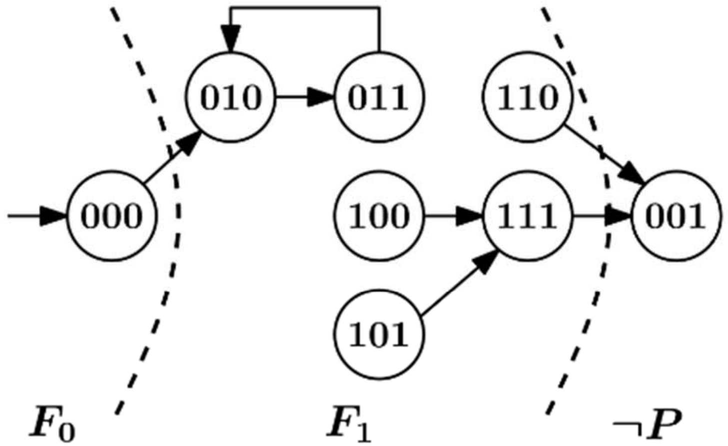
Consider the following synchronous Kripke structure K, with states of the form $x_1x_2x_3$. The only initial state is 000, and we are given a property P that holds everywhere but in 001.



We wish to prove that P is always true using PDR. We began the algorithm, obtaining the frames F_0 and F_1 as shown in the figure.

Task 4a [4 points]. Starting from the figure, carry out two iterations of the first variant of the PDR (from k=1, until k=3) shown in class. Clearly indicate the steps and the frames at the end of each iteration. Is the property P verified at the end? Why/Why not?

function PDR(Model M)

    if SAT($S_0 \wedge \neg P$) **or** SAT($S_0 \wedge R \wedge \neg P'$) **then FAIL**

    F_0 := S_0; F_1 := P; k :=1;

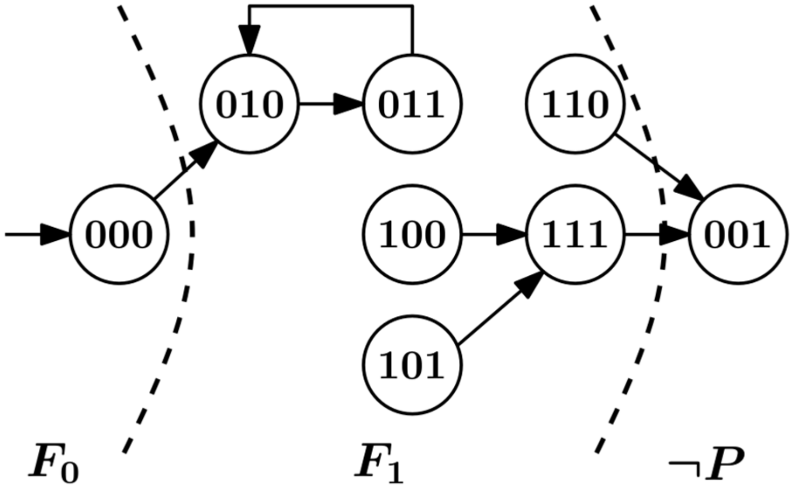
    while(true)

        while(s := SAT($F_k \wedge R \wedge \neg P'$))

            removeBad(k, s)

        k++; F_k := P

        if $\exists 0 \leq i < k-1 : F_i := F_{i+1}$ **then SUCCEED**



// post: $\neg SAT(F_i \wedge s)$

function removeBad($i \in N$, state s)

    if SAT($S_0 \wedge c$) **then FAIL**

    while(t := SAT($F_{i-1} \wedge R \wedge s'$))

        removeBad($i-1, t$)

    $\forall 0 < j \leq i : F_j \coloneqq F_j \wedge \neg$s

function PDR(Model M)

if SAT($S_0 \wedge \neg P$) **or** SAT($S_0 \wedge R \wedge \neg P'$) **then FAIL**

F_0 := S_0; F_1 := P; k :=1;

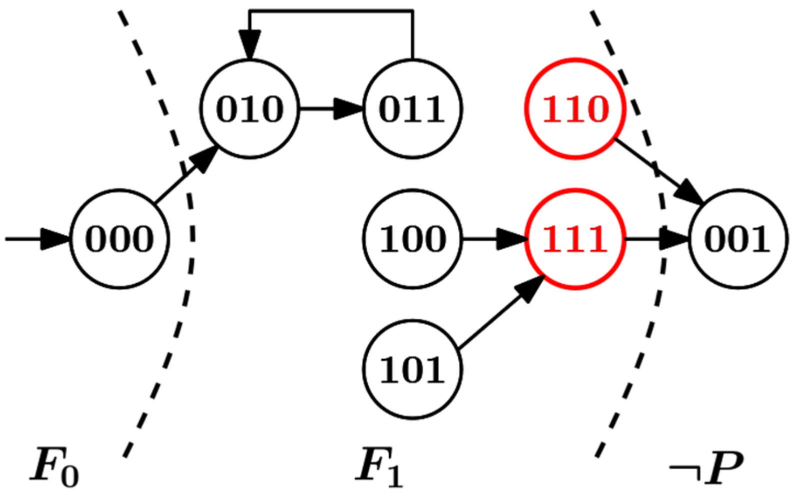
while(true)

while(s := SAT($F_k \wedge R \wedge \neg P'$))

removeBad(k, s)

k++; F_k := P

if $\exists 0 \leq i < k-1 : F_i := F_{i+1}$ **then SUCCEED**



// post: $\neg SAT(F_i \wedge s)$

function removeBad($i \in N$, state s)

if SAT($S_0 \wedge c$) **then FAIL**

while(t := SAT($F_{i-1} \wedge R \wedge s'$))

removeBad($i-1, t$)

$\forall 0 < j \leq i : F_j \coloneqq F_j \wedge \neg$s

function PDR(Model M)

  if SAT($S_0 \wedge \neg P$) **or** SAT($S_0 \wedge R \wedge \neg P'$) **then FAIL**

  F_0 := S_0; F_1 := P; k :=1;

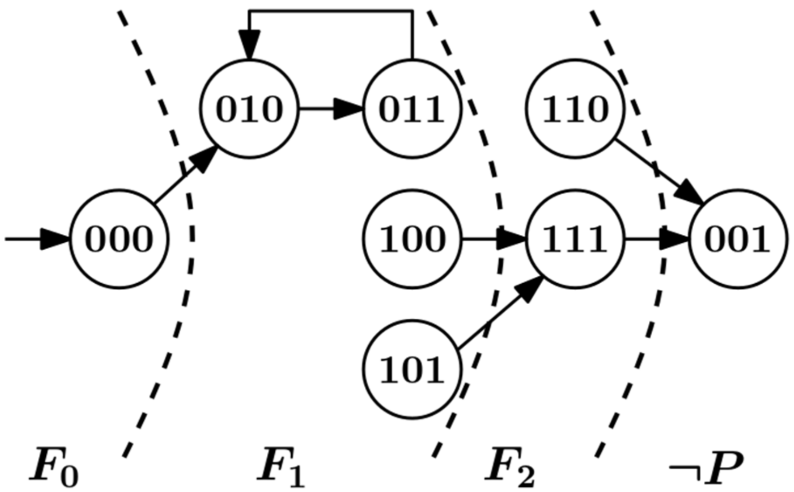
  while(true)

    while(s := SAT($F_k \wedge R \wedge \neg P'$))

      removeBad(k, s)

    k++; F_k := P

    if $\exists 0 \leq i < k-1 \colon F_i := F_{i+1}$ **then SUCCEED**



// post: $\neg SAT(F_i \wedge s)$

function removeBad($i \in N$, state s)

  if SAT($S_0 \wedge c$) **then FAIL**

  while(t := SAT($F_{i-1} \wedge R \wedge s'$))

    removeBad($i-1, t$)

  $\forall 0 < j \leq i \colon F_j \coloneqq F_j \wedge \neg$s

function PDR(Model M)

if SAT($S_0 \wedge \neg P$) **or** SAT($S_0 \wedge R \wedge \neg P'$) **then FAIL**

F_0 := S_0; F_1 := P; k :=1;

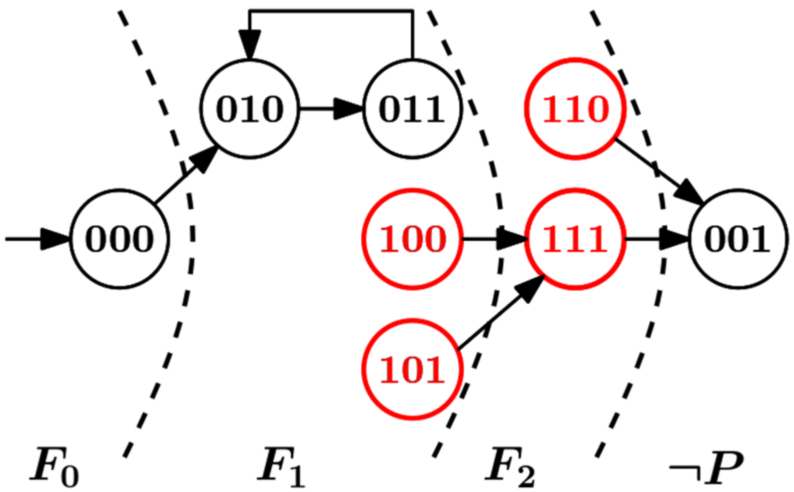
while(true)

while(s := SAT($F_k \wedge R \wedge \neg P'$))

removeBad(k, s)

k++; F_k := P

if $\exists 0 \leq i < k-1 : F_i := F_{i+1}$ **then SUCCEED**



// post: $\neg SAT(F_i \wedge s)$

function removeBad($i \in N$, state s)

if SAT($S_0 \wedge c$) **then FAIL**

while(t := SAT($F_{i-1} \wedge R \wedge s'$))

removeBad($i-1, t$)

$\forall 0 < j \leq i : F_j \coloneqq F_j \wedge \neg$s

function PDR(Model M)

if SAT($S_0 \wedge \neg P$) **or** SAT($S_0 \wedge R \wedge \neg P'$) **then FAIL**

F_0 := S_0; F_1 := P; k :=1;

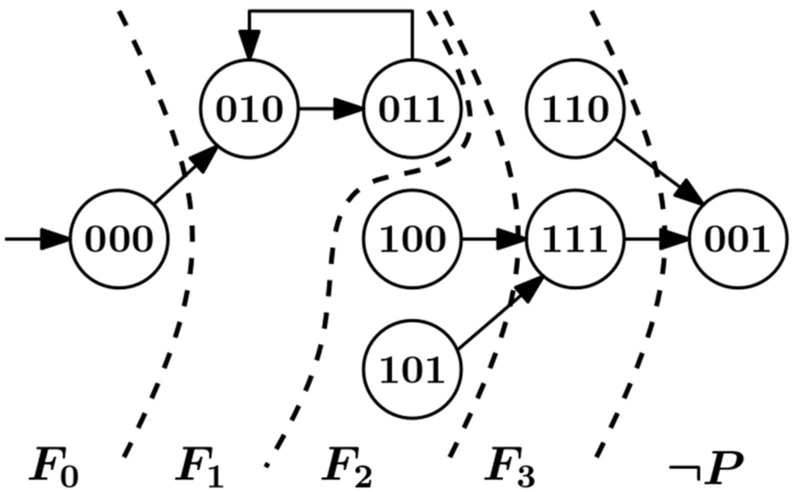
while(true)

while(s := SAT($F_k \wedge R \wedge \neg P'$))

removeBad(k, s)

k++; F_k := P

if $\exists 0 \leq i < k-1 : F_i := F_{i+1}$ **then SUCCEED**



// post: $\neg SAT(F_i \wedge s)$

function removeBad($i \in N$, state s)

if SAT($S_0 \wedge c$) **then FAIL**

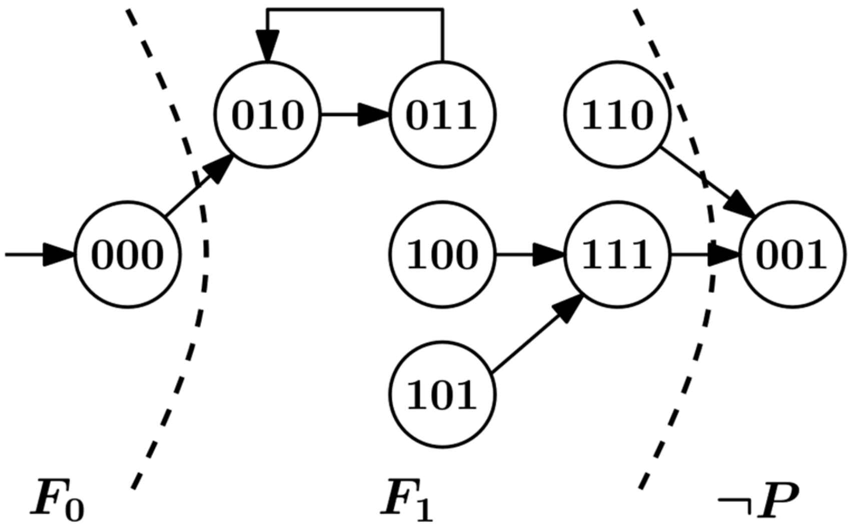
while(t := SAT($F_{i-1} \wedge R \wedge s'$))

removeBad($i-1, t$)

$\forall 0 < j \leq i : F_j \coloneqq F_j \wedge \neg$s

All frames are still different, so P is not verified.

