CS228 : Logic for Computer Science

Module 5: Temporal Logic

Lecture 25: CTL, Expressiveness, and Model Checking

Instructor: S. Akshay

IIT Bombay, India

Recap: CTL

$\varphi ::= \bot \mid \top \mid p \mid (\neg\varphi) \mid (\varphi \vee \varphi) \mid (\varphi \wedge \varphi) \mid (\varphi \to \varphi) \mid$
$AX\varphi \mid EX\varphi \mid AF\varphi \mid EF\varphi \mid AG\varphi \mid EG\varphi \mid A[\varphi\ U\ \varphi] \mid E[\varphi\ U\ \varphi]$
where $p \in AP$ are atomic propositions.

- A CTL formula is evaluated at a *state* s of a model M: $M, s \models \varphi$.

	X (next)	F (eventually)	G (always)	U (until)
A: all paths				
E: some path				

Recap: CTL

$\varphi ::= \bot \mid \top \mid p \mid (\neg\varphi) \mid (\varphi \vee \varphi) \mid (\varphi \wedge \varphi) \mid (\varphi \to \varphi) \mid$
$A X\varphi \mid E X\varphi \mid A F\varphi \mid E F\varphi \mid A G\varphi \mid E G\varphi \mid A[\varphi\ U\ \varphi] \mid E[\varphi\ U\ \varphi]$
where $p \in AP$ are atomic propositions.

- A CTL formula is evaluated at a *state* s of a model M: $M, s \models \varphi$.
- Every temporal operator (X, F, G, U) has a *path quantifier* in front: A (*all* paths from s) or E (*some* path from s).

	X (next)	F (eventually)	G (always)	U (until)
A: all paths				
E: some path				

Recap: CTL

$\varphi ::= \bot \mid \top \mid p \mid (\neg\varphi) \mid (\varphi \vee \varphi) \mid (\varphi \wedge \varphi) \mid (\varphi \to \varphi) \mid$
$AX\varphi \mid EX\varphi \mid AF\varphi \mid EF\varphi \mid AG\varphi \mid EG\varphi \mid A[\varphi\ U\ \varphi] \mid E[\varphi\ U\ \varphi]$
where $p \in AP$ are atomic propositions.

- A CTL formula is evaluated at a *state* s of a model M: $M, s \models \varphi$.
- Every temporal operator (X, F, G, U) has a *path quantifier* in front: A (*all* paths from s) or E (*some* path from s).

	X (next)	F (eventually)	G (always)	U (until)
A: all paths	$AX\,\varphi$ φ at every next state	$AF\,\varphi$ every path reaches φ	$AG\,\varphi$ on every path, φ throughout	$A[\varphi\,U\,\psi]$ on every path, φ until ψ
E: some path				

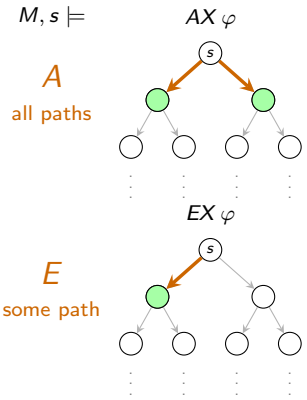
Recap: CTL

$\varphi ::= \bot \mid \top \mid p \mid (\neg\varphi) \mid (\varphi \vee \varphi) \mid (\varphi \wedge \varphi) \mid (\varphi \to \varphi) \mid$
$AX\varphi \mid EX\varphi \mid AF\varphi \mid EF\varphi \mid AG\varphi \mid EG\varphi \mid A[\varphi\ U\ \varphi] \mid E[\varphi\ U\ \varphi]$
where $p \in AP$ are atomic propositions.

- A CTL formula is evaluated at a *state* s of a model M: $M, s \models \varphi$.
- Every temporal operator (X, F, G, U) has a *path quantifier* in front: A (*all* paths from s) or E (*some* path from s).

	X (next)	F (eventually)	G (always)	U (until)
A: all paths	$AX\,\varphi$ φ at every next state	$AF\,\varphi$ every path reaches φ	$AG\,\varphi$ on every path, φ throughout	$A[\varphi\,U\,\psi]$ on every path, φ until ψ
E: some path	$EX\,\varphi$ φ at some next state	$EF\,\varphi$ some path reaches φ	$EG\,\varphi$ some path with φ throughout	$E[\varphi\,U\,\psi]$ some path with φ until ψ

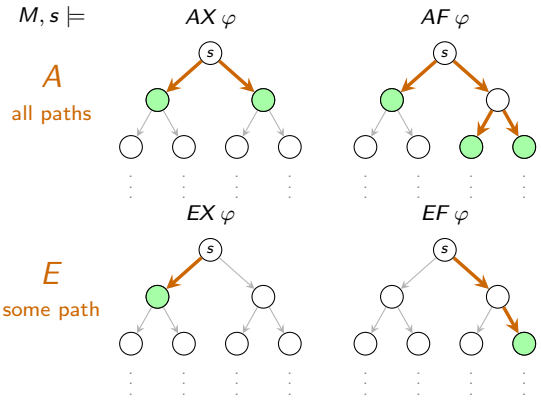
Recap: CTL semantics on a computation tree



φ-state ψ-state does not matter ⟶ the path(s) the formula talks about: all branches for A, one for E

Atoms: $M, s \models p$ iff $p \in L(s)$; $\neg, \wedge, \vee, \rightarrow$ are as usual.

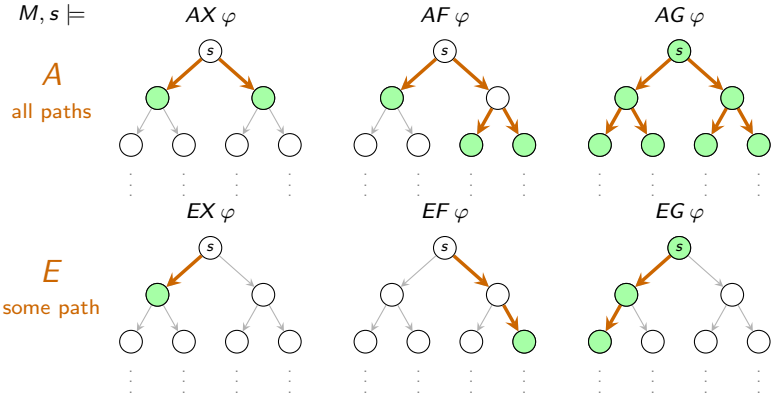
Recap: CTL semantics on a computation tree



φ-state ψ-state does not matter the path(s) the formula talks about: all branches for A, one for E

Atoms: $M, s \models p$ iff $p \in L(s)$; $\neg, \wedge, \vee, \rightarrow$ are as usual.

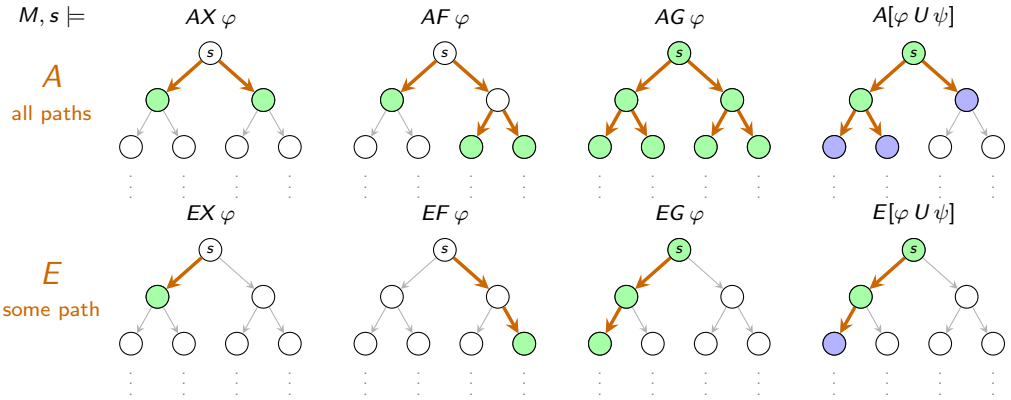
Recap: CTL semantics on a computation tree



φ-state ψ-state does not matter the path(s) the formula talks about: all branches for A, one for E

Atoms: $M, s \models p$ iff $p \in L(s)$; $\neg, \wedge, \vee, \rightarrow$ are as usual.

Recap: CTL semantics on a computation tree

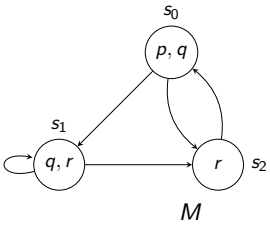


φ-state ψ-state does not matter the path(s) the formula talks about: all branches for A, one for E

Atoms: $M, s \models p$ iff $p \in L(s)$; $\neg, \wedge, \vee, \rightarrow$ are as usual.

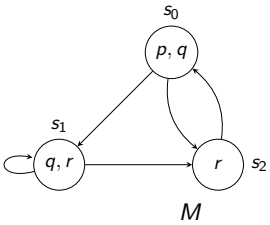
Recall example M from before...

