Key points of this lecture

Global model checking algorithm, recursive on state sub-formulas (i.e. bottom-up on the parse tree).

Satisfaction set characterisations of CTL formulas in existential normal form.

Dedicated algorithms for CTL formulas in existential normal form, which exploit the satisfaction set characterisations.

The complexity of CTL model checking is polynomial in the size of the transition system and the CTL formula.

Lecture 03 - Model Checking CTL

- Formal semantics and naïve algorithms
- Existential normal form CTL (ECTL)
- Global model checking algorithm
- Satisfaction sets characterisation of ECTL
- MC algorithms for basic cases
- MC algorithms for EU
- MC algorithms for EG
- Overall complexity

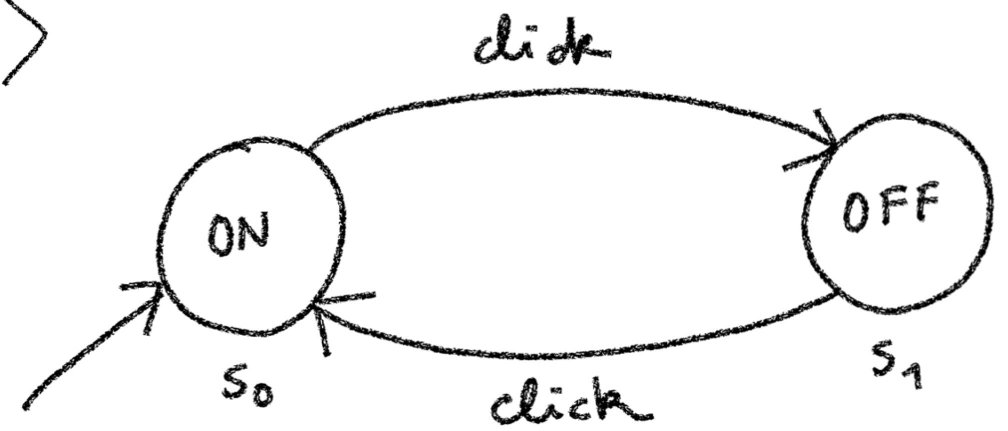
Transition Systems

Def. A transition system is a tuple

$$\langle S, A, \rightarrow, L, AP, I \rangle$$

such that

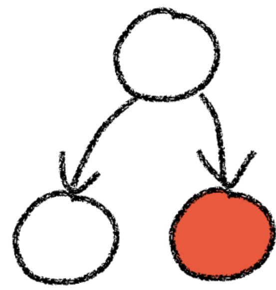
- S is a set of states
- A is a set of "Actions"
- $\rightarrow \subseteq S \times A \times S$ is a set of transitions
- $L : S \longrightarrow 2^{AP}$ is a labelling function
- AP is a finite state of "Atomic Propositions"
- $I \subseteq S$ is the set of initial states



Intuition of "next" operator

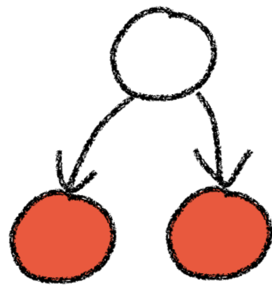
$\exists \circ \phi$

"in the next state ϕ holds"



$\forall \circ \phi$

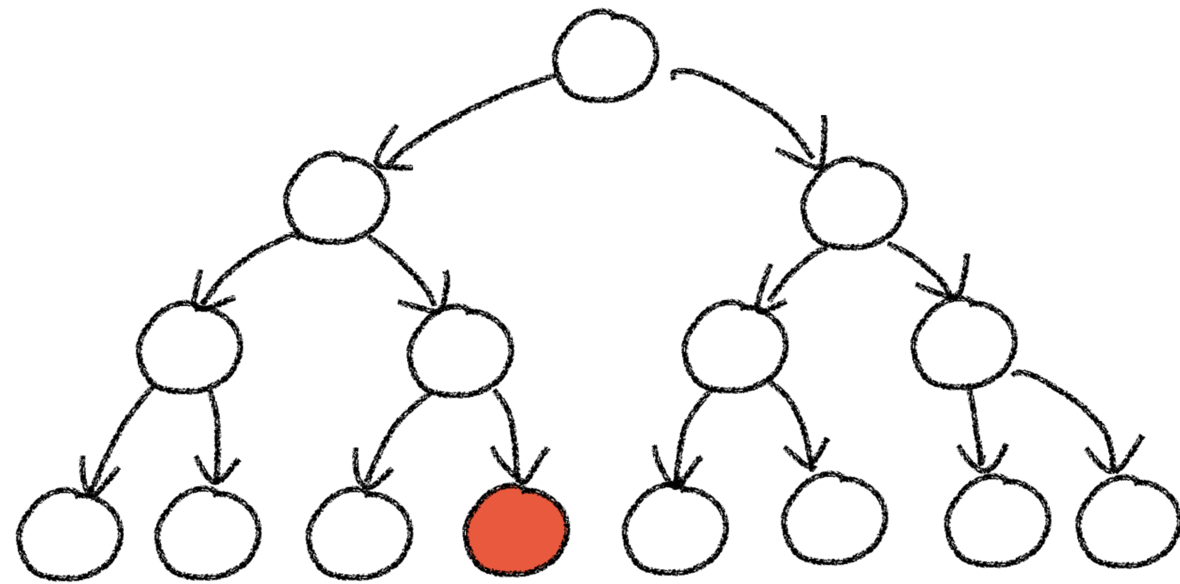
"in all next states ϕ holds"



Intuition of “eventually” and “always” operators

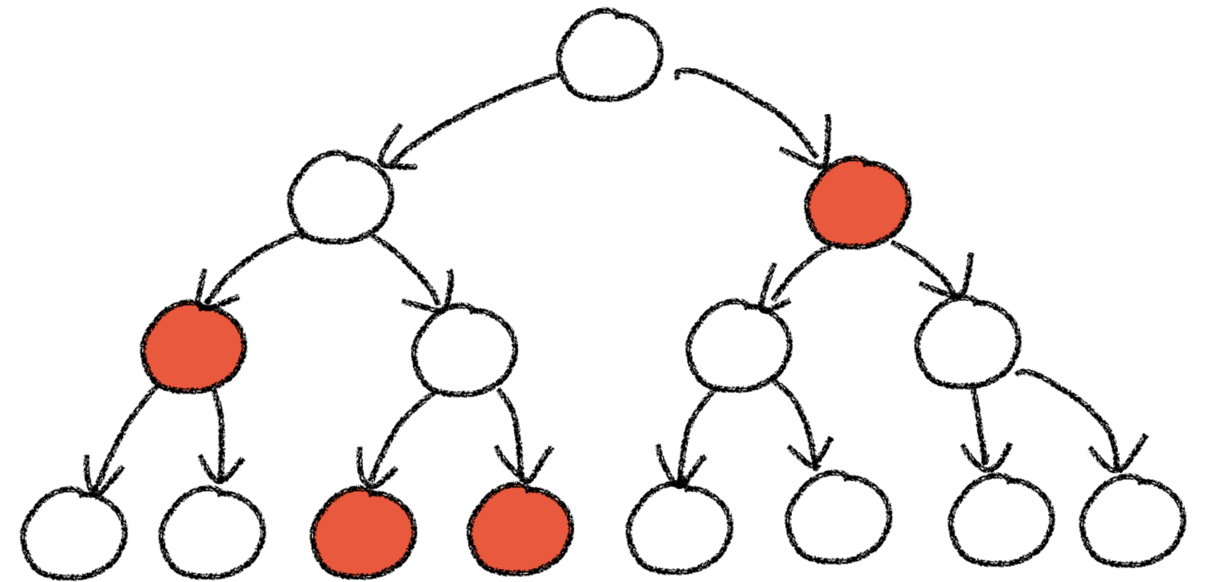
$\exists \Diamond \phi$

"ϕ holds potentially"



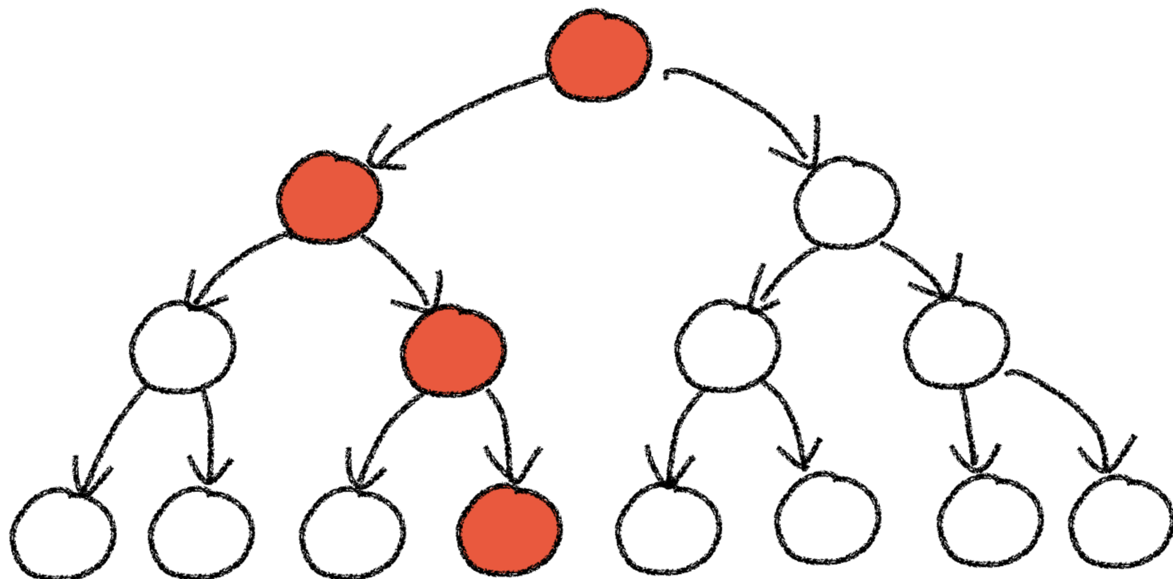
$\forall \Diamond \phi$

"ϕ is inevitable"



