CS 181u Applied Logic

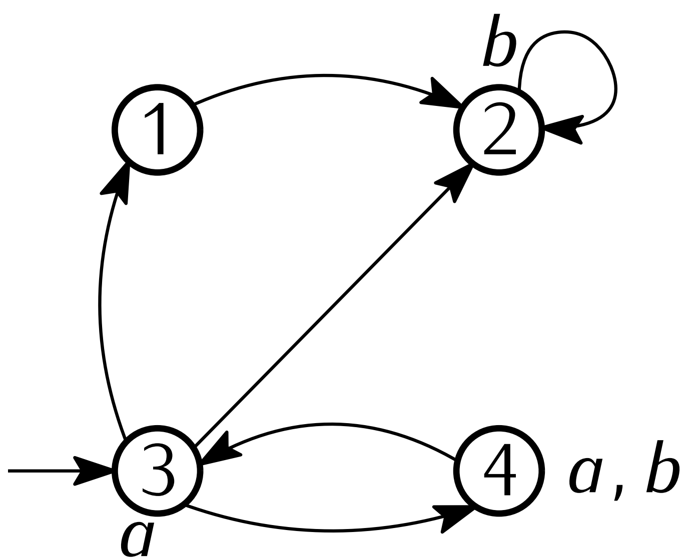
Lecture 12

Modeling a State Machine in νSMV

Current HW Problem 4: translate a transition system into a νSMV model. Write and verify some CTL properties. Generate example paths of CTL properties.

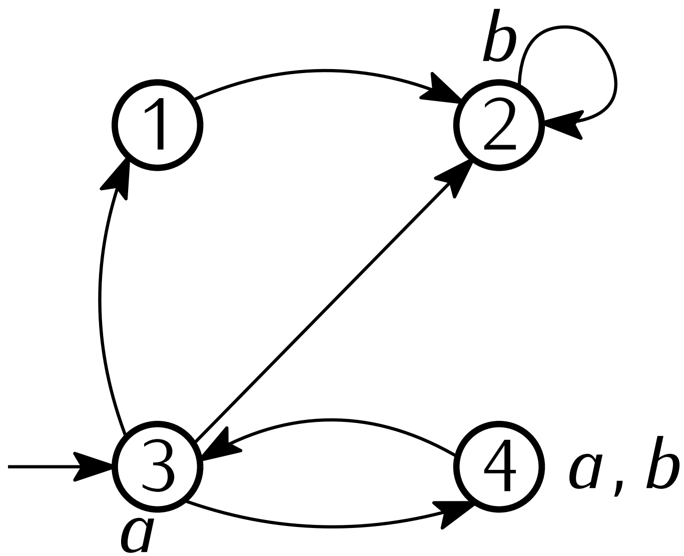
Modeling a State Machine in νSMV

Current HW Problem 4: translate a transition system into a νSMV model. Write and verify some CTL properties. Generate example paths of CTL properties.



Modeling a State Machine in νSMV

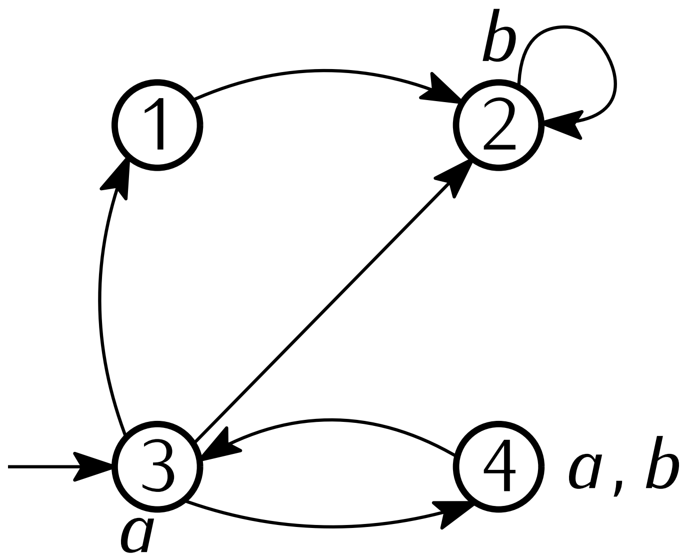
Current HW Problem 4: translate a transition system into a νSMV model. Write and verify some CTL properties. Generate example paths of CTL properties.




```
MODULE main

VAR
  s : 1..4;

ASSIGN
  init(s) := 3;

  next(s) :=
    case
      s = 1 : 2;
      s = 2 : 2;
      s = 3 : {1,2,4};
      s = 4 : 3;
    esac;

DEFINE
  a := s = 3 | s = 4;
  b := s = 2 | s = 4;
```

Modeling a State Machine in νSMV

Current HW Problem 4: translate a transition system into a νSMV model. Write and verify some CTL properties. Generate example paths of CTL properties.

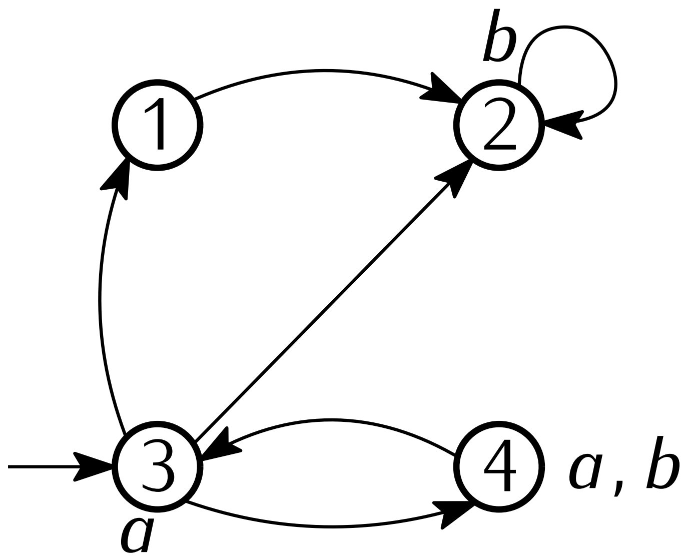



```
MODULE main

VAR
  s : 1..4;

ASSIGN
  init(s) := 3;

  next(s) :=
    case
      s = 1 : 2;
      s = 2 : 2;
      s = 3 : {1,2,4};
      s = 4 : 3;
    esac;

DEFINE
  a := s = 3 | s = 4;
  b := s = 2 | s = 4;
```

Instantaneous logical equivalence.

Modeling a State Machine in νSMV

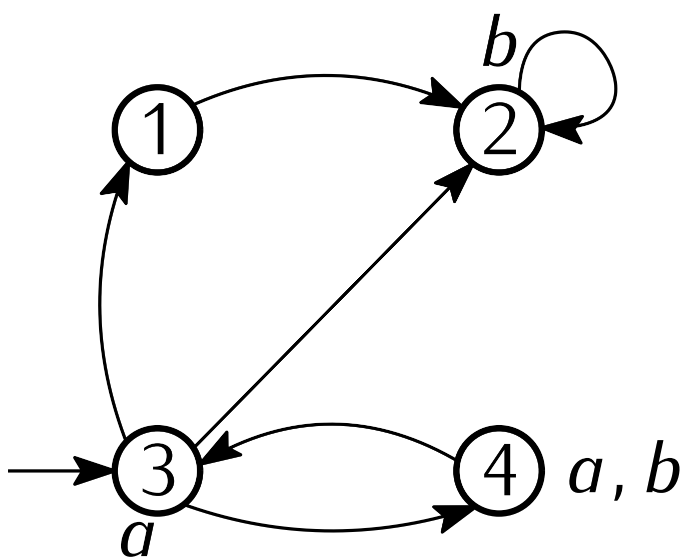
Current HW Problem 4: translate a transition system into a νSMV model. Write and verify some CTL properties. Generate example paths of CTL properties.



CTL property: It is possible that eventually b is always true.

Modeling a State Machine in νSMV

Current HW Problem 4: translate a transition system into a νSMV model. Write and verify some CTL properties. Generate example paths of CTL properties.



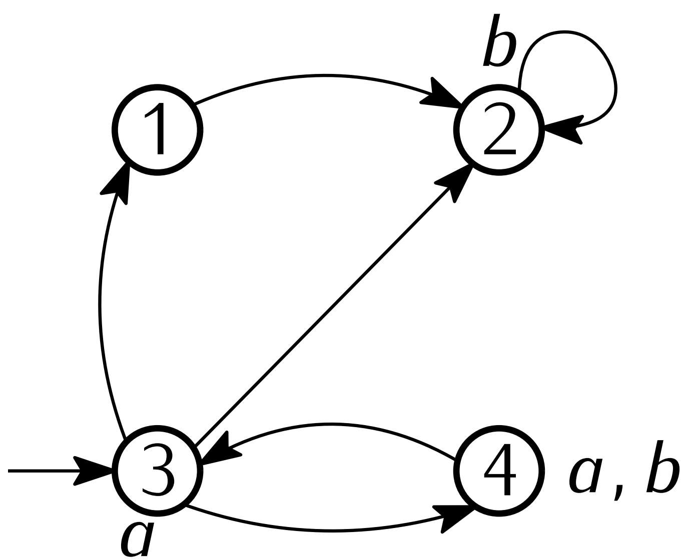
CTL property: It is possible that eventually b is always true.

$EF\ AG\ b$

CTLSPEC EF AG b

Modeling a State Machine in νSMV

Current HW Problem 4: translate a transition system into a νSMV model. Write and verify some CTL properties. Generate example paths of CTL properties.



CTL property: It is possible that eventually b is always true.

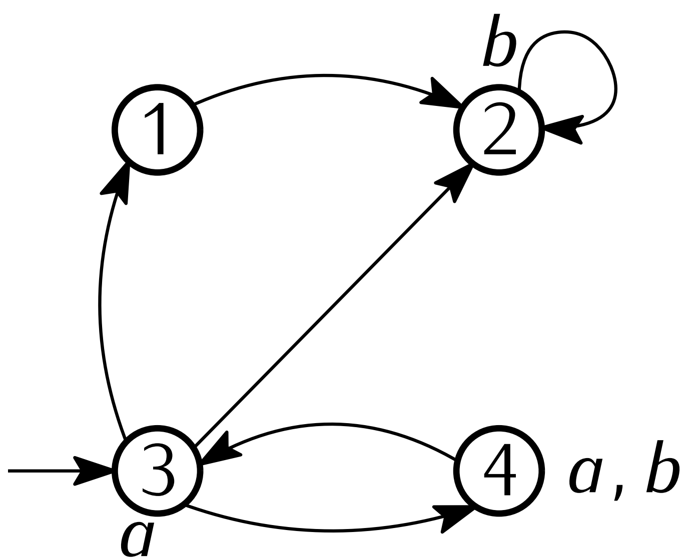
$EF\ AG\ b$

CTLSPEC EF AG b

How to produce a path that satisfies $EF\ AG\ b$?

Modeling a State Machine in νSMV

Current HW Problem 4: translate a transition system into a νSMV model. Write and verify some CTL properties. Generate example paths of CTL properties.



CTL property: It is possible that eventually b is always true.

$EF\ AG\ b$

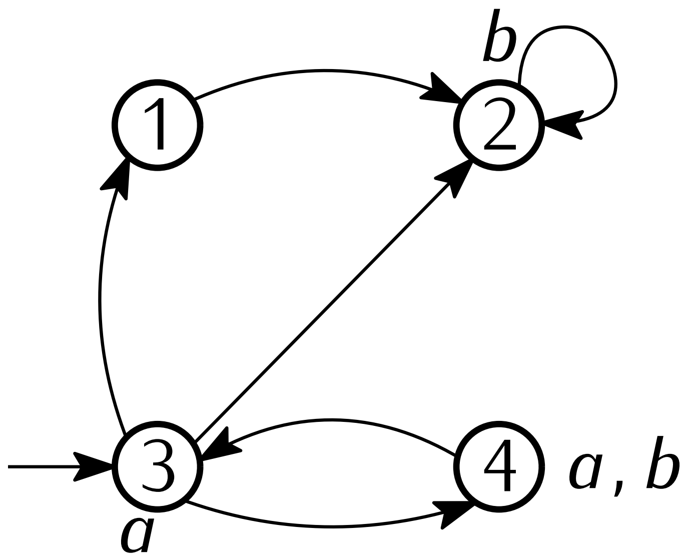
CTLSPEC EF AG b

How to produce a path that satisfies $EF\ AG\ b$?

Recall: if $\mathcal{M} \not\models \phi$ then νSMV produces a counterexample.

Modeling a State Machine in νSMV

Current HW Problem 4: translate a transition system into a νSMV model. Write and verify some CTL properties. Generate example paths of CTL properties.



CTL property: It is possible that eventually b is always true.

$EF\ AG\ b$

CTLSPEC EF AG b

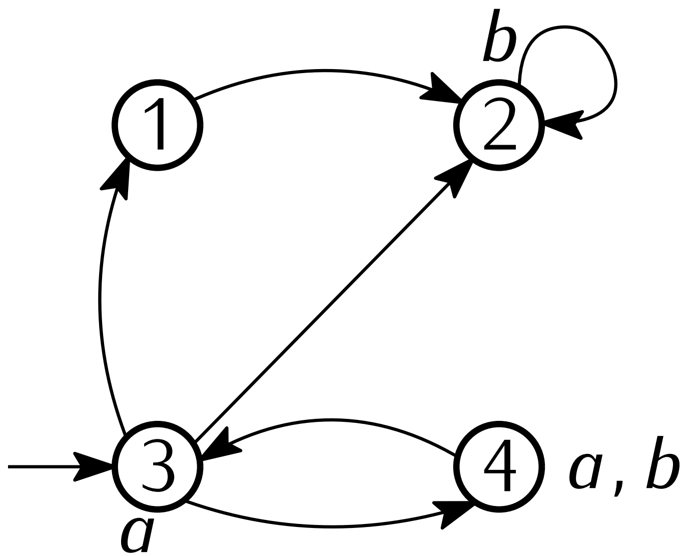
How to produce a path that satisfies $EF\ AG\ b$?

Recall: if $\mathcal{M} \not\models \phi$ then νSMV produces a counterexample.

If $\mathcal{M} \models \phi$ then checking $\mathcal{M} \models \neg\phi$ produces a counterexample to $\neg\phi$, which is a path π such that $\pi \models \phi$.

Modeling a State Machine in νSMV

Current HW Problem 4: translate a transition system into a νSMV model. Write and verify some CTL properties. Generate example paths of CTL properties.



CTL property: It is possible that eventually b is always true.

$EF\ AG\ b$

CTLSPEC EF AG b

$\boxed{\text{CTLSPEC ! EF AG b}}$

How to produce a path that satisfies $EF\ AG\ b$?

Recall: if $\mathcal{M} \not\models \phi$ then νSMV produces a counterexample.

If $\mathcal{M} \models \phi$ then checking $\mathcal{M} \models \neg\phi$ produces a counterexample to $\neg\phi$, which is $\boxed{\text{a path } \pi \text{ such that } \pi \models \phi}$.

Some useful νSMV capabilities

Using the c-preprocessor (cpp)

```
# define SIZE 10
MODULE main ...
 VAR
   x :  1 ..  SIZE
```

Some useful νSMV capabilities

Using the c-preprocessor (cpp)

```
# define SIZE 10
MODULE main ...
 VAR
   x :  1 ..  SIZE
```

Not the same as the νSMV keyword `DEFINE`!

Some useful νSMV capabilities

Using the c-preprocessor (cpp)

```
# define SIZE 10
MODULE main ...
 VAR
   x :  1 ..  SIZE
```

Not the same as the νSMV keyword `DEFINE`!

`$ NuSMV -pre cpp filename.smv`

"Find and replace" all instances of SIZE with 10

Some useful νSMV capabilities

Using the c-preprocessor (cpp)

```
# define SIZE 10
MODULE main ...
 VAR
   x :  1 ..  SIZE
```

Not the same as the νSMV keyword `DEFINE`!

`$ NuSMV -pre cpp filename.smv`

"Find and replace" all instances of SIZE with 10

Bounded model checking

Some useful νSMV capabilities

Using the c-preprocessor (cpp)

```
# define SIZE 10
MODULE main ...
 VAR
   x :  1 ..  SIZE
```

Not the same as the νSMV keyword `DEFINE`!

```
$ NuSMV -pre cpp filename.smv
```

"Find and replace" all instances of SIZE with 10

Bounded model checking

```
$ NuSMV -bmc filename.smv
```

Default bound = 10

Some useful νSMV capabilities

Using the c-preprocessor (cpp)

```
# define SIZE 10
MODULE main ...
 VAR
   x :  1 ..  SIZE
```

Not the same as the νSMV keyword `DEFINE`!

```
$ NuSMV -pre cpp filename.smv
```

"Find and replace" all instances of SIZE with 10

Bounded model checking

```
$ NuSMV -bmc filename.smv
```

Default bound = 10

```
$ NuSMV -bmc -bmc_length 20 filename.smv
```

Some useful νSMV capabilities

Using the c-preprocessor (cpp)

```
# define SIZE 10
MODULE main ...
 VAR
   x :  1 ..  SIZE
```

Not the same as the νSMV keyword `DEFINE`!

```
$ NuSMV -pre cpp filename.smv
```

"Find and replace" all instances of SIZE with 10

Bounded model checking

```
$ NuSMV -bmc filename.smv
```

Default bound $= 10$

```
$ NuSMV -bmc -bmc_length 20 filename.smv
```

Searches for counterexamples starting with length $= 1$ and going up to length_bound. Returns the smallest counterexample found.

Some useful νSMV capabilities

The case statement

```
x :=
  case
    y > 10 :  z - 1;
    TRUE : z + 1;
  esac
```

Some useful *v*SMV capabilities

The case statement

```
x :=
  case
    y > 10 :  z - 1;
    TRUE : z + 1;
  esac
```

The if-then-else operator: test-exp ? then-exp : else-exp

```
x := y > 10 ?  z + 1 :  z - 1;
```

Some useful *v*SMV capabilities

The case statement

```
x :=
  case
    y > 10 :  z - 1;
    TRUE : z + 1;
  esac
```

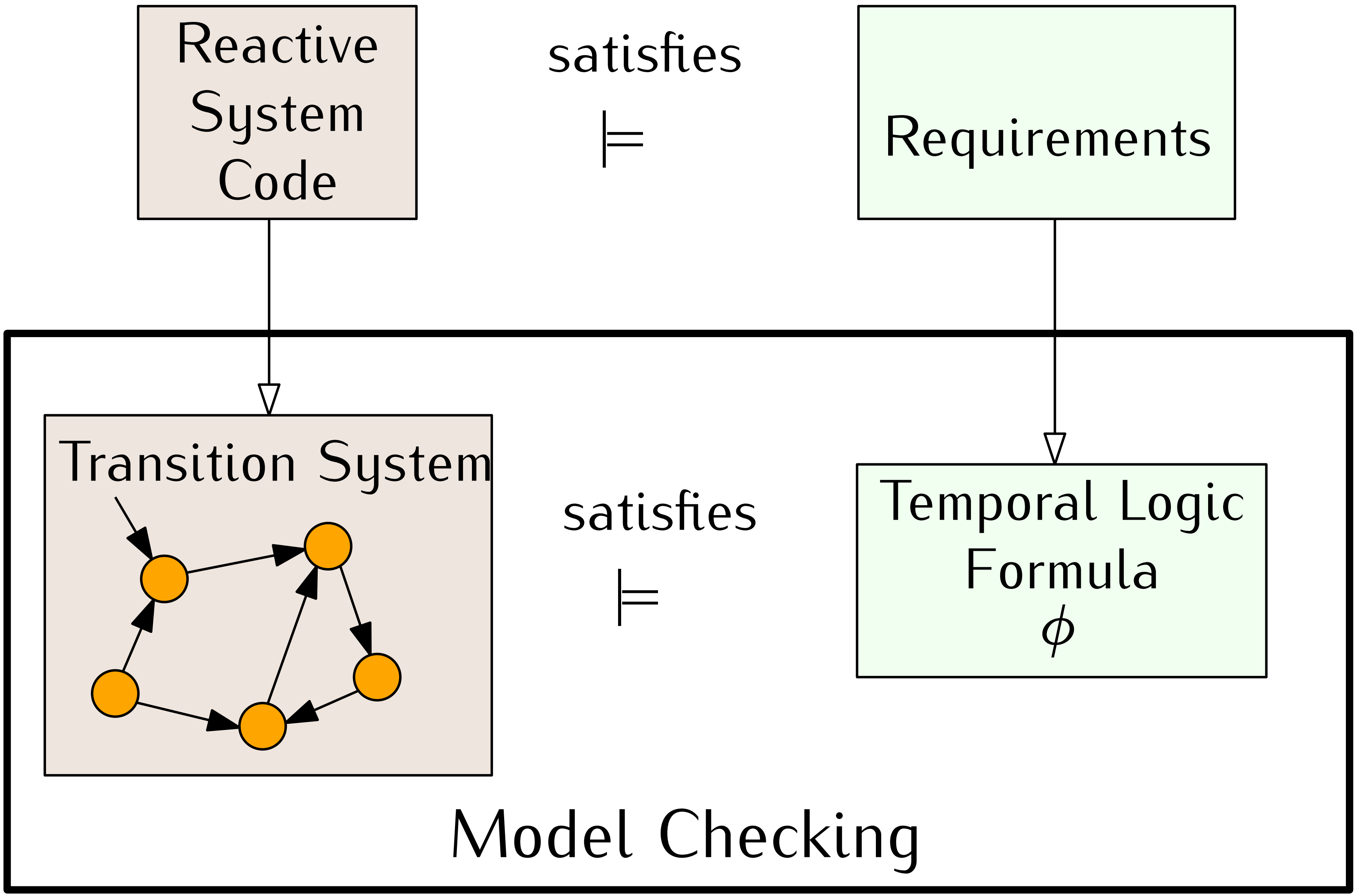
The if-then-else operator: test-exp ? then-exp : else-exp

```
x := y > 10 ?  z + 1 :  z - 1;
```

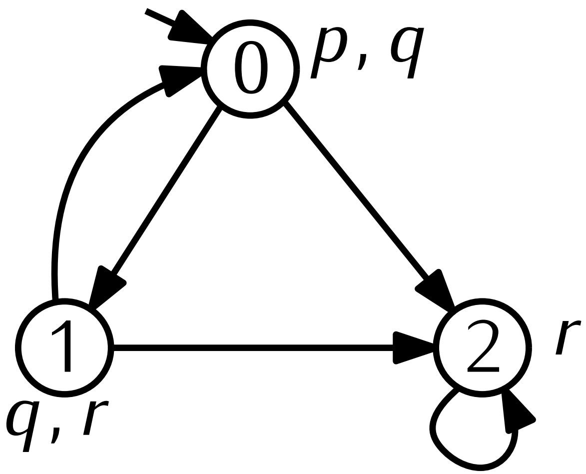
Arrays: array min .. max of enum-type-exp

```
array 0 ..  9 of boolean
array -4 ..  3 of {4, a, error}
```

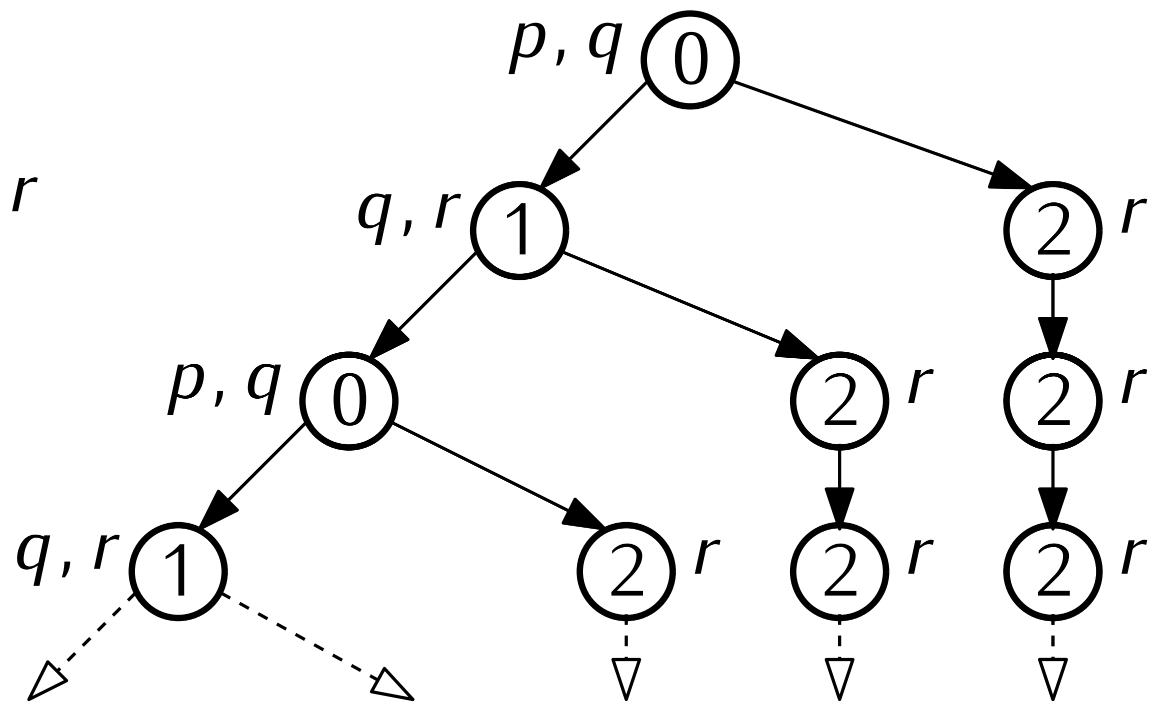
The Big Picture



Computation Tree Logic (CTL) Review



Computation tree for $\mathcal{M}$



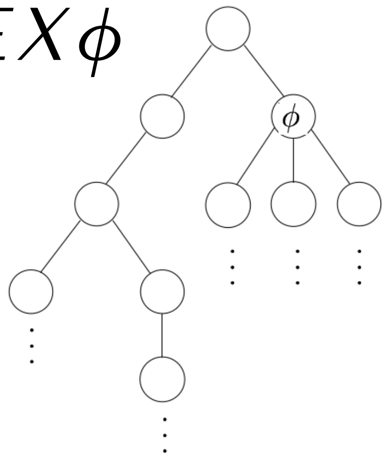
Computation Tree Logic (CTL) expresses properties of "alternative timelines".

$$\mathcal{M} \models \phi \Leftrightarrow \forall s \in I \;\; s \models \phi$$

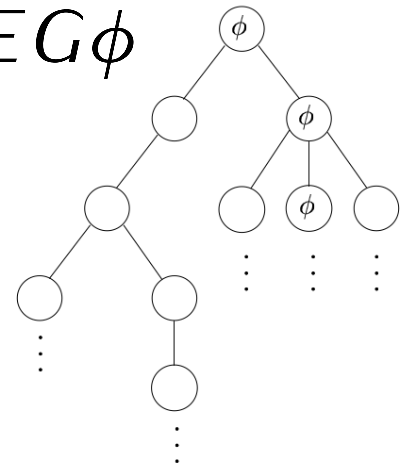
CTL Model Checking

CTL review $AG\phi$ $EG\phi$ $AF\phi$ $EF\phi$ $AX\phi$ $EX\phi$

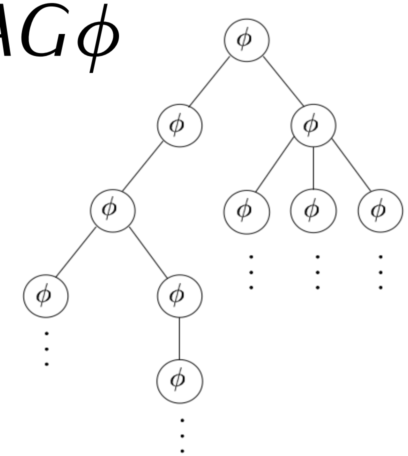
$EX\phi$



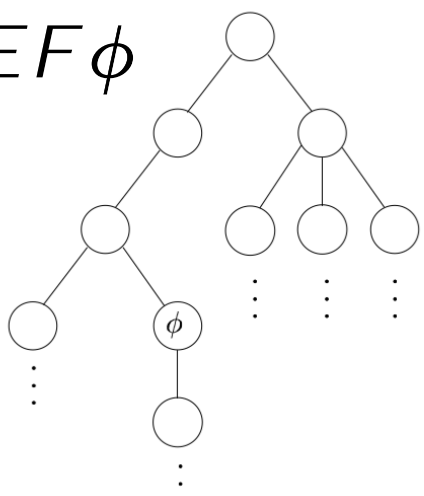
$EG\phi$



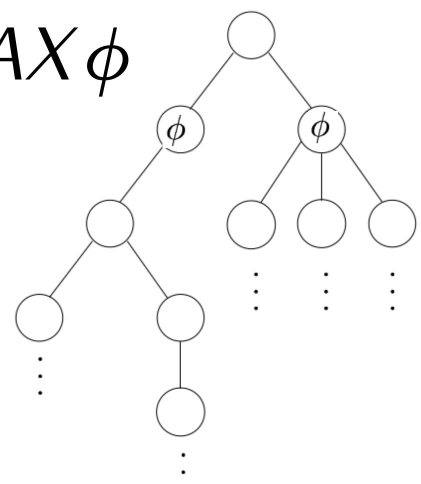
$AG\phi$



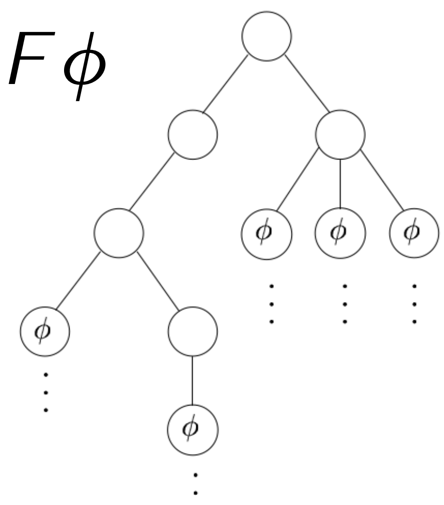
$EF\phi$



$AX\phi$



$AF\phi$



CTL Model Checking

$$\mathcal{M} \models \phi \Leftrightarrow \forall s \in I \quad s \models \phi$$

Given $\mathcal{M}$ and CTL formula ϕ
we want to check if $\mathcal{M} \models \phi$.

CTL Model Checking

$$\mathcal{M} \models \phi \Leftrightarrow \forall s \in I \;\; s \models \phi$$

Given $\mathcal{M}$ and CTL formula ϕ
we want to check if $\mathcal{M} \models \phi$.

One idea: come up with some algorithm that looks at the set of initial states I and outputs *true* or *false* depending on if $s \models \phi$ for all $s \in I$.

CTL Model Checking

$$\mathcal{M} \models \phi \Leftrightarrow \forall s \in I \quad s \models \phi$$

Given $\mathcal{M}$ and CTL formula ϕ
we want to check if $\mathcal{M} \models \phi$.

One idea: come up with some algorithm that looks at the set of initial states I and outputs *true* or *false* depending on if $s \models \phi$ for all $s \in I$.

A ***slightly*** **different idea:** figure out the set of states $S' \subseteq S$ such that for all $s \in S'$, $s \models \phi$. Then check if $I \subseteq S'$.

CTL Model Checking

$$\mathcal{M} \models \phi \Leftrightarrow \forall s \in I \;\; s \models \phi$$

Given $\mathcal{M}$ and CTL formula ϕ
we want to check if $\mathcal{M} \models \phi$.

One idea: come up with some algorithm that looks at the set of initial states I and outputs *true* or *false* depending on if $s \models \phi$ for all $s \in I$.

A *slightly* different idea: figure out the set of states $S' \subseteq S$ such that for all $s \in S'$, $s \models \phi$. Then check if $I \subseteq S'$.

This is easier.

CTL Model Checking

$$\mathcal{M} \models \phi \Leftrightarrow \forall s \in I \quad s \models \phi$$

Todays goal: an algorithm for CTL that does the following:

Input: $\mathcal{M}$ and ϕ
Output: all states of $\mathcal{M}$ that satisfy ϕ

Recall: Why so many operators?

AG EG AF EF AX EX AU EU

Recall: Why so many operators?

AG EG AF EF AX EX AU EU

The acts of the mind, wherein it exerts its power over simple ideas, are chiefly these three: Combining several simple ideas into one compound one, and thus all complex ideas are made. The second is bringing two ideas, whether simple or complex, together, and setting them by one another so as to take a view of them at once, without uniting them into one, by which it gets all its ideas of relations. The third is separating them from all other ideas that accompany them in their real existence: this is called abstraction, and thus all its general ideas are made.

SICP by Abelson, Sussman, and Sussman quoting John Locke from his *Essay Concerning Human Understanding*

Why so many operators?

We don't actually need any of them if we are OK with always writing temporal properties using first order logic and quantifying over states and paths.

Why so many operators?

We don't actually need any of them if we are OK with always writing temporal properties using first order logic and quantifying over states and paths.

However, they are useful to have on hand to state things concisely, like when writing νSMV specifications.

Why so many operators?

We don’t actually need any of them if we are OK with always writing temporal properties using first order logic and quantifying over states and paths.

However, they are useful to have on hand to state things concisely, like when writing νSMV specifications.

On the other hand, when performing meta-analysis of CTL, we need to examine each operator.

Why so many operators?

We don't actually need any of them if we are OK with always writing temporal properties using first order logic and quantifying over states and paths.

