Lecture 4
Model Checking and Logic Synthesis

Nok Wongpiromsarn

Richard M. Murray

Ufuk Topcu

EECI, 14 May 2012

Outline

- Model checking: what it is, how it works, how it is used
- Computational complexity of model checking
- Closed system synthesis
- Examples using SPIN model checker

The basic idea behind model checking

Given:

- Transition system TS
- LTL formula Φ

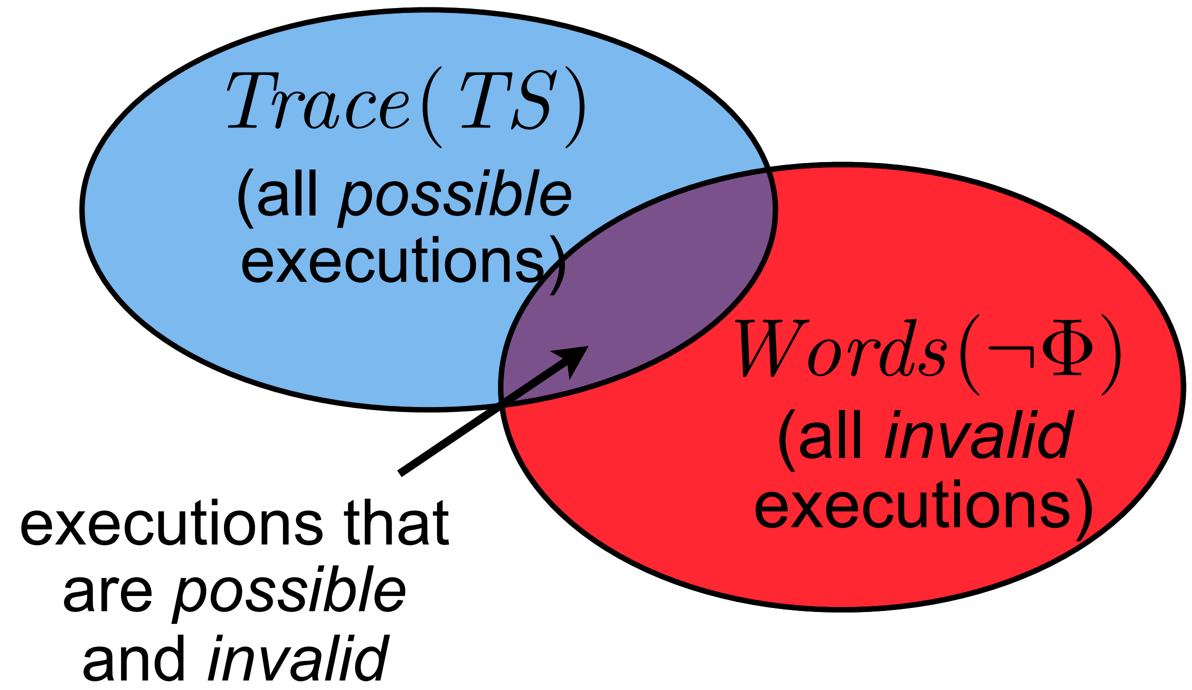
Question: Does TS satisfy Φ, i.e.,

$$TS \models \Phi \ ?$$

The basic idea behind model checking

Given:

- Transition system TS
- LTL formula Φ



Question: Does TS satisfy Φ, i.e.,

$$TS \models \Phi \text{ ?}$$

Answer (conceptual)**:**

$$TS \models \Phi$$ [*TS* satisfies Φ]

$$\Updownarrow$$

$$Trace(TS) \subseteq Words(\Phi)$$ [All executions of *TS* satisfy Φ]

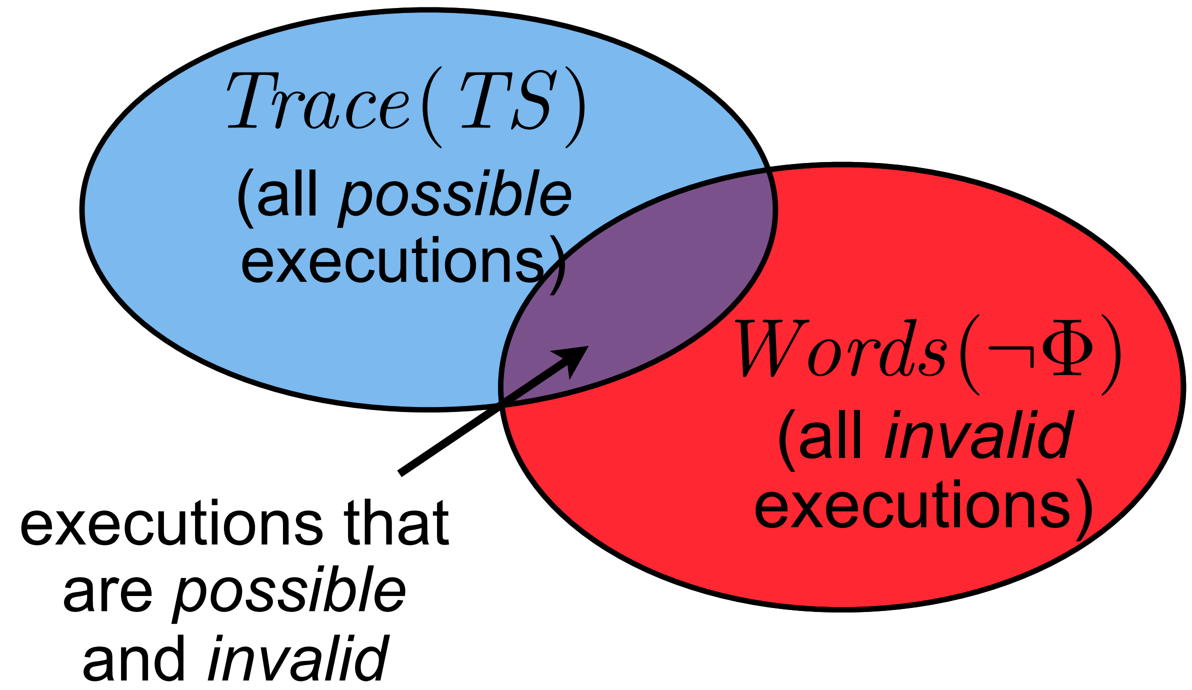
$$\Updownarrow$$

$$Trace(TS) \cap Words(\neg\Phi) = \emptyset$$ [No execution of *TS* violates Φ]

The basic idea behind model checking

Given:

- Transition system TS
- LTL formula Φ



Question: Does TS satisfy Φ, i.e.,

$$TS \models \Phi \text{ ?}$$

Answer (conceptual):

$TS \models \Phi$ [*TS* satisfies Φ]

$\Updownarrow$

$Trace(TS) \subseteq Words(\Phi)$ [All executions of *TS* satisfy Φ]

$\Updownarrow$

$Trace(TS) \cap Words(\neg\Phi) = \emptyset$ [No execution of *TS* violates Φ]

How to determine whether $Trace(TS) \cap Words(\neg\Phi) = \emptyset$ **?**

Preliminaries: LTL $\rightarrow$ Buchi automata

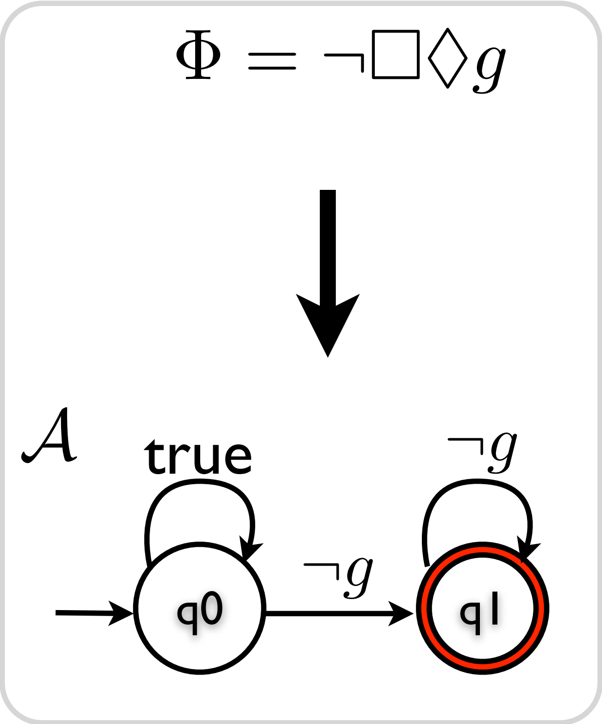
Theorem. *There exists an algorithm that takes an LTL formula* Φ *and returns a Büchi automaton* $\mathcal{A}$ *such that*

$$Words(\Phi) = \mathcal{L}_\omega(\mathcal{A})$$

Preliminaries: LTL $\rightarrow$ Buchi automata

Theorem. *There exists an algorithm that takes an LTL formula* Φ *and returns a Büchi automaton* $\mathcal{A}$ *such that*

$$Words(\Phi) = \mathcal{L}_\omega(\mathcal{A})$$

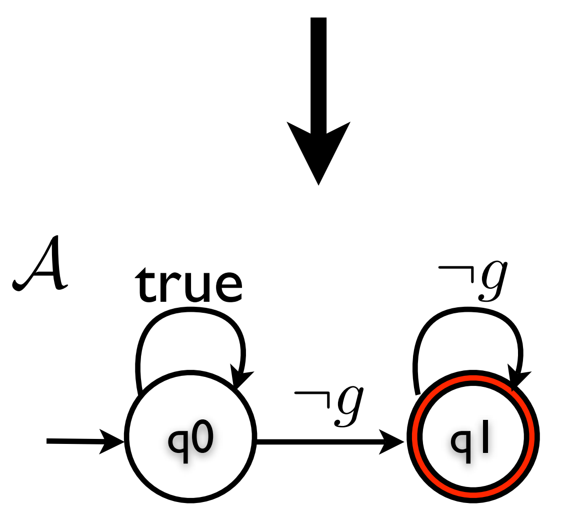


Preliminaries: LTL → Buchi automata

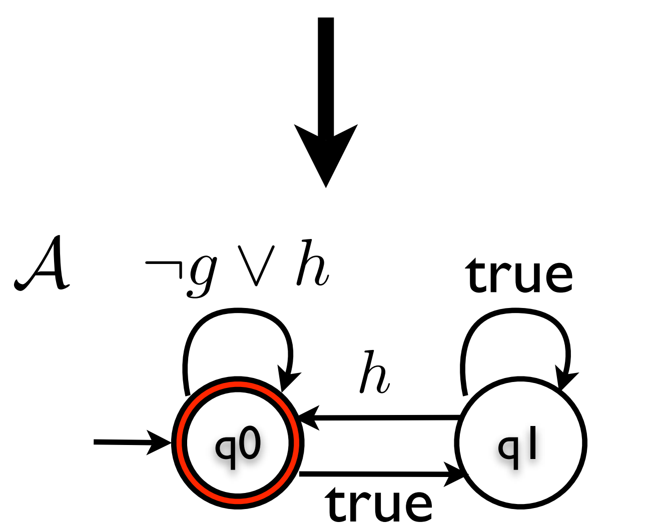
Theorem. *There exists an algorithm that takes an LTL formula* Φ *and returns a Büchi automaton* $\mathcal{A}$ *such that*

$$Words(\Phi) = \mathcal{L}_\omega(\mathcal{A})$$

$$\Phi = \neg\Box\Diamond g$$



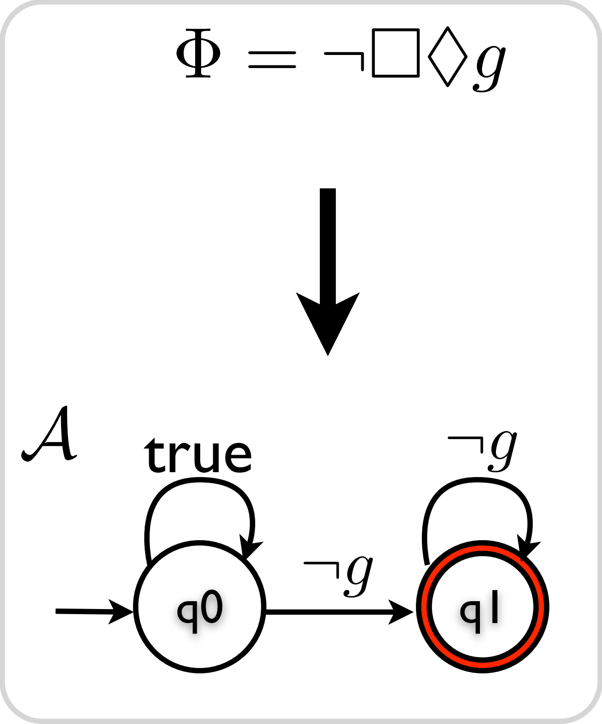
$$\Phi = \Box(g \to \Diamond h)$$

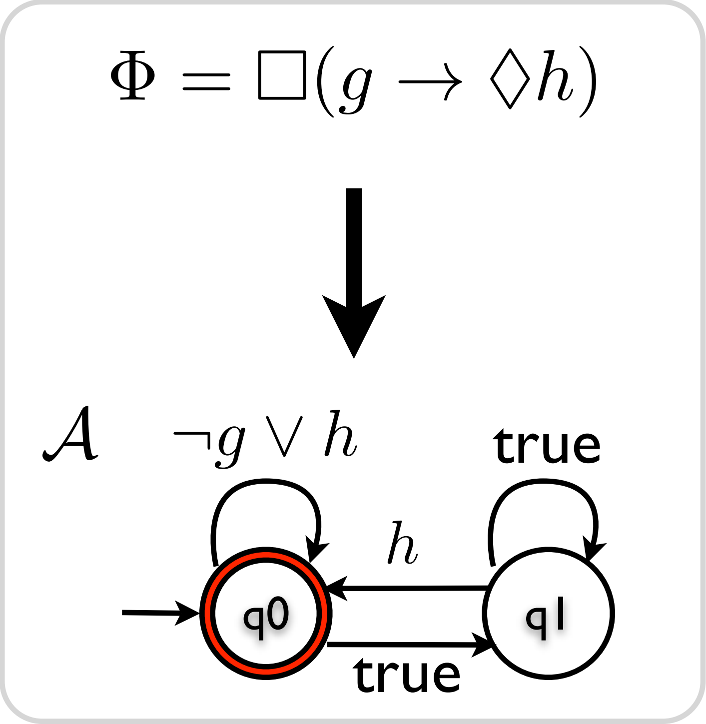


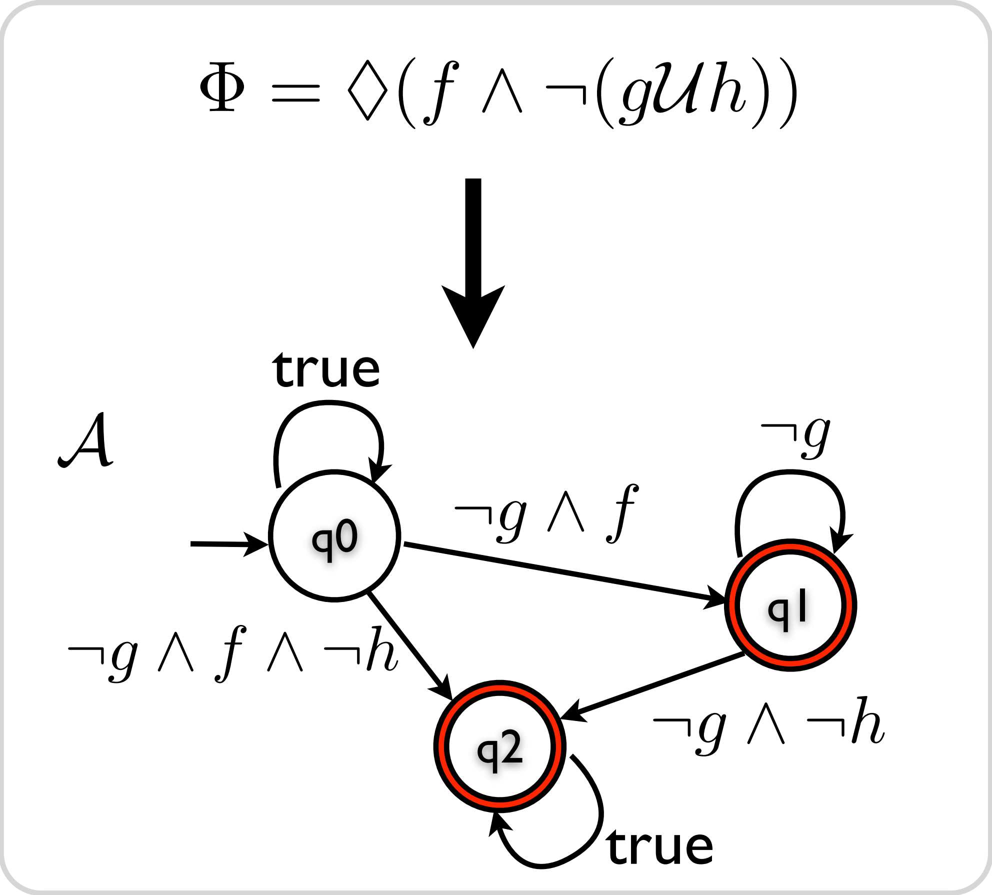
Preliminaries: LTL → Buchi automata

Theorem. *There exists an algorithm that takes an LTL formula* Φ *and returns a Büchi automaton* $\mathcal{A}$ *such that*

$$Words(\Phi) = \mathcal{L}_\omega(\mathcal{A})$$







A tool for constructing Buchi automata from LTL formulas: LTL2BA [http://www.lsv.ens-cachan.fr/~gastin/ltl2ba/index.php]

Preliminaries: transition system ⊗ Buchi automaton

Transition system:

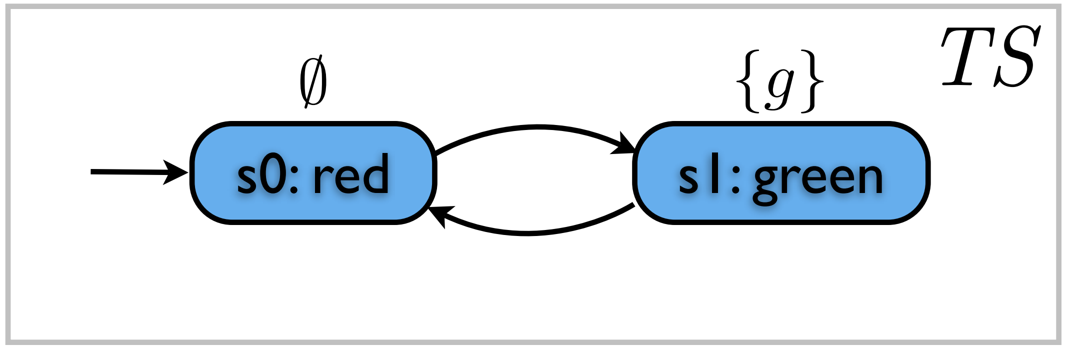
$$TS = (S, \mathrm{Act}, \rightarrow, I, \mathrm{AP}, L)$$

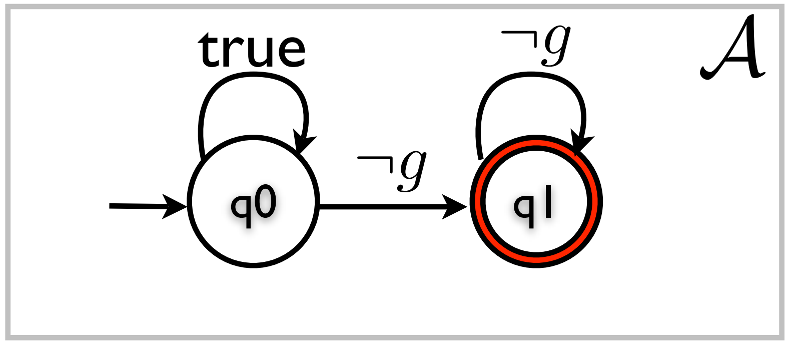
Nondeterministic Buchi automaton:

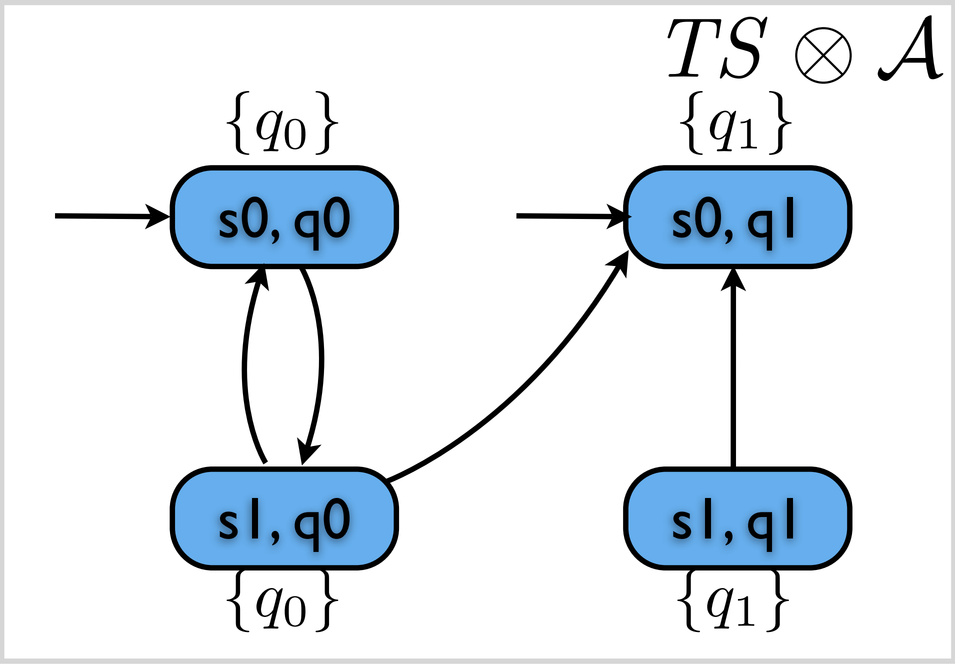
$$\mathcal{A} = (Q, 2^{\mathrm{AP}}, \delta, Q_0, F)$$

Define the product automaton: $TS \otimes \mathcal{A} = (S', \mathrm{Act}, \rightarrow', I', \mathrm{AP}', L')$, where

- $S' = S \times Q$
- $\forall s, t \in S, q, p \in Q$ with $s \xrightarrow{\alpha} t$ and $q \xrightarrow{L(t)} p$, there exists $\langle s, q \rangle \xrightarrow{\alpha}{}' \langle t, p \rangle$
- $I' = \{\langle s_0, q \rangle : s_0 \in I \text{ and } \exists q_0 \in Q_0 \text{ s.t. } q_0 \xrightarrow{L(s_0)} q\}$
- $AP' = Q$
- $L' : S \times Q \rightarrow 2^Q$ and $L'(\langle s, q \rangle) = \{q\}$







Preliminaries

Transition system: $TS = (S, \mathrm{Act}, \rightarrow, I, \mathrm{AP}, L)$

Nondeterministic Buchi automaton: $\mathcal{A} = (Q, 2^{\mathrm{AP}}, \delta, Q_0, F)$

Preliminaries

Transition system: $TS = (S, \mathrm{Act}, \rightarrow, I, \mathrm{AP}, L)$

Nondeterministic Buchi automaton: $\mathcal{A} = (Q, 2^{\mathrm{AP}}, \delta, Q_0, F)$

Theorem: $Trace(TS) \cap \mathcal{L}_\omega(\mathcal{A}) \neq \emptyset \quad \Leftrightarrow \quad TS \otimes \mathcal{A} \not\models \text{“eventually forever ”} \neg F$

Preliminaries

Transition system: $TS = (S, \mathrm{Act}, \rightarrow, I, \mathrm{AP}, L)$

Nondeterministic Buchi automaton: $\mathcal{A} = (Q, 2^{\mathrm{AP}}, \delta, Q_0, F)$

not in *F*

Theorem: $Trace(TS) \cap \mathcal{L}_\omega(\mathcal{A}) \neq \emptyset \quad \Leftrightarrow \quad TS \otimes \mathcal{A} \not\models \text{"eventually forever"} \neg F$

Preliminaries

Transition system: $TS = (S, \mathrm{Act}, \rightarrow, I, \mathrm{AP}, L)$

Nondeterministic Buchi automaton: $\mathcal{A} = (Q, 2^{\mathrm{AP}}, \delta, Q_0, F)$

not in *F*

Theorem: $Trace(TS) \cap \mathcal{L}_\omega(\mathcal{A}) \neq \emptyset \quad \Leftrightarrow \quad TS \otimes \mathcal{A} \not\models \text{"eventually forever "} \neg F$

Proof idea (⇐): Pick a path π' in $TS \otimes \mathcal{A}$ s.t. $\pi' \not\models \text{"eventually forever"} \neg F$, and let π be its projection to *TS*. Then,

- *trace*(π) ∈ *Trace*(*TS*) -- by definition of product
- *trace*(π) ∈ $\mathcal{L}_\omega(\mathcal{A})$ -- by hypothesis and by definition of product (*L*'(‹*s*,*q*›) = {*q*})

Preliminaries

Transition system: $TS = (S, \mathrm{Act}, \rightarrow, I, \mathrm{AP}, L)$

Nondeterministic Buchi automaton: $\mathcal{A} = (Q, 2^{\mathrm{AP}}, \delta, Q_0, F)$

not in *F*

Theorem: $Trace(TS) \cap \mathcal{L}_\omega(\mathcal{A}) \neq \emptyset \quad \Leftrightarrow \quad TS \otimes \mathcal{A} \not\models \text{"eventually forever"} \neg F$

Proof idea (⇐): Pick a path π' in $TS \otimes \mathcal{A}$ s.t. $\pi' \not\models \text{"eventually forever"} \neg F$, and let π be its projection to *TS*. Then,

- *trace*(π) ∈ *Trace*(*TS*) -- by definition of product
- *trace*(π) ∈ $\mathcal{L}_\omega(\mathcal{A})$ -- by hypothesis and by definition of product (*L*'(‹*s*,*q*›) = {*q*})

$TS \otimes \mathcal{A} \quad \not\models \text{"eventually forever"} \neg F$

$\Updownarrow$

There exists a state *x* in $TS \otimes \mathcal{A}$

- *x* is reachable
- *L'*(*x*) ⊆ *F*
- *x* is on a directed cycle

} graph search, e.g., (nested) depth-first search

Preliminaries

Transition system: $TS = (S, \mathrm{Act}, \rightarrow, I, \mathrm{AP}, L)$

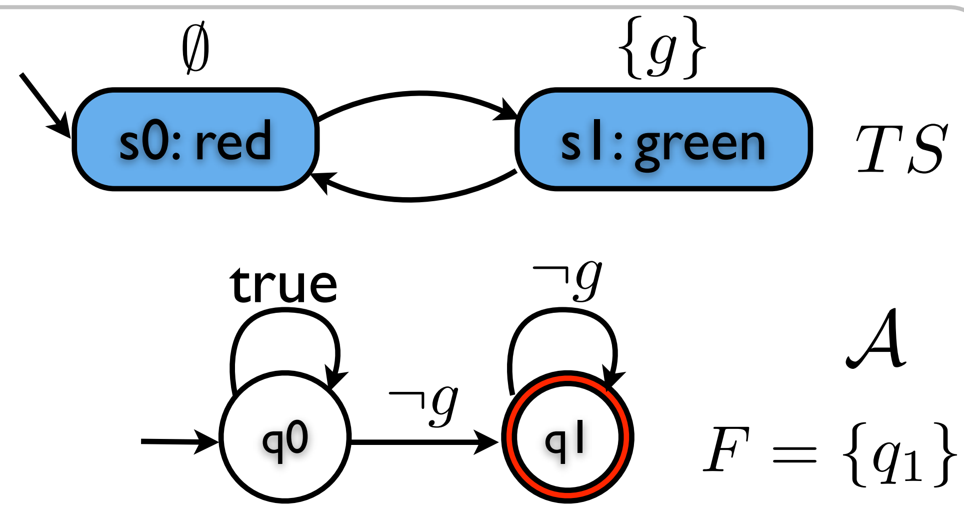
Nondeterministic Buchi automaton: $\mathcal{A} = (Q, 2^{\mathrm{AP}}, \delta, Q_0, F)$

not in *F*

Theorem: $Trace(TS) \cap \mathcal{L}_\omega(\mathcal{A}) \neq \emptyset \quad \Leftrightarrow \quad TS \otimes \mathcal{A} \not\models$ "eventually forever" $\neg F$

