COL:750/7250

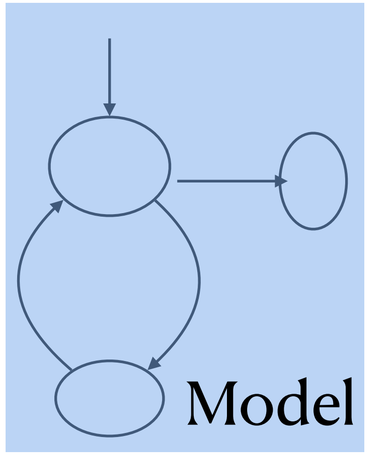
Foundations of Automatic Verification

Instructor: Priyanka Golia

Course Webpage

https://priyanka-golia.github.io/teaching/COL-750-COL7250/index.html

Does Code Satisfy Requirements ?



Does Model Satisfy Logical formulation: LTL/CTL Formula ?

Model Checking

Model Checking Algorithm

$$M, s \models F?$$

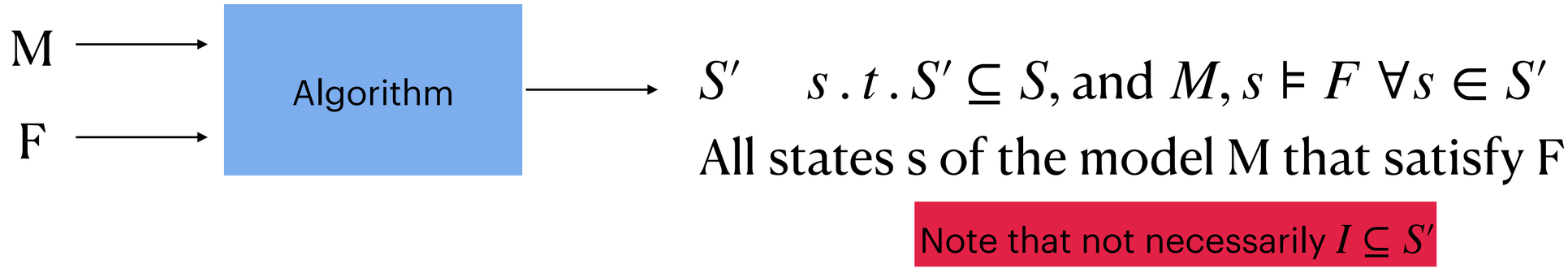
Yes, if $M, s \models F, \forall s \in I$

No, if $\exists s \in I\ s\,.\,t\,.\,M, s \not\models F,$

A path of the system M demonstrating that M can't satisfy F

Model Checking Algorithm

$$M, s \models F?$$



CTL Model Checking Algorithm — Labelling Algorithm

1. Input — a Model M, and a CTL formula F.
2. Output — S' (the set of states of M that satisfy the formula F.)

Key idea — build from sub-formulas.

1. Label the states of M with the sub-formulas of F that are satisfied at the state.
2. Starting with smallest sub formula and working towards F.

$$F = (\forall \Box (p \rightarrow \forall \Diamond q))$$

q $\forall \Diamond q$ p $p \rightarrow \forall \Diamond q$ $\forall \Box (p \rightarrow \forall \Diamond q)$

CTL Model Checking Algorithm — Labelling Algorithm

1. Input — a Model M, and a CTL formula F.
2. Output — S' (the set of states of M that satisfy the formula F.)

Key idea — build from sub-formulas.

1. Constructs the set of states where the formula holds:
$[F] := s \in S$ **s.t.** $M, s \vDash F$
2. Then, compare the $[F]$ with the set of initial states : $I \subseteq [F]$?

Compute $[F]$ in "bottom-up" on the structure of formula — computing $[F_i]$ for each sub-formula F_i of F.

$F = (\forall \square (p \rightarrow \forall \Diamond q))$ q $\forall \Diamond q$ p $p \rightarrow \forall \Diamond q$ $\forall \square (p \rightarrow \forall \Diamond q)$

CTL Model Checking Algorithm — Labelling Algorithm

Compute $[F]$ in "bottom-up" on the structure of formula — computing $[F_i]$ for each sub-formula F_i of F.

How to compute $[F_i]$?

