Embedded Systems

Arijit Mondal
Dept. of Computer Science & Engineering
Indian Institute of Technology Patna
`arijit@iitp.ac.in`

Verification

Introduction

- **The goal of verification**
  - To ensure 100% correct in functionality and timing
  - Spend 50∼70% of time to verify a design

- **Functional verification**
  - Simulation
  - Formal proof

- **Timing verification**
  - Dynamic timing simulation (DTS)
  - Static timing analysis (STA)

Verification vs Test

Verification

- Verifies correctness of design i.e., check if the design meets the specifications.
- Simulation or formal methods.
- Performed once prior to manufacturing.
- Required for reliability of design.

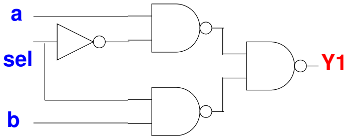
Test

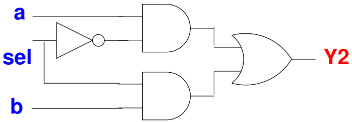
- Checks correctness of manufactured hardware.
- Two-stage process:
  - Test generation: CAD tools executed once during design for ATPG
  - Test application: TPs tests applied to ALL hardware samples
- Test application performed on every manufactured device.
- Responsible for reliability of devices.

Simulation

- Need to drive the circuit with the stimulus
  - Exhaustive simulation
    - Drive the circuit with all possible stimulus
  - Non-exhaustive simulations
    - Drive the circuit with selected stimulus
    - To find appropriate subset is a complex problem
    - May not cover all cases
- Number of test cases may be exponential

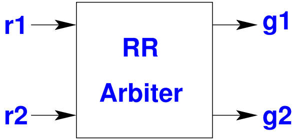
Verification of Combinational Circuits





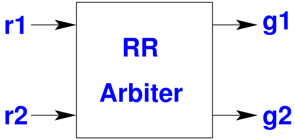
- **Are Y1 and Y2 equivalent?**
  - $Y1 = \overline{\overline{(a \land \neg sel)} \land \overline{(b \land sel)}}$
  - $Y2 = (a \land \neg sel) \lor (b \land sel)$
- **Canonical structure of Binary Decision Diagram can be exploited to compare Boolean functions like Y1 & Y2**

Verification of Sequential Circuits



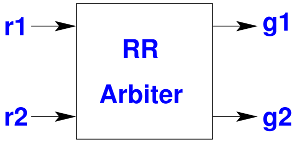
- Properties span across cycle boundaries
- Example: Two way round robin arbiter
  - If the request bit r1 is true in a cycle then the grant bit g1 has to be true within the next two clock cycles

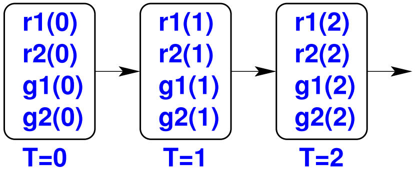
Verification of Sequential Circuits



- Properties span across cycle boundaries
- Example: Two way round robin arbiter
  - If the request bit r1 is true in a cycle then the grant bit g1 has to be true within the next two clock cycles
- Need temporal logic to specify the behavior

Verification of Sequential Circuits



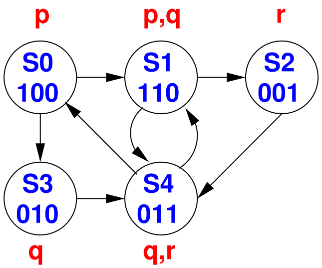


- *If the request bit r1 is true in a cycle then the grant bit g1 has to be true within the next two clock cycles*
- $\forall t[r1(t) \rightarrow g1(t+1) \vee g1(t+2)]$
- **In propositional temporal logic time (t) is implicit**
  - **always r1 $\rightarrow$ (next g1) $\vee$ (next next g1)**

Temporal logic

- **The truth value of a temporal logic is defined with respect to a model.**
- **Temporal logic formula is not statically true or false in a model.**
- **The models of temporal logic contain several states and a formula can be true in some states and false in others.**
- **Example:**
  - **I am** *always* **happy.**
  - **I will** *eventually* **be happy.**
  - **I will be happy** *until* **I do something wrong.**
  - **I am happy.**

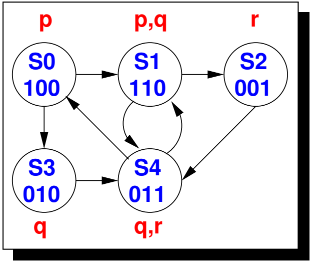
Kripke Structure



- $M = (AP, S, S_0, T, L)$
  - AP — Set of **atomic proposition**
  - S — Set of **states**
  - S_0 — Set of **initial states**
  - T — **Total transition relation** ($T \subseteq S \times S$)
  - L — **Labeling function** ($S \rightarrow 2^{AP}$)

Path

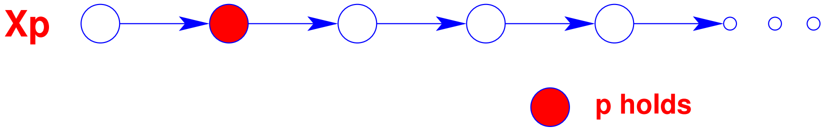
- **A path $\pi = s_0, s_1, \ldots$ in a Kripke structure is a sequence of states such that $\forall i, (s_i, s_{i+1}) \in T$**
- **Sample paths**
  - $S0, S1, S2, S4, S1, \ldots$
  - $S0, S3, S4, S0, \ldots$
  - $S0, S1, S4, S1, \ldots$
  - $\pi = \underbrace{s_0, s_1, \ldots, s_k}_{\textbf{prefix of } \pi_k \textbf{ in } \pi}, s_{k+1} \ldots$
  - $\pi = s_0, s_1, \ldots, \underbrace{s_k, s_{k+1} \ldots}_{\textbf{suffix of } \pi^k \textbf{ in } \pi}$



Temporal operators

- Two fundamental path operators
  - Next operator
    - Xp — property p holds in the next state
  - Until operator
    - p U q — property p holds in all states upto the state where property q holds
- Derived operators
  - Eventual/Future operator
    - Fp — property p holds eventually (in some future states)
  - Always/Globally operator
    - Gp — property p holds always (at all states)
- All these operators are interpreted over the paths in Kripke structure under consideration
- All Boolean operators are supported by the temporal logics

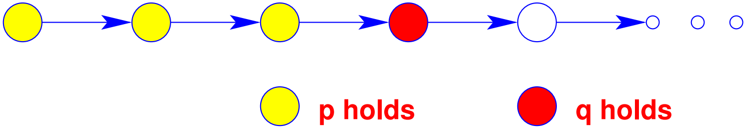
The next operator (X)



- p holds in the next state of the path
- Formally
  - $\pi \models \mathrm{Xp}$ **iff** $\pi^1 \models \mathrm{p}$

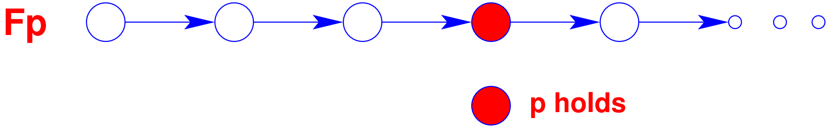
The until operator (U)

pUq



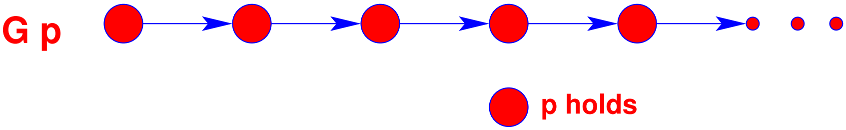
- q holds eventually and p holds until q holds
- Formally
  - $\pi \models p\,\mathsf{U}\,q$ **iff** $\exists k$ **such that** $\pi^k \models q$ **and** $\forall j, 0 \leq j < k$ **we have** $\pi^j \models p$

The eventual operator (F)



- p holds eventually (in future)
- Formally
  - $\pi \models Fp$ iff $\exists k$ such that $\pi^k \models p$
  - This can be written as true U p

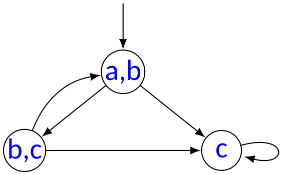
The always operator (G)

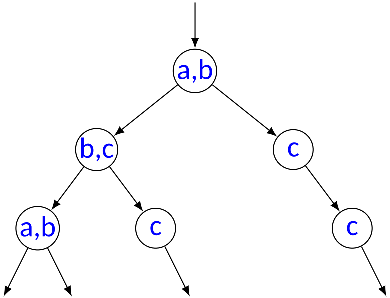


- p holds always (globally)
- Formally
  - $\pi \models Gp$ **iff** $\forall k$ **we have** $\pi^k \models p$
  - **This can be written as** $\neg(\text{true}\ U\ \neg p)$ **or** $\neg F \neg p$

Branching Time Logic

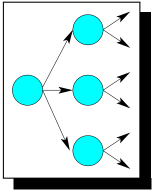
- **Interpreted over computation tree**



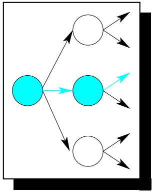


Path Quantifier

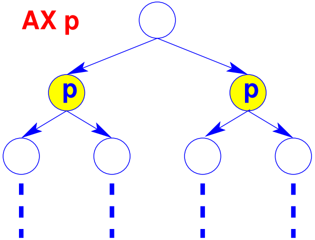
- A: “For all paths …”

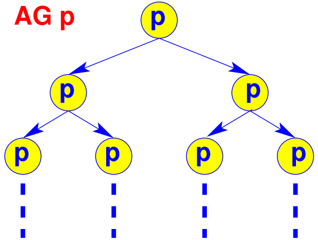


- E: “There exists a path …”



Universal Path Quantification



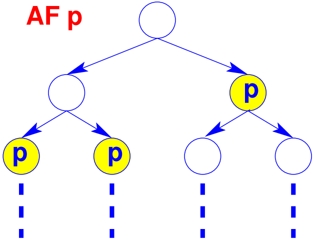


In all the next states p holds.

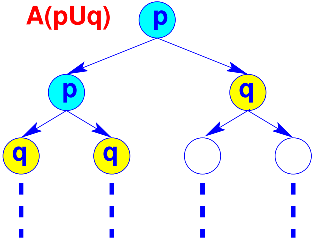
Along all the paths p holds forever.

Universal Path Quantification

AF p



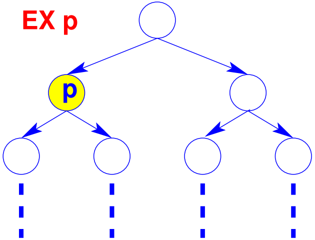
A(pUq)



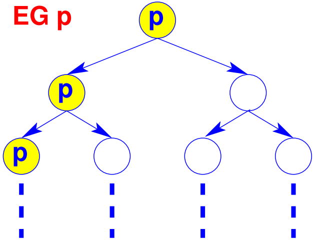
Along all the paths p holds eventually.

Along all the paths p holds until q holds.

Existential Path Quantification



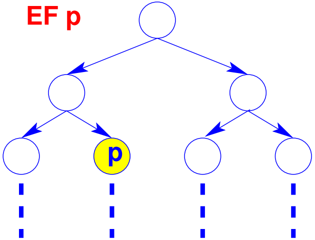
There exists a next state where p holds.



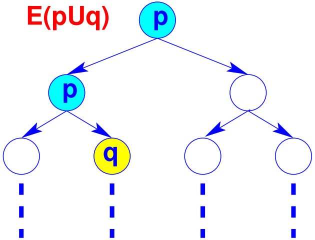
there exists a path along which p holds forever.

Existential Path Quantification

EF p



E(pUq)



There exists a path along which p holds eventually.

There exists a path along which p holds until q holds.