However, they are useful to have on hand to state things concisely, like when writing νSMV specifications.
On the other hand, when performing meta-analysis of CTL, we need to examine each operator.

Hence, it is good to reduce everything down to a smallest set of sufficiently expressive operators.

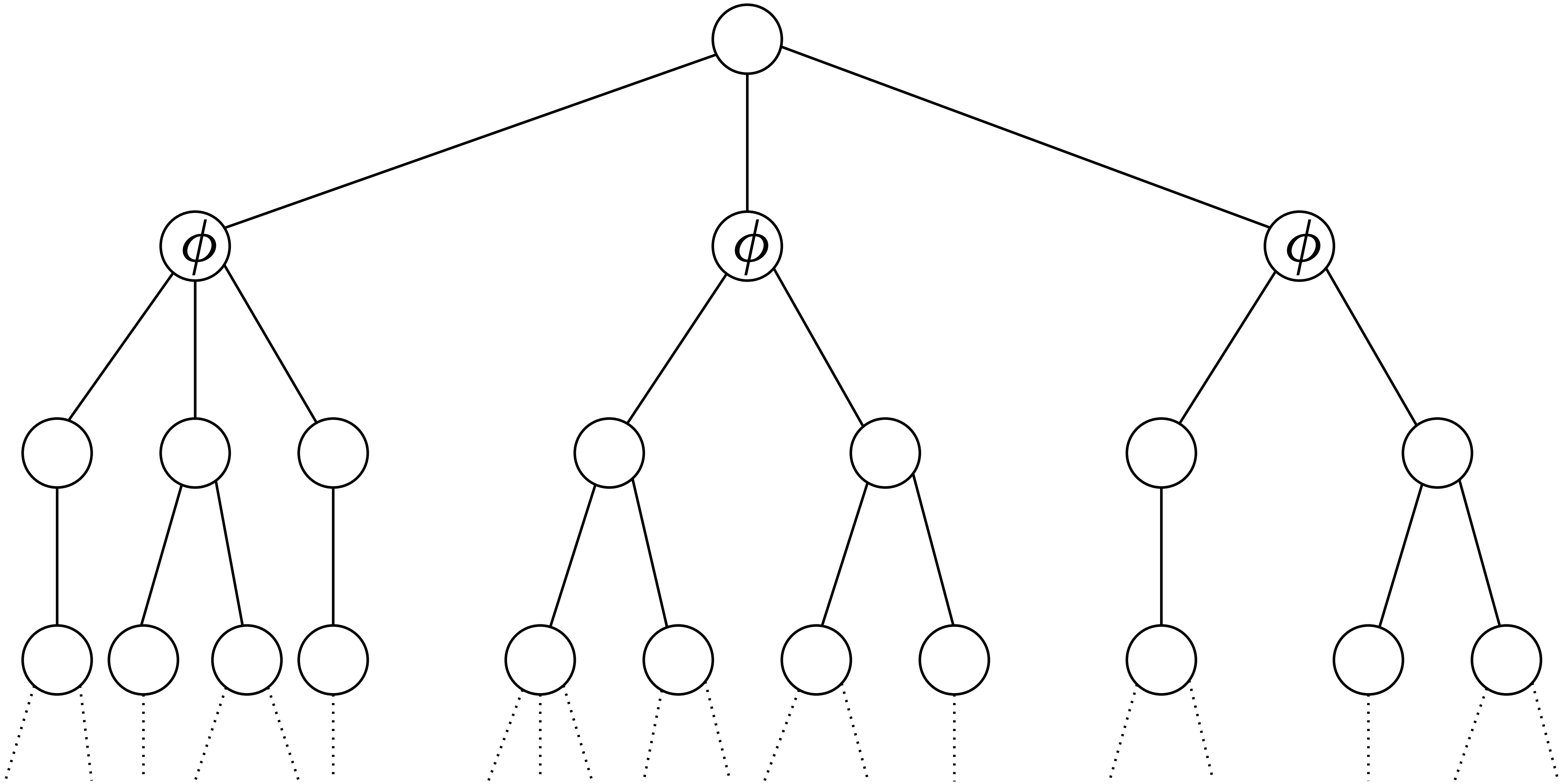
Adequate set of operators for CTL

Let's eliminate as many operators as possible by writing them in terms of other operators.

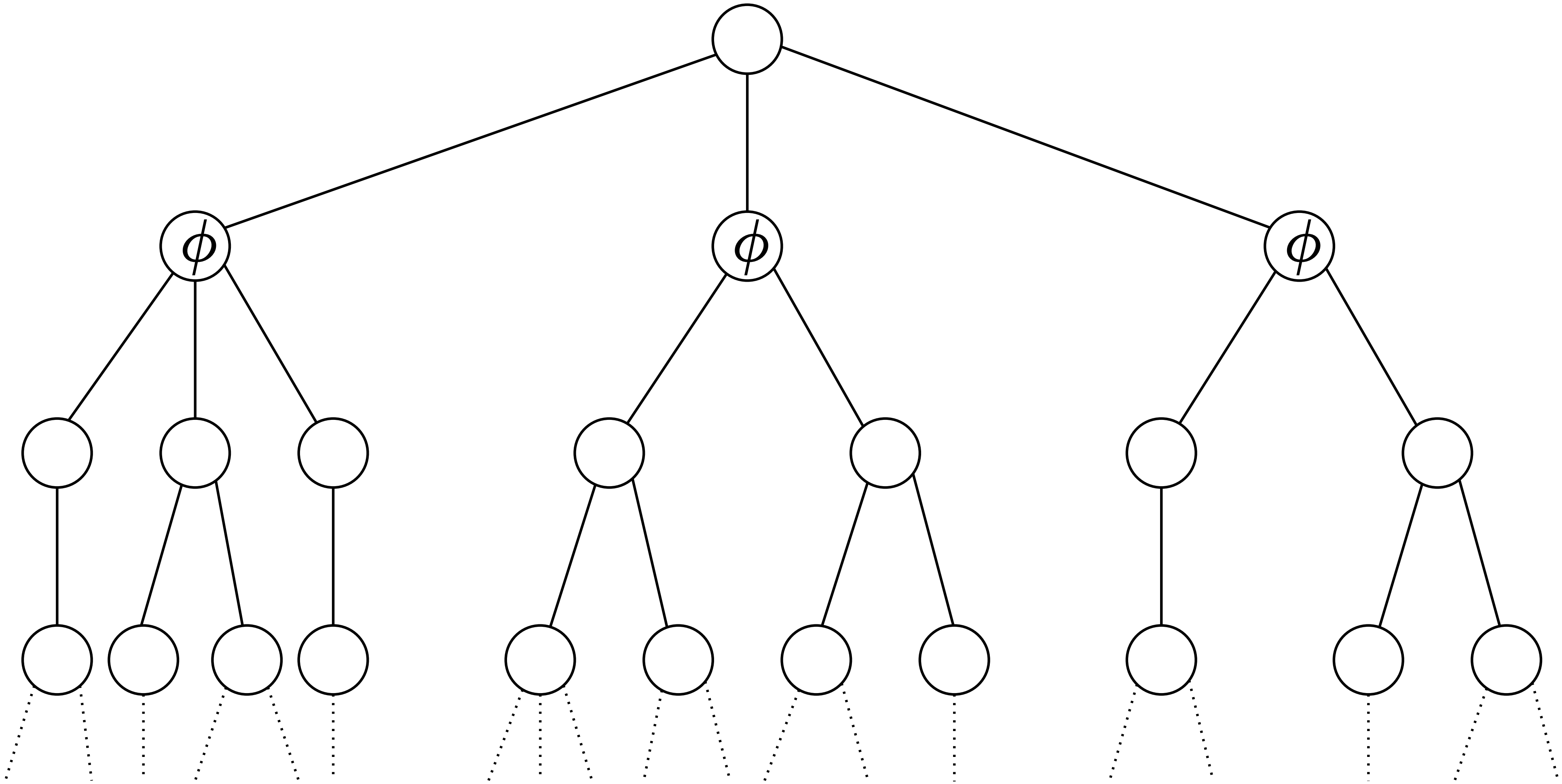
In fact, let's try to write everything in terms of E-properties EX, EU, and EG, and Boolean operations $\neg, \wedge$, and $\vee$.

(On your HW, you wrote some operators in terms of EX, EU, and AU.)

Get rid of $AX\phi$

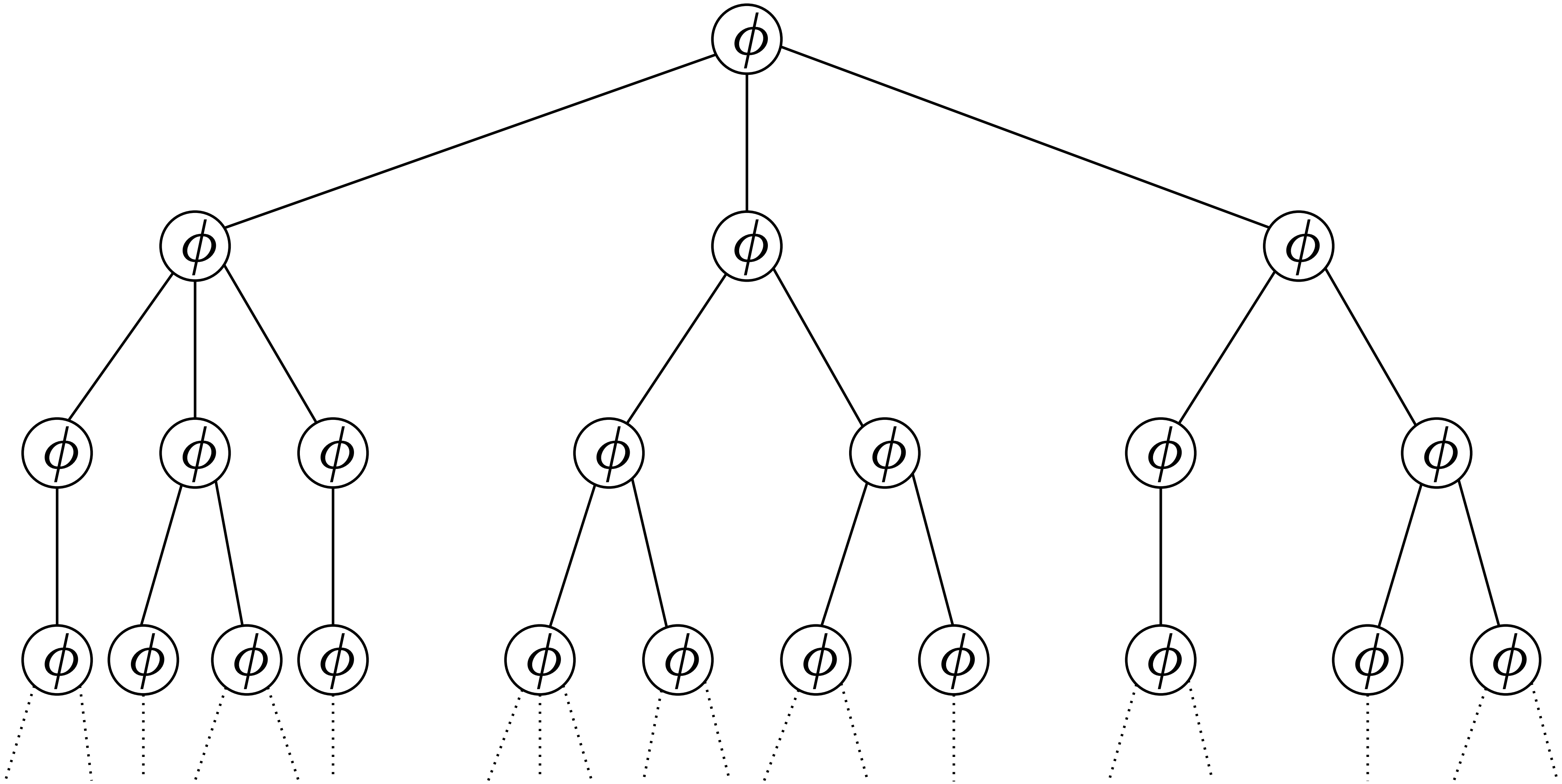


Get rid of $AX\phi$

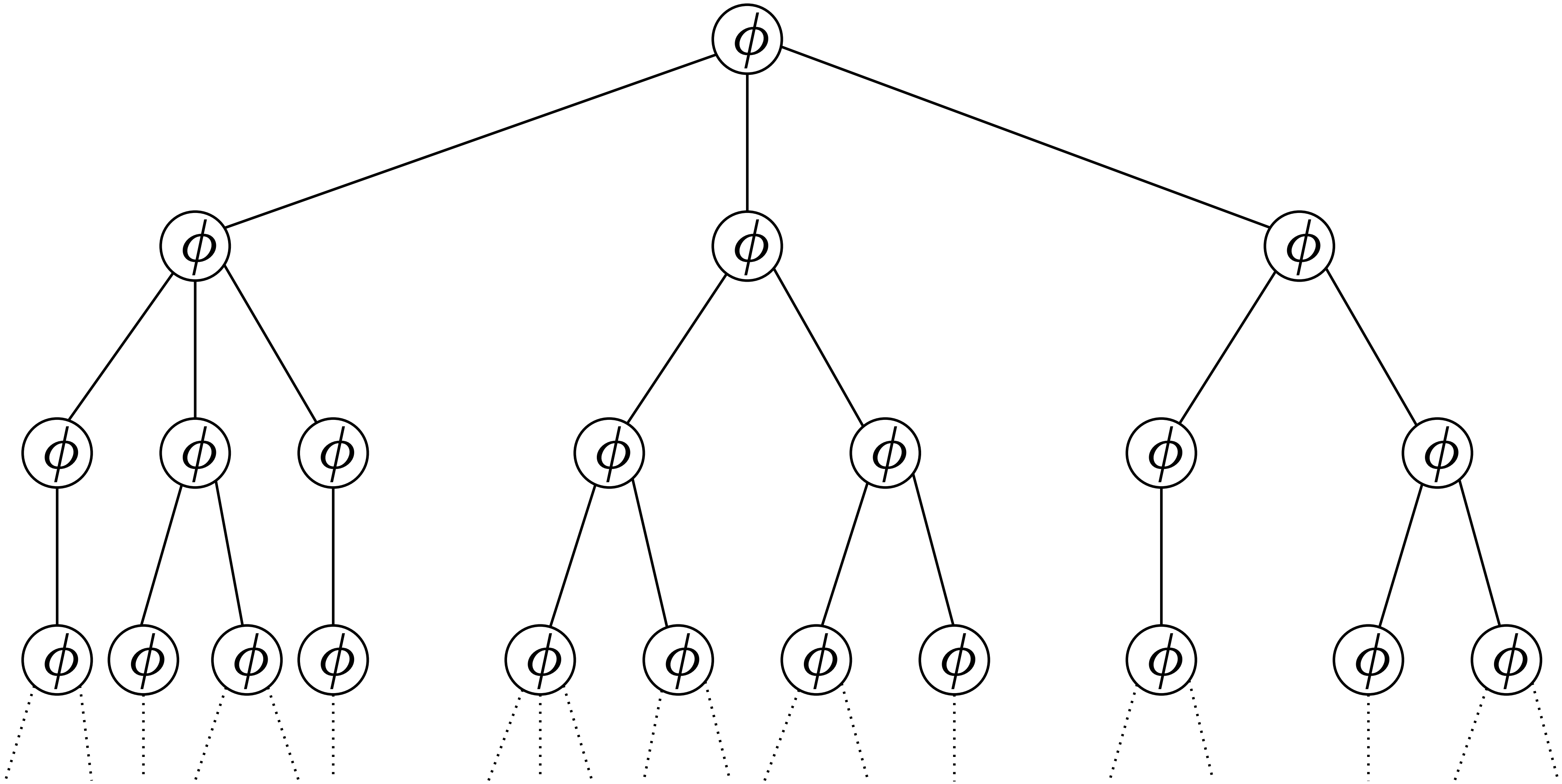


$$AX\phi \equiv \neg EX\neg\phi$$

Get rid of $AG\phi$

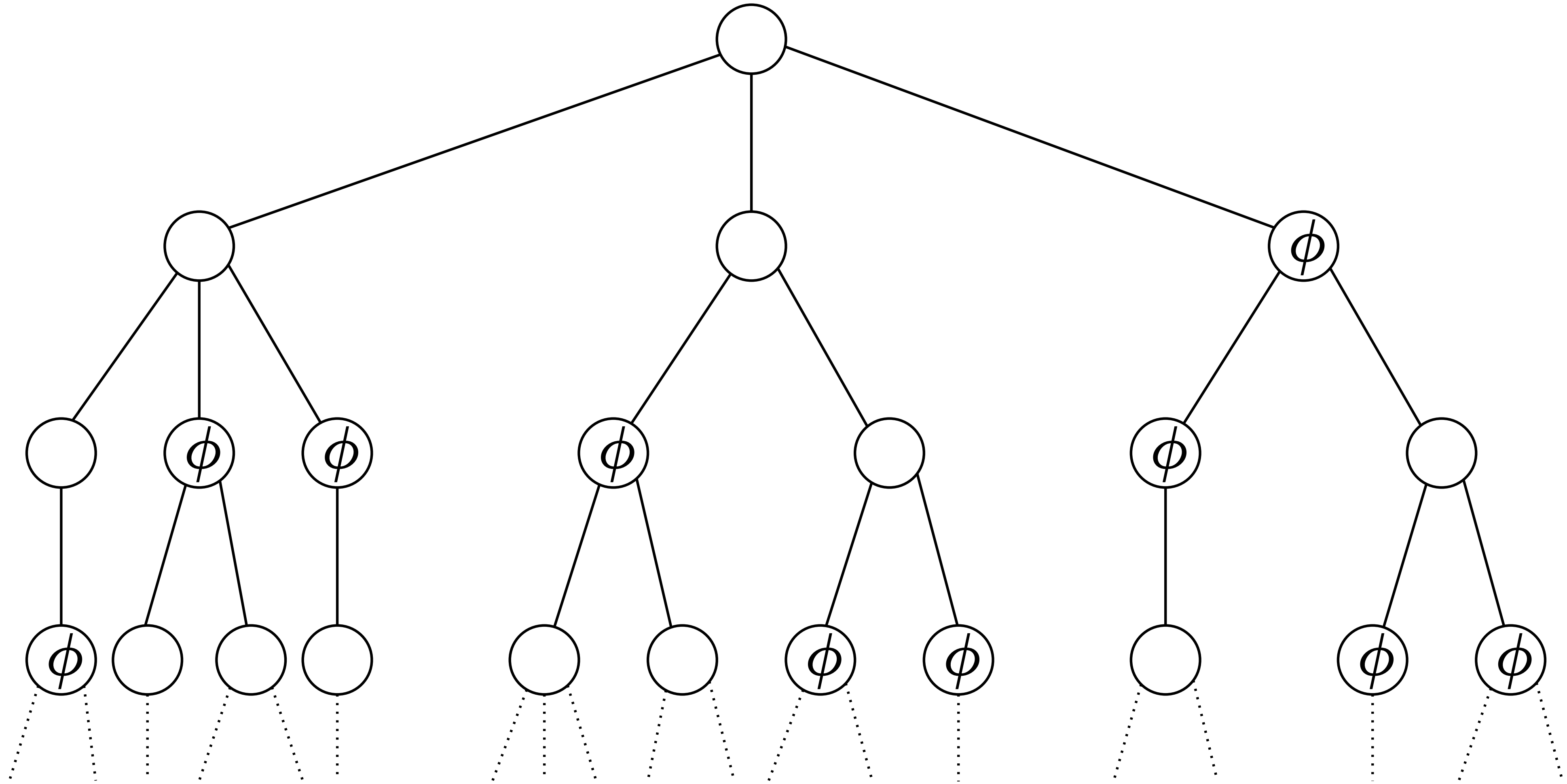


Get rid of $AG\phi$

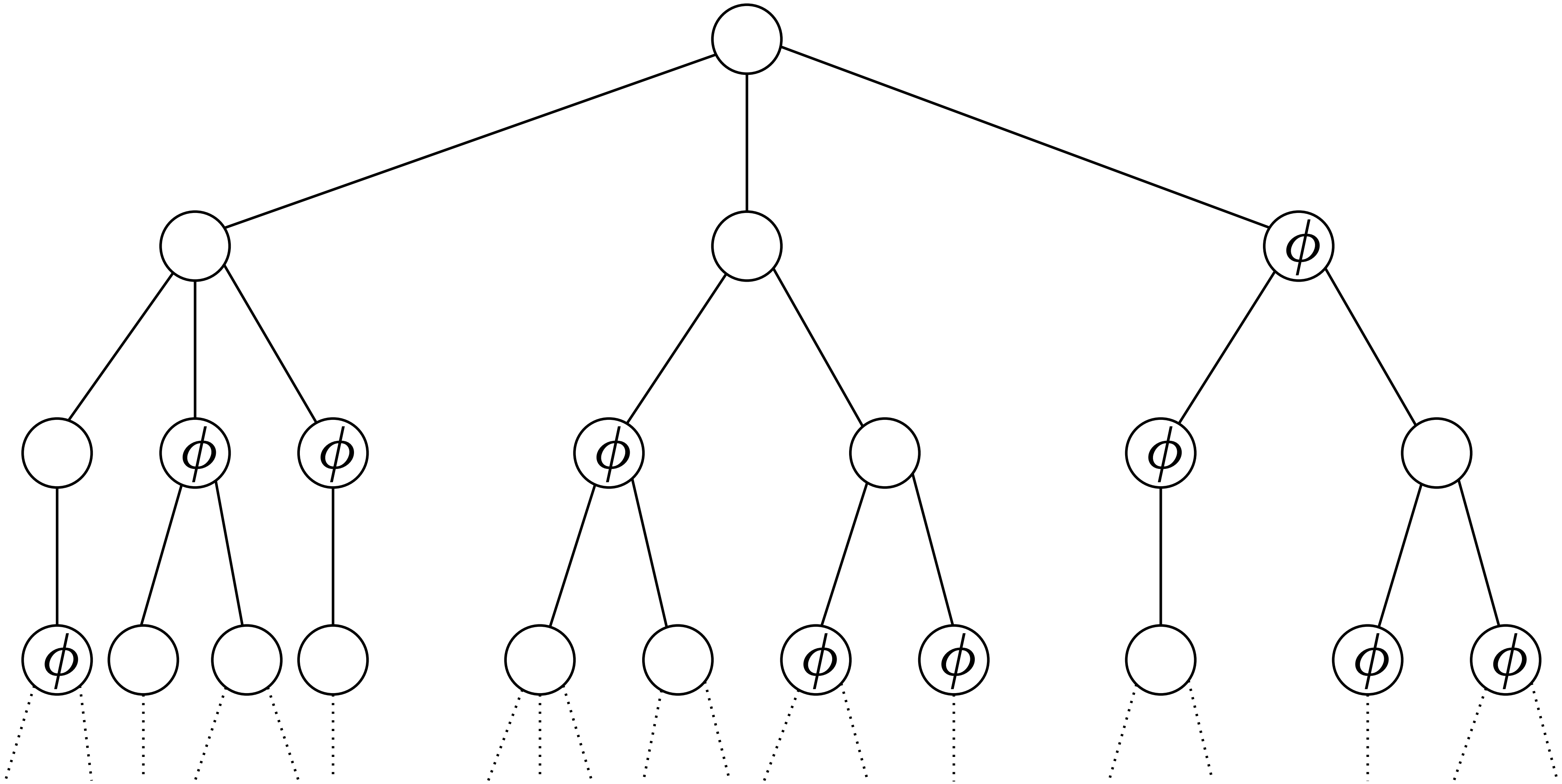


$$AG\phi \equiv \neg EF \neg \phi$$

Get rid of $AF\phi$



Get rid of $AF\phi$



$$AF\phi \equiv \neg EG\neg\phi$$

How to deal with $\phi\ AU \psi$?

$\phi\ AU\ \psi$ means that for all paths $\phi\ U\ \psi$

How to deal with $\phi\ AU\psi$?

$\phi\ AU\ \psi$ means that for all paths $\phi\ U\ \psi$
$\phi\ AU\ \psi$ means that there is no path where $\neg(\phi\ U\ \psi)$

$$\phi AU\psi \equiv \neg(\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad)$$

How to deal with $\phi\ AU\psi$?

$\phi\ AU\ \psi$ means that for all paths $\phi\ U\ \psi$

$\phi\ AU\ \psi$ means that there is no path where $\neg(\phi\ U\ \psi)$

What kinds of paths satisfy $\neg(\phi\ U\ \psi)$?

$$\phi AU\psi \equiv \neg(\qquad\qquad\qquad\qquad\qquad\qquad)$$

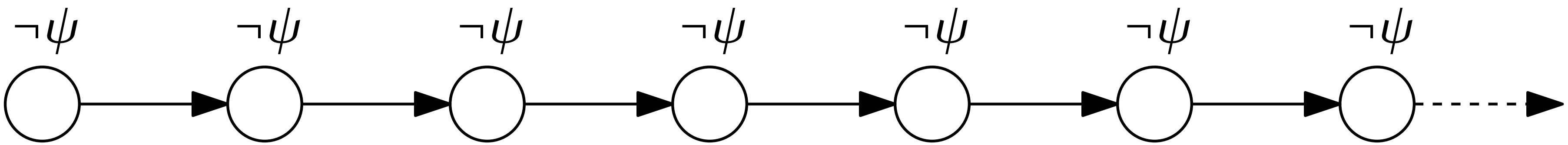
How to deal with $\phi\ AU\psi$?

$\phi\ AU\ \psi$ means that for all paths $\phi\ U\ \psi$

$\phi\ AU\ \psi$ means that there is no path where $\neg(\phi\ U\ \psi)$

What kinds of paths satisfy $\neg(\phi\ U\ \psi)$?

Either ψ never holds:



$$\phi AU\psi \equiv \neg(\ EG\neg\psi \qquad\qquad\qquad\qquad\qquad\qquad)$$

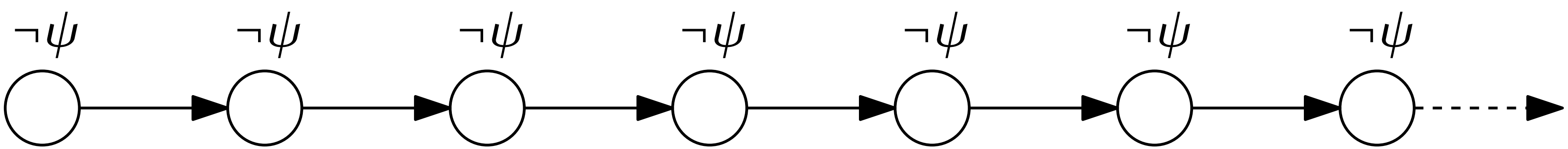
How to deal with $\phi\ AU\psi$?

$\phi\ AU\ \psi$ means that for all paths $\phi\ U\ \psi$

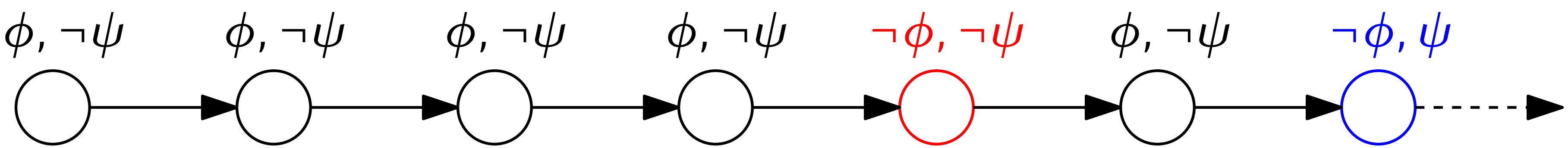
$\phi\ AU\ \psi$ means that there is no path where $\neg(\phi\ U\ \psi)$

What kinds of paths satisfy $\neg(\phi\ U\ \psi)$?

Either ψ never holds:



Or ϕ stops holding sometime before ψ holds



$$\phi AU\psi \equiv \neg(\ EG\neg\psi \qquad\qquad\qquad\qquad)$$

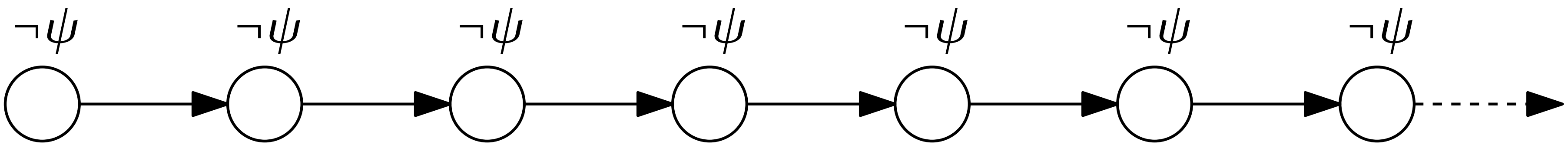
How to deal with $\phi\ AU\psi$?

$\phi\ AU\ \psi$ means that for all paths $\phi\ U\ \psi$

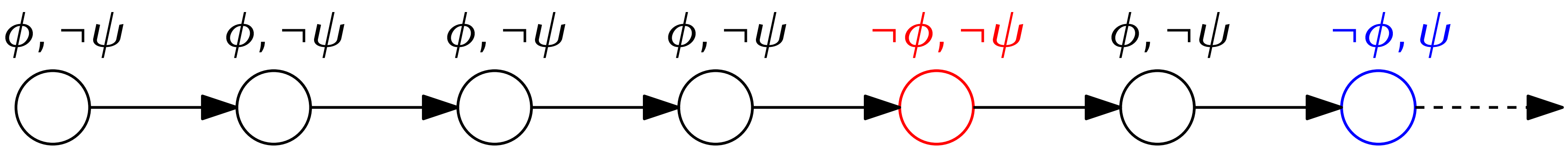
$\phi\ AU\ \psi$ means that there is no path where $\neg(\phi\ U\ \psi)$

What kinds of paths satisfy $\neg(\phi\ U\ \psi)$?

Either ψ never holds:



Or ϕ stops holding sometime before ψ holds



$$\phi AU\psi \equiv \neg(\ EG\neg\psi \quad \vee \quad \neg\psi\ EU\ (\neg\phi \wedge \neg\psi))$$

Summarizing:

$$AX\phi \equiv \neg EX \neg \phi$$

$$AG\phi \equiv \neg EF \neg \phi$$

$$AF\phi \equiv \neg EG \neg \phi$$

$$\phi AU \psi \equiv \neg(EG \neg \phi \quad \vee \quad \neg \phi \; EU \; (\neg \phi \wedge \neg \psi))$$

Summarizing:

$$
\begin{aligned}
AX\phi &\equiv \neg EX\neg\phi \\
AG\phi &\equiv \neg EF\neg\phi \\
AF\phi &\equiv \neg EG\neg\phi \\
\phi AU\psi &\equiv \neg(\, EG\neg\phi \quad \vee \quad \neg\phi\ EU\ (\neg\phi \wedge \neg\psi))
\end{aligned}
$$

All of the A-properties can be written in terms of the E-properties and Boolean connectives.

Summarizing:

$$\boxed{\begin{aligned} AX\phi \\ AG\phi \\ AF\phi \\ \phi AU\psi \end{aligned}} \equiv \boxed{\begin{aligned} &\neg EX\neg\phi \\ &\neg EF\neg\phi \\ &\neg EG\neg\phi \\ &\neg(EG\neg\phi \quad \vee \quad \neg\phi\ EU\ (\neg\phi \wedge \neg\psi)) \end{aligned}}$$

All of the *A*-properties can be written in terms of the *E*-properties and Boolean connectives.

This is called **existential negation normal form** for CTL.

Summarizing:

$$\boxed{\begin{array}{r} AX\phi \\ AG\phi \\ AF\phi \\ \phi AU\psi \end{array}} \begin{array}{c} \equiv \\ \equiv \\ \equiv \\ \equiv \end{array} \boxed{\begin{array}{l} \neg EX\neg\phi \\ \neg EF\neg\phi \\ \neg EG\neg\phi \\ \neg(EG\neg\phi \quad \vee \quad \neg\phi\ EU\ (\neg\phi \wedge \neg\psi)) \end{array}}$$

All of the A-properties can be written in terms of the E-properties and Boolean connectives.

This is called **existential negation normal form** for CTL.