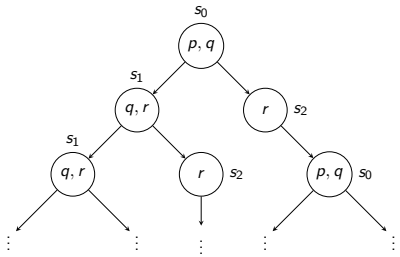
From any given state, we can unwind the system as an infinite tree of computation paths.



Recall example M from before...

From any given state, we can unwind the system as an infinite tree of computation paths.

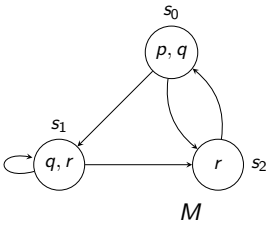




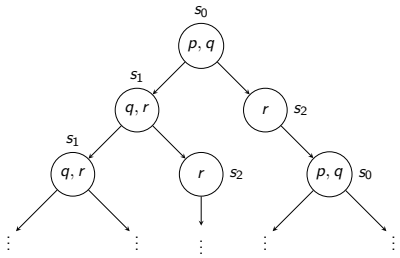
unwinding of M from s_0

Recall example M from before...

From any given state, we can unwind the system as an infinite tree of computation paths.



M



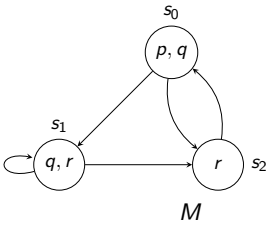
unwinding of M from s_0

Exercise 24.11 Which of the following hold? Why or why not? Justify!

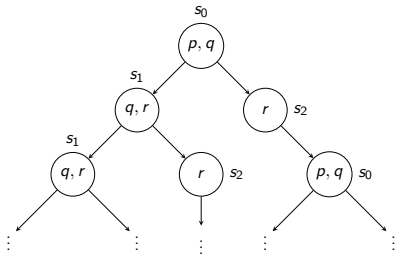
1. $M, s_0 \models (p \wedge q) \wedge \neg r$
2. $M, s_0 \models EX(q \wedge r)$
3. $M, s_0 \models AX(q \wedge r)$
4. $M, s_0 \models \neg EF(p \wedge r)$

Recall example M from before...

From any given state, we can unwind the system as an infinite tree of computation paths.



M



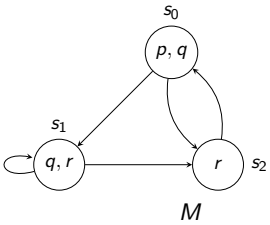
unwinding of M from s_0

Exercise 24.11 Which of the following hold? Why or why not? Justify!

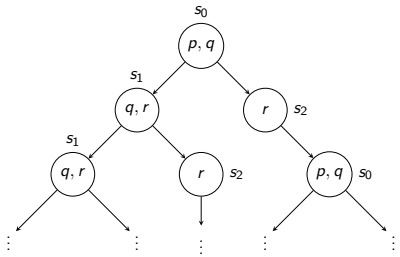
1. $M, s_0 \models (p \wedge q) \wedge \neg r$
2. $M, s_0 \models EX(q \wedge r)$
3. $M, s_0 \models AX(q \wedge r)$
4. $M, s_0 \models \neg EF(p \wedge r)$
5. $M, s_1 \models EG(q \wedge r)$
6. $M, s_0 \models AF\ r$
7. $M, s_0 \models E[(p \wedge q)\ U\ (q \wedge r)]$
8. $M, s_0 \models A[(p \wedge q)\ U\ (q \wedge r)]$

Recall example M from before...

From any given state, we can unwind the system as an infinite tree of computation paths.



M



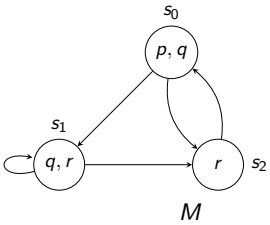
unwinding of M from s_0

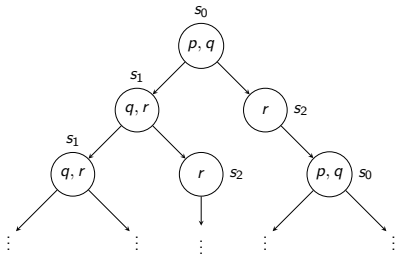
Exercise 24.11 Which of the following hold? Why or why not? Justify!

1. $M, s_0 \models (p \wedge q) \wedge \neg r$ ✓
2. $M, s_0 \models EX(q \wedge r)$ ✓
3. $M, s_0 \models AX(q \wedge r)$ ×
4. $M, s_0 \models \neg EF(p \wedge r)$ ✓
5. $M, s_1 \models EG(q \wedge r)$ ✓
6. $M, s_0 \models AF\ r$ ✓
7. $M, s_0 \models E[(p \wedge q)\ U\ (q \wedge r)]$ ✓
8. $M, s_0 \models A[(p \wedge q)\ U\ (q \wedge r)]$ ×

Recall example M from before...

From any given state, we can unwind the system as an infinite tree of computation paths.





unwinding of M from s_0

Exercise 24.11 Which of the following hold? Why or why not? Justify!

1. $M, s_0 \models (p \wedge q) \wedge \neg r$ ✓
2. $M, s_0 \models EX(q \wedge r)$ ✓
3. $M, s_0 \models AX(q \wedge r)$ ×
4. $M, s_0 \models \neg EF(p \wedge r)$ ✓
5. $M, s_1 \models EG(q \wedge r)$ ✓
6. $M, s_0 \models AF\ r$ ✓
7. $M, s_0 \models E[(p \wedge q)\ U\ (q \wedge r)]$ ✓
8. $M, s_0 \models A[(p \wedge q)\ U\ (q \wedge r)]$ ×

Is $\neg EF\,\varphi$ "the same as" $AG\,\neg\varphi$? Is $\neg AX\,\varphi$ equivalent to $EX\,\neg\varphi$? And so on...

Proving equivalence in CTL

Same idea as for LTL, now on states

Two CTL formulas φ, ψ are equivalent, $\varphi \equiv \psi$, if for all M, s, $M, s \models \varphi$ iff $M, s \models \psi$.

Proving equivalence in CTL

Same idea as for LTL, now on states

Two CTL formulas φ, ψ are equivalent, $\varphi \equiv \psi$, if for all M, s, $M, s \models \varphi$ iff $M, s \models \psi$.

Claim: $\neg AX\varphi \equiv EX\neg\varphi$. Proof for an arbitrary model M and state s of M:

$$M, s \models \neg AX\varphi$$
$$\iff M, s \not\models AX\varphi \qquad \text{semantics of } \neg$$

Proving equivalence in CTL

Same idea as for LTL, now on states

Two CTL formulas φ, ψ are equivalent, $\varphi \equiv \psi$, if for all M, s, $M, s \models \varphi$ iff $M, s \models \psi$.

Claim: $\neg AX\varphi \equiv EX\neg\varphi$. Proof for an arbitrary model M and state s of M:

$$
\begin{aligned}
& M, s \models \neg AX\varphi \\
\iff\ & M, s \not\models AX\varphi && \text{semantics of } \neg \\
\iff\ & \text{not (for all } s_1 \text{ with } s \to s_1\text{: } M, s_1 \models \varphi) && \text{semantics of } AX
\end{aligned}
$$

Proving equivalence in CTL

Same idea as for LTL, now on states

Two CTL formulas φ, ψ are equivalent, $\varphi \equiv \psi$, if for all M, s, $M, s \models \varphi$ iff $M, s \models \psi$.

Claim: $\neg AX\varphi \equiv EX\neg\varphi$. Proof for an arbitrary model M and state s of M:

	$M, s \models \neg AX\varphi$	
$\iff$	$M, s \not\models AX\varphi$	semantics of $\neg$
$\iff$	not (for all s_1 with $s \to s_1$: $M, s_1 \models \varphi$)	semantics of AX
$\iff$	for some s_1 with $s \to s_1$: $M, s_1 \not\models \varphi$	negating $\forall$

Proving equivalence in CTL

Same idea as for LTL, now on states

Two CTL formulas φ, ψ are equivalent, $\varphi \equiv \psi$, if for all M, s, $M, s \models \varphi$ iff $M, s \models \psi$.

Claim: $\neg AX\varphi \equiv EX\neg\varphi$. Proof for an arbitrary model M and state s of M:

	$M, s \models \neg AX\varphi$	
$\iff$	$M, s \not\models AX\varphi$	semantics of $\neg$
$\iff$	not (for all s_1 with $s \to s_1$: $M, s_1 \models \varphi$)	semantics of AX
$\iff$	for some s_1 with $s \to s_1$: $M, s_1 \not\models \varphi$	negating $\forall$
$\iff$	for some s_1 with $s \to s_1$: $M, s_1 \models \neg\varphi$	semantics of $\neg$

Proving equivalence in CTL

Same idea as for LTL, now on states

Two CTL formulas φ, ψ are equivalent, $\varphi \equiv \psi$, if for all M, s, $M, s \models \varphi$ iff $M, s \models \psi$.

