

From Checking Program Correctness To Measuring Program Fitness

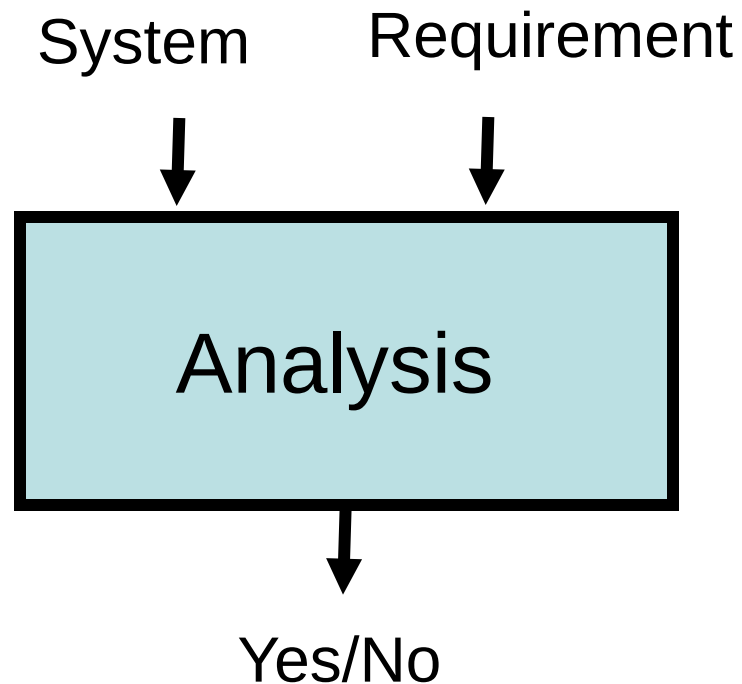
Tom Henzinger
IST Austria

Joint work with Pavol Cerny,
Krishnendu Chatterjee, Jan Otop,
Arjun Radhakrishna, and many others.

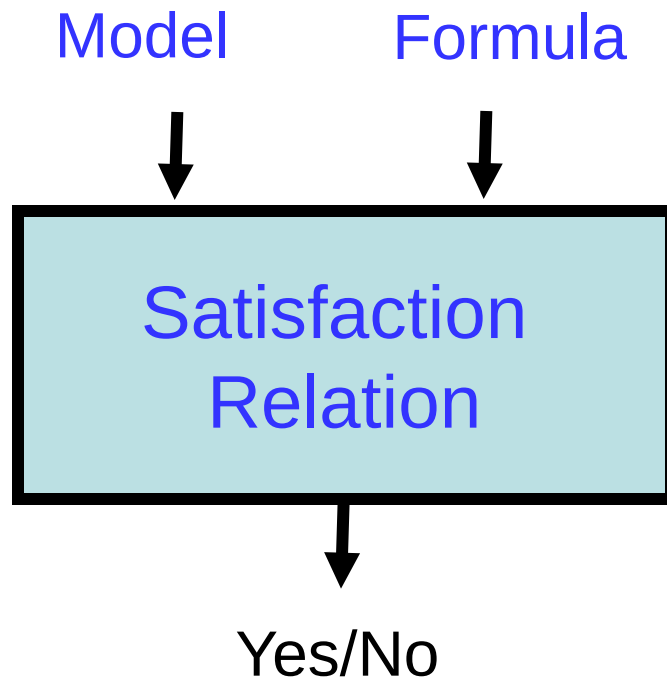


European
Research
Council

Formal Verification



Model Checking



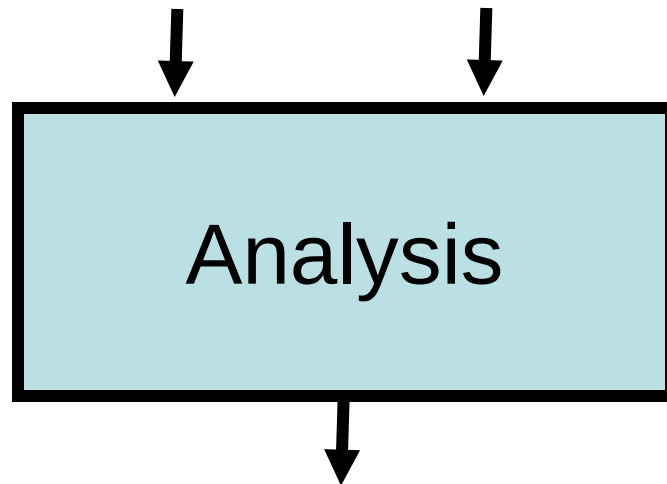
Model Checking

Finite
Automaton

Model

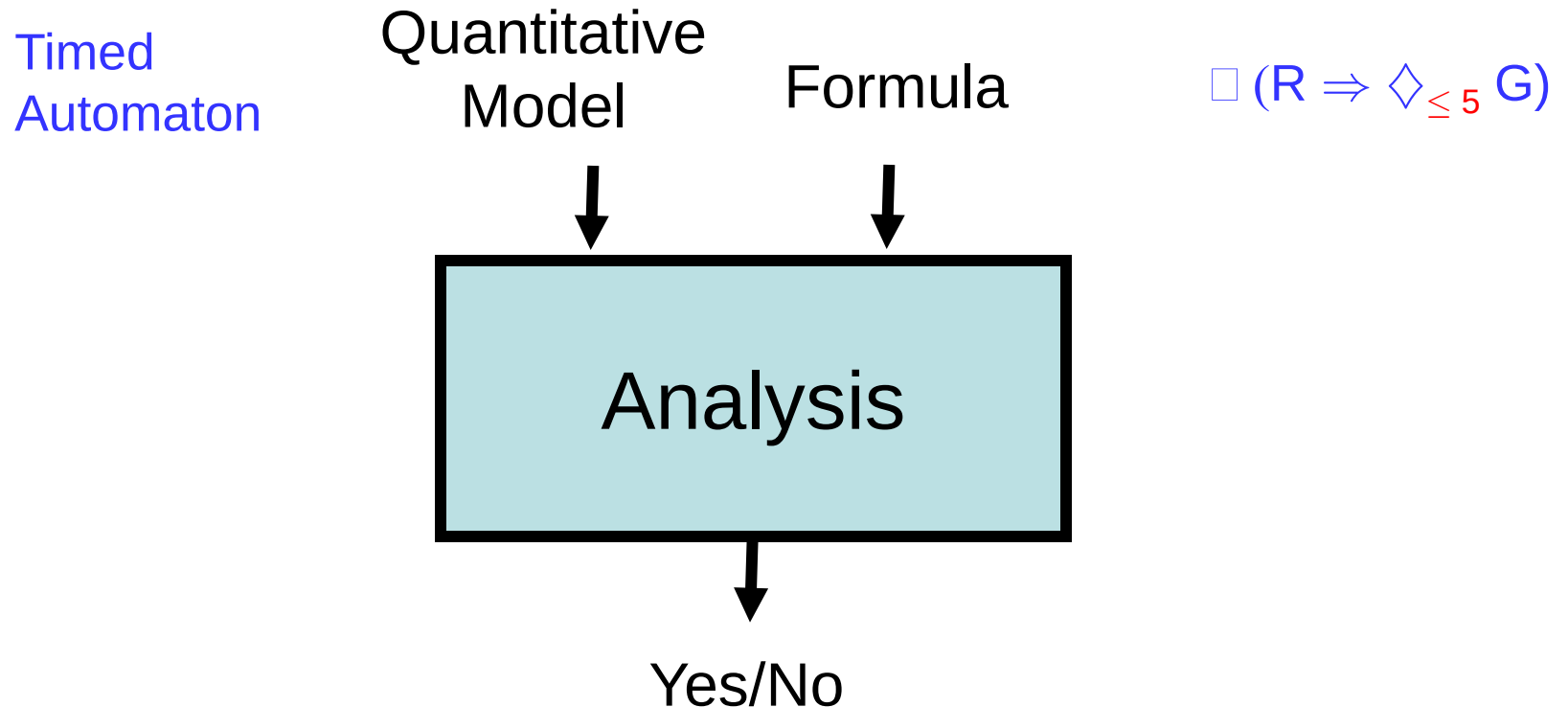
Formula

$\Box (R \Rightarrow \Diamond G)$



Yes/No

Quantitative Model Checking



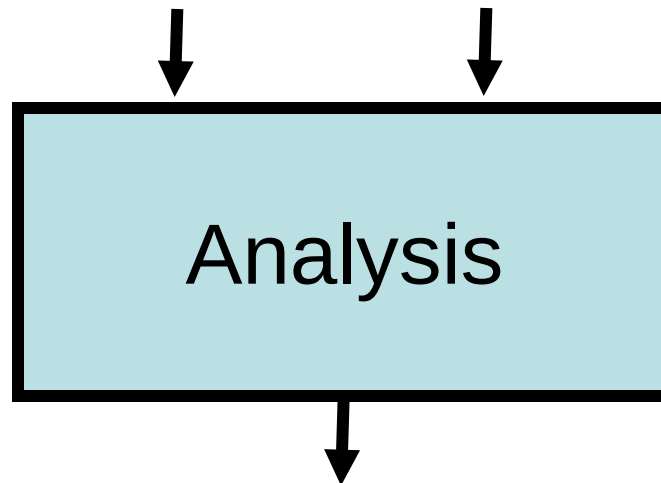
Quantitative Model Checking

Markov
Process

Quantitative
Model

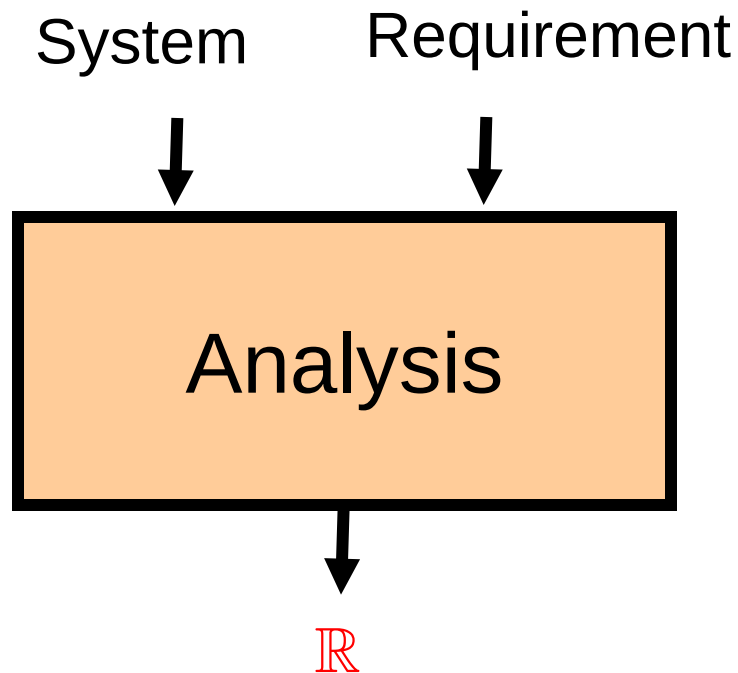
Formula

$\forall \square (R \Rightarrow$
 $\text{Pr}(\diamond G) \geq 0.5)$



Yes/No

Our Agenda



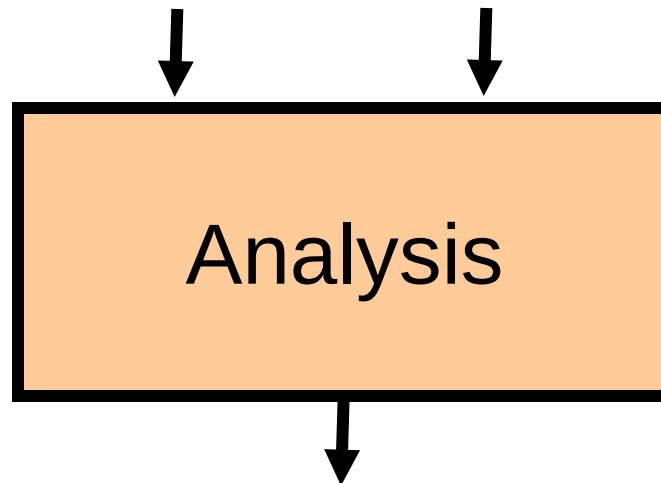
- measure of “fit” between system and requirement
- could involve performance, quality, cost, etc.

Model Measuring

Quantitative
Model

Formula

$\square (R \Rightarrow \diamond G)$



Analysis

R

The less time between
R and G, the better.