Case Analysis

Recall given $M := < S, I, R, L >$

$\perp$ — no states are labelled with $\perp$ $[\perp] = \{\}$

P — label s with p if $p \in L(s)$ $[p] = \{s \mid p \in L(s)\}$

$\neg F_1$ — label s with $\neg F_1$ if s is NOT already labelled with F_1 $[\neg F_1] = S \backslash [F_1]$

$F_1 \wedge F_2$ — label s with $F_1 \wedge F_2$ if s is already labelled both F_1 and F_2

$[F_1 \wedge F_2] = [F_1] \cap [F_2]$

CTL Model Checking Algorithm — Labelling Algorithm

Case Analysis

Recall given $M := < S, I, R, L >$

$\exists \mathbf{N} F_1$

If any state s is labelled with F_1 if one of its successor is labelled with F_1

$$[\exists \mathbf{N} F_1] = \{s \in S \mid \exists s' < s, s' > \in R \wedge s' \in [F_1]\}$$

$[\exists \mathbf{N} F_1]$ is called pre-image of $[F_1]$ (pre($[F_1]$)

CTL Model Checking Algorithm — Labelling Algorithm

Case Analysis

Recall given $M := < S, I, R, L >$

$\exists \Box F$

$\Box F \equiv F \wedge \mathbf{N}(\Box F)$

$\exists \Box F \equiv F \wedge \exists \mathbf{N}(\exists \Box F)$

Label any state with $\exists \Box F_1$ if

1. it is labelled with F_1 and one of its successor is labelled with $\exists \Box F_1$ until there is no change.

$[\exists \Box F] = [F] \cap pre([\exists \Box F])$

CTL Model Checking Algorithm — Labelling Algorithm

Case Analysis

Recall given $M := < S, I, R, L >$

$$\exists \Box F$$

$$\Box F \equiv F \wedge \mathbf{N}(\Box F)$$

$$\exists \Box F \equiv F \wedge \exists \mathbf{N}(\exists \Box F)$$

$$[\exists \Box F] = [F] \cap pre([\exists \Box F])$$

We can compute this inductively.

$$X_1 = [F_1]$$

$$X_2 = X_1 \cap pre(X_1)$$

..

$$X_{j+1} = X_j \cap pre(X_j)$$

Since $X_{j+1} \subseteq X_j$ for every $j \geq 0$, thus a fix point always exists.

CTL Model Checking Algorithm — Labelling Algorithm

Case Analysis

Recall given $M := < S, I, R, L >$

$\exists F_1 \mathbf{U} F_2$

$F_1 \mathbf{U} F_2 \equiv F_2 \vee (F_1 \wedge \mathbf{N}(F_1 \mathbf{U} F_2))$

$\exists F_1 \mathbf{U} F_2 \equiv F_2 \vee (F_1 \wedge \exists \mathbf{N} \exists (F_1 \mathbf{U} F_2))$

Label any state with $\exists F_1 \mathbf{U} F_2$ if

1. it is labelled with F_2, or
2. it is labelled with F_1 and one of its successor is labelled with $\exists F_1 \mathbf{U} F_2$ until there is no change.

$$[\exists F_1 \mathbf{U} F_2] = [F_2] \cup ([F_1] \cap pre([\exists (F_1 \mathbf{U} F_2)]))$$

CTL Model Checking Algorithm — Labelling Algorithm

Case Analysis

Recall given $M := < S, I, R, L >$

$$\exists F_1 \mathbf{U} F_2$$

$$F_1 \mathbf{U} F_2 \equiv F_2 \vee (F_1 \wedge \mathbf{N}(F_1 \mathbf{U} F_2))$$

$$\exists F_1 \mathbf{U} F_2 \equiv F_2 \vee (F_1 \wedge \exists \mathbf{N} \exists (F_1 \mathbf{U} F_2))$$

$$[\exists F_1 \mathbf{U} F_2] = [F_2] \cup ([F_1] \cap pre([\exists (F_1 \mathbf{U} F_2)]))$$

We can compute this inductively.

$$X_1 = [F_2]$$

$$X_2 = X_1 \cup ([F_1] \cap pre(X_1))$$

..

$$X_{j+1} = X_j \cup ([F_1] \cap pre(X_j))$$

Since $X_{j+1} \supseteq X_j$ for every $j \geq 0$, thus a fix point always exists.

