Formal Methods for System Design

Chapter 4: Computation tree logic

Mickael Randour

Mathematics Department, UMONS

October 2023

UMONS
Université de Mons

1 CTL: a specification language for BT properties

2 CTL model checking

3 CTL vs. LTL

4 CTL*

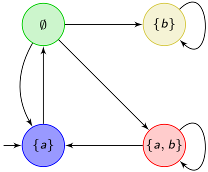
1 CTL: a specification language for BT properties

2 CTL model checking

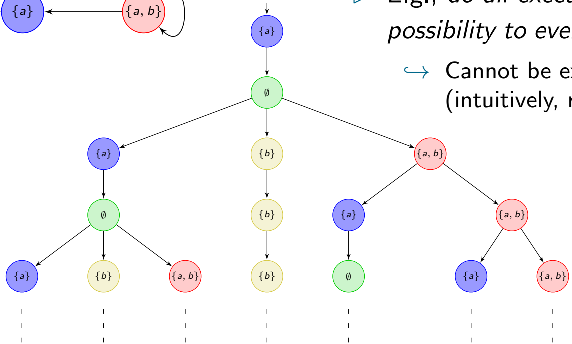
3 CTL vs. LTL

4 CTL*

Branching time semantics: a reminder



- **Branching time semantics** deals with the *execution (or computation) tree*.
  - ▷ Infinite unfolding considering all branching possibilities.
  - ▷ E.g., *do all executions always have the possibility to eventually reach* {b}*?* Yes.
    - ↪ Cannot be expressed as a LT property (intuitively, requires *branching*).



Intuition

- In LTL, $s \models \phi$ means that **all** paths starting in s satisfy ϕ.
  - ▷ Implicit universal quantification.
  - ▷ *Could be made explicit by writing $s \models \forall\, \phi$.*
- What if we want to talk about **some** paths?
  - ▷ E.g., *does there exist* a path satisfying ϕ starting in s?
  - ▷ *Could be expressed using the duality between universal and existential quantification: $s \models \exists\, \phi$ iff $s \not\models \forall\, \neg\phi$.*
- What if the property is more complex? E.g., do *all* executions always have the *possibility* to eventually reach {b}?
  - ▷ $s \models \forall\Box\Diamond b$ **does not work** as it requires all paths to always return in {b}, not just to have the *possibility* to do so.
  - ▷ **Not expressible in LTL.** We need **nesting of path quantifiers** ($\forall$,$\exists$).
  - ↪ $s \models \forall\Box\, \exists\Diamond b$ is a **CTL formula**: "for *all* paths, it is always the case (i.e., at every step along the branch) that *there exists* a path (which can be branching) that eventually reaches *b*."

CTL vs. LTL

Different notions of time

- In LTL, we reason about paths and their traces.
  - ▷ Time is **linear**: along a trace, any point has only one possible future.
- In CTL, we reason about the computation tree and its branching behavior.
  - ▷ Time is **branching**: any point along an execution (i.e., node in the tree) has several possible futures.

$\Longrightarrow$ **We will see that the expressiveness of LTL and CTL are incomparable. . .**

. . . and we will sketch CTL*, a logic which subsumes both LTL and CTL.

CTL in a nutshell (1/2)

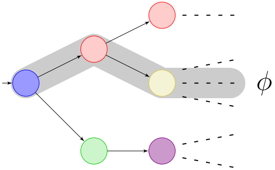
In CTL, we have two types of formulae.

State formulae are assertions about atomic propositions in states *and their branching structure*.

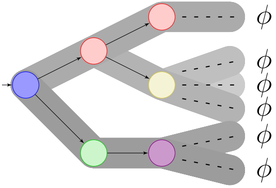
↪ Written in uppercase Greek letters: e.g., Φ, Ψ.

- **Atomic propositions** $a \in AP$ (represented as $\{a\}$, $\{b\}$, etc).
- **Boolean combinations of formulae**: $\neg\Phi$, $\Phi \wedge \Psi$, $\Phi \vee \Psi$.
- **Path quantification** using *path formulae*.
  - ↪ Path formulae written in lowercase Greek letters: e.g., ϕ, ψ.

Existential quantification $\exists\phi$.



Universal quantification $\forall\phi$.

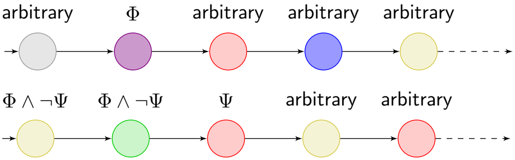


CTL in a nutshell (2/2)

Path formulae use temporal operators.

next $\bigcirc \Phi$

until $\Phi \cup \Psi$



Differences between CTL path formulae and LTL formulae

Path formulae

- cannot be combined with boolean connectives;
- do not allow nesting of temporal modalities.

In CTL, every temporal operator must be in the immediate scope of a path quantifier!

E.g., $s \models \forall\Box\,\exists\Diamond b$ is a valid CTL formula but $s \models \forall\Box\Diamond b$ is not.

CTL syntax

Core syntax

CTL syntax

Given the set of atomic propositions AP, CTL *state formulae* are formed according to the following grammar:

$$\Phi ::= \mathsf{true} \mid a \mid \Phi \wedge \Psi \mid \neg\Phi \mid \exists\phi \mid \forall\phi$$

where $a \in AP$ and ϕ is a path formula. CTL *path formulae* are formed according to the following grammar:

$$\phi ::= \bigcirc \Phi \mid \Phi \,\mathsf{U}\, \Psi$$

where Φ and Ψ are state formulae.

$\Longrightarrow$ **The syntax enforces the presence of a path quantifier before every temporal operator.**

$\hookrightarrow$ When we just say *CTL formula*, we mean CTL *state* formula.

CTL syntax

Examples (1/2)

CTL syntax reminder

$$\Phi ::= \text{true} \mid a \mid \Phi \wedge \Psi \mid \neg\Phi \mid \exists\phi \mid \forall\phi \qquad \phi ::= \bigcirc \Phi \mid \Phi \,\mathsf{U}\, \Psi$$

- Is $\Phi = \exists\bigcirc a$ a valid CTL formula?
  - ▷ **Yes**, because $\phi = \bigcirc a$ is a valid path formula, hence $\Phi = \exists\phi$ is a valid state formula.
- Is $\Phi = a \wedge b$ a valid CTL formula?
  - ▷ **Yes**, because $\Psi_1 = a$ and $\Psi_2 = b$ are valid state formulae, hence $\Phi = \Psi_1 \wedge \Psi_2$ is a valid state formula.
- Is $\Phi = \forall(a \wedge \exists\bigcirc b)$ a valid CTL formula?
  - ▷ **No**, because $\phi = a \wedge \exists\bigcirc b$ is not a valid path formula (should be $\bigcirc \Psi$ or $\Psi_1 \,\mathsf{U}\, \Psi_2$).

CTL syntax

Examples (2/2)

CTL syntax reminder

$\Phi ::= \text{true} \mid a \mid \Phi \wedge \Psi \mid \neg\Phi \mid \exists\phi \mid \forall\phi \qquad \phi ::= \bigcirc\Phi \mid \Phi \,\mathsf{U}\, \Psi$

- Is $\Phi = \exists((\forall\bigcirc a) \,\mathsf{U}\, (a \wedge b))$ a valid CTL formula?
  - ▷ **Yes**, because $\Psi_1 = \forall\bigcirc a$ and $\Psi_2 = a \wedge b$ are valid state formulae, hence $\phi = \Psi_1 \,\mathsf{U}\, \Psi_2$ is a valid path formula, hence $\Phi = \exists\phi$ is a valid state formula.
- Is $\Phi = \exists\bigcirc (a \,\mathsf{U}\, b)$ a valid CTL formula?
  - ▷ **No**, because $\phi = a \,\mathsf{U}\, b$ is a valid *path* formula whereas we require a *state* formula at this position. I.e., one needs to insert quantification for the U operator.