Task 4b [3 points]. As before, perform two iterations of PDR starting from the figure. This time use "naive generalization" during the removal of bad states, as shown in class. Clearly indicate the steps and the frames at the end of each iteration. Is the property P verified at the end? Why/Why not?



function PDR(Model M)

  if SAT($S_0 \wedge \neg P$)) **or** $SAT(S\_0 \wedge R \wedge \neg P')$ **then FAIL**

  F_0 := S_0; F_1 := P; k :=1;

  while(true)

    while(s := SAT($F_k \wedge R \wedge \neg P'$))

      removeBad(k, s)

    k++; F_k := P

    if $\exists 0 \leq i < k-1 : F_i := F_{i+1}$ **then SUCCEED**

// post: $\neg SAT(F_i \wedge s)$

function removeBad($i \in N$, state s)

  if SAT($S_0 \wedge c$) **then FAIL**

  while(t := SAT($F_{i-1} \wedge R \wedge s'$))

    removeBad($i-1, t$)

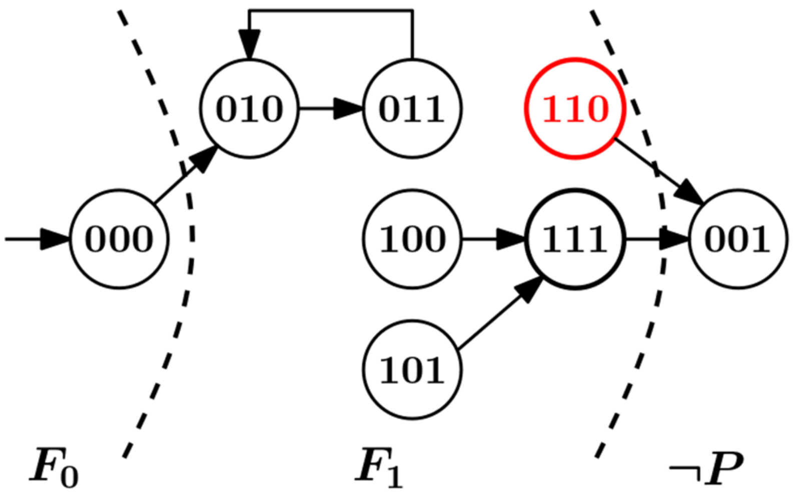
  g := generalizeNaive(i, s)

  $\forall 0 < j \leq i : F_j \coloneqq F_j \wedge \neg g$

function generalizeNaive(i, state s)

  return a shortest cube c such that

  - $c \leftarrow s$
  - $\neg\mathrm{SAT}(\mathrm{F_{i-1}} \wedge R \wedge c')$
  - $\neg\mathrm{SAT}(S_0 \wedge c)$



function PDR(Model M)

  if SAT($S_0 \land \neg P$)) **or** $SAT(S\_0 \land R \land \neg P')$ **then FAIL**

  F_0 := S_0; F_1 := P; k :=1;

  while(true)

    while(s := SAT($F_k \land\ R \land \neg P'$))

      removeBad(k, s)

    k++; F_k := P

    if $\exists 0 \le i < k-1 : F_i := F_{i+1}$ **then SUCCEED**

// post: $\neg SAT(F_i \land s)$

function removeBad($i \in N$, state s)

  if SAT($S_0 \land c$) **then FAIL**

  while(t := SAT($F_{i-1} \land R \land s'$))

    removeBad($i-1, t$)

  g := generalizeNaive(i, s)

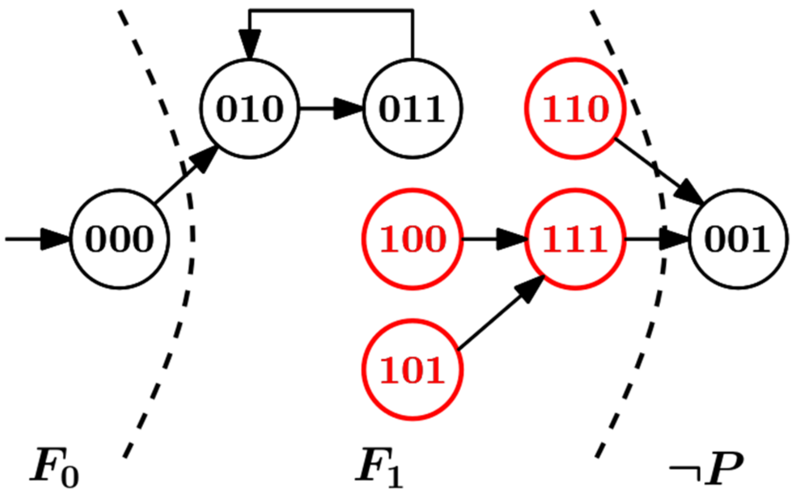
  $\forall 0 < j \le i : F_j \coloneqq F_j \land \neg g$

function generalizeNaive(i, state s)

  return a shortest cube c such that

  - $c \leftarrow s$
  - $\neg\mathrm{SAT}(\mathrm{F_{i-1}} \land R \land c')$
  - $\neg\mathrm{SAT}(S_0 \land c)$

i=1 , s=110,
c=x_1



function PDR(Model M)

if SAT($S_0 \wedge \neg P$)) **or** $SAT(S\_0 \wedge R \wedge \neg P')$ **then FAIL**

$F_0 := S_0$; $F_1 := P$; k :=1;

while(true)

while(s := SAT($F_k \wedge\ R \wedge \neg P'$))

removeBad(k, s)

k++; $F_k := P$

if $\exists 0 \leq i < k-1 \colon F_i := F_{i+1}$ **then SUCCEED**

// post: $\neg SAT(F_i \wedge s)$

function removeBad($i \in N$, state s)

if SAT($S_0 \wedge c$) **then FAIL**

while(t := SAT($F_{i-1} \wedge R \wedge s'$))

removeBad($i-1, t$)

g := generalizeNaive(i, s)

$\forall 0 < j \leq i \colon F_j \coloneqq F_j \wedge \neg g$

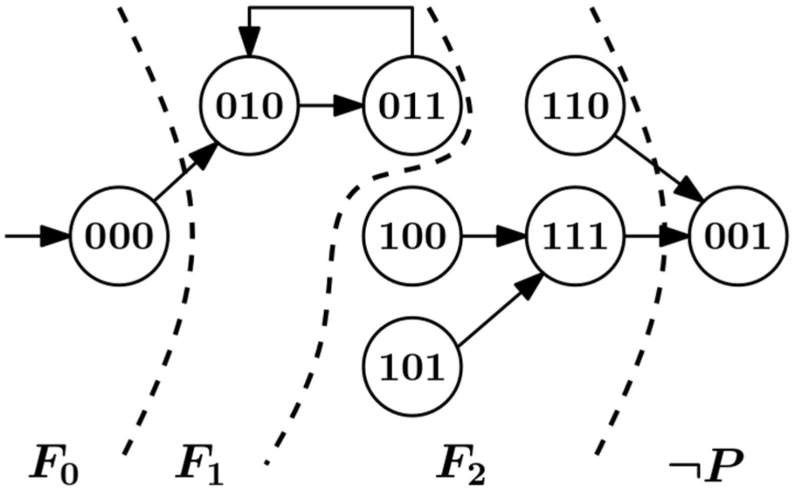
function generalizeNaive(i, state s)

return a shortest cube c such that

- $c \leftarrow s$
- $\neg\mathrm{SAT}(\mathrm{F_{i-1}} \wedge R \wedge c')$
- $\neg\mathrm{SAT}(S_0 \wedge c)$

i=1 , s=110,

c=x_1



SCOS Secure & Correct Systems

function PDR(Model M)

if SAT($S_0 \wedge \neg P$)) **or** $SAT(S\_0 \wedge R \wedge \neg P')$ **then FAIL**

F_0 := S_0; F_1 := P; k :=1;

while(true)

while(s := SAT($F_k \wedge\ R \wedge \neg P'$))

removeBad(k, s)

k++; F_k := P

if $\exists 0 \leq i < k-1: F_i := F_{i+1}$ **then SUCCEED**

// post: $\neg SAT(F_i \wedge s)$

function removeBad($i \in N$, state s)

if SAT($S_0 \wedge c$) **then FAIL**

while(t := SAT($F_{i-1} \wedge R \wedge s'$))

removeBad($i-1, t$)

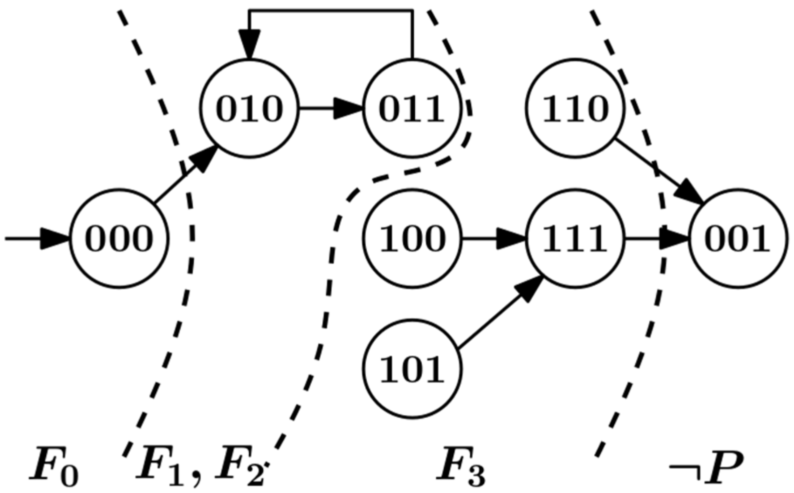
g := generalizeNaive(i, s)

$\forall 0 < j \leq i: F_j \coloneqq F_j \wedge \neg g$

function generalizeNaive(i, state s)

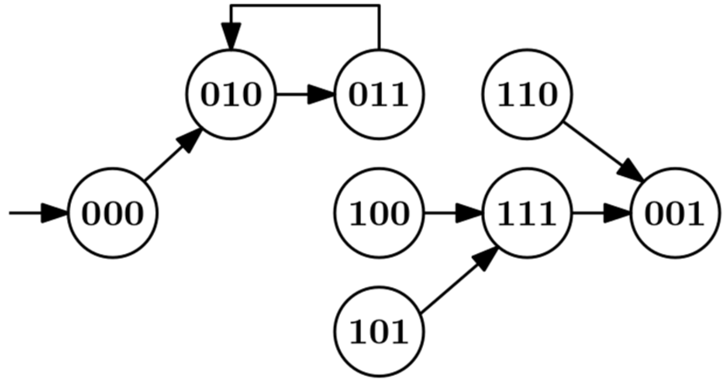
return a shortest cube c such that

- $c \leftarrow s$
- $\neg$SAT($F_{i-1} \wedge R \wedge c'$)
- $\neg$SAT($S_0 \wedge c$)



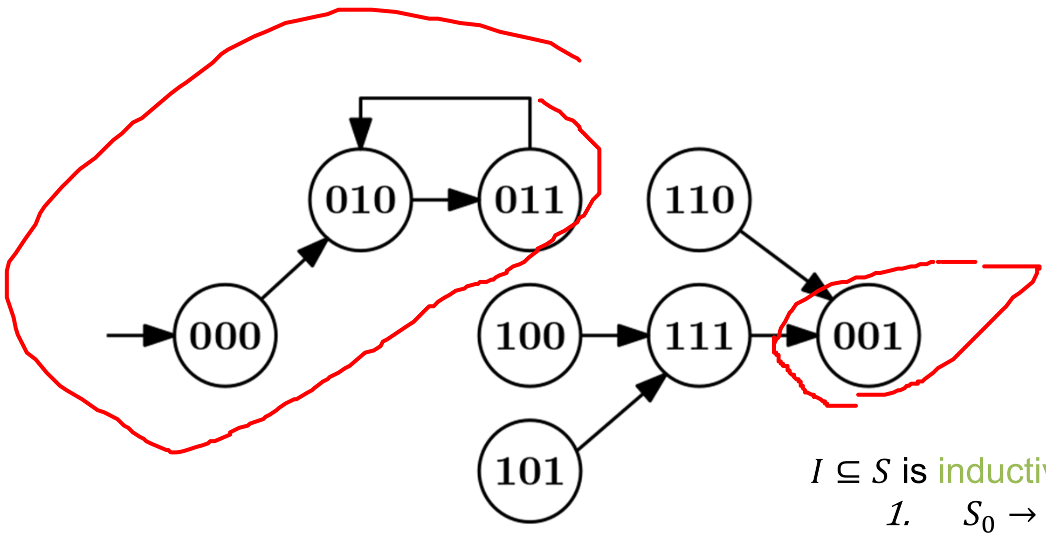
We repeat the same steps.

$F_1 = F_2$, so P is verified.



Task 4c [3 points]. Which of the following statements are false? Justify your answer.

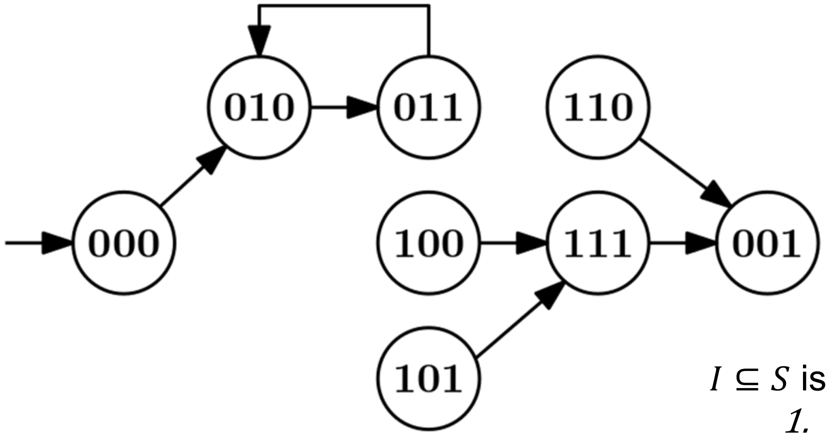
- The set $\neg x_1$ is inductive.
- The set $\neg x_3$ is inductive.
- The set $\neg x_2$ is inductive relative to $\neg x_1$.
- The set $\neg x_3$ is inductive relative to $\neg x_1$.



$I \subseteq S$ is inductive if

1. $S_0 \rightarrow I \ (S_0 \subseteq I)$
2. $I \wedge R \rightarrow I' \ (postimage(I) \subseteq I)$

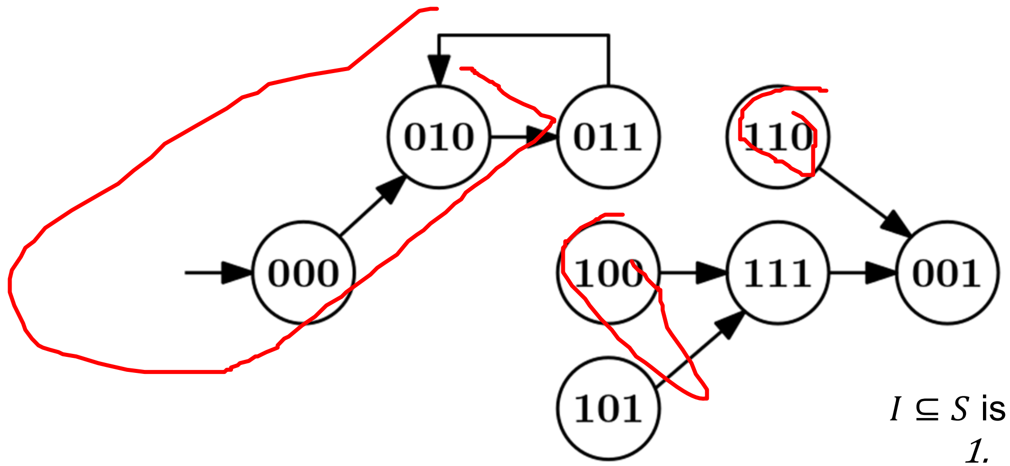
– The set $\neg x_1$ is inductive.



$I \subseteq S$ is inductive if

1. $S_0 \rightarrow I$ $(S_0 \subseteq I)$
2. $I \wedge R \rightarrow I'$ $(postimage(I) \subseteq I)$

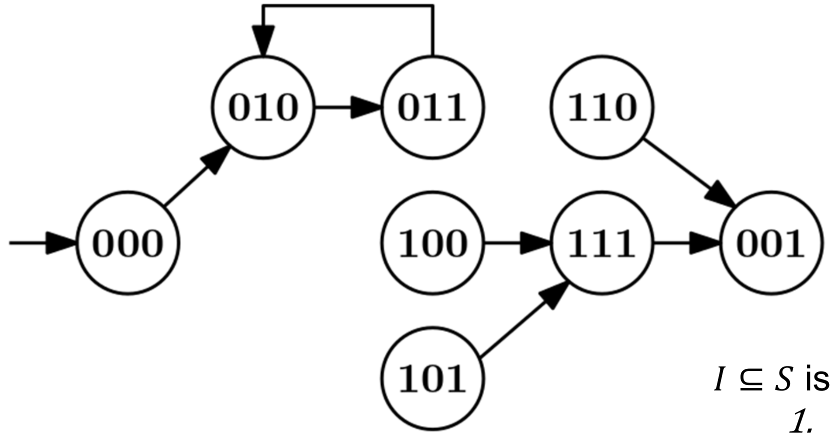
– The set $\neg x_1$ is inductive. TRUE



$I \subseteq S$ is inductive if

1. $S_0 \rightarrow I \ \ (S_0 \subseteq I)$
2. $I \wedge R \ \rightarrow I' \ (postimage(I) \subseteq I)$

– The set $\neg x_3$ is inductive.

