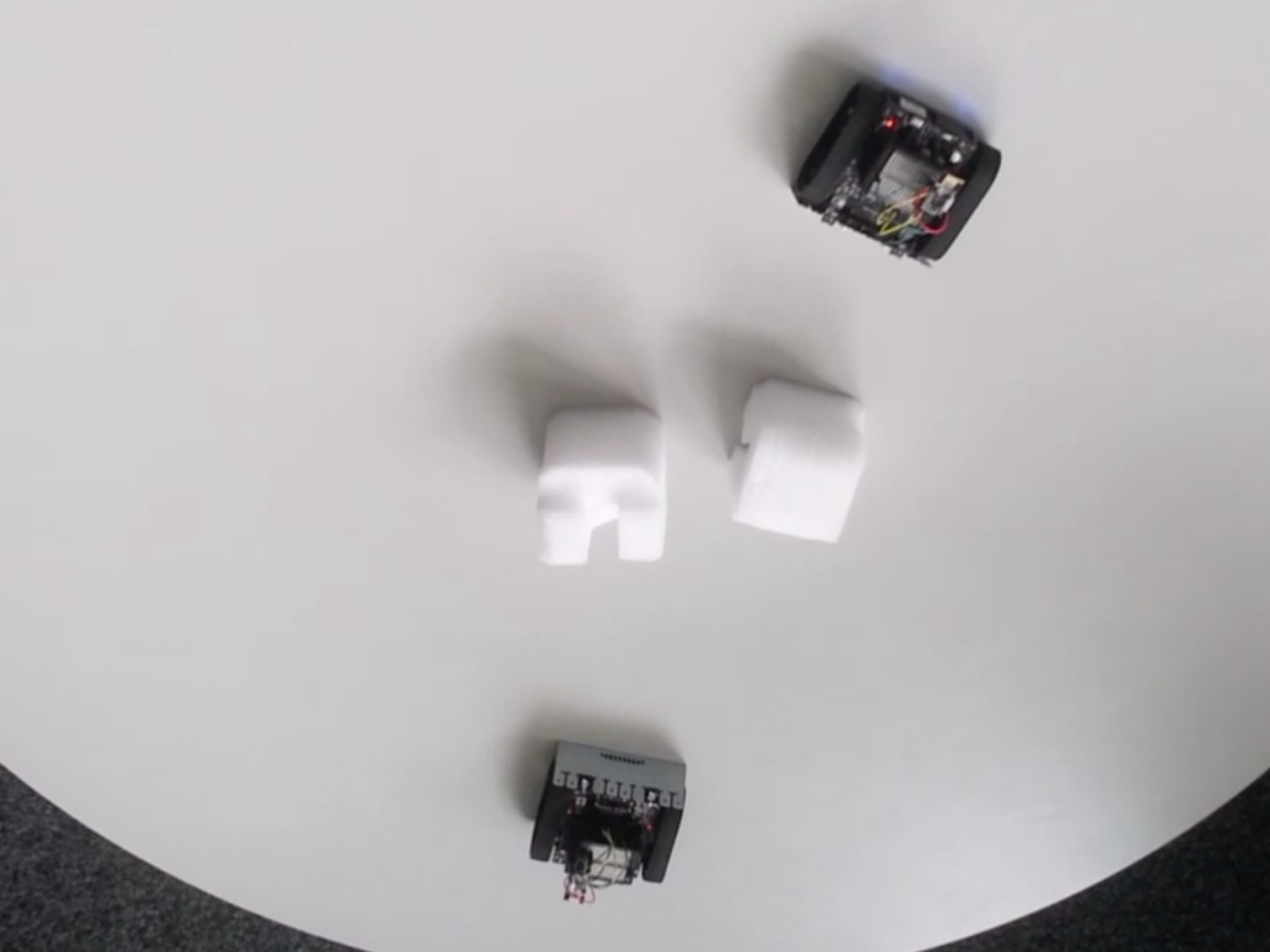
Reactive Synthesis

Bernd Finkbeiner
Universität des Saarlandes

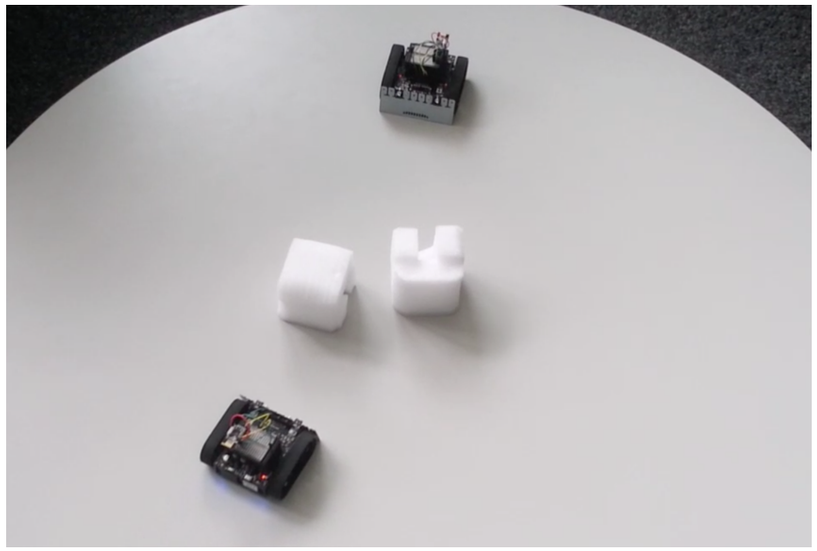
Summer School Marktoberdorf 2016



Fundamental premise of synthesis

It is easier

... to say what a system should do



1. **Eventually** all garbage must be cleared from the table.
2. The robots may **never** push each other.

... than how it should be done.

```
#include <avr/eeprom.h>
#include <LiquidCrystal.h>
#include <VirtualWire.h>
#include <AESLib.h>
const int SAFEWINDOW = 50;
long lastcount;
char *controller;
unsigned char msg[16];
uint8_t key[] = {0,1,2,3,4,5,6,7
#pragma pack 1
//uint8_t key[16];
typedef struct {
    unsigned long serial;
    long counter;
    char command;
    long extra;
    int code;
    char termin;
} PackedData;
typedef struct {
    long counter;
    unsigned long serial;
    uint8_t skey[16];
```

```
    Serial.println("");
    lcd.setCursor(0,1);
    lcd.print(StoredSett
    lcd.setCursor(0,0);
    lcd.print("SN: ");
    lcd.setCursor(4,0);
    lcd.print(StoredSett
    pinMode(2, OUTPUT);
    vw_set_ptt_inverted(
    vw_set_rx_pin(A5);
    vw_setup(4000); // B
    vw_rx_start();
}
void pairing(void)
{
    uint8_t buf[VW_MAX_M
    uint8_t buflen = VW_
    lcd.setCursor(0,0);
    lcd.print(" Pairing!
    long start =millis()
    int ledState = LOW;
    long previousMillis
    do{
```

```
case 'a':
    lcd.clear();
    lcd.setCursor(0
    lcd.print("OPEN
    //do the servo
    break;
case 'b':
    lcd.clear();
    lcd.setCursor(0
    lcd.print("CLOS
    //do the servo
    break;
case 'c':
    lcd.clear();
    lcd.setCursor(0
    lcd.print("NEXT
    //do the servo
    break;
```

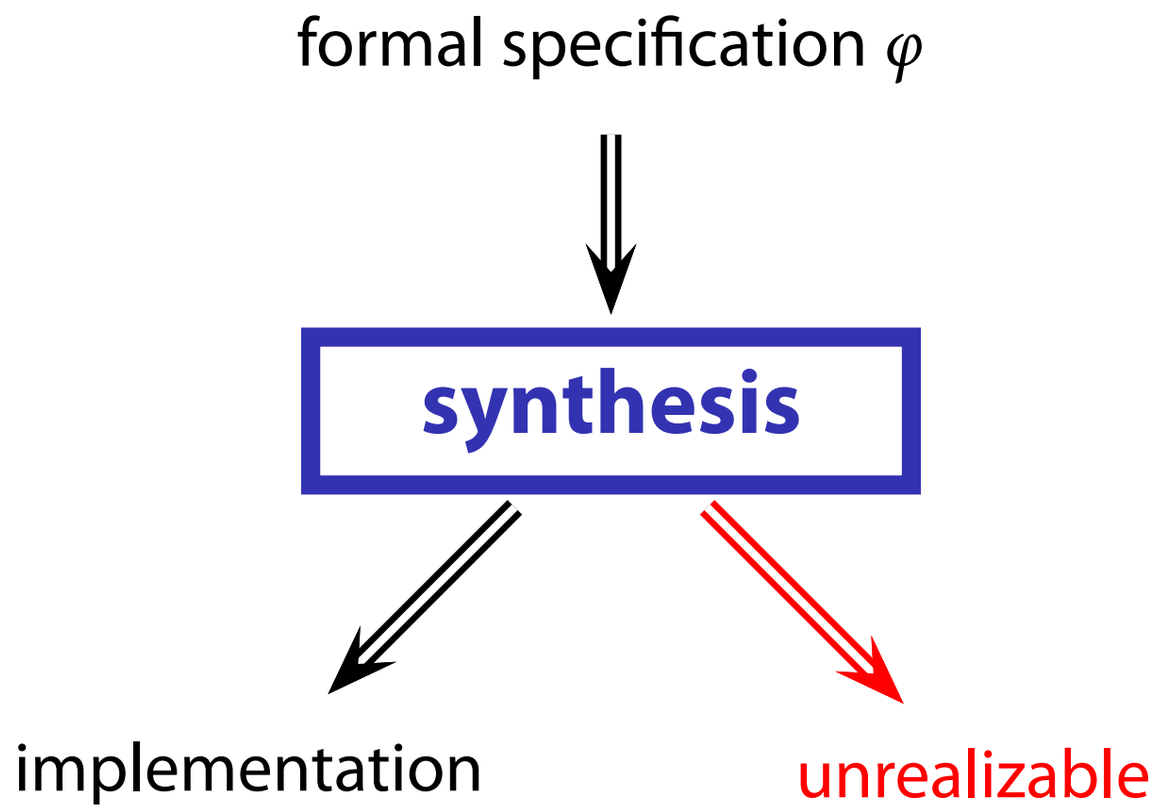
```
    } while ((millis()-start)<10000); //for 5 seconds
}
void loop()
{

  if (read_LCD_buttons()!=0)
   {
  pairing();
   }
   uint8_t buf[VW_MAX_MESSAGE_LE
   uint8_t buflen = VW_MAX_MESSA
   if (vw_get_message(buf, &bufl
   {
       Serial.print("Encrypted:"
       memcpy(msg, buf, buflen);
       for(int k=0;k<16;k++)
       Serial.print(msg[k],HEX);
       Serial.println("");
```

```
long currentcounter;
if(sdData->code==555){
    lcd.clear();
    lcd.setCursor(0,0);
    lcd.print(sdData->serial
    lcd.setCursor(0,1);
    lcd.print(sdData->counte
    //do the job if the cour
    currentcounter= sdData->
    if((currentcounter-Store
    {
        lcd.setCursor(0,0);
        lcd.print("Access G
        StoredSettings.count
        eeprom_write_block((
        //do the stuff here
        char command = sdDat
        switch (command) {
```

```
int adc_key_in;
// read the buttons
int read_LCD_buttons()
{
    adc_key_in = analogRead(0); // read the va
    if (adc_key_in < 50) return 1;// btnRIGHT;
    if (adc_key_in < 250) return 2;// btnUP;
    if (adc_key_in < 450) return 3;//btnDOWN;
    if (adc_key_in < 650) return 4;//btnLEFT;
    if (adc_key_in < 850) return 5;//btnSELECT
    return 0;//btnNONE; // when all others fai
}
void setup()
{
    Serial.begin(9600);
    lcd.begin(16, 2); // start the library
```

Synthesis



Synthesized implementation is guaranteed to satisfy φ !

Focus

- Reactive systems
  - Continuous interaction with environment
  - Correctness depends on temporal properties (temporal logic, automata)
- Finite state space
- Focus on control, not data transformation
- Typical examples:
  - Reactive layer of cyberphysical systems
  - Hardware circuits

Cyberphysical example: Autonomous driving

- Reactive traffic planner decides
  whether vehicle should stay
  in the travel lane or
  perform a passing maneuver,
  whether it should go or stop,
  whether it is allowed to reverse, etc.

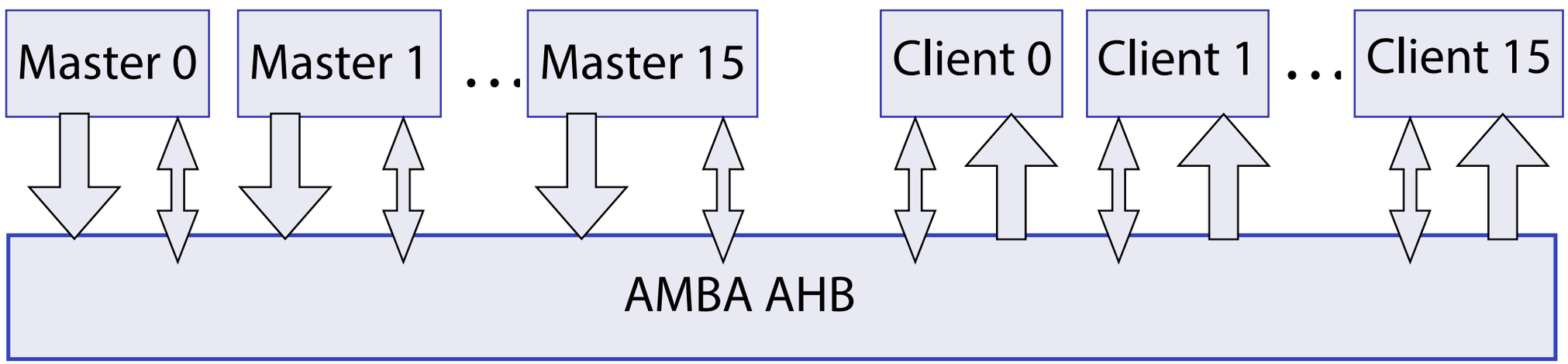
- Hierarchical control: reactive traffic planner interacts with mission control (above) and path planner (below).
- Specification consists of
  - traffic rules:
    for example “no collision”, “obey speed limits”
  - goals:
    for example “eventually the checkpoint should be reached”

[Murray et al, 2012]

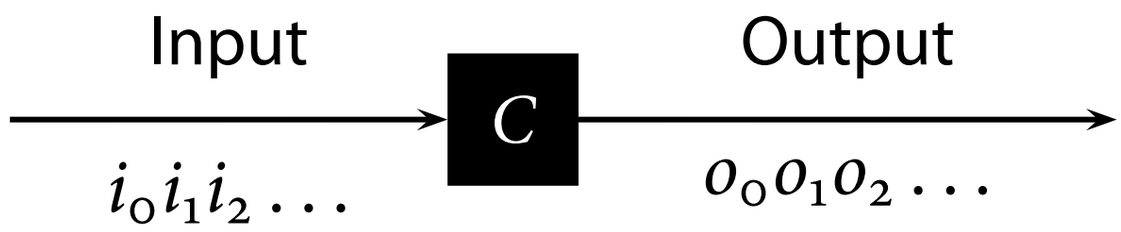
Hardware example: AMBA AHB Bus

- High-performance on-chip bus
- Data, address, and control signals
- Up to 16 masters and 16 clients
- Specification consists of
  - 12 guarantees:
    for example “when a locked unspecified length burst starts, new access does not start until current master (i) releases bus by lowering HBUSREQi.”
  - 3 assumptions:
    for example “the clients indicate infinitely often that they have finished processing the data by lowering HREADY”

[Bloem et al, 2007]



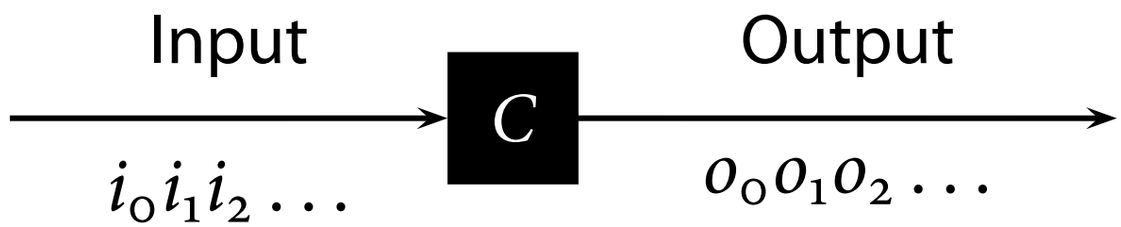
The reactive synthesis problem



Alonzo Church (1957)

Given a requirement φ
on the input-output behavior of a boolean circuit,
compute a circuit C that satisfies φ.

Game theoretic formulation



Synthesis game

- Player 0 produces outputs, Player 1 produces inputs.
- Game is played in infinitely many rounds.
- In each round, first Player 1 produces an input, then Player 0 produces an output.
- Player 0 wins if the resulting sequence of inputs and outputs satisfies φ.

Example: Synthesis of an arbiter circuit

An arbiter circuit receives **requests** r_1, r_2 from two clients and produces **grants** g_1, g_2.

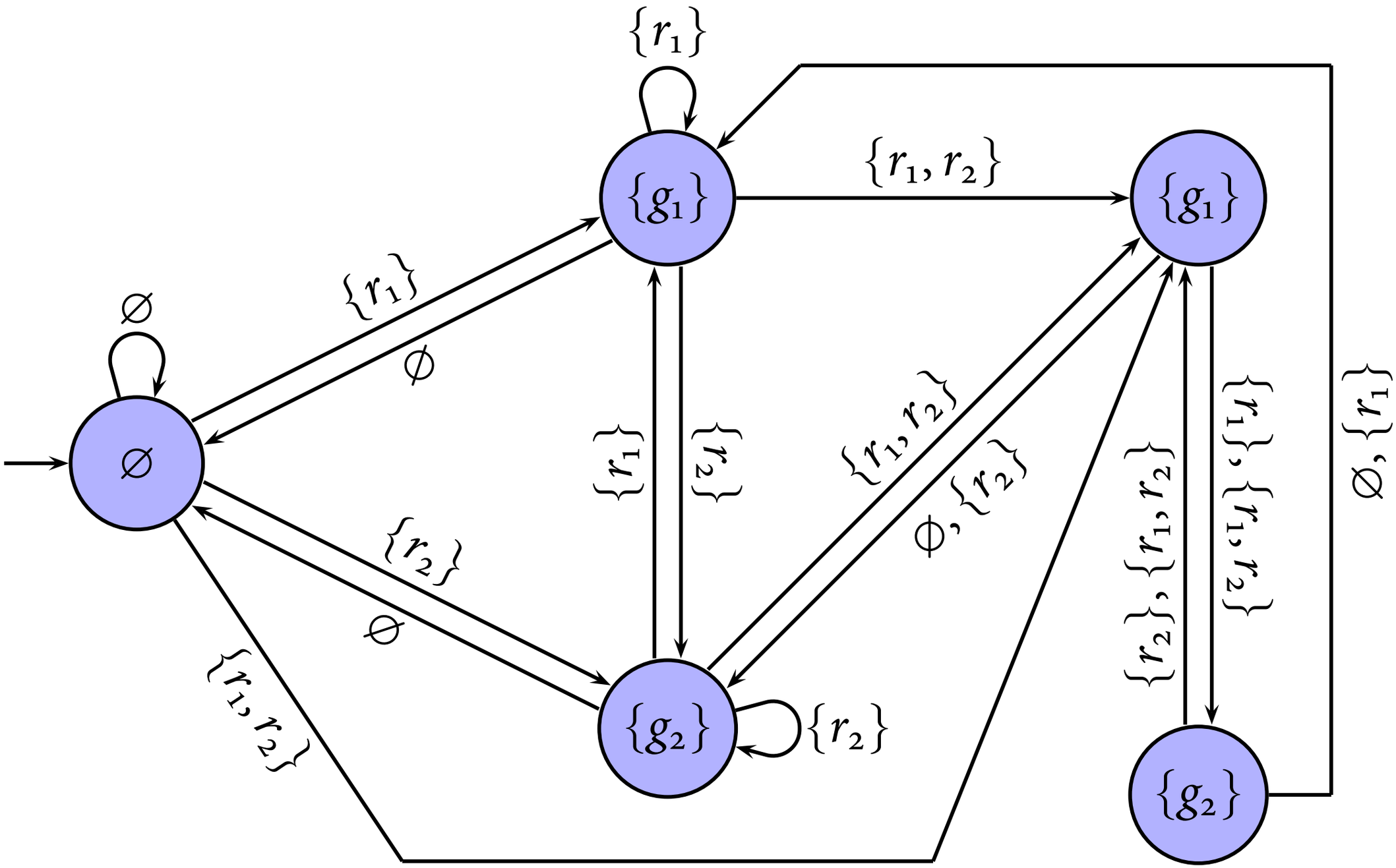
The **specification** φ of the arbiter is the conjunction of the following properties:

1. **Mutual exclusion:** at no point in time should there be both g_1 and g_2 in the output.
2. **Response:** every request r_i from the client i (for $i \in \{1, 2\}$) should eventually be followed by grant g_i for client i.

Winning strategy:

- Initial output is $\varnothing$.
- If input is $\varnothing$ (*no request*) respond with $\varnothing$.
- If input is $\{r_1\}$ (*only client 1 requests grant*) respond with $\{g_1\}$.
- If input is $\{r_2\}$ (*only client 2 requests grant*) respond with $\{g_2\}$.
- If input is $\{r_1, r_2\}$ (*both clients request grants*) **...**

Winning strategy



Reactive synthesis

- **The problem is hard.**
  - Synthesis from LTL specifications is **2EXPTIME hard**.
  - Synthesis of distributed systems (where the processes have incomplete information) is in general **undecidable**.
- **There has been a lot of progress in the last 10 years.**
  - Specification languages with lower complexity
  - Bounded synthesis
  - Applications, e.g., in hardware design and robotics
- **Synthesis competition www.syntcomp.org**
  - ≈ 3500 benchmarks

Overview

Part I. Infinite Games

Fundamental algorithms to solve infinite games played over finite graphs.

Part II. Synthesis from Logical Specifications

Synthesis from specifications given as formulas of a temporal logic. The quest for an efficient and expressive specification language.

Part III. Bounded Synthesis

Finding simple solutions fast. The quest for structurally simple implementations.

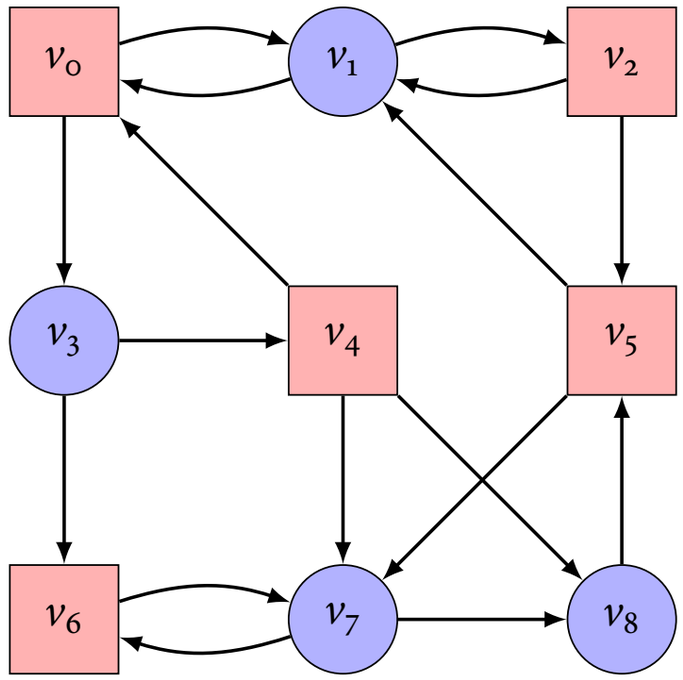
Part IV. Distributed Synthesis

Synthesizing systems that consist of multiple distributed components.

Part I: Infinite Games

1. **Definitions**
2. Reachability games
3. Büchi games
4. Parity games

Game arenas and plays



A **game arena** $\mathcal{A} = (V, V_0, V_1, E)$ consists of

- a finite set V of states,
- a subset $V_0 \subseteq V$ of states owned by Player 0 (circles),
- a subset $V_1 = V \setminus V_0$ of states owned by Player 1 (boxes),
- an edge relation $E \subseteq V \times V$ such that every state $v \in V$ has at least one outgoing edge $(p, p') \in E$.

A **play** is an infinite path through $\mathcal{A}$.

Strategies

A **strategy** for Player i in $\mathcal{A}$ is a function $\sigma : V^* \cdot V_i \to V$ such that $(v_n, \sigma(v_0 v_1 \dots v_n)) \in E$ for every prefix $v_0 v_1 \dots v_n$ of a play.

A play $v_0 v_1 \dots$ **is consistent with** strategy σ, if $v_{n+1} = \sigma(v_0 \dots v_n)$ whenever $v_n \in V_i$.

Special types of strategies:

- **Positional strategies:** $\sigma(v_0 v_1 \dots v_n) = \sigma(v_n)$
  strategy only depends on last state
- **Finite-state strategies:** implemented by some FSM

Winning conditions

- A **reachability game** $\mathcal{G} = (\mathcal{A}, R)$ consists of an arena $\mathcal{A}$ and a set $S \subseteq V$ of states. Player 0 wins a play π if π visits R at least once, otherwise Player 1 wins.
- A **Büchi game** $\mathcal{G} = (\mathcal{A}, F)$ consists of an arena $\mathcal{A}$ and a set $F \subseteq V$ of states. Player 0 wins a play π if π visits F infinitely often, otherwise Player 1 wins.
- A **parity game** $\mathcal{G} = (\mathcal{A}, \alpha)$ consists of an arena $\mathcal{A}$ and a coloring function $\alpha : V \to \mathbb{N}$. Player 0 wins a play π if the highest color that is seen infinitely often is even, otherwise Player 1 wins.

Winning regions

A strategy σ is **winning** for Player i from some state v if all plays that are consistent with σ and that start in v are won by Player i.

The **winning region** $W_i(\mathcal{G})$ is the set of states from which Player i has a winning strategy.

A game is **determined** if $V = W_0 \cup W_1$.

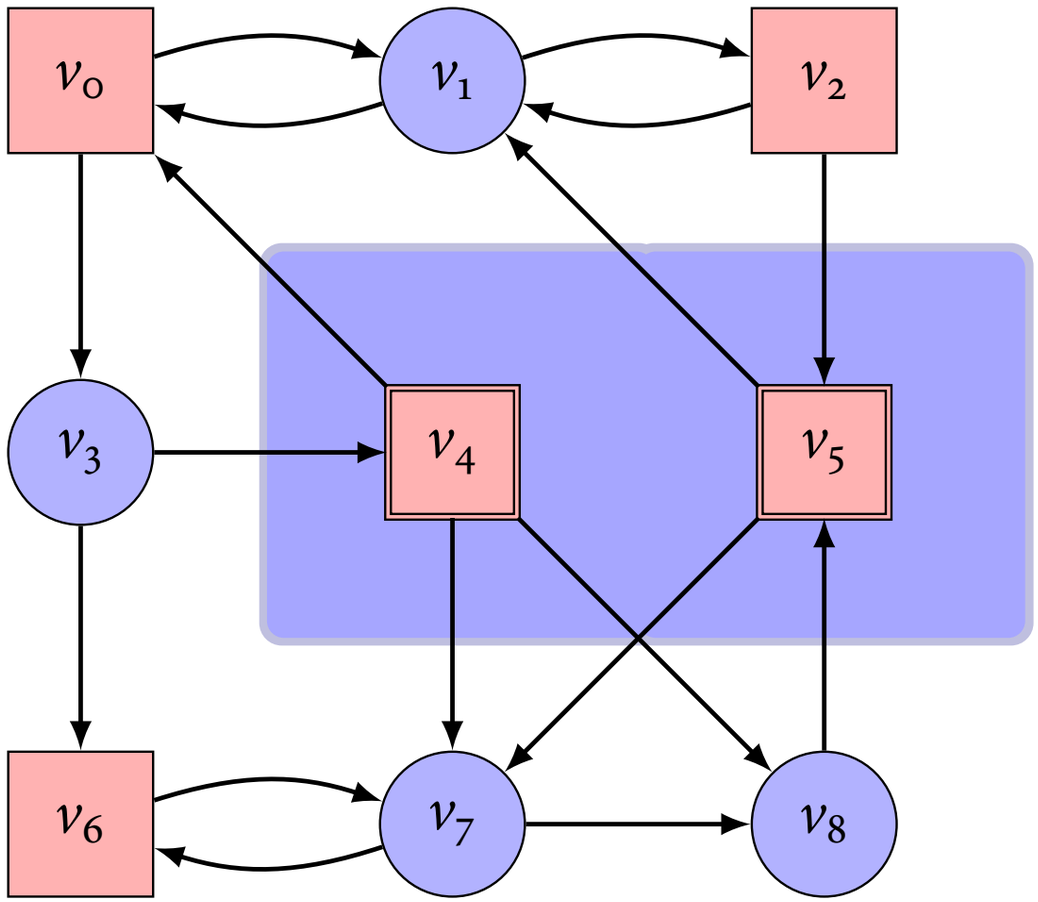
Solving a game means to determine the winning region and the winning strategies.

Part I: Infinite Games

1. Definitions
2. **Reachability games**
3. Büchi games
4. Parity games

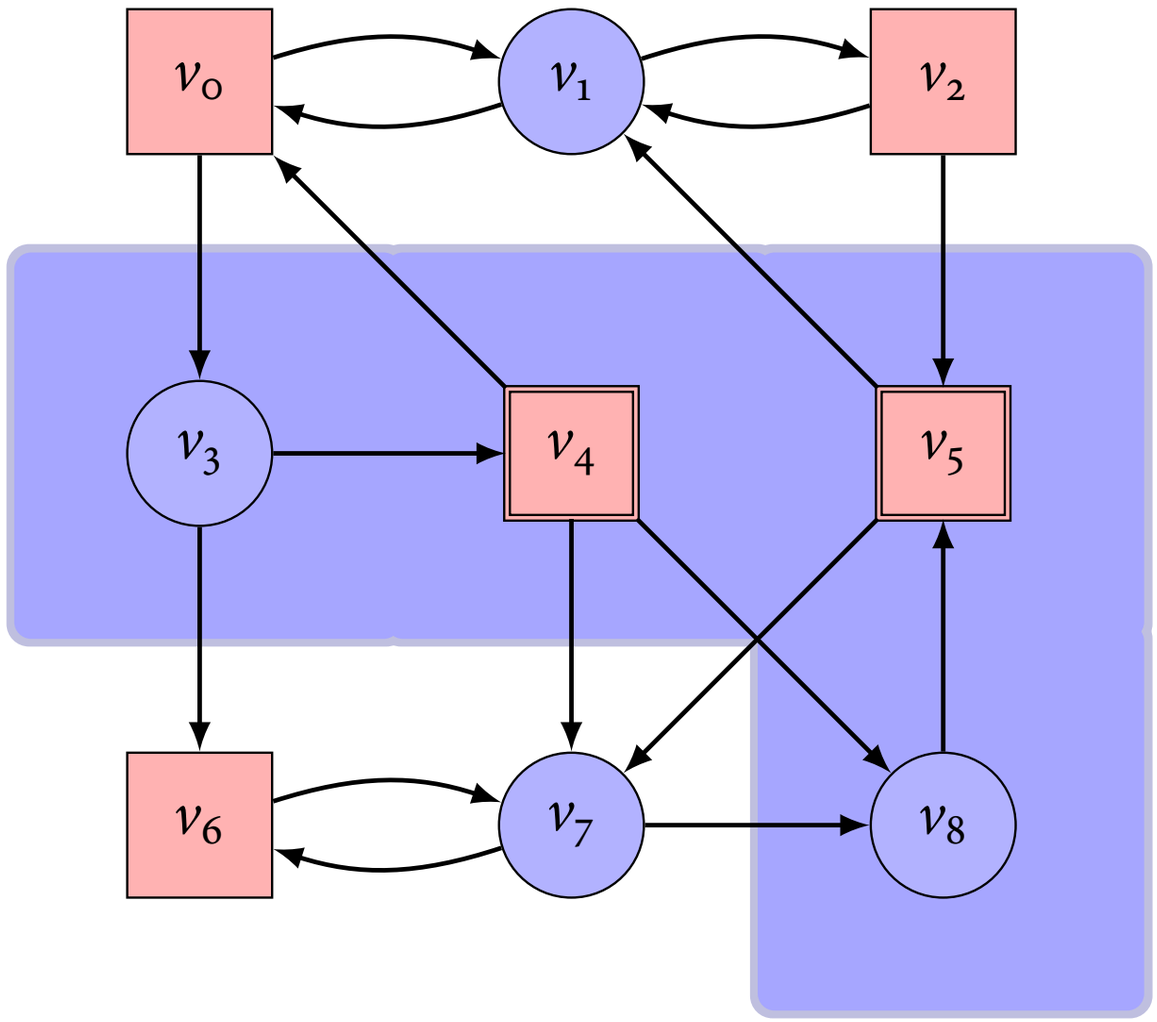
Reachability games

Reachability game: Player o wins a play π if π visits R at least once



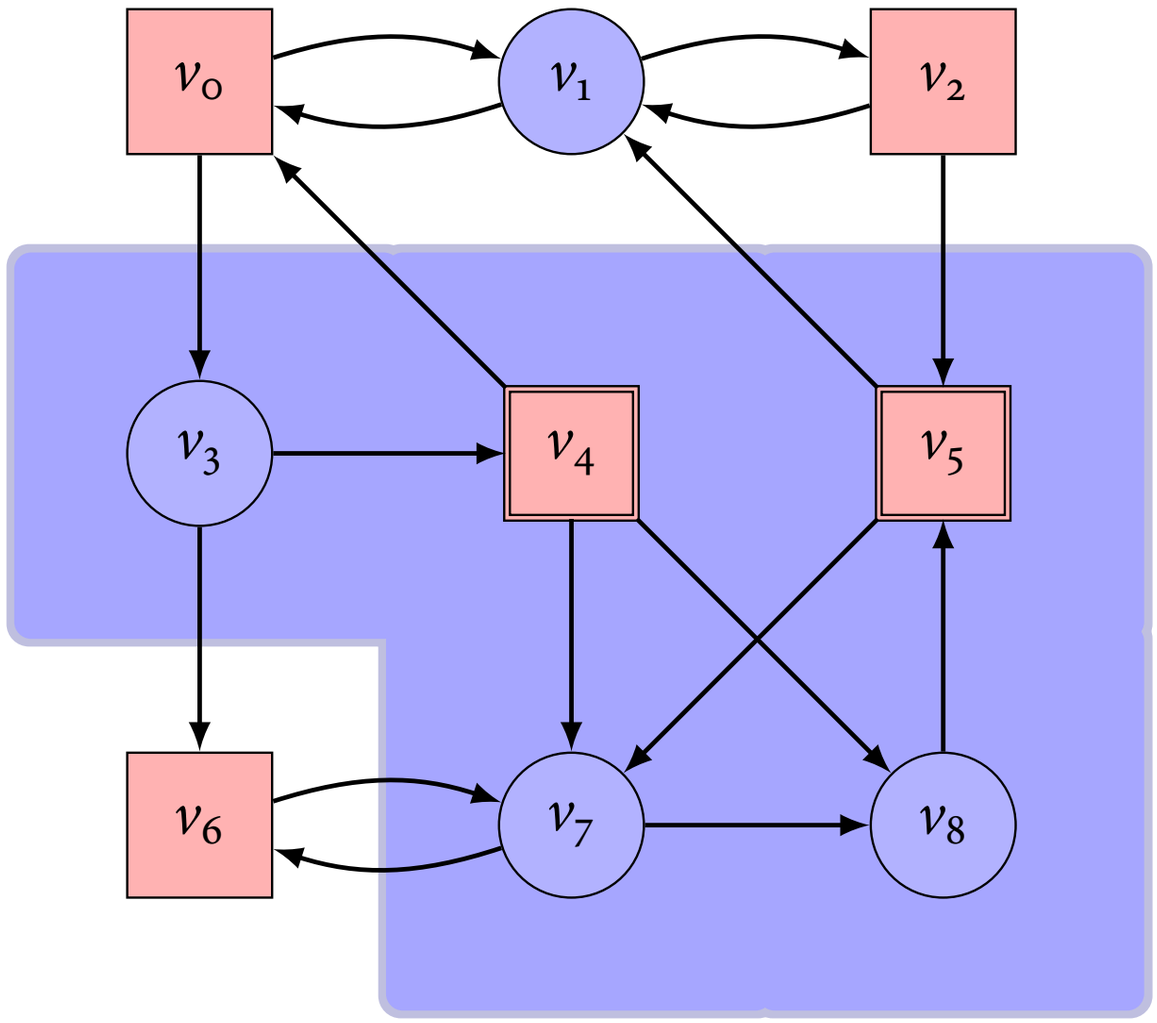
Reachability games

Reachability game: Player o wins a play π if π visits R at least once



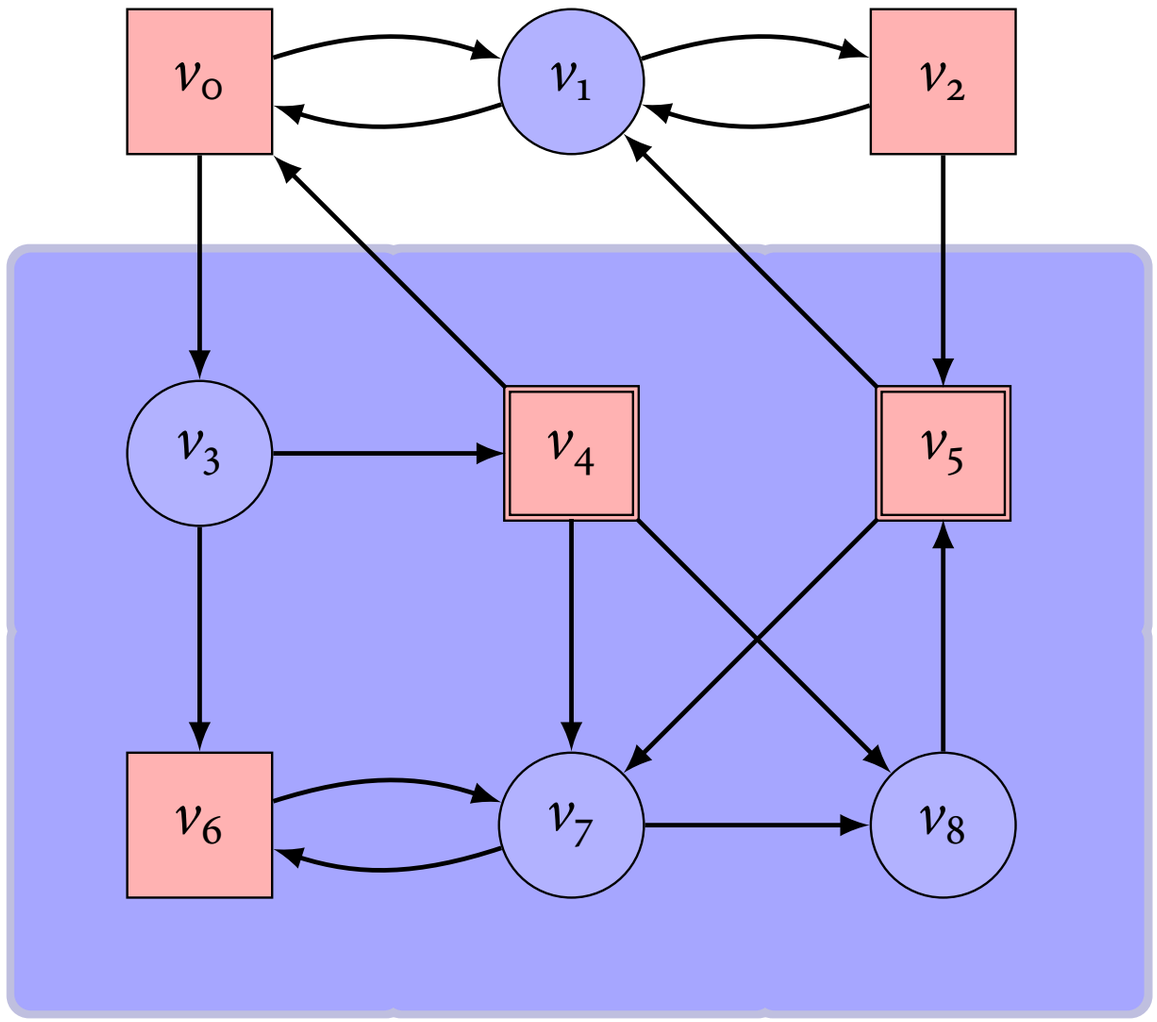
Reachability games

Reachability game: Player o wins a play π if π visits R at least once



Reachability games

Reachability game: Player o wins a play π if π visits R at least once



Attractor construction

$$
\begin{aligned}
Attr_i^0(R) &= R \\
Attr_i^{n+1}(R) &= Attr_i^n(R) \cup CPre_i(Attr_i^n(R))
\end{aligned}
$$

$$
Attr_i(R) = \bigcup_{n \in \mathbb{N}} Attr_i^n(R)
$$

where

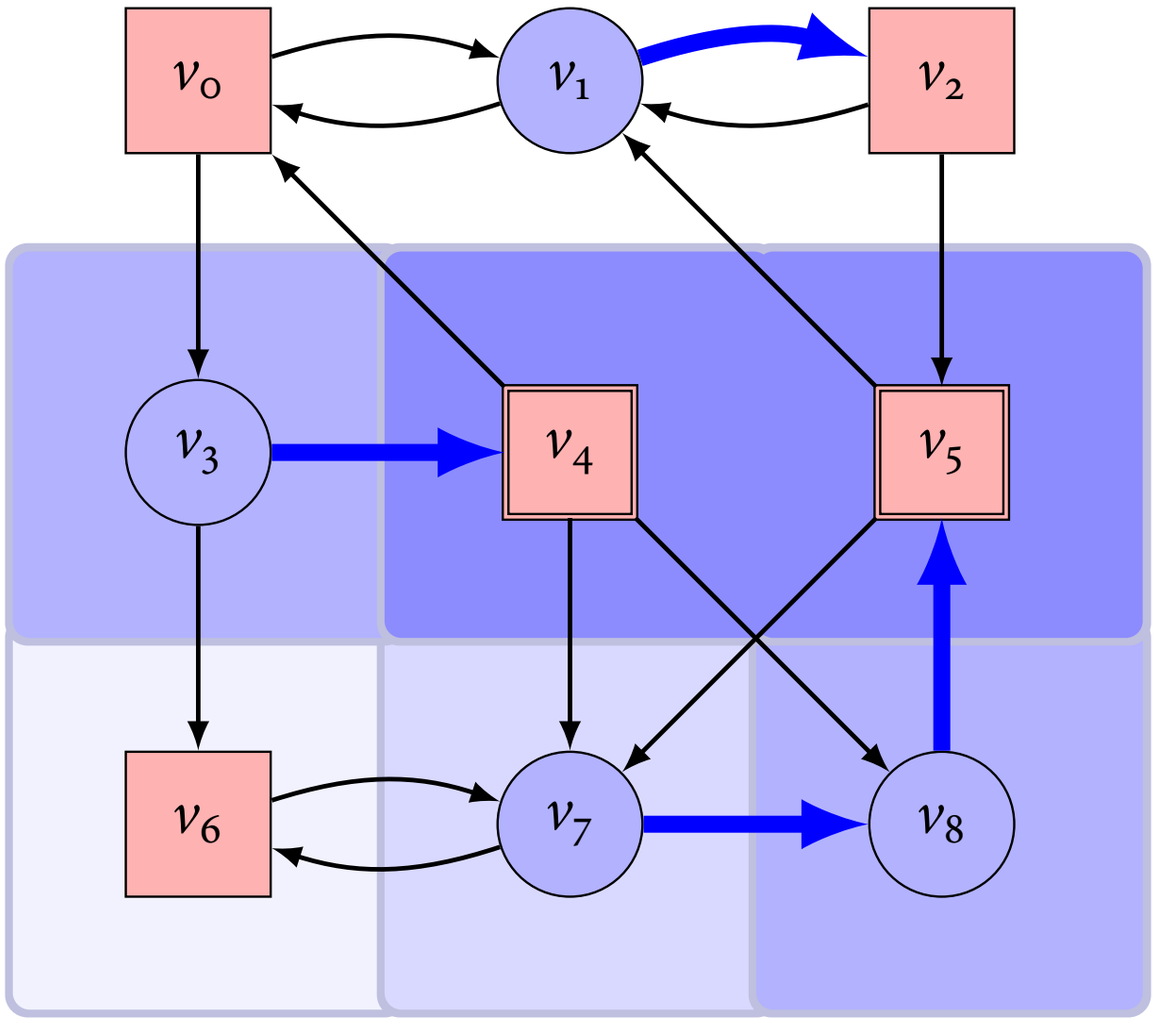
$$
\begin{aligned}
CPre_i(R) = &\{ v \in V_i \mid \exists v' \in V. (v, v') \in E \wedge v' \in R \} \\
&\cup \{ v \in V_{1-i} \mid \forall v' \in V. (v, v') \in E \Rightarrow v' \in R \}
\end{aligned}
$$