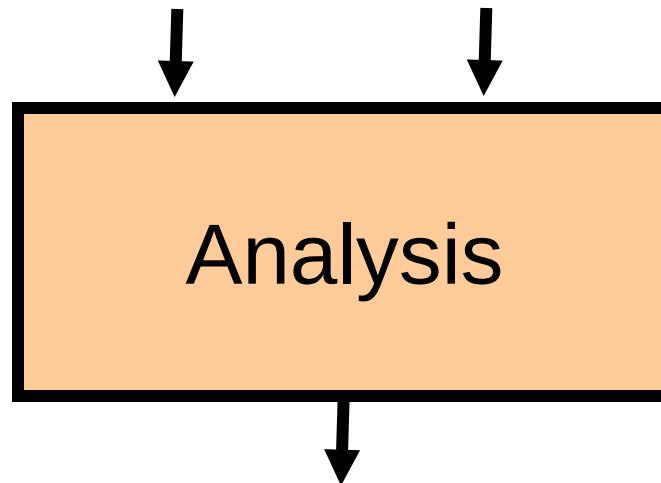
Worst or average response time

Model Measuring

Model

Formula

$\Box (R \Rightarrow \Diamond G)$

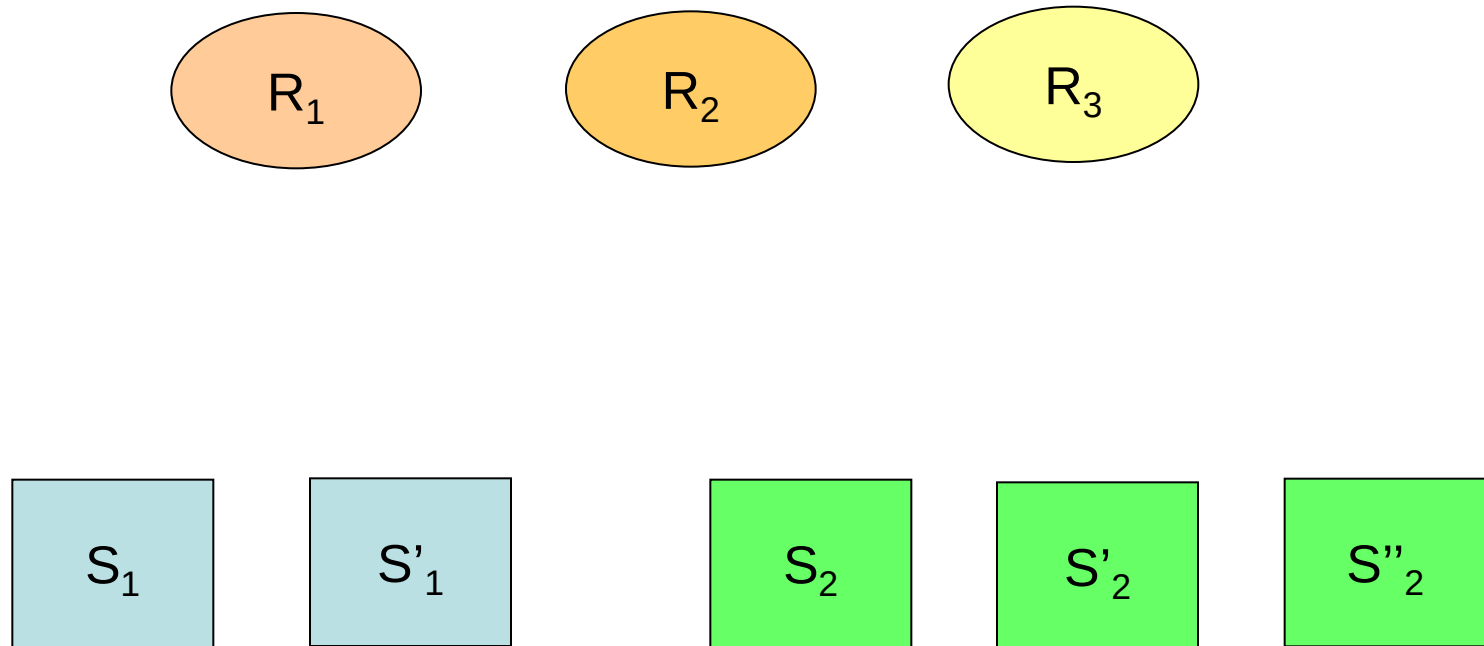


The fewer “unnecessary”
grants G , the better.

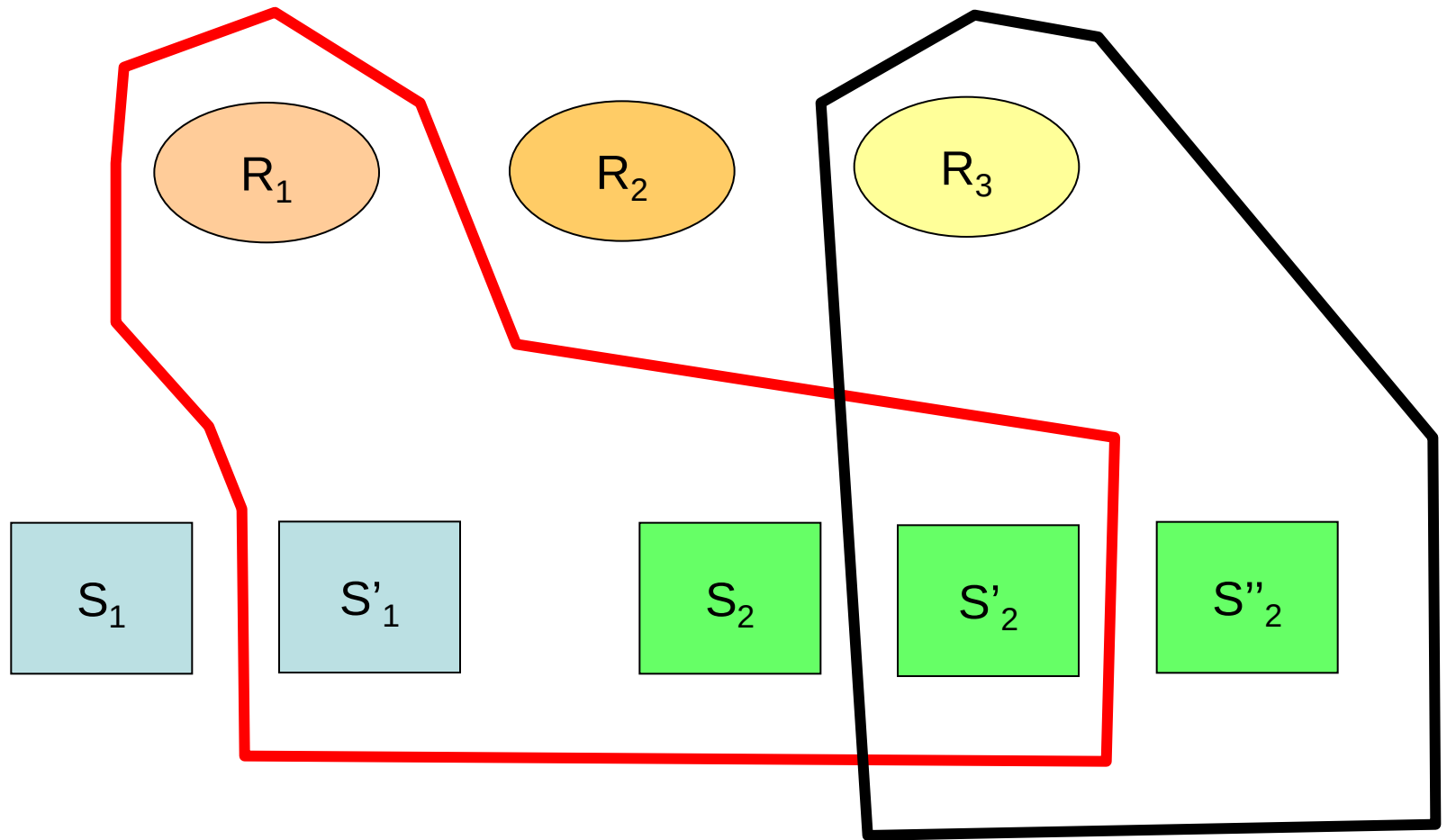
$\mathbb{R}$

Max or avg number of grants per request

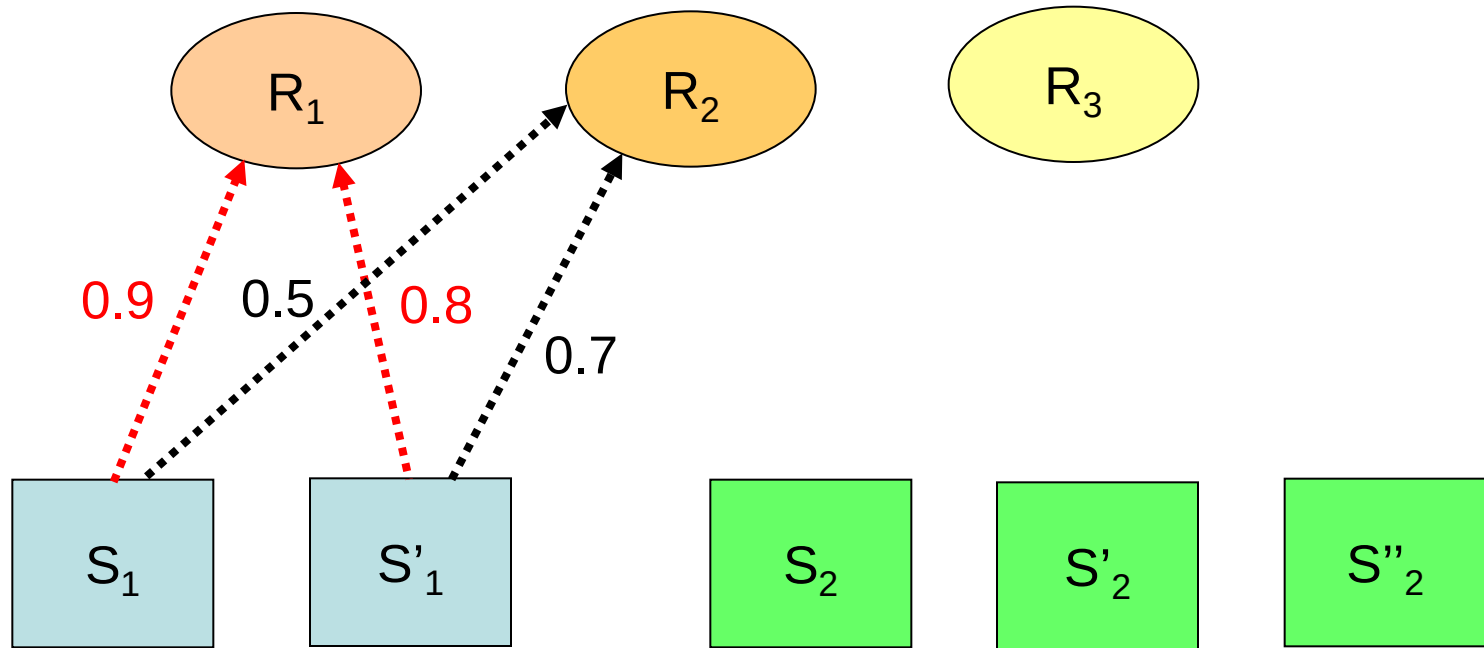
Systems and Requirements



Correctness = Preorder



Fitness = Metric



Example: Incompatible Requirements

$$\Box (R_1 \Rightarrow G_1)$$

$$\Box (R_2 \Rightarrow G_2)$$

$$\Box \neg (G_1 \wedge G_2)$$

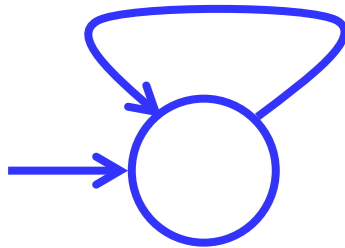
Example: Incompatible Requirements

$$\Box (R_1 \Rightarrow G_1)$$

$$\Box (R_2 \Rightarrow G_2)$$

$$\Box \neg (G_1 \wedge G_2)$$

$R_1 r_2 / G_1 g_2$
 $r_1 R_2 / g_1 G_2$
 $R_1 R_2 / G_1 g_2$
 $r_1 r_2 / g_1 g_2$

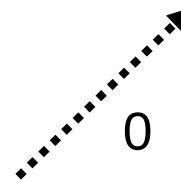
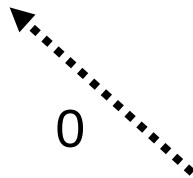


Example: Incompatible Requirements

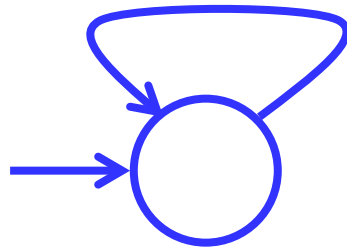
$$\Box (R_1 \Rightarrow G_1)$$

$$\Box (R_2 \Rightarrow G_2)$$

$$\Box \neg (G_1 \wedge G_2)$$



$R_1 r_2 / G_1 g_2$
 $r_1 R_2 / g_1 G_2$
 $R_1 R_2 / G_1 g_2$
 $r_1 r_2 / g_1 g_2$

