LTL Reactive Synthesis with a Few Hints

Mrudula Balachander, Emmanuel Filiot, and Jean-François Raskin
Université libre de Bruxelles

IFIP Working group 2.2 meeting
Tallinn - July 2024

Synthesis

LTL Reactive Synthesis

$$\boxed{\text{Env}} \parallel \boxed{\ ?\ } \models \varphi$$

- Sys is constructed by an algorithm
- Sys is **correct** by construction
- Underlying theory: 2 player zero-sum **games** played on graphs
- Env is **adversarial** (worst-case assumption)
- Correct Sys = **Winning strategy**
- Main argument of Synthesis: **What** versus **How**

LTL Reactive Synthesis

$\forall \quad \bar{I} = i_0 i_1 \ldots i_n \ldots \in I^\omega \longrightarrow$ **Sys** $\longrightarrow \bar{O} = o_0 o_1 \ldots o_n \ldots \in O^\omega$

environment

synchronous response

$$i_0 o_0 i_1 o_1 \ldots i_n o_n \ldots \models \varphi$$

- Problem: given a LTL formula φ over $AP = I \uplus O$, decide if there exists a strategy $\sigma : I^* \cdot I \rightarrow O$ such that for **all** sequences of inputs (=env. is adversarial) $\bar{I} = i_0 i_1 \ldots i_n \ldots \in I^\omega$:

$$i_0 \cdot \sigma(i_0) \cdot i_1 \cdot \sigma(i_0 i_1) \cdot \ldots \cdot i_n \cdot \sigma(i_0 i_1 \ldots i_n) \cdot \ldots \models \varphi$$

and if so, construct such a strategy.

LTL Reactive Synthesis

An example - **Mutual exclusion**

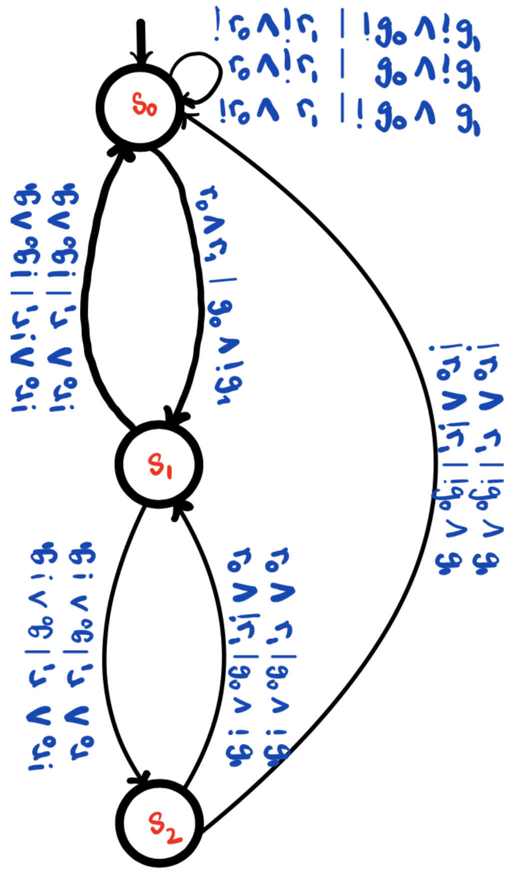
- **Input AP**: r_0, r_1
- **Output AP**: g_0, g_1
- **CORE Spec**:
  - $\varphi_{\text{CORE}} \equiv \square(\neg g_0 \vee \neg g_1) \wedge \square(r_0 \rightarrow \Diamond g_0) \wedge \square(r_1 \rightarrow \Diamond g_1)$

CORE Spec = properties that you would check on any solution to mutual exclusion (e.g. Peterson, Dedecker, etc.) - Remember “What vs. How”

Mealy Machine that Realizes Spec

$$\forall \quad \bar{I} = i_0 i_1 \dots i_n \dots \in I^\omega \longrightarrow \boxed{M} \longrightarrow \bar{O} = o_0 o_1 \dots o_n \dots \in O^\omega$$

$$i_0 o_0 i_1 o_1 \dots i_n o_n \dots \models \varphi_{\text{CORE}}$$



$$\models \varphi_{\text{CORE}}$$

- A Mealy machine is an **input-complete** deterministic automaton with **outputs** that encodes a **strategy**
- M realizes φ if $\forall \bar{I} \in I^\omega : M(\bar{I}) \models \varphi_{\text{CORE}}$, noted $M \models \varphi_{\text{CORE}}$

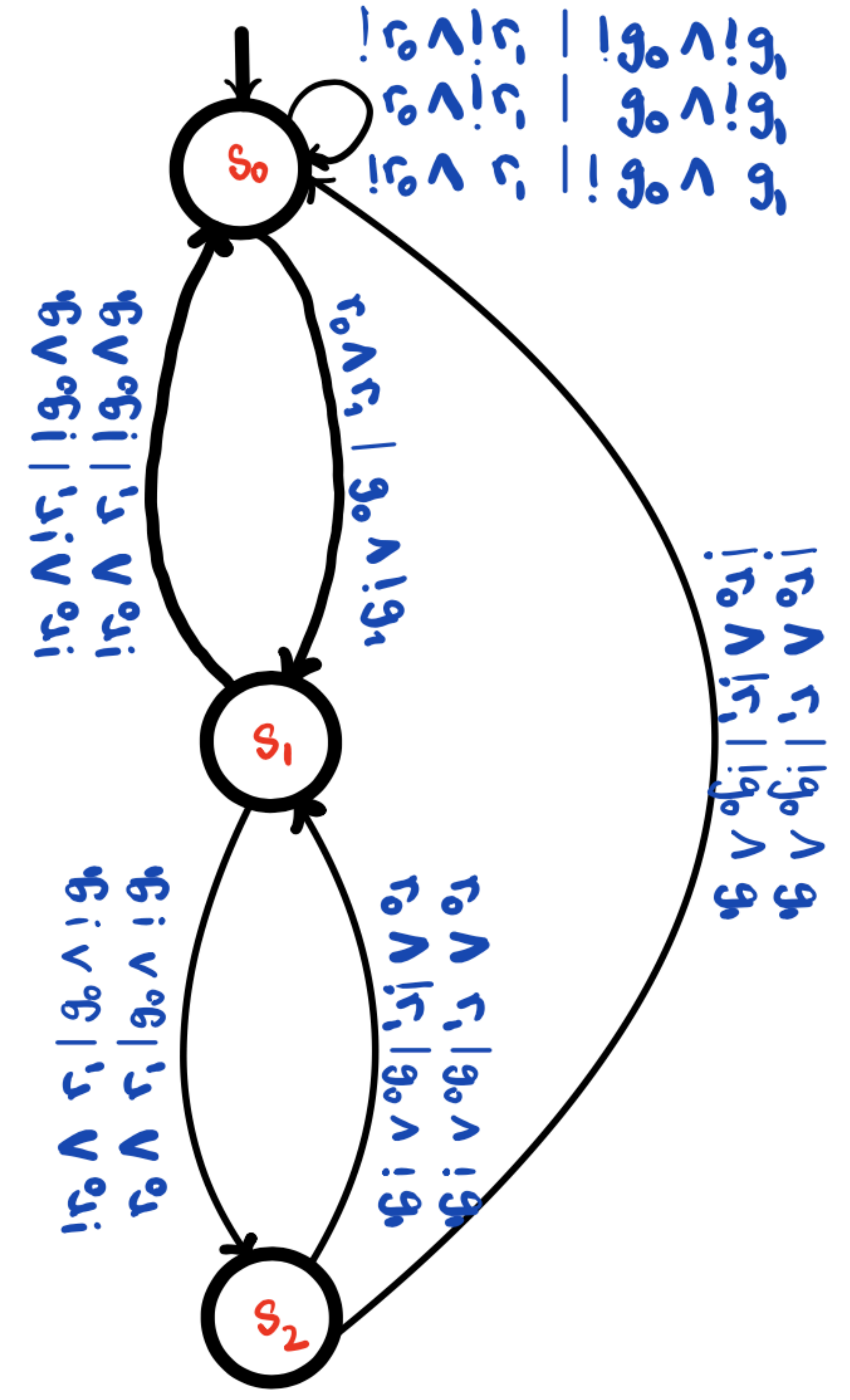
LTL Reactive Synthesis

An example - Mutual exclusion

- **Input AP**: r_0, r_1
- **Output AP**: g_0, g_1
- **CORE Spec**:
    - $\varphi_{\text{CORE}} \equiv \Box(\neg g_0 \vee \neg g_1) \wedge \Box(r_0 \rightarrow \Diamond g_0) \wedge \Box(r_1 \rightarrow \Diamond g_1)$

Synthesis algorithm

?



Solution=Winning Strategy=**Mealy Machine**

LTL Reactive Synthesis

- LTL reactive synthesis is **2ExpTime-C**
- Nevertheless, “efficient” implementations exist
  (medium size spec - $\approx$ one page) - e.g. Acacia (ULB-U Antwerpen) - STRIX (TUM)
- Demo: Mutual exclusion (STRIX)

LTL Reactive Synthesis

- LTL reactive synthesis is **2ExpTime-C**
- Nevertheless, “efficient” implementations exist
  (medium size spec - ≈ one page) - e.g. Acacia (ULB-U Antwerpen) - STRIX (TUM)
- Demo: Mutual exclusion (STRIX)

Assumptions:

```
true
```

Guarantees:

```
G (!grant_0 | !grant_1)
G (request_0 -> F grant_0)
G (request_1 -> F grant_1)
```

Input propositions:

```
request_0, request_1
```

Output propositions:

```
grant_0, grant_1
```

LTL Reactive Synthesis

- LTL reactive synthesis is **2ExpTime-C**
- Nevertheless, “efficient” implementations exist
  (medium size spec - ≈ one page) - e.g. Acacia (ULB-U Antwerpen) - STRIX (TUM)
- Demo: Mutual exclusion (STRIX)

Assumptions:

```
true
```

Guarantees:

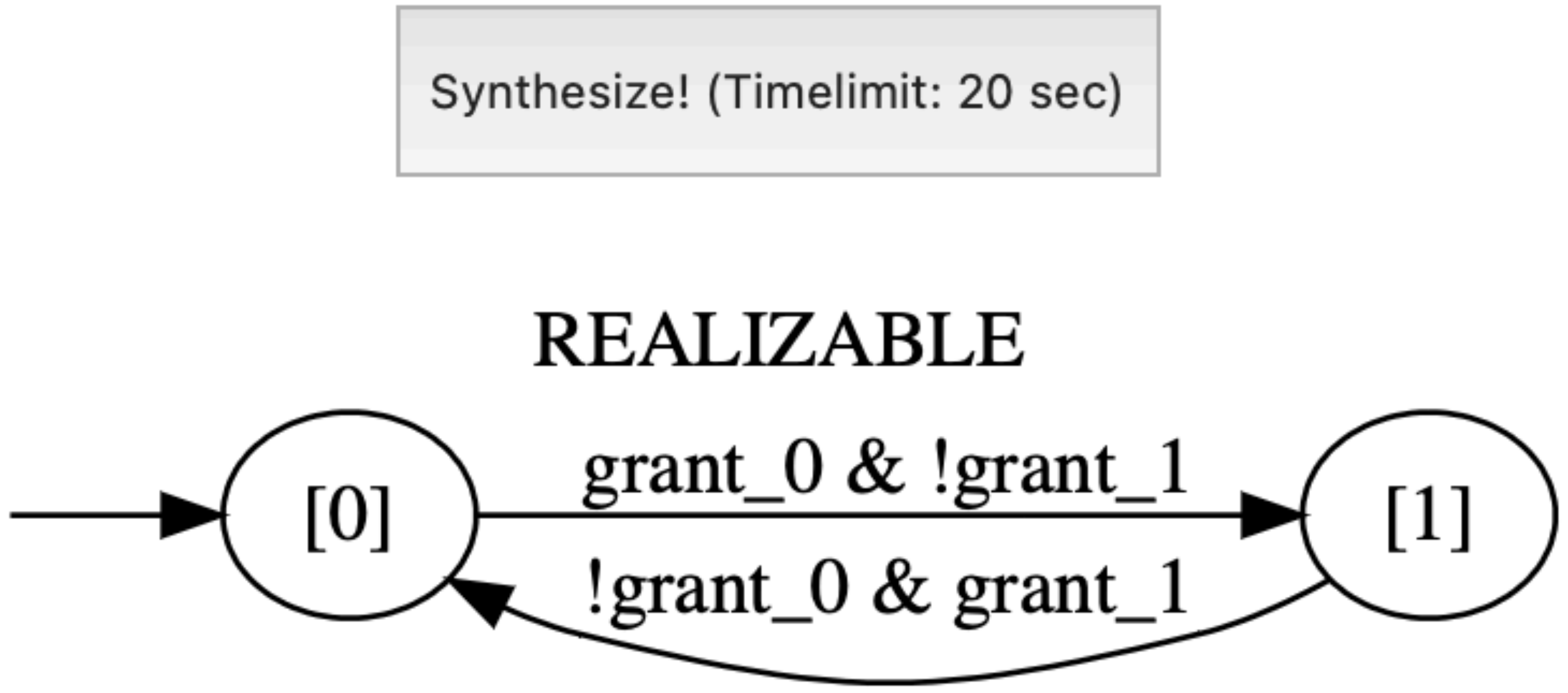
```
G (!grant_0 | !grant_1)
G (request_0 -> F grant_0)
G (request_1 -> F grant_1)
```