Duality between Always & Eventual operators

- $Gp = p \wedge (\textbf{next}\ p) \wedge (\textbf{next next}\ p) \wedge (\textbf{next next next}\ p) \ldots$

  $= \neg(\neg(p \wedge (\textbf{next}\ p) \wedge (\textbf{next next}\ p) \wedge (\textbf{next next next}\ p) \wedge \ldots))$

  applying De Morgan's law

  $= \neg(\neg p \vee (\textbf{next}\ \neg p) \vee (\textbf{next next}\ \neg p) \vee (\textbf{next next next}\ \neg p) \vee \ldots)$

  $= \neg(F\neg p)$

- **Therefore we have**
  - $Gp = \neg F \neg p$
  - $Fp = \neg G \neg p$

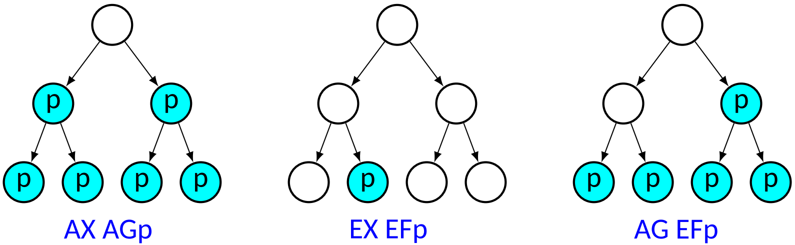
Computation Tree Logic (CTL)

- **Syntax:**
  - **Given a set of Atomic Propositions (AP):**
    - **All Boolean formulas of over AP are CTL properties**
    - **If f and g are CTL properties then so are $\neg f$, AXf, $A(f\,U\,g)$, EXf and $E(f\,U\,g)$,**
  - **Properties like AFp, AGp, EGp, EFp can be derived from the above**

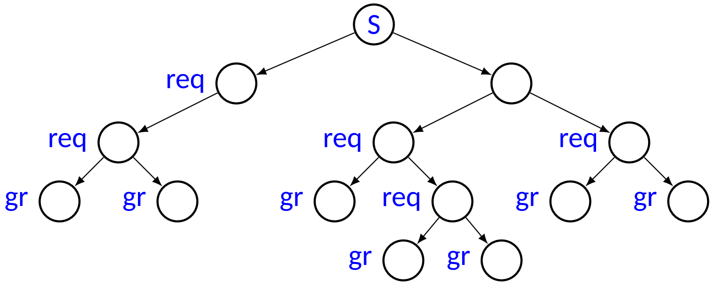
- **Semantics:**
  - **The property Af is true at a state s of the Kripke structure iff the path property f holds on all paths starting from s**
  - **The property Ef is true at a state s of the Kripke structure iff the path property f holds on some path starting from s**

Nested properties in CTL

- *AX AGp*
  - **From all the next state *p* holds forever along all paths**
- *EX EFp*
  - **There exist a next state from where there exist a path to a state where *p* holds**
- *AG EFp*
  - **From any state there exist a path to a state where *p* holds**

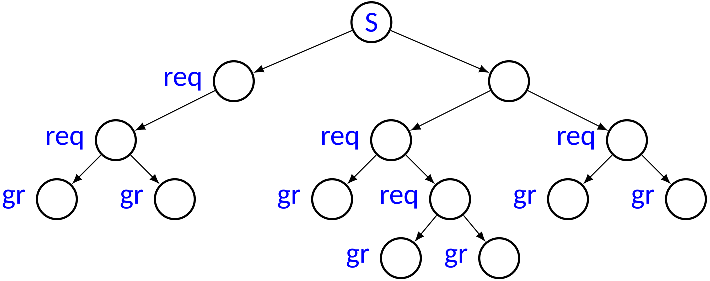


CTL example



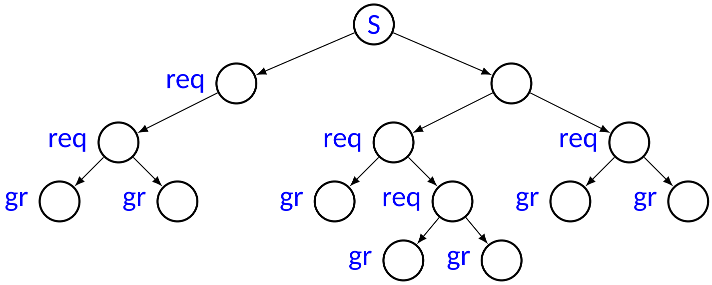
- **From S the system always makes a request in future:**

CTL example



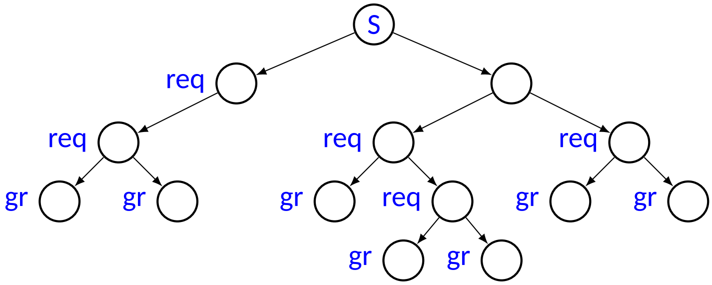
- **From *S* the system always makes a request in future:** *AF req*

CTL example



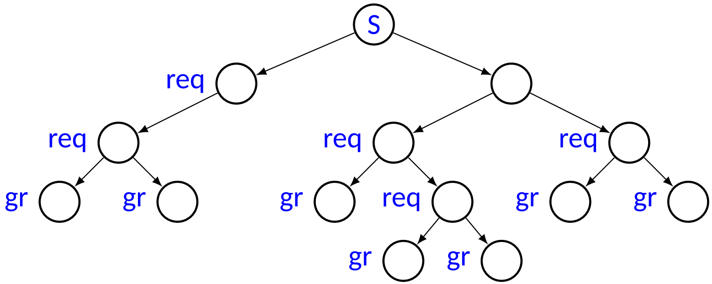
- **From *S* the system always makes a request in future:** *AF req*
- **All requests are eventually granted:**

CTL example



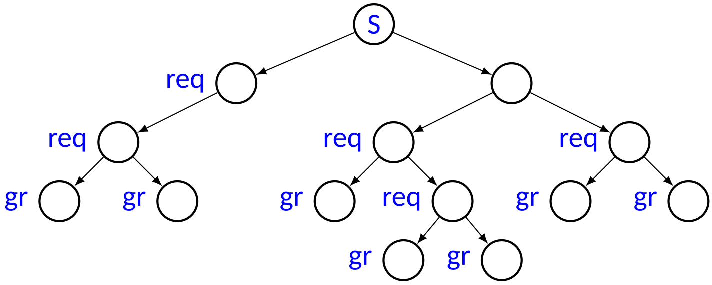
- **From S the system always makes a request in future:** $AF\,req$
- **All requests are eventually granted:** $AG(req \rightarrow AF\,gr)$

CTL example



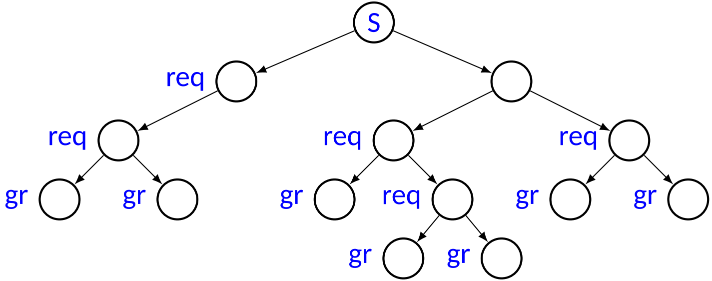
- **From S the system always makes a request in future:** $AF\, req$
- **All requests are eventually granted:** $AG(req \rightarrow AF\, gr)$
- **Sometimes requests are immediately granted:**

CTL example



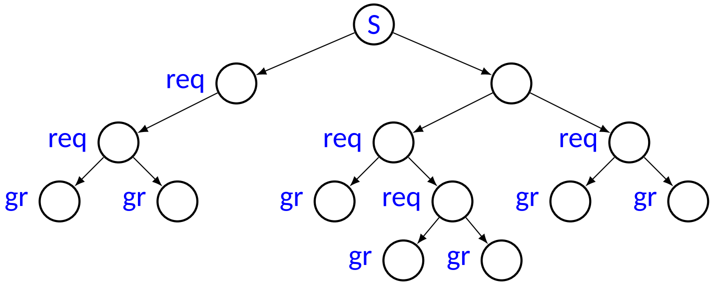
- **From** S **the system always makes a request in future:** $AF\, req$
- **All requests are eventually granted:** $AG(req \rightarrow AF\, gr)$
- **Sometimes requests are immediately granted:** $EF(req \rightarrow EX\, gr)$

CTL example



- **From S the system always makes a request in future:** $AF\,req$
- **All requests are eventually granted:** $AG(req \rightarrow AF\,gr)$
- **Sometimes requests are immediately granted:** $EF(req \rightarrow EX\,gr)$
- **Requests are held till grant is received:**

CTL example



- **From** S **the system always makes a request in future:** $AF\, req$
- **All requests are eventually granted:** $AG(req \rightarrow AF\, gr)$
- **Sometimes requests are immediately granted:** $EF(req \rightarrow EX\, gr)$
- **Requests are held till grant is received:** $AG(req \rightarrow A(req\, U\, gr))$

Real Time properties

- **Real time systems**
  - **Predictable response time are necessary for correct operation**
  - **Safety critical systems like controller for aircraft, industrial machinery are a few examples**

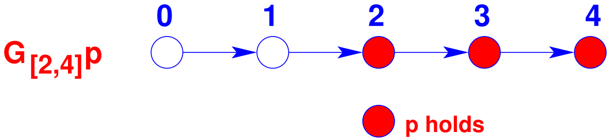
- **It is difficult to express complex timing properties**
  - **Simple: “event p will happen in future”**
    - Fp
  - **Complex: “event p will happen within at most n time units”**
    - $p \vee (Xp) \vee (XXp) \vee \ldots ([XX \ldots \textbf{n times}]p)$

Bounded Temporal Operators

- **Specify real-time constraints**
  - **Over bounded traces**

- **Various bounded temporal operators**
  - $G_{[m,n]}p$ — p **always holds between mth and nth time step**
  - $F_{[m,n]}p$ — p **eventually holds between mth and nth time step**
  - $X_{[m]}p$ — p **holds at the mth time step**
  - $p\, U_{[m,n]}\, q$ — q **eventually holds between mth and nth time step and** p **holds until that point of time**

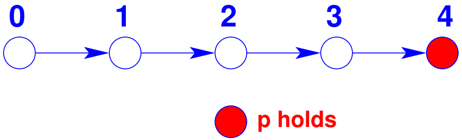
Examples



- p holds always between 2nd and 4th time step

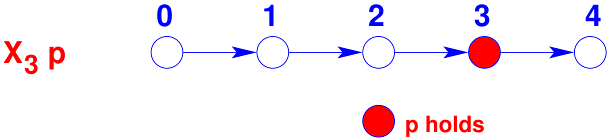
Examples

$F_{[2,4]}p$



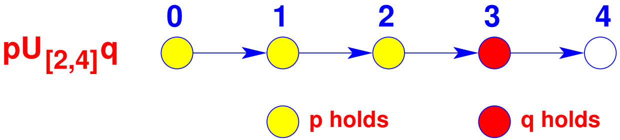
- p holds eventually between 2nd and 4th time step

Examples



- p holds in the 3rd time step

Examples

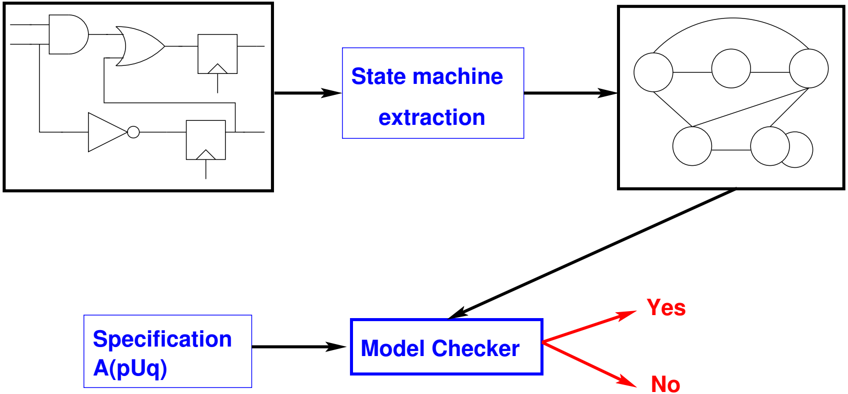


- q holds eventually between 2nd and 4th time step and p holds until q holds

Timing properties

- **Whenever request is recorded grant should take place within 4 time units**
  - $AG(posedge(req) \rightarrow AF_{[0,4]}\, posedge(gr))$
- **The arbiter will provide exactly 64 time units to high priority user in each grant**
  - $AG(posedge(hpusing) \rightarrow$
    $A(\neg negedge(hpusing)\, U_{[64,64]}\, negedge(hpusing)))$