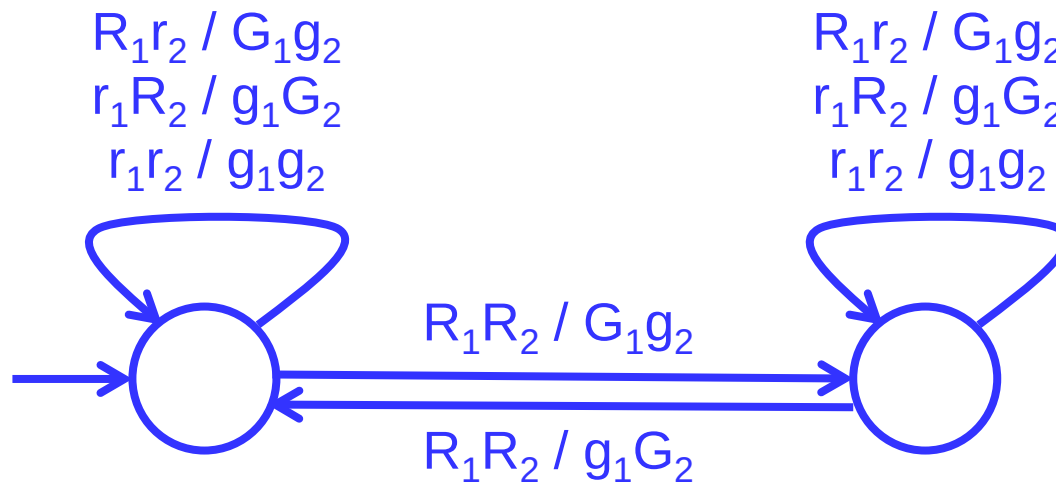


Example: Incompatible Requirements

$$\Box (R_1 \Rightarrow G_1)$$

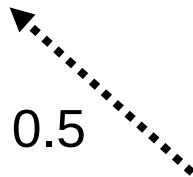
$$\Box (R_2 \Rightarrow G_2)$$

$$\Box \neg (G_1 \wedge G_2)$$

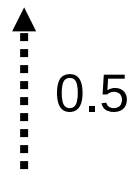


Example: Incompatible Requirements

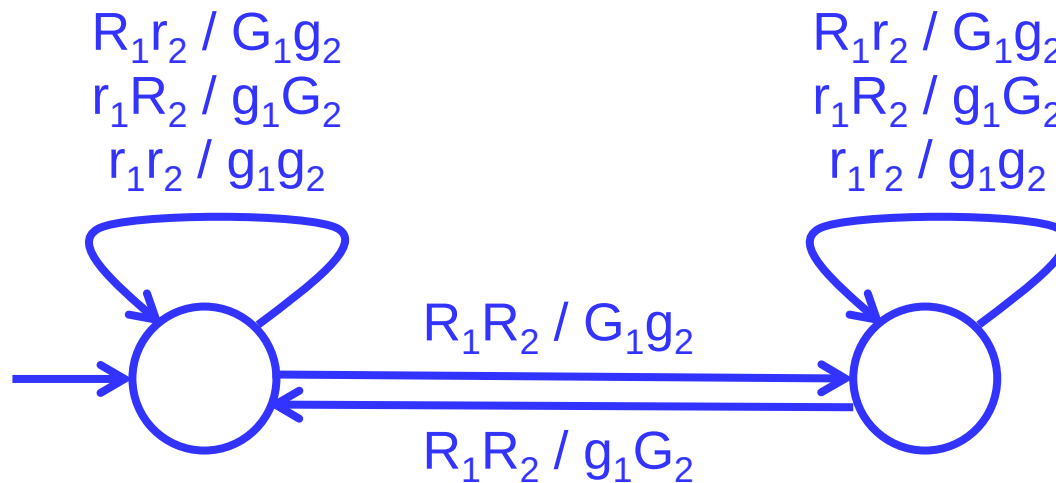
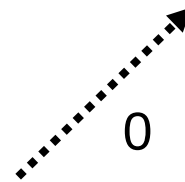
$$\Box (R_1 \Rightarrow G_1)$$



$$\Box (R_2 \Rightarrow G_2)$$



$$\Box \neg (G_1 \wedge G_2)$$



Defining Fitness Measures

Q1 Assign values to behaviors

Q2 Assign values to models

Q3 Assign values (“distances”) to pairs of models

Defining Fitness Measures

- Q1 Assign values to behaviors
 Boolean case: correct vs. incorrect behaviors

- Q2 Assign values to models
 Boolean case: automaton = generator of behaviors

- Q3 Assign values (“distances”) to pairs of models
 Boolean case: preorders between automata

Q1 Assign Values To Behaviors

a. Probabilities

Q1 Assign Values To Behaviors

- a. Probabilities
- b. Resource use

Weighted Automata

Q

Σ

$q_0 \in Q$

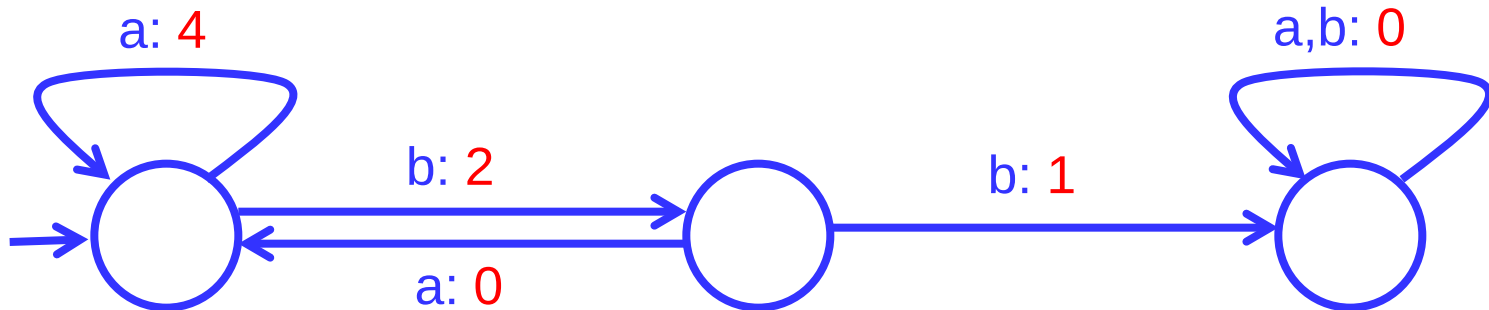
$\delta \subseteq Q \times \Sigma \times \mathbb{R} \times Q$

states

observations (letters)

initial state

transition function



Q1 Assign Values To Behaviors

a. Probabilities

b. Resource use

behavior $x = q_1 a_1 v_1 q_2 a_2 v_2 q_3 a_3 v_3 \dots$

where $q_i \in Q \dots$ states

$a_i \in \Sigma \dots$ observations

$v_i \in \mathbb{R} \dots$ weights (local resource consumption)

value function $\text{val}(x) = \text{val}(v_1 v_2 v_3 \dots) \in \mathbb{R}$

worst case vs. average case vs. accumulative
(e.g. power consumption)

Different Value Functions

Max value: $\text{val}(v_1 v_2 v_3 \dots) = \sup\{ v_i : i \geq 1 \}$
(only 0 and 1 weights: finite automaton)

Limsup value: $\text{val} = \lim_{n \rightarrow \infty} \sup\{ v_i : i \geq n \}$
(only 0 and 1 weights: Buechi automaton)

Weighted Automata

Q

Σ

$q_0 \in Q$

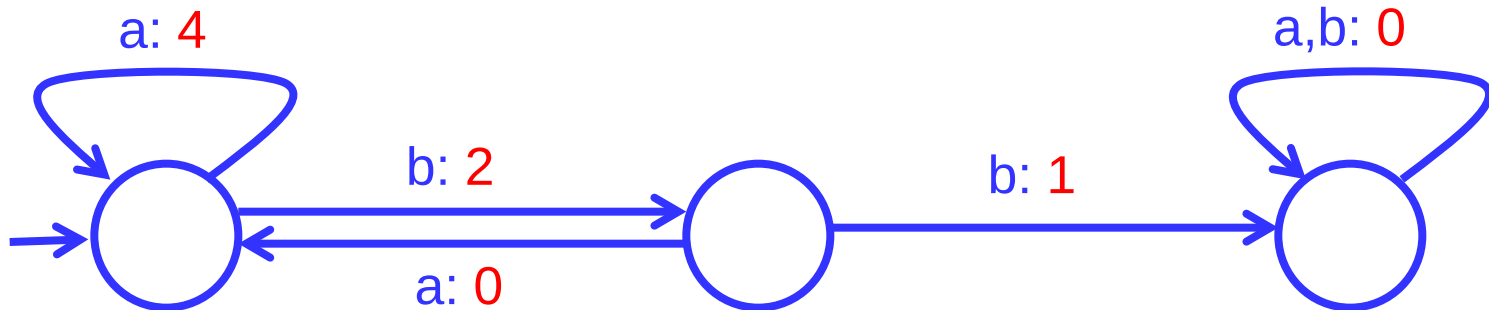
$\delta \subseteq Q \times \Sigma \times \mathbb{R} \times Q$

states

observations (letters)

initial state

transition function



Different Value Functions

Max value: $\text{val}(v_1 v_2 v_3 \dots) = \sup\{ v_i : i \geq 1 \}$
(only 0 and 1 weights: finite condition)

Limsup value: $\text{val} = \lim_{n \rightarrow \infty} \sup\{ v_i : i \geq n \}$
(only 0 and 1 weights: Buchi condition)

Limavg value: $\text{val} = \liminf_{n \rightarrow \infty} 1/n \cdot \sum_{1 \leq i \leq n} v_i$

Weighted Automata

Q

Σ

$q_0 \in Q$

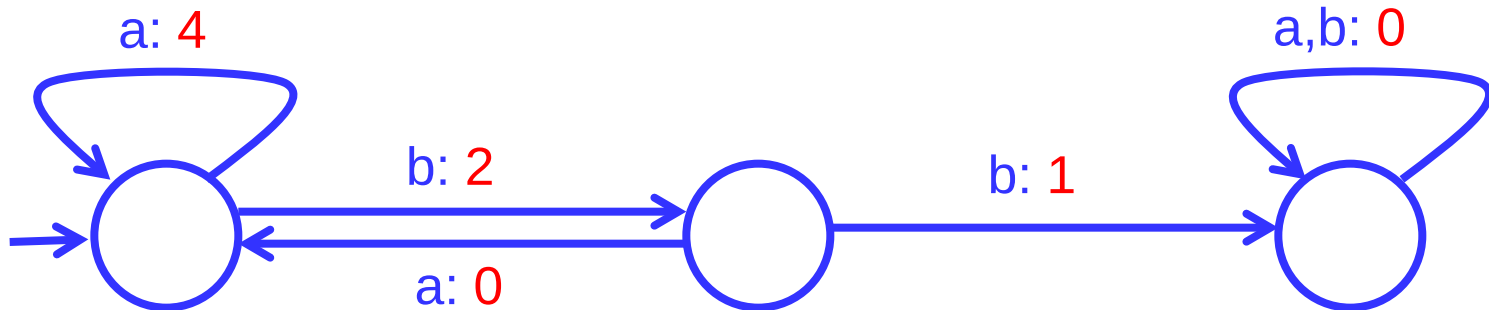
$\delta \subseteq Q \times \Sigma \times \mathbb{R} \times Q$

states

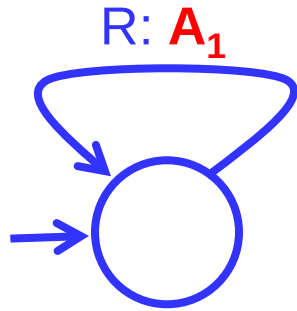
observations (letters)

initial state

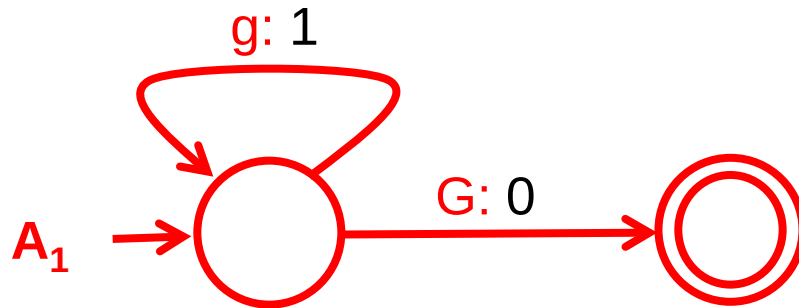
transition function



Example: Average Response Time



Master automaton: limavg



Slave automaton: sum

Different Value Functions

Max value: $\text{val}(v_1 v_2 v_3 \dots) = \sup\{ v_i : i \geq 1 \}$
(only 0 and 1 weights: finite condition)

Limsup value: $\text{val} = \lim_{n \rightarrow \infty} \sup\{ v_i : i \geq n \}$
(only 0 and 1 weights: Buchi condition)

Limavg value: $\text{val} = \liminf_{n \rightarrow \infty} 1/n \cdot \sum_{1 \leq i \leq n} v_i$

Energy value: $\text{val} = \inf\{ v : (\forall n \geq 1) (v \geq \sum_{1 \leq i \leq n} v_i) \}$

$$\text{val}(2, 5, 0, -3, -3, 5, 0, -3, -3, 5, 0, -3, -3, \dots) = 7$$

Different Value Functions

Max value: $\text{val}(v_1 v_2 v_3 \dots) = \sup\{ v_i : i \geq 1 \}$
(only 0 and 1 weights: finite condition)

Limsup value: $\text{val} = \lim_{n \rightarrow \infty} \sup\{ v_i : i \geq n \}$
(only 0 and 1 weights: Buchi condition)

Limavg value: $\text{val} = \liminf_{n \rightarrow \infty} 1/n \cdot \sum_{1 \leq i \leq n} v_i$

Energy value: $\text{val} = \inf\{ v : (\forall n \geq 1) (v \geq \sum_{1 \leq i \leq n} v_i) \}$

Discounting: $\text{val} = \sum_{i \geq 1} d^i \cdot v_i$ for some $0 < d < 1$

Q1 Assign Values To Behaviors

a. Probabilities

b. Resource use

worst case vs. average case vs. accumulative
(e.g., response time, power consumption)

c. Quality measures

discounting vs. long-run averaging

Reliability

a: ok

b: fail

Discounted value ($0 < d < 1$): □ a

aaaaaaaaaaa...