Furthermore, $EF\phi \equiv \top\ EU\phi$

We only need EX, EU, EG

CTL model checking algorithm

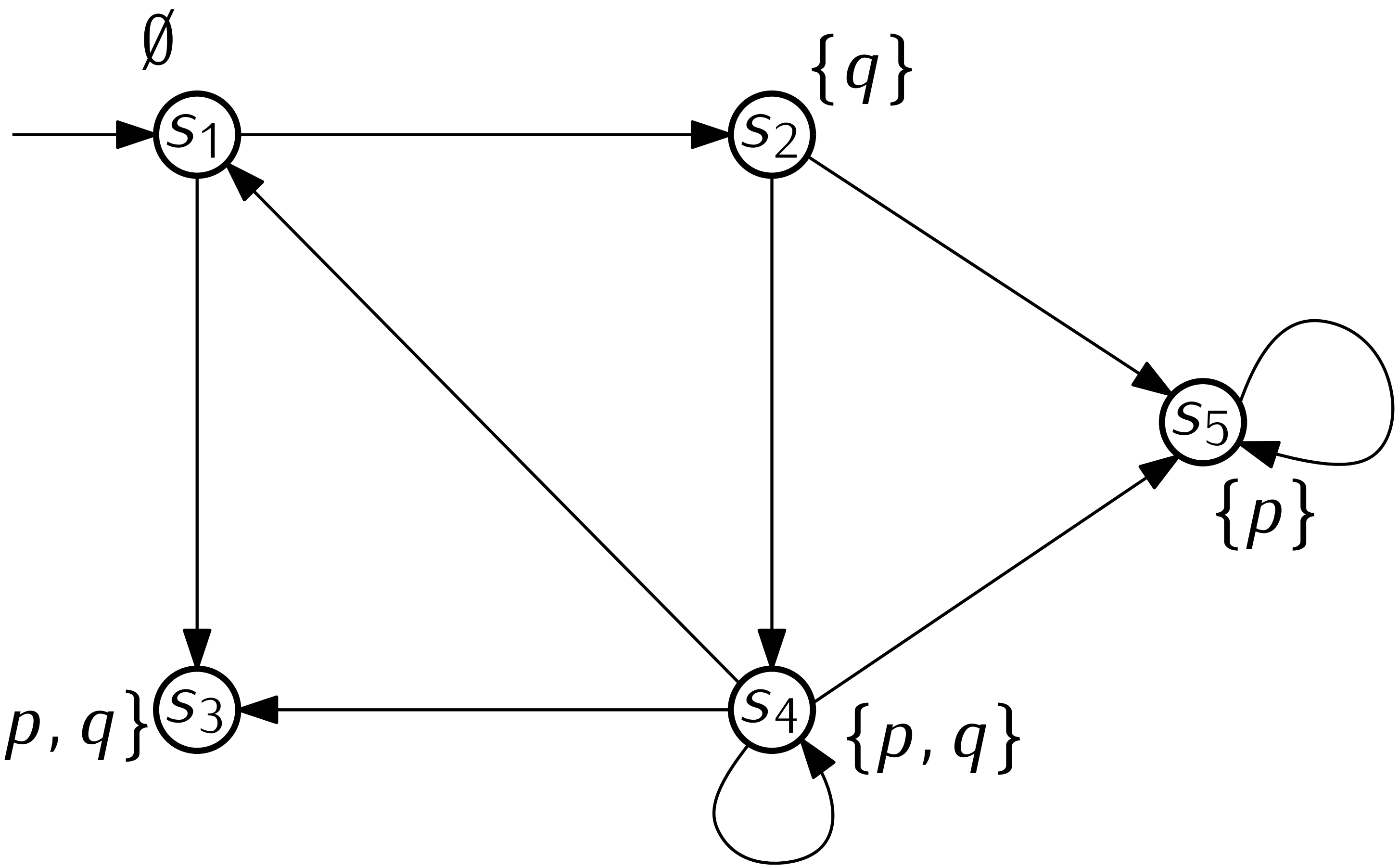
The main idea

First convert everything into existential negation normal form using previous reductions, so that we have only formulas with EX, EG, EU.

For each of the operators EX, EG, EU, give a method to determine the correposind set of states that satisfy the property.

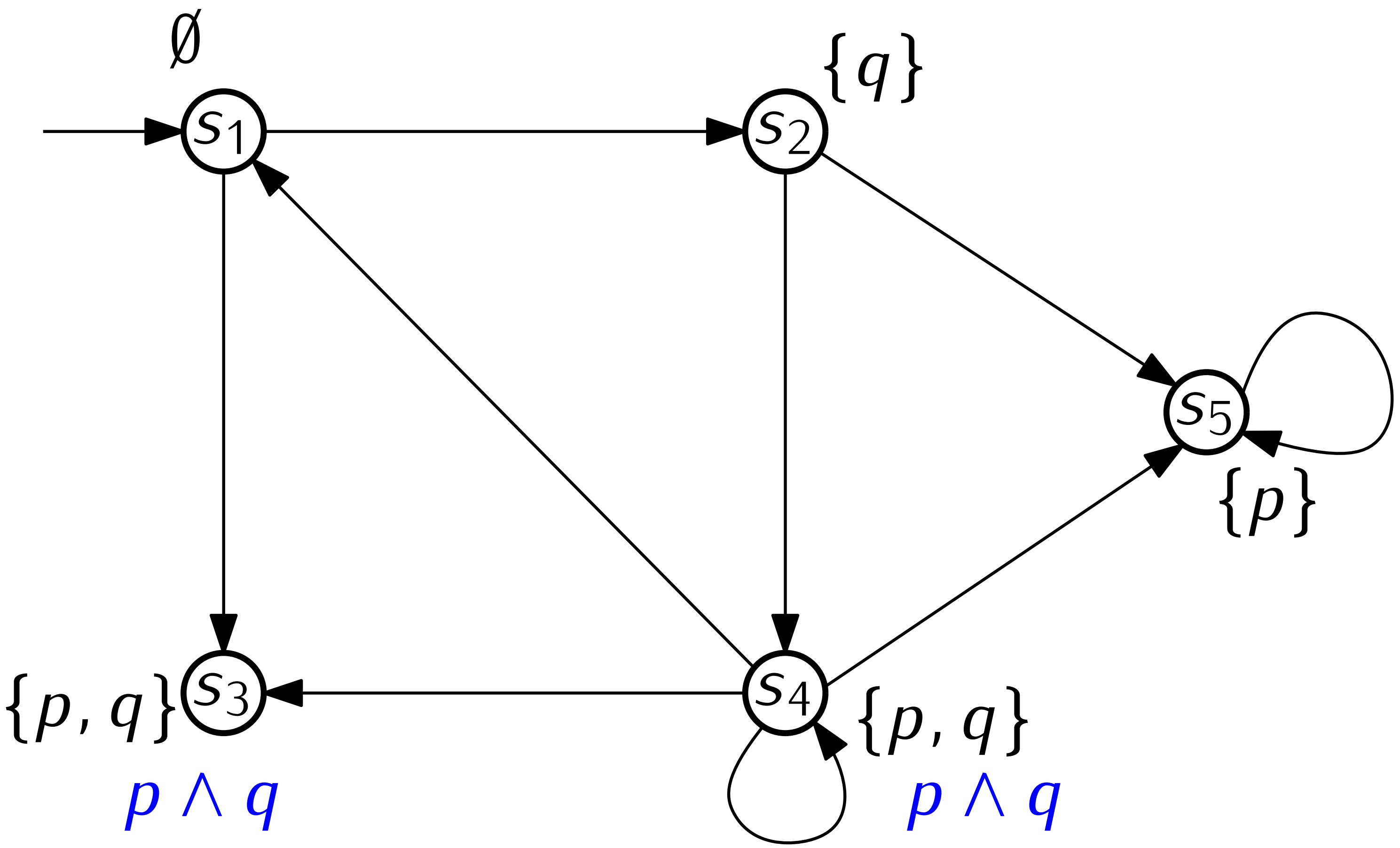
The Algorithm for $EX\ \phi$

First, an example $\quad EX(p \wedge q)$



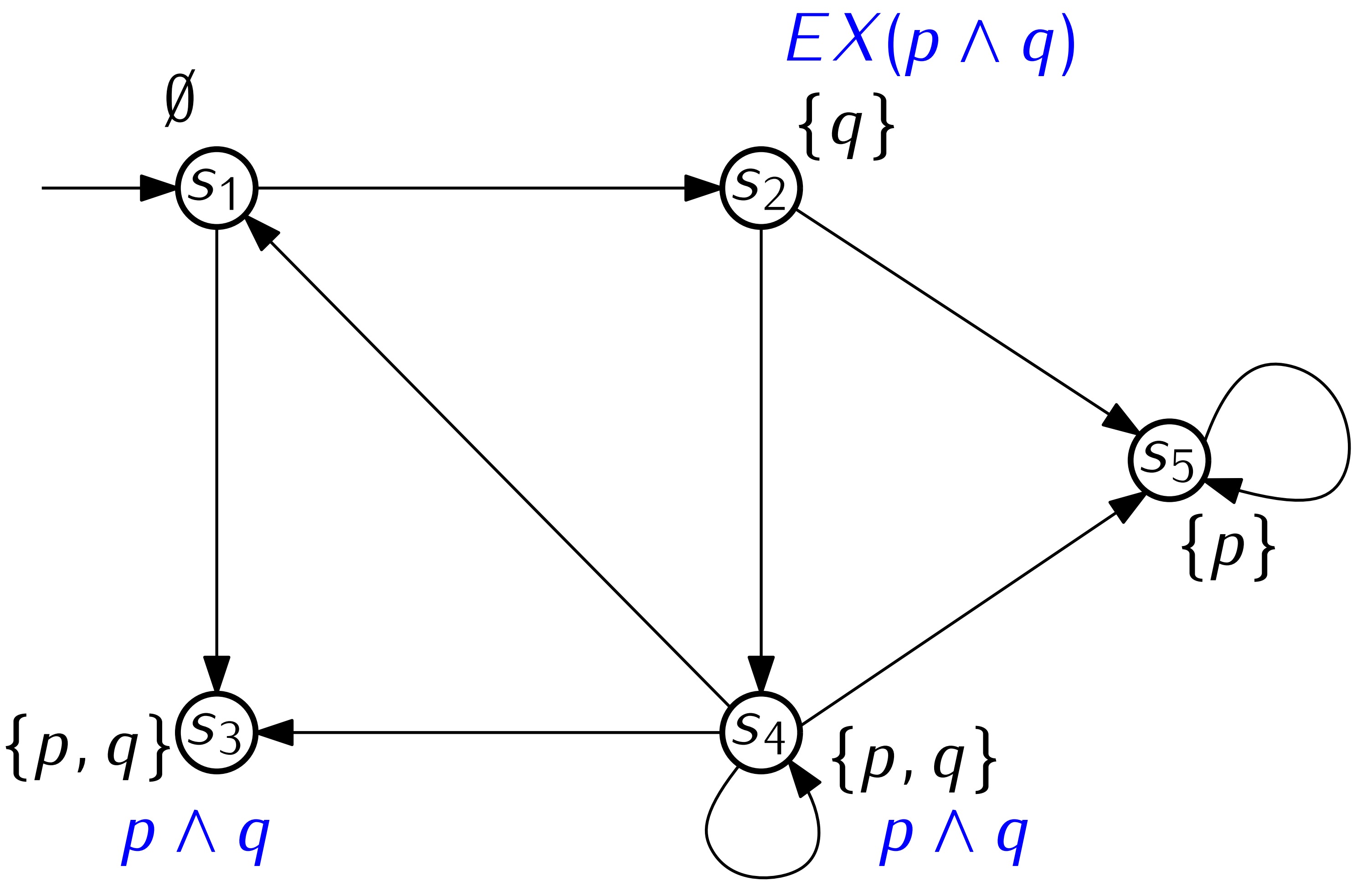
The Algorithm for $EX\ \phi$

First, an example $EX(p \wedge q)$



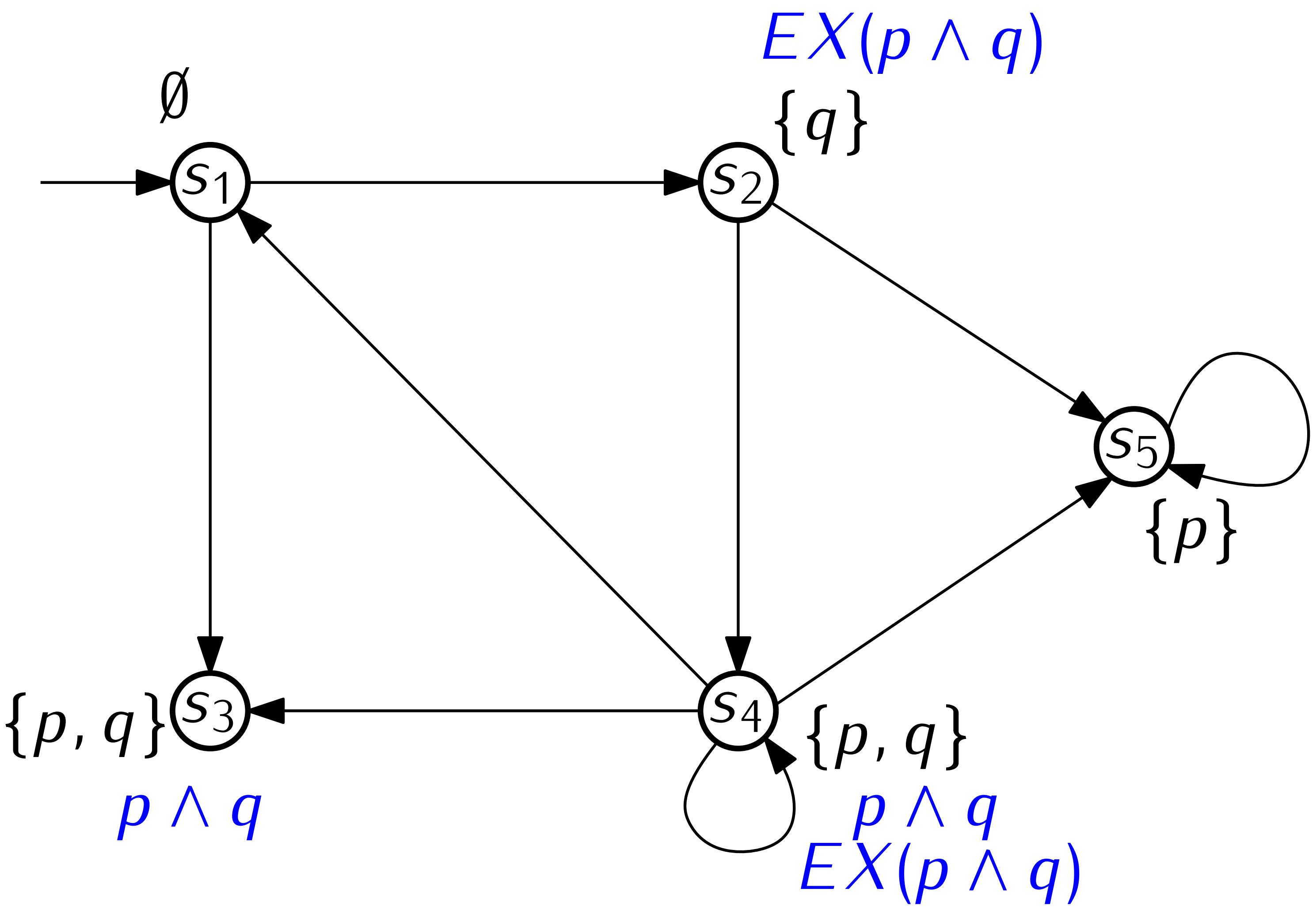
The Algorithm for $EX\ \phi$

First, an example $EX(p \wedge q)$



The Algorithm for $EX\ \phi$

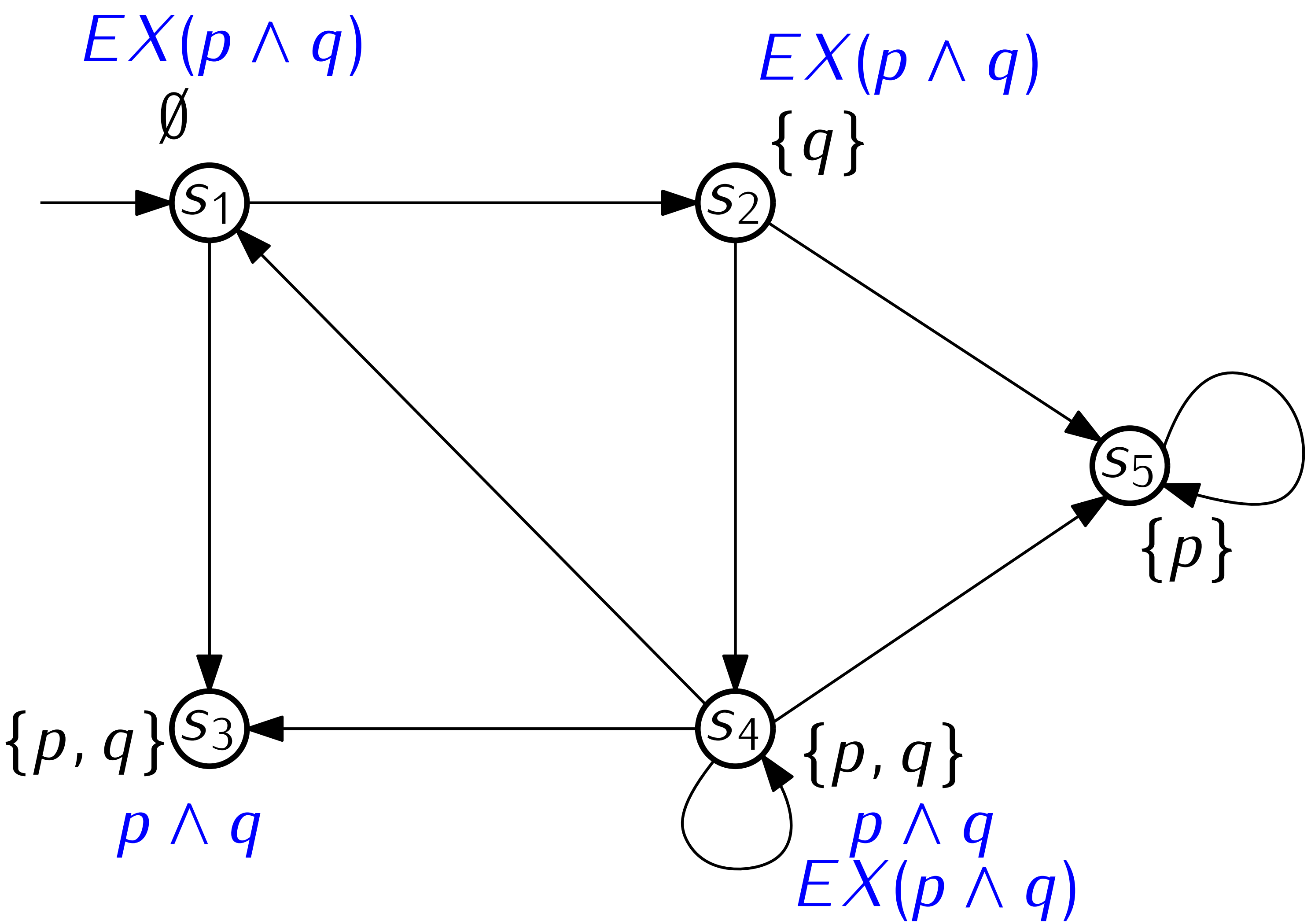
First, an example $\qquad EX(p \wedge q)$



The Algorithm for $EX\ \phi$

First, an example

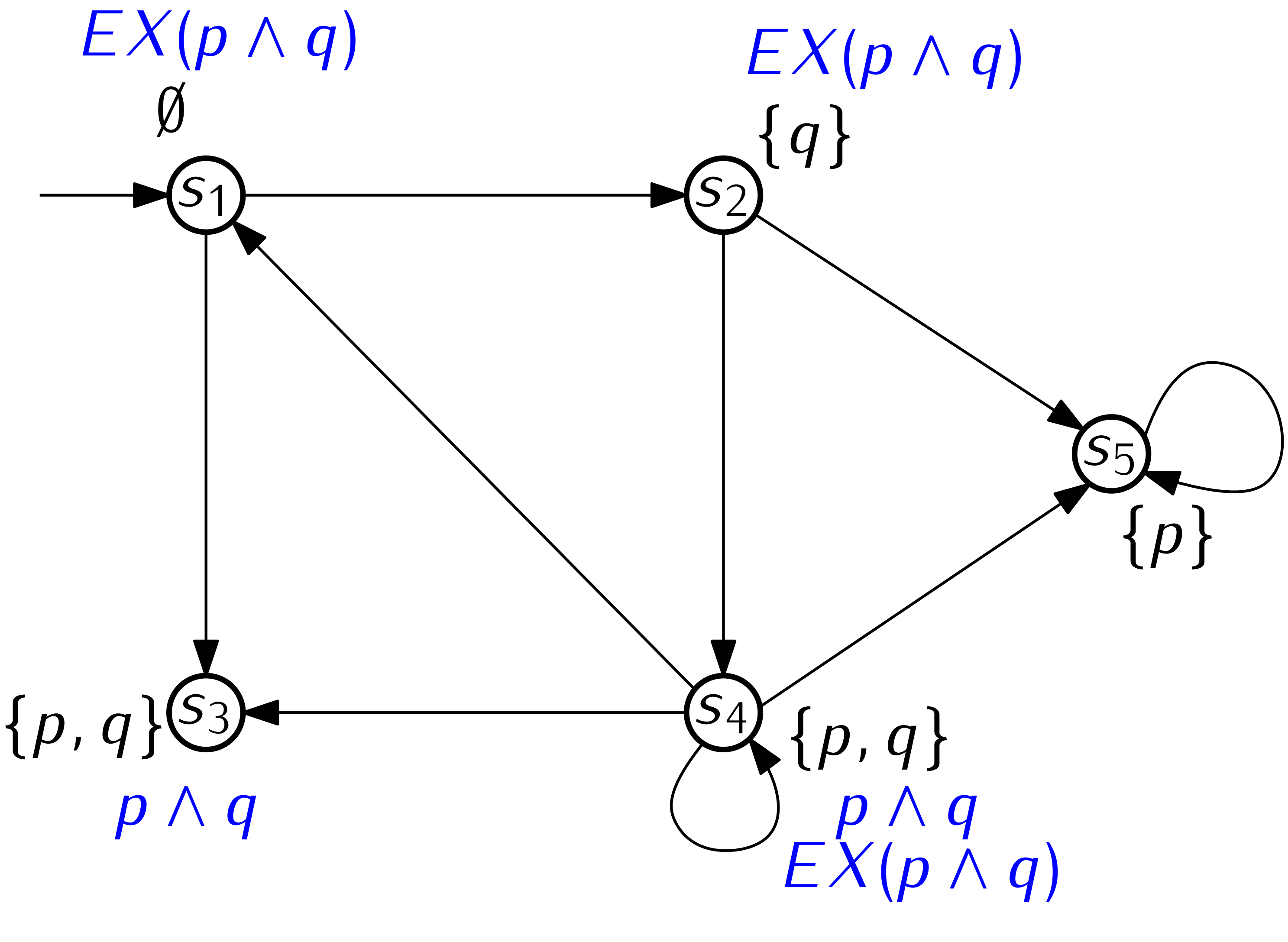
$EX(p \wedge q)$



The Algorithm for $EX\ \phi$

First, an example

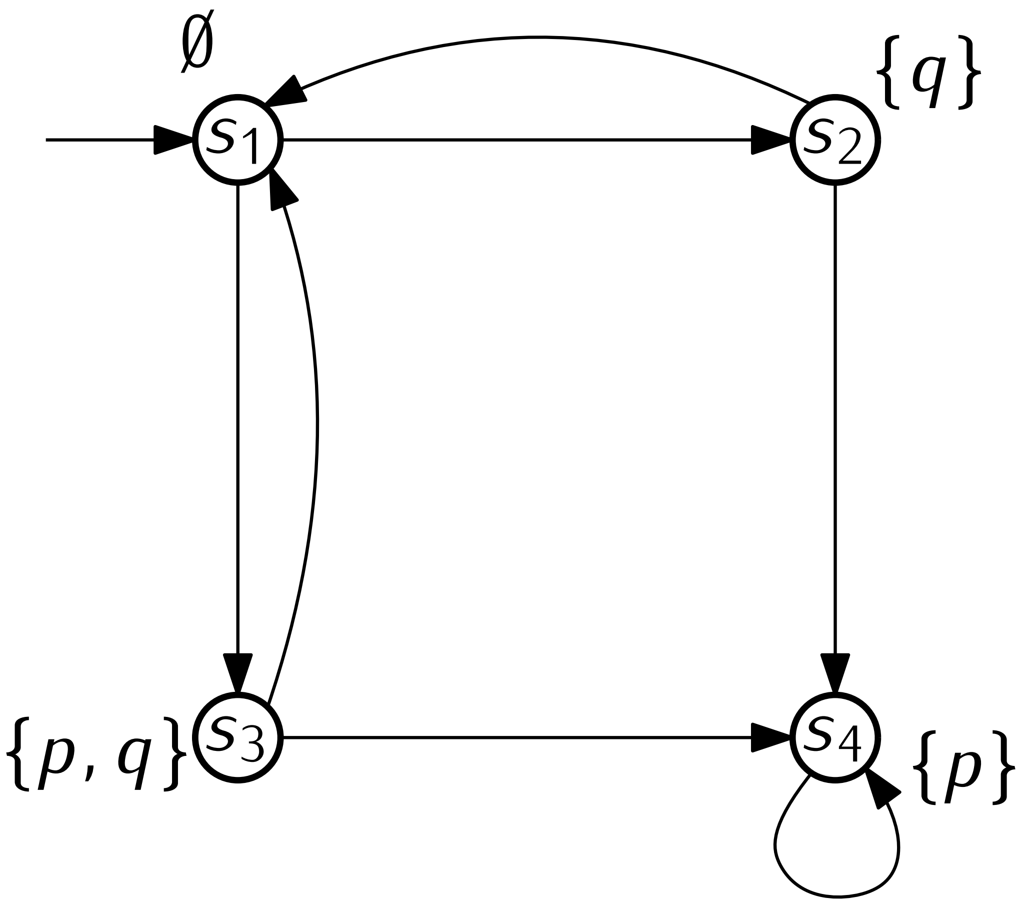
$$EX(p \wedge q)$$



$$s_1, s_2, s_4 \models EX(p \wedge q)$$

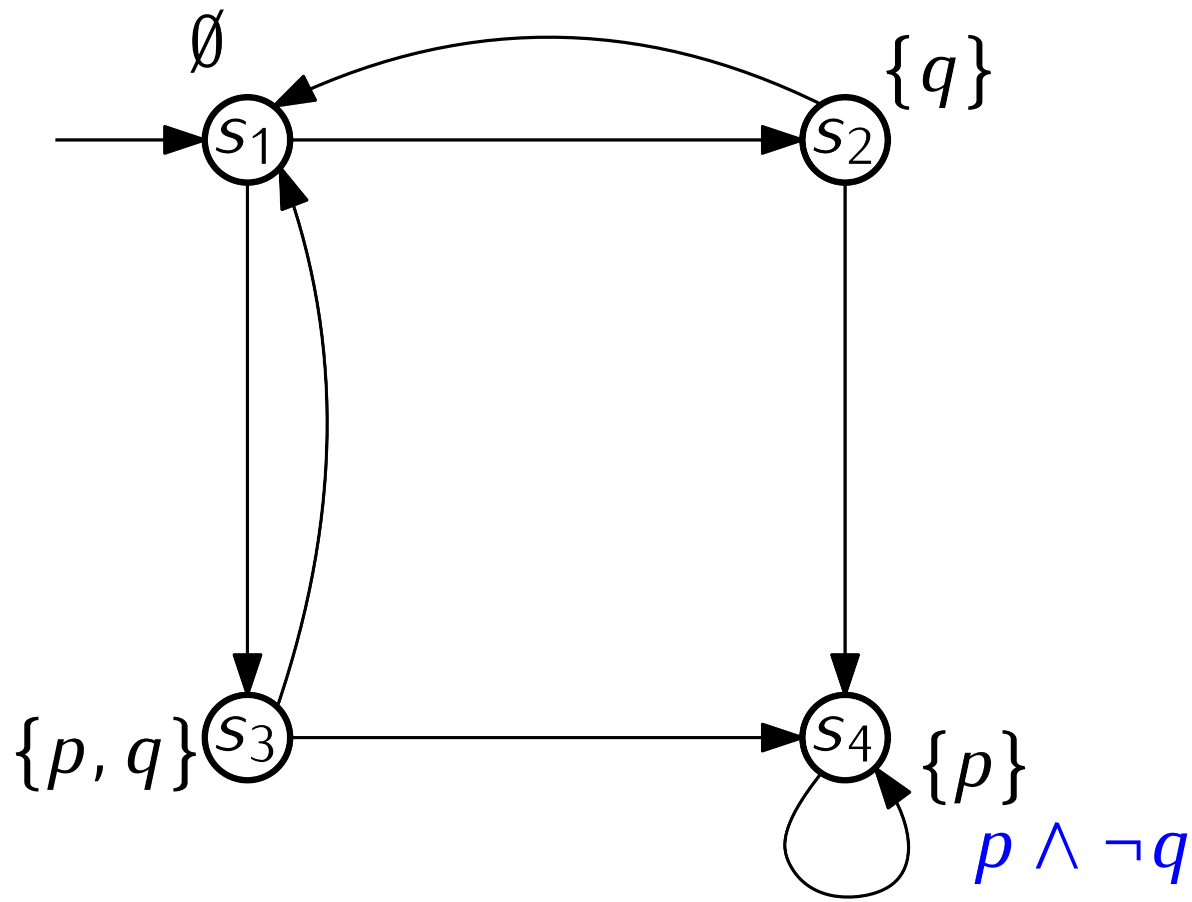
The Algorithm for $EX\ \phi$

Another example $\qquad EX(p \wedge \neg q)$



The Algorithm for $EX\ \phi$

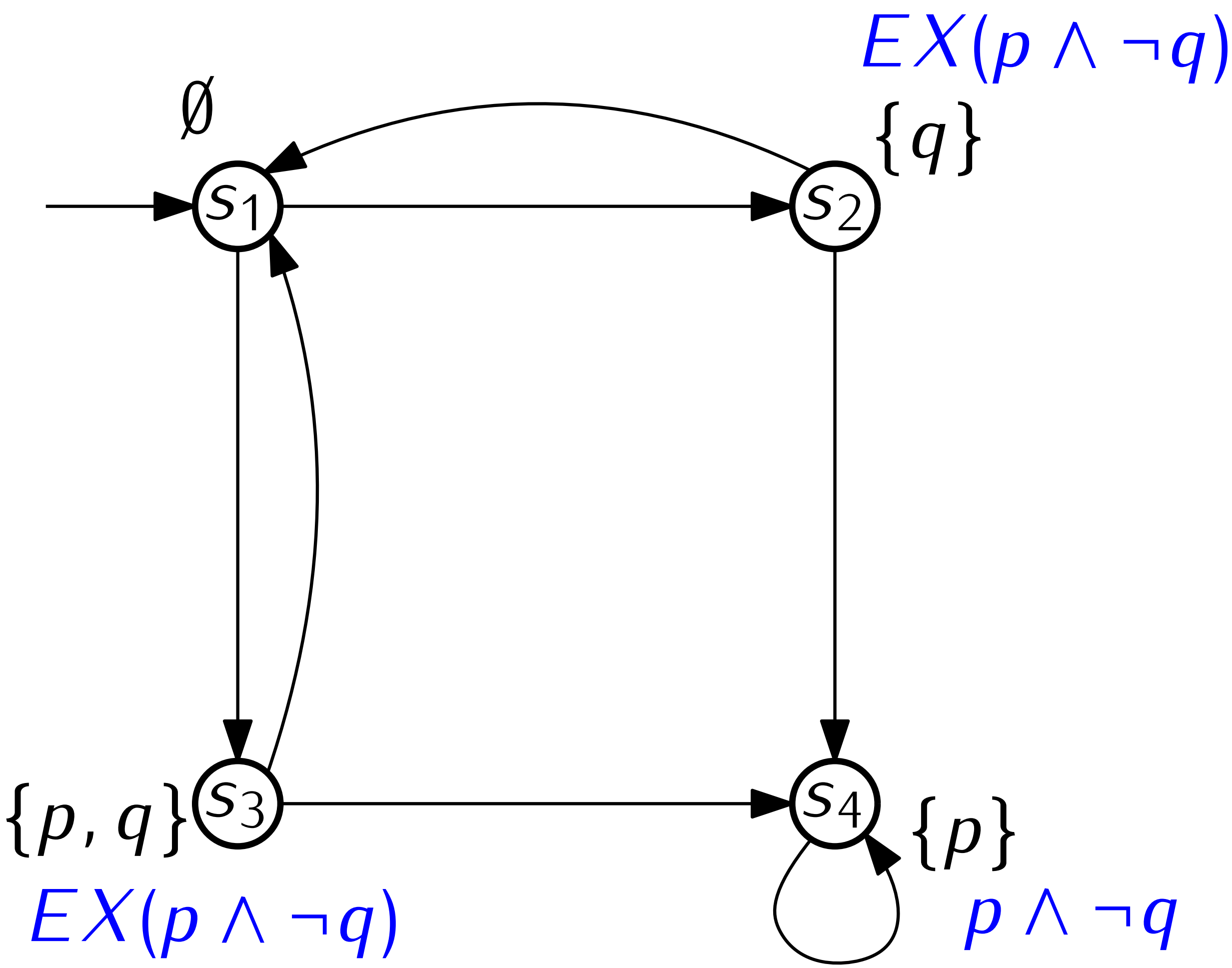
Another example $\qquad\qquad EX(p \wedge \neg q)$