CTL Model Checking Algorithm — Labelling Algorithm

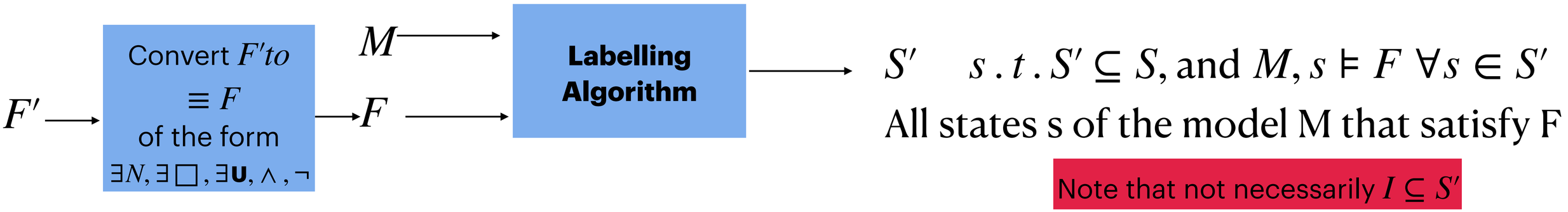
Binary operator $\bot, \neg, \wedge$

Temporal operator $\exists \mathbf{N}, \exists \square, \exists \mathbf{U}$

Form an adequate set of CTL formulas.

Any given CTL formula can be converted to equivalent CTL formula using only these operator.

$$M, s \models F?$$



CTL Model Checking Algorithm — Labelling Algorithm

Function Label(F, M){

Case F *of* :

True	return S
False	return {}
p	return $\{s \in S \mid p \in L(s)\}$
$\neg F_1$	return $S \backslash Label(F_1)$
$F_1 \wedge F_2$	return $Label(F_1) \cap Label(F_2)$
$\exists \mathbf{N} F_1$	return pre($Label(F_1)$)
$\exists \Box F_1$	return $Label\_EG(Label(F_1))$
$\exists F_1 \mathbf{U} F_2$	return $Label\_EU(Label(F_1), Label(F_2))$

End Case

CTL Model Checking Algorithm — Labelling Algorithm

$[\exists \mathbf{N} F_1] = pre([F_1])$

$[\exists \mathbf{N} F_1] = \{s \in S \mid \exists s' < s, s' > \in R \wedge s' \in [F_1]\}$

pre($[F_1]$) {

X = {}

For each s' in $[F_1]$ do:

For each s in S do:

If $< s, s' > \in R$:

$X = X \cup s$

Return X

}

CTL Model Checking Algorithm — Labelling Algorithm

$[\exists \Box F_1]$ $[\exists \Box F] = [F] \cap pre([\exists \Box F])$

$Label\_EG([F_1])$ {

X = $[F_1]$
Y = {}

While $X \neq Y$ do:

Y = X

$X = X \cap pre(X)$

Return X

}

CTL Model Checking Algorithm — Labelling Algorithm

$[\exists F_1 \mathbf{U} F_2]$ $[\exists F_1 \mathbf{U} F_2] = [F_2] \cup ([F_1] \cap pre([\exists (F_1 \mathbf{U} F_2)]))$

$Label\_EU([F_1], [F_2])\ \{$

X = $[F_2]$
Y = S

While $X \neq Y$ do:

Y = X

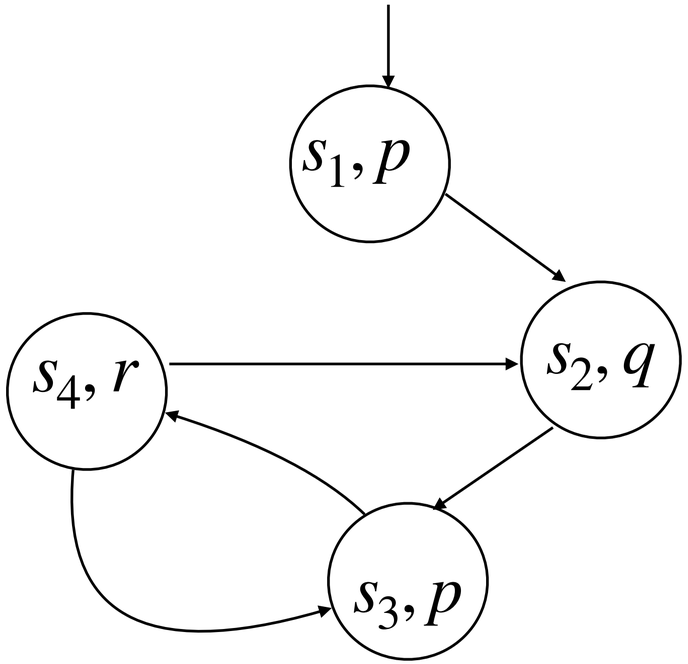
$X = X \cup ([F_1] \cap pre(X))$

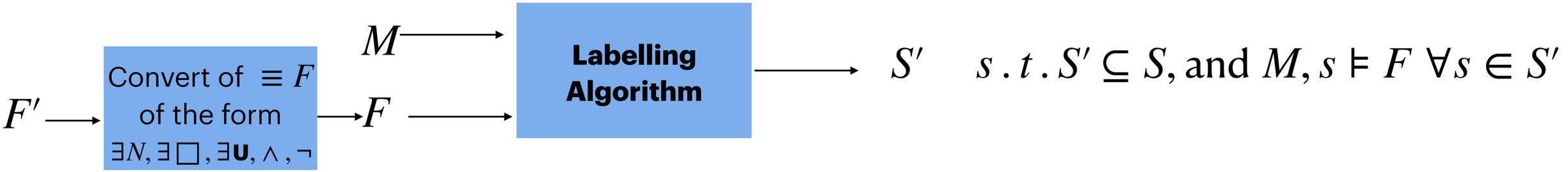
Return X

}

Labelling Algorithm — Example

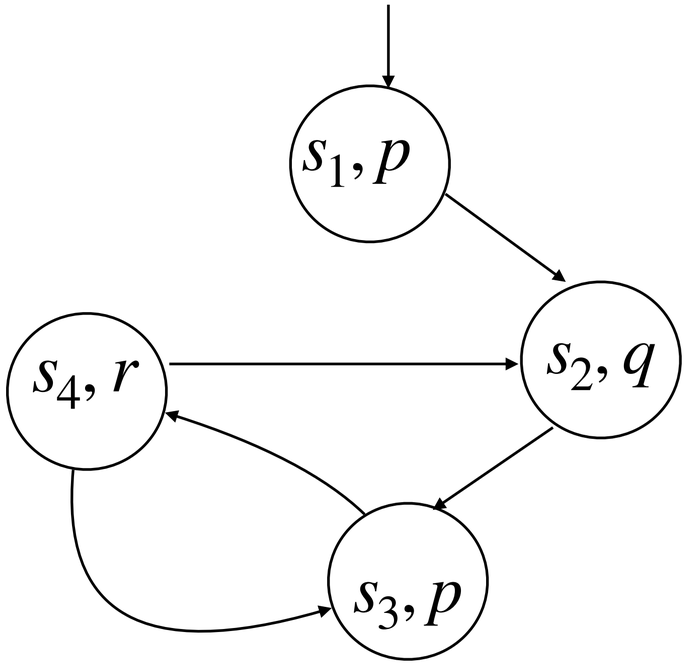
$$F' = (\forall \Box (p \rightarrow \forall \Diamond q))$$





