Chapter 6: Computation Tree Logic

Prof. Ali Movaghar
Verification of Reactive Systems
Spring 2012
Outline

- We introduce **Computation Tree Logic (CTL)**, a branching temporal logic for specifying system properties.
- A comparison to LTL based on expressiveness is provided.
- The chapter concludes with an extension called CTL* which subsumes both CTL and LTL.
Branching Time vs. Linear Time

- In **Linear time** at each moment of time there is only one possible successor state and thus each time moment has a unique possible future.

- We can state properties over all possible computations that start in a state,

- but not easily about *some of such computations*
Branching Time vs. Linear Time (cont.)

- Sometimes we can overcome these problems:
- To check whether there exists some computation starting in $s$ that satisfies $\phi$ we may check whether $s \models \forall \neg \phi$
But sometimes we cannot:

The property “for every computation it is always possible to return to the initial state” cannot be stated or checked by LTL.

- Although it can be checked by a stronger property
- We may require a computation to *always* return to the initial state: \( s \models \forall \Box \Diamond \text{start} \)
Branching Time vs. Linear Time (cont.)

- We could solve this Problem if we could write the property:
  - $\forall \Box \exists \Diamond \text{start}$: in any state ($\Box$) of any possible computation ($\forall$), there is a possibility ($\exists$) to eventually return to the start state ($\Diamond$ start)
    
    (This is actually a CTL property)

- But this not a valid nesting of an LTL property
Branching Time vs. Linear Time (cont.)

- **Branching time** refers to the fact that at each moment there may be several different possible futures.
- Instead of a sequence of states (a path) what we have here is a Tree of states.
- The tree rooted at state $s$ represents all possible infinite computations in the transition system that start in $s$. 
Computation Tree Logic: Syntax (State formulae)

- CTL has a **two-stage syntax**: 
- CTL state formulae
  - are formed over the set $AP$ of atomic proposition according to the following grammar:
    \[ \Phi ::= \text{true} \mid a \mid \Phi_1 \land \Phi_2 \mid \neg \Phi \mid \exists \phi \mid \forall \phi \]
    where $a \in AP$ and $\phi$ is a path formula
- Note how every path formula is preceded by a quantifier
Computation Tree Logic: Syntax (Path formulae)

- CTL path formulae
  - are formed according to the following grammar:
    \[ \phi ::= \circ \Phi \mid \Phi_1 \mathbin{U} \Phi_2 \]
    where \( \Phi, \Phi_1 \) and \( \Phi_2 \) are state formulae

(From now on, capital greek letters denote state formulae, and the small ones denote path formulae)

- Path formulae are as in LTL built by the next-step and until operators, but they must not be combined with Boolean connectives
Computation Tree Logic: Deriving the Temporal Modalities

- **eventually:**
  - $\exists \diamond \Phi = \exists (\text{true U } \Phi)$  
    “$\Phi$ holds potentially”
  - $\forall \diamond \Phi = \forall (\text{true U } \Phi)$  
    “$\Phi$ is inevitable”

- **always:**
  - $\exists \Box \Phi = \neg \forall \diamond \neg \Phi$  
    “potentially always $\Phi$”
  - $\forall \Box \Phi = \neg \exists \diamond \neg \Phi$  
    “invariantly $\Phi$”
“always” \( \Phi \) cannot be obtained from the equation \( \Box \Phi = \neg \Diamond \neg \Phi \) as in LTL, since propositional logic operators cannot be applied to path formulae.

We should exploit the duality of existential and universal quantification:

- \( \exists \Box \Phi \) is not defined as \( \exists \neg \Diamond \neg \Phi \), but rather as \( \neg \forall \Diamond \neg \Phi \)
Example 6.3.: The mutual exclusion property is described in CTL by the formula:
\[ \forall \Box (\neg \text{crit1} \lor \neg \text{crit2}) \]

CTL formulae of the form \[ \forall \Box \forall \Diamond \Phi \] expresses that \( \Phi \) is infinitely often true on all paths thus the CTL formula \( (\forall \Box \forall \Diamond \text{crit1}) \land (\forall \Box \forall \Diamond \text{crit2}) \) specifies the Liveness property.
Example 6.3.

“∀ □ ∃ ◊ start” expresses that in every reachable system state it is possible to return to the starting state(s).
Given a transition system $\mathcal{TS} = (S, \text{Act}, \rightarrow, I, \text{AP}, L)$, the semantics of CTL formulae is defined by two satisfaction relations (both denoted by $|=_{\mathcal{TS}}$, or briefly $|=_{\mathcal{TS}}$): one for the state formulae and one for the path formulae.
The satisfaction relation $|=\,$ is defined for state formulae by:

- $s |= a \iff a \in L(s)$
- $s |= \neg \Phi \iff \text{not } s |= \Phi$
- $s |= \Phi \land \Psi \iff (s |= \Phi) \text{ and } (s |= \Psi)$
- $s |= \exists \varphi \iff \pi |= \varphi \text{ for some } \pi \in \text{Paths}(s)$
- $s |= \forall \varphi \iff \pi |= \varphi \text{ for all } \pi \in \text{Paths}(s)$
CTL Semantics: Satisfaction Relation

For path $\pi$, the satisfaction relation $\models$ for path formulae is defined by:

\[
\pi \models \Diamond \Phi \quad \text{iff} \quad \pi[1] \models \Phi \\
\pi \models \Phi \lor \Psi \quad \text{iff} \quad \exists j \geq 0. \ (\pi[j] \models \Psi \land (\forall 0 \leq k < j. \pi[k] \models \Phi))
\]
CTL Semantics: Semantics for Transition Systems

- Given a transition system $TS$, the satisfaction set $\text{Sat}_{TS}(\Phi)$, or briefly $\text{Sat}(\Phi)$, for CTL-state formula $\Phi$ is defined by:
  - $\text{Sat}(\Phi) = \{s \in S| s \models \Phi\}$

- The transition system $TS$ satisfies CTL formula $\Phi$ if and only if $\Phi$ holds in all initial states of $TS$:
  - $TS \models \Phi$ if and only if $\forall s_0 \in I. s_0 \models \Phi$.

- This is equivalent to $I \subseteq \text{Sat}(\Phi)$.
CTL Semantics: Semantics derivation

For path fragment $\pi = s_0 \ s_1 \ s_2 \ \ldots$, The semantics of the derived path operator “eventually” is:

- $\pi \models \Diamond \Phi$ iff $s_j \models \Phi$ for some $j \geq 0$

As we know, $\exists \Box \Phi = \neg \forall \Diamond (\neg \Phi)$, from which we can derive:

- $s \models \neg \forall \Diamond (\neg \Phi)$ iff $\neg (\forall \pi \in \text{Paths}(s). \pi \models \Diamond (\neg \Phi))$, so:
- $s \models \exists \Box \Phi$ iff $\exists \pi \in \text{Paths}(s). \pi[j] \models \Phi$ for all $j \geq 0$
Therefore, \( \Phi \) can be understood as a CTL path formula with the semantics:

\[
\pi = s_0 \ s_1 \ s_2 \ldots \models \Box \Phi \text{ if and only if } s_j \models \Phi \text{ for all } j \geq 0
\]

- Read Example 6.6
Computation Tree Logic: Semantics

**Example 6.7.** Consider the LTS below with the set of propositions \( AP=\{a, b\} \)

\[ TS \models \exists \diamond (\exists \square a) \]

\[ TS \models \exists (aU(\neg a \land \forall (\neg aUb))) \]

(The little graphs show states where the leading formulae hold)
Computation Tree Logic: Semantics

- **Infinitely Often**: We are going to prove the following:
  - if $s \models \forall \Box \forall \lozenge a$ then $\forall \pi \in \text{Paths}(s). \pi[i] \models a$ for infinitely many $i$

  - The reverse of this Property is also true, as proved in Remark 6.8.

- Let $s$ be a state, such that $s \models \forall \Box \forall \lozenge a$

- We show that every infinite path fragment $\pi$ starting in $s$ passes through an a-state infinitely often
Computation Tree Logic: Semantics (Cont.)

- From \( s \models \forall \Box \forall \Diamond a \) we have \( \pi \models \Box \forall \Diamond a \)
- Let \( \pi = s_0 s_1 s_2 \ldots \in \text{Paths}(s) \) and \( j \geq 0 \). We demonstrate that there exists an index \( i \geq j \) with \( s_i \models a \)
- So, for \( s_j \) we have \( s_j \models \forall \Diamond a \)
- \( \pi[j \ldots] = s_j s_{j+1} \ldots \in \text{Paths}(s_j) \)
  Hence for all \( j \) we have:
  \[
  s_j s_{j+1} s_{j+2} \ldots \models \Diamond a
  \]
  Which means there exists an \( i \) for which \( s_i \models a \) holds
- We can conclude that path \( \pi \) visits an \( a \)-state infinitely often.
The weak-until operator in CTL cannot be defined directly starting from the LTL definition $\phi W \psi = (\phi U \psi) \lor (\Box \phi)$ since it does not yield a syntactically correct CTL formula.

It should be defined using the LTL equivalence law

$\phi W \psi = \neg ((\phi \land \neg \psi) U (\neg \phi \land \neg \psi))$
The weak-until operator can be defined in CTL by:

- $\exists (\Phi W \Psi) = \neg \forall ((\Phi \land \neg \Psi) U (\neg \Phi \land \neg \Psi))$
- $\forall (\Phi W \Psi) = \neg \exists ((\Phi \land \neg \Psi) U (\neg \Phi \land \neg \Psi))$

So, for $\exists (\Phi W \Psi)$ we can say that:

- $\pi \models \Phi W \Psi$ iff $\pi \models \Phi U \Psi$ or $\pi \models \Box (\Phi \land \neg \Psi)$,
- Or $\pi \models \Phi W \Psi$ iff $\pi \models \Phi U \Psi$ or $\pi \models \Box \Phi$.