CTL syntax

Derived operators

Boolean operators false, $\vee$, $\oplus$, $\rightarrow$, $\leftrightarrow$ derived as for LTL.

Other derivations also similar:

$$\exists\Diamond\Phi \equiv \exists(\text{true}\,\mathsf{U}\,\Phi)$$ ***potentially***

$$\forall\Diamond\Phi \equiv \forall(\text{true}\,\mathsf{U}\,\Phi)$$ ***inevitably***

$$\exists\Box\Phi \equiv \neg\forall\Diamond\neg\Phi$$ ***potentially always***

$$\forall\Box\Phi \equiv \neg\exists\Diamond\neg\Phi$$ ***invariantly***

$$\exists(\Phi\,\mathsf{W}\,\Psi) \equiv \neg\forall\big((\Phi\wedge\neg\Psi)\,\mathsf{U}\,(\neg\Phi\wedge\neg\Psi)\big)$$ ***weak until***

$$\forall(\Phi\,\mathsf{W}\,\Psi) \equiv \neg\exists\big((\Phi\wedge\neg\Psi)\,\mathsf{U}\,(\neg\Phi\wedge\neg\Psi)\big)$$

Would $\forall\Box\Phi \equiv \forall\neg\Diamond\neg\Phi$ be a correct derivation (similar to LTL)?

No! Because $\neg$ cannot be applied to *path* formulae.

$\Longrightarrow$ **Derivations are based on the duality between $\exists$ and $\forall$.**

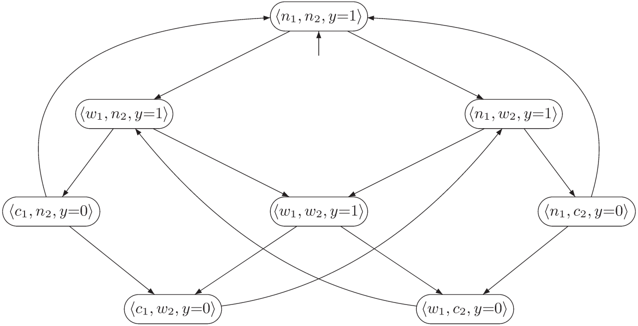
CTL syntax

Precedence order

Same rules as for LTL, with quantifiers $\exists$, $\forall$ directly linked to the following path formula.

Formalizing LT/BT properties in CTL

Safety

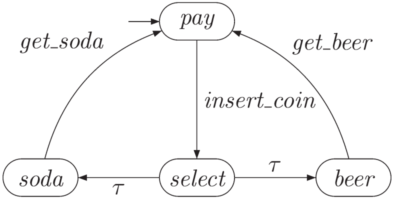


TS for semaphore-based mutex [BK08] (Ch. 2).

- ▷ $AP = \{crit_1, crit_2\}$, natural labeling.
- ▷ In LTL, $\neg\Diamond(crit_1 \wedge crit_2)$ or $\Box(\neg crit_1 \vee \neg crit_2)$.
- ↪ In CTL, $\neg\exists\Diamond(crit_1 \wedge crit_2)$ or $\forall\Box(\neg crit_1 \vee \neg crit_2)$.

Formalizing LT/BT properties in CTL

Liveness



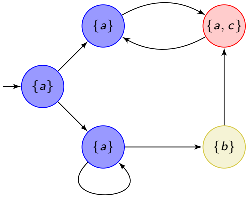
Beverage vending machine [BK08] (Ch. 2).

- ▷ $AP = \{paid, drink\}$, natural labeling.
- ▷ In LTL, $\Box\Diamond drink$.
- ↪ In CTL, $\forall\Box\forall\Diamond drink$. Intuitively, for all paths, it is true at every step that all futures will eventually reach *drink*.

$\Longrightarrow$ Formal proof after proper definition of the semantics.

Formalizing LT/BT properties in CTL

Persistence (1/3)



Ensure that from some point on, a holds but b does not.

- ▷ In LTL, $\Diamond\Box(a \wedge \neg b)$.

↪ In CTL. . . ?

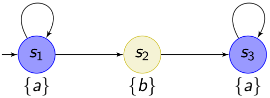
This property cannot be expressed in CTL!

$\Longrightarrow$ Informal argument in the next slide. . .

Formalizing LT/BT properties in CTL

Persistence (2/3)

Take a simpler TS $\mathcal{T}$:



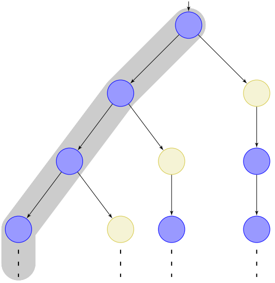
It clearly satisfies LTL formula $\phi = \Diamond\Box a$.

As all paths, the highlighted one must satisfy $\Diamond\forall\Box a$ for Φ to hold.

But there is no state along this path where $\forall\Box a$ holds as we can always branch to b! $\Longrightarrow \mathcal{T} \not\models \Phi$.

Best guess for equivalent CTL formula: $\Phi = \forall\Diamond\forall\Box a$ (we want this to be true on *all* paths).

But what is the execution tree?



Formalizing LT/BT properties in CTL

Persistence (3/3)

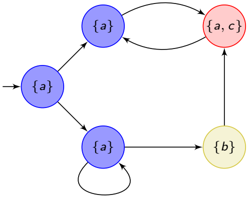
Intuition.

- In LTL, time is *linear*.
  - ▷ Either we have a path that do branch to b, thus $\Box a$ is true after b. Or we never branch and $\Box a$ is true from the initial state.
- In CTL, time is *branching*.
  - ▷ We have to use the $\forall$ quantifier (as we want to characterize all paths).
  - ▷ But then $\Diamond\forall\Box a$ asks to reach a state where **all possible futures** satisfy $\Box a$.
  - ▷ Not possible because of the possibility of branching.

Hence, **even if all branches satisfy $\Diamond\Box a$**, the CTL formula **requires the additional (and not verified) existence of nodes in the tree whose subtrees only contain paths satisfying $\Box a$**.

Formalizing LT/BT properties in CTL

Typical BT property



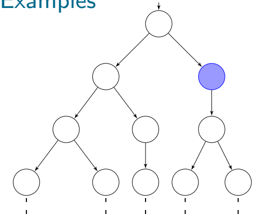
Along all paths, it is always *possible* to reach $\{a, c\}$.

- ▷ Not expressible in LTL: in linear time, either you reach or you do not. Reasoning about possible futures requires branching time.

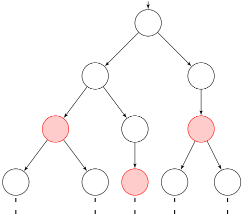
↪ In CTL, $\forall\Box\exists\Diamond(a \wedge c)$.

CTL semantics

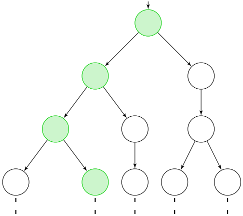
Examples



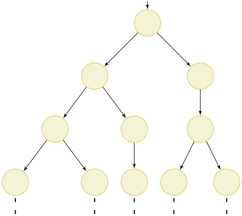
$\exists\bigcirc\, blue$



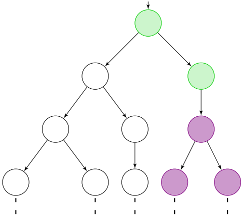
$\forall\Diamond\, red$



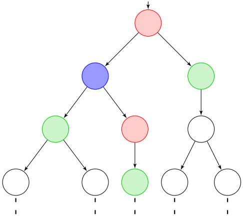
$\exists\Box\, green$



$\forall\Box\, yellow$



$\exists(green \,\mathsf{U}\, \forall\Box\, violet)$



$\forall\big((red \vee blue) \,\mathsf{U}\, green\big)$

CTL semantics

For state formulae

Let $\mathcal{T} = (S, Act, \longrightarrow, I, AP, L)$ be a TS without terminal states, $a \in AP$, $s \in S$, Φ and Ψ be CTL state formulae and ϕ be a CTL path formula.