Input propositions:

```
request_0, request_1
```

Output propositions:

```
grant_0, grant_1
```



LTL Reactive Synthesis

- LTL reactive synthesis is **2ExpTime-C**
- Nevertheless, “efficient” implementations exist
  (medium size spec - ≈ one page) - e.g. Acacia (ULB-U Antwerpen) - STRIX (TUM)
- Demo: Mutual exclusion (STRIX)

Assumptions:

```
true
```

Guarantees:

```
G (!grant_0 | !grant_1)
G (request_0 -> F grant_0)
G (request_1 -> F grant_1)
```

Input propositions:

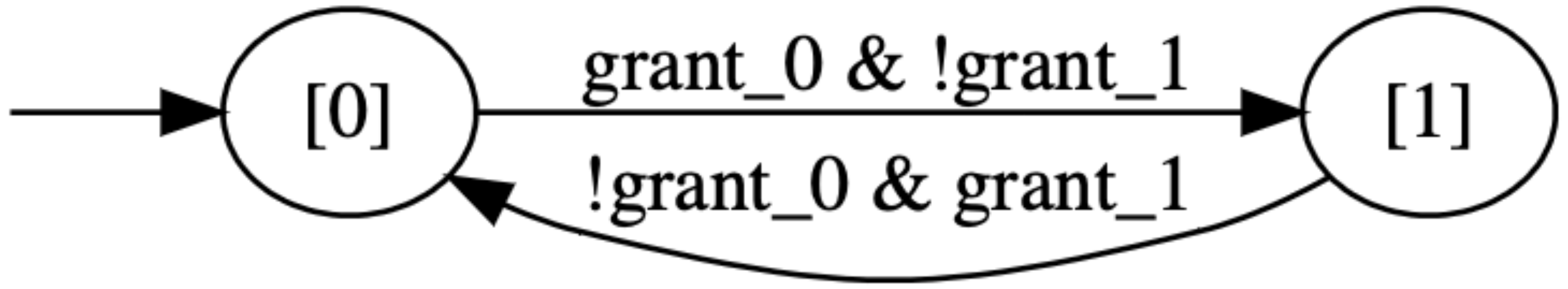
```
request_0, request_1
```

Output propositions:

```
grant_0, grant_1
```

Synthesize! (Timelimit: 20 sec)

REALIZABLE



Quality of solution ?
Small is beautiful ?
is “What only” sufficient ?

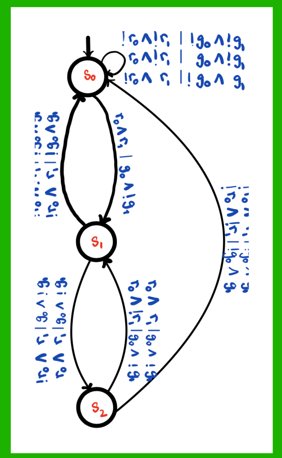
Space of solutions of φ_{CORE}

How to drive the synthesis procedure to good solutions?

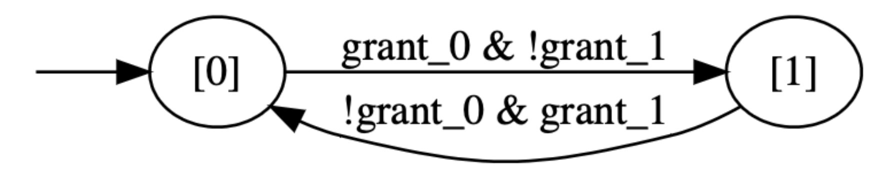
Mealy Machines

Mealy machines that realizes φ_{CORE}

"Good"solutions



...



...

Quality of solutions

- The **“What only”** may lead to solutions that are not of practical interest
  (e.g. unsolicited grants)
- Remedy ? Give a “**complete”** specification
- Example: Mutual exclusion without **unsolicited grants**

Quality of solutions

- The **"What only"** may lead to solutions that are not of practical interest (e.g. unsolicited grants)
- Remedy ? Give a "**complete**" specification
- Example: Mutual exclusion without **unsolicited grants**

Assumptions:

```
true
```

Guarantees:

```
G ((grant_0 & G !request_0) -> (F !grant_0))
G ((grant_1 & G !request_1) -> (F !grant_1))
G ((grant_0 & X (!request_0 & !grant_0)) -> X (request_0 R !grant_0))
G ((grant_1 & X (!request_1 & !grant_1)) -> X (request_1 R !grant_1))
G (!grant_0 | !grant_1)
request_0 R !grant_0
request_1 R !grant_1
G (request_0 -> F grant_0)
G (request_1 -> F grant_1)
```

Input propositions:

```
request_0, request_1
```

Output propositions:

```
grant_0, grant_1
```

Quality of solutions

- The **“What only”** may lead to solutions that are not of practical interest (e.g. unsolicited grants)
- Remedy ? Give a “**complete**” specification
- Example: Mutual exclusion without **unsolicited grants**

Assumptions:

```
true
```

Guarantees:

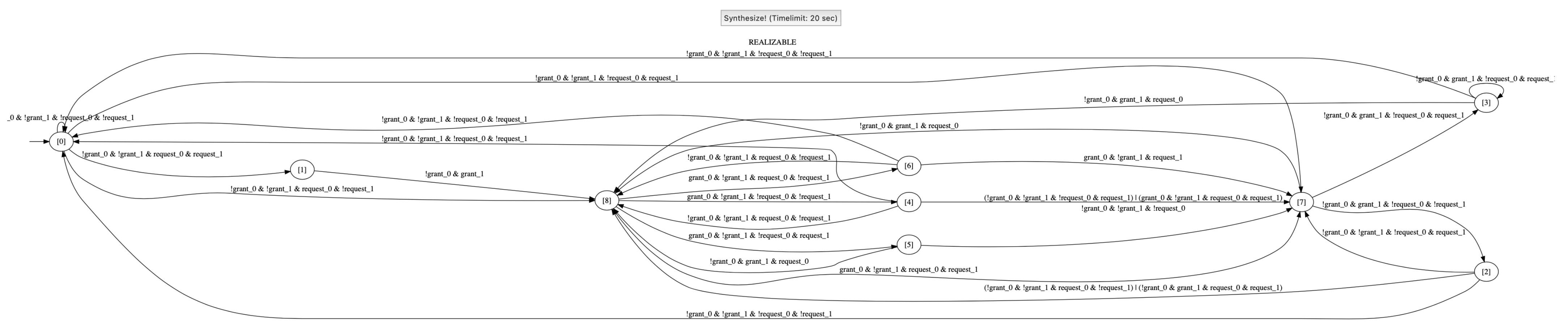
```
G ((grant_0 & G !request_0) -> (F !grant_0))
G ((grant_1 & G !request_1) -> (F !grant_1))
G ((grant_0 & X (!request_0 & !grant_0)) -> X (request_0 R !grant_0))
G ((grant_1 & X (!request_1 & !grant_1)) -> X (request_1 R !grant_1))
G (!grant_0 | !grant_1)
request_0 R !grant_0
request_1 R !grant_1
G (request_0 -> F grant_0)
G (request_1 -> F grant_1)
```

Input propositions:

```
request_0, request_1
```

Output propositions:

```
grant_0, grant_1
```



Quality of solutions

- The **"What only"** may lead to solutions that are not of practical interest
- A…
  s…
- Ex…
  exc…
  uns… …nts

Assumptions:

```
1 true
```

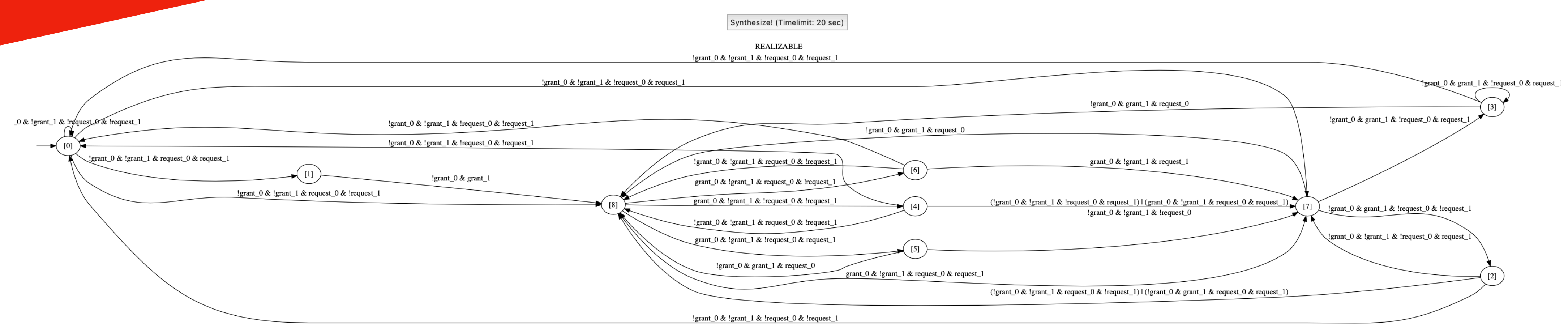
Guarant…

BUT

Specifying low level requirements in LTL may be difficult/cumbersome

It is not clear that it is the Mealy machine that we are looking for

Specifying lower level requirements clear goes against the very idea of synthesis!