Claim: $\neg AX\varphi \equiv EX\neg\varphi$. Proof for an arbitrary model M and state s of M:

	$M, s \models \neg AX\varphi$	
$\iff$	$M, s \not\models AX\varphi$	semantics of $\neg$
$\iff$	not (for all s_1 with $s \to s_1$: $M, s_1 \models \varphi$)	semantics of AX
$\iff$	for some s_1 with $s \to s_1$: $M, s_1 \not\models \varphi$	negating $\forall$
$\iff$	for some s_1 with $s \to s_1$: $M, s_1 \models \neg\varphi$	semantics of $\neg$
$\iff$	$M, s \models EX\neg\varphi$	semantics of EX

Equivalences and Adequate sets

$$\varphi ::= \bot \mid \top \mid p \mid (\neg\varphi) \mid (\varphi \vee \varphi) \mid (\varphi \wedge \varphi) \mid (\varphi \rightarrow \varphi) \mid$$
$$AX\varphi \mid EX\varphi \mid AF\varphi \mid EF\varphi \mid AG\varphi \mid EG\varphi \mid A[\varphi\ U\ \varphi] \mid E[\varphi\ U\ \varphi]$$

Equivalences and Adequate sets

$$\varphi ::= \bot \mid \top \mid p \mid (\neg\varphi) \mid (\varphi \vee \varphi) \mid (\varphi \wedge \varphi) \mid (\varphi \rightarrow \varphi) \mid$$
$$AX\varphi \mid EX\varphi \mid AF\varphi \mid EF\varphi \mid AG\varphi \mid EG\varphi \mid A[\varphi \ U \ \varphi] \mid E[\varphi \ U \ \varphi]$$

Exercise 25.1 (PS10) Check the following equivalences

1. $\neg AF\varphi \equiv EG\neg\varphi$
2. $\neg EF\varphi \equiv AG\neg\varphi$
3. $\neg AX\varphi \equiv EX\neg\varphi$ ✓

Equivalences and Adequate sets

$$\varphi ::= \bot \mid \top \mid p \mid (\neg\varphi) \mid (\varphi \vee \varphi) \mid (\varphi \wedge \varphi) \mid (\varphi \to \varphi) \mid$$
$$AX\varphi \mid EX\varphi \mid AF\varphi \mid EF\varphi \mid AG\varphi \mid EG\varphi \mid A[\varphi \ U \ \varphi] \mid E[\varphi \ U \ \varphi]$$

Exercise 25.1 (PS10) Check the following equivalences

1. $\neg AF\varphi \equiv EG\neg\varphi$
2. $\neg EF\varphi \equiv AG\neg\varphi$
3. $\neg AX\varphi \equiv EX\neg\varphi$ ✓
4. $AF\varphi \equiv A[\top \ U \ \varphi]$
5. $EF\varphi \equiv E[\top \ U \ \varphi]$

Equivalences and Adequate sets

$$\varphi ::= \bot \mid \top \mid p \mid (\neg\varphi) \mid (\varphi \vee \varphi) \mid (\varphi \wedge \varphi) \mid (\varphi \to \varphi) \mid$$
$$AX\varphi \mid EX\varphi \mid AF\varphi \mid EF\varphi \mid AG\varphi \mid EG\varphi \mid A[\varphi \ U \ \varphi] \mid E[\varphi \ U \ \varphi]$$

Exercise 25.1 (PS10) Check the following equivalences

1. $\neg AF\varphi \equiv EG\neg\varphi$
2. $\neg EF\varphi \equiv AG\neg\varphi$
3. $\neg AX\varphi \equiv EX\neg\varphi$
4. $AF\varphi \equiv A[\top \ U \ \varphi]$
5. $EF\varphi \equiv E[\top \ U \ \varphi]$
6. ✓ $AF\varphi \equiv \varphi \vee AX \ AF\varphi$

Equivalences and Adequate sets

$$\varphi ::= \bot \mid \top \mid p \mid (\neg\varphi) \mid (\varphi \vee \varphi) \mid (\varphi \wedge \varphi) \mid (\varphi \to \varphi) \mid$$
$$AX\varphi \mid EX\varphi \mid AF\varphi \mid EF\varphi \mid AG\varphi \mid EG\varphi \mid A[\varphi \ U \ \varphi] \mid E[\varphi \ U \ \varphi]$$

Exercise 25.1 (PS10) Check the following equivalences

1. $\neg AF\varphi \equiv EG\neg\varphi$
2. $\neg EF\varphi \equiv AG\neg\varphi$
3. $\neg AX\varphi \equiv EX\neg\varphi$
4. $AF\varphi \equiv A[\top \ U \ \varphi]$
5. $EF\varphi \equiv E[\top \ U \ \varphi]$
6. ✓ $AF\varphi \equiv \varphi \vee AX \ AF\varphi$

From the above equivalences (which ones?), it follows that

AU, EU, EX are a universal/adequate set of CTL temporal connectives

Exercise 25.2. Show that $\{EG, EU, EX\}$ is also adequate. Are there yet other sets?

Equivalences and Adequate sets

$$\varphi ::= \bot \mid \top \mid p \mid (\neg\varphi) \mid (\varphi \vee \varphi) \mid (\varphi \wedge \varphi) \mid (\varphi \to \varphi) \mid$$
$$AX\varphi \mid EX\varphi \mid AF\varphi \mid EF\varphi \mid AG\varphi \mid EG\varphi \mid A[\varphi\ U\ \varphi] \mid E[\varphi\ U\ \varphi]$$

Exercise 25.1 (PS10) Check the following equivalences

1. $\neg AF\varphi \equiv EG\neg\varphi$
2. $\neg EF\varphi \equiv AG\neg\varphi$
3. $\neg AX\varphi \equiv EX\neg\varphi$
4. $AF\varphi \equiv A[\top\ U\ \varphi]$
5. $EF\varphi \equiv E[\top\ U\ \varphi]$
6. ✓ $AF\varphi \equiv \varphi \vee AX\ AF\varphi$

From the above equivalences (1–5), it follows that

AU, EU, EX are a universal/adequate set of CTL temporal connectives

Exercise 25.2. Show that $\{EG, EU, EX\}$ is also adequate. Are there yet other sets?

Equivalences and Adequate sets

$$\varphi ::= \bot \mid \top \mid p \mid (\neg\varphi) \mid (\varphi \vee \varphi) \mid (\varphi \wedge \varphi) \mid (\varphi \to \varphi) \mid$$
$$AX\varphi \mid EX\varphi \mid AF\varphi \mid EF\varphi \mid AG\varphi \mid EG\varphi \mid A[\varphi\ U\ \varphi] \mid E[\varphi\ U\ \varphi]$$

Exercise 25.1 (PS10) Check the following equivalences

1. $\neg AF\varphi \equiv EG\neg\varphi$
2. $\neg EF\varphi \equiv AG\neg\varphi$
3. $\neg AX\varphi \equiv EX\neg\varphi$ ✓
4. $AF\varphi \equiv A[\top\ U\ \varphi]$
5. $EF\varphi \equiv E[\top\ U\ \varphi]$
6. $AF\varphi \equiv \varphi \vee AX\ AF\varphi$

From the above equivalences (1–5), it follows that

AU, EU, EX are a universal/adequate set of CTL temporal connectives

Exercise 25.2. Show that $\{EG, EU, EX\}$ is also adequate. Are there yet other sets?

We can also define Weak-until and Release, AR, ER, AW, EW in CTL as for LTL.

Some practical examples of CTL specifications

Take atomic propositions: **busy**, **ready**, **started**, **req**, **grant**, **enabled**, **running**, etc.

1. It is possible to reach a state where **started** holds but **ready** does not.

Some practical examples of CTL specifications

Take atomic propositions: **busy**, **ready**, **started**, **req**, **grant**, **enabled**, **running**, etc.

1. It is possible to reach a state where **started** holds but **ready** does not.
   $EF(\ \textbf{started} \wedge \neg \textbf{ready})$

Some practical examples of CTL specifications

Take atomic propositions: **busy**, **ready**, **started**, **req**, **grant**, **enabled**, **running**, etc.

1. It is possible to reach a state where **started** holds but **ready** does not.
   $EF(\ \textbf{started} \wedge \neg \textbf{ready})$
2. In any state, if there is a request (of some resource), then it will be eventually granted.

Some practical examples of CTL specifications

Take atomic propositions: **busy**, **ready**, **started**, **req**, **grant**, **enabled**, **running**, etc.

1. It is possible to reach a state where **started** holds but **ready** does not.
   $EF($ **started** $\wedge \neg$**ready**$)$
2. In any state, if there is a request (of some resource), then it will be eventually granted.
   $AG($**req** $\rightarrow AF$ **grant**$)$. recall the LTL formula!

Some practical examples of CTL specifications

Take atomic propositions: **busy**, **ready**, **started**, **req**, **grant**, **enabled**, **running**, etc.

