PROGRAM SYNTHESIS

RUZICA PISKAC
YALE UNIVERSITY

Oregon Programming Languages
Summer School 2023

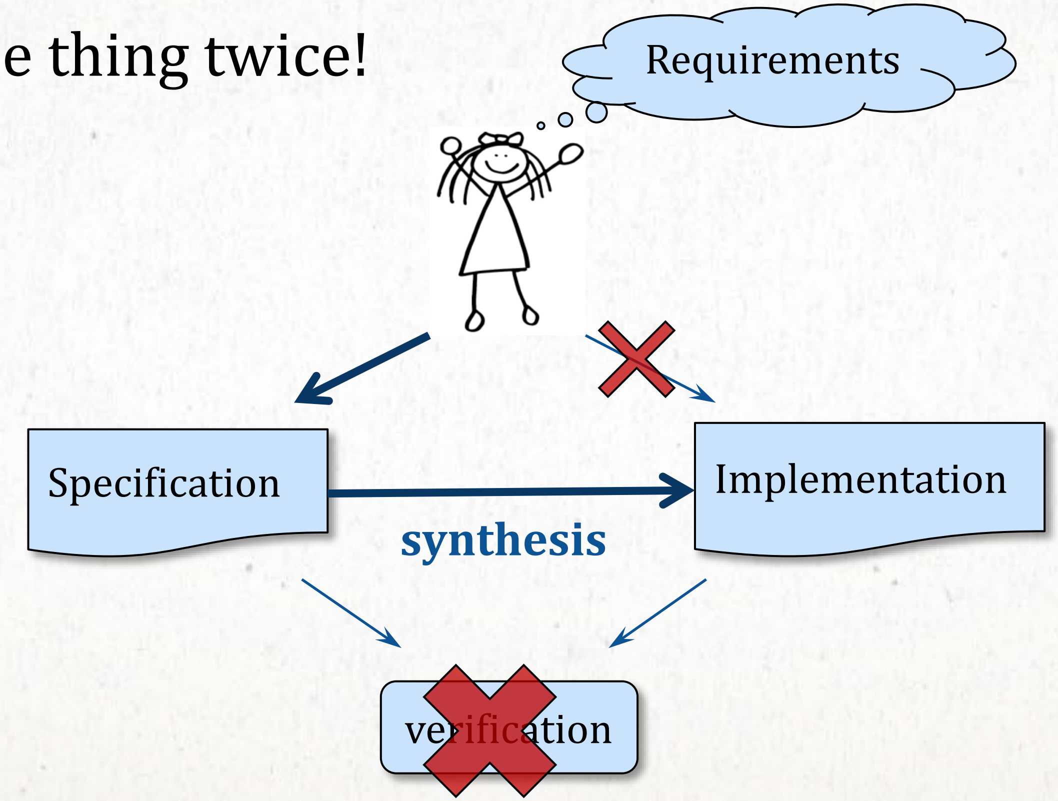
ACKNOWLEDGMENTS

- Based liberal theft of ideas and reuse of slides from **Rajeev Alur**, **Roderick Bloem**, Krishnendu Chatterjee, Ruediger Ehlers, **Bernd Finkbeiner**, **Priyanka Golia,** Andreas Griesmayer, Tom Henzinger, Georg Hofferek, Swen Jacobs, Barbara Jobstmann, Ayrat Khalimov, Bettina Koenighofer, Robert Koenighofer, Andreas Morgenstern, Nir Piterman, Amir Pnueli, Yaniv, Sa'ar, Swen Schewe, Klaus Schneider, Armando Solar-Lezama, Stefan Staber, **Emina Torlak**, Moshe Vardi, and many others