Our proposal:
Add scenarios (examples)
to φ_{CORE}

$$\{\,!r_0, !r_1\,\}\,.\,\{\,!g_0, !g_1\,\}\#\{r_0, !r_1\,\}\,.\,\{g_0, !g_1\}\#\{\,!r_0, r_1\,\}\,.\,\{\,!g_0, g_1\,\}$$

Requirement engineering and scenarios

Formal spec and scenarios are complementary

- Scenarios are accepted in RE as an adequate tool to **elicit** requirements
- Scenarios are **easy** to produce: the designer controls **both** the inputs and the outputs
- … avoiding the main difficulty of reactive system design: having to cope with **all** possible environment inputs

Requirement engineering and scenarios

Formal spec and scenarios are complementary

- Scenarios are accepted in RE as an adequate tool to **elicit** requirements
- Scenarios are **easy** to produce: the designer controls **both** the inputs and the outputs
- … avoiding the main difficulty of reactive system design: having to cope with **all** possible environment inputs

- Scenarios (=examples=hints) as an alternative to **guide** the search for “good” solutions: the synthesis algorithm must now produce solutions **compatible** with the examples

Another (more appealing) approach

LTL spec + Hints (scenarios)

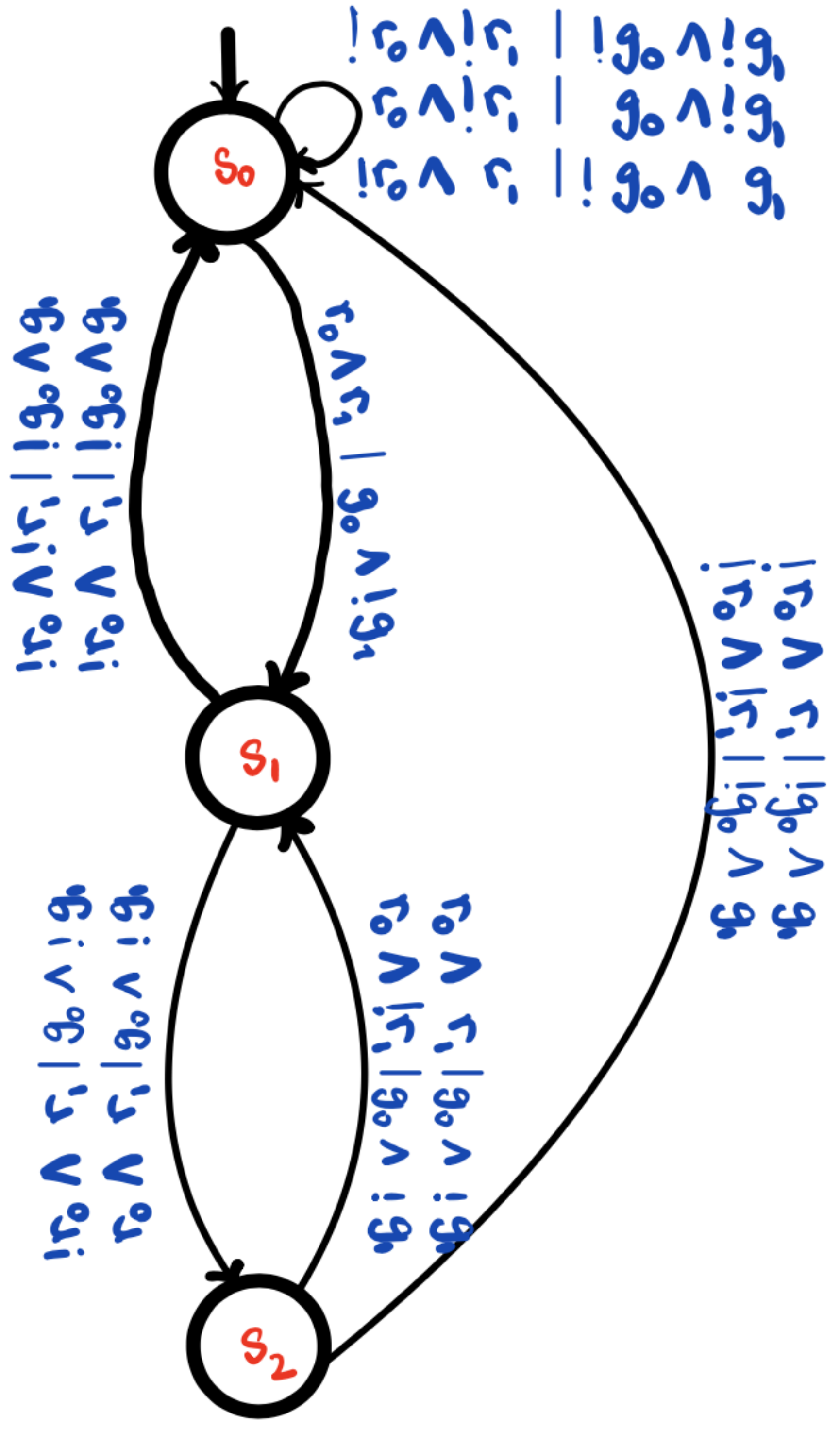
- $\varphi_{\text{CORE}} \equiv \Box(\neg g_0 \vee \neg g_1) \wedge \Box(r_0 \rightarrow \Diamond g_0) \wedge \Box(r_1 \rightarrow \Diamond g_1)$
- + a few **Hints**:
    - $\{!r_0, !r_1\} . \{!g_0, !g_1\} \# \{r_0, !r_1\} . \{g_0, !g_1\} \# \{!r_0, r_1\} . \{!g_0, g_1\}$
    - $\{r_0, r_1\} . \{g_0, !g_1\} \# \{!r_0, !r_1\} . \{!g_0, g_1\}$

Another (more appealing) approach

LTL spec + Hints (scenarios)

- $\varphi_{\text{CORE}} \equiv \Box(\neg g_0 \vee \neg g_1) \wedge \Box(r_0 \rightarrow \Diamond g_0) \wedge \Box(r_1 \rightarrow \Diamond g_1)$
- + a few **Hints**:
  - $\{!r_0, !r_1\} . \{!g_0, !g_1\} \# \{r_0, !r_1\} . \{g_0, !g_1\} \# \{!r_0, r_1\} . \{!g_0, g_1\}$
  - $\{r_0, r_1\} . \{g_0, !g_1\} \# \{!r_0, !r_1\} . \{!g_0, g_1\}$

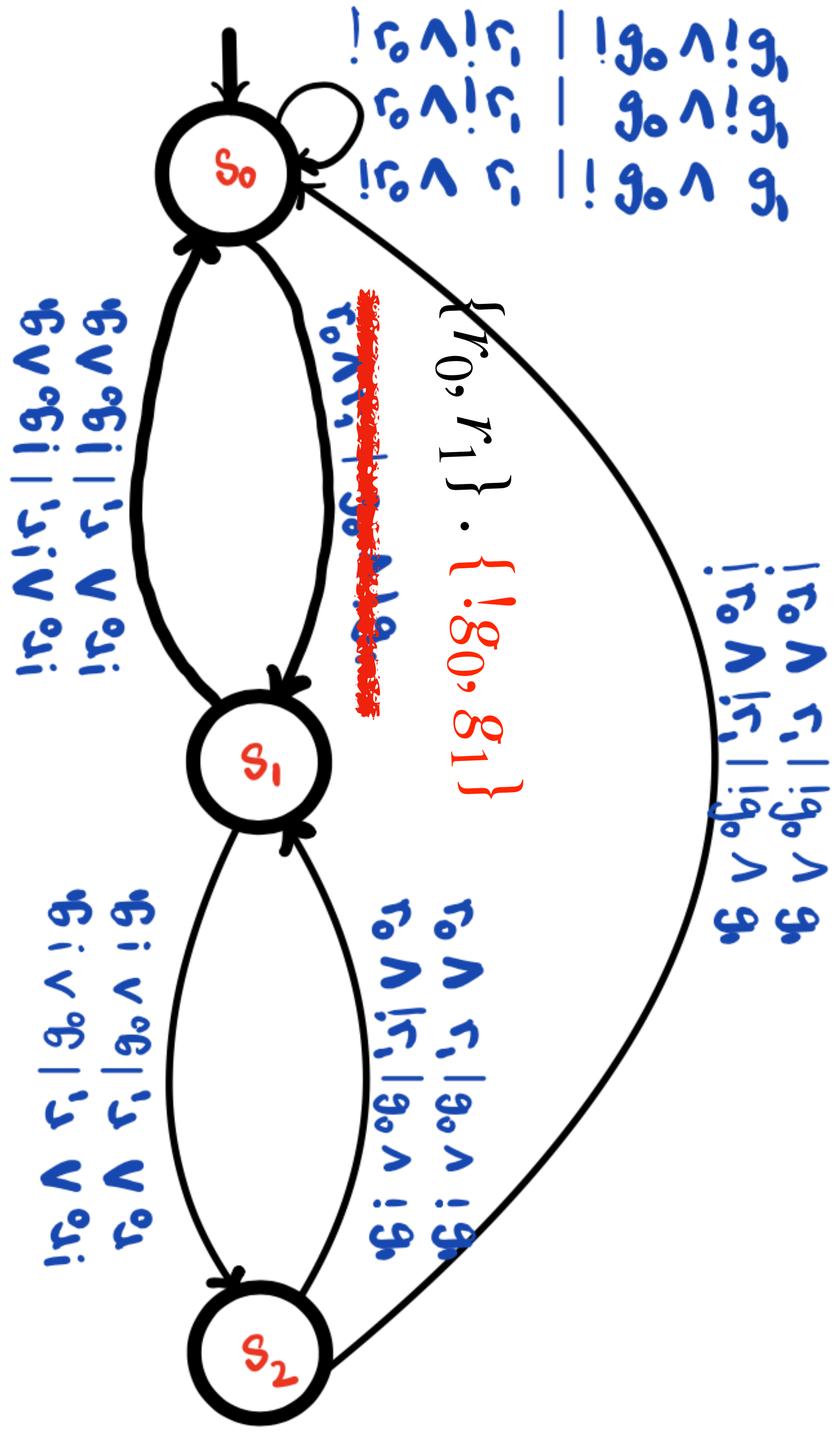
Learning+Synthesis



Another (more appealing) approach

LTL spec + Hints (scenarios)

- If you want a solution where g_1 comes before g_0 in case of concurrent request: **change the scenario** !
- **Hints**:
  - $\{!r_0, !r_1\} . \{!g_0, !g_1\}\#\{r_0, !r_1\} . \{g_0, !g_1\}\#\{!r_0, r_1\} . \{!g_0, g_1\}$
  - $\{r_0, r_1\} . \{!g_0, g_1\}\#\{!r_0, !r_1\} . \{g_0, !g_1\}$