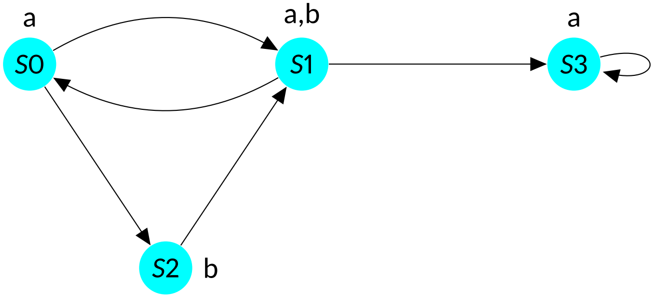
Formal Verification



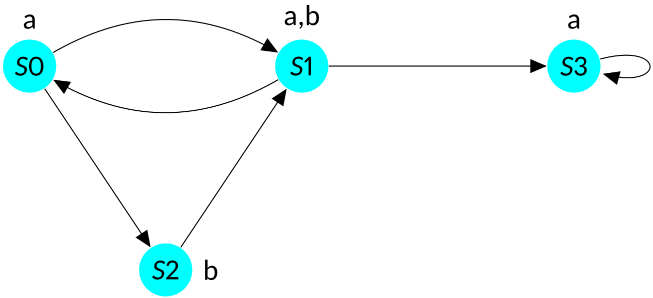
Formal Property Verification

- The formal method is called “Model Checking”
  - The algorithm has two inputs
    - A finite state state machine (FSM) that represents the implementation
    - A formal property that represent 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

Example: Explicit State Model

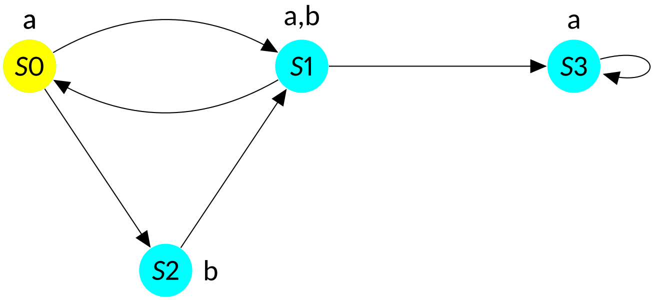


Example: EX



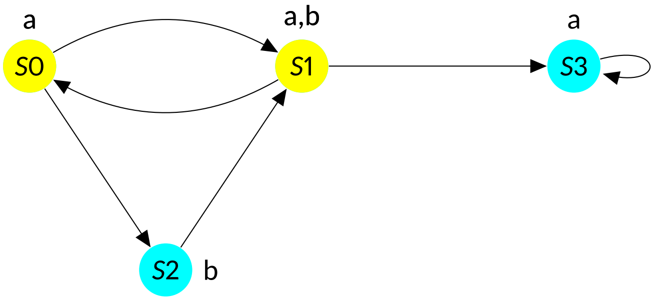
EXa

Example: EX



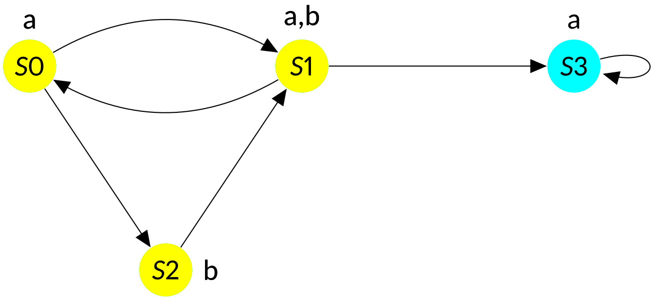
EXa

Example: EX



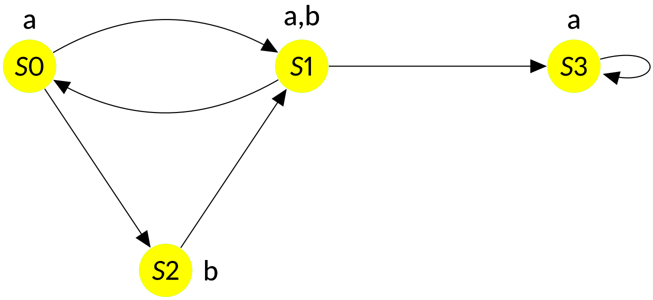
EXa

Example: EX



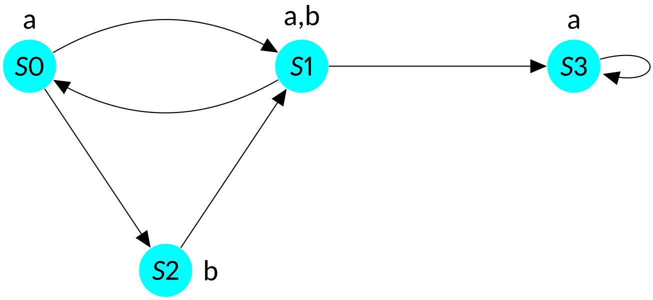
EXa

Example: EX



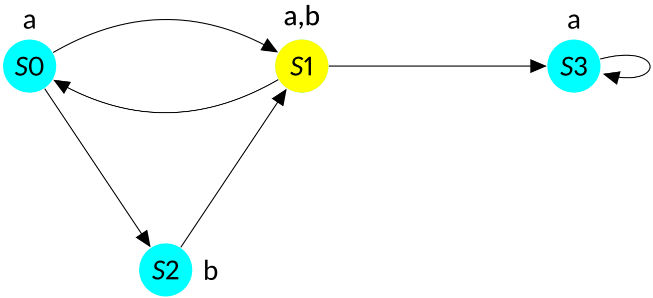
EXa

Example: AX



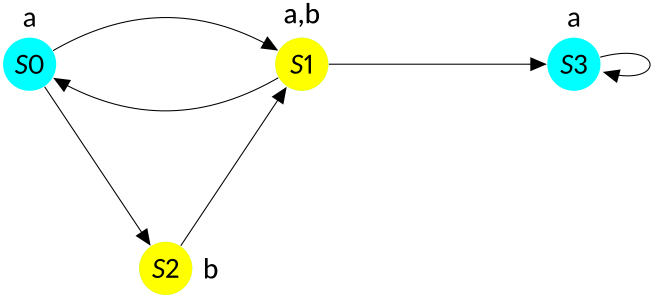
AXa

Example: AX



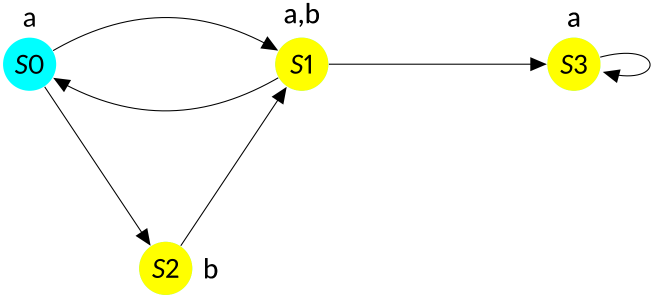
$\boxed{AXa}$

Example: AX



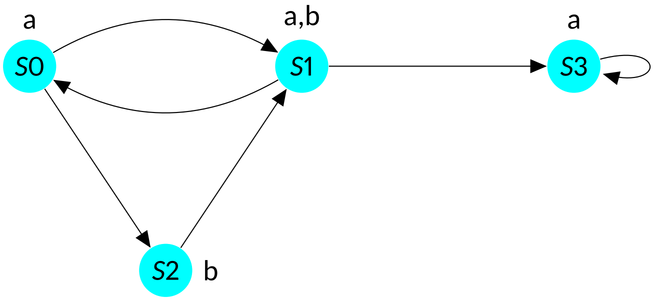
AXa

Example: AX



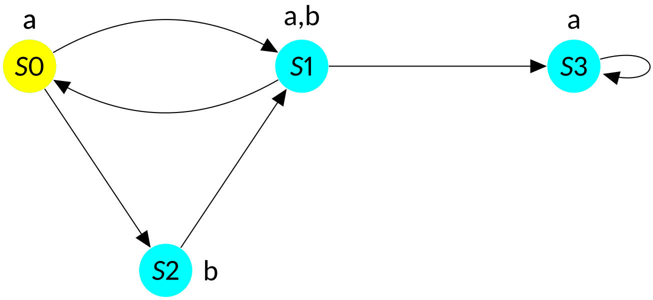
$\boxed{AXa}$

Example: EG



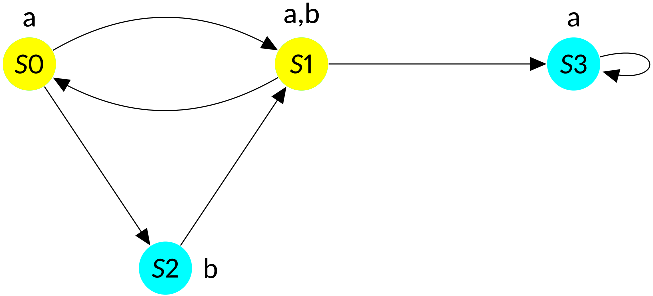
$\boxed{EGa}$

Example: EG



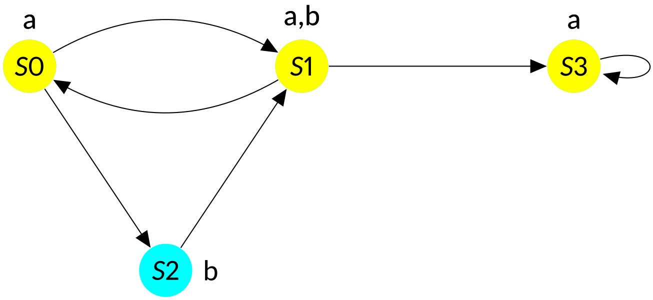
EGa

Example: EG



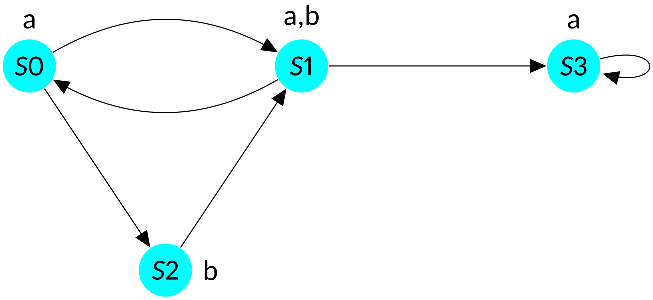
$\boxed{EGa}$

Example: EG



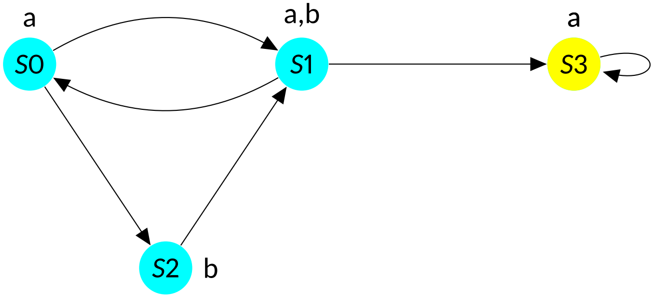
$\boxed{EGa}$

Example: AG



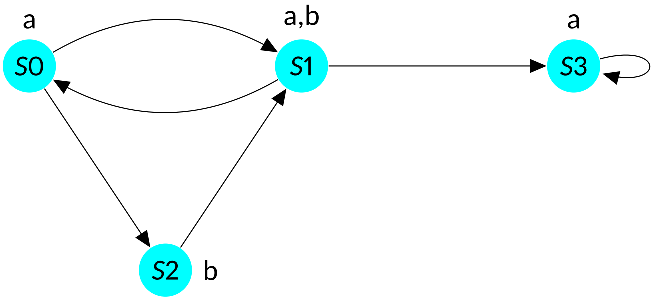
$\boxed{AGa}$

Example: AG



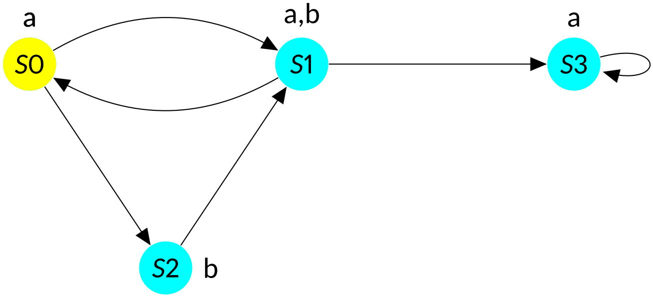
$\boxed{AGa}$

Example: AU



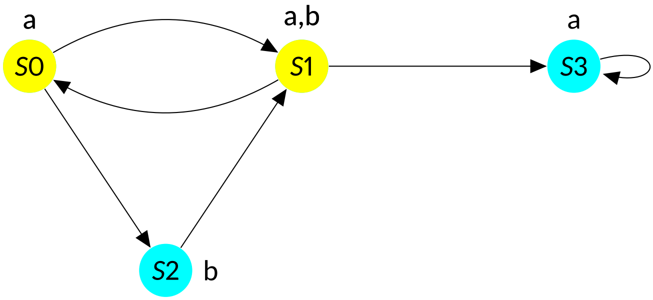
$\boxed{A(a \, U \, b)}$

Example: AU



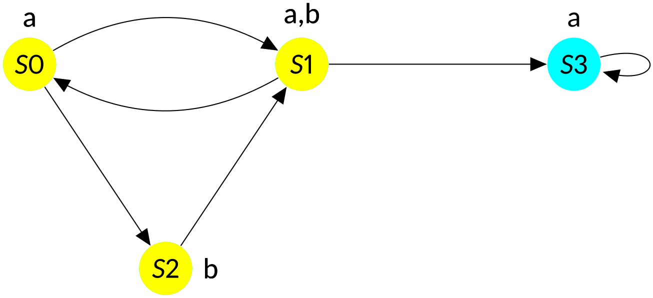
$\boxed{A(a \; U \; b)}$

Example: AU



$\boxed{A(a\ U\ b)}$

Example: AU



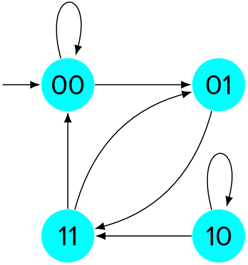
$\boxed{A(a \; U \; b)}$

Symbolic Representation

- **Represents set of transition as function $\delta(\text{old}, \text{new})$**
  - **Yields 1 if there is a transition from** old **to** new
  - **Can be represented as Boolean function by encoding the states with Boolean variables**