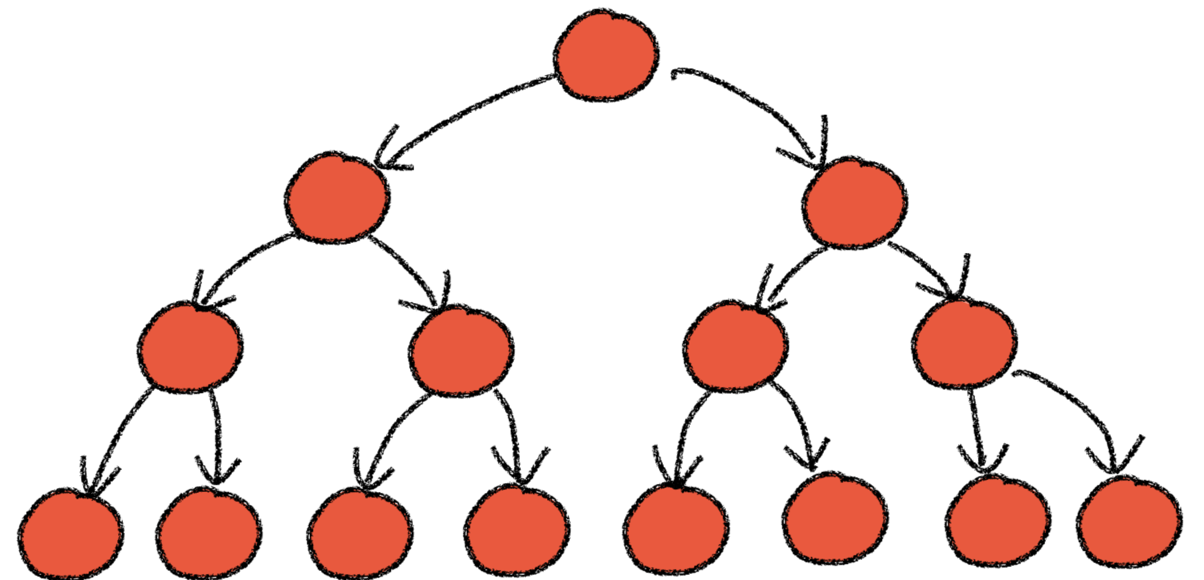
$\exists \Box \phi$

"potentially always ϕ"



$\forall \Box \phi$

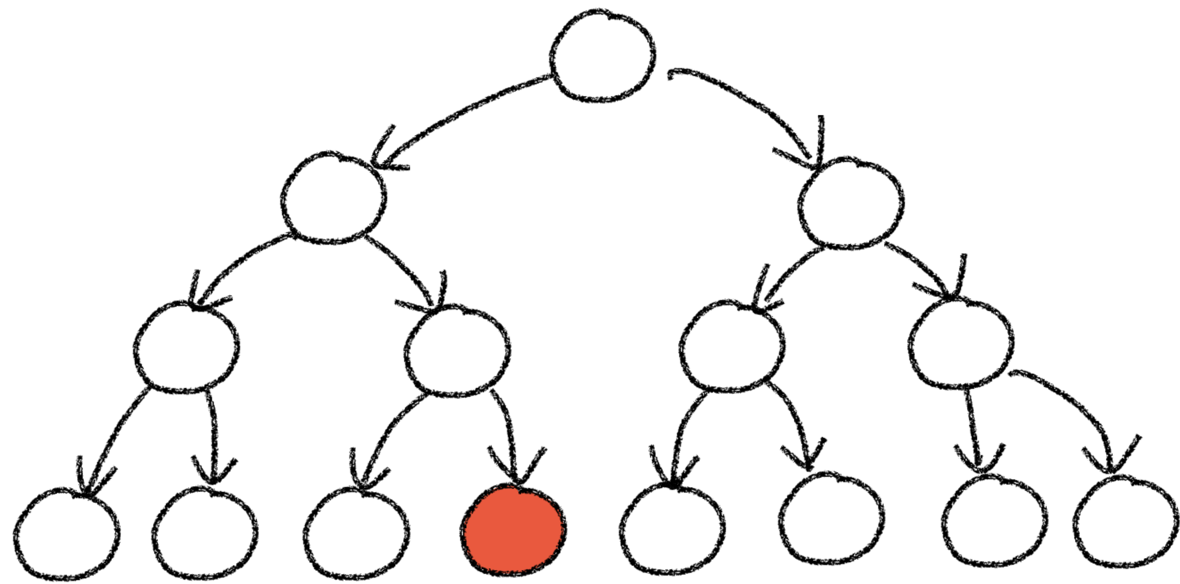
"Globally always ϕ"



Intuition of the "Until" operator

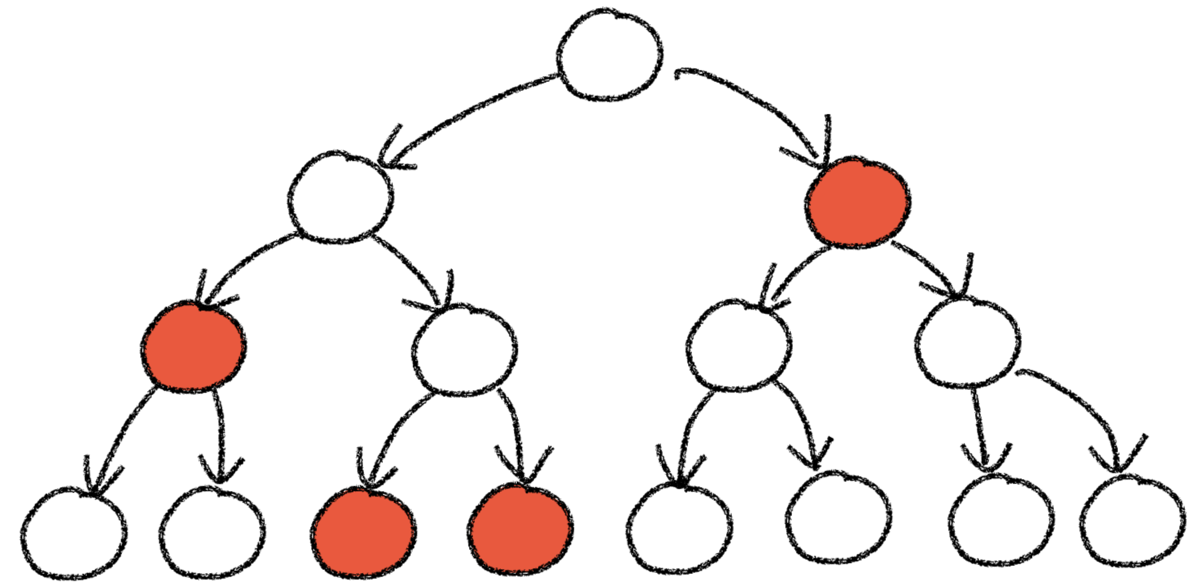
$\exists \Diamond \phi$

"ϕ holds potentially"



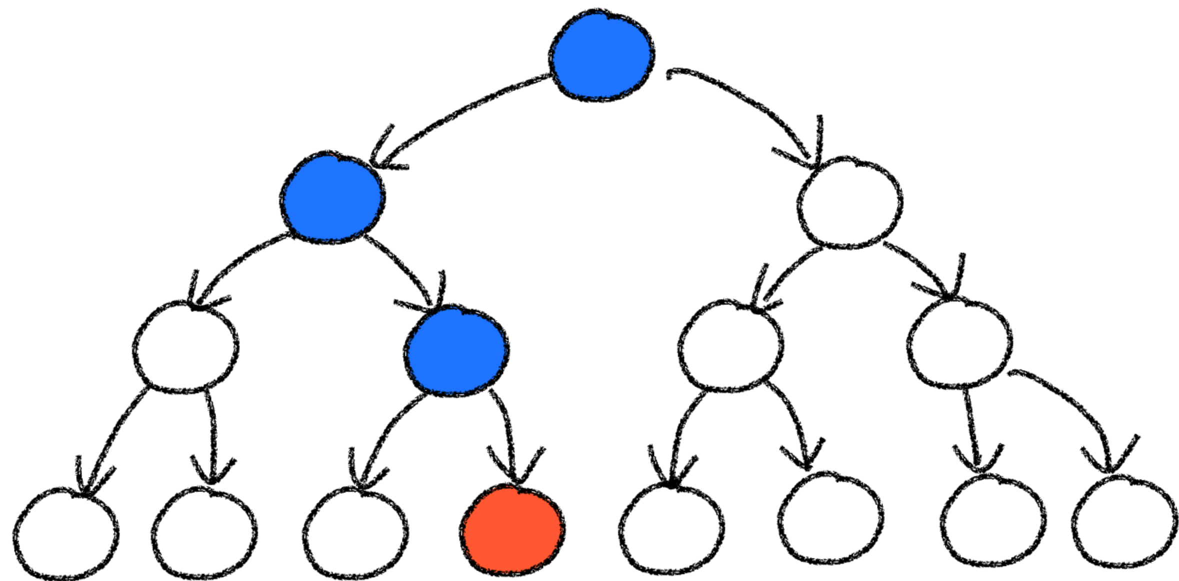
$\forall \Diamond \phi$

"ϕ is inevitable"