Satisfaction for state formulae

$s \models \Phi$ iff formula Φ holds in state s.

$$
\begin{array}{lll}
s \models \text{true} & & \\
s \models a & \text{iff} & a \in L(s) \\
s \models \Phi \wedge \Psi & \text{iff} & s \models \Phi \text{ and } s \models \Psi \\
s \models \neg\Phi & \text{iff} & s \not\models \Phi \\
s \models \exists\phi & \text{iff} & \exists\, \pi \in Paths(s),\ \pi \models \phi \\
s \models \forall\phi & \text{iff} & \forall\, \pi \in Paths(s),\ \pi \models \phi
\end{array}
$$

CTL semantics

For path formulae

Let $\pi = s_0 s_1 s_2 \ldots$.

Satisfaction for path formulae

$\pi \models \phi$ iff path π satisfies ϕ.

$$\begin{array}{lll} \pi \models \bigcirc\Phi & \text{iff} & s_1 \models \Phi \\ \pi \models \Phi \,\mathsf{U}\, \Psi & \text{iff} & \exists j \geq 0,\ s_j \models \Psi \text{ and } \forall\, 0 \leq i < j,\ s_i \models \Phi \\ \pi \models \Diamond\Phi & \text{iff} & \exists j \geq 0,\ s_j \models \Phi \\ \pi \models \Box\Phi & \text{iff} & \forall j \geq 0,\ s_j \models \Phi \end{array}$$

CTL semantics

For transition systems

Let $\mathcal{T} = (S, Act, \longrightarrow, I, AP, L)$ be a TS and Φ a CTL state formula over AP.

Definition: satisfaction set

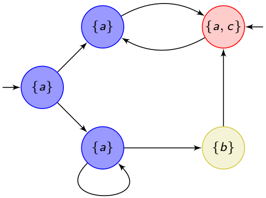
The **satisfaction set** $Sat_{\mathcal{T}}(\Phi)$ (or briefly, $Sat(\Phi)$) for formula Φ is

$$Sat(\Phi) = \{s \in S \mid s \models \Phi\}.$$

TS $\mathcal{T}$ satisfies Φ, denoted $\mathcal{T} \models \Phi$, iff Φ holds in all initial states, i.e.,

$$\mathcal{T} \models \Phi \;\text{ iff }\; I \subseteq Sat(\Phi).$$

Example



Notice the two initial states.

$\mathcal{T} \models \forall \bigcirc a$	$\mathcal{T} \not\models \exists \Diamond b$	$\mathcal{T} \not\models \forall (a \, \mathsf{U} \, b)$
$\mathcal{T} \not\models \exists (a \, \mathsf{U} \, b)$	$\mathcal{T} \not\models \forall \Box a$	$\mathcal{T} \models \forall \Box \exists \Diamond \forall \Box \forall \Diamond c$
$\mathcal{T} \models \exists \Box a$	$\mathcal{T} \models \forall (a \, \mathsf{W} \, b)$	$\mathcal{T} \models \forall \Box (c \rightarrow \forall \bigcirc a)$
$\mathcal{T} \models \exists (a \, \mathsf{U} \, c)$	$\mathcal{T} \models \exists \Box \neg b$	$\mathcal{T} \models \exists \Box \exists \Diamond b \rightarrow \neg c$

$\Longrightarrow$ **Blackboard solution.**

Playing with the semantics

Infinitely often (1/3)

Earlier, we claimed that the CTL formula $\Phi = \forall\Box\forall\Diamond a$ is *equivalent* to the LTL formula $\phi = \Box\Diamond a$, i.e., for all TS $\mathcal{T}$, $\mathcal{T} \models \Phi$ iff $\mathcal{T} \models \phi$.

$\Longrightarrow$ **Let's prove it!**

We prove the more precise statement: $\forall\, s \in S,\ s \models \Phi \iff s \models \phi$, which implies the result for TSs.

Playing with the semantics

Infinitely often (2/3)

$s \models \Phi \implies s \models \phi.$

1. Let $s \models \Phi$. We must prove that $\forall \pi = s_0 s_1 s_2 \ldots \in Paths(s)$, $\pi \models \phi$, i.e., for all $j \geq 0$, there exists $i \geq j$ such that $s_i \models a$.
2. Since $s \models \forall\Box\forall\Diamond a$ and $\pi \in Paths(s)$, we have $\pi \models \Box\forall\Diamond a$.
3. Hence, $s_j \models \forall\Diamond a$.
4. Since $\pi[j..] = s_j s_{j+1} \ldots \in Paths(s_j)$, we have that $\pi[j..] \models \Diamond a$.
5. Hence, there exists $i \geq j$ such that $s_i \models a$.
6. This holds for all j so we are done.

Playing with the semantics

Infinitely often (3/3)

$s \models \Phi \impliedby s \models \phi.$

1. Let $s \models \phi$. We must prove that $s \models \forall\Box\forall\Diamond a$, i.e, that $\forall\, \pi = s_0 s_1 s_2 \ldots \in \mathit{Paths}(s)$, $\pi \models \Box\forall\Diamond a$.
2. I.e., that for all $j \geq 0$, $s_j \models \forall\Diamond a$.
3. Let $j \geq 0$ and fix any path $\pi' = s_j s'_{j+1} s'_{j+2} \ldots \in \mathit{Paths}(s_j)$. We must show that $\pi' \models \Diamond a$.
4. But, then $\pi'' = s_0 s_1 \ldots s_j s'_{j+1} s'_{j+2} \ldots \in \mathit{Paths}(s)$. Hence, $\pi'' \models \Box\Diamond a$ by hypothesis.
5. Hence, there exists $i > j$ such that $s'_i \models a$.
6. Therefore, $\pi' \models \Diamond a$.
7. This holds for any path $\pi' \in \mathit{Paths}(s_j)$ so $s_j \models \forall\Diamond a$.
8. Since it holds for all j, $\pi \models \Box\forall\Diamond a$.
9. Finally, it holds for all $\pi \in \mathit{Paths}(s)$, thus $s \models \Phi$.

Semantics of negation

States

Negation for states

For $s \in S$ and a CTL formula Φ over AP,

$$s \not\models \Phi \iff s \models \neg\Phi.$$

Intuitively, due to the duality between $\forall$ and $\exists$ and the semantics of negation for path formulae (see LTL, either a path satisfies ϕ or it satisfies $\neg\phi$).

Semantics of negation

Transition systems

Negation for TSs

For TS $\mathcal{T} = (S, Act, \longrightarrow, I, AP, L)$ and a CTL formula Φ over AP:

$$\mathcal{T} \not\models \Phi$$
$$\not\Downarrow \; \Uparrow$$
$$\mathcal{T} \models \neg\Phi$$

We have that $\mathcal{T} \not\models \Phi$ iff $I \not\subseteq Sat(\Phi)$

iff $\exists s \in I,\; s \not\models \Phi$

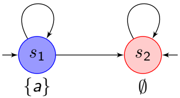
iff $\exists s \in I,\; s \models \neg\Phi$

But it may be the case that $\mathcal{T} \not\models \Phi$ and $\mathcal{T} \not\models \neg\Phi$ if

$\exists s_1, s_2 \in I$ such that $s_1 \models \Phi$ and $s_2 \models \neg\Phi$.