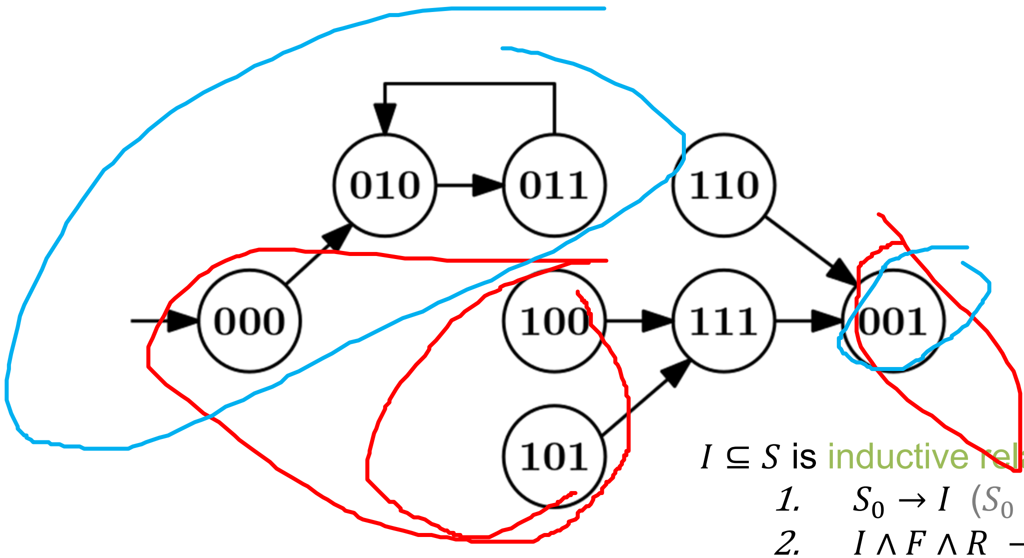


$I \subseteq S$ is inductive if

1. $S_0 \rightarrow I$ $(S_0 \subseteq I)$
2. $I \wedge R \rightarrow I'$ $(postimage(I) \subseteq I)$

– The set $\neg x_3$ is inductive.
FALSE

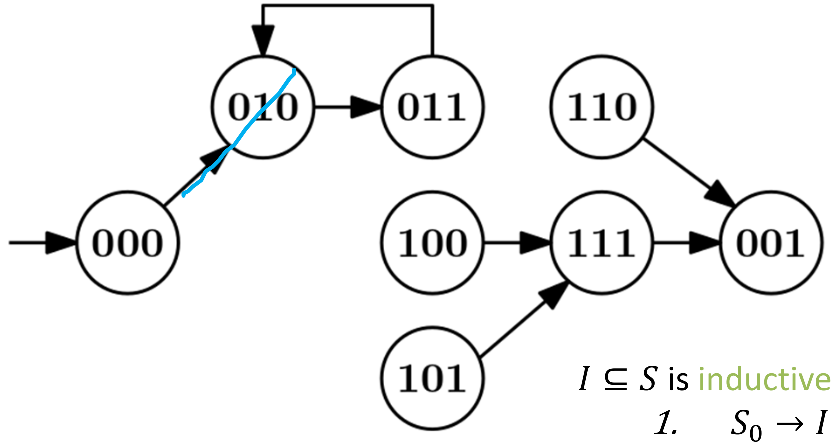
010 -> 011



$I \subseteq S$ is inductive relative to F if

1. $S_0 \rightarrow I \;\; (S_0 \subseteq I)$
2. $I \wedge F \wedge R \;\rightarrow I' \; (postimage(F \cap I) \subseteq I)$

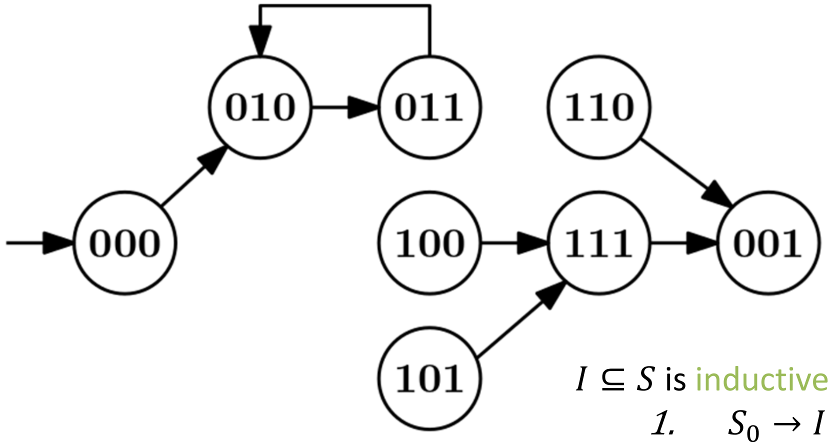
– The set $\neg x_2$ is inductive relative to $\neg x_1$.



$I \subseteq S$ is inductive relative to F if

1. $S_0 \rightarrow I$ $(S_0 \subseteq I)$
2. $I \wedge F \wedge R \rightarrow I'$ $(postimage(F \cap I) \subseteq I)$

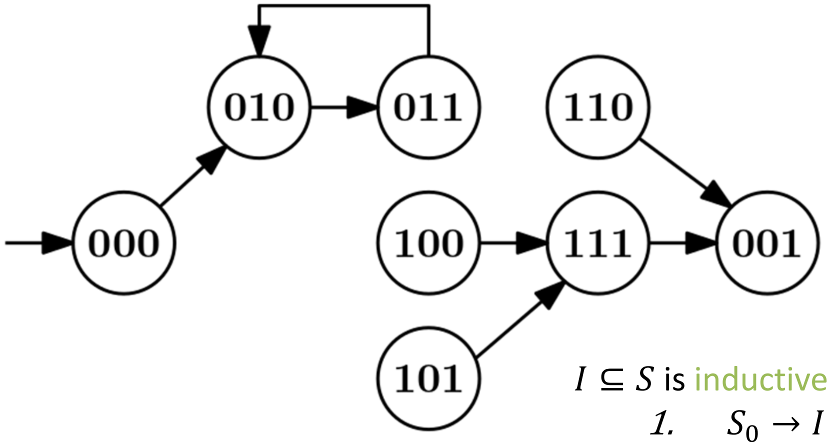
– The set $\neg x_2$ is inductive relative to $\neg x_1$. 000 -> 010
  FALSE



$I \subseteq S$ is inductive relative to F if

1. $S_0 \rightarrow I$ $(S_0 \subseteq I)$
2. $I \wedge F \wedge R \rightarrow I'$ $(postimage(F \cap I) \subseteq I)$

– The set $\neg x_3$ is inductive relative to $\neg x_1$.



$I \subseteq S$ is inductive relative to F if

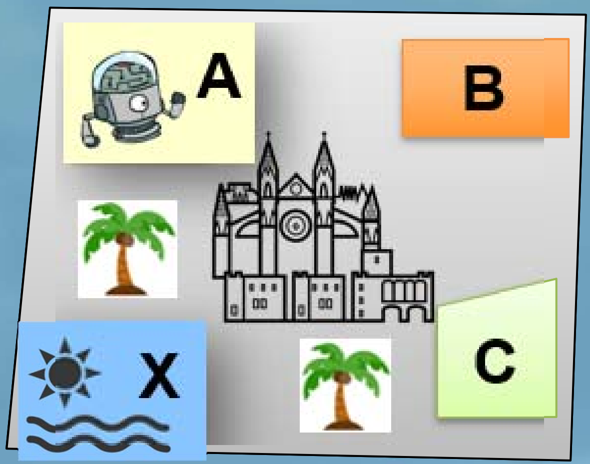
1. $S_0 \rightarrow I \;\; (S_0 \subseteq I)$
2. $I \wedge F \wedge R \;\rightarrow I' \; (postimage(F \cap I) \subseteq I)$

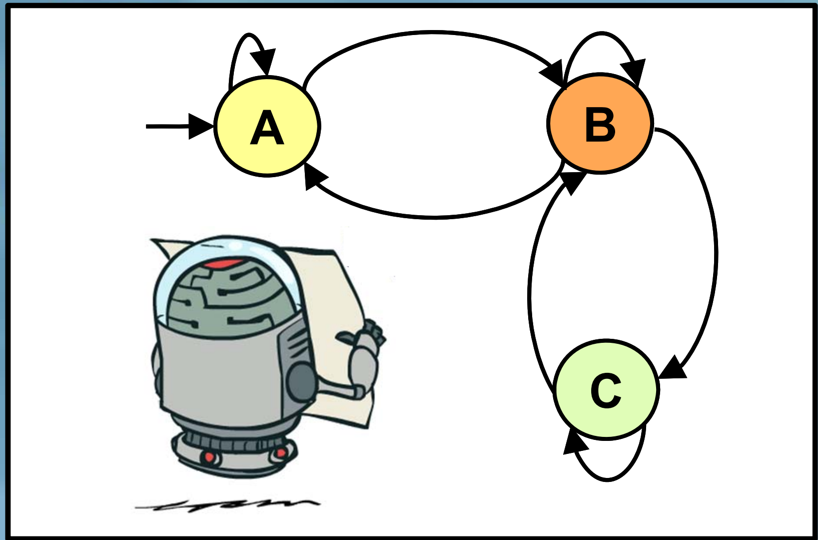
– The set $\neg x_3$ is inductive relative to $\neg x_1$.
FALSE

010 -> 011

Graz University of Technology
Institute for Applied Information
Processing and Communications

Temporal Logic





Model Checking SS21

Warm Up

Translate sentences to formulas

- “If today is Tuesday, tomorrow is Wednesday.”

- “This lecture is exciting and not boring.”

Warm Up

Translate sentences to formulas

- "If today is Thursday, then tomorrow is Friday."

  p... today is Tuesday, q… tomorrow is Wednesday

  $$p \rightarrow q$$

- "This lecture is exciting and not boring."

  p... This lecture is exciting ,
  q… This lecture is boring

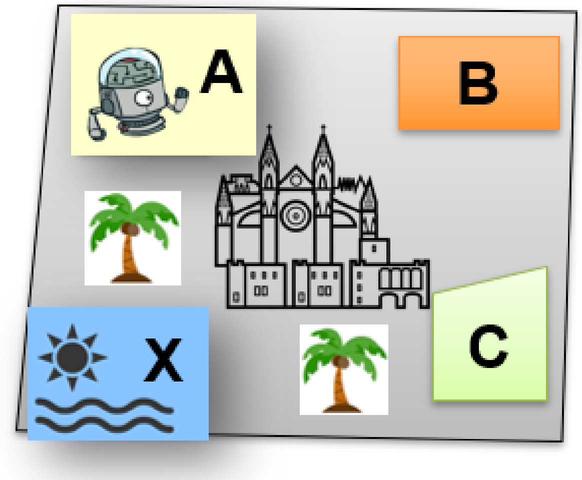
  $$p \land \neg q$$

Warm Up

- Whenever it rains, I have an umbrella
- When it rains, worms come out after a while
- I will not pay before you deliver the goods

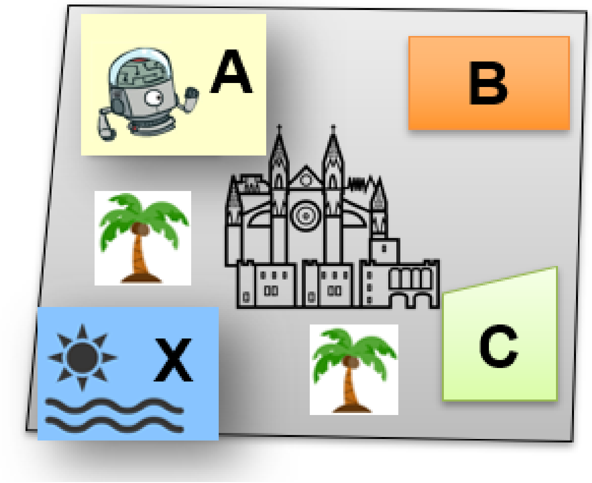
Modeling a reactive system

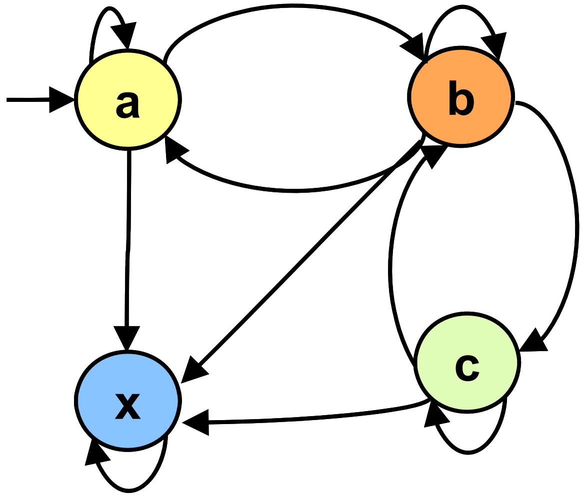
Kripke structure



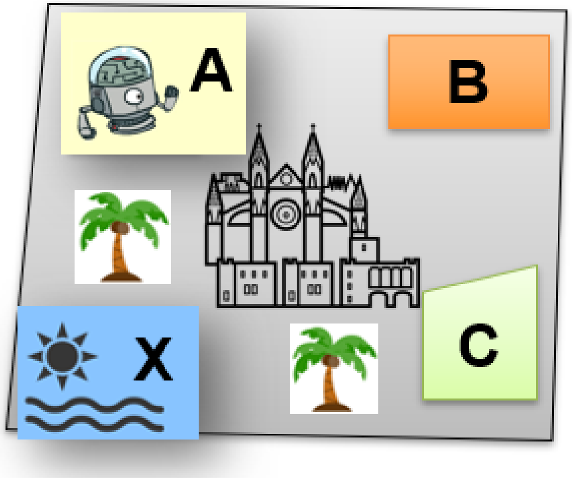
Modeling a reactive system

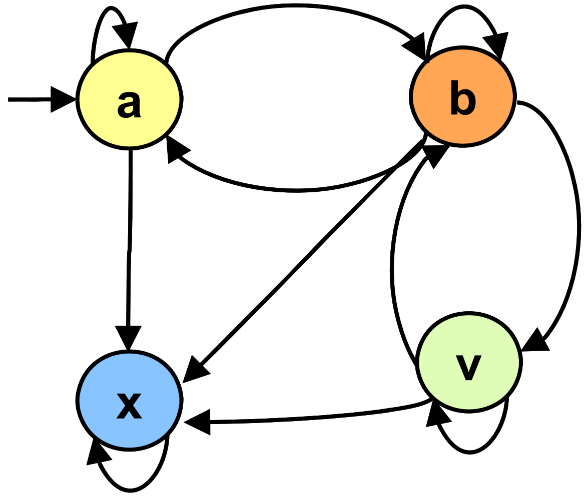
Kripke structure





Properties of Kripke Structures





Properties

ToDo *Write properties as formulas*

- Always when the robot visits A, it visits C within the next two steps.
- The robot can visit C within the next two steps after visiting A