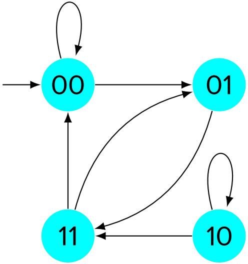
Symbolic Representation

- **Represents set of transition as function** $\delta(\mathrm{old}, \mathrm{new})$
  - **Yields 1 if there is a transition from** old **to** new
  - **Can be represented as Boolean function by encoding the states with Boolean variables**



Symbolic Representation

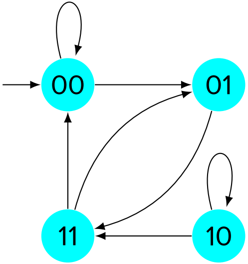
- **Represents set of transition as function** $\delta(\text{old}, \text{new})$
  - **Yields 1 if there is a transition from** old **to** new
  - **Can be represented as Boolean function by encoding the states with Boolean variables**



- o_1, o_2 — **Old state variables**
- n_1, n_2 — **New state variables**

Symbolic Representation

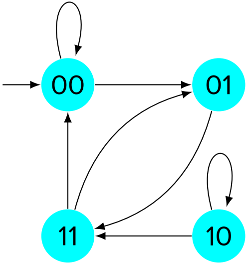
- **Represents set of transition as function $\delta(\text{old}, \text{new})$**
  - **Yields 1 if there is a transition from** old **to** new
  - **Can be represented as Boolean function by encoding the states with Boolean variables**



- o_1, o_2 — **Old state variables**
- n_1, n_2 — **New state variables**
- $\delta = \bar{o}_1\bar{o}_2\bar{n}_1 n_2 \vee \bar{o}_1\bar{o}_2\bar{n}_1\bar{n}_2 \vee \bar{o}_1 o_2 n_1 n_2 \vee o_1\bar{o}_2 n_1\bar{n}_2$
  $\vee\ o_1\bar{o}_2 n_1 n_2 \vee o_1 o_2\bar{n}_1\bar{n}_2 \vee o_1 o_2\bar{n}_1 n_2$

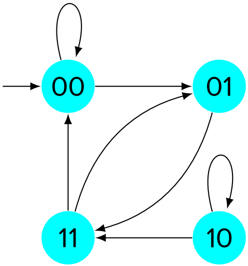
Symbolic Representation

- **Represents set of transition as function** $\delta(\text{old}, \text{new})$
  - **Yields 1 if there is a transition from** old **to** new
  - **Can be represented as Boolean function by encoding the states with Boolean variables**



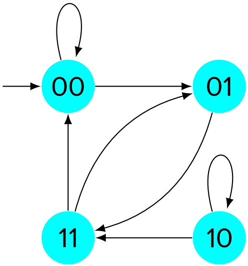
- o_1, o_2 — **Old state variables**
- n_1, n_2 — **New state variables**
- $\delta = \bar{o}_1\bar{o}_2\bar{n}_1 n_2 \vee \bar{o}_1\bar{o}_2\bar{n}_1\bar{n}_2 \vee \bar{o}_1 o_2 n_1 n_2 \vee o_1\bar{o}_2 n_1\bar{n}_2$
  $\vee\, o_1\bar{o}_2 n_1 n_2 \vee o_1 o_2\bar{n}_1\bar{n}_2 \vee o_1 o_2 \bar{n}_1 n_2$
- **New states =**$\exists\langle o_1 o_2\rangle[S(\langle o_1 o_2\rangle) \wedge \delta(\langle o_1 o_2\rangle, \langle n_1 n_2\rangle)]$

Breadth First Reachability



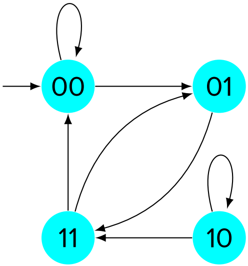
- R_i **is the set of states that can be reached in** *i* **transitions**
- **Reaches fix point when** $R_n = R_{n+1}$
  - **Fix point always exists as it has finite number of states**

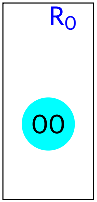
Breadth First Reachability



- R_i **is the set of states that can be reached in** i **transitions**
- **Reaches fix point when** $R_n = R_{n+1}$
  - **Fix point always exists as it has finite number of states**

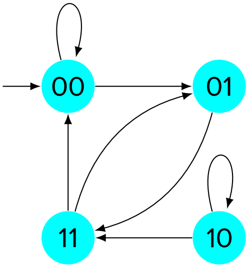
Breadth First Reachability

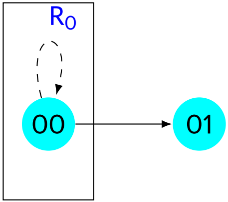




- R_i **is the set of states that can be reached in *i* transitions**
- **Reaches fix point when** $R_n = R_{n+1}$
  - **Fix point always exists as it has finite number of states**

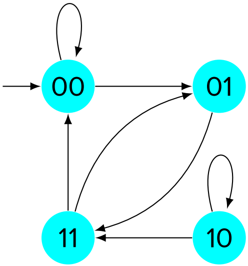
Breadth First Reachability

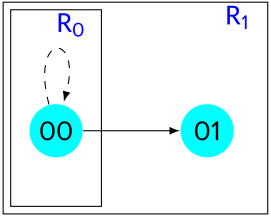




- R_i **is the set of states that can be reached in *i* transitions**
- **Reaches fix point when** $R_n = R_{n+1}$
  - **Fix point always exists as it has finite number of states**

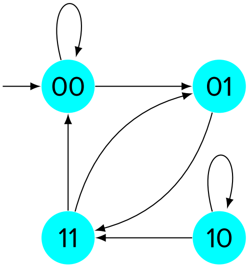
Breadth First Reachability

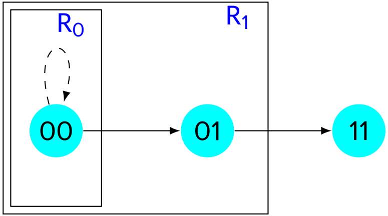




- R_i **is the set of states that can be reached in *i* transitions**
- **Reaches fix point when** $R_n = R_{n+1}$
  - **Fix point always exists as it has finite number of states**

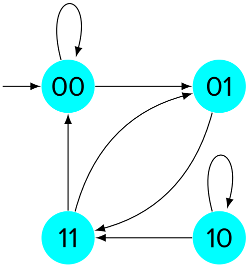
Breadth First Reachability

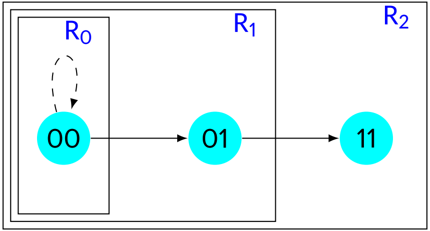




- R_i **is the set of states that can be reached in *i* transitions**
- **Reaches fix point when** $R_n = R_{n+1}$
  - **Fix point always exists as it has finite number of states**

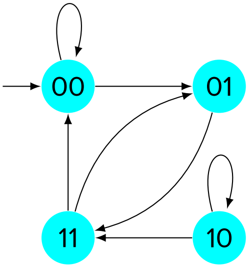
Breadth First Reachability

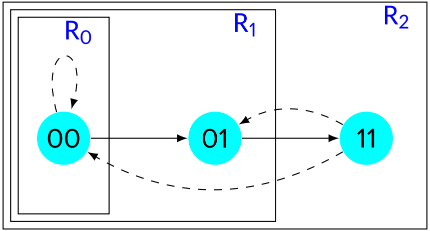




- R_i **is the set of states that can be reached in** ***i*** **transitions**
- **Reaches fix point when** $R_n = R_{n+1}$
  - **Fix point always exists as it has finite number of states**

Breadth First Reachability



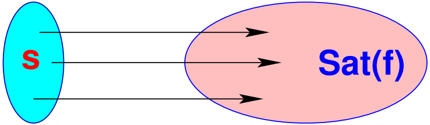


- R_i **is the set of states that can be reached in** i **transitions**
- **Reaches fix point when** $R_n = R_{n+1}$
  - **Fix point always exists as it has finite number of states**

CTL Model Checking

- It checks whether a given CTL formula f holds on a given Kripke structure M i.e., $M \models f$
- Need to have modalities for EX, EU and EG
  - Other modalities can be expressed using EX, EU and EG
    - $AF\, f \equiv \neg EG \neg f$
    - $AG\, f \equiv \neg EF \neg f$
    - $A(f\, U\, g) \equiv (\neg EG \neg g) \wedge (\neg E[\neg g\, U\, (\neg f \wedge \neg g)])$
- Basic procedure
  - The set Sat(f) of all states satisfying f is computed recursively
  - $M \models f$ if and only if $S_0 \subseteq Sat(f)$

CTL Model Checking: EXf



- $\mathsf{Post}(s) = \{s' \in S \mid (s, s') \in T\}$
- $\mathsf{Sat}(EXf) = \{s \in S \mid \mathsf{Post}(s) \cap \mathsf{Sat}(f) \neq \emptyset\}$

CTL Model Checking: EXf

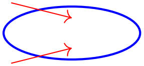
function CheckEX(f)

$S_f = \{s \in S \mid f \in L(s)\}$
while $S_f \neq \emptyset$
    Choose $s \in S_f$
    $S_f = S_f - \{s\}$
    for all t **such that** $(t, s) \in T$
        if $f \notin L(t)$
            $L(t) = L(t) \cup \{EXf\}$
        endif
    end for
end while

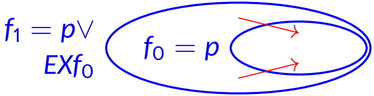
CTL Model Checking: EFp

$f_0 = p$

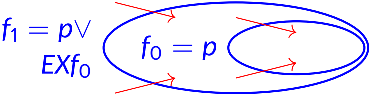
CTL Model Checking: EFp

$f_0 = p$ 

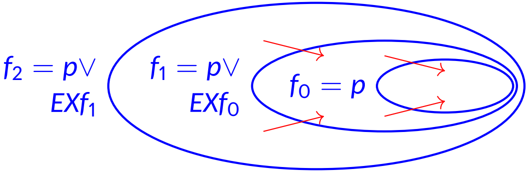
CTL Model Checking: EFp



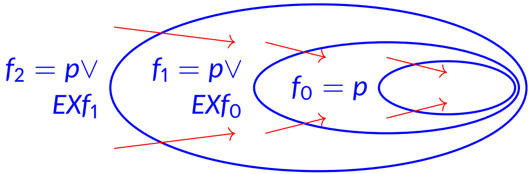
CTL Model Checking: EFp



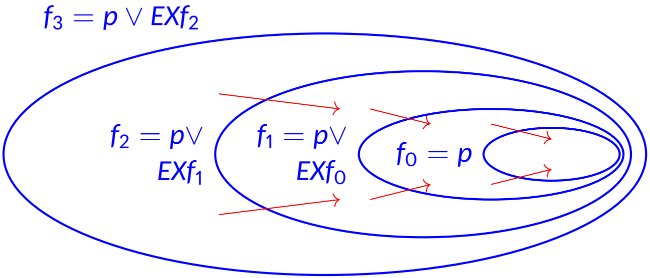
CTL Model Checking: EFp



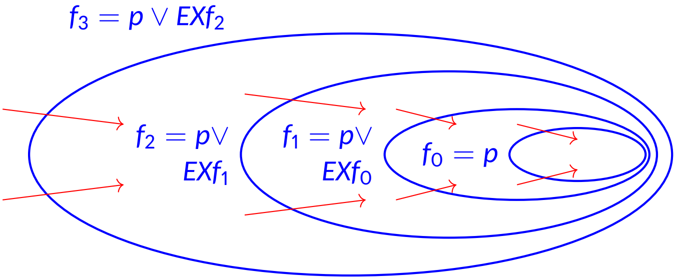
CTL Model Checking: EFp



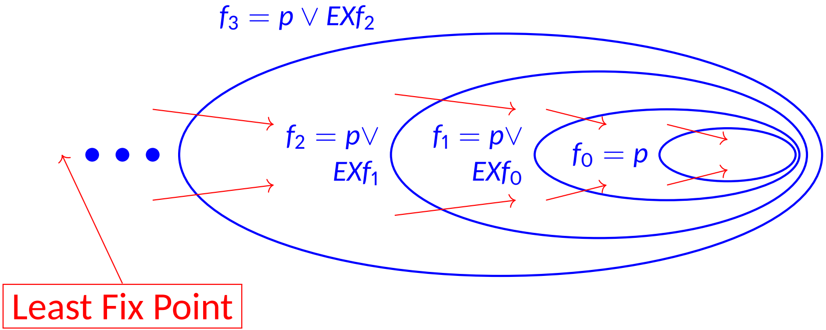
CTL Model Checking: EFp



CTL Model Checking: EFp



CTL Model Checking: EFp



CTL Model Checking: f $=$ EF p

- **Given a model $M = \langle AP, S, S_0, T, L \rangle$ and S_p the set of states satisfying p in M**

function CheckEF(S_p)