1

aaaaaaaaab...

$1 - d^8$

aaab...

$1 - d^3$

b...

0

Reliability

a: ok

b: fail

Discounted value ($0 < d < 1$): $\square a$

aaaaaaaaaaa...

1

aaaaaaaaab...

$1 - d^8$

aaab...

$1 - d^3$

b...

0

Long-run average value:

$\limavg a$

aaaaaaaaaaa...

1

abaabaaab...

1

aaabaaabaaab...

$3/4$

babbabbba...

0

aaaaaabbba...

0

Q2 Assign Values To Models

behavior
word
language

$$x = q_1 a_1 v_1 q_2 a_2 v_2 \dots$$

$$w = a_1 a_2 a_3 \dots \in \Sigma^\omega$$

$$L: \Sigma^\omega \rightarrow \mathbb{B}$$

“boolean language”

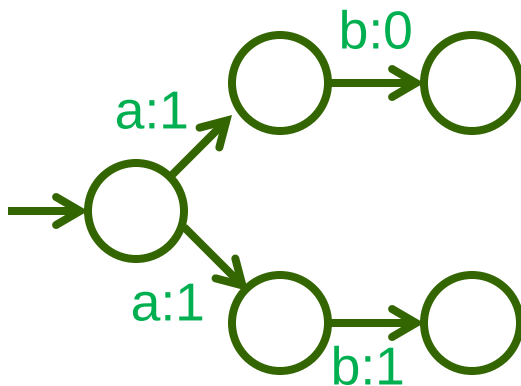
$$A: \Sigma^\omega \rightarrow \mathbb{R}$$

“quantitative language”

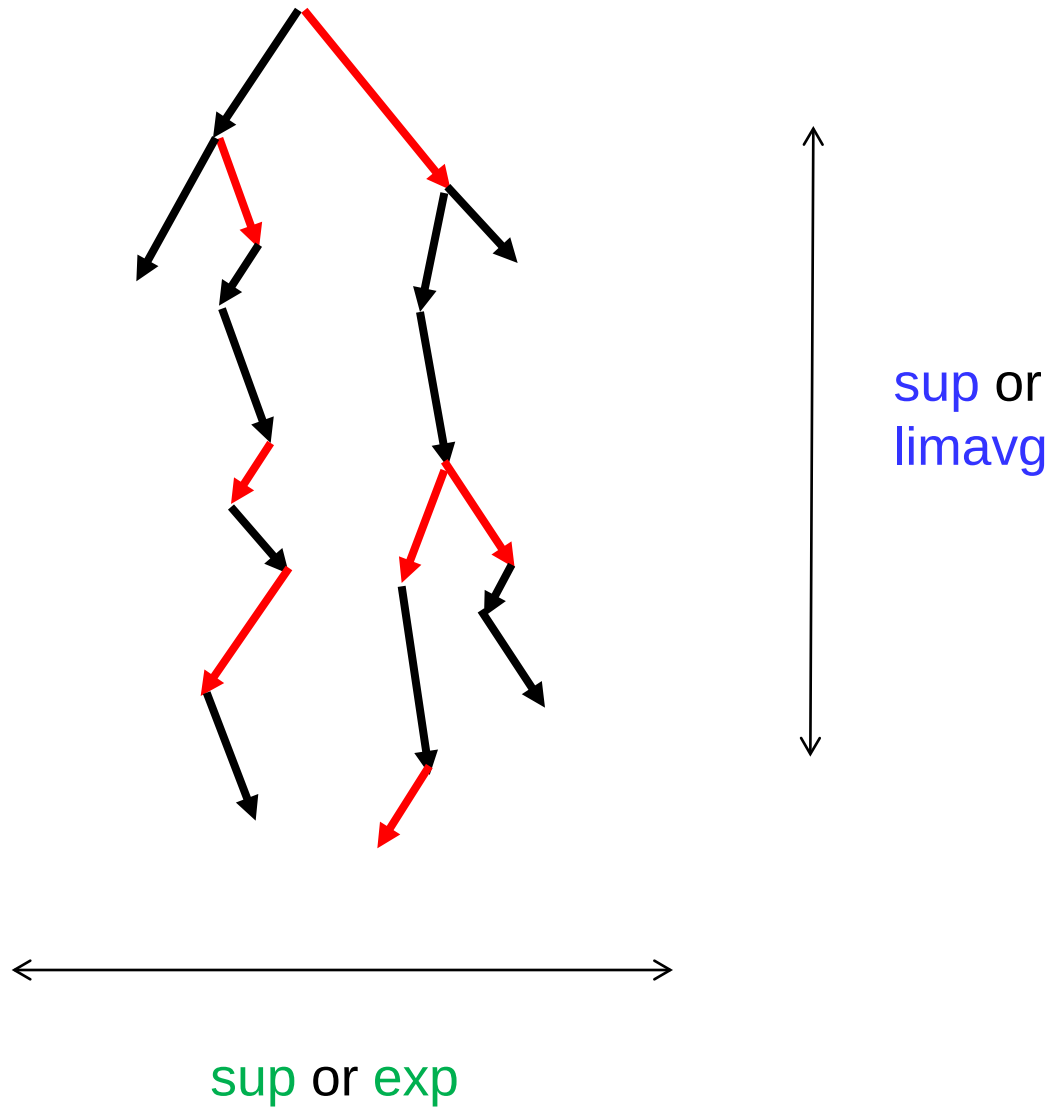
Q2 Assign Values To Models

behavior $x = q_1 a_1 v_1 q_2 a_2 v_2 \dots$
word $w = a_1 a_2 a_3 \dots \in \Sigma^\omega$
language $A: \Sigma^\omega \rightarrow \mathbb{R}$

$$A(w) = \sup_x \{ \text{val}(x) : \text{obs}(x) = w \}$$



$$A(ab) = \max(\text{val}(1,1), \text{val}(1,0)) = 1$$

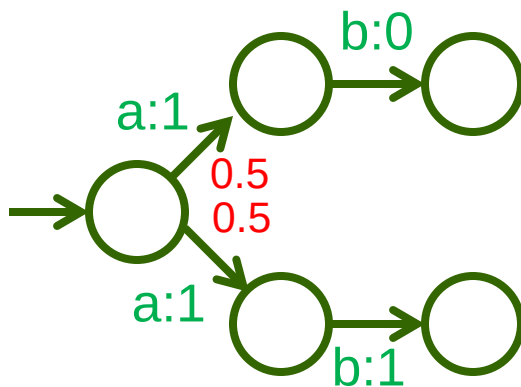


Q2 Assign Values To Models

behavior $x = q_1 a_1 v_1 q_2 a_2 v_2 \dots$
word $w = a_1 a_2 a_3 \dots \in \Sigma^\omega$
languages $A, B: \Sigma^\omega \rightarrow \mathbb{R}$

$A(w) = \sup_x \{ \text{val}(x) : \text{obs}(x) = w \}$ worst case

$B(w) = \exp_x \{ \text{val}(x) : \text{obs}(x) = w \}$ avg case



$$B(ab) = \exp(\text{val}(1,1), \text{val}(1,0)) = 0.75$$

Q3 Assign Distances Between Models

behavior $x = q_1 a_1 v_1 q_2 a_2 v_2 \dots$
word $w = a_1 a_2 a_3 \dots \in \Sigma^\omega$
programs $A, B: \Sigma^\omega \rightarrow \mathbb{R}$

$A(w) = \sup_x \{ \text{val}(x) : \text{obs}(x) = w \}$ worst case

$B(w) = \exp_x \{ \text{val}(x) : \text{obs}(x) = w \}$ avg case

$\text{diff}(A, B) = \sup_w \{ |A(w) - B(w)| \}$

Correctness = Language Inclusion

Given two models A and B, is $L(A) \subseteq L(B)$?

For finite and Buchi automata: deterministic B or PSPACE.

Correctness = Language Inclusion

Given two models A and B, is $L(A) \subseteq L(B)$?

i.e. $\forall w \in \Sigma^\omega : L(A)(w) \leq L(B)(w)$

For finite and Buchi automata: deterministic B or PSPACE.

Correctness = Language Inclusion

Given two models A and B, is $L(A) \subseteq L(B)$?

i.e. $\forall w \in \Sigma^\omega : L(A)(w) \leq L(B)(w)$

For finite and Buchi automata: deterministic B or PSPACE.

For max and limsup automata: same.

For limavg and energy automata: functional B or undecidable.

For discounted automata: open.

Computing Distances

Linear time (language based): sometimes impossible

Branching time (tree based): solving graph games

Computing Distances

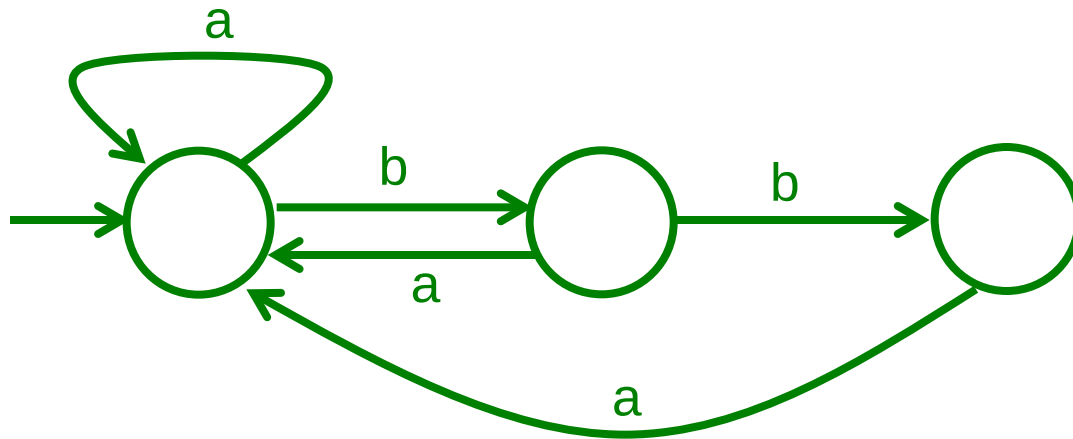
Linear time (language based): sometimes impossible