Winning regions of a reachability game

Winning region of Player 0: $W_0(\mathcal{G}) = Attr_0(R)$

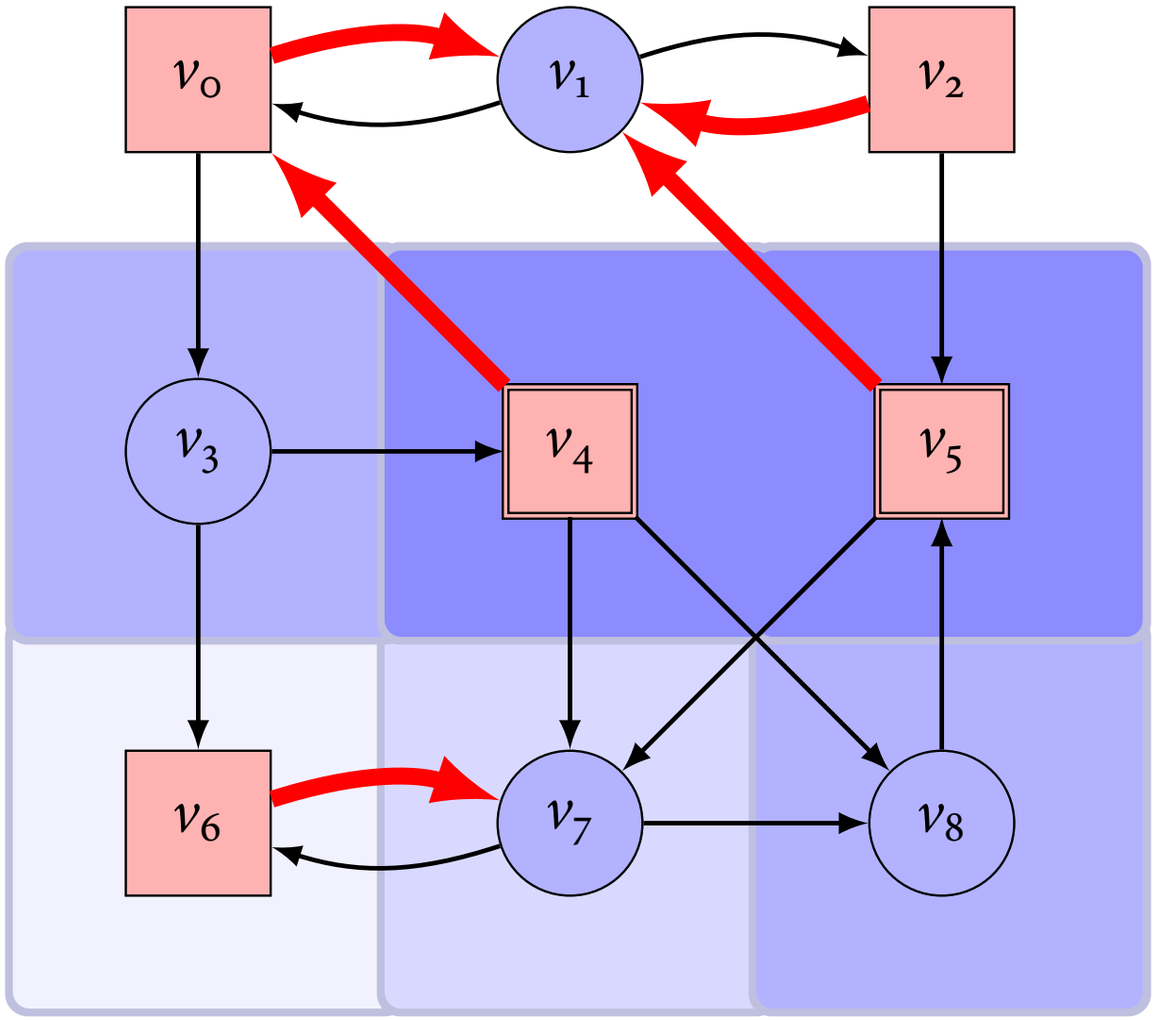
Winning region of Player 1: $W_1(\mathcal{G}) = V \setminus Attr_0(R)$

Winning strategy of Player o (Attractor Strategy)



The winning strategy for Player o always moves to $Attr_{\mathrm{o}}^{n}(R)$ for the smallest possible n. If no such successor exits from some state, the strategy is not winning from that state and it moves to an arbitrary successor.

Winning strategy of Player 1 (Safety Strategy)



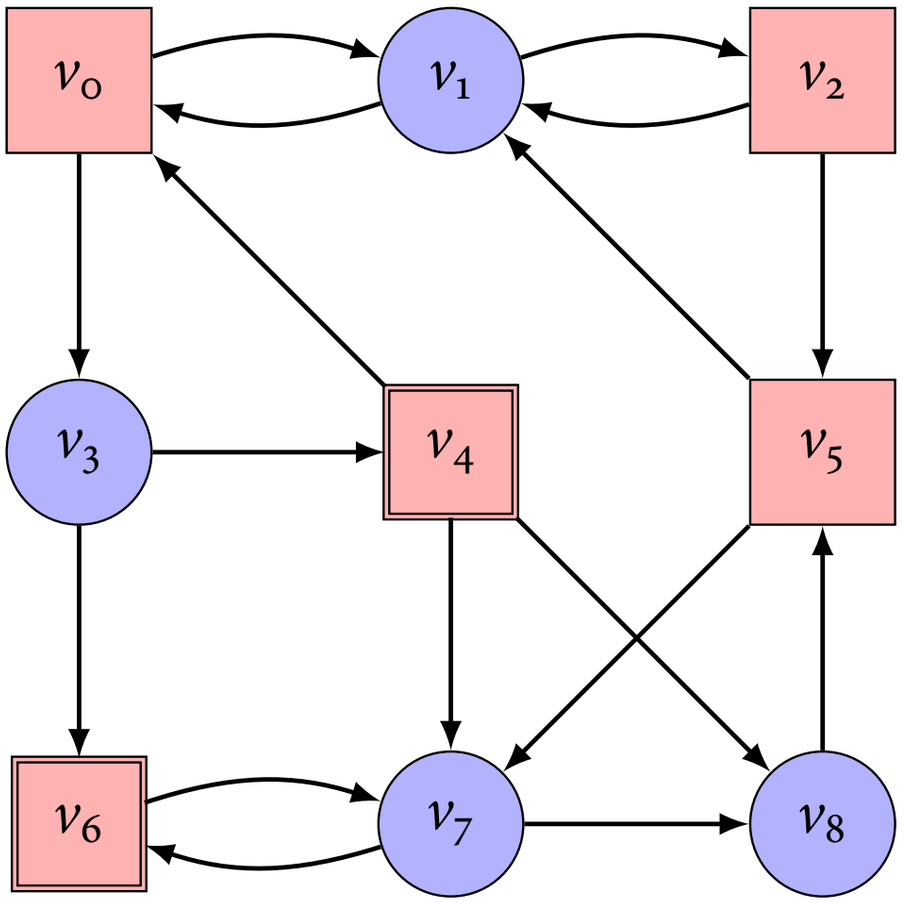
The winning strategy for Player 1 always avoids $Attr_0(R)$. If only successors in $Attr_0(R)$ exist from some state, the strategy is not winning from that state and it moves to an arbitrary successor.

Part I: Infinite Games

1. Definitions
2. Reachability games
3. **Büchi games**
4. Parity games

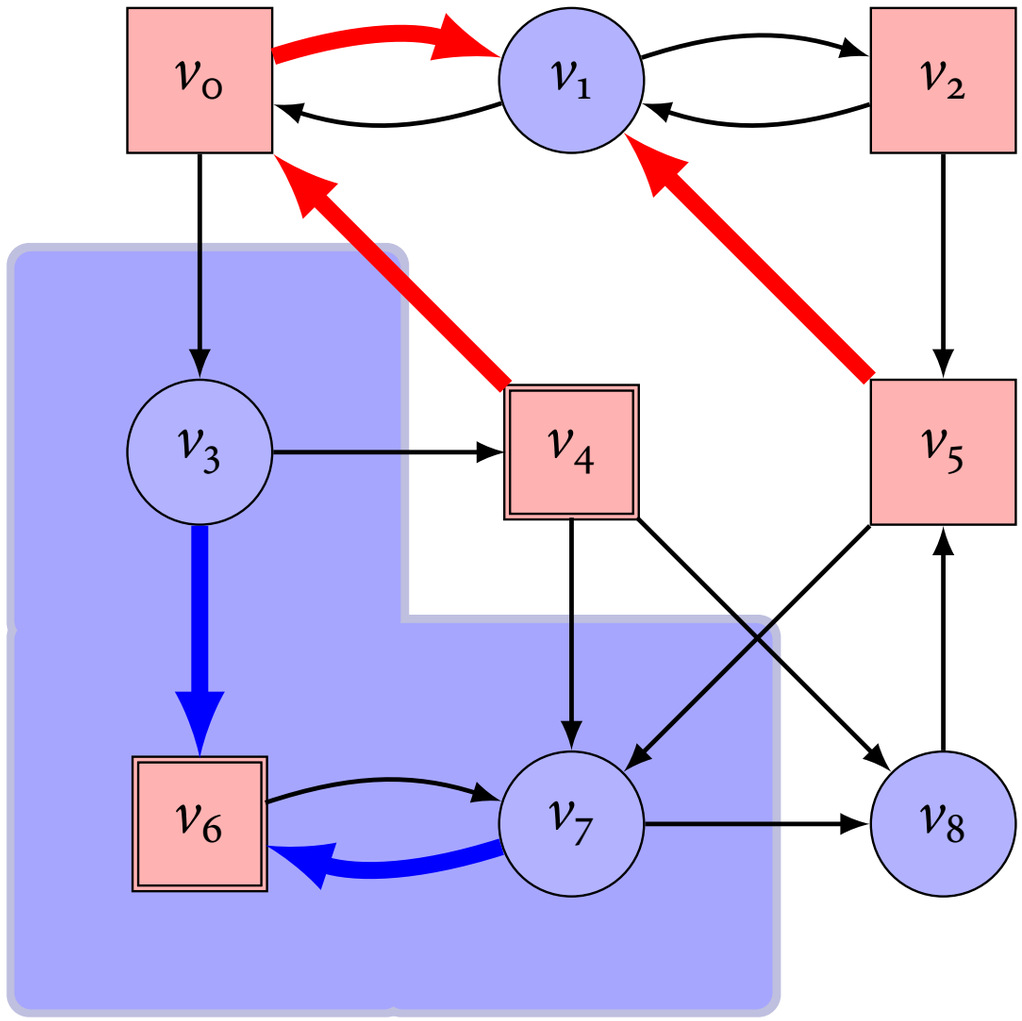
Büchi games

Büchi game: Player 0 wins a play π if π visits F infinitely often, otherwise Player 1 wins.



Büchi games

Büchi game: Player 0 wins a play π if π visits F infinitely often, otherwise Player 1 wins.



Recurrence construction

$$
\begin{aligned}
Recur^{0}(F) &= F \\
W_1^n(F) &= V \setminus Attr_0(Recur^n(F)) \\
Recur^{n+1}(F) &= Recur^n(F) \setminus CPre_1(W_1^n(F))
\end{aligned}
$$

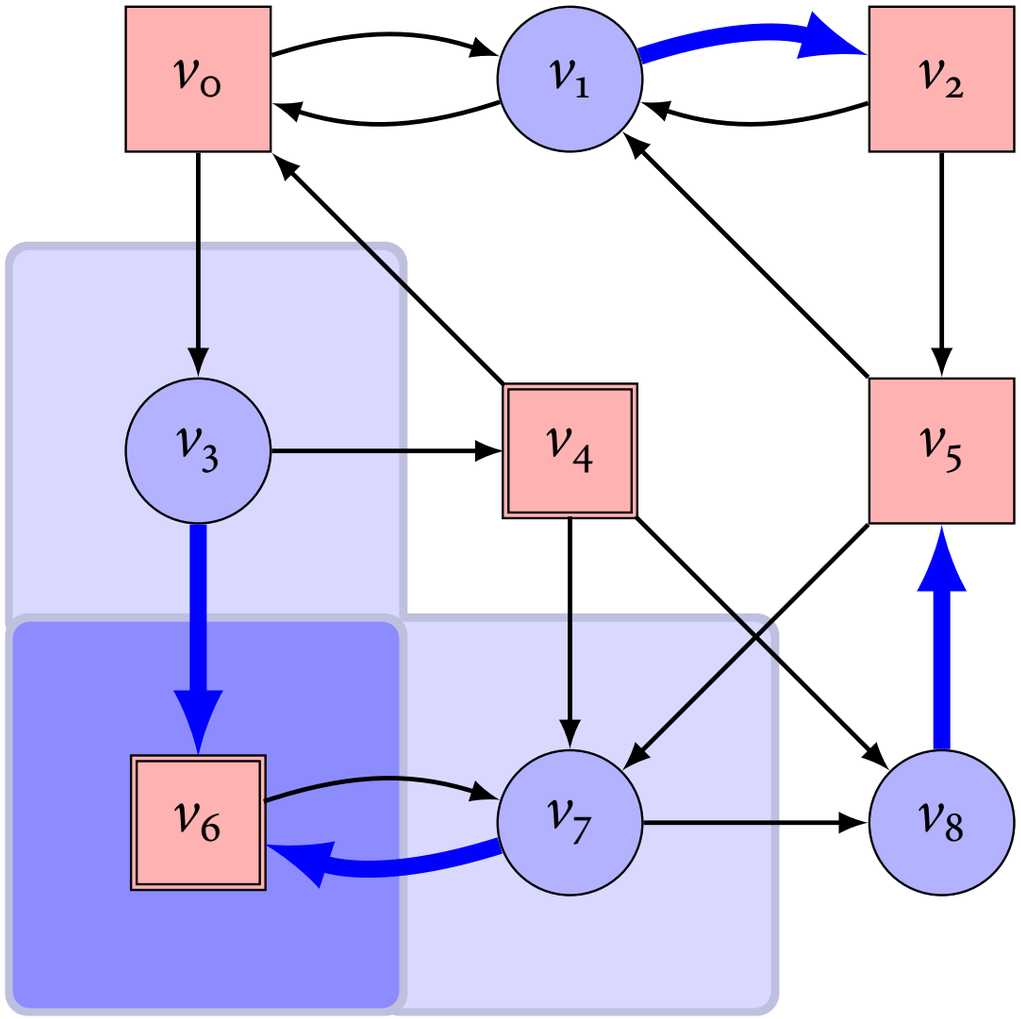
$$
Recur(F) = \bigcap_{n \in \mathbb{N}} Recur^n(F)
$$

Winning regions of a Büchi game

Winning region of Player 0: $W_0(\mathcal{G}) = Attr_0(Recur(F))$

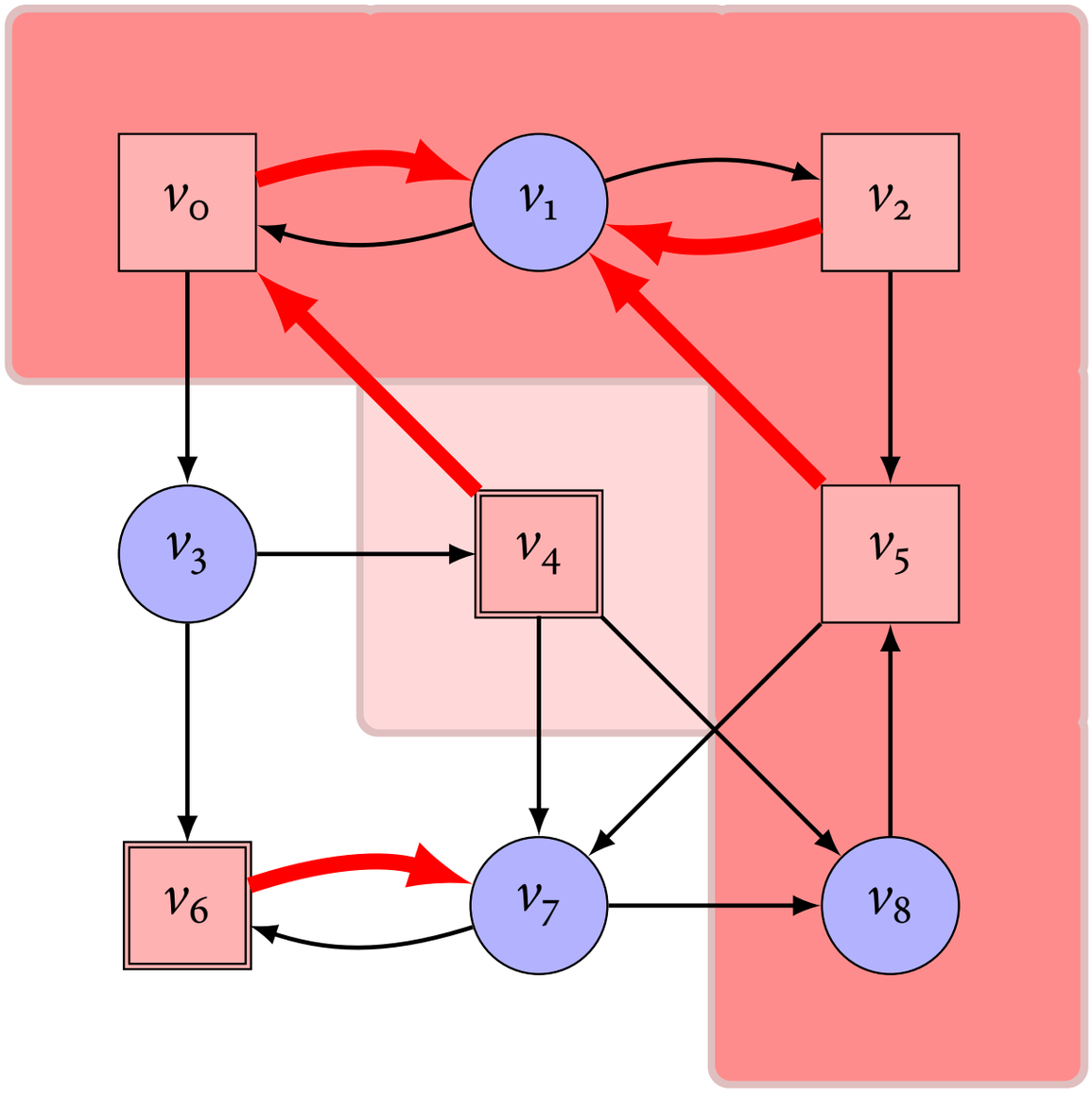
Winning region of Player 1: $W_1(\mathcal{G}) = V \setminus Attr_0(Recur(F))$

Winning strategy of Player o (Büchi Strategy)



The winning strategy for Player o always moves to $Attr_{\mathrm{o}}^{n}(Recur(F))$ for the smallest possible n. If no such successor exits from some state, the strategy is not winning from that state and it moves to an arbitrary successor.

Winning strategy of Player 1 (co-Büchi Strategy)



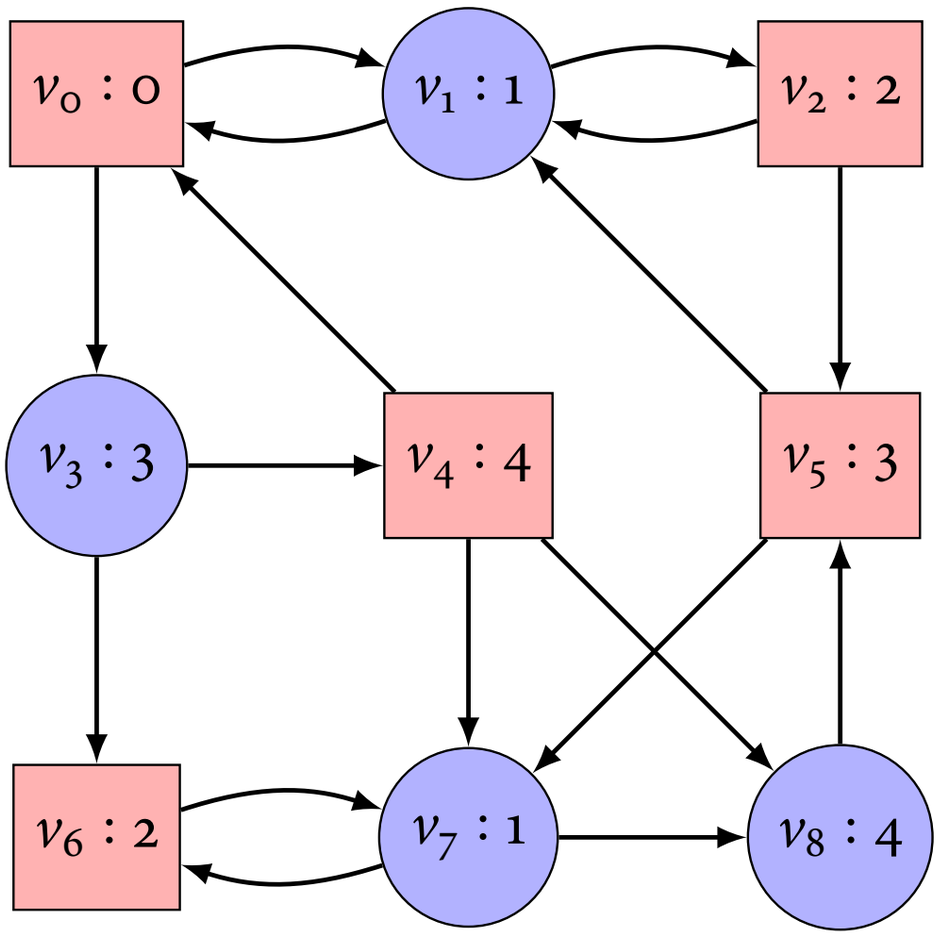
The winning strategy for Player 1 moves from a state in W_1^n to W_1^{n-1} whenever possible, and stays in W_1^n otherwise. Outside W_1, the strategy is not winning and it moves to an arbitrary successor.

Part I: Infinite Games

1. Definitions
2. Reachability games
3. Büchi games
4. **Parity games**

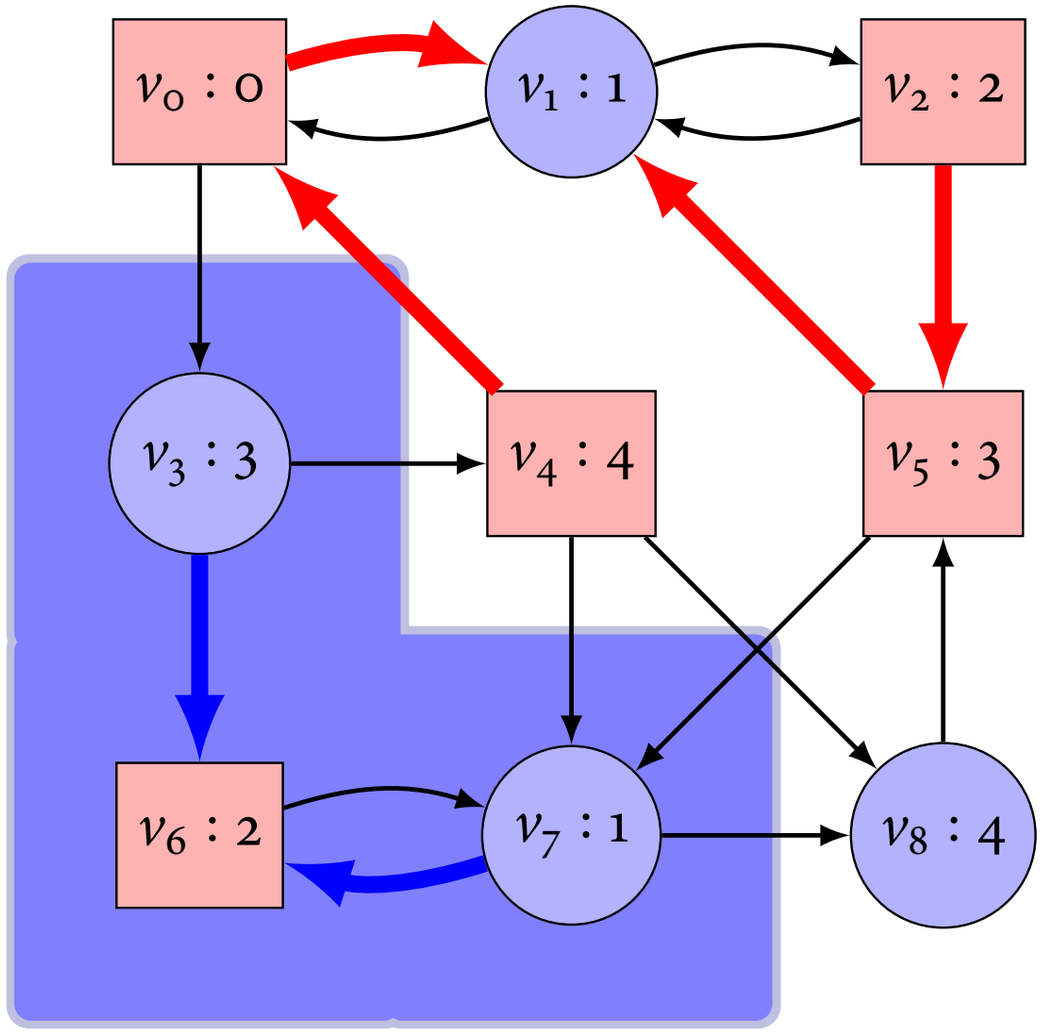
Parity games

Parity game: Player o wins a play π if the highest color that is seen infinitely often is even.



Parity games

Parity game: Player o wins a play π if the highest color that is seen infinitely often is even.



McNaughton's Algorithm

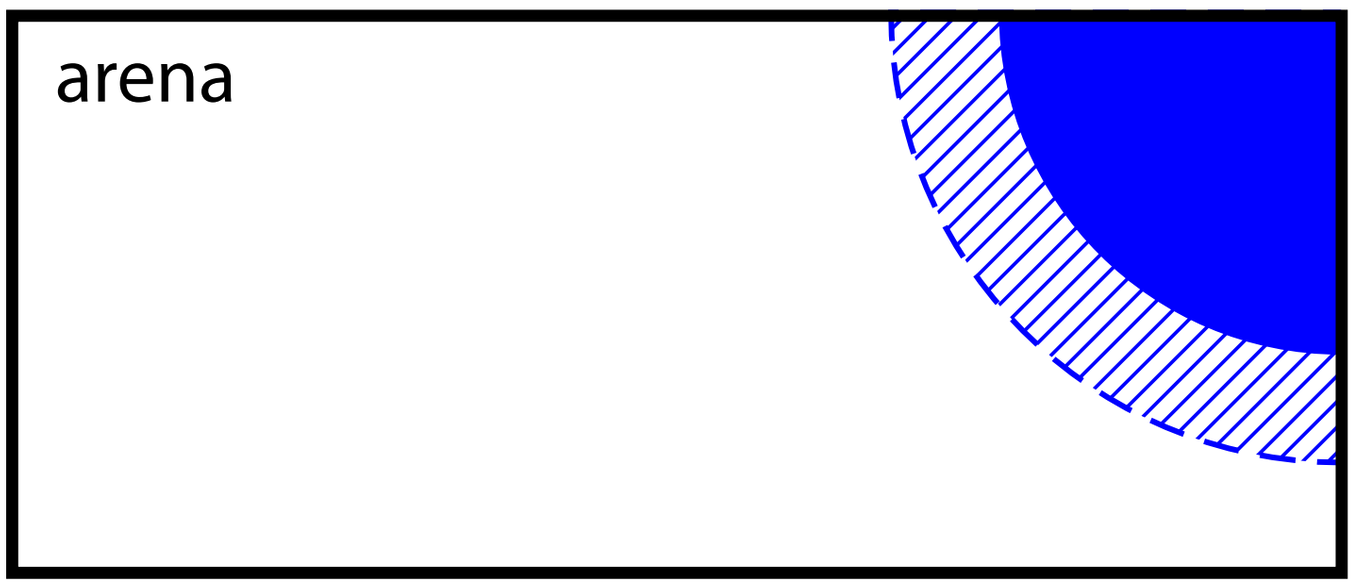
$McNaughton(\mathcal{G})$

1. $c :=$ **highest color in** $\mathcal{G}$
2. <u>if</u> $c = 0$ or $V = \emptyset$
   <u>then return</u> $(V, \emptyset)$
3. set i to $c \mod 2$
4. set W_{1-i} to $\emptyset$
5. <u>repeat</u>
   - 5.1 $\mathcal{G}' := \mathcal{G} \setminus Attr_i(\alpha^{-1}(c), \mathcal{G})$
   - 5.2 $(W'_0, W'_1) := McNaughton(\mathcal{G}')$
   - 5.3 <u>if</u> $(W'_{1-i} = \emptyset)$ <u>then</u>
     - 5.3.1 $W_i := V \setminus W_{1-i}$
     - 5.3.2 <u>return</u> (W_0, W_1)
   - 5.4 $W_{1-i} := W_{1-i} \cup Attr_{1-i}(W'_{1-i}, \mathcal{G})$
   - 5.5 $\mathcal{G} := \mathcal{G} \setminus Attr_{1-i}(W'_{1-i}, \mathcal{G})$

McNaughton's Algorithm

$McNaughton(\mathcal{G})$

1. $c :=$ highest color in $\mathcal{G}$
2. if $c = 0$ or $V = \emptyset$
   then return $(V, \emptyset)$
3. set i to $c \mod 2$
4. set W_{1-i} to $\emptyset$

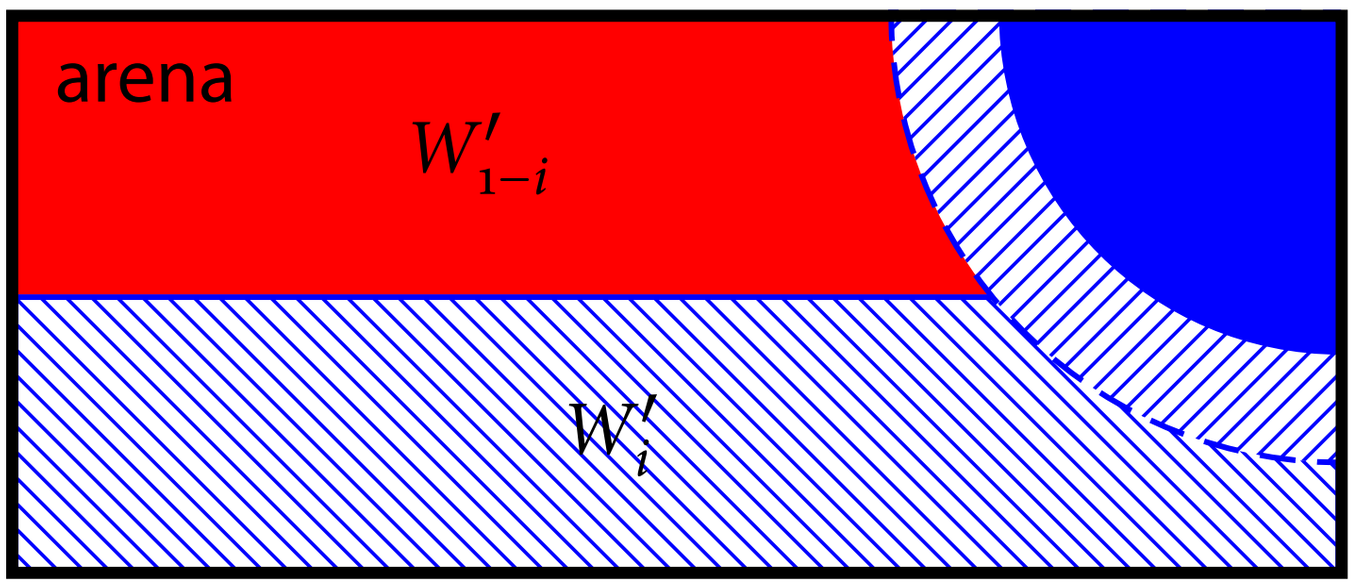


5. repeat
   - 5.1 $\mathcal{G}' := \mathcal{G} \setminus Attr_i(\alpha^{-1}(c), \mathcal{G})$
   - 5.2 $(W'_0, W'_1) := McNaughton(\mathcal{G}')$
   - 5.3 if $(W'_{1-i} = \emptyset)$ then
     - 5.3.1 $W_i := V \setminus W_{1-i}$
     - 5.3.2 return (W_0, W_1)
   - 5.4 $W_{1-i} := W_{1-i} \cup Attr_{1-i}(W'_{1-i}, \mathcal{G})$
   - 5.5 $\mathcal{G} := \mathcal{G} \setminus Attr_{1-i}(W'_{1-i}, \mathcal{G})$

McNaughton's Algorithm

$McNaughton(\mathcal{G})$

1. $c :=$ highest color in $\mathcal{G}$
2. <u>if</u> $c = 0$ or $V = \emptyset$
   <u>then return</u> $(V, \emptyset)$
3. set i to $c \mod 2$
4. set W_{1-i} to $\emptyset$

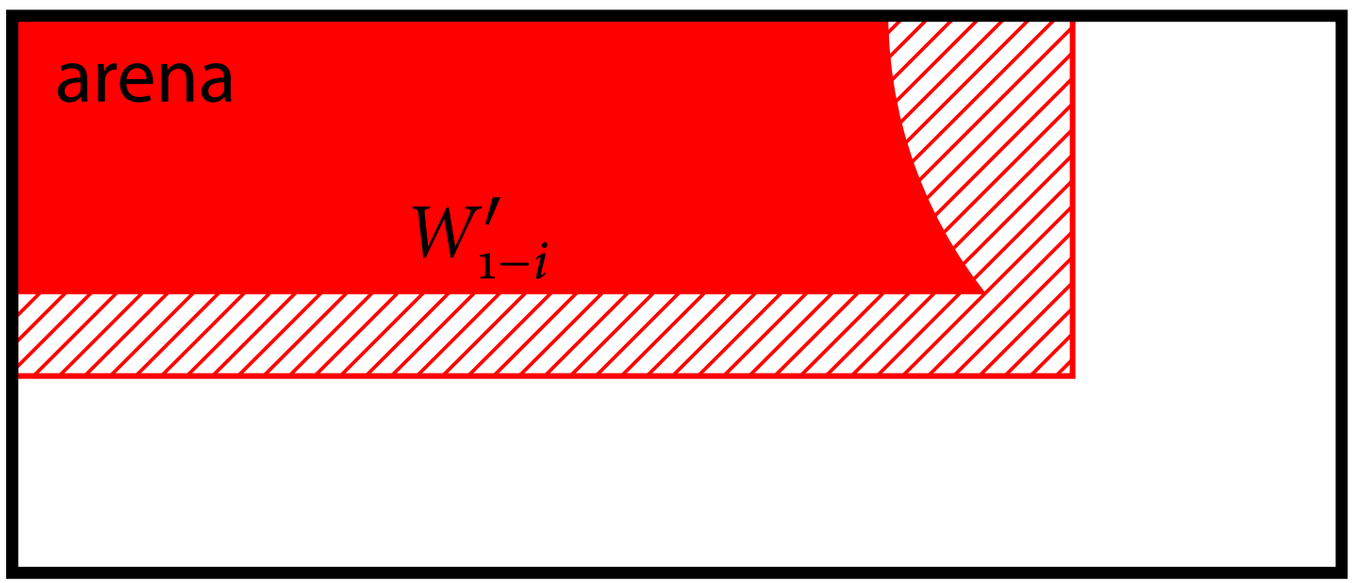


5. <u>repeat</u>
   - 5.1 $\mathcal{G}' := \mathcal{G} \setminus Attr_i(\alpha^{-1}(c), \mathcal{G})$
   - 5.2 $(W'_0, W'_1) := McNaughton(\mathcal{G}')$
   - 5.3 <u>if</u> $(W'_{1-i} = \emptyset)$ <u>then</u>
     - 5.3.1 $W_i := V \setminus W_{1-i}$
     - 5.3.2 <u>return</u> (W_0, W_1)
   - 5.4 $W_{1-i} := W_{1-i} \cup Attr_{1-i}(W'_{1-i}, \mathcal{G})$
   - 5.5 $\mathcal{G} := \mathcal{G} \setminus Attr_{1-i}(W'_{1-i}, \mathcal{G})$

McNaughton's Algorithm

$McNaughton(\mathcal{G})$

1. $c :=$ highest color in $\mathcal{G}$
2. <u>if</u> $c = 0$ or $V = \emptyset$
   <u>then return</u> $(V, \emptyset)$
3. set i to $c \mod 2$
4. set W_{1-i} to $\emptyset$



5. <u>repeat</u>
   - 5.1 $\mathcal{G}' := \mathcal{G} \setminus Attr_i(\alpha^{-1}(c), \mathcal{G})$
   - 5.2 $(W'_0, W'_1) := McNaughton(\mathcal{G}')$
   - 5.3 <u>if</u> $(W'_{1-i} = \emptyset)$ <u>then</u>
     - 5.3.1 $W_i := V \setminus W_{1-i}$
     - 5.3.2 <u>return</u> (W_0, W_1)
   - 5.4 $W_{1-i} := W_{1-i} \cup Attr_{1-i}(W'_{1-i}, \mathcal{G})$
   - 5.5 $\mathcal{G} := \mathcal{G} \setminus Attr_{1-i}(W'_{1-i}, \mathcal{G})$

Complexity

- McNaughton's algorithm has complexity $O(m\, n^{d-1})$ for games with n states, m edges, and d colors.
- Currently best known algorithm: $O(m\, n^{\frac{d}{3}})$
- **Open problem:** is solving parity games in polynomial time?