- Synthesizing good synthesis slides is a group effort

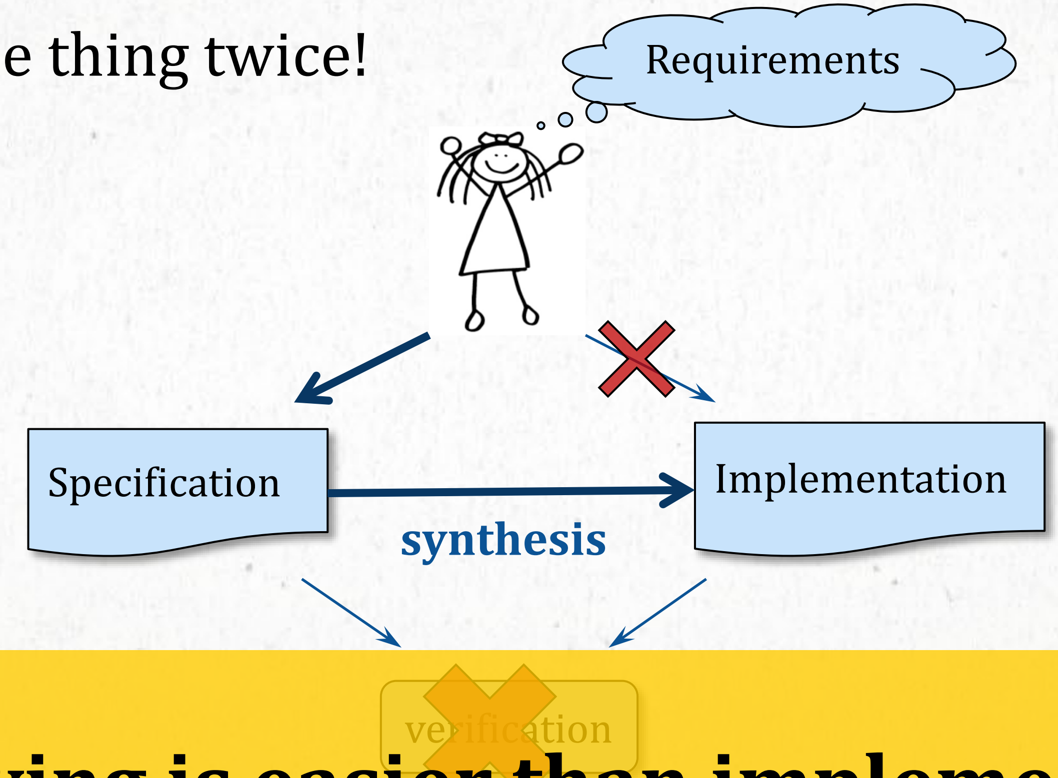
CONSTRUCT CORRECT SYSTEMS AUTOMATICALLY FROM SPEC

Don't do same thing twice!



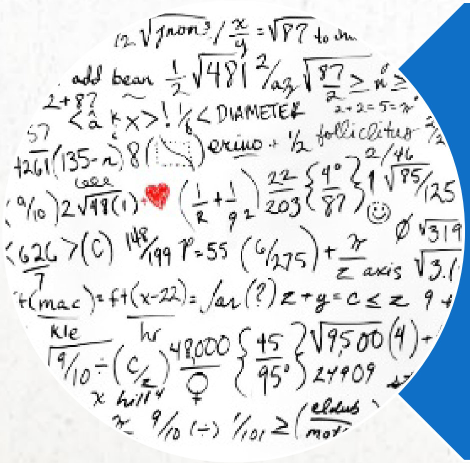
CONSTRUCT CORRECT SYSTEMS AUTOMATICALLY FROM SPEC

Don’t do same thing twice!



Specifying is easier than implementing!

Removes need to Code!