$Q \leftarrow \phi$;

$Q' \leftarrow S_p$;

while $Q \neq Q'$ **do**

$Q \leftarrow Q'$

$Q' \leftarrow Q \cup \{s \mid \exists s' [T(s, s') \wedge Q(s')]\}$

end while

$S_f \leftarrow Q'$

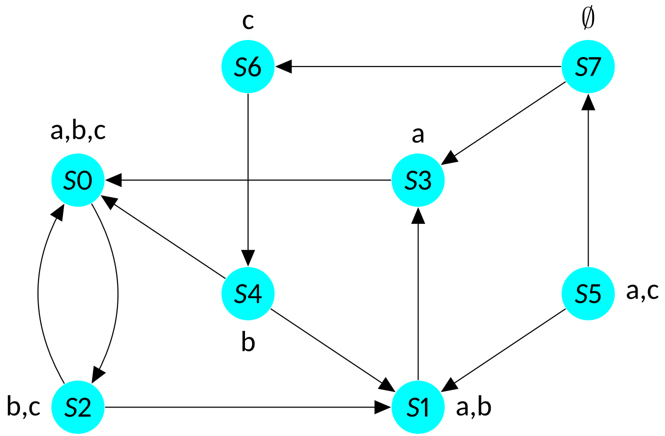
return S_f

CTL Model Checking: EFp

function CheckEF(p)

1. $S_p = \{s \in S \mid p \in L(s)\}$
2. **for all** $s \in S_p$ **do** $L(s) = L(s) \cup \{EFp\}$
3. **while** $S_p \neq \emptyset$
4. **Choose** $s \in S_p$
5. $S_p = S_p - \{s\}$
6. **for all** t **such that** $(t, s) \in T$
7. **if** $\{EFp\} \notin L(t)$
8. $L(t) = L(t) \cup \{EFp\}$
9. $S_p = S_p \cup t$
10. **endif**
11. **end for**
12. **end while**

Example: $g = \mathrm{EF}(\overline{(a \oplus c)} \wedge (a \oplus b))$

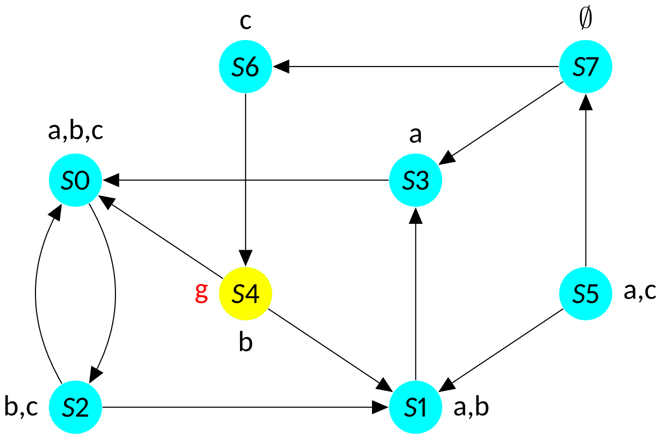


Let $p = (\overline{(a \oplus c)} \wedge (a \oplus b))$

$S_p = \{\}$

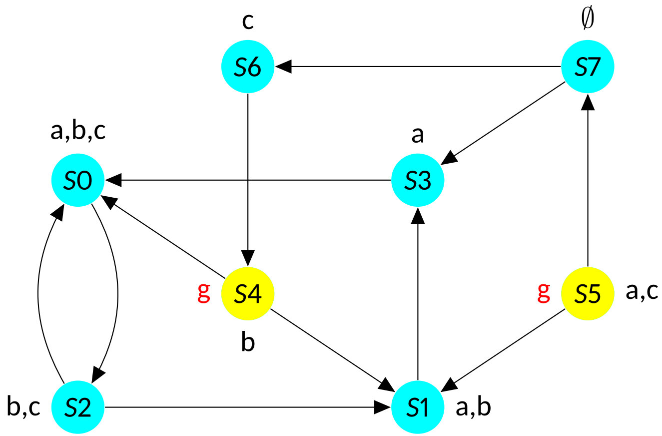
$\{\} \models g$

Example: $g = EF(\overline{(a \oplus c)} \wedge (a \oplus b))$



$\boxed{S4 \models g.}$ $\boxed{S_p = \{S4\}}$ $\boxed{\{S4\} \models g}$

Example: $g = EF(\overline{(a \oplus c)} \wedge (a \oplus b))$

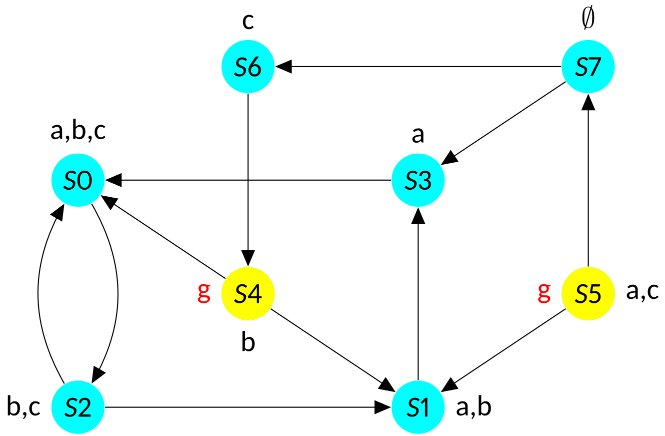


$S5 \models g.$

$S_p = \{S4, S5\}$

$\{S4, S5\} \models g$

Example: $g = \mathbf{EF}(\overline{(a \oplus c)} \wedge (a \oplus b))$

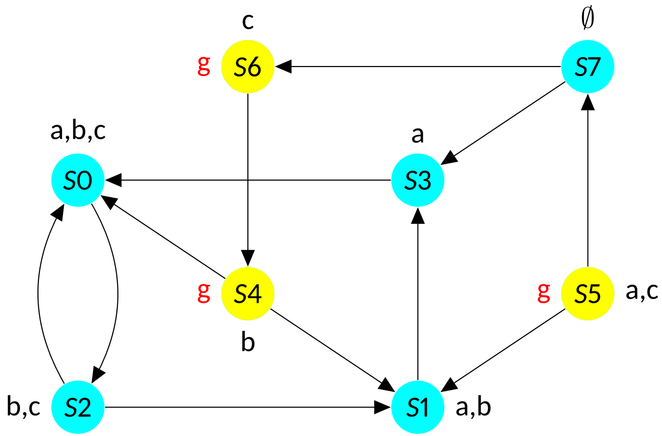


Choose S4.

$S_p = \{S5\}$

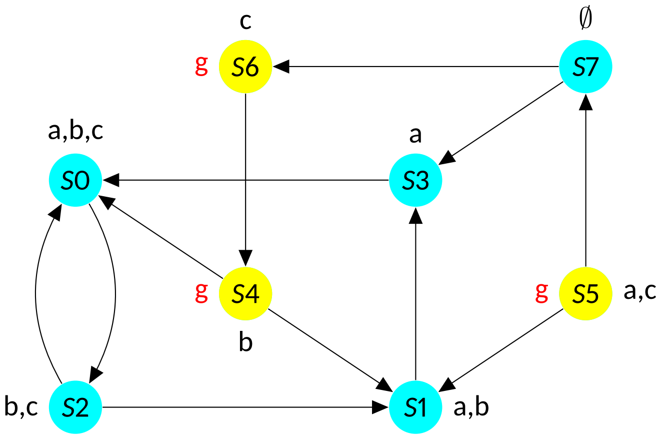
$\{S4, S5\} \models g$

Example: $g = \mathrm{EF}(\overline{(a \oplus c)} \wedge (a \oplus b))$



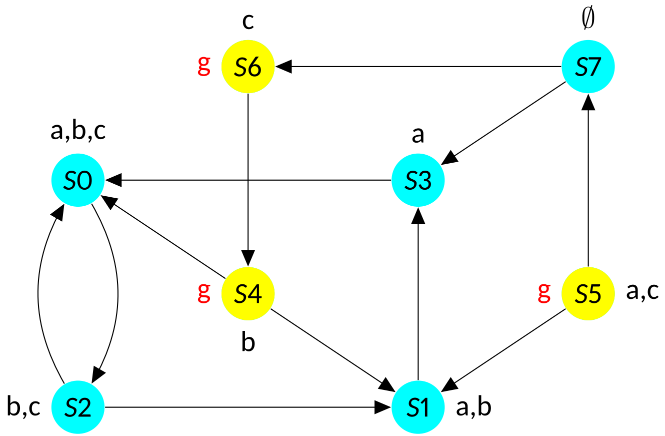
$\boxed{S6 \models g.}$ $\boxed{S_p = \{S5\}}$ $\boxed{\{S4, S5\} \models g}$

Example: $g = \mathrm{EF}(\overline{(a \oplus c)} \wedge (a \oplus b))$



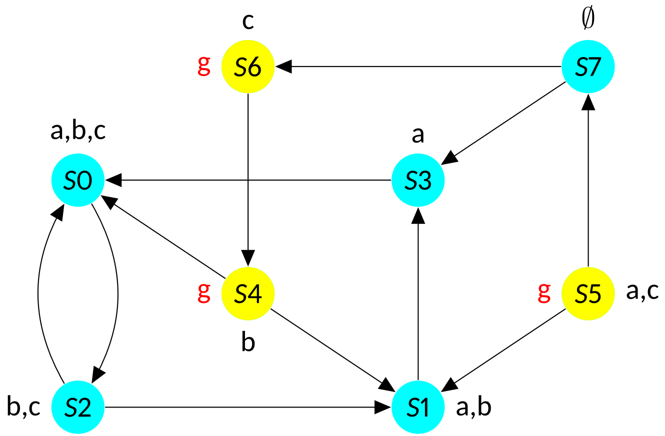
$\boxed{S_p = \{S5, S6\}}$ $\boxed{\{S4, S5, S6\} \models g}$

Example: $g = \text{EF}(\overline{(a \oplus c)} \wedge (a \oplus b))$



$\boxed{\text{Choose S5.}}$ $\boxed{S_p = \{S6\}}$ $\boxed{\{S4, S5, S6\} \models g}$

Example: $g = EF(\overline{(a \oplus c)} \wedge (a \oplus b))$

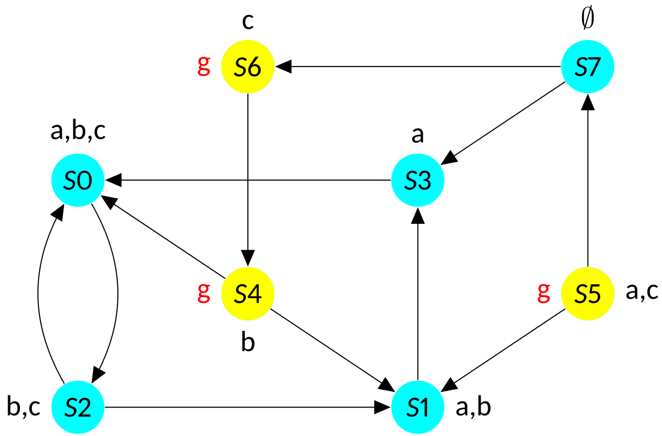


No predecessor.