Overview

Part I. Infinite Games

Fundamental algorithms to solve infinite games played over finite graphs.

Part II. Synthesis from Logical Specifications

Synthesis from specifications given as formulas of a temporal logic. The quest for an efficient and expressive specification language.

Part III. Bounded Synthesis

Finding simple solutions fast. The quest for structurally simple implementations.

Part IV. Distributed Synthesis

Synthesizing systems that consist of multiple distributed components.

Part II: Synthesis from Logical Specifications

1. **Linear-time temporal logic (LTL)**
2. Branching-time temporal logic (CTL/CTL*)
3. GR(1)

Linear-time temporal logic

Linear-time temporal logic (LTL) (Pnueli, 1977)

- **propositional logic**
  - p for some atomic proposition p
  - $\neg\varphi$
  - $\varphi \wedge \psi$
- **temporal operators**
  - X φ
  - φ U ψ

  derived operators:
  - F $\varphi \equiv \mathit{true}$ U φ
  - G $\varphi \equiv \neg(\text{F } \neg\varphi)$

Example: Synthesis of an arbiter circuit

An arbiter circuit receives **requests** r_1, r_2 from two clients and produces **grants** g_1, g_2.

The **specification** φ of the arbiter is the conjunction of the following properties:

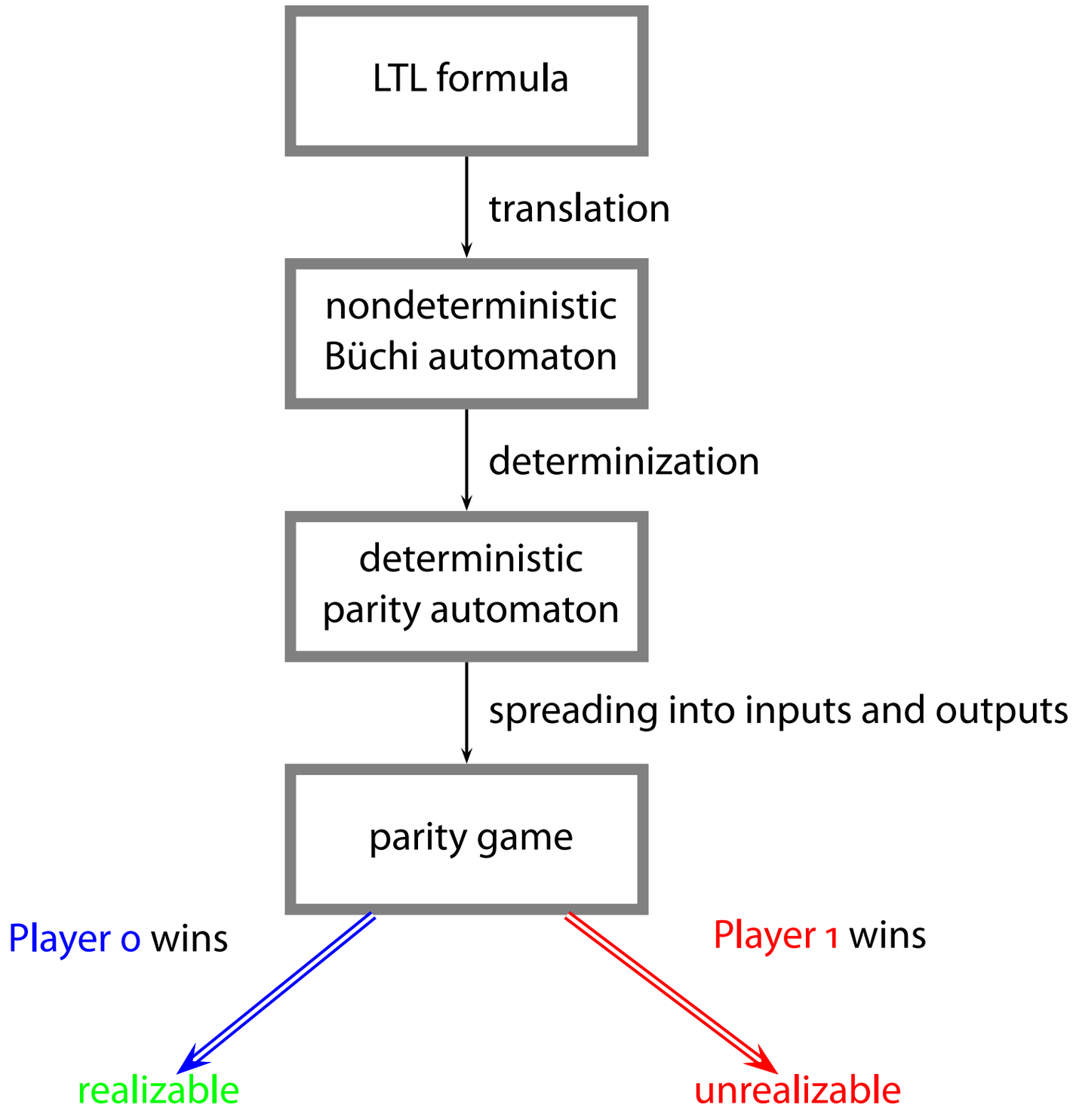
1. **Mutual exclusion:** at no point in time should there be both g_1 and g_2 in the output.

$$\mathsf{G}\,\neg(g_1 \wedge g_2)$$

2. **Response:** every request r_i from the client i (for $i \in \{1, 2\}$) should eventually be followed by grant g_i for client i.

$$\mathsf{G}\,r_1 \Rightarrow \mathsf{X}\,\mathsf{F}\,g_1$$
$$\wedge$$
$$\mathsf{G}\,r_2 \Rightarrow \mathsf{X}\,\mathsf{F}\,g_2$$

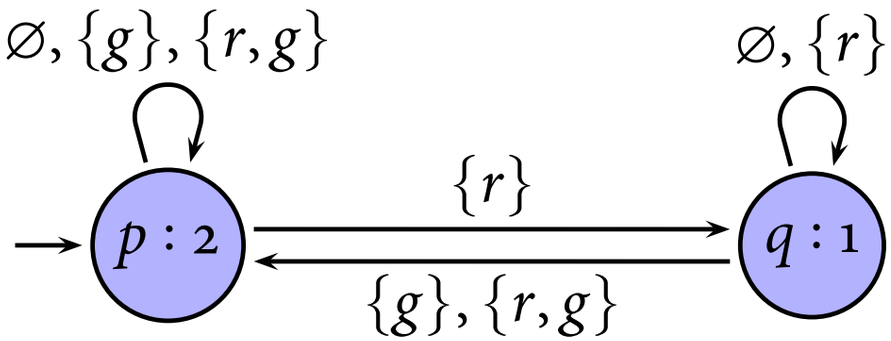
LTL synthesis

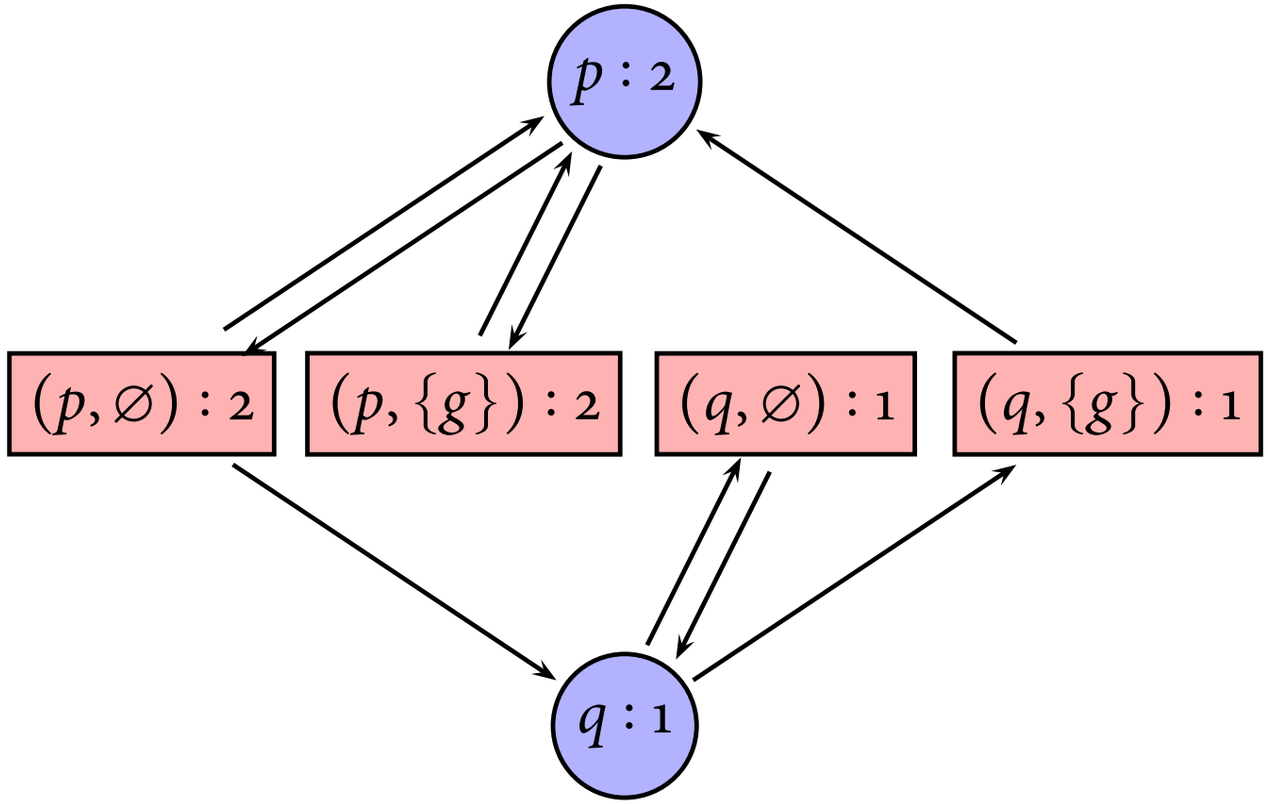


Simplified example

Simple response property:

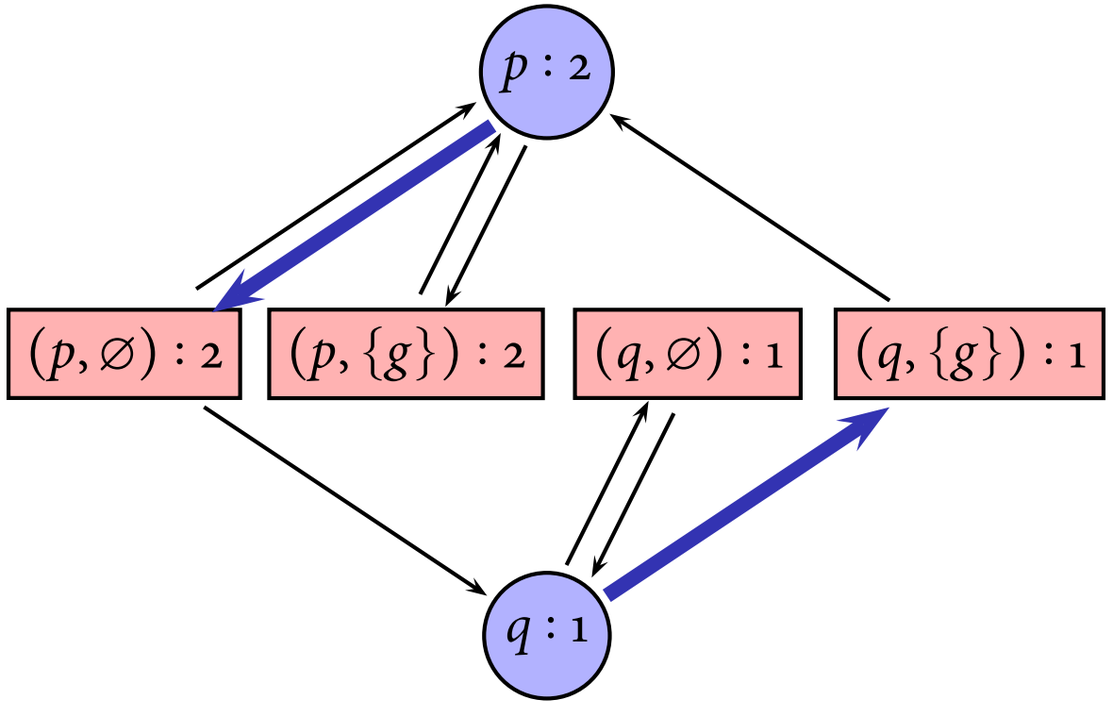
$$\mathsf{G}\,(r \Rightarrow \mathsf{F}\,g)$$

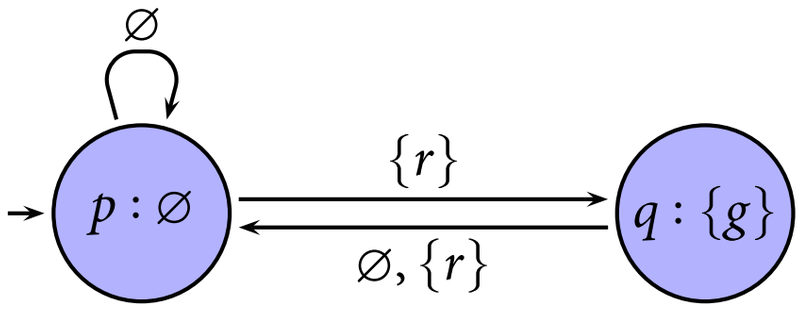




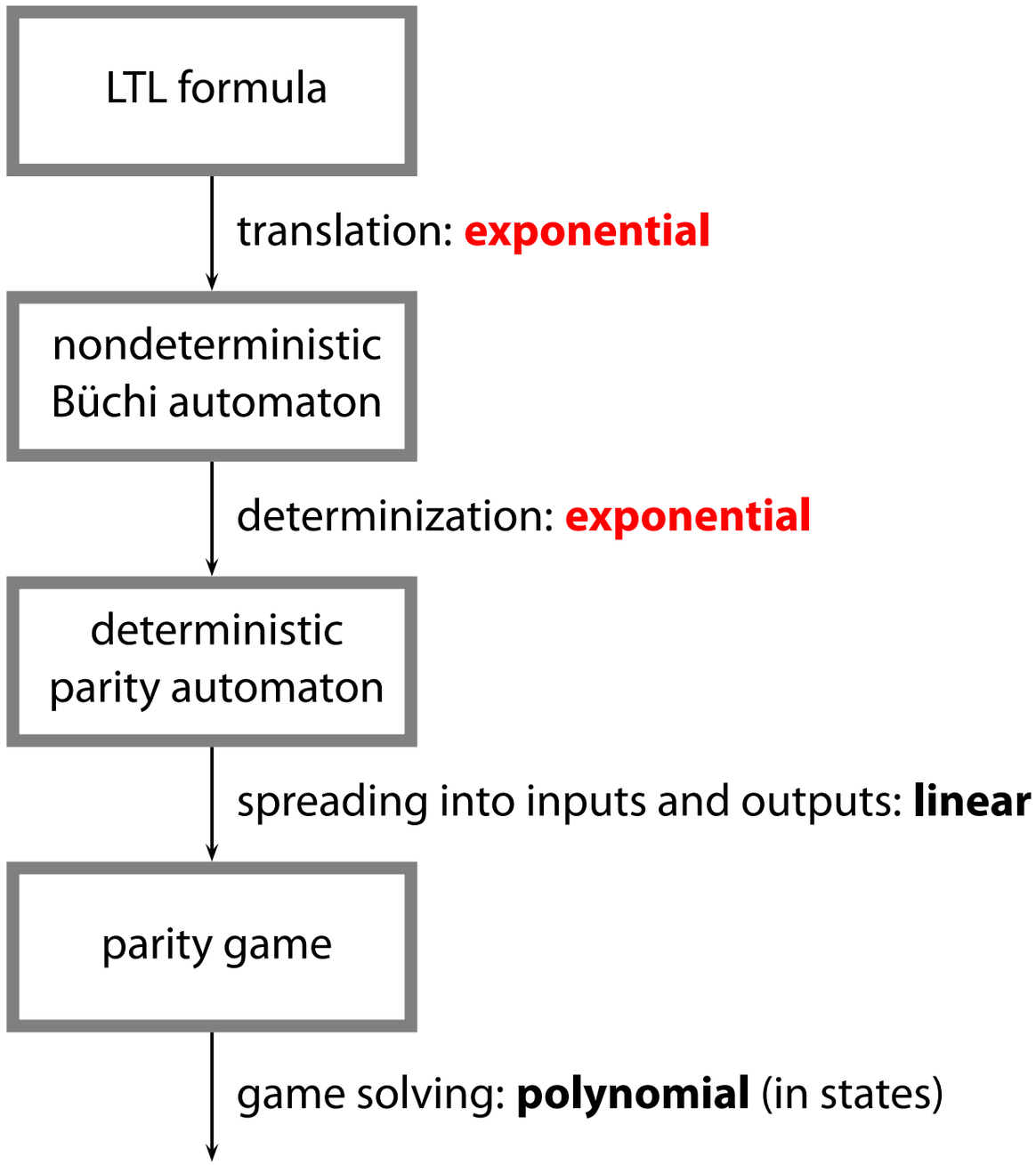
From strategies to implementations

A winning strategy for the Player o in the parity game can be represented as an FSM.





Complexity



LTL synthesis

LTL synthesis is 2EXPTIME complete.

Why deterministic parity automata?

Why deterministic automata?

- Nondeterministic automata may have rejecting runs on accepted sequences.
- Player o may lose a play just because the wrong run was chosen.
- **Example:** $(\mathsf{F}\,\mathsf{G}\,(i \wedge \mathsf{X}o)) \vee (\mathsf{G}\,\mathsf{F}\,(\neg i \wedge \mathsf{X}\neg o))$
  - Player o has a winning strategy (copy input i to output o)
  - Player o cannot choose between the two disjuncts.

Why not deterministic Büchi automata?

- Not every LTL formula can be translated into an equivalent deterministic Büchi automaton.
- Example: $\mathsf{F}\,\mathsf{G}\,p$

Part II: Synthesis from Logical Specifications

1. Linear-time temporal logic (LTL)
2. **Branching-time temporal logic (CTL/CTL*)**
3. GR(1)

Branching-time temporal logics

CTL* (Emerson/Halpern 1985)

- CTL* **state formulas**:

$$\Phi ::= p \mid \Phi_1 \wedge \Phi_2 \mid \neg\Phi \mid \mathsf{E}\varphi \mid \mathsf{A}\varphi$$

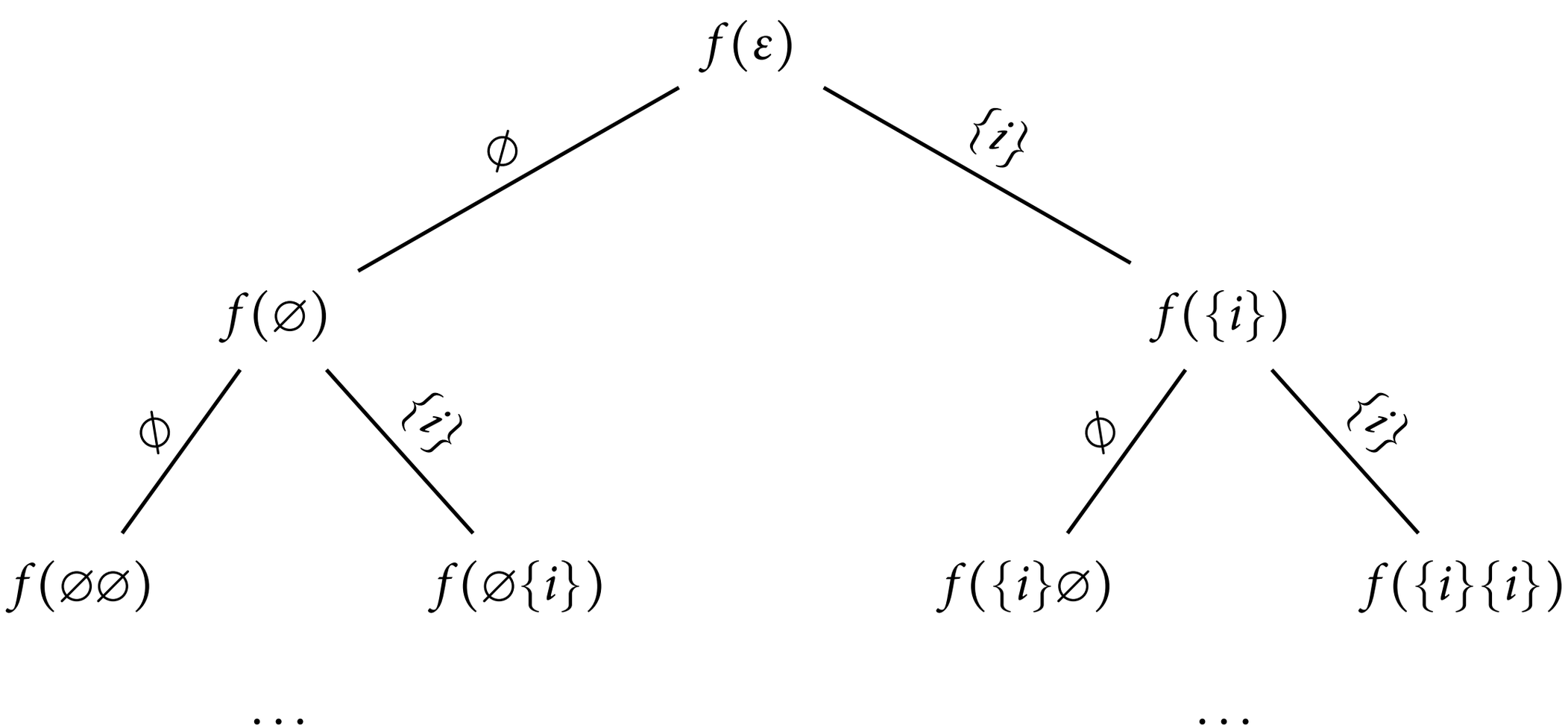
- CTL* **path formulas**:

$$\varphi ::= \Phi \mid \varphi_1 \wedge \varphi_2 \mid \neg\varphi \mid \mathsf{X}\varphi \mid \varphi_1 \,\mathcal{U}\, \varphi_2$$

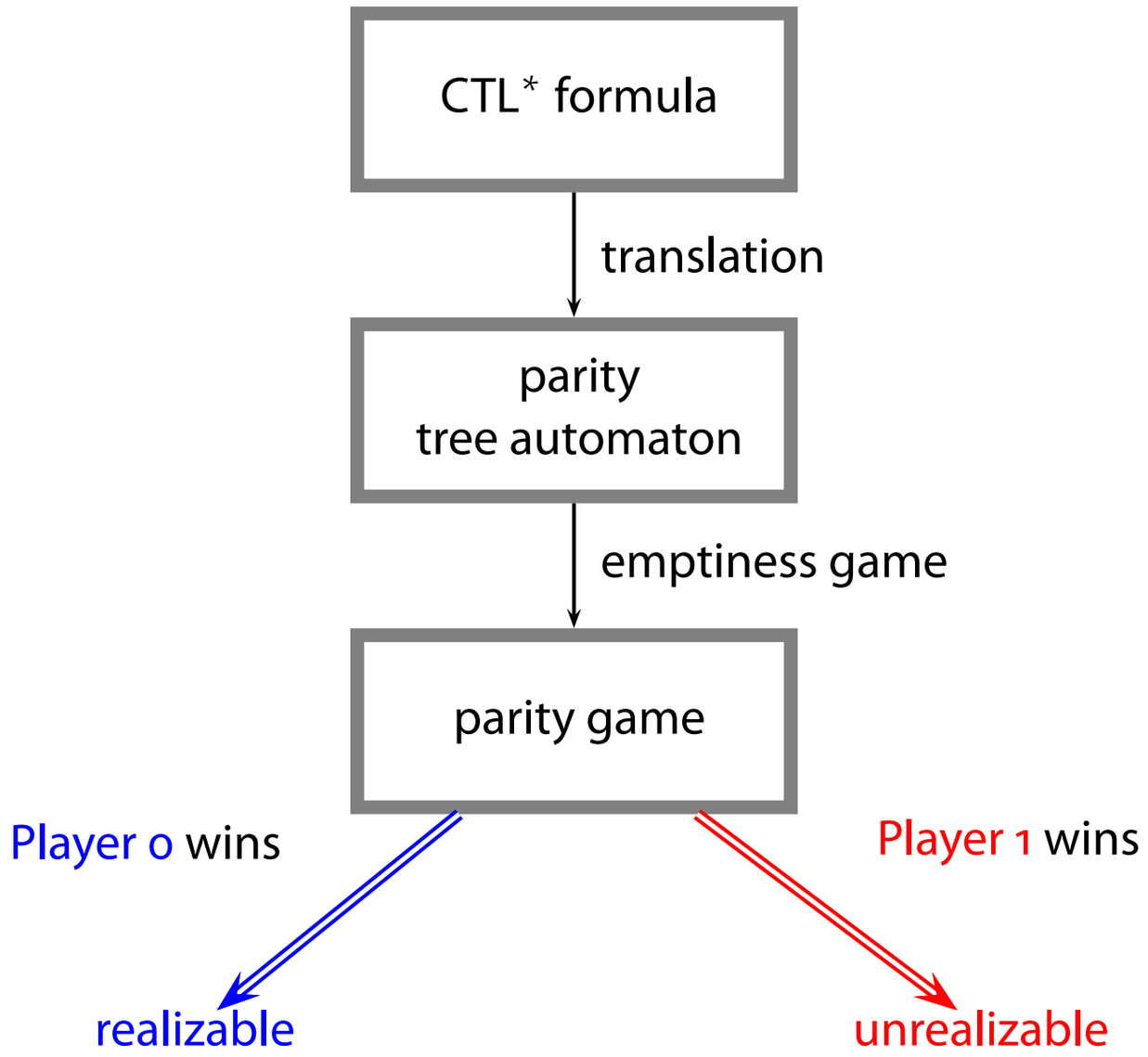
Strategy trees

- For a given set Υ of **directions**,
  the **(full infinite) tree** is the set Υ^* of finite sequences over Υ.
- A Σ**-labeled** Υ**-tree** is a function $\Upsilon^* \to \Sigma$.

A strategy can be seen as a *Output*-labeled *Input*-tree:



CTL* synthesis



Tree automata

We use tree automata to represent sets of strategies.

A **parity tree automaton** over Σ-labeled Υ-trees is a tuple $\mathcal{A} = (Q, q_0, T, \alpha)$, where

- Q is a finite set of **states**,
- $q_0 \in Q$ is an **initial state**,
- $T \subseteq Q \times \Sigma \times (\Upsilon \to Q)$ is a set of **transitions**, and
- $\alpha : Q \to \mathbb{N}$ is a coloring function.

A run of the tree automaton on a given tree annotates each node of the tree with a state of the tree automaton such that each node satisfies, together with its children, some transition.
The tree is accepted, if, on every path, the highest color that is visited infinitely often is even.

Example

Existential response property:

$$\mathsf{A}\,\mathsf{G}\,(r \Rightarrow \mathsf{E}\,\mathsf{F}\,g)$$

$$Q = \{p, q\}$$
$$q_0 = p$$
$$\begin{aligned} T = \{&(p, \varnothing, \varnothing \mapsto p, \{r\} \mapsto q) \\ &(p, \{g\}, \varnothing \mapsto p, \{r\} \mapsto p) \\ &(q, \varnothing, \varnothing \mapsto p, \{r\} \mapsto q) \\ &(q, \{g\}, \varnothing \mapsto p, \{r\} \mapsto p)\} \end{aligned}$$
$$\alpha(p) = 2, \alpha(q) = 1$$

Emptiness game

Language emptiness of a tree automaton can be checked by solving the **emptiness game**. The language is **non-empty** iff Player 0 **wins**.

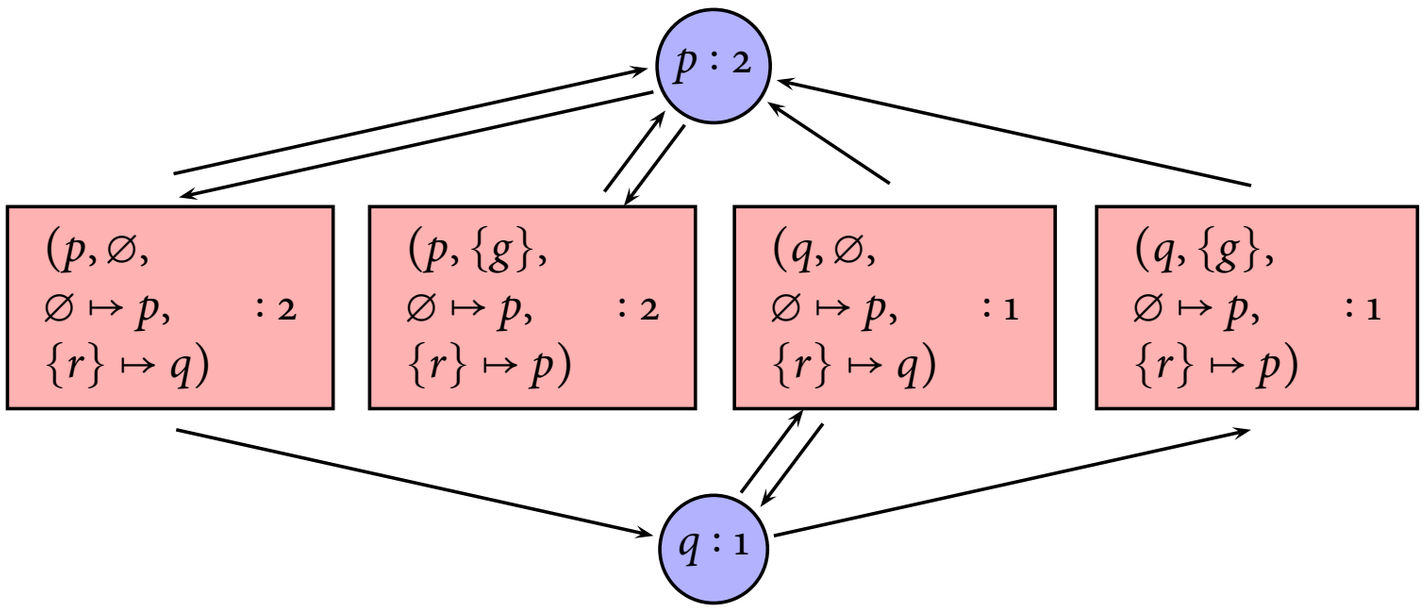
- Player 0 picks transitions
- Player 1 picks successor state

$Q = \{p, q\}$

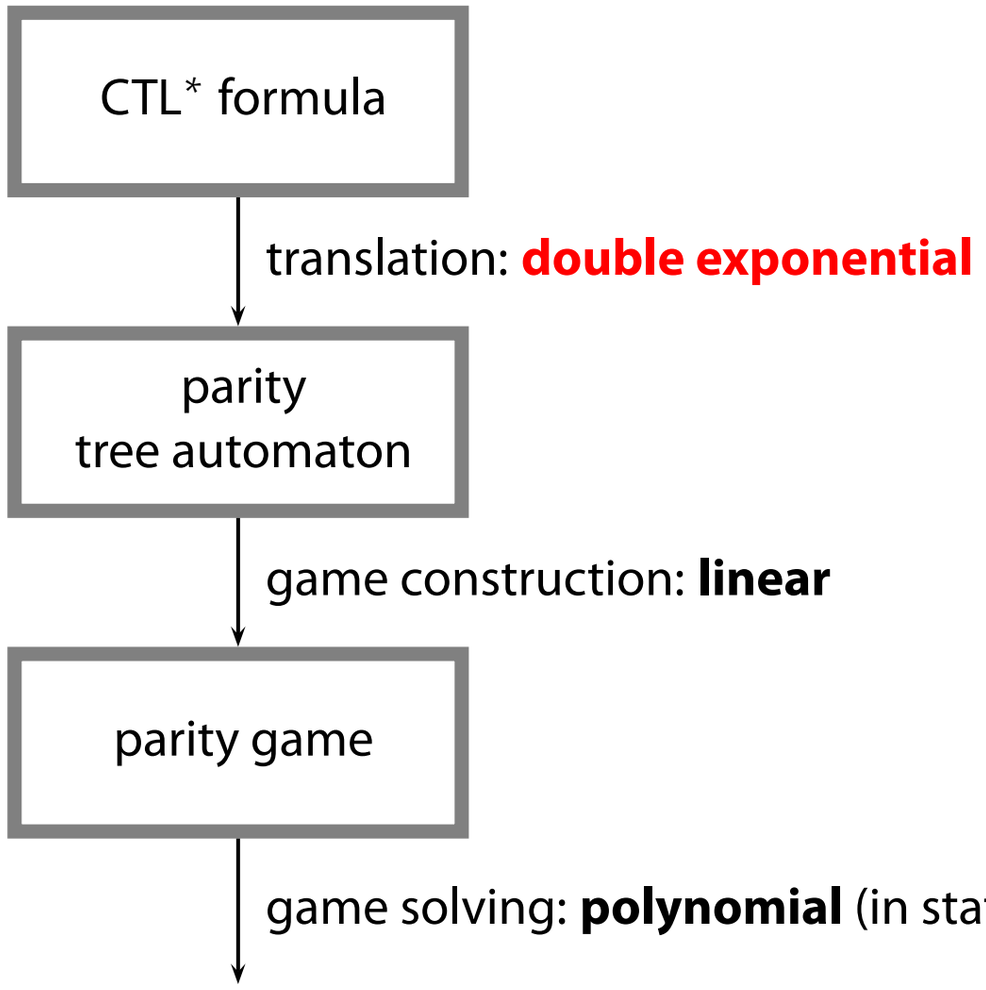
$q_0 = p$

$T = \{(p, \varnothing, \varnothing \mapsto p, \{r\} \mapsto q)$
$(p, \{g\}, \varnothing \mapsto p, \{r\} \mapsto p)$
$(q, \varnothing, \varnothing \mapsto p, \{r\} \mapsto q)$
$(q, \{g\}, \varnothing \mapsto p, \{r\} \mapsto p)\}$

$\alpha(p) = 2, \alpha(q) = 1$



Complexity



CTL* synthesis

CTL* synthesis is 2EXPTIME complete.

Branching-time temporal logics

CTL (Clarke/Emerson 1982)

CTL* with the restriction that every temporal operator is immediately preceded by a path quantifier.

CTL synthesis is EXPTIME-complete.

translation CTL → tree automaton **single exponential**

(Vardi/Wolper 1986)

Problem with CTL

It is difficult to specify environment assumptions in CTL.

Example:

- "Every request is followed by a grant" can be expressed:
  $\mathsf{A}\,\mathsf{G}\,(r \Rightarrow \mathsf{A}\,\mathsf{F}\,g)$
- "If there are infinitely many requests then there are infinitely many grants" cannot be expressed.

Part II: Synthesis from Logical Specifications

1. Linear-time temporal logic (LTL)
2. Branching-time temporal logic (CTL/CTL*)
3. **GR(1)**

General Reactivity (1)

GR(1) specifications have the following form:

$$A_1 \wedge A_2 \wedge \ldots \wedge A_m \;\Rightarrow\; G_1 \wedge G_2 \wedge \ldots \wedge G_n,$$

Assumptions A_i and **guarantees** G_i are restricted to the following types of formulas:

- **initialization properties**: state formulas
- **safety properties** of the form $\mathsf{G}\,(\varphi \rightarrow \mathsf{X}\,\psi)$,
- **liveness properties** of the form $\mathsf{G}\,\mathsf{F}\,\varphi$.