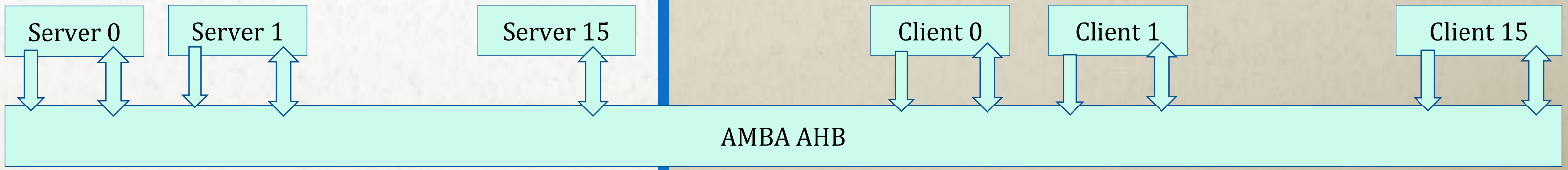


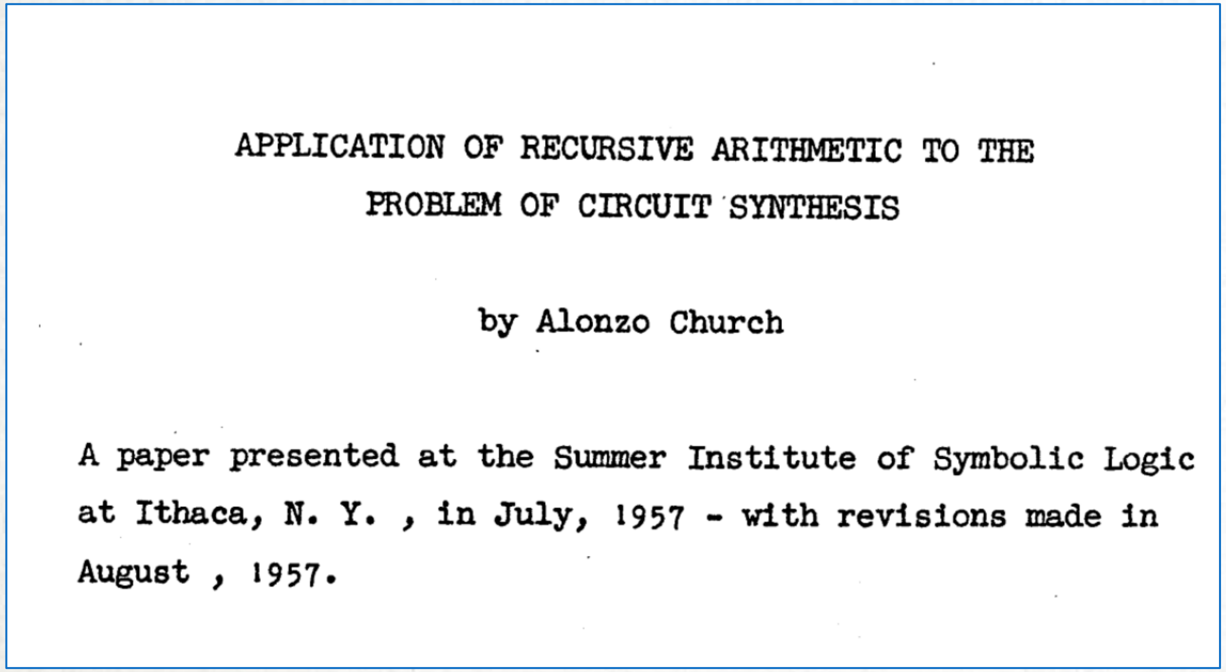
Correct by Construction!

THIS TUTORIAL

- Reactive synthesis – from Church's synthesis problem to scalable software
- Deductive synthesis – from the seminar Manna-Waldinger paper to scalable software
- Can reactive and deductive synthesis be friends?
- Syntax-guided synthesis
- New applications of software synthesis

REACTIVE SYNTHESIS



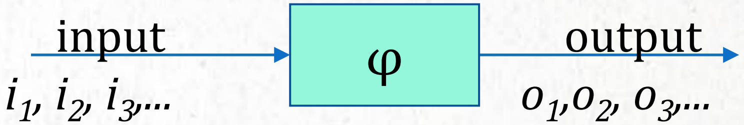


APPLICATION OF RECURSIVE ARITHMETIC TO THE PROBLEM OF CIRCUIT SYNTHESIS

by Alonzo Church

A paper presented at the Summer Institute of Symbolic Logic at Ithaca, N. Y. , in July, 1957 - with revisions made in August , 1957.

- Reactive systems: embedded systems, GUIs, robots, hardware circuits, …
- Church synthesis problem (1957):
  - Given a requirement φ on the input-output behavior of a Boolean circuit, compute a circuit C that satisfies φ.



- Reactive synthesis: given a specification written in LTL (linear temporal logic), automatically compute the program that satisfies the specification

TODAY'S LECTURE

Finite State Reactive systems

- Continuous interaction with environment
- Do not terminate
- Discrete time
- Correctness statements are temporal (temporal logic, automata)

Tomorrow's lecture: functions

- Start with input, terminate with output (non-termination = bug)
- Correctness is input/output relation (Hoare logic)