1. It is possible to reach a state where **started** holds but **ready** does not.
   $EF(\ \mathbf{started} \wedge \neg \mathbf{ready})$
2. In any state, if there is a request (of some resource), then it will be eventually granted.
   $AG(\mathbf{req} \rightarrow AF\ \mathbf{grant})$. recall the LTL formula! $G(\mathbf{req} \rightarrow F\ \mathbf{grant})$

Some practical examples of CTL specifications

Take atomic propositions: **busy**, **ready**, **started**, **req**, **grant**, **enabled**, **running**, etc.

1. It is possible to reach a state where **started** holds but **ready** does not.
   $EF(\ \mathbf{started} \wedge \neg \mathbf{ready})$
2. In any state, if there is a request (of some resource), then it will be eventually granted.
   $AG(\mathbf{req} \rightarrow AF\ \mathbf{grant})$. recall the LTL formula! $G(\mathbf{req} \rightarrow F\ \mathbf{grant})$
3. If a process is enabled infinitely often, then it runs infinitely often.

Some practical examples of CTL specifications

Take atomic propositions: **busy**, **ready**, **started**, **req**, **grant**, **enabled**, **running**, etc.

1. It is possible to reach a state where **started** holds but **ready** does not.
   $EF($ **started** $\wedge \neg$**ready**$)$
2. In any state, if there is a request (of some resource), then it will be eventually granted.
   $AG($**req** $\rightarrow AF$ **grant**$)$. recall the LTL formula! $G($**req** $\rightarrow F$ **grant**$)$
3. If a process is enabled infinitely often, then it runs infinitely often. Expressible in CTL??

Some practical examples of CTL specifications

Take atomic propositions: **busy**, **ready**, **started**, **req**, **grant**, **enabled**, **running**, etc.

1. It is possible to reach a state where **started** holds but **ready** does not.
   $EF($ **started** $\wedge \neg$**ready**$)$
2. In any state, if there is a request (of some resource), then it will be eventually granted.
   $AG($**req** $\rightarrow AF$ **grant**$)$. recall the LTL formula! $G($**req** $\rightarrow F$ **grant**$)$
3. If a process is enabled infinitely often, then it runs infinitely often. Expressible in CTL??
   Recall the LTL formula for this.

Some practical examples of CTL specifications

Take atomic propositions: **busy**, **ready**, **started**, **req**, **grant**, **enabled**, **running**, etc.

1. It is possible to reach a state where **started** holds but **ready** does not.
   $EF($ **started** $\wedge \neg$**ready**$)$
2. In any state, if there is a request (of some resource), then it will be eventually granted.
   $AG($**req** $\rightarrow AF$ **grant**$)$. recall the LTL formula! $G($**req** $\rightarrow F$ **grant**$)$
3. If a process is enabled infinitely often, then it runs infinitely often. Expressible in CTL??
   Recall the LTL formula for this. What does $AG\ AF$ **enabled** $\rightarrow AG\ AF$ **running** do?

Some practical examples of CTL specifications

Take atomic propositions: **busy**, **ready**, **started**, **req**, **grant**, **enabled**, **running**, etc.

1. It is possible to reach a state where **started** holds but **ready** does not.
   $EF($ **started** $\wedge \neg$**ready**$)$
2. In any state, if there is a request (of some resource), then it will be eventually granted.
   $AG($**req** $\rightarrow AF$ **grant**$)$. recall the LTL formula! $G($**req** $\rightarrow F$ **grant**$)$
3. If a process is enabled infinitely often, then it runs infinitely often. Expressible in CTL??
   Recall the LTL formula for this. What does $AG\ AF$ **enabled** $\rightarrow AG\ AF$ **running** do?
   Weaker: it only says that if all paths have enabled infinitely often, then all have running infinitely often.

Some practical examples of CTL specifications

Take atomic propositions: **busy**, **ready**, **started**, **req**, **grant**, **enabled**, **running**, etc.

1. It is possible to reach a state where **started** holds but **ready** does not.
   $EF($ **started** $\wedge \neg$**ready**$)$
2. In any state, if there is a request (of some resource), then it will be eventually granted.
   $AG($**req** $\rightarrow AF$ **grant**$)$. recall the LTL formula! $G($**req** $\rightarrow F$ **grant**$)$
3. If a process is enabled infinitely often, then it runs infinitely often. Expressible in CTL??
   Recall the LTL formula for this. What does $AG\ AF$ **enabled** $\rightarrow AG\ AF$ **running** do?
   Weaker: it only says that if all paths have enabled infinitely often, then all have running infinitely often.
4. From any state we reach, we can get to some ready state.

Some practical examples of CTL specifications

Take atomic propositions: **busy**, **ready**, **started**, **req**, **grant**, **enabled**, **running**, etc.

1. It is possible to reach a state where **started** holds but **ready** does not.
   $EF($ **started** $\wedge \neg$**ready**$)$
2. In any state, if there is a request (of some resource), then it will be eventually granted.
   $AG($**req** $\rightarrow AF$ **grant**$)$. recall the LTL formula! $G($**req** $\rightarrow F$ **grant**$)$
3. If a process is enabled infinitely often, then it runs infinitely often. Expressible in CTL??
   Recall the LTL formula for this. What does $AG\ AF$ **enabled** $\rightarrow AG\ AF$ **running** do?
   Weaker: it only says that if all paths have enabled infinitely often, then all have running infinitely often.
4. From any state we reach, we can get to some ready state. $AG\ EF$ **ready**.

Some practical examples of CTL specifications

Take atomic propositions: **busy**, **ready**, **started**, **req**, **grant**, **enabled**, **running**, etc.

1. It is possible to reach a state where **started** holds but **ready** does not.
   $EF($ **started** $\wedge \neg$**ready**$)$
2. In any state, if there is a request (of some resource), then it will be eventually granted.
   $AG($**req** $\rightarrow AF$ **grant**$)$. recall the LTL formula! $G($**req** $\rightarrow F$ **grant**$)$
3. If a process is enabled infinitely often, then it runs infinitely often. Expressible in CTL??
   Recall the LTL formula for this. What does $AG\ AF$ **enabled** $\rightarrow AG\ AF$ **running** do?
   Weaker: it only says that if all paths have enabled infinitely often, then all have running infinitely often.
4. From any state we reach, we can get to some ready state. $AG\ EF$ **ready**. Can this be expressed in LTL?

Expressiveness between CTL and LTL

- Some properties are expressible both in LTL and CTL.

Expressiveness between CTL and LTL

- Some properties are expressible both in LTL and CTL.
  - E.g., every request is eventually granted: CTL $AG(p \rightarrow AFq)$, $G(p \rightarrow Fq)$.

Expressiveness between CTL and LTL

- Some properties are expressible both in LTL and CTL.
  - E.g., every request is eventually granted: CTL $AG(p \rightarrow AFq)$, $G(p \rightarrow Fq)$.

Theorem: CTL and LTL are expressively incomparable

1. There exist properties that are expressible in CTL but not LTL.

2. There exist properties that are expressible in LTL but not CTL.

Expressiveness between CTL and LTL

- Some properties are expressible both in LTL and CTL.
  - E.g., every request is eventually granted: CTL $AG(p \rightarrow AFq)$, $G(p \rightarrow Fq)$.

Theorem: CTL and LTL are expressively incomparable

1. There exist properties that are expressible in CTL but not LTL.
   - e.g., from any state we can get to a ready state: CTL $AG\ EFp$.

2. There exist properties that are expressible in LTL but not CTL.

Expressiveness between CTL and LTL

- Some properties are expressible both in LTL and CTL.
  - E.g., every request is eventually granted: CTL $AG(p \rightarrow AFq)$, $G(p \rightarrow Fq)$.

Theorem: CTL and LTL are expressively incomparable

1. There exist properties that are expressible in CTL but not LTL.
   - e.g., from any state we can get to a ready state: CTL $AG\ EFp$.
   - Not expressible in LTL. Intuitively, it cannot be reduced to a property of a path or all paths.
2. There exist properties that are expressible in LTL but not CTL.

Expressiveness between CTL and LTL

- Some properties are expressible both in LTL and CTL.
  - E.g., every request is eventually granted: CTL $AG(p \rightarrow AFq)$, $G(p \rightarrow Fq)$.

Theorem: CTL and LTL are expressively incomparable

1. There exist properties that are expressible in CTL but not LTL.
   - e.g., from any state we can get to a ready state: CTL $AG\ EFp$.
   - Not expressible in LTL. Intuitively, it cannot be reduced to a property of a path or all paths.
2. There exist properties that are expressible in LTL but not CTL.
   - e.g., if p occurs infinitely often, then so does q: LTL $GFp \rightarrow GFq$.

Expressiveness between CTL and LTL

- Some properties are expressible both in LTL and CTL.
  - E.g., every request is eventually granted: CTL $AG(p \rightarrow AFq)$, $G(p \rightarrow Fq)$.

Theorem: CTL and LTL are expressively incomparable

1. There exist properties that are expressible in CTL but not LTL.
   - e.g., from any state we can get to a ready state: CTL $AG\ EFp$.
   - Not expressible in LTL. Intuitively, it cannot be reduced to a property of a path or all paths.
2. There exist properties that are expressible in LTL but not CTL.
   - e.g., if p occurs infinitely often, then so does q: LTL $GFp \rightarrow GFq$.
   - Not expressible in CTL. Intuitively, we need to talk about that very path, so it's a path property!