The Algorithm for $EX\ \phi$

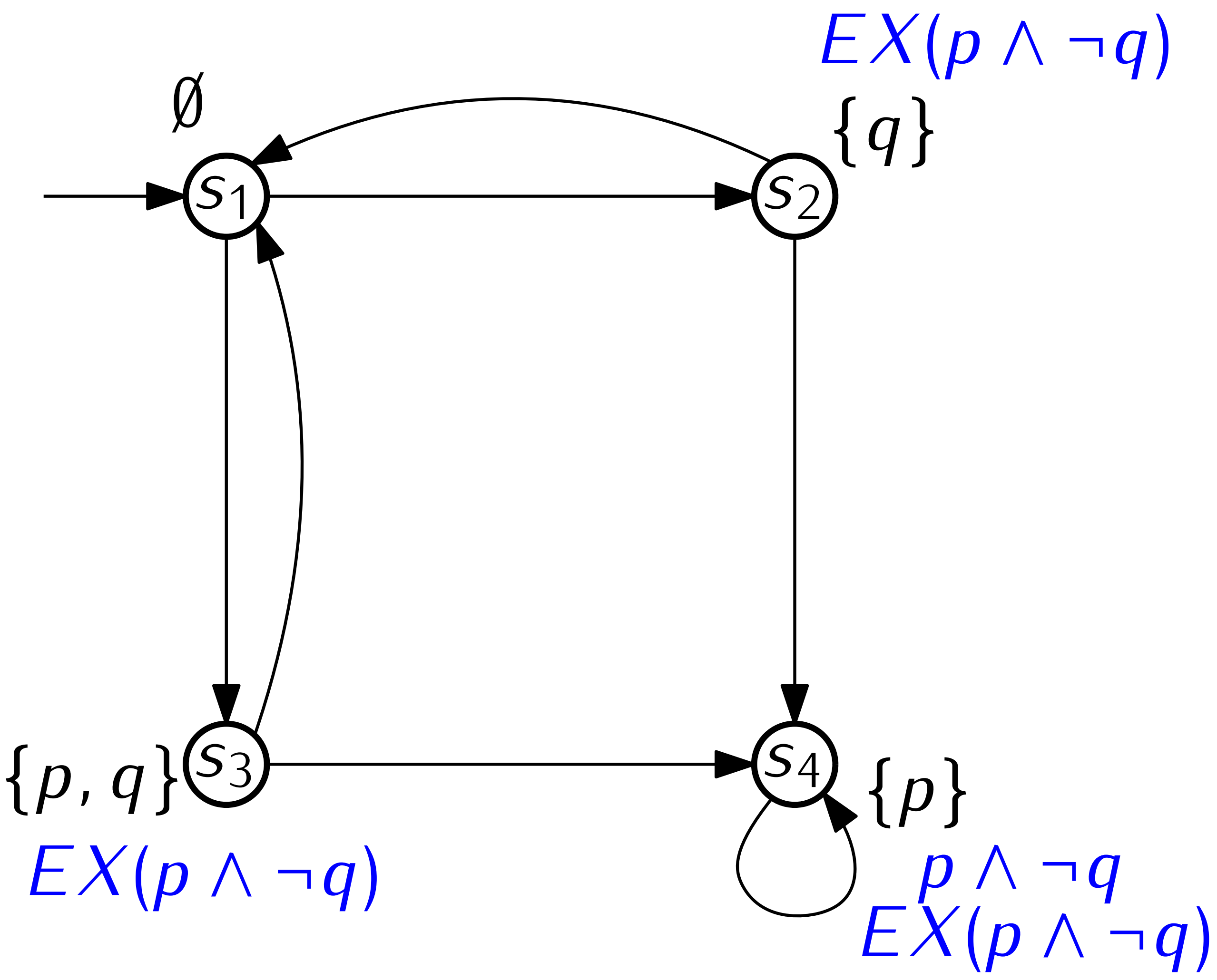
Another example

$$EX(p \wedge \neg q)$$



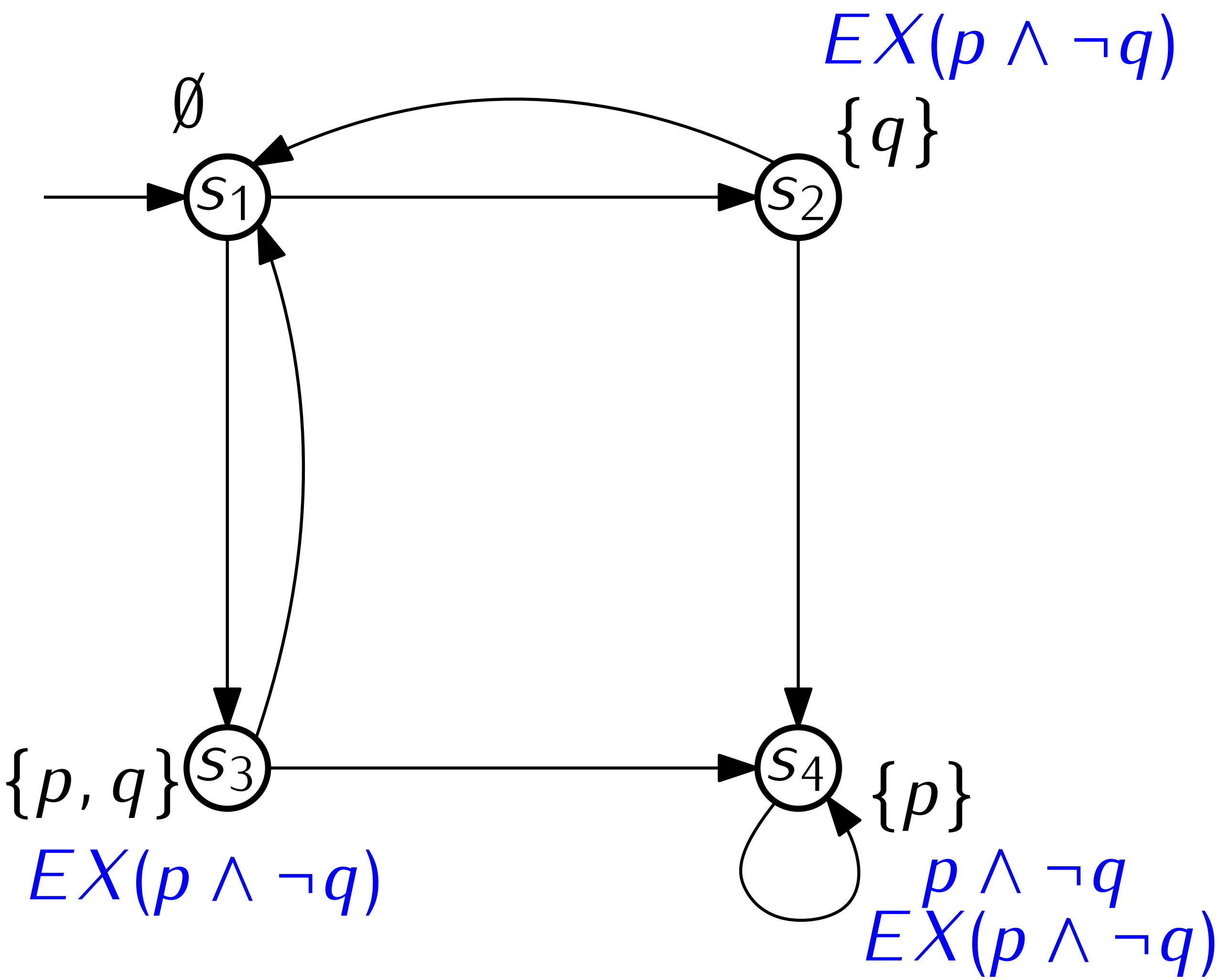
The Algorithm for $EX\ \phi$

Another example $EX(p \land \neg q)$



The Algorithm for $EX\ \phi$

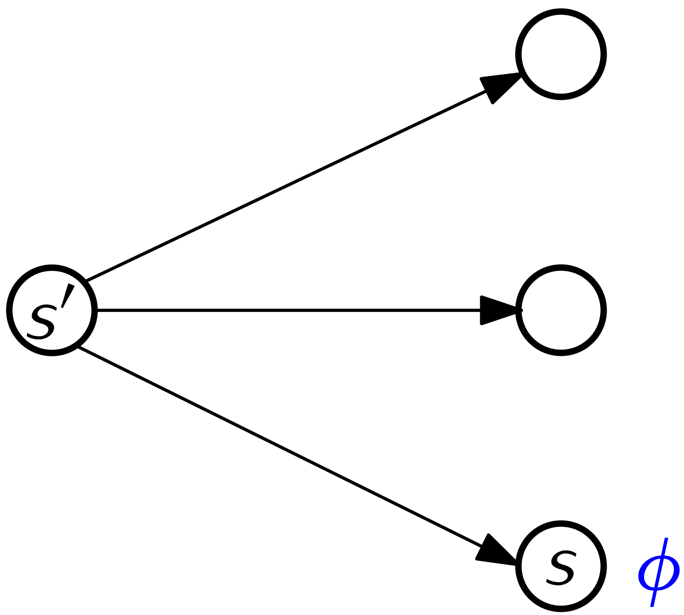
Another example $\qquad EX(p \wedge \neg q)$



$s_2, s_3, s_4 \models EX(p \wedge q)$

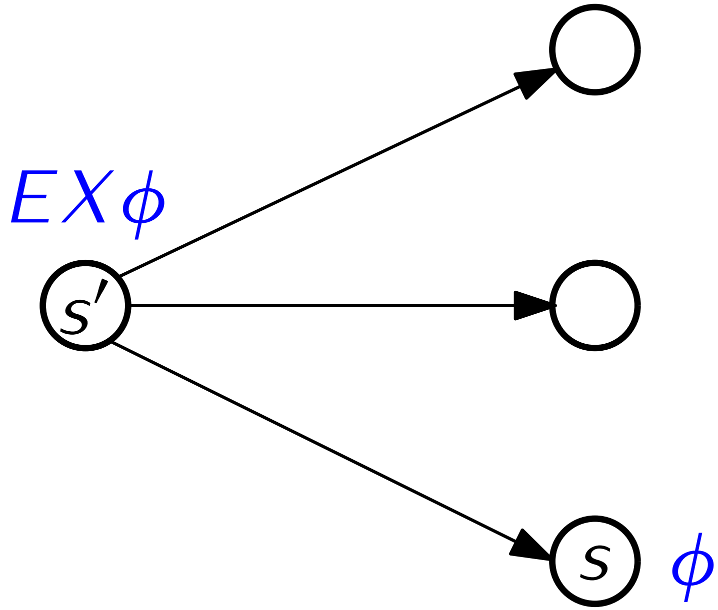
The Algorithm for $EX\ \phi$

After labelling all states s that satisfy ϕ, label and state s' with $EX\phi$ if there is a transition from s' to s.



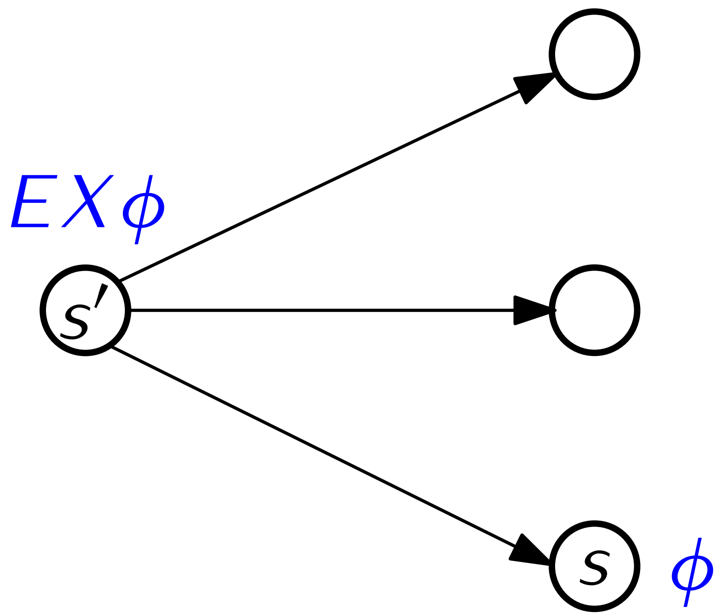
The Algorithm for $EX\ \phi$

After labelling all states s that satisfy ϕ, label and state s' with $EX\phi$ if there is a transition from s' to s.



The Algorithm for $EX\ \phi$

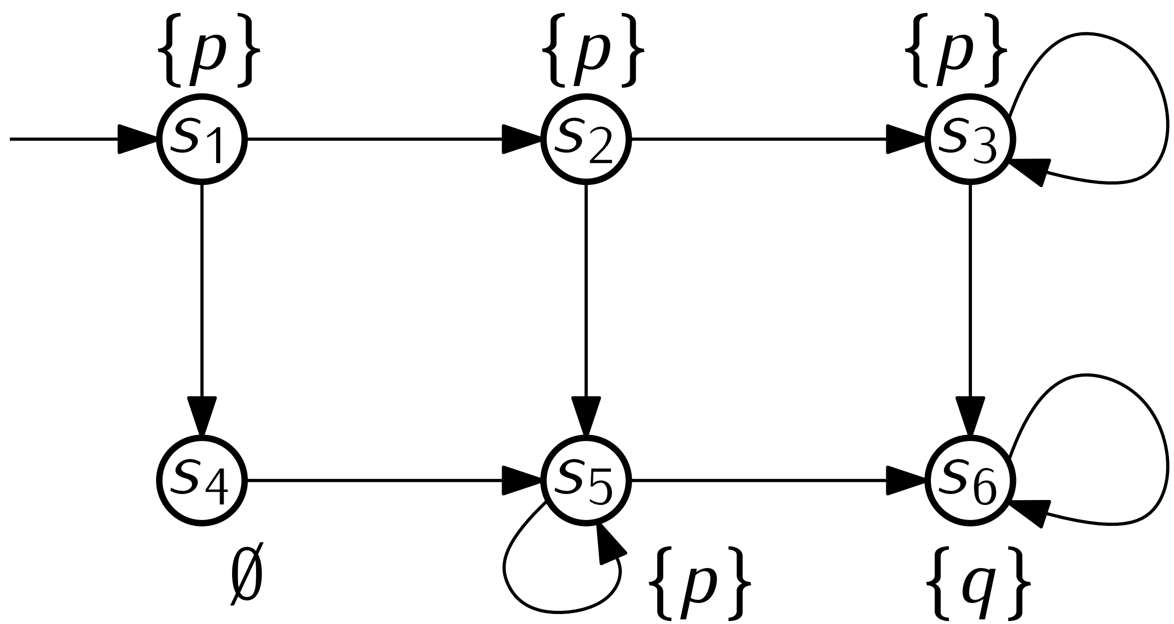
After labelling all states s that satisfy ϕ, label and state s' with $EX\phi$ if there is a transition from s' to s.