Semantics of negation

Example



Consider CTL formula $\Phi = \exists\square a$. Do we have that $\mathcal{T} \models \Phi$?

Beware of erroneous intuition!

$\mathcal{T} \models \exists\phi \;\not\Longleftrightarrow\; \exists\,\sigma \in \mathit{Traces}(\mathcal{T}),\ \sigma \models \phi$.

Indeed, Φ must hold in **all** initial states.

$\hookrightarrow$ Here it does not in $s_2 \implies \mathcal{T} \not\models \Phi$.

Do we have that $\mathcal{T} \models \neg\Phi = \forall\lozenge\neg a$?

$\hookrightarrow$ No. Because of path $(s_1)^\omega$, $s_1 \not\models \neg\Phi \implies \mathcal{T} \not\models \neg\Phi$.

Surprising equivalence.

$\mathcal{T} \not\models \neg\exists\phi \iff \exists\,\sigma \in \mathit{Traces}(\mathcal{T}),\ \sigma \models \phi$.

Equivalence of CTL formulae

Definition

Equivalence of CTL formulae

CTL (state) formulae Φ and Ψ over AP are *equivalent*, denoted $\Phi \equiv \Psi$, if and only if, for all TS $\mathcal{T}$ over AP,

$$Sat(\Phi) = Sat(\Psi).$$

In particular, $\Phi \equiv \Psi \iff (\forall \mathcal{T},\ \mathcal{T} \models \Phi \iff \mathcal{T} \models \Psi)$.

$\Longrightarrow$ **Let us review some computational rules.**

Equivalence of CTL formulae

Duality for path quantifiers

$$
\begin{aligned}
\forall \bigcirc \Phi &\equiv \neg\exists \bigcirc \neg\Phi \\
\exists \bigcirc \Phi &\equiv \neg\forall \bigcirc \neg\Phi \\
\forall \Diamond \Phi &\equiv \neg\exists \Box \neg\Phi \\
\exists \Diamond \Phi &\equiv \neg\forall \Box \neg\Phi \\
\forall(\Phi \,\mathsf{U}\, \Psi) &\equiv \neg\exists(\neg\Psi \,\mathsf{U}\, (\neg\Phi \wedge \neg\Psi)) \wedge \neg\exists\Box\neg\Psi \\
&\equiv \neg\exists((\Phi \wedge \neg\Psi) \,\mathsf{U}\, (\neg\Phi \wedge \neg\Psi)) \wedge \neg\exists\Box(\Phi \wedge \neg\Psi) \\
&\equiv \neg\exists((\Phi \wedge \neg\Psi) \,\mathsf{W}\, (\neg\Phi \wedge \neg\Psi)) \\
\exists(\Phi \,\mathsf{U}\, \Psi) &\equiv \neg\forall((\Phi \wedge \neg\Psi) \,\mathsf{W}\, (\neg\Phi \wedge \neg\Psi))
\end{aligned}
$$

Equivalence of CTL formulae

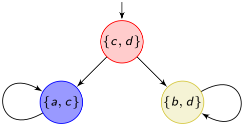
Distribution

$$\forall\Box(\Phi \wedge \Psi) \quad \equiv \quad \forall\Box\Phi \wedge \forall\Box\Psi$$
$$\exists\Diamond(\Phi \vee \Psi) \quad \equiv \quad \exists\Diamond\Phi \vee \exists\Diamond\Psi$$

Similar to LTL $\Box(\phi \wedge \psi) \equiv \Box\phi \wedge \Box\psi$ and $\Diamond(\phi \vee \psi) \equiv \Diamond\phi \vee \Diamond\psi$.

But not all laws can be lifted!

$$\exists\Box(\Phi \wedge \Psi) \quad \not\equiv \quad \exists\Box\Phi \wedge \exists\Box\Psi$$
$$\forall\Diamond(\Phi \vee \Psi) \quad \not\equiv \quad \forall\Diamond\Phi \vee \forall\Diamond\Psi$$



$\mathcal{T} \models \forall\Diamond(a \vee b)$ but $\mathcal{T} \not\models \forall\Diamond a \vee \forall\Diamond b$

$\mathcal{T} \models \exists\Box c \wedge \exists\Box d$ but $\mathcal{T} \not\models \exists\Box(c \wedge d)$

Equivalence of CTL formulae

Expansion laws

In LTL, we had:

$$\begin{aligned}
\phi \,\mathsf{U}\, \psi &\equiv \psi \vee (\phi \wedge \bigcirc (\phi \,\mathsf{U}\, \psi)) \\
\Diamond \phi &\equiv \phi \vee \bigcirc \Diamond \phi \\
\Box \phi &\equiv \phi \wedge \bigcirc \Box \phi
\end{aligned}$$

In CTL, we have:

$$\begin{aligned}
\forall(\Phi \,\mathsf{U}\, \Psi) &\equiv \Psi \vee (\Phi \wedge \forall\bigcirc \forall(\Phi \,\mathsf{U}\, \Psi)) \\
\forall\Diamond \Phi &\equiv \Phi \vee \forall\bigcirc \forall\Diamond \Phi \\
\forall\Box \Phi &\equiv \Phi \wedge \forall\bigcirc \forall\Box \Phi \\
\exists(\Phi \,\mathsf{U}\, \Psi) &\equiv \Psi \vee (\Phi \wedge \exists\bigcirc \exists(\Phi \,\mathsf{U}\, \Psi)) \\
\exists\Diamond \Phi &\equiv \Phi \vee \exists\bigcirc \exists\Diamond \Phi \\
\exists\Box \Phi &\equiv \Phi \wedge \exists\bigcirc \exists\Box \Phi
\end{aligned}$$

Existential normal form (ENF)

ENF for CTL

Goal

Retain the full expressiveness of CTL but permit *only existential quantifiers* (thanks to negation and duality).

ENF for CTL

Given atomic propositions AP, CTL formulae in *existential normal form* are given by:

$$\Phi ::= \mathsf{true} \mid a \mid \Phi \wedge \Psi \mid \neg\Phi \mid \exists\bigcirc \Phi \mid \exists(\Phi \mathbin{\mathsf{U}} \Psi) \mid \exists\Box\Phi$$

where $a \in AP$.

Every CTL formula can be rewritten in ENF... but the translation can cause an exponential blowup (because of the rewrite rule for $\forall \mathsf{U}$).

Positive normal form (PNF)

Weak-until PNF for CTL (1/2)

Goal

Retain the full expressiveness of CTL but permit *only negations of atomic propositions*.

Weak-until PNF for LTL

Given atomic propositions AP, CTL state formulae in *weak-until positive normal form* are given by:

$$\Phi ::= \mathsf{true} \mid \mathsf{false} \mid a \mid \neg a \mid \Phi \wedge \Psi \mid \Phi \vee \Psi \mid \exists \phi \mid \forall \phi$$

where $a \in AP$ and path formulae are given by:

$$\phi ::= \bigcirc \Phi \mid \Phi \,\mathsf{U}\, \Psi \mid \Phi \,\mathsf{W}\, \Psi.$$

Positive normal form (PNF)

Weak-until PNF for CTL (2/2)

Every CTL formula can be rewritten in PNF... but the translation can cause an exponential blowup (because of the rewrite rules for $\forall\,\mathsf{U}$ and $\exists\,\mathsf{U}$).