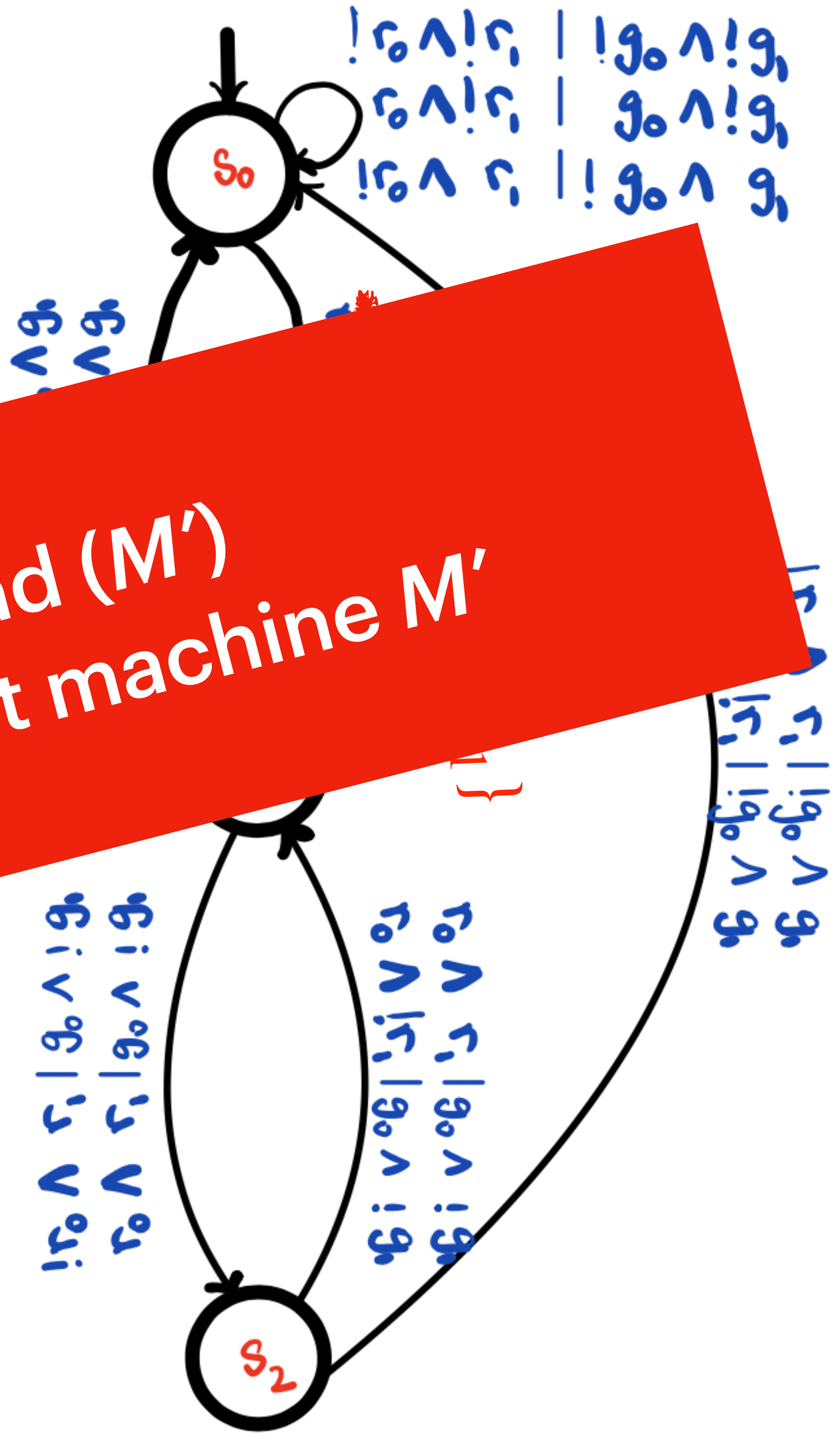


Another (more appealing) approach

LTL spec + Hints (scenarios)

- If you want a solution where $g_1$comes before g_0 in case of concurrent request: **change the scenario** !
- **Hints**:
  - $\{!r_0, !r_1\} \,.\, \{!g_0, !g_1\}\#\{r_0, !r_1\} \,.\, \{g_0, !g$
  - $\{r_0, r_1\} \,.\, \{\, !g_0, g$

If the solution M is not the one that User has in mind (M′)
then add an example that differentiates M and the target machine M′



$$(\varphi_{\mathsf{CORE}}, E_0) \longrightarrow M_0$$

$(\varphi_{\mathsf{CORE}}, E_0) \longrightarrow M_0$

If designer not happy

$(\varphi_{\mathsf{CORE}}, E_0 \cup \{e_1, e_2, \ldots, e_n\}) \longrightarrow M_1$

$(\varphi_{\mathsf{CORE}}, E_0) \longrightarrow M_0$

If designer not happy

$(\varphi_{\mathsf{CORE}}, E_0 \cup \{e_1, e_2, \ldots, e_n\}) \longrightarrow M_1$

If designer not happy

$(\varphi_{\mathsf{CORE}}, E_1 \cup \{e'_1, e'_2, \ldots, e'_m\}) \longrightarrow M_2$

...

Until designer is happy

LTL Reactive Synthesis with a Few Hints

The problem definition

- Given a (*i*) LTL formula φ and (*ii*) a prefix-closed set of examples (scenarios) $E \subseteq (I \cdot O)*$, construct a Mealy Machine M that is:
    - compatible with E and
    - such that for all $\forall \bar{I} = i_0 i_1 \dots i_n \dots \in I^\omega : M(\bar{I}) \models \varphi$

LTL Reactive Synthesis with a Few Hints

The problem definition

- Given a (*i*) LTL formula φ and (*ii*) a prefix-closed set of examples (scenarios) $E \subseteq (I \cdot O)^*$, construct a Mealy Machine M that is:
    - compatible with E and
    - such that for all $\forall \bar{I} = i_0 i_1 \ldots i_n \ldots \in I^\omega : M(\bar{I}) \models \varphi$

+ informal requirement: generalize E

Our solution - two-phase algorithm

Mix formal methods and learning

- Phase 1: **learn** a pre-Mealy machine that **generalizes** the examples in ***E*** and which **maintains realizability** (checked using game-based synthesis) of φ_{CORE}

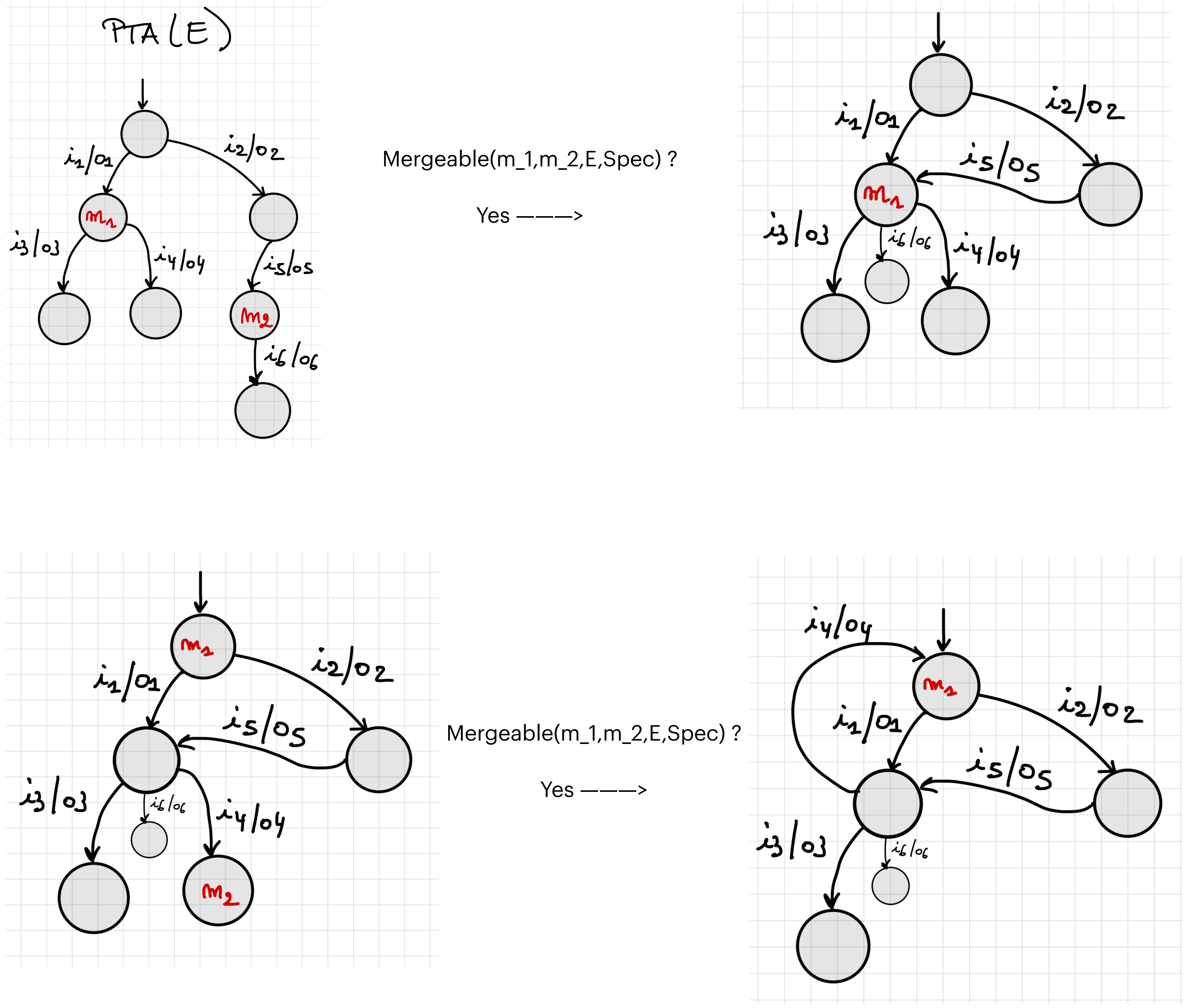
  intermediary output: a **pre**-Mealy machine (usually not input-complete)

- Phase 2: **complete** (using game-based synthesis) the pre-Mealy machine into a complete Mealy machine that realizes φ_{CORE} while maintaining compatibility with the examples in ***E***

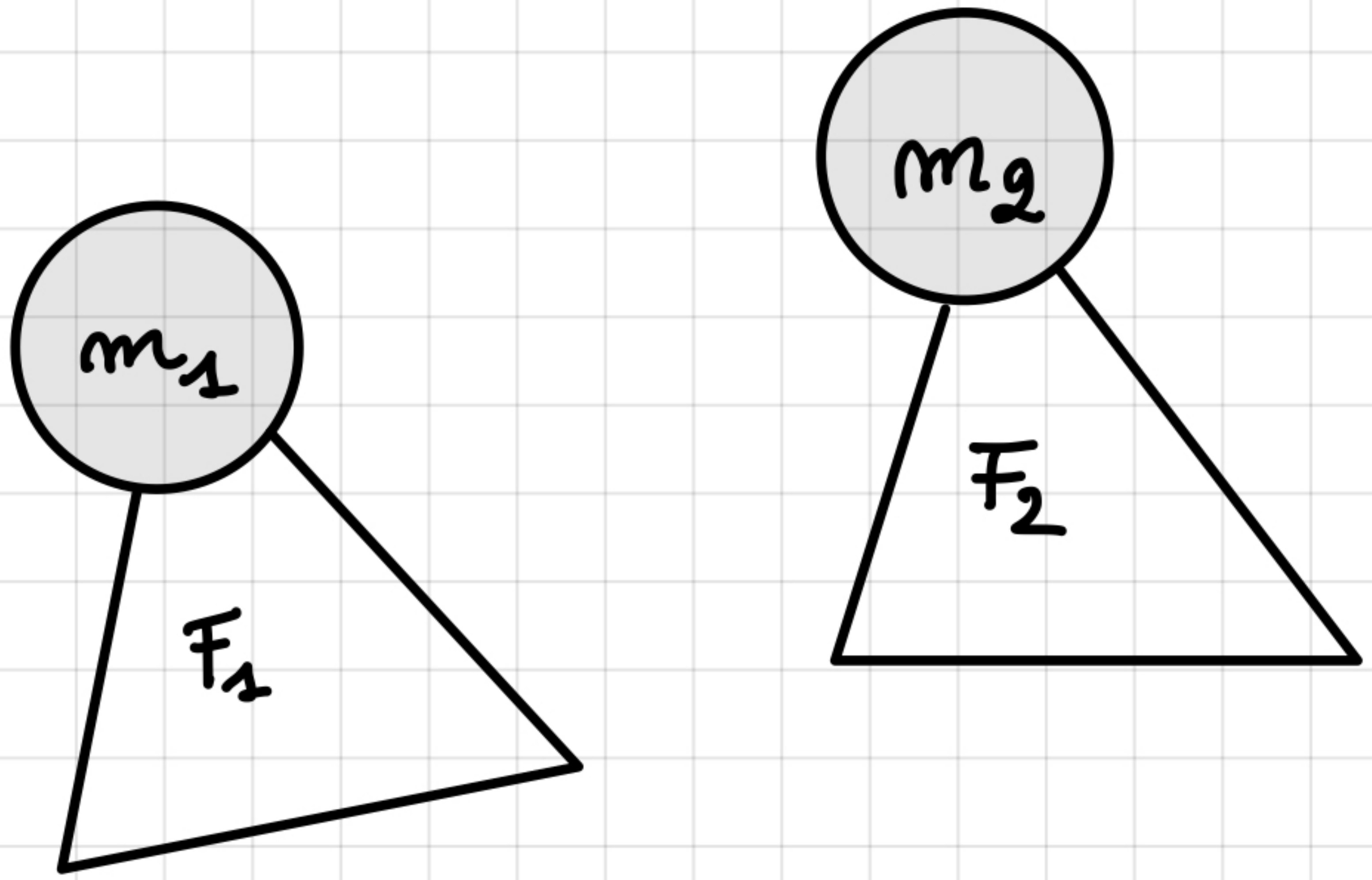
Phase 1 - Generalization

Learning automata from examples constrained by Spec realizability

- **RPNI** style learning:
  Start with PTA(E)=prefix tree automaton of the examples in ***E***
- **Merge** states when **possible** in order to **generalize** from the examples
- Mergeable?(m_1,m_2,***E***,φ_{CORE})
  - Yes, if the resulting pre-Mealy Machine is compatible with ***E*** and can be completed into a (full) Mealy Machine that realizes φ_{CORE}