$S_p = \{S6\}$

$\{S4, S5, S6\} \models g$

Example: $g = EF(\overline{(a \oplus c)} \wedge (a \oplus b))$

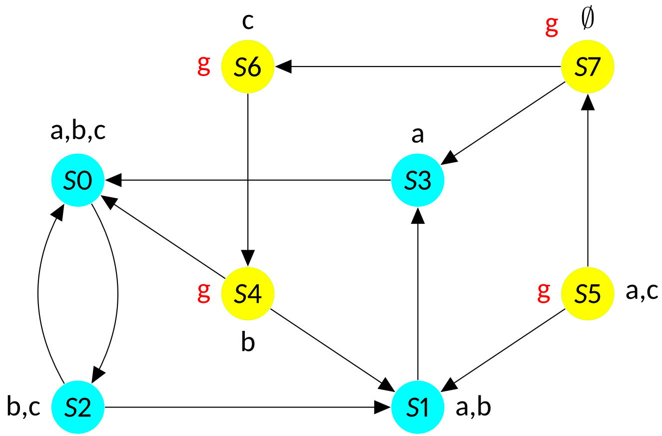


Choose S6.

$S_p = \{\}$

$\{S4, S5, S6\} \models g$

Example: $g = \mathbf{EF}(\overline{(a \oplus c)} \wedge (a \oplus b))$

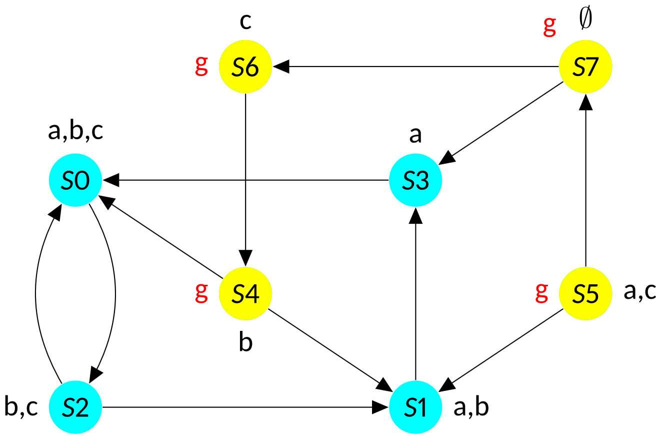


$S7 \models g.$

$S_p = \{\}$

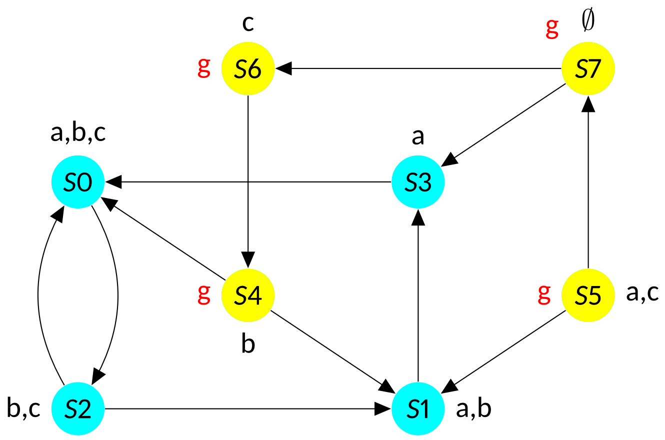
$\{S4, S5, S6\} \models g$

Example: $g = \mathbf{EF}(\overline{(a \oplus c)} \wedge (a \oplus b))$



$\boxed{S_p = \{S7\}}$ $\boxed{\{S4, S5, S6, S7\} \models g}$

Example: $g = \mathrm{EF}(\overline{(a \oplus c)} \wedge (a \oplus b))$

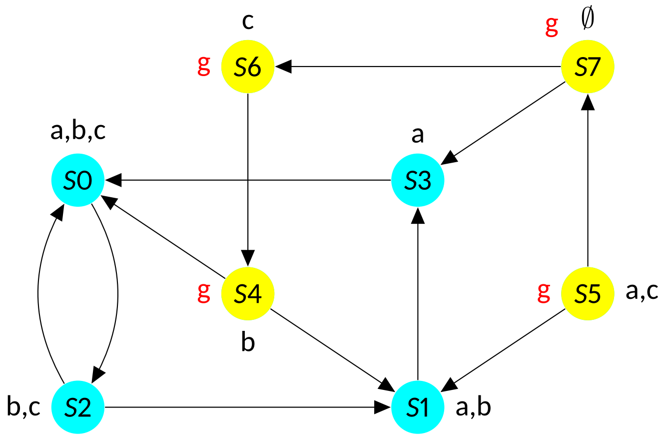


Choose S7.

$S_p = \{\}$

$\{S4, S5, S6, S7\} \models g$

Example: $g = \mathbf{EF}(\overline{(a \oplus c)} \wedge (a \oplus b))$

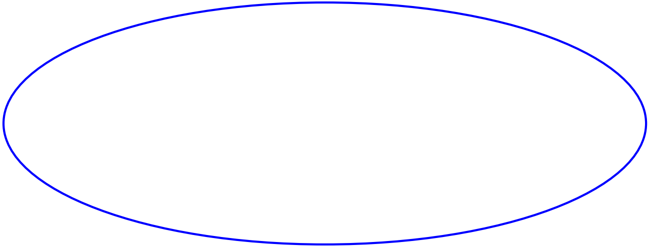


Done.

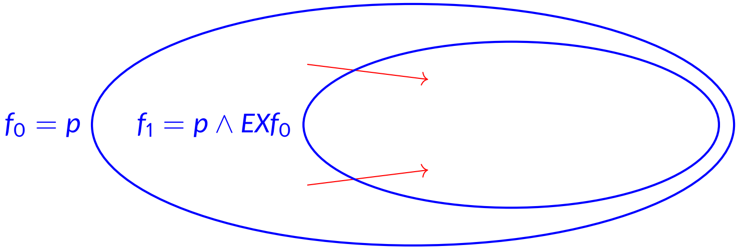
$$\boxed{\{S4, S5, S6, S7\} \models g}$$

CTL Model Checking: EGp

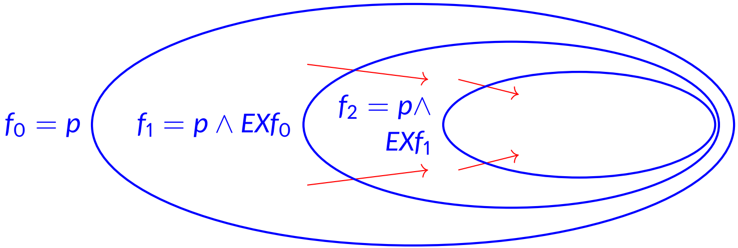
$f_0 = p$



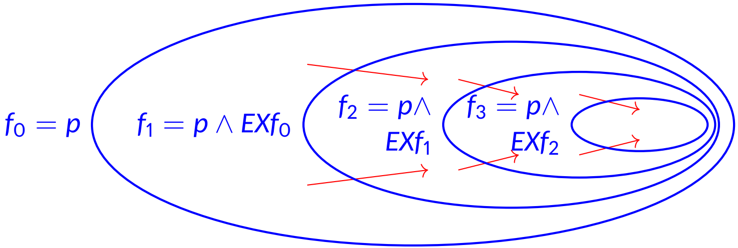
CTL Model Checking: EGp



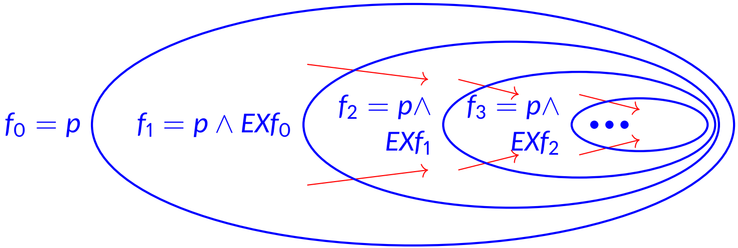
CTL Model Checking: EGp



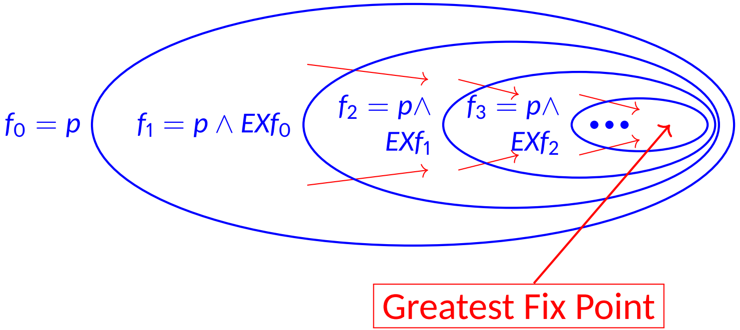
CTL Model Checking: EGp



CTL Model Checking: EGp



CTL Model Checking: EGp



CTL Model Checking: f $=$ EG p

- **Given a model $M = \langle AP, S, S_0, T, L \rangle$ and S_p the set of states satisfying p in M**

function CheckEG(S_p)
  $Q \leftarrow \phi$; $Q' \leftarrow S_p$;

  while $Q \neq Q'$ **do**
    $Q \leftarrow Q'$
    $Q' \leftarrow Q \cap \{s \mid \exists s'[T(s, s') \wedge Q(s')]\}$
  end while

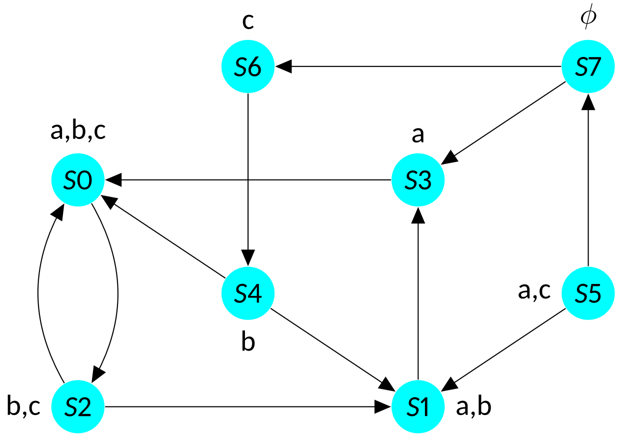
  $S_f \leftarrow Q'$
  return S_f

CTL Model Checking: EGp

function CheckEG(p)

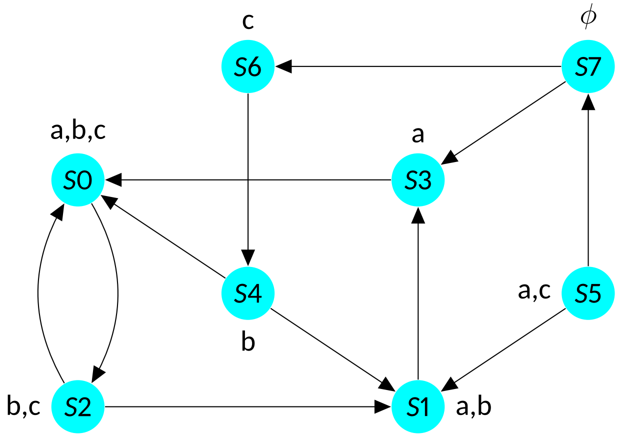
1. $S_p = \{s \in S \mid p \in L(s)\}$
2. $SCC = \{C \mid$ **C is nontrivial SCC of** $S_p\}$
3. $R = \bigcup_{C \in SCC} \{s \mid s \in C\}$
4. **for all** $s \in R$ **do** $L(s) = L(s) \cup \{EGp\}$
5. **while** $R \neq \emptyset$
6.   **Choose** $s \in R$
7. $R = R - \{s\}$
8.   **for all** t **such that** $(t, s) \in T$ **and** $t \in S_p$
9.   **if** $\{EGp\} \notin L(t)$
10. $L(t) = L(t) \cup \{EGp\}$
11. $R = R \cup \{t\}$
12.   **endif**
13.   **end for**
14. **end while**

Example: $g = \mathbf{EG\ b}$



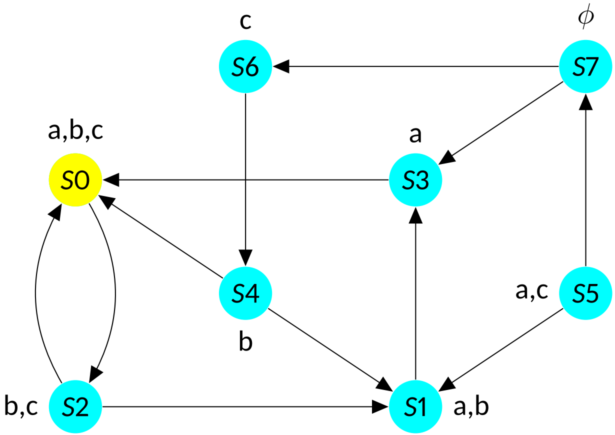
Let $p = b$

Example: g = EG b



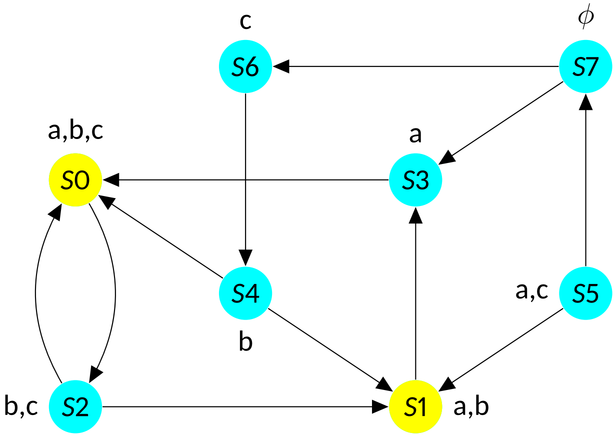
Find states satisfying b.