$\Longrightarrow$ **As for LTL, can be avoided by introducing a "release" operator.**

$$\exists(\Phi\,\mathsf{R}\,\Psi) \equiv \neg\forall((\neg\Phi)\,\mathsf{U}\,(\neg\Psi))$$
$$\forall(\Phi\,\mathsf{R}\,\Psi) \equiv \neg\exists((\neg\Phi)\,\mathsf{U}\,(\neg\Psi))$$

1 CTL: a specification language for BT properties

2 CTL model checking

3 CTL vs. LTL

4 CTL*

CTL model checking

Decision problem

Definition: CTL model checking problem

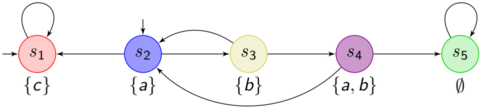
Given a TS $\mathcal{T}$ and a CTL formula Φ, decide if $\mathcal{T} \models \Phi$ or not.

$\Longrightarrow$ Model checking algorithm via **recursive computation of the satisfaction set $Sat(\Phi)$**.

Intuition.

- Use the *parse tree* of Φ (decomposition in subformulae).
- Compute $Sat(a)$ for all leaves in the tree ($a \in AP$).
- Compute satisfaction sets of nodes in a bottom-up fashion, using the satisfactions sets of their children.
- In the root, obtain $Sat(\Phi)$ and check that $I \subseteq Sat(\Phi)$ to conclude whether $\mathcal{T} \models \Phi$ or not.

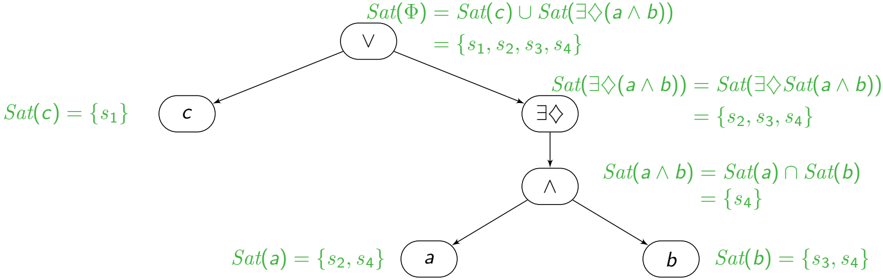
Toy example



CTL formula $\Phi = c \lor \exists\Diamond(a \land b)$.

$\Longrightarrow$ **We have to check that $I = \{s_1, s_2\} \subseteq Sat(\Phi)$.**

Parse tree of Φ:



$\Longrightarrow$ **Finally $I \subseteq Sat(\Phi)$, thus $\mathcal{T} \models \Phi$.**

Formulae in ENF

Throughout this section, we assume formulae are written in ENF.

Reminder: ENF for CTL

Given atomic propositions AP, CTL formulae in *existential normal form* are given by:

$$\Phi ::= \mathsf{true} \mid a \mid \Phi \wedge \Psi \mid \neg\Phi \mid \exists\bigcirc\Phi \mid \exists(\Phi \mathbin{\mathsf{U}} \Psi) \mid \exists\Box\Phi$$

where $a \in AP$.

Assume we have $Sat(\Phi)$ and $Sat(\Psi)$, we need algorithms for:

- $Sat(\Phi \wedge \Psi)$ and $Sat(\neg\Phi)$: easy, *intersection* and *complement*.
- $Sat(\exists\bigcirc\Phi)$, $Sat(\exists(\Phi \mathbin{\mathsf{U}} \Psi))$ and $Sat(\exists\Box\Phi)$.

In practice, one can either rewrite any formula in ENF (but with a potential blow-up), or design specific algorithms to deal with $\forall$ quantifiers (based on similar ideas).

Main algorithm

Key concept: bottom-up traversal of the parse tree of Φ.
For formulae in ENF,

- leaves can be true or $a \in AP$,
- inner nodes can be $\neg$, $\wedge$, $\exists\bigcirc$, $\exists\,\mathsf{U}$, or $\exists\Box$.

Each node represents a subformula Ψ of Φ and $Sat(\Psi)$ is the set of states where Ψ holds.

Intuition

When we compute $Sat(\Psi)$ in a node, it is as if we label all states of $Sat(\Psi)$ with a new proposition a_Ψ such that $a_\Psi \in L(s)$ iff $s \models \Psi$. This label can then be used to compute the parent formula.

E.g., computing $Sat(\exists\bigcirc\,\Psi)$ is now computing $Sat(\exists\bigcirc\, a_\Psi)$: there is no need to reconsider the child formula Ψ, just the corresponding labeling of states.

Characterization of Sat (1/2)

Let $\mathcal{T} = (S, Act, \longrightarrow, I, AP, L)$ be a TS without terminal state.
For all CTL formulae Φ, Ψ over AP, we have:

$$\begin{aligned}
Sat(\text{true}) &= S \\
Sat(a) &= \{s \in S \mid a \in L(s)\} \text{ for } a \in AP \\
Sat(\Phi \wedge \Psi) &= Sat(\Phi) \cap Sat(\Psi) \\
Sat(\neg\Phi) &= S \setminus Sat(\Phi) \\
Sat(\exists \bigcirc \Phi) &= \{s \in S \mid Post(s) \cap Sat(\Phi) \neq \emptyset\}
\end{aligned}$$

$\hookrightarrow$ All states that have a successor in $Sat(\Phi)$.

Characterization of Sat (2/2)

$Sat(\exists(\Phi \, \mathsf{U} \, \Psi))$ is the smallest subset T of S such that

1. $Sat(\Psi) \subseteq T$,
2. $s \in Sat(\Phi) \land Post(s) \cap T \neq \emptyset \implies s \in T$.

$\hookrightarrow$ (1) must hold because $\Phi \, \mathsf{U} \, \Psi$ is satisfied directly, and (2) says that if Φ holds now and there exists a successor where $\exists(\Phi \, \mathsf{U} \, \Psi)$ holds, then $\exists(\Phi \, \mathsf{U} \, \Psi)$ holds also now (cf. expansion law).

$Sat(\exists\Box\Phi)$ is the largest subset T of S such that

1. $T \subseteq Sat(\Phi)$,
2. $s \in T \implies Post(s) \cap T \neq \emptyset$.

$\hookrightarrow$ (1) must hold because states outside $Sat(\Phi)$ directly falsify $\exists\Box\Phi$, and (2) says that if $\exists\Box\Phi$ holds now, then there must exist a successor where $\exists\Box\Phi$ still holds (cf. expansion law).

Computation of Sat: algorithm (1/3)

Input: TS $\mathcal{T} = (S, Act, \longrightarrow, I, AP, L)$ and CTL formula Φ in ENF
Output: $Sat(\Phi) = \{s \in S \mid s \models \Phi\}$

if $\Phi = \text{true}$ **then**
  return S
else if $\Phi = a \in AP$ **then**
  return $\{s \in S \mid a \in L(s)\}$
else if $\Phi = \Psi_1 \wedge \Psi_2$ **then**
  return $Sat(\Psi_1) \cap Sat(\Psi_2)$
else if $\Phi = \neg\Psi$ **then**
  return $S \setminus Sat(\Psi)$
else if $\Phi = \exists\bigcirc\,\Psi$ **then**
  return $\{s \in S \mid Post(s) \cap Sat(\Psi) \neq \emptyset\}$
$\vdots$

Computation of Sat: algorithm (2/3)

$\vdots$
else if $\Phi = \exists(\Psi_1 \mathsf{U} \Psi_2)$ **then**
  $T := Sat(\Psi_2)$ // smallest fixed point computation
  while $A := \{s \in Sat(\Psi_1) \setminus T \mid Post(s) \cap T \neq \emptyset\} \neq \emptyset$ **do**
    $T := T \cup A$
  return T
$\vdots$

$\hookrightarrow$ We iteratively compute an **increasing** sequence of sets T_i s.t. $T_0 = Sat(\Psi_2)$ and $T_{i+1} = T_i \cup \{s \in Sat(\Psi_1) \mid Post(s) \cap T_i \neq \emptyset\}$, i.e., T_i represents all states that can reach $Sat(\Psi_2)$ in at most i steps via a path of states in $Sat(\Psi_1)$.