Branching time (tree based): solving graph games

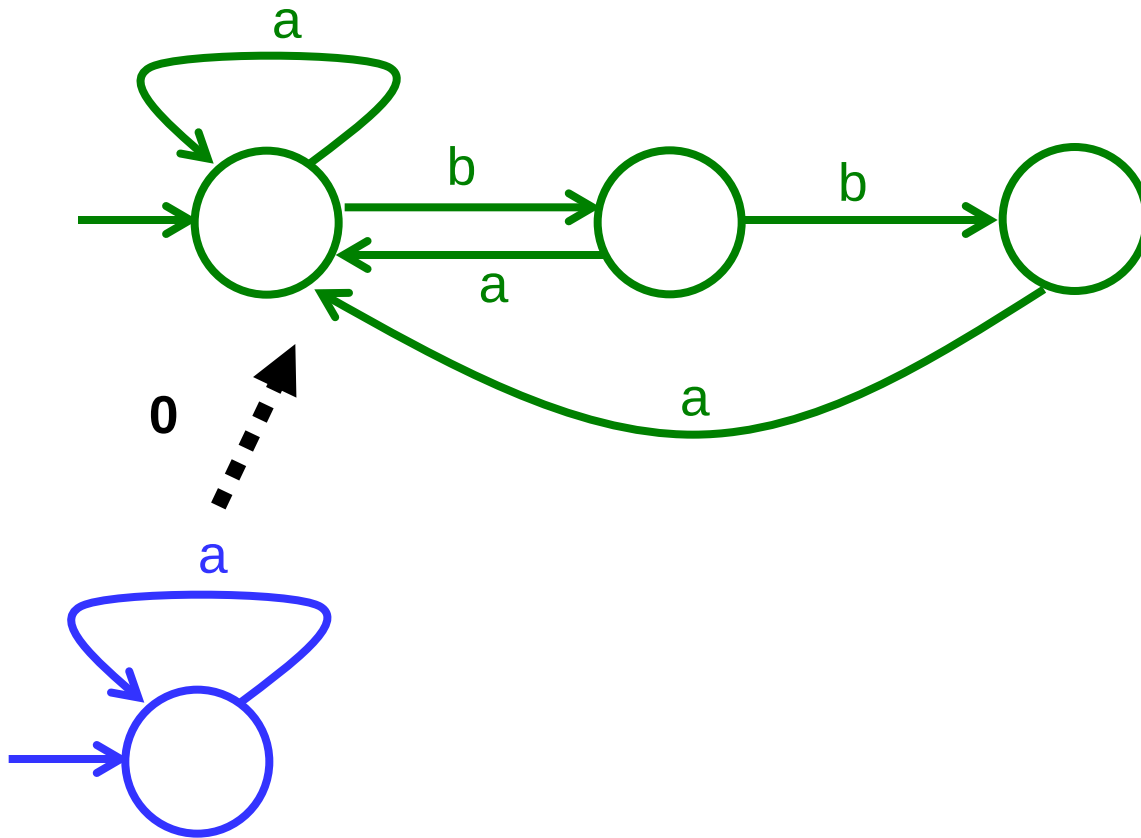
1 Simulation-like preorders

2 Reactive synthesis

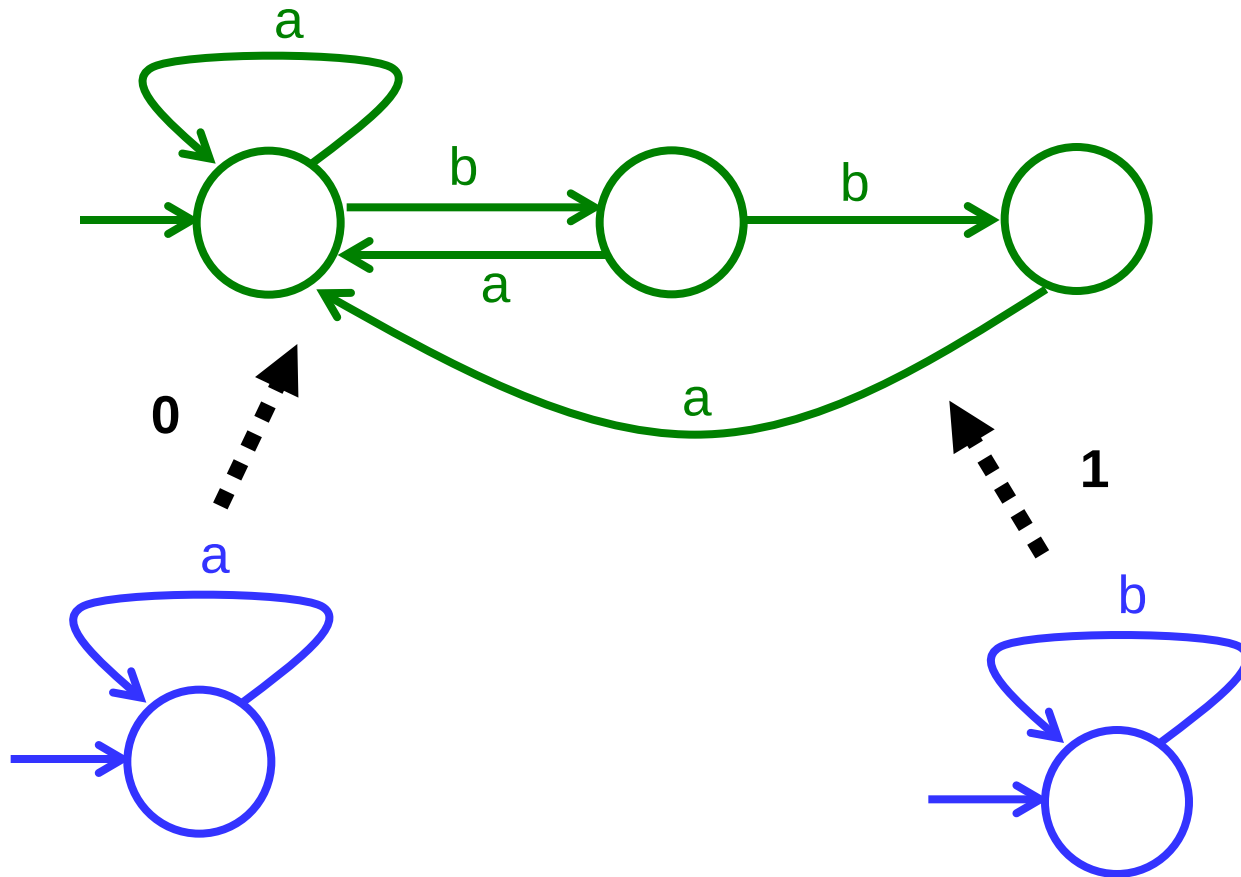
Simulation Preorder



Simulation Preorder



Simulation Preorder



Simulation Game

Player 1:

chooses a transition of the implementation

Player 2:

matches the letter by choosing a transition of the specification

Player 1 wins if Player 2 cannot match:

implementation incorrect with respect to specification

Quantitative Simulation Game for **Incorrect** Implementations

Player 1:
chooses a transition of the implementation

Player 2:
matches the letter by choosing a transition of the specification (weight 0),
or chooses an illegal transition of the specification (weight 1)

Quantitative Simulation Game for **Incorrect** Implementations

Player 1:

chooses a transition of the implementation

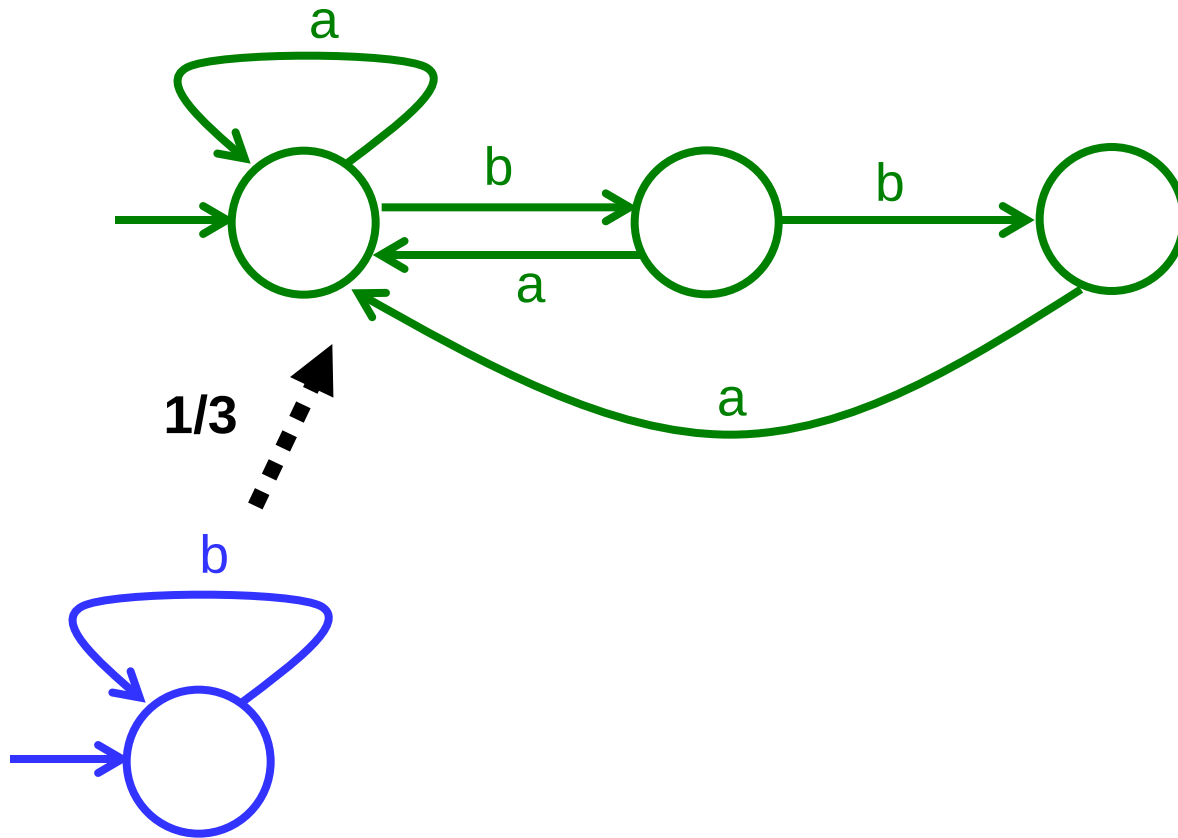
Player 2:

matches the letter by choosing a transition of the specification (weight 0),
or chooses an illegal transition of the specification (weight 1)

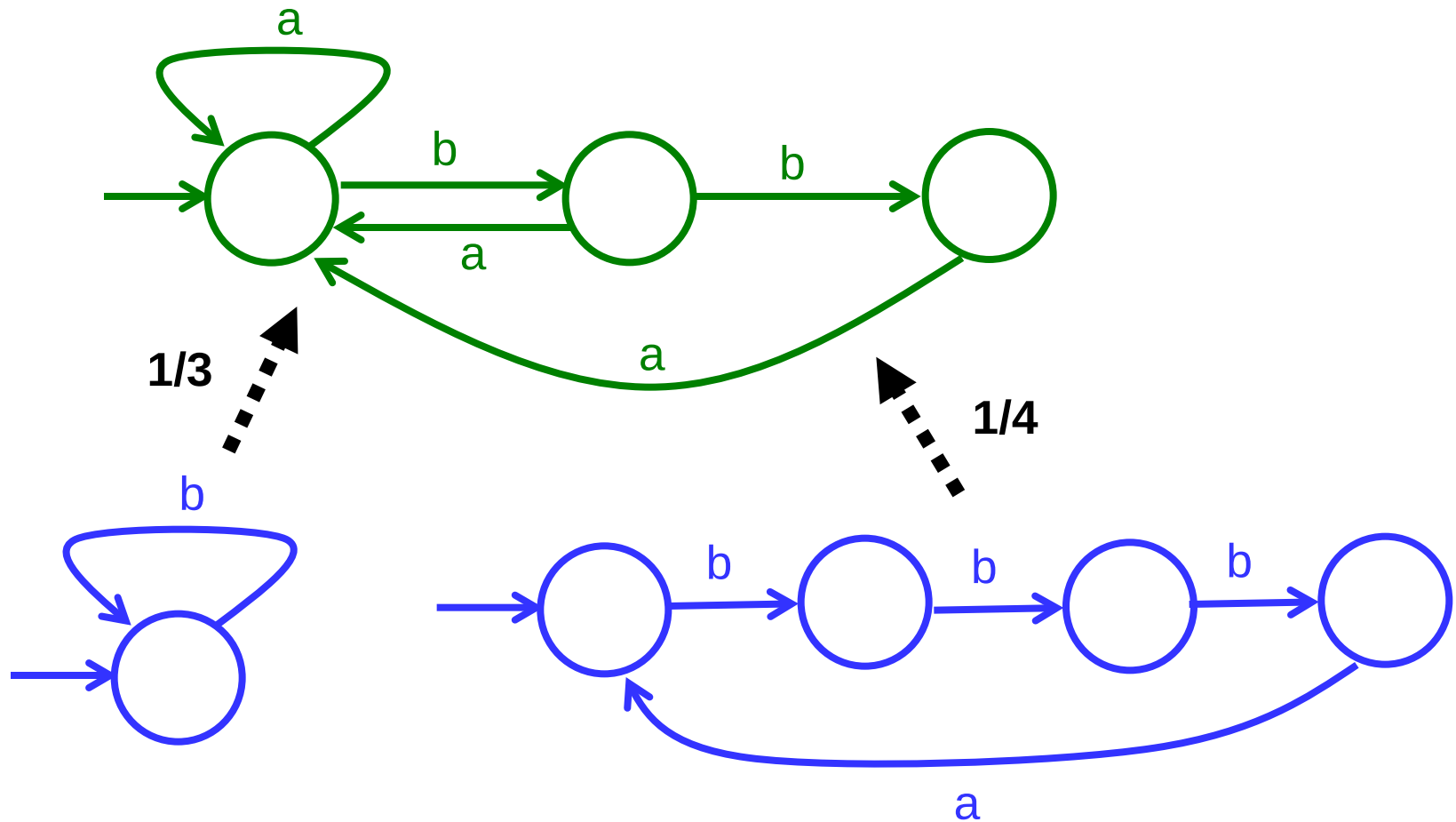
Player 1 tries to make Player 2 choose as many illegal transitions as possible: maximize limavg of weights

The more illegal transitions of the specification are needed to simulate the implementation, the greater the distance.

