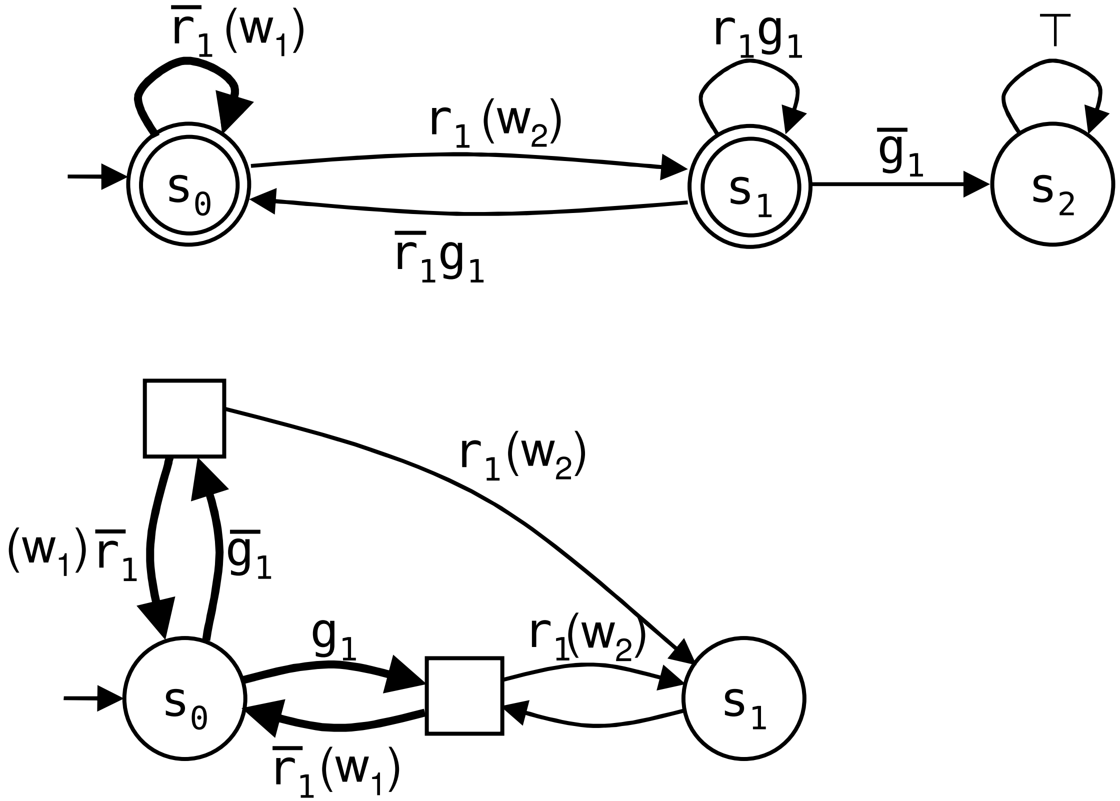
maximum simple cycle ratio=1

1-robust

Synthesizing Systems

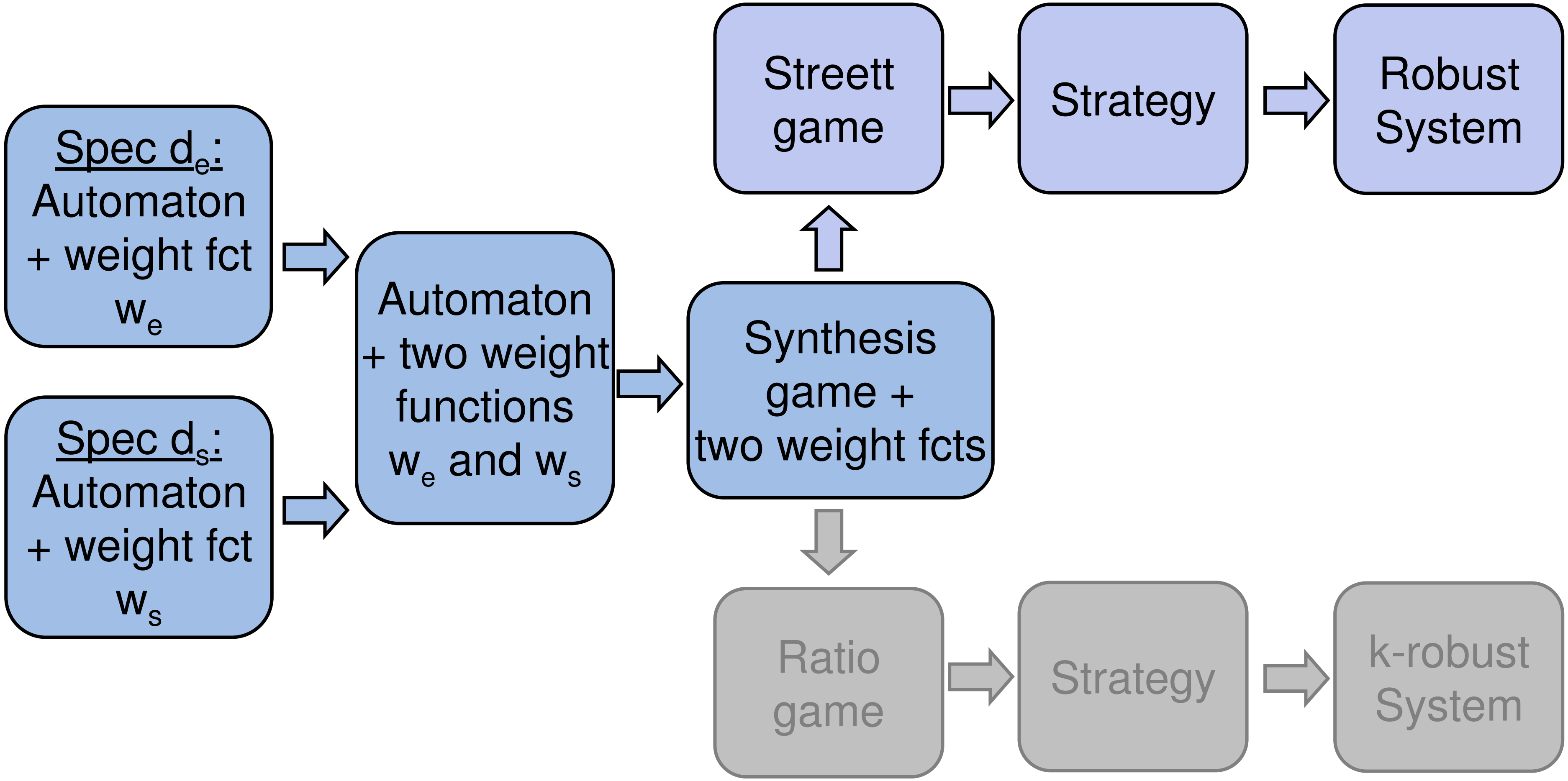


Synthesis Game: Example



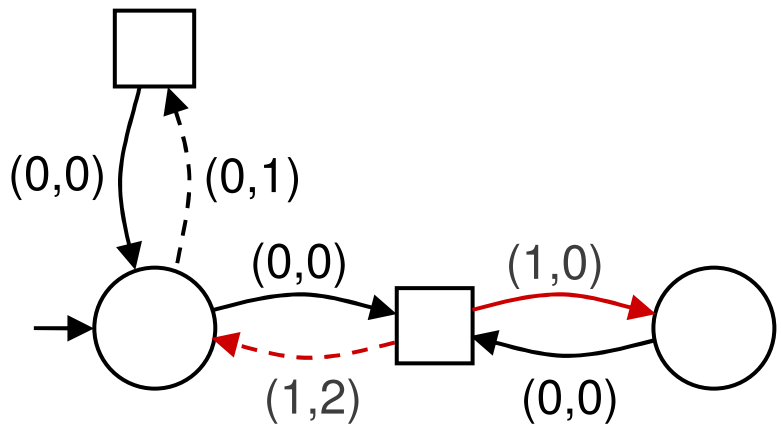
- Two players
- Alternately, pick signal assignments
- Add weights accordingly
- Winning objective?