Example: $(\mathsf{G}\,\mathsf{F}\,r) \;\Rightarrow (\mathsf{G}\,\mathsf{F}\,g)$

General Reactivity (1)

GR(1) games are (comparatively) small:

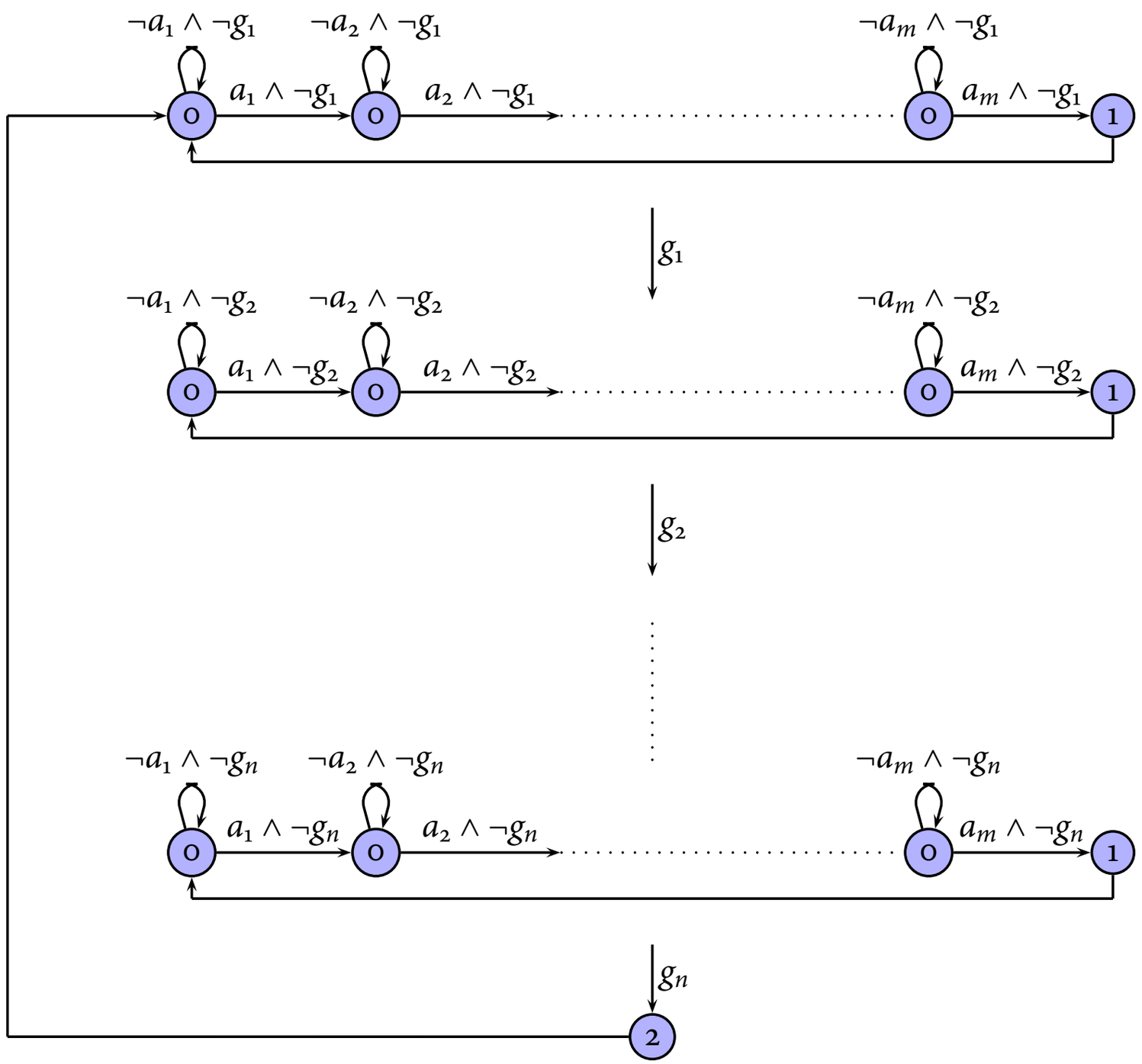
- safety properties of the form $\mathsf{G}(\varphi \rightarrow \mathsf{X}\,\psi)$
  - lead to an exponential state space (keep track of the "active" X-formulas)
  - lead to states can be stored efficiently using BDDs and other symbolic data structures
- liveness properties properties of the form $\mathsf{G}\,\mathsf{F}\,\varphi$ lead to parity games with 3 colors (see next slide)

GR (1) realizability can be checked in exponential time.

(Piterman/Pnueli/Sa'ar, 2006)

General Reactivity (1)

$$(\mathsf{G}\,\mathsf{F}\,a_1)\wedge(\mathsf{G}\,\mathsf{F}\,a_2\wedge)\wedge\ldots\wedge(\mathsf{G}\,\mathsf{F}\,a_m)\;\rightarrow\;(\mathsf{G}\,\mathsf{F}\,g_1)\wedge(\mathsf{G}\,\mathsf{F}\,g_2)\wedge\ldots\wedge(\mathsf{G}\,\mathsf{F}\,g_n)$$



Pre-synthesis

- Specifications often cannot directly be expressed in GR(1). For example:

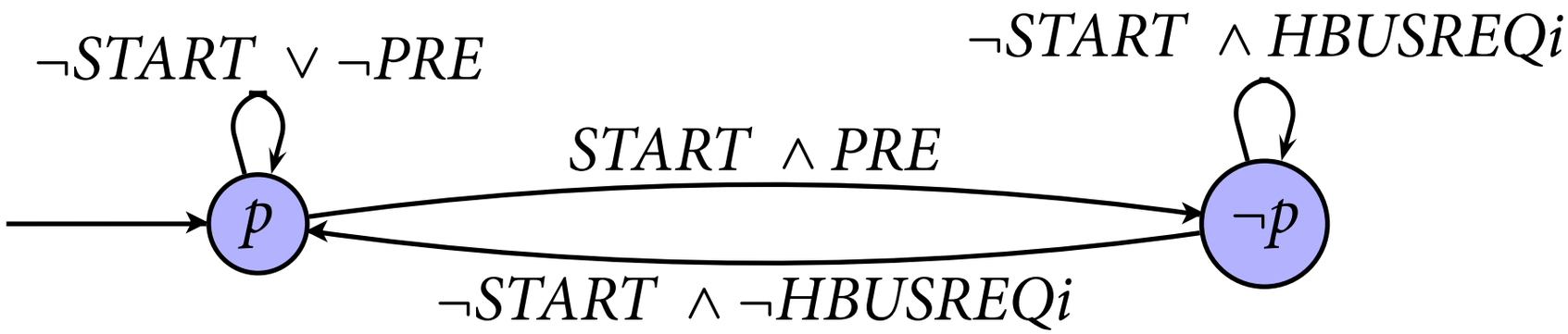
$$\mathsf{G}\left((START \wedge PRE) \Rightarrow \mathsf{X}\left(\neg START\ \mathsf{U}\ \neg HBUSREQi\right)\right)$$

(from the AMBA specification) requires an Until operator.

- **Pre-synthesis** often sidesteps this problem: synthesize a deterministic monitor automaton such that the property can be expressed in terms of the states of the monitor. The monitor itself is specified in GR(1).
- The construction of the monitor is double exponential...

Pre-synthesis

Example: $\mathsf{G}\left((START \wedge PRE) \Rightarrow \mathsf{X}\left(\neg START\ \mathsf{U}\ \neg HBUSREQi\right)\right)$



$$
\begin{aligned}
& p \\
\wedge\quad & \mathsf{G}\left(p \wedge \left(\neg START \vee \neg PRE\right)\right) \Rightarrow \mathsf{X}p \\
\wedge\quad & \mathsf{G}\left(p \wedge START \wedge PRE\right) \Rightarrow \mathsf{X}\neg p \\
\wedge\quad & \mathsf{G}\,\neg p \Rightarrow \neg START \\
\wedge\quad & \mathsf{G}\left(\neg p \wedge HBUSREQi\right) \Rightarrow \mathsf{X}\neg p \\
\wedge\quad & \mathsf{G}\left(\neg p \wedge \neg HBUSREQi\right) \Rightarrow \mathsf{X}\neg p \\
\wedge\quad & \mathsf{G}\,\mathsf{F}\,p
\end{aligned}
$$

Overview

Part I. Infinite Games

Fundamental algorithms to solve infinite games played over finite graphs.

Part II. **Synthesis from Logical Specifications**

Synthesis from specifications given as formulas of a temporal logic. The quest for an efficient and expressive specification language.

Part III. Bounded Synthesis

Finding simple solutions fast. The quest for structurally simple implementations.

Part IV. Distributed Synthesis

Synthesizing systems that consist of multiple distributed components.

Bounded synthesis

Synthesis

- Is there an implementation that satisfies the specification?

Bounded Synthesis

- Is there an implementation with **no more than N states**?

[F./Schewe, 2007]

From input to output complexity

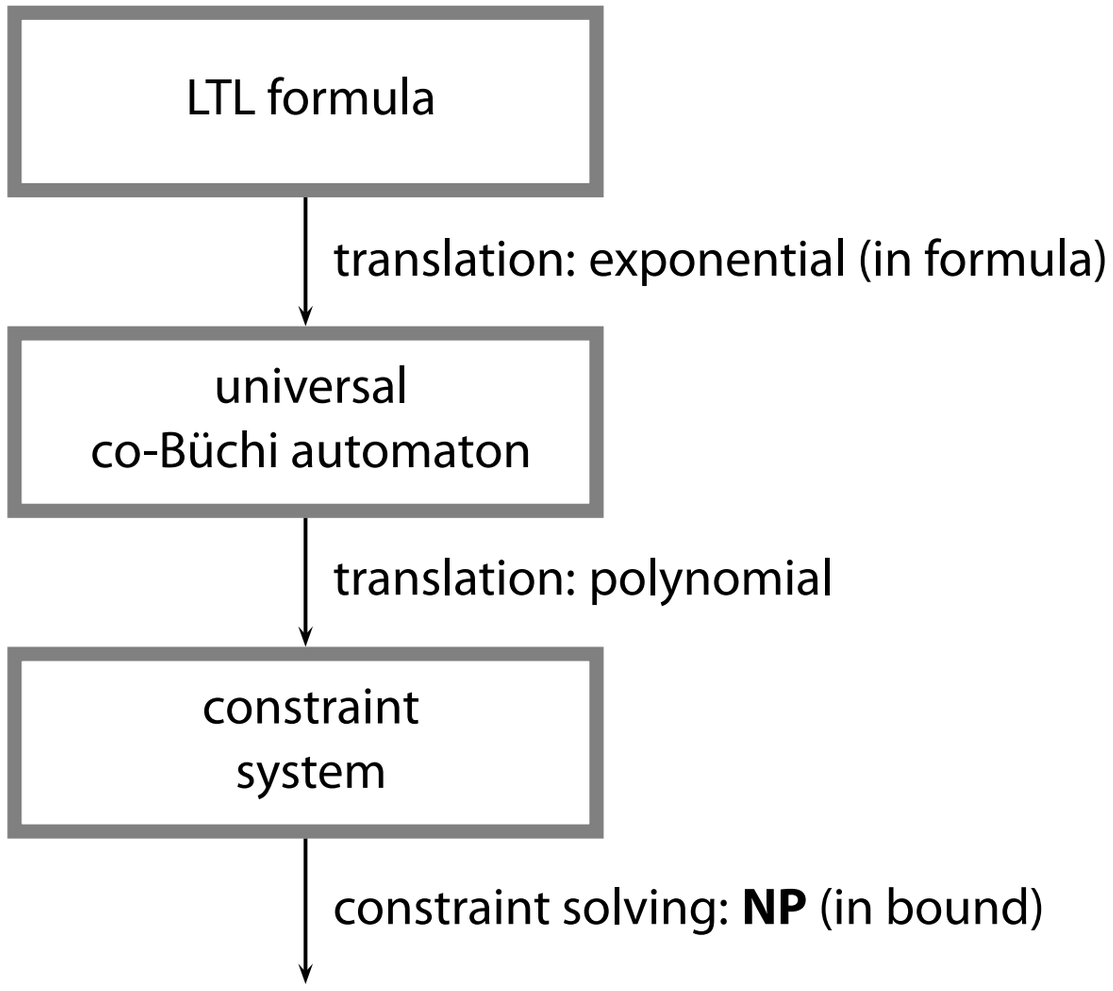
Synthesis

- 2EXPTIME in length of LTL formula (input)

Bounded Synthesis

- NP-complete in size of implementation (output)

Bounded LTL synthesis



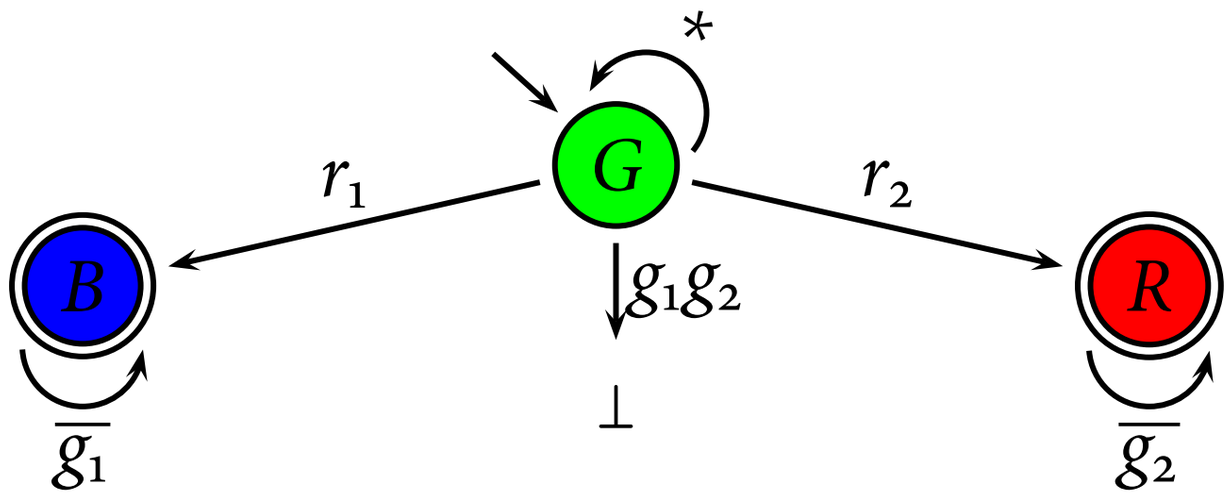
Output complexity

LTL synthesis is NP-complete in the size of implementation.

Part III: Bounded Synthesis

1. Universal co-Büchi automata
2. Constraint systems
3. Towards structurally simple implementations

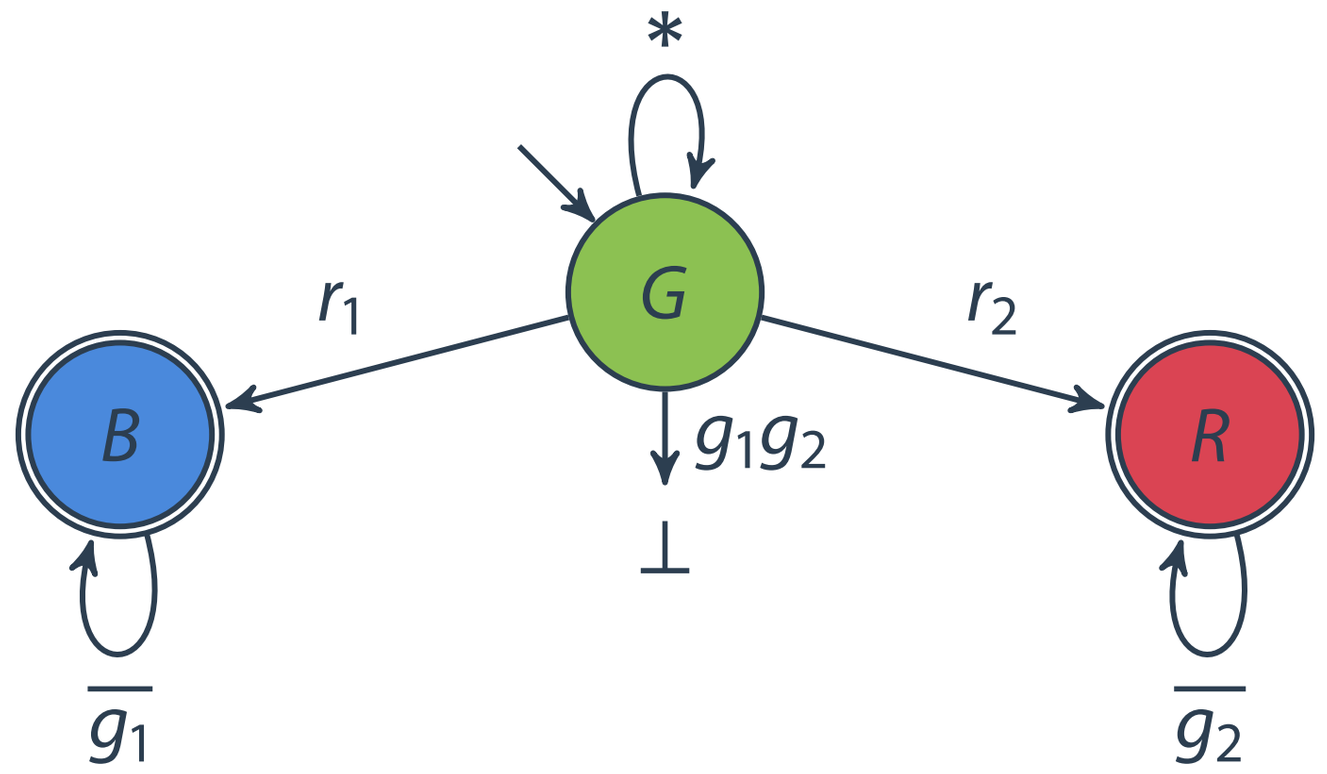
Universal co-Büchi automata

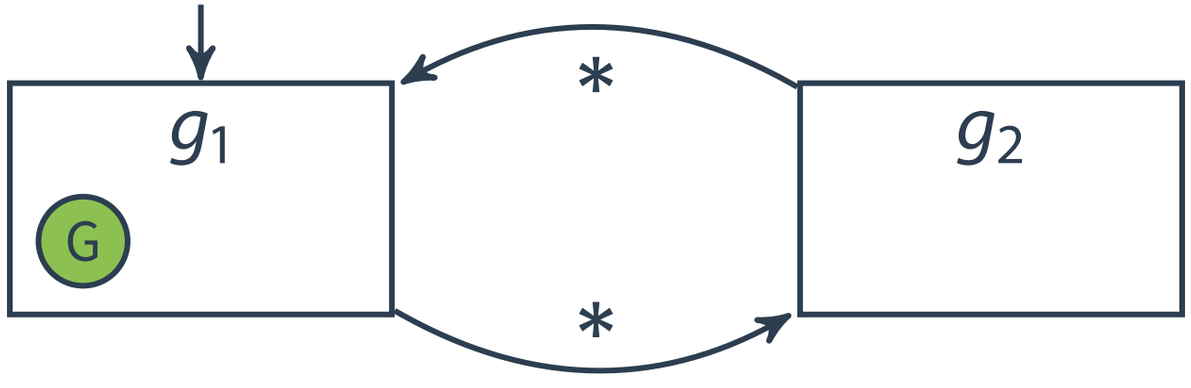


Example: Arbiter

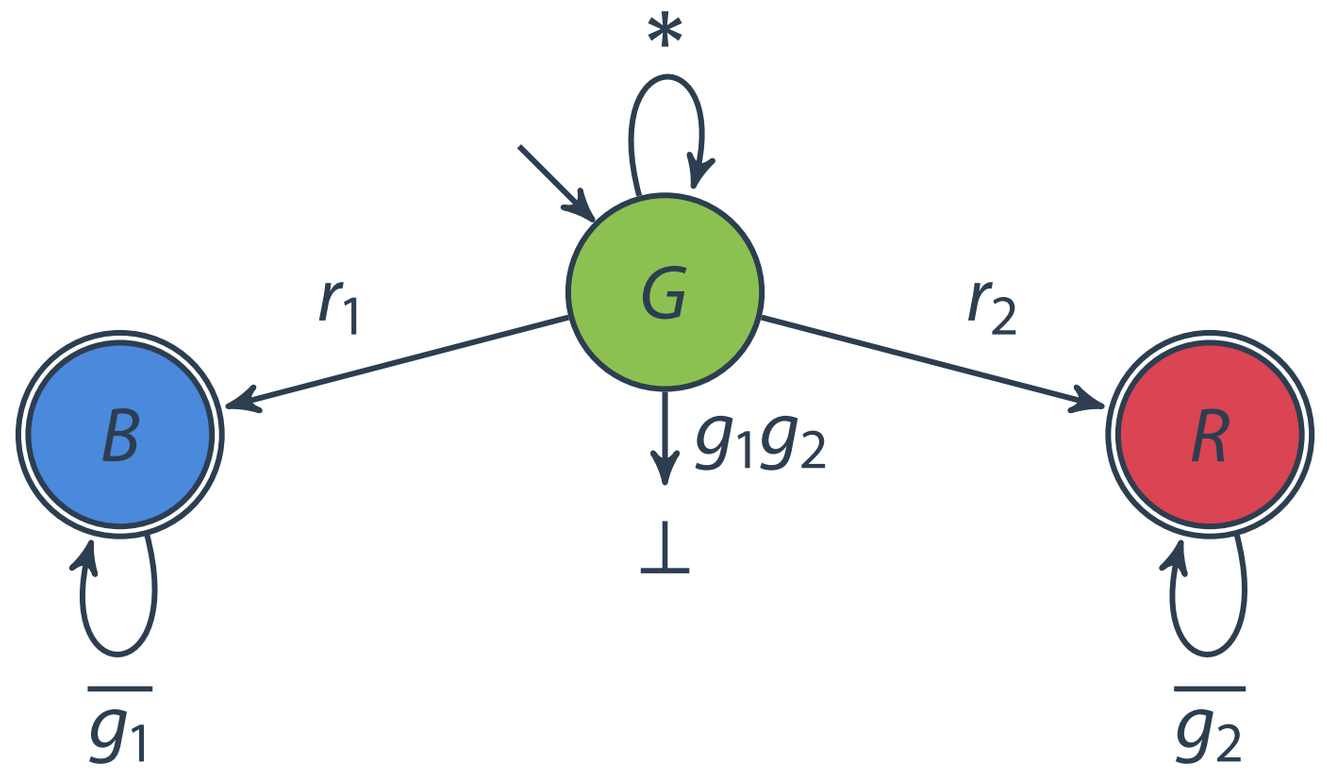
$$\varphi = \mathsf{G}\,(r_1 \rightarrow \mathsf{XF}\,g_1) \wedge \mathsf{G}\,(r_2 \rightarrow \mathsf{XF}g_2) \wedge \mathsf{G}\neg(g_1 \wedge g_2)$$