Simulation Distance



Simulation Distance



Quantitative Simulation Game for **Correct** Implementations

Player 2:

decides if the implementation must choose a legal transition (**weight 0**),
or is allowed to choose an illegal transition (**weight 1**)

Player 1:

chooses a legal transition of the implementation,
or if allowed, chooses an illegal transition of the implementation

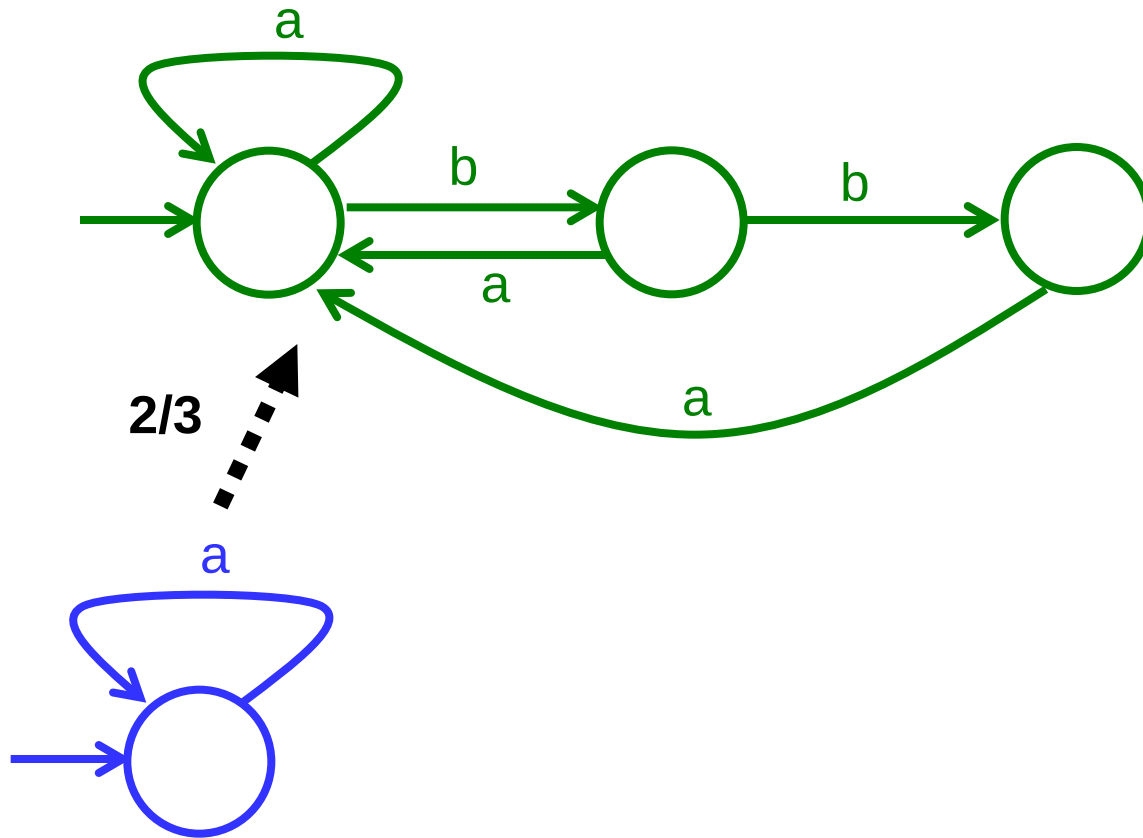
Player 2:

matches the letter by choosing a transition of the specification

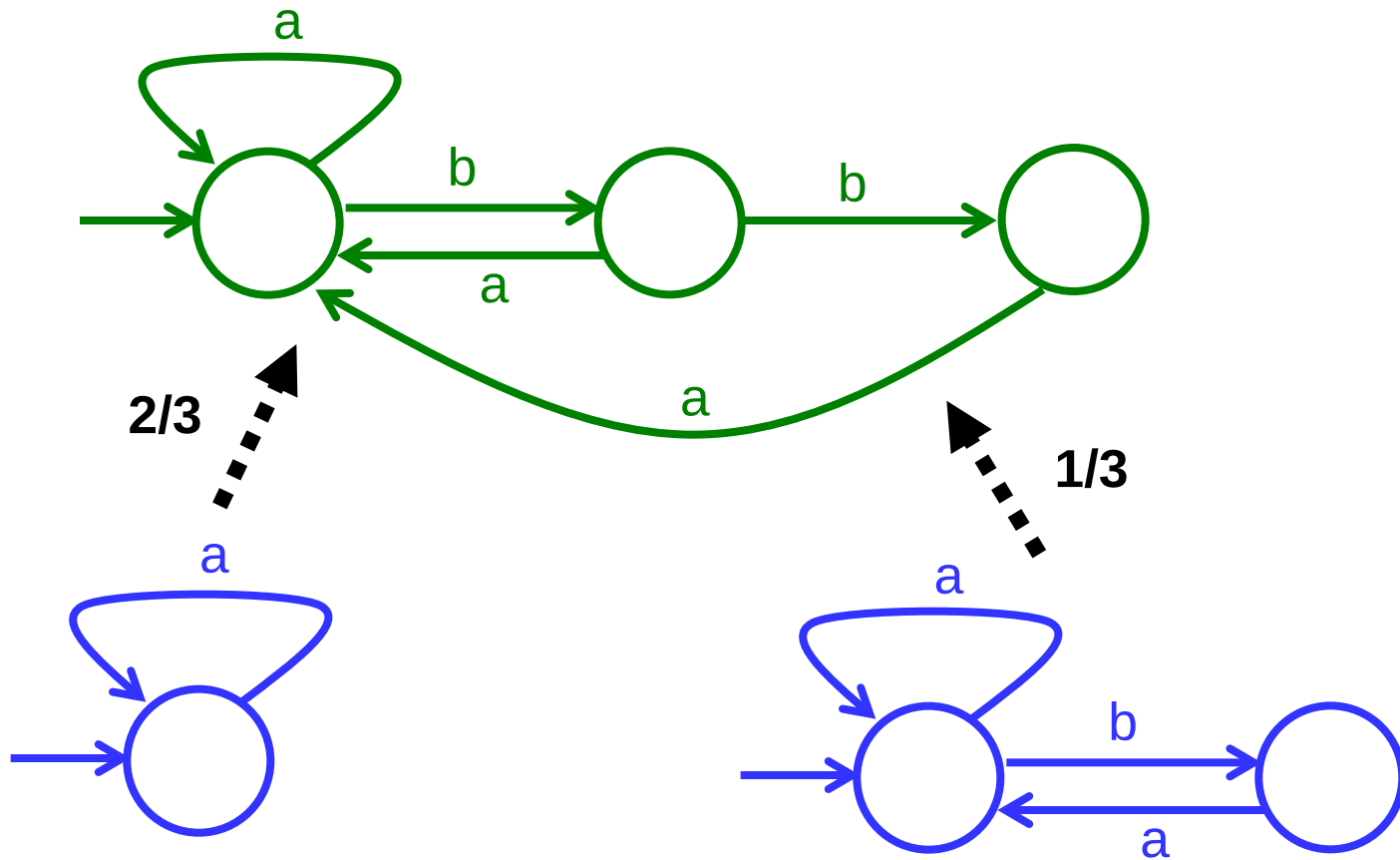
Player 2 tries to allow Player 1 choose as many illegal transitions as possible: maximize limavg of weights

The more illegal transitions of the implementation can be tolerated by the specification, the greater the robustness.