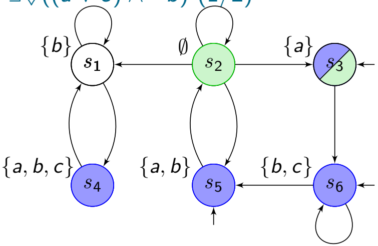
Computation of Sat: algorithm (3/3)

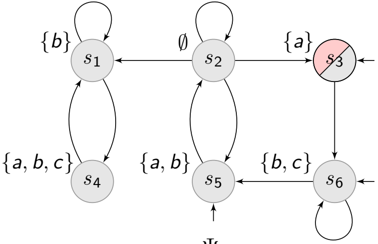
$\vdots$
else if $\Phi = \exists\Box\Psi$ **then**
$T := Sat(\Psi)$ // largest fixed point computation
while $A := \{s \in T \mid Post(s) \cap T = \emptyset\} \neq \emptyset$ **do**
$T := T \setminus A$
return T

$\hookrightarrow$ We iteratively compute a **decreasing** sequence of sets T_i s.t. $T_0 = Sat(\Psi)$ and $T_{i+1} = T_i \cap \{s \in Sat(\Psi) \mid Post(s) \cap T_i \neq \emptyset\}$, i.e., T_i represents all states from which there exists a path staying in $Sat(\Psi)$ for at least i steps.

Examples

$\Phi = \exists\Diamond((a \vee c) \wedge \neg b)$ (1/2)



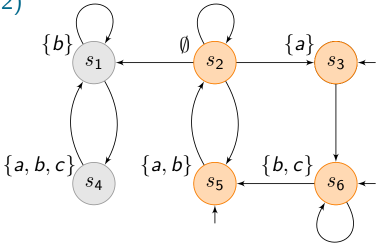


Formula $\Phi = \exists\Diamond((a \vee c) \wedge \neg b) \equiv \exists\Big(\underbrace{\text{true}}_{\Psi_4} \,\mathsf{U}\, \overbrace{\big(\underbrace{(a \vee c)}_{\Psi_1} \wedge \underbrace{\neg b}_{\Psi_2}\big)}^{\Psi_3}\Big)$

1. $Sat(\Psi_1) = Sat(a) \cup Sat(c) = \{s_3, s_4, s_5, s_6\}$
2. $Sat(\Psi_2) = S \setminus Sat(b) = \{s_2, s_3\}$
3. $Sat(\Psi_3) = Sat(\Psi_1) \cap Sat(\Psi_2)$, $Sat(\Psi_4) = S$
4. $Sat(\Phi) = \exists\Psi_4 \,\mathsf{U}\, \Psi_3 \implies$ **Algorithm in the next slide.**

Examples

$\Phi = \exists\Diamond((a \vee c) \wedge \neg b)$ (2/2)



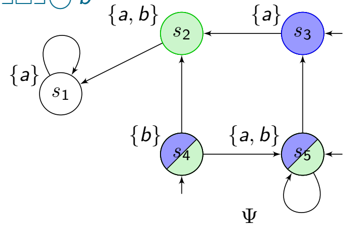
We obtain $Sat(\Phi) = \exists\Psi_4 \, \mathsf{U} \, \Psi_3$ via smallest fixed point computation:

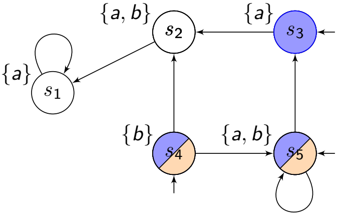
- $T_0 = Sat(\Psi_3) = \{s_3\}$
- $T_1 = T_0 \cup \{s \in Sat(\Psi_4) \mid Post(s) \cap T_0 \neq \emptyset\} = \{s_2, s_3\}$
- $T_2 = T_1 \cup \{s \in Sat(\Psi_4) \mid Post(s) \cap T_1 \neq \emptyset\} = \{s_2, s_3, s_5\}$
- $T_3 = T_2 \cup \{s \in Sat(\Psi_4) \mid Post(s) \cap T_2 \neq \emptyset\} = \{s_2, s_3, s_5, s_6\}$
- $T_4 = T_3 \cup \{s \in Sat(\Psi_4) \mid Post(s) \cap T_3 \neq \emptyset\} = T_3 = Sat(\Phi)$

$$I = \{s_3, s_5, s_6\} \subseteq Sat(\Phi) \implies \mathcal{T} \models \Phi = \exists\Diamond((a \vee c) \wedge \neg b)$$

Examples

$\Phi = \exists\Box\exists\bigcirc b$





Formula $\Phi = \exists\Box \overbrace{\exists\bigcirc b}^{\Psi}$

1. $Sat(b) = \{s_2, s_4, s_5\}$
2. $Sat(\Psi) = \{s \in S \mid Post(s) \cap Sat(b) \neq \emptyset\} = \{s_3, s_4, s_5\}$
3. We obtain $Sat(\Phi) = \exists\Box\Psi$ via largest fixed point computation:

- $T_0 = Sat(\Psi) = \{s_3, s_4, s_5\}$
- $T_1 = T_0 \cap \{s \in Sat(\Psi) \mid Post(s) \cap T_0 \neq \emptyset\} = \{s_4, s_5\}$
- $T_2 = T_1 \cap \{s \in Sat(\Psi) \mid Post(s) \cap T_1 \neq \emptyset\} = T_1 = Sat(\Phi)$

$$I = \{s_3, s_4, s_5\} \not\subseteq Sat(\Phi) \implies \mathcal{T} \not\models \Phi = \exists\Box\exists\bigcirc b$$

Complexity of CTL model checking

- Clever implementations of algorithms for $\exists(\Psi_1 \mathsf{U} \Psi_2)$ and $\exists\Box\Psi$ take time $\mathcal{O}(|S| + |\longrightarrow|)$.

  $\Longrightarrow$ **See the book for detailed algorithms.**

- Main algorithm to compute $Sat(\Phi)$ is a bottom-up traversal of the parse tree: $\mathcal{O}(|\Phi|)$.

Complexity of the algorithm

The time complexity is $\mathcal{O}(|\mathcal{T}| \cdot |\Phi|)$.

$\Longrightarrow$ **CTL model checking is in polynomial time!**

$\Longrightarrow$ **So... much more efficient than LTL which is** PSPACE**-complete?**

$\Longrightarrow$ **Not really... need to consider the whole picture, including succinctness!**

1 CTL: a specification language for BT properties

2 CTL model checking

3 CTL vs. LTL

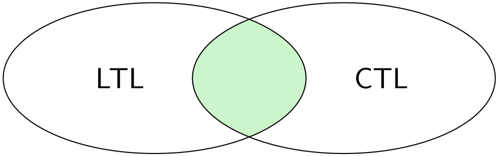
4 CTL*

Expressiveness

Incomparable logics

We have seen that:

- some properties are expressible in LTL but not in CTL (e.g., $\phi = \Diamond\Box a$),
- some properties are expressible in CTL but not in LTL (e.g., $\Phi = \forall\Box\exists\Diamond a$),
- some properties can be expressed in both logics (e.g., $\phi = \Box\Diamond a$ is equivalent to $\Phi = \forall\Box\forall\Diamond a$).



Can we characterize the intersection?

Expressiveness

Equivalent formulae

Recall the notion of equivalent formulae.