Expressiveness between CTL and LTL

- Some properties are expressible both in LTL and CTL.
  - E.g., every request is eventually granted: CTL $AG(p \rightarrow AFq)$, $G(p \rightarrow Fq)$.

Theorem: CTL and LTL are expressively incomparable

1. There exist properties that are expressible in CTL but not LTL.
   - e.g., from any state we can get to a ready state: CTL $AG\ EFp$.
   - Not expressible in LTL. Intuitively, it cannot be reduced to a property of a path or all paths.
2. There exist properties that are expressible in LTL but not CTL.
   - e.g., if p occurs infinitely often, then so does q: LTL $GFp \rightarrow GFq$.
   - Not expressible in CTL. Intuitively, we need to talk about that very path, so it's a path property!

Finally, are there properties that are expressible neither in LTL nor in CTL?

Expressiveness between CTL and LTL

- Some properties are expressible both in LTL and CTL.
  - E.g., every request is eventually granted: CTL $AG(p \to AFq)$, $G(p \to Fq)$.

Theorem: CTL and LTL are expressively incomparable

1. There exist properties that are expressible in CTL but not LTL.
   - e.g., from any state we can get to a ready state: CTL $AG\ EFp$.
   - Not expressible in LTL. Intuitively, it cannot be reduced to a property of a path or all paths.
2. There exist properties that are expressible in LTL but not CTL.
   - e.g., if p occurs infinitely often, then so does q: LTL $GFp \to GFq$.
   - Not expressible in CTL. Intuitively, we need to talk about that very path, so it's a path property!

Finally, are there properties that are expressible neither in LTL nor in CTL? There exists a path with infinitely many p's!

Expressiveness between CTL and LTL

- Some properties are expressible both in LTL and CTL.
  - E.g., every request is eventually granted: CTL $AG(p \to AFq)$, $G(p \to Fq)$.

Theorem: CTL and LTL are expressively incomparable

1. There exist properties that are expressible in CTL but not LTL.
   - e.g., from any state we can get to a ready state: CTL $AG\ EFp$.
   - Not expressible in LTL. Intuitively, it cannot be reduced to a property of a path or all paths.
2. There exist properties that are expressible in LTL but not CTL.
   - e.g., if p occurs infinitely often, then so does q: LTL $GFp \to GFq$.
   - Not expressible in CTL. Intuitively, we need to talk about that very path, so it's a path property!

Finally, are there properties that are expressible neither in LTL nor in CTL? There exists a path with infinitely many p's!

- Question: How do we prove these inexpressibility results formally?
- This is often difficult (and fun!). But there are techniques...

Formally proving Inexpressibility results: An example!

- Given an LTL formula ψ and a CTL formula φ, we say $\psi \equiv \varphi$ if for all M, s, $M, s \models \psi$ iff $M, s \models \varphi$. For example, $AG(p \rightarrow AFq) \equiv G(p \rightarrow Fq)$.

Formally proving Inexpressibility results: An example!

- Given an LTL formula ψ and a CTL formula φ, we say $\psi \equiv \varphi$ if for all M, s, $M, s \models \psi$ iff $M, s \models \varphi$. For example, $AG(p \rightarrow AFq) \equiv G(p \rightarrow Fq)$.

Theorem: There exists no LTL formula equivalent to $\varphi = (AG\ EF\ p)$.

Formally proving Inexpressibility results: An example!

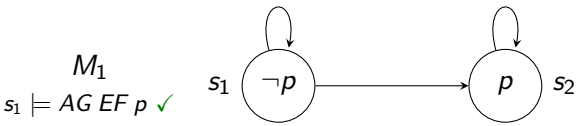
- Given an LTL formula ψ and a CTL formula φ, we say $\psi \equiv \varphi$ if for all M, s, $M, s \models \psi$ iff $M, s \models \varphi$. For example, $AG(p \rightarrow AFq) \equiv G(p \rightarrow Fq)$.

Theorem: There exists no LTL formula equivalent to $\varphi = (AG\ EF\ p)$.

Proof: Suppose not, let ψ be an LTL formula s.t, $\psi \equiv \varphi$, i.e., $\forall M, s$, $M, s \models \psi$ iff $M, s \models \varphi$.

Formally proving Inexpressibility results: An example!

- Given an LTL formula ψ and a CTL formula φ, we say $\psi \equiv \varphi$ if for all M, s, $M, s \models \psi$ iff $M, s \models \varphi$. For example, $AG(p \rightarrow AFq) \equiv G(p \rightarrow Fq)$.



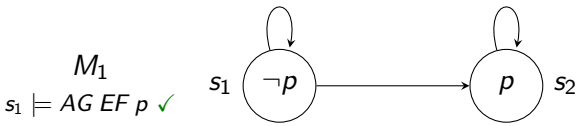
Theorem: There exists no LTL formula equivalent to $\varphi = (AG\ EF\ p)$.

Proof: Suppose not, let ψ be an LTL formula s.t, $\psi \equiv \varphi$, i.e., $\forall M, s$, $M, s \models \psi$ iff $M, s \models \varphi$.

- Consider M_1, s_1. $M_1, s_1 \models (AG\ EF\ p)$ so $M_1, s_1 \models \psi$.

Formally proving Inexpressibility results: An example!

- Given an LTL formula ψ and a CTL formula φ, we say $\psi \equiv \varphi$ if for all M, s, $M, s \models \psi$ iff $M, s \models \varphi$. For example, $AG(p \rightarrow AFq) \equiv G(p \rightarrow Fq)$.



Theorem: There exists no LTL formula equivalent to $\varphi = (AG\ EF\ p)$.

Proof: Suppose not, let ψ be an LTL formula s.t, $\psi \equiv \varphi$, i.e., $\forall M, s$, $M, s \models \psi$ iff $M, s \models \varphi$.

- Consider M_1, s_1. $M_1, s_1 \models (AG\ EF\ p)$ so $M_1, s_1 \models \psi$.
- By definition of satisfaction in LTL, for all paths starting at s_1 in M_1, LTL formula ψ holds.

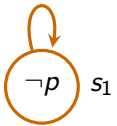
Formally proving Inexpressibility results: An example!

- Given an LTL formula ψ and a CTL formula φ, we say $\psi \equiv \varphi$ if for all M, s, $M, s \models \psi$ iff $M, s \models \varphi$. For example, $AG(p \rightarrow AFq) \equiv G(p \rightarrow Fq)$.

M_1

$s_1 \models AG\ EF\ p$ ✓





M_2

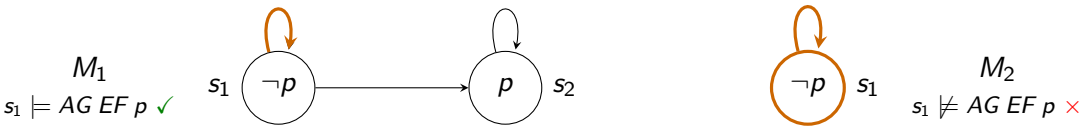
Theorem: There exists no LTL formula equivalent to $\varphi = (AG\ EF\ p)$.

Proof: Suppose not, let ψ be an LTL formula s.t, $\psi \equiv \varphi$, i.e., $\forall M, s$, $M, s \models \psi$ iff $M, s \models \varphi$.

- Consider M_1, s_1. $M_1, s_1 \models (AG\ EF\ p)$ so $M_1, s_1 \models \psi$.
- By definition of satisfaction in LTL, for all paths starting at s_1 in M_1, LTL formula ψ holds.
- Consider M_2: paths from s_1 in M_2 are a subset of paths from s_1 in M_1, hence $M_2, s_1 \models \psi$.

Formally proving Inexpressibility results: An example!

- Given an LTL formula ψ and a CTL formula φ, we say $\psi \equiv \varphi$ if for all M, s, $M, s \models \psi$ iff $M, s \models \varphi$. For example, $AG(p \rightarrow AFq) \equiv G(p \rightarrow Fq)$.



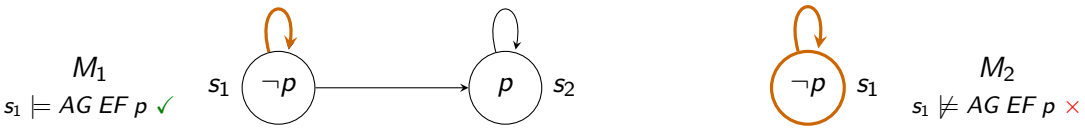
Theorem: There exists no LTL formula equivalent to $\varphi = (AG\ EF\ p)$.

Proof: Suppose not, let ψ be an LTL formula s.t, $\psi \equiv \varphi$, i.e., $\forall M, s$, $M, s \models \psi$ iff $M, s \models \varphi$.

- Consider M_1, s_1. $M_1, s_1 \models (AG\ EF\ p)$ so $M_1, s_1 \models \psi$.
- By definition of satisfaction in LTL, for all paths starting at s_1 in M_1, LTL formula ψ holds.
- Consider M_2: paths from s_1 in M_2 are a subset of paths from s_1 in M_1, hence $M_2, s_1 \models \psi$.
- But $M_2, s_1 \not\models \varphi$, hence contradiction.