Proof idea (⇐): Pick a path π' in $TS \otimes \mathcal{A}$ s.t. $\pi' \not\models$ "eventually forever" $\neg F$, and let π be its projection to *TS*. Then,

- *trace*(π) ∈ *Trace*(*TS*) -- by definition of product
- *trace*(π) ∈ $\mathcal{L}_\omega(\mathcal{A})$ -- by hypothesis and by definition of product (*L*'(‹*s*,*q*›) = {*q*})



$TS \otimes \mathcal{A} \quad \not\models$ "eventually forever" $\neg F$

$\Updownarrow$

There exists a state *x* in $TS \otimes \mathcal{A}$

- *x* is reachable
- *L'(x)* ⊆*F*
- *x* is on a directed cycle

} graph search, e.g., (nested) depth-first search

Preliminaries

Transition system: $TS = (S, \mathrm{Act}, \rightarrow, I, \mathrm{AP}, L)$

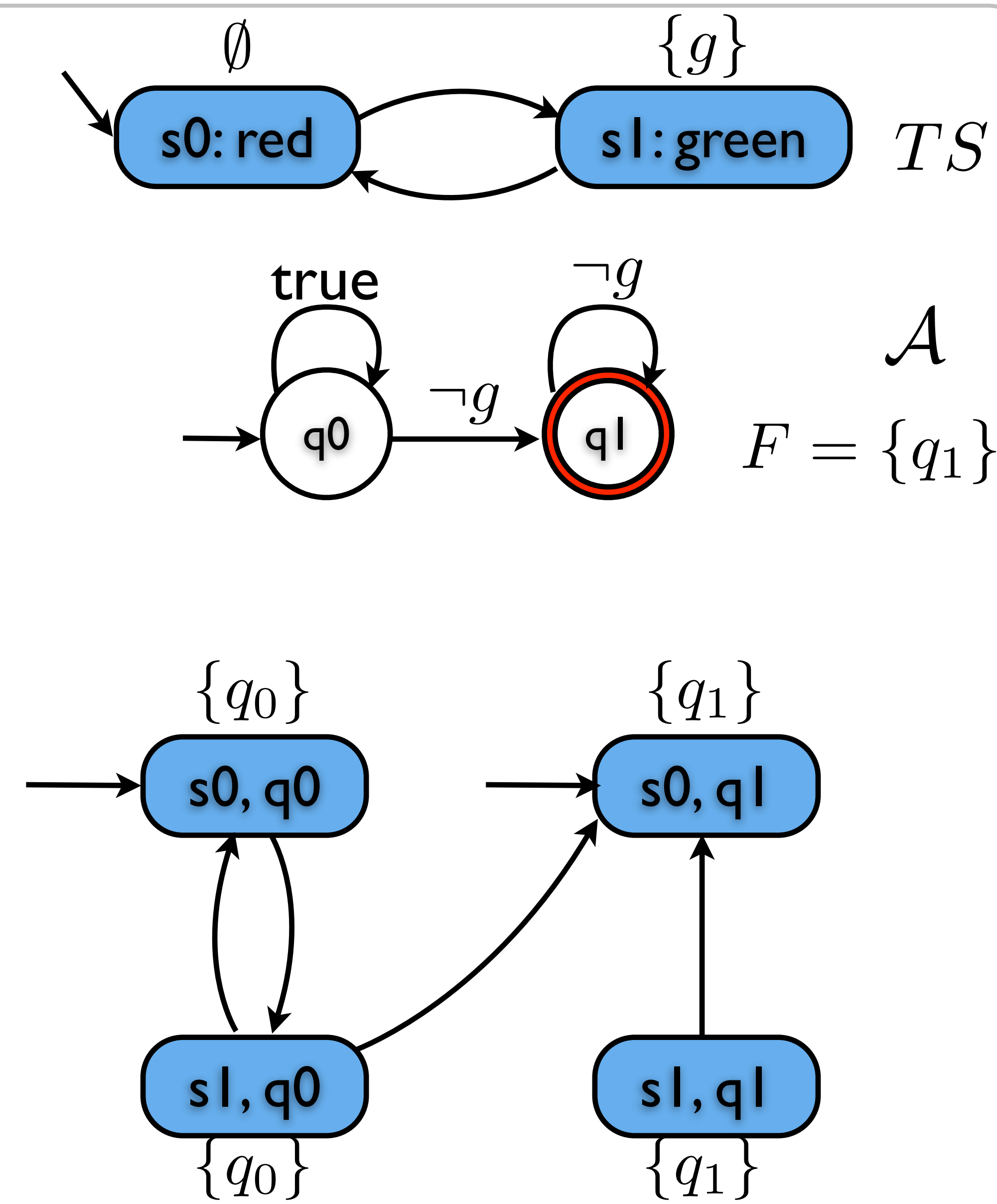
Nondeterministic Buchi automaton: $\mathcal{A} = (Q, 2^{\mathrm{AP}}, \delta, Q_0, F)$

not in *F*

Theorem: $Trace(TS) \cap \mathcal{L}_\omega(\mathcal{A}) \neq \emptyset \quad \Leftrightarrow \quad TS \otimes \mathcal{A} \not\models \text{"eventually forever"} \neg F$

Proof idea (⇐): Pick a path π' in $TS \otimes \mathcal{A}$ s.t. $\pi' \not\models \text{"eventually forever"} \neg F$, and let π be its projection to *TS*. Then,

- *trace*(π) ∈ *Trace*(*TS*) -- by definition of product
- *trace*(π) ∈ $\mathcal{L}_\omega(\mathcal{A})$ -- by hypothesis and by definition of product (*L*'(‹*s*,*q*›) = {*q*})



$TS \otimes \mathcal{A} \quad \not\models \text{"eventually forever"} \neg F$

⇕

There exists a state *x* in $TS \otimes \mathcal{A}$

- *x* is reachable
- *L'(x)* ⊆ *F*
- *x* is on a directed cycle

} graph search, e.g., (nested) depth-first search

Preliminaries

Transition system: $TS = (S, \mathrm{Act}, \rightarrow, I, \mathrm{AP}, L)$

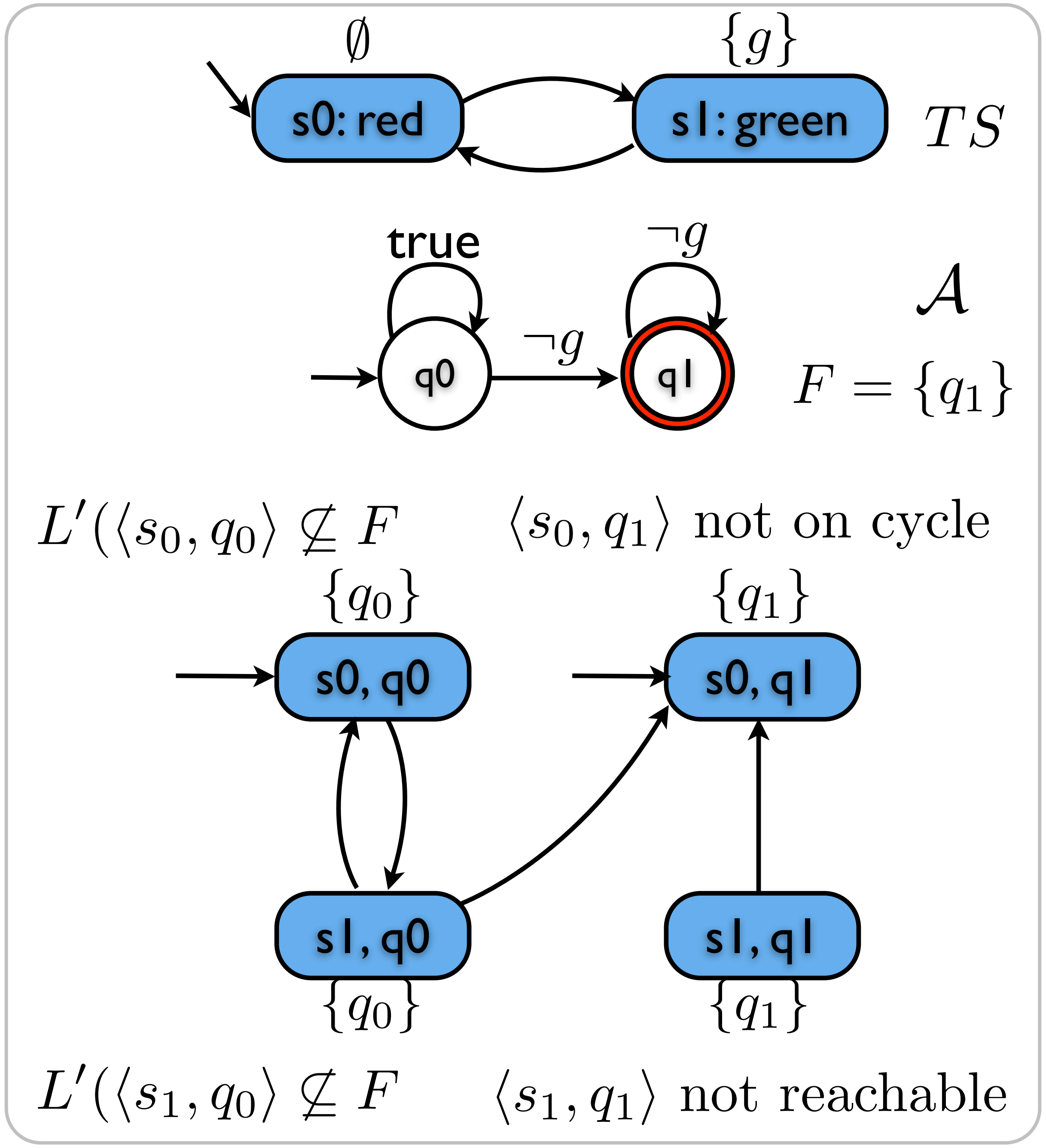
Nondeterministic Buchi automaton: $\mathcal{A} = (Q, 2^{\mathrm{AP}}, \delta, Q_0, F)$

not in *F*

Theorem: $Trace(TS) \cap \mathcal{L}_\omega(\mathcal{A}) \neq \emptyset \quad \Leftrightarrow \quad TS \otimes \mathcal{A} \not\models$ "eventually forever" $\neg F$

Proof idea (⇐): Pick a path π‘ in $TS \otimes \mathcal{A}$ s.t. $\pi' \not\models$ "eventually forever" $\neg F$, and let π be its projection to *TS*. Then,

- *trace*(π) ∈ *Trace*(*TS*) -- by definition of product
- *trace*(π) ∈ $\mathcal{L}_\omega(\mathcal{A})$ -- by hypothesis and by definition of product (*L*’(‹*s*,*q*›) = {*q*})



$TS \otimes \mathcal{A} \quad \not\models$ "eventually forever" $\neg F$

$\Updownarrow$

There exists a state *x* in $TS \otimes \mathcal{A}$

- *x* is reachable
- *L’(x)* ⊆*F*
- *x* is on a directed cycle

} graph search, e.g., (nested) depth-first search

Putting together

Given:

- Transition system TS
- LTL formula Φ
- NBA $\mathcal{A}_{\neg\Phi}$ accepting $\neg\Phi$ with the set *F* of accepting states

$$TS \not\models \Phi$$

Putting together

Given:

- Transition system TS
- LTL formula Φ
- NBA $\mathcal{A}_{\neg\Phi}$ accepting $\neg\Phi$ with the set *F* of accepting states

$$TS \not\models \Phi$$

$$\Updownarrow$$

$$Trace(TS) \not\subseteq Words(\Phi)$$

Putting together

Given:

- Transition system TS
- LTL formula Φ
- NBA $\mathcal{A}_{\neg\Phi}$ accepting $\neg\Phi$ with the set *F* of accepting states

$$TS \not\models \Phi$$

$$\Updownarrow$$

$$Trace(TS) \not\subseteq Words(\Phi)$$

$$\Updownarrow$$

$$Trace(TS) \cap Words(\neg\Phi) \neq \emptyset$$

Putting together

Given:

- Transition system TS
- LTL formula Φ
- NBA $\mathcal{A}_{\neg\Phi}$ accepting $\neg\Phi$ with the set *F* of accepting states

$$TS \not\models \Phi$$

$$\Updownarrow$$

$$Trace(TS) \not\subseteq Words(\Phi)$$

$$\Updownarrow$$

$$Trace(TS) \cap Words(\neg\Phi) \neq \emptyset$$

$$\Updownarrow$$

$$Trace(TS) \cap \mathcal{L}_\omega(\mathcal{A}_{\neg\Phi}) \neq \emptyset$$

Putting together

Given:

- Transition system TS
- LTL formula Φ
- NBA $\mathcal{A}_{\neg\Phi}$ accepting $\neg\Phi$ with the set *F* of accepting states

$$TS \not\models \Phi$$

$$\Updownarrow$$

$$Trace(TS) \not\subseteq Words(\Phi)$$

$$\Updownarrow$$

$$Trace(TS) \cap Words(\neg\Phi) \neq \emptyset$$

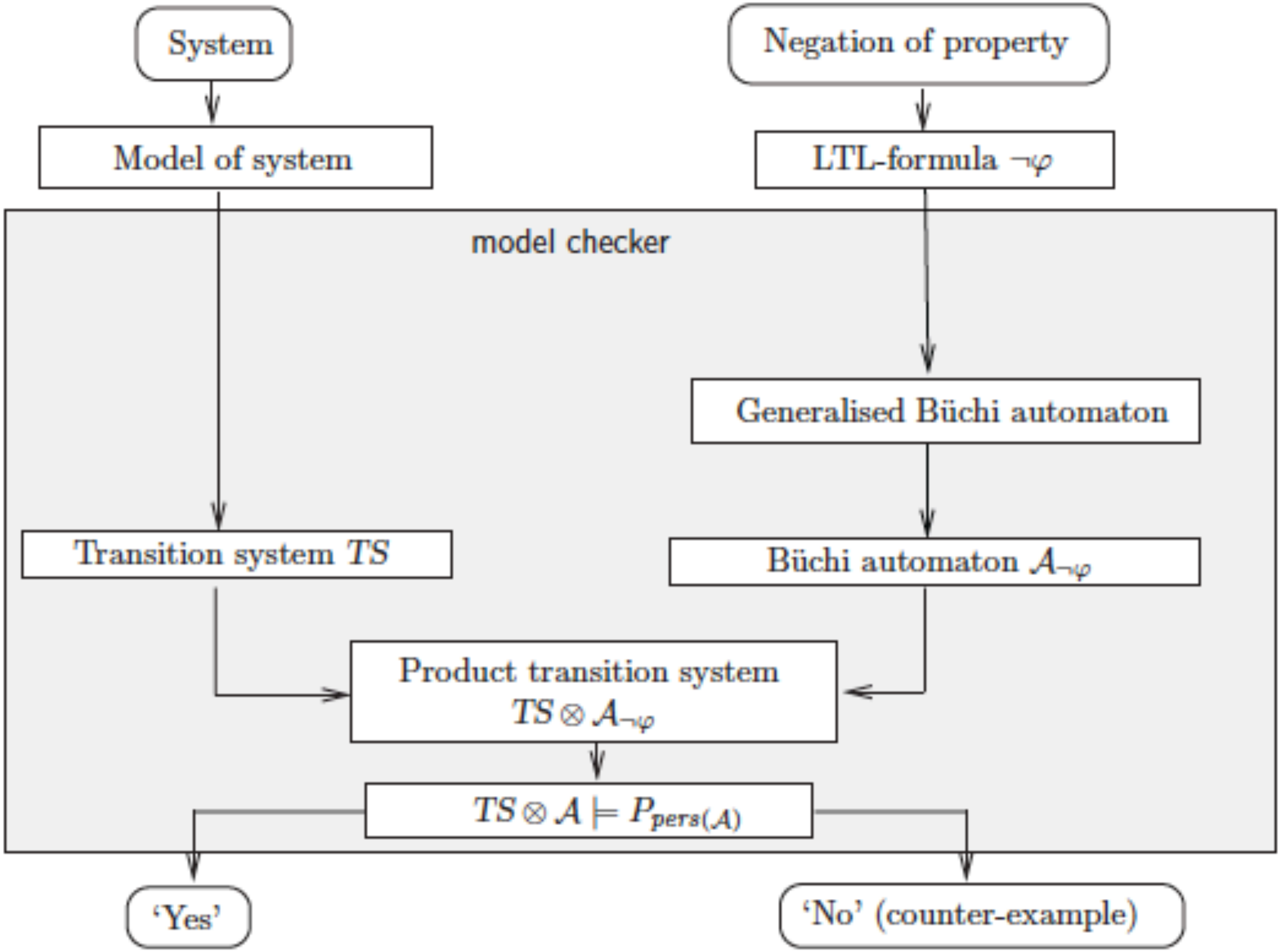
$$\Updownarrow$$

$$Trace(TS) \cap \mathcal{L}_\omega(\mathcal{A}_{\neg\Phi}) \neq \emptyset$$

$$\Updownarrow$$

$$TS \otimes \mathcal{A}_{\neg\Phi} \not\models \text{“eventually forever”} \neg F$$

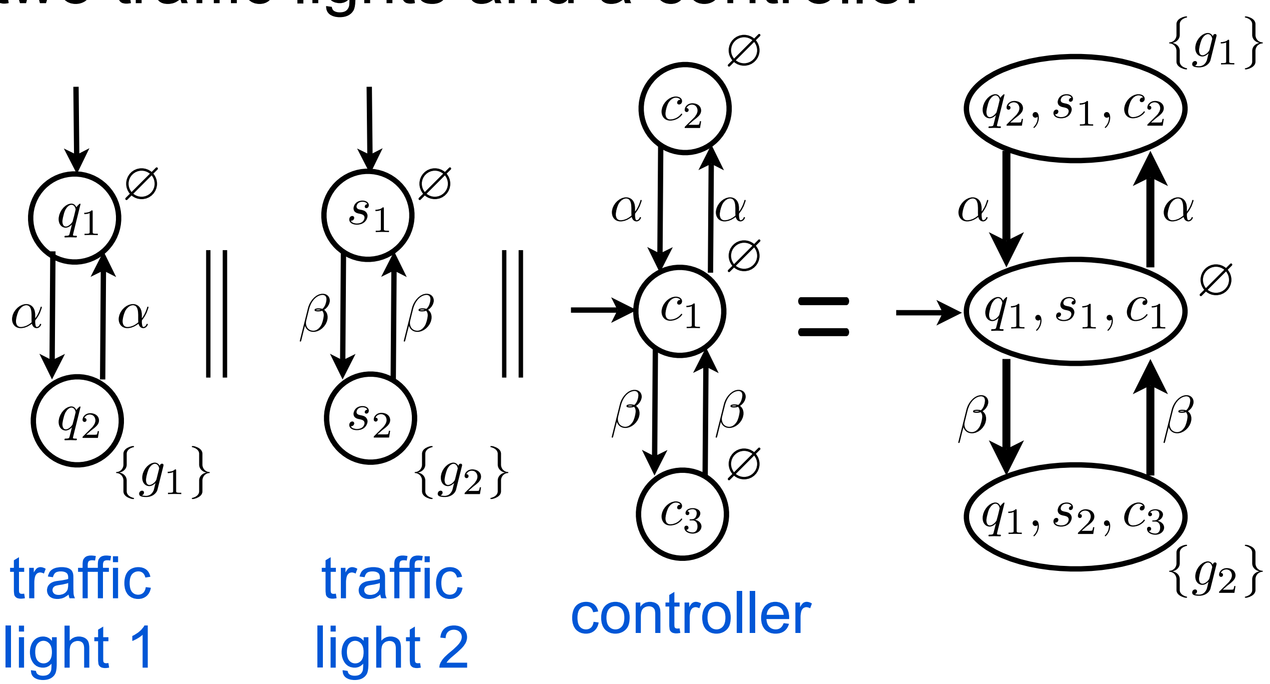
The process flow of model checking



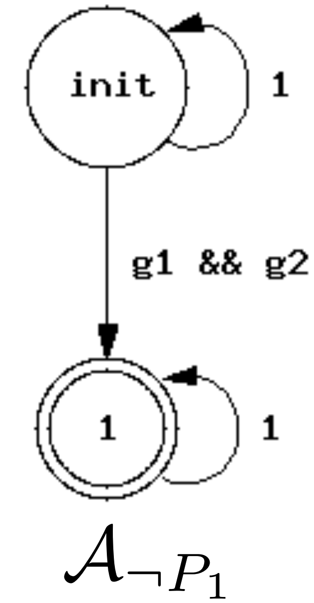
Efficient model checking tools automate the process: SPIN, nuSMV, TLC,...

Example 1: traffic lights (property verified)

System *TS*: synchronous composition of two traffic lights and a controller

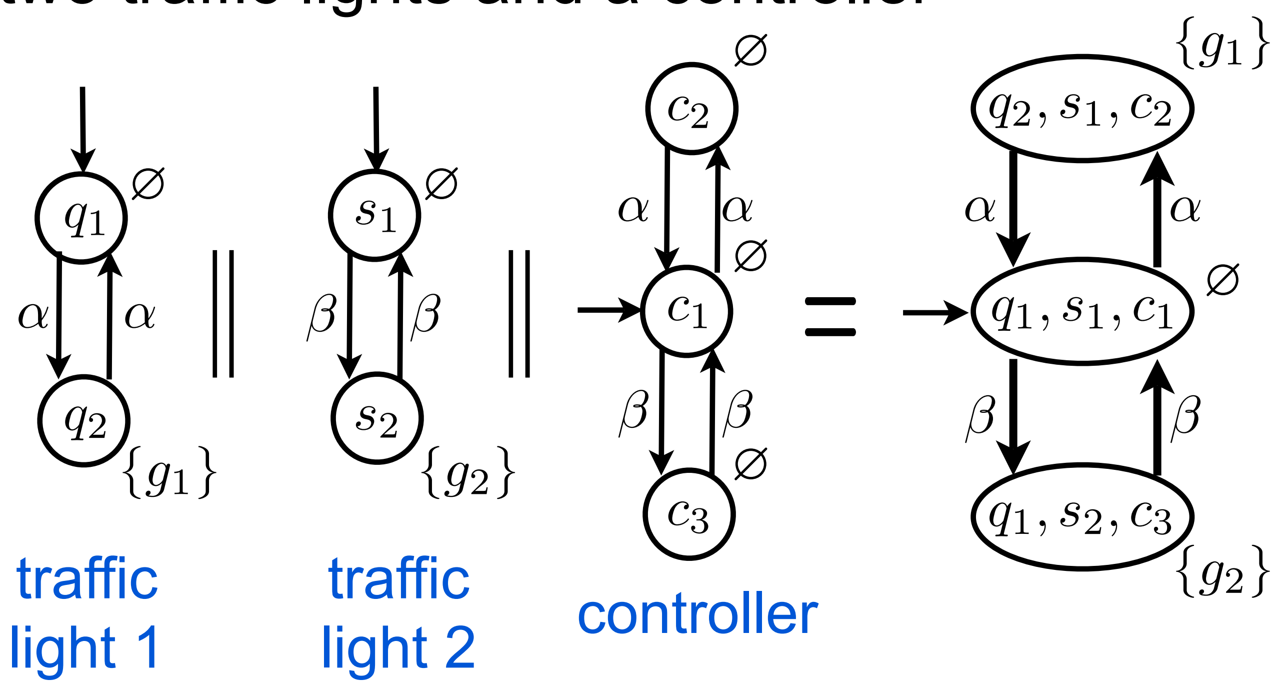


Specification P_1:
"The light are never green simultaneously."



Example 1: traffic lights (property verified)

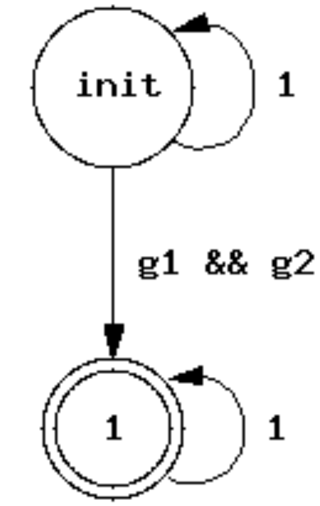
System *TS*: synchronous composition of two traffic lights and a controller



Property verified:

$$TS \models P_1$$

Specification P_1:
"The light are never green simultaneously."



$\mathcal{A}_{\neg P_1}$

SPIN code:

System model (synchronous composition of the modules):

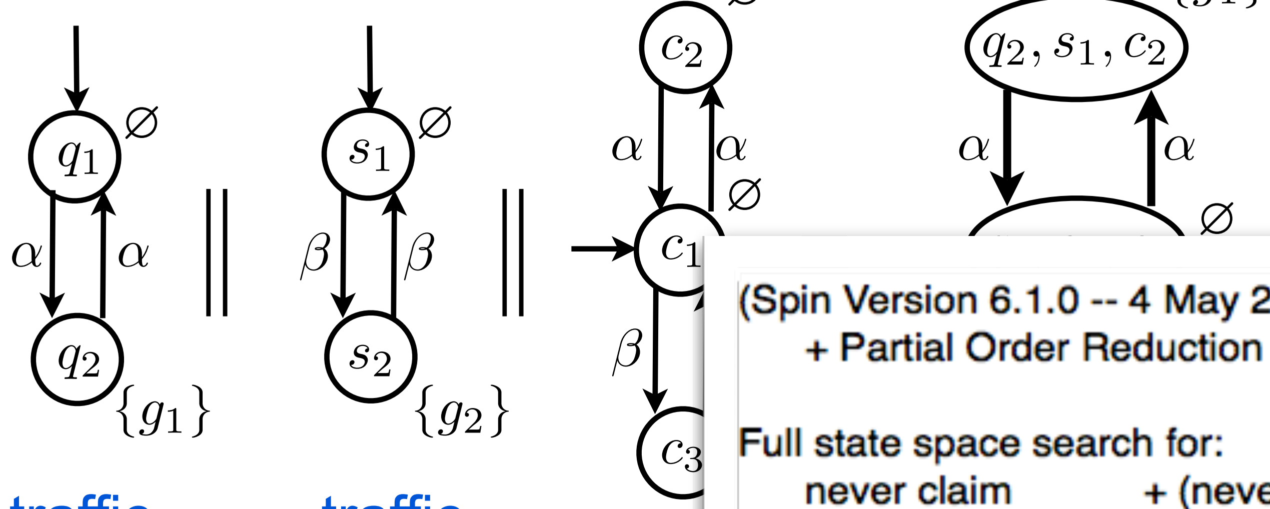
```
:: atomic{ (g1==0 && g2==0) -> g1=1; g2=0 }
:: atomic{ (g1==0 && g2==0) -> g1=0; g2=1 }
:: atomic{ (g1==1 && g2==0) -> g1=0; g2=0 }
:: atomic{ (g1==0 && g2==1) -> g1=0; g2=0 }
```

$\mathcal{A}_{\neg P_1}$ from LTL2BA:

```
T0_init :    /* init */
     if
     :: (1) -> goto T0_init
     :: (g1 && g2) -> goto accept_all
     fi;
accept_all :    /* 1 */
```

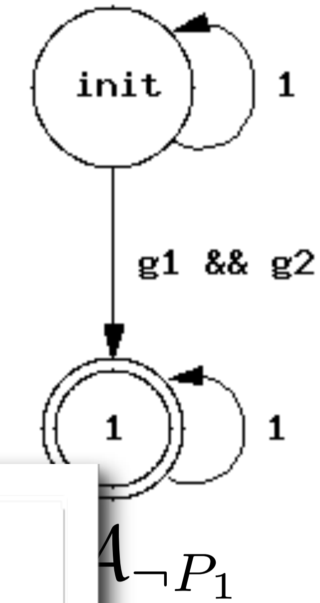
Example 1: traffic lights (property verified)

System *TS*: synchronous composition of two traffic lights and a controller



traffic light 1 | traffic light 2 | contr…

Specification P_1: "The light are never green simultaneously."