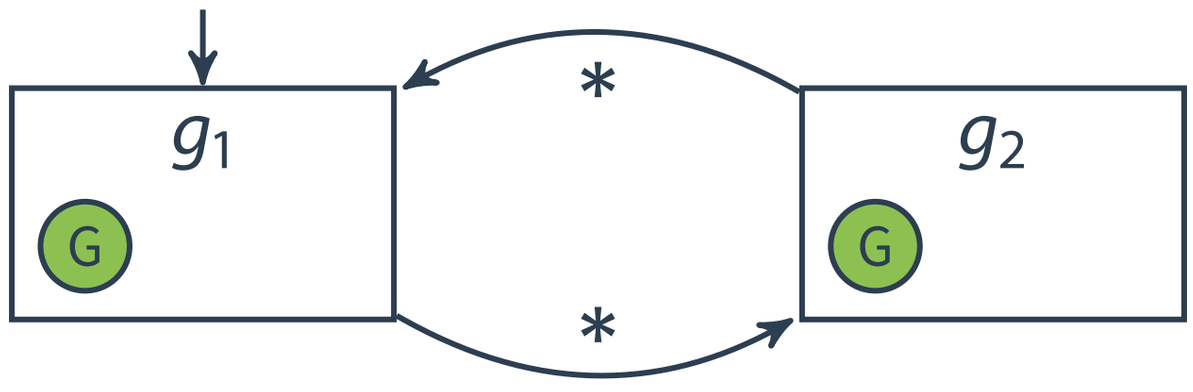
Acceptance of an FSM



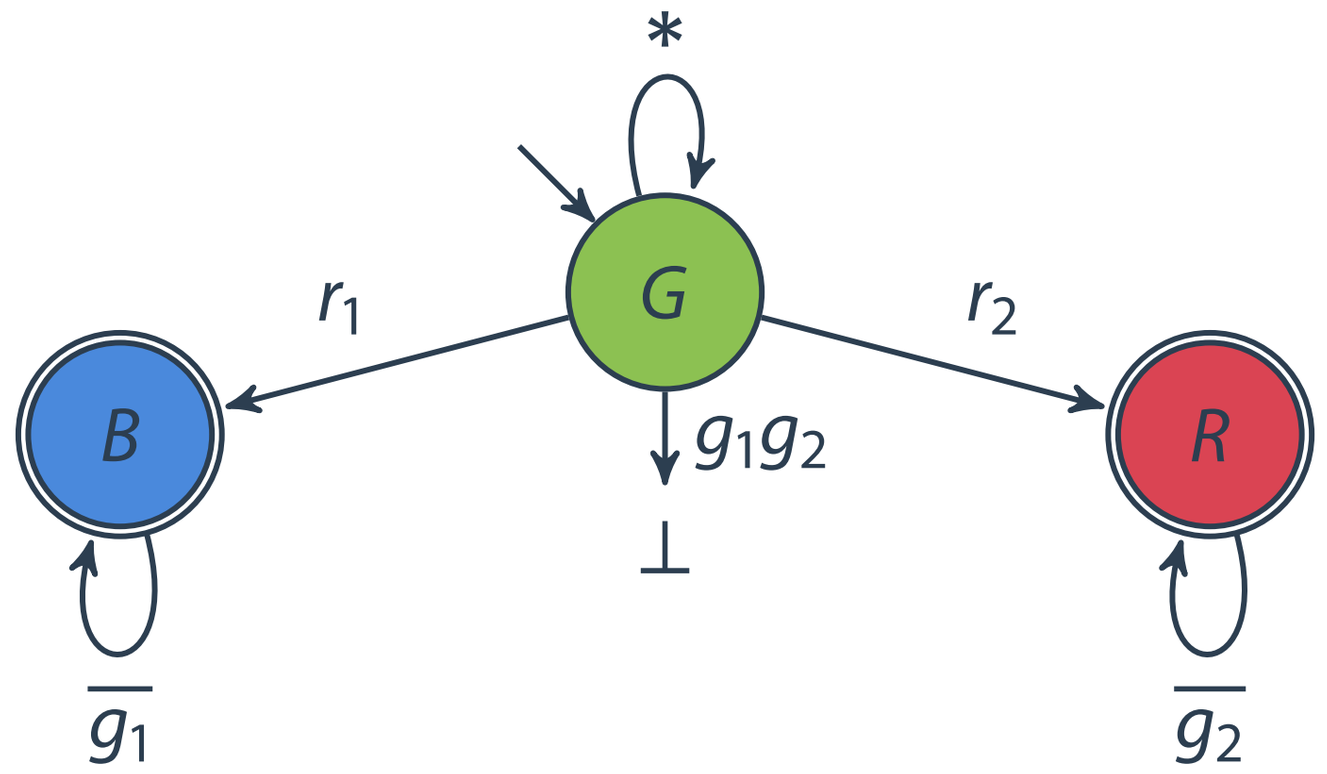


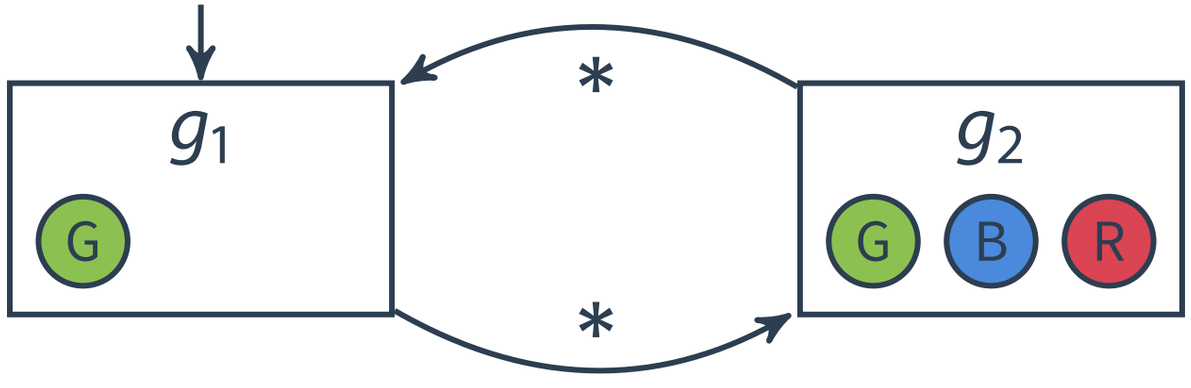
Acceptance of an FSM



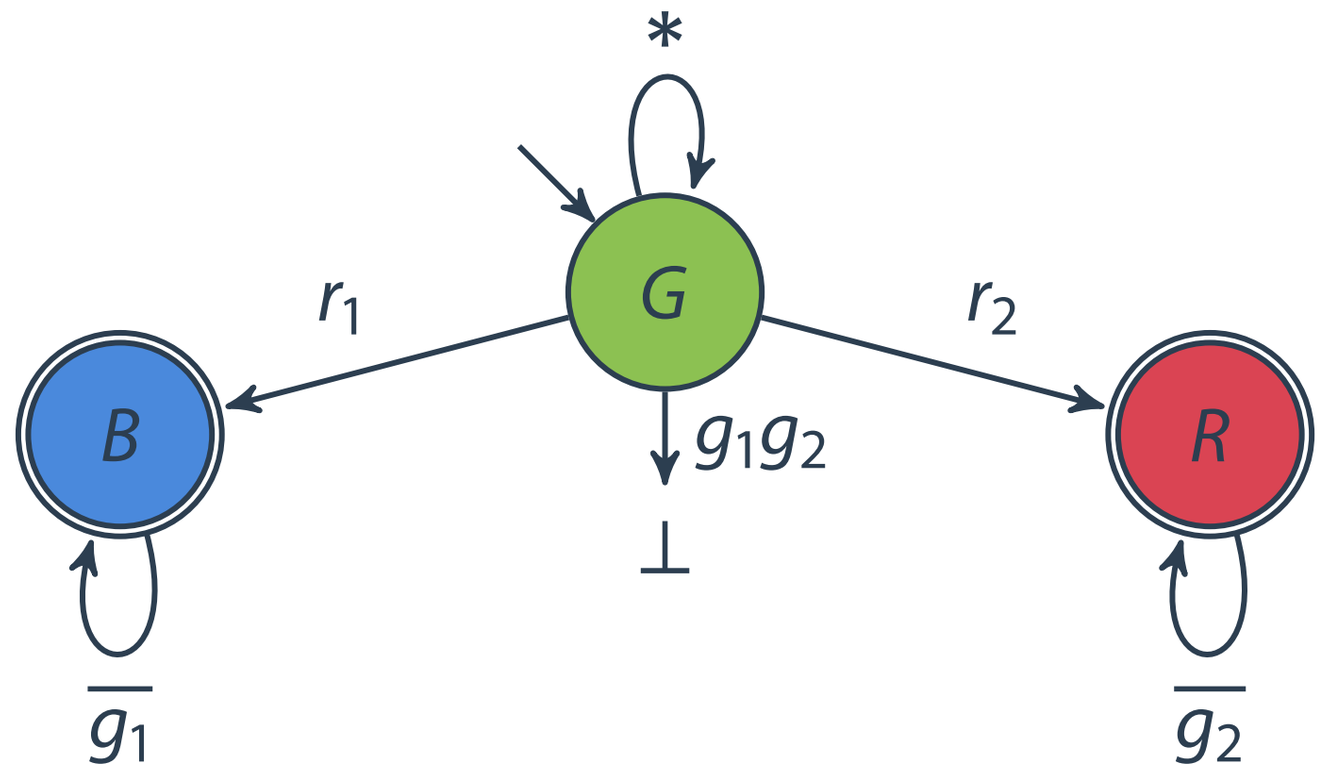


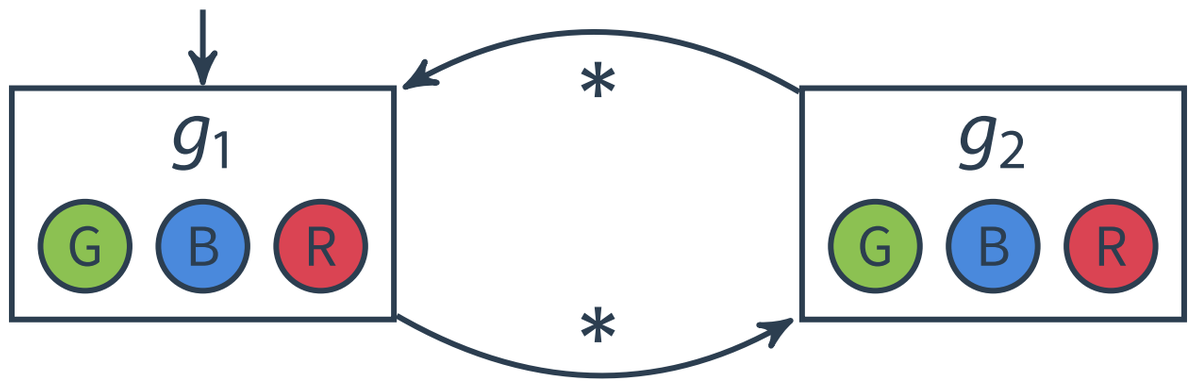
Acceptance of an FSM



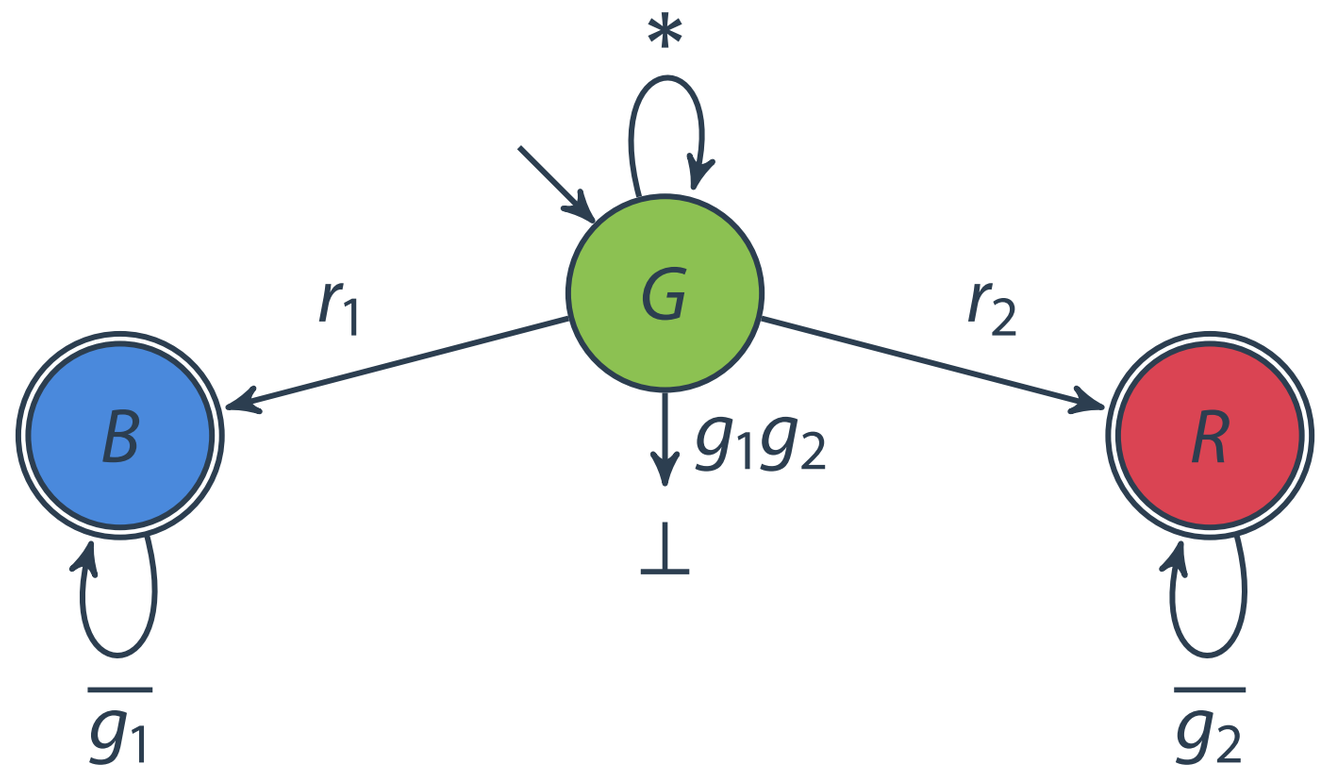


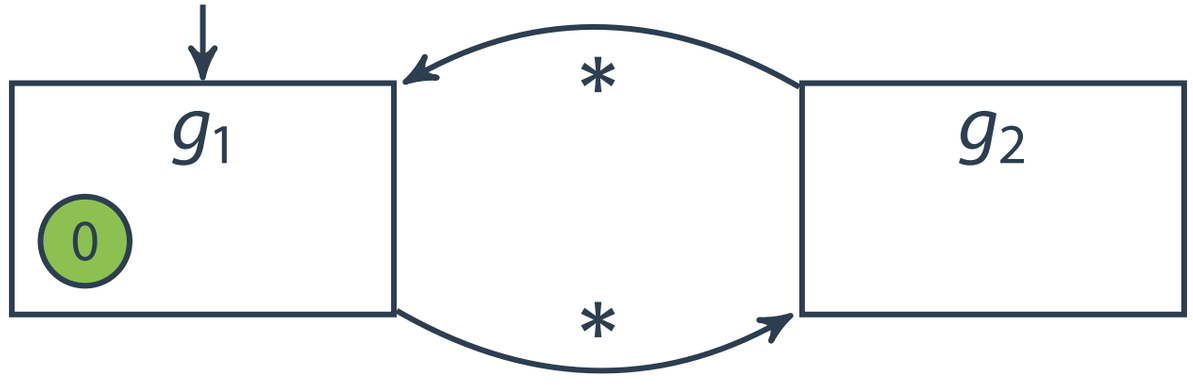
Acceptance of an FSM



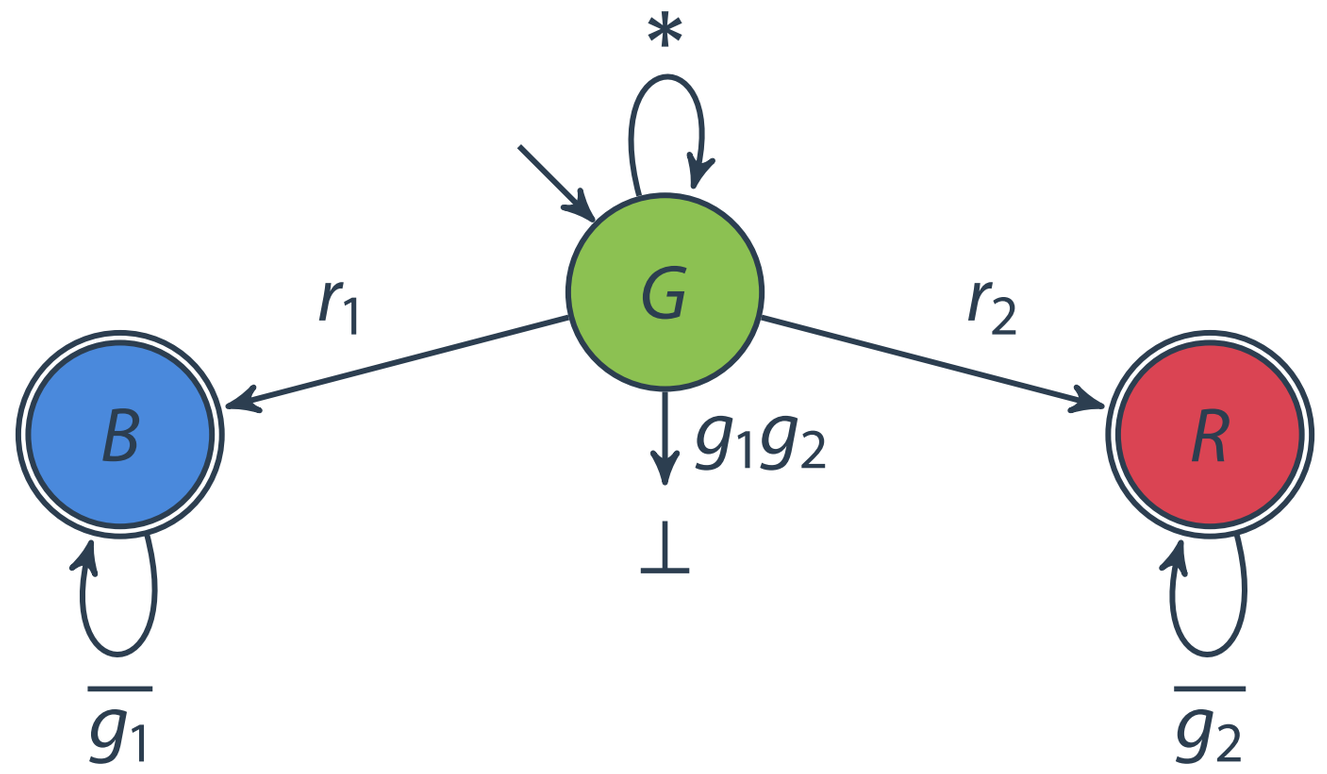


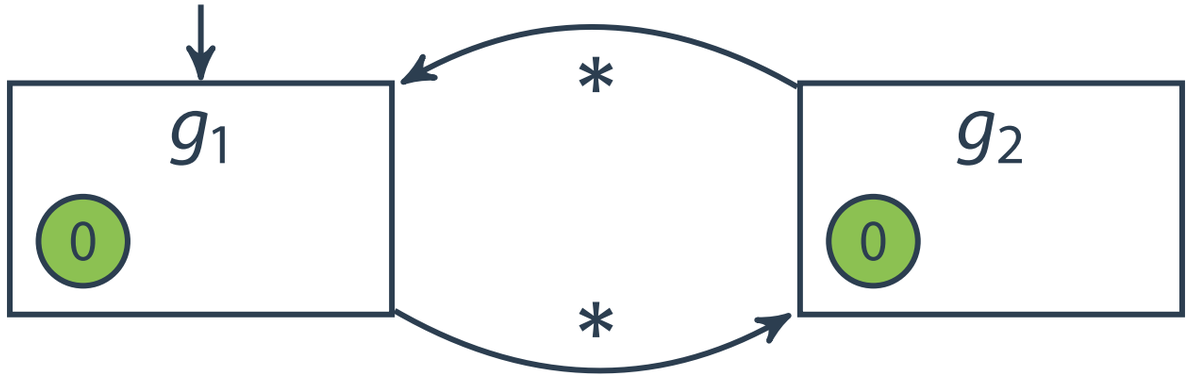
Acceptance of an FSM



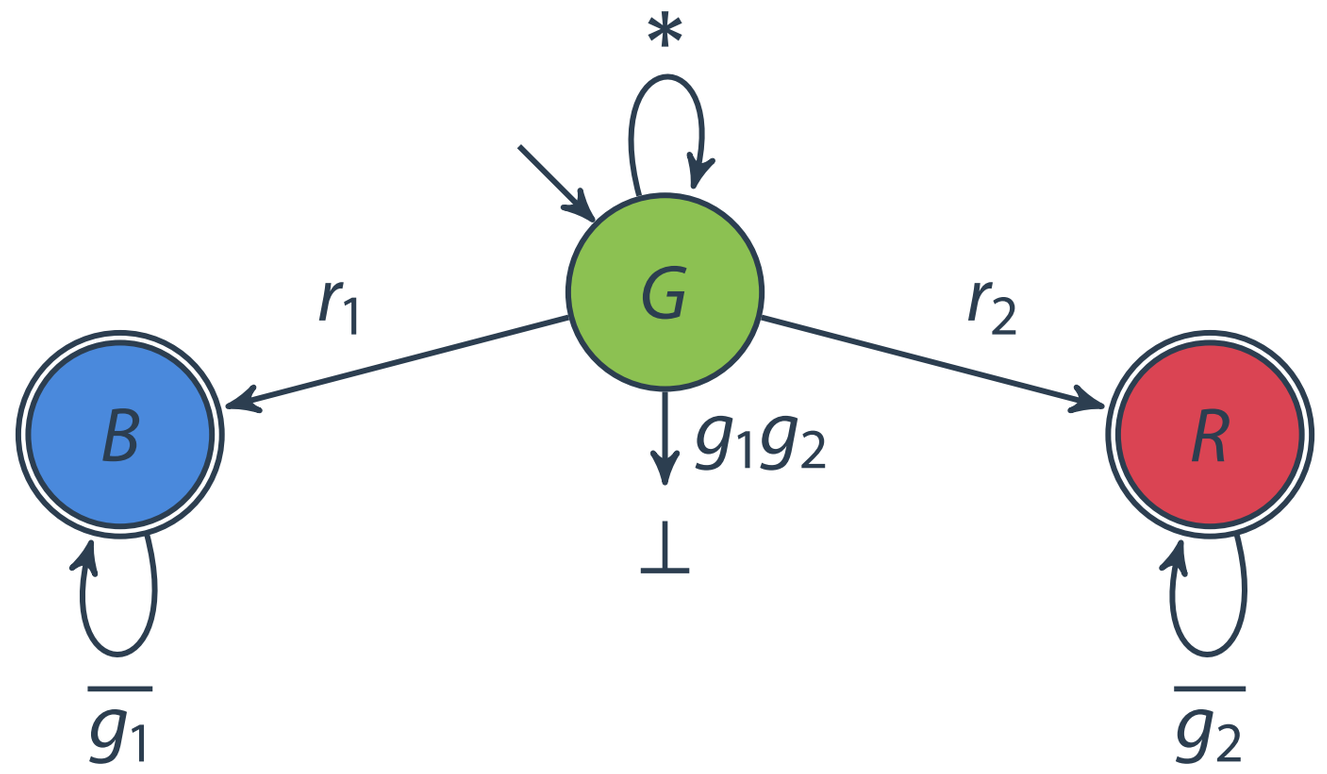


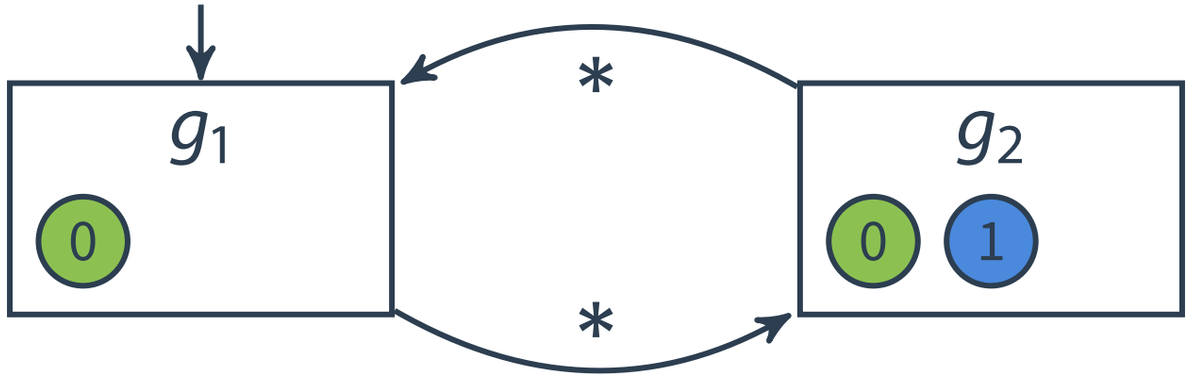
Acceptance of an FSM



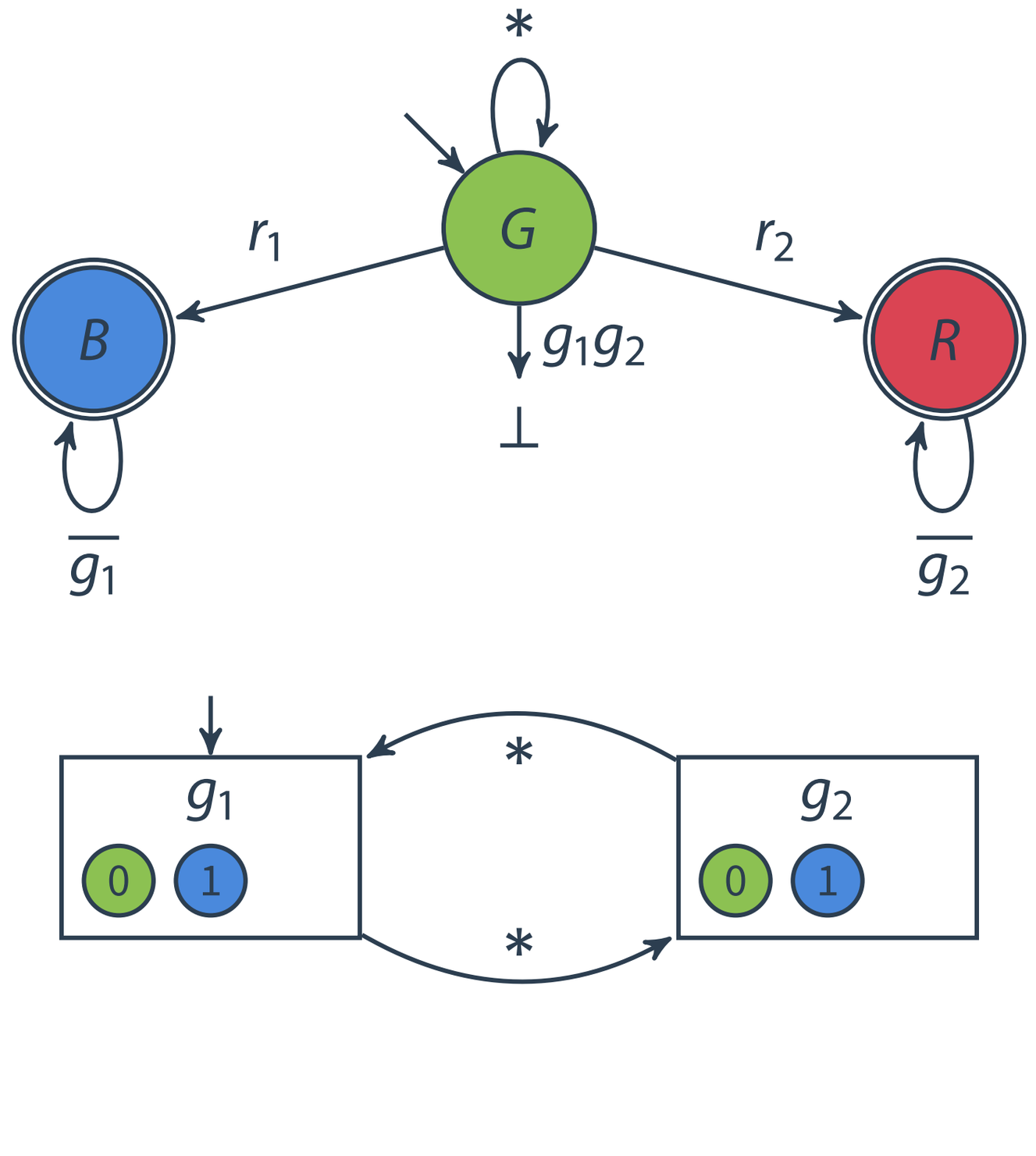


Acceptance of an FSM

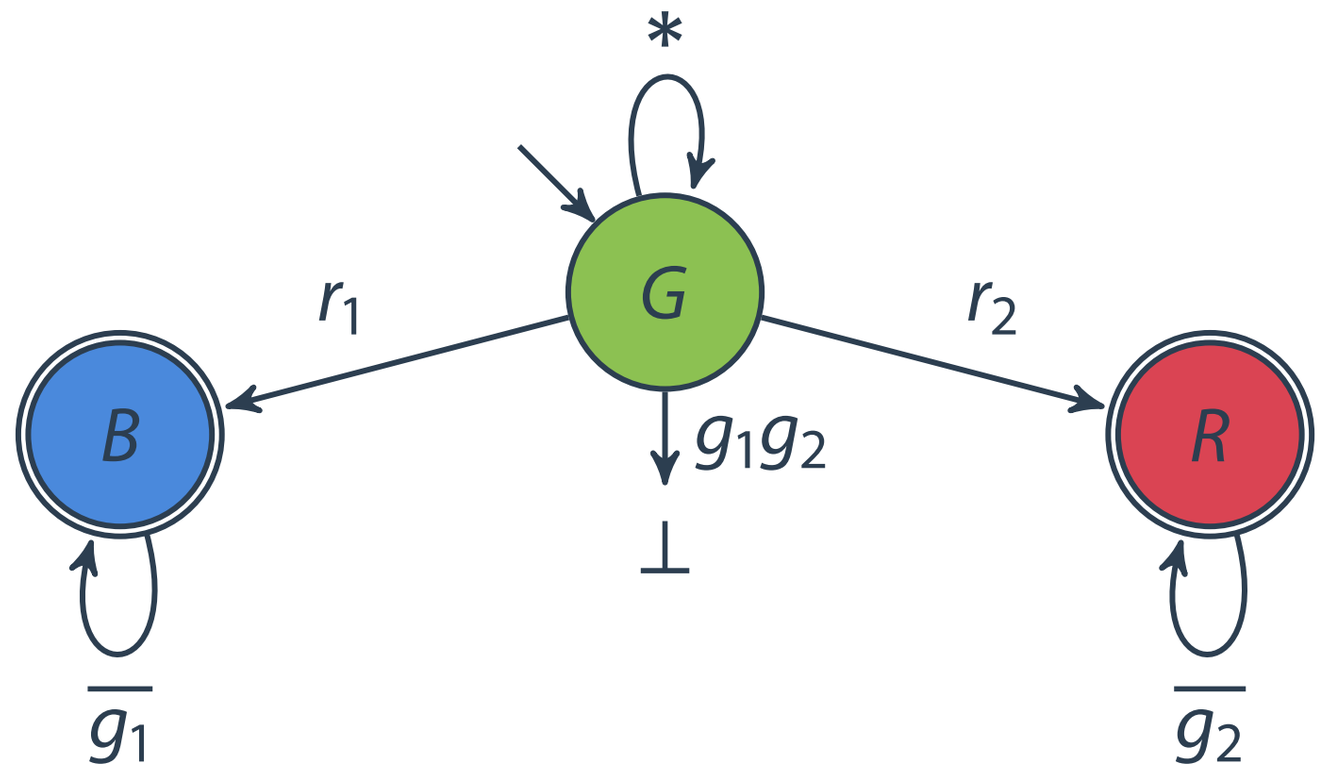


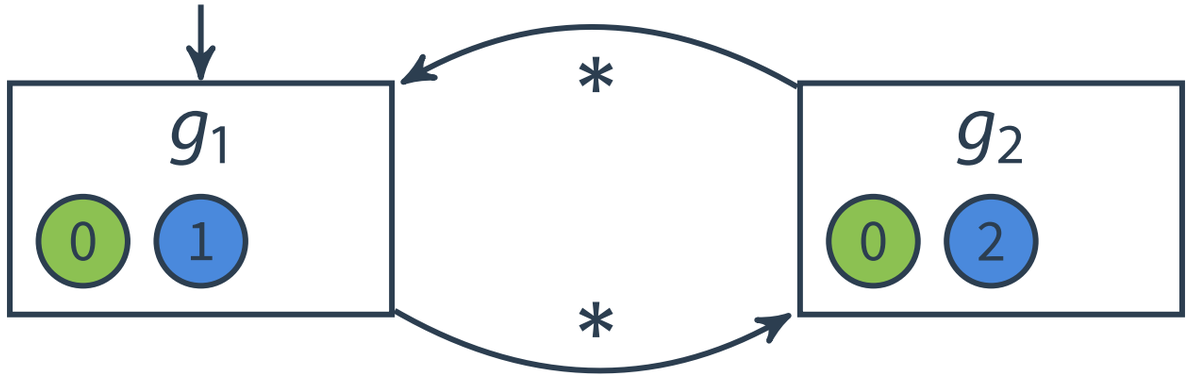


Acceptance of an FSM

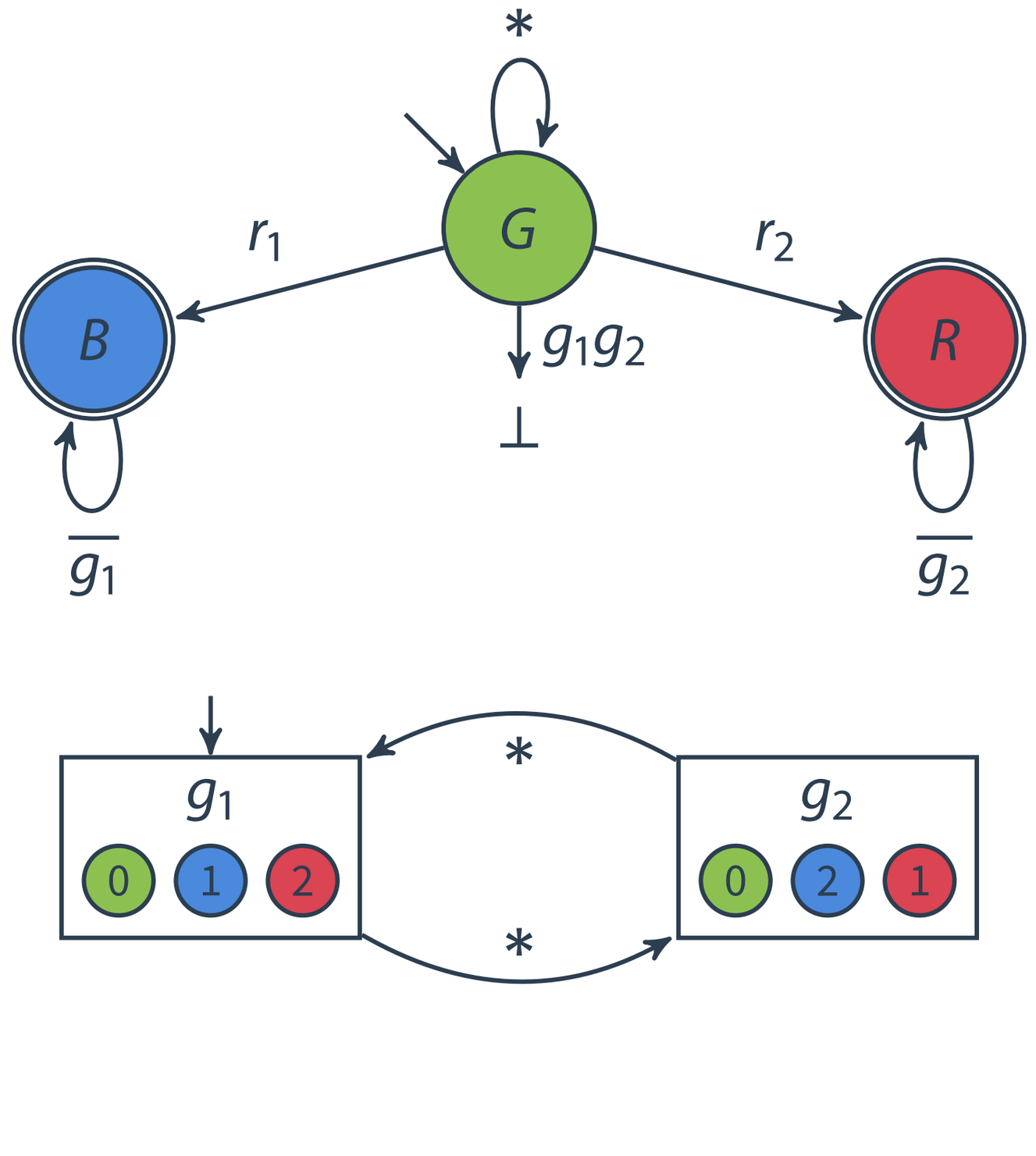


Acceptance of an FSM





Acceptance of an FSM



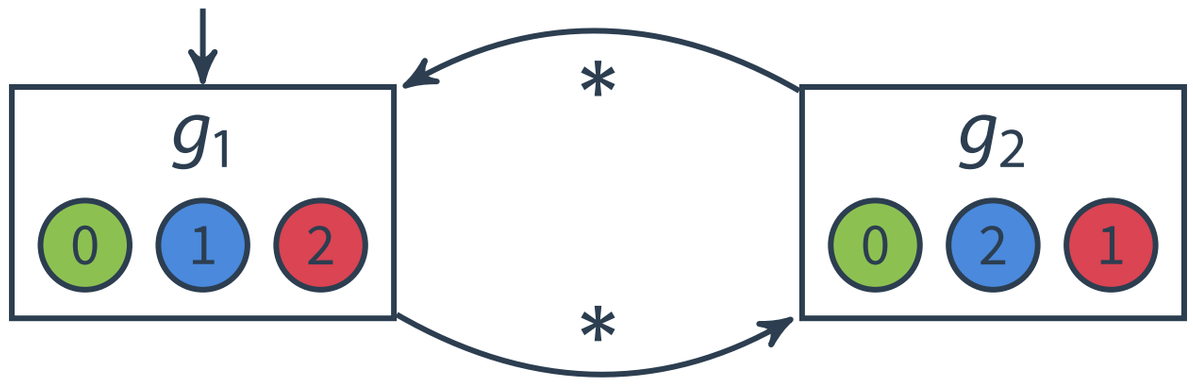
Part III: Bounded Synthesis

1. Universal co-Büchi automata
2. **Constraint systems**
3. Towards structurally simple implementations

Annotated FSM

Annotation

- collects the paths of the run tree that lead to a state in the FSM
- for each automaton state, indicates whether state visited on some path, and if so, max number of visits to rejecting states



Annotated FSM

Annotation

- collects the paths of the run tree that lead to a state in the FSM
- for each automaton state, indicates whether state visited on some path, and if so, max number of visits to rejecting states

Theorem – Completeness

An FSM is accepted by a universal co-Büchi automaton
$\Leftrightarrow$ it has a valid annotation.

Constraint System

The constraint system specifies the existence of an annotated FSM.

Representation of the FSM

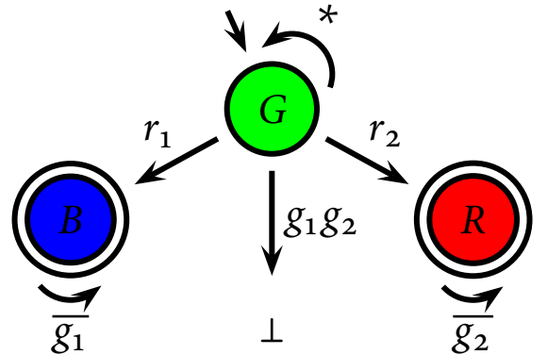
- **states**: $\mathbb{N}_N$
- **labeling**: functions $\nu : \mathbb{N}_N \to \mathbb{B}$
- **transitions**: functions $\tau_{in} : \mathbb{N}_N \to \mathbb{N}_N$

Representation of annotation

- **state occurrence**: functions $\lambda_q^{\mathbb{B}} : \mathbb{N}_N \to \mathbb{B}$
- **rejecting bound**: functions $\lambda_q^{\#} : \mathbb{N}_N \to \mathbb{N}$

Constraints

- $\lambda^{\mathbb{B}}_{G}(o)$
- $\forall t.\, \lambda^{\mathbb{B}}_{G}(t) \rightarrow \lambda^{\mathbb{B}}_{G}(\tau_{\overline{r}_1\overline{r}_2}(t)) \wedge \lambda^{\#}_{G}(\tau_{\overline{r}_1\overline{r}_2}(t)) \geq \lambda^{\#}_{G}(t)$
  $\wedge\ \lambda^{\mathbb{B}}_{G}(\tau_{\overline{r}_1 r_2}(t)) \wedge \lambda^{\#}_{G}(\tau_{\overline{r}_1 r_2}(t)) \geq \lambda^{\#}_{G}(t)$
  $\wedge\ \lambda^{\mathbb{B}}_{G}(\tau_{r_1\overline{r}_2}(t)) \wedge \lambda^{\#}_{G}(\tau_{r_1\overline{r}_2}(t)) \geq \lambda^{\#}_{G}(t)$
  $\wedge\ \lambda^{\mathbb{B}}_{G}(\tau_{r_1 r_2}(t)) \wedge \lambda^{\#}_{G}(\tau_{r_1 r_2}(t)) \geq \lambda^{\#}_{G}(t)$
- $\forall t.\, \lambda^{\mathbb{B}}_{G}(t) \rightarrow \neg g_1(t) \vee \neg g_2(t)$



- $\forall t.\, \lambda^{\mathbb{B}}_{G}(t) \wedge r_1(t) \rightarrow \lambda^{\mathbb{B}}_{B}(\tau_{\overline{r}_1\overline{r}_2}(t)) \wedge \lambda^{\#}_{B}(\tau_{\overline{r}_1\overline{r}_2}(t)) > \lambda^{\#}_{G}(t)$
  $\wedge\ \lambda^{\mathbb{B}}_{B}(\tau_{\overline{r}_1 r_2}(t)) \wedge \lambda^{\#}_{B}(\tau_{\overline{r}_1 r_2}(t)) > \lambda^{\#}_{G}(t)$
  $\wedge\ \lambda^{\mathbb{B}}_{B}(\tau_{r_1\overline{r}_2}(t)) \wedge \lambda^{\#}_{B}(\tau_{r_1\overline{r}_2}(t)) > \lambda^{\#}_{G}(t)$
  $\wedge\ \lambda^{\mathbb{B}}_{B}(\tau_{r_1 r_2}(t)) \wedge \lambda^{\#}_{B}(\tau_{r_1 r_2}(t)) > \lambda^{\#}_{G}(t)$
- $\forall t.\, \lambda^{\mathbb{B}}_{B}(t) \wedge \neg g_1(t) \rightarrow \lambda^{\mathbb{B}}_{B}(\tau_{\overline{r}_1\overline{r}_2}(t)) \wedge \lambda^{\#}_{B}(\tau_{\overline{r}_1\overline{r}_2}(t)) > \lambda^{\#}_{B}(t)$
  $\wedge\ \lambda^{\mathbb{B}}_{B}(\tau_{\overline{r}_1 r_2}(t)) \wedge \lambda^{\#}_{B}(\tau_{\overline{r}_1 r_2}(t)) > \lambda^{\#}_{B}(t)$
  $\wedge\ \lambda^{\mathbb{B}}_{B}(\tau_{r_1\overline{r}_2}(t)) \wedge \lambda^{\#}_{B}(\tau_{r_1\overline{r}_2}(t)) > \lambda^{\#}_{B}(t)$
  $\wedge\ \lambda^{\mathbb{B}}_{B}(\tau_{r_1 r_2}(t)) \wedge \lambda^{\#}_{B}(\tau_{r_1 r_2}(t)) > \lambda^{\#}_{B}(t)$

Experiments

Instance	SMT Z3		SAT MiniSat		QBF RAReQS+B		DQBF iDQ	
	mealy	moore	mealy	moore	mealy	moore	mealy	moore
arbiter-2	0.22	0.21	0.21	0.21	0.19	0.19	0.23	0.23
arbiter-3	0.45	0.39	0.30	0.31	0.21	0.21	0.62	0.63
arbiter-4	1428	2234	0.73	0.76	0.25	0.25	1.71	1.83
arbiter-5	TO	TO	4.15	3.72	0.40	0.40	24.6	24.6
arbiter-6	TO	TO	21.0	21.0	0.71	0.71	76.8	79.9
arbiter-7	TO	TO	155.6	102.5	7.02	5.98	294.6	294.3
arbiter-8	TO	TO	2384	TO	397.6	406.6	TO	TO

[Faymonville/F./Rabe/Tentrup, 2015]

Propositional encodings

- **SAT: complete unrolling**

$$\exists \lambda^{\mathbb{B}}_{t,q}, \lambda^{\#}_{t,q}, \nu_s, \tau_{t,i,t'}$$

- **QBF: input symbolic encoding**

$$\exists \lambda^{\mathbb{B}}_{t,q}, \lambda^{\#}_{t,q}, \nu_s, \forall i.\ \exists \tau_{t,t'}$$

- **DQBF: state and input symbolic encoding**

$$\forall t.\ \exists \lambda^{\mathbb{B}}_{q}, \lambda^{\#}_{q}, \nu_s, \tau_{t,i,t'}\ \forall i.\ \exists \tau$$
$$\forall t'.\ \exists \lambda'^{\mathbb{B}}_{q}, \lambda'^{\#}_{q}$$

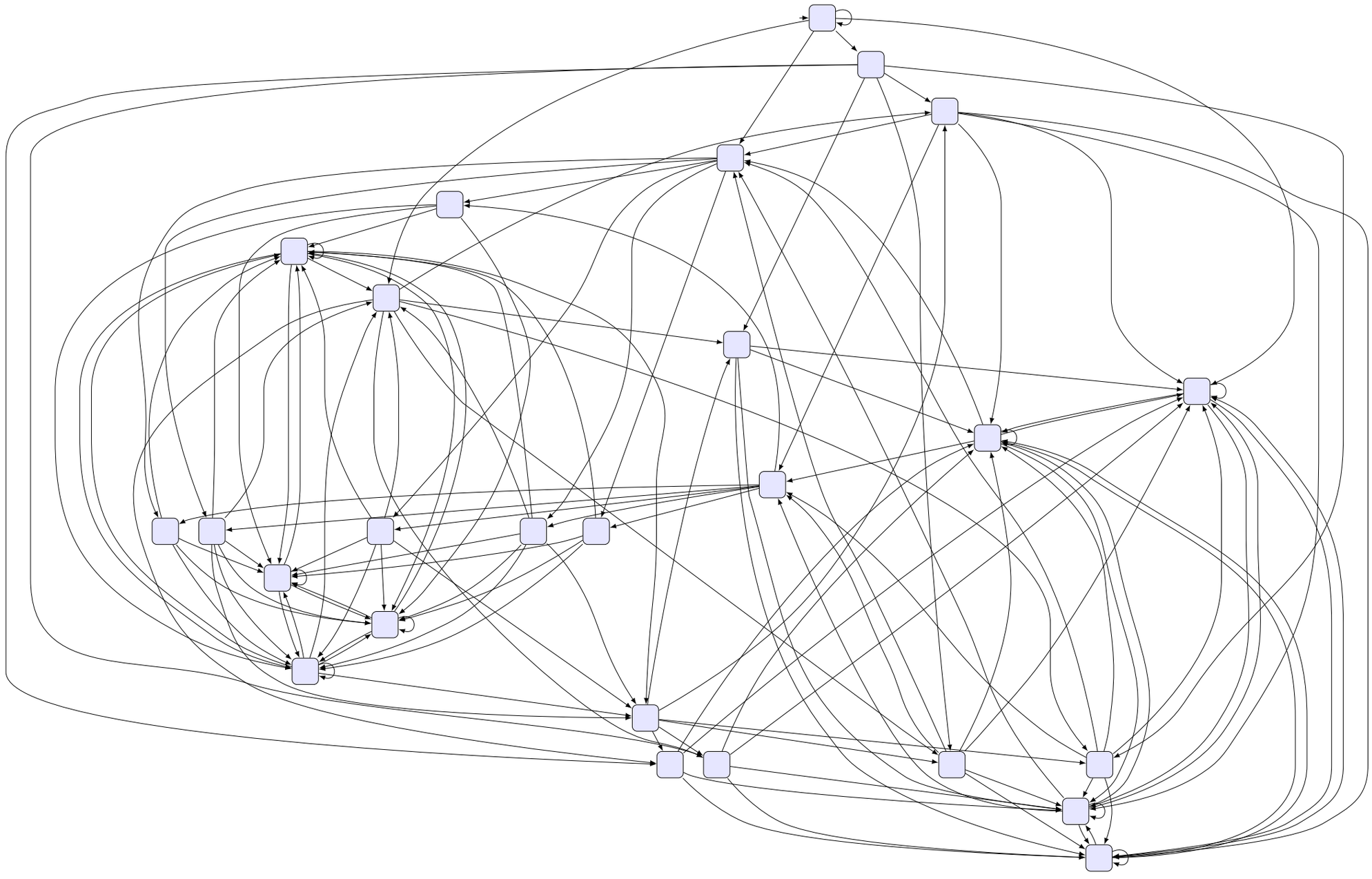
$$\bigwedge_q (t = t') \Rightarrow (\lambda^{\mathbb{B}}_q = \lambda'^{\mathbb{B}}_q) \wedge (\lambda^{\#}_q = \lambda'^{\#}_q)$$

[Faymonville/F./Rabe/Tentrup, 2015]

Part III: Bounded Synthesis

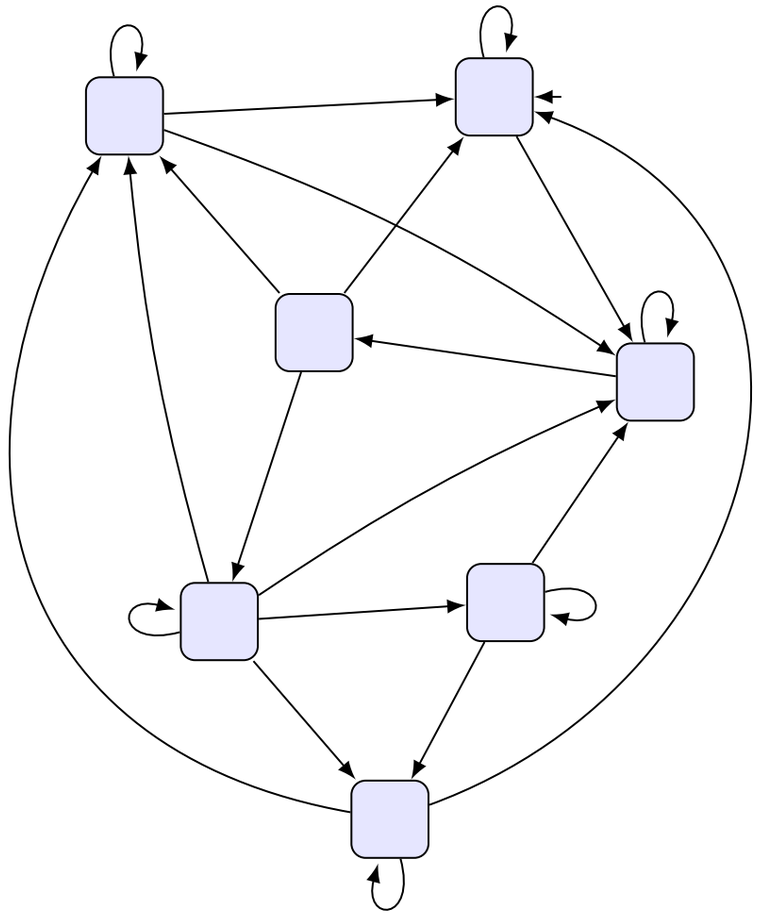
1. Universal co-Büchi automata
2. Constraint systems
3. **Towards structurally simple implementations**

Standard synthesis (Acacia+ v2.3)



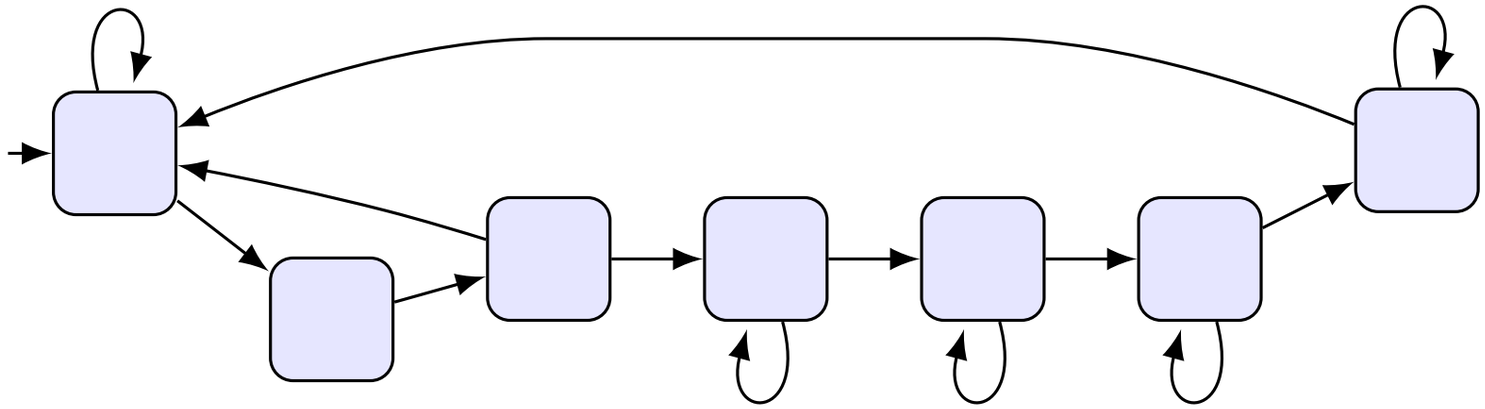
TBURST4 component (AMBA)

Bounded synthesis



TBURST4 component (AMBA)

Bounded cycle synthesis



TBURST4 component (AMBA)

Towards structurally simple implementations

- **Bounded synthesis** minimizes the number of states.
- FSMs with a minimal number of states may still have a complicated control structure.
- Additional parameters (besides number of states) are needed.
- **Bounded cycle synthesis** additionally minimizes the number of simple cycles.
- The number of cycles is an explosive parameter:
  - The number of cycles of an FSM is exponentially bounded in the size of the FSM.
  - There is a realizable LTL formula φ such that every implementation has at least triply-exponentially many cycles in the size of φ.

Experiments

Benchmark	Size			Cycles			Time (s)				
								SAT		UNSAT	
	$\mathcal{A}_\varphi$	Aca+	BoSy/ BoCy	Aca+	BoSy	BoCy	Aca+	BoSy	BoCy	BoSy	BoCy
ARBITER[2]	6	26	2	5439901	3	3	0.261	0.847	0.868	0.300	0.836
ARBITER[3]	20	111	3	$>$ 9999999	8	4	0.511	9.170	9.601	3.916	9.481
ARBITER[4]	64	470	4	$>$ 9999999	8	5	12.981	105.527	109.180	56.853	106.803
LOCK[2]	12	4	3	12	6	5	0.459	0.395	0.522	0.165	0.487
LOCK[3]	20	4	3	12	5	5	55.917	1.037	1.245	0.433	1.107
LOCK[4]	36	–	3	–	6	5	$>$ 999	4.419	4.761	1.407	3.726
ENCODE[2]	3	6	2	41	3	3	0.473	0.071	0.089	0.048	0.084
ENCODE[3]	5	16	3	90428	8	8	1.871	0.292	0.561	0.200	0.503
ENCODE[4]	5	20	4	$>$ 9999999	24	24	4.780	1.007	16.166	0.579	$>$ 999
DECODE	1	4	1	8	1	1	0.328	0.055	0.051	–	–
SHIFT	3	6	2	31	3	3	0.387	0.060	0.072	0.041	0.071
TBURST4	103	14	7	61	19	7	0.634	8.294	206.604	6.261	$>$ 999
TINCR	43	5	3	7	5	2	0.396	2.262	2.279	0.845	2.221
TSINGLE	22	8	4	12	5	4	0.372	1.863	2.143	1.165	2.067

[F./Klein 2016]

Overview

Part I. Infinite Games

Fundamental algorithms to solve infinite games played over finite graphs.

Part II. Synthesis from Logical Specifications

Synthesis from specifications given as formulas of a temporal logic. The quest for an efficient and expressive specification language.

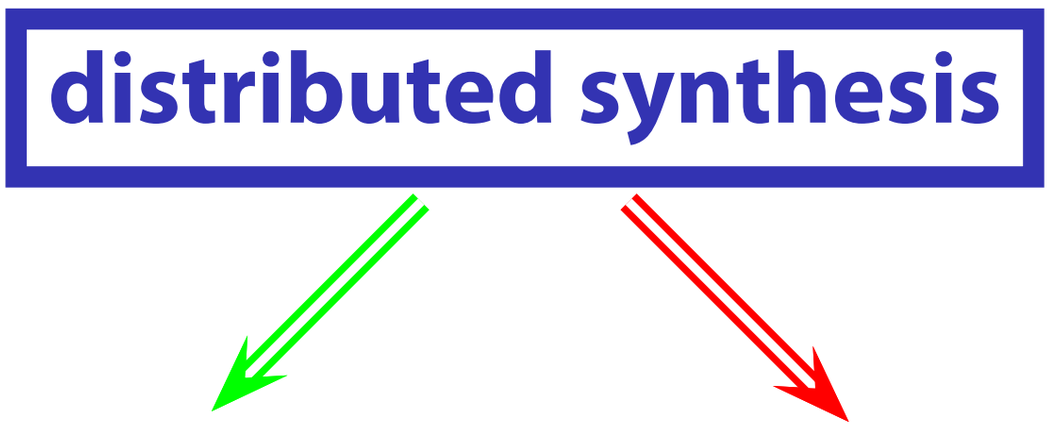
Part III. Bounded Synthesis

Finding simple solutions fast. The quest for structurally simple implementations.