Formally proving Inexpressibility results: An example!

- Given an LTL formula ψ and a CTL formula φ, we say $\psi \equiv \varphi$ if for all M, s, $M, s \models \psi$ iff $M, s \models \varphi$. For example, $AG(p \rightarrow AFq) \equiv G(p \rightarrow Fq)$.



Theorem: There exists no LTL formula equivalent to $\varphi = (AG\ EF\ p)$.

Proof: Suppose not, let ψ be an LTL formula s.t, $\psi \equiv \varphi$, i.e., $\forall M, s$, $M, s \models \psi$ iff $M, s \models \varphi$.

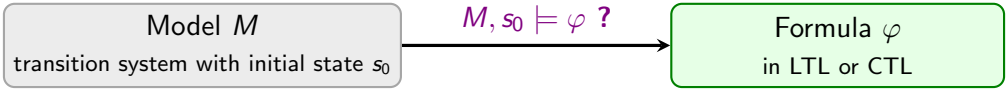
- Consider M_1, s_1. $M_1, s_1 \models (AG\ EF\ p)$ so $M_1, s_1 \models \psi$.
- By definition of satisfaction in LTL, for all paths starting at s_1 in M_1, LTL formula ψ holds.
- Consider M_2: paths from s_1 in M_2 are a subset of paths from s_1 in M_1, hence $M_2, s_1 \models \psi$.
- But $M_2, s_1 \not\models \varphi$, hence contradiction.

Exercise 25.3. Is $(FGp) \equiv (AF\ AG\ p)$? Exhibit a model that distinguishes them.

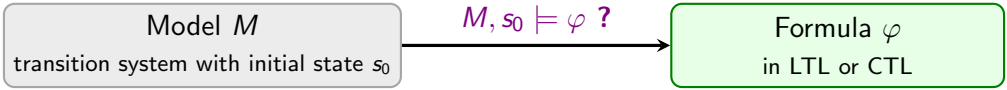
Topic 25.1

Model checking

The model checking problem



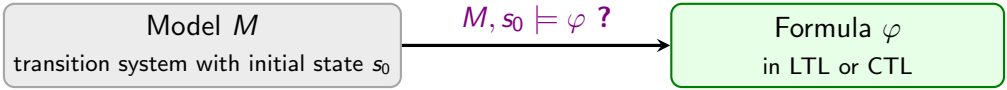
The model checking problem



Basic Model checking Question

Given a formula φ in LTL or CTL, and given a transition system/model M with an initial state s_0: is $M, s_0 \models \varphi$?

The model checking problem

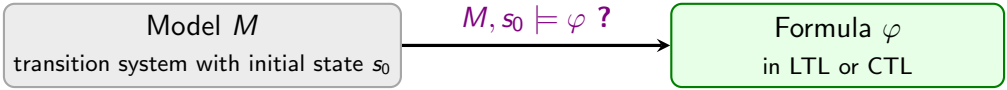


Basic Model checking Question

Given a formula φ in LTL or CTL, and given a transition system/model M with an initial state s_0: is $M, s_0 \models \varphi$?

- LTL model checking beyond the scope of this course. Great for reading further!

The model checking problem

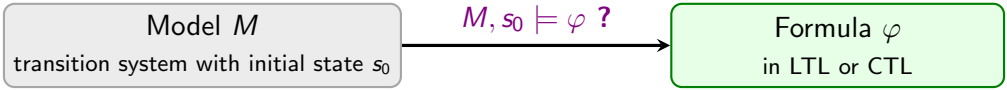


Basic Model checking Question

Given a formula φ in LTL or CTL, and given a transition system/model M with an initial state s_0: is $M, s_0 \models \varphi$?

- LTL model checking beyond the scope of this course. Great for reading further!
- CTL model checking, though, has a simple polynomial algorithm. Coming up!

The model checking problem



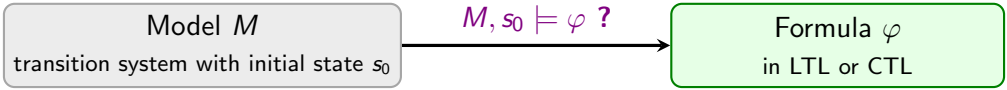
Basic Model checking Question

Given a formula φ in LTL or CTL, and given a transition system/model M with an initial state s_0: is $M, s_0 \models \varphi$?

- LTL model checking beyond the scope of this course. Great for reading further!
- CTL model checking, though, has a simple polynomial algorithm. Coming up!

In fact, we will show something slightly stronger.

The model checking problem



Basic Model checking Question

Given a formula φ in LTL or CTL, and given a transition system/model M with an initial state s_0: is $M, s_0 \models \varphi$?

- LTL model checking beyond the scope of this course. Great for reading further!
- CTL model checking, though, has a simple polynomial algorithm. Coming up!

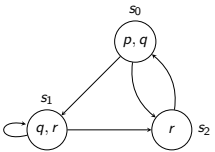
In fact, we will show something slightly stronger.

Given a CTL formula and model, we will output the set of all states that satisfy the formula!

A labelling algo for CTL-model checking

Input: CTL formula φ and model $M = (S, \to, L)$
Output: Set of states of M that satisfy φ.

- Convert φ into a CTL formula φ' that only uses $EG, EU, EX, \wedge, \neg, \bot$.



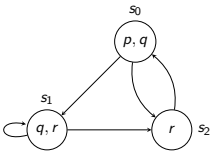
Example: $\varphi = \neg p \wedge EX(q \wedge r)$ (so $\varphi' = \varphi$)

	s_0	s_1	s_2

A labelling algo for CTL-model checking

Input: CTL formula φ and model $M = (S, \to, L)$
Output: Set of states of M that satisfy φ.

- Convert φ into a CTL formula φ' that only uses $EG, EU, EX, \wedge, \neg, \bot$.
- Idea: Label states of M with all subformulas of φ' that are satisfied there, smallest subformulas first.



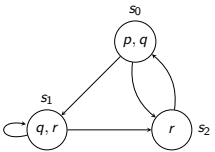
Example: $\varphi = \neg p \wedge EX(q \wedge r)$ (so $\varphi' = \varphi$)

	s_0	s_1	s_2
p			
q			
r			
$\neg p$			
$q \wedge r$			
$EX(q \wedge r)$			
φ			

A labelling algo for CTL-model checking

Input: CTL formula φ and model $M = (S, \to, L)$
Output: Set of states of M that satisfy φ.

- Convert φ into a CTL formula φ' that only uses $EG, EU, EX, \wedge, \neg, \bot$.
- Idea: Label states of M with all subformulas of φ' that are satisfied there, smallest subformulas first.
- Take the next subformula ψ (all its proper subformulas are already labelled):
  - $\psi = \bot$. Nothing to do.



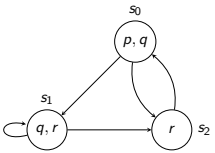
Example: $\varphi = \neg p \wedge EX(q \wedge r)$ (so $\varphi' = \varphi$)

	s_0	s_1	s_2
p			
q			
r			
$\neg p$			
$q \wedge r$			
$EX(q \wedge r)$			
φ			

A labelling algo for CTL-model checking

Input: CTL formula φ and model $M = (S, \to, L)$
Output: Set of states of M that satisfy φ.

- Convert φ into a CTL formula φ' that only uses $EG, EU, EX, \wedge, \neg, \bot$.
- Idea: Label states of M with all subformulas of φ' that are satisfied there, smallest subformulas first.
- Take the next subformula ψ (all its proper subformulas are already labelled):
  - $\psi = \bot$. Nothing to do.
  - $\psi = p$. Label s with p if $p \in L(s)$.



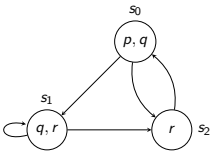
Example: $\varphi = \neg p \wedge EX(q \wedge r)$ (so $\varphi' = \varphi$)

	s_0	s_1	s_2
p	✓		
q	✓	✓	
r		✓	✓
$\neg p$			
$q \wedge r$			
$EX(q \wedge r)$			
φ			

A labelling algo for CTL-model checking

Input: CTL formula φ and model $M = (S, \to, L)$
Output: Set of states of M that satisfy φ.

- Convert φ into a CTL formula φ' that only uses $EG, EU, EX, \wedge, \neg, \bot$.
- Idea: Label states of M with all subformulas of φ' that are satisfied there, smallest subformulas first.
- Take the next subformula ψ (all its proper subformulas are already labelled):
  - $\psi = \bot$. Nothing to do.
  - $\psi = p$. Label s with p if $p \in L(s)$.
  - $\psi = \neg\psi'$. Label s with $\neg\psi'$ unless it is already labelled ψ'.