Example: g = EG b



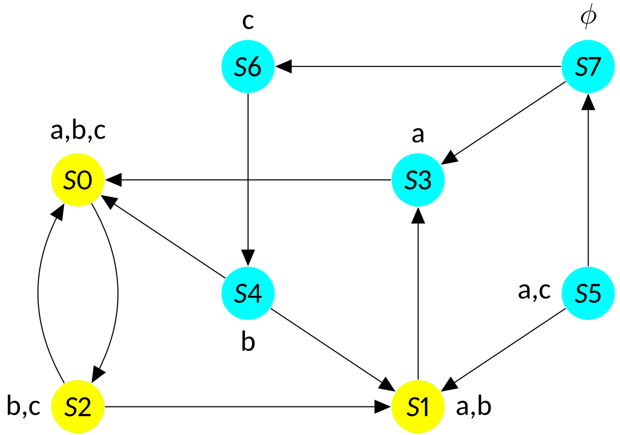
$S_p = \{S0\}$

Example: g = EG b



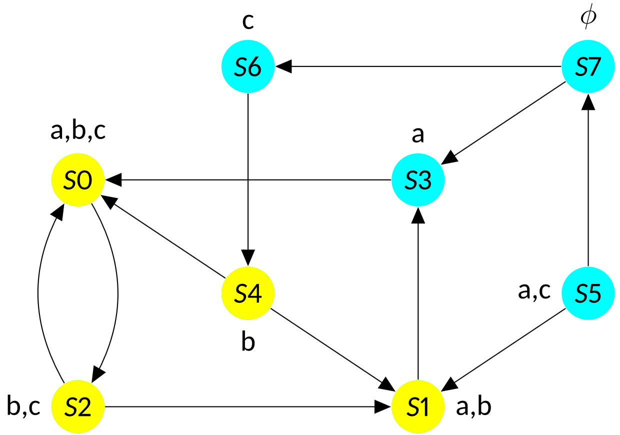
$S_p = \{S0, S1\}$

Example: $\mathsf{g} = \mathsf{EG}\ \mathsf{b}$



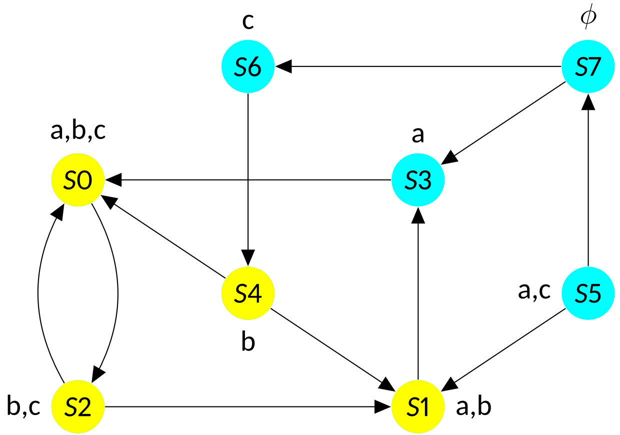
$\boxed{S_p = \{S0, S1, S2\}}$

Example: $g = \mathbf{EG}\ b$



$S_p = \{S0, S1, S2, S4\}$

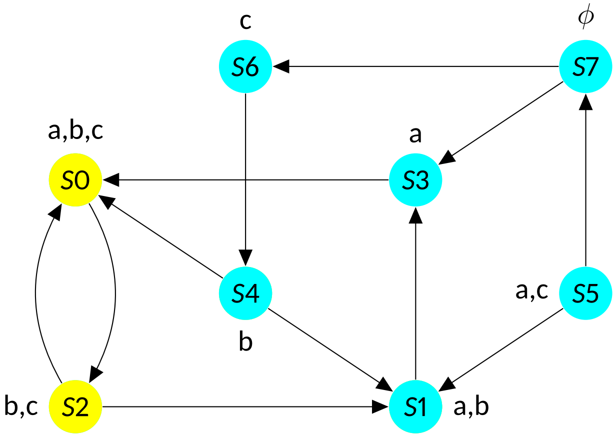
Example: g = EG b



Find SCC using S_p.

$S_p = \{S0, S1, S2, S4\}$

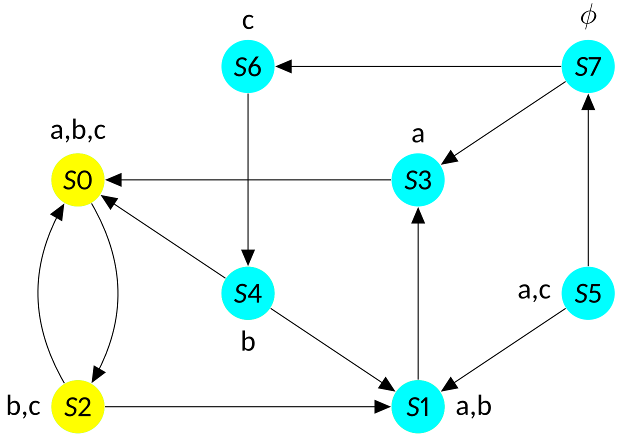
Example: $g = \mathbf{EG}\ b$



Find SCC using S_p.

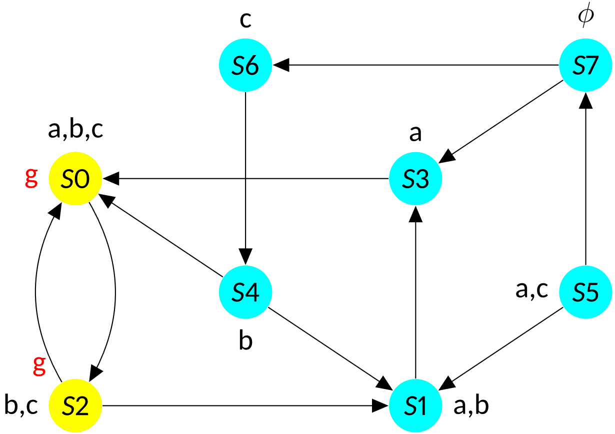
$S_p = \{S0, S1, S2, S4\}$

Example: $g = \text{EG b}$



$R = \{S0, S2\}$

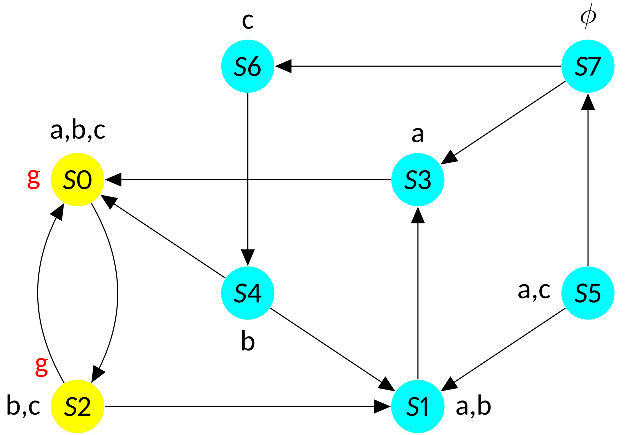
Example: g = EG b



$R = \{S0, S2\}$

$S_g = \{S0, S2\}$

Example: $g = \text{EG b}$

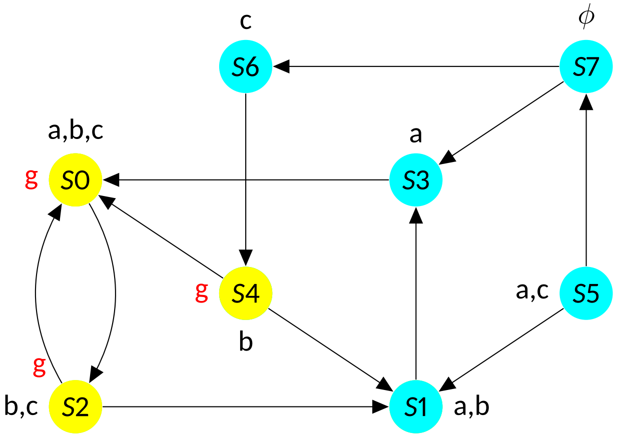


Choose S0.

$R = \{S2\}$

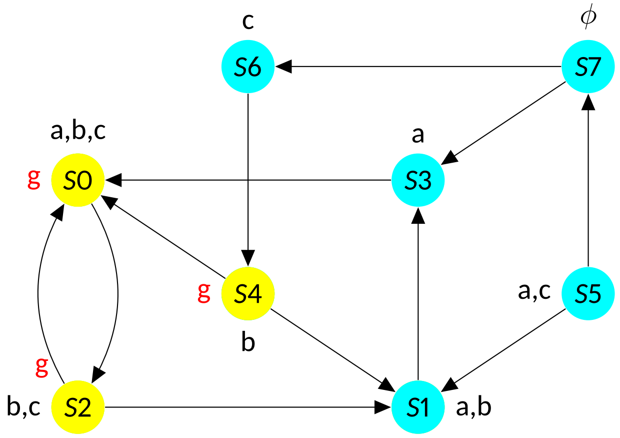
$S_g = \{S0, S2\}$

Example: g = EG b



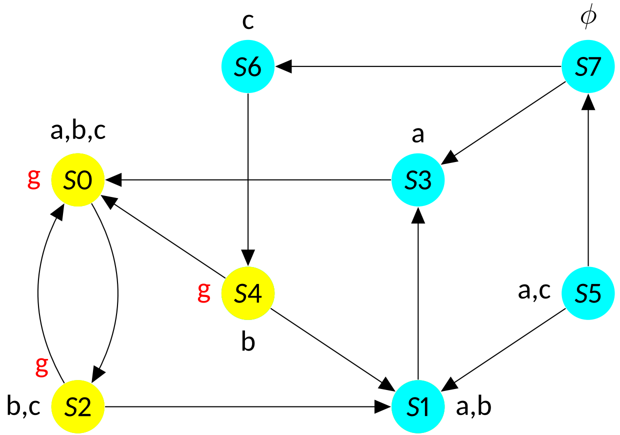
$\boxed{S4 \models g.}$ $\boxed{R = \{S2\}}$ $\boxed{S_g = \{S0, S2\}}$

Example: g = EG b



$\boxed{S4 \models g.}$ $\boxed{R = \{S2, S4\}}$ $\boxed{S_g = \{S0, S2, S4\}}$

Example: g $=$ EG b

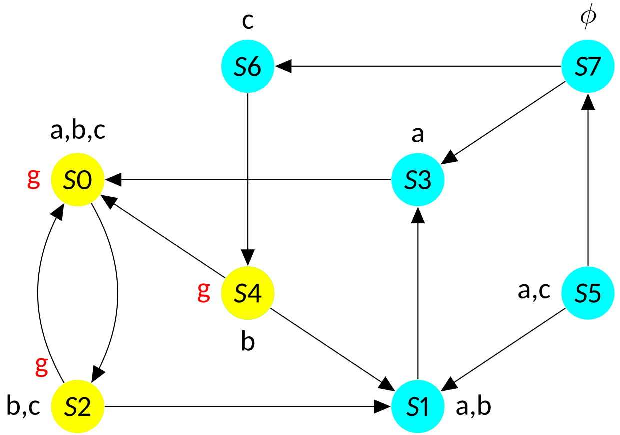


Choose S2.

$R = \{S4\}$

$S_g = \{S0, S2, S4\}$

Example: g $=$ EG b

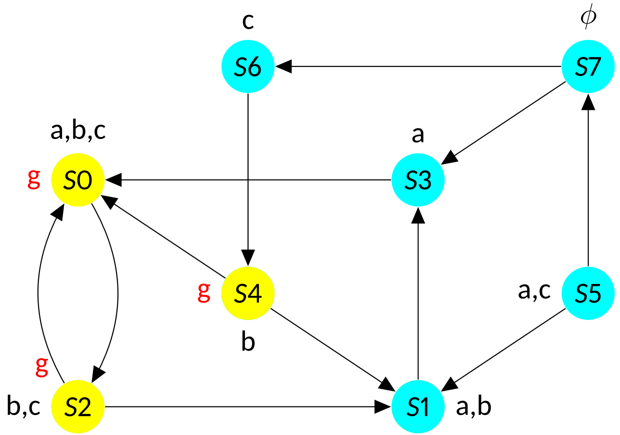


Nothing to explore.