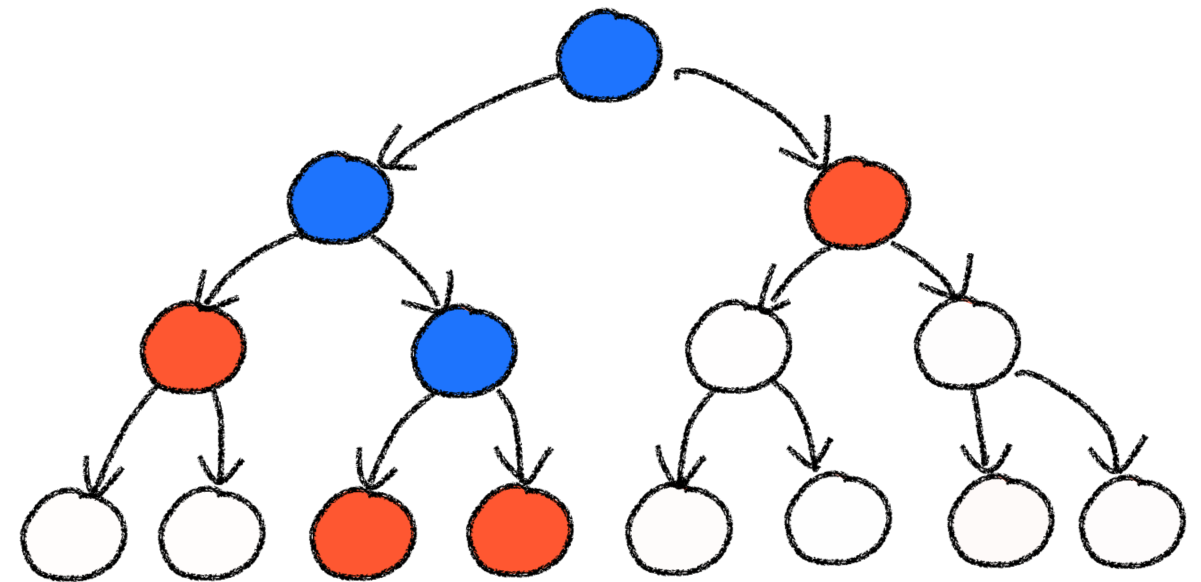
$\exists\, \phi_1 \cup \phi_2$

"there is a path s.t. $\phi_1 \cup \phi_2$ holds"



$\forall\, \phi_1 \cup \phi_2$

"in <u>all</u> paths $\phi_1 \cup \phi_2$ holds"



CTL - formal semantics

We define the formal semantics of CTL as 2 relations.

A relation between states and state formulas

$s \vDash true$ (holds always)

$s \vDash p$ iff iff $p \in L(s)$

$s \vDash \neg\phi$ iff $s \nvDash \phi$

$s \vDash \phi_1 \wedge \phi_2$ iff $s \vDash \phi_1$ and $s \vDash \phi_2$

$s \vDash \exists\psi$ iff $\exists\pi \in Paths(s) \,.\, \pi \vDash \psi$

$s \vDash \forall\psi$ iff $\forall\pi \in Paths(s) \,.\, \pi \vDash \psi$

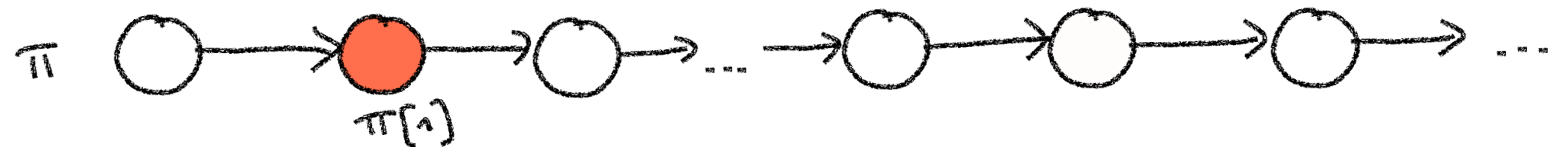
and a relation between paths and path formulas (next slide)

$\pi \vDash \psi$ iff …

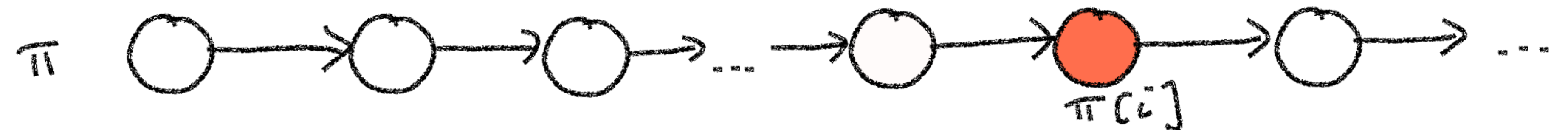
Semantics of path formulas

And here is the formal semantics of path formulas

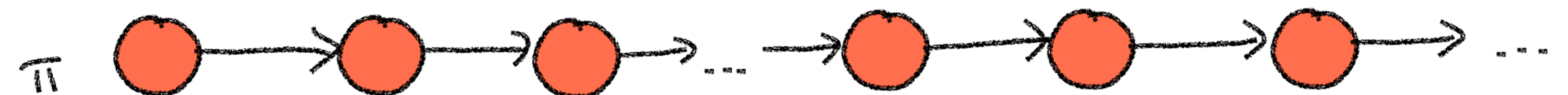
$\pi \models \bigcirc \phi$ iff $\pi[1] \models \phi$



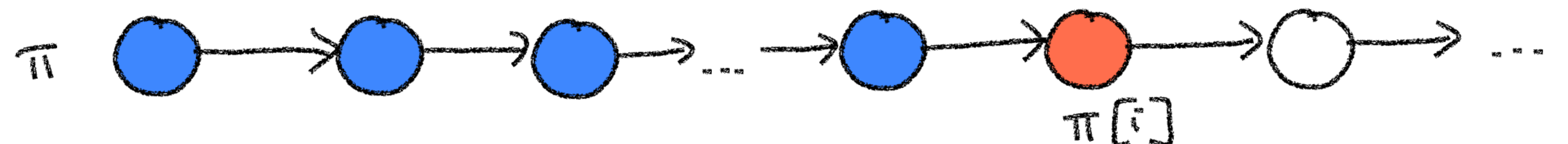
$\pi \models \Diamond \phi$ iff $\exists i \in \mathbb{N} \,.\, \pi[i] \models \phi$



$\pi \models \square \phi$ iff $\forall i \in \mathbb{N} \,.\, \pi[i] \models \phi$



$\pi \models \phi_1 \mathsf{U} \phi_2$ iff $\exists i \in \mathbb{N} \,.\, \pi[i] \models \phi_2 \wedge \forall 0 \leq j < i \,.\, \pi[j] \models \phi_1$



Naïve algorithms for state formulas

$s \models true$

$s \models p$ iff iff $p \in L(s)$

$s \models \neg\phi$ iff $s \not\models \phi$

$s \models \phi_1 \wedge \phi_2$ iff $s \models \phi_1$ and $s \models \phi_2$

$s \models \exists\phi$ iff $\exists \pi \in Paths(s)\,.\,\pi \models \psi$

$s \models \forall\phi$ iff $\forall \pi \in Paths(s)\,.\,\pi \models \psi$

easy programming task

```
modelCheck(s, φ) =
    for each π in Paths(s) do
        if modelCheck(π, ψ) return true;
    end
    return false;
```

Naïve algorithms for path formulas

$\pi \vDash \bigcirc \phi$ iff $\pi[1] \vDash \phi$

↳ modelCheck(π, ○ϕ) = return modelCheck(π[1], ϕ)

$\pi \vDash \Diamond \phi$ iff $\exists i \in \mathbb{N} \,.\, \pi[i] \vDash \phi$

↳ modelCheck(π, ◇ϕ) = for each state in π…

$\pi \vDash \Box \phi$ iff $\forall i \in \mathbb{N} \,.\, \pi[i] \vDash \phi$

↳ modelCheck(π, □ϕ) = for each state in π…

$\pi \vDash \phi_1 \mathsf{U} \phi_2$ iff $\exists i \in \mathbb{N} \,.\, \pi[i] \vDash \phi_2 \wedge \forall 0 \leq j < i \,.\, \pi[j] \vDash \phi_1$

↳ modelCheck(π, ϕ₁Uϕ₂) = for each state in π…

Lecture 03 - Model Checking CTL

- Formal semantics and naïve algorithms
- Existential normal form CTL (ECTL)
- Global model checking algorithm
- Satisfaction sets characterisation of ECTL
- MC algorithms for basic cases
- MC algorithms for EU
- MC algorithms for EG
- Overall complexity

CTL grammars recall

We can get rid of the “always” and “eventually” operators

$$\phi ::= true \mid p \mid \neg\phi \mid \phi_1 \vee \phi_2 \mid \exists\psi \mid \forall\psi$$

$$\psi ::= \bigcirc\phi \mid \Diamond\phi \mid \Box\phi \mid \phi_1 \cup \phi_2$$

Alternatively, we can get rid of the universal quantifier

$$\phi ::= true \mid p \mid \neg\phi \mid \phi_1 \vee \phi_2 \mid \exists\psi \mid \forall\psi$$

$$\psi ::= \bigcirc\phi \mid \Diamond\phi \mid \Box\phi \mid \phi_1 \cup \phi_2$$

The above grammar yields CTL formulas in so-called “existential normal form” (we shall consider it when we will see the model-checking algorithms)

Existential normal form (ECTL)

The grammar for ECTL as we saw it in Lecture 02

$$\phi ::= true \mid p \mid \neg\phi \mid \phi_1 \vee \phi_2 \mid \exists\psi$$

$$\psi ::= \bigcirc\phi \mid \Box\phi \mid \phi_1 \mathsf{U} \phi_2$$

Equivalent grammar with just state formulas

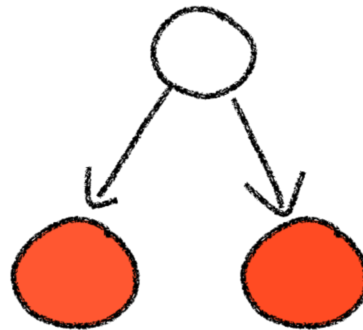
$$\phi ::= true \mid p \mid \neg\phi \mid \phi_1 \vee \phi_2 \mid \exists\bigcirc\psi \mid \exists\Box\phi \mid \exists\phi_1 \mathsf{U} \phi_2$$

NOTE: we focus on this grammar to reduce the number of algorithms, but dedicated algorithms for all CTL operators can be designed.