Property verified:

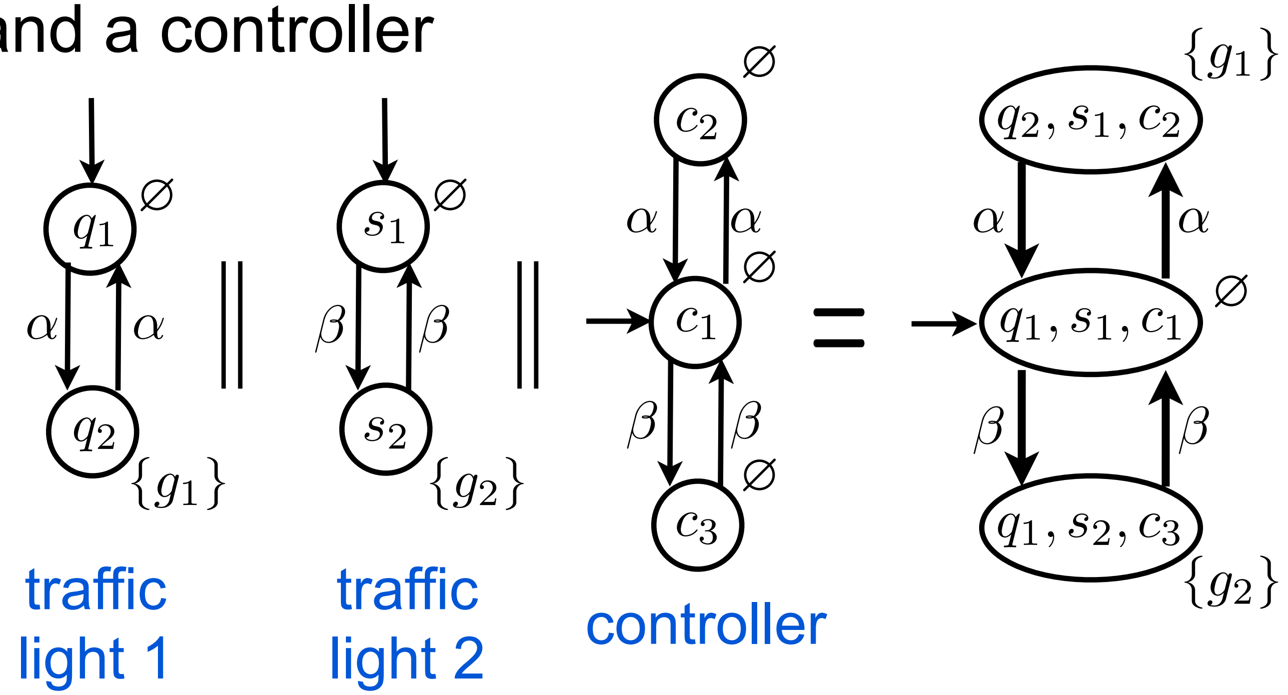
$$TS \models P_1$$

```
(Spin Version 6.1.0 -- 4 May 2011)
      + Partial Order Reduction

Full state space search for:
      never claim            + (never_0)
      assertion violations   + (if within scope of claim)
      acceptance  cycles     + (fairness disabled)
      invalid end states     - (disabled by never claim)

State-vector 28 byte, depth reached 3, errors: 0
        3 states, stored
        2 states, matched
        5 transitions (= stored+matched)
        0 atomic steps
hash conflicts:         0 (resolved)

Stats on memory usage (in Megabytes):
    0.000    equivalent memory usage for states (stored*(State-vector + overhead))
    0.289    actual memory usage for states (unsuccessful compression: 180519.05%)
          state-vector as stored = 101063 byte + 28 byte overhead
    4.000    memory used for hash table (-w19)
    0.534    memory used for DFS stack (-m10000)
    4.730    total actual memory usage
```

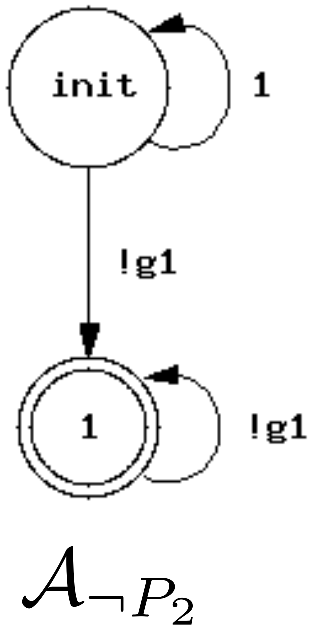
```
; g2=0 }
; g2=1 }
; g2=0 }
; g2=0 }
accept_all :    /* 1 */
```

Example 2: traffic lights (counterexample found → property not verified)

System *TS*: composition of two traffic lights and a controller

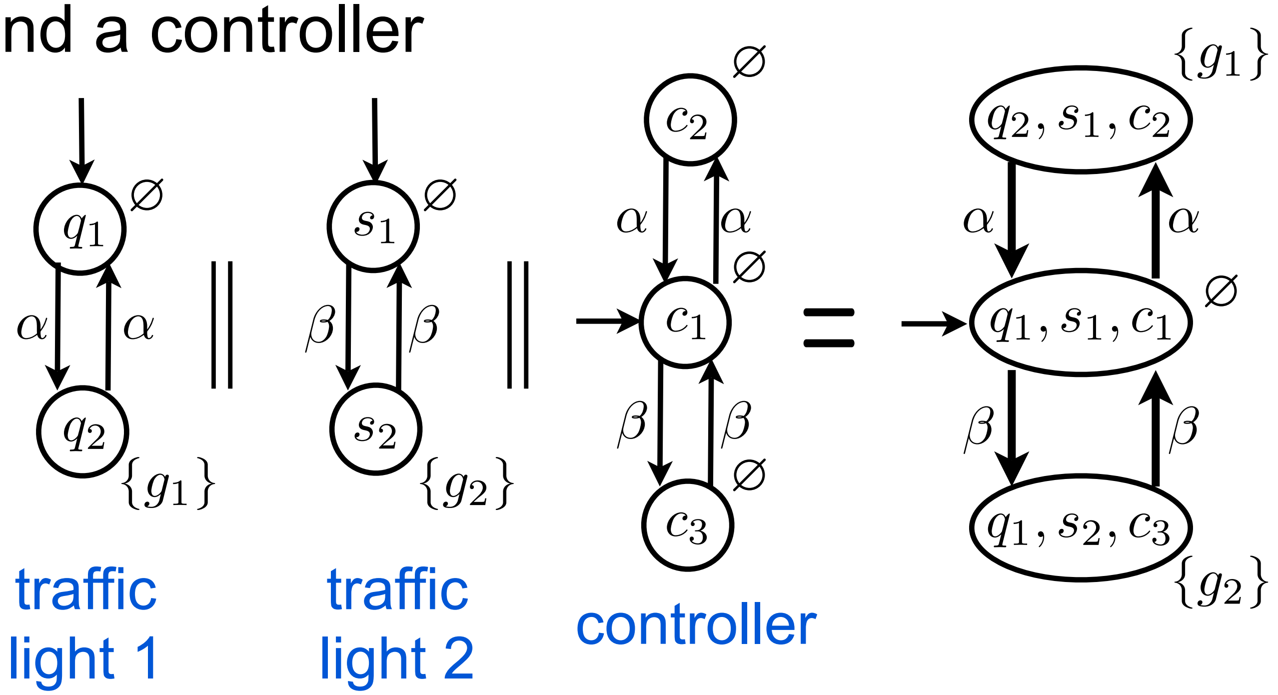


Specification P_2: "The first light is infinitely often green."

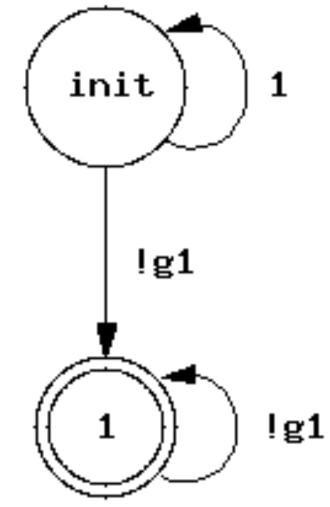


Example 2: traffic lights (counterexample found → property not verified)

System *TS*: composition of two traffic lights and a controller



Specification P_2:
"The first light is infinitely often green."

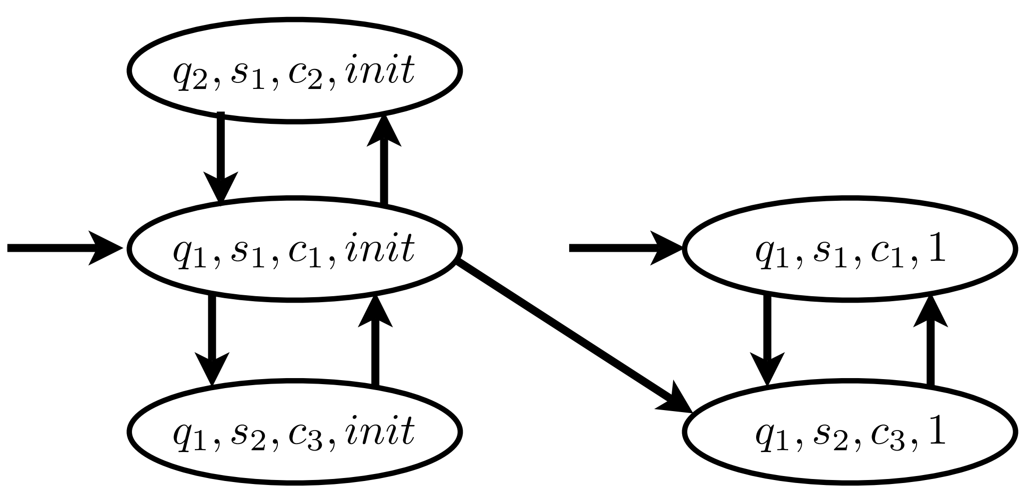


$\mathcal{A}_{\neg P_2}$

Property not verified: $TS \not\models P_2$

Counterexample:

$$(\langle q_1, s_1, c_1, 1\rangle\langle q_1, s_2, c_3, 1\rangle)^\omega$$



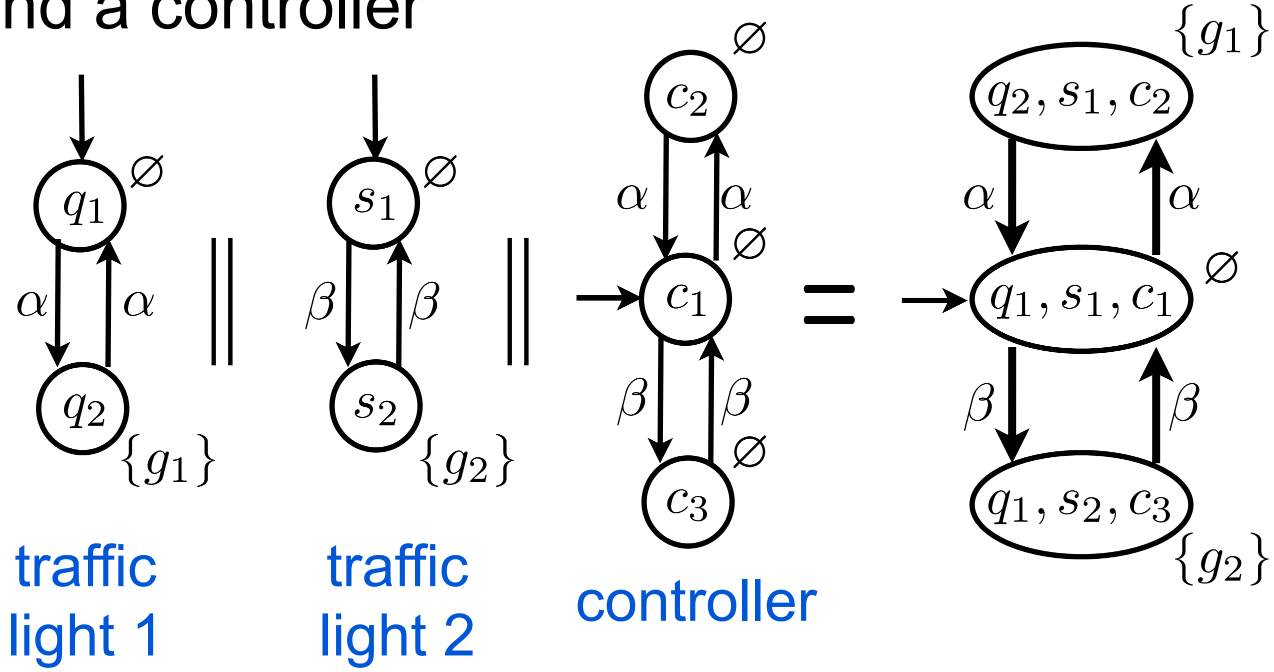
$TS \otimes \mathcal{A}_{\neg P_2}$

Counterexample from SPIN output:

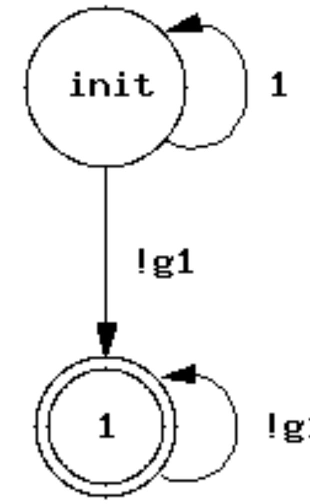
```
<<<<<START OF CYCLE>>>>>
Never claim moves to line 21      [((!(g1))]
    : (state 5)   [((((g1==0)&&(g2==0)))]
    : (state 6)   [g1 = 0]
    : (state 7)   [g2 = 1]
    : (state 13)  [((((g1==0)&&(g2==1)))]
    : (state 14)  [g1 = 0]
    : (state 15)  [g2 = 0]
spin: trail ends after 8 steps
```

Example 2: traffic lights (counterexample found → property not verified)

System *TS*: composition of two traffic lights and a controller



Specification P_2:
"The first light is infinitely often green."

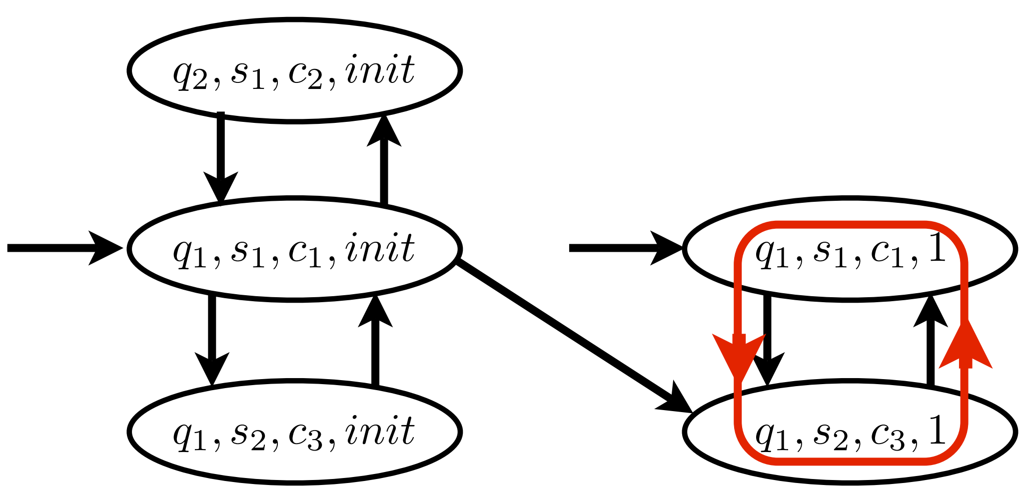


$\mathcal{A}_{\neg P_2}$

Property not verified: $TS \not\models P_2$

Counterexample:

$$(\langle q_1, s_1, c_1, 1\rangle \langle q_1, s_2, c_3, 1\rangle)^\omega$$



$TS \otimes \mathcal{A}_{\neg P_2}$

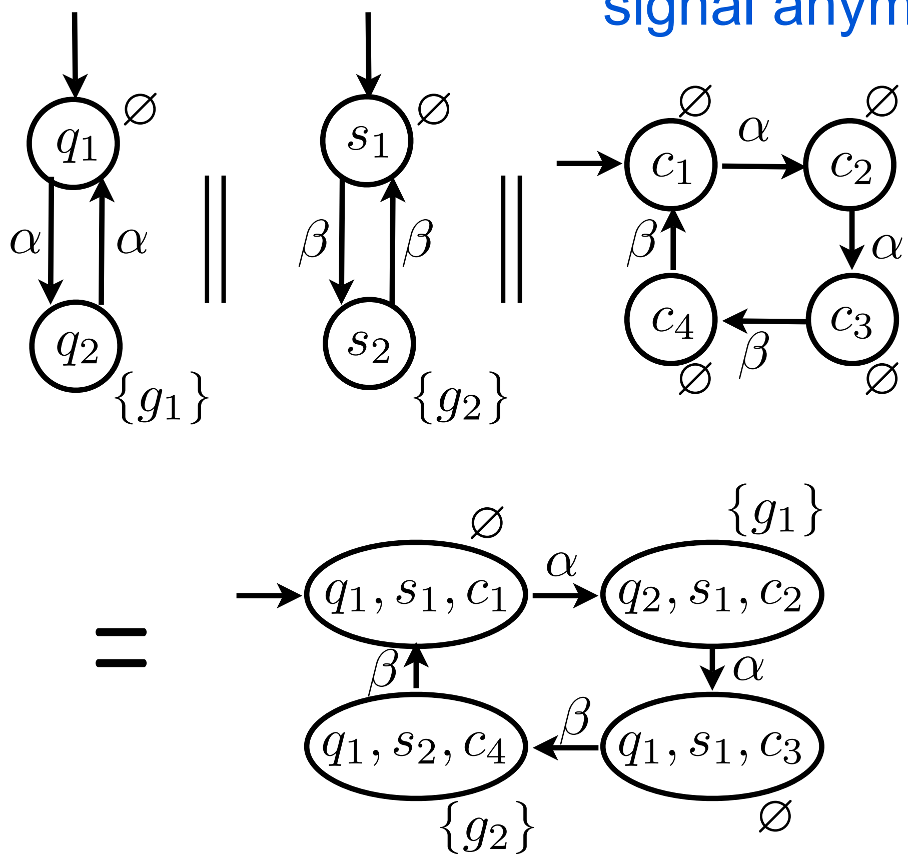
Counterexample from SPIN output:

```
<<<<<START OF CYCLE>>>>>
Never claim moves to line 21      [((!(g1))]
    : (state 5)   [(((g1==0)&&(g2==0)))]
    : (state 6)   [g1 = 0]
    : (state 7)   [g2 = 1]
    : (state 13)  [(((g1==0)&&(g2==1)))]
    : (state 14)  [g1 = 0]
    : (state 15)  [g2 = 0]
spin: trail ends after 8 steps
```

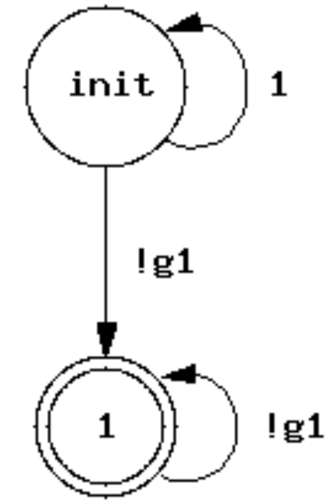
Example 3: traffic lights (counterexample used to modify the controller)

System *TS*: composition of two traffic lights and a modified controller

new controller: β^ω is not a valid control signal anymore



Specification P_2: "The first light is infinitely often green."

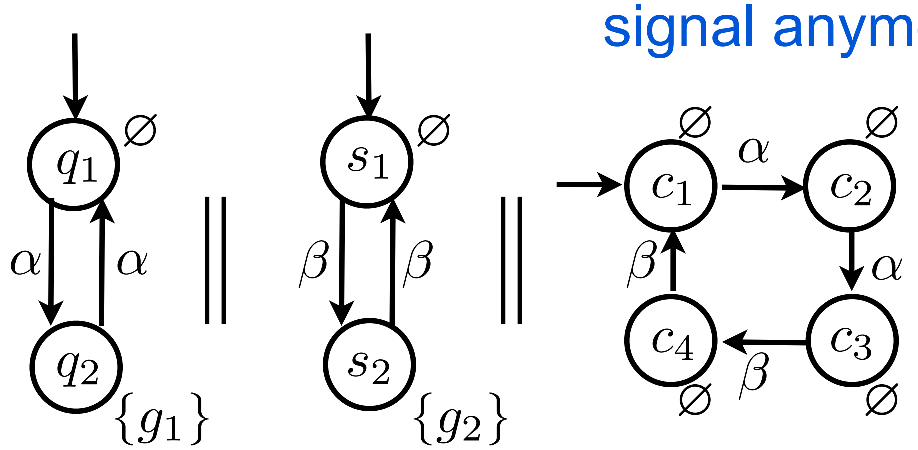


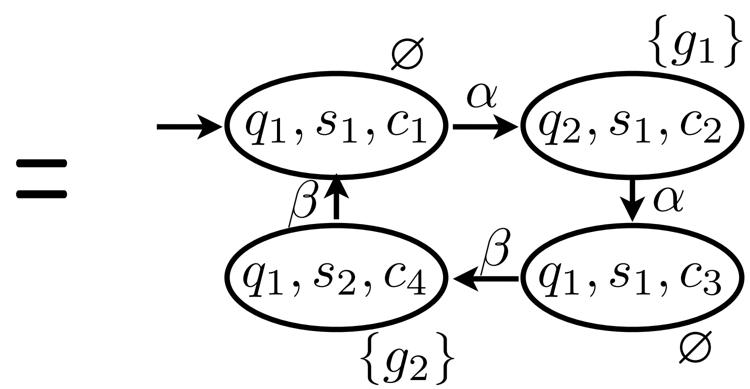
$\mathcal{A}_{\neg P_2}$

Example 3: traffic lights (counterexample used to modify the controller)

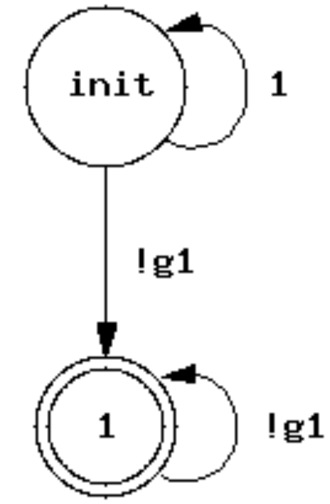
System *TS*: composition of two traffic lights and a modified controller

new controller: β^ω is not a valid control signal anymore





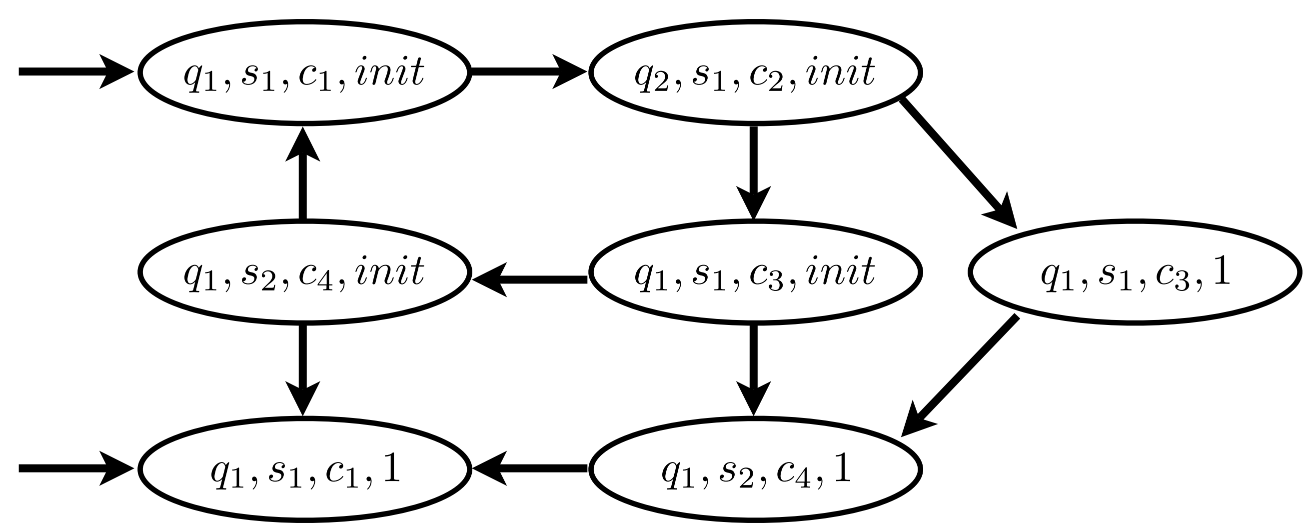
Specification P_2:
"The first light is infinitely often green."