$R = \{S4\}$

$S_g = \{S0, S2, S4\}$

Example: $g = \mathbf{EG}\ b$

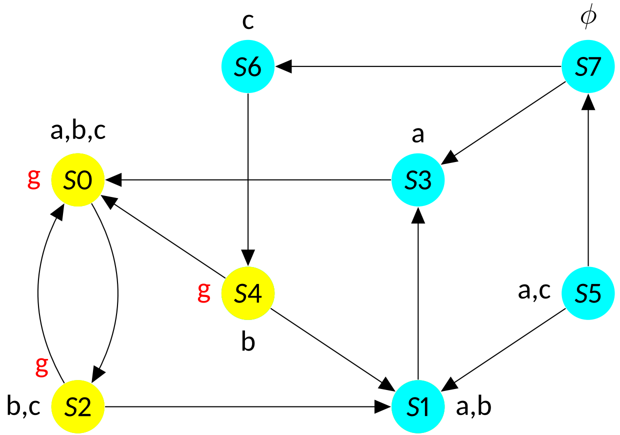


Choose S4.

$R = \{\}$

$S_g = \{S0, S2, S4\}$

Example: g = EG b

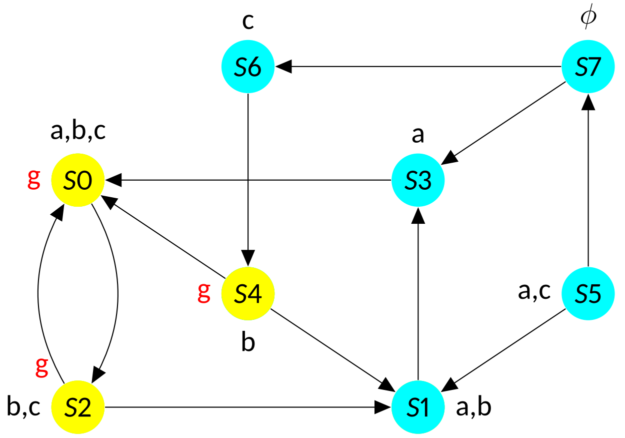


Nothing to explore.

$R = \{\}$

$S_g = \{S0, S2, S4\}$

Example: g $=$ EG b



Done.

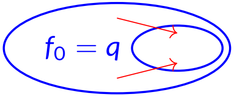
$S_g = \{S0, S2, S4\}$

CTL Model Checking: E(p U q)

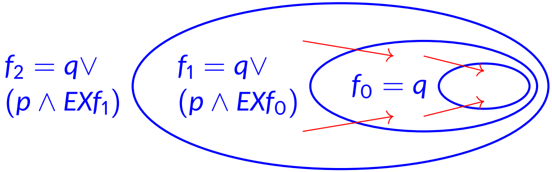
$f_0 = q$

CTL Model Checking: E(p U q)

$f_1 = q \vee$
$(p \wedge EXf_0)$

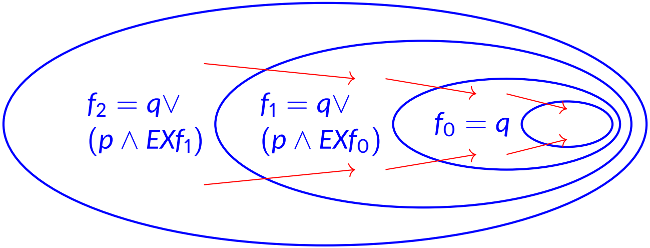


CTL Model Checking: E(p U q)

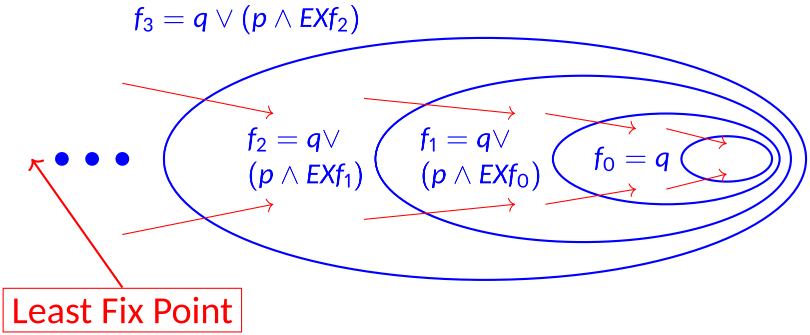


CTL Model Checking: E(p U q)

$f_3 = q \vee (p \wedge EXf_2)$



CTL Model Checking: E(p U q)



CTL Model Checking: E(p U q)

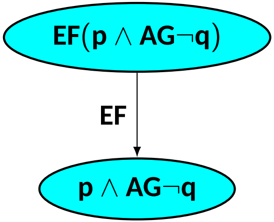
function CheckEU(p,q)

$S_q = \{s \in S \mid q \in L(s)\}$
for all $s \in S_q$ **do** $L(s) = L(s) \cup \{E(p\,U\,q)\}$
while $S_q \neq \emptyset$
    Choose $s \in S_q$
    $S_q = S_q - \{s\}$
    for all t **such that** $(t, s) \in T$
        if $\{E(p\,U\,q)\} \notin L(t)$ **and** $p \in L(t)$
            $L(t) = L(t) \cup \{E(p\,U\,q)\}$
            $S_q = S_q \cup \{t\}$
        endif
    end for
end while

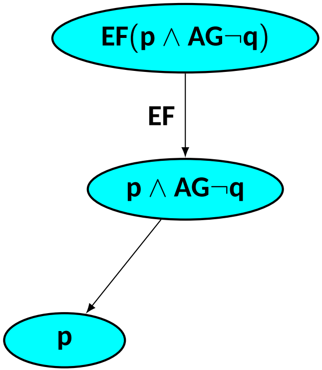
Nested CTL query

$\mathbf{EF(p \wedge AG\neg q)}$

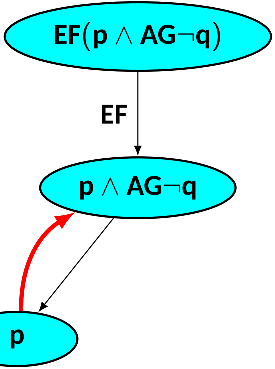
Nested CTL query



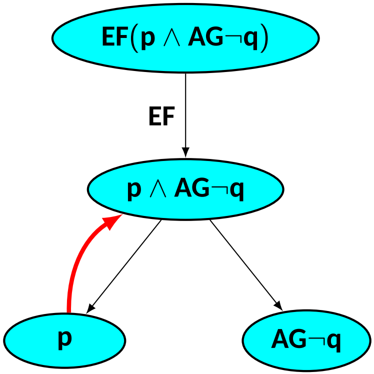
Nested CTL query



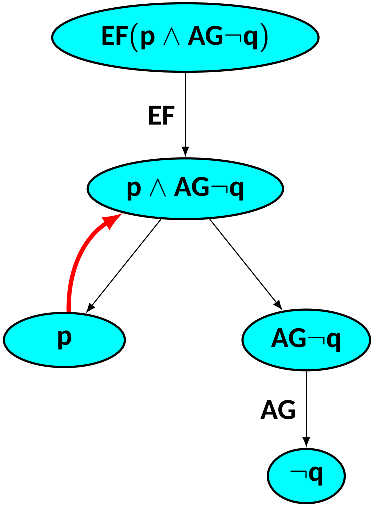
Nested CTL query



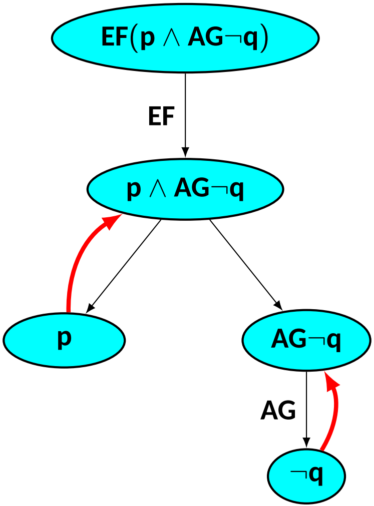
Nested CTL query



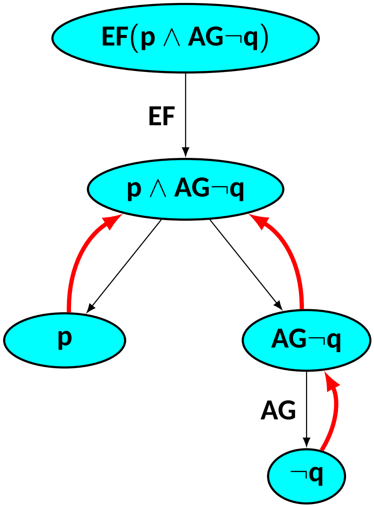
Nested CTL query



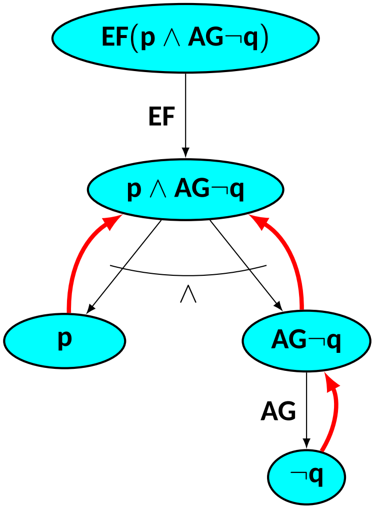
Nested CTL query



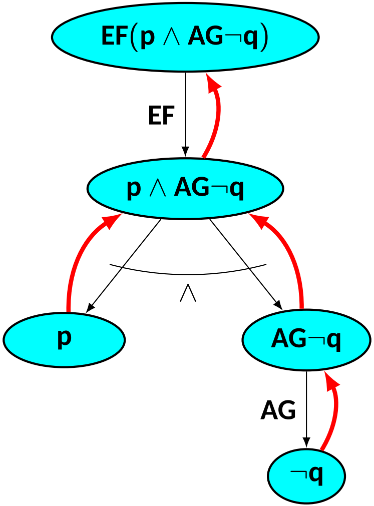
Nested CTL query



Nested CTL query



Nested CTL query



Verification of RTCTL query

- $A(p\, U_{\leq k}\, q) \equiv q \vee (p \wedge AX\, A(p\, U_{\leq k-1}\, q))$ **if** $k > 1$
- $E(p\, U_{\leq k}\, q) \equiv q \vee (p \wedge EX\, E(p\, U_{\leq k-1}\, q))$ **if** $k > 1$
- $A(p\, U_{\leq 0}\, q) \equiv q \equiv E(p\, U_{\leq 0}\, q)$

- **Similar fix point characterization of CTL modalities can be used**
- **For qualative CTL queries** $k = |S|$

RTCTL Model Checking: $f = E(p\, U_{\leq k}\, q)$

function CheckEU(p,q,k)

$N_f^0 = \{s \in S \mid q \in L(s)\}$
for all $s \in N_f^0$ **do** $L(s) = L(s) \cup \{E(p\, U_{\leq k}\, q)\}$
$j = 0;$
while $j < k$
    $TEMP = N_f^j$
    while $N_f^j \neq \emptyset$
        Choose $s \in TEMP$; $TEMP = TEMP - \{s\}$
        for all t **such that** $(t, s) \in T$
            if $\{E(p\, U_{\leq k}\, q)\} \notin L(t)$ **and** $p \in L(t)$
                $L(t) = L(t) \cup \{E(p\, U_{\leq k}\, q)\}$; $N_f^{j+1} = N_f^{j+1} \cup \{t\}$
            endif
        end for
    end while
    $j = j + 1;$
end while

Verification of RTCTL query

- $E(p\,U_{[a,b]}\,q) \equiv p \wedge (EX\,E(p\,U_{[a-1,b-1]}\,q))$ **if** $a > 0$ **and** $b > 0$
- $E(p\,U_{[0,b]}\,q) \equiv q \vee (p \wedge EX\,E(p\,U_{[0,b-1]}\,q))$ **if** $b > 0$
- $E(p\,U_{[0,0]}\,q) \equiv q$

Verification of RTCTL query

- $E(p\,U_{[a,b]}\,q) \equiv p \wedge (EX\,E(p\,U_{[a-1,b-1]}\,q))$ **if** $a > 0$ **and** $b > 0$
- $E(p\,U_{[0,b]}\,q) \equiv q \vee (p \wedge EX\,E(p\,U_{[0,b-1]}\,q))$ **if** $b > 0$
- $E(p\,U_{[0,0]}\,q) \equiv q$

- **Steps:**
  - **Compute set of states where p is true for a steps**
    - **If fix point is reached before a steps, skip to the second case**
  - **Compute set of states where E(p U q) is true for b steps**
    - **If fix point is reached before (b-a) steps, skip to the third case**

Complexity

- **Linear in the size of the model**
- **Linear in the size of the CTL formula**

- **Complexity is** $O(|F| \times M)$
  - **Model size —** M
  - **Formula size —** $|F|$