Definition: equivalent formulae

CTL formula Φ and LTL formula ϕ over AP are *equivalent*, denoted $\Phi \equiv \phi$ if for all TS $\mathcal{T}$, $\mathcal{T} \models \Phi \iff \mathcal{T} \models \phi$.

Here is a way to know if a CTL formula admits an equivalent one in LTL.

Criterion for transformation from CTL to LTL

Let Φ be a CTL formula, and ϕ be the LTL formula obtained by eliminating all path quantifiers from Φ. Then, either $\Phi \equiv \phi$ or there exists no LTL formula equivalent to Φ.

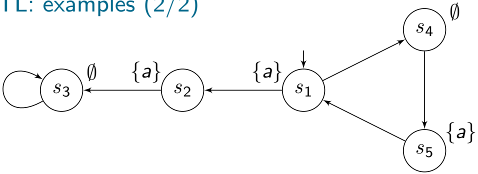
Expressiveness

Comparing LTL and CTL: examples (1/2)

- We proved that $\phi = \Box\Diamond a \equiv \Phi = \forall\Box\forall\Diamond a$, and indeed, ϕ is obtained from Φ by removing all quantifiers.
- We argued that $\Phi = \forall\Diamond\forall\Box a \not\equiv \phi = \Diamond\Box a$. Hence, **there is no equivalent to Φ in LTL**.

Expressiveness

Comparing LTL and CTL: examples (2/2)



Consider formula $\Phi = \forall\Diamond(a \wedge \forall\bigcirc a)$ and its potential LTL equivalent, $\phi = \Diamond(a \wedge \bigcirc a)$.

- $\mathcal{T} \models \phi$ because $s_1 \models \phi$:
  - ▷ All paths in $Paths(s_1)$ contain $s_1 \rightarrow s_2$, or $s_5 \rightarrow s_1$, or both.
  - ▷ Any suffix $s_1 s_2 \ldots$ satisfies $(a \wedge \bigcirc a)$, and so does any suffix $s_5 s_1 \ldots$
  - ▷ Hence all paths satisfy ϕ.
- $\mathcal{T} \not\models \Phi$ because of path $s_1 s_2 s_3^\omega$.
  - ▷ None of s_1, s_2 and s_3 satisfies $(a \wedge \forall\bigcirc a)$ (look at s_4 for s_1).

$\Longrightarrow$ **CTL formula $\Phi = \forall\Diamond(a \wedge \forall\bigcirc a)$ has no LTL equivalent.**

Model checking efficiency

Let $\mathcal{T} = (S, Act, \longrightarrow, I, AP, L)$ be a TS, and Φ (resp. ϕ) a CTL (resp. LTL) formula over AP.

- Model checking Φ requires linear time in both the model and the formula: $\mathcal{O}(|\mathcal{T}| \cdot |\Phi|)$.
- Model checking ϕ requires linear time in the model but exponential time in the formula: $\mathcal{O}(|\mathcal{T}|) \cdot 2^{\mathcal{O}(|\Phi|)}$.

Hence, CTL model checking is more efficient, right?

No!

Because LTL can be exponentially more succinct!

$\hookrightarrow$ That is, given a CTL formula, the LTL equivalent can be exponentially shorter.

LTL can be exponentially more succinct than CTL

Proof sketch (1/3)

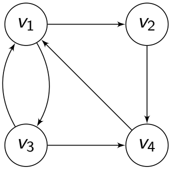
1. Take an NP-complete problem and show that it can be solved by model checking a polynomial-size LTL formula on a polynomial-size model.
2. Show that the LTL formula has an equivalent in CTL (of exponential size).
3. If an equivalent CTL formula of *polynomial size* existed, we would be able to solve the NP-complete problem in polynomial time, hence to prove that $\mathrm{P} = \mathrm{NP}$.

Hence, unless $\mathrm{P} = \mathrm{NP}$, some properties can be expressed in LTL through exponentially shorter formulae than in CTL.

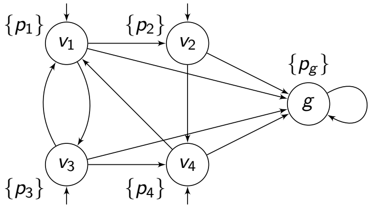
Chosen problem: deciding the existence of a Hamiltonian path (i.e., visiting each vertex exactly once) in a directed graph.

LTL can be exponentially more succinct than CTL

Proof sketch (2/3)



Directed graph.



Transition system.

Encoding of the problem:

- ▷ Make all vertices initial states and add an additional state g reachable from all other states.
- ▷ Label of vertex $v_i = p_i$, label of $g = p_g$.
- ▷ Let n be the number of vertices of the graph. Consider LTL formula $\phi = (\Diamond p_1 \land \ldots \land \Diamond p_n) \land \bigcirc^n p_g$.
- ▷ *Paths satisfying ϕ in the TS correspond to Hamiltonian paths in the graph.*

LTL can be exponentially more succinct than CTL

Proof sketch (3/3)

Reduction:

- The graph contains a Hamiltonian path iff $\mathcal{T} \not\models \neg\phi$ with $\phi = (\Diamond p_1 \wedge \ldots \wedge \Diamond p_n) \wedge \bigcirc^n p_g$.
- Observe that TS $\mathcal{T}$ and formula ϕ are both of polynomial size.
- No contradiction with NP-completeness since LTL model checking is PSPACE-complete.

Encoding in CTL?

- Yes but enumerates all possible Hamiltonian paths! E.g.,

$$\begin{aligned}\Phi = \; & (p_1 \wedge \exists \bigcirc (p_2 \wedge \exists \bigcirc (p_3 \wedge \exists \bigcirc p_4))) \\ \vee & (p_1 \wedge \exists \bigcirc (p_2 \wedge \exists \bigcirc (p_4 \wedge \exists \bigcirc p_3))) \\ \vee & (p_1 \wedge \exists \bigcirc (p_3 \wedge \exists \bigcirc (p_2 \wedge \exists \bigcirc p_4))) \vee \ldots\end{aligned}$$

$\Longrightarrow$ **Exponential formula:** $|\Phi| = \mathcal{O}(n \cdot n!)$

$\Longrightarrow$ **No polynomial encoding can exist unless P = NP because CTL model checking is in P.**

Other differences between LTL and CTL

Fairness

LTL

- Unconditional, strong and weak fairness can be formalized in LTL.
- Fairness can be incorporated into classical LTL model checking: $\mathcal{T} \models_{fair} \phi$ iff $\mathcal{T} \models (fair \rightarrow \phi)$.

CTL

- Most fairness constraints cannot be encoded in CTL. E.g., strong fairness $\Box\Diamond a \rightarrow \Box\Diamond b$ is equivalent to $\Diamond\Box\neg a \vee \Box\Diamond b$ and persistence ($\Diamond\Box\neg a$) is not expressible in CTL.
- Need for $\forall(fair \rightarrow \phi)$ and $\exists(fair \wedge \phi)$ but not possible in CTL (no connectives on path formulae).

$\Longrightarrow$ **In CTL, fairness requires specific techniques.**

$\Longrightarrow$ Adapt the semantics of $\exists\phi$ and $\forall\phi$ to interpret them on **fair** paths, with fairness constraint seen as an LTL formula over CTL state formulae.

$\Longrightarrow$ **Not discussed here. See the book for more.**

Other differences between LTL and CTL

Implementation relation

LTL

- LTL is preserved by *trace inclusion* (PSPACE-c.).
- (Bi)simulation is a sound but incomplete alternative, computable in polynomial time.

(bi)simulation
$\Downarrow \not\Uparrow$
trace inclusion

CTL

- Bisimulation preserves **full** CTL.
- Simulation preserves **the universal fragment** of CTL.