Propositional Temporal Logic

AP – a set of atomic propositions, p,q∈AP

Temporal operators:

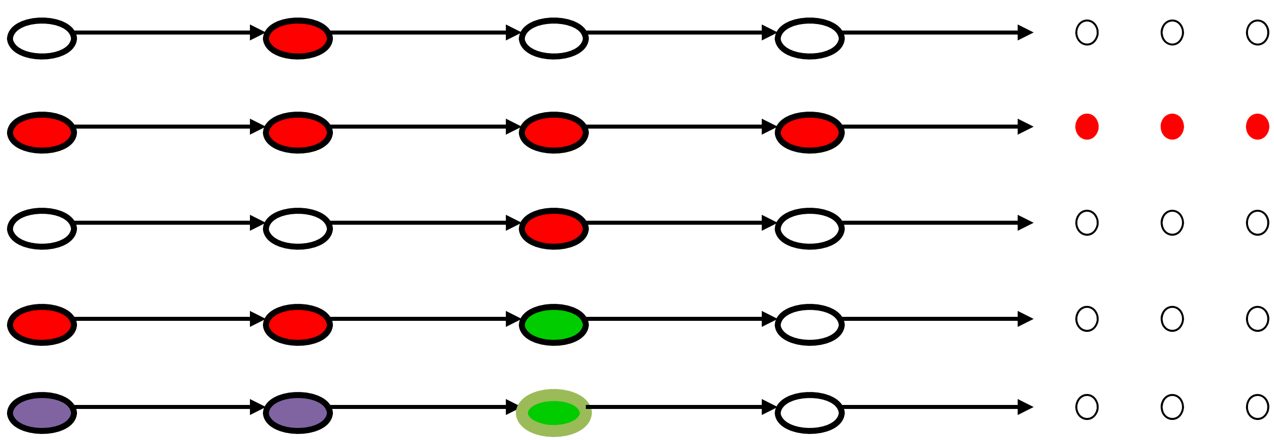
Xp

Gp

Fp

pUq

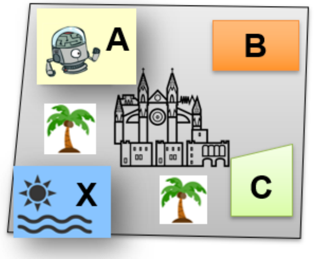
pRq

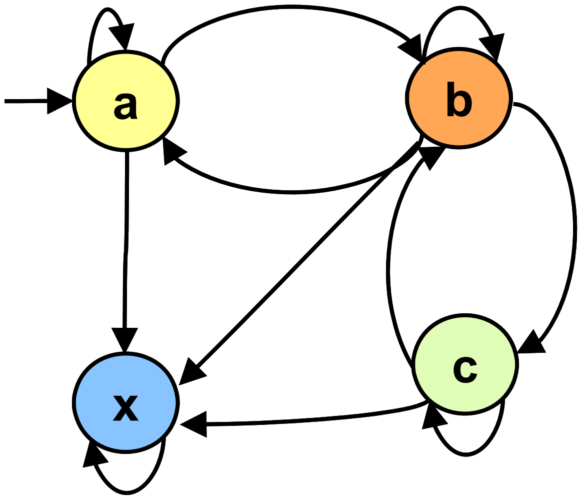


Path quantifiers: **A** for **all** paths

E there **exists** a path

Properties of Kripke Structures





Temporal Operators
X… next
G… globally
F… eventually

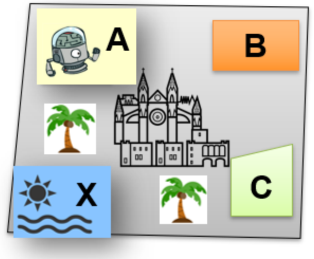
Path quantifiers
A for **all** paths
E there **exists** a path

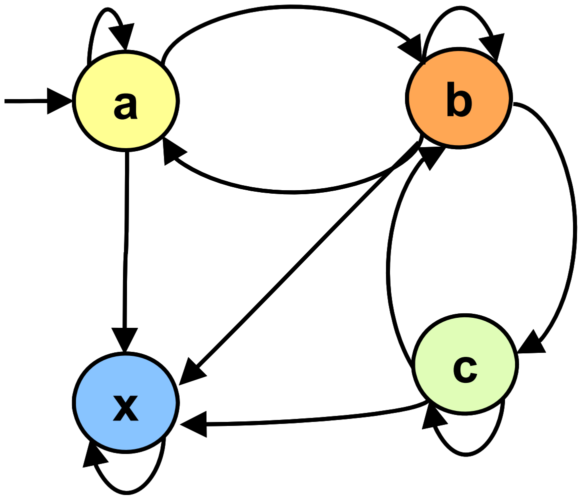
Properties

ToDo *Write properties as formulas*

- Always when the robot visits A, it visits C within the next two steps.
- The robot can visit C within the next two steps after visiting A

Properties of Kripke Structures





Temporal Operators
X… next
G… globally
F… eventually

Path quantifiers
A for **all** paths
E there **exists** a path

Properties

ToDo *Write properties as formulas*

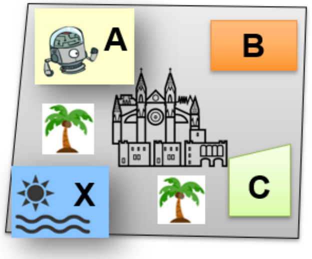
- Always when the robot visits A, it visits C within the next two steps.

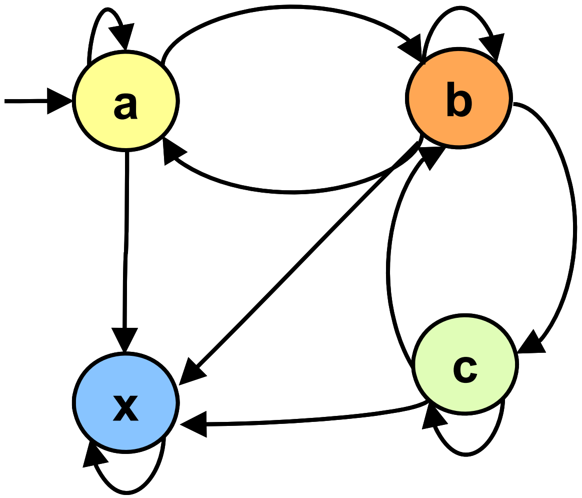
$$A\ G\ (a \rightarrow Xc \vee XXc)$$

- The robot can visit C within the next two steps after visiting A

$$E\ G\ (a \rightarrow Xc \vee XXc)$$

Properties of Kripke Structures





Temporal Operators
X… next
G… globally
F… eventually

Path quantifiers
A for **all** paths
E there **exists** a path

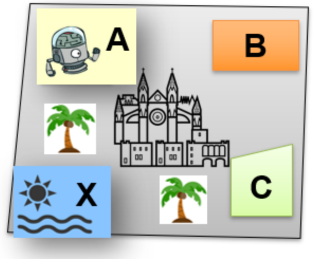
Properties

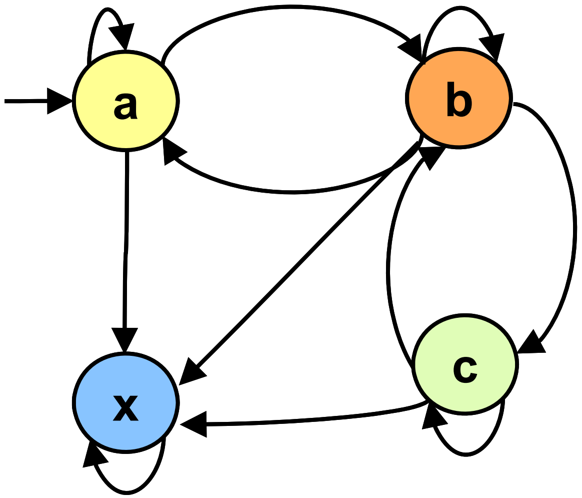
ToDo *Write properties as formulas*

- The robot *never* visits **X**
- It is possible that the robot *never* visits **X**

SCOS Secure & Correct Systems

Properties of Kripke Structures





Temporal Operators
X… next
G… globally
F… eventually

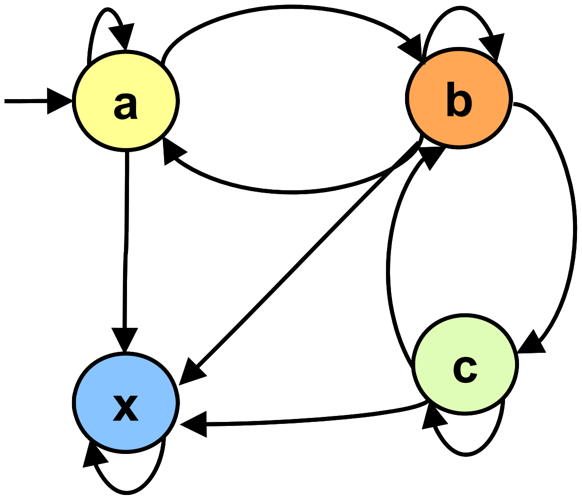
Path quantifiers
A for **all** paths
E there **exists** a path

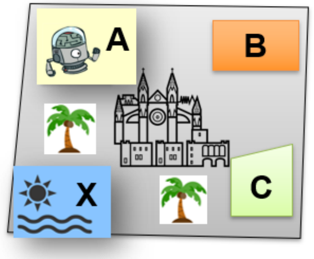
Properties

Write properties as formulas

- The robot *never* visits **X** $\quad A\ G\ \neg x$
- It is possible that the robot *never* visits **X** $\quad E\ G\ \neg x$

Properties of Kripke Structures





Temporal Operators
X… next
G… globally
F… eventually

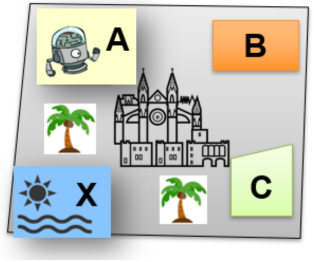
Path quantifiers
A for **all** paths
E there **exists** a path

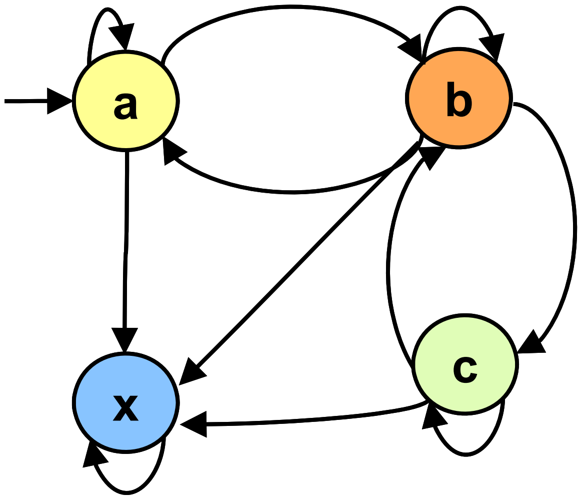
Properties

Write properties as formulas

- The robot can visit A and C *infinitely often.*
- The robot always visits A *infinitely often*, but C only *finitely often*.

Properties of Kripke Structures





Temporal Operators
X… next
G… globally
F… eventually

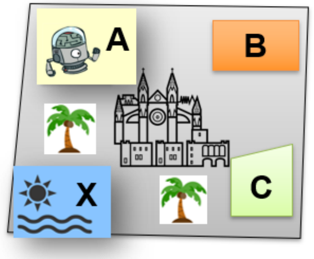
Path quantifiers
A for **all** paths
E there **exists** a path

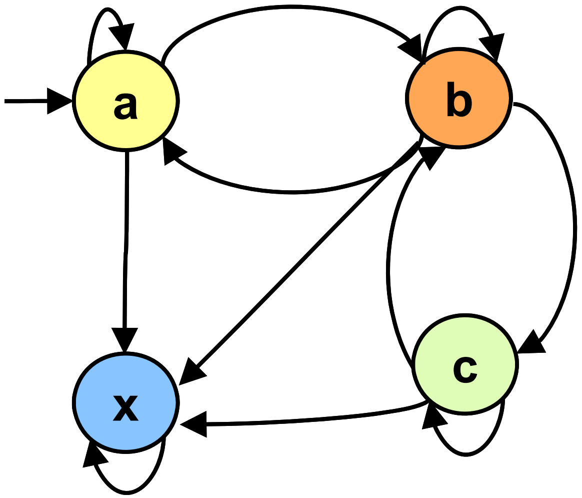
Properties

Write properties as formulas

- The robot can visit A and C *infinitely often.* $A\ (GF\ a \wedge GF\ c)$
- The robot always visits A *infinitely often*, but C only *finitely often*. $E\ (GF\ a \wedge FG \neg c)$

Properties of Kripke Structures





Temporal Operators
X… next
G… globally
F… eventually

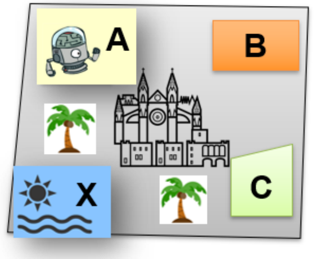
Path quantifiers
A for **all** paths
E there **exists** a path

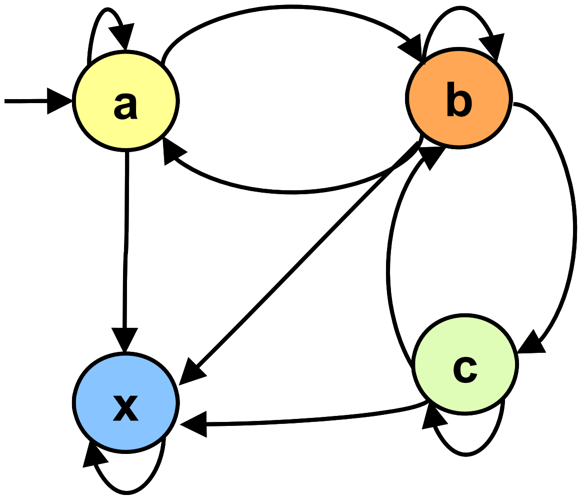
Properties

Write properties as formulas

- If the robot visits **A** *infinitely often,*
  it should also visit **C** *finitely often*.