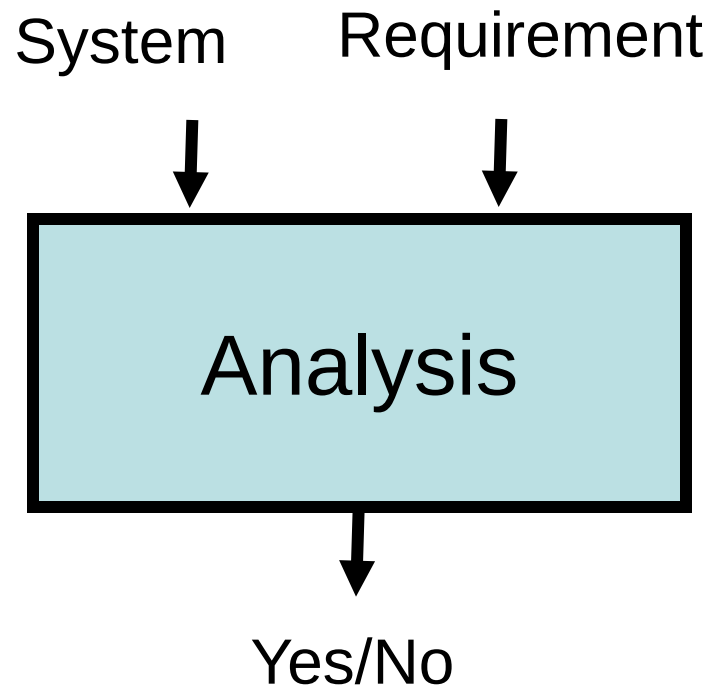
Robustness Distance



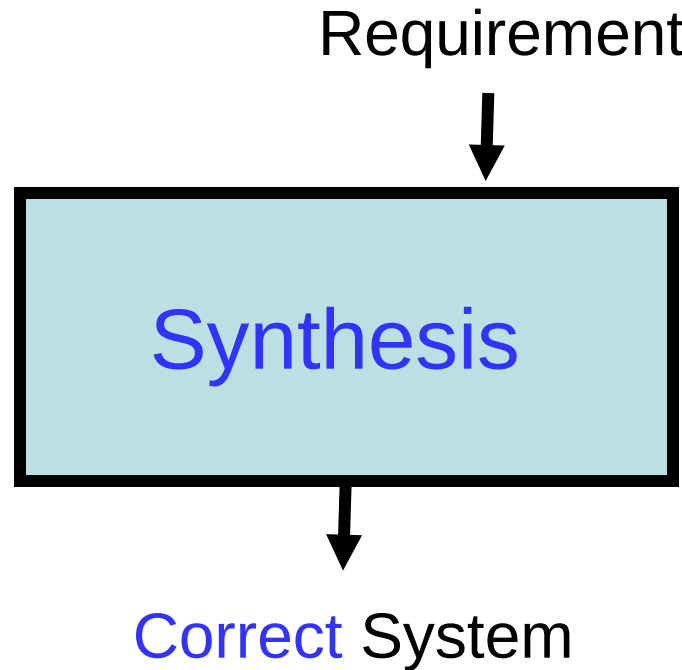
Robustness Distance



Formal Verification



Automatic Programming



Reactive Synthesis

Temporal
Formula

$$\Box (R \Rightarrow \Diamond G)$$

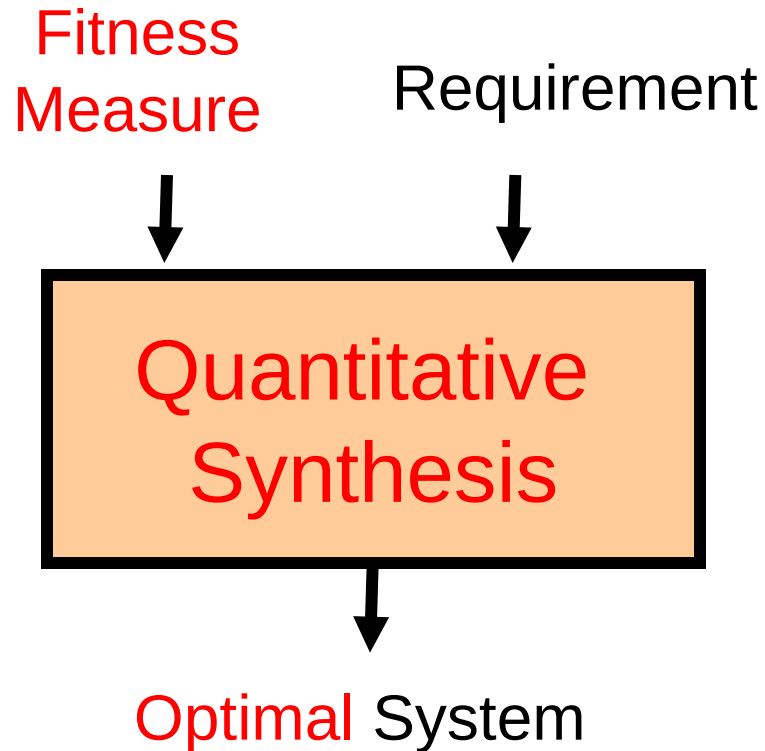


Graph Game with
Boolean Objective

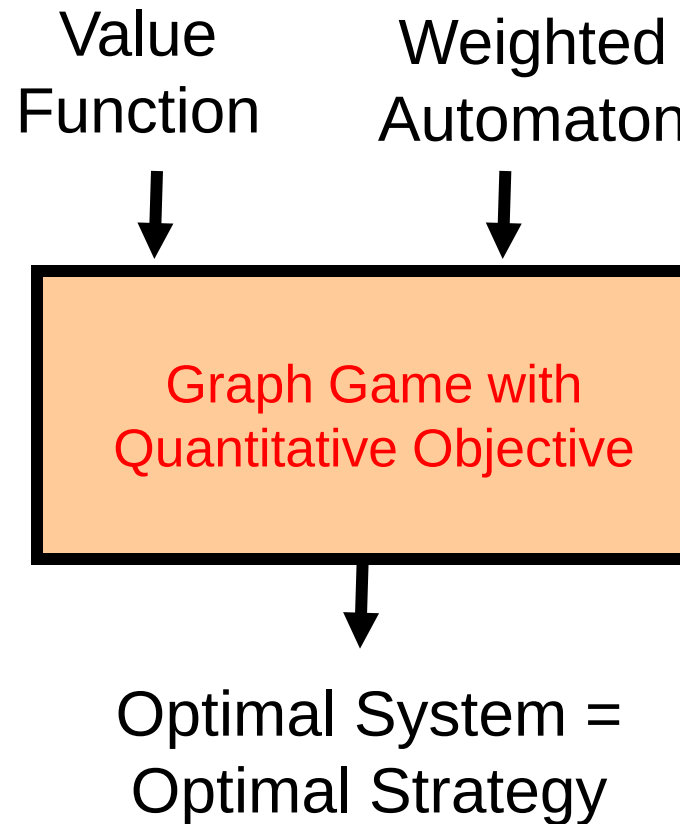


Correct System =
Winning Strategy

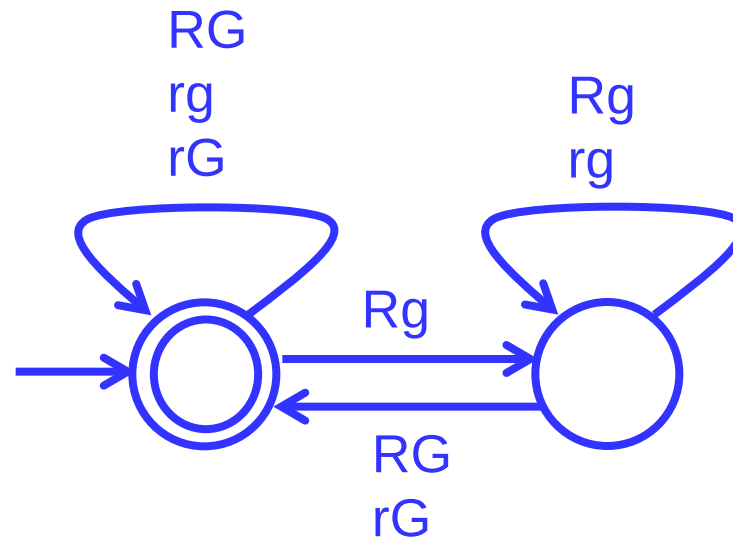
Not all solutions are equally desirable ...



Quantitative Synthesis

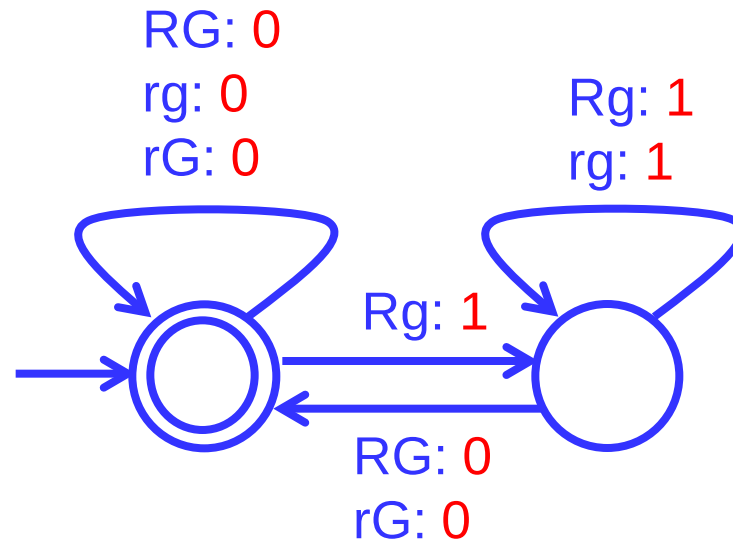


Request-Grant Buchi Automaton



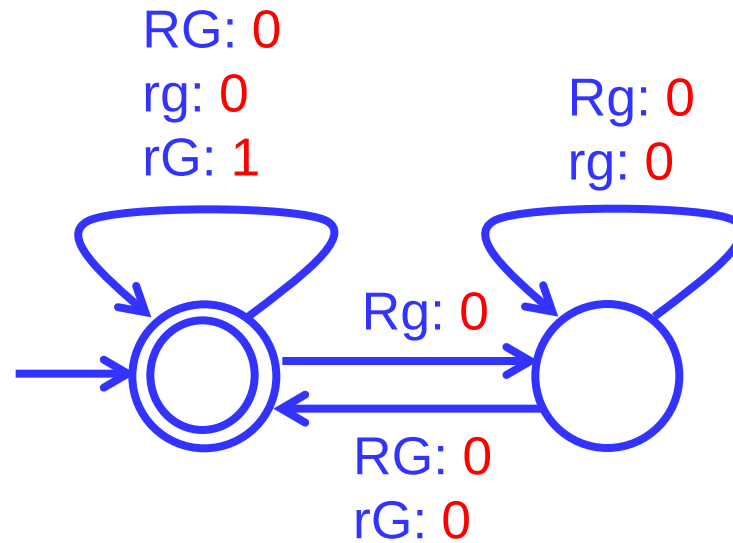
Every request is followed by a grant.

Request-Grant limavg Automaton 1



Following a request, all steps until the next grant are penalized.

Request-Grant limavg Automaton 2



All unnecessary grants are penalized.

Partial Program Synthesis

(Bodik & Solar-Lezama):

given a nondeterministic program sketch
(a program with “holes”)

and a set of requirements
(safety and liveness),

find a deterministic program completion
that satisfies the requirements.

Practical relevance (OSDI, SOSPP).

Concurrent Program Sketch

Thread P_1 :

init $x := 0$;

loop

choice

| $x := x+1 \bmod 2$

| $x := 0$

end choice

end loop.

Thread P_2 :

init $y := 0$;

loop

choice

| $y := x$

| $y := x+1 \bmod 2$

end choice

end loop.

