Model Checking

CS60030 FORMAL SYSTEMS

PALLAB DASGUPTA,
FNAE, FASc,
A K Singh Distinguished Professor in AI,
Dept of Computer Science & Engineering
Indian Institute of Technology Kharagpur
Email: pallab@cse.iitkgp.ac.in
Web: http://cse.iitkgp.ac.in/~pallab

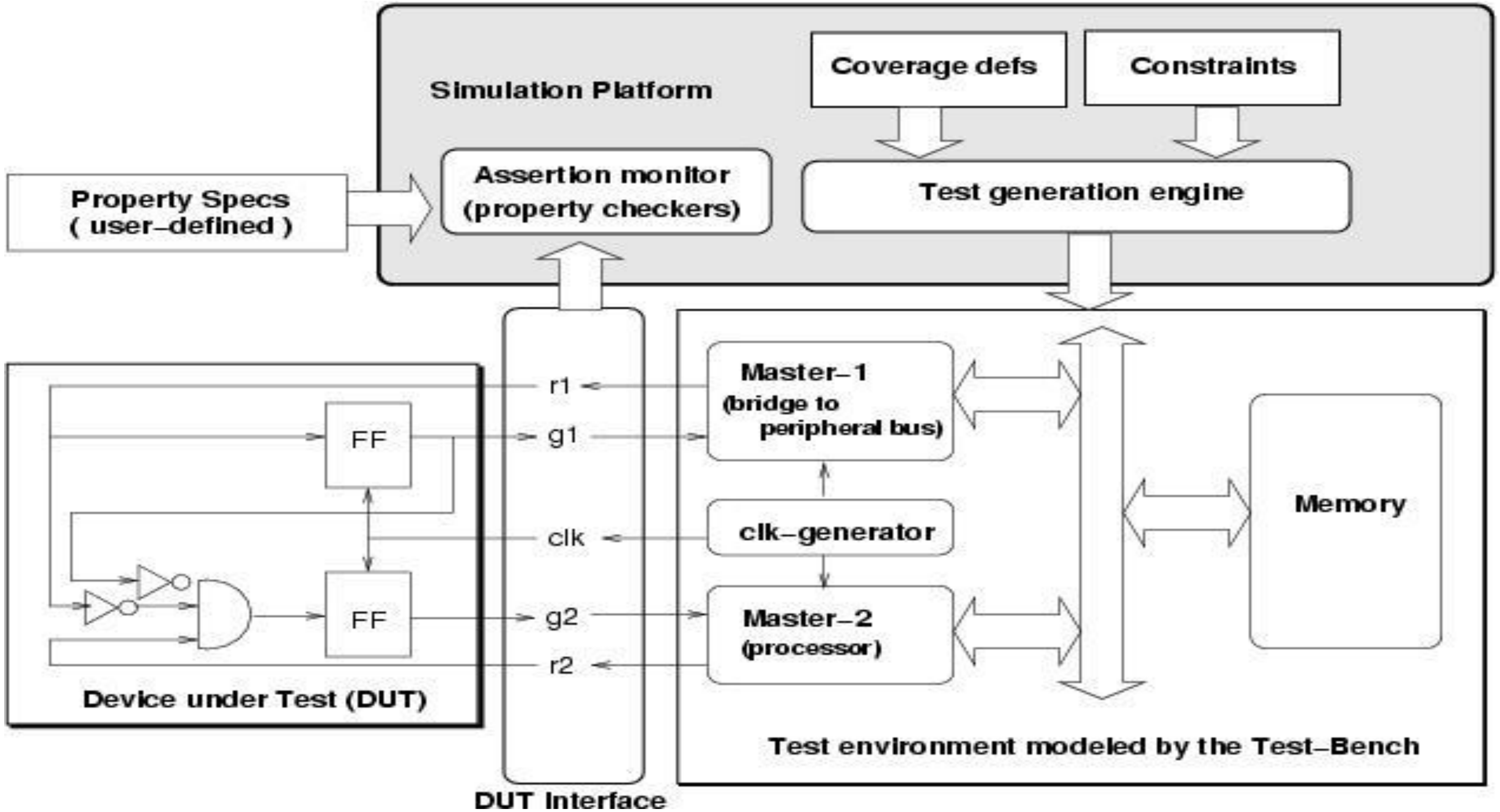
INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR

FMSAFE
FORMAL METHODS FOR SAFETY CRITICAL SYSTEMS

Formal Property Verification

- What is *formal property verification?*
  - Verification of *formal properties?*
  - *Formal methods* for property verification?
- Both are important requirements
- Broad Classification
  - Dynamic property verification (DPV)
  - Static/Formal property verification (FPV)

Dynamic Property Verification (DPV)



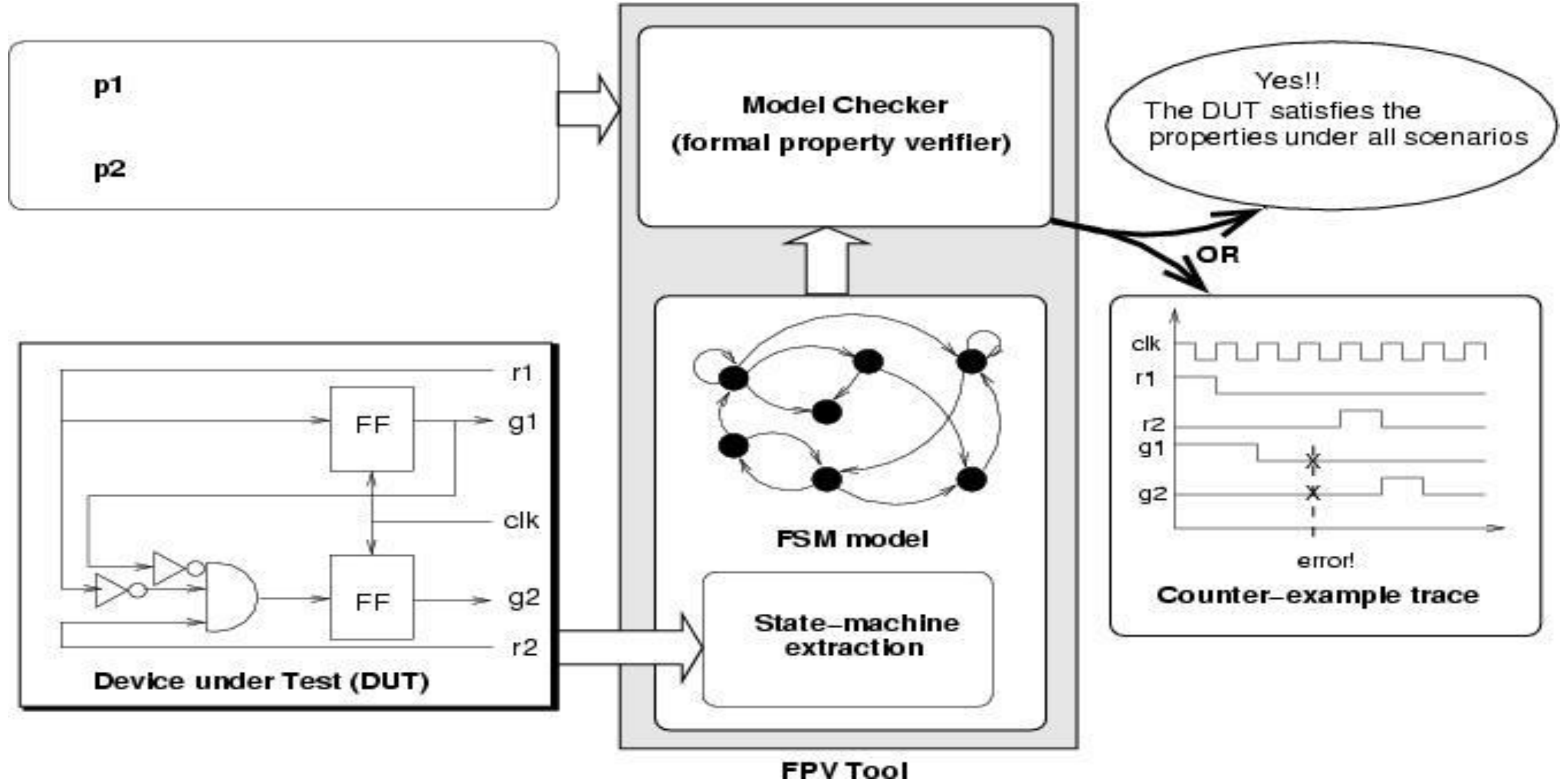
Formal Property Verification (FPV)

Temporal Logics (Timed / Untimed, Linear Time / Branching Time): ***LTL, CTL***

Early Languages: ***Forspec (Intel), Sugar (IBM), Open Vera Assertions (Synopsys)***

Current IEEE Standards: ***SystemVerilog Assertions (SVA),***

Property Specification Language (PSL)



Formal Property Verification

The formal method is called *"Model Checking"*

- The algorithm has two inputs
  - A finite state transition system (FSM) representing the implementation
  - A formal property representing the specification
- The algorithm checks whether the FSM *"models"* the property
  - This is an exhaustive search of the FSM to see whether it has any path / state that refutes the property.