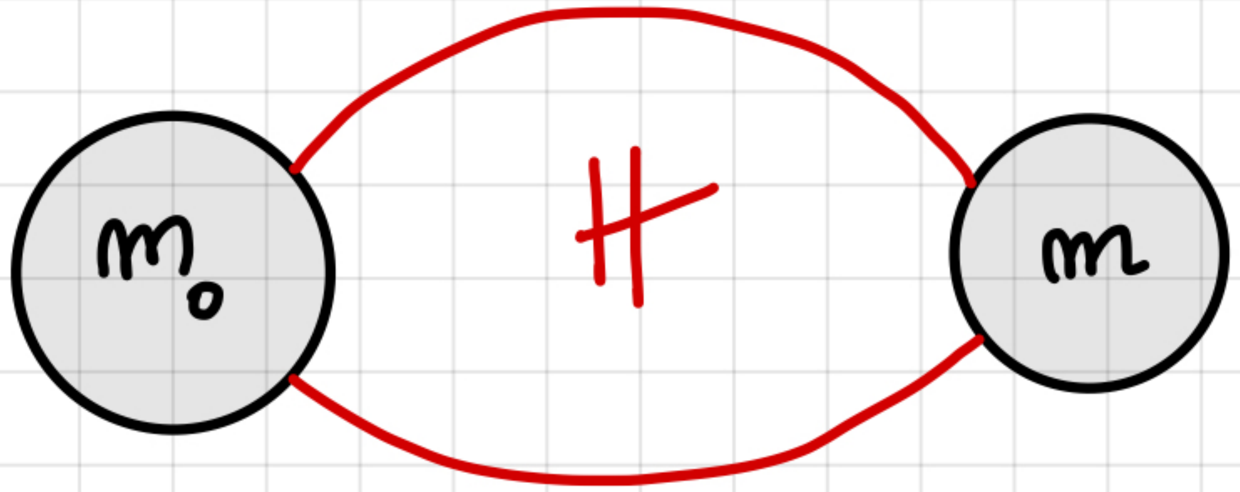


Mergeable?(m_1,m_2,*E*,Spec)



① F_1 and F_2 must agree on all $\bar{I} \in I^*$ for which F_1 and F_2 are defined

② $M \xrightarrow{\text{merge}(m_1, m_2)} M'$

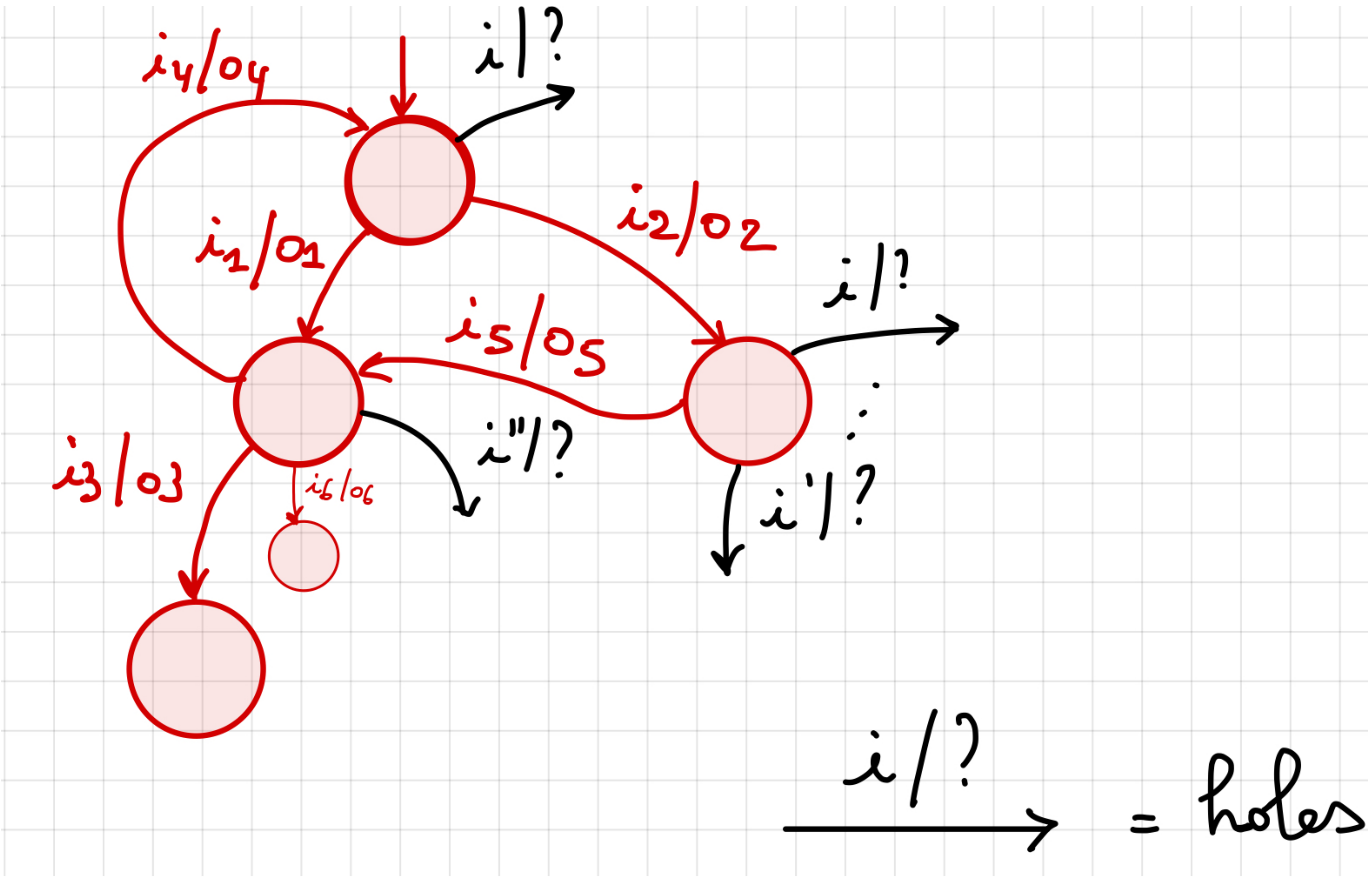


Check that from m, we can construct a strategy that works $\forall h \in H$!

Phase 2 - Completion

From pre-Mealy Machine to a (full) Mealy Machine that realizes Spec

- Given a pre Mealy machine ***M*** that generalizes the set of examples ***E*** and that can be completed into a Mealy machine ***M'*** that realizes φ



- Complete **holes** in the machine.
- Heuristics: try to avoid creating new states and **reuse** existing red ones (idea: **generalize** examples).

How to maintain efficiency

Exploit the most general strategy

- **Difficulty:**

  Theorem (**Mergeable complexity**): (Even) for a regular specification Spec φ_{CORE} given as a *deterministic Büchi automaton*, deciding Mergeable$(M, m, m', \varphi_{\text{CORE}}, E)$ is **ExpTime-C**. (a subset construction is needed)

How to maintain efficiency

Exploit the most general strategy

- **Difficulty:**

  Theorem (**Mergeable complexity**): (Even) for a regular specification Spec given as a *deterministic Büchi automaton*, deciding $\mathsf{Mergeable}(M, m, m', \varphi_{\mathsf{CORE}}, E)$ is **ExpTime-C**. (a subset construction is needed)

- … in the two-phase algorithm, we need to use $\mathsf{Mergeable}(M, m, m', \varphi_{\mathsf{CORE}}, E)$ multiple times, and in the worst-case the parity automaton A_φ associated to the LTL spec φ is already doubly exponential in $|\varphi|$.
- Can we avoid this complexity problem ? **YES**

How to maintain efficiency

Exploit the most general strategy

- **Theorem** Given (φ, E), SynthLearn(φ, E) returns a Mealy machine M such that $E \subseteq L(M)$ and $L_\omega(M) \subseteq [\![\varphi]\!]$ if it exists, in worst-case doubly exponential time in $|\varphi|$ and polynomial in $|E|$. Otherwise it returns UNREAL.

- More precisely, our algorithm is
  - polynomial in the size of E and
  - polynomial in a well-chosen symbolic representation the of set of Mealy machines that realize φ which is computed by Acacia-Bonzai for solving plain LTL synthesis for φ.
- So, generalizing from E comes at an additional **polynomial cost**.