From CTL to ECTL

We can go from CTL to ECTL using these equivalences

$$\forall \bigcirc \phi \equiv \neg \exists \bigcirc \neg \phi$$



$$\forall(\phi_1 \mathsf{U} \phi_2) \equiv \neg \exists((\neg \phi_2)\mathsf{U}(\neg \phi_1 \wedge \neg \phi_2)) \wedge \neg \exists \square \neg \phi_2$$

"you cannot reach a state that satisfies neither ϕ_1 nor ϕ_2 without traversing a state that satisfies ϕ_2"

"ϕ_2 is inevitable"

Lecture 03 - Model Checking CTL

- Formal semantics and naïve algorithms
- Existential normal form CTL (ECTL)
- Global model checking algorithm
- Satisfaction sets characterisation of ECTL
- MC algorithms for basic cases
- MC algorithms for EU
- MC algorithms for EG
- Overall complexity

Semantics over a TS - recall

We say that a transition system satisfies a formula if all its initial states satisfy the formula:

$$T \models \phi \text{ iff } \forall s \in I \,.\, s \models \phi$$

or, equivalently,

$$T \models \phi \text{ iff } I \subseteq sat(\phi)$$

We use the latter in our algorithm

```
modelCheck(TS,phi) = {
   return I ⊆ sat(phi);
}
```

Satisfaction sets - recall

We define the satisfaction set of a CTL state formula as the set of states that satisfy the formula

$$sat(\phi) = \{s \mid s \vDash \phi\}$$

Main algorithm

Now that we have our main algorithm

```
modelCheck(TS,phi) = {
  return I ⊆ sat(phi);
}
```

All we need to do is to implement function **`sat(…)`**

```
sat(true) = …
sat(not phi) = …
sat(phi1 or phi2) = …
sat(EX phi) = …
sat(EG phi) = …
sat(E(phi1 U phi2) = …
```

Recursion

We will see that **`sat(…)`** is recursive

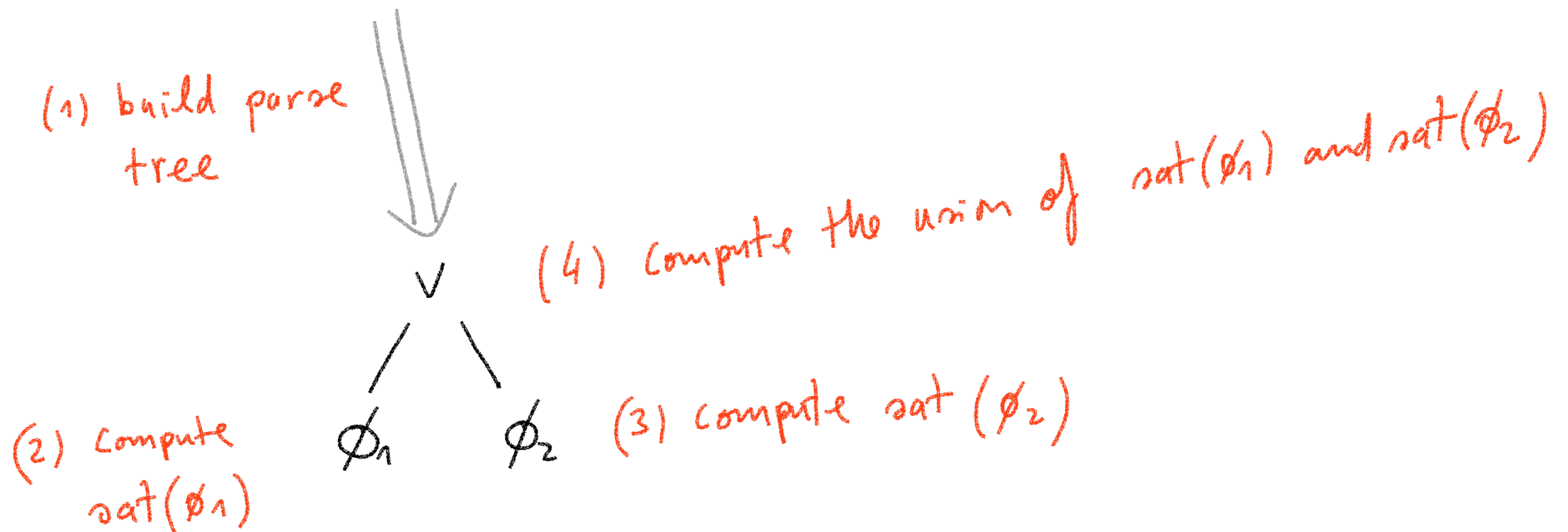
```
sat(true) = …
sat(not phi) = S \ sat(phi)
sat(phi1 or phi2) = sat(phi1) U sat(phi2)
sat(EX phi) = …
sat(EG phi) = …
sat(E(phi1 U phi2) = …
```

Hence, invoking **`sat(phi)`** will recursively compute the satisfaction sets for all sub-formulas of **`phi`**

You can think of satisfaction sets being computed bottom-up on the parse tree of **`phi`**

Bottom-up computation

You can think of satisfaction sets being computed bottom-up on the parse tree of the formula



Memoisation

We can apply memoisation to avoid recomputing twice the same satisfaction sets.

This can happen in general, but especially in the transformation from CTL to ECTL:

$$\forall(\phi_1 \mathsf{U} \phi_2) \equiv \neg\exists((\neg\phi_2)\mathsf{U}(\neg\phi_1 \wedge \neg\phi_2)) \wedge \neg\exists\Box\neg\phi_2$$

It is also convenient when doing model checking, especially by hand.

One way to do it: Once *sat(phi)* is computed,

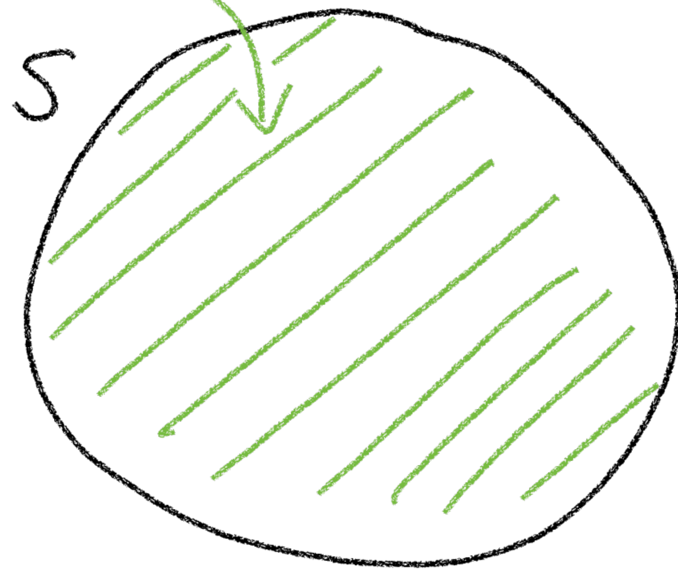
1. create a new atomic proposition *p_phi*
2. label every state in *sat(phi)* with *p_phi*
3. replace every occurrence of *phi* by *p*

Lecture 03 - Model Checking CTL

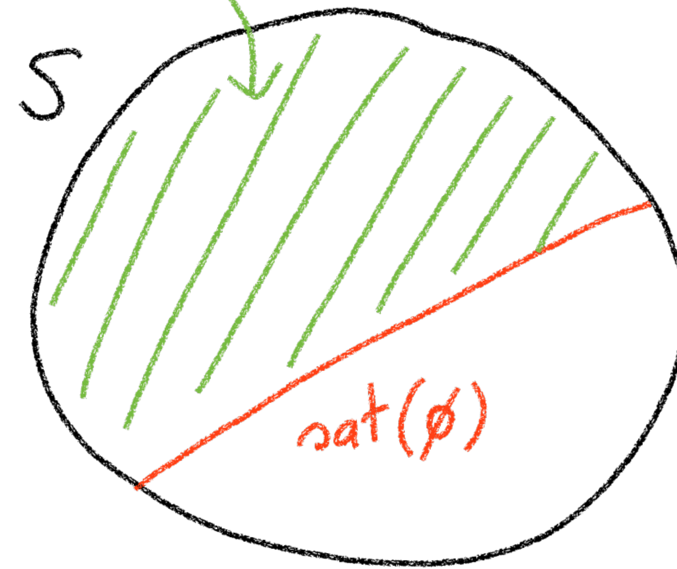
- Formal semantics and naïve algorithms
- Existential normal form CTL (ECTL)
- Global model checking algorithm
- Satisfaction sets characterisation of ECTL
- MC algorithms for basic cases
- MC algorithms for EU
- MC algorithms for EG
- Overall complexity