Transition Systems and Kripke Structures

A *transition system* TS is a tuple $(S, Act, \rightarrow, I, AP, L)$ where

- S is a set of states
- Act is a set of actions
- $\rightarrow \subseteq S \times Act \times S$ is a transition relation
- $I \subseteq S$ is a set of initial states
- AP is a set of atomic propositions
- $L : S \rightarrow 2^{AP}$ is a labeling function

S and Act are either finite or countably infinite

A *Kripke Structure* TS is a tuple $(S, \rightarrow, I, AP, L)$ where

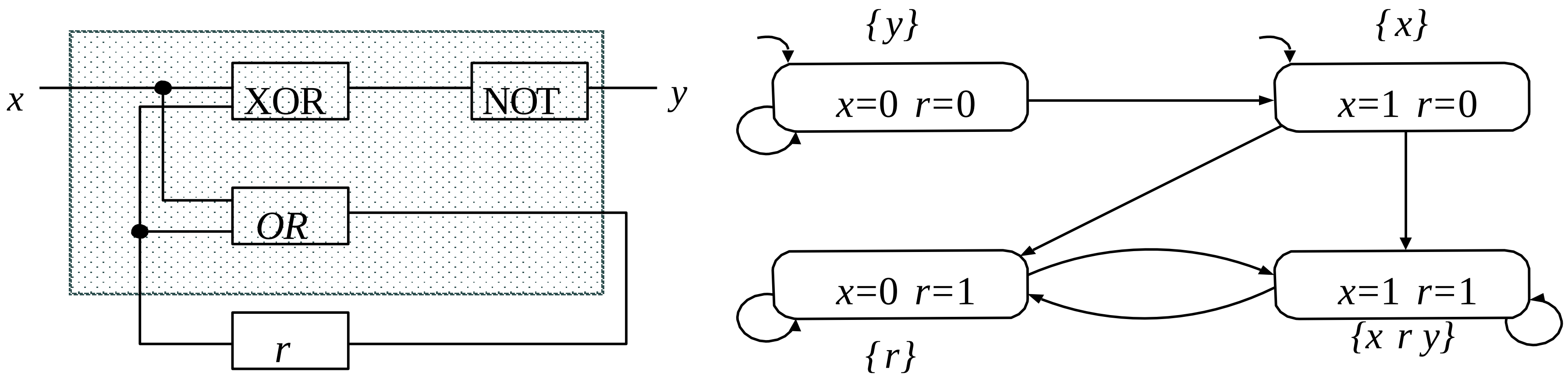
- S is a set of states (inputs are part of the state)
- $\rightarrow \subseteq S \times S$ is a transition relation
- $I \subseteq S$ is a set of initial states
- AP is a set of atomic propositions
- $L : S \rightarrow 2^{AP}$ is a labeling function

$\rightarrow$ is a total relation, that is, every state has a next state (could be itself)

S is finite

In this discussion we shall use the notion of Kripke structures

Modeling Sequential Circuits as Kripke Structures



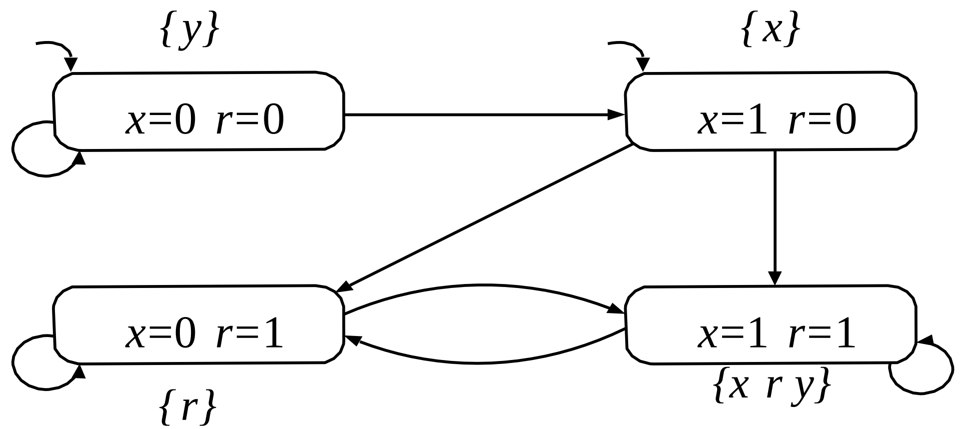
A simple hardware circuit with Input variable x, Output variable y, and Register r

Output function $\neg(x \oplus r)$ and register evaluation function $x \vee r$

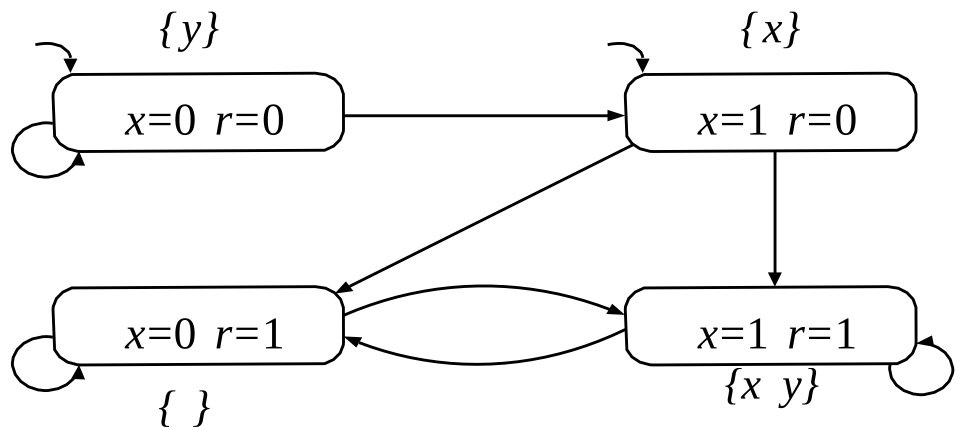
Atomic Propositions

Consider two possible state-labelings:

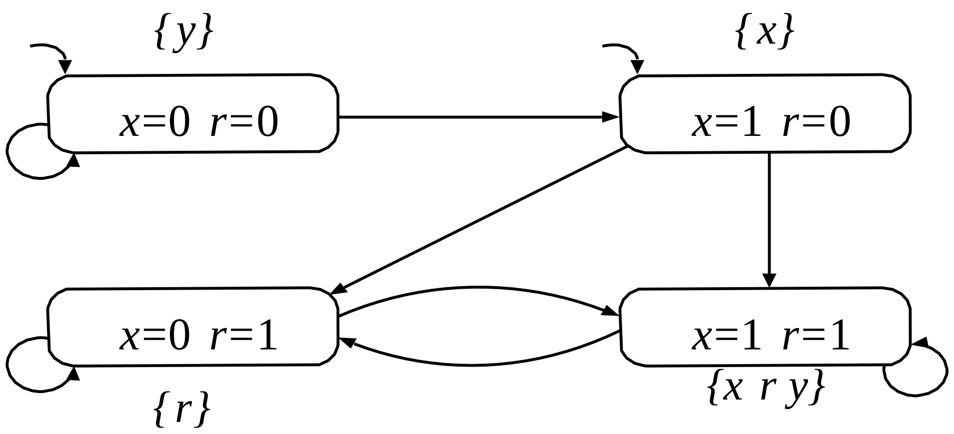
- Let $AP = \{ x, y, r \}$
  - $L((x = 0, r = 1)) = \{ r \}$ and $L((x = 1, r = 1)) = \{ x, r, y \}$
  - $L((x = 0, r = 0)) = \{ y \}$ and $L((x = 1, r = 0)) = \{ x \}$
  - property e.g., “once the register is one, it remains one”



- Let $AP' = \{ x, y \}$ – the register evaluations are now “invisible”
  - $L((x = 0, r = 1)) = \emptyset$ and $L((x = 1, r = 1)) = \{ x, y \}$
  - $L((x = 0, r = 0)) = \{ y \}$ and $L((x = 1, r = 0)) = \{ x \}$
  - property e.g., “the output bit y is set infinitely often”



Automata over Infinite Words



- A run of this state machine is an infinite sequence of states.
  - If we observe only the state labels, then each state is viewed as a combination of labels (note that two states can have same labels)