Illustration

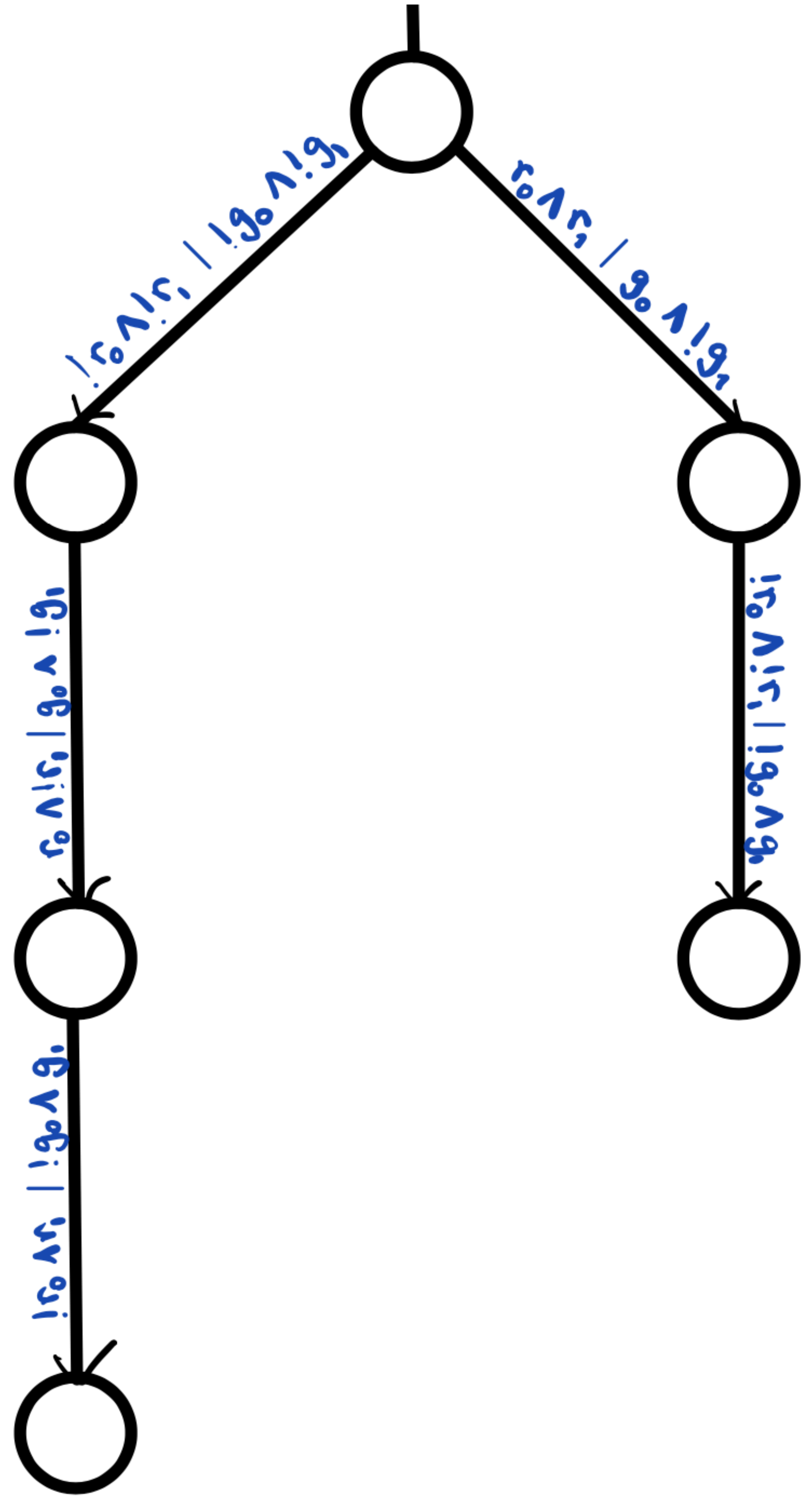
Phase 1: generalization of E

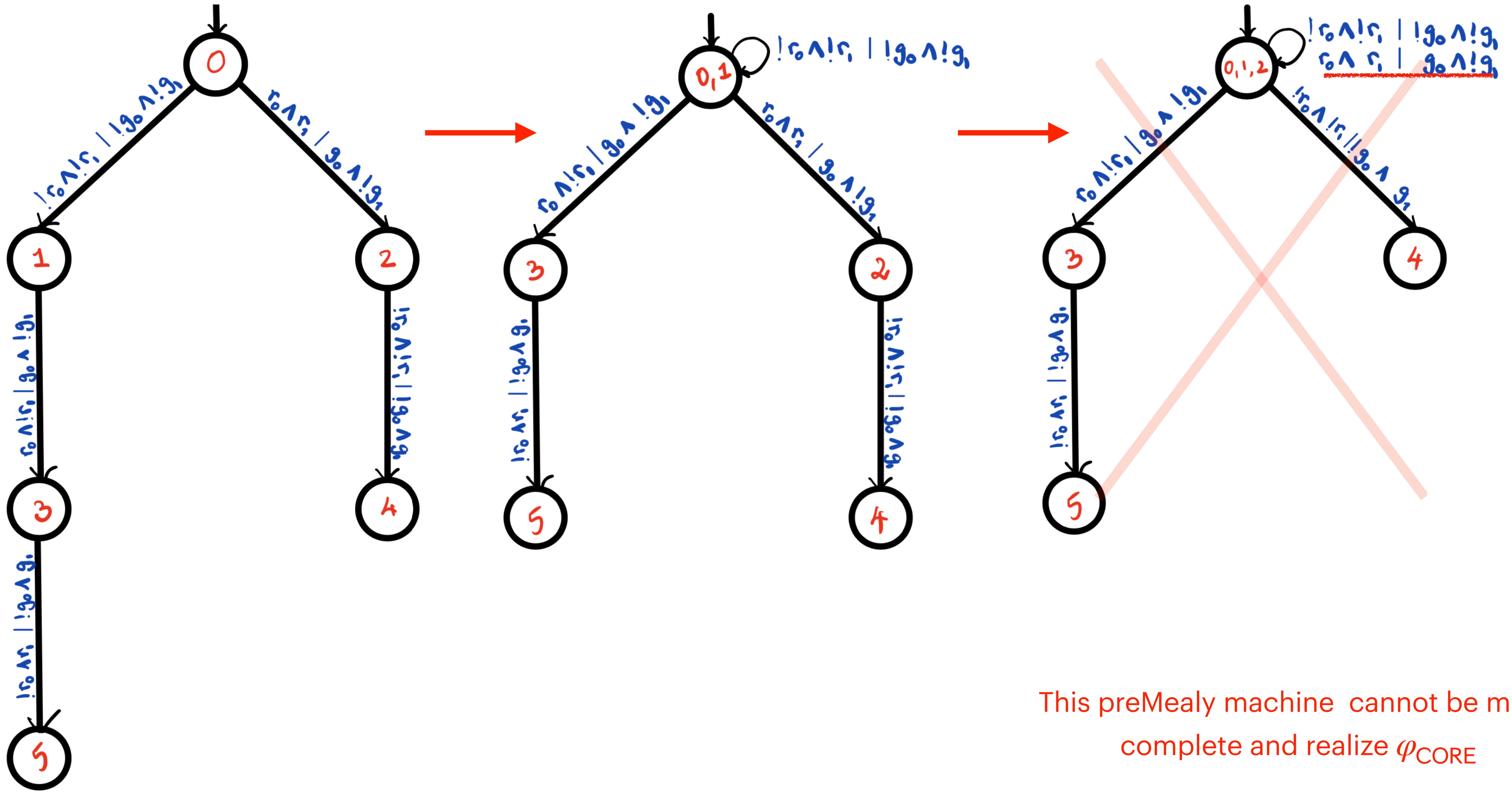
Specification:

$$\varphi_{\text{CORE}} \equiv \square(\neg g_0 \vee \neg g_1) \wedge \square(r_0 \rightarrow \lozenge g_0) \wedge \square(r_1 \rightarrow \lozenge g_1)$$

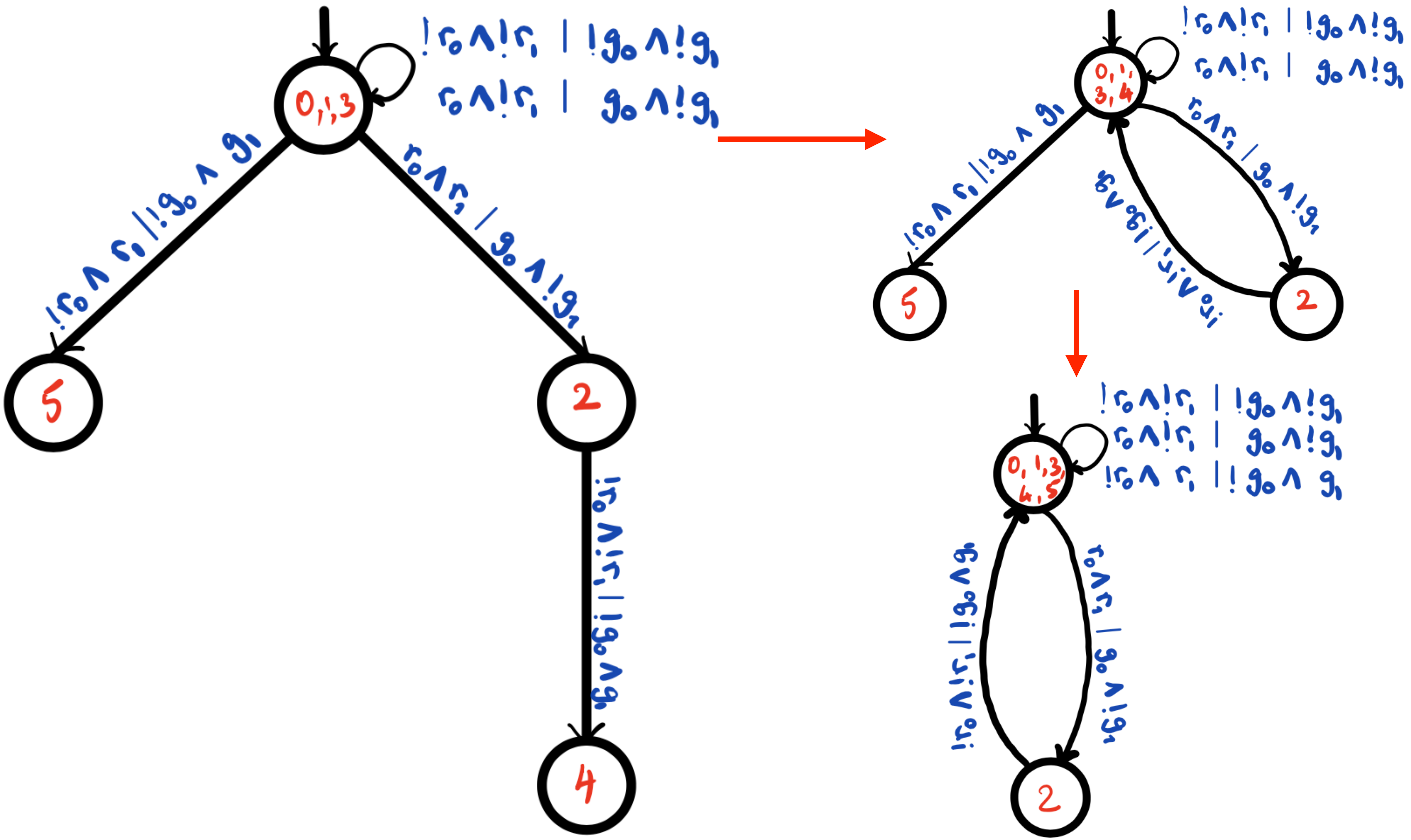
Examples E:

- $\{\,!r_0, !r_1\} \,.\, \{\,!g_0, !g_1\}\#\{r_0, !r_1\} \,.\, \{g_0, !g_1\}\#\{\,!r_0, r_1\} \,.\, \{\,!g_0, g_1\}$
- $\{r_0, r_1\} \,.\, \{g_0, !g_1\}\#\{\,!r_0, !r_1\} \,.\, \{\,!g_0, g_1\}$





This preMealy machine cannot be made complete and realize φ_{CORE}


0,1,3
!r0 ∧ !r1 | !g0 ∧ !g1
r0 ∧ !r1 | g0 ∧ !g1
5
2
4
0,1,3,4
2
0,1,3,4,5
!r0 ∧ r1 | !g0 ∧ g1