$\mathcal{A}_{\neg P_2}$

Property verified:

$$TS \vDash P_2$$



$TS \otimes \mathcal{A}_{\neg P_2}$

Computational complexity of model checking

Transition system: $TS = (S, \mathrm{Act}, \rightarrow, I, \mathrm{AP}, L)$. Specification: Φ

Problem size:

$$\left(\begin{array}{c}\#\text{ of reachable}\\\text{states in } TS\end{array}\right) \times \left(\begin{array}{c}\#\text{ of states}\\\text{in } \mathcal{A}_{\neg\Phi}\end{array}\right) \times \left(\begin{array}{c}\text{size of one}\\\text{state in bytes}\end{array}\right)$$

$O(|S|)$ $\qquad$ $2^{O(|\neg\Phi|)}$

"length" of $\neg\Phi$, e.g., # of operators in $\neg\Phi$

Computational complexity of model checking

Transition system: $TS = (S, \text{Act}, \rightarrow, I, \text{AP}, L)$. Specification: Φ

Problem size:

$$\begin{pmatrix} \#\text{ of reachable} \\ \text{states in } TS \end{pmatrix} \times \begin{pmatrix} \#\text{ of states} \\ \text{in } \mathcal{A}_{\neg\Phi} \end{pmatrix} \times \begin{pmatrix} \text{size of one} \\ \text{state in bytes} \end{pmatrix}$$

$$O(|S|) \qquad 2^{O(|\neg\Phi|)}$$

$|\neg\Phi|$: "length" of $\neg\Phi$, e.g., # of operators in $\neg\Phi$

Potential reductions:

- Restrict the ranges of variables
- Use abstraction, separation of concerns, generalization
- Use compressed representation of the state space (e.g. BDD)
  - Used in symbolic model checkers, e.g., SMV, NuSMV
- **Partial order reduction** (avoid computing equivalent paths)

Computational complexity of model checking

Transition system: $TS = (S, \mathrm{Act}, \rightarrow, I, \mathrm{AP}, L)$. Specification: Φ

Problem size:

$$\begin{pmatrix} \#\text{ of reachable} \\ \text{states in } TS \end{pmatrix} \times \begin{pmatrix} \#\text{ of states} \\ \text{in } \mathcal{A}_{\neg\Phi} \end{pmatrix} \times \begin{pmatrix} \text{size of one} \\ \text{state in bytes} \end{pmatrix}$$

$O(|S|)$ $\quad$ $2^{O(|\neg\Phi|)}$ → "length" of $\neg\Phi$, e.g., # of operators in $\neg\Phi$

Potential reductions:

- Restrict the ranges of variables
- Use abstraction, separation of concerns, generalization
- Use compressed representation of the state space (e.g. BDD)
  - Used in symbolic model checkers, e.g., SMV, NuSMV
- **Partial order reduction** (avoid computing equivalent paths)

- Use separable properties, instead of large, combined ones

Computational complexity of model checking

Transition system: $TS = (S, \text{Act}, \rightarrow, I, \text{AP}, L)$. Specification: Φ

Problem size:

$$\begin{pmatrix} \text{\# of reachable} \\ \text{states in } TS \end{pmatrix} \times \begin{pmatrix} \text{\# of states} \\ \text{in } \mathcal{A}_{\neg\Phi} \end{pmatrix} \times \begin{pmatrix} \text{size of one} \\ \text{state in bytes} \end{pmatrix}$$

$O(|S|)$ $\qquad$ $2^{O(|\neg\Phi|)}$

$|\neg\Phi|$ → "length" of $\neg\Phi$, e.g., # of operators in $\neg\Phi$

Potential reductions:

- Restrict the ranges of variables
- Use abstraction, separation of concerns, generalization
- Use compressed representation of the state space (e.g. BDD)
  - Used in symbolic model checkers, e.g., SMV, NuSMV
- **Partial order reduction** (avoid computing equivalent paths)

- Use separable properties, instead of large, combined ones

- Lossy compression, e.g., hash-compact and bitstate hashing
  - May result in incompleteness
- Lossless compression and alternate state representation methods
  - May increase time while reduce memory

Computational complexity of model checking

Transition system: $TS = (S, \mathrm{Act}, \rightarrow, I, \mathrm{AP}, L)$. Specification: Φ

Problem size:

$$\left(\begin{array}{c} \#\text{ of reachable} \\ \text{states in } TS \end{array} \right) \times \left(\begin{array}{c} \#\text{ of states} \\ \text{in } \mathcal{A}_{\neg\Phi} \end{array} \right) \times \left(\begin{array}{c} \text{size of one} \\ \text{state in bytes} \end{array} \right)$$

$O(|S|)$ $\quad 2^{O(|\neg\Phi|)}$ → "length" of $\neg\Phi$, e.g., # of operators in $\neg\Phi$

Potential reductions:

- Restrict the ranges of variables
- Use abstraction, separation of concerns, generalization
- Use compressed representation of the state space (e.g. BDD)
  - Used in symbolic model checkers, e.g., SMV, NuSMV
- **Partial order reduction** (avoid computing equivalent paths)

- Use separable properties, instead of large, combined ones

- Lossy compression, e.g., hash-compact and bitstate hashing
  - May result in incompleteness
- Lossless compression and alternate state representation methods
  - May increase time while reduce memory

"**On-the-fly**" construction of TS, $\mathcal{A}_{\neg\Phi}$ and the product automaton (while searching the automaton) to avoid constructing the complete state space

Computational complexity of model checking

Transition system: $TS = (S, \mathrm{Act}, \rightarrow, I, \mathrm{AP}, L)$. Specification: Φ

Problem size:

$$\left(\begin{array}{c}\#\text{ of reachable}\\ \text{states in } TS\end{array}\right) \times \left(\begin{array}{c}\#\text{ of states}\\ \text{in } \mathcal{A}_{\neg\Phi}\end{array}\right) \times \left(\begin{array}{c}\text{size of one}\\ \text{state in bytes}\end{array}\right)$$

$O(|S|)$ $\qquad 2^{O(|\neg\Phi|)}$ → "length" of $\neg\Phi$, e.g., # of operators in $\neg\Phi$

Potential reductions:

- Restrict the ranges of variables
- Use abstraction, separation of concerns, generalization
- Use compressed representation of the state space (e.g. BDD)
  - Used in symbolic model checkers, e.g., SMV, NuSMV
- **Partial order reduction** (avoid computing equivalent paths)

- Use separable properties, instead of large, combined ones

- Lossy compression, e.g., hash-compact and bitstate hashing
  - May result in incompleteness
- Lossless compression and alternate state representation methods
  - May increase time while reduce memory

"**On-the-fly**" construction of TS, $\mathcal{A}_{\neg\Phi}$ and the product automaton (while searching the automaton) to avoid constructing the complete state space

Time complexity of DFS: $O(\#\text{ of states} + \#\text{ of transitions in } TS \otimes A_{\neg\Phi})$

Closed system synthesis

Closed system: behaviors are generated purely by the system itself without any external influence

Given:

- A transition system *P*
- An LTL formula Φ

Compute: A path π of P such that

$$\pi \models \Phi$$

Closed system synthesis

Closed system: behaviors are generated purely by the system itself without any external influence

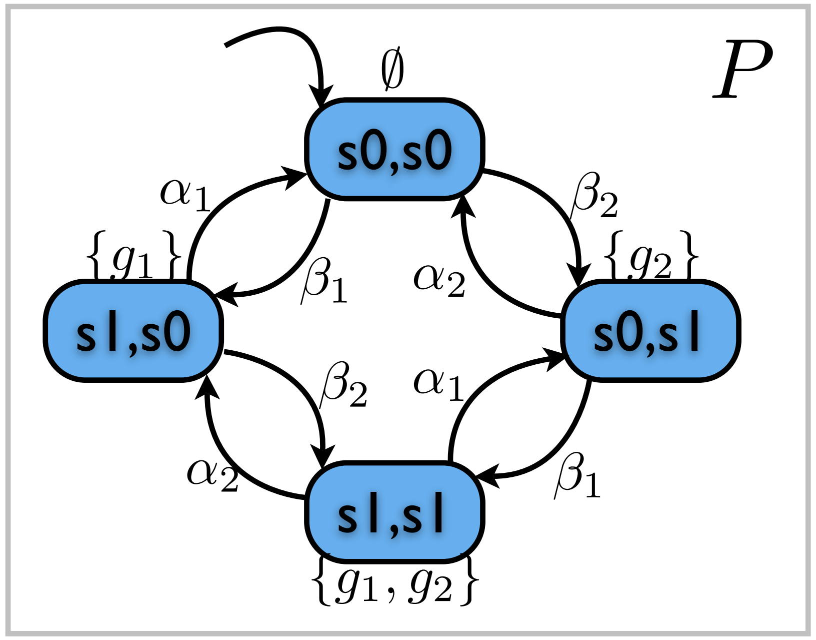
Given:

- A transition system *P*
- An LTL formula Φ

Compute: A path π of P such that

$$\pi \models \Phi$$

P: composition of two traffic lights



$$\Phi \quad = \quad \Box\neg(g_1 \wedge g_2) \wedge \Box\Diamond g_1 \wedge \Box\Diamond g_2$$

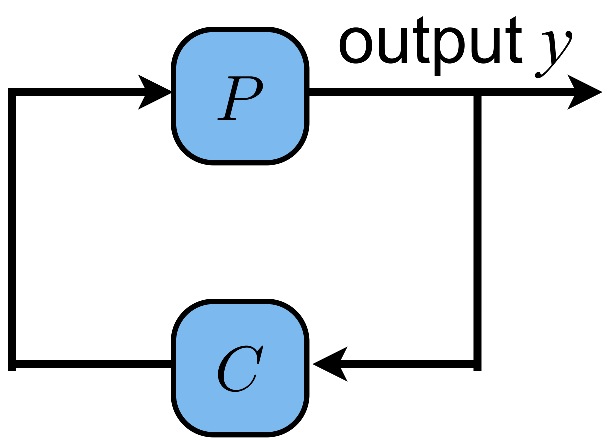
Sample paths of *P*:

$$\pi_1 \quad = \quad (\langle s_0 s_0\rangle\langle s_1 s_0\rangle\langle s_1 s_1\rangle\langle s_0 s_1\rangle)^\omega \text{ ✗}$$

$$\pi_2 \quad = \quad (\langle s_0 s_0\rangle\langle s_0 s_1\rangle)^\omega \text{ ✗}$$

$$\pi_3 \quad = \quad (\langle s_0 s_0\rangle\langle s_1 s_0\rangle\langle s_0 s_0\rangle\langle s_0 s_1\rangle)^\omega \text{ ✓}$$

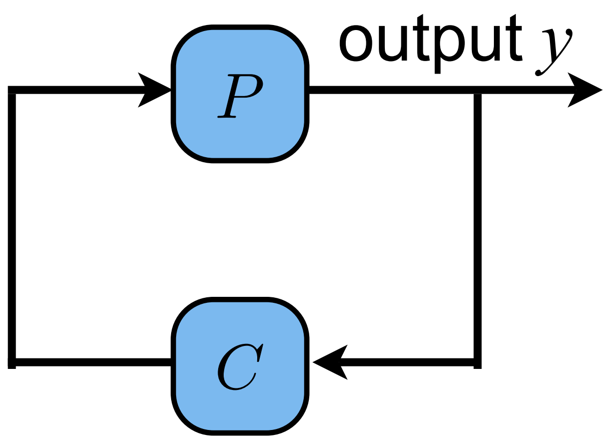
Closed system synthesis--a "controls" interpretation



The controller *C* is a function $C : M \times S \to Act$

- The controller keeps some history of states
- It picks the next action for *P* such that the resulting path satisfies the specification Φ (i.e., *C* constrains the paths system can take.

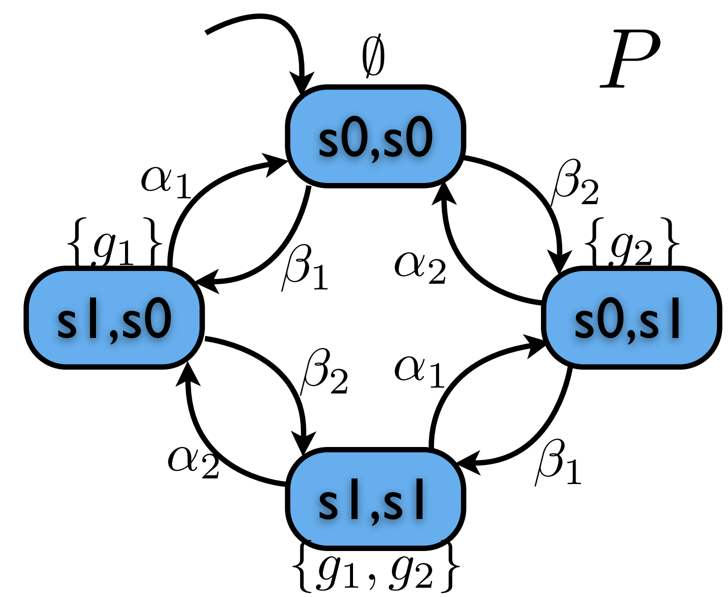
Closed system synthesis--a "controls" interpretation



memory domain

The controller *C* is a function $C : M \times S \rightarrow Act$

- The controller keeps some history of states
- It picks the next action for *P* such that the resulting path satisfies the specification Φ (i.e., *C* constrains the paths system can take.

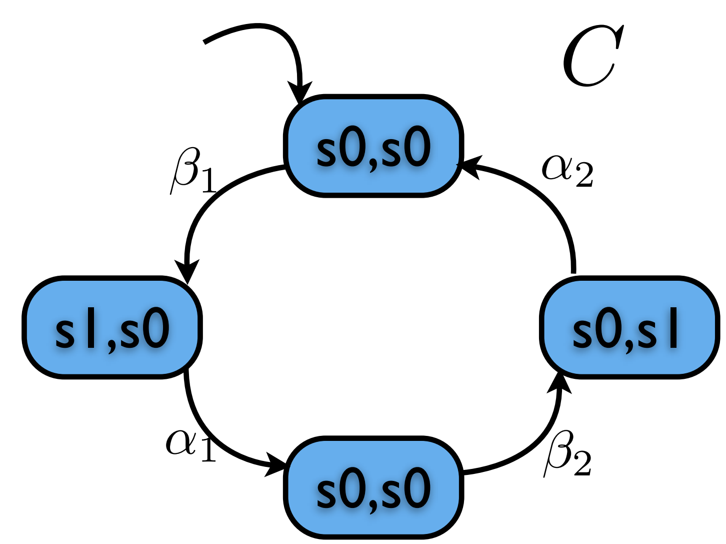


Let *M* be a sequence of length 1, i.e., the controller keeps only the previous state

$$\begin{aligned}
C(\emptyset, \langle s_0 s_0 \rangle) &= \beta_1 \\
C(\langle s_0 s_1 \rangle, \langle s_0 s_0 \rangle) &= \beta_1 \\
C(\langle s_1 s_0 \rangle, \langle s_0 s_0 \rangle) &= \beta_2 \\
C(\langle s_0 s_0 \rangle, \langle s_1 s_0 \rangle) &= \alpha_1 \\
C(\langle s_0 s_0 \rangle, \langle s_0 s_1 \rangle) &= \alpha_2
\end{aligned}$$

$$\Rightarrow \pi = (\langle s_0 s_0 \rangle \langle s_1 s_0 \rangle \langle s_0 s_0 \rangle \langle s_0 s_1 \rangle)^\omega$$

and $\pi \models \Phi = \Box\neg(g_1 \wedge g_2) \wedge \Box\Diamond g_1 \wedge \Box\Diamond g_2$



A solution approach

- Closed system synthesis can be formulated as a non-emptiness of the specification or satisfiability problem

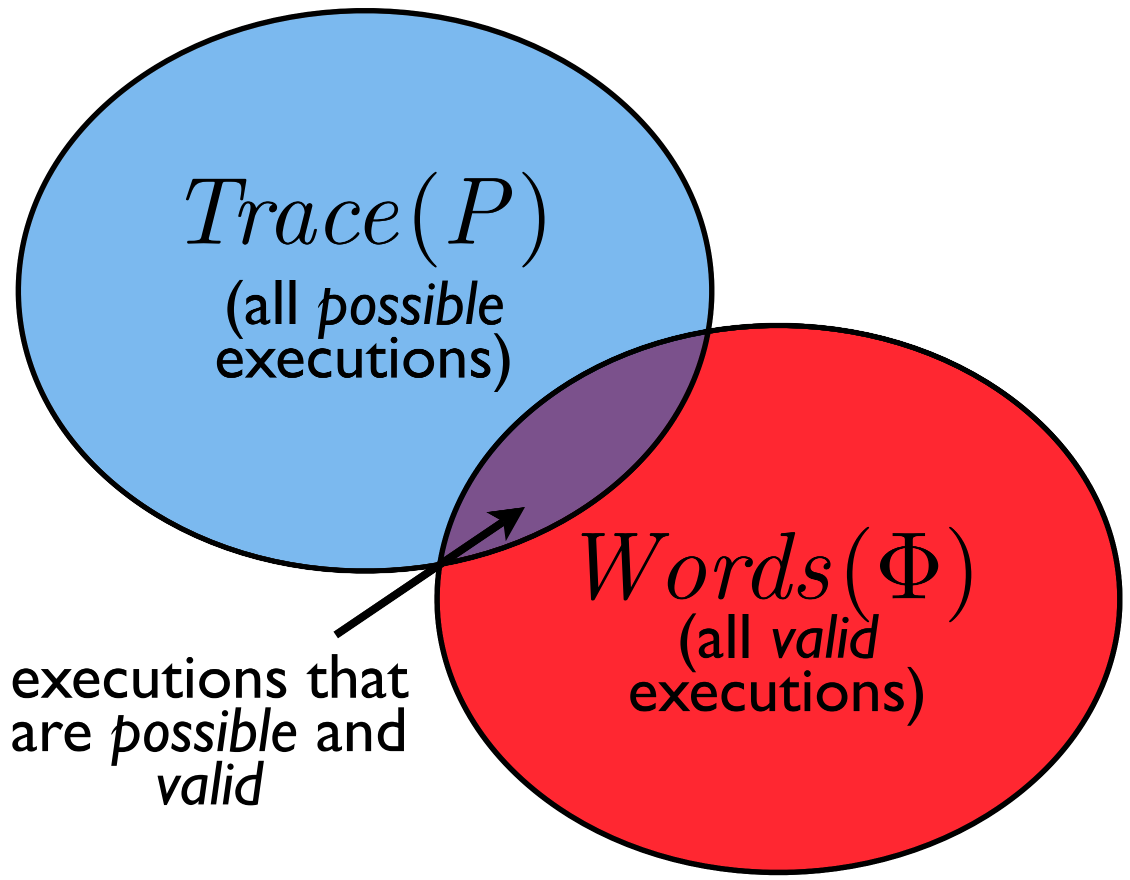
$$\exists y \cdot \Phi(y)$$

A solution approach

- Closed system synthesis can be formulated as a non-emptiness of the specification or satisfiability problem

$$\exists y \cdot \Phi(y)$$

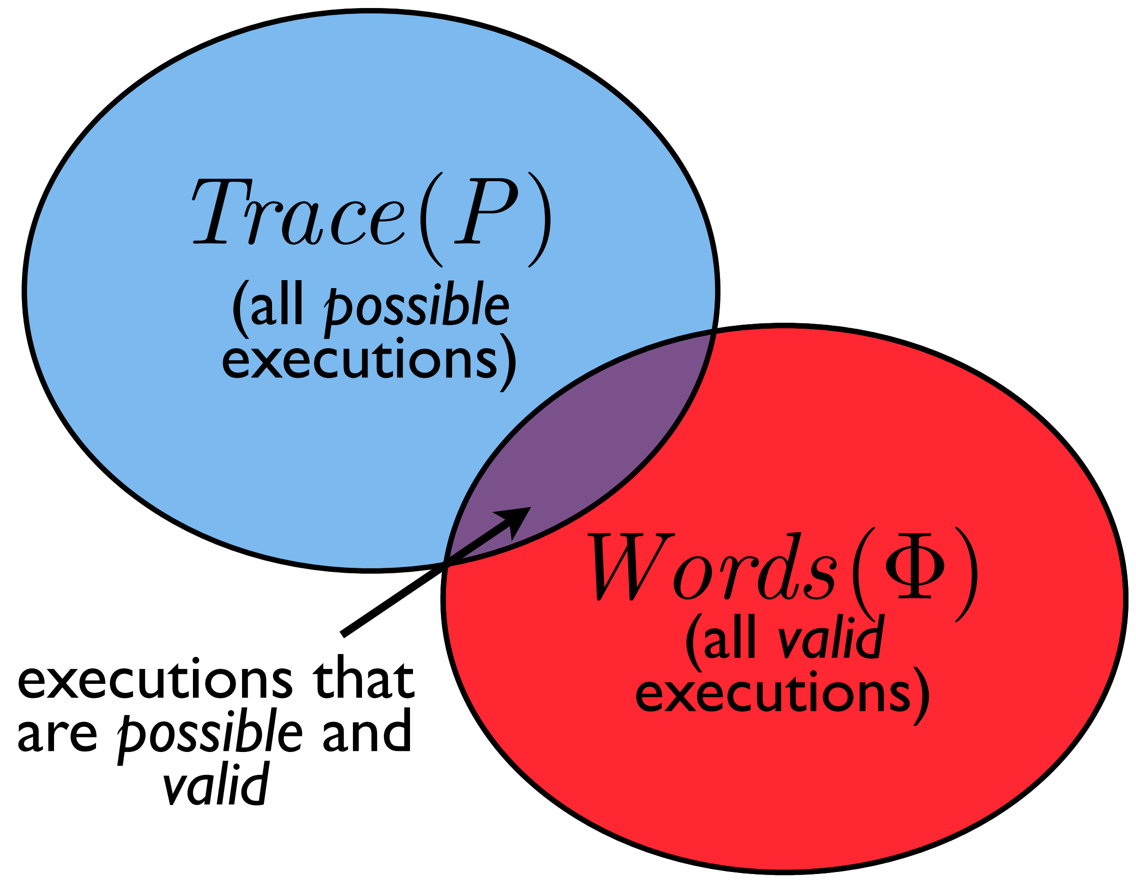
- For synthesis problems, “interesting” behaviors are “good” behaviors (as opposed to verification problems where “interesting behaviors are “bad” behaviors)



A solution approach

- Closed system synthesis can be formulated as a non-emptiness of the specification or satisfiability problem

$$\exists y \cdot \Phi(y)$$



- For synthesis problems, "interesting" behaviors are "good" behaviors (as opposed to verification problems where "interesting behaviors are "bad" behaviors)

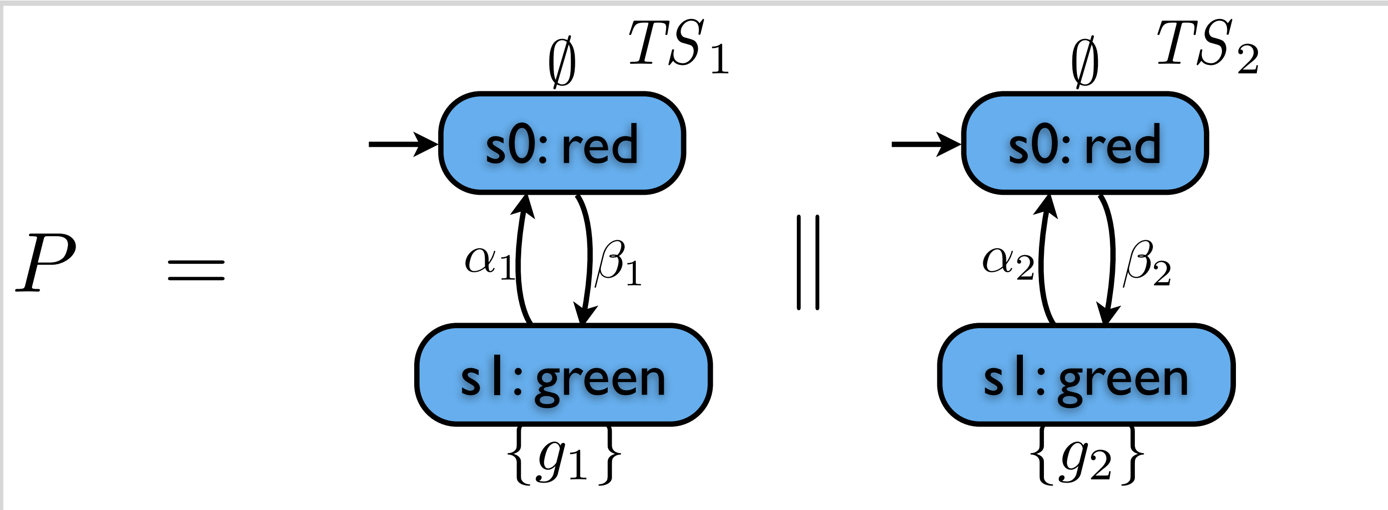
- Construct a verification model and claim that

$$Trace(P) \cap Words(\Phi) = \emptyset$$

  - A counterexample provided in case of negative result is a path π of *P* that satisfies Φ
  - Positive result means $Trace(P) \cap Words(\Phi) = \emptyset$, i.e., a path π of *P* that satisfies Φ does not exist

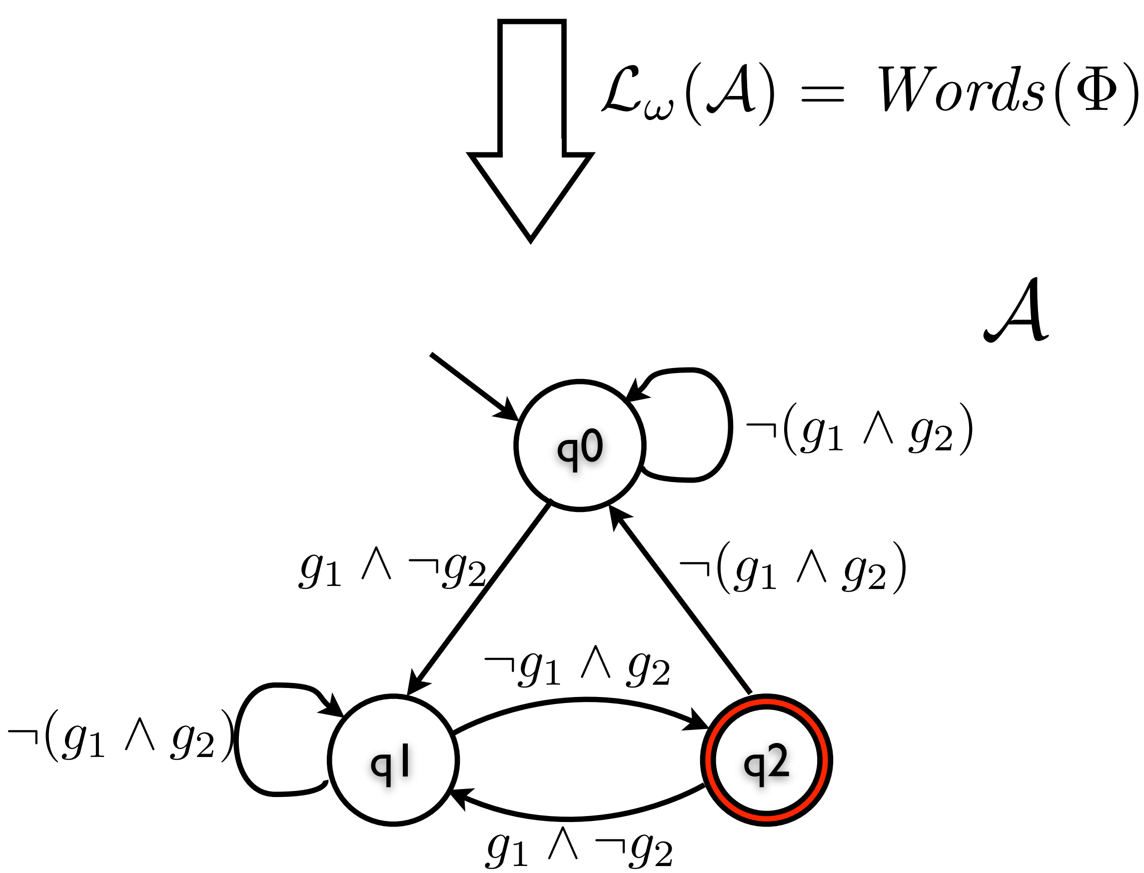
Example: traffic lights

System model:



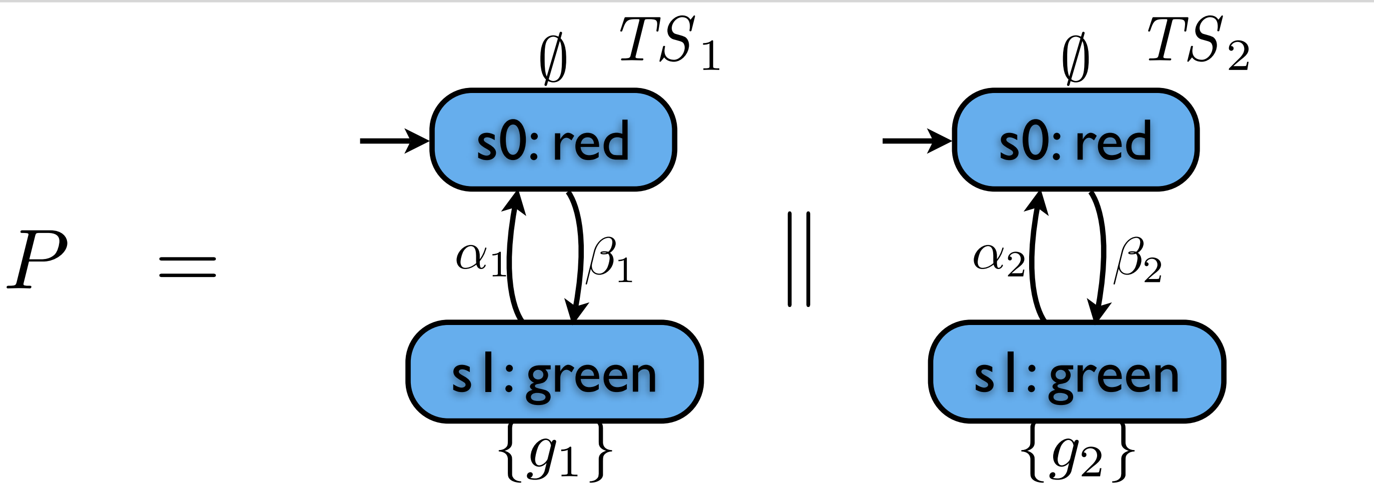
Specification:

$$\Phi \quad = \quad \Box\neg(g_1 \wedge g_2) \wedge \Box\Diamond g_1 \wedge \Box\Diamond g_2$$