Runs of the transition system:

Σ = { { }, {x}, {y}, {r}, {x y}, {x r}, {r y}, {x r y} } = 2^{AP}

Each run of the system belongs to $(2^{AP})^\omega$ that is, the set of infinite words over Σ

Runs(TS) ⊆ $(2^{AP})^\omega$

Runs of the formal property:

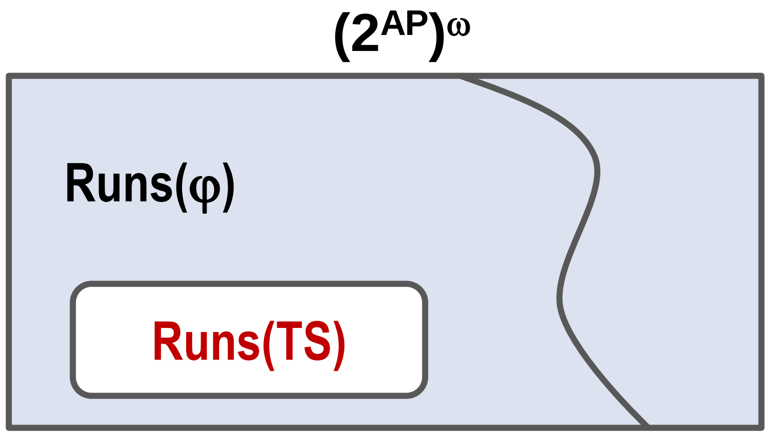
Linear time properties are also defined over Σ = 2^{AP}

Each run in $(2^{AP})^\omega$ either satisfies a given formal property φ or is a counterexample

Runs(φ) ⊆ $(2^{AP})^\omega$

TS ⊨ φ (read as TS models φ) iff Runs(TS) ⊆ Runs(φ)

Model Checking Linear Time Properties



- Linear Temporal Logic (LTL) captures an expressive subset of Omega Regular Languages
  - SVA is derived from LTL
- Given a LTL property, φ, to determine whether TS ⊨ φ we do the following:
  - Since TS ⊨ φ iff Runs(TS) ⊆ Runs(φ), it follows that
    Runs(TS) ∩ [$(2^{AP})^\omega$ – Runs(φ)] = ∅
  - We create an automaton, $B_{\neg\varphi}$, which accepts runs satisfying ¬φ, that is, runs in $(2^{AP})^\omega$ – Runs(φ)
  - We compute the product of TS with $B_{\neg\varphi}$ and check whether the product has any accepting run.
    - If not then TS |= φ.
    - Otherwise, the accepting run is a counter-example trace.

Nondeterministic Büchi automata

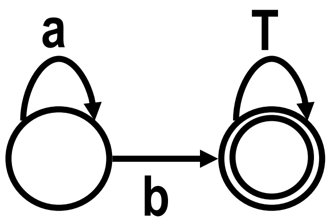
- NFA (and DFA) are incapable of accepting infinite words

- A nondeterministic Büchi automaton (NBA) $\mathcal{A}$ is a tuple $(Q, \Sigma, \delta, Q_0, F)$ where:
  - Q is a finite set of states with $Q_0 \subseteq Q$ a set of initial states
  - Σ is an alphabet
  - $\delta : Q \times \Sigma \to 2^Q$ is a transition function
  - $F \subseteq Q$ is a set of accept (or: final) states

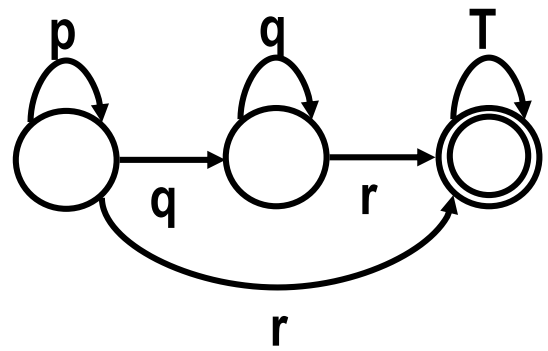
- NBAs are structurally similar to NFAs.
- But they have separate *acceptance criteria*
  - An NFA accepts its (finite) input if some run of the NFA reaches an accept state at the end of the input
  - *A Büchi automaton accepts its infinite length input if at least one of the accept states is visited infinitely often*

Linear Time Properties can be converted to NBA

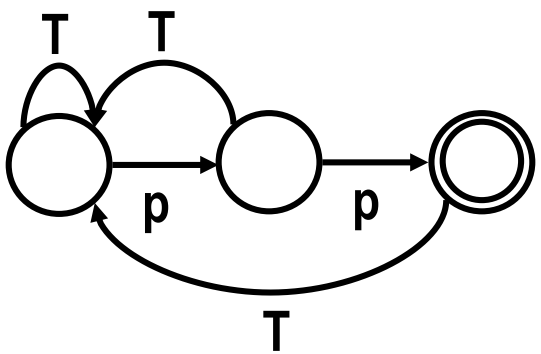
a U b



p U (q U r)



GF(p ∧ Xp)

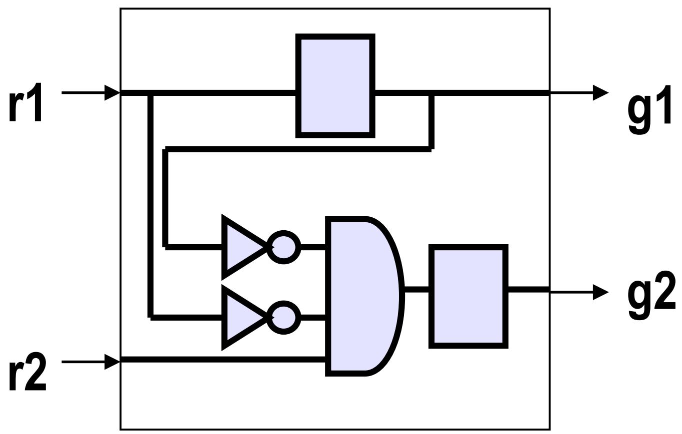


Here the Büchi acceptance criteria ensures that p ∧ Xp is satisfied infinitely often.

DFAs and NFAs are equally powerful, and therefore many algorithms convert a NFA to a DFA before product construction.

Non-deterministic Büchi automata are strictly more powerful than deterministic Büchi automata. Therefore we do not attempt to convert a NBA to a DBA.

Our running example: *Priority Arbiter*



Design-under-test (DUT)

Specification: Formal Property

- One of the grant lines is always asserted
- In Linear Temporal Logic: $G(g1 \vee g2)$

We wish to check whether: $TS(DUT) \models G(g1 \vee g2)$

The Kripke Structure: TS(DUT)

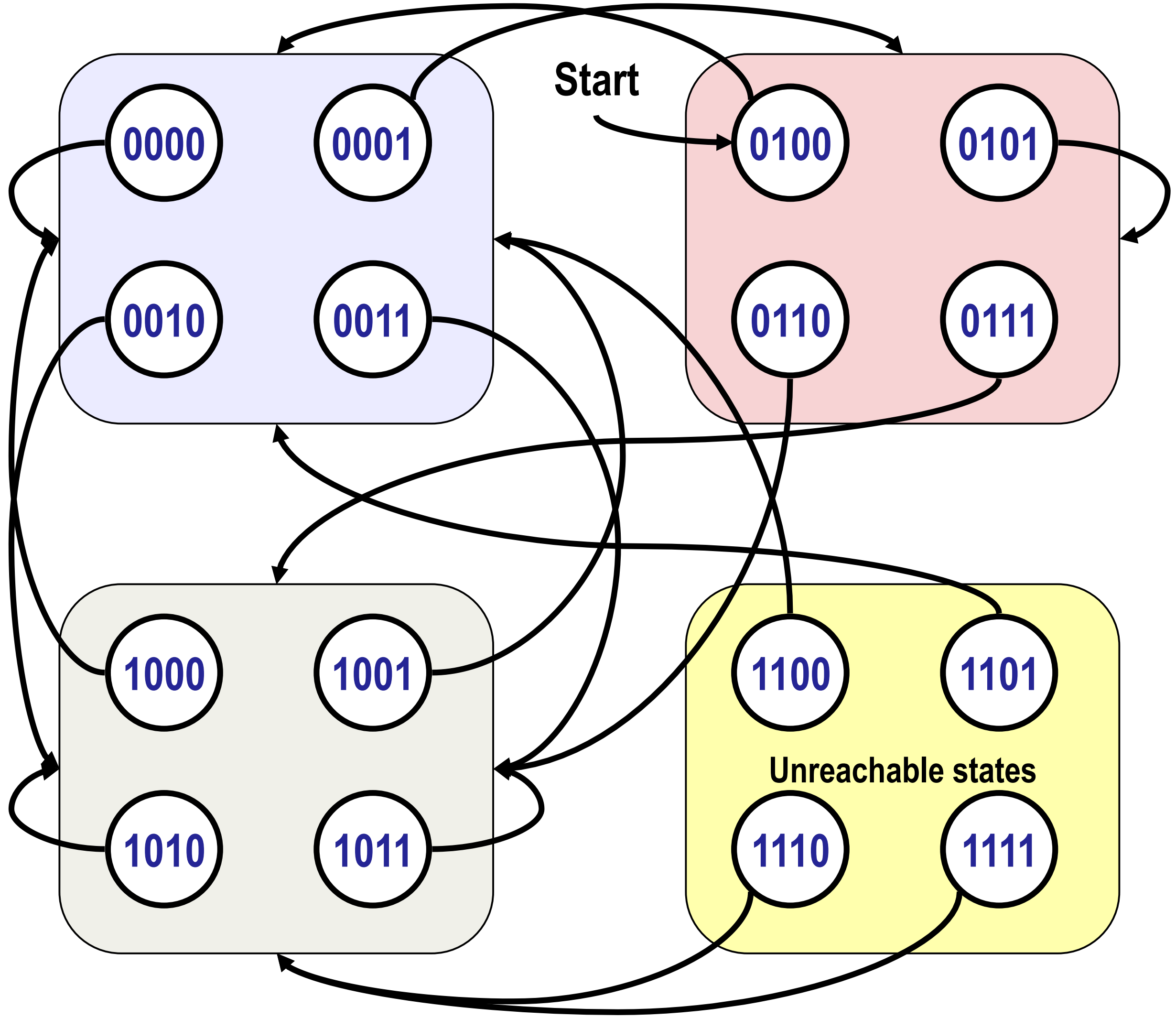
Transition Relation:

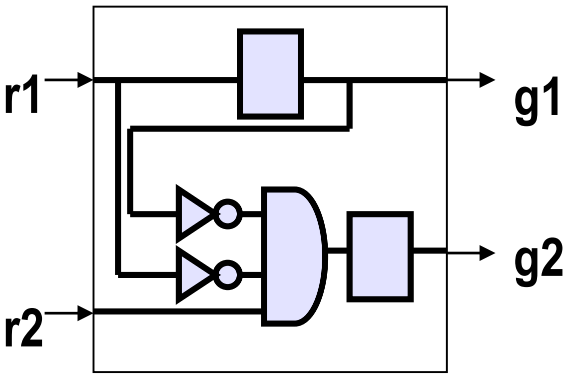
$g'_1 \Leftrightarrow r_1$

$g'_2 \Leftrightarrow \neg r_1 \land r_2 \land \neg g_1$

Initial State:

$r_1=0,\ r_2=0,\ g_1=0,\ g_2=1$





PS g_1g_2	I/P r_1r_2	NS $g'_1g'_2$	Next I/P
00	00	00	xx
00	01	01	xx
00	10	10	xx
00	11	10	xx
01	00	00	xx
01	01	01	xx
01	10	10	xx
01	11	10	xx
10	00	00	xx
10	01	00	xx
10	10	10	xx
10	11	10	xx
11	00	00	xx
11	01	00	xx
11	10	10	xx
11	11	10	xx

This is only for demonstration !!

We will never create this explicitly, but encode it in SAT / BDD

Now we handle the specification

Our property: $\varphi = G[\, g_1 \vee g_2 \,]$

- Either of the grant lines is always active

We will create the automaton, $\mathcal{A}$, for $\neg\varphi$

- $\neg\varphi = F[\, \neg g_1 \wedge \neg g_2 \,]$
- Sometime both grant lines will be inactive

We will then search for a common run between this automaton and the TS(DUT) from the implementation

Intuitive steps towards creating the automaton for the property

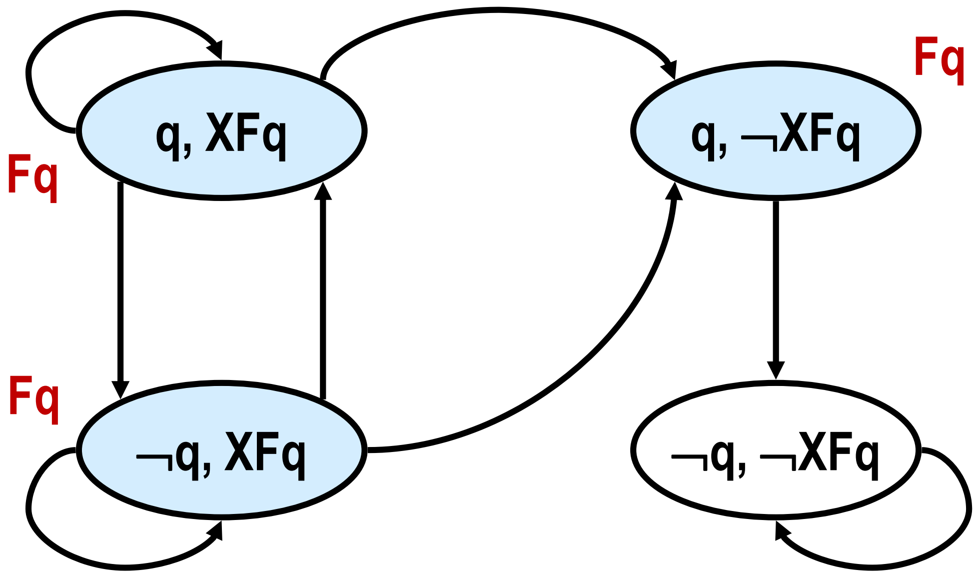
- Let us consider our property $F(\neg g_1 \wedge \neg g_2)$ *// Eventually q is true*
- Using q as a short form for $\neg g_1 \wedge \neg g_2$ we can rewrite it as:

  $Fq = q \vee XFq$ *// Either q is true now or Fq is true in the next state*

- Therefore we can classify the states in a run into the following types:
  - States that satisfy q
  - States that do not satisfy q but satisfy XFq
  - States that do not satisfy q and do not satisfy XFq
  - The first two types are labeled by Fq

The automaton for our property

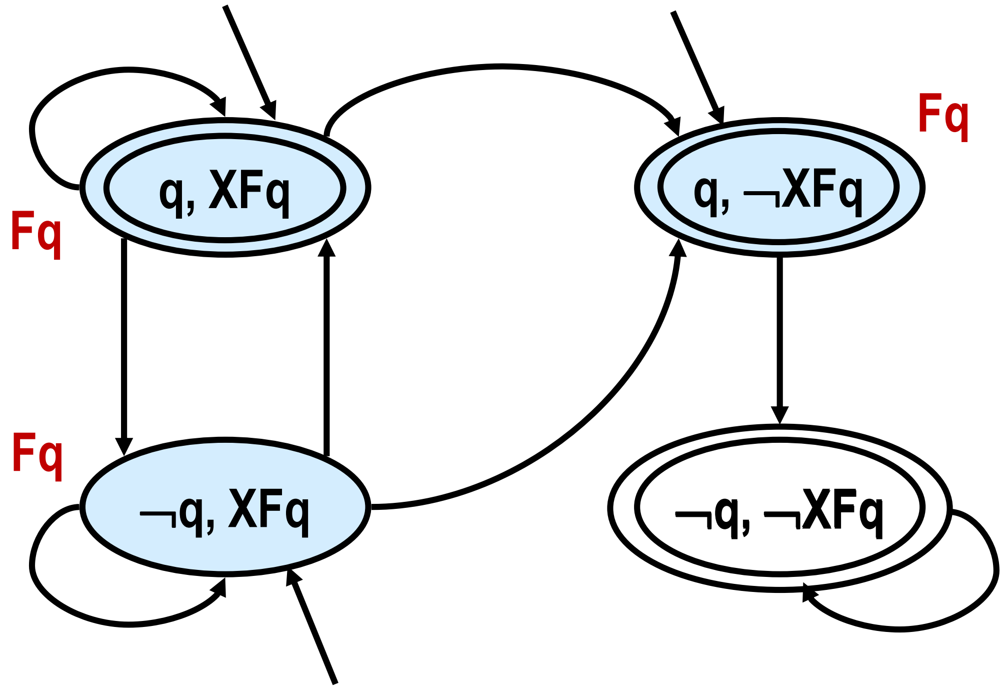
Our property: **Fq** *where* **q** = $\neg g_1 \wedge \neg g_2$



- States that satisfy q and states that do not satisfy q but satisfy XFq are labeled with **Fq**
- We add the following edges:
  - From states satisfying XFq to states labeled with Fq
  - From states satisfying ¬XFq to states satisfying ¬q
- But the self loop in the state labeled {¬q, XFq} is problematic
  - It allows the satisfaction of q to be postponed forever, in which case Fq does not hold