Call this process $\mathrm{SAT}_{EX}(\phi)$

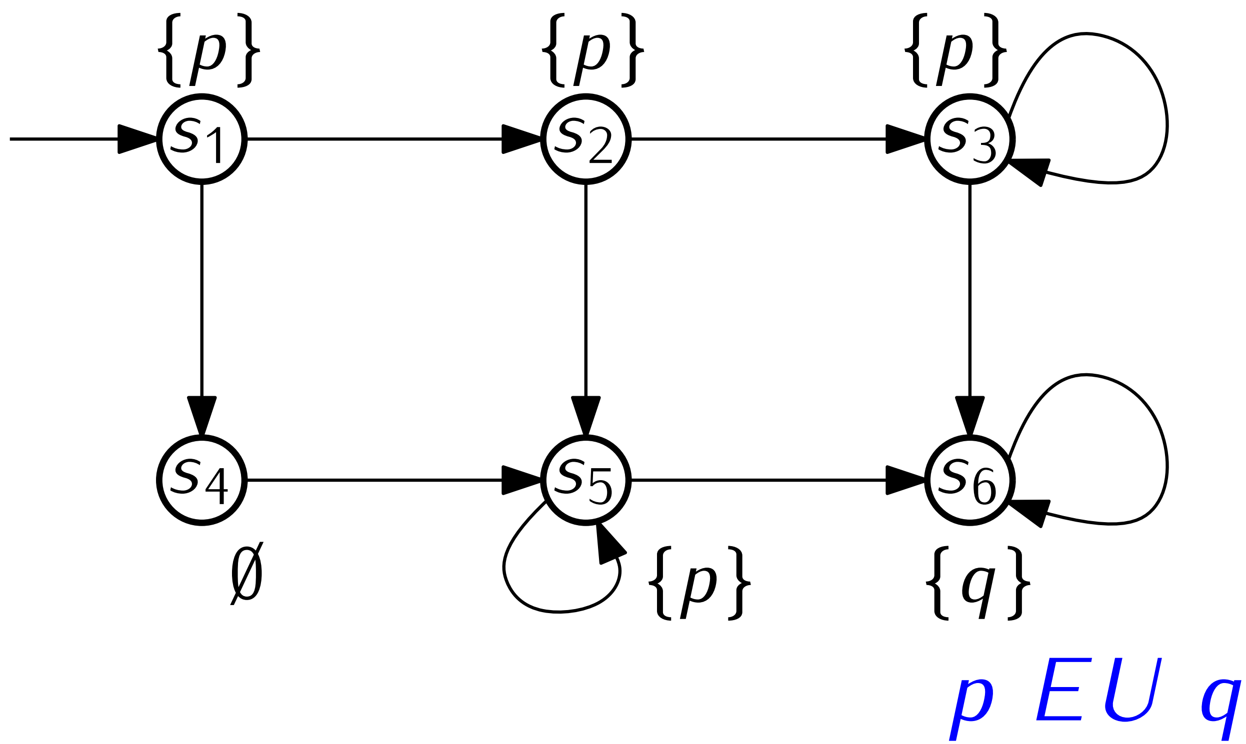
The Algorithm for $\phi\ EU\ \psi$

First, an example $\quad p\ EU\ q$



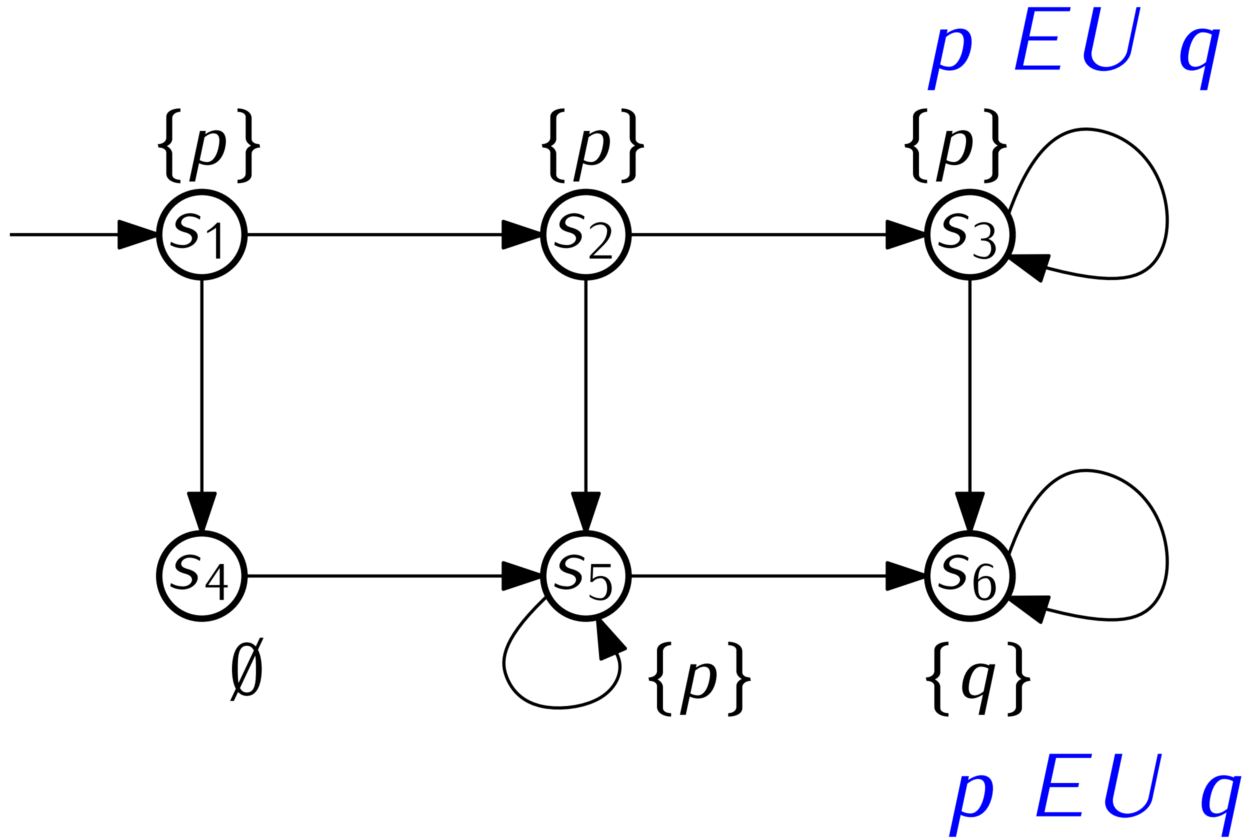
The Algorithm for $\phi\ EU\ \psi$

First, an example $\quad p\ EU\ q$



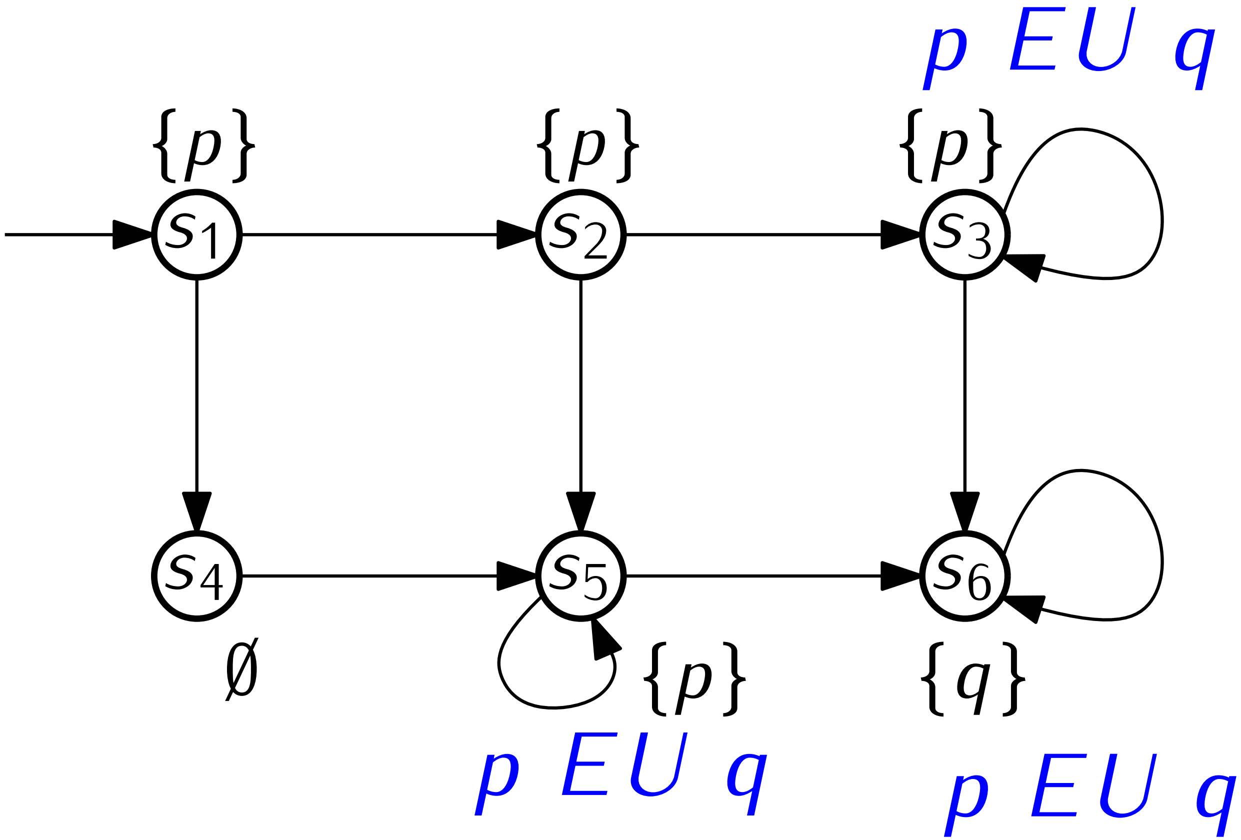
The Algorithm for $\phi \ EU \ \psi$

First, an example $\quad p \ EU \ q$



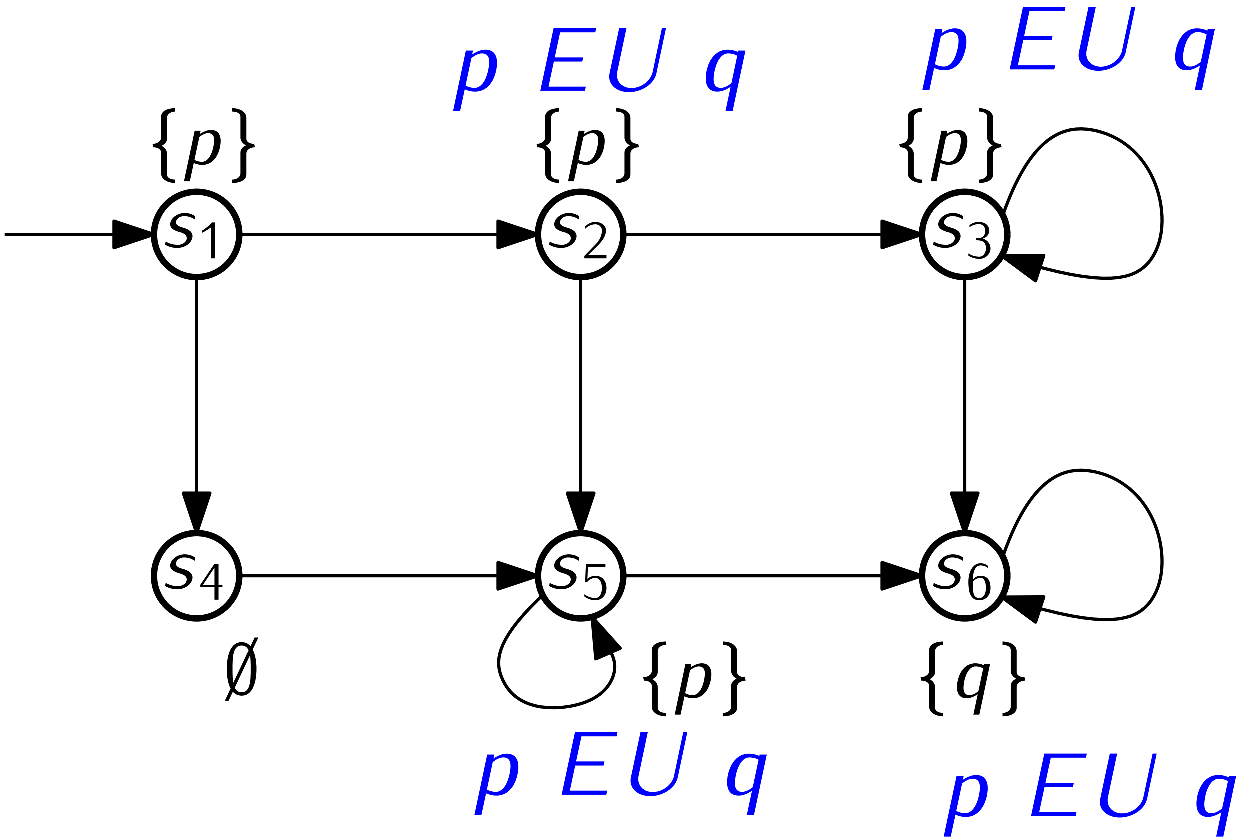
The Algorithm for $\phi\ EU\ \psi$

First, an example $\quad p\ EU\ q$



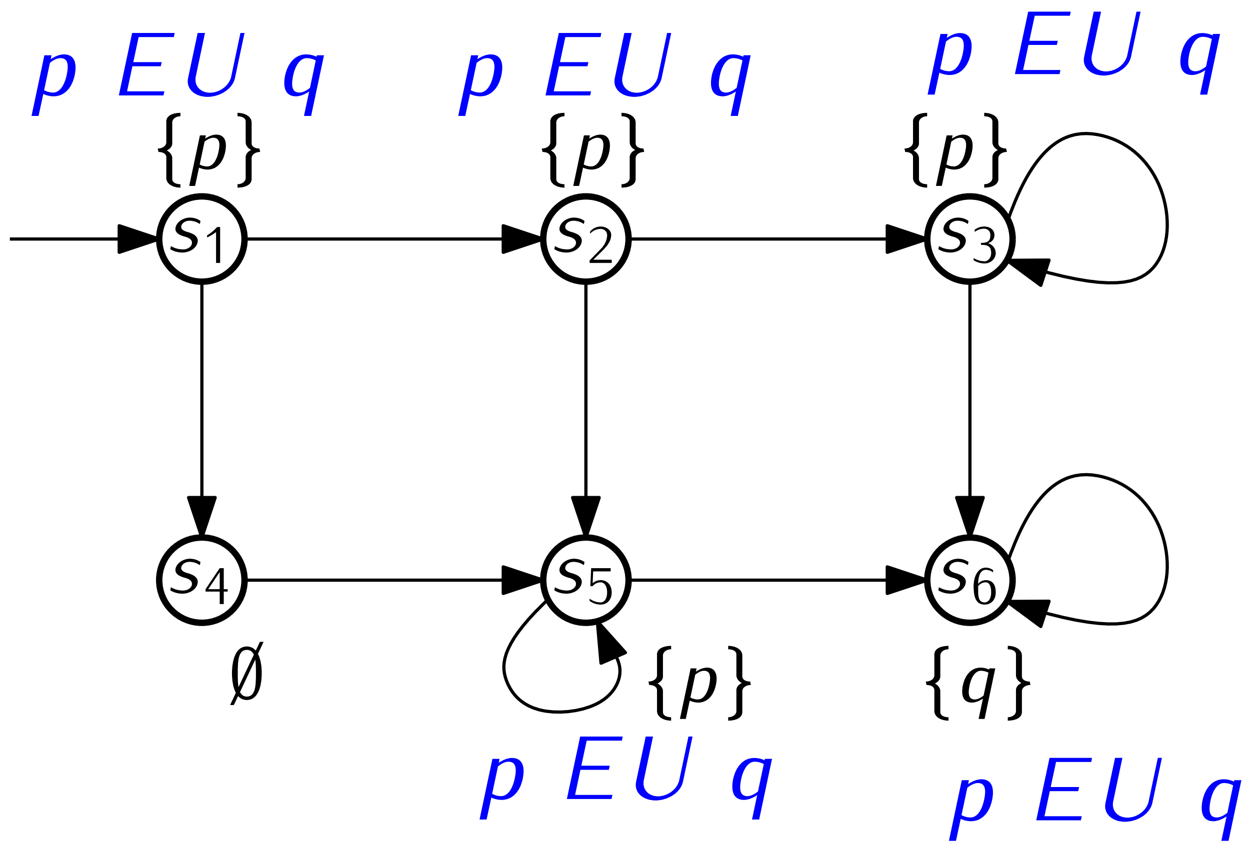
The Algorithm for $\phi \; EU \; \psi$

First, an example $\qquad p \; EU \; q$



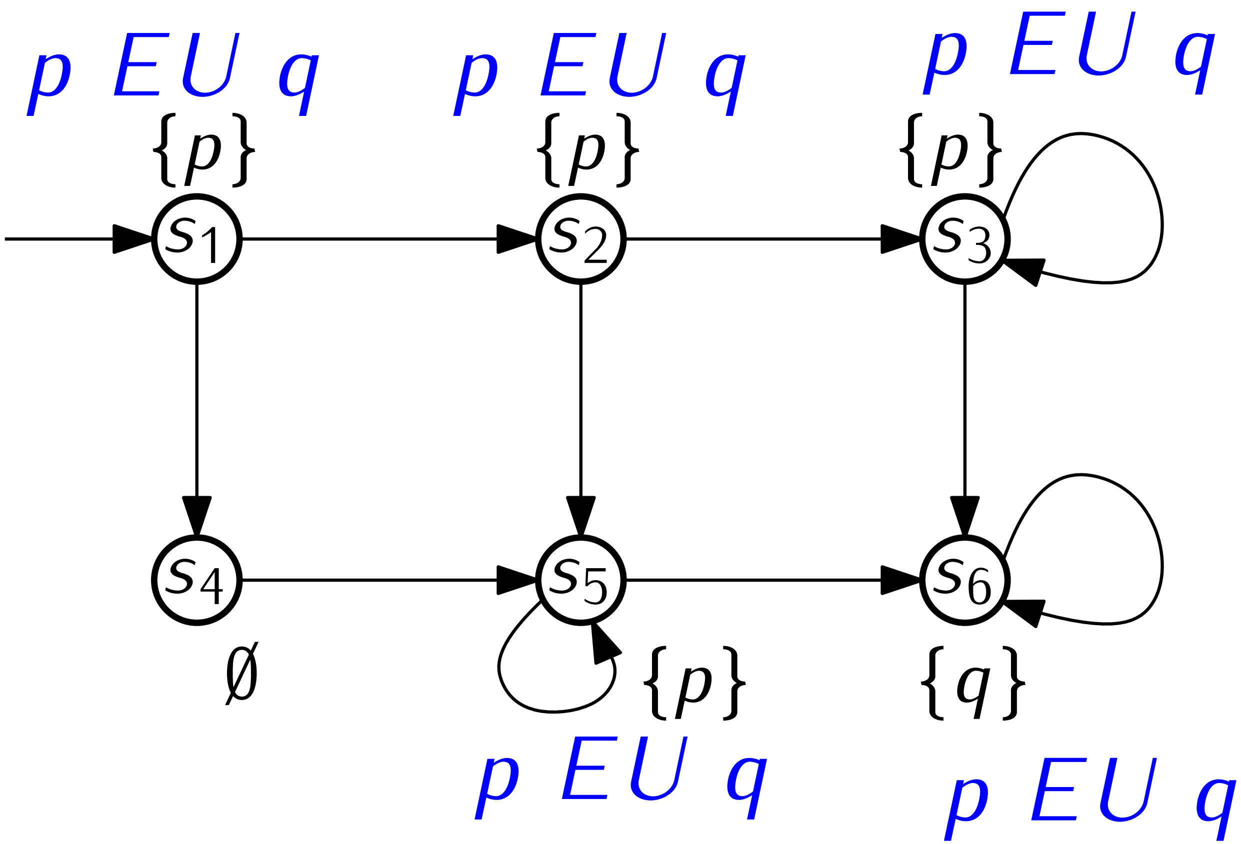
The Algorithm for $\phi\ EU\ \psi$

First, an example $p\ EU\ q$



The Algorithm for $\phi \; EU \; \psi$

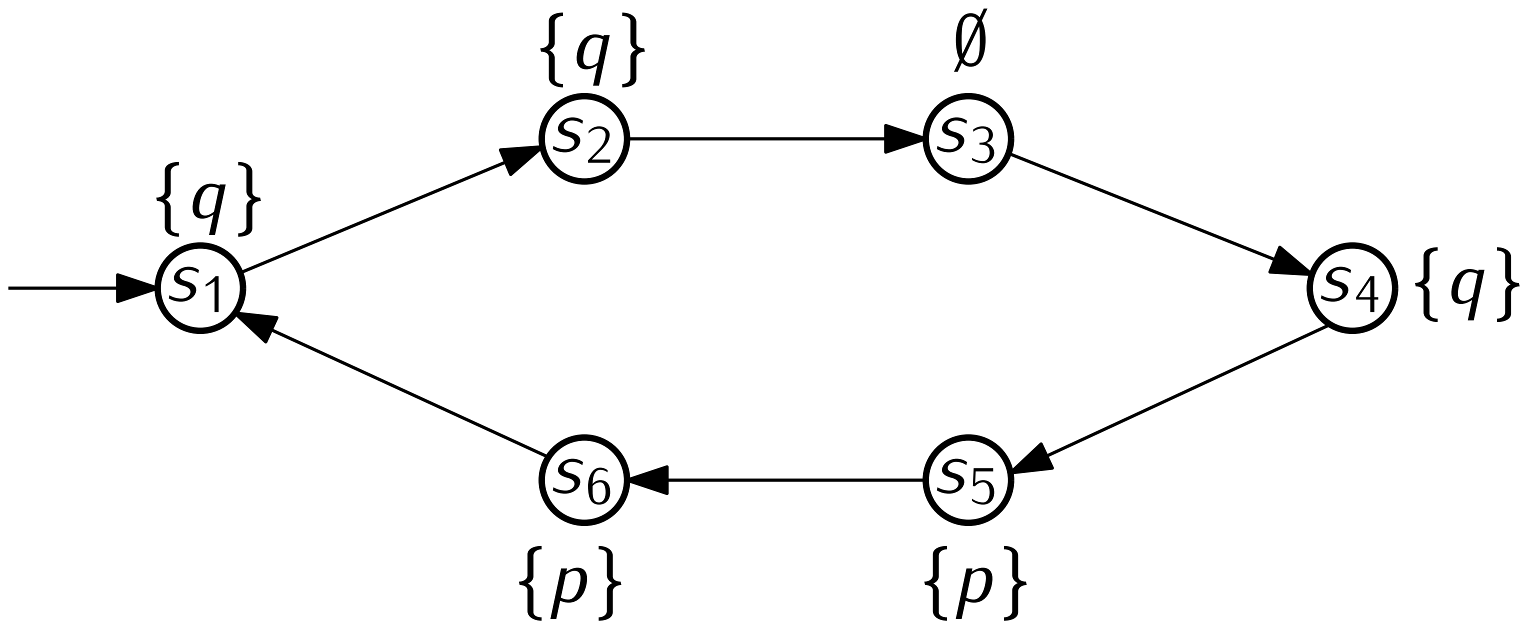
First, an example $\qquad p \; EU \; q$



$$s_1, s_2, s_3, s_5, s_6 \models (p \; EU \; q)$$

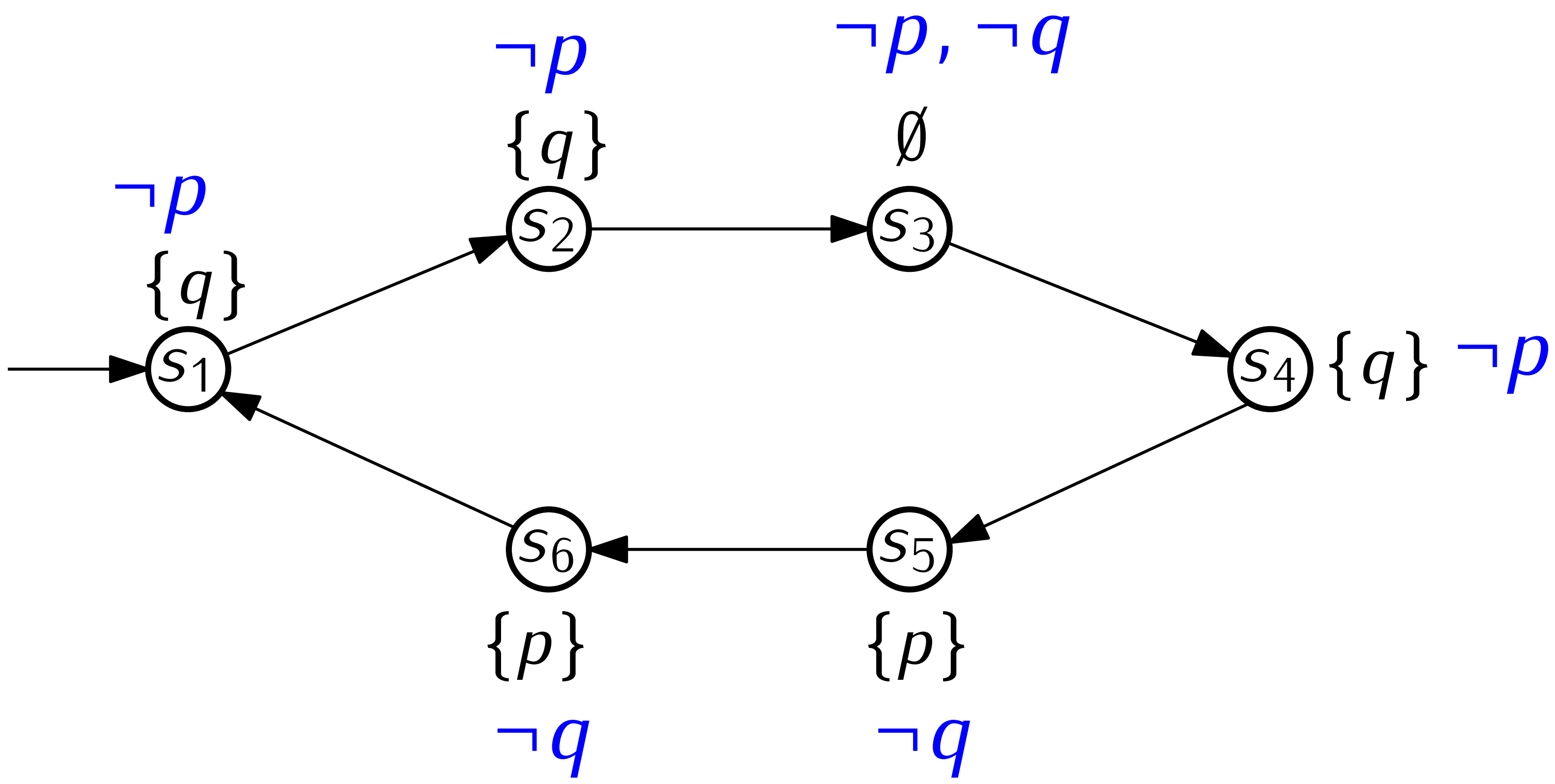
The Algorithm for $\phi\ EU\ \psi$

Another example $\neg p\ EU\ \neg q$



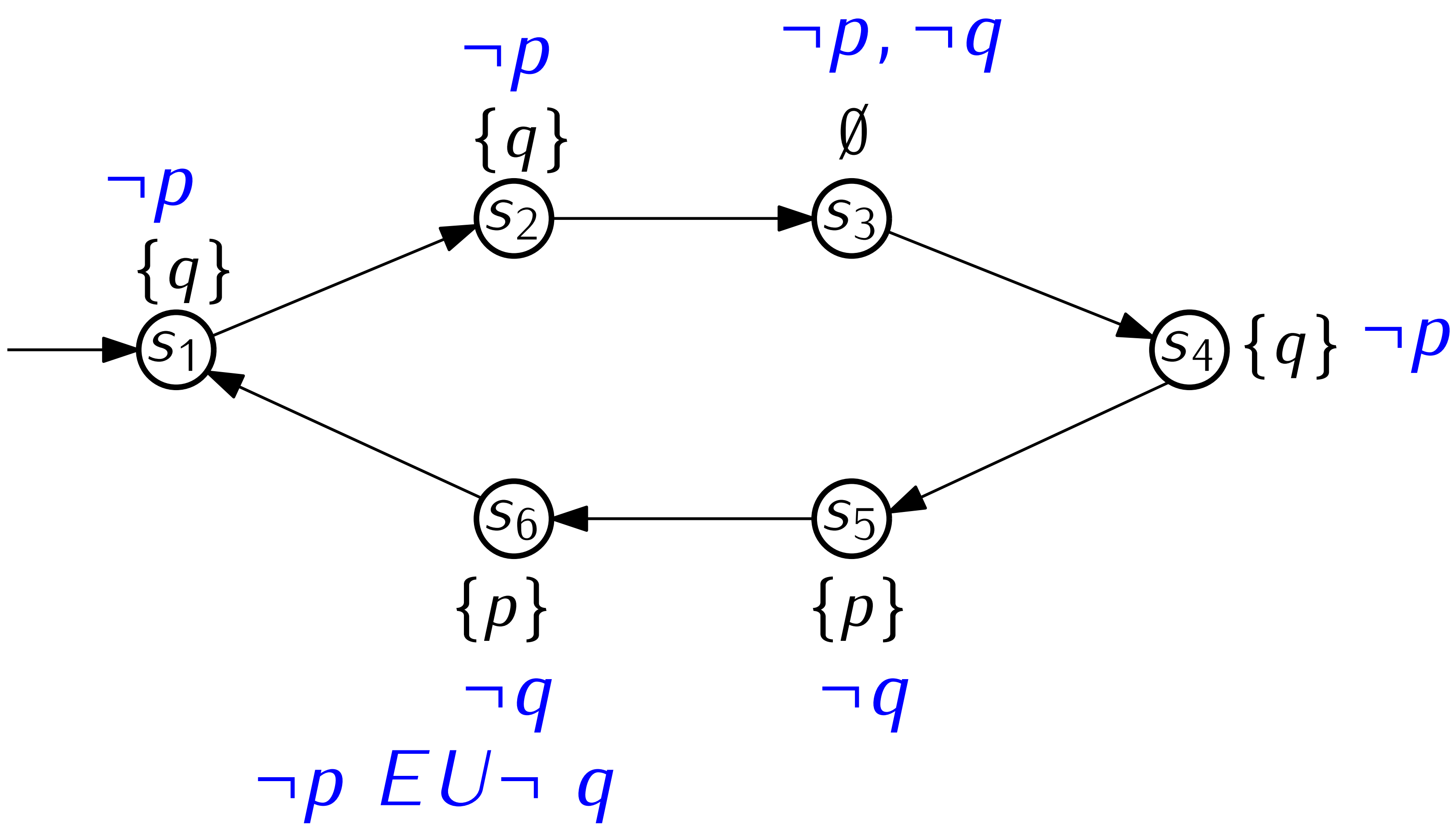
The Algorithm for $\phi\ EU\ \psi$

Another example $\qquad \neg p\ EU\ \neg q$



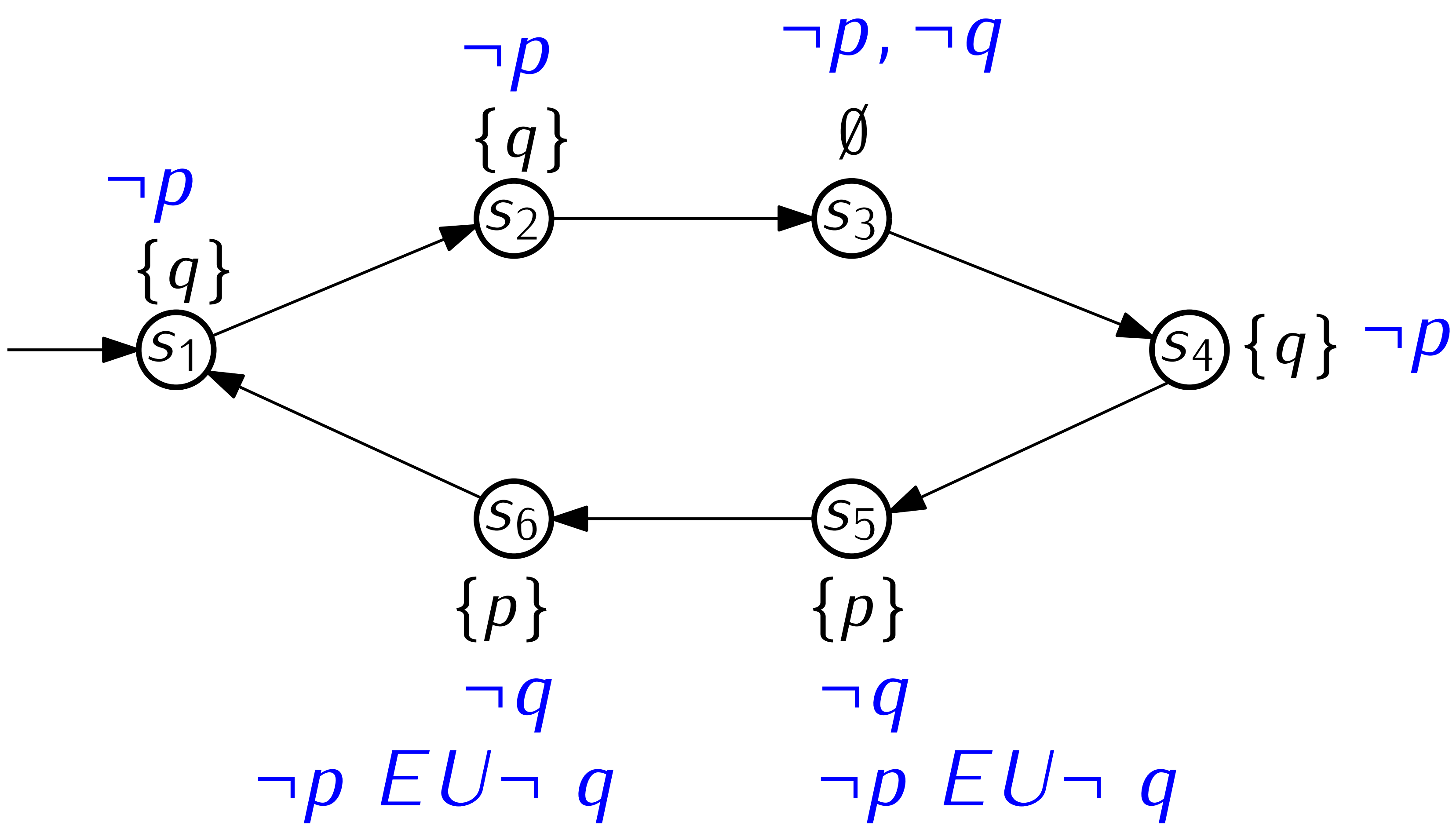
The Algorithm for $\phi\ EU\ \psi$

Another example $\neg p\ EU\ \neg q$



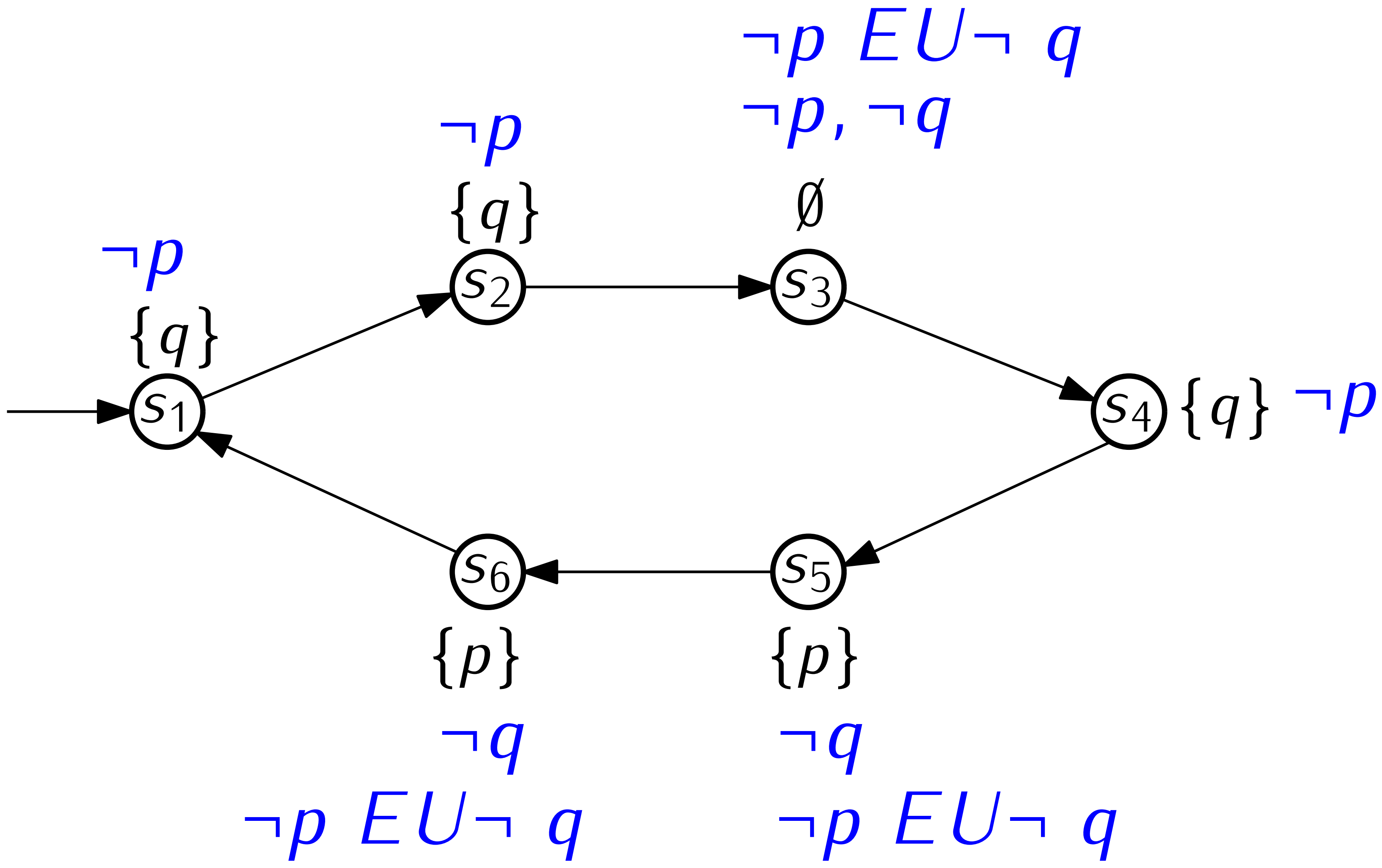
The Algorithm for $\phi\ EU\ \psi$

Another example $\neg p\ EU\ \neg q$



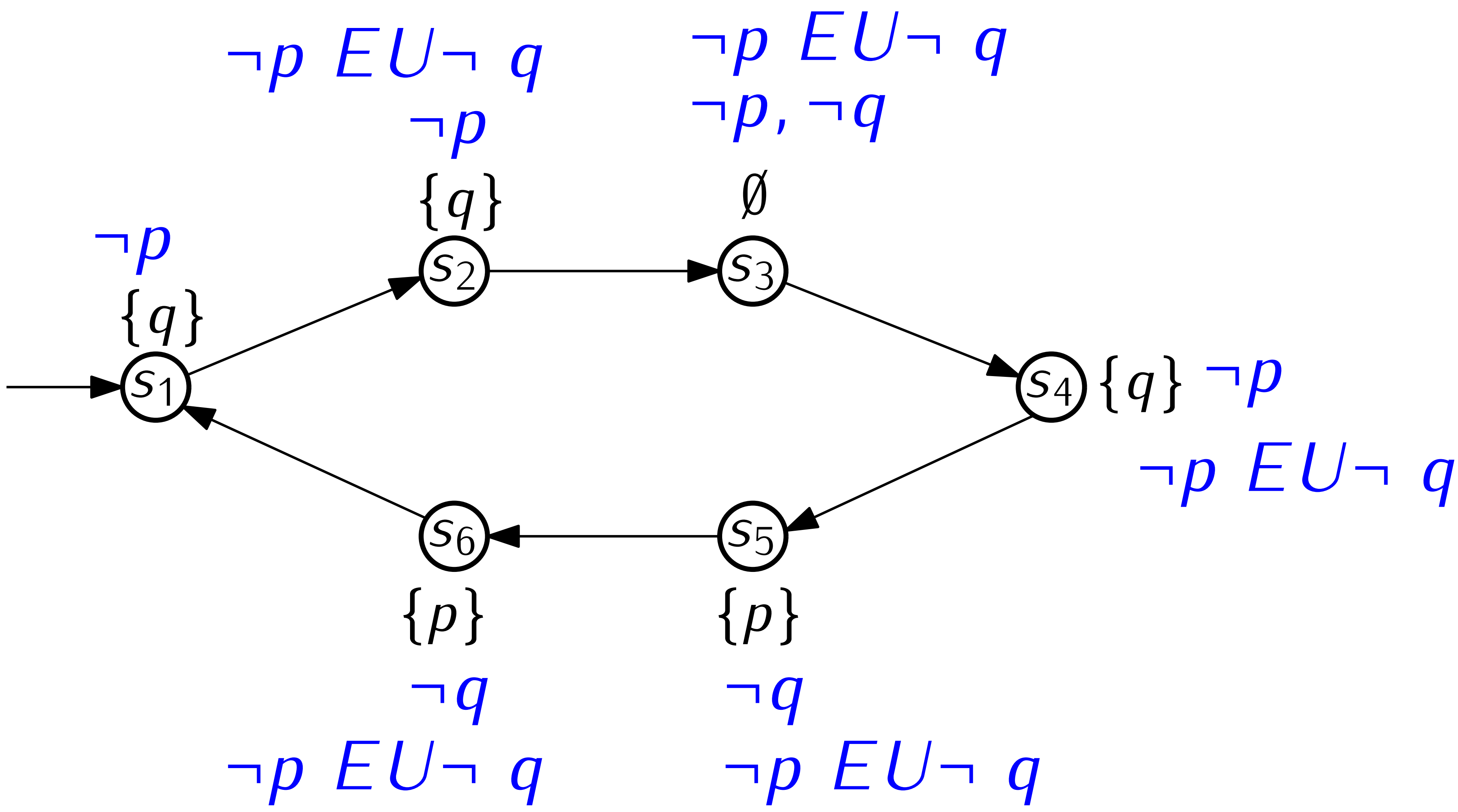
The Algorithm for $\phi\ EU\ \psi$

Another example $\neg p\ EU\ \neg q$



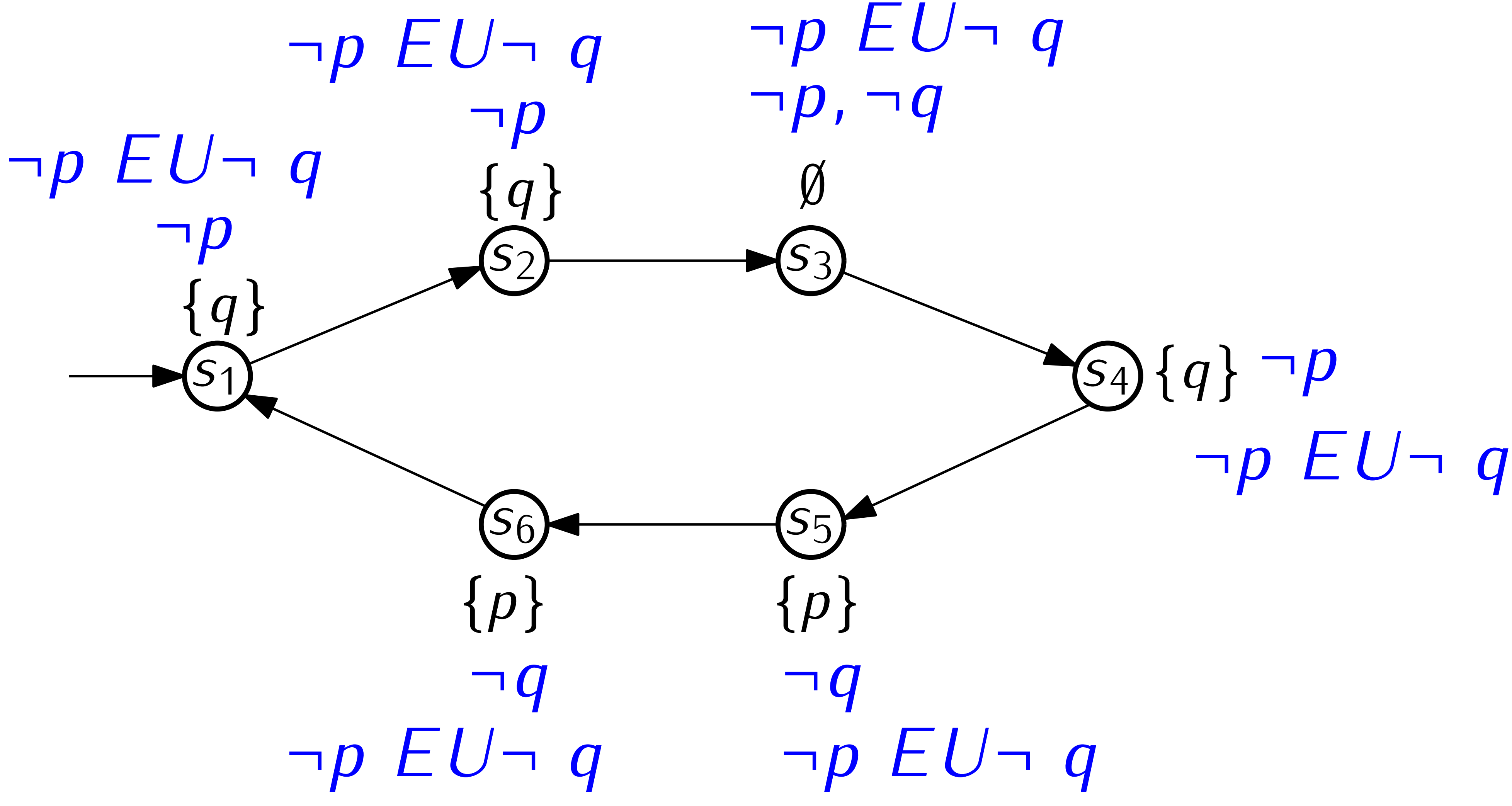
The Algorithm for $\phi \; EU \; \psi$

Another example $\neg p \; EU \; \neg q$



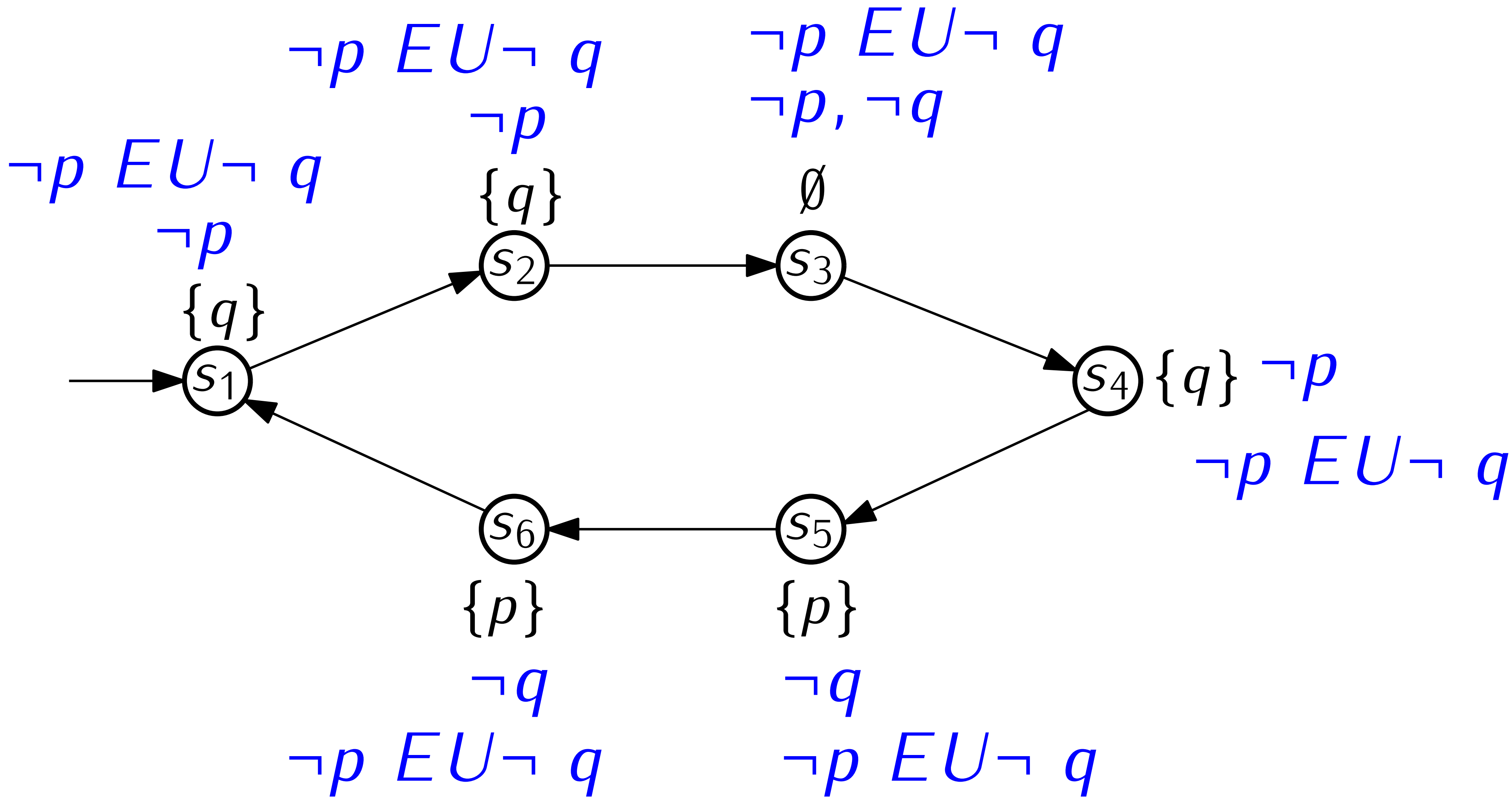
The Algorithm for $\phi \; EU \; \psi$

Another example $\neg p \; EU \; \neg q$



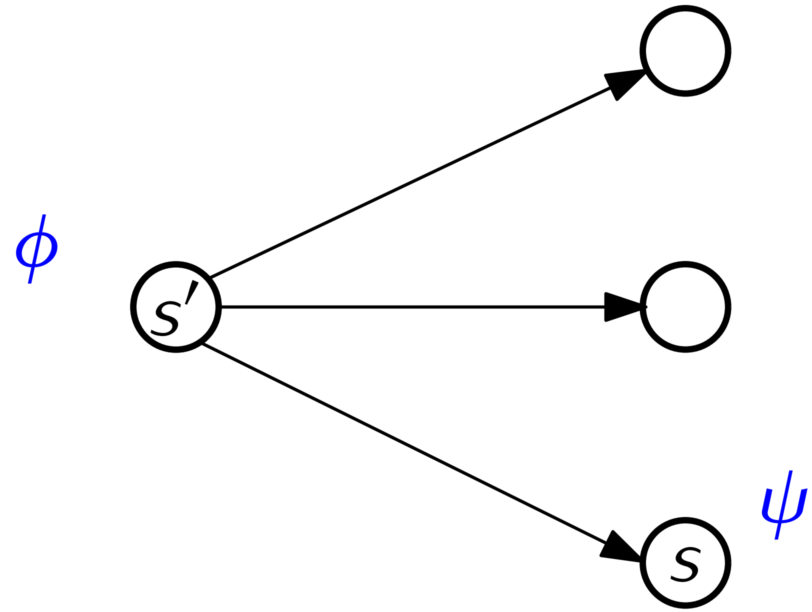
The Algorithm for $\phi\ EU\ \psi$

Another example $\qquad \neg p\ EU\ \neg q$

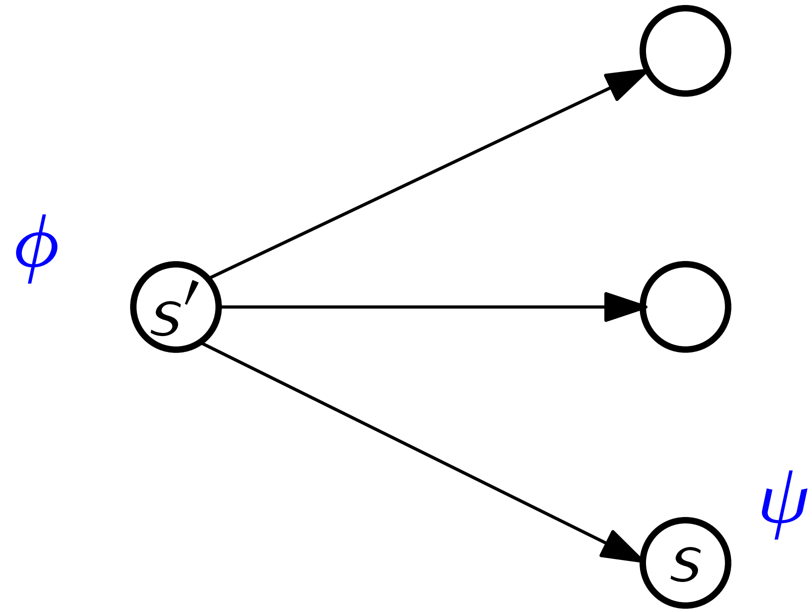


$$s_1, s_2, s_3, s_4, s_5, s_6 \models (\neg p\ EU\ \neg q)$$

The Algorithm for $\phi \; EU \; \psi$

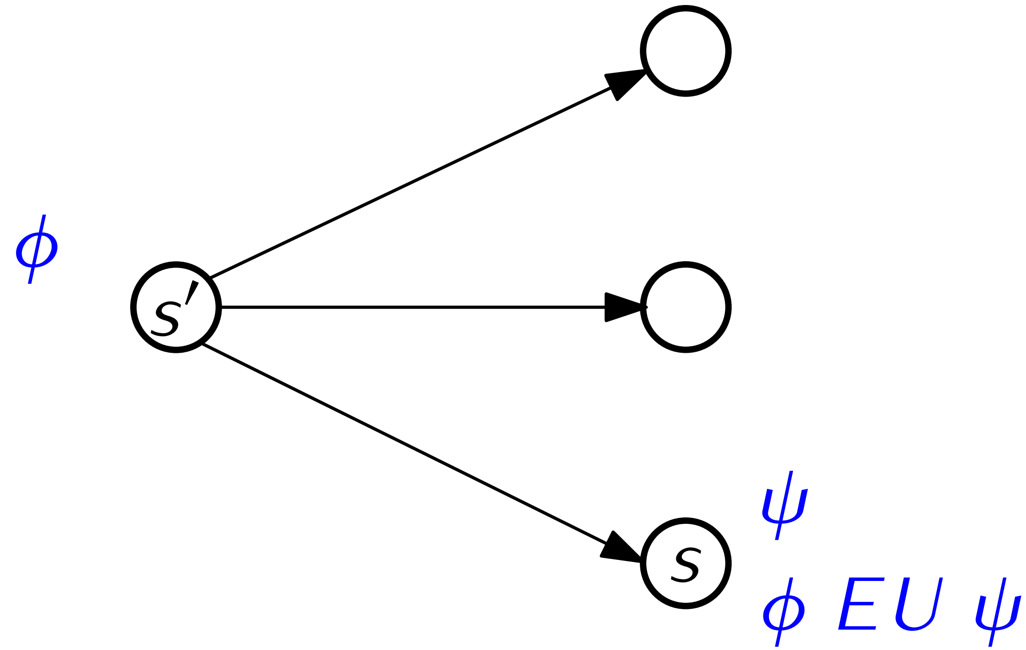


The Algorithm for ϕ EU ψ



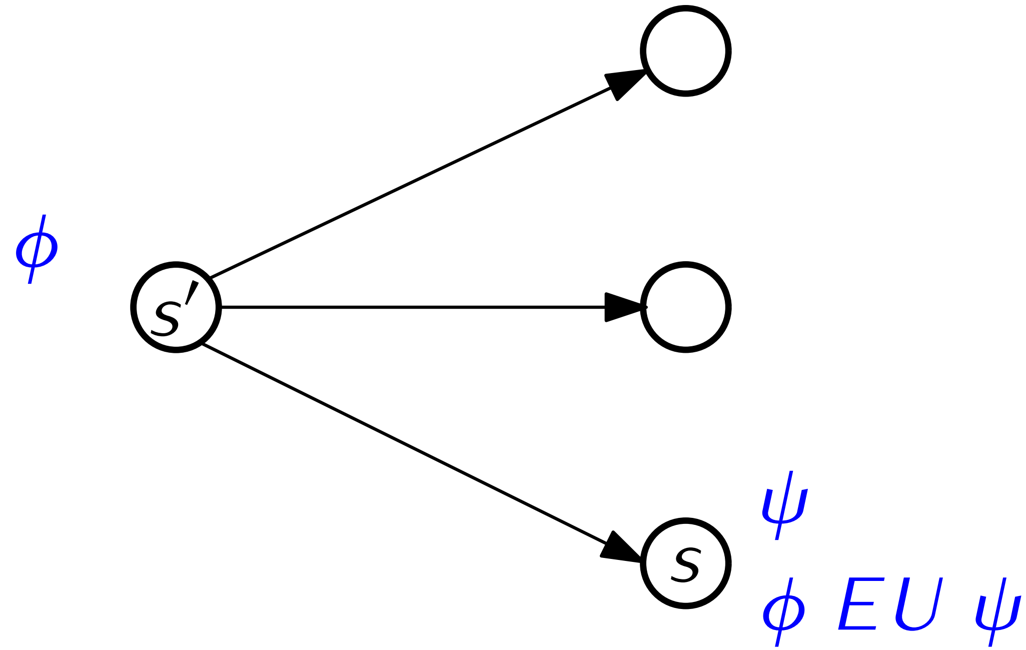
If a state is labelled with ψ label it with ϕ EU ψ.