$\hookrightarrow$ Allows only quantifier $\forall$.

- Equivalently, simulation preserves **the existential fragment** of CTL.

$\hookrightarrow$ Allows only quantifier $\exists$ (recall $\forall\phi \equiv \neg\exists\neg\phi$).

$\Longrightarrow$ **Different logics, different implementation relations.**

LTL vs. CTL

Wrap-up

Notion of time	*Linear*	*Branching*
Behavior in state s	path-based: $Traces(s)$	state-based: computation tree of s
Temporal logic	LTL: path formulae ϕ $s \models \phi$ iff $\forall \pi \in Paths(s),\ \pi \models \phi$	CTL: state formulae Φ path quantifiers $\exists\phi$, $\forall\phi$
Model checking complexity	PSPACE-complete	P
Implementation relation	trace inclusion and equivalence (PSPACE-complete)	(bi)simulation (polynomial time)

1 CTL: a specification language for BT properties

2 CTL model checking

3 CTL vs. LTL

4 CTL*

Why?

Because LTL and CTL are incomparable.

- CTL* extends CTL by allowing **arbitrary nesting of path quantifiers** with temporal operators $\bigcirc$ and U.
- **CTL* subsumes both CTL and LTL.**

$\Longrightarrow$ **Here, we only take a quick glance at CTL*. For full discussion, including model checking algorithms, see the book.**

CTL* syntax

Core syntax

CTL* syntax

Given the set of atomic propositions AP, CTL* *state formulae* are formed according to the following grammar:

$$\Phi ::= \text{true} \mid a \mid \Phi \wedge \Psi \mid \neg\Phi \mid \exists\phi$$

where $a \in AP$ and ϕ is a path formula. CTL* *path formulae* are formed according to the following grammar:

$$\phi ::= \Phi \mid \phi \wedge \psi \mid \neg\phi \mid \bigcirc\,\phi \mid \phi \,\mathsf{U}\, \psi$$

where Φ is a state formula and ϕ, ψ are path formulae.

As for LTL and CTL, we obtain derived propositional logics operators $\vee$, $\rightarrow$, ... Moreover,

$$\Diamond\phi = \text{true}\,\mathsf{U}\,\phi \quad \text{and} \quad \Box\phi = \neg\Diamond\neg\phi \quad \text{and} \quad \forall\phi = \neg\exists\neg\phi$$

CTL* syntax

Examples (1/2)

CTL* syntax reminder

$\Phi ::= \text{true} \mid a \mid \Phi \land \Psi \mid \neg\Phi \mid \exists\phi \quad \phi ::= \Phi \mid \phi \land \psi \mid \neg\phi \mid \bigcirc \phi \mid \phi \,\mathsf{U}\, \psi$

- Is $\Phi = \exists\bigcirc a$ a valid CTL* formula? (yes for CTL)
  - ▷ **Yes**, because $\phi = \bigcirc a$ is a valid path formula, hence $\Phi = \exists\phi$ is a valid state formula.
- Is $\Phi = a \land b$ a valid CTL* formula? (yes for CTL)
  - ▷ **Yes**, because $\Psi_1 = a$ and $\Psi_2 = b$ are valid state formulae, hence $\Phi = \Psi_1 \land \Psi_2$ is a valid state formula.
- Is $\Phi = \forall(a \land \exists\bigcirc b)$ a valid CTL* formula? (no for CTL)
  - ▷ **Yes**, because $\Psi = a \land \exists\bigcirc b$ is a valid state formula and any state formula Ψ can be taken as a path formula $\phi = \Psi$.

CTL* syntax

Examples (2/2)

CTL* syntax reminder

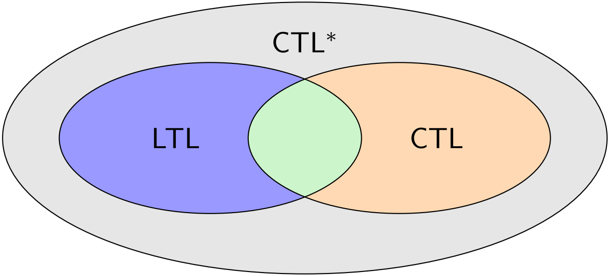
$\Phi ::= \mathrm{true} \mid a \mid \Phi \wedge \Psi \mid \neg\Phi \mid \exists\phi \quad \phi ::= \Phi \mid \phi \wedge \psi \mid \neg\phi \mid \bigcirc\phi \mid \phi \,\mathsf{U}\, \psi$

- Is $\Phi = \exists((\forall\bigcirc a) \,\mathsf{U}\, (a \wedge b))$ a valid CTL* formula? (yes for CTL)
  - ▷ **Yes**, because $\Psi_1 = \forall\bigcirc a$ and $\Psi_2 = a \wedge b$ are valid state formulae, hence $\phi = \Psi_1 \,\mathsf{U}\, \Psi_2$ is a valid path formula, hence $\Phi = \exists\phi$ is a valid state formula.
- Is $\Phi = \exists\bigcirc(a \,\mathsf{U}\, b)$ a valid CTL* formula? (no for CTL)
  - ▷ **Yes**, because $\phi = a \,\mathsf{U}\, b$ is a valid *path* formula and we can use it directly after $\bigcirc$ without an additional quantifier in CTL*.

Semantics

The semantics of CTL* follows naturally from the one of CTL.

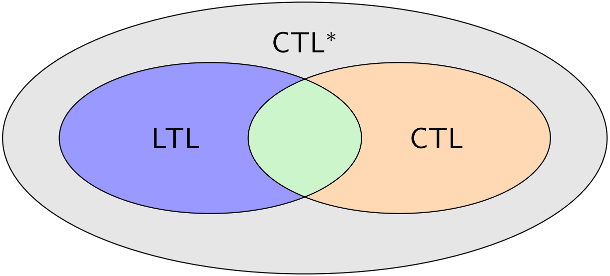
Expressiveness



- **Any CTL formula is also a CTL* formula.**
  - ▷ Indeed, the syntax of CTL is a subset of the one of CTL*.
- **Any LTL formula ϕ has an equivalent CTL* formula.**
  - ▷ We have $\mathcal{T} \models \phi \iff \mathcal{T} \models \Phi = \forall\phi$.

$\Longrightarrow$ CTL* is strictly more expressive than LTL and CTL, i.e., there exist CTL* formulae that cannot be expressed neither in LTL nor in CTL.

Expressiveness



Examples of formulae belonging to the different sets

- LTL formula $\phi = \Diamond\Box a$ cannot be expressed in CTL.
- CTL formula $\Phi = \forall\Box\exists\Diamond a$ cannot be expressed in LTL.
- LTL formula $\phi = \Box\Diamond a$ is equivalent to CTL $\Phi = \forall\Box\forall\Diamond a$.
- CTL* formula $\Phi = \forall\Diamond\Box a \wedge \forall\Box\exists\Diamond b$ is not expressible in LTL nor in CTL.

CTL* model checking

- The algorithm for CTL* combines the respective algorithms for LTL and CTL.
- Its complexity is dominated by the complexity of LTL model checking.

Complexity of the algorithm

The time complexity is $\mathcal{O}(|\mathcal{T}|) \cdot 2^{\mathcal{O}(|\Phi|)}$.

Complexity of the model checking problem for CTL*

The CTL* model checking problem is PSPACE-complete.

$\Longrightarrow$ **Since LTL model checking is reducible to CTL* model checking.**

Implementation relations

Similarly to CTL,

- bisimulation preserves **full** CTL*;
- simulation preserves **the existential and universal fragments** of CTL*.

References I

C. Baier and J.-P. Katoen.
Principles of model checking.
MIT Press, 2008.