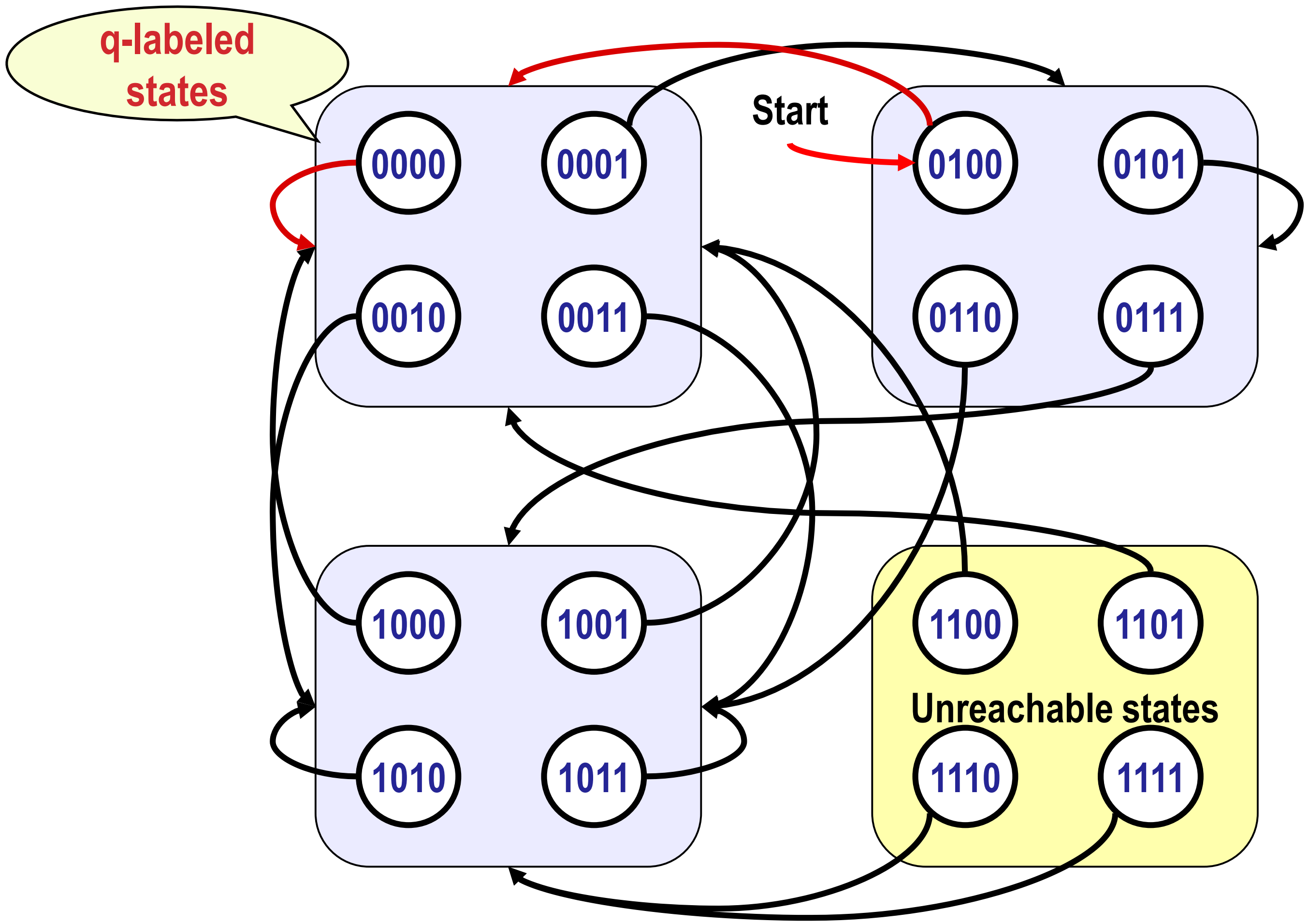
The Büchi Automaton

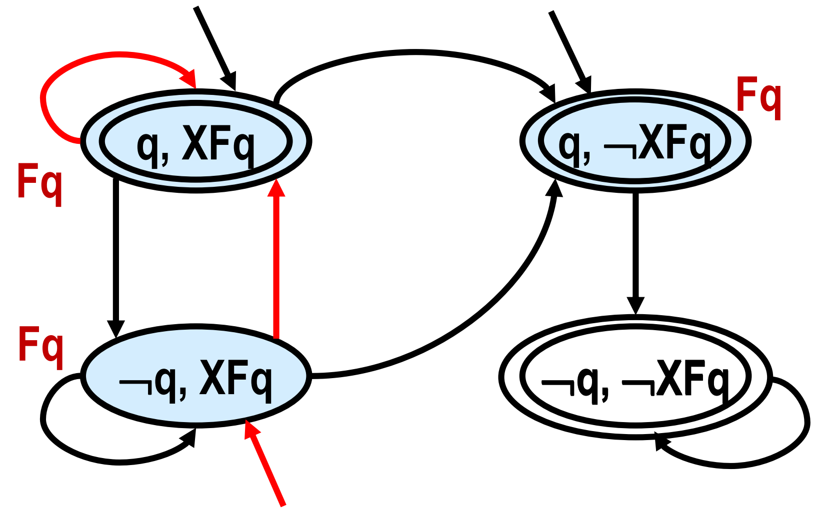
Our property: Fq *where* $q = \neg g_1 \wedge \neg g_2$



- The self loop in the state labeled {¬q, XFq} is problematic
  - It allows the satisfaction of q to be postponed forever, in which case Fq does not hold
- By defining the remaining three states as ***accept states***, we force the accepting runs to come out of the state labeled {¬q, XFq}
  - Recall that the Büchi acceptance criterion states that accept states must be visited infinitely often.

Is the product non-empty?





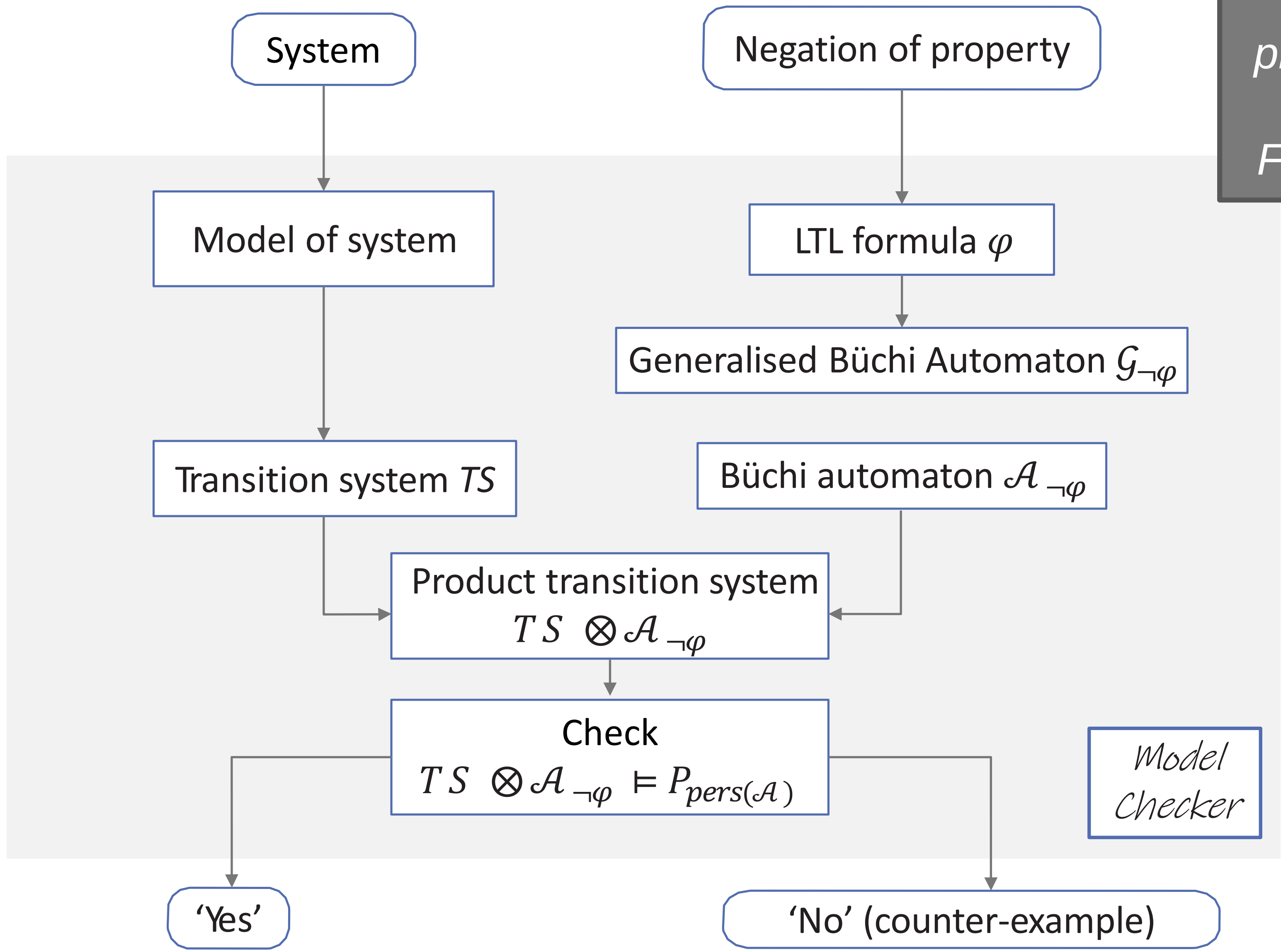
The common run is shown in red. Product is non-empty.