Synthesizing Systems

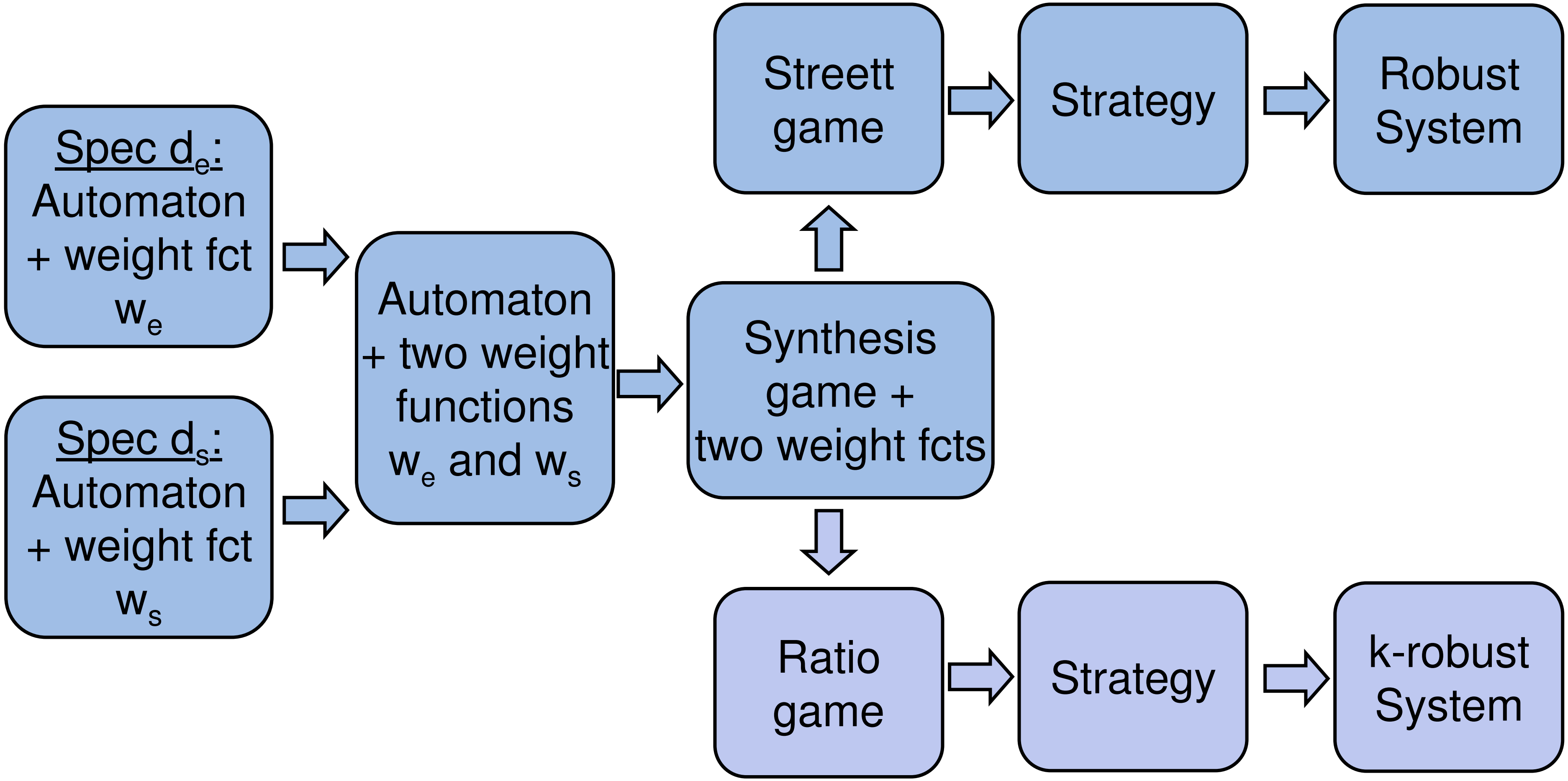


Synthesizing Robust Systems

- Play is winning: if infinitely many edges with $w_s>0$, then infinitely many edges with $w_e>0$
- Two sets E_s, E_e of edges: one with $w_s>0$ and on with $w_e>0$
- Winning objective: Streett with 1-pair (E_s,E_e)



Synthesizing Systems



Ratio Game

- A novel game type
- Formally, value of a play $\rho = q_0 q_1 q_2 \ldots$ is

$$v(\rho) = \lim_{m\to\infty} \liminf_{l\to\infty} \frac{\mathrm{sum}_{i=m..l}(w_s(q_i, q_{i+1}))}{1+\mathrm{sum}_{i=m..l}(w_e(q_i, q_{i+1}))}$$

- Objective of Player System: minimize $v(\rho)$
- In paper, we show that in ratio games
  - both players have memoryless winning strategies and
  - reduction to mean-payoff game (decision)

Conclusion

- A notion of robustness based on error functions
- Algorithms to
  - verify robustness and k-robustness
  - synthesize robust systems with minimal k (based on our ratio games)