SYNTHESIS

Given

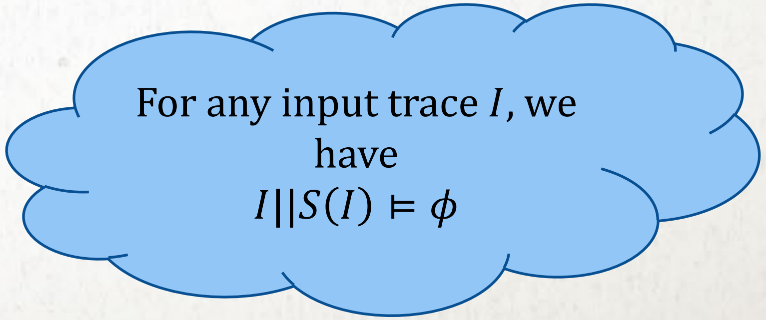
Input and output signals

Specification ϕ of behavior

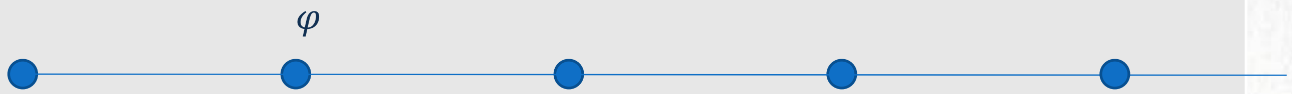
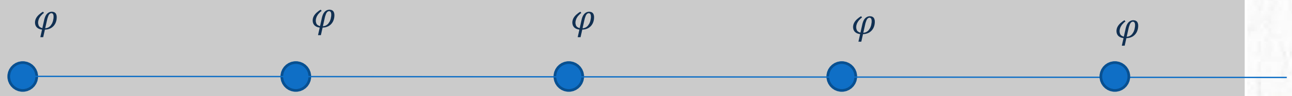
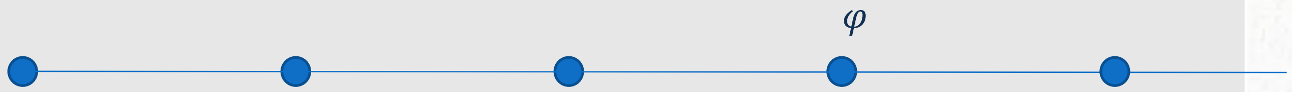
Determine

Realizability: Is there a system that realizes specification?

Synthesis: If system exists, **construct** it



LINEAR TEMPORAL LOGIC (LTL)

LTL	PSL	Meaning
p	p	
$X\varphi$	next φ	
$G\varphi$	always φ	
$F\varphi$	eventually! φ	
$\varphi\ U\ \psi$	φ *until!* ψ	

plus Boolean connectors ($\vee$, $\wedge$, $\neg$, $\rightarrow$) and nesting

LTL SYNTAX

- If φ is an atomic propositional formula, it is a formula in LTL
- If φ and ψ are LTL formulas, so are $\varphi \wedge \psi$, $\varphi \vee \psi$, $\neg \varphi$, $\varphi \,\mathrm{U}\, \psi$ (until), X φ (next), Fφ (eventually), G φ (always)
- Interpretation: over computations $\pi\!: \omega \Rightarrow 2^V$ which assigns truth values to the elements of V at each time instant

$\pi \vDash \mathrm{X}\,\varphi$ iff $\pi^1 \vDash \varphi$

$\pi \vDash \mathrm{G}\,\varphi$ iff $\forall i \cdot \pi^i \vDash \varphi$

$\pi \vDash \mathrm{F}\varphi$ iff $\exists i \cdot \pi^i \vDash \varphi$

$\pi \vDash \varphi\,\mathrm{U}\,\psi$ iff $\exists i \cdot \pi^i \vDash \psi \wedge \forall j \cdot 0 \leq j < i \Rightarrow \pi^j \vDash \varphi$

Here, π^i is the i th state on a path

EXPRESSING PROPERTIES IN LTL

- Good for safety ($G \neg$) and liveness (F) properties
- Express:
  - When a request occurs, it will eventually be acknowledged
    - $G\,(\mathit{request} \Rightarrow F\, \mathit{acknowledge})$
  - A path contains infinitely many q' s
    - $G\,F\,q$
  - At most a finite number of states in a path satisfy $\neg q$ (or property q eventually stabilizes)
    - $F\,G\,q$
  - Action s precedes p after q
    - $[\neg q\ U\ (q \wedge [\neg p\ U\ s])]$
    - Note: hard to do correctly.

SATISFIABILITY & REALIZABILITY

Satisfiability:

Is there a trace that satisfies spec?

Realizability:

Is there a system that satisfies spec?