Conclusion: TS(DUT) ⊨ G(g1 ∨ g2) is not true. The counterexample is the run in red.

Computational facts

- If a LTL property has k sub-formulas, then the number of states in its automaton may have $O(2^k)$ states
  - Decomposing the property into a conjunction of smaller properties helps in containing the size of this automaton
  - It also helps the FPV tool to prune away parts of the implementation before making the emptiness check
- LTL model checking is PSPACE-complete, but linear in the size of the implementation
  - However, the main bottleneck is in the size of the implementation, which is why we use succinct representations.

LTL Model Checking – An Overview



A persistence property for an NBA $\mathcal{A}$ is FG ("no final state")

"Elementary" Sets for φ

- For an LTL-property φ, the set closure(φ) consists of:
  - All sub-formulas ψ of φ and their negation $\neg\psi$.

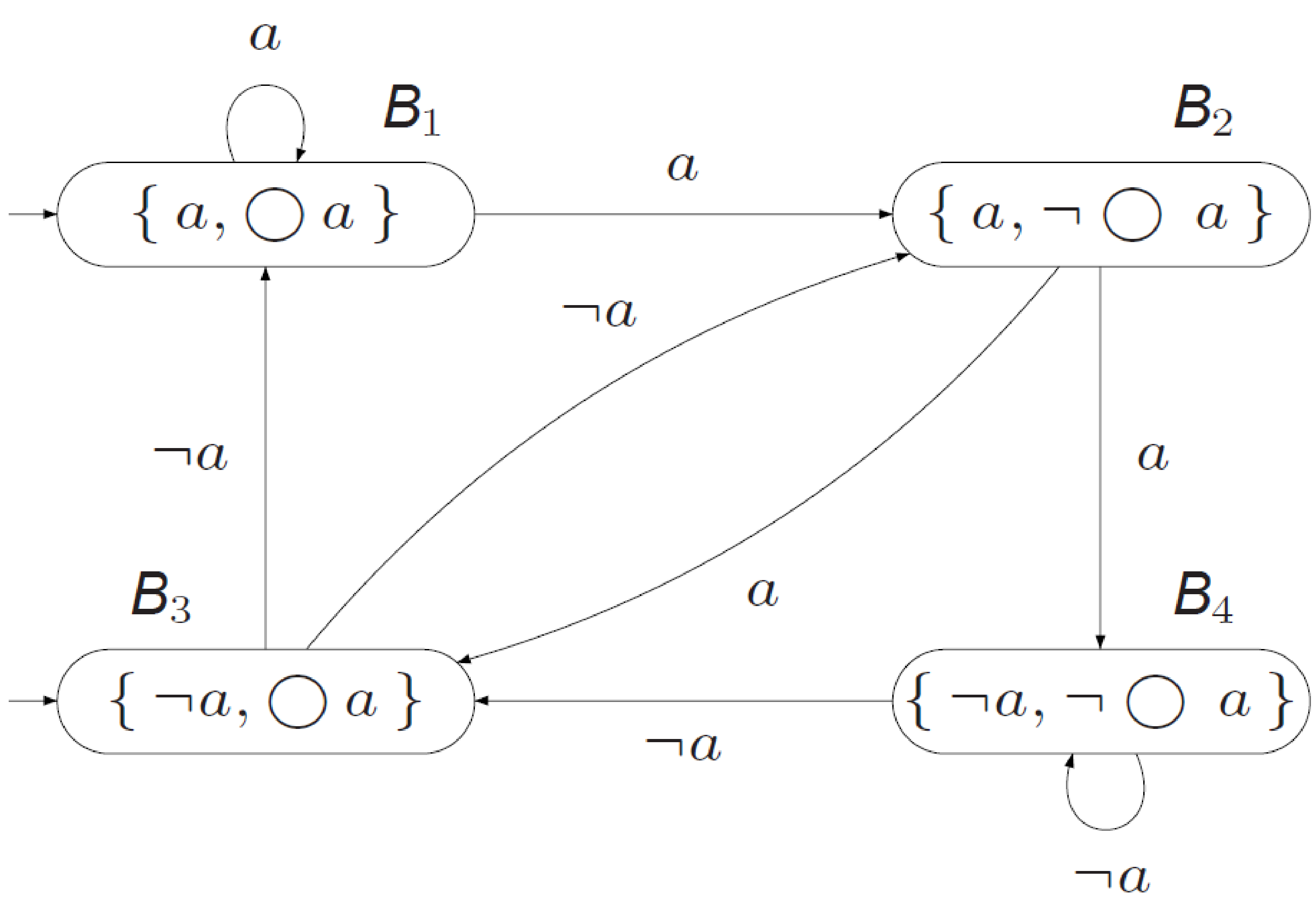
The set B $\subseteq$ closure(φ) is elementary if:

1. B is logically consistent - if for all $\varphi_1 \wedge \varphi_2$, $\psi \in$ closure(φ):
   - $\varphi_1 \wedge \varphi_2 \in B \iff \varphi_1 \in B$ and $\varphi_2 \in B$
   - $\psi \in B \implies \neg\psi \notin B$
   - true $\in$ closure(φ) $\implies$ true $\in B$
2. B is locally consistent – if for all φ_1 U $\varphi_2 \in$ closure(φ):
   - $\varphi_2 \in B \implies \varphi_1 \mathsf{U} \varphi_2 \in B$
   - $\varphi_1 \mathsf{U} \varphi_2 \in B$ and $\varphi_2 \notin B \implies \varphi_1 \in B$
3. B is maximal – for all $\psi \in$ closure(φ):
   - $\psi \notin B \implies \neg\psi \in B$