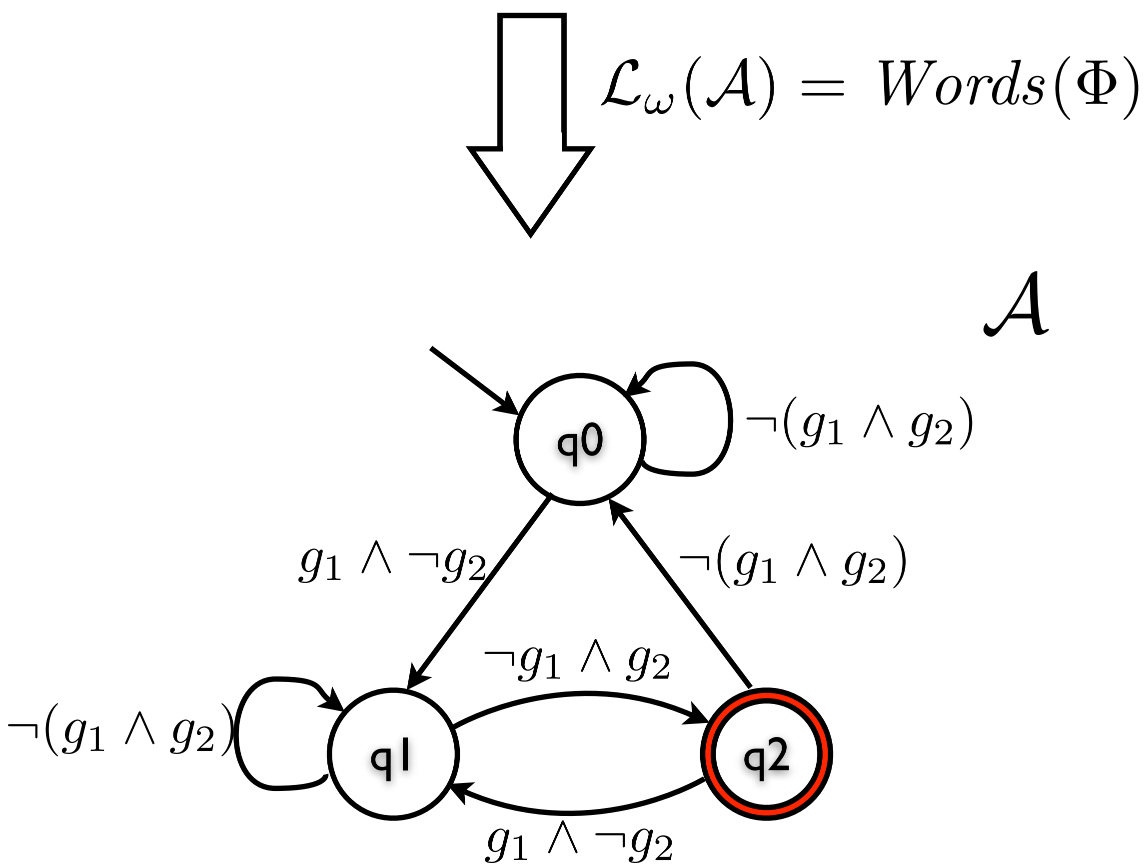
Example: traffic lights

System model:



Specification:

$$\Phi \; = \; \Box\neg(g_1 \wedge g_2) \wedge \Box\Diamond g_1 \wedge \Box\Diamond g_2$$



SPIN code:

System model (asynchronous composition):

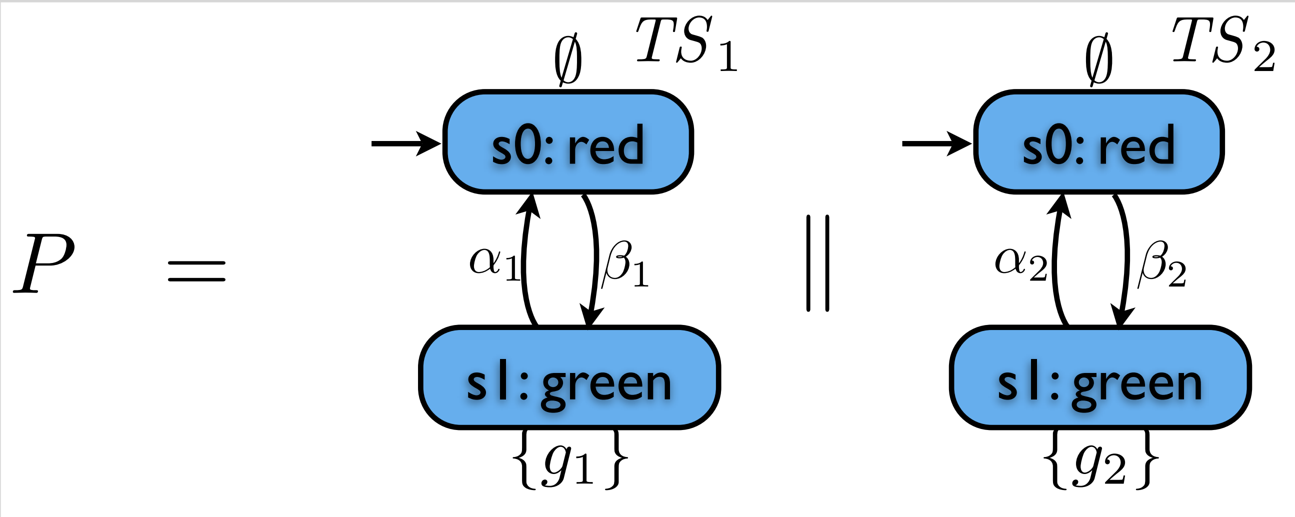
```
active proctype TL1() {
   do
   ::  atomic{ g1 == 0 -> g1 = 1}
   ::  atomic{ g1 == 1 -> g1 = 0 }
   od
}
active proctype TL2() {
   do
   ::  atomic{ g2 == 0 -> g2 = 1}
   ::  atomic{ g2 == 1 -> g2 = 0 }
   od
}
```

Automaton from LTL2BA:

```
T0_init:
   if
   ::  (!g1) || (!g2) -> goto T0_init
   ::  (g1 && !g2) -> goto T1_S1
   fi;
T1_S1:
   if
   ::  (!g1) || (!g2) -> goto T1_S1
   ::  (!g1 && g2) -> goto accept_S1
   fi;
accept_S1:
   if
   ::  (!g1) || (!g2) -> goto T0_init
   ::  (g1 && !g2) -> goto T1_S1
   fi;
}
```

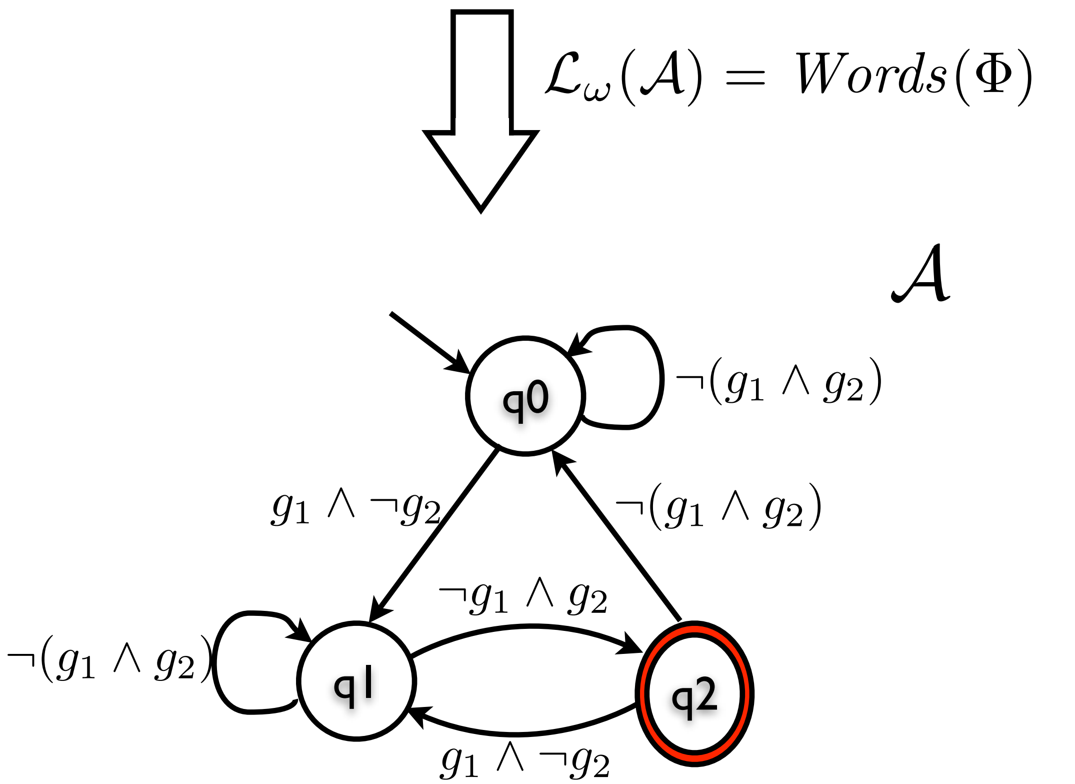
Solution to the traffic light problem

System model:



Specification:

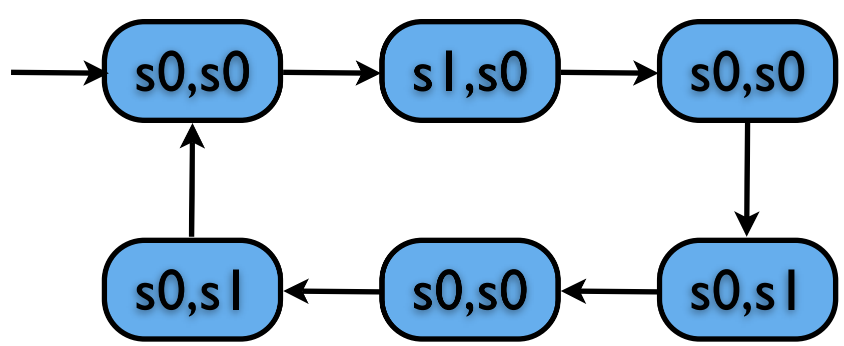
$$\Phi = \Box\neg(g_1 \wedge g_2) \wedge \Box\Diamond g_1 \wedge \Box\Diamond g_2$$



Solution from SPIN output:

```
<<<<<START OF CYCLE>>>>>

(state 1)     [((g1==0))]
(state 2)     [g1 = 1]

(state 4)     [((g1==1))]
(state 5)     [g1 = 0]

(state 1)     [((g2==0))]
(state 2)     [g2 = 1]

(state 4)     [((g2==1))]
(state 5)     [g2 = 0]

(state 1)     [((g2==0))]
(state 2)     [g2 = 1]

(state 4)     [((g2==1))]
(state 5)     [g2 = 0]
```

$$\pi = (\langle s_0 s_0\rangle\langle s_1 s_0\rangle\langle s_0 s_0\rangle\langle s_0 s_1\rangle\langle s_0 s_0\rangle\langle s_0 s_1\rangle)^\omega$$



Example: the farmer puzzle

A farmer wants to cross a river in a little boat with a wolf, a goat and a cabbage.

Constraints:

- The boat is only big enough to carry the farmer plus one other animal or object.
- The wolf will eat the goat if the farmer is not present.
- The goat will eat the cabbage if the farmer is not present.

How can the farmer get both animals and the cabbage safely across the river?

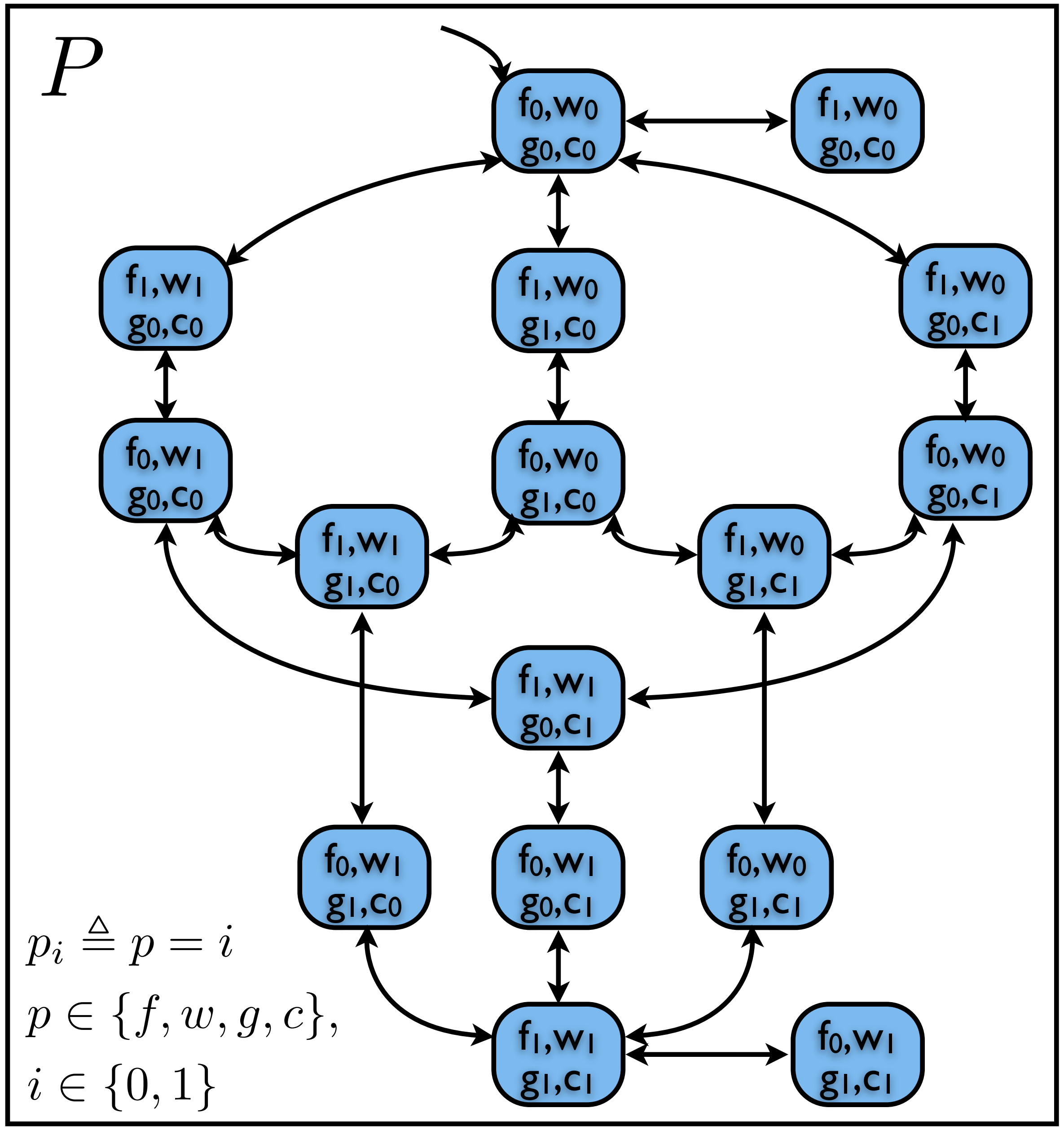
Example: the farmer puzzle

A farmer wants to cross a river in a little boat with a wolf, a goat and a cabbage.

Constraints:

- The boat is only big enough to carry the farmer plus one other animal or object.
- The wolf will eat the goat if the farmer is not present.
- The goat will eat the cabbage if the farmer is not present.

How can the farmer get both animals and the cabbage safely across the river?



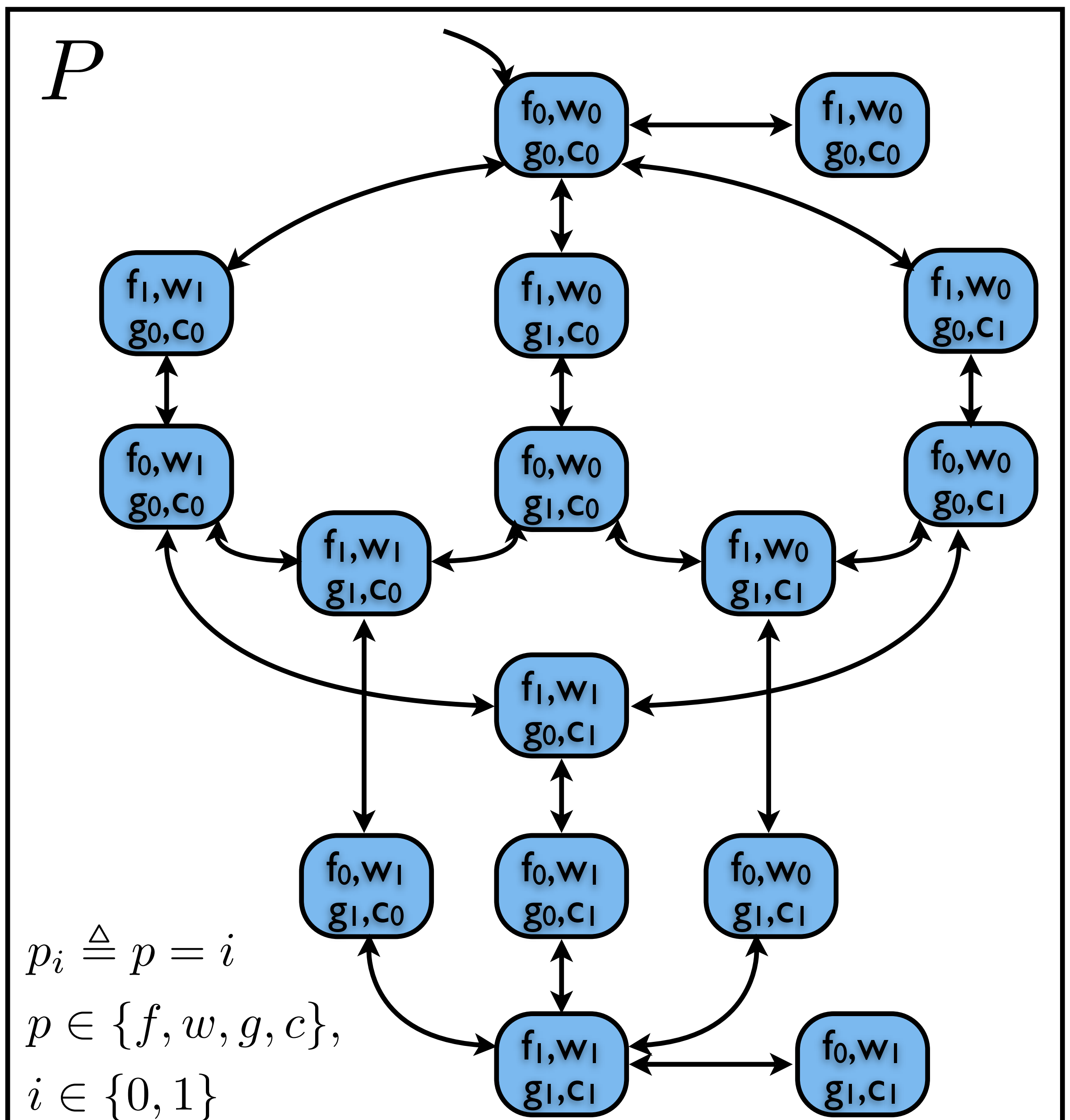
Example: the farmer puzzle

A farmer wants to cross a river in a little boat with a wolf, a goat and a cabbage.

Constraints:

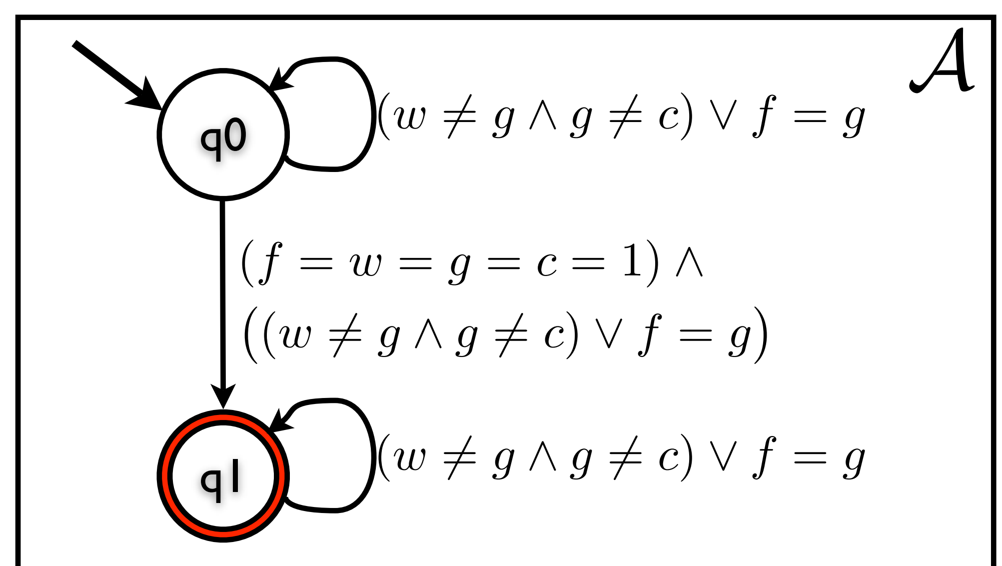
- The boat is only big enough to carry the farmer plus one other animal or object.
- The wolf will eat the goat if the farmer is not present.
- The goat will eat the cabbage if the farmer is not present.

How can the farmer get both animals and the cabbage safely across the river?



$$\begin{aligned}\Phi \quad = \quad & \Diamond(f = w = g = c = 1) \wedge \\ & \Box(w \neq g \vee f = g) \wedge \\ & \Box(g \neq c \vee f = g)\end{aligned}$$

$$\mathcal{L}_\omega(\mathcal{A}) = Words(\Phi)$$



Solving the farmer puzzle (using SPIN)

A farmer wants to cross a river in a little boat with a wolf, a goat and a cabbage.

Constraints:

- The boat is only big enough to carry the farmer plus one other animal or object.
- The wolf will eat the goat if the farmer is not present.
- The goat will eat the cabbage if the farmer is not present.

System model in SPIN:

```
active proctype P() {
   do
   ::  f=1-f
   ::  atomic{ f==g -> f=1-f; g=1-g }
   ::  atomic{ f==w -> f=1-f; w=1-w }
   ::  atomic{ f==c -> f=1-f; c=1-c }
   od
}
```

- `f=1-f`: farmer crosses the river alone
- `atomic{ f==g -> f=1-f; g=1-g }`: farmer and goat cross the river
- `atomic{ f==w -> f=1-f; w=1-w }`: farmer and wolf cross the river
- `atomic{ f==c -> f=1-f; c=1-c }`: farmer and cabbage cross the river

Specification:

$$\begin{aligned}\Phi \quad = \quad & \Diamond(f = w = g = c = 1) \wedge \\ & \Box(w \neq g \vee f = g) \wedge \\ & \Box(g \neq c \vee f = g)\end{aligned}$$

Solving the farmer puzzle (using SPIN)

A farmer wants to cross a river in a little boat with a wolf, a goat and a cabbage.

Constraints:

- The boat is only big enough to carry the farmer plus one other animal or object.
- The wolf will eat the goat if the farmer is not present.
- The goat will eat the cabbage if the farmer is not present.

System model in SPIN:

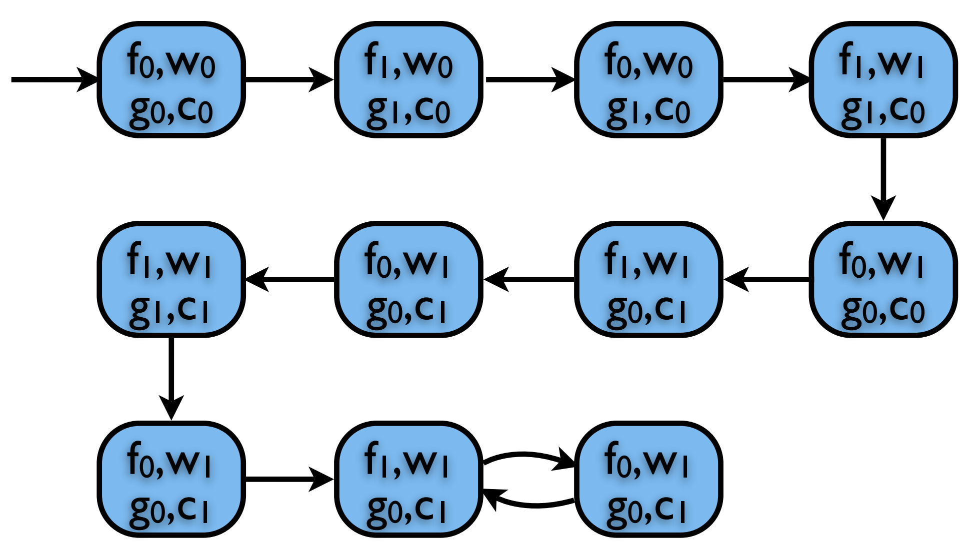
```
active proctype P() {
   do
   ::  f=1-f
   ::  atomic{ f==g -> f=1-f; g=1-g }
   ::  atomic{ f==w -> f=1-f; w=1-w }
   ::  atomic{ f==c -> f=1-f; c=1-c }
   od
}
```

- `f=1-f`: farmer crosses the river alone
- `atomic{ f==g -> f=1-f; g=1-g }`: farmer and goat cross the river
- `atomic{ f==w -> f=1-f; w=1-w }`: farmer and wolf cross the river
- `atomic{ f==c -> f=1-f; c=1-c }`: farmer and cabbage cross the river

Specification:

$$\begin{aligned}\Phi \quad = \quad & \Diamond(f = w = g = c = 1) \wedge \\ & \Box(w \neq g \vee f = g) \wedge \\ & \Box(g \neq c \vee f = g)\end{aligned}$$

A solution:



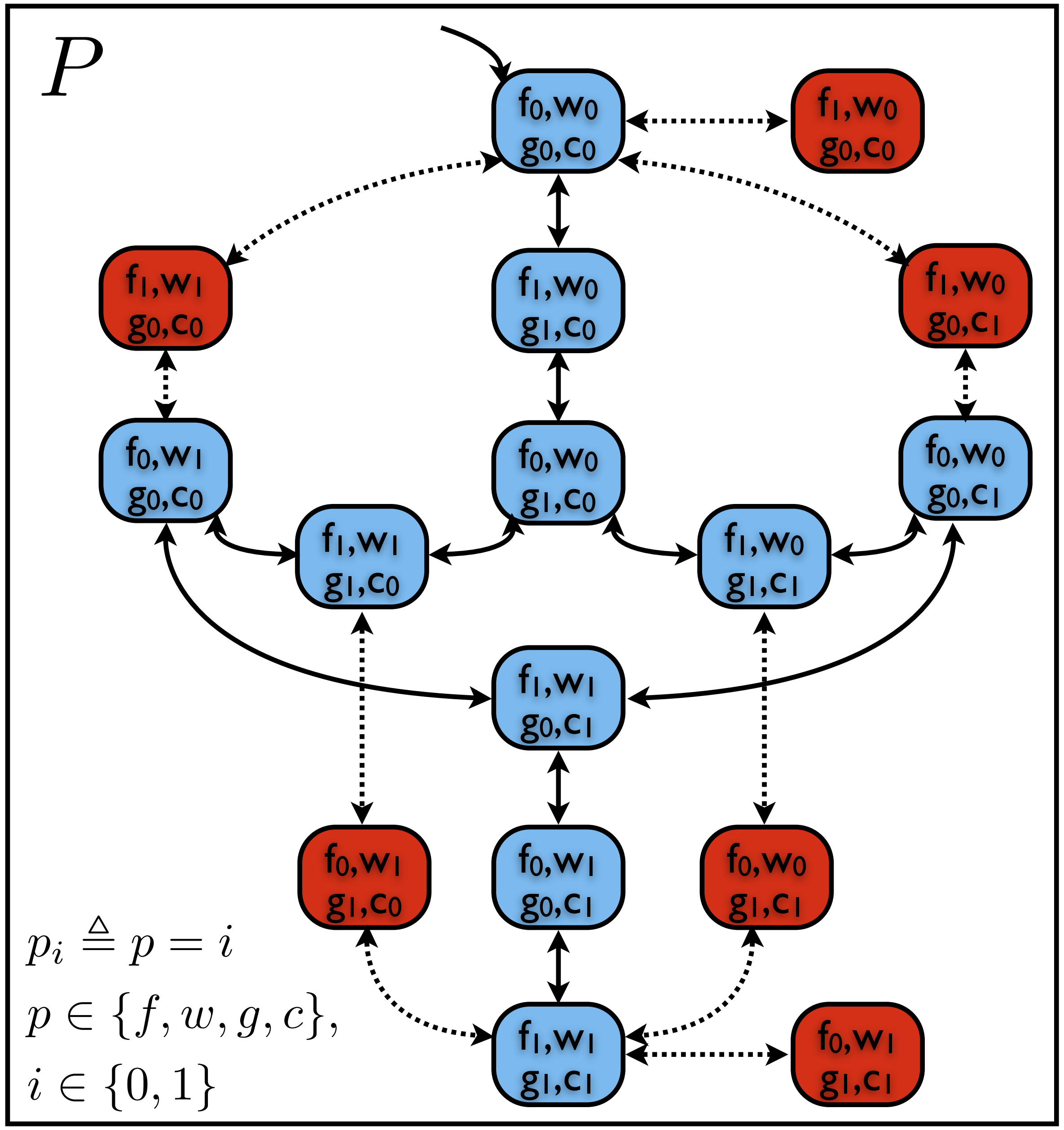
Alternative solution

A farmer wants to cross a river in a little boat with a wolf, a goat and a cabbage.