Satisfaction sets characterisation of propositional fragment

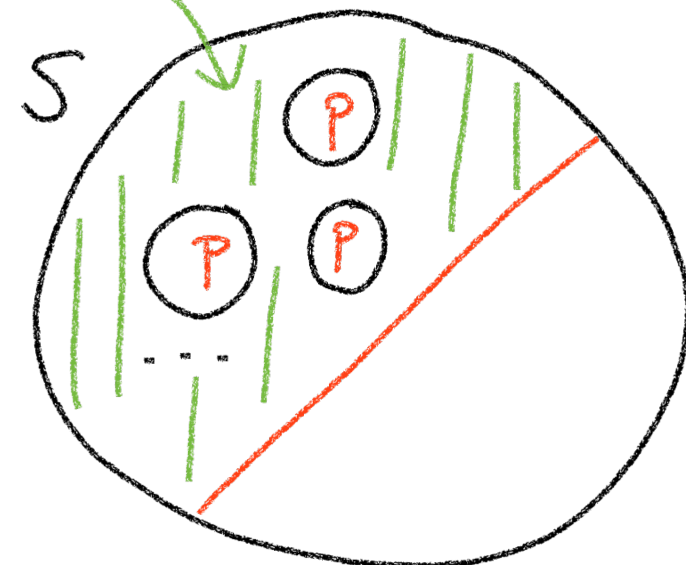
$sat(true) = S$



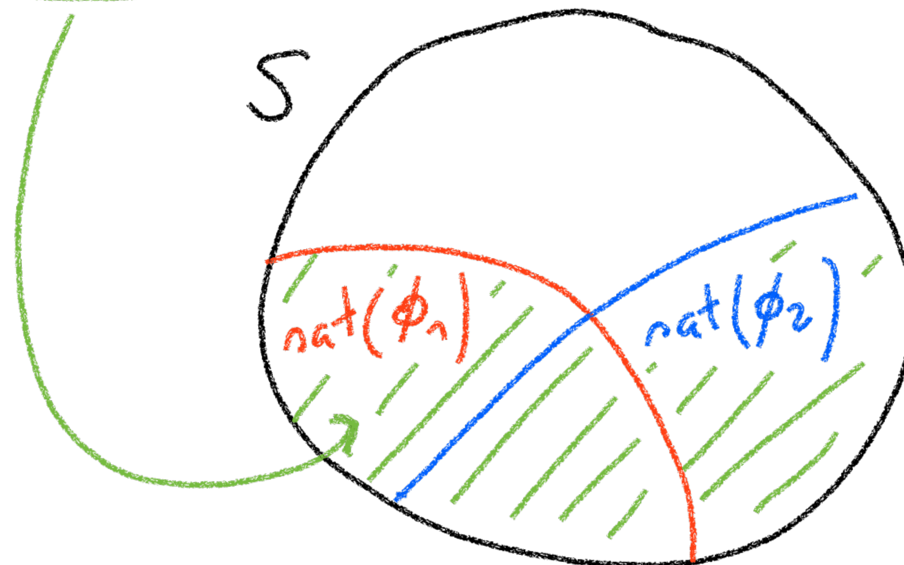
$sat(\neg \phi) = S \backslash sat(\phi)$



$sat(p) = \{ s \in S \mid p \in L(s) \}$

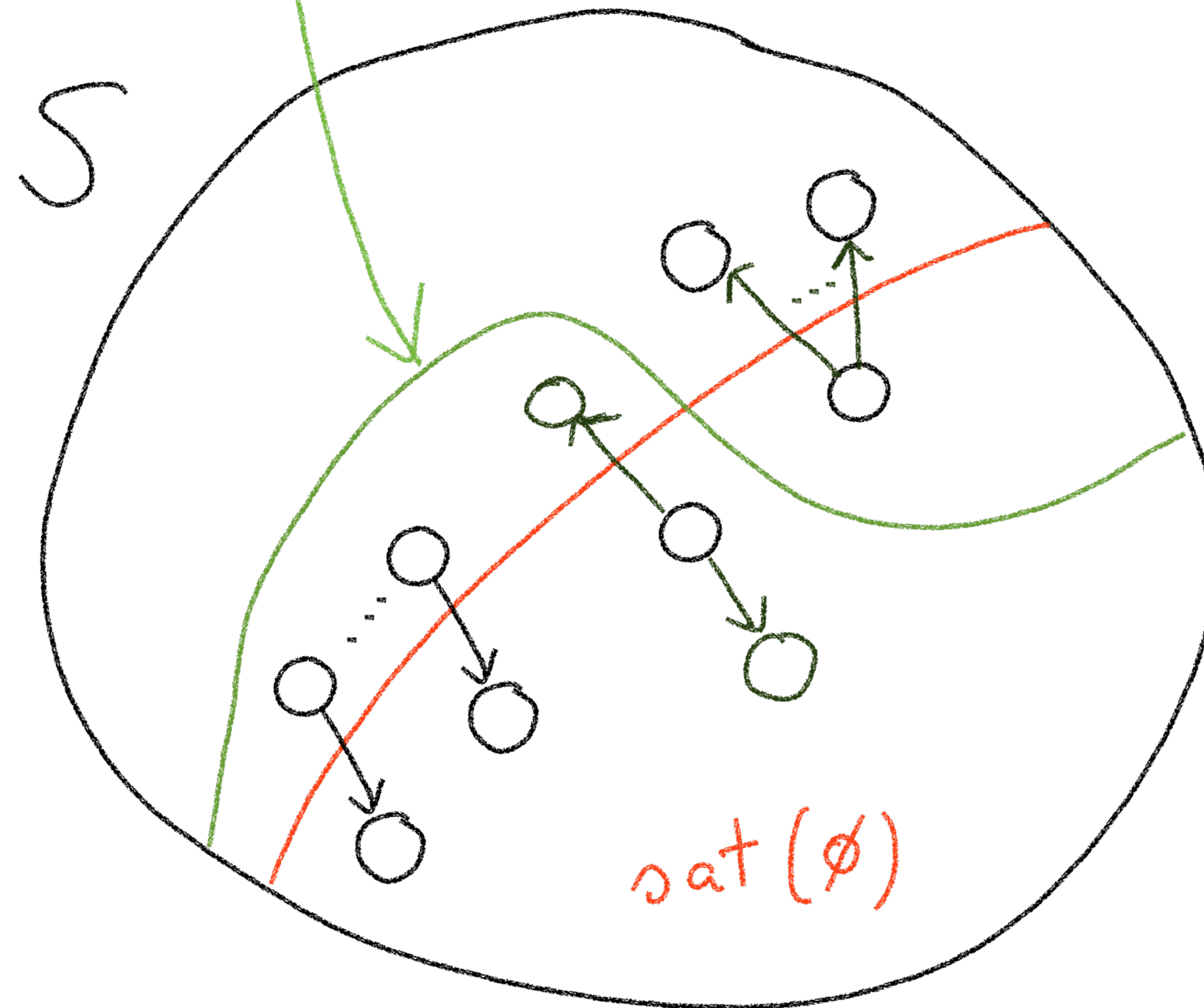


$sat(\phi_1 \vee \phi_2) = sat(\phi_1) \cup sat(\phi_2)$



Satisfaction sets characterisation for EX

$$sat(\exists \bigcirc \phi) = \{s \in S \mid Post(s) \cap \mathrm{sat}(\phi) \neq \varnothing\}$$



Satisfaction sets characterisation for EG and EU

Not so trivial, we will see them in detail in a while…

Lecture 03 - Model Checking CTL

- Formal semantics and naïve algorithms
- Existential normal form CTL (ECTL)
- Global model checking algorithm
- Satisfaction sets characterisation of ECTL
- MC algorithms for basic cases
- MC algorithms for EU
- MC algorithms for EG
- Overall complexity

Algorithms for basic cases

So far we have seen the following basic cases

$$sat(true) = S$$

$$sat(\neg\phi) = S \backslash sat(\phi)$$

$$sat(p) = \{s \in S \mid p \in L(s)\}$$

$$sat(\phi_1 \vee \phi_2) = sat(\phi_1) \cup sat(\phi_2)$$

$$sat(\exists \bigcirc \phi) = \{s \in S \mid Post(s) \cap \mathrm{sat}(\phi) \neq \varnothing\}$$

These cases can be easily implemented with a suitable representation of states and transitions, amenable for set operations (complement, union, etc.)

NOTE: A popular approach is to use BDDs, which we are not going to cover in this course.

Lecture 03 - Model Checking CTL

- Formal semantics and naïve algorithms
- Existential normal form CTL (ECTL)
- Global model checking algorithm
- Satisfaction sets characterisation of ECTL
- MC algorithms for basic cases
- MC algorithms for EU
- MC algorithms for EG
- Overall complexity

Let's start simply with EF

Before we see the algorithm of EU, let us first look at the simplest case of EF.

Remember: $\exists \Diamond \phi \equiv \mathit{true} \; \mathsf{U} \; \phi$

You may also remember this expansion law:

$$\exists \Diamond \phi \equiv \phi \vee \exists \bigcirc \exists \Diamond \phi$$

Expansion law for EF

The expansion law for EF

$$\exists\Diamond\phi \equiv \phi \vee \exists\bigcirc\exists\Diamond\phi$$

Provides a recursive definition definition of *sat*

$$\begin{aligned} sat(\exists\Diamond\phi) &\equiv sat(\phi \vee \exists\bigcirc\exists\Diamond\phi) \\ &\equiv sat(\phi) \cup sat(\exists\bigcirc\exists\Diamond\phi) \\ &\equiv sat(\phi) \cup \{s \in S \mid Post(s) \cap sat(\exists\Diamond\phi) \neq \emptyset\} \end{aligned}$$

A fix point computation

From our recursive definition

$$sat(\exists\Diamond\phi) \equiv sat(\phi) \cup \{s \in S \mid Post(s) \cap sat(\exists\Diamond\phi) \neq \emptyset\}$$

We know that we look for a set T such that

$$(1) \quad T \supseteq sat(\phi)$$

$$(2) \quad Post(s) \cap T \neq \emptyset \Rightarrow s \in T$$

T = S

(1) ✓

(2) ✓

T = ∅

It can be shown that $sat(\exists\Diamond\phi)$

is the ~~smallest~~ largest set T satisfying (1) and (2)