Which yields: $\exists (\Phi W \Psi) = \exists (\Phi U \Psi) \lor \exists \Box \Phi$
Computation Tree Logic: Semantics (Negation)

- It is possible that the statements $\text{TS} \models \Phi$ and $\text{TS} \models \neg \Phi$ both hold
  - Because of the fact that initial states are not necessarily unique

- $\text{TS} \not\models \neg \exists \phi$ iff $\exists \pi \in \text{Paths(TS)}. (\pi \models \phi)$

- Thus the statement $\text{TS} \not\models \neg \exists \phi$ holds if and only if there exists an initial state $s_0 \in I$ with $s_0 \not\models \neg \exists \phi$, (i.e. $s_0 \models \exists \phi$)
On the other hand, $TS \models \exists \phi$ requires that $s_0 \models \exists \phi$ for all $s_0 \in I$.

In the above figure we have that both $TS \not\models \neg \exists \Box a$ and $TS \not\models \exists \Box a$ hold.
CTL Semantics: Transition Systems with Terminal States

For a finite maximal path fragment $\pi = s_0 s_1 s_2 \ldots s_n$ of length $n$, $s_n$ is a terminal state:

- $\pi \models \Diamond \Phi$ iff $n > 0$ and $s_1 \models \Phi$,
- $\pi \models \Phi \cup \Psi$ iff there exists an index $j \in \mathbb{N}$ with $j \leq n$, and $s_i \models \Phi$, for $i = 0, 1, \ldots, j-1$, and $s_j \models \Psi$.

- $s \models \forall \Diamond \text{false}$ iff $s$ is a terminal state. Furthermore:
  - $\pi \models \Box \Phi$ iff there exists an index $j \leq n$ with $s_j \models \Phi$,
  - $\pi \models \Diamond \Phi$ iff for all $j \in \mathbb{N}$ with $j \leq n$ we have $s_j \models \Phi$. 

27
Computation Tree Logic: Equivalence of CTL Formulae

- CTL formulae $\Phi$ and $\Psi$ are called equivalent whenever for any state $s$ it holds that $s \models \Phi$ if and only if $s \models \Psi$.

**Definition:** CTL formulae $\Phi$ and $\Psi$ (over $\text{AP}$) are called equivalent, denoted $\Phi \equiv \Psi$, if $\text{Sat}(\Phi) = \text{Sat}(\Psi)$ for all transition systems $\text{TS}$ over $\text{AP}$.

Recall that the notion of CTL Formula is used for a CTL state formula.
Computation Tree Logic: Equivalence of CTL Formulae

Useful equivalency Rules:

- Duality laws:

\[
\begin{align*}
\forall \Diamond \Phi & \equiv \neg \exists \Diamond \neg \Phi \\
\exists \Diamond \Phi & \equiv \neg \forall \Diamond \neg \Phi \\
\forall \Box \Phi & \equiv \neg \exists \Box \neg \Phi \\
\exists \Box \Phi & \equiv \neg \forall \Box \neg \Phi \\
\forall (\Phi \cup \Psi) & \equiv \neg \exists (\neg \Psi \cup (\neg \Phi \land \neg \Psi)) \land \neg \exists \Box \neg \Psi \\
& \equiv \neg \exists ((\Phi \land \neg \Psi) \cup (\neg \Phi \land \neg \Psi)) \land \neg \exists \Box (\Phi \land \neg \Psi) \\
& \equiv \neg \exists ((\Phi \land \neg \Psi) \mathsf{W} (\neg \Phi \land \neg \Psi))
\end{align*}
\]
Equivalence of CTL Formulae (Cont.)

- Expansion Laws:

\[
\begin{align*}
\forall (\Phi U \Psi) & \equiv \Psi \lor (\Phi \land \forall O \forall (\Phi U \Psi)) \\
\forall \diamond \Phi & \equiv \Phi \land \forall O \forall \diamond \Phi \\
\forall \square \Phi & \equiv \Phi \land \forall O \forall \square \Phi \\
\exists (\Phi U \Psi) & \equiv \Psi \lor (\Phi \land \exists O \exists (\Phi U \Psi)) \\
\exists \diamond \Phi & \equiv \Phi \land \exists O \exists \diamond \Phi \\
\exists \square \Phi & \equiv \Phi \land \exists O \exists \square \Phi
\end{align*}
\]

- $\exists(\varphi U \psi)$ is equivalent to the fact that the current state either satisfies $\psi$, or it satisfies $\varphi$, and for some direct successor state, $\exists(\varphi U \psi)$ holds.
Equivalence of CTL Formulae (Cont.)