SATISFIABILITY & REALIZABILITY

Satisfiability: Is there a trace that satisfies the spec?

Realizability: Is there a system that satisfies the spec?


```
input req1, req2

output grant1, grant2

G( (req1 → grant1) ∧ (req2 → grant2) )

G ¬( grant1 ∧ grant2 )
```

req1	F	F	F	F	F	F
req2	F	F	F	F	F	F
grant1	F	F	F	F	F	F
grant2	F	F	F	F	F	F

Satisfiable?

Yes

req1	T	T	T	T	T	T
req2	T	T	T	T	T	T
grant1						
grant2						

Realizable?

No

Inputs universally quantified

SATISFIABILITY & REALIZABILITY

Satisfiability: Is there a trace that satisfies the spec?

Realizability: Is there a system that satisfies the spec?

Realizability $\neq$ Satisfiability

`input req1`

`output grant1`

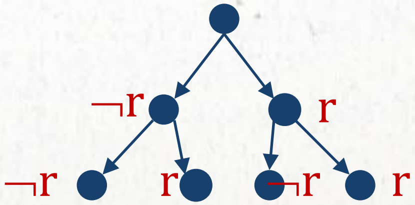
`G( grant1 ↔ X req1 )`

Satisfiable?

Yes (No matter how we set **`grant1`**).

Realizable?

No, clairvoyant!



FORMAL VERIFICATION

Given:

System provides outputs

A specification

One Player: (not a game!)

- Environment provides inputs

System is good if it fulfills the spec **for all possible inputs**

SYNTHESIS IS A GAME

Given:

~~System provides outputs~~

A specification

Two Players (a game!)

- Environment provides inputs
- **System provides outputs**

System is good if it fulfills the spec **for all possible inputs**

REACTIVE SYNTHESIS SETTINGS

Reactive Systems

- Constant interaction
- No Termination
- E.g. Cell phones, Operating Systems, Powerpoint

Finite State

- Non-terminating, finite systems are graphs with loops
- Not our current focus: functions
  - "Create a function that computes sqrt(2)"

EXAMPLE I: CHESS

- Environment determines black moves
- System determines white moves
- Winning condition:
  - If all black moves are legal, then all white moves are legal and eventually, white reaches checkmate

Easy to specify!

CHECKERS AND SYSTEMS

Checkers are passive

Judge if given behavior is OK

Used in verification

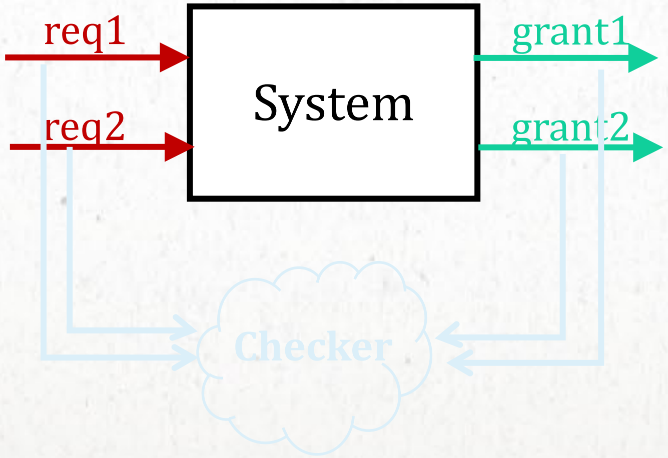
Systems are active

Construct correct behavior

Result of synthesis

Property Synthesis: systems not checkers

Checks moves



Thinks of moves

SYNTHESIS

1. Specify
2. Create Game
3. Solve Game
4. Create System

EXAMPLE II: ARBITER

1. **Specify**
2. Create Game
3. Solve Game
4. Create System

Input: r0, r1

Output: g0, g1

What is the specification?