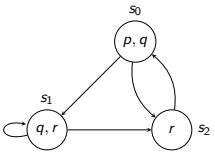
Example: $\varphi = \neg p \wedge EX(q \wedge r)$ (so $\varphi' = \varphi$)

	s_0	s_1	s_2
p	✓		
q	✓	✓	
r		✓	✓
$\neg p$		✓	✓
$q \wedge r$			
$EX(q \wedge r)$			
φ			

A labelling algo for CTL-model checking

Input: CTL formula φ and model $M = (S, \to, L)$
Output: Set of states of M that satisfy φ.

- Convert φ into a CTL formula φ' that only uses $EG, EU, EX, \wedge, \neg, \bot$.
- Idea: Label states of M with all subformulas of φ' that are satisfied there, smallest subformulas first.
- Take the next subformula ψ (all its proper subformulas are already labelled):
  - $\psi = \bot$. Nothing to do.
  - $\psi = p$. Label s with p if $p \in L(s)$.
  - $\psi = \neg\psi'$. Label s with $\neg\psi'$ unless it is already labelled ψ'.
  - $\psi = \psi_1 \wedge \psi_2$. Label s with ψ if it is already labelled with ψ_1 and ψ_2.



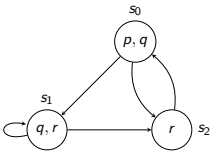
Example: $\varphi = \neg p \wedge EX(q \wedge r)$ (so $\varphi' = \varphi$)

	s_0	s_1	s_2
p	✓		
q	✓	✓	
r		✓	✓
$\neg p$		✓	✓
$q \wedge r$		✓	
$EX(q \wedge r)$			
φ			

A labelling algo for CTL-model checking

Input: CTL formula φ and model $M = (S, \to, L)$
Output: Set of states of M that satisfy φ.

- Convert φ into a CTL formula φ' that only uses $EG, EU, EX, \wedge, \neg, \bot$.
- Idea: Label states of M with all subformulas of φ' that are satisfied there, smallest subformulas first.
- Take the next subformula ψ (all its proper subformulas are already labelled):
  - $\psi = \bot$. Nothing to do.
  - $\psi = p$. Label s with p if $p \in L(s)$.
  - $\psi = \neg\psi'$. Label s with $\neg\psi'$ unless it is already labelled ψ'.
  - $\psi = \psi_1 \wedge \psi_2$. Label s with ψ if it is already labelled with ψ_1 and ψ_2.
  - $\psi = EX\psi_1$. Label s with $EX\,\psi_1$ if some successor of s is labelled ψ_1.



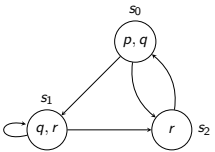
Example: $\varphi = \neg p \wedge EX(q \wedge r)$ (so $\varphi' = \varphi$)

	s_0	s_1	s_2
p	✓		
q	✓	✓	
r		✓	✓
$\neg p$		✓	✓
$q \wedge r$		✓	
$EX(q \wedge r)$	✓	✓	
φ			

A labelling algo for CTL-model checking

Input: CTL formula φ and model $M = (S, \to, L)$
Output: Set of states of M that satisfy φ.

- Convert φ into a CTL formula φ' that only uses $EG, EU, EX, \wedge, \neg, \bot$.
- Idea: Label states of M with all subformulas of φ' that are satisfied there, smallest subformulas first.
- Take the next subformula ψ (all its proper subformulas are already labelled):
  - $\psi = \bot$. Nothing to do.
  - $\psi = p$. Label s with p if $p \in L(s)$.
  - $\psi = \neg\psi'$. Label s with $\neg\psi'$ unless it is already labelled ψ'.
  - $\psi = \psi_1 \wedge \psi_2$. Label s with ψ if it is already labelled with ψ_1 and ψ_2.
  - $\psi = EX\psi_1$. Label s with $EX\,\psi_1$ if some successor of s is labelled ψ_1.



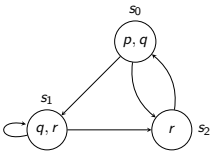
Example: $\varphi = \neg p \wedge EX(q \wedge r)$ (so $\varphi' = \varphi$)

	s_0	s_1	s_2
p	✓		
q	✓	✓	
r		✓	✓
$\neg p$		✓	✓
$q \wedge r$		✓	
$EX(q \wedge r)$	✓	✓	
φ		✓	

A labelling algo for CTL-model checking

Input: CTL formula φ and model $M = (S, \to, L)$
Output: Set of states of M that satisfy φ.

- Convert φ into a CTL formula φ' that only uses $EG, EU, EX, \wedge, \neg, \bot$.
- Idea: Label states of M with all subformulas of φ' that are satisfied there, smallest subformulas first.
- Take the next subformula ψ (all its proper subformulas are already labelled):
  - $\psi = \bot$. Nothing to do.
  - $\psi = p$. Label s with p if $p \in L(s)$.
  - $\psi = \neg\psi'$. Label s with $\neg\psi'$ unless it is already labelled ψ'.
  - $\psi = \psi_1 \wedge \psi_2$. Label s with ψ if it is already labelled with ψ_1 and ψ_2.
  - $\psi = EX\psi_1$. Label s with $EX\,\psi_1$ if some successor of s is labelled ψ_1.



Example: $\varphi = \neg p \wedge EX(q \wedge r)$ (so $\varphi' = \varphi$)

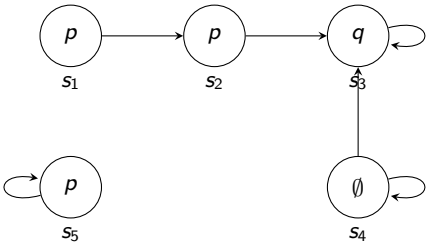
	s_0	s_1	s_2
p	✓		
q	✓	✓	
r		✓	✓
$\neg p$		✓	✓
$q \wedge r$		✓	
$EX(q \wedge r)$	✓	✓	
φ		✓	

Output: $\{s_1\}$, so $M, s_0 \not\models \varphi$

A labelling algo for CTL-model checking (contd.)

- Case $\psi = E[\psi_1\ U\ \psi_2]$. If any s is labelled ψ_2, label it with ψ. Repeat till fixpoint: label any state with ψ if it is labelled ψ_1 and has a successor labelled ψ.

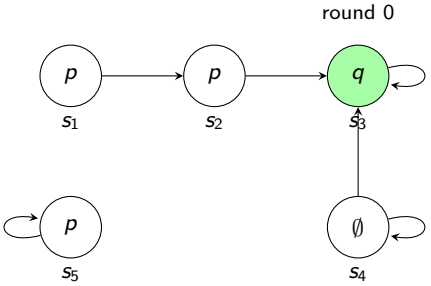
Example: $\psi = E[p\ U\ q]$, so $\psi_1 = p$ and $\psi_2 = q$



A labelling algo for CTL-model checking (contd.)

- Case $\psi = E[\psi_1\ U\ \psi_2]$. If any s is labelled ψ_2, label it with ψ. Repeat till fixpoint: label any state with ψ if it is labelled ψ_1 and has a successor labelled ψ.

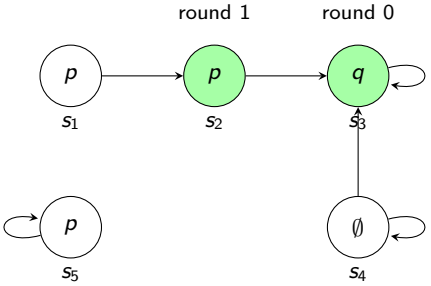
Example: $\psi = E[p\ U\ q]$, so $\psi_1 = p$ and $\psi_2 = q$



A labelling algo for CTL-model checking (contd.)

- Case $\psi = E[\psi_1\ U\ \psi_2]$. If any s is labelled ψ_2, label it with ψ. Repeat till fixpoint: label any state with ψ if it is labelled ψ_1 and has a successor labelled ψ.

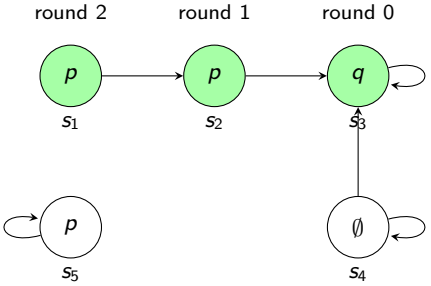
Example: $\psi = E[p\ U\ q]$, so $\psi_1 = p$ and $\psi_2 = q$



A labelling algo for CTL-model checking (contd.)

- Case $\psi = E[\psi_1\ U\ \psi_2]$. If any s is labelled ψ_2, label it with ψ. Repeat till fixpoint: label any state with ψ if it is labelled ψ_1 and has a successor labelled ψ.

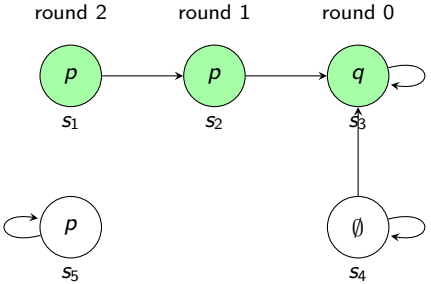
Example: $\psi = E[p\ U\ q]$, so $\psi_1 = p$ and $\psi_2 = q$



A labelling algo for CTL-model checking (contd.)