Distributive Laws:

\[
\text{distributive laws}
\]

\[
\forall \Box (\Phi \land \Psi) \equiv \forall \Box \Phi \land \forall \Box \Psi
\]

\[
\exists \Diamond (\Phi \lor \Psi) \equiv \exists \Diamond \Phi \lor \exists \Diamond \Psi
\]
Computation Tree Logic: Normal Forms

- The basic operators $\exists \circ$, $\exists U$, and $\forall U$ would have been sufficient to define the syntax of CTL. Although by the following definition we could omit all the universal quantifiers as well.
**Definition:** For $a \in AP$, the set of CTL state formulae in existential normal form (ENF, for short) is given by:

- $\Phi ::= true \mid \alpha \mid \Phi_1 \land \Phi_2 \mid \neg \Phi \mid \exists \diamond \Phi \mid \exists(\Phi_1 \cup \Phi_2) \mid \exists \square \Phi$.

**Theorem 6.14:** For each CTL formula there exists an equivalent CTL formula in ENF.
**CTL Normal Forms: Positive Normal Form**

- **Definition**: The set of CTL state formulae in positive normal form (PNF, for short) is given by:
  
  - \( \Phi ::= \text{true} \mid \text{false} \mid a \mid \neg a \mid \Phi_1 \land \Phi_2 \mid \Phi_1 \lor \Phi_2 \mid \exists \phi \mid \forall \phi \)

  where \( a \in \text{AP} \) and the path formulae are given by

  - \( \phi ::= \Box \Phi \mid \Phi_1 \lor \Phi_2 \mid \Phi_1 \lor \Phi_2 \).

- **Theorem 6.16**: For each CTL formula there exists an equivalent CTL formula in PNF.
CTL Normal Forms: Release Operator

- In the rules for $\forall U$ and $\exists U$ the number of occurrences of $\Psi$ (and $\Phi$) is doubled, the length of an equivalent CTL formula may be exponentially longer than the original CTL formula.

- This exponential blowup can be avoided by using the release operator, which in CTL can be defined by:
  - $\exists (\Phi \mathcal{R} \Psi) = \neg \forall ((\neg \Phi) U (\neg \Psi))$
  - $\forall (\Phi \mathcal{R} \Psi) = \neg \exists ((\neg \Phi) U (\neg \Psi))$
Expressiveness of CTL vs. LTL

There are properties that one can express in CTL, but that cannot be expressed in LTL, and vice versa.

- Hence, the logics CTL and LTL are incomparable according to their expressiveness.

**Definition:** CTL formula $\Phi$ and LTL formula $\phi$ (both over AP) are equivalent, denoted $\Phi \equiv \phi$, if for any transition system $TS$ over AP:

- $TS \models \Phi$ if and only if $TS \models \phi$
Expressiveness of CTL vs. LTL (Cont.)

- Dropping all (universal and existential) quantifiers is a safe way to generate an equivalent LTL formula, provided there are equivalent LTL formulae:
  - As an example, the LTL formula $\Diamond a$ is equivalent to the CTL formula $\forall \Diamond a$
  - However, $\forall \Diamond \forall \Box a$ and $\Diamond \Box a$ are not equivalent.
Theorem 6.18: Let $\Phi$ be a CTL formula, and $\phi$ the LTL formula that is obtained by eliminating all path quantifiers in $\Phi$. Then:

- $\Phi \equiv \phi$

or

- There does not exist any LTL formula that is equivalent to $\Phi$. 
Lemma 6.19. (Persistence): The CTL formula $\forall \Diamond \forall \Box a$ and the LTL formula $\Diamond \Box a$ are not equivalent.

By theorem 6.18 it can be asserted that there is no LTL expression whatsoever which can describe the property $\forall \Diamond \forall \Box a$. 
Expressiveness of CTL vs. LTL (Cont.)

- **Lemma 6.20.** (Eventually an $a$-State with only direct $a$-Successors):
  - The CTL formula $\forall \Diamond (a \land \forall \bigcirc a)$ and the LTL formula $\Diamond (a \land \bigcirc a)$ are not equivalent.

- In addition to CTL formulas inexpressible in LTL, there are also LTL formulas which cannot be expressed in CTL.
Theorem 6.21. (Incomparable Expressiveness of CTL and LTL):

a) There exist LTL formulae for which no equivalent CTL formula exists. This holds for, for instance:
   - $\diamond \square a$ and $\diamond (a \land \Diamond a)$.

b) There exist CTL formulae for which no equivalent LTL formula exists. This holds for, for instance:
   - $\forall \forall \diamond a$ and $\forall \diamond (a \land \forall \Diamond a)$ and $\forall \square \exists \diamond a$. 
CTL Model Checking

- **CTL model checker:**
  - A decision algorithm that checks whether \( TS \models \Phi \) for a given transition system \( TS \) and a CTL formula \( \Phi \)

- Throughout this section, it is assumed that \( TS \) is finite, and has no terminal states

- CTL model checking can be performed by a recursive procedure that calculates the satisfaction set for all subformulæ of \( \Phi \)
CTL Model Checking (Cont.)