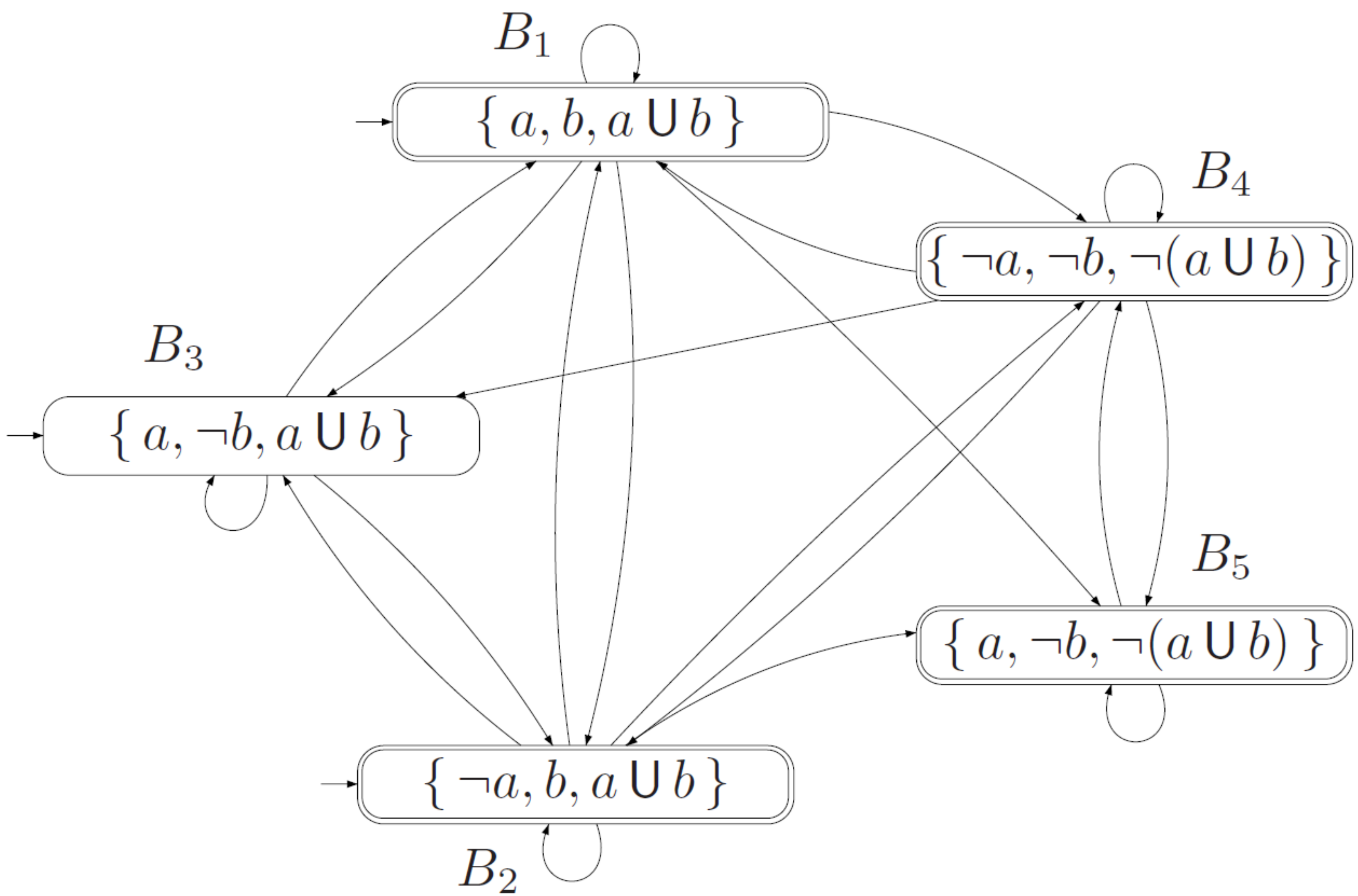
The GNBA for the LTL-property φ

- A Generalized NBA has multiple sets of accept states, $F_1, \ldots, F_k$ each of which must be visited infinitely often in an accepting run
- For the LTL-property φ, let $\mathcal{G}_\varphi = (Q, 2^{AP}, \delta, Q_0, \mathcal{F})$, where
  - Q is the set of elementary sets of formulas $B \subseteq closure(\varphi)$.
    - $Q_0 = \{ B \in Q \mid \varphi \in B \}$
  - $\mathcal{F} = \{ \{ B \in Q \mid \varphi_1 \, U \, \varphi_2 \notin B \text{ or } \varphi_2 \in B \} \mid \varphi_1 \, U \, \varphi_2 \in closure(\varphi) \}$
  - The transition relation $\delta: Q \times 2^{AP} \longrightarrow Q$ is given by:
    - $\delta(B, B \cap AP)$ is the set of all elementary sets of formulas B' satisfying:
      - For every $X\psi \in closure(\varphi)$:
        $X\psi \in B \Leftrightarrow \psi \in B'$
        AND
      - For every $\varphi_1 \, U \, \varphi_2 \in closure(\varphi)$:
        $\varphi_1 \, U \, \varphi_2 \in B \Leftrightarrow (\varphi_2 \in B \vee (\varphi_1 \in B \wedge \varphi_1 \, U \, \varphi_2 \in B'))$

GNBA for $\varphi = \bigcirc a$



GNBA for φ = a U b



Emptiness Check

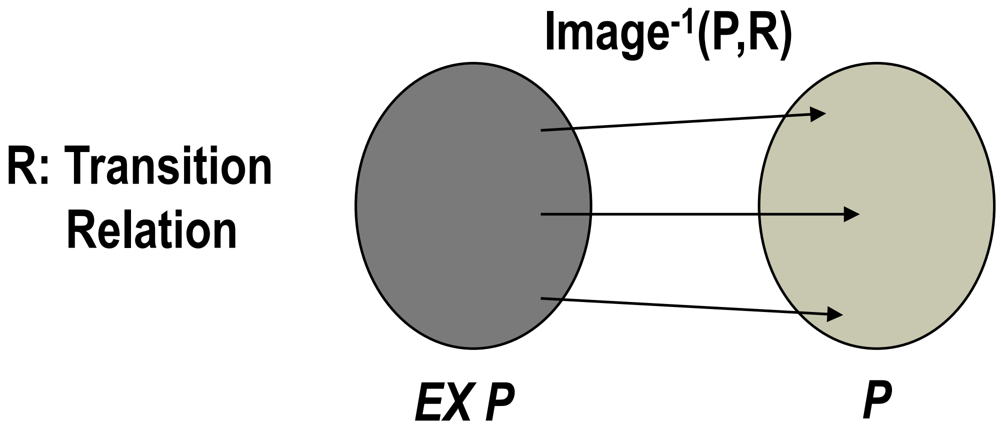
- Emptiness check for a NFA is to find whether any accepting run exists
  - Can be decided by finding whether any accept state is reachable
  - We can do this using the symbolic reachability methods discussed earlier

- Emptiness check for a NBA is to find whether any accepting run exists using the Büchi acceptance criterion
  - Can be decided by finding whether any strongly connected component containing one or more accept states is reachable
  - Once we find the states in strongly connected components with accept states, we can use the symbolic reachability methods to find whether such components are reachable from the initial states
  - How to find strongly connected components using symbolic search?

CTL Model Checking

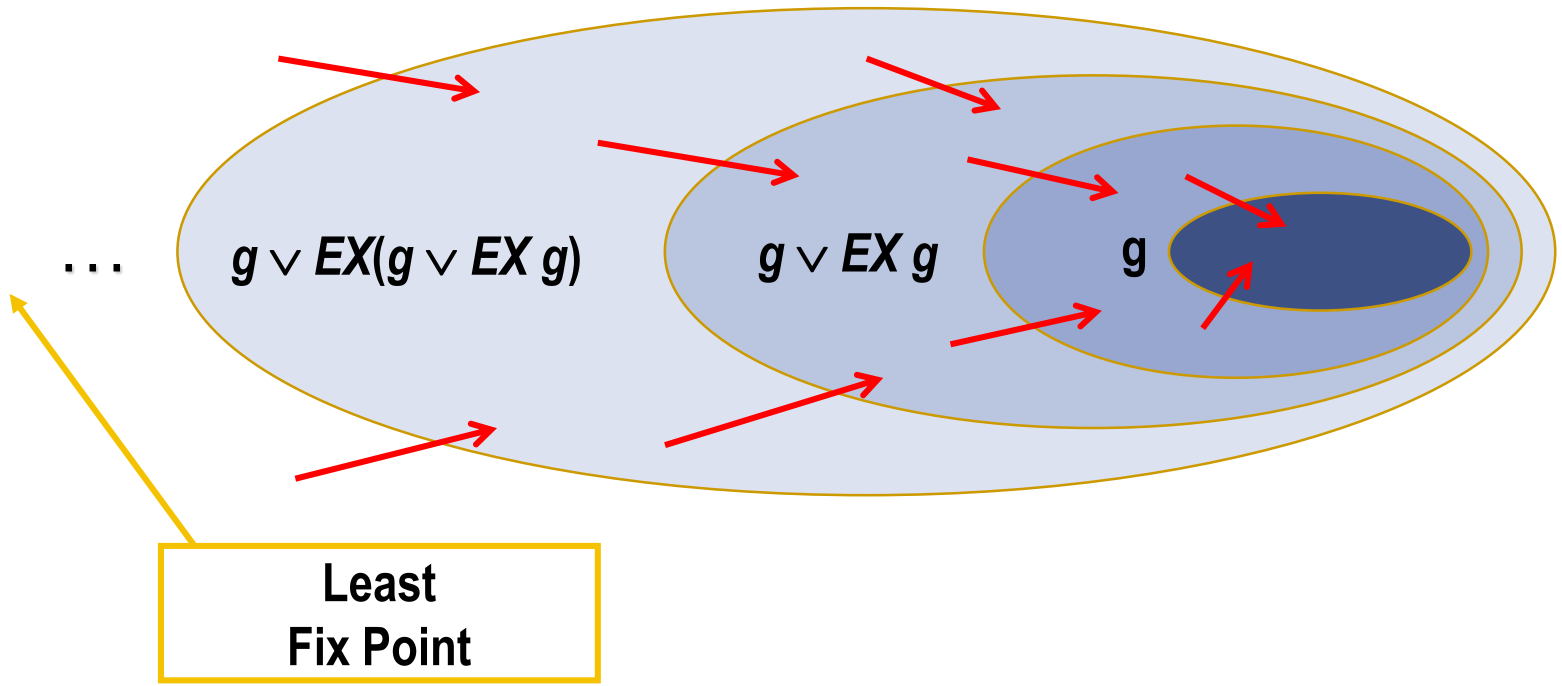
- Need only to show methodology for EX, EU, EG.
- Other modalities can be expressed in terms of EX, EU, EG.
    - $AFp = \neg EG \neg p$
    - $AGp = \neg EF \neg p$
    - $A(p \; U \; q) = \neg E[\neg q \; U \; (\neg p \wedge \neg q)] \wedge \neg EG \neg q$