Constraints:

- The boat is only big enough to carry the farmer plus one other animal or object.
- The wolf will eat the goat if the farmer is not present.
- The goat will eat the cabbage if the farmer is not present.

How can the farmer get both animals and the cabbage safely across the river?



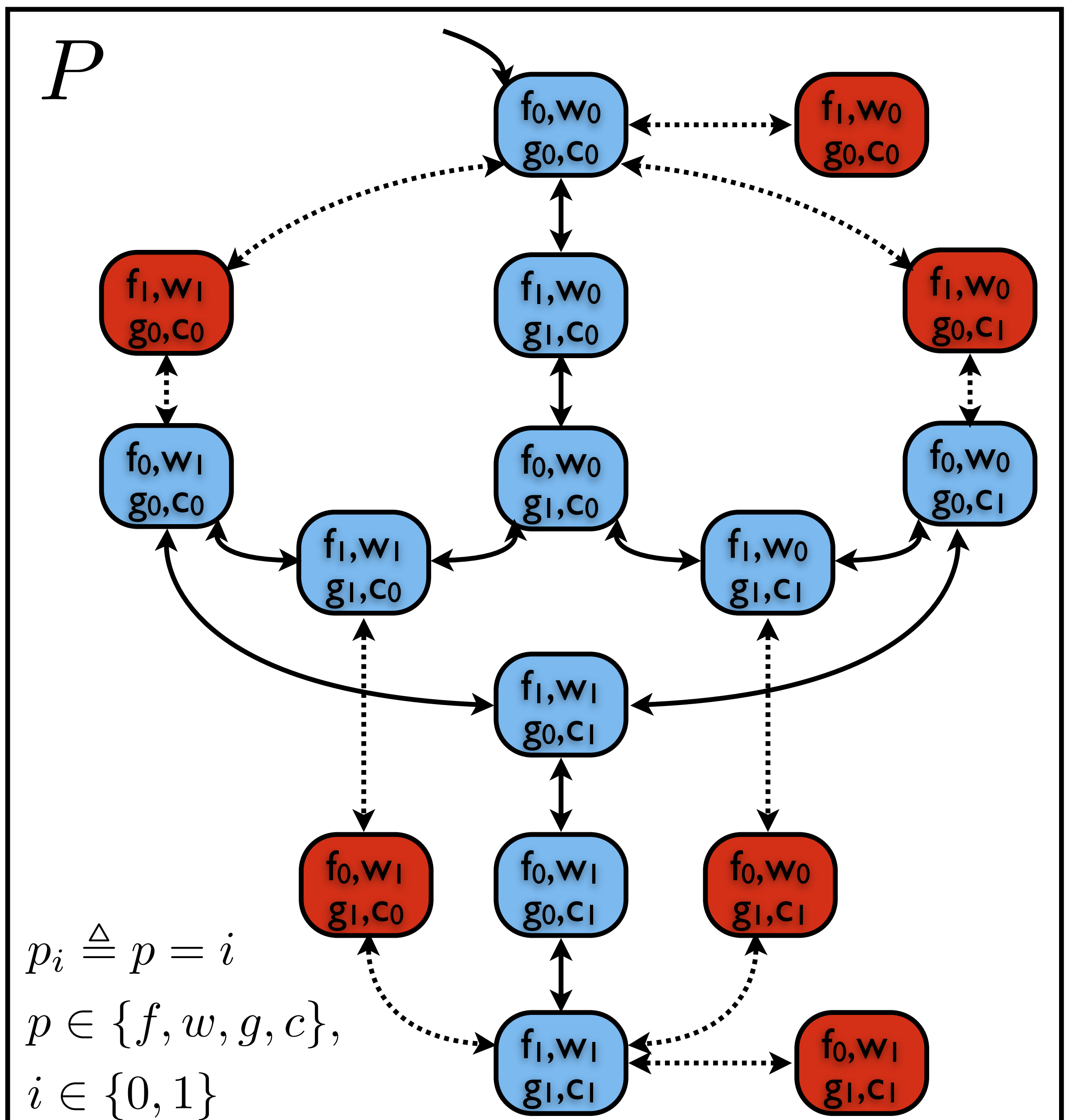
Alternative solution

A farmer wants to cross a river in a little boat with a wolf, a goat and a cabbage.

Constraints:

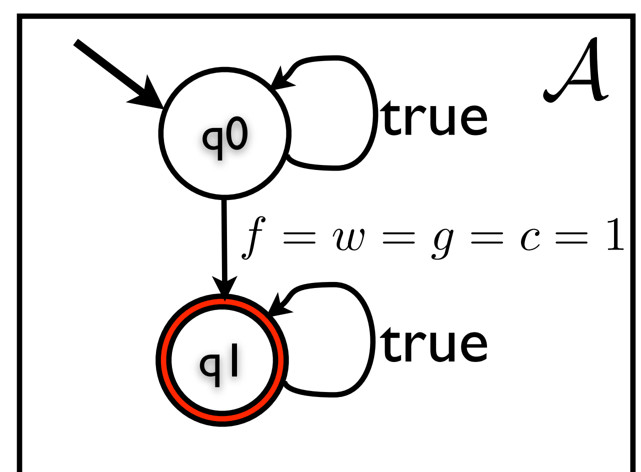
- The boat is only big enough to carry the farmer plus one other animal or object.
- The wolf will eat the goat if the farmer is not present.
- The goat will eat the cabbage if the farmer is not present.

How can the farmer get both animals and the cabbage safely across the river?



$$\Phi \quad = \quad \Diamond(f = w = g = c = 1)$$

$$\mathcal{L}_\omega(\mathcal{A}) = Words(\Phi)$$



Alternative solution

A farmer wants to cross a river in a little boat with a wolf, a goat and a cabbage.

Constraints:

- The boat is only big enough to carry the farmer plus one other animal or object.
- The wolf will eat the goat if the farmer is not present.
- The goat will eat the cabbage if the farmer is not present.

System model in SPIN:

```
active proctype P() {
  do
  :: atomic{ (g!=c && g!=w) -> f=1-f }
  :: atomic{ f==g -> f=1-f; g=1-g }
  :: atomic{ (f==w && g!=c) -> f=1-f; w=1-w }
  :: atomic{ (f==c && g!=w) -> f=1-f; c=1-c }
  od
}
```

farmer can cross only when goat and cabbage are not at the same place and goat and wolf are not

farmer and goat can cross only when they are at the same place

farmer and wolf can cross only when they are at the same place and goat and cabbage are not

farmer and cabbage can cross only when they are at the same place and goat and wolf are not

Specification:

$$\Phi \quad = \quad \Diamond(f = w = g = c = 1)$$

Alternative solution

A farmer wants to cross a river in a little boat with a wolf, a goat and a cabbage.

Constraints:

- The boat is only big enough to carry the farmer plus one other animal or object.
- The wolf will eat the goat if the farmer is not present.
- The goat will eat the cabbage if the farmer is not present.

System model in SPIN:

```
active proctype P() {
  do
  :: atomic{ (g!=c && g!=w) -> f=1-f }
  :: atomic{ f==g -> f=1-f; g=1-g }
  :: atomic{ (f==w && g!=c) -> f=1-f; w=1-w }
  :: atomic{ (f==c && g!=w) -> f=1-f; c=1-c }
  od
}
```

farmer can cross only when goat and cabbage are not at the same place and goat and wolf are not

farmer and goat can cross only when they are at the same place

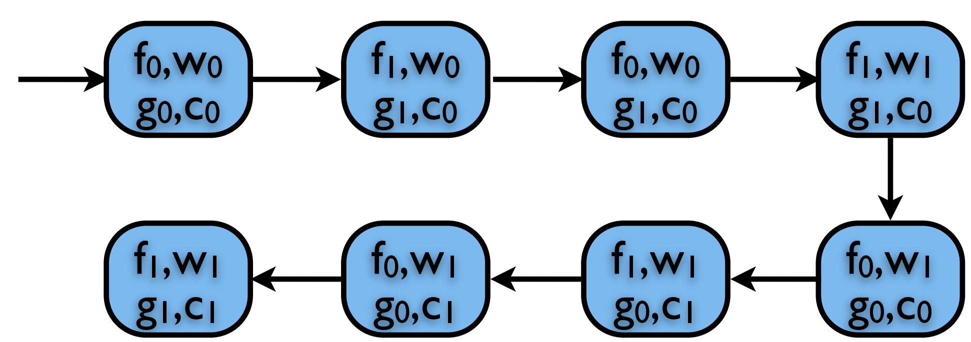
farmer and wolf can cross only when they are at the same place and goat and cabbage are not

farmer and cabbage can cross only when they are at the same place and goat and wolf are not

Specification:

$$\Phi \quad = \quad \Diamond(f = w = g = c = 1)$$

Another solution:

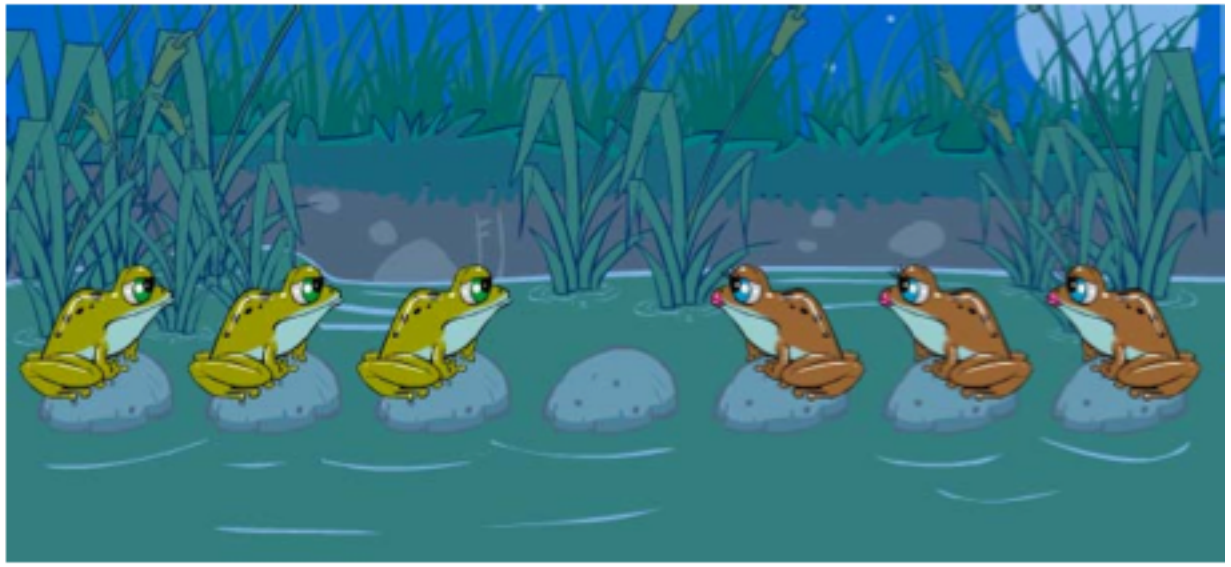


Example: frog puzzle

Find a way to send all the yellow frogs to the right hand side of the pond and send all the brown frogs to the left hand side.

Constraints:

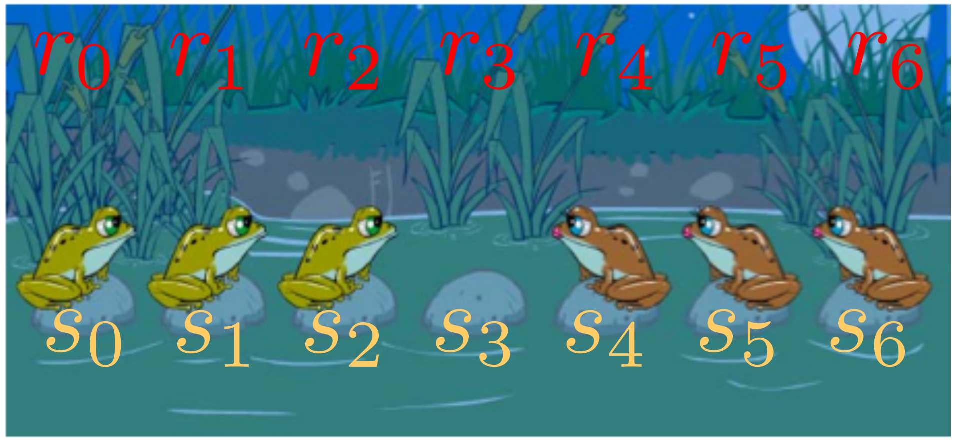
- Frogs can only jump in the direction they are facing.
- Frogs can either jump one rock forward if the next rock is empty or they can jump over a frog if the next rock has a frog on it and the rock after it is empty.

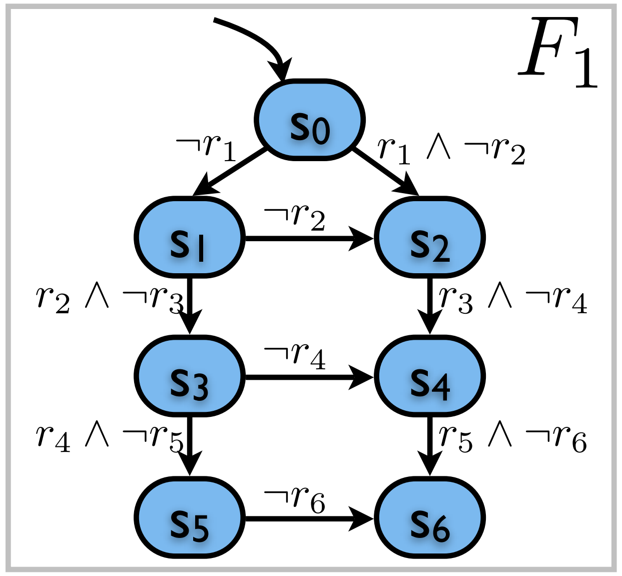


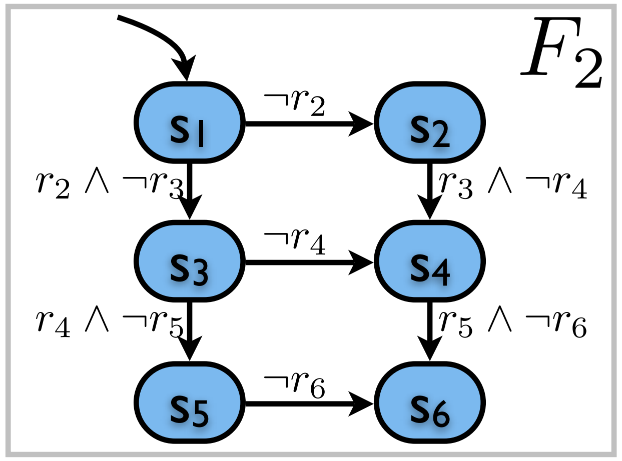
http://www.hellam.net/maths2000/frogs.html

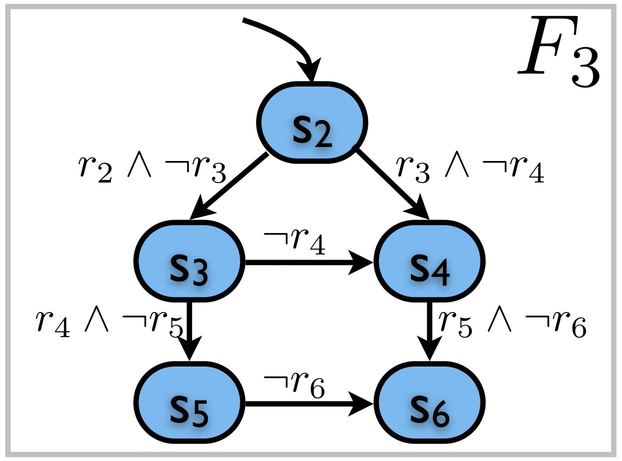
Solving the frog puzzle as logic synthesis

- Rock *i* is not occupied or occupied $r_i \in \{0, 1\}$
- State of frog *i*: $s(F_i) \in \{s_0, s_1 \dots, s_6\}$
- Transition system of frog *i*: F_i
- Overall system model: $P = F_1 \parallel F_2 \parallel \cdots \parallel F_6$



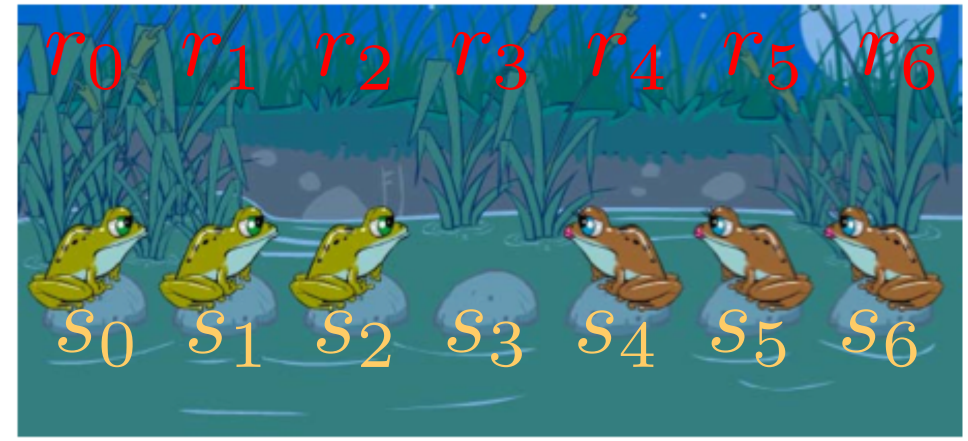


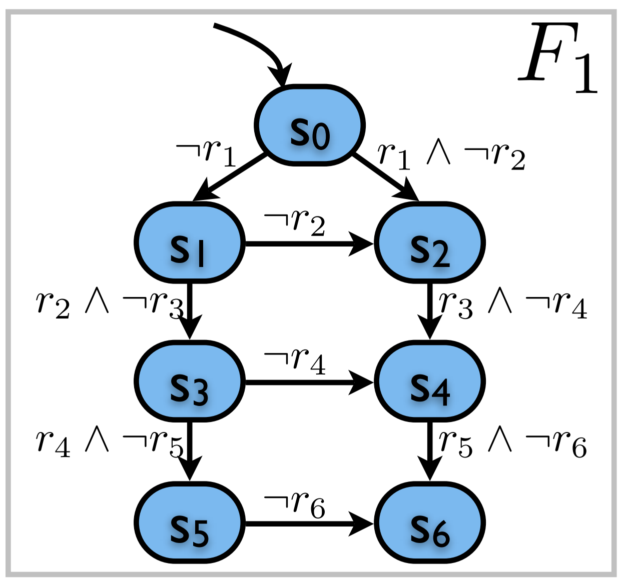


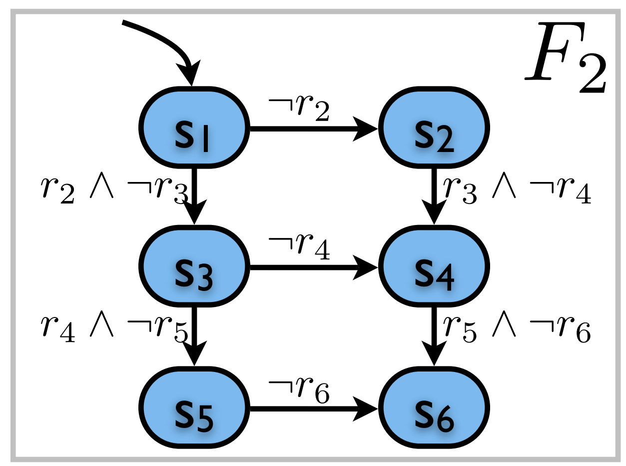


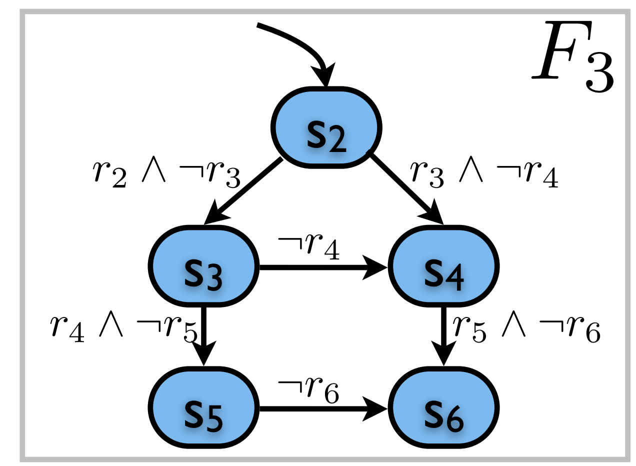
Solving the frog puzzle as logic synthesis

- Rock *i* is not occupied or occupied $r_i \in \{0, 1\}$
- State of frog *i*: $s(F_i) \in \{s_0, s_1 \ldots, s_6\}$
- Transition system of frog *i*: F_i
- Overall system model: $P = F_1 \parallel F_2 \parallel \cdots \parallel F_6$

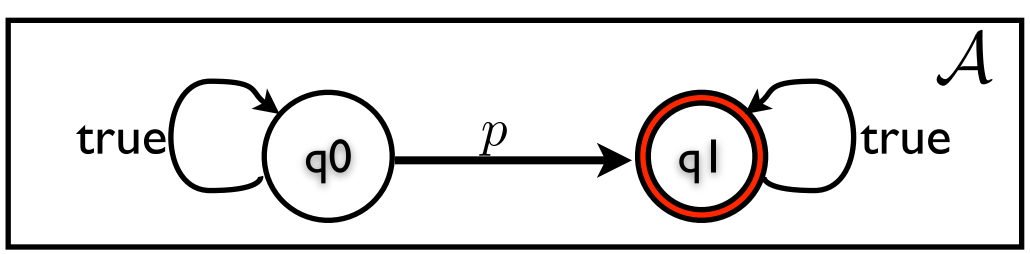








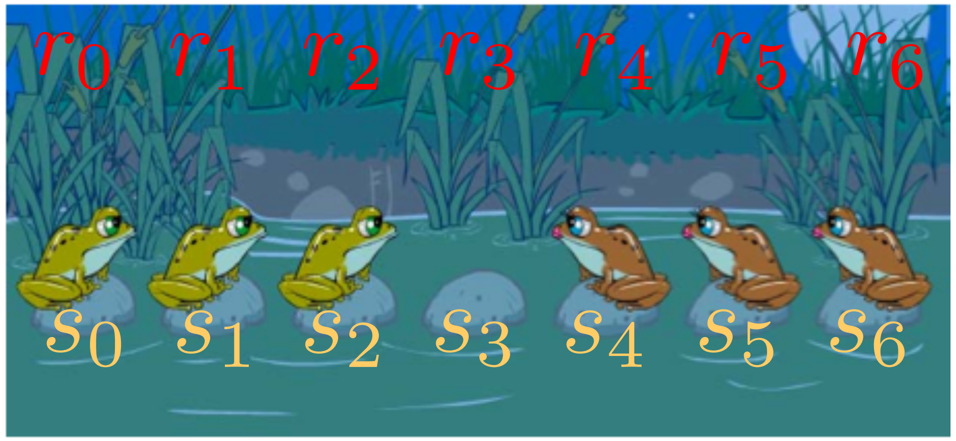
$$\Phi = \Diamond\big(s(F_1), s(F_2), s(F_3) \in \{s_4, s_5, s_6\} \wedge s(F_4), s(F_5), s(F_6) \in \{s_0, s_1, s_2\}\big)$$

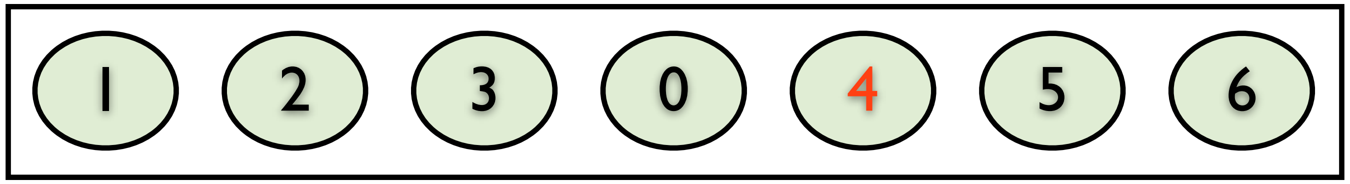


$$p \triangleq \big(s(F_1), s(F_2), s(F_3) \in \{s_4, s_5, s_6\} \wedge s(F_4), s(F_5), s(F_6) \in \{s_0, s_1, s_2\}\big)$$

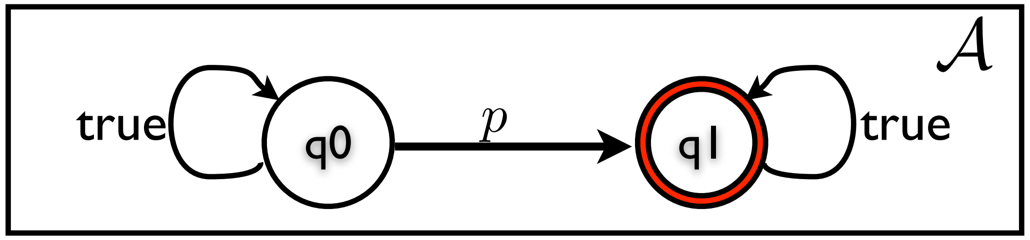
Solving the frog puzzle as logic synthesis

- Rock *i* is not occupied or occupied $r_i \in \{0, 1\}$
- State of frog *i*: $s(F_i) \in \{s_0, s_1 \ldots, s_6\}$
- Transition system of frog *i*: F_i
- Overall system model: $P = F_1 \parallel F_2 \parallel \cdots \parallel F_6$





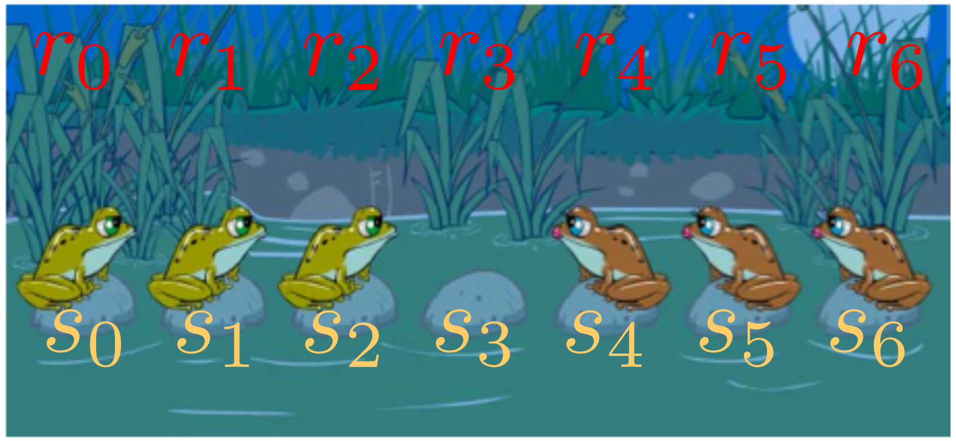
$$\Phi = \Diamond\big(s(F_1), s(F_2), s(F_3) \in \{s_4, s_5, s_6\} \wedge s(F_4), s(F_5), s(F_6) \in \{s_0, s_1, s_2\}\big)$$