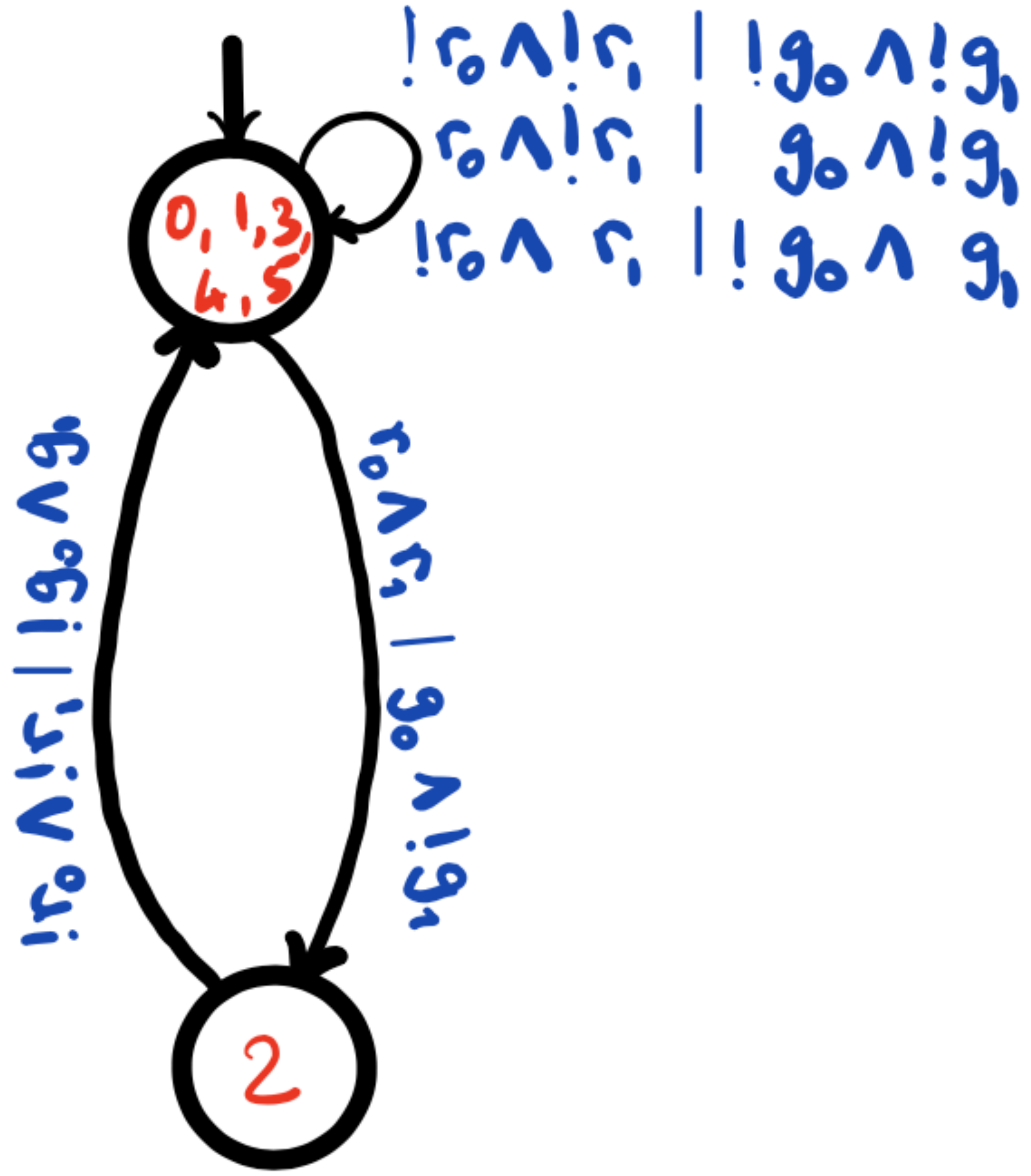
Phase 2: complete the preMealy machine

Specification:

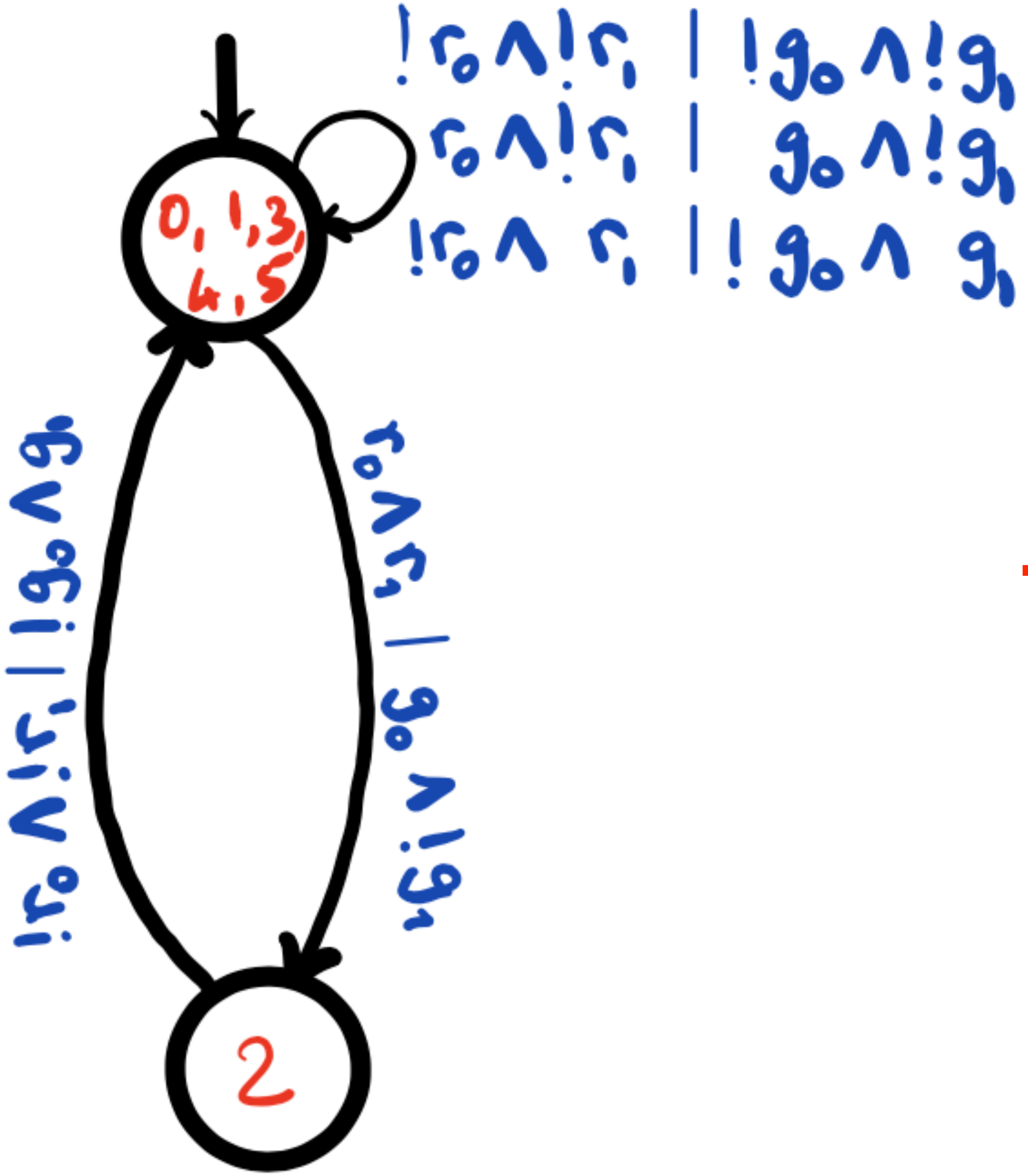
$\varphi_{\text{CORE}} \equiv \square(\neg g_0 \vee \neg g_1) \wedge \square(r_0 \rightarrow \Diamond g_0) \wedge \square(r_1 \rightarrow \Diamond g_1)$

Examples E:

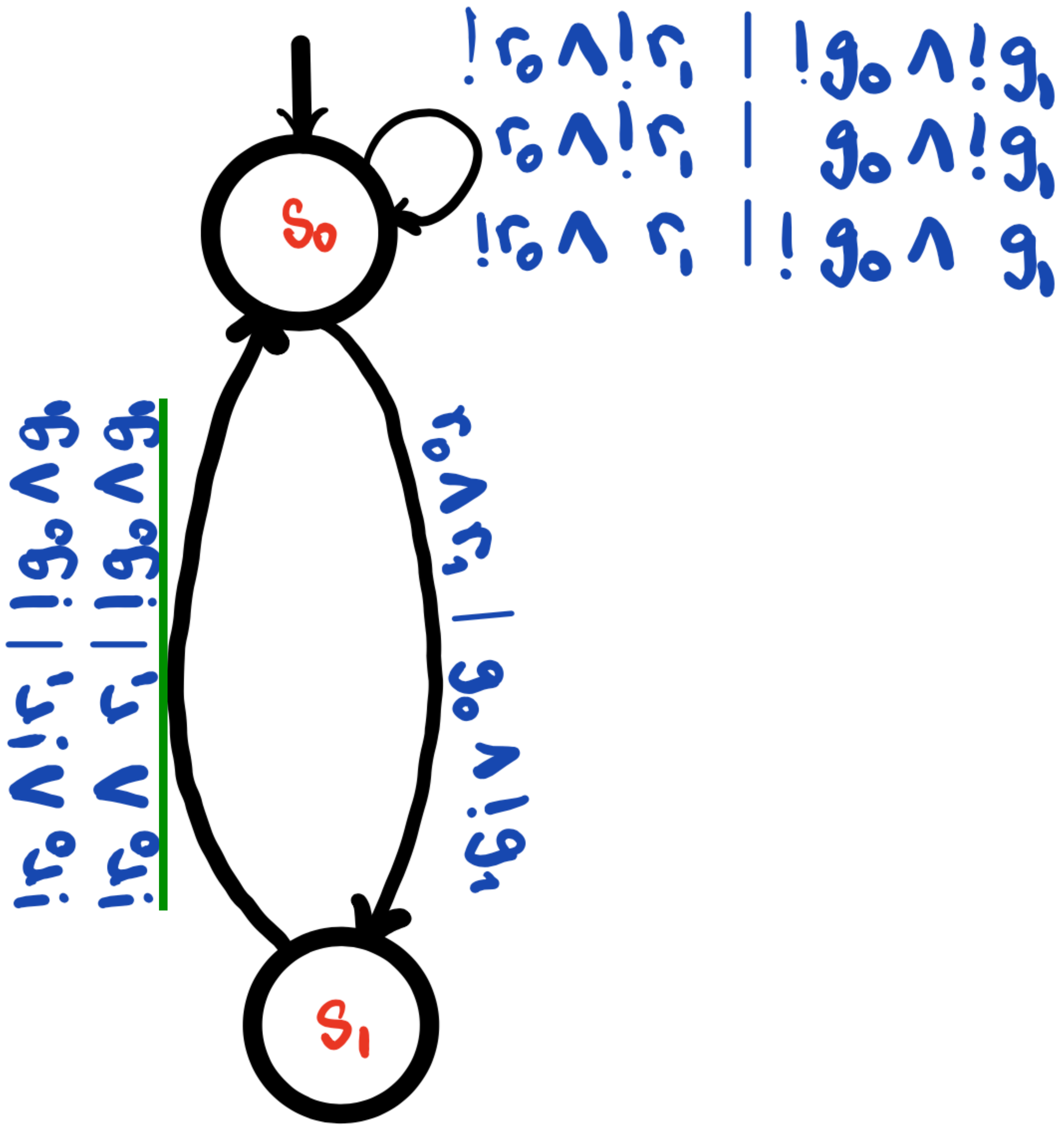
- $\{!r_0, !r_1\} . \{!g_0, !g_1\} \# \{r_0, !r_1\} . \{g_0, !g_1\} \# \{!r_0, r_1\} . \{!g_0, g_1\}$
- $\{r_0, r_1\} . \{g_0, !g_1\} \# \{!r_0, !r_1\} . \{!g_0, g_1\}$