Example: EX p



$$EXp = \{ v \mid \exists v' \, (v, v') \in R \wedge p \in \mathcal{L}(v') \}$$

Example: EF g



Given a model M = ⟨ AP, S, S0, R, L ⟩ and S_g the sets of states satisfying g in M

procedure **CheckEF** (S_g)

Q := emptyset; Q' := S_g ;

while Q ≠ Q' do

  Q := Q';

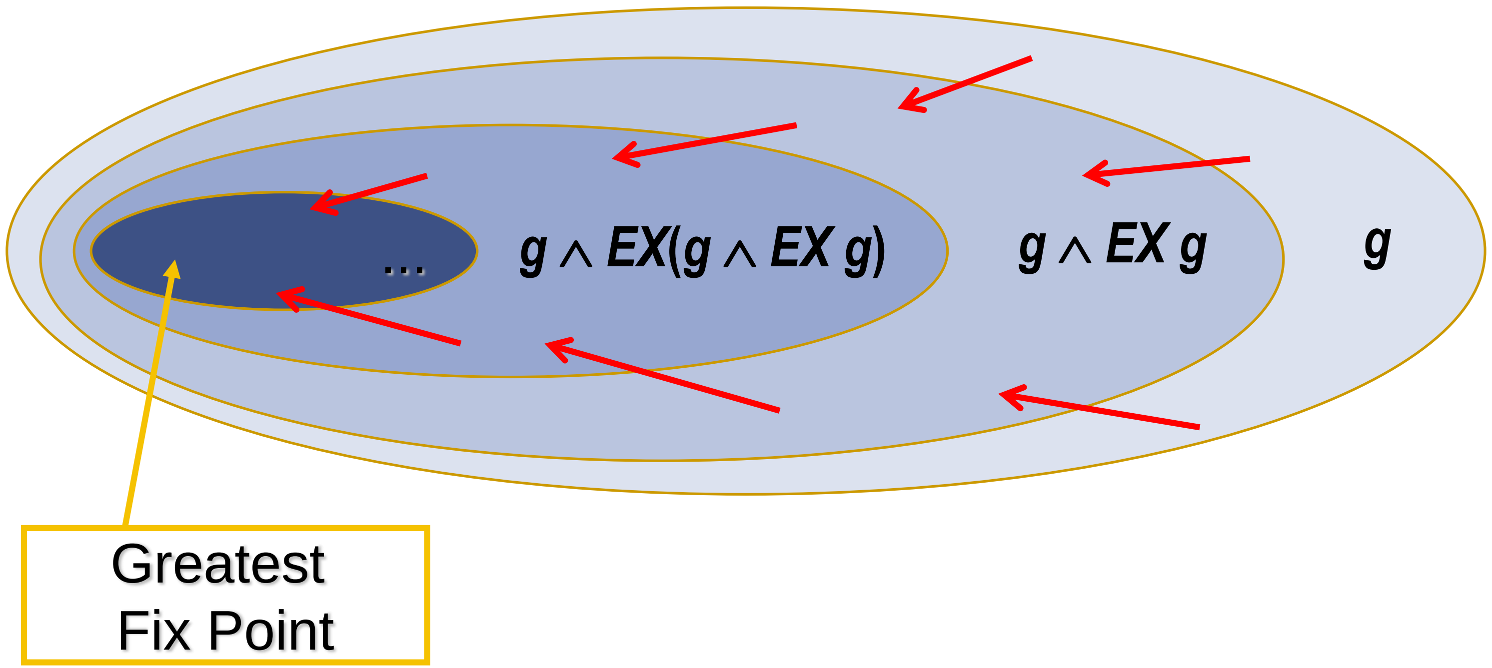
  Q' := Q ∪ { s | ∃s' [R(s,s') ∧ Q(s')] }

end while

S_f := Q ; return(S_f)

Example: EG g

EG g is calculated as



Given a model M = ⟨ AP, S, S0, R, L ⟩ and S_g the sets of states satisfying g in M

procedure **CheckEG** (S_g)

Q := S ; Q' := Sg ;

while Q $\neq$ Q' do

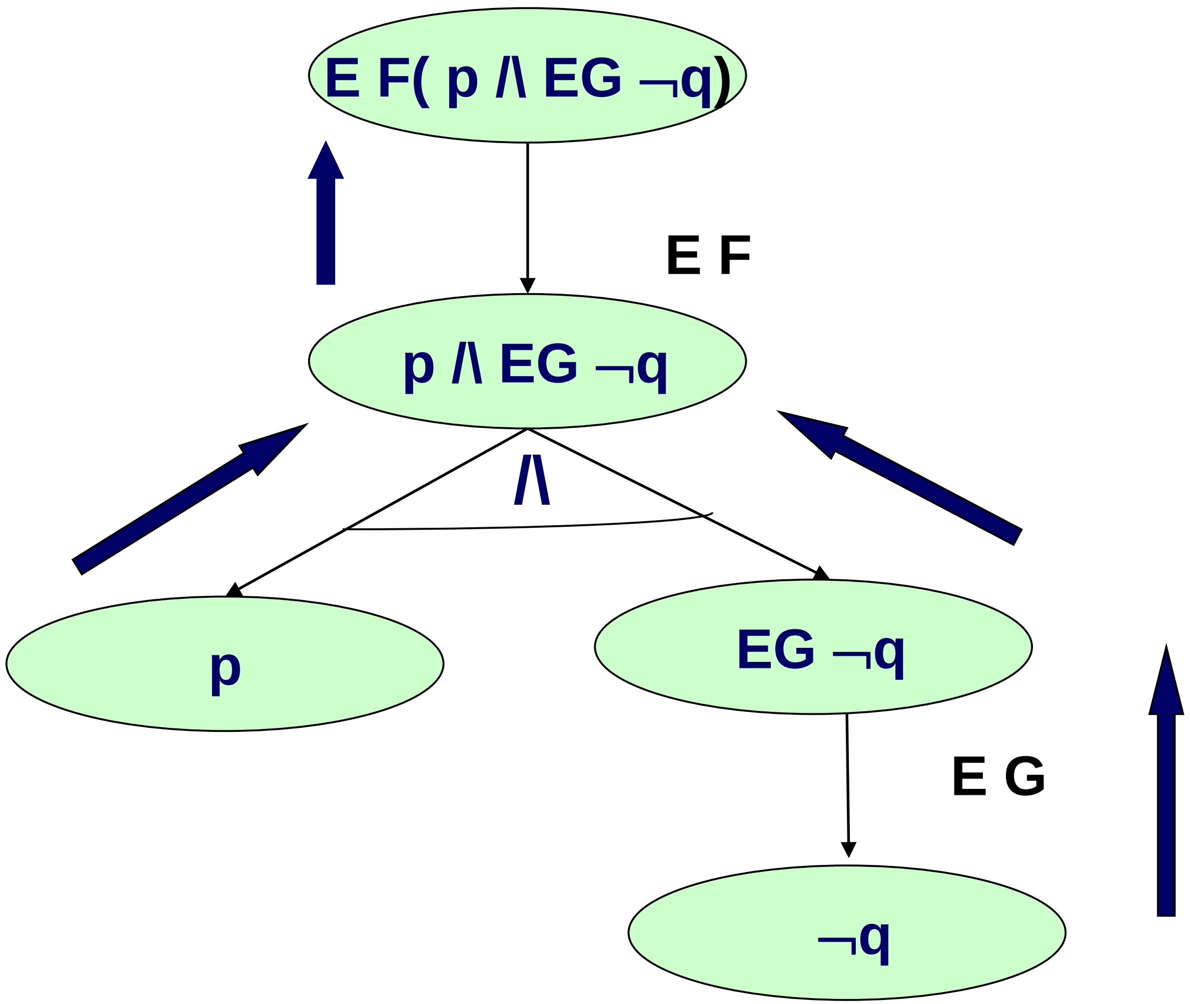
    Q := Q';

    Q' := Q $\cap$ { s | $\exists$s' [R(s,s') $\wedge$ Q(s')] }

end while

S_f := Q ; return(S_f)

Checking Nested Formulas



Bottom Up

Checking Nested formulas

EF (p ∧ EG ¬q)

- p state
- q state
- ¬p ∧ ¬q state

EF(p ∧ EG ¬q)