- We consider CTL formulae in ENF (CTL formulae built by the basic modalities $\exists \Box$, $\exists U$, and $\exists \square$.)
  - This requires algorithms that generate $\text{Sat}(\exists \Box \Phi)$, $\text{Sat}(\exists (\Phi U \Psi))$, and $\text{Sat}(\exists \square \Phi)$ when $\text{Sat}(\Phi)$ and $\text{Sat}(\Psi)$ are given.
CTL Model Checking: Global Model-checking Procedure

- The basic procedure for CTL model checking is:
  - The set $\text{Sat}(\Phi)$ of all states satisfying $\Phi$ is computed recursively.
  - It follows that $\text{TS} \models \Phi$ if and only if $I \subseteq \text{Sat}(\Phi)$, where $I$ is the set of initial states of $\text{TS}$.
- It is called global because satisfaction of all other states is verified besides the initial states.
Sat(Φ) is computed in a bottom-up traversal of the parse tree of the CTL state formula Φ:

- The nodes of the parse tree represent the subformulae of Φ.
- The leaves stand for the constant true or an atomic proposition a ∈ AP.
- All inner nodes are labeled with an operator. For ENF formulae the labels of the inner nodes are ¬, ∧, ∃□, ∃U, or ∃□.
CTL Model Checking: Global Model-checking Procedure (Cont.)

- The children of a node $v$ stand for the maximal genuine subformulae of the formula $\Psi_v$ that is represented by $v$.
- The Operator labeling a node says how to combine its maximal genuine subformulae.
As soon as a certain subformula $\Psi$ is computed, $a_\Psi$ is added to $L(s)$ for every $s \in \text{Sat}(\Psi)$ rather as an atomic proposition satisfied by $s$. 
Example 6.22:

\[ \Phi = \exists \bigcirc a \land \exists (b \lor \exists \square \neg c) \]
Once the subformulae $\Psi$ and $\Psi'$ are treated, they can be replaced by the new atomic propositions $a_1, a_2$.

The formula that is to be treated for the root node simply thus is: $\Phi = a_1 \land a_2$.

Sat($\Phi$) results from intersecting $\text{Sat}(a_1) = \text{Sat}(\Psi)$ and $\text{Sat}(a_2) = \text{Sat}(\Psi')$. 
Theorem 6.23: For all CTL formulae $\Phi, \Psi$ over AP it holds that:

(a) $Sat(true) = S$,

(b) $Sat(a) = \{ s \in S \mid a \in L(s) \}$, for any $a \in AP$,

(c) $Sat(\Phi \land \Psi) = Sat(\Phi) \cap Sat(\Psi)$,

(d) $Sat(\neg \Phi) = S \setminus Sat(\Phi)$,

(e) $Sat(\exists \bigcirc \Phi) = \{ s \in S \mid Post(s) \cap Sat(\Phi) \neq \emptyset \}$,
(f) \( \text{Sat}(\exists (\Phi \cup \Psi)) \) is the smallest subset \( T \) of \( S \), such that

1. \( \text{Sat}(\Psi) \subseteq T \) and
2. \( s \in \text{Sat}(\Phi) \) and \( \text{Post}(s) \cap T \neq \emptyset \) implies \( s \in T \),

(g) \( \text{Sat}(\exists \Box \Phi) \) is the largest subset \( T \) of \( S \), such that

3. \( T \subseteq \text{Sat}(\Phi) \) and
4. \( s \in T \) implies \( \text{Post}(s) \cap T \neq \emptyset \).

These two last characterizations are based upon equation induced by the expansion laws for \( \exists (\Phi \cup \Psi) \) and \( \exists \Box \Phi \) respectively.
Fixpoint Representation

- Let $TS = (S, Act, \rightarrow, I, AP, L)$ be an arbitrary finite Transition System with no terminal state. The set $P(S)$ of all subsets of $S$ form a *lattice* under the set inclusion ordering.

- Based on first order logic, a subset $S'$ can be viewed as a predicate which is true for every $s_i$ which is a member of $S'$.

- Each element $S'$ of the lattice is a predicate on $S$.

- A function that maps $P(S)$ to $P(S)$ is called a *Predicate Transformer*.
Fixpoint theory

Let $\tau : \mathcal{P}(S) \rightarrow \mathcal{P}(S)$ be a predicate transformer, then:

- $\tau$ is **Monotonic** provided that $P \subseteq Q$ implies $\tau(P) \subseteq \tau(Q)$;
- $\tau$ is **U-continuous** provided that $P_1 \subseteq P_2 \subseteq \ldots$ implies $\tau(\bigcup_i P_i) = \bigcup_i \tau(P_i)$;
- $\tau$ is **$\cap$-continuous** provided that $P_1 \subseteq P_2 \subseteq \ldots$ implies $\tau(\bigcap_i P_i) = \bigcap_i \tau(P_i)$. 