- Case $\psi = E[\psi_1\ U\ \psi_2]$. If any s is labelled ψ_2, label it with ψ. Repeat till fixpoint: label any state with ψ if it is labelled ψ_1 and has a successor labelled ψ.

Example: $\psi = E[p\ U\ q]$, so $\psi_1 = p$ and $\psi_2 = q$



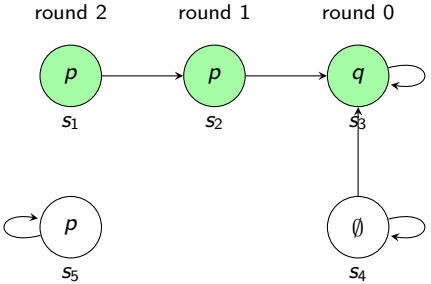
Round 3 adds nothing: output $\{s_1, s_2, s_3\}$

s_4 is not p; s_5 is p but never reaches q

A labelling algo for CTL-model checking (contd.)

- Case $\psi = E[\psi_1\ U\ \psi_2]$. If any s is labelled ψ_2, label it with ψ. Repeat till fixpoint: label any state with ψ if it is labelled ψ_1 and has a successor labelled ψ.
- Last case: $\psi = EG\,\psi_1$: next slide.

Example: $\psi = E[p\ U\ q]$, so $\psi_1 = p$ and $\psi_2 = q$



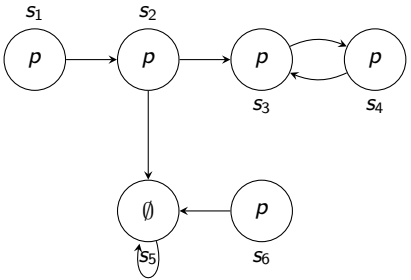
Round 3 adds nothing: output $\{s_1, s_2, s_3\}$

s_4 is not p; s_5 is p but never reaches q

A labelling algo for CTL-model checking: the EG case

- Last case: $\psi = EG\psi_1$.

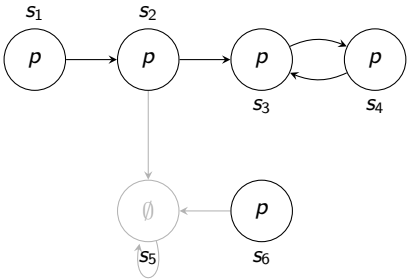
Example: $\psi = EG\, p$, so $\psi_1 = p$



A labelling algo for CTL-model checking: the EG case

- Last case: $\psi = EG\psi_1$.
  - Consider the subgraph consisting of all states labelled ψ_1.

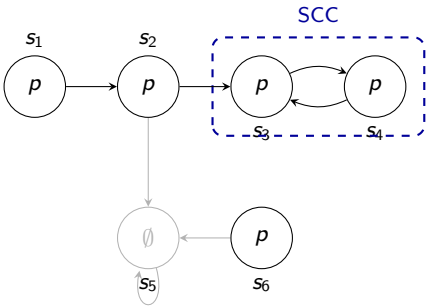
Example: $\psi = EG\, p$, so $\psi_1 = p$



A labelling algo for CTL-model checking: the EG case

- Last case: $\psi = EG\psi_1$.
  - Consider the subgraph consisting of all states labelled ψ_1.
  - In this, find maximum strongly connected components (MSCCs)[1].

Example: $\psi = EG\, p$, so $\psi_1 = p$

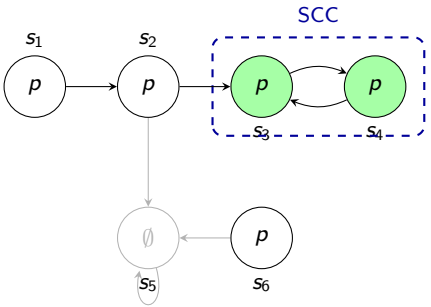


[1] A set of states is *strongly connected* if every state in it has a path to every other state in it. A maximal such set is an MSCC; we only need those with at least one edge (a single state counts only if it has a self-loop).

A labelling algo for CTL-model checking: the EG case

- Last case: $\psi = EG\psi_1$.
  - Consider the subgraph consisting of all states labelled ψ_1.
  - In this, find maximum strongly connected components (MSCCs)[1].
  - Use backwards breadth-first search (BFS) on this subgraph to find all states that can reach such an MSCC. Label each such state with ψ.

Example: $\psi = EG\ p$, so $\psi_1 = p$

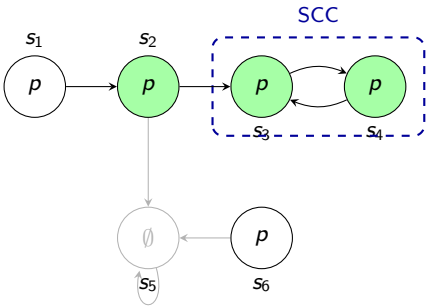


[1] A set of states is *strongly connected* if every state in it has a path to every other state in it. A maximal such set is an MSCC; we only need those with at least one edge (a single state counts only if it has a self-loop).

A labelling algo for CTL-model checking: the EG case

- Last case: $\psi = EG\psi_1$.
  - Consider the subgraph consisting of all states labelled ψ_1.
  - In this, find maximum strongly connected components (MSCCs)[1].
  - Use backwards breadth-first search (BFS) on this subgraph to find all states that can reach such an MSCC. Label each such state with ψ.

Example: $\psi = EG\ p$, so $\psi_1 = p$

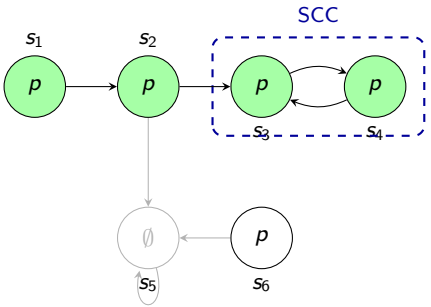


[1] A set of states is *strongly connected* if every state in it has a path to every other state in it. A maximal such set is an MSCC; we only need those with at least one edge (a single state counts only if it has a self-loop).

A labelling algo for CTL-model checking: the EG case

- Last case: $\psi = EG\psi_1$.
  - Consider the subgraph consisting of all states labelled ψ_1.
  - In this, find maximum strongly connected components (MSCCs)[1].
  - Use backwards breadth-first search (BFS) on this subgraph to find all states that can reach such an MSCC. Label each such state with ψ.

Example: $\psi = EG\ p$, so $\psi_1 = p$

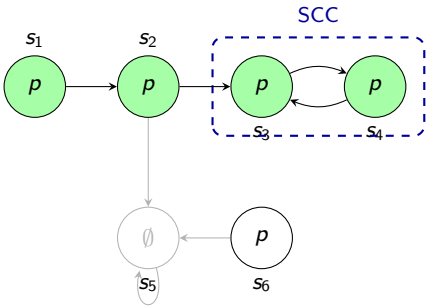


[1] A set of states is *strongly connected* if every state in it has a path to every other state in it. A maximal such set is an MSCC; we only need those with at least one edge (a single state counts only if it has a self-loop).

A labelling algo for CTL-model checking: the EG case

- Last case: $\psi = EG\psi_1$.
  - Consider the subgraph consisting of all states labelled ψ_1.
  - In this, find maximum strongly connected components (MSCCs)[1].
  - Use backwards breadth-first search (BFS) on this subgraph to find all states that can reach such an MSCC. Label each such state with ψ.

Example: $\psi = EG\ p$, so $\psi_1 = p$



Output: $\{s_1, s_2, s_3, s_4\}$

s_6 is p, but its only successor s_5 is not: it cannot stay in p forever

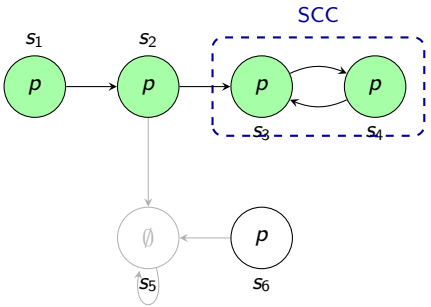
[1] A set of states is *strongly connected* if every state in it has a path to every other state in it. A maximal such set is an MSCC; we only need those with at least one edge (a single state counts only if it has a self-loop).

A labelling algo for CTL-model checking: the EG case

- Last case: $\psi = EG\psi_1$.
  - Consider the subgraph consisting of all states labelled ψ_1.
  - In this, find maximum strongly connected components (MSCCs)[1].
  - Use backwards breadth-first search (BFS) on this subgraph to find all states that can reach such an MSCC. Label each such state with ψ.

This completes the algo. At end we check if initial state is labelled by input CTL formula φ'.

Example: $\psi = EG\, p$, so $\psi_1 = p$



Output: $\{s_1, s_2, s_3, s_4\}$

s_6 is p, but its only successor s_5 is not: it cannot stay in p forever

[1] A set of states is *strongly connected* if every state in it has a path to every other state in it. A maximal such set is an MSCC; we only need those with at least one edge (a single state counts only if it has a self-loop).