A fix point computation

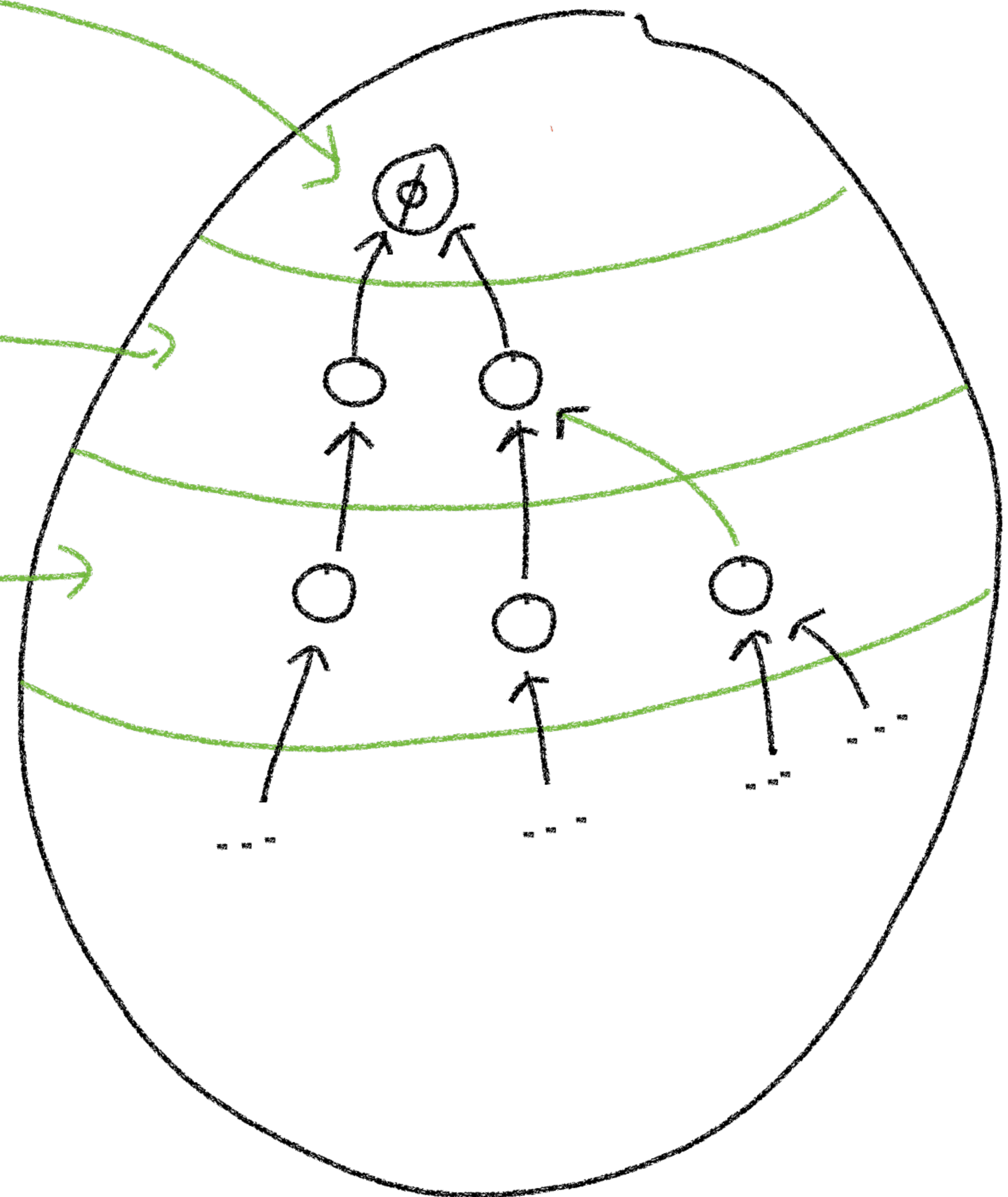
We can then just perform a standard least fix point calculation

$T_0 := sat(\phi)$

$T_1 := T_0 \cup \{s \in S \setminus T_0 \mid Post(s) \cap T_0 \neq \varnothing\}$

$T_2 := T_1 \cup \{s \in S \setminus T_1 \mid Post(s) \cap T_1 \neq \varnothing\}$

…



Algorithm for EF

Finally, the algorithm…

$T := sat(\phi);$ // the set of states to be returned as a result

$W := sat(\phi);$ // working set (states to be explored)

while $W \neq \emptyset$ do

  remove some s from W;

  foreach $s' \in Pre(s)$ do

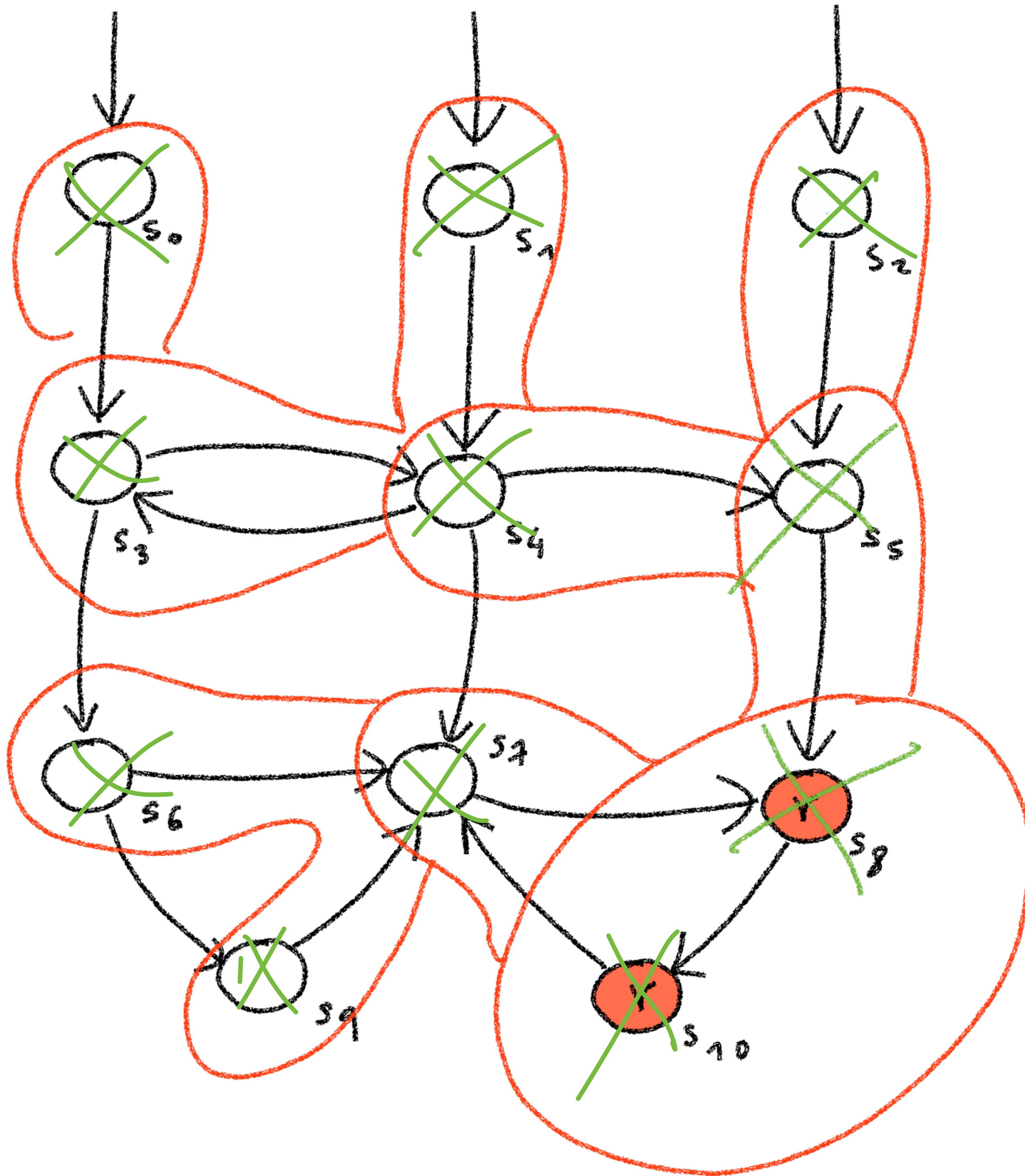
    if $s' \notin T$ then

      $T := T \cup s';$

      $W := W \cup s';$

return T;

Example



$\exists \Diamond r$

T

Let us now extend what we saw to the general case of the Until operator

Expansion law for EU

The expansion law for EU

$$\exists \phi_1 \mathsf{U} \phi_2 \;\equiv\; \phi_2 \vee (\phi_1 \wedge \exists \bigcirc \exists \phi_1 \mathsf{U} \phi_2)$$

Provides a recursive definition definition of *sat*

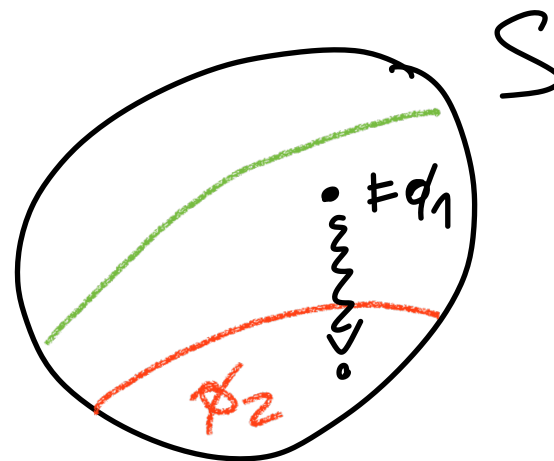
$$\begin{aligned}
sat(\exists \phi_1 \mathsf{U} \phi_2) &\equiv sat(\phi_2 \vee (\phi_1 \wedge \exists \bigcirc \exists \phi_1 \mathsf{U} \phi_2)) \\
&\equiv sat(\phi_2) \cup sat((\phi_1 \wedge \exists \bigcirc \exists \phi_1 \mathsf{U} \phi_2)) \\
&\equiv sat(\phi_2) \cup (sat(\phi_1) \cap sat(\exists \bigcirc \exists \phi_1 \mathsf{U} \phi_2)) \\
&\equiv sat(\phi_2) \cup (sat(\phi_1) \cap \{s \in S \mid Post(s) \cap sat(\exists \phi_1 \mathsf{U} \phi_2) \neq \varnothing\}) \\
&\equiv sat(\phi_2) \cup \{s \in sat(\phi_1) \mid Post(s) \cap sat(\exists \phi_1 \mathsf{U} \phi_2) \neq \varnothing\}
\end{aligned}$$

A fix point computation

From our recursive definition

$$sat(\exists\phi_1 \mathsf{U} \phi_2) \equiv sat(\phi_2) \cup \{s \in sat(\phi_1) \mid Post(s) \cap sat(\exists\phi_1 \mathsf{U} \phi_2) \neq \varnothing\}$$