Fixpoint Theory: Predicate Transformer Properties

- **Lemma 1:** If $S$ is finite and $\tau$ is monotonic, then $\tau$ is also $U$-continuous and $\bigcap$-continuous.

- **Lemma 2:** If $\tau$ is monotonic, then for every $i$, $\tau^i(False) \subseteq \tau^{i+1}(False)$ and $\tau^i(True) \supseteq \tau^{i+1}(True)$.

- **Lemma 3:** If $\tau$ is monotonic and $S$ is finite, then there is an integer $i_0$ such that for every $j \geq i_0$, $\tau^i(False) = \tau^{i_0}(False)$. Similarly, there is some $j_0$ such that for every $j \geq j_0$, $\tau^i(True) = \tau^{j_0}(True)$. 
Lemma 4: If $\tau$ is monotonic and $S$ is finite, then there is an integer $i_0$ such that $\mu Z . \tau(Z) = \tau^{i_0}(\text{False})$. Similarly, there is some $j_0$ such that $\nu Z . \tau(Z) = \tau^{j_0}(\text{True})$. 
Fixpoint theory: Greatest and Least Fixpoints

- Let $\tau^i(Z)$ denote $i$ applications of $\tau$ to $Z$.
- According to Tarski, any monotonic predicate transformer $\tau$ on $P(S)$ always has a least fixpoint $\mu Z \cdot \tau(Z)$, and a greatest fixpoint $\nu Z \cdot \tau(Z)$ where
  
  $\mu Z \cdot \tau(Z) = \cap \{Z \mid \tau(Z) \subseteq Z\}$ and
  
  $\nu Z \cdot \tau(Z) = \cup \{Z \mid \tau(Z) \supseteq Z\}$.

- Additionally, if $\tau$ is $U$-continuous $\mu Z \cdot \tau(Z) = U_i \tau^i$ (False) and if $\tau$ is $\cap$-continuous $\nu Z \cdot \tau(Z) = \cap_i \tau^i$ (True).
CTL operators as greatest or least fixpoints

If we identify each CTL formula $f$ with the predicate $\{s \mid M, s \models f \}$, then each of the basic CTL operators may be characterized as a least or greatest fixpoint of an appropriate predicate transformer as follows:

- $\forall \Box f_1 = \mu Z . f_1 \lor \forall \bigcirc Z$
- $\exists \Box f_1 = \mu Z . f_1 \lor \exists \bigcirc Z$
- $\forall \square f_1 = \nu Z . f_1 \land \forall \bigcirc Z$
- $\exists \square f_1 = \nu Z . f_1 \land \exists \bigcirc Z$
- $\forall [f_1 U f_2] = \mu Z . f_2 \lor (f_1 \land \forall \bigcirc Z)$
- $\exists [f_1 U f_2] = \mu Z . f_2 \lor (f_1 \land \exists \bigcirc Z)$
Procedure for computing least fixpoint

function Lfp(Tau : PredicateTransformer) : Predicate
  Q := False;
  Q' := Tau(Q);
  while (Q ≠ Q') do
    Q := Q';
    Q' := Tau(Q');
  end while;
  return(Q);
end function
Procedure for computing greatest fixpoint

function Gfp(Tau: PredicateTransformer): Predicate
    Q := True;
    Q' := Tau(Q);
    while (Q ≠ Q') do
        Q := Q';
        Q' := Tau(Q');
    end while;
    return(Q);
end function
Fixpoint Theory: Sequence of Approximations

Original Transition System

\( \tau^1 \) (False)

\( \tau^2 \) (False)

\( \tau^3 \) (False)
Consider the case of $\exists(\Phi U \Psi)$:

$\exists(\Phi U \Psi) \equiv \Psi \lor (\Phi \land \exists \Box \exists(\Phi U \Psi))$

We can describe the CTL formula $\exists(\Phi U \Psi)$ as a fixed-point of the logical equation

$F \equiv \Psi \lor (\Phi \land \exists \Box F)$

A fixed-point is simply a solution of the above equation
There are other solutions too: $\exists (\Phi W \Psi)$

- But $\exists (\Phi U \Psi)$ is the least solution to the former equation
- Whereas $\exists (\Phi W \Psi)$ is the largest solution

In a similar way, $\exists \square \Phi$ can be considered as the greatest fixed point of the logical equation

- $F = \Phi \land \exists \Diamond F$
CTL Model Checking: Fixed-point computation

- As an example, the algorithm to compute $\exists (\Phi_1 \cup \Phi_2)$ is given below:

  $\exists (\Phi_1 \cup \Phi_2)$:
  
  $T := \text{Sat}(\Phi_2)$;  (* compute the smallest fixed point *)
  
  while $\{ s \in \text{Sat}(\Phi_1) \setminus T \mid \text{Post}(s) \cap T \neq \emptyset \} \neq \emptyset$ do
    
    let $s \in \{ s \in \text{Sat}(\Phi_1) \setminus T \mid \text{Post}(s) \cap T \neq \emptyset \}$;