Part IV. Distributed Synthesis

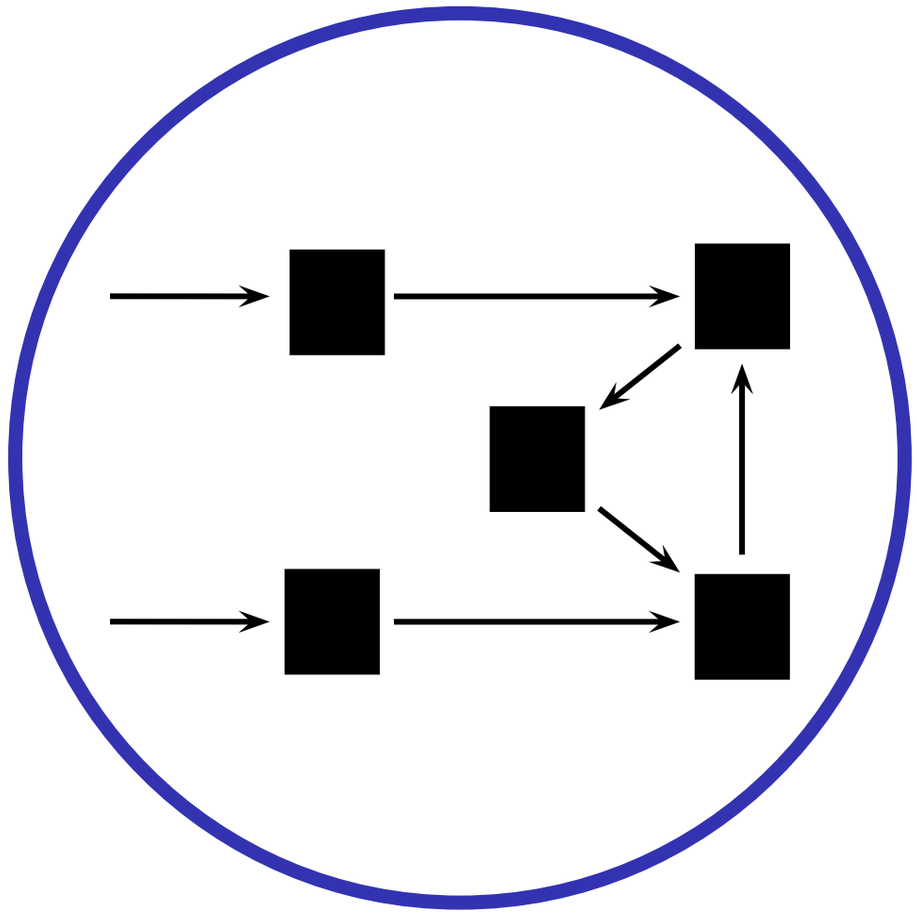
Synthesizing systems that consist of multiple distributed components.

specification



implementation

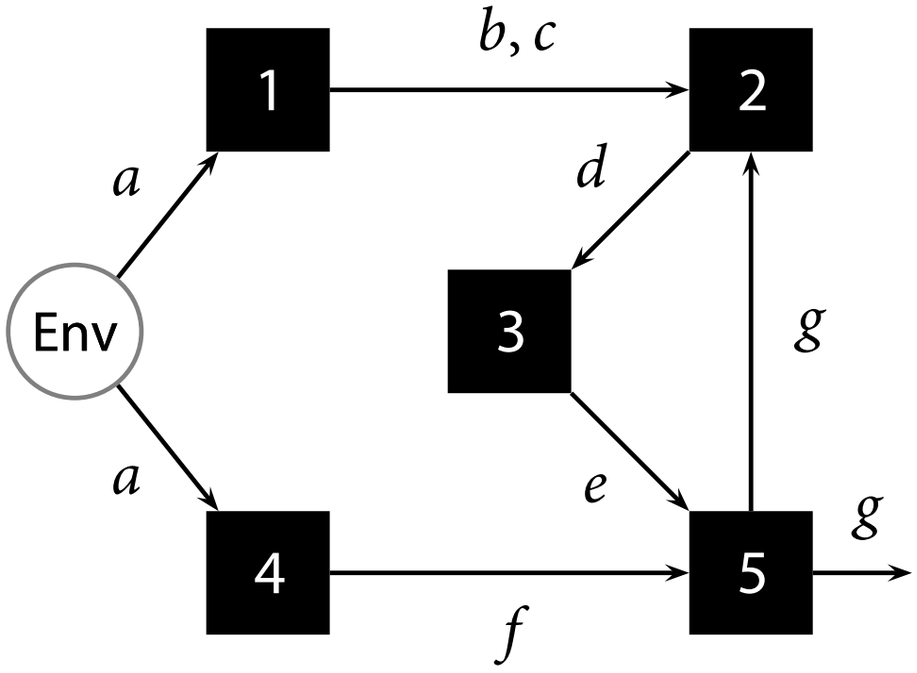
unrealizable



Part IV: Distributed Synthesis

1. **Synthesis in the Pnueli/Rosner model**
2. Bounded synthesis of distributed systems
3. Synthesis in the causal memory model

Pnueli/Rosner model



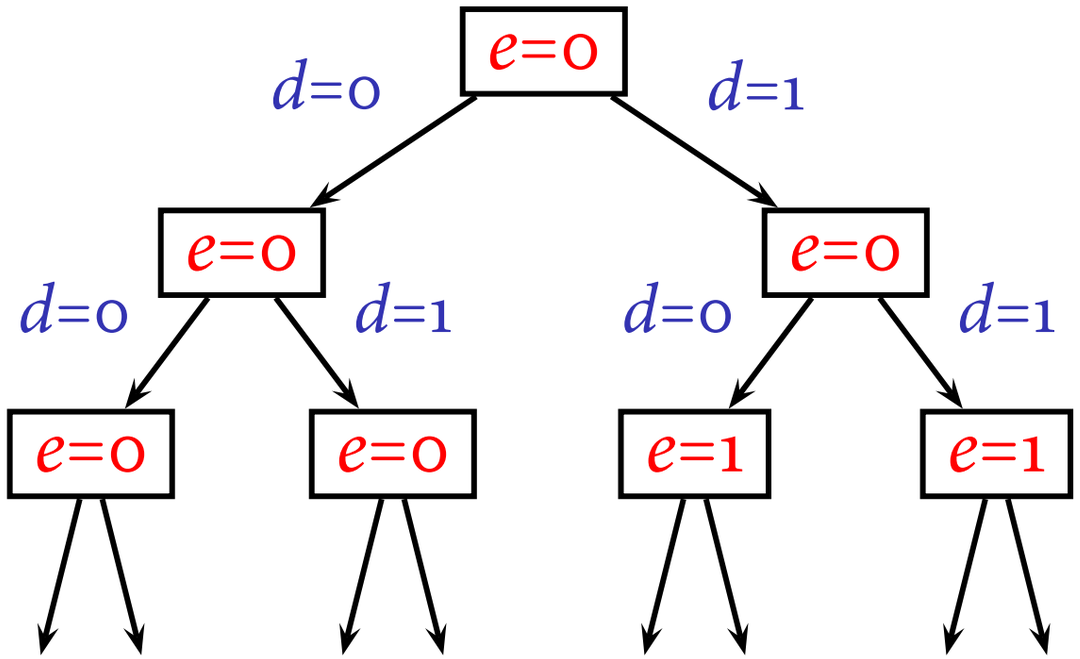
Architectures

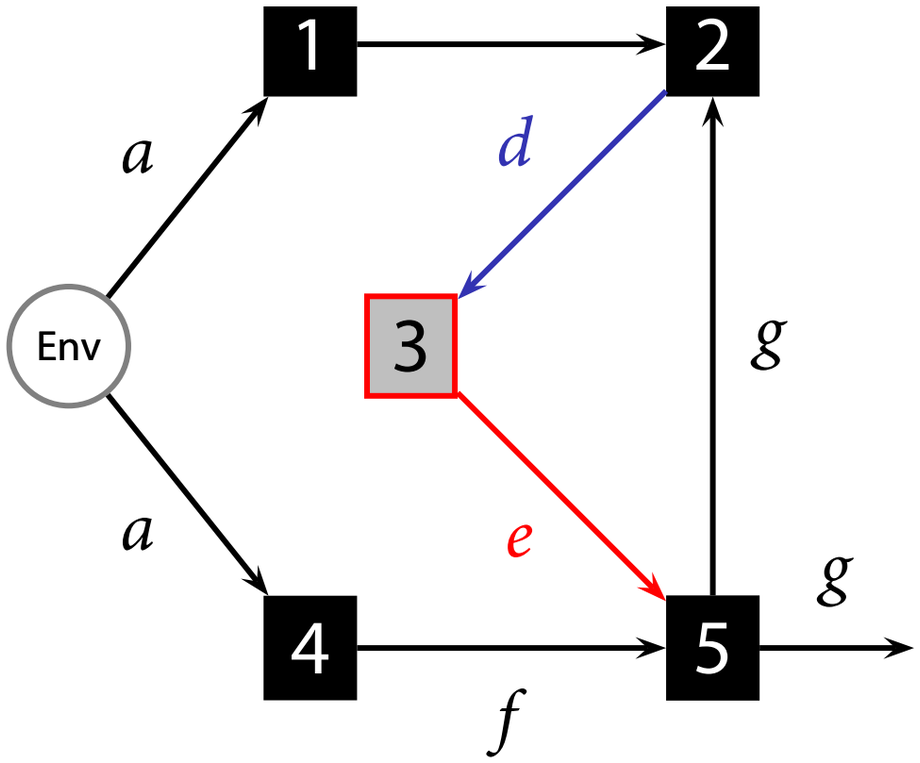
Nodes:
- system processes -- unknown implementation
- environment -- unconstrained behavior

Edges:
- communication structure
- variables

[Pnueli/Rosner,1989]

Pnueli/Rosner model

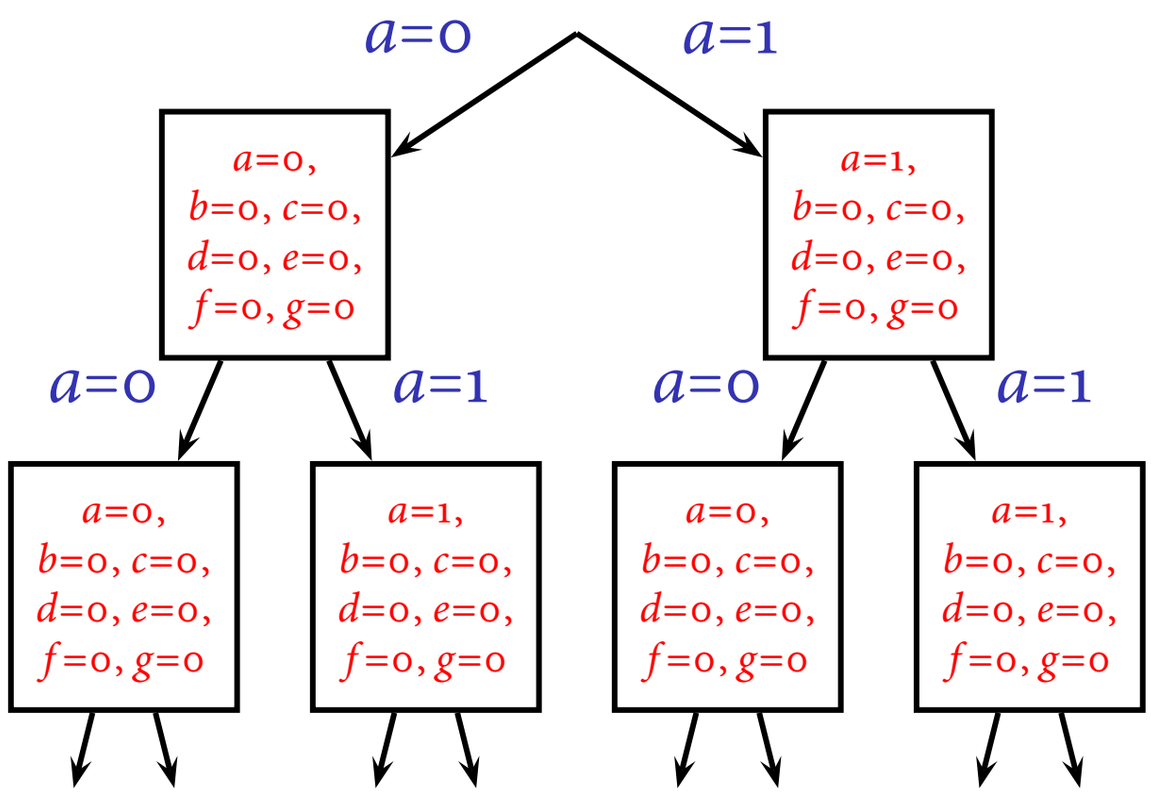


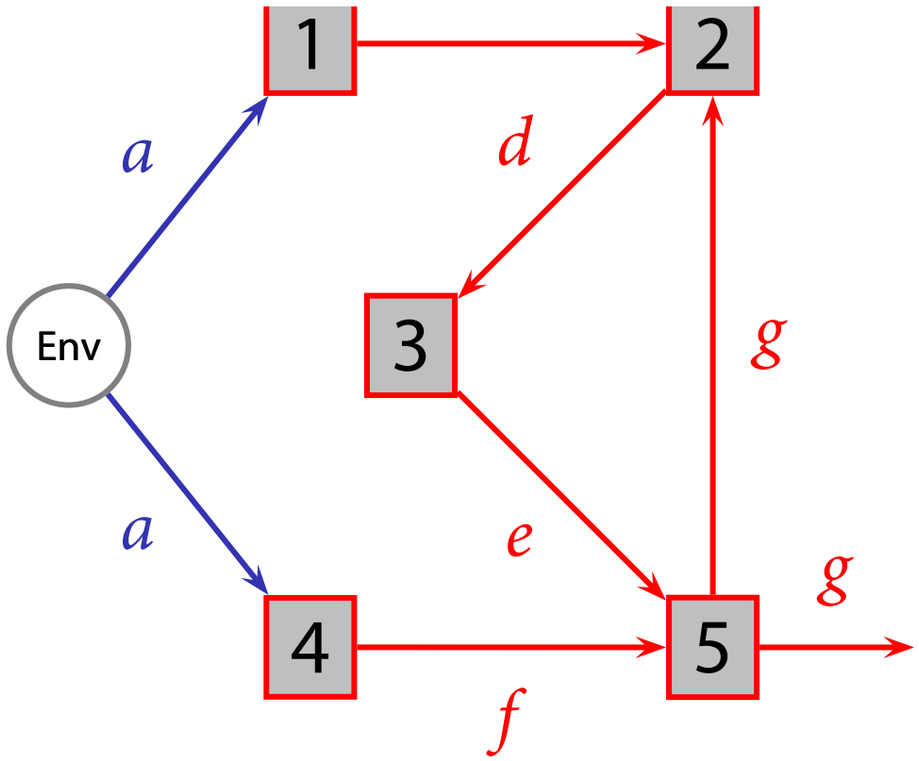


Implementation

The implementation defines for each process p
with input variables I_p and output variables O_p
a strategy tree with directions 2^{I_p} and labels 2^{O_p}.

Pnueli/Rosner model



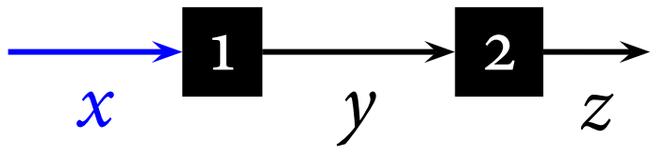


Specification

The combination of the process strategies defines the computation tree with directions 2^I and labels 2^V, where I are the global input variables and V is the set of variables.

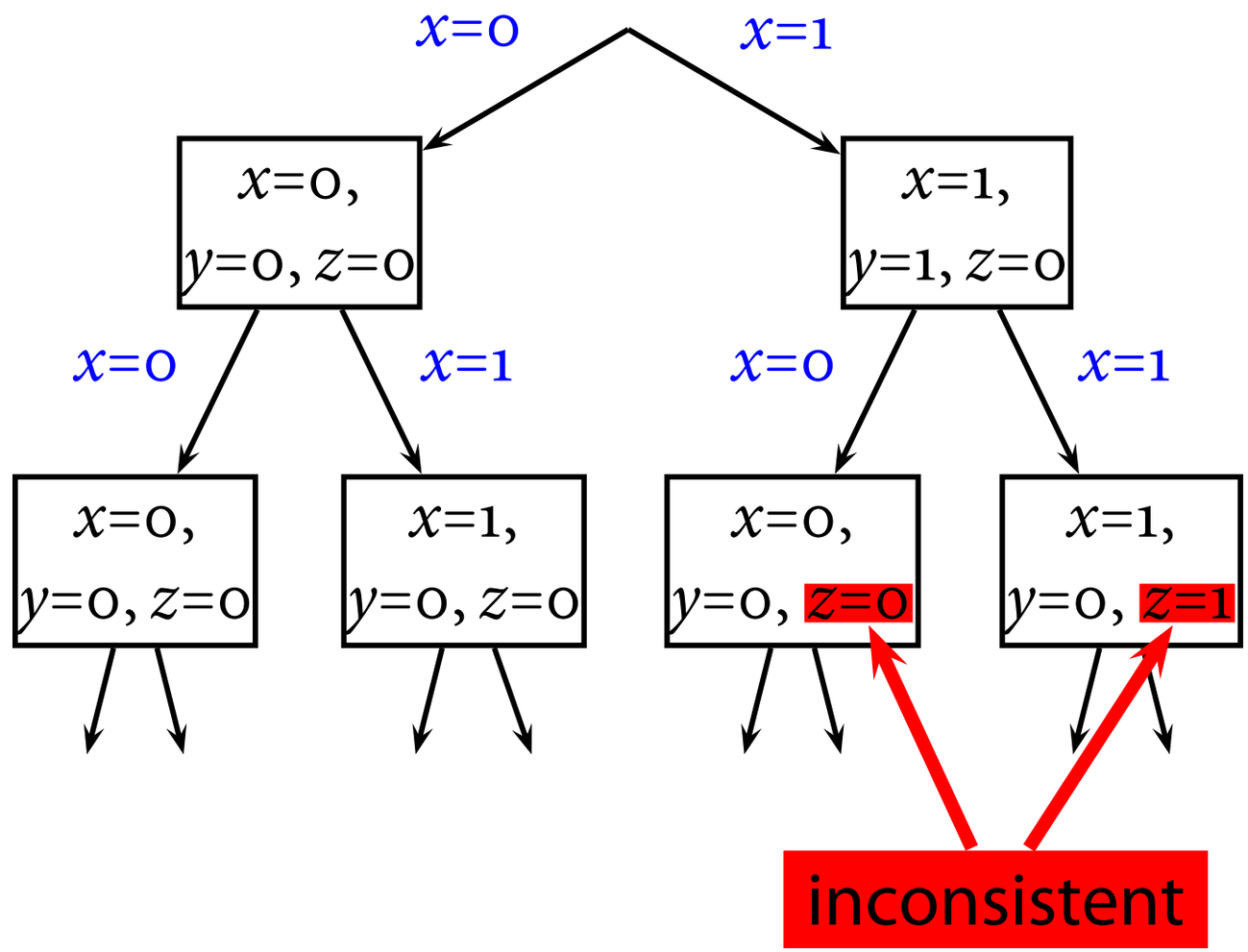
The implementation is correct iff the computation tree satisfies the given specification, e.g., some formula in a temporal logic.

Partial observation

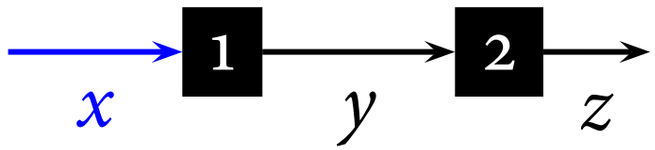


process 2 does not know x

decisions of process 2
must not depend on x.

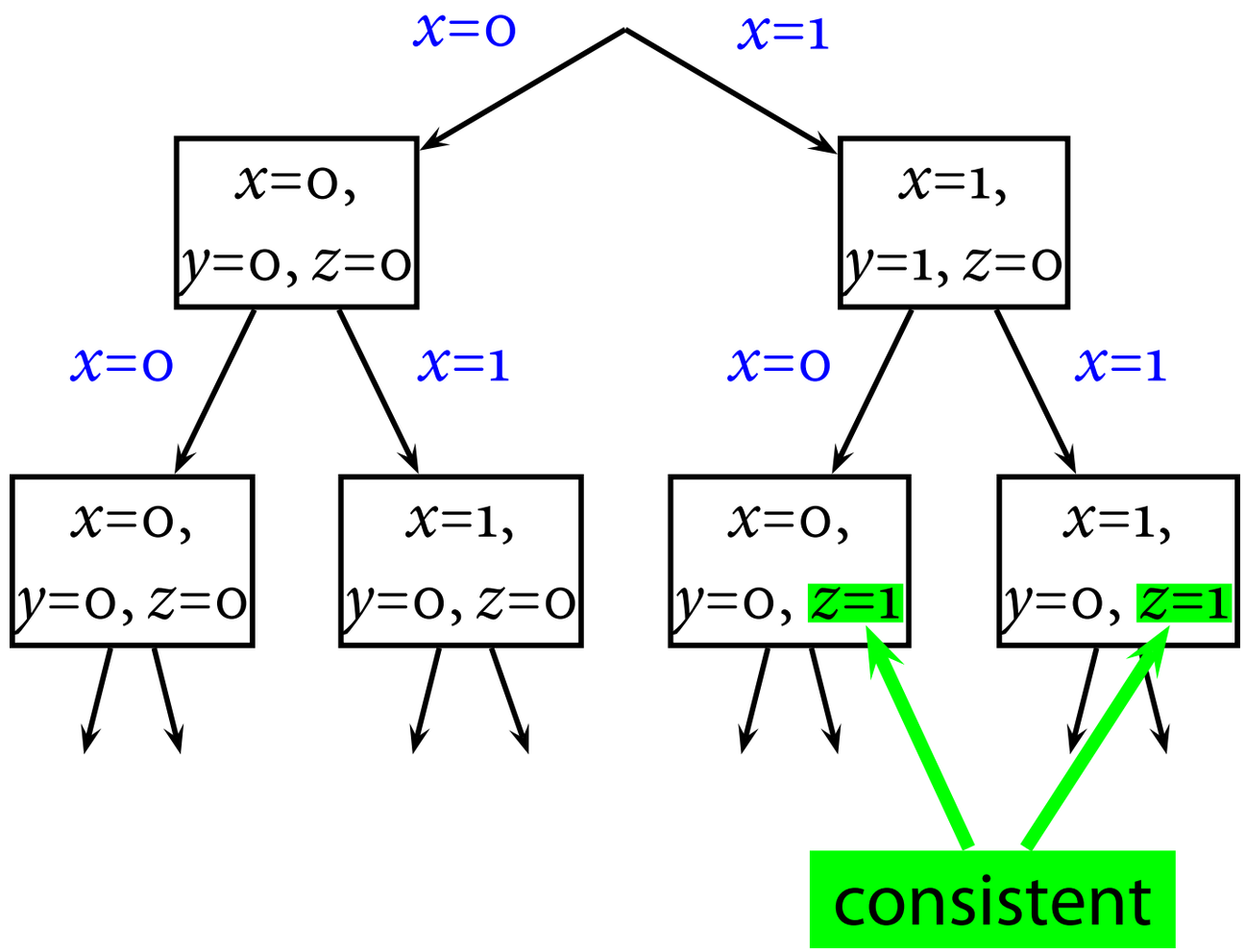


Partial observation



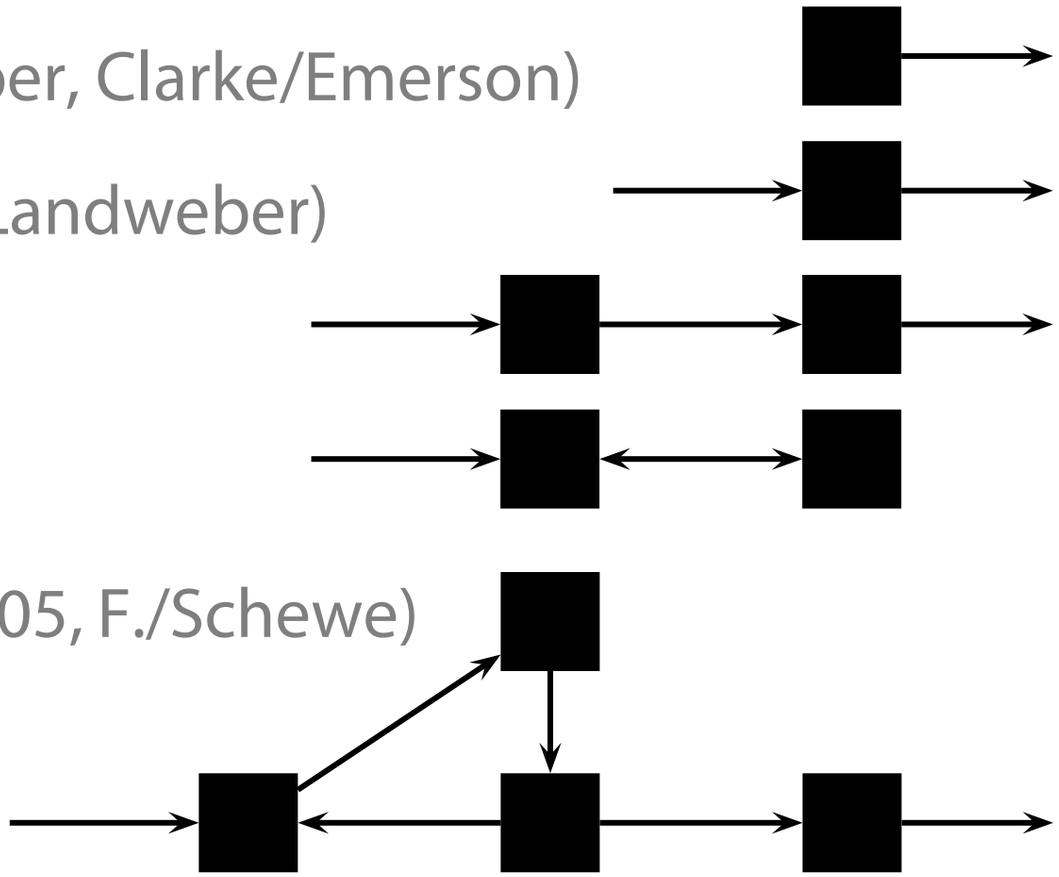
process 2 does not know x

decisions of process 2
must not depend on x.



Pnueli/Rosner model: Decidability

- Closed systems (1981 Manna/Wolper, Clarke/Emerson)
- Open systems (1969 Rabin, Büchi, Landweber)
- Pipelines (1990, Pnueli/Rosner)
- Rings (2001, Kupferman/Vardi)
- Weakly-ordered architectures (2005, F./Schewe)

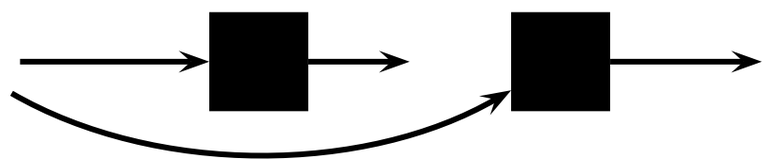


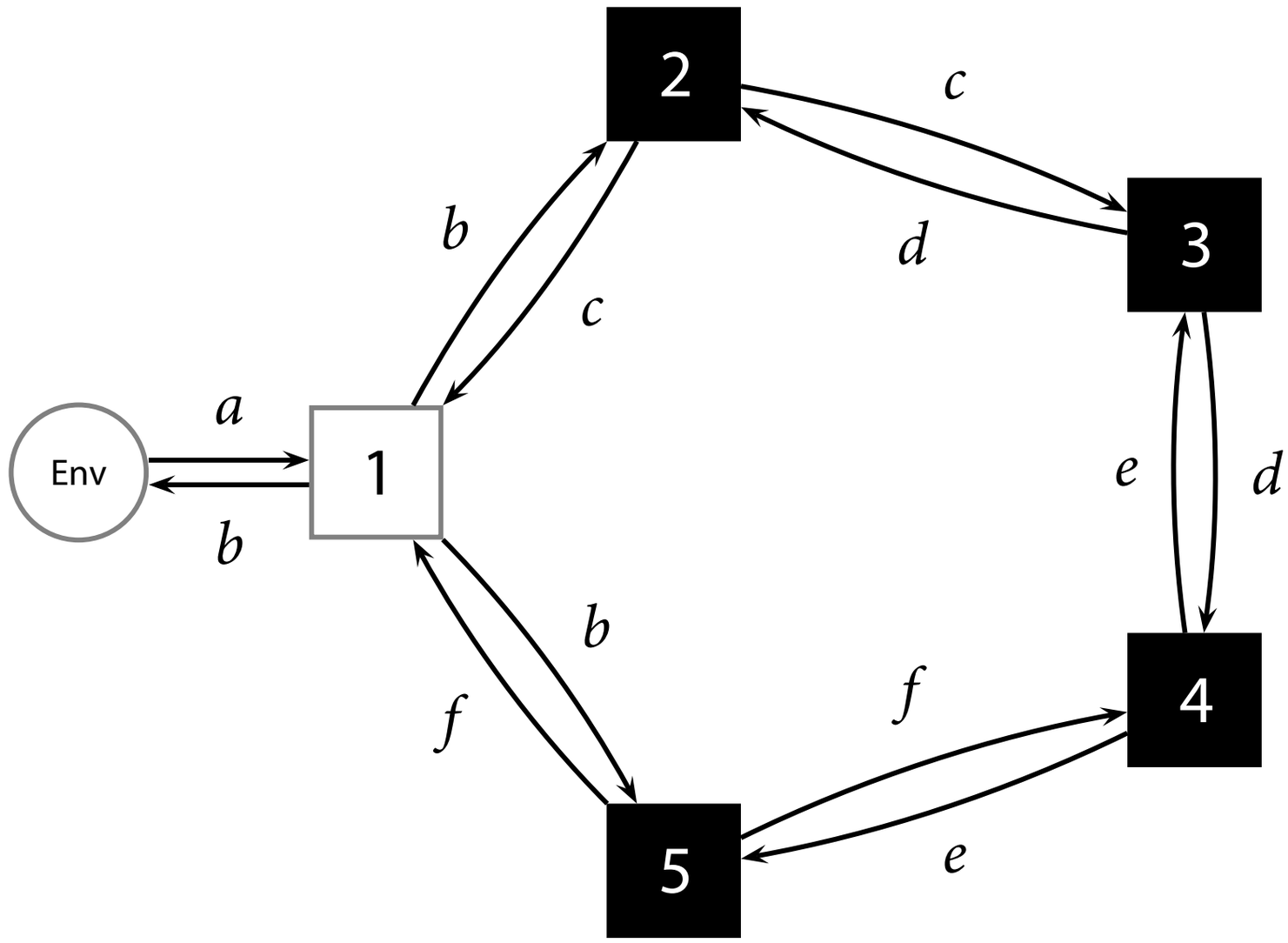
decidable

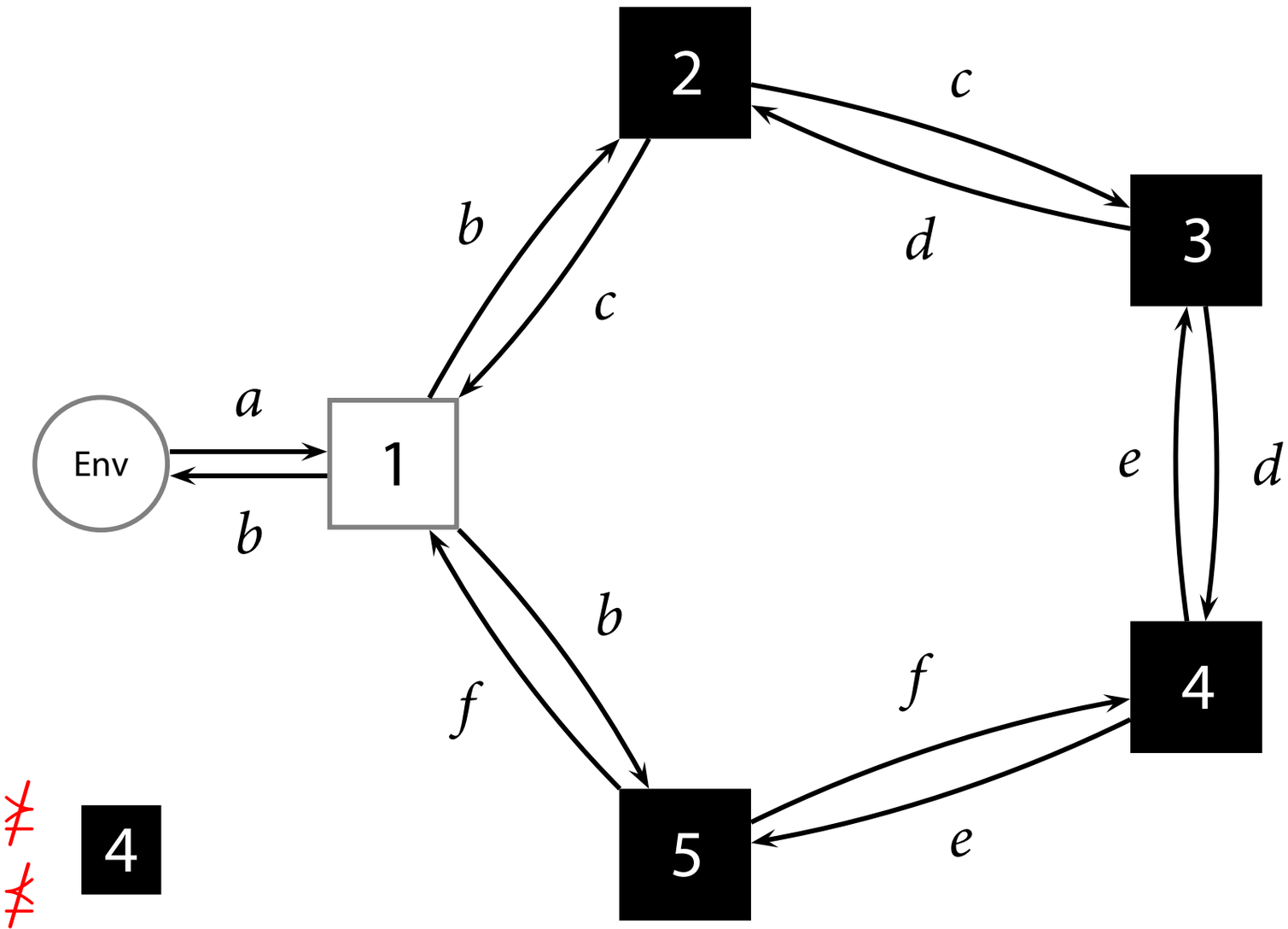
---

undecidable

- Independent processes
  (1990, Pnueli/Rosner)
- Architectures with information forks (2005, F./Schewe)

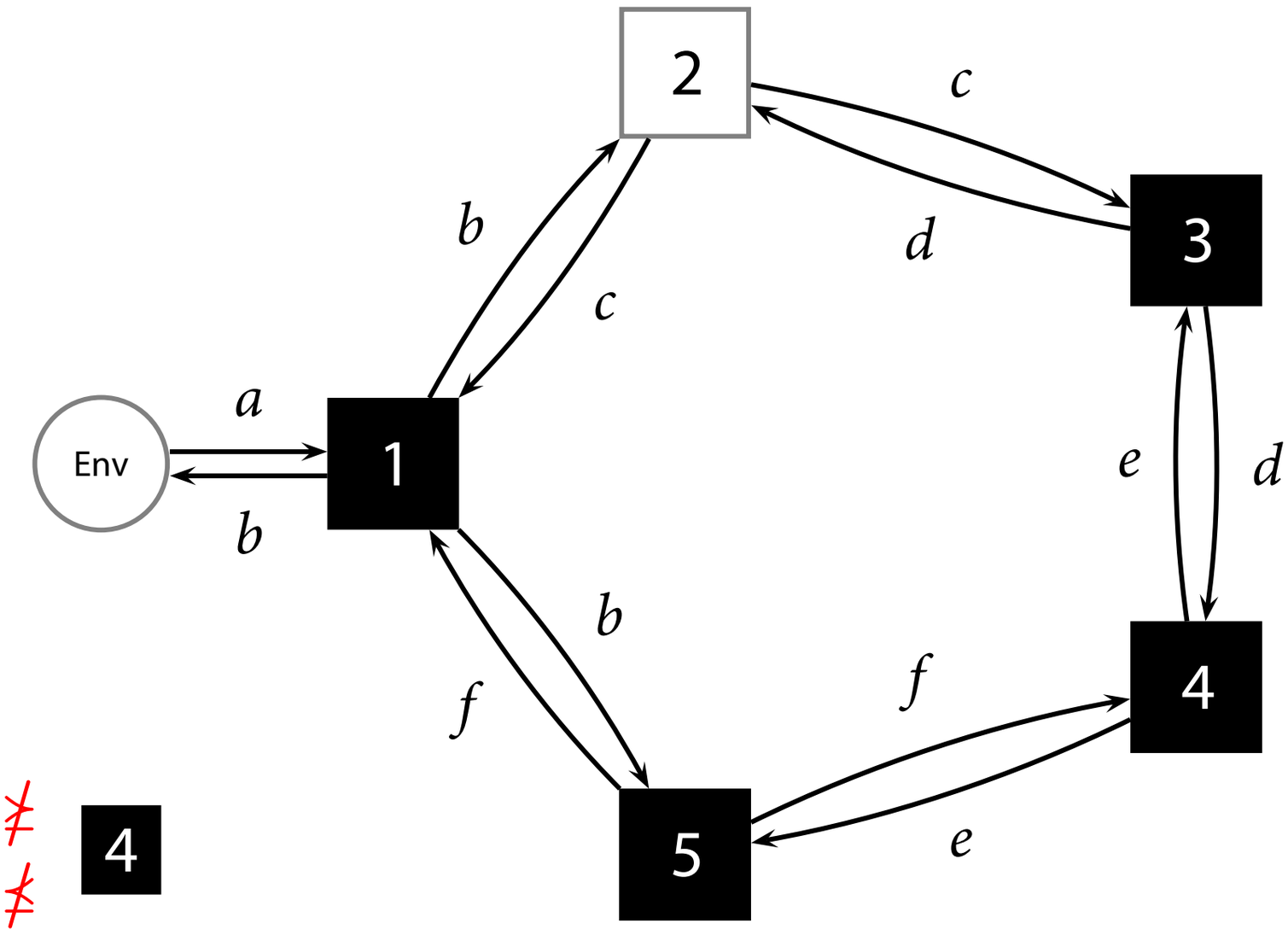



2
c
3
b
d
c
a
Env
1
e
d
b
h
b
f
f
4
5
e



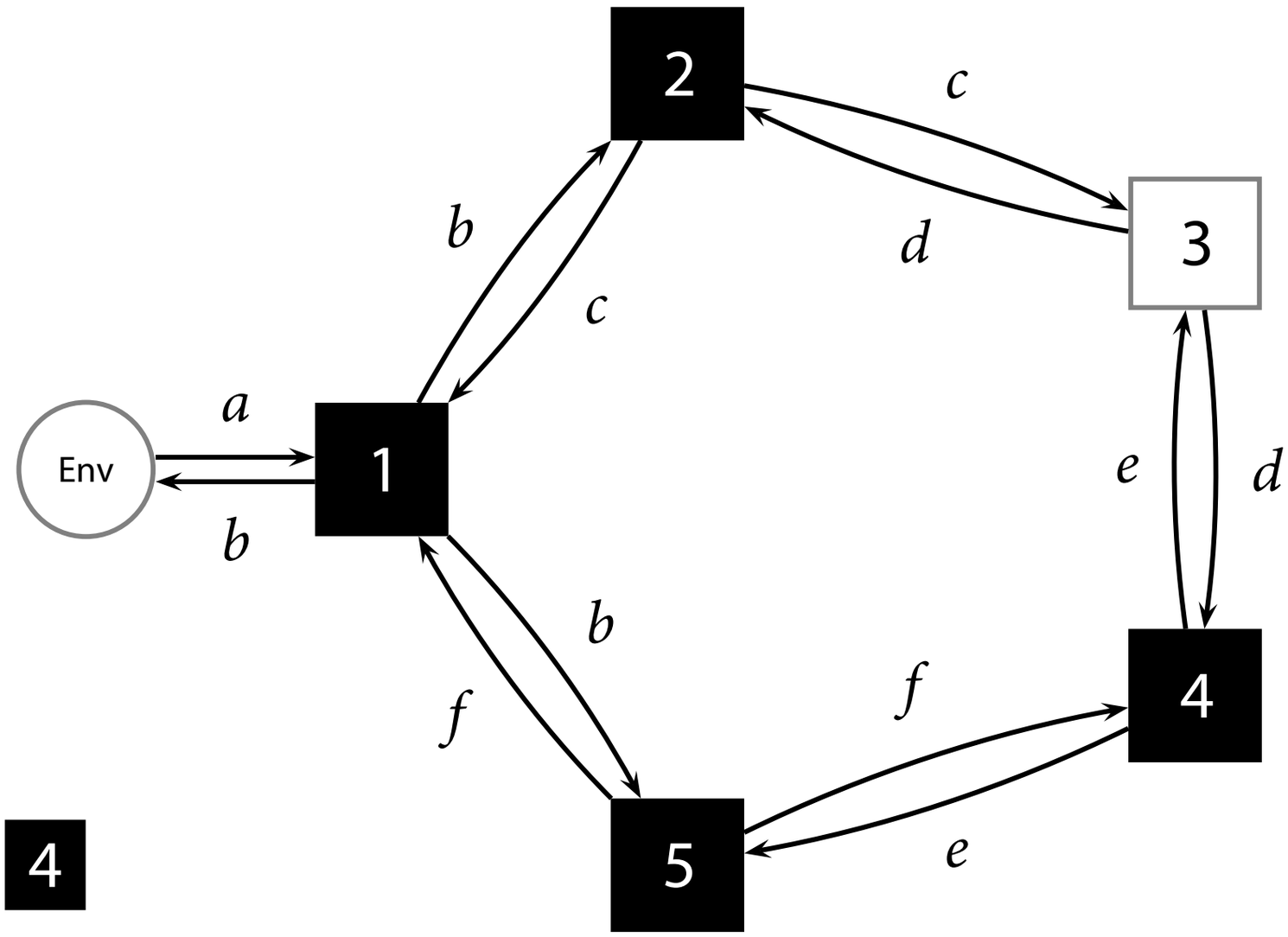
$2 \simeq 5 \succeq 3 \not\simeq 4$

information fork



1 ≥ 5 ≥ 3 ≠ 4

information fork

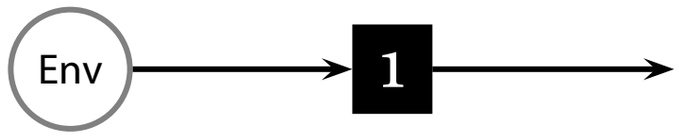


1 ≥ 2 ≃ 5 ≥ 4

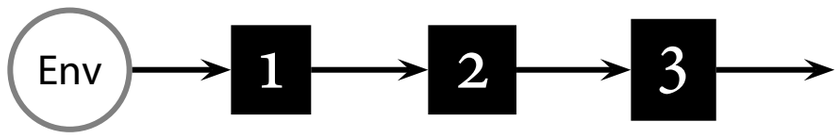
weakly ordered

Synthesis in the Pnueli/Rosner model

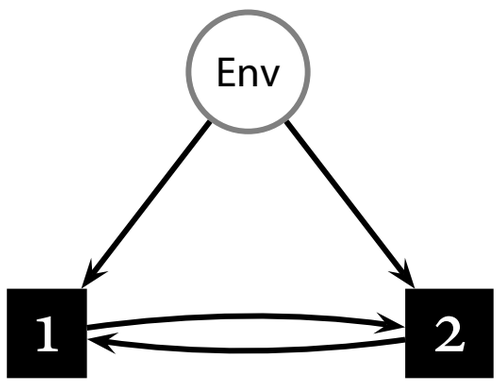
1-process architectures --- 2EXPTIME



Pipeline architectures --- non-elementary



2-process arbiter architecture --- undecidable

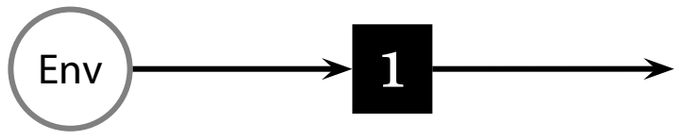


Part IV: Distributed Synthesis

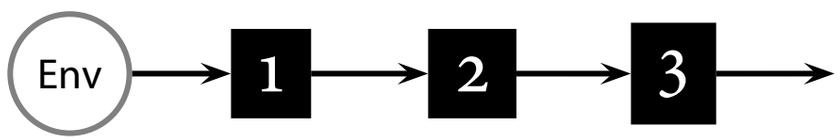
1. Synthesis in the Pnueli/Rosner model
2. **Bounded synthesis of distributed systems**
3. Synthesis in the causal memory model

Bounded synthesis of distributed systems

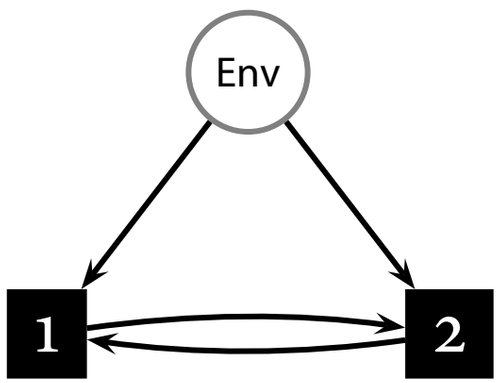
1-process architectures --- NP



Pipeline architectures --- NP



2-process arbiter architecture --- NP



Constraint System

The constraint system specifies the existence of an annotated FSM.