EXAMPLE II: ARBITER

1. **Specify**
2. Create Game
3. Solve Game
4. Create System

Input: r0, r1

Output: g0, g1

$G(r0 \rightarrow Fg0)$

$G(r1 \rightarrow Fg1)$

$G(\neg g0 \vee \neg g1)$

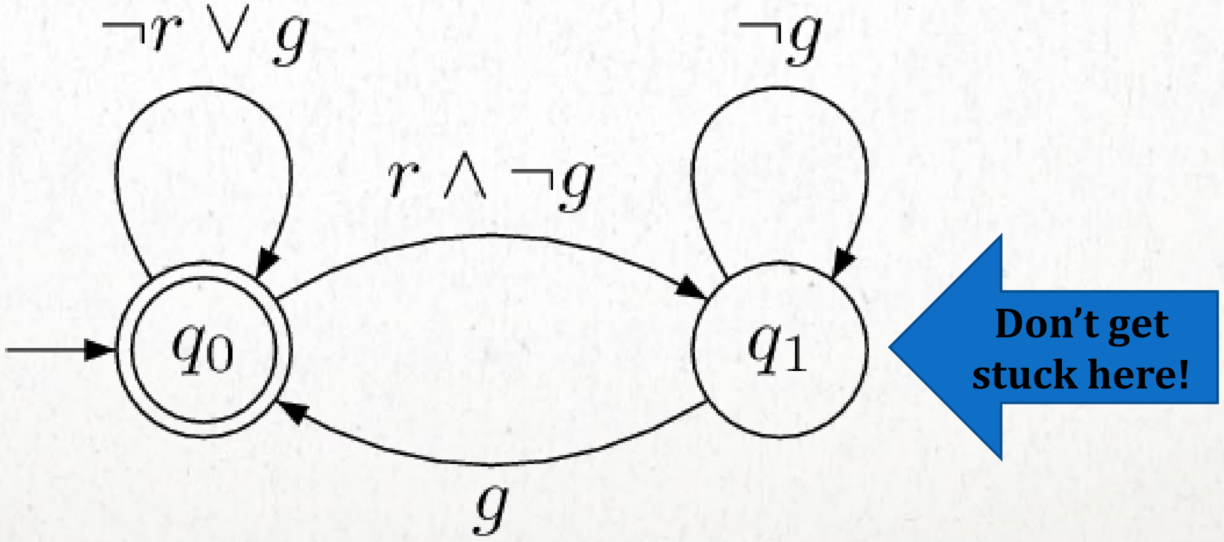
ARBITER SPECIFICATION

1. Specify
2. **Create Game**
3. Solve Game
4. Create System

Deterministic *Büchi automaton* for

$G(r \rightarrow Fg)$

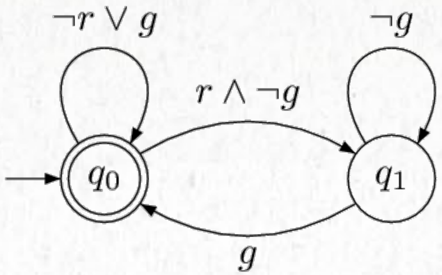
Accepting states must be visited infinitely often



ARBITER GAME

Game for

$G(r \rightarrow Fg)$



1. Specify
2. **Create Game**
3. Solve Game
4. Create System

Box: environment

Circle: system

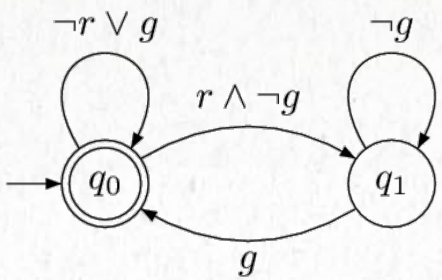
Accepting states must be visited infinitely often