The Algorithm for $\phi\ EU\ \psi$



If a state is labelled with ψ label it with $\phi\ EU\ \psi$.

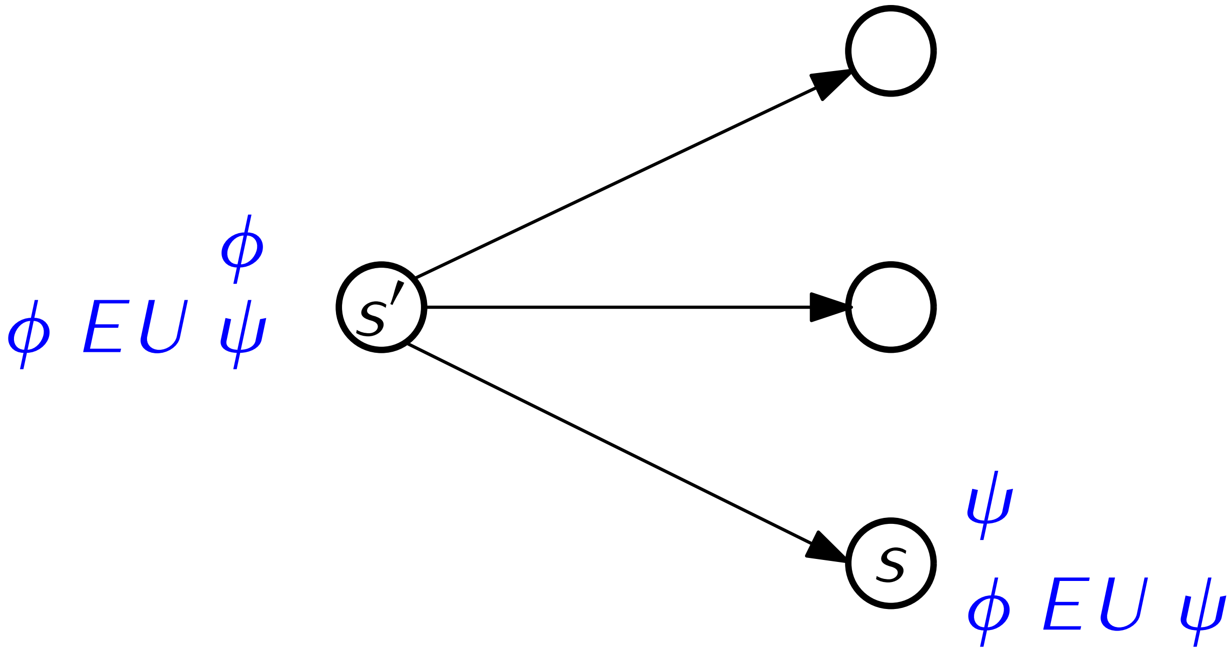
The Algorithm for $\phi\ EU\ \psi$



If a state is labelled with ψ label it with $\phi\ EU\ \psi$.

For any state s' labelled with ϕ, if at least one successor state s is labelled with $\phi\ EU\ \psi$, then label s' with $\phi\ EU\ \psi$ as well. Repeat until labels stop changing.

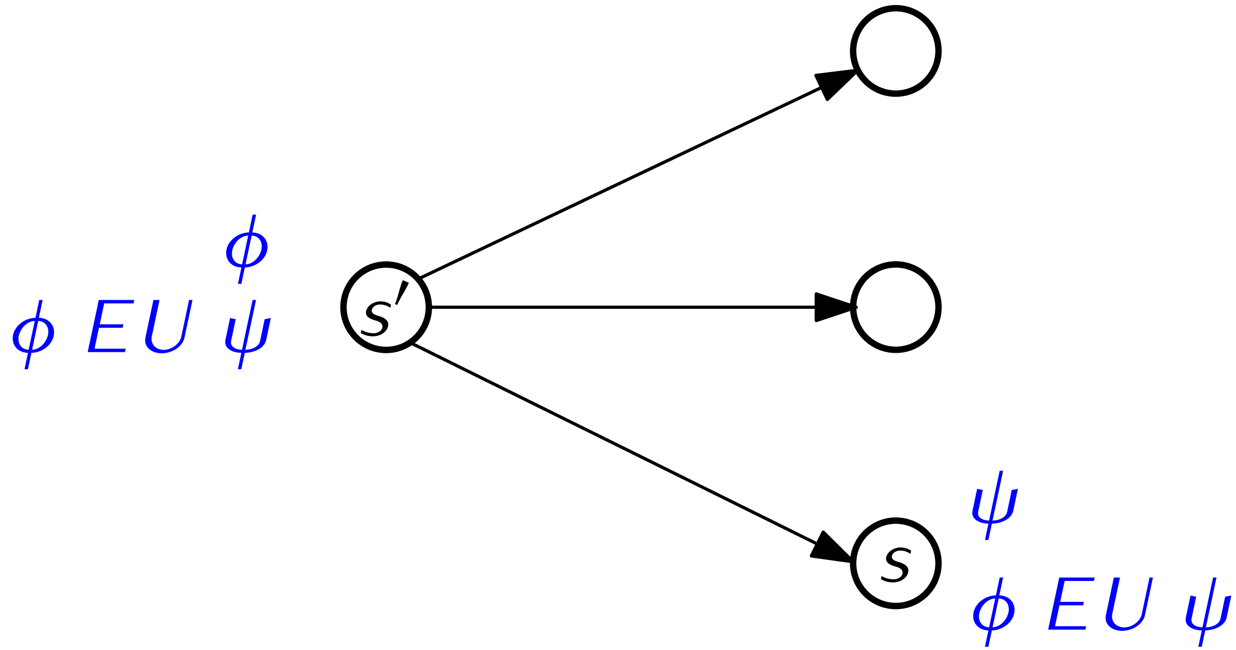
The Algorithm for $\phi \; EU \; \psi$



If a state is labelled with ψ label it with $\phi \; EU \; \psi$.

For any state s' labelled with ϕ, if at least one successor state s is labelled with $\phi \; EU \; \psi$, then label s' with $\phi \; EU \; \psi$ as well. Repeat until labels stop changing.

The Algorithm for $\phi\ EU\ \psi$



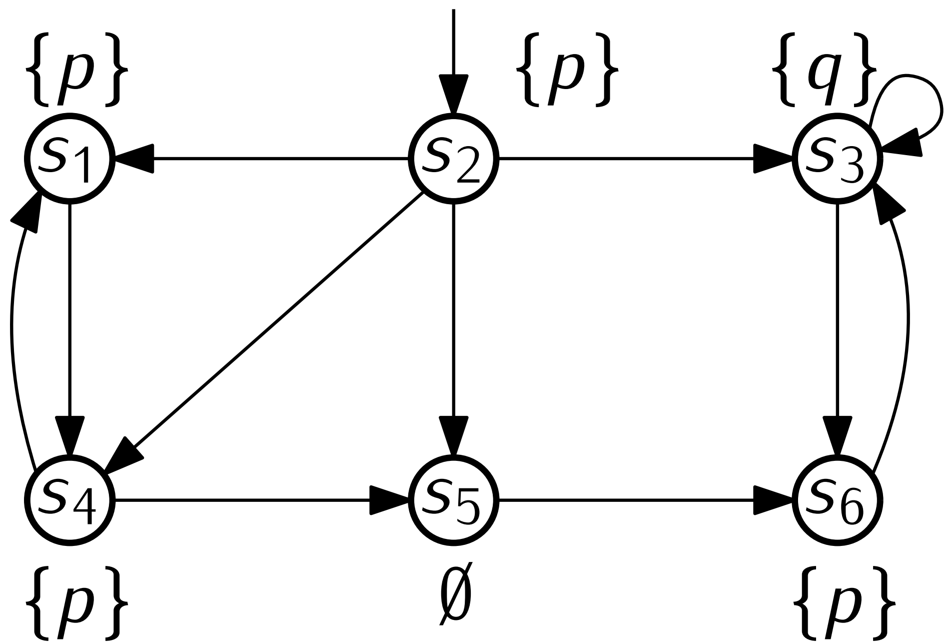
If a state is labelled with ψ label it with $\phi\ EU\ \psi$.

For any state s' labelled with ϕ, if at least one successor state s is labelled with $\phi\ EU\ \psi$, then label s' with $\phi\ EU\ \psi$ as well. Repeat until labels stop changing.

Call this process $\mathrm{SAT}_{EU}(\phi, \psi)$

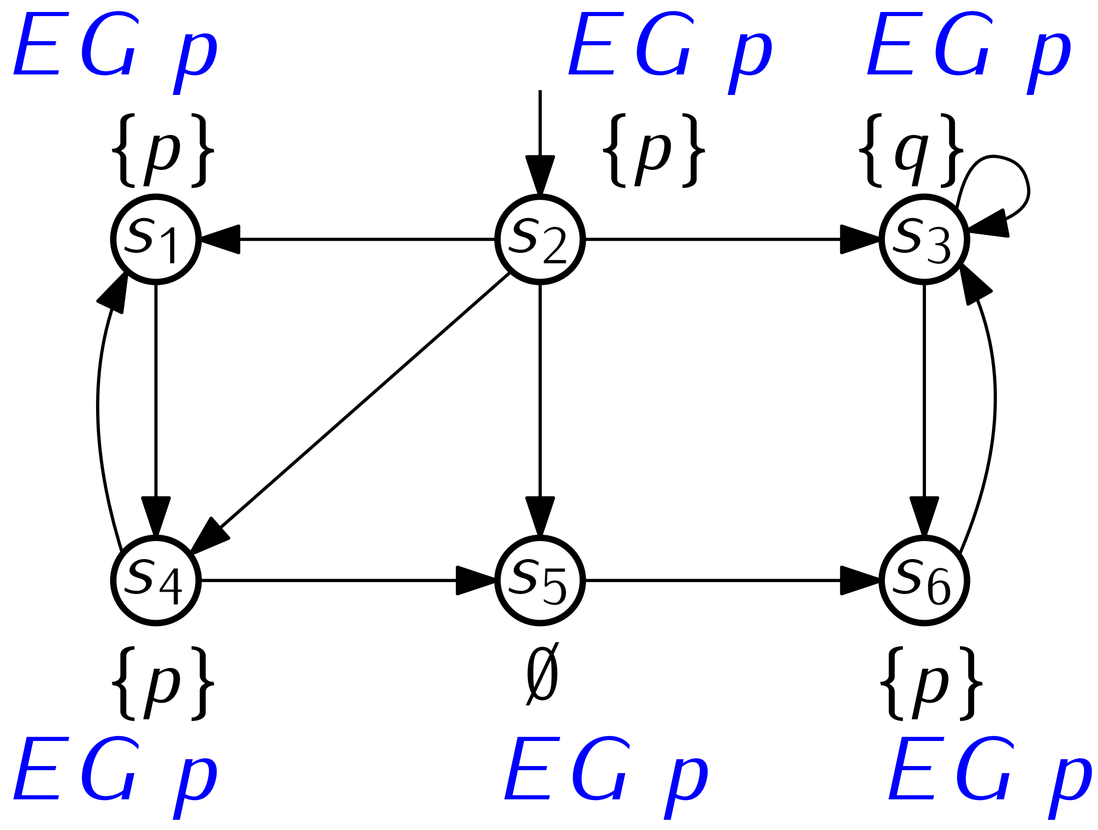
The Algorithm for $EG\ \phi$

First, an example $EG\ p$



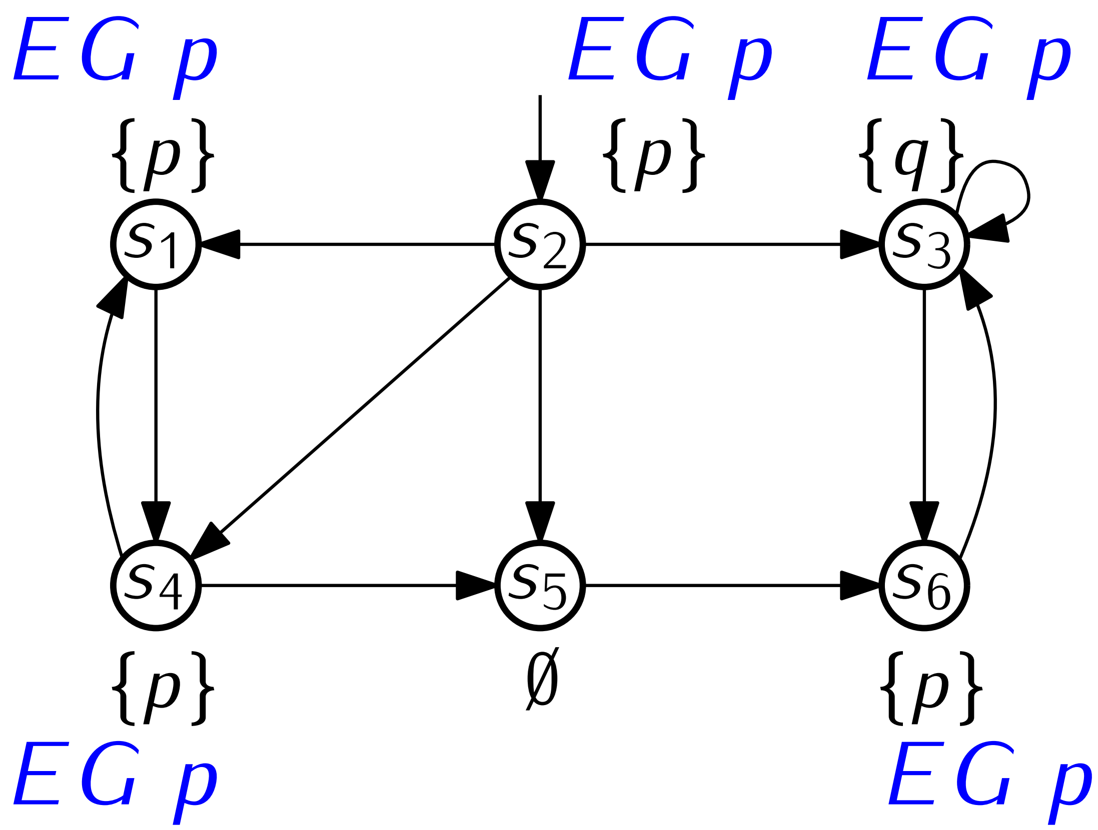
The Algorithm for $EG\ \phi$

First, an example $EG\ p$



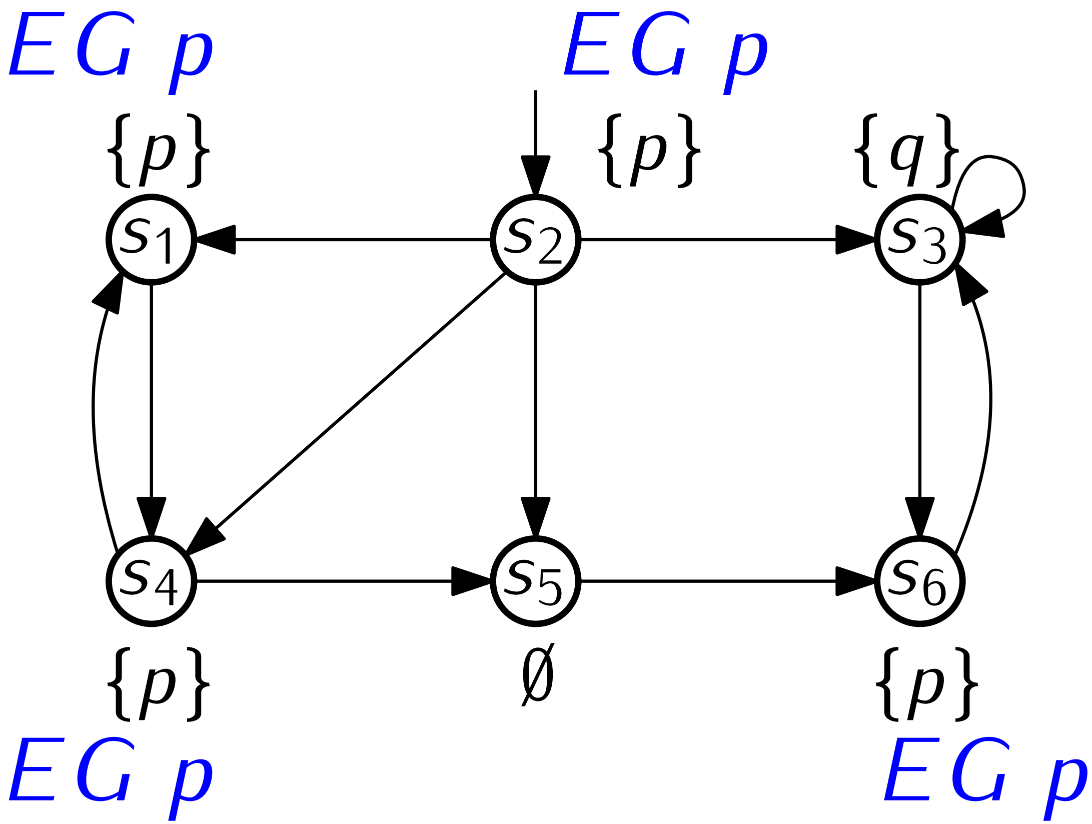
The Algorithm for $EG\ \phi$

First, an example $\qquad EG\ p$



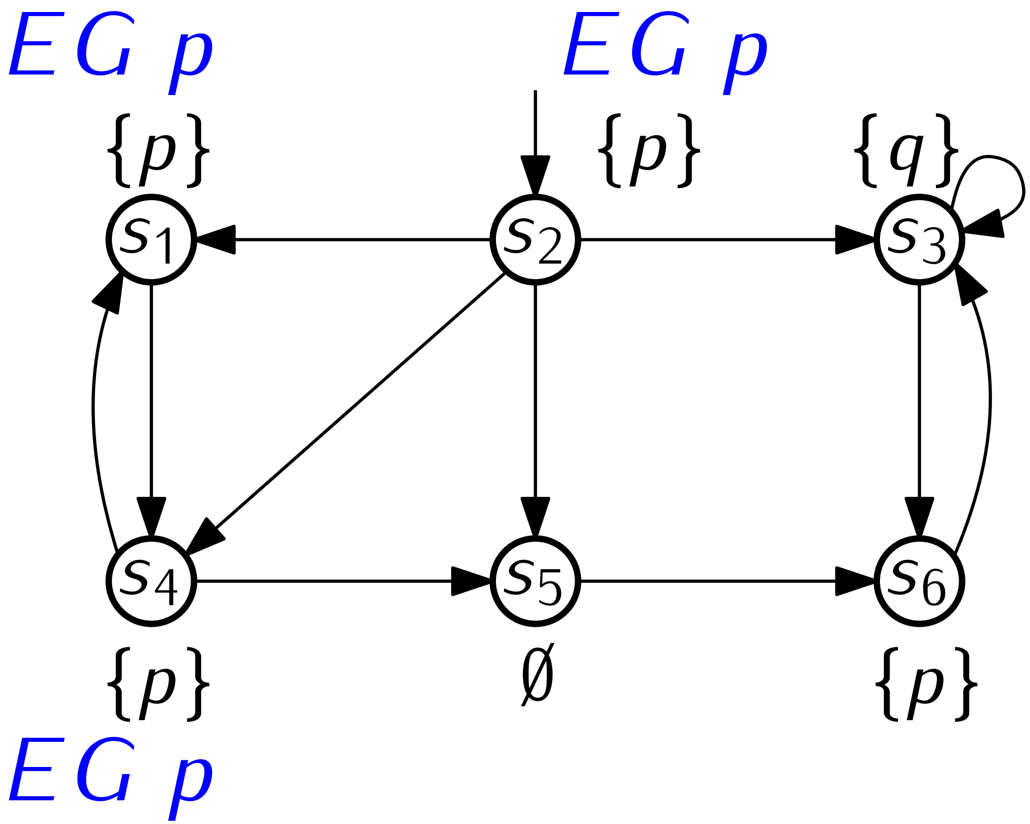
The Algorithm for $EG\ \phi$

First, an example $EG\ p$



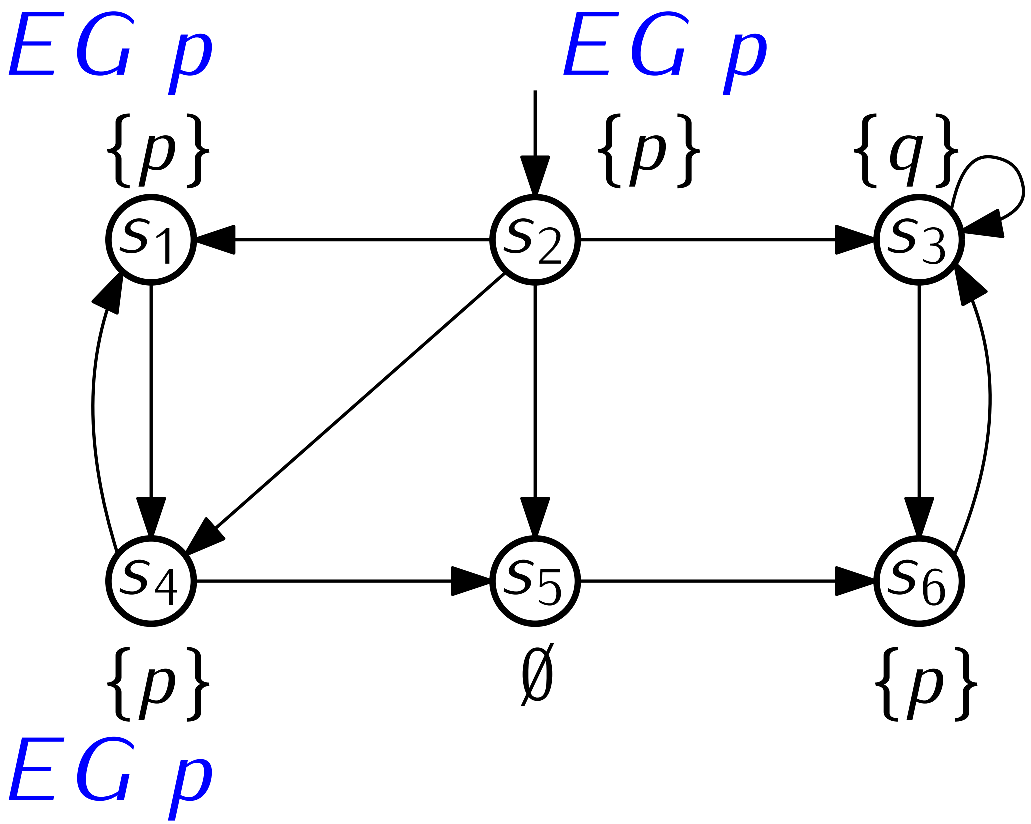
The Algorithm for $EG\ \phi$

First, an example $\quad EG\ p$



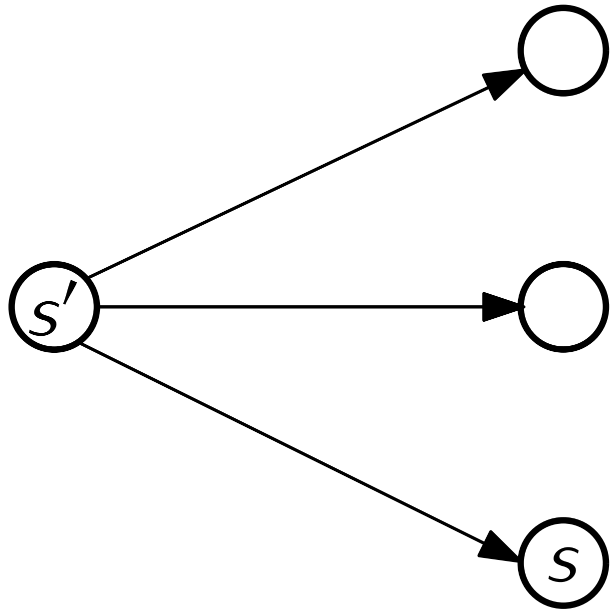
The Algorithm for $EG\ \phi$

First, an example $\qquad EG\ p$

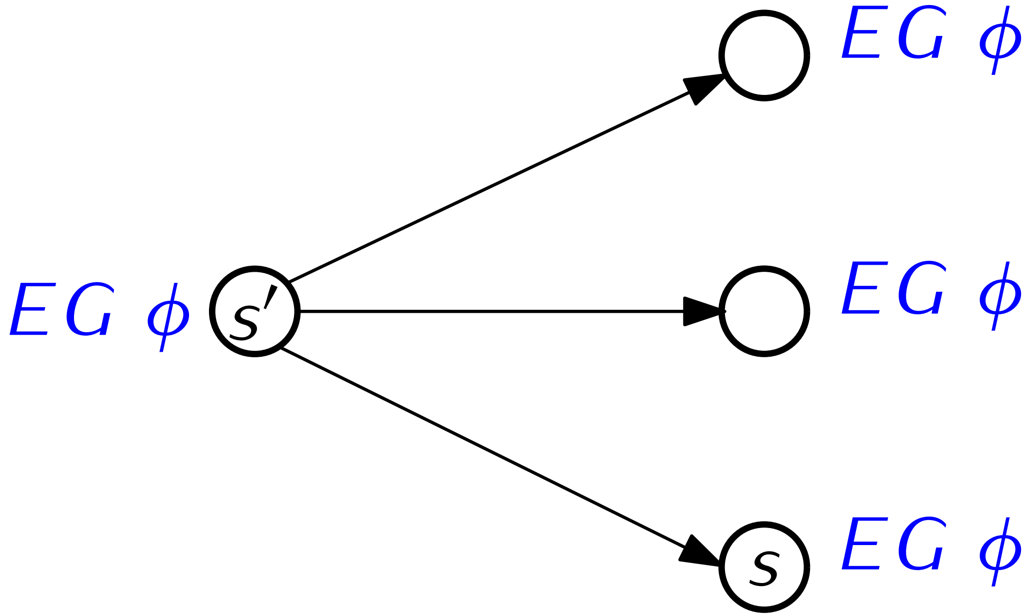


$$s_1, s_2, s_4 \models EG\ p$$

The Algorithm for $EG\ \phi$

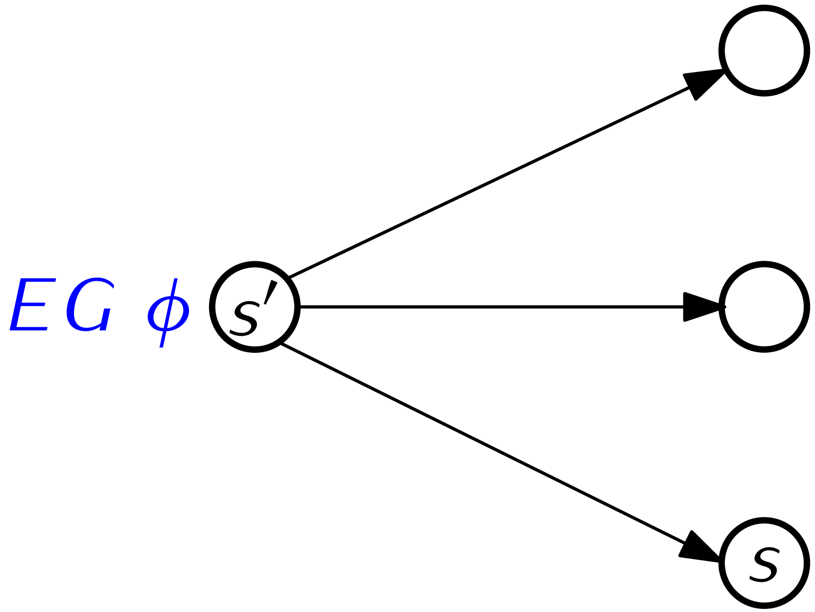


The Algorithm for $EG\ \phi$



Label all states with $EG\ \phi$

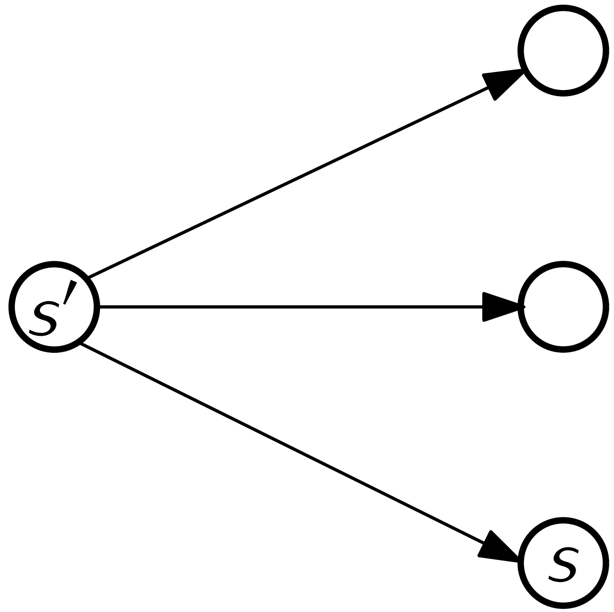
The Algorithm for $EG\ \phi$



Label all states with $EG\ \phi$

Delete $EG\ \phi$ from any state not labelled with ϕ.