    $T := T \cup \{ s \}$;

  od;

  return $T$;

$S \setminus T$: all members of $S$ which are not a member of $T$
Computation of the Satisfaction Sets

Algorithm 14 Computation of the satisfaction sets

Input: finite transition system $\text{TS}$ with state set $S$ and CTL formula $\Phi$ in ENF
Output: $\text{Sat}(\Phi) = \{ s \in S \mid s \vDash \Phi \}$

(* recursive computation of the sets $\text{Sat}(\Psi)$ for all subformulae $\Psi$ of $\Phi$ *)

switch($\Phi$):

true : return $S$;
a : return $\{ s \in S \mid a \in L(s) \}$;
$\Phi_1 \land \Phi_2$ : return $\text{Sat}(\Phi_1) \cap \text{Sat}(\Phi_2)$;
$\neg \Psi$ : return $S \setminus \text{Sat}(\Psi)$;
$\exists \bigcirc \Psi$ : return $\{ s \in S \mid \text{Post}(s) \cap \text{Sat}(\Psi) \neq \emptyset \}$;
$\exists(\Phi_1 \cup \Phi_2)$ : $T := \text{Sat}(\Phi_2)$; (* compute the smallest fixed point *)
while $\{ s \in \text{Sat}(\Phi_1) \setminus T \mid \text{Post}(s) \cap T \neq \emptyset \} \neq \emptyset$ do
    let $s \in \{ s \in \text{Sat}(\Phi_1) \setminus T \mid \text{Post}(s) \cap T \neq \emptyset \}$;
    $T := T \cup \{ s \}$;
endwhile;
return $T$;

$\exists \square \Phi$ : $T := \text{Sat}(\Phi)$; (* compute the greatest fixed point *)
while $\{ s \in T \mid \text{Post}(s) \cap T = \emptyset \} \neq \emptyset$ do
    let $s \in \{ s \in T \mid \text{Post}(s) \cap T = \emptyset \}$;
    $T := T \setminus \{ s \}$;
endwhile;
return $T$;

end switch
Example 6.26. Checking the formula $\exists \Diamond \Phi$ with $\Phi = ((a = c) \land (a \neq b))$ in the following transition system:
As we know, $\exists \diamond \Phi = \exists (\text{true} \lor \Phi)$

Invoking the former algorithm given, Sat(\text{true}) and Sat((a = c) \land (a \neq b)) are recursively computed. The states of set T are:
In the next two steps, $s_6$ and $s_7$ are added respectively, finally yielding the satisfaction set:
CTL Model Checking: Alternative Algorithms

- A possibility to compute $\text{Sat}(\exists \square \Phi)$ is to only consider the $\Phi$-states of a transition system $TS$ and ignore all $\neg \Phi$-states:

- from the TS given in example 6.26, consider the example with $\Phi = b$:
In the next step, all nontrivial strongly connected components (SCCs) in the state graph induced by $TS[\Phi]$ are computed:

- Nontrivial SCC: A SCC containing at least one transition
Finally, all states which satisfy $\Phi$ and can reach a state in the strongly connected components are added:

**Theorem 6.29.** $s \models \exists \Box \Phi$ iff $s \models \Phi$ and there is a nontrivial SCC in $TS[\Phi]$ reachable from $s$. 
Algorithm 15 Enumerative backward search for computing $\text{Sat}(\exists (\Phi \cup \Psi))$

Input: finite transition system $TS$ with state set $S$ and CTL formula $\exists (\Phi \cup \Psi)$
Output: $\text{Sat}(\exists (\Phi \cup \Psi)) = \{ s \in S | s \models \exists (\Phi \cup \Psi) \}$

\[
\begin{align*}
E & := \text{Sat}(\Psi); \\
T & := E; \\
\text{while } E \neq \emptyset \text{ do} \\
& \quad \text{let } s' \in E; \\
& \quad E := E \setminus \{ s' \}; \\
& \quad \text{for all } s \in \text{Pre}(s') \text{ do} \\
& \quad \quad \text{if } s \in \text{Sat}(\Phi) \setminus T \text{ then } E := E \cup \{ s \}; T := T \cup \{ s \} \text{ fi} \\
& \quad \text{od} \\
& \quad \text{od} \\
& \text{return } T
\end{align*}
\]
CTL Model Checking: Alternative Algorithms

Algorithm 16 Enumerative backward search to compute $\text{Sat}(\exists \square \Phi)$