We know that we look for a set T such that



(1) $T \supseteq sat(\phi_2)$

(2) $s \in sat(\phi_1)$ and $Post(s) \cap T \neq \varnothing$ implies $s \in T$

It can be shown that $sat(\exists\phi_1 \mathsf{U} \phi_2)$

is the smallest set T satisfying (1) and (2)

A fix point computation

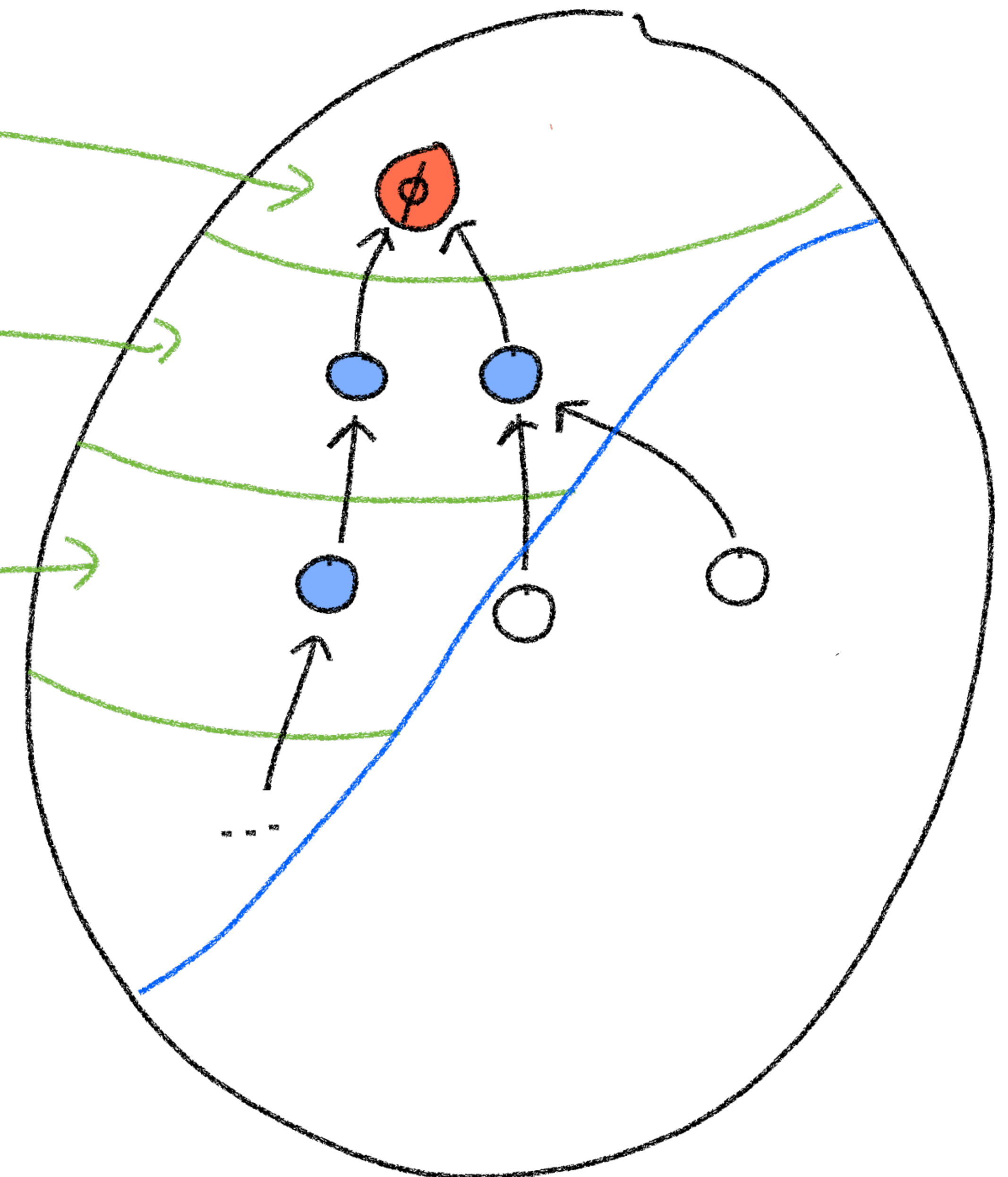
We can then just perform a standard least fix point calculation

$T_0 := sat(\phi)$

$T_1 := T_0 \cup \{s \in sat(\phi_1) \setminus T_0 \mid Post(s) \cap T_0 \neq \emptyset\}$

$T_2 := T_1 \cup \{s \in sat(\phi_1) \setminus T_1 \mid Post(s) \cap T_1 \neq \emptyset\}$

…



Algorithm for EU

Finally, the algorithm…

$T := sat(\phi);$

$W := sat(\phi);$

while $W \neq \varnothing$ do

  remove some s from W;

  foreach $s' \in Pre(s)$ do

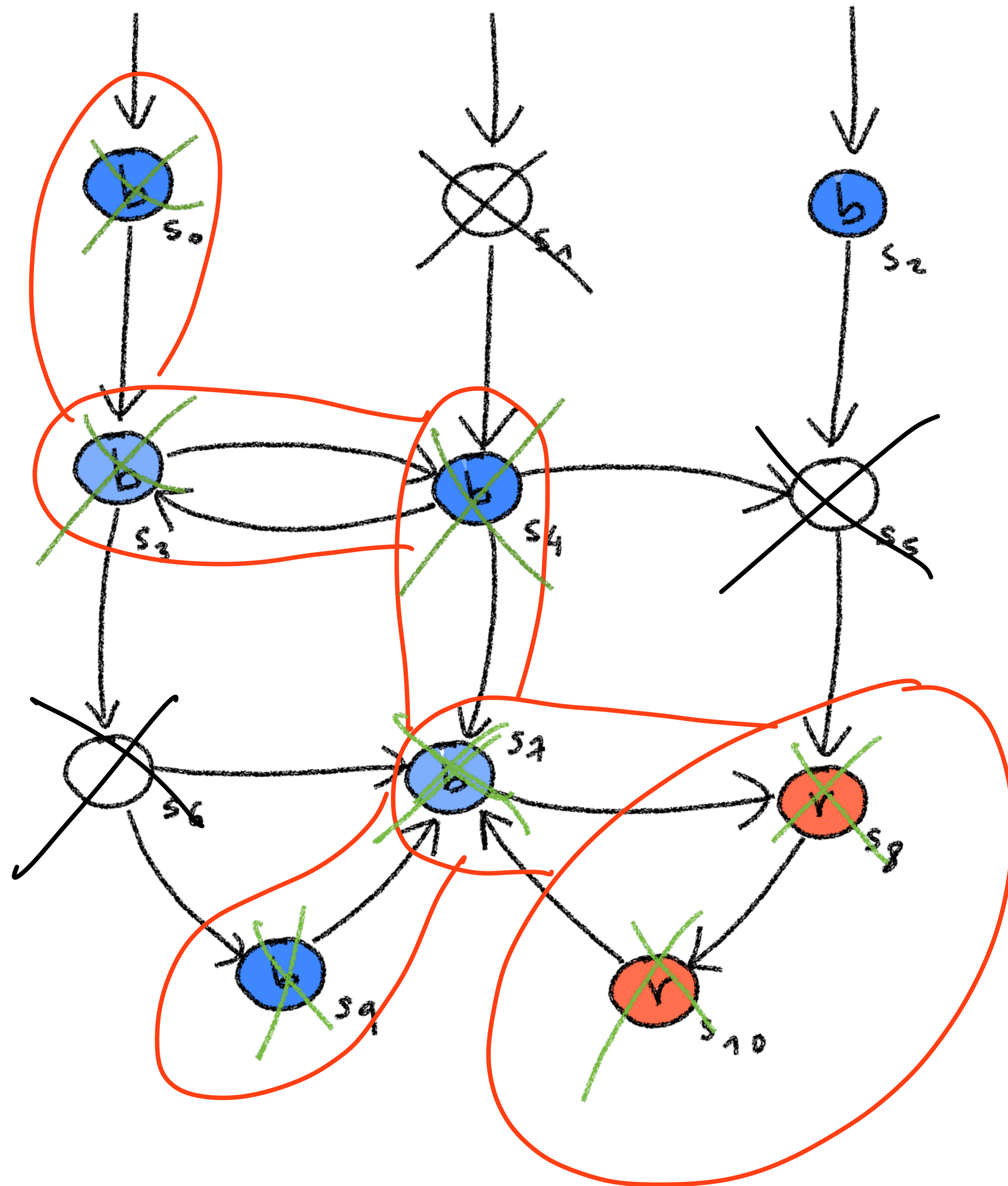
    if $s' \notin T$ and $s' \in \mathrm{sat}(\phi_1)$ then

      $T := T \cup s';$

      $W := W \cup s';$

return T;

An example



$\exists(b \; U \; r)$

T

W

Lecture 03 - Model Checking CTL

- Formal semantics and naïve algorithms
- Existential normal form CTL (ECTL)
- Global model checking algorithm
- Satisfaction sets characterisation of ECTL
- MC algorithms for basic cases
- MC algorithms for EU
- **MC algorithms for EG**
- Overall complexity

Perhaps unsurprisingly we can tell a similar story for EG

Expansion law for EG

The expansion law for EG

$$\exists \Box \phi \equiv \phi \wedge \exists \bigcirc \exists \Box \phi$$

Provides a recursive definition definition of *sat*

$$\begin{aligned} sat(\exists \Box \phi) &\equiv sat(\phi \wedge \exists \bigcirc \exists \Box \phi) \\ &\equiv sat(\phi) \cap sat(\exists \bigcirc \exists \Box \phi) \\ &\equiv sat(\phi) \cap \{s \in S \mid Post(s) \cap sat(\exists \Box phi) \neq \varnothing\} \\ &\equiv \{s \in sat(\phi) \mid Post(s) \cap sat(\exists \Box phi) \neq \varnothing\} \end{aligned}$$