The Algorithm for $EG\ \phi$

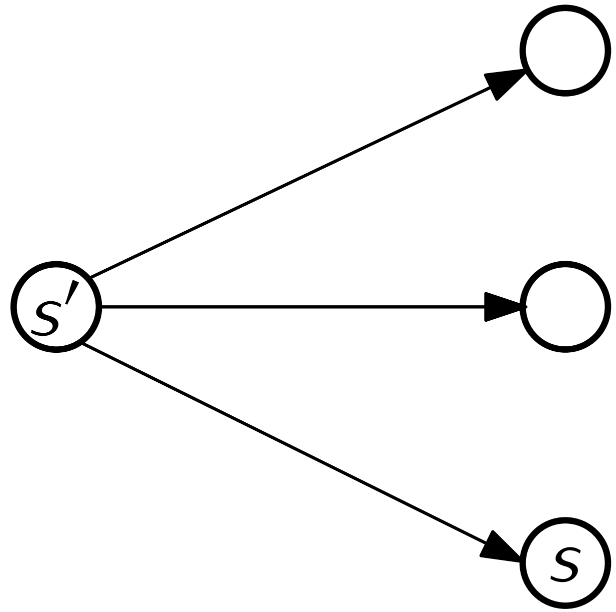


Label all states with $EG\ \phi$

Delete $EG\ \phi$ from any state not labelled with ϕ.

Delete $EG\ \phi$ from any state where none of its successors is labelled with $EG\ \phi$. Repeat until no more labels can be deleted.

The Algorithm for $EG\ \phi$



Label all states with $EG\ \phi$

Delete $EG\ \phi$ from any state not labelled with ϕ.

Delete $EG\ \phi$ from any state where none of its successors is labelled with $EG\ \phi$. Repeat until no more labels can be deleted.

Call this process $\mathrm{SAT}_{EG}(\phi)$

Summary so far:

Rewrite everything in terms of EX, EG, EU.

$$AX\phi \equiv \neg EX \neg \phi$$

$$AG\phi \equiv \neg EF \neg \phi$$

$$AF\phi \equiv \neg EG \neg \phi$$

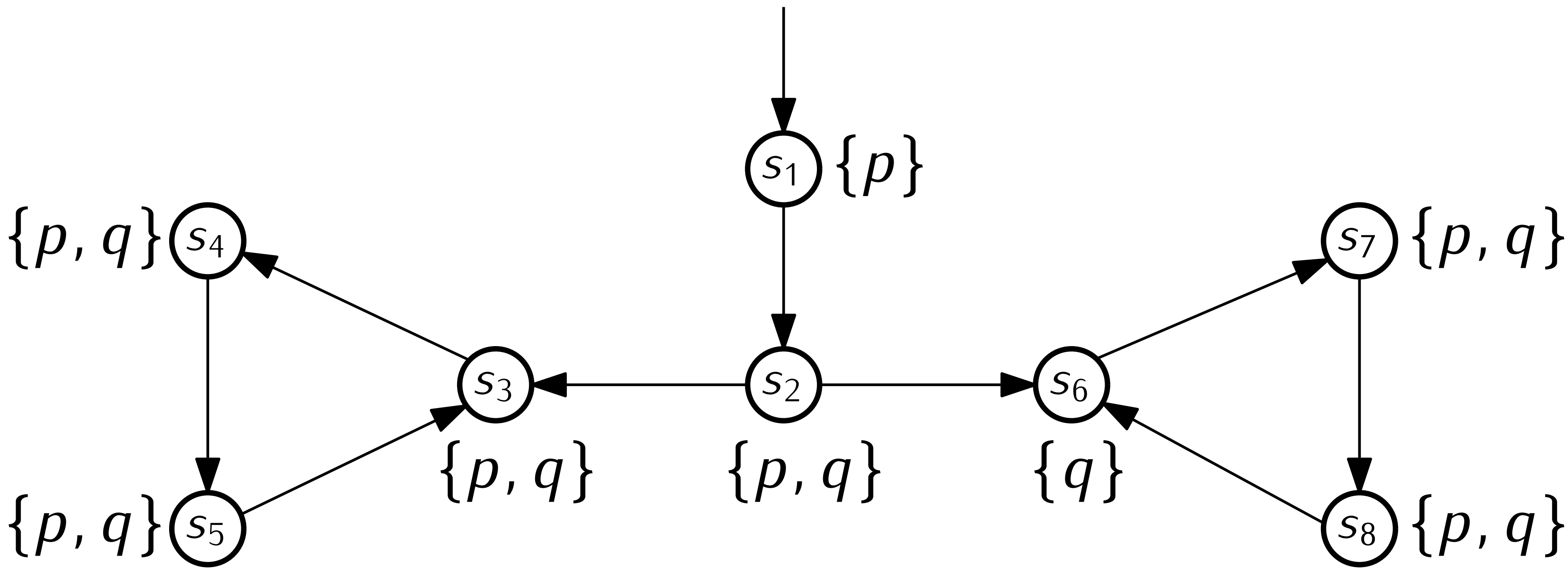
$$\phi AU \psi \equiv \neg(EG \neg \phi \quad \vee \quad \neg \phi \; EU \; (\neg \phi \wedge \neg \psi))$$

Procedures for determining the set of satisfied states

$$\mathrm{SAT}_{EX}(\phi),\ \mathrm{SAT}_{EG}(\phi),\ \mathrm{SAT}_{EU}(\phi, \psi)$$

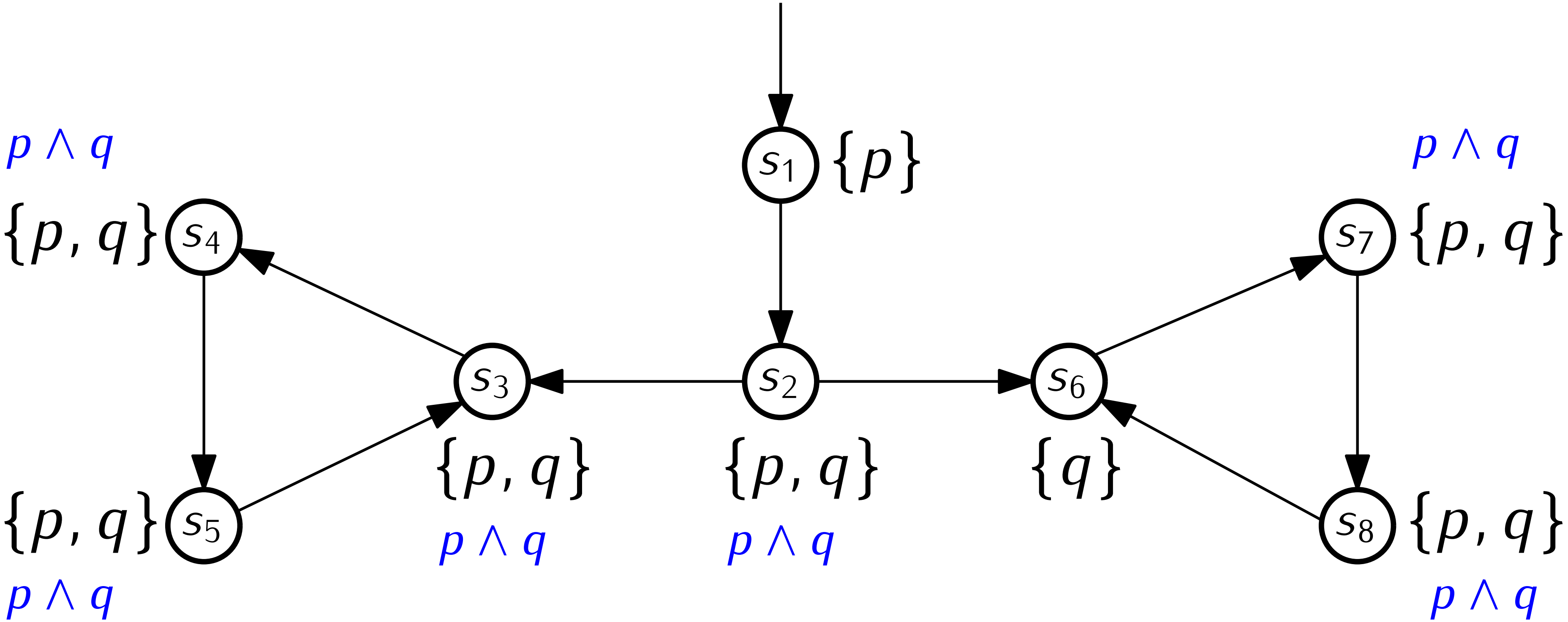
The General CTL Model Checking Algorithm

Example $EX\ EG\ (p \wedge q)$



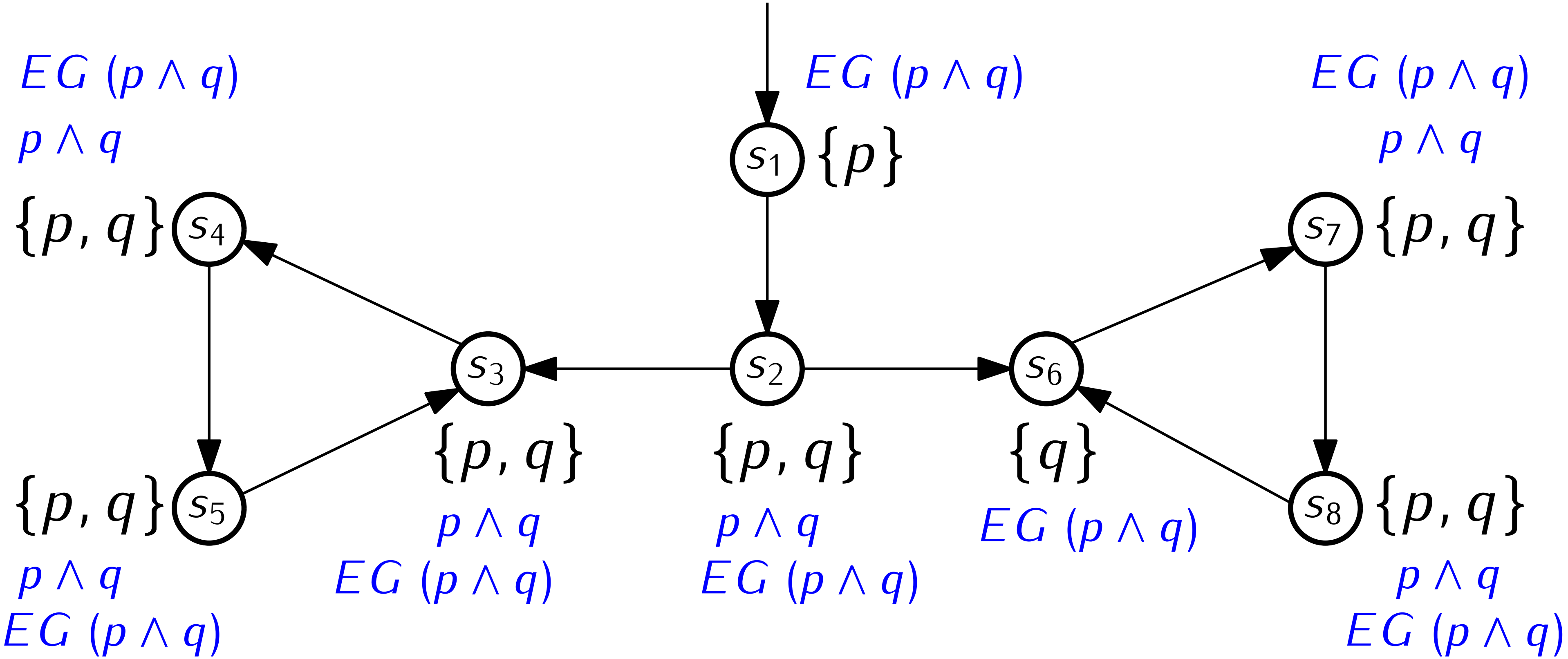
The General CTL Model Checking Algorithm

Example $EX\ EG\ (\underline{p \wedge q})$



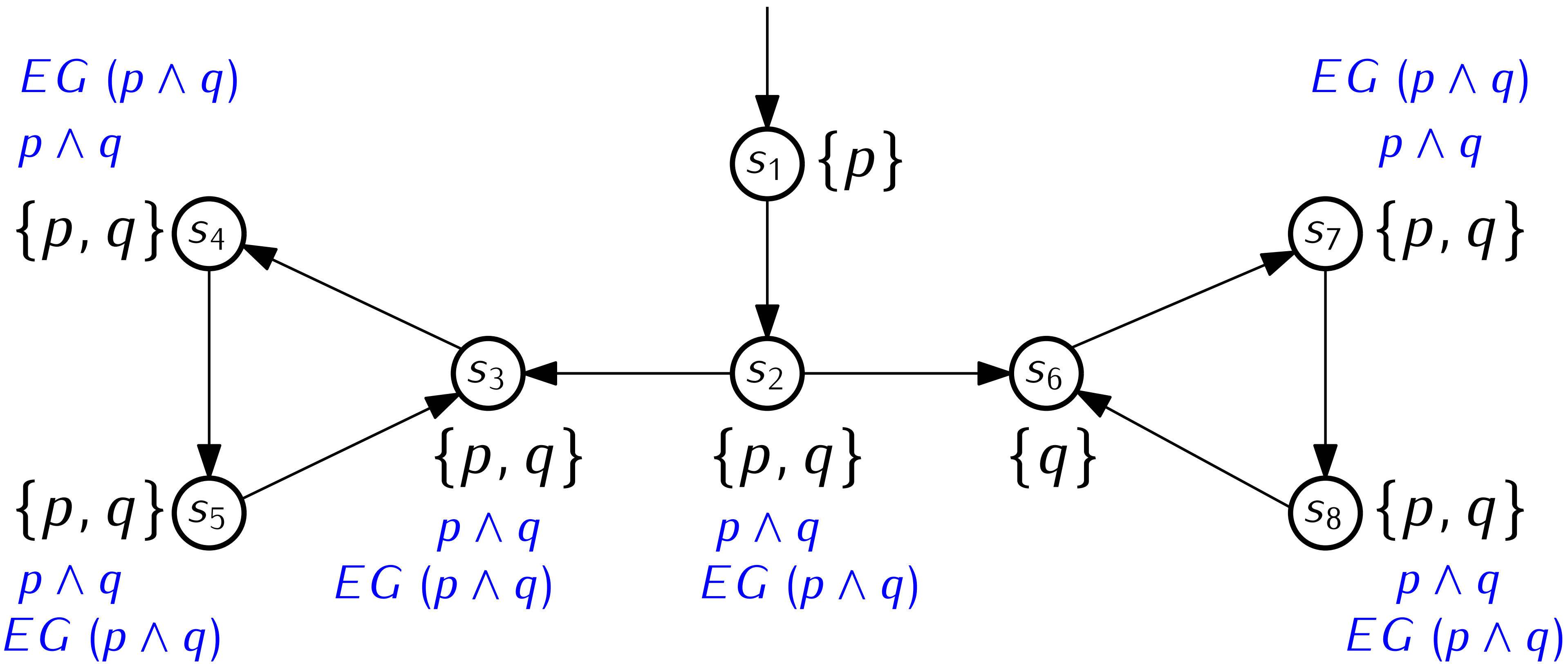
The General CTL Model Checking Algorithm

Example $EX\ \underline{EG\ (p \wedge q)}$



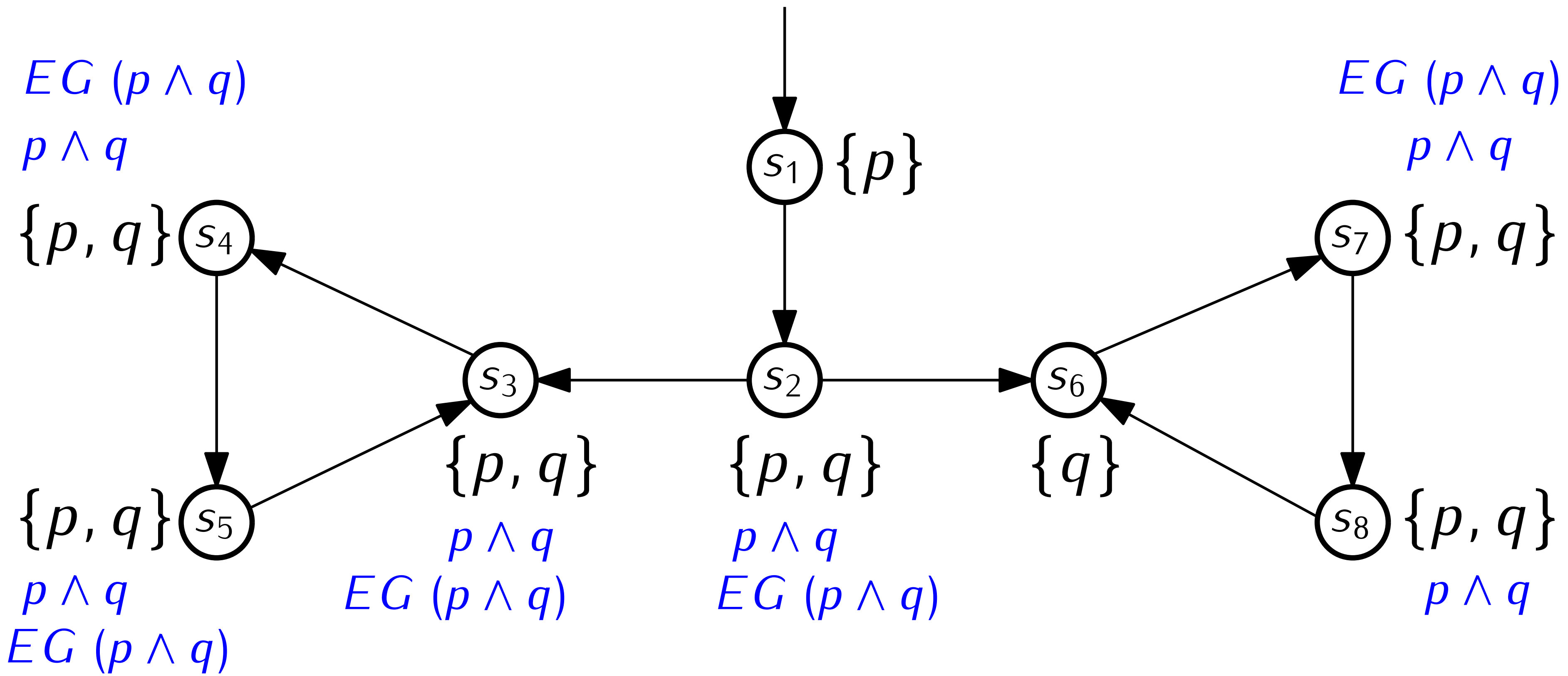
The General CTL Model Checking Algorithm

Example $EX\ \underline{EG\ (p \wedge q)}$



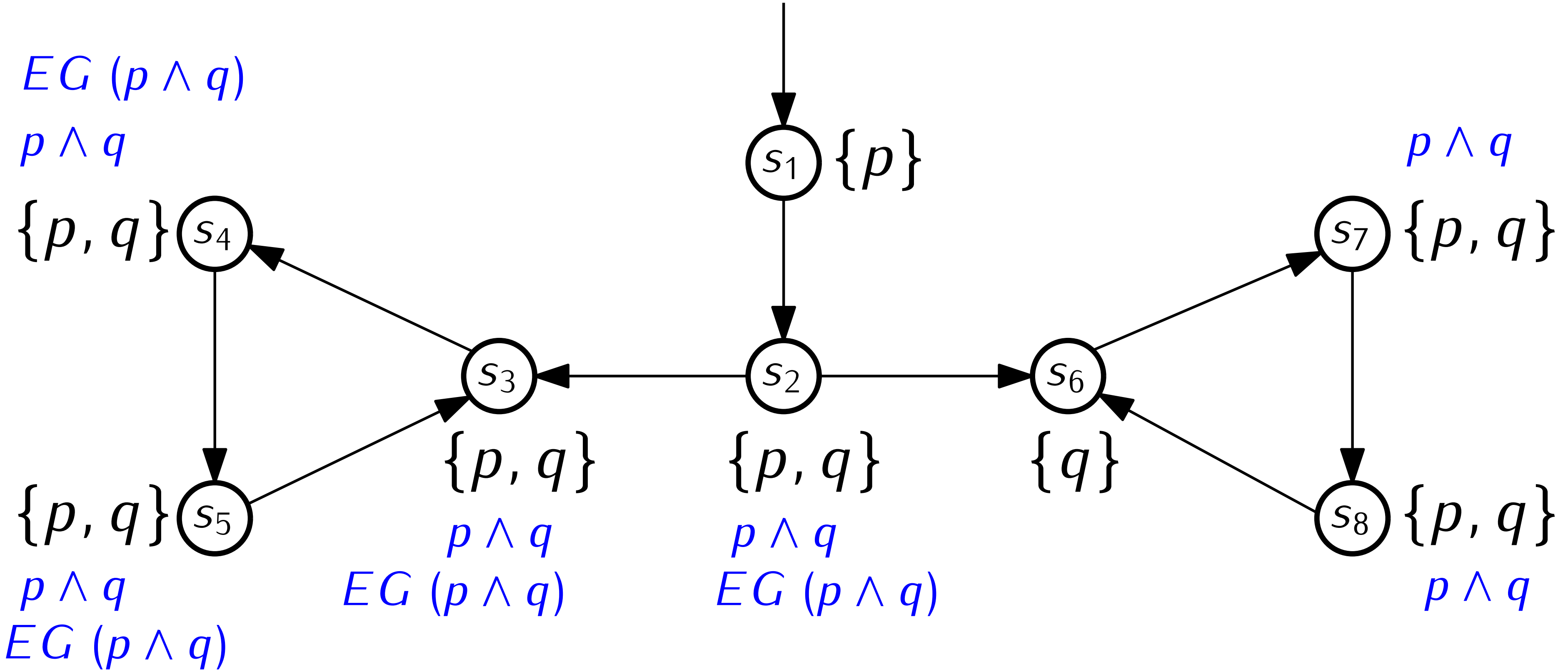
The General CTL Model Checking Algorithm

Example $EX\ \underline{EG\ (p \wedge q)}$



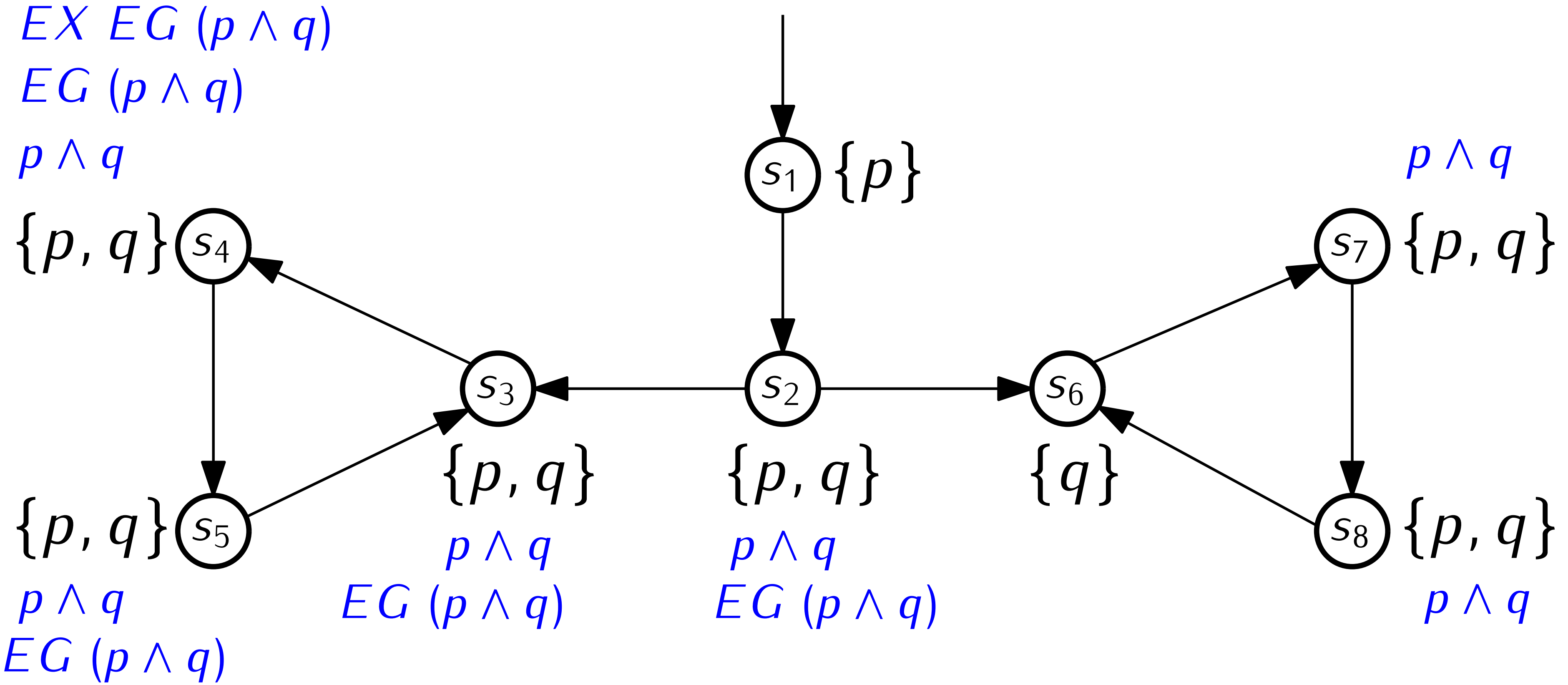
The General CTL Model Checking Algorithm

Example $EX\ \underline{EG\ (p \wedge q)}$



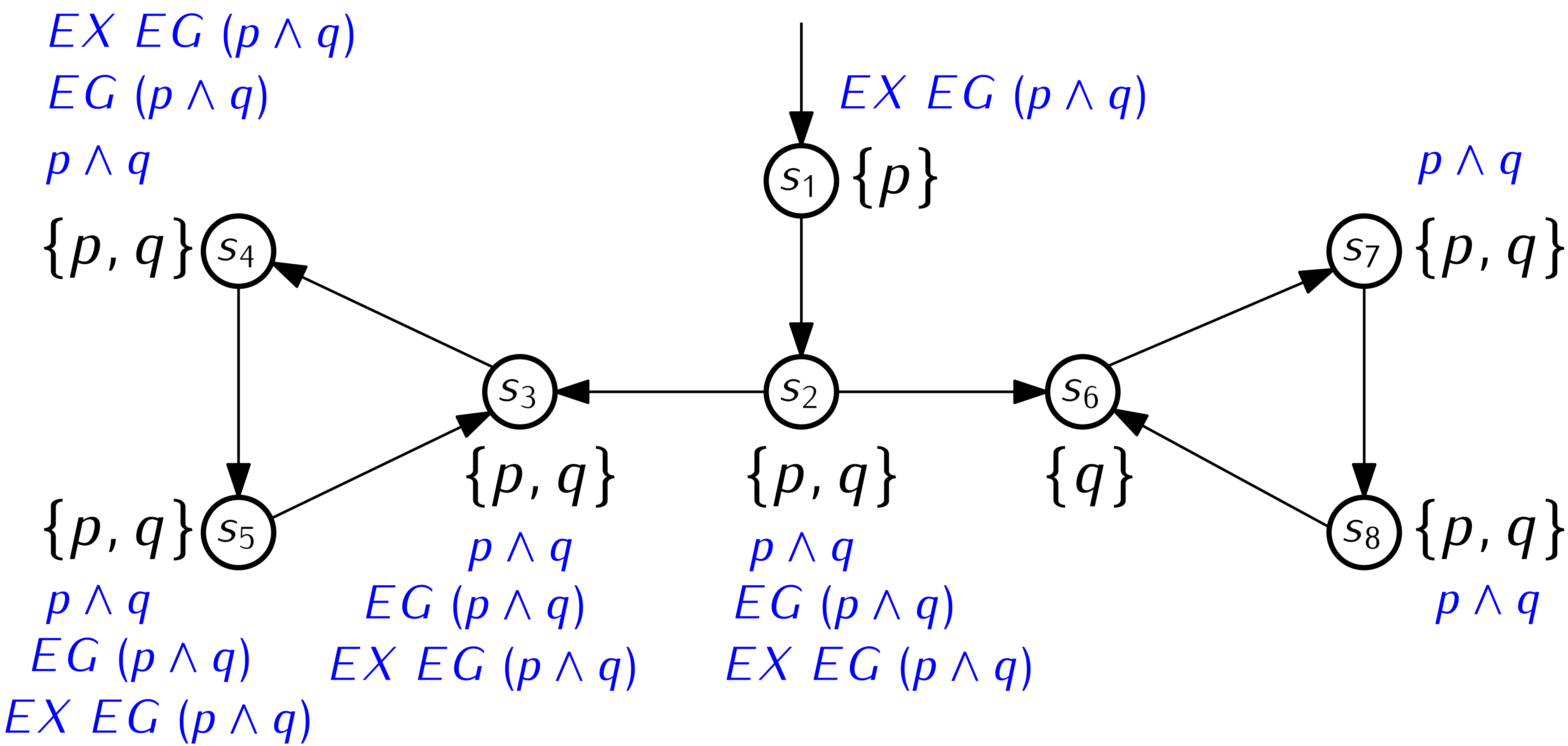
The General CTL Model Checking Algorithm

Example $EX\ EG\ (p \wedge q)$



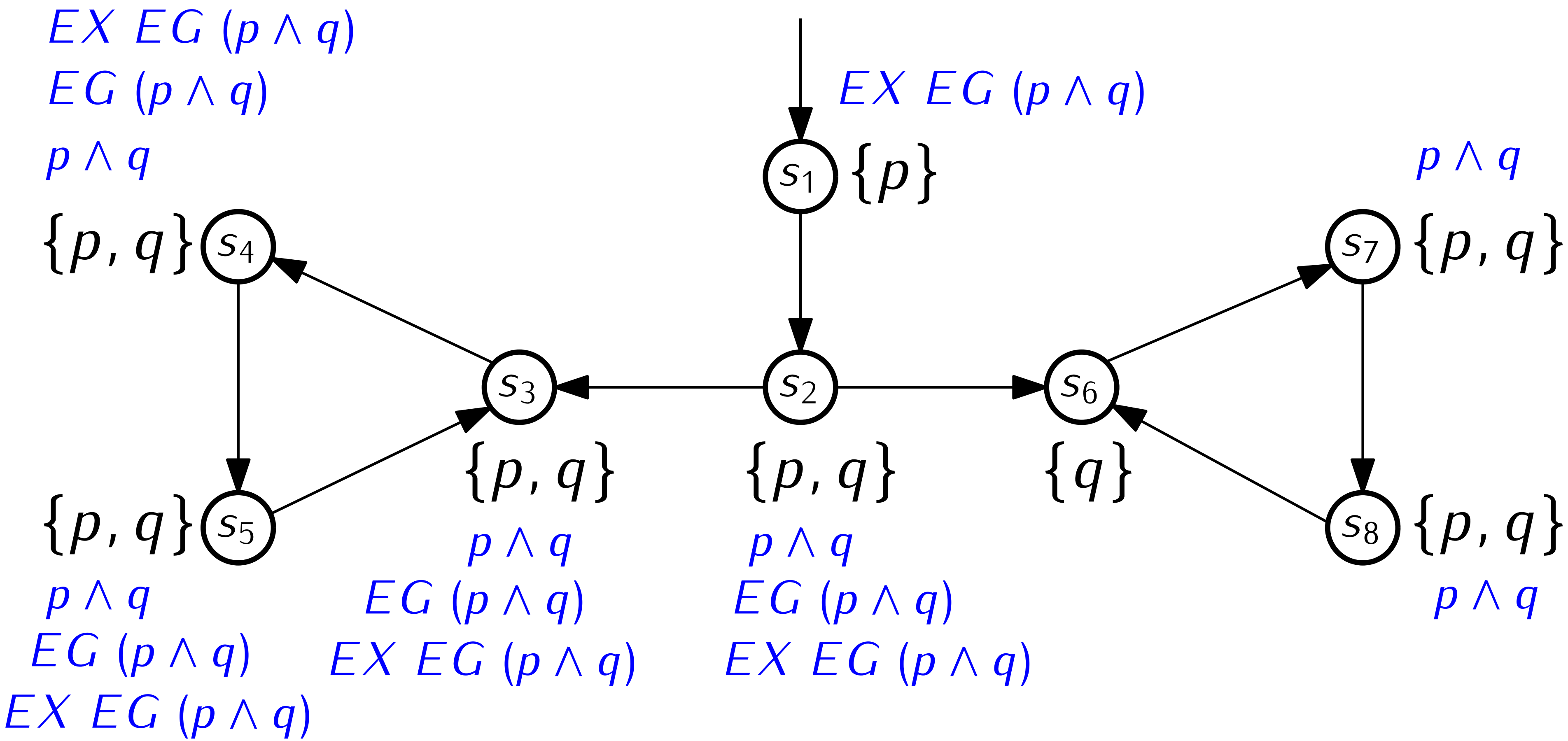
The General CTL Model Checking Algorithm