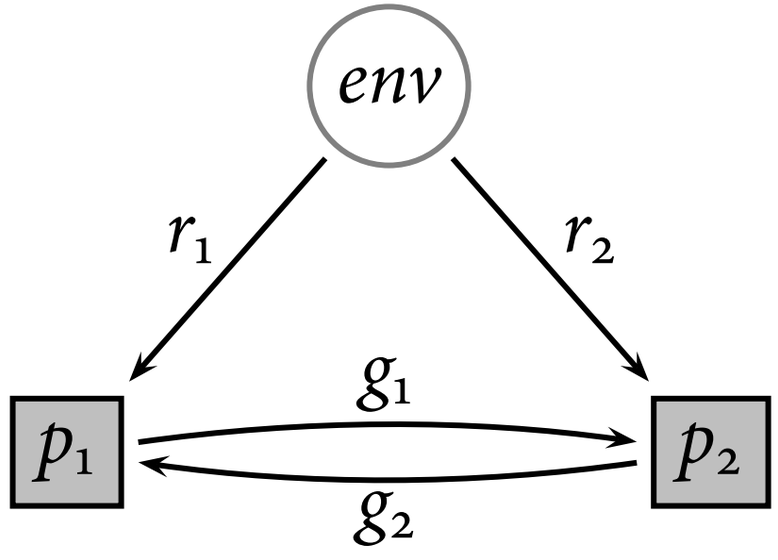
Representation of the FSM

- **states**: $\mathbb{N}_N$
- **labeling**: functions $\nu : \mathbb{N}_N \to \mathbb{B}$
- **transitions**: functions $\tau_{in} : \mathbb{N}_N \to \mathbb{N}_N$

Representation of annotation

- **state occurrence**: functions $\lambda_q^{\mathbb{B}} : \mathbb{N}_N \to \mathbb{B}$
- **rejecting bound**: functions $\lambda_q^{\#} : \mathbb{N}_N \to \mathbb{N}$

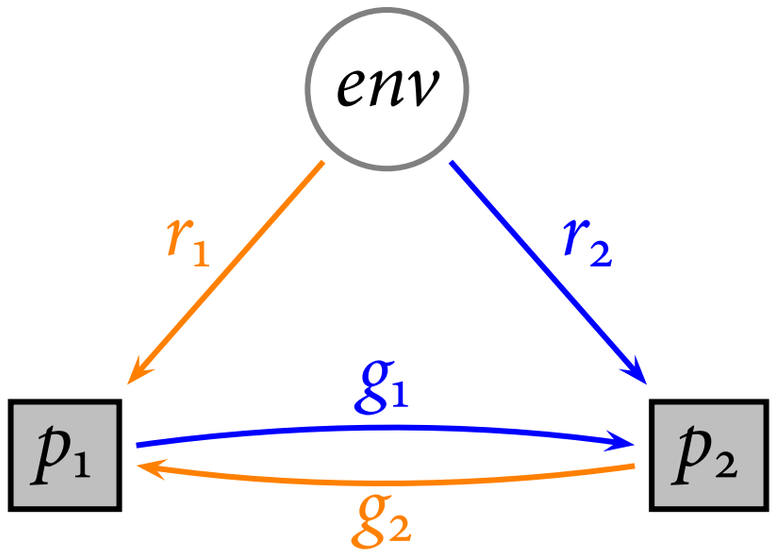
Extended constraint system



Local transition system

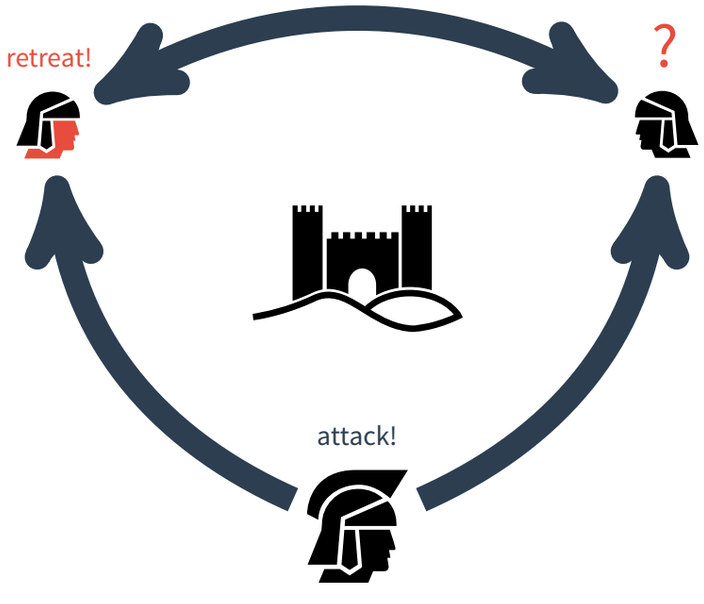
- **projection** from global states to local states for process p_i:
  $proj_i : \mathbb{N}_N \to \mathbb{N}_{N_i}$
- **local transition function** for local input and local state
  $\tau_{i;inp} : \mathbb{N}_{N_i} \to \mathbb{N}_{N_i}$

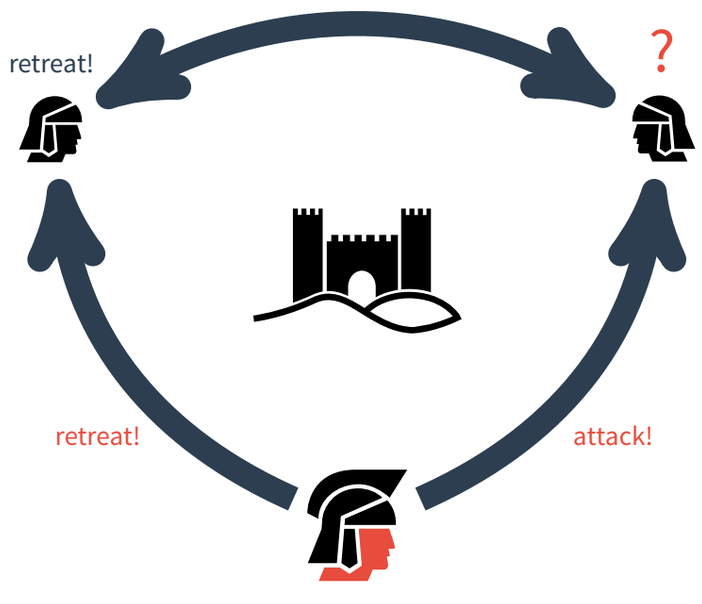
Consistency constraint



- $\forall t.\ \tau_{1;r_1,g_2(proj_2(t))}\big(proj_1(t)\big) = proj_1(\tau_{r_1 r_2}(t)) = proj_1(\tau_{r_1 \overline{r}_2}(t))$
  $\wedge\ \tau_{1;\overline{r_1},g_2(proj_2(t))}\big(proj_1(t)\big) = proj_1(\tau_{\overline{r}_1 r_2}(t)) = proj_1(\tau_{\overline{r}_1 \overline{r}_2}(t))$
- $\forall t.\ \tau_{2;r_2,g_1(proj_1(t))}\big(proj_2(t)\big) = proj_2(\tau_{r_1 r_2}(t)) = proj_2(\tau_{\overline{r}_1 r_2}(t))$
  $\wedge\ \tau_{2;\overline{r_2},g_1(proj_1(t))}\big(proj_2(t)\big) = proj_2(\tau_{r_1 \overline{r}_2}(t)) = proj_2(\tau_{\overline{r}_1 \overline{r}_2}(t))$

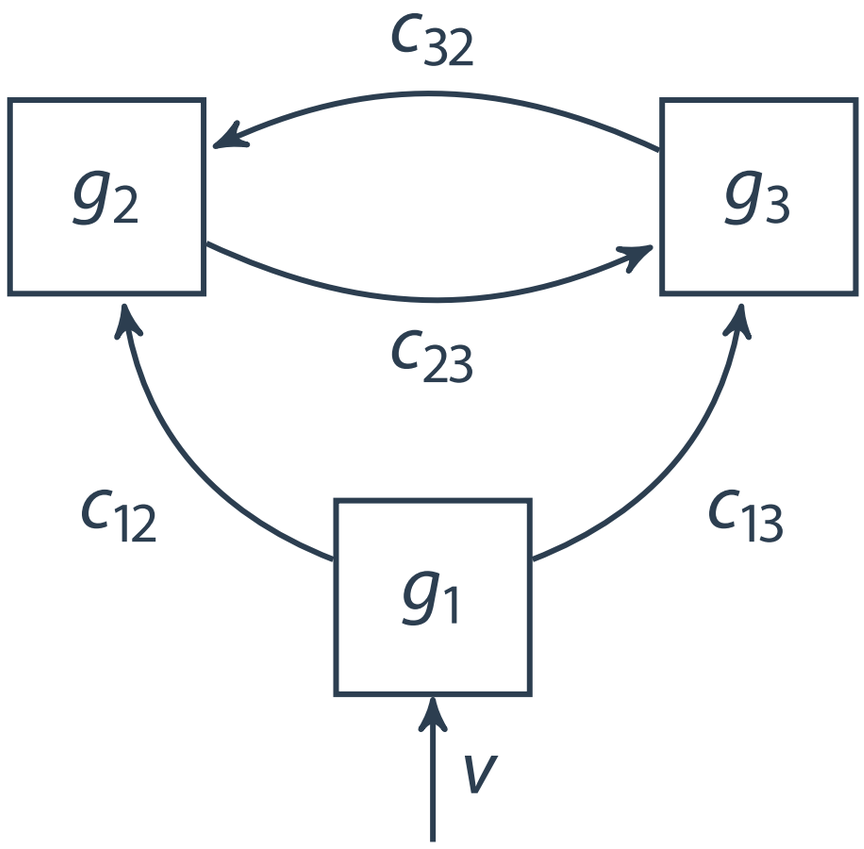
The unrealizable Byzantine Generals

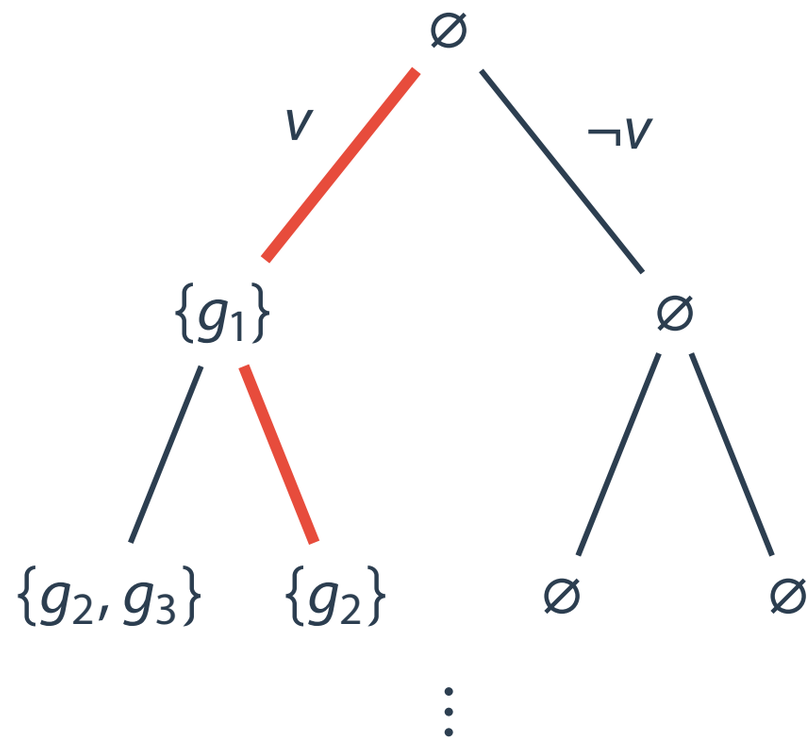




- 3 generals, consisting of 1 commander and 2 lieutenants, need to agree on a plan of attack
- One of them is a traitor
- Specification:
  - if the commander is the traitor, the lieutenants should reach consensus
  - if one of the lieutenants is the traitor, the loyal lieutenant should agree with the commander
- The specification is **unrealizable.**

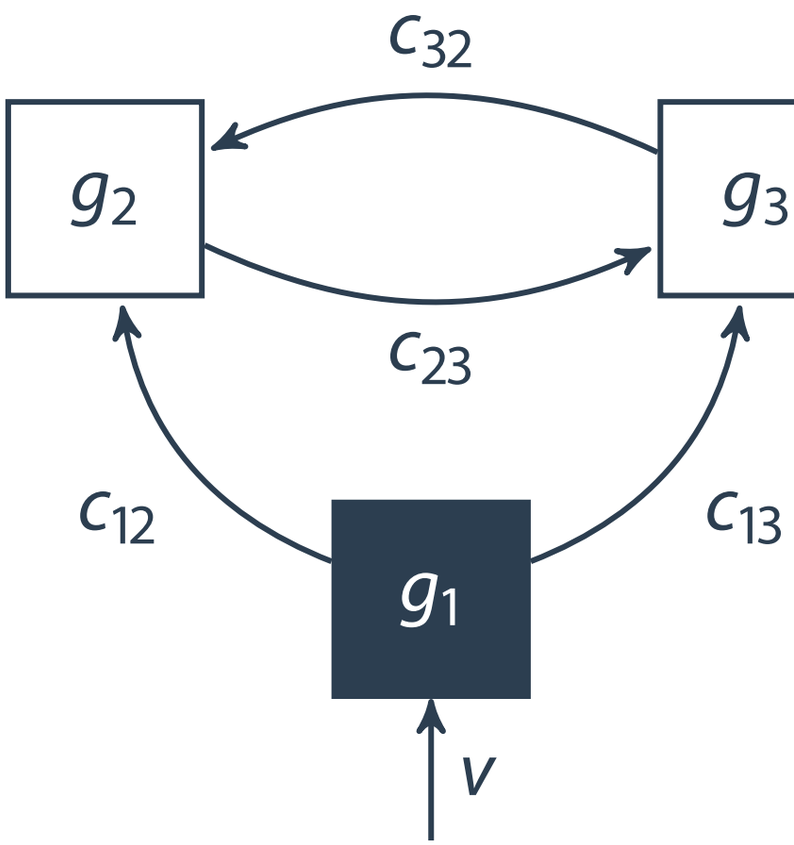
Counterexamples in model checking

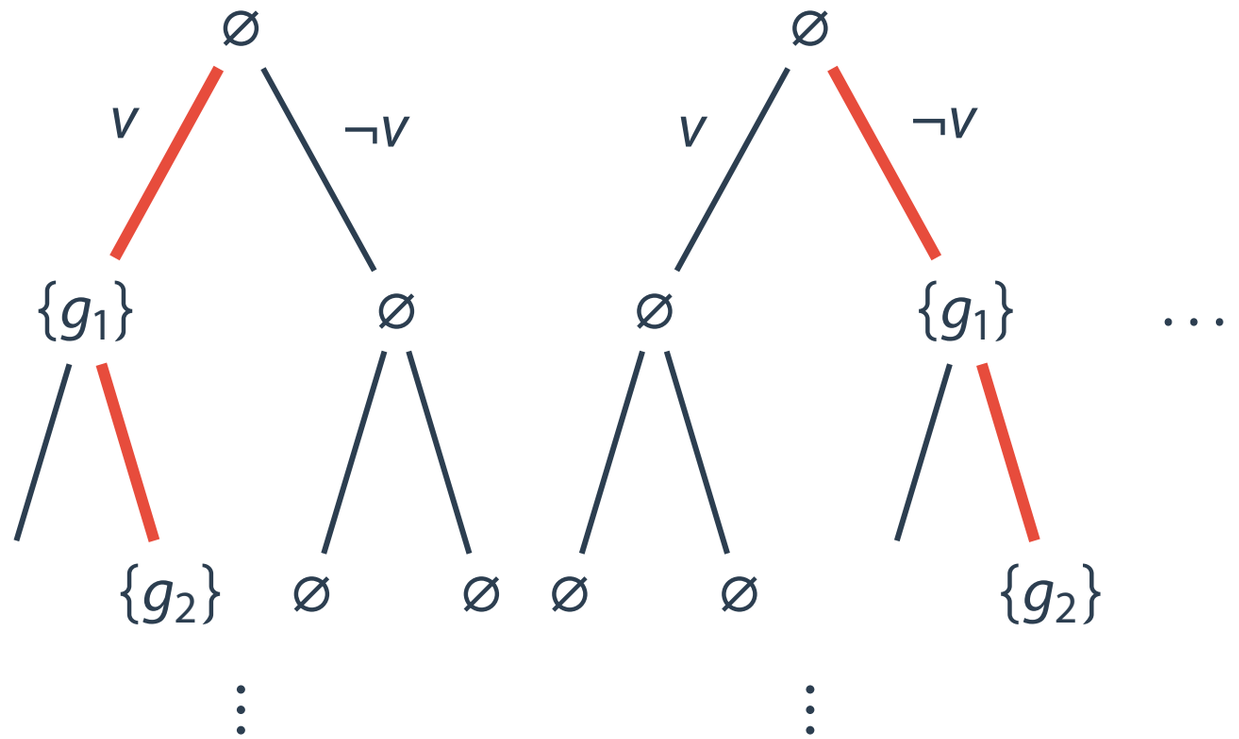




A counterexample in model checking is **one path** that violates φ.

Counterexamples in synthesis





A counterexample to realizability is a **set of paths** such that for **all** implementations **at least one** path violates φ.

Bounded counterexamples

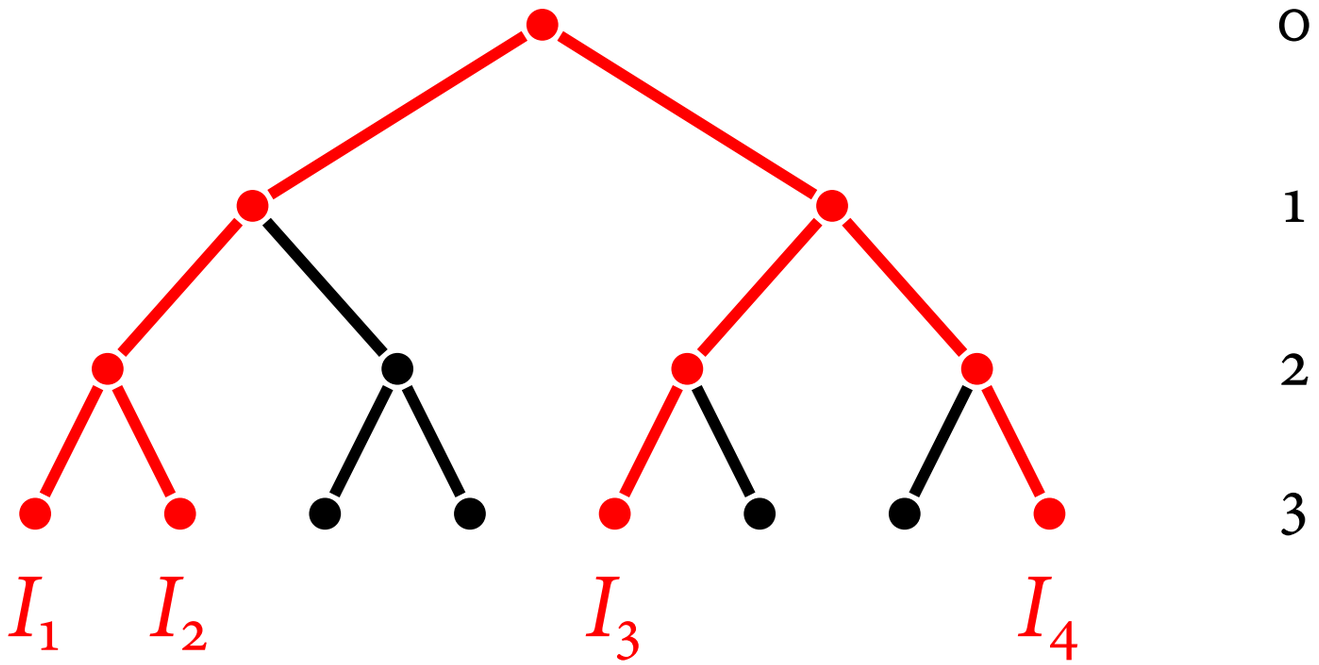
A counterexample to realizability is a **set of paths** such that for **all** implementations **at least one** path violates φ.

- **Bounded-size counterexamples:**
  bounded number of paths in the set
- **Bounded-length counterexamples:**
  paths with bounded length

Bounded counterexamples are a sufficient criterion
(but not in general necessary) for **unrealizability.**

[F./Tentrup 2014]

Encoding in QBF



$$\exists I_1^0, I_1^1, \ldots, I_1^k, \quad \ldots \quad, I_n^0, I_n^1, \ldots, I_n^k .$$

$$\forall S_1^0, S_1^1, \ldots, S_1^k, \quad \ldots \quad, S_n^0, S_n^1, \ldots, S_n^k .$$

$$\mathsf{consistent}(S_1^0, \ldots, S_1^k, \ldots, S_n^0, \ldots, S_n^k)$$

$$\rightarrow \bigvee_{1 \le i \le n} \mathsf{counterexample}(\varphi_i)$$

Part IV: Distributed Synthesis

1. Synthesis in the Pnueli/Rosner model
2. Bounded synthesis of distributed systems
3. **Synthesis in the causal memory model**

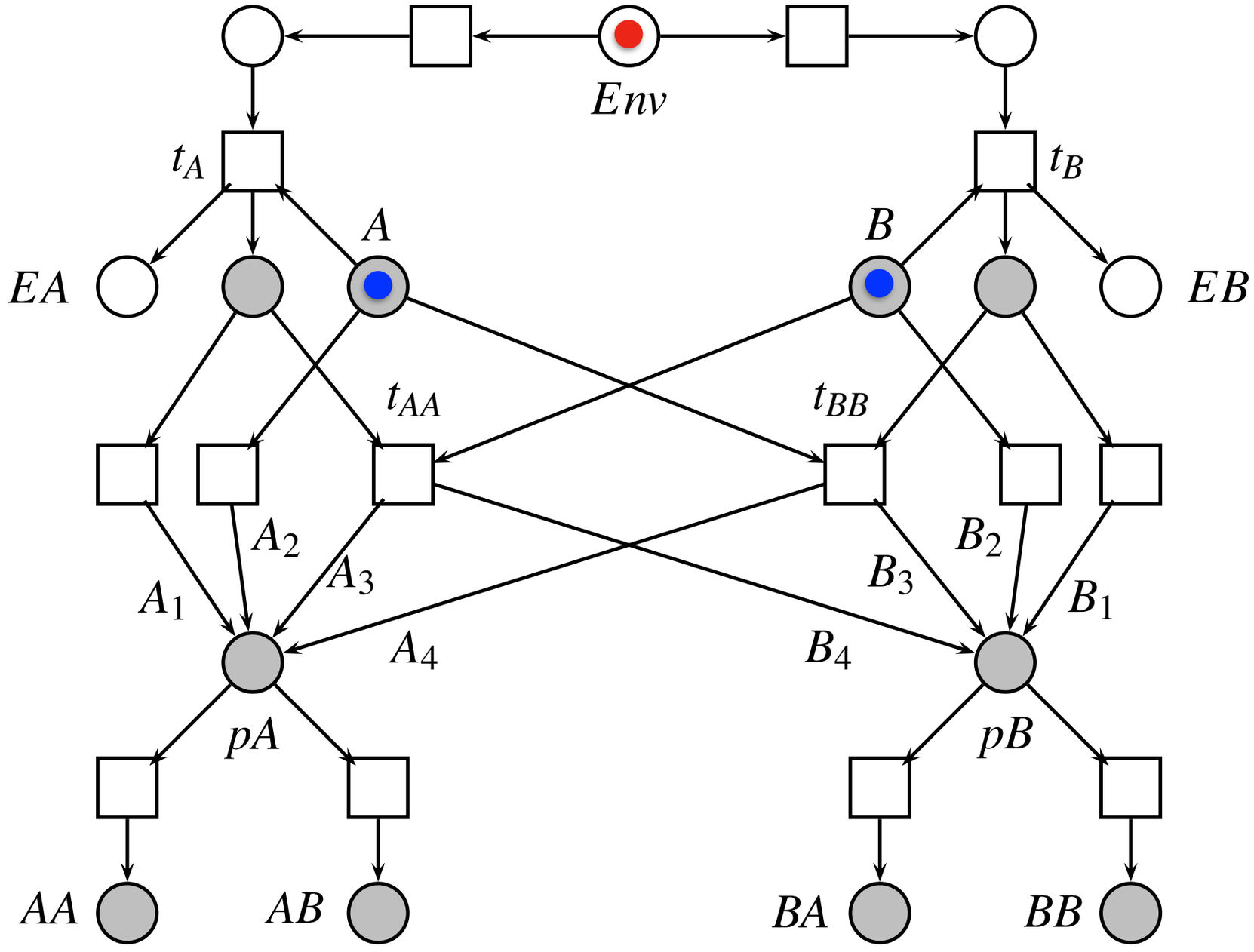
The causal memory model

- processes memorize their causal history
- processes communicate causal history to each other during each synchronization
- this abstracts from the content of a communication
  (it is part of the synthesis problem to determine what should be communicated during the synchronization)

- Hope: Since the communication is less restricted than under partial observation, the undecidability results do not carry over.

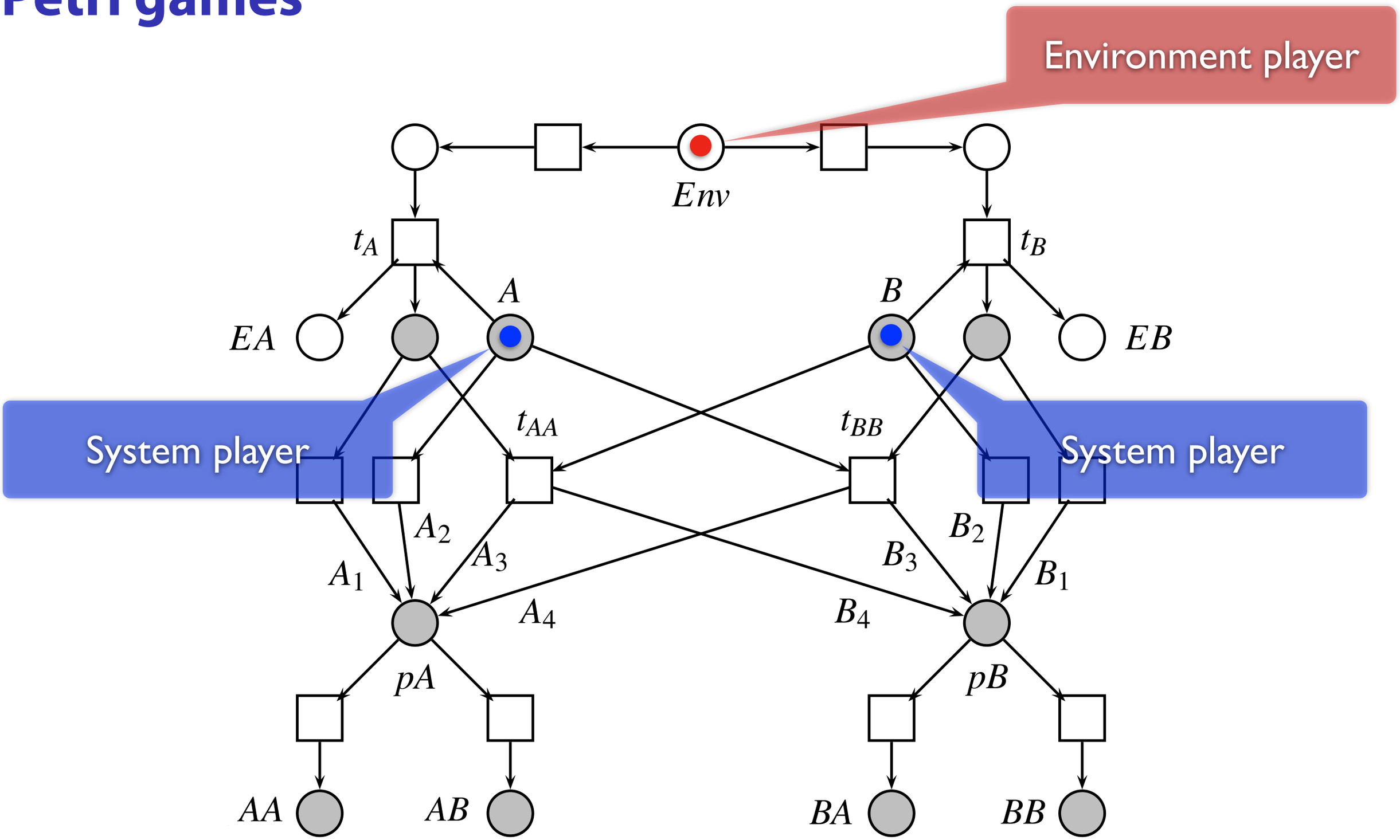
[Gastin/Lerman/Zeitoun, 2004]

Petri games



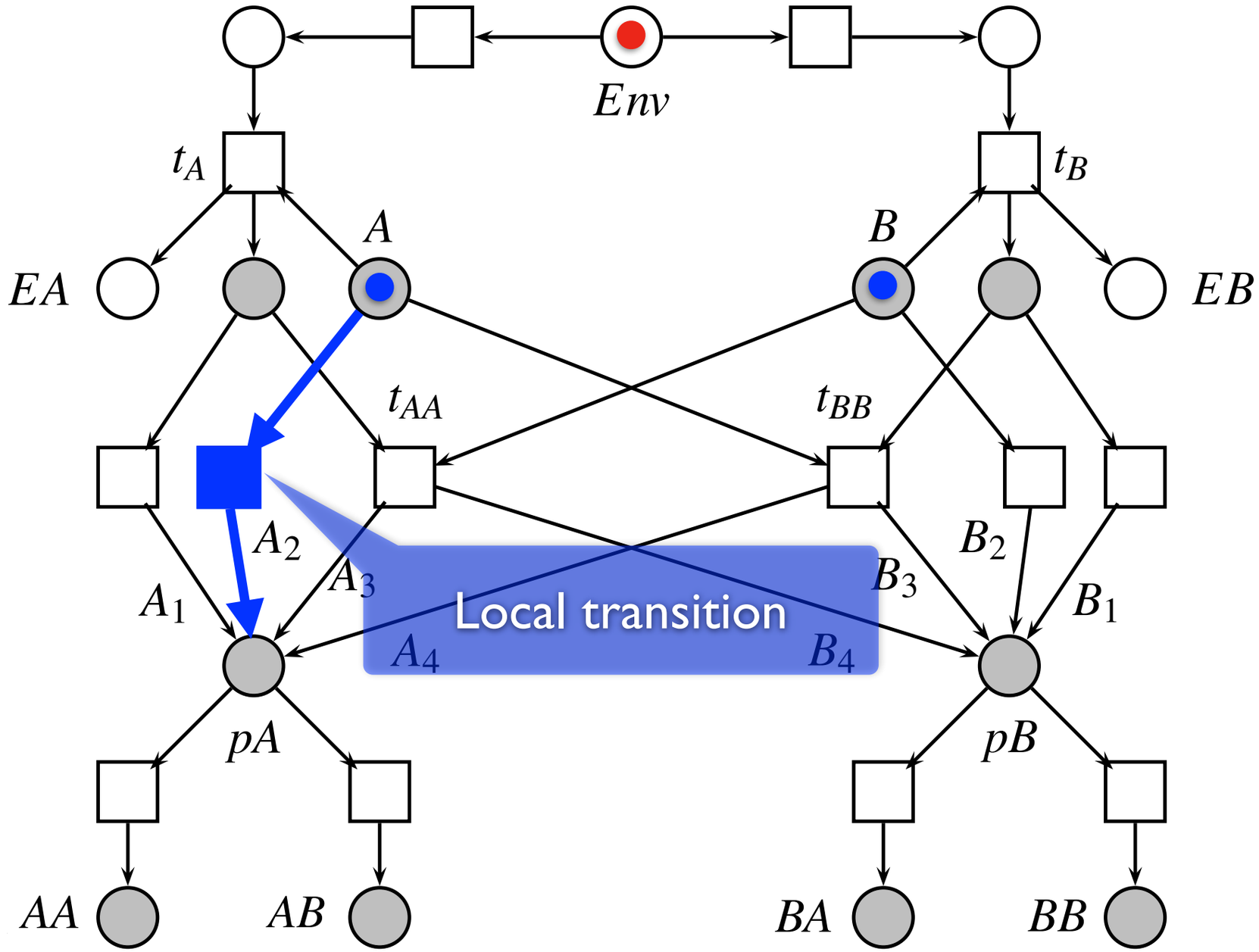
[F./Olderog, 2014]

Petri games



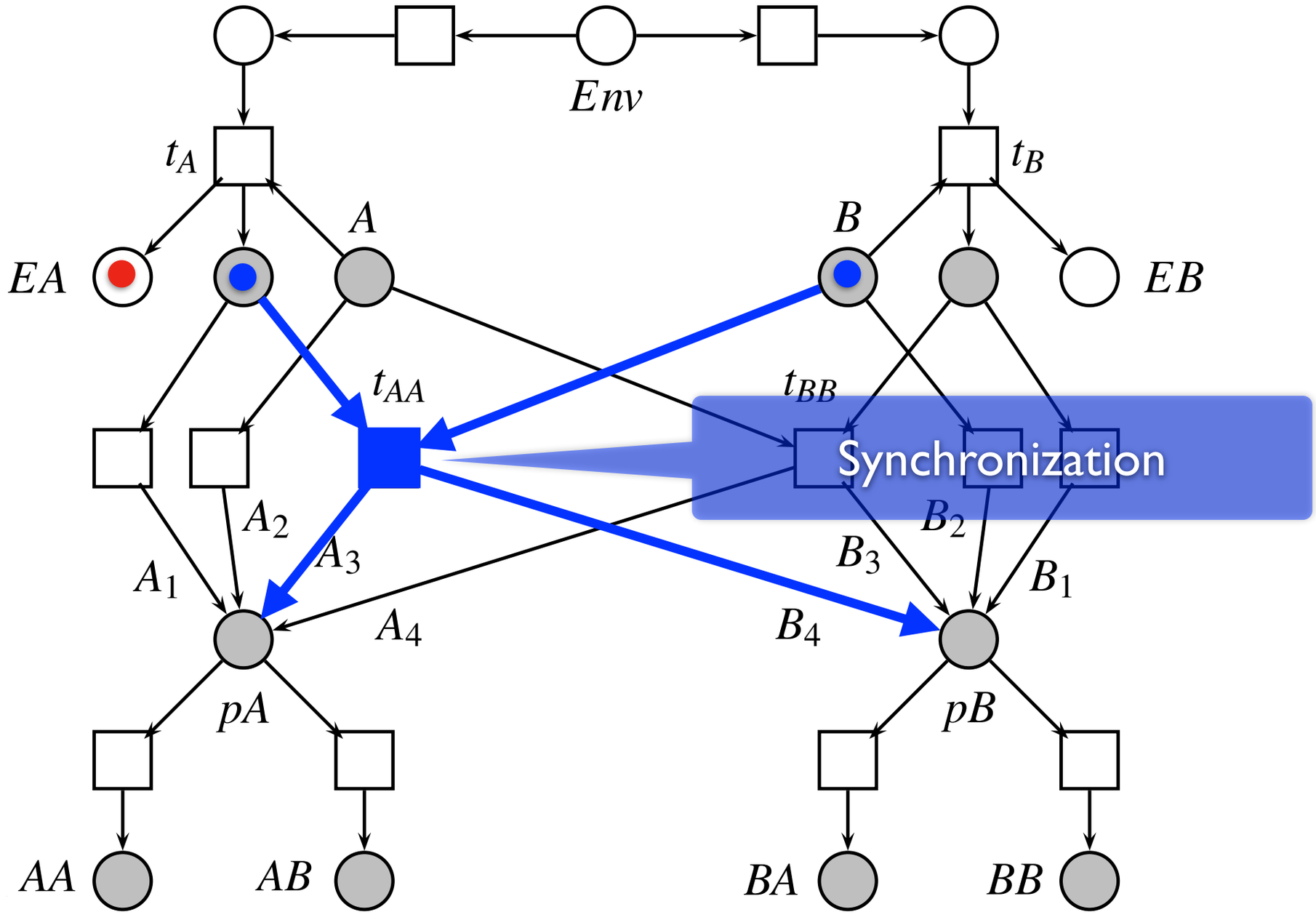
[F./Olderog, 2014]