Properties of Kripke Structures





Temporal Operators
X… next
G… globally
F… eventually

Path quantifiers
A for **all** paths
E there **exists** a path

Properties

Write properties as formulas

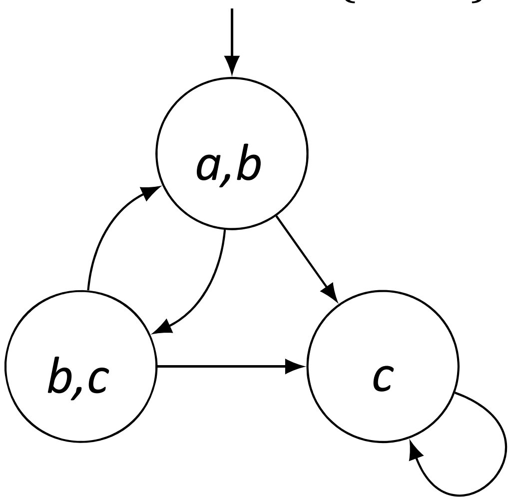
- If the robot visits **A** *infinitely often,* it should also visit **C** *finitely often.*

$$A\ (GF\ a \rightarrow GFc)$$

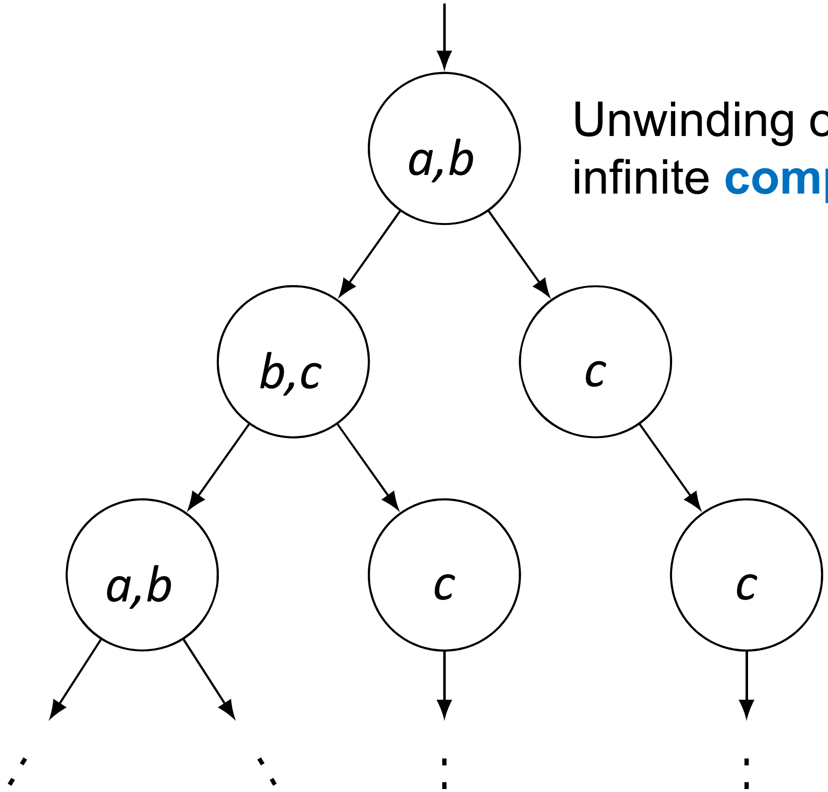
Computation Tree Logic - CTL*

- Defines properties of computation trees of Kripke structures

Kripke structure M,
labeled with $AP = \{a, b, c\}$

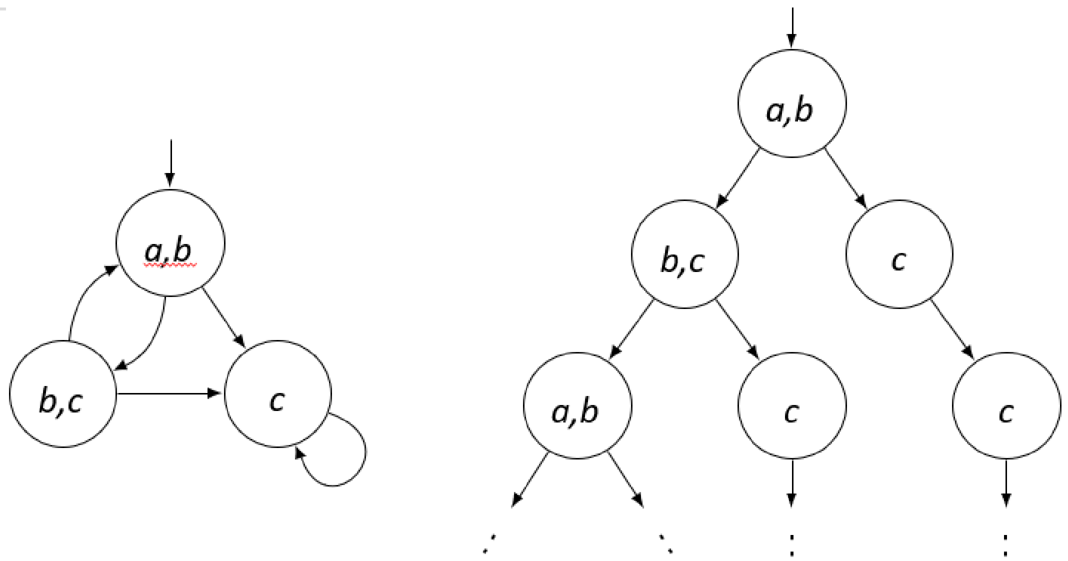


Unwinding of M into infinite **computation tree**



Paths and Suffixes

- $\pi = s_0, s_1, \ldots$ is an *infinite* path in M from a state s if
  - $s = s_0$ and
  - for all $i \geq 0$, $(s_i, s_{i+1}) \in R$



Propositional Temporal Logic

Temporal operators:

- Describe properties that hold along π

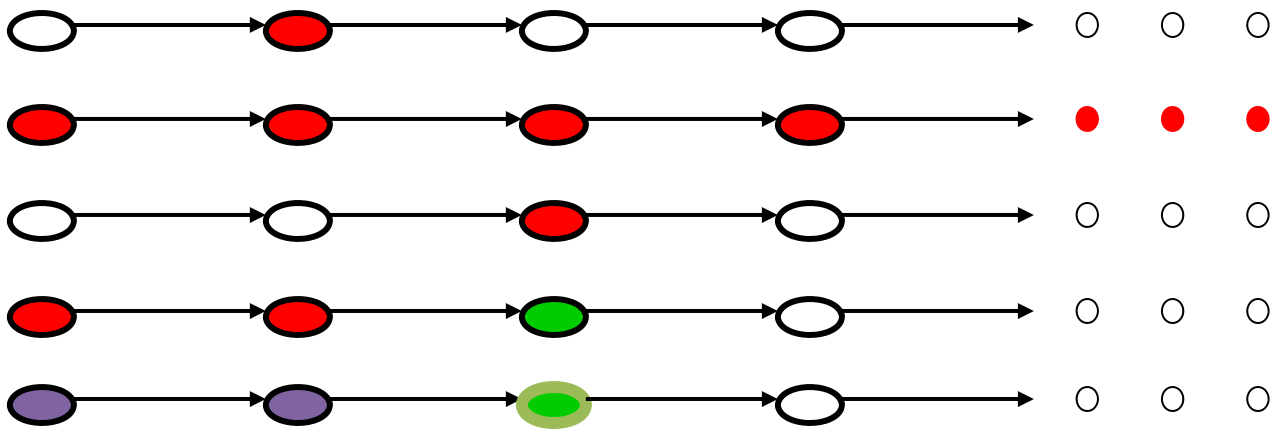
Xp

Gp

Fp

pUq

pRq

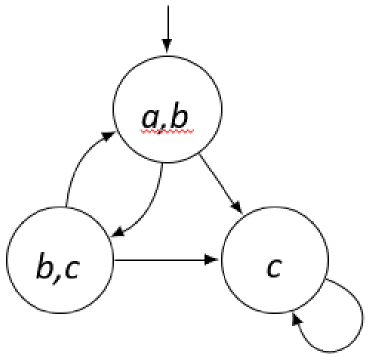


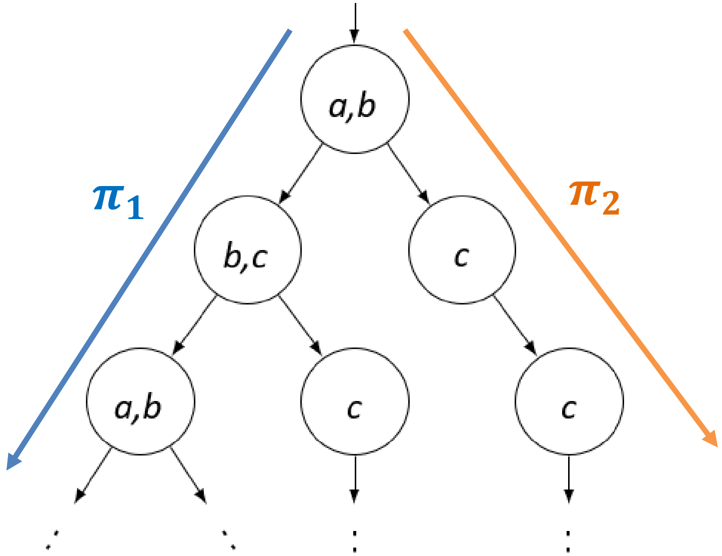
Propositional Temporal Logic

Path quantifiers:

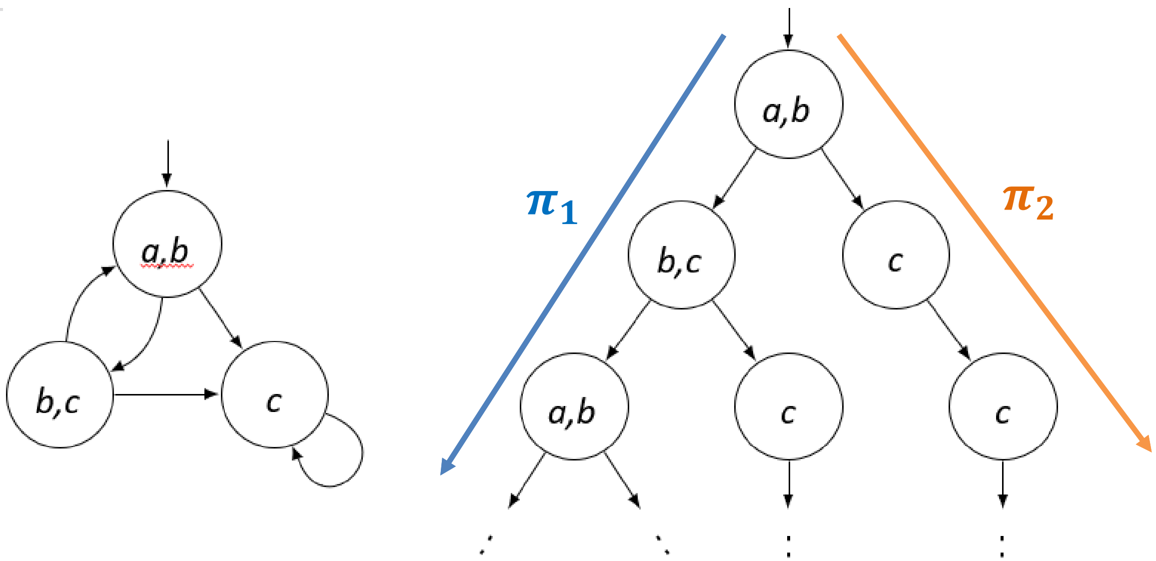
- **A** for **all** paths starting from **s** have property φ
- **E** there **exists** a path starting from **s** have property φ
- Use combination of **A and E** to describe branching structure in tree

State Formulas and Path Formulas



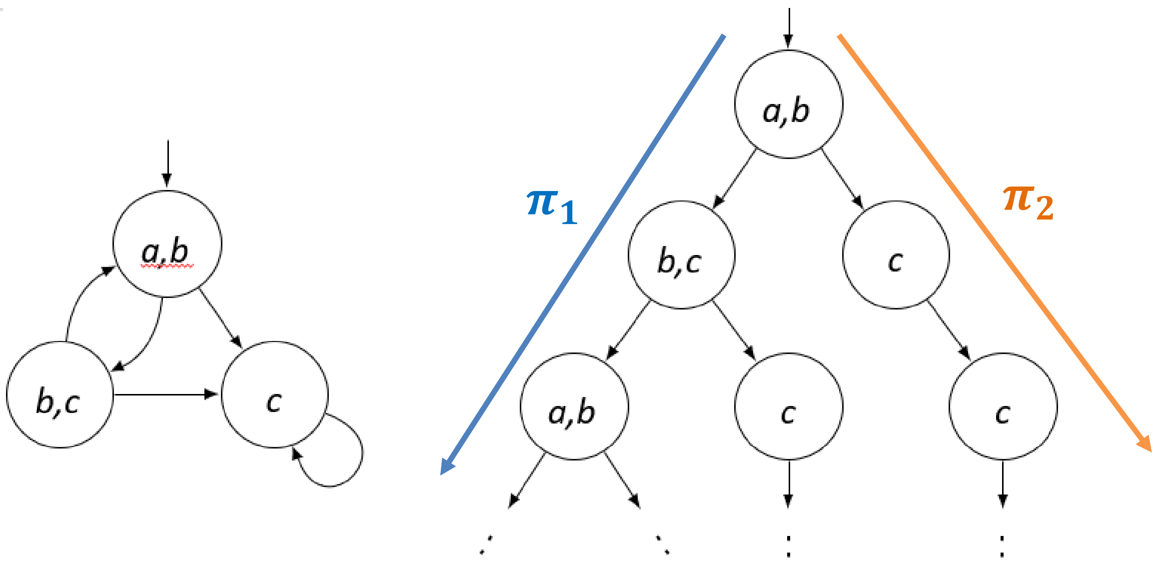


State Formulas and Path Formulas



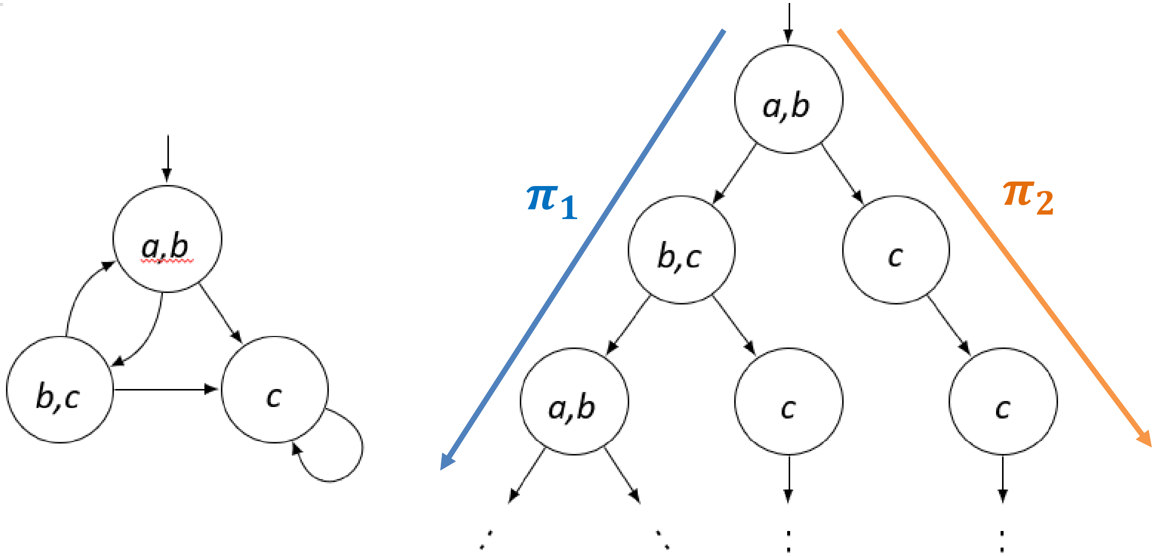
- Path Formulas:
  - $\pi_1 \vDash \mathrm{Gb}$
  - $\pi_2 \nvDash \mathrm{Gb}$