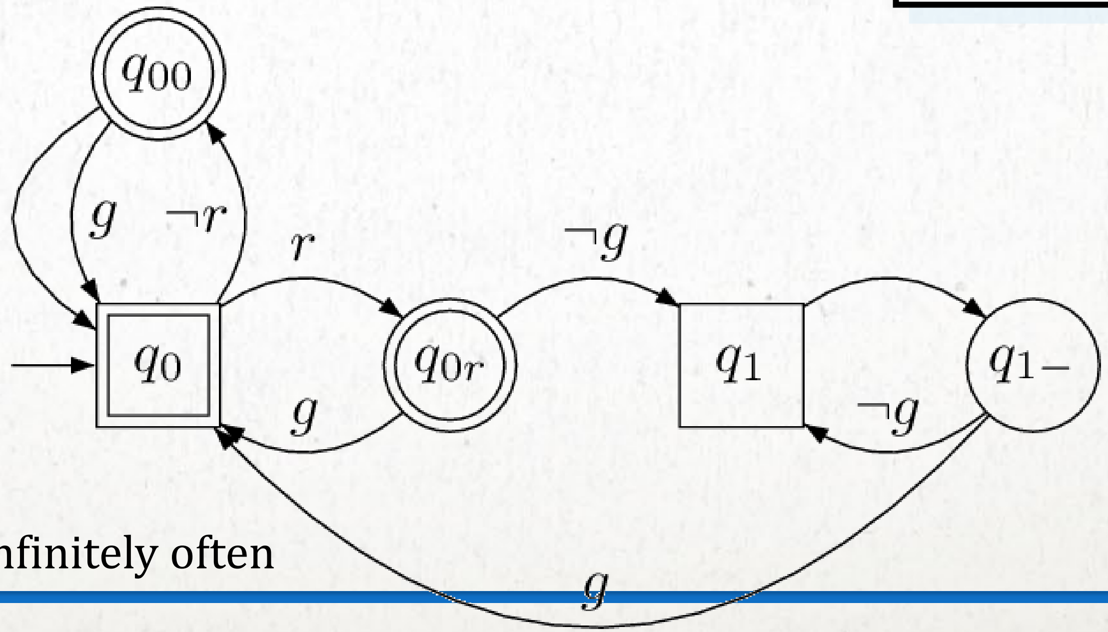
ARBITER GAME

1. Specify
2. **Create Game**
3. Solve Game
4. Create System

Game for

$G(r \rightarrow Fg)$





Box: environment

Circle: system

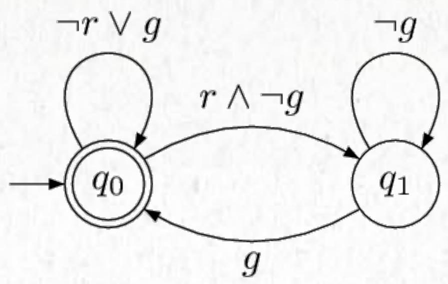
Accepting states must be visited infinitely often