Example $EX\ EG\ (p \wedge q)$



The General CTL Model Checking Algorithm

Example $\quad EX\ EG\ (p \wedge q)$

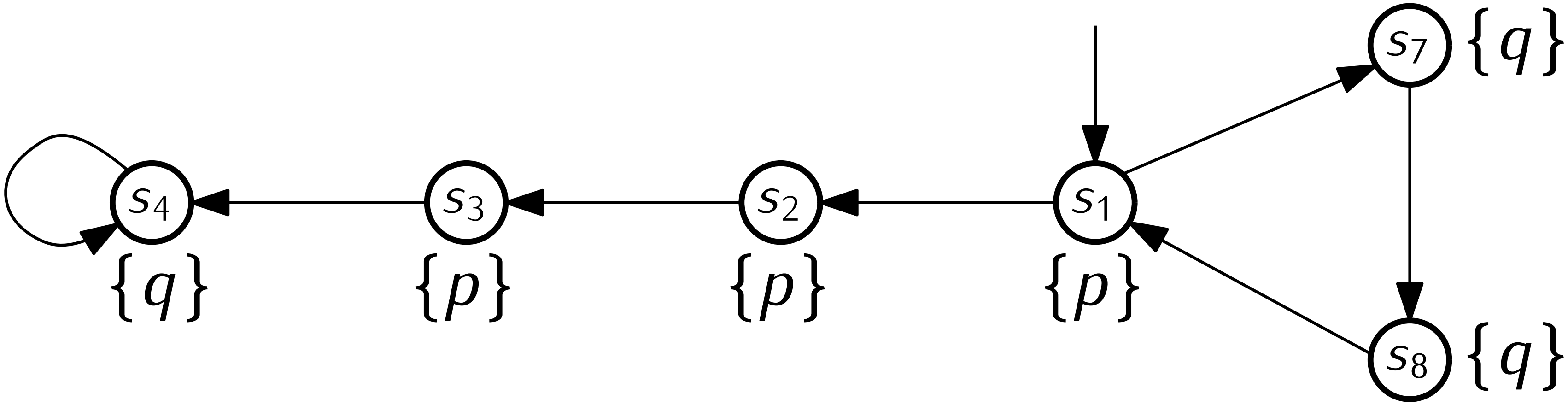


$$s_1, s_2, s_3, s_4, s_5, s_6 \models EX\ EG\ (p \wedge q) \ \wedge s_1 \in I$$

$$\Rightarrow \ \mathcal{M} \models EX\ EG\ (p \wedge q)$$

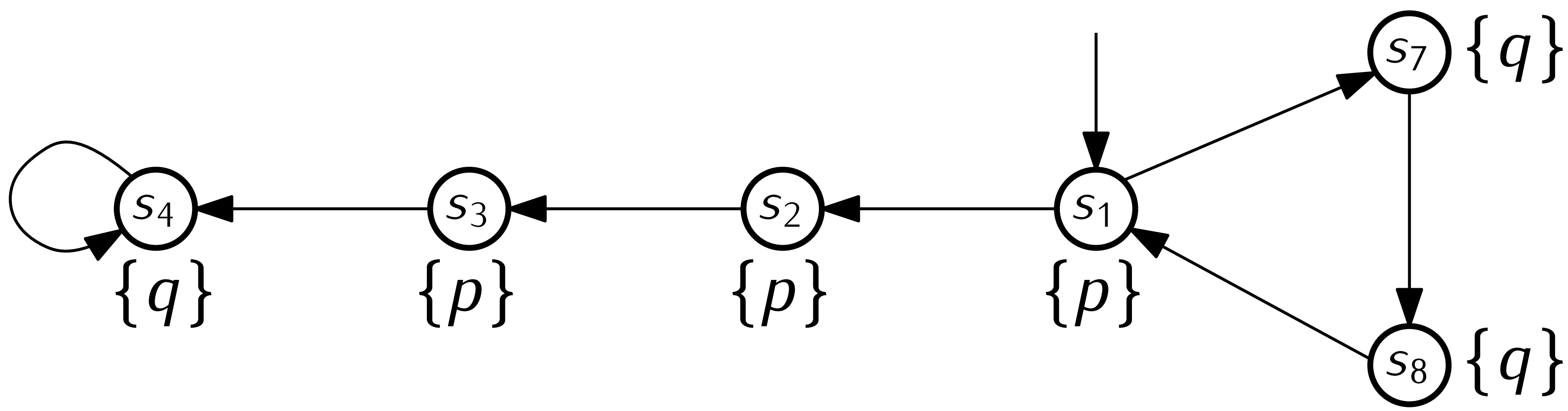
The General CTL Model Checking Algorithm

Example $p \; EU(EG \; q)$



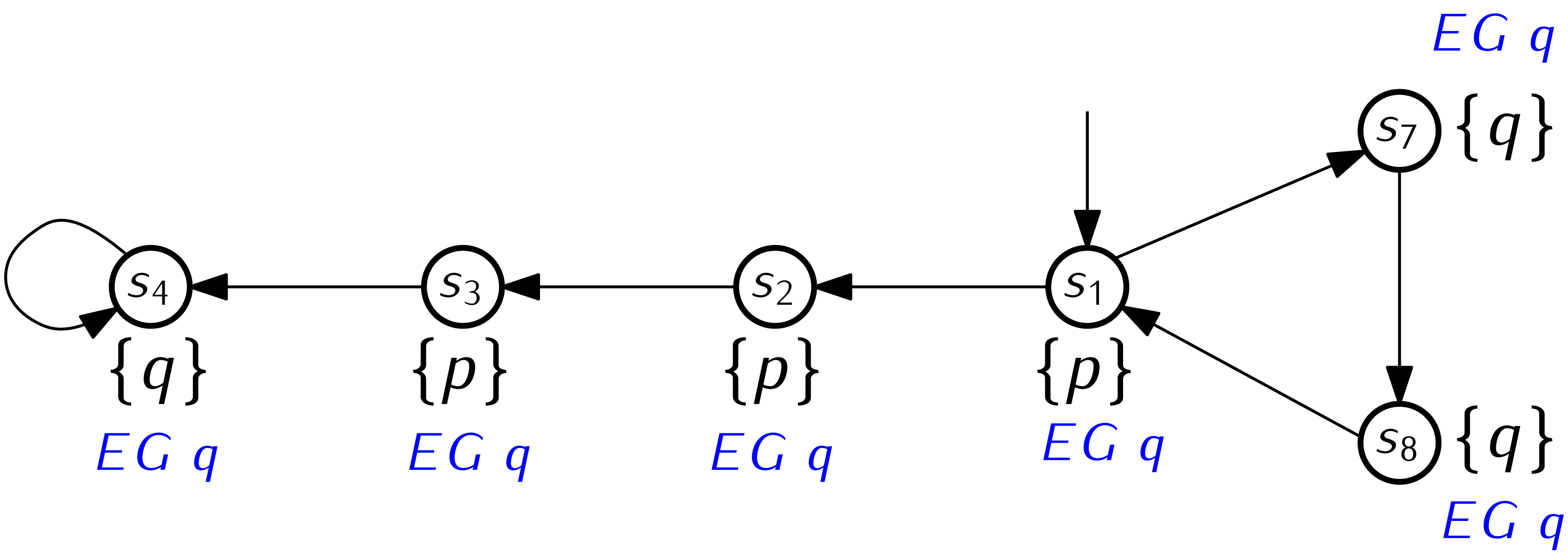
The General CTL Model Checking Algorithm

Example $p \; EU(\underline{EG \; q})$



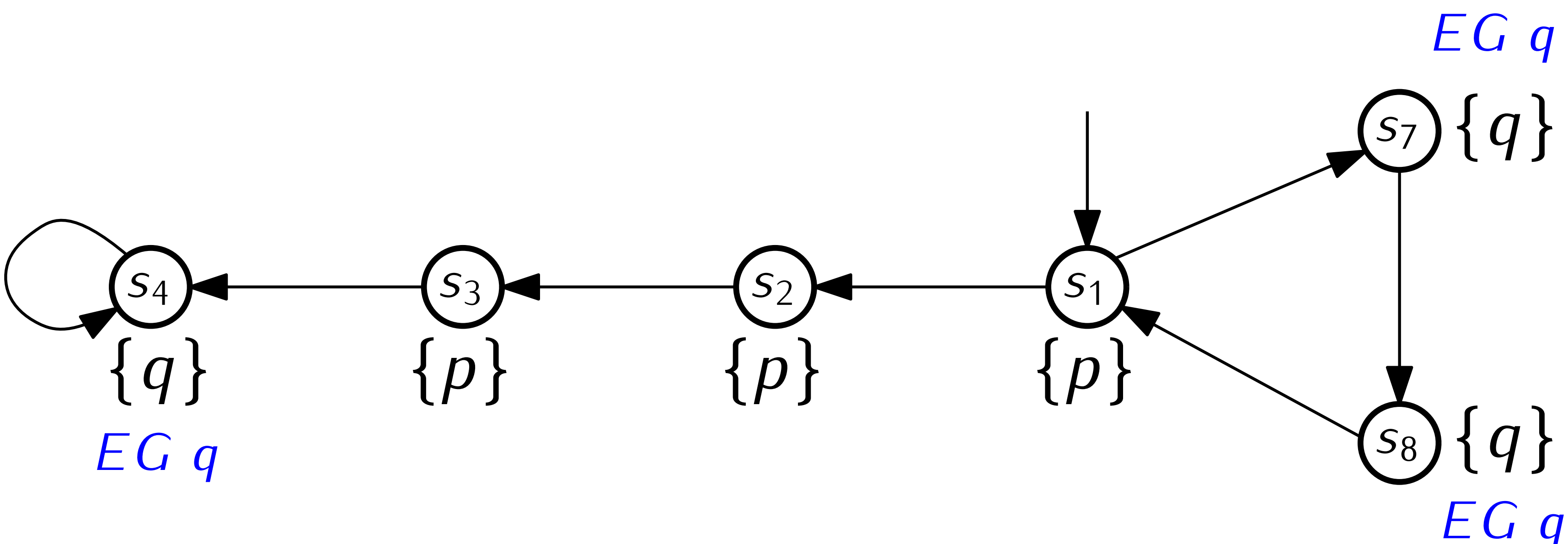
The General CTL Model Checking Algorithm

Example $p\ EU(\underline{EG\ q})$



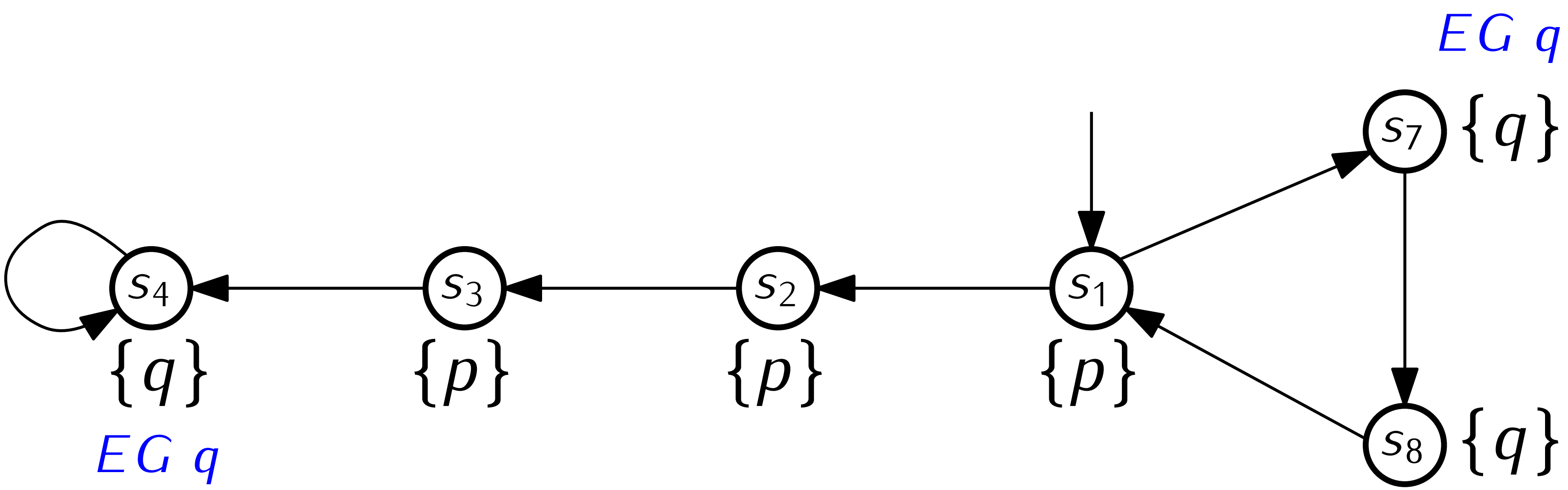
The General CTL Model Checking Algorithm

Example $p\ EU(\underline{EG\ q})$



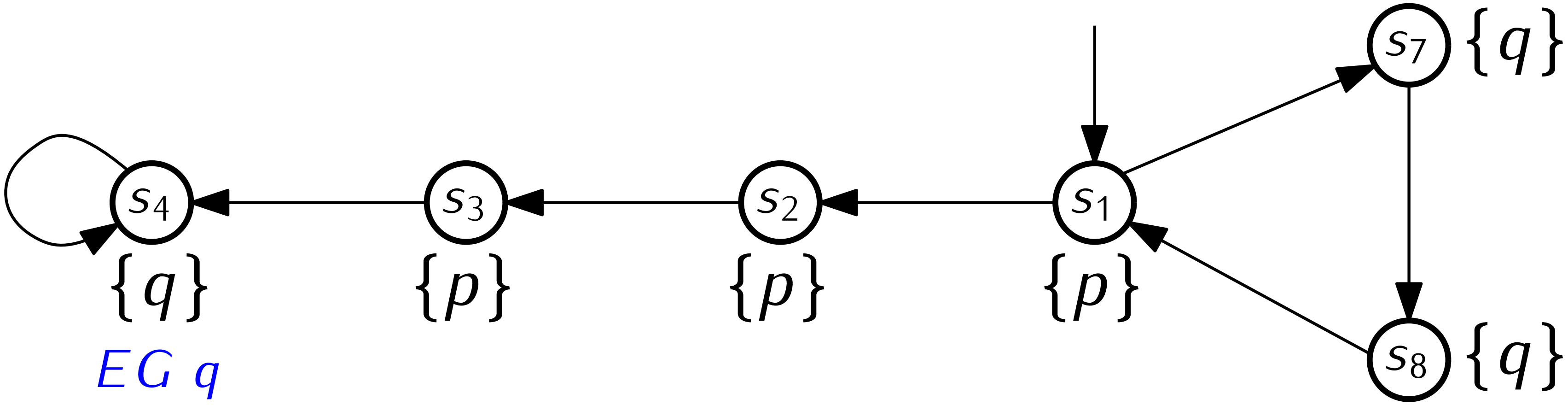
The General CTL Model Checking Algorithm

Example $p\ EU(\underline{EG\ q})$



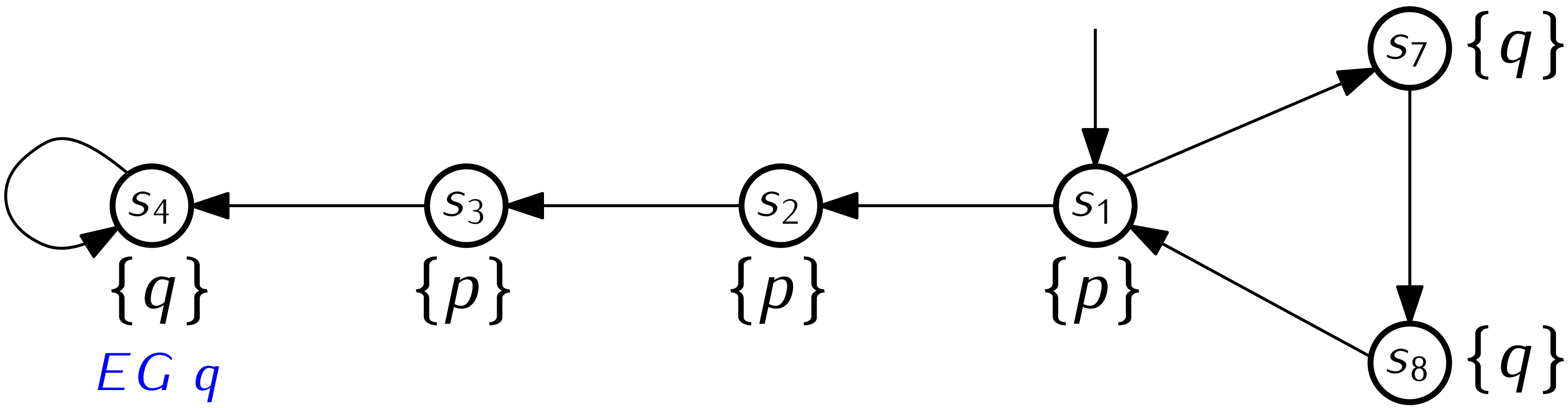
The General CTL Model Checking Algorithm

Example $p \; EU(\underline{EG \; q})$



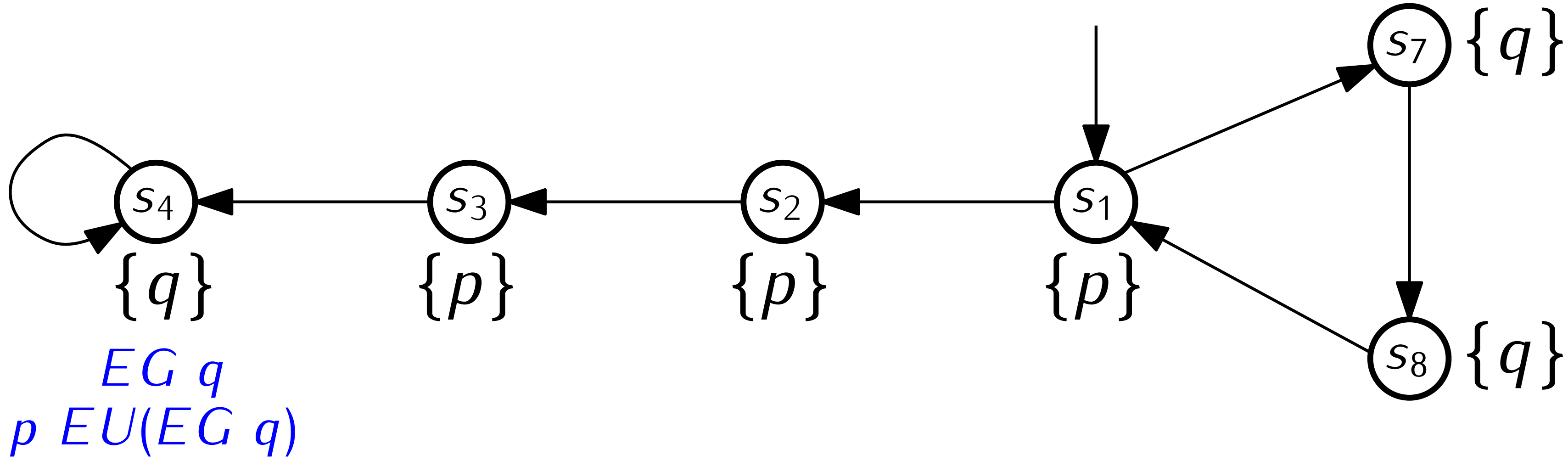
The General CTL Model Checking Algorithm

Example $\underline{p \; EU(EG \; q)}$



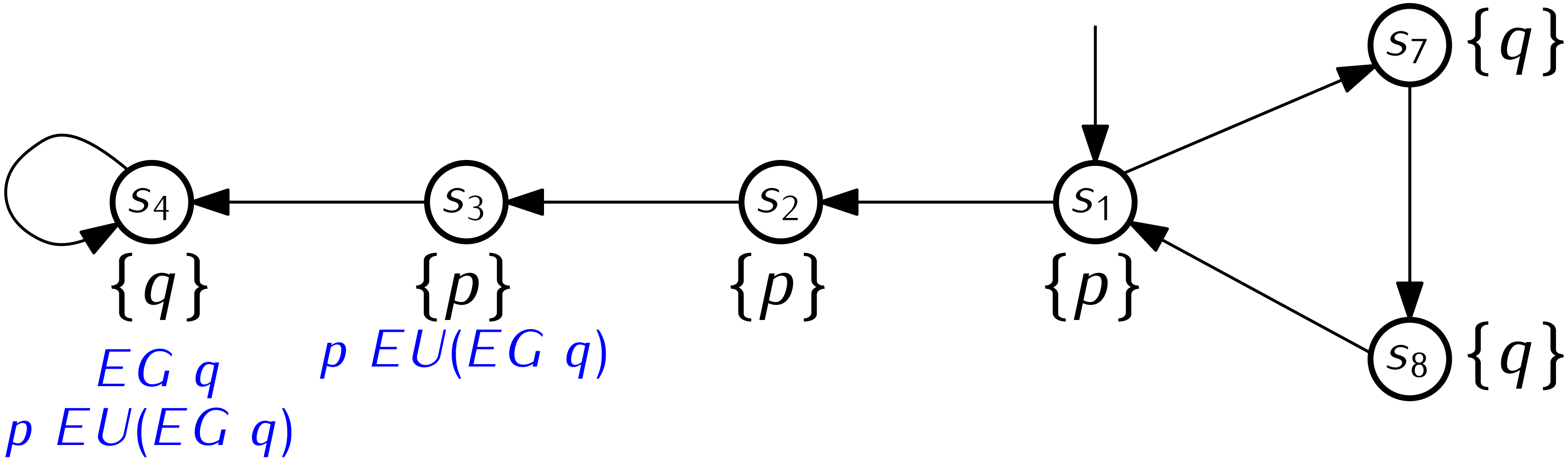
The General CTL Model Checking Algorithm

Example $p\ EU(EG\ q)$



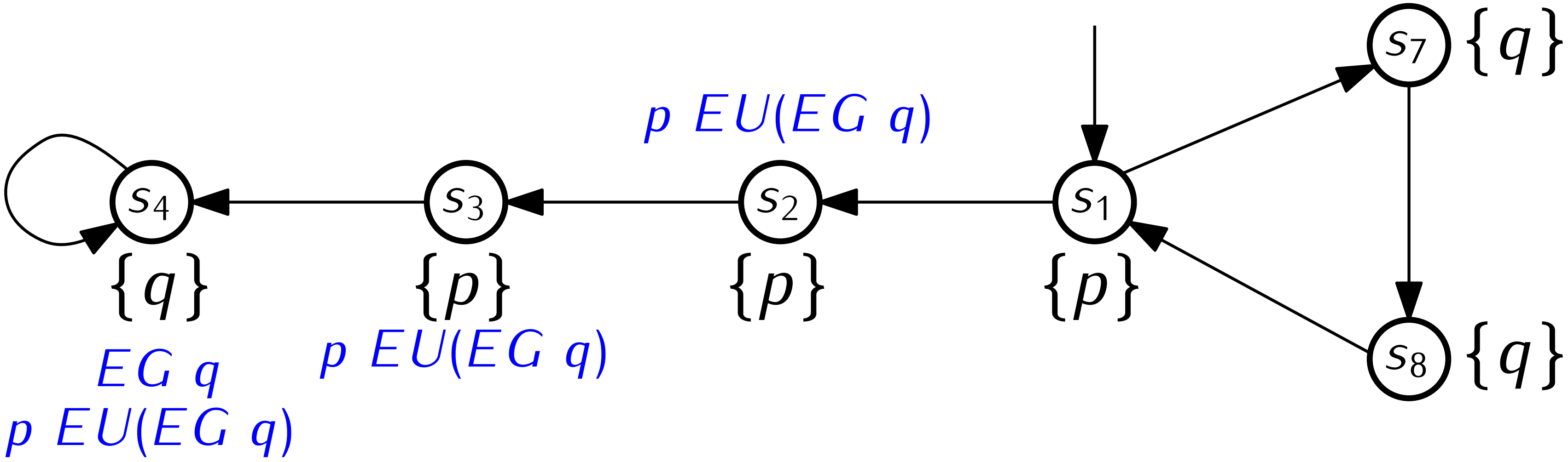
The General CTL Model Checking Algorithm

Example $p\ EU(EG\ q)$



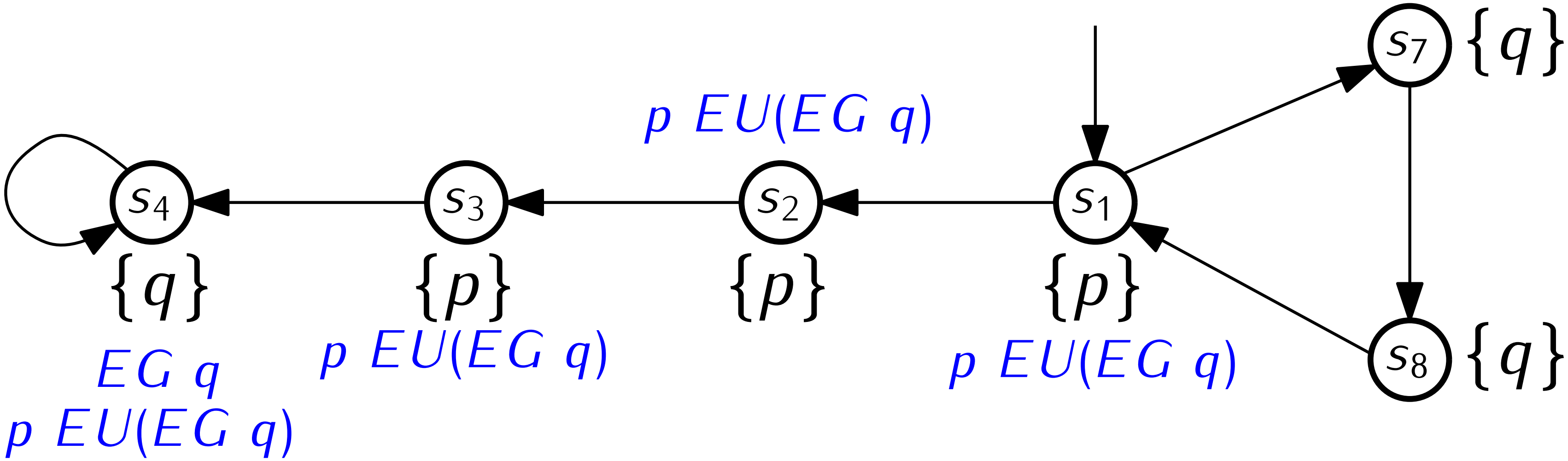
The General CTL Model Checking Algorithm

Example $\quad p\ EU(EG\ q)$



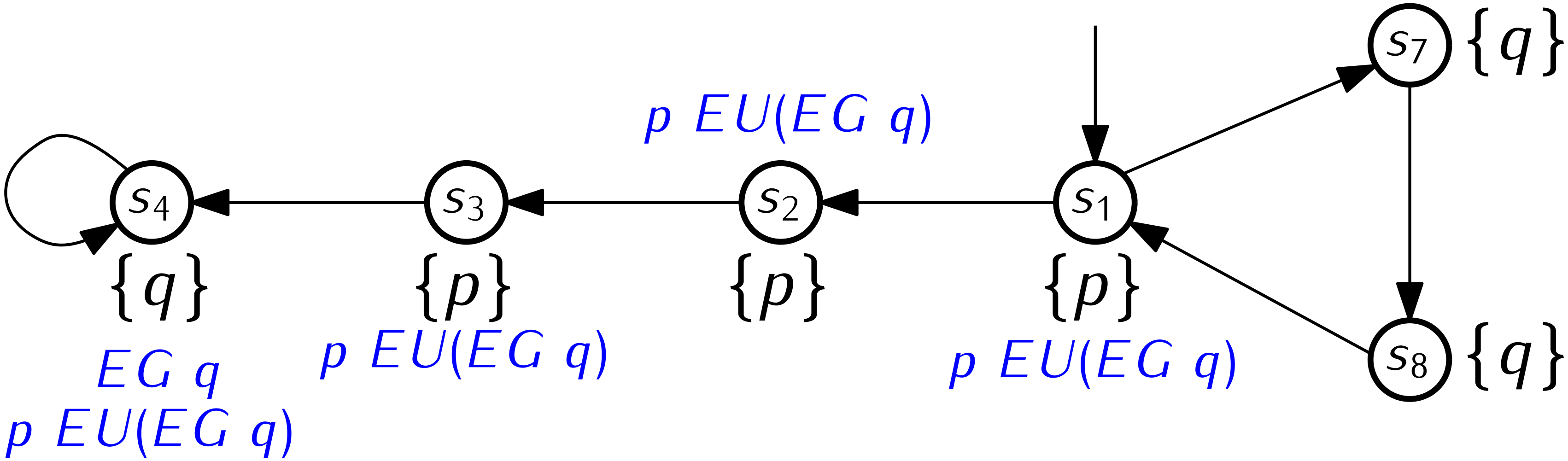
The General CTL Model Checking Algorithm

Example $p\ EU(EG\ q)$



The General CTL Model Checking Algorithm

Example $p\ EU(EG\ q)$



$$s_1, s_2, s_3, s_4 \models p\ EU(EG\ q)\ \wedge s_1 \in I\ \Rightarrow\ \mathcal{M} \models p\ EU(EG\ q)$$

The General CTL Model Checking Algorithm

$\mathsf{SAT}(\phi) =$

The General CTL Model Checking Algorithm

$\mathsf{SAT}(\phi) =$

case

The General CTL Model Checking Algorithm

$\text{SAT}(\phi) =$

case

ϕ is $\top$: return S

The General CTL Model Checking Algorithm

$\mathrm{SAT}(\phi) =$

case

ϕ is $\top$: return S

ϕ is p_i : return $\{s : p \in L(S)\}$

The General CTL Model Checking Algorithm

$\text{SAT}(\phi) =$

case

ϕ is $\top$: return S

ϕ is p_i : return $\{s : p \in L(S)\}$

ϕ is $\phi \land \psi$: return $\text{SAT}(\phi) \cap \text{SAT}(\psi)$

The General CTL Model Checking Algorithm

SAT(ϕ) $=$

case

ϕ is $\top$: return S

ϕ is p_i : return $\{s : p \in L(S)\}$

ϕ is $\phi \wedge \psi$: return SAT(ϕ)$\cap$ SAT(ψ)

ϕ is $\phi \wedge \psi$: return SAT(ϕ)$\cup$ SAT(ψ)

The General CTL Model Checking Algorithm

$\mathrm{SAT}(\phi) =$

case

ϕ is $\top$: return S

ϕ is p_i : return $\{s : p \in L(S)\}$

ϕ is $\phi \wedge \psi$: return $\mathrm{SAT}(\phi) \cap \mathrm{SAT}(\psi)$

ϕ is $\phi \wedge \psi$: return $\mathrm{SAT}(\phi) \cup \mathrm{SAT}(\psi)$

ϕ is $\neg \phi$: return $S - \mathrm{SAT}(\phi)$

The General CTL Model Checking Algorithm

SAT(ϕ) $=$

case

ϕ is $\top$: return S

ϕ is p_i : return $\{s : p \in L(S)\}$

ϕ is $\phi \wedge \psi$: return SAT(ϕ)$\cap$ SAT(ψ)

ϕ is $\phi \wedge \psi$: return SAT(ϕ)$\cup$ SAT(ψ)

ϕ is $\neg\phi$: return S – SAT(ϕ)

ϕ is $EX\phi$: return SAT$_{EX}$(ϕ)

The General CTL Model Checking Algorithm

SAT(ϕ) $=$

case

ϕ is $\top$: return S

ϕ is p_i : return $\{s : p \in L(S)\}$

ϕ is $\phi \wedge \psi$: return SAT(ϕ)$\cap$ SAT(ψ)

ϕ is $\phi \wedge \psi$: return SAT(ϕ)$\cup$ SAT(ψ)

ϕ is $\neg\phi$: return S – SAT(ϕ)

ϕ is $EX\phi$: return SAT$_{EX}$(ϕ)

ϕ is $EU\phi$: return SAT$_{EU}$(ϕ)

The General CTL Model Checking Algorithm

SAT(ϕ) =

case

ϕ is $\top$: return S

ϕ is p_i : return $\{s : p \in L(S)\}$

ϕ is $\phi \wedge \psi$: return SAT(ϕ)$\cap$ SAT(ψ)

ϕ is $\phi \wedge \psi$: return SAT(ϕ)$\cup$ SAT(ψ)

ϕ is $\neg\phi$: return S – SAT(ϕ)

ϕ is $EX\phi$: return SAT$_{EX}$(ϕ)

ϕ is $EU\phi$: return SAT$_{EU}$(ϕ)

ϕ is $EG\phi$: return SAT$_{EG}$(ϕ)

esac