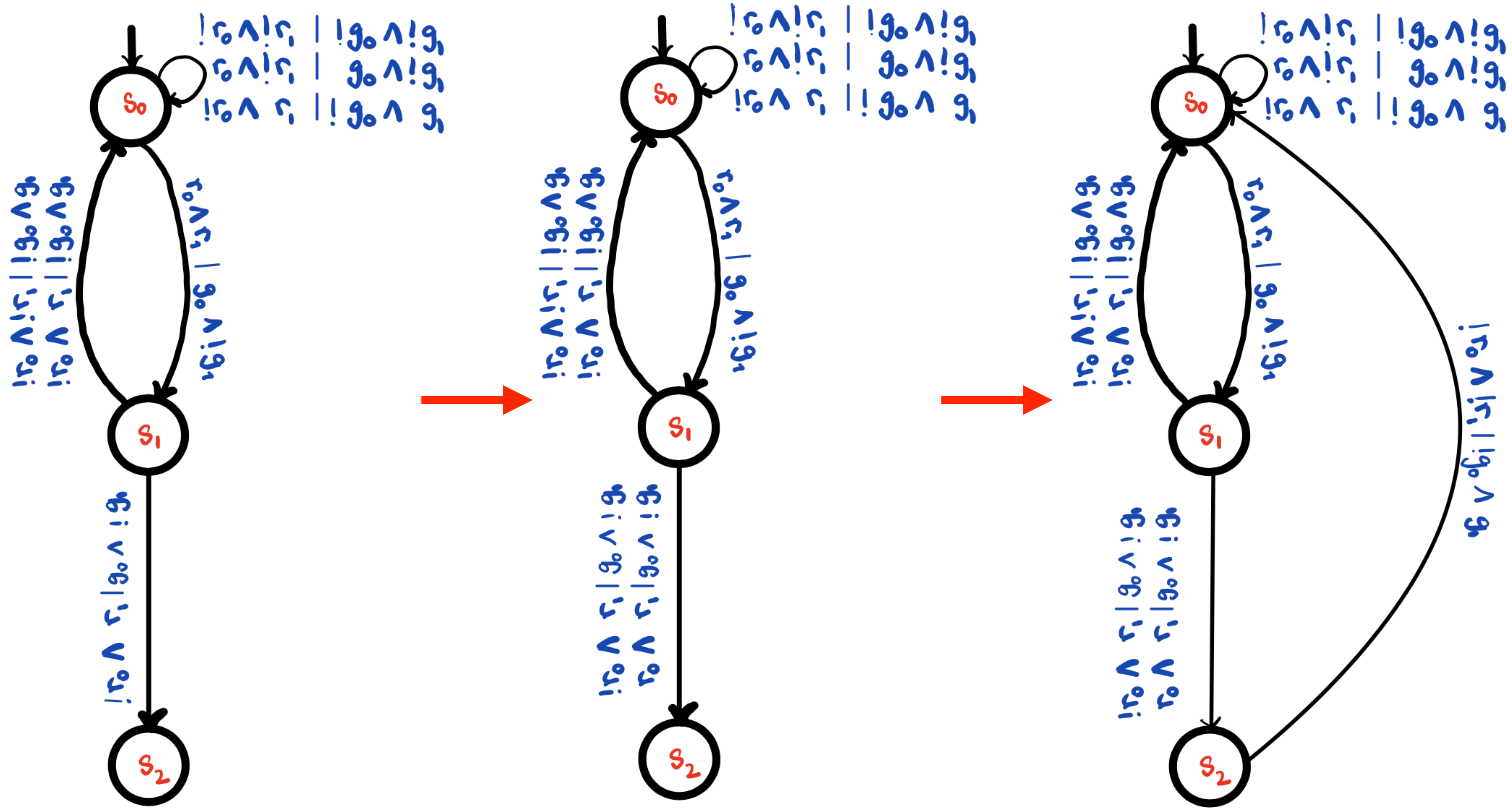


PreMealy machine obtained from the learning phase (generalization of the examples)

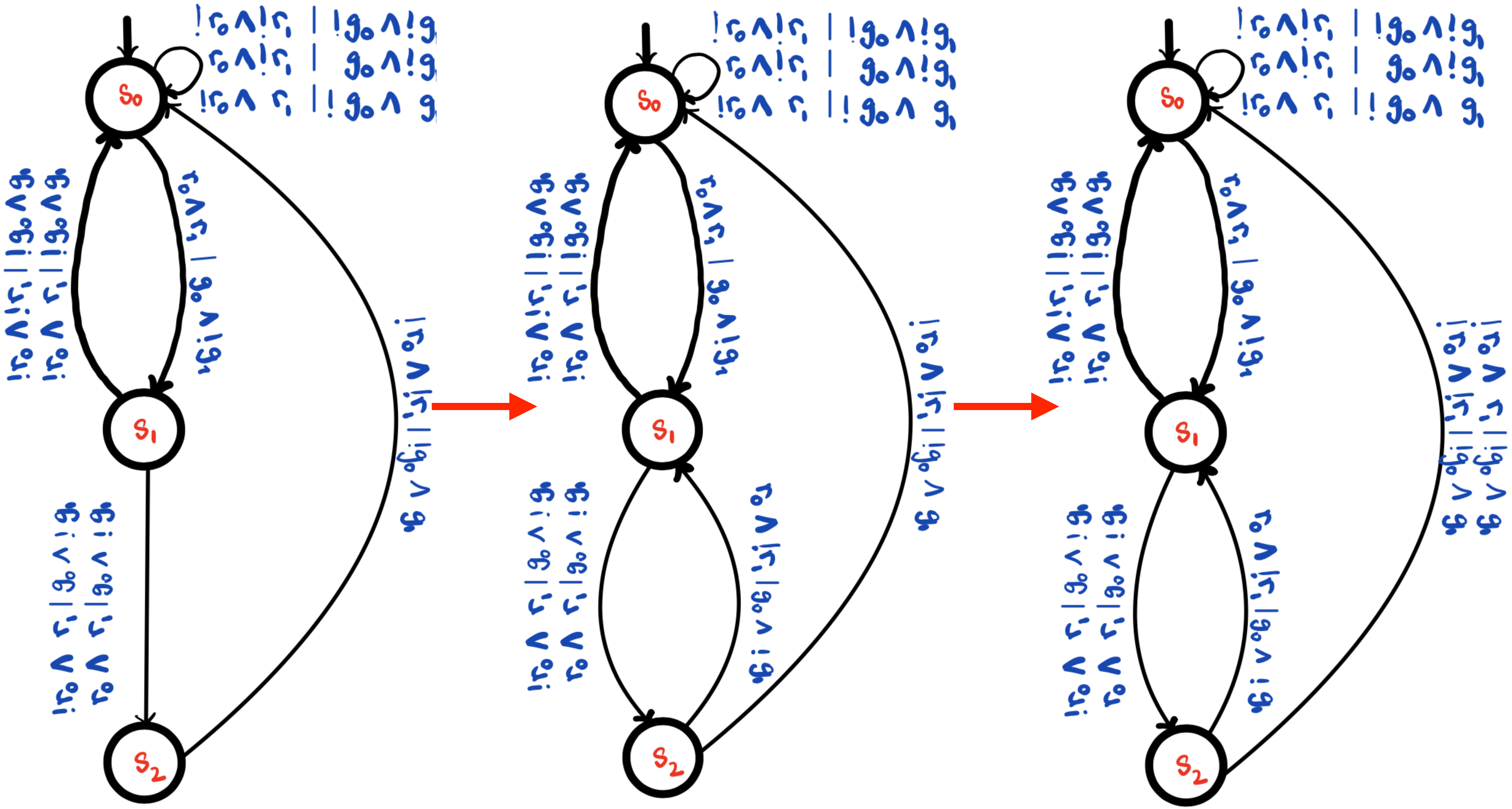


filling holes





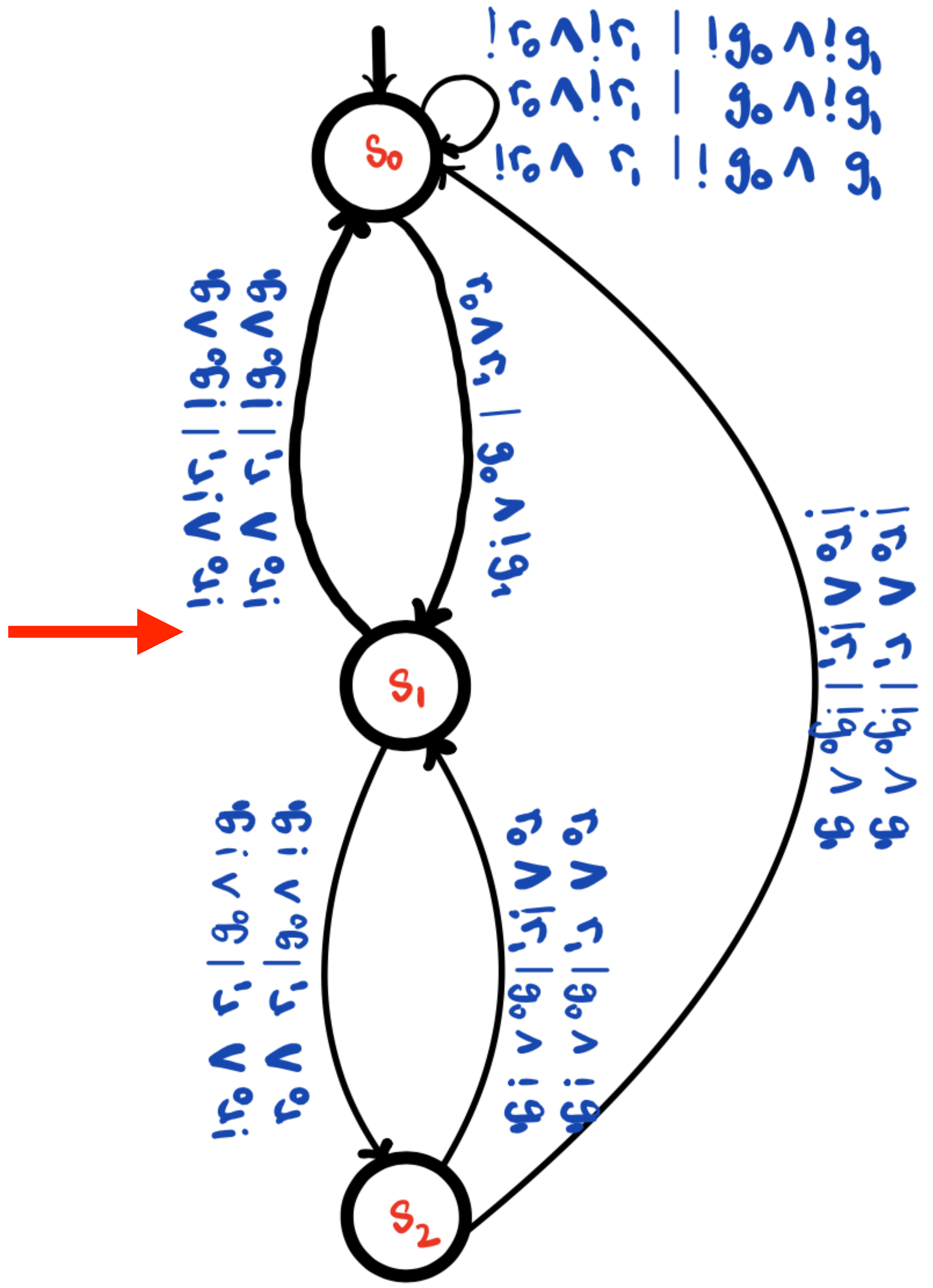
Heuristic: try to reuse as much as possible states that were introduced by the learning phase
(try to imitate decisions that are illustrated by the examples)



Heuristic: try to reuse as much as possible states that were introduced by the learning phase (try to imitate decisions that are illustrated by the examples)

Final (complete) Mealy machine that enforces

φ_{CORE}



A few theorems

- Theorem 1 (**Termination** and **correctness**): For all (φ, E), SynthLearn(φ, E) terminates and returns a Mealy machine M such that (i) $M \models \varphi$ and (ii) is compatible with *E*, if it exists, and return **UNREAL** otherwise.

- Theorem 2 (Mealy **completeness**): For all specifications φ, for all Mealy machines M such that $M \models \varphi$, there is a set of examples E_M of size **polynomial** in M and such that SynthLearn$(\varphi, E_M) = M$.

- Theorem 3 (**Polynomial** additional cost): our algorithm is polynomial in the size of E and in a symbolic representation the of set of Mealy machines that realize φ (which is computed by Acacia-Bonzai for solving plain LTL synthesis for φ.)

Further works

Easy and more ambitious

- Negative examples: hist $\cdot\ i/\neg o$
- Infinite examples given as deterministic i/o $\omega-$regular expressions, e.g. $\left(\{!r_1,!r_2\}.\{!g_1,!g_2\}\#\{r_1,!r_2\}.\{g_1,!g_2\}\#\{!r_1,r_2\}.\{!g_1,g_2\}\right)^{\omega}$
- Symbolic examples and symbolic Mealy machines: $\psi_0^i \cdot \psi_0^o \sharp \psi_1^i \cdot \psi_1^o \sharp \ldots \sharp \psi_n^i / o$
- How to learn **programs** manipulating variables, queues, or stacks that realize a spec φ_{CORE} with a similar approach ?