State Formulas and Path Formulas



- Path Formulas:
  - $\pi_1 \models \mathrm{G}b$
  - $\pi_2 \not\models \mathrm{G}b$
- State Formulas:
  - $s_0 \models \mathrm{EG}\ b$
  - $s_0 \not\models \mathrm{AG}\ b$

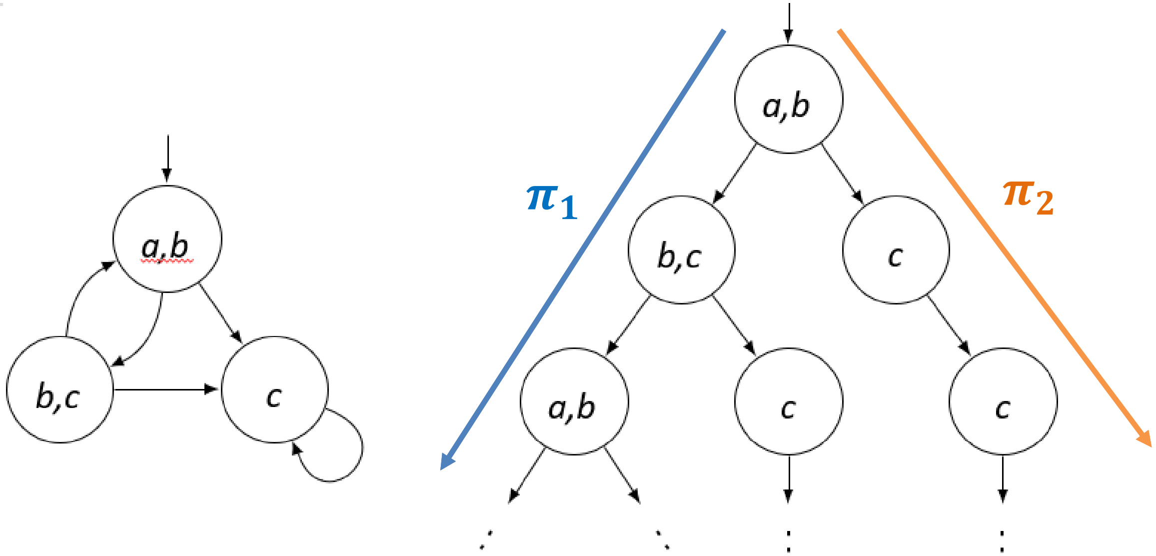
State Formulas and Path Formulas



Does s_0 satisfy the following formula?

- $s_0 \square \text{EXX}\ (a \wedge b)$
- $s_0 \square \text{EXAX}\ (a \wedge b)$

State Formulas and Path Formulas



- Does s_0 satisfy the following formula?
  - $s_0 \vDash \text{EXX}\,(\text{a} \wedge b)$
  - $s_0 \nvDash \text{EXAX}\,(\text{a} \wedge b)$

Syntax of CTL*

Two types of formulas in the inductive definition

- State formulas
- Path formulas

Syntax of CTL*: State Formulas

State formulas are true in a specific state

Syntax of CTL*: State Formulas

State formulas are true in a specific state

Inductive definition of state formulas:

Syntax of CTL*: State Formulas

State formulas are true in a specific state

Inductive definition of state formulas:

- $p \in AP$

Syntax of CTL*: State Formulas

State formulas are true in a specific state

Inductive definition of state formulas:

- $p \in AP$
- $\neg f_1, f_1 \vee f_2, f_1 \wedge f_2$ where f_1, f_2 are state formulas

Syntax of CTL*: State Formulas

State formulas are true in a specific state

Inductive definition of state formulas:

- $p \in AP$
- $\neg f_1, f_1 \vee f_2, f_1 \wedge f_2$ where f_1, f_2 are state formulas
- $\boldsymbol{E}g, \boldsymbol{A}g$ where g is a path formula

Syntax of CTL*: Path Formulas

Path formulas are true along a specific path

Syntax of CTL*: Path Formulas

Path formulas are true along a specific path

Inductive definition of path formulas:

Syntax of CTL*: Path Formulas

Path formulas are true along a specific path

Inductive definition of path formulas:

- If f is a state formula, then f is also a path formula

Syntax of CTL*: Path Formulas

Path formulas are true along a specific path

Inductive definition of path formulas:

- If f is a state formula, then f is also a path formula
- $\neg g_1, g_1 \vee g_2, g_1 \wedge g_2, \boldsymbol{X} g_1, \boldsymbol{G} g_1, \boldsymbol{F} g_1, g_1 \boldsymbol{U} g_2,$ $g_1 \boldsymbol{R} g_2$ where g_1, g_2 are path formulas

Syntax of CTL*: Path Formulas

Path formulas are true along a specific path

Inductive definition of path formulas:

- If f is a state formula, then f is also a path formula
- $\neg g_1, g_1 \vee g_2, g_1 \wedge g_2, \boldsymbol{X} g_1, \boldsymbol{G} g_1, \boldsymbol{F} g_1, g_1 \boldsymbol{U} g_2$, $g_1 \boldsymbol{R} g_2$ where g_1, g_2 are path formulas

CTL* is the set of all state formulas

Semantics of CTL*

- Kripke Structure $M = (S, S_0, R, AP, L)$
- $\pi = s_0, s_1, \ldots$ is an infinite path in M
- π^i – the suffix of π, starting at s_i

Semantics of CTL*

- Kripke Structure $M = (S, S_0, R, AP, L)$
- $\pi = s_0, s_1, \ldots$ is an infinite path in M
- π^i – the suffix of π, starting at s_i

- For state formulas:
  - $M, s \vDash f$ … the **state** formula f holds in state s of M

Semantics of CTL*

- Kripke Structure $M = (S, S_0, R, AP, L)$
- $\pi = \pi_0, \pi_1, \ldots$ is an infinite path in M
- π^i – the suffix of π, starting at s_i

- For state formulas:
  - $M, s \models f$ … the **state** formula f holds in state s of M

- For path formulas:
  - $M, \pi \models g$ … the **path** formula g holds along π in M

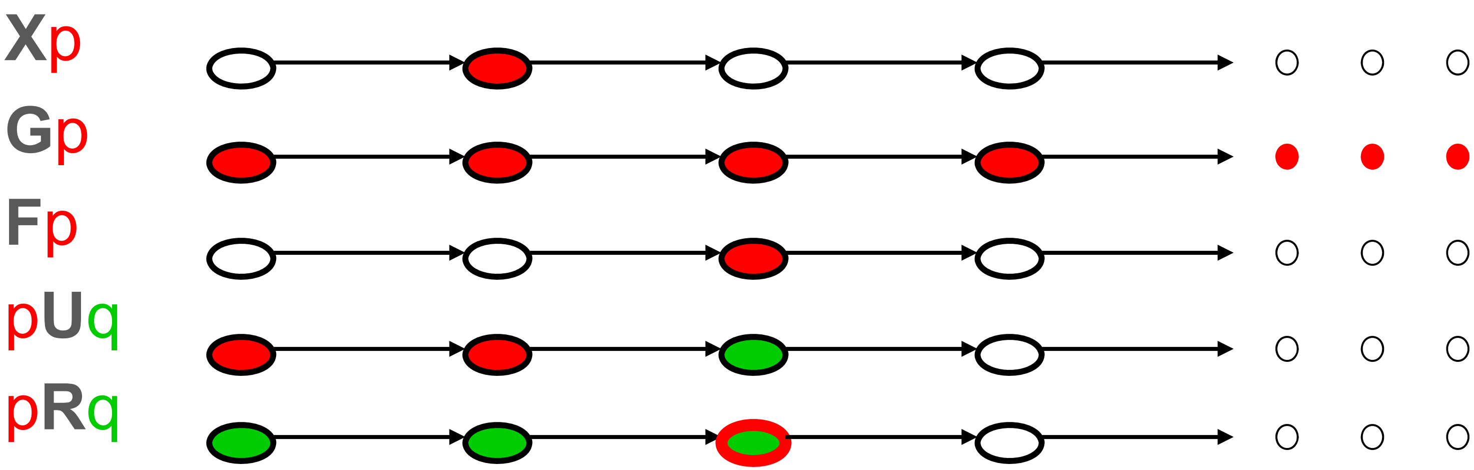
Semantics of CTL*

State formulas:

- $M, s \models p \quad \Leftrightarrow \quad p \in L(s)$
- $M, s \models \mathbf{E}\, f \quad \Leftrightarrow$ there is a path π from s such that $M, \pi \models f$
- $M, s \models \mathbf{A}\, g \quad \Leftrightarrow$ for every path π from s, $M, \pi \models g$
- Boolean combination ($\wedge$, $\vee$, $\neg$) – the usual semantics

Semantics of path formulas - summary

If p,q are state formulas, then:

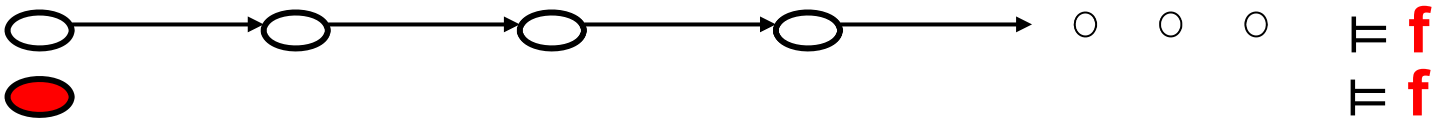


But in the general case, they can be path formulas

Semantics of CTL*

Path formulas:

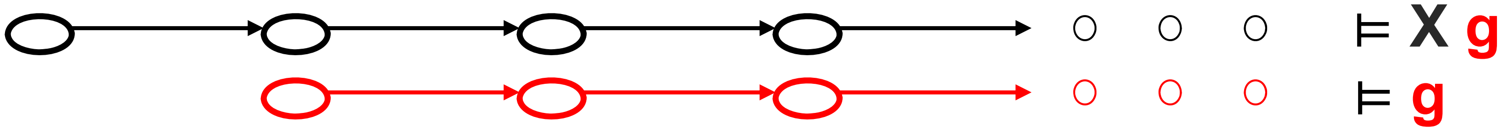
- M, π ⊨ f, where f is a state formula ⇔ $M, \pi_0 \models f$



Semantics of CTL*

Path formulas:

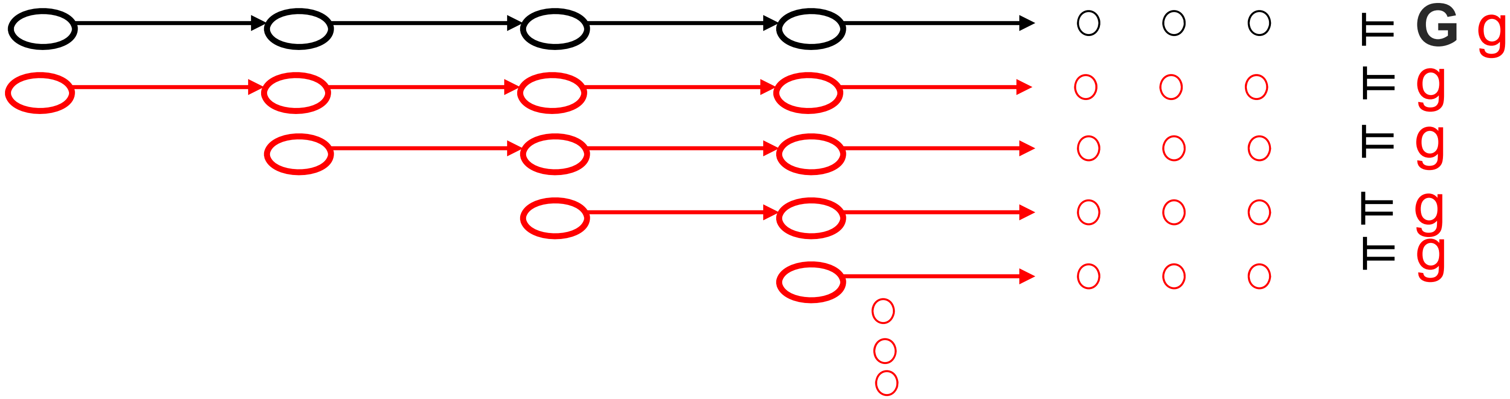
- $M, \pi \models \mathbf{X}g$, where g is a path formula $\Leftrightarrow$ $M, \pi^1 \models g$



Semantics of CTL*

Path formulas:

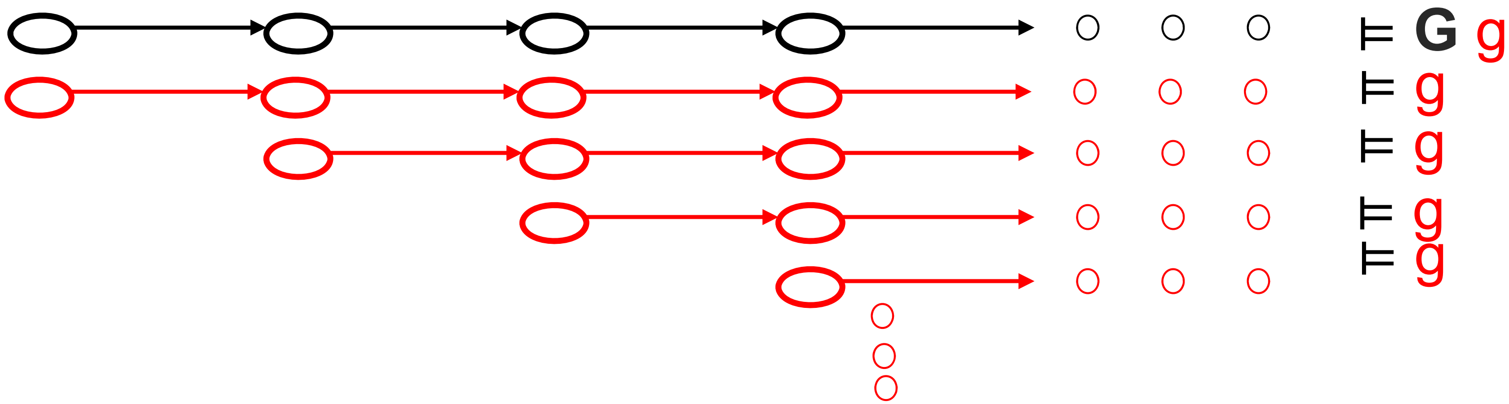
- $M, \pi \models \mathbf{G}g \Leftrightarrow$ for every $i \geq 0$, $M, \pi^i \models g$



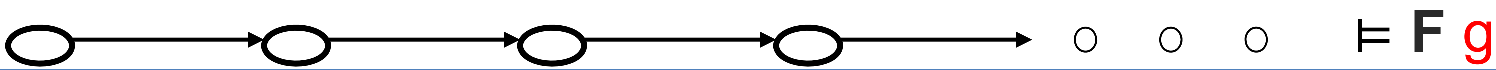
Semantics of CTL*

Path formulas:

- $M, \pi \models \mathbf{G}g \Leftrightarrow$ for every $i \geq 0$, $M, \pi^i \models g$