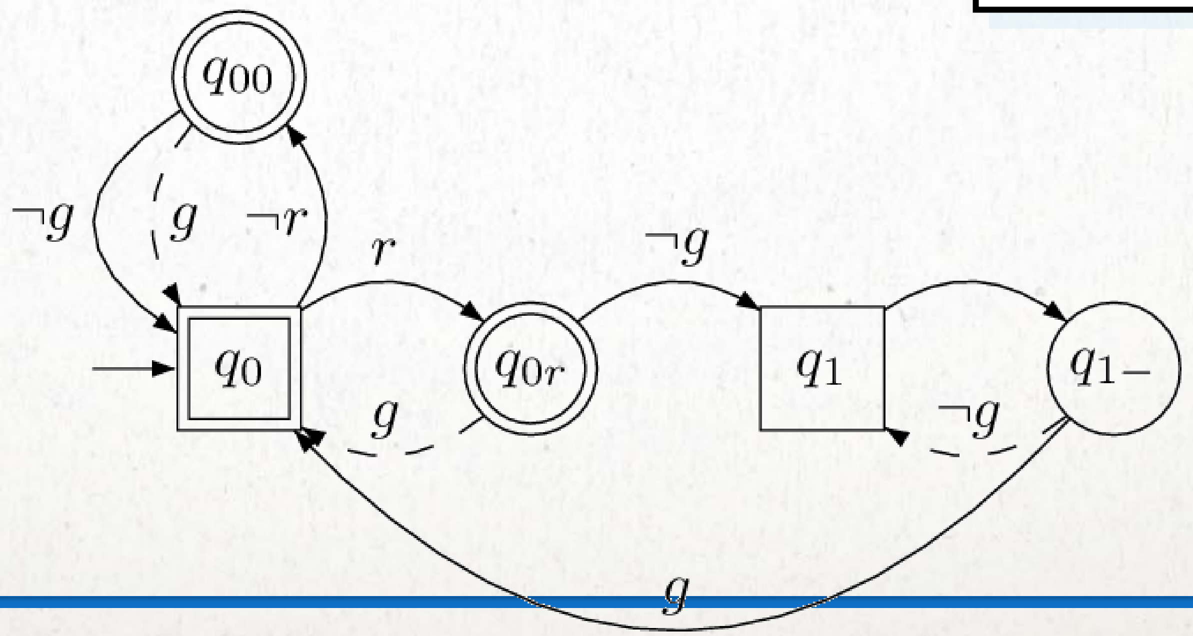
ARBITER STRATEGY

1. Specify
2. Create Game
3. **Solve Game**
4. Create System

Game for

$G(r \rightarrow Fg)$





Box: environment

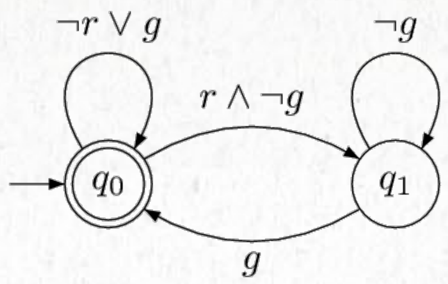
Circle: system

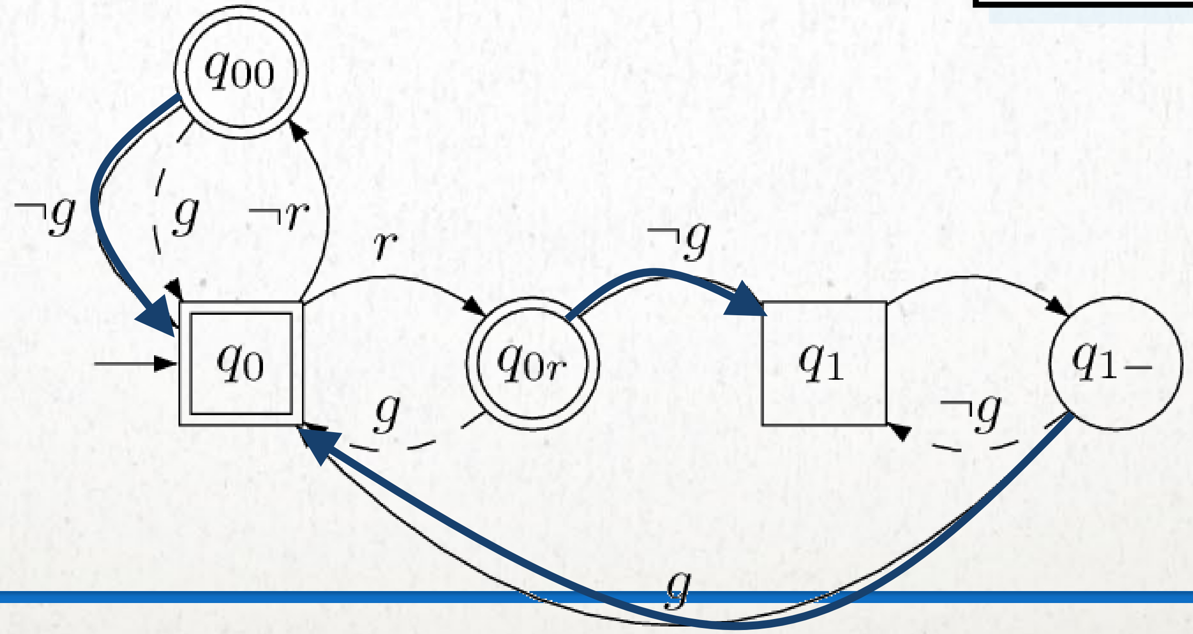
ARBITER STRATEGY

1. Specify
2. Create Game
3. **Solve Game**
4. Create System

Game for

G(r → Fg)





Box: environment

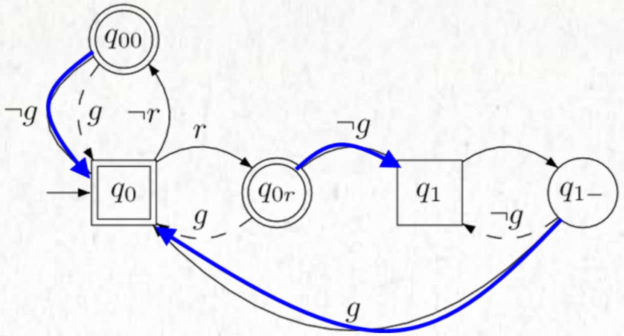
Circle: system

ARBITER STRATEGY

1. Specify
2. Create Game
3. Solve Game
4. **Create System**

Game for

G(r → Fg)




```
initial state = q0
while(){
  r = getinput();
  if(state==q0 && r==0) {g=0; state=q0}
  if(state==q0 && r==1) {g=0; state=q1}
  if(state==q1           ) {g=1; state=q0}
}
```