Labelling Algorithm — Example

$$F' = (\forall \Box (p \rightarrow \forall \Diamond q))$$



$$\neg F' = \exists \Diamond \neg (p \rightarrow \forall \Diamond q)$$

$$\neg F' = \exists \Diamond (p \wedge \neg (\forall \Diamond q))$$

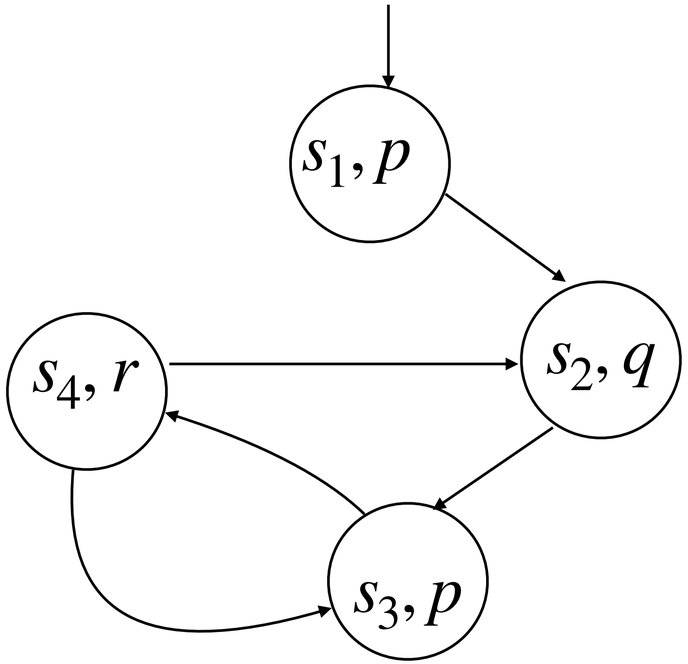
$$\neg F' = \exists \Diamond (p \wedge \exists \Box \neg q))$$

$$\neg F' = \exists (True \ \mathbf{U} \ (p \wedge \exists \Box \neg q)))$$

Labelling Algorithm — Example

$$\neg F' = True\ \mathbf{U}\ (p \wedge \exists \Box \neg q))$$

$$F = \neg(\neg F')$$



$[\text{True}] = \{1,2,3,4\}$

$[\neg q] = \{1,3,4\}$

$[\exists \Box \neg q] = \{3,4\}$

$[p] = \{1,3\}$

$[p \wedge \exists \Box \neg q] = \{3\}$

$[True \mathbf{U} (p \wedge \exists \Box \neg q)] = \{1,2,3,4\}$

$[\neg F'] = \{1,2,3,4\}$

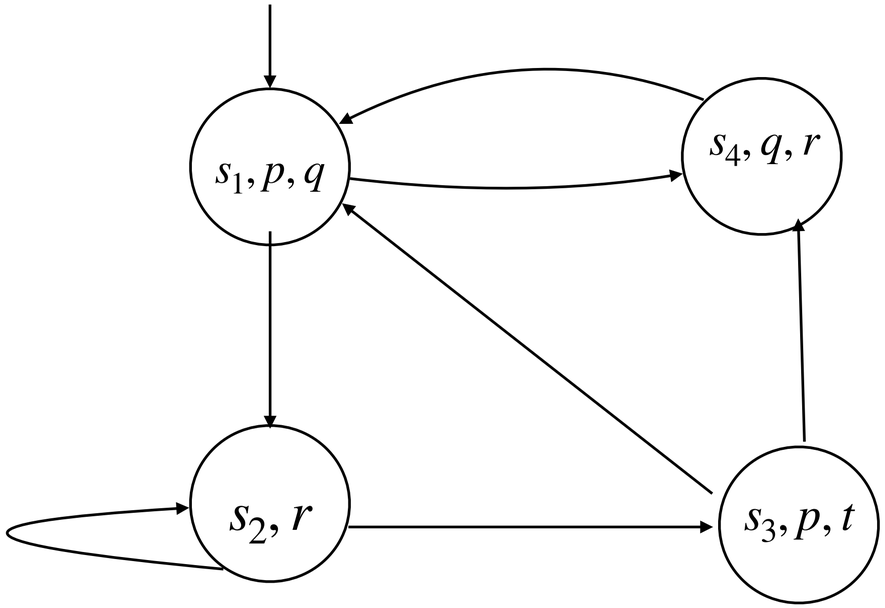
$[\text{F}] = \{\}$

Labelling Algorithm — Example

$$F = \forall \Box (\forall \Diamond q)$$

$$R = \mathbf{NN}(r)$$

$$S = \forall \Box (\exists \Diamond (p \vee r))$$



Exercise: find out the complexity of the labelling algorithm in terms of
number of connectives of the formula (say f), and
number of states of the model (say |S|), and
number of transitions in the model (say |R|).

How to improve *Label_EG* algorithm?

Recall strongly connected components.

1. Restrict the graph to states satisfying F_1.
2. Find the maximal strongly connected components (SCC)
3. Use backward breadth first search on the restricted graph to find any state that can reach SCC.