Petri games

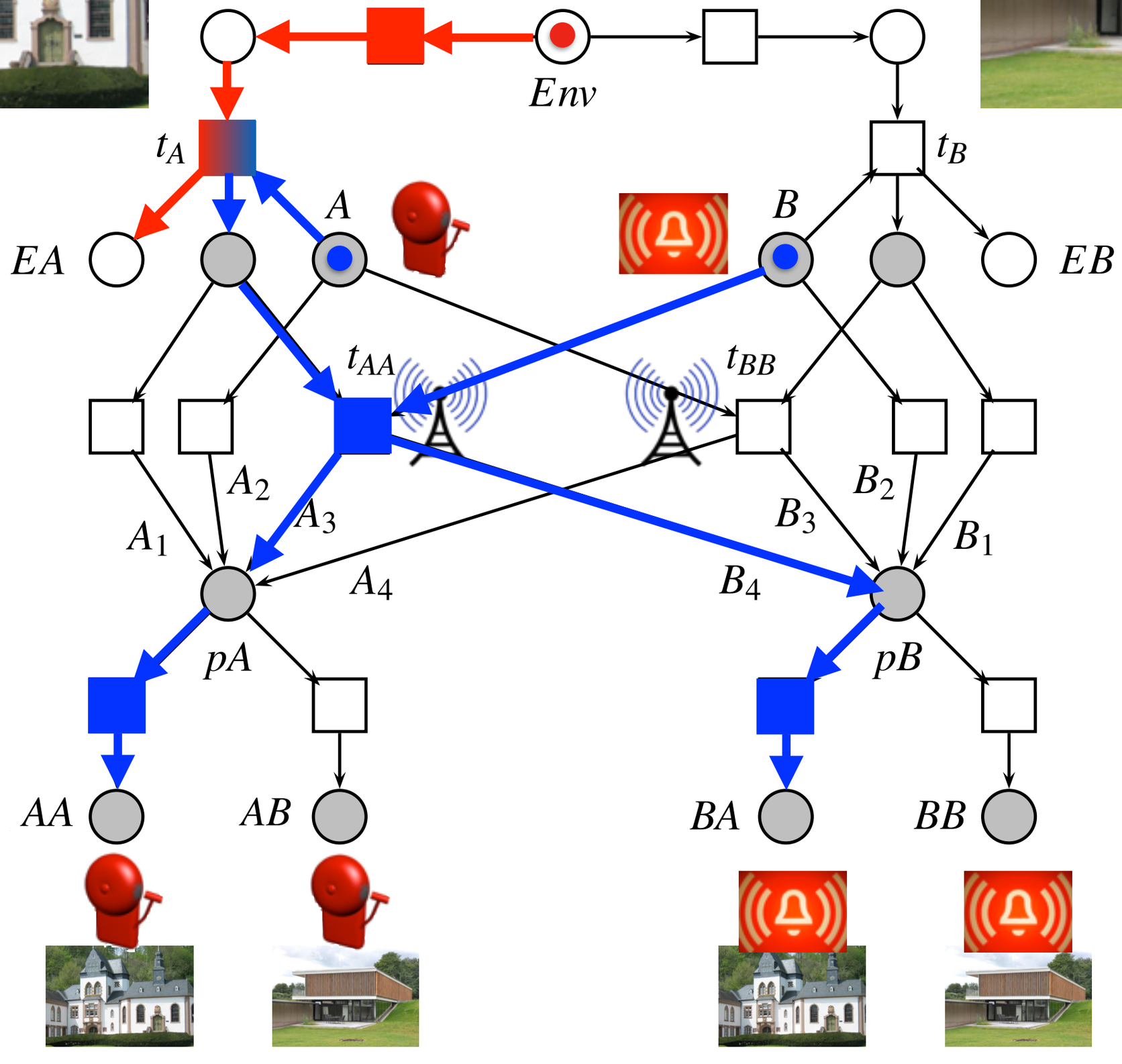


[F./Olderog, 2014]

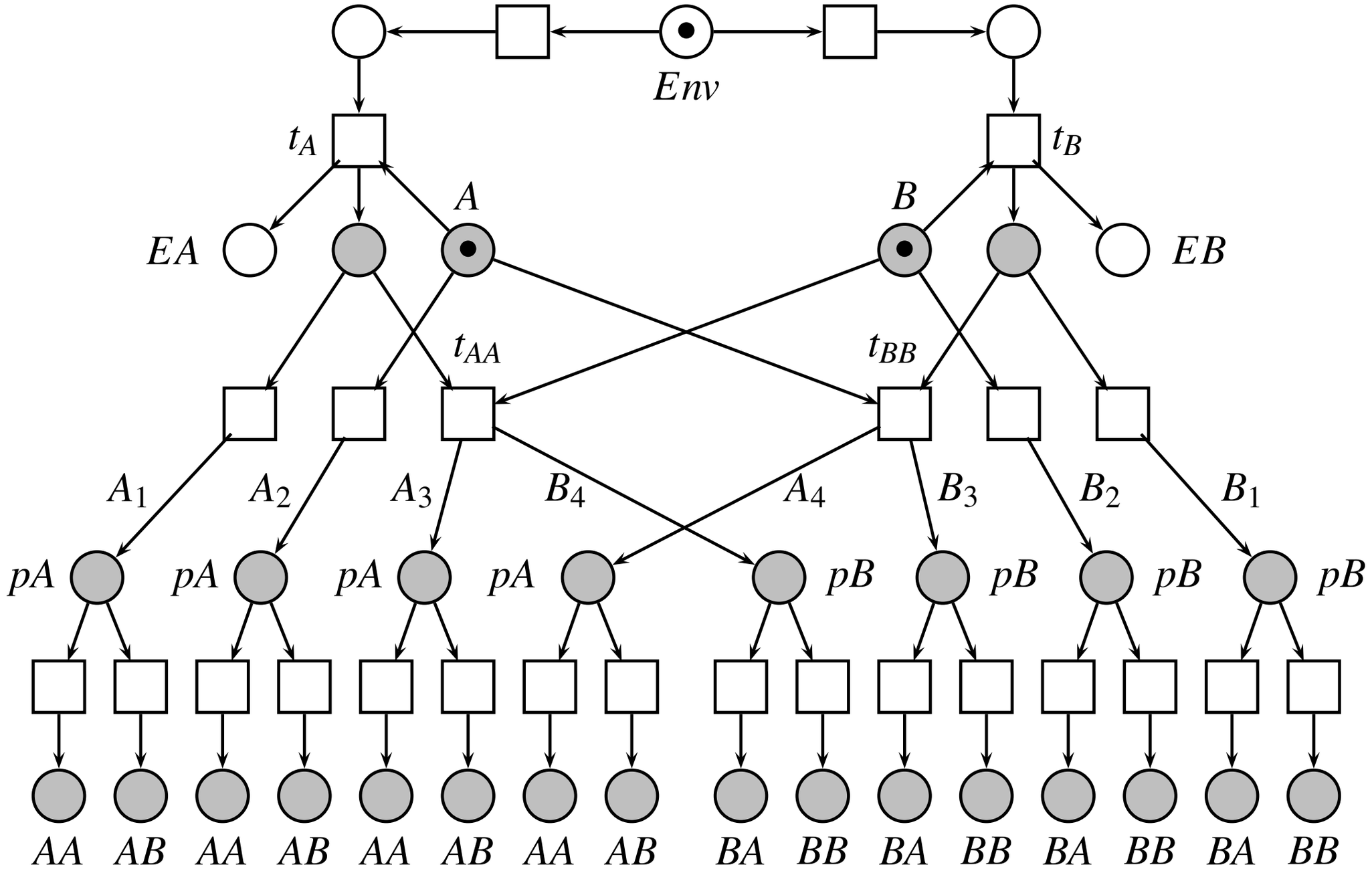
Petri games



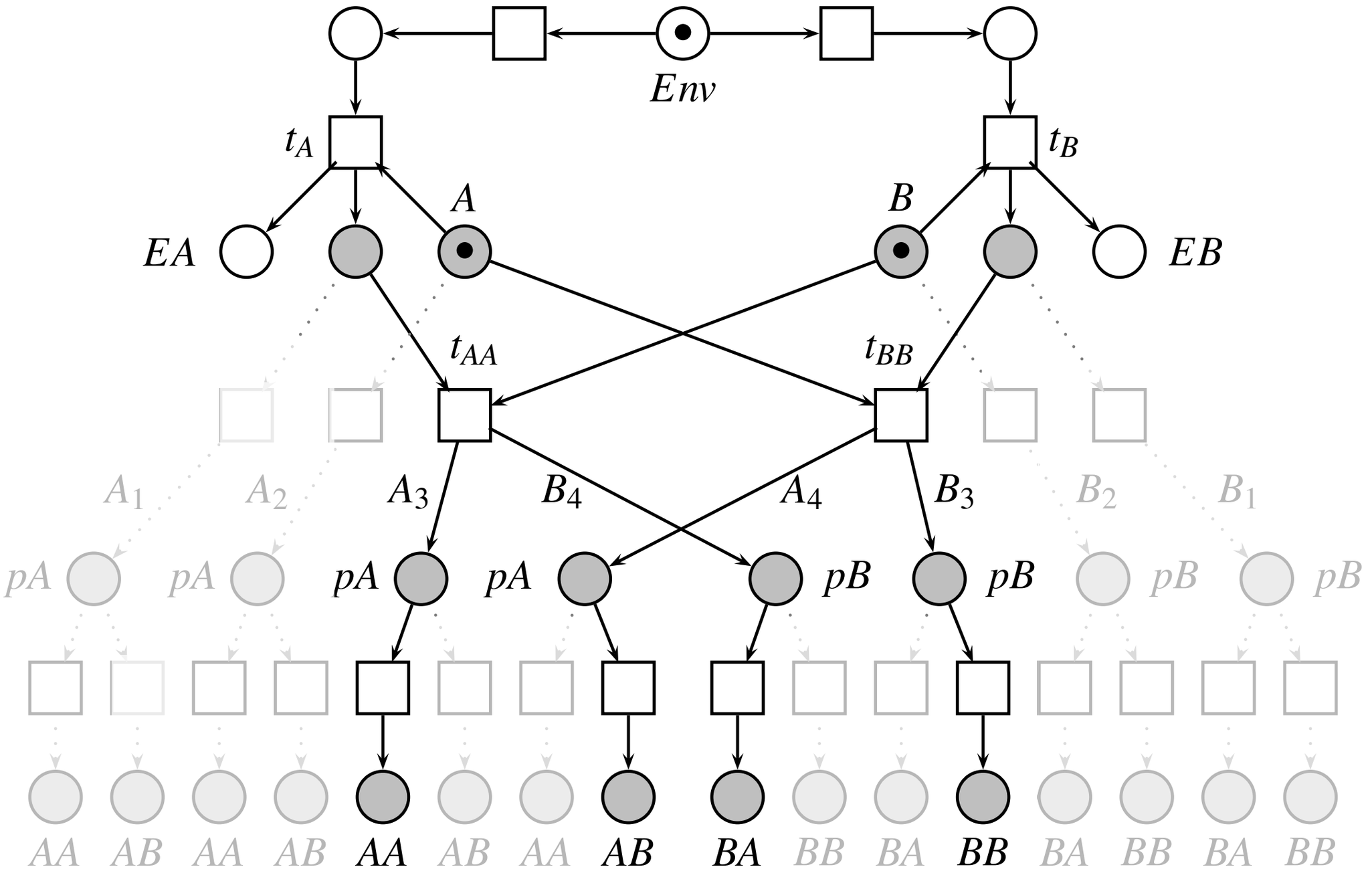
[F./Olderog, 2014]


Env
t_A
t_B
A
B
EA
EB
t_{AA}
t_{BB}
A_1
A_2
A_3
A_4
B_1
B_2
B_3
B_4
pA
pB
AA
AB
BA
BB

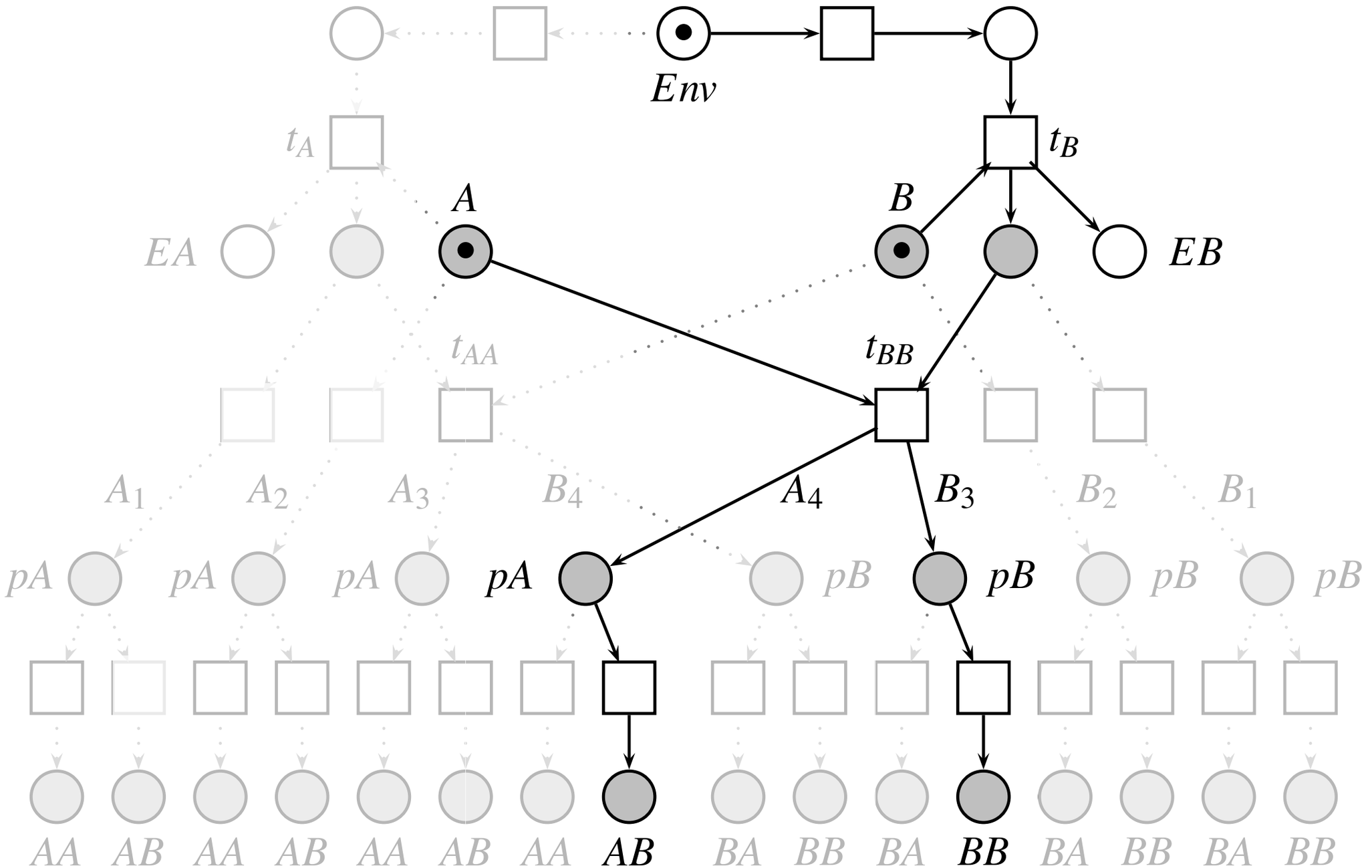
Petri games: Unfolding



Petri games: Global strategy



Petri games: Play



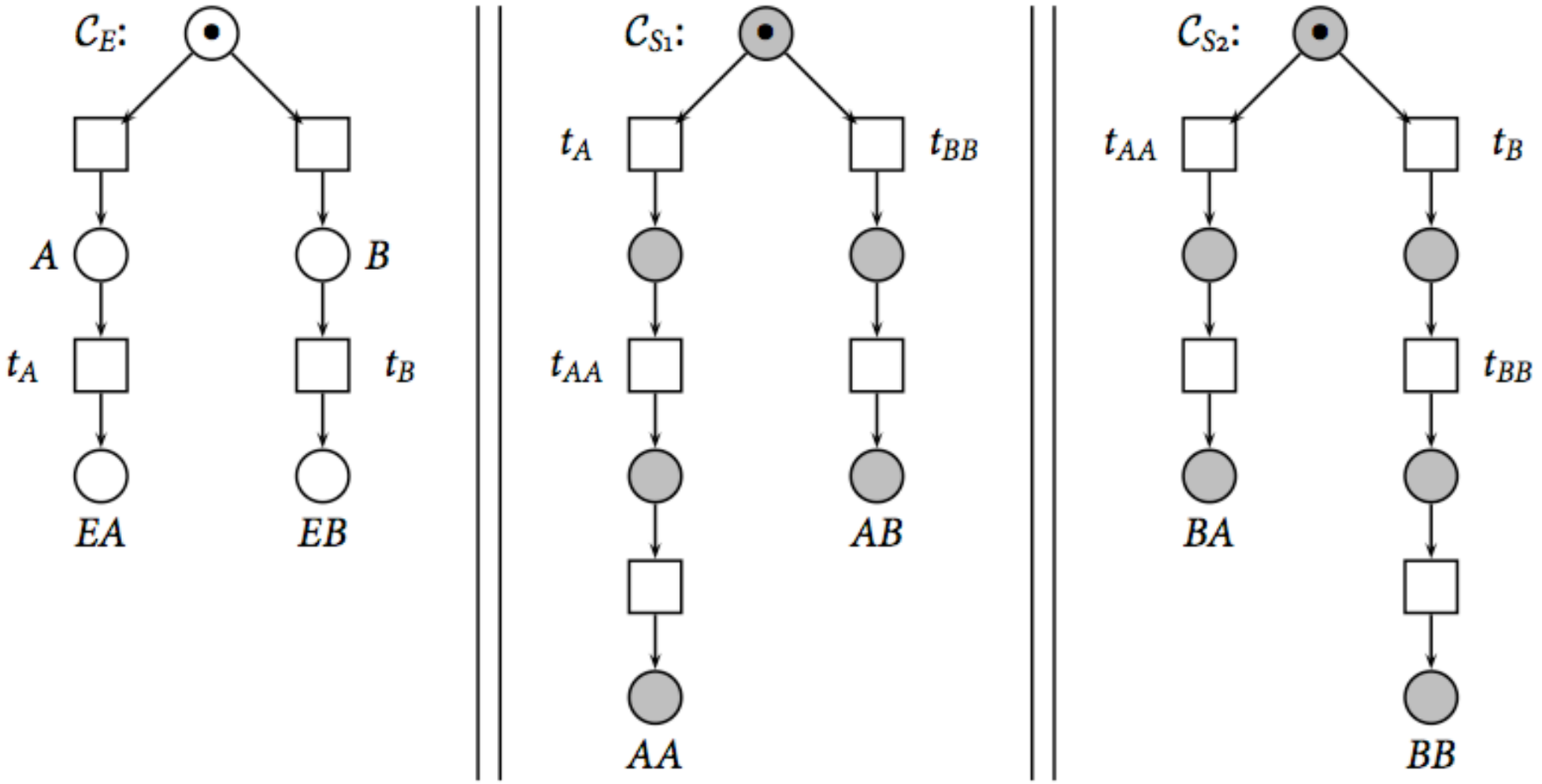
Strategy distribution

- A global strategy σ is distributable if σ can be represented as the parallel composition of local controllers for the environment and the system players in the sense that the reachable part of the parallel composition is isomorphic to σ.

Distribution Lemma

Every global strategy for a concurrency-preserving Petri game is distributable.

Petri games: Distributed strategy



$$\mathcal{N}_1 \,||\, \mathcal{N}_2 = (\mathcal{P}_1 \cup \mathcal{P}_2, \mathcal{T}_1 \cup \mathcal{T}_2, \mathcal{F}_1 \cup \mathcal{F}_2, In_1 \cup In_2)$$

Solving Petri games

EXPTIME-completeness

For Petri games with

- one environment player,
- a bounded number of system players,
- a safety objective (given as a set of safe places)

the question whether the system players have a deadlock-avoiding winning strategy is EXPTIME-complete. If a winning strategy for the system players exists, it can be constructed in exponential time.

Proof via reduction to Büchi games

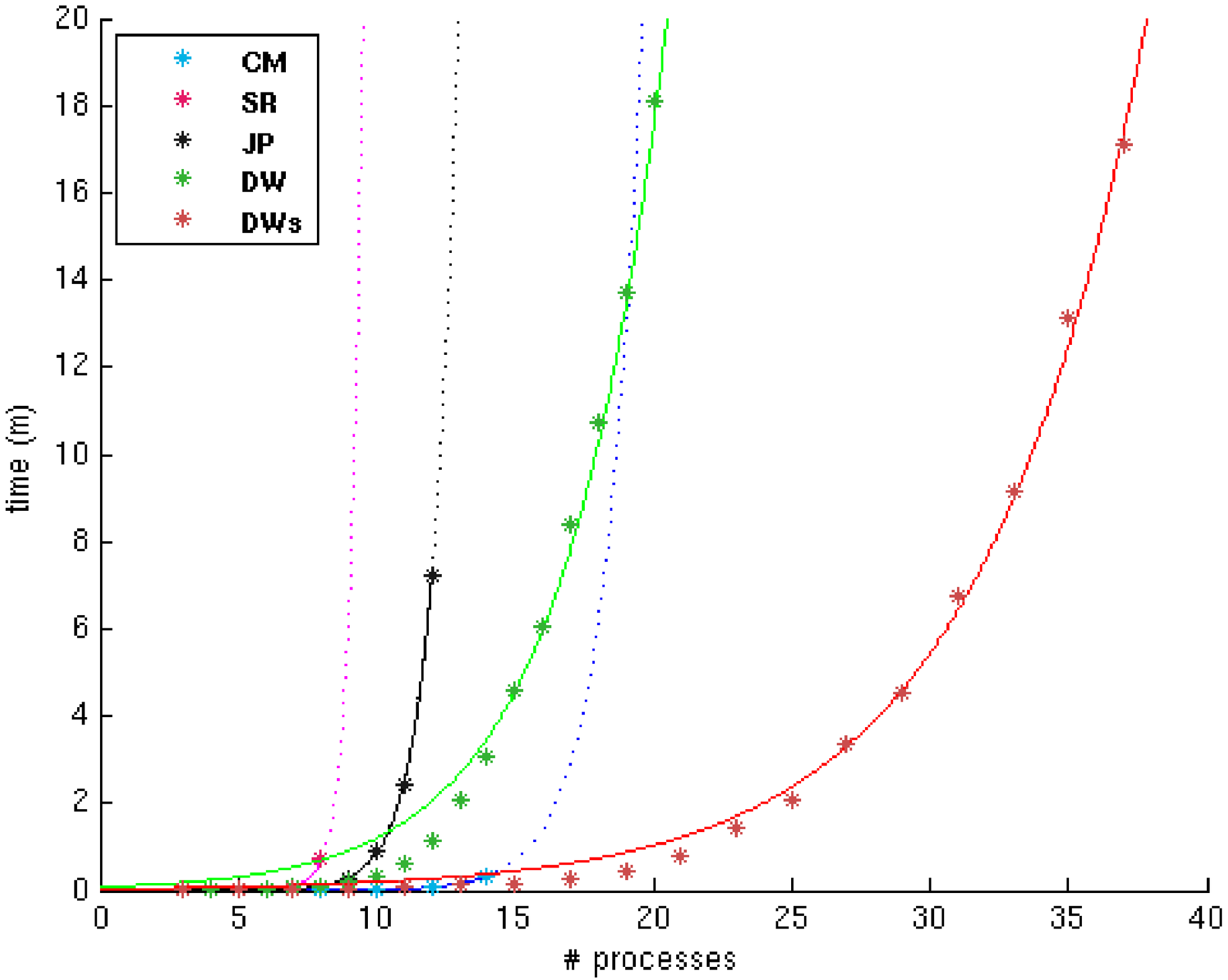
[F./Olderog, 2014]

ADAM

- Tool for the automated synthesis of distributed systems based on Petri games
- BDD-based symbolic representation of sets of cuts/mcuts
- Various optimizations (e.g., net invariants)
- Case studies from manufacturing with many (>10) independent agents

[F./Gieseking/Olderog, 2015]

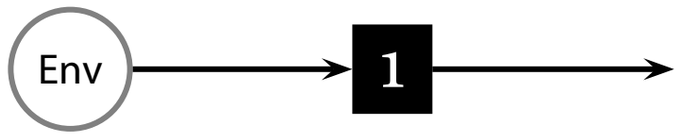
ADAM


CM
SR
JP
DW
DWs
time (m)
processes
0
2
4
6
8
10
12
14
16
18
20
0
5
10
15
20
25
30
35
40

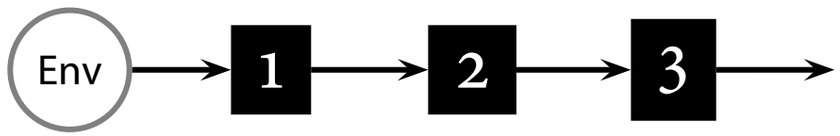
Conclusions

Reactive synthesis is a (really) hard problem

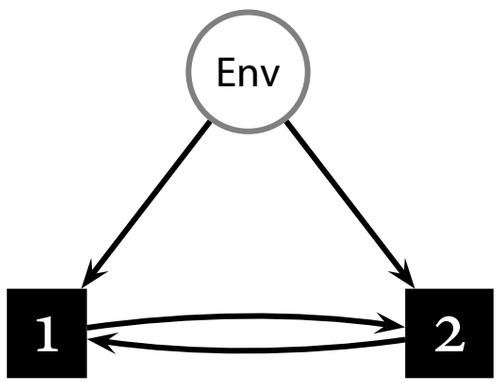
1-process architectures --- 2EXPTIME



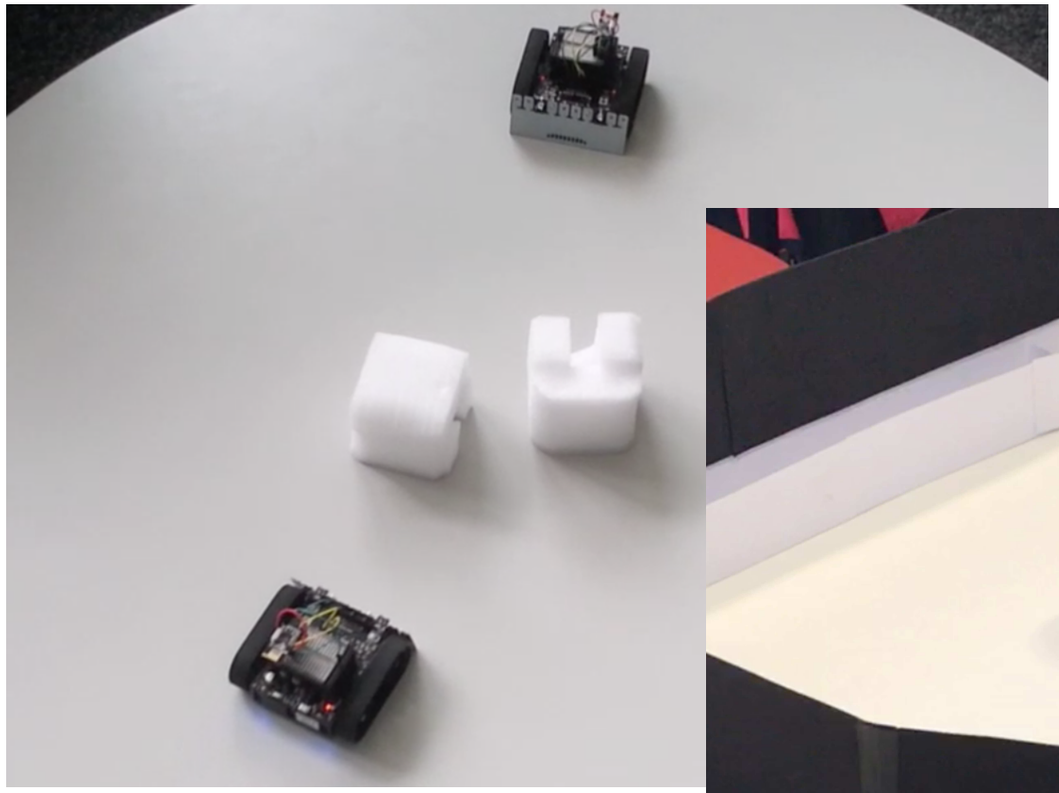
Pipeline architectures --- non-elementary

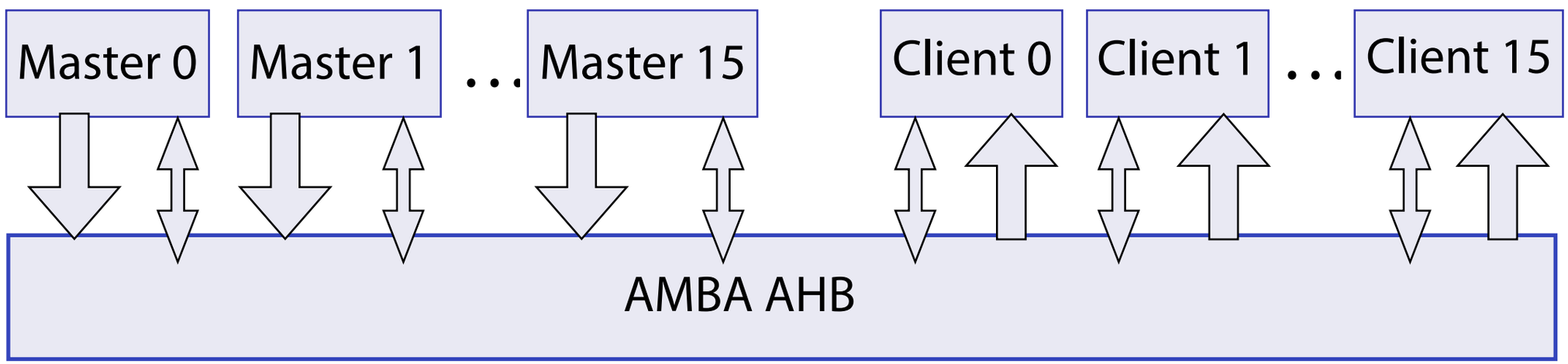


2-process arbiter architecture --- undecidable



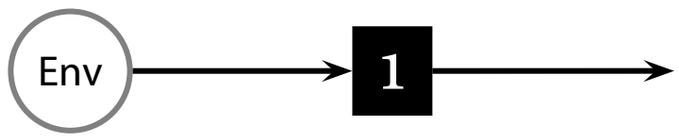
Success stories



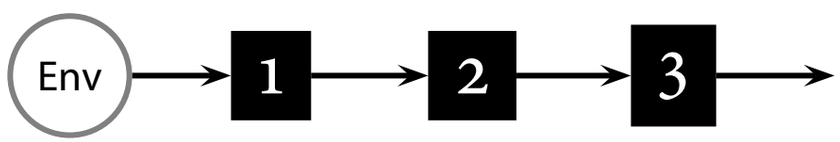


Bounded synthesis

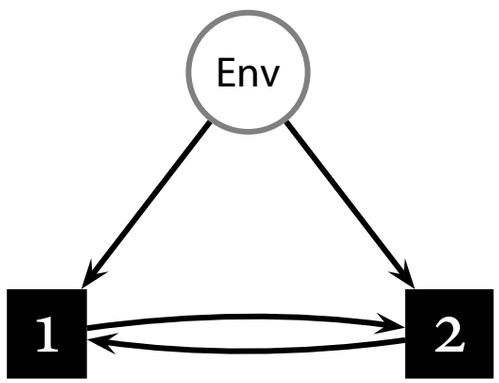
1-process architectures --- NP



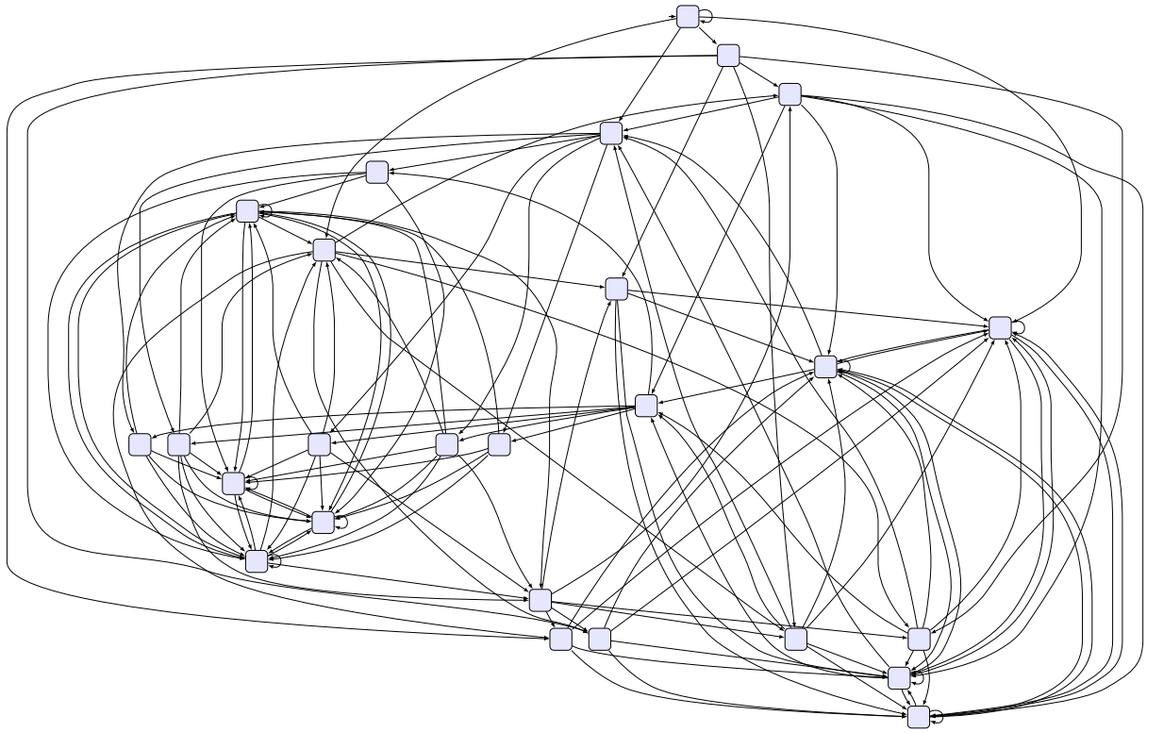
Pipeline architectures --- NP

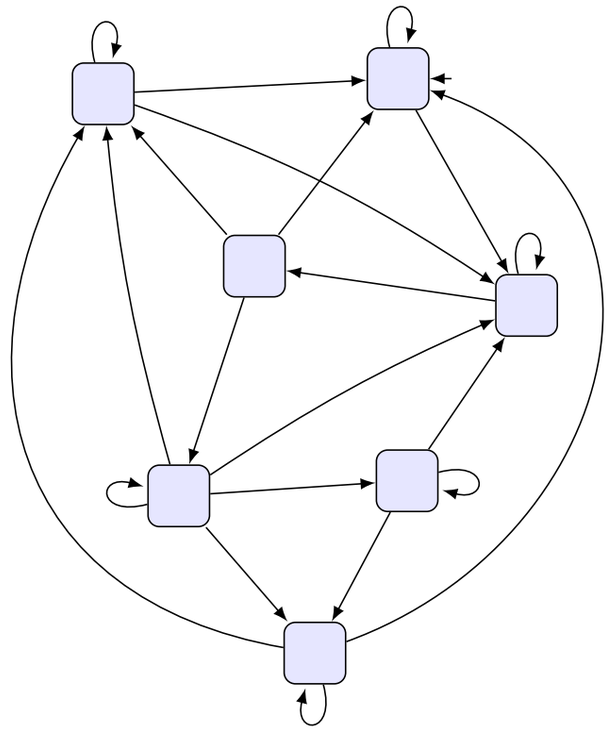


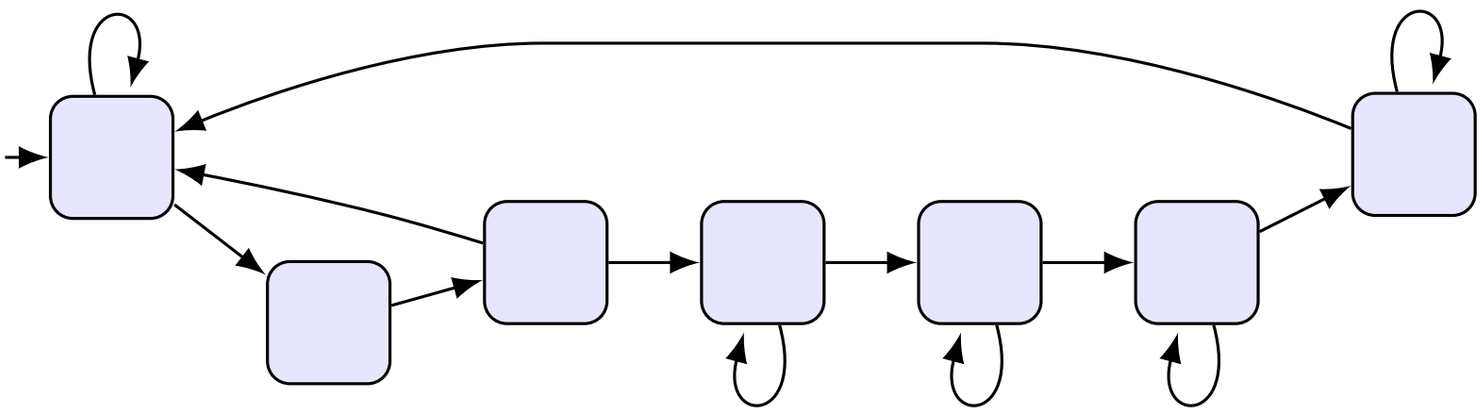
2-process arbiter architecture --- NP



Bounded cycle synthesis







Beyond bounded synthesis:
Output-sensitive algorithms for reactive synthesis

- Organize the search space such that the best solutions are found first.
- Parameters beyond size: structural complexity, logical complexity, degree of dependency between components, ...
- Find better implementations.
- Find them faster.

PhD/visitors/PostDoc positions available

www.osares.eu