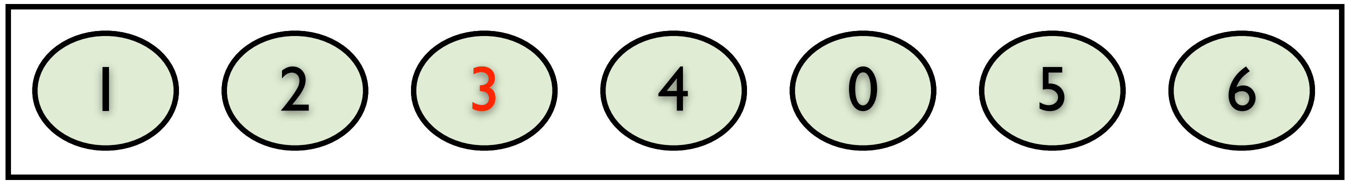


$$p \triangleq \big(s(F_1), s(F_2), s(F_3) \in \{s_4, s_5, s_6\} \wedge s(F_4), s(F_5), s(F_6) \in \{s_0, s_1, s_2\}\big)$$

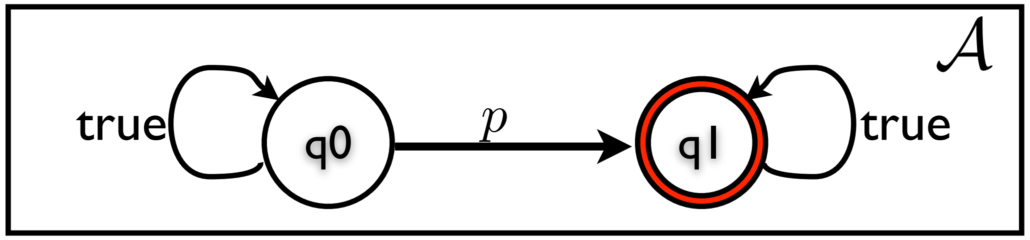
Solving the frog puzzle as logic synthesis

- Rock *i* is not occupied or occupied $r_i \in \{0, 1\}$
- State of frog *i*: $s(F_i) \in \{s_0, s_1 \ldots, s_6\}$
- Transition system of frog *i*: F_i
- Overall system model: $P = F_1 \parallel F_2 \parallel \cdots \parallel F_6$





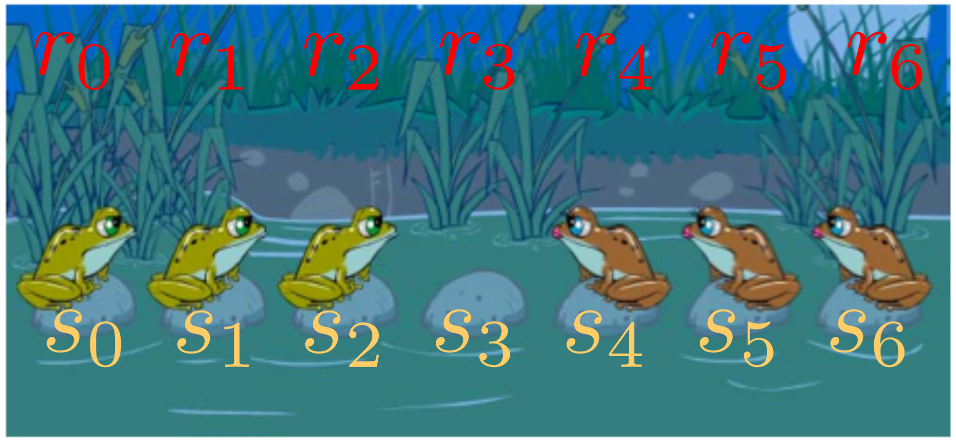
$$\Phi = \Diamond\big(s(F_1), s(F_2), s(F_3) \in \{s_4, s_5, s_6\} \wedge s(F_4), s(F_5), s(F_6) \in \{s_0, s_1, s_2\}\big)$$

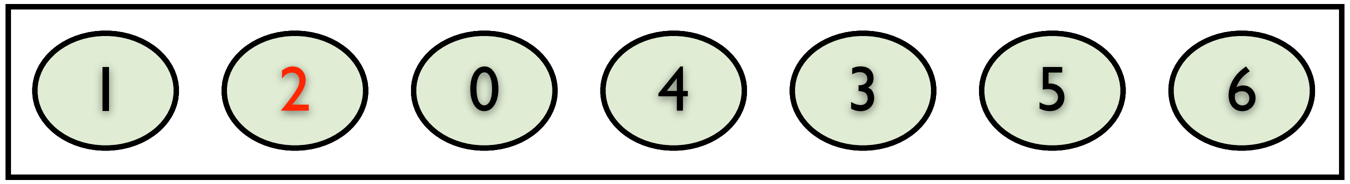


$$p \triangleq \big(s(F_1), s(F_2), s(F_3) \in \{s_4, s_5, s_6\} \wedge s(F_4), s(F_5), s(F_6) \in \{s_0, s_1, s_2\}\big)$$

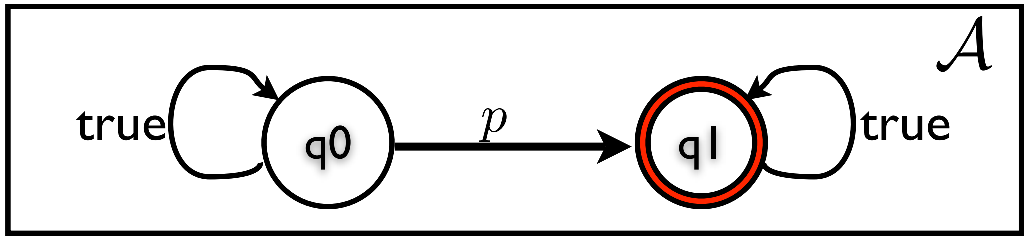
Solving the frog puzzle as logic synthesis

- Rock *i* is not occupied or occupied $r_i \in \{0, 1\}$
- State of frog *i*: $s(F_i) \in \{s_0, s_1 \dots, s_6\}$
- Transition system of frog *i*: F_i
- Overall system model: $P = F_1 \parallel F_2 \parallel \cdots \parallel F_6$





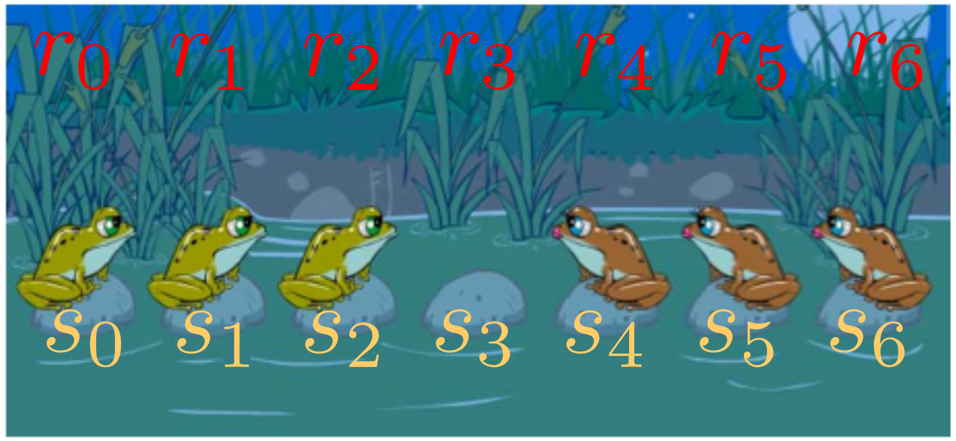
$$\Phi = \Diamond\big(s(F_1), s(F_2), s(F_3) \in \{s_4, s_5, s_6\} \wedge s(F_4), s(F_5), s(F_6) \in \{s_0, s_1, s_2\}\big)$$

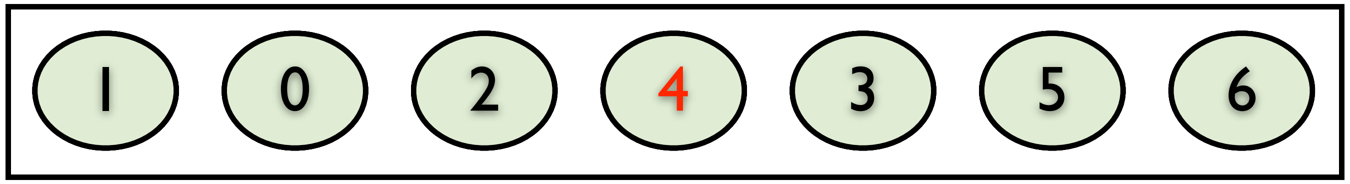


$$p \triangleq \big(s(F_1), s(F_2), s(F_3) \in \{s_4, s_5, s_6\} \wedge s(F_4), s(F_5), s(F_6) \in \{s_0, s_1, s_2\}\big)$$

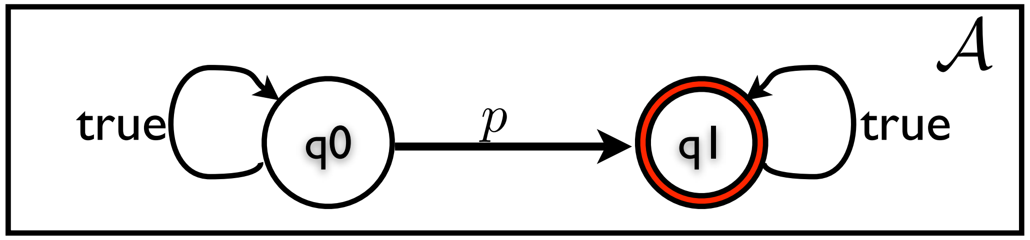
Solving the frog puzzle as logic synthesis

- Rock *i* is not occupied or occupied $r_i \in \{0, 1\}$
- State of frog *i*: $s(F_i) \in \{s_0, s_1 \ldots, s_6\}$
- Transition system of frog *i*: F_i
- Overall system model: $P = F_1 \parallel F_2 \parallel \cdots \parallel F_6$





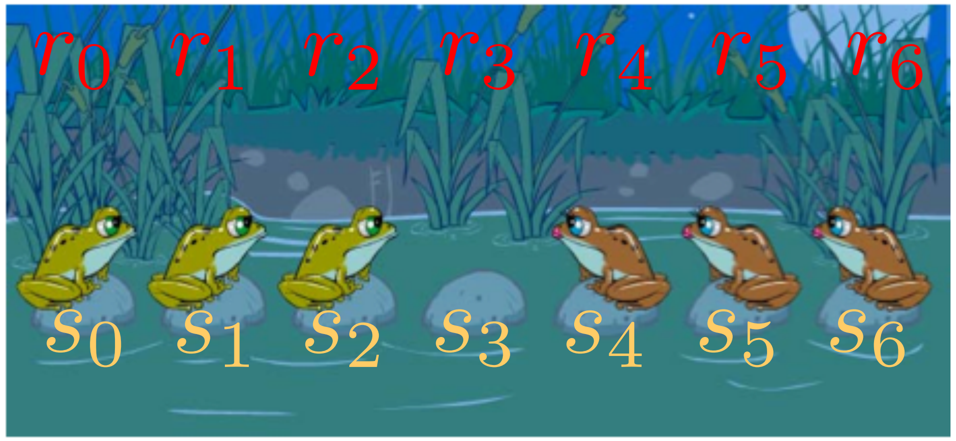
$$\Phi = \Diamond\big(s(F_1), s(F_2), s(F_3) \in \{s_4, s_5, s_6\} \wedge s(F_4), s(F_5), s(F_6) \in \{s_0, s_1, s_2\}\big)$$

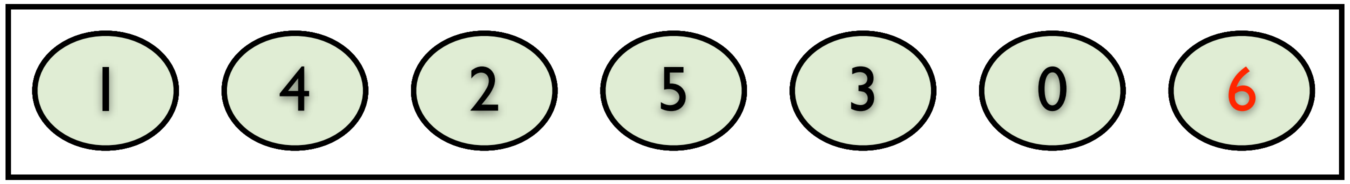


$$p \triangleq \big(s(F_1), s(F_2), s(F_3) \in \{s_4, s_5, s_6\} \wedge s(F_4), s(F_5), s(F_6) \in \{s_0, s_1, s_2\}\big)$$

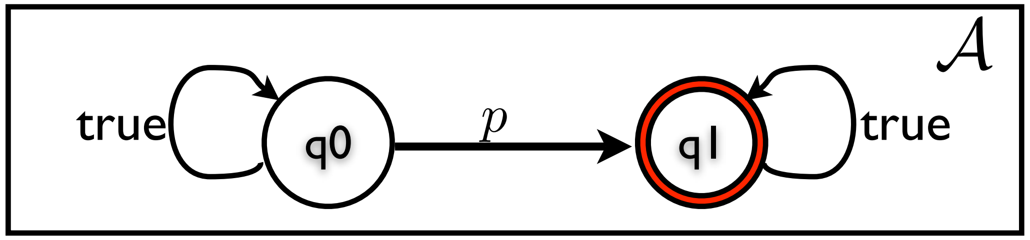
Solving the frog puzzle as logic synthesis

- Rock i is not occupied or occupied $r_i \in \{0, 1\}$
- State of frog i: $s(F_i) \in \{s_0, s_1 \ldots, s_6\}$
- Transition system of frog i: F_i
- Overall system model: $P = F_1 \parallel F_2 \parallel \cdots \parallel F_6$





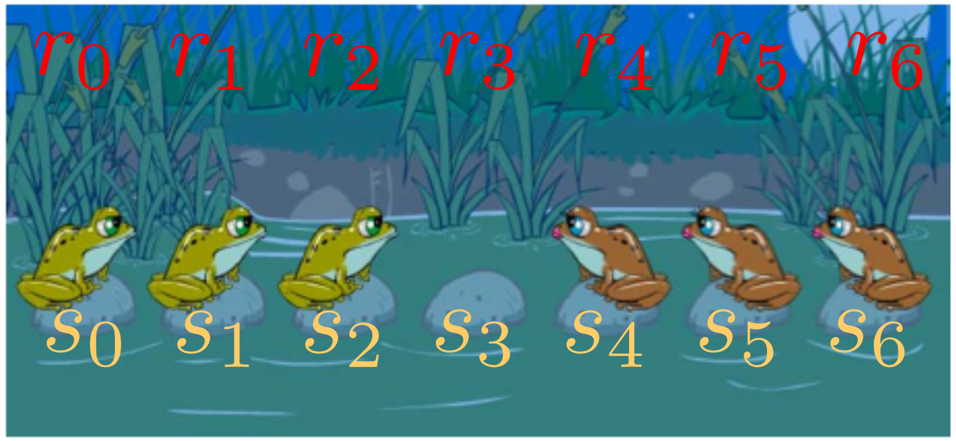
$$\Phi = \Diamond\big(s(F_1), s(F_2), s(F_3) \in \{s_4, s_5, s_6\} \wedge s(F_4), s(F_5), s(F_6) \in \{s_0, s_1, s_2\}\big)$$



$$p \triangleq \big(s(F_1), s(F_2), s(F_3) \in \{s_4, s_5, s_6\} \wedge s(F_4), s(F_5), s(F_6) \in \{s_0, s_1, s_2\}\big)$$

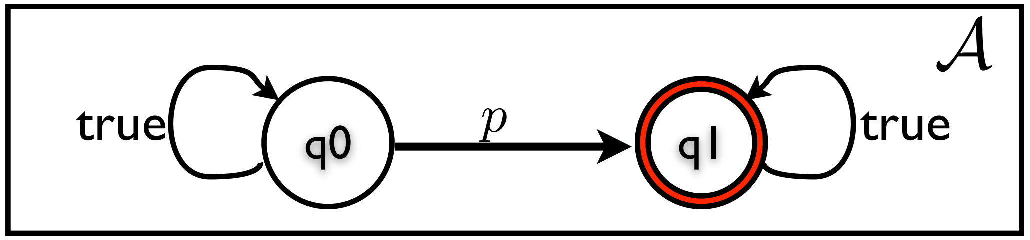
Solving the frog puzzle as logic synthesis

- Rock *i* is not occupied or occupied $r_i \in \{0,1\}$
- State of frog *i*: $s(F_i) \in \{s_0, s_1 \ldots, s_6\}$
- Transition system of frog *i*: F_i
- Overall system model: $P = F_1 \parallel F_2 \parallel \cdots \parallel F_6$



1 4 2 5 3 6 0

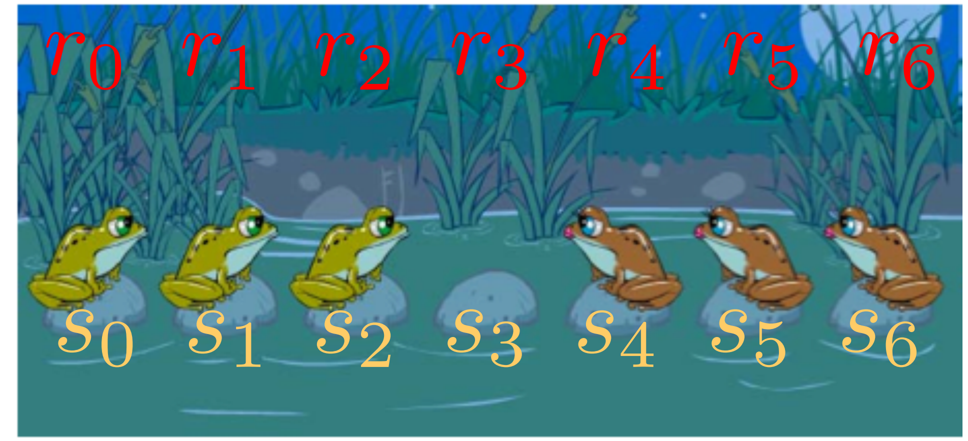
$$\Phi = \Diamond\big(s(F_1), s(F_2), s(F_3) \in \{s_4, s_5, s_6\} \wedge s(F_4), s(F_5), s(F_6) \in \{s_0, s_1, s_2\}\big)$$

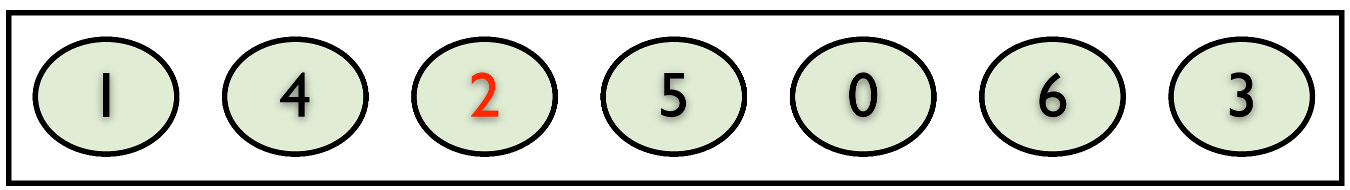


$$p \triangleq \big(s(F_1), s(F_2), s(F_3) \in \{s_4, s_5, s_6\} \wedge s(F_4), s(F_5), s(F_6) \in \{s_0, s_1, s_2\}\big)$$

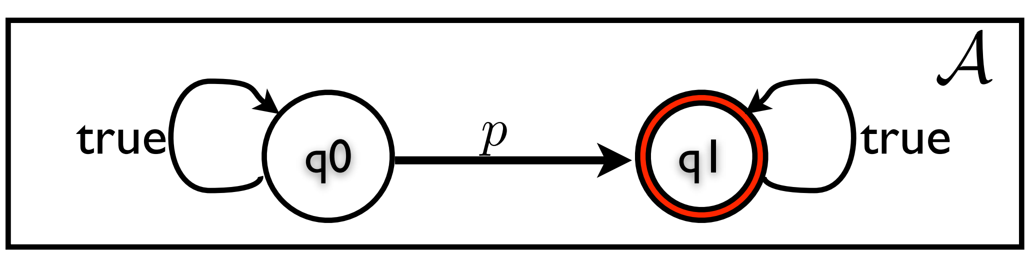
Solving the frog puzzle as logic synthesis

- Rock *i* is not occupied or occupied $r_i \in \{0,1\}$
- State of frog *i*: $s(F_i) \in \{s_0, s_1 \ldots, s_6\}$
- Transition system of frog *i*: F_i
- Overall system model: $P = F_1 \parallel F_2 \parallel \cdots \parallel F_6$





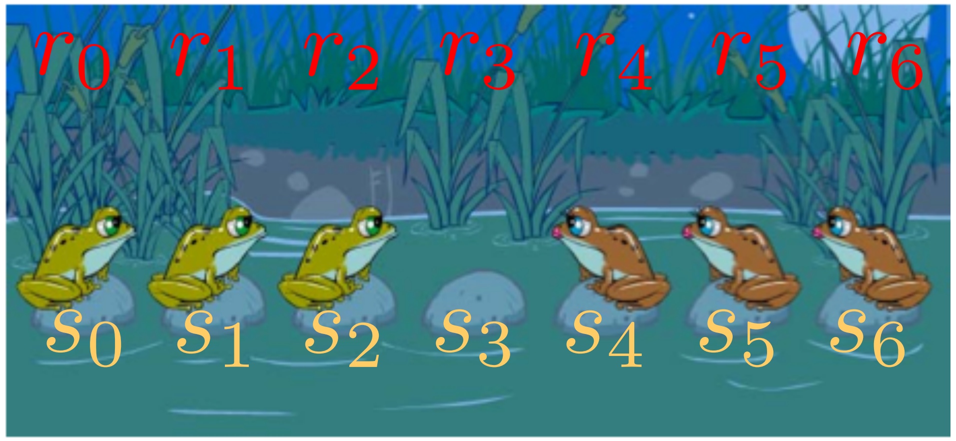
$$\Phi = \Diamond\big(s(F_1), s(F_2), s(F_3) \in \{s_4, s_5, s_6\} \wedge s(F_4), s(F_5), s(F_6) \in \{s_0, s_1, s_2\}\big)$$

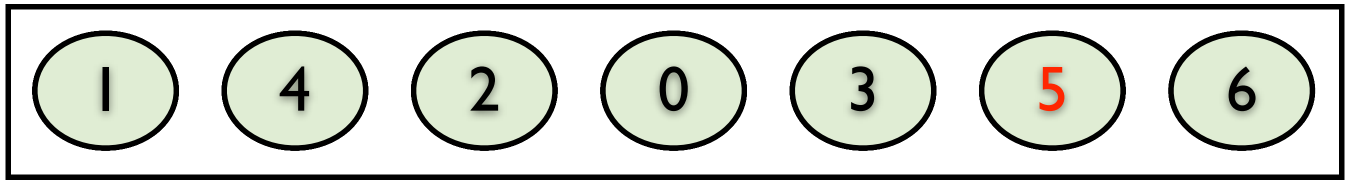


$$p \triangleq \big(s(F_1), s(F_2), s(F_3) \in \{s_4, s_5, s_6\} \wedge s(F_4), s(F_5), s(F_6) \in \{s_0, s_1, s_2\}\big)$$

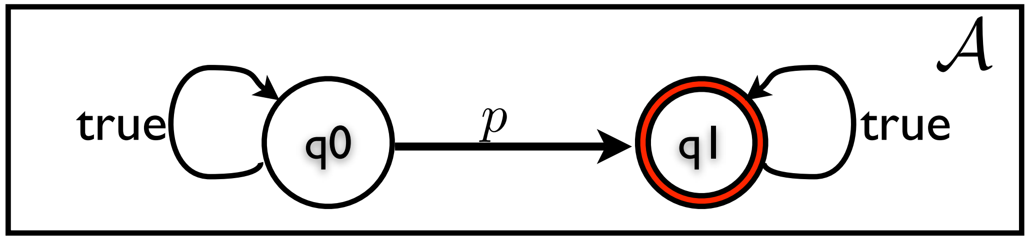
Solving the frog puzzle as logic synthesis

- Rock *i* is not occupied or occupied $r_i \in \{0,1\}$
- State of frog *i*: $s(F_i) \in \{s_0, s_1 \ldots, s_6\}$
- Transition system of frog *i*: F_i
- Overall system model: $P = F_1 \parallel F_2 \parallel \cdots \parallel F_6$





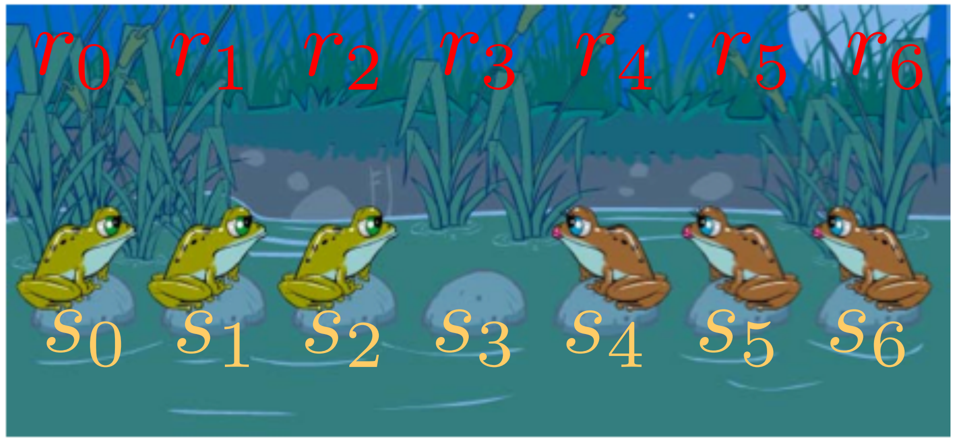
$$\Phi = \Diamond\big(s(F_1), s(F_2), s(F_3) \in \{s_4, s_5, s_6\} \wedge s(F_4), s(F_5), s(F_6) \in \{s_0, s_1, s_2\}\big)$$

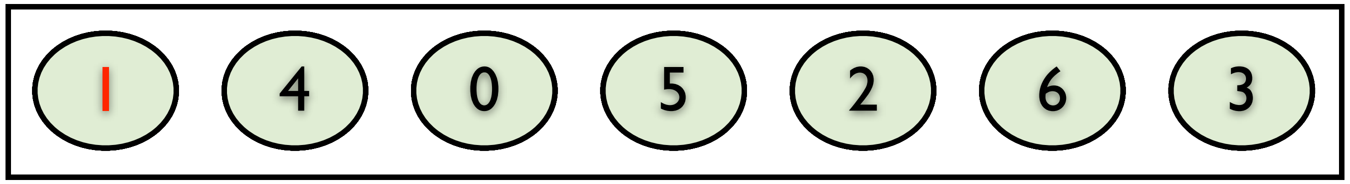


$$p \triangleq \big(s(F_1), s(F_2), s(F_3) \in \{s_4, s_5, s_6\} \wedge s(F_4), s(F_5), s(F_6) \in \{s_0, s_1, s_2\}\big)$$