- $M, \pi \models \mathbf{F}g \Leftrightarrow$

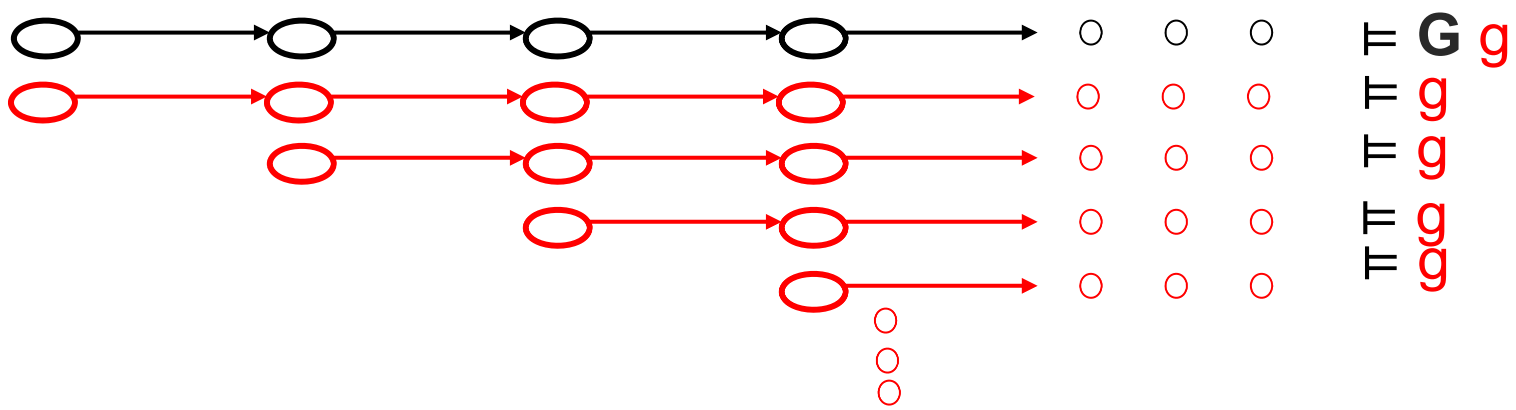


$\models \mathbf{G}\, g$

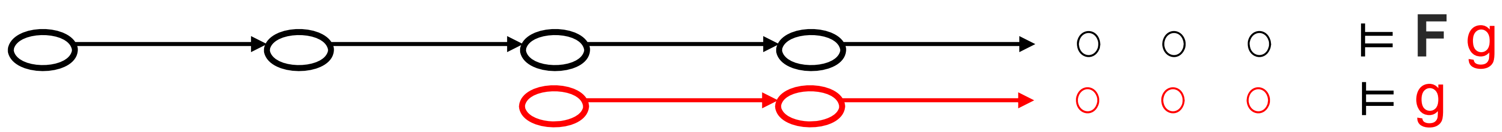
Semantics of CTL*

Path formulas:

- $M, \pi \models \mathbf{G}g \Leftrightarrow$ for every $i \geq 0$, $M, \pi^i \models g$



- $M, \pi \models \mathbf{F}g \Leftrightarrow$ there exists $k \geq 0$, such that $M, \pi^k \models g$

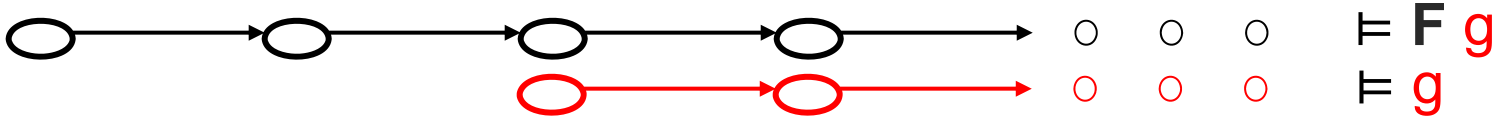


SCOS Secure & Correct Systems

Semantics of CTL*

Path formulas:

- $M, \pi \models \mathbf{F}g \Leftrightarrow$ there exists $k \geq 0$, such that $M, \pi^k \models g$

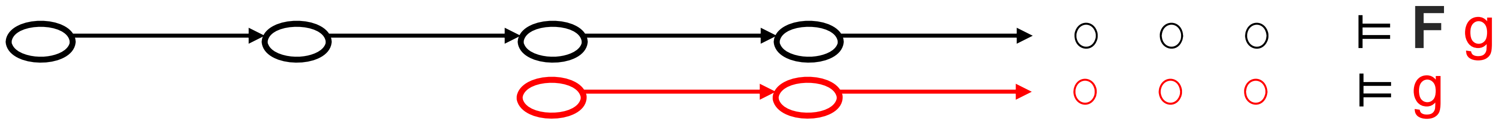


- $M, \pi \models g_1\ \mathbf{U}\ g_2 \Leftrightarrow$

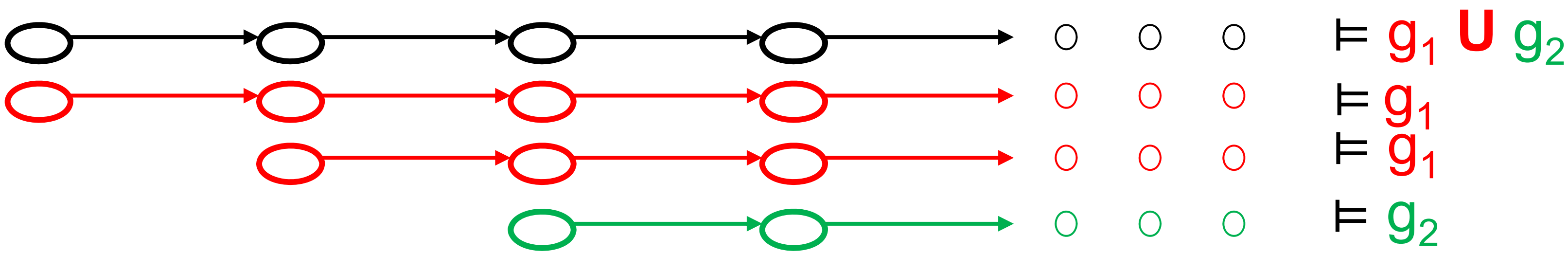
Semantics of CTL*

Path formulas:

- $M, \pi \models \mathbf{F}g \Leftrightarrow$ there exists $k \geq 0$, such that $M, \pi^k \models g$



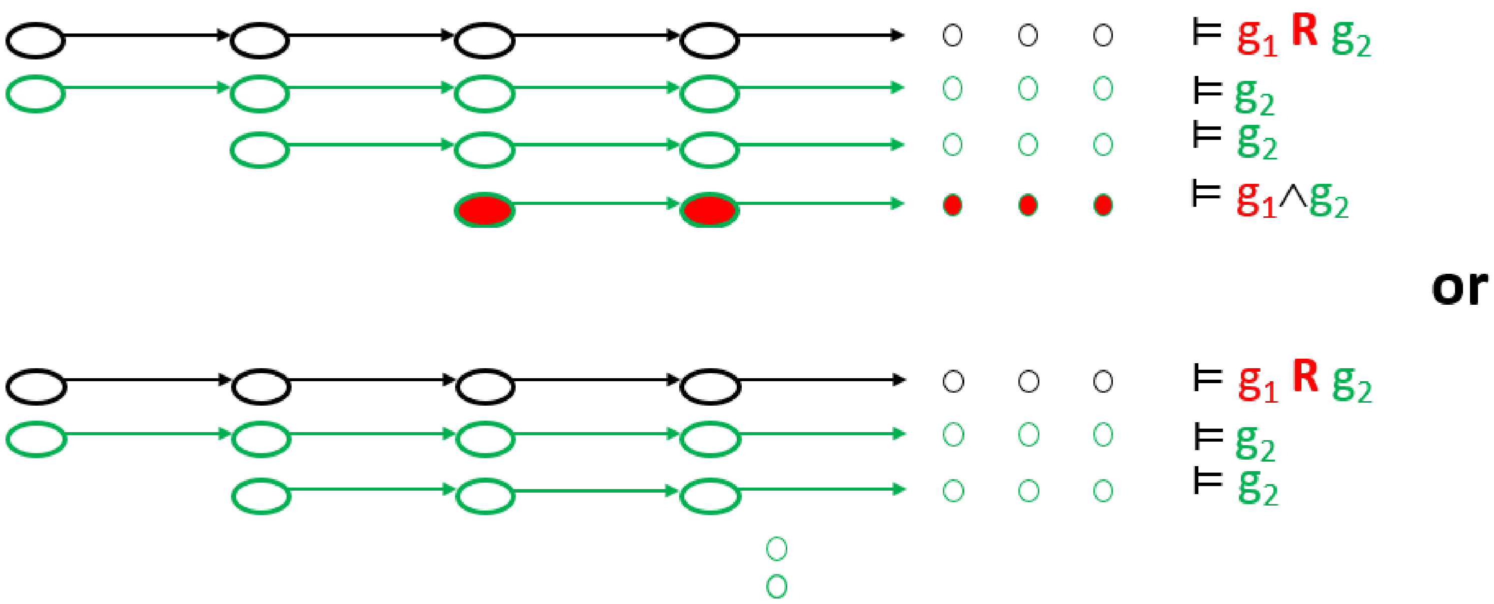
- $M, \pi \models g_1 \,\mathbf{U}\, g_2 \Leftrightarrow$ there exists $k \geq 0$, such that $M, \pi^k \models g_2$
  and for every $0 \leq j < k$, $M, \pi^j \models g_1$



Semantics of CTL*

Path formulas:

- M, $\pi \vDash g_1$ **R** $g_2 \Leftrightarrow$ for all $j \geq 0$, if for every $i<j$ M, $\pi^i \nvDash g_1$ then M, $\pi^j \vDash g_2$



More about R („release“)

- Intuitively, once g_1 becomes true, it “releases” g_2 If g_1 never becomes true then g_2 stays true forever

ToDo Rewrite it using U, F, G, or X

- g_1 **R** g_2 $\equiv$

More about R („release“)

- Intuitively, once g_1 becomes true, it “releases” g_2. If g_1 never becomes true then g_2 stays true forever

- $g_1 \mathbf{R}\, g_2 \equiv (g_2 \mathbf{U}\, (g_1 \wedge g_2)) \vee \mathbf{G}\, g_2$

Semantics of CTL*

ToDo

- $M \models f \Leftrightarrow$ $\boxed{}$

Semantics of CTL*

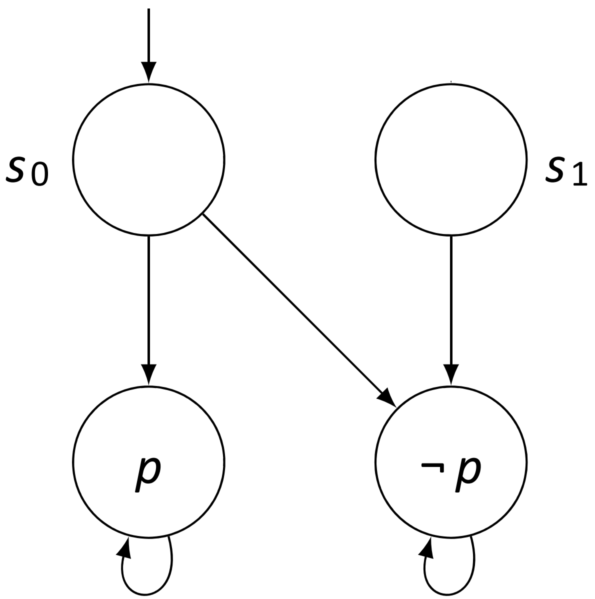
- $M \models f \Leftrightarrow$ for all initial states $s_0 \in S_0$: $M, s_0 \models f$

Semantics of CTL*

- $M \models f \iff$ for all initial states $s_0 \in S_0$: $M, s_0 \models f$

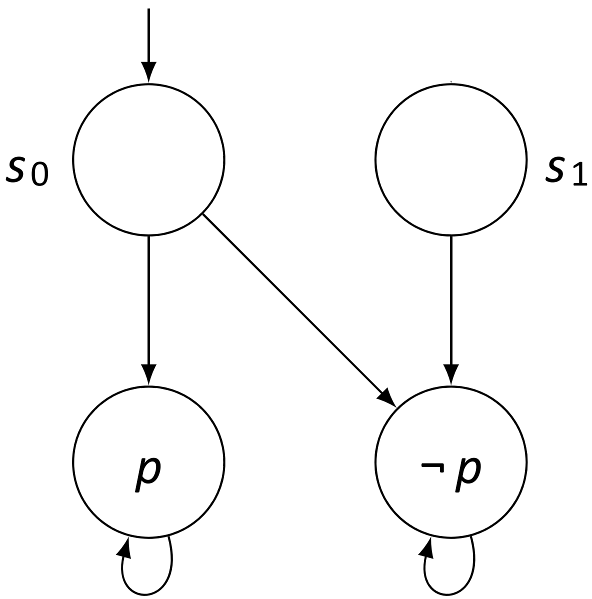
ToDo

- Example: Does $M \models EX\, p$ or $M \models \neg EX\, p$?



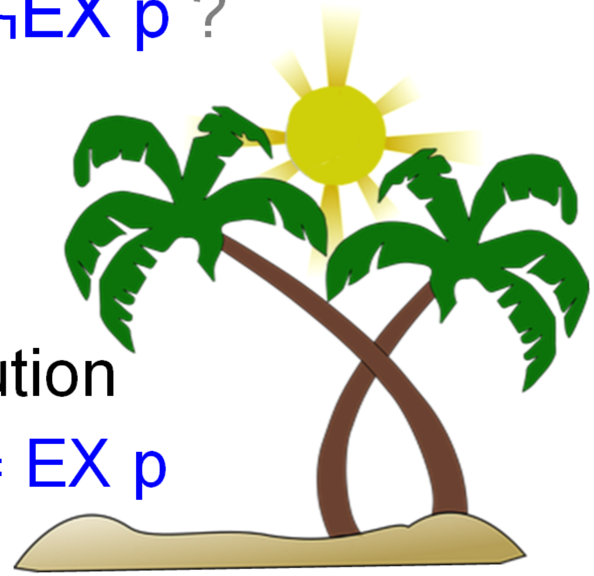
Semantics of CTL*

- $M \models f \Leftrightarrow$ for all initial states $s_0 \in S_0$, $M, s_0 \models f$
- Example: Does $M \models EX\ p$ or $M \models \neg EX\ p$?