Analysis Questions

$$\forall x (x \geq y)$$

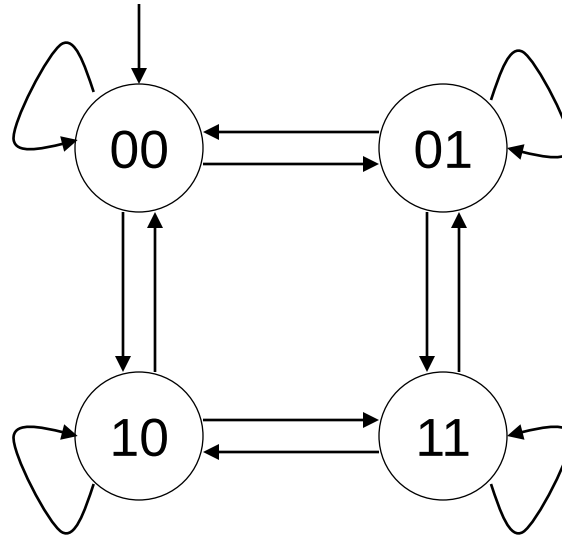
$$\exists x (x \geq y)$$

Analysis Questions

X $\forall \square (x \geq y)$

✓ $\exists \square (x \geq y)$

Model
checking



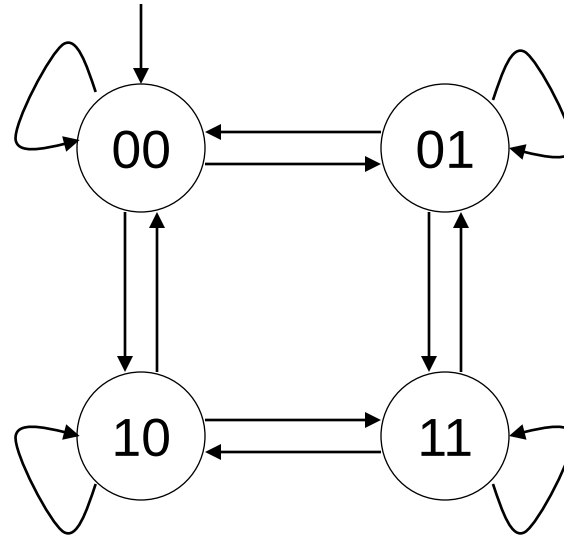
Synthesis Questions

X $\forall \square (x \geq y)$

✓ $\exists \square (x \geq y)$

$\langle\langle P_1 \rangle\rangle \square (x \geq y)$

$\langle\langle P_2 \rangle\rangle \square (y = 0)$

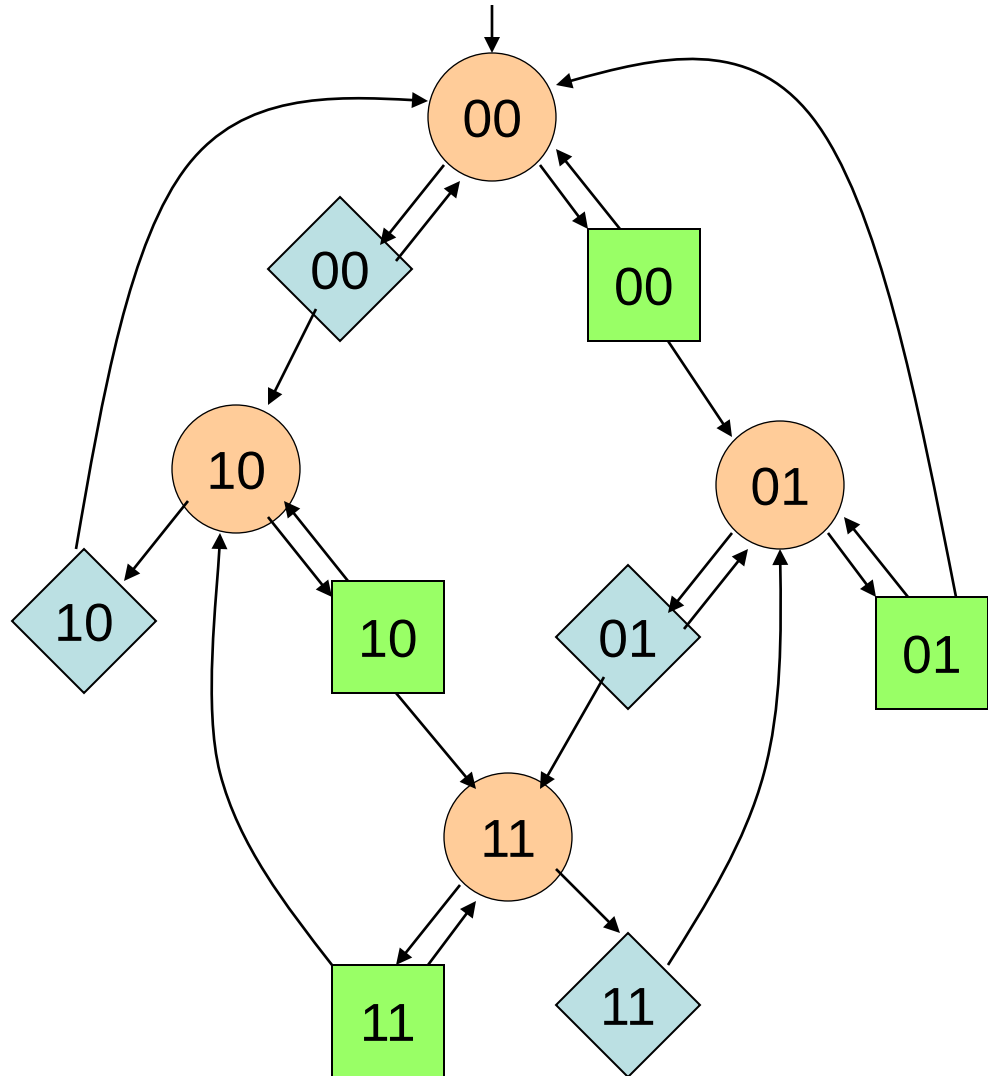


Synthesis Questions

Computing
winning strategies
in a graph game

X $\langle\langle P_1 \rangle\rangle \square (x \geq y)$

✓ $\langle\langle P_2 \rangle\rangle \square (y = 0)$



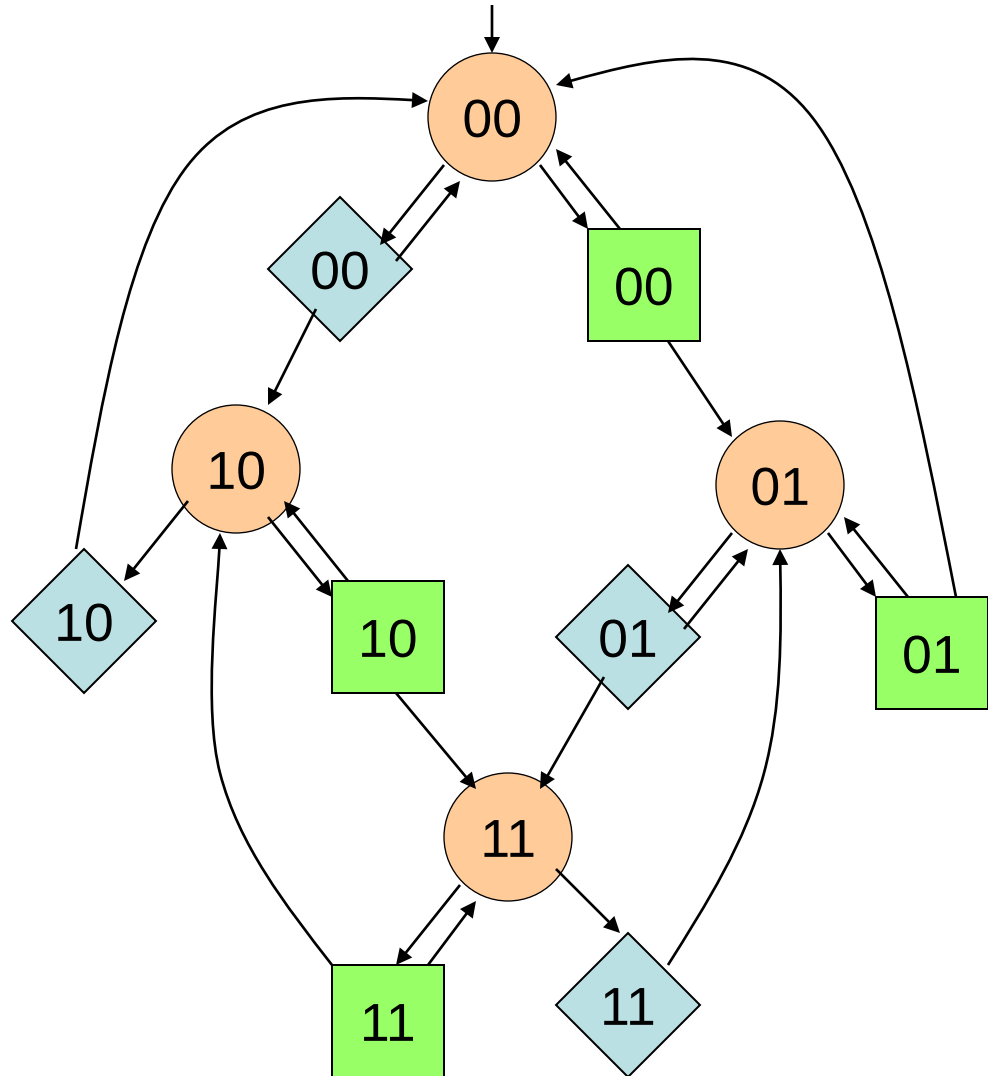
Co-Synthesis Question

$\langle\langle P_1 \rangle\rangle \square (x \geq y)$



$\langle\langle P_2 \rangle\rangle \square (y = 0)$

Computing
equilibria
in a graph game



Example: Mutual Exclusion

```
while( true ) {  
    flag[1] := true; turn := 2;
```

choice

```
| while( flag[1] ) nop;  
| while( flag[2] ) nop;  
| while( turn=1 ) nop;  
| while( turn=2 ) nop;  
| while( flag[1] & turn=2 ) nop;  
| while( flag[1] & turn=1 ) nop;  
| while( flag[2] & turn=1 ) nop;  
| while( flag[2] & turn=2 ) nop;  
end choice;
```

```
CS1 := true; fin_wait;  
CS1 := false; arbitrary_wait;  
}
```

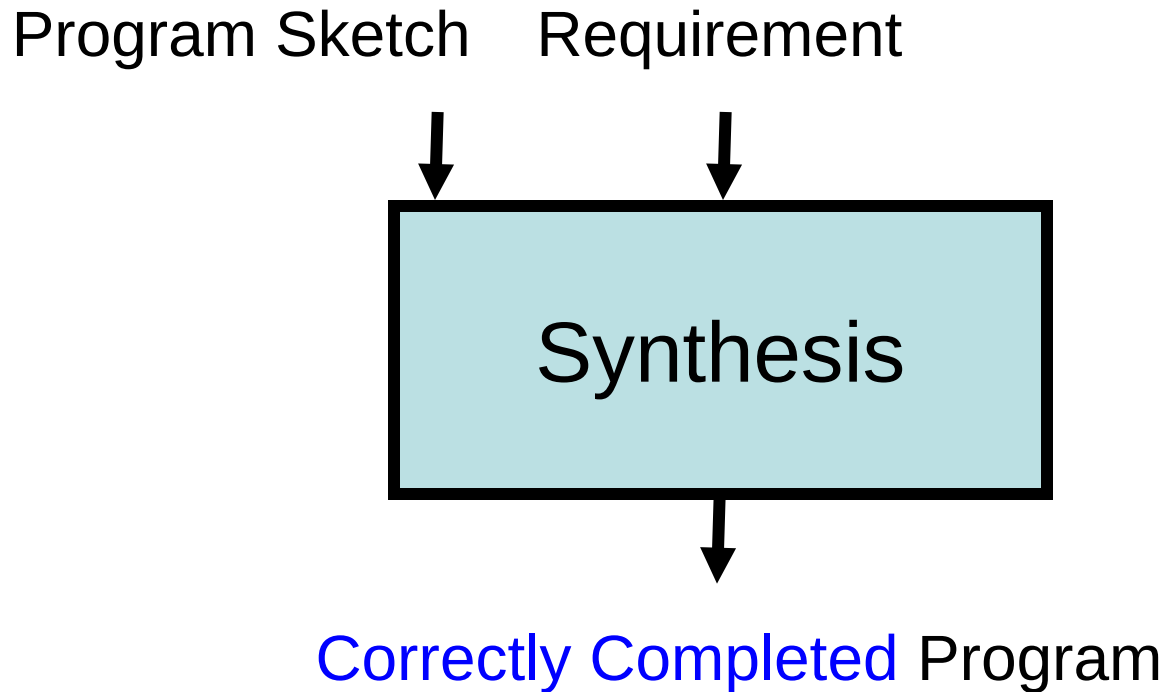
```
while( true ) {  
    flag[2] := true; turn :=1;
```

choice

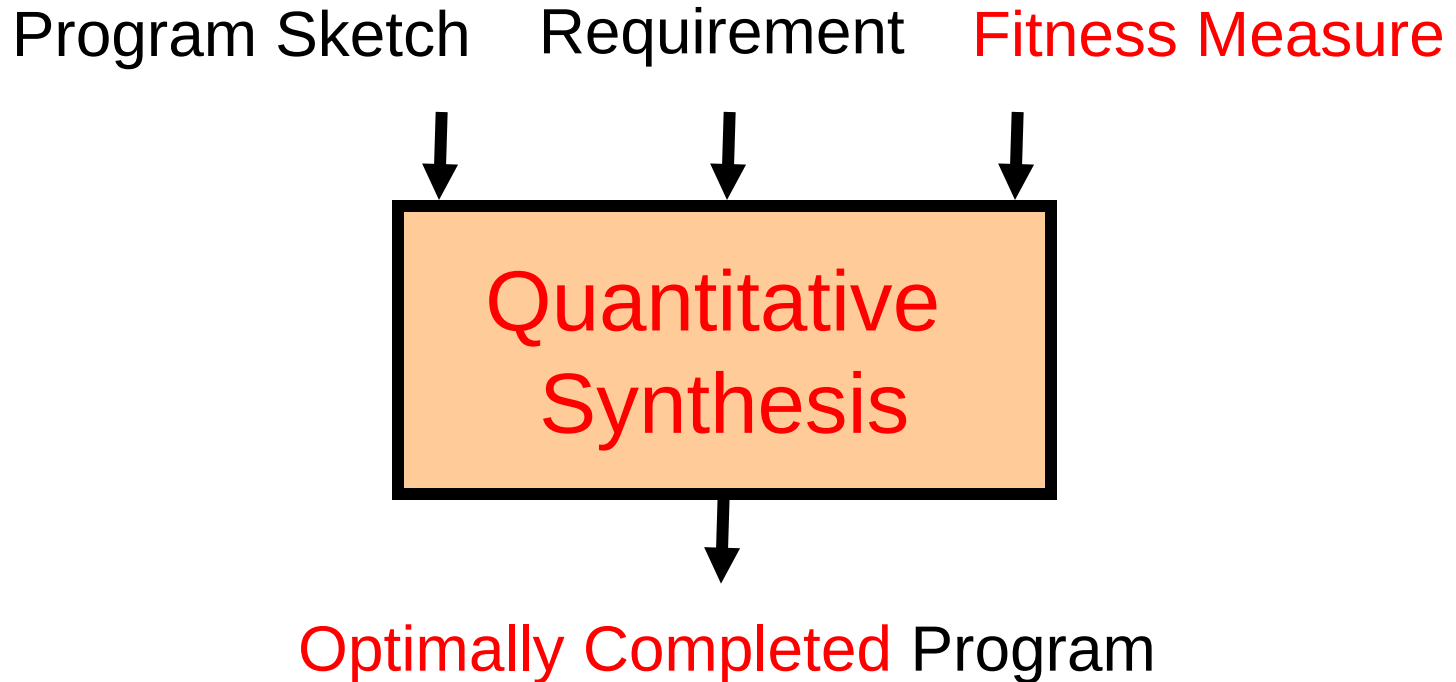
```
| while( flag[1] ) nop;  
| while( flag[2] ) nop;  
| while( turn=1 ) nop;  
| while( turn=2 ) nop;  
| while( flag[1] & turn=2 ) nop;  
| while( flag[1] & turn=1 ) nop;  
| while( flag[2] & turn=1 ) nop;  
| while( flag[2] & turn=2 ) nop;  
end choice;
```

```
CS2 := true; fin_wait;  
CS2 := false; arbitrary_wait;  
}
```

Partial Program Synthesis



Computer-Aided Programming



Example: Lock Placement

-given sequentially correct program

-add locks so that the program is safe for concurrent execution and performs optimally among all solutions

-weighted automaton to model performance on specific architecture

Thread P_1 :

```
loop
access x;
access y
end loop.
```

Thread P_2 :

```
loop
access x;
access y
end loop.
```

Fine grained vs. coarse grained locks:

-fine grained locks allow more interleavings,
and therefore cause less waiting of threads

-coarse grained locks cause fewer context switches,
which are expensive

Thread P_1 :

loop

access x;

access y

end.

Thread P_2 :

loop

access x;

access y

end.

Fine grained vs. coarse grained locks:

-fine grained locks allow more interleavings,
and therefore cause less waiting of threads

-coarse grained locks cause fewer context switches,
which are expensive

Thread P_1 :

loop

access x;

access y

end.

Thread P_2 :

loop

access x;

access y

end.

Conclusions

- We can move from program correctness criteria to program preference metrics.

Conclusions

- We can move from program correctness criteria to program preference metrics.
- Fitness metrics for programs can measure time and probabilities, but also resource use, reliability, robustness, etc.

Conclusions

- We can move from program correctness criteria to program preference metrics.
- Fitness metrics for programs can measure time and probabilities, but also resource use, reliability, robustness, etc.
- Nested weighted automata over infinite words can define fitness measures.

Conclusions

- We can move from program correctness criteria to program preference metrics.
- Fitness metrics for programs can measure time and probabilities, but also resource use, reliability, robustness, etc.
- Nested weighted automata over infinite words can define fitness measures.
- Graph games with quantitative objectives offer algorithmic solutions for computer-aided programming.

Some References

- 1 Simulation distances
[CONCUR 2010]
- 2 Nested weighted automata
[LICS 2015]
- 3 Lock placement
[CAV 2015]
- 4 Co-synthesis
[TACAS 2007]