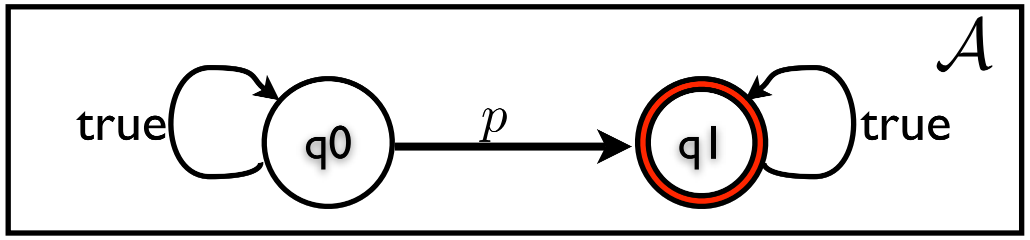
Solving the frog puzzle as logic synthesis

- Rock *i* is not occupied or occupied $r_i \in \{0, 1\}$
- State of frog *i*: $s(F_i) \in \{s_0, s_1 \ldots, s_6\}$
- Transition system of frog *i*: F_i
- Overall system model: $P = F_1 \parallel F_2 \parallel \cdots \parallel F_6$





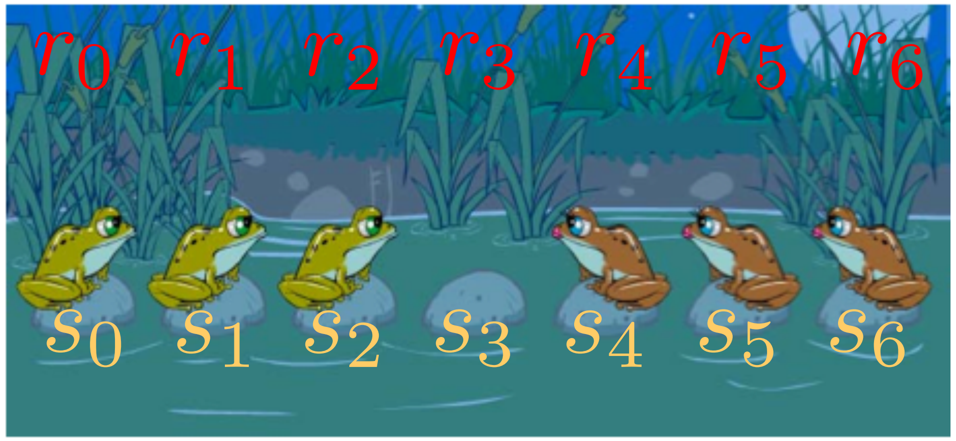
$$\Phi = \Diamond\big(s(F_1), s(F_2), s(F_3) \in \{s_4, s_5, s_6\} \wedge s(F_4), s(F_5), s(F_6) \in \{s_0, s_1, s_2\}\big)$$

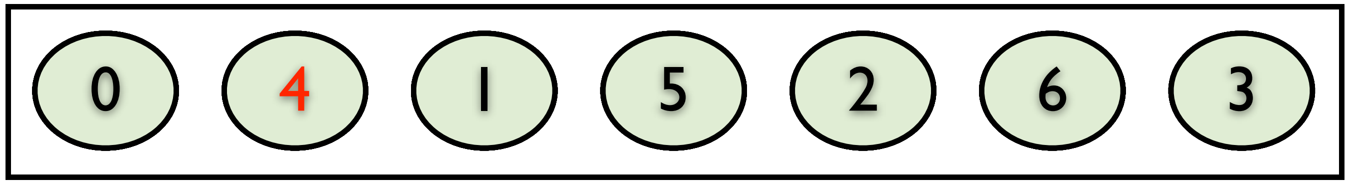


$$p \triangleq \big(s(F_1), s(F_2), s(F_3) \in \{s_4, s_5, s_6\} \wedge s(F_4), s(F_5), s(F_6) \in \{s_0, s_1, s_2\}\big)$$

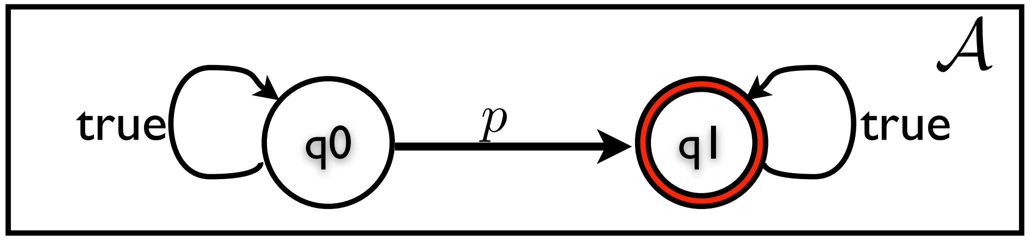
Solving the frog puzzle as logic synthesis

- Rock *i* is not occupied or occupied $r_i \in \{0,1\}$
- State of frog *i*: $s(F_i) \in \{s_0, s_1 \ldots, s_6\}$
- Transition system of frog *i*: F_i
- Overall system model: $P = F_1 \parallel F_2 \parallel \cdots \parallel F_6$





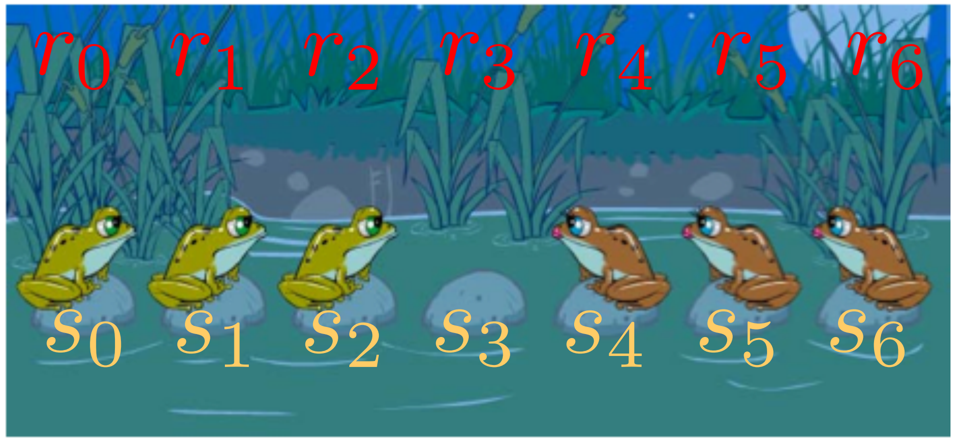
$$\Phi = \Diamond\big(s(F_1), s(F_2), s(F_3) \in \{s_4, s_5, s_6\} \wedge s(F_4), s(F_5), s(F_6) \in \{s_0, s_1, s_2\}\big)$$

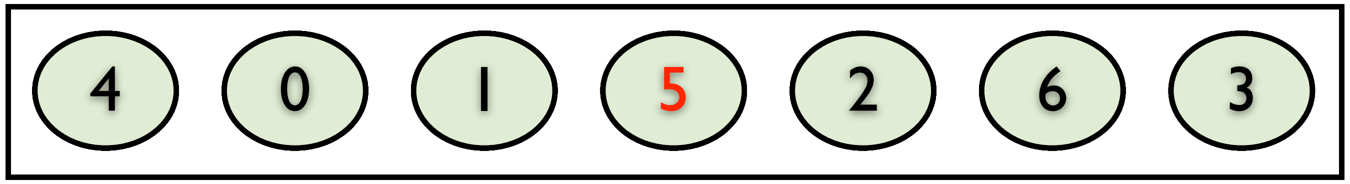


$$p \triangleq \big(s(F_1), s(F_2), s(F_3) \in \{s_4, s_5, s_6\} \wedge s(F_4), s(F_5), s(F_6) \in \{s_0, s_1, s_2\}\big)$$

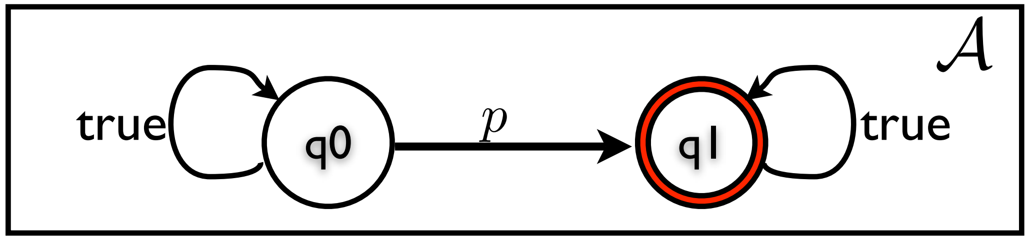
Solving the frog puzzle as logic synthesis

- Rock i is not occupied or occupied $r_i \in \{0,1\}$
- State of frog i: $s(F_i) \in \{s_0, s_1 \ldots, s_6\}$
- Transition system of frog i: F_i
- Overall system model: $P = F_1 \parallel F_2 \parallel \cdots \parallel F_6$





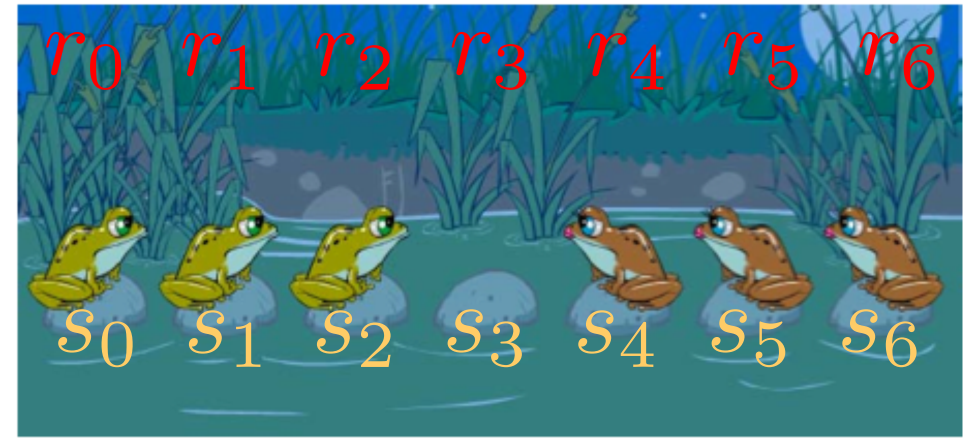
$$\Phi = \Diamond\big(s(F_1), s(F_2), s(F_3) \in \{s_4, s_5, s_6\} \wedge s(F_4), s(F_5), s(F_6) \in \{s_0, s_1, s_2\}\big)$$

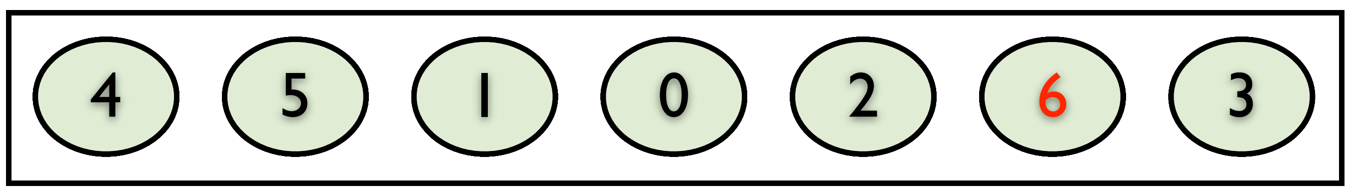


$$p \triangleq \big(s(F_1), s(F_2), s(F_3) \in \{s_4, s_5, s_6\} \wedge s(F_4), s(F_5), s(F_6) \in \{s_0, s_1, s_2\}\big)$$

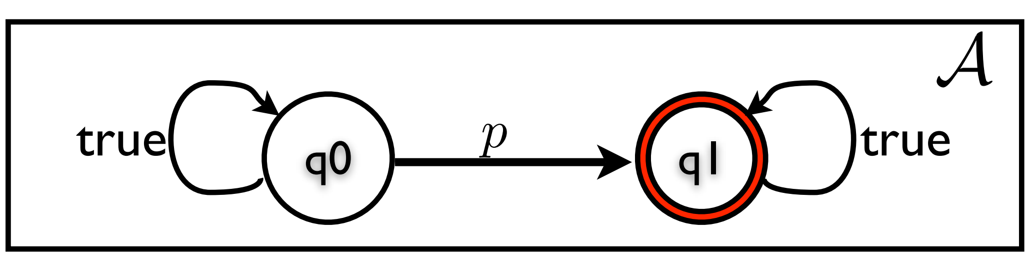
Solving the frog puzzle as logic synthesis

- Rock *i* is not occupied or occupied $r_i \in \{0,1\}$
- State of frog *i*: $s(F_i) \in \{s_0, s_1 \ldots, s_6\}$
- Transition system of frog *i*: F_i
- Overall system model: $P = F_1 \parallel F_2 \parallel \cdots \parallel F_6$





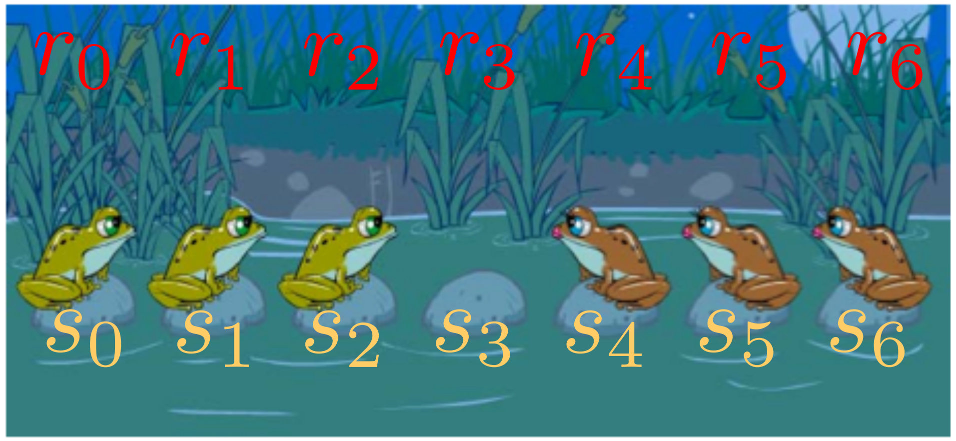
$$\Phi = \Diamond\big(s(F_1), s(F_2), s(F_3) \in \{s_4, s_5, s_6\} \wedge s(F_4), s(F_5), s(F_6) \in \{s_0, s_1, s_2\}\big)$$

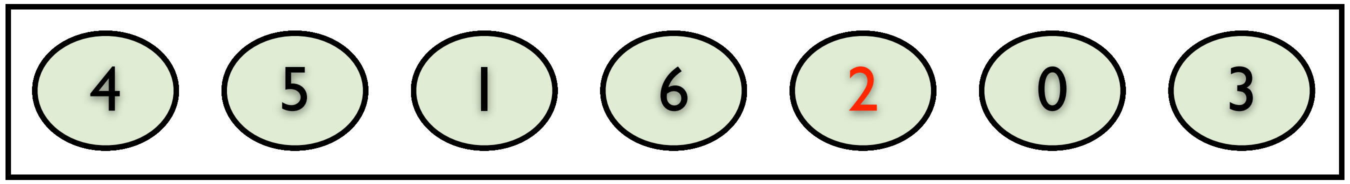


$$p \triangleq \big(s(F_1), s(F_2), s(F_3) \in \{s_4, s_5, s_6\} \wedge s(F_4), s(F_5), s(F_6) \in \{s_0, s_1, s_2\}\big)$$

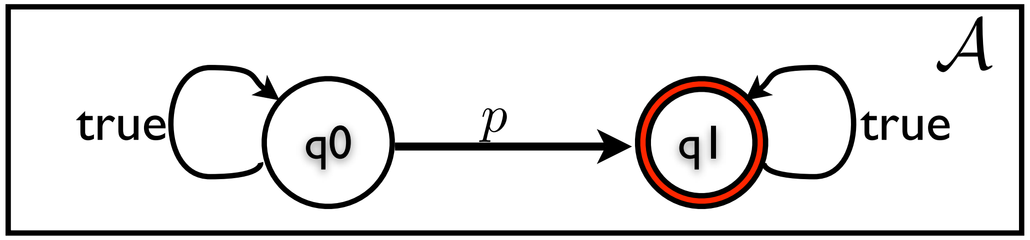
Solving the frog puzzle as logic synthesis

- Rock *i* is not occupied or occupied $r_i \in \{0, 1\}$
- State of frog *i*: $s(F_i) \in \{s_0, s_1 \ldots, s_6\}$
- Transition system of frog *i*: F_i
- Overall system model: $P = F_1 \parallel F_2 \parallel \cdots \parallel F_6$





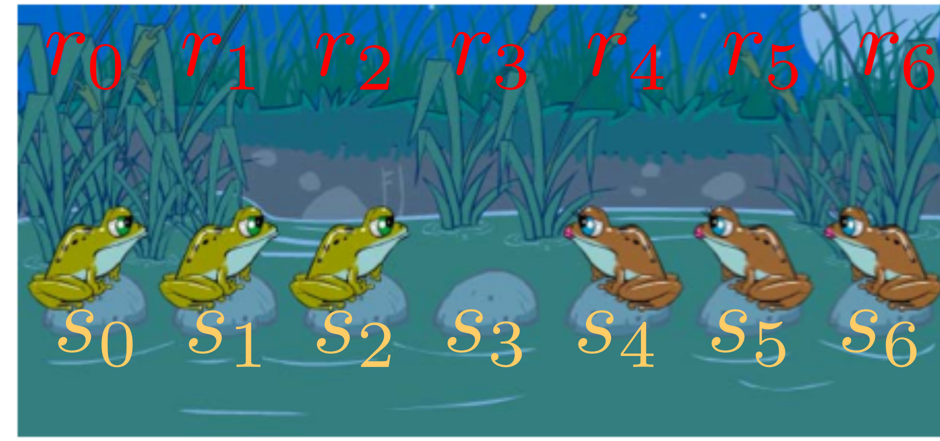
$$\Phi = \Diamond\big(s(F_1), s(F_2), s(F_3) \in \{s_4, s_5, s_6\} \wedge s(F_4), s(F_5), s(F_6) \in \{s_0, s_1, s_2\}\big)$$

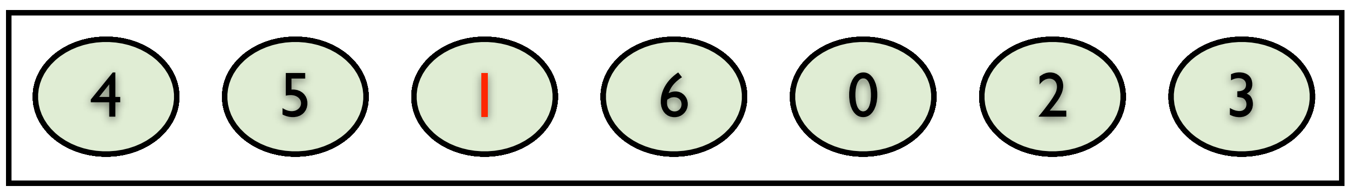


$$p \triangleq \big(s(F_1), s(F_2), s(F_3) \in \{s_4, s_5, s_6\} \wedge s(F_4), s(F_5), s(F_6) \in \{s_0, s_1, s_2\}\big)$$

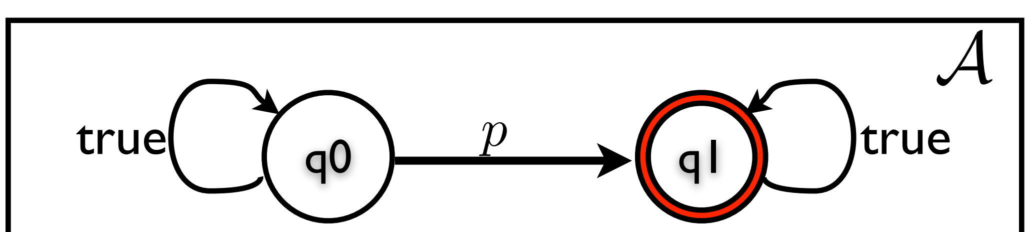
Solving the frog puzzle as logic synthesis

- Rock *i* is not occupied or occupied $r_i \in \{0, 1\}$
- State of frog *i*: $s(F_i) \in \{s_0, s_1 \ldots, s_6\}$
- Transition system of frog *i*: F_i
- Overall system model: $P = F_1 \parallel F_2 \parallel \cdots \parallel F_6$





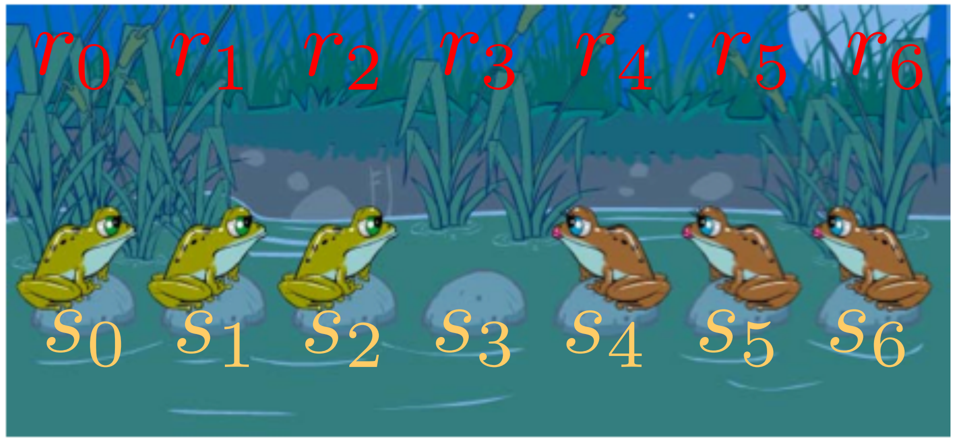
$$\Phi = \Diamond\big(s(F_1), s(F_2), s(F_3) \in \{s_4, s_5, s_6\} \wedge s(F_4), s(F_5), s(F_6) \in \{s_0, s_1, s_2\}\big)$$

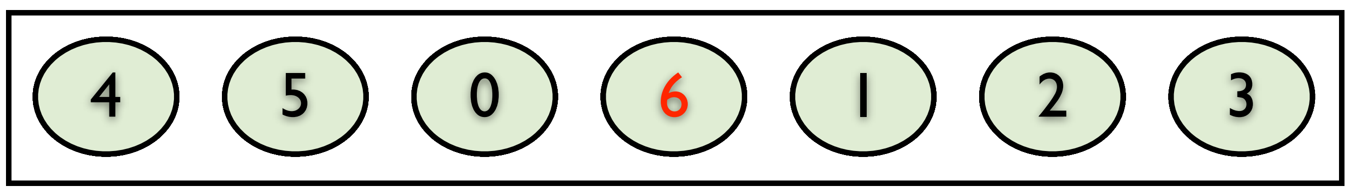


$$p \triangleq \big(s(F_1), s(F_2), s(F_3) \in \{s_4, s_5, s_6\} \wedge s(F_4), s(F_5), s(F_6) \in \{s_0, s_1, s_2\}\big)$$

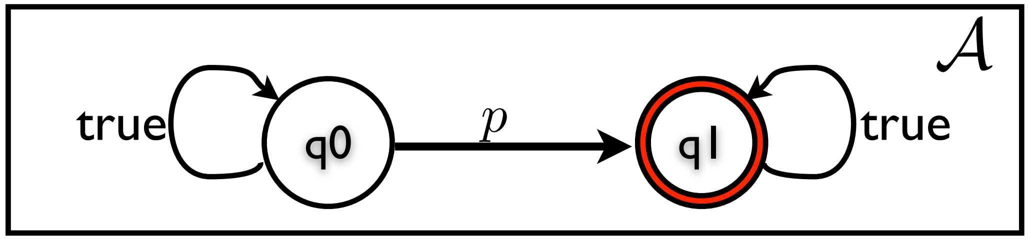
Solving the frog puzzle as logic synthesis

- Rock *i* is not occupied or occupied $r_i \in \{0, 1\}$
- State of frog *i*: $s(F_i) \in \{s_0, s_1 \ldots, s_6\}$
- Transition system of frog *i*: F_i
- Overall system model: $P = F_1 \parallel F_2 \parallel \cdots \parallel F_6$





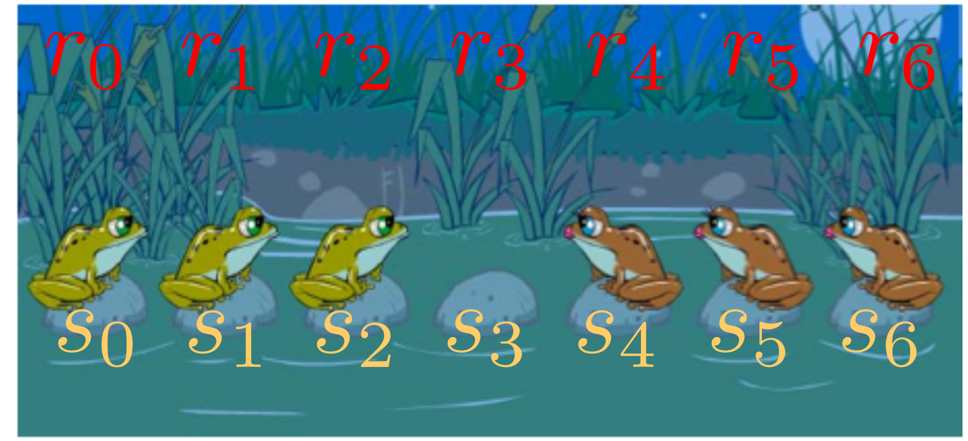
$$\Phi = \Diamond\big(s(F_1), s(F_2), s(F_3) \in \{s_4, s_5, s_6\} \wedge s(F_4), s(F_5), s(F_6) \in \{s_0, s_1, s_2\}\big)$$

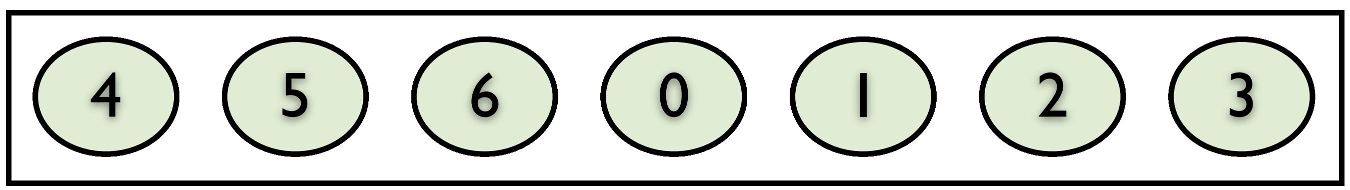


$$p \triangleq \big(s(F_1), s(F_2), s(F_3) \in \{s_4, s_5, s_6\} \wedge s(F_4), s(F_5), s(F_6) \in \{s_0, s_1, s_2\}\big)$$

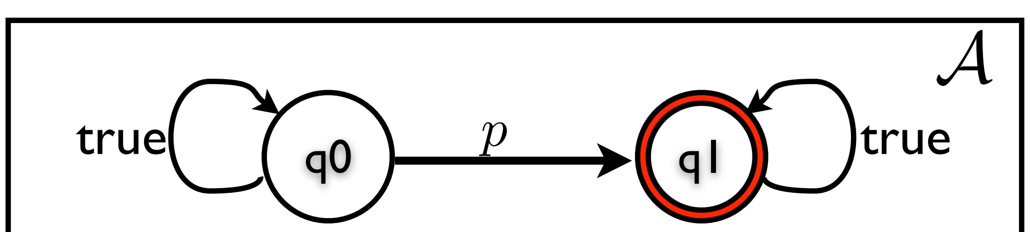
Solving the frog puzzle as logic synthesis

- Rock *i* is not occupied or occupied $r_i \in \{0, 1\}$
- State of frog *i*: $s(F_i) \in \{s_0, s_1 \ldots, s_6\}$
- Transition system of frog *i*: F_i
- Overall system model: $P = F_1 \parallel F_2 \parallel \cdots \parallel F_6$





$$\Phi = \Diamond\big(s(F_1), s(F_2), s(F_3) \in \{s_4, s_5, s_6\} \wedge s(F_4), s(F_5), s(F_6) \in \{s_0, s_1, s_2\}\big)$$



$$p \triangleq \big(s(F_1), s(F_2), s(F_3) \in \{s_4, s_5, s_6\} \wedge s(F_4), s(F_5), s(F_6) \in \{s_0, s_1, s_2\}\big)$$