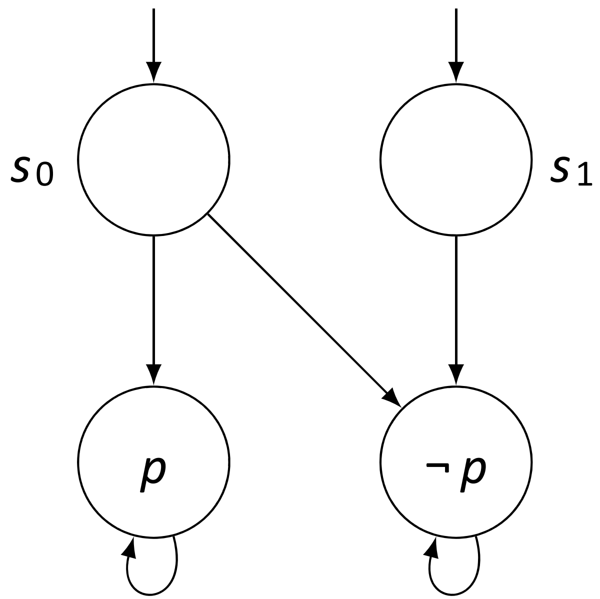
Solution

$M \models EX\ p$



Semantics of CTL*

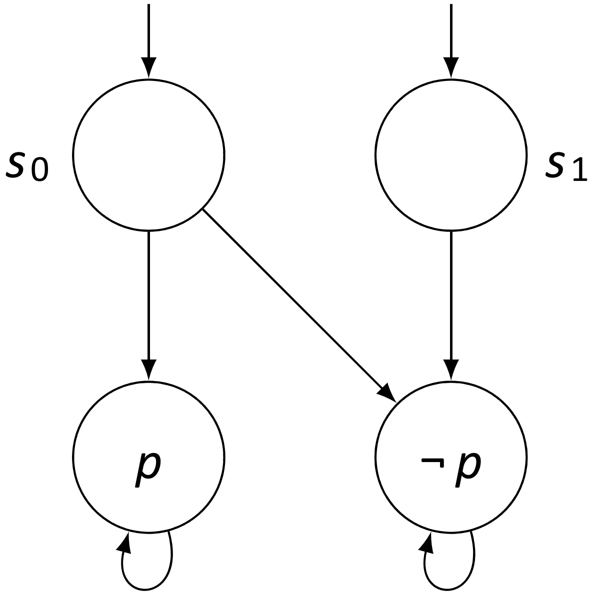
- $M \models f \Leftrightarrow$ for all initial states $s_0 \in S_0$, $M, s_0 \models f$
- Example: Does $M \models EX\ p$ or $M \models \neg EX\ p$?



Semantics of CTL*

- $M \models f \;\Leftrightarrow\;$ for all initial states $s_0 \in S_0$, $\;M, s_0 \models f$

- Example: Does $M \models EX\, p$ or $M \models \neg EX\, p$?



Neither

Exercise 1

Question:

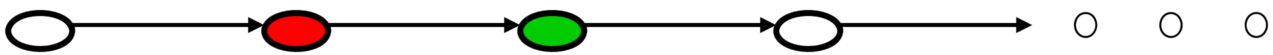
- Given a, b $\in$ AP
  How do all paths that satisfy (**F**b) **U** a look like?

Exercise 1

Question:

- Given a, b $\in$ AP
  How do all paths that satisfy (**F**b) **U** a look like?

(**F**b) **U** a



ToDo Exercise 2

Question:

For $p \in AP$, what is the meaning of the following formulas? That is, when does π satisfy each of the formulas:

- $\pi \models \mathbf{GF}\ p$
- $\pi \models \mathbf{FG}\ p$

Exercise 2

Question:

For p ∈ AP, what are the meaning of the following formulas? That is, when does π satisfy each of the formulas:

- $\pi \vDash \mathbf{GF}\ p$ Infinitely often p along π
- $\pi \vDash \mathbf{FG}\ p$ Finitely often ¬p along π

ToDo Exercise 2

Question:

For $p \in AP$, what are the meaning of the following formulas? That is, when does s satisfy each of the formulas:

- $s \models \mathbf{EGF}\ p$
- $s \models \mathbf{EG}\ \mathbf{EF}\ p$

- $\pi \models \mathbf{GF}\ p$ Infinitely often p along π
- $\pi \models \mathbf{FG}\ p$ Finitely often $\neg p$ along π

Exercise 2

Question:

For p ∈ AP, what are the meaning of the following formulas? That is, when does s satisfy each of the formulas:

- $s \models$ **EGF** p — There exists a path with satisfies infinitely often p
- $s \models$ **EG EF** p — There exists a path in which we can reach p from all states

- $\pi \models$ **GF** p — Infinitely often p along π
- $\pi \models$ **FG** p — Finitely often $\neg$p along π

ToDo Exercise 3

Question:

When does π satisfy the formula:

- $\pi \models (\mathbf{G}a)\ \mathbf{U}\ (\mathbf{G}b)$

Answer:

Exercise 3

Question:

When does π satisfy the formula:

- $\pi \vDash (\mathbf{G}a)\ \mathbf{U}\ (\mathbf{G}b)$

Answer:

- $(\mathbf{G}a)\ \mathbf{U}\ (\mathbf{G}b) \equiv \mathbf{G}b \vee (\mathbf{G}a \wedge \mathbf{FG}b)$

Properties of CTL*

The operators $\vee$, $\neg$, **X**, **U**, **E** are sufficient to express any CTL* formula:

- $f \wedge g \;\equiv\; \neg(\neg f \vee \neg g)$
- $f\,\mathbf{R}\,g \;\equiv\; \neg(\neg f\,\mathbf{U}\,\neg g)$
- $\mathbf{F}\,f \;\equiv\; \text{true}\,\mathbf{U}\,f$
- $\mathbf{G}\,f \;\equiv\; \neg\,\mathbf{F}\,\neg f$
- $\mathbf{A}\,(f) \;\equiv\; \neg\,\mathbf{E}\,(\neg f)$

Negation Normal Form (NNF)

- Formulas in Negation Normal Form (NNF) are formulas in which negations are applied only to atomic propositions

- Every CTL* formula is equivalent to a CTL* formula in NNF
- Negations can be "pushed" inwards.
  $\neg\, \mathbf{E}\, f \equiv \mathbf{A}\, \neg f$
  $\neg\, \mathbf{G}\, f \equiv \mathbf{F}\, \neg f$
  $\neg\, \mathbf{X}\, f \equiv \mathbf{X}\, \neg f$
  $\neg\, (\, f\, \mathbf{U}\, g\,) \equiv (\, \neg f\, \mathbf{R}\, \neg g\,)$

Negation Normal Form (NNF)

- Negations can be "pushed" inwards.
  $\neg\, \mathbf{E}\, f \;\equiv\; \mathbf{A}\, \neg f$
  $\neg\, \mathbf{G}\, f \;\equiv\; \mathbf{F}\, \neg f$
  $\neg\, \mathbf{X}\, f \;\equiv\; \mathbf{X}\, \neg f$
  $\neg\, (\, f\, \mathbf{U}\, g\,) \equiv (\, \neg f\, \mathbf{R}\, \neg g\,)$

- Example:
  Transforming a formula into NNF:

- $\neg(\, (a\, \mathbf{U}\, b\,) \vee \mathbf{F}\, c) \equiv$

Negation Normal Form (NNF)

- Negations can be "pushed" inwards.
  $\neg \mathbf{E}\, f \equiv \mathbf{A} \neg f$
  $\neg \mathbf{G}\, f \equiv \mathbf{F} \neg f$
  $\neg \mathbf{X}\, f \equiv \mathbf{X} \neg f$
  $\neg (f \,\mathbf{U}\, g) \equiv (\neg f \,\mathbf{R}\, \neg g)$

- Example:
  Transforming a formula into NNF:

- $\neg((a \,\mathbf{U}\, b) \vee \mathbf{F}\, c) \equiv (\neg(a \,\mathbf{U}\, b) \wedge \neg \mathbf{F}\, c) \equiv$
  $(((\neg a) \,\mathbf{R}\, (\neg b)) \wedge (\mathbf{G} \neg c)$

Useful sublogics of CTL*

- CTL, ACTL and ACTL* are branching-time temporal logics
  - Can describe the branching of the computation tree by applying nested path quantifications
- LTL is a linear-time temporal logic
  - Describes the paths in the computation tree, using only **one, outermost universal quantification**
- CTL and LTL are most widely used

LTL/CTL/CTL*

LTL consists of state formulas of the form **A**f

- f is a path formula, containing no path quantifiers
- LTL is interpreted over infinite computation paths

CTL consists of state formulas, where path quantifiers and temporal operators appear in pairs:

- **AG**, **AU**, **AX**, **AF**, **AR**, **EG**, **EU**, **EX**, **EF**, **ER**
- CTL is interpreted over infinite computation trees

CTL* allows any combination of temporal operators and path quantifiers.
It includes both LTL and CTL

LTL

State formulas:

- **A**f where f is a path formula

Path formulas:

- $p \in AP$
- $\neg f_1$, $f_1 \lor f_2$, $f_1 \land f_2$, $\mathbf{X} f_1$, $\mathbf{G} f_1$, $\mathbf{F} f_1$, $f_1 \mathbf{U} f_2$, $f_1 \mathbf{R} f_2$

  where f_1 and f_2 are path formulas

LTL is the set of all state formulas

CTL

CTL is the set of all state formulas, defined below (by means of state formulas only):

- $p \in AP$
- $\neg g_1$, $g_1 \vee g_2$, $g_1 \wedge g_2$
- **AX** g_1, **AG** g_1, **AF** g_1, **A** $(g_1$ **U** $g_2)$, **A** $(g_1$ **R** $g_2)$
- **EX** g_1, **EG** g_1, **EF** g_1, **E** $(g_1$ **U** $g_2)$, **E** $(g_1$ **R** $g_2)$

where g_1 and g_2 are state formulas

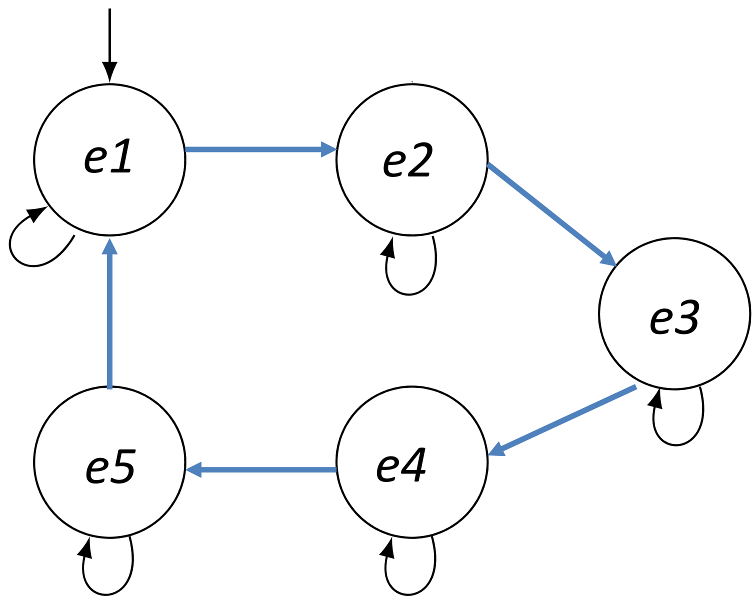

THANK YOU!

Semantics of CTL*

- $M \vDash f \Leftrightarrow$ for all initial states $s_0 \in S_0$: $M, s_0 \vDash f$

ToDo

- Example: Does $M \vDash EX\, p$ or $M \vDash \neg EX\, p$?



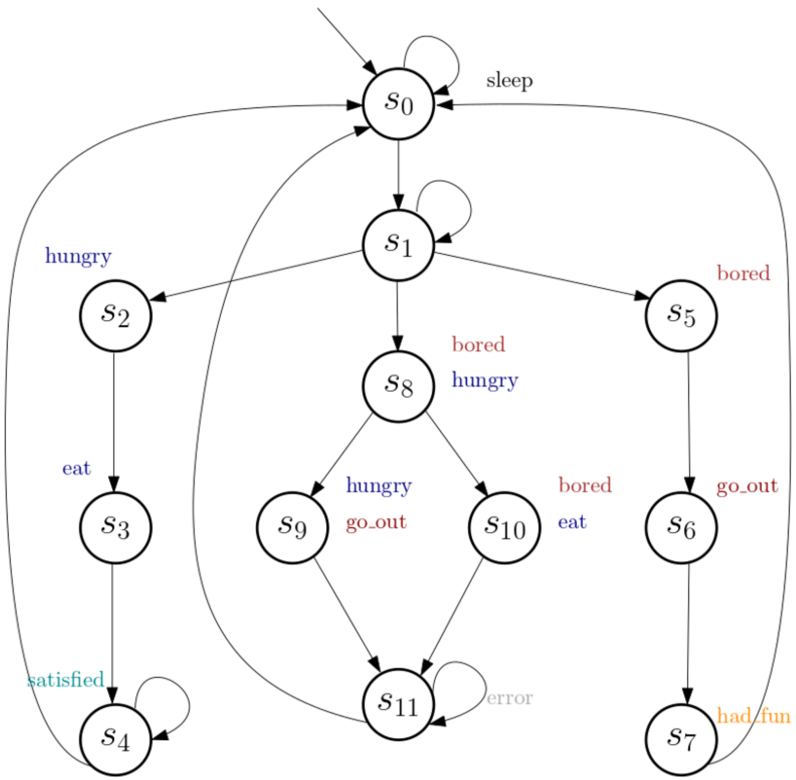
Model Checking Homework 5

Deadline: 14 April 2022 4:00pm
Send solution to: modelchecking@iaik.tugraz.at

Tempesta is a cheerful and simple dog. She is so simple that her behaviour can be represented by a Kripke structure with just 12 states (see figure). Every hour Tempesta undergoes a transition available from whatever state she is in. The atomic propositions AP = {sleep, hungry, bored, eat, go_out, satisfied, had_fun, error} indicate her activity.

This is a description of her behaviour:

Tempesta starts the day asleep (s_0). At each hour, she might wake up (s_1), and in the next hour migl be hungry (s_2), bored (s_5) or both (s_8). If she is only hungry, she will eat (s_3), be satisfied (s_4) and remain satisfied until she sleeps again (s_0). If she is only bored, she will go out (s_6), have fun (s_7) and be so tired that she directly goes to sleep (s_0). If she is bored and hungry at the same time, she will either try to go out while hungry (s_9) or eat while bored (s_{10}), but she will not manage and thus will enter into an error state (s_{11}), in which she will remain until she goes to sleep (s_0).



Task 4a [8 points *]. Translate these sentences to CTL* formulas. Indicate for each formula whether it in CTL, LTL, both or neither. Indicate also if the Kripke structure in the figure satisfies your sentence, and give an informal explanation of why.

1. From any state, Tempesta will eventually be hungry, and once she is hungry, she will eventually be satisfied in the future.
2. When Tempesta is hungry and not bored, she will sleep before reaching an error.
3. It is possible that Tempesta never wakes up from her sleep.
4. It is possible that Tempesta never stops eating.
5. Before Tempesta goes to sleep forever, which will eventually happen, she will have eaten.
6. It can be the case that Tempesta is asleep, and continues asleep for two hours more.
7. In any case, after Tepesta eats she is satisfied
8. When Tempesta wakes up, she requires at least 2 hours to sleep again
9. It is possible that Tempesta is infinitely often hungry and finitely often bored.

(*) Each sentence is worth 1 point, there are 9 sentences, so you are allowed one mistake.

Task 4b [2 points]. Tempesta wants to improve herself to enter her error state less often. When she is hungry and bored, if she decides to eat, she will (1) suppress the boredom while eating, (2) go out later and then being (3) satisfied and tired, (4) go to sleep. Write a modified Kripke structure that implements this new behaviour. From the previous sentences, is there any that the previous model of Tempesta did not satisfy and this new model satisfies? If so, which one?