A fix point computation

From our recursive definition

$$sat(\exists \Box \phi) \equiv \{s \in sat(\phi) \mid Post(s) \cap sat(\exists \Box \phi) \neq \emptyset\}$$

We know that we look
for a set T such that

(1) $T \subseteq sat(\phi)$

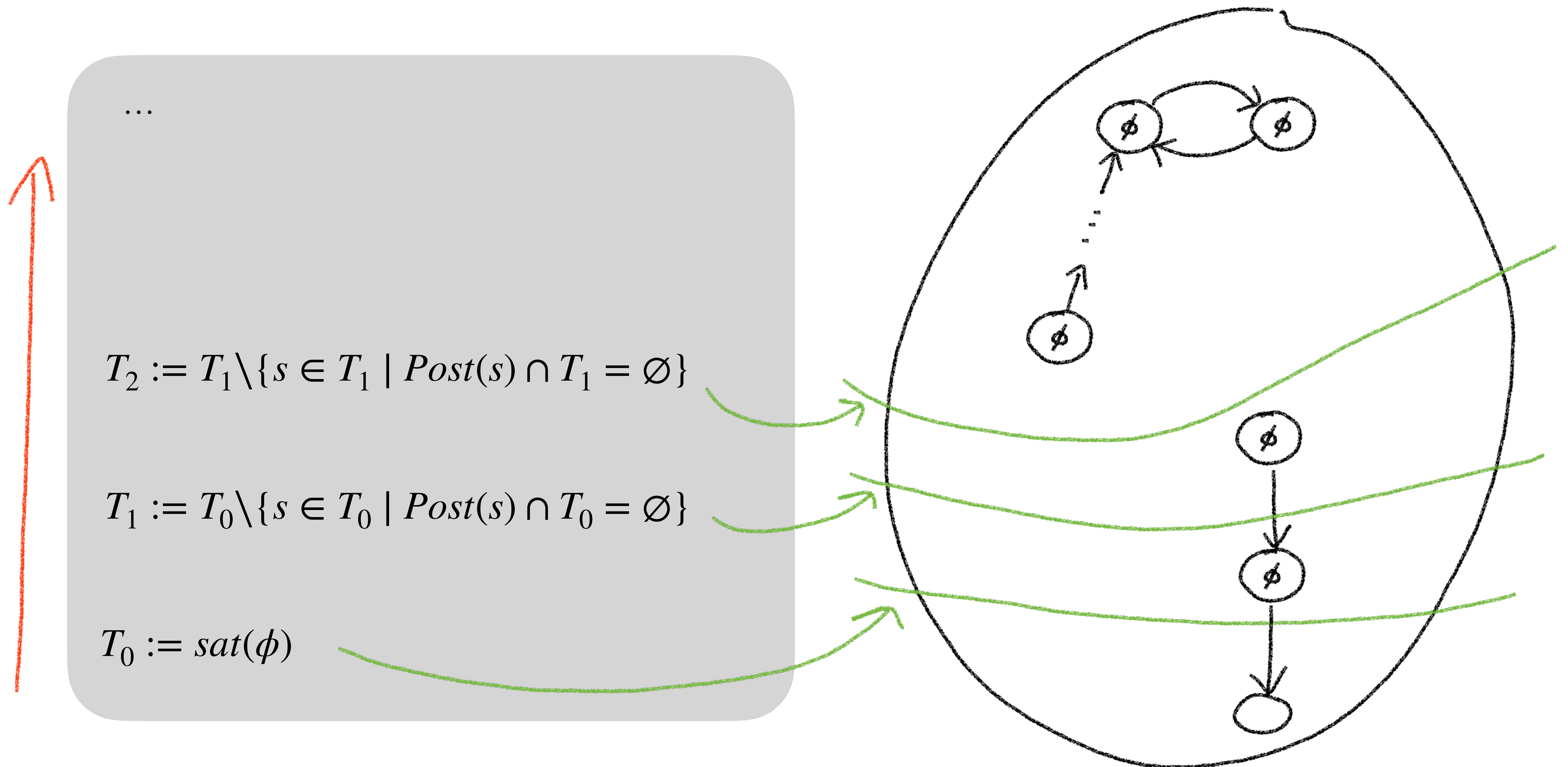
(2) $s \in T$ implies $Post(s) \cap T \neq \emptyset$

It can be shown that $sat(\exists \Box \phi)$

is the largest set T satisfying (1) and (2)

A fix point computation

We can then just perform a standard greatest fix point calculation



Finally, the algorithm…

$T := sat(\phi);$ // the set of states to be returned as a result

$W := S \backslash sat(\phi);$ // working set (states to be explored)

while $W \neq \varnothing$ do

    remove some s from W;

    foreach $s' \in Pre(s)$ do

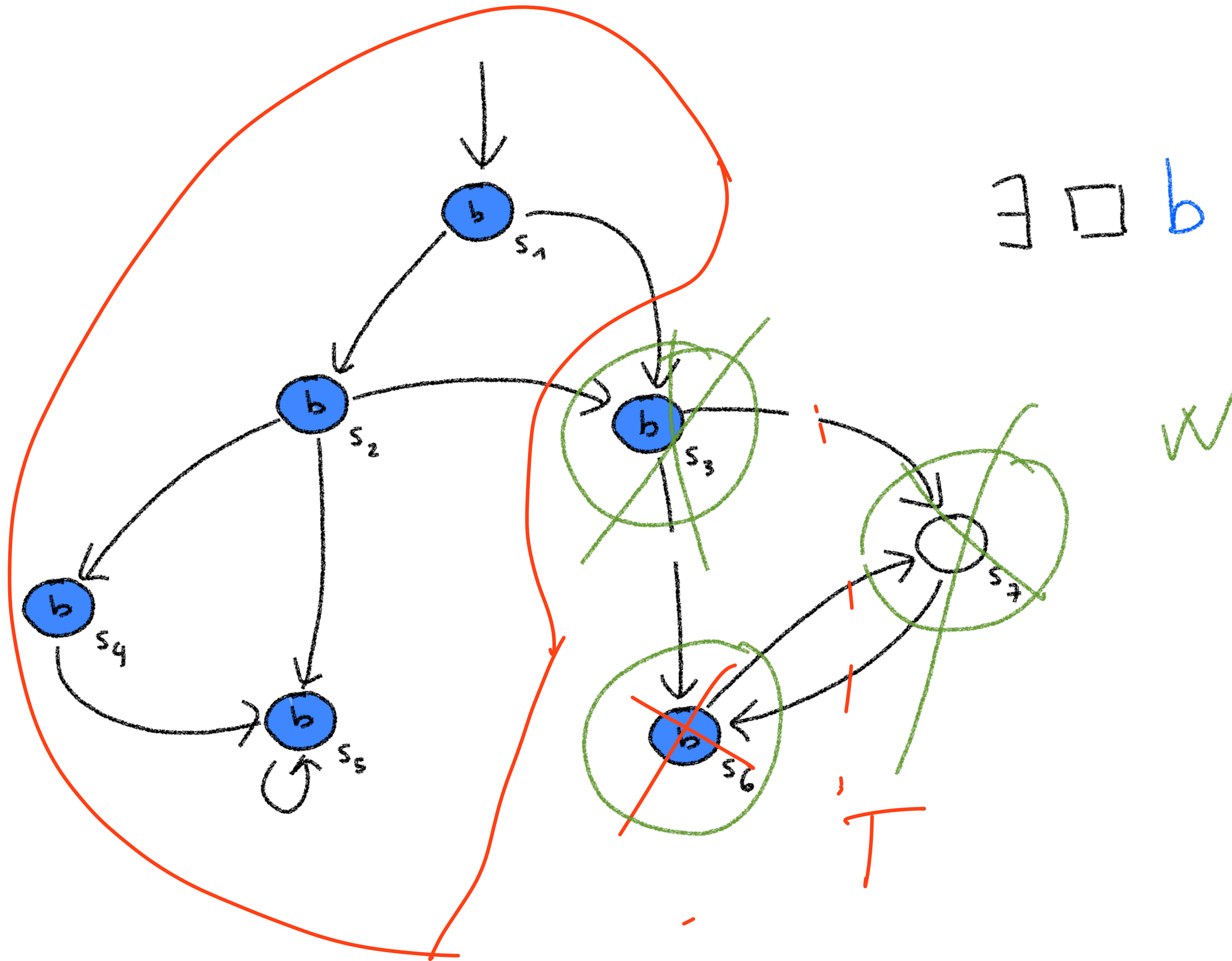
        if $s' \in T$ and $Post(s') \cap T = \varnothing$ then

            $T := T \backslash s';$

            $W := W \cup s';$

return T;

An example



Lecture 03 - Model Checking CTL

- Formal semantics and naïve algorithms
- Existential normal form CTL (ECTL)
- Global model checking algorithm
- Satisfaction sets characterisation of ECTL
- MC algorithms for basic cases
- MC algorithms for EU
- MC algorithms for EG
- Overall complexity

Overall *sat* (…) algorithm

We have now completed all cases of our algorithm:

```
sat(true) = …
sat(not phi) = …
sat(phi1 or phi2) = …
sat(EX phi) = …
sat(EG phi) = …
sat(E(phi1 U phi2) = …
```

The overall complexity is linear in the size of the transition system and the formula

Lecture 03 - Model Checking CTL

- Formal semantics and naïve algorithms
- Existential normal form CTL (ECTL)
- Global model checking algorithm
- Satisfaction sets characterisation of ECTL
- MC algorithms for basic cases
- MC algorithms for EU
- MC algorithms for EG
- Overall complexity

Key points so far

Global model checking algorithm, recursive on state sub-formulas (i.e. bottom-up on the parse tree).

Satisfaction set characterisations of CTL formulas in existential normal form.

Dedicated algorithms for CTL formulas in existential normal form, which exploit the satisfaction set characterisations.

The complexity of CTL model checking is polynomial in the size of the transition system and the CTL formula.