Input: finite transition system $TS$ with state set $S$ and CTL formula $\exists \square \Phi$
Output: $\text{Sat}(\exists \square \Phi) = \{ s \in S \mid s \models \exists \square \Phi \}$

\[
\begin{align*}
E & := S \setminus \text{Sat}(\Phi); \quad (* \text{E contains any not visited } s' \text{ with } s' \not\models \exists \square \Phi *) \\
T & := \text{Sat}(\Phi); \quad (* \text{T contains any } s \text{ for which } s \models \exists \square \Phi \text{ has not yet been disproven } *) \\
\text{for all } s \in \text{Sat}(\Phi) \text{ do } & \text{count}[s] := | \text{Post}(s) |; \quad \text{od} \quad (* \text{initialize array count } *) \\
\text{while } E \neq \emptyset \text{ do } & \\
\quad \text{let } s' \in E; \\
\quad E & := E \setminus \{ s' \}; \\
\quad \text{for all } s \in \text{Pre}(s') \text{ do } \\
\quad \quad \text{if } s \in T \text{ then } & \text{count}[s] := \text{count}[s] - 1; \\
\quad \quad \text{if } \text{count}[s] = 0 \text{ then } & T := T \setminus \{ s \}; \\
\quad \quad \text{E} & := E \cup \{ s \}; \\
\quad \quad \text{fi} \\
\quad \text{fi} \\
\quad \text{od} \\
\text{od} \\
\text{return } T
\end{align*}
\]
CTL Model Checking: Time and Space Complexity

- Let TS be a finite transition system with N states and K transitions.
- The time complexity of Algorithm 14 (see page 348, partly described in slide 54) is in:
  - $O((N+K) \cdot |\Phi|)$
Theorem 6.30. For transition system TS with N states and K transitions, and CTL formula Φ, the CTL model-checking problem TS |= Φ can be determined in time

\[ O((N+K) \cdot |\Phi|) \]

CTL Model-checking is Linear in contrast to the exponential time Model-checking algorithm given for LTL.
CTL Model Checking: Time and Space Complexity (Cont.)

- But this should not be interpreted as “CTL model checking is more efficient than LTL model checking”.

- LTL formulae may be exponentially shorter than any of their equivalent formulation in CTL.
Example 6.31. The Hamiltonian Path Problem:

From Lemma 5.45, it follows that the Hamiltonian path problem for digraphs with \( n \) nodes can be encoded in LTL formula \( \phi_n \) whose length is polynomial in the number of nodes, \( n \).
We want to find a Hamiltonian path in a directed graph \( G = (V,E) \) with \( V = \{v_1, \ldots, v_n\} \)

We want to define a \( \Phi_n \) such that:

\( G \) contains a Hamiltonian path if and only if \( TS \models \neg \Phi_n \)
Define TS = (V ∪ {b}, {τ}, →, V, V, L): The labels correspond to graph nodes, and the transitions are the same as edges, except that there is a transition to b added to every node.

In this example, (a) is the original graph:
Let $\Phi_n$ be defined as follows:

$$\Phi_n = \bigvee_{(i_1, \ldots, i_n) \text{ permutation of } (1, \ldots, n)} \Psi(v_{i_1}, v_{i_2}, \ldots, v_{i_n})$$

Such that $\Psi(v_{i_1}, \ldots, v_{i_n})$ is a CTL formula that is satisfied if and only if $v_{i_1}, v_{i_2}, \ldots, v_{i_n}$ is a Hamiltonian path in $G$. 
The formulae $\Psi(v_{i1}, \ldots, v_{in})$ are inductively defined as follows:

1. $\Psi(v_i) = v_i$
2. $\Psi(v_{i1}, v_{i2}, \ldots, v_{in}) = v_{in} \land \exists \bigcirc \Psi(v_{i1}, \ldots, v_{in})$ if $n > 1$

Stated in words, $\Psi(v_{i1}, v_{i2}, \ldots, v_{in})$ holds if there exists a path on which $v_{i1}, v_{i2}, \ldots$ successively hold.
According to this definition, it is easy to infer that:

$$\text{Sat}(\psi(v_{i_1}, v_{i_2}, \ldots, v_{i_n})) =$$

- $$\{v_{i_1}\}$$ if $$v_{i_1}, v_{i_2}, \ldots, v_{i_n}$$ is a Hamiltonian path in $$G$$
- $$\emptyset$$ otherwise

Thus, $$\text{TS} \not\models \neg\Phi_n$$ iff there is an initial state $$s$$ of $$\text{TS}$$ for which $$s \models \Phi$$, which is, there exists a $$v$$ in $$G$$ and a permutation $$i, \ldots, i_{of} 1, \ldots, n$$ such that $$v = v_i$$ and $$v_{i_1}, \ldots, v_{i_n}$$ is a Hamiltonian path in $$G$$.
By the enumeration of all possible permutations we obtain a formula with a length exponential in the number of vertices in the graph.

And we do not expect a smaller formulae, as we know the Hamiltonian path problem is not yet solved polynomially.
Fairness in CTL

- Fairness assumptions can be added as premise to the LTL formula to be verified.
  - $\text{TS} \models_{\text{fair}} \phi$ if and only if $\text{TS} \models (\text{fair} \rightarrow \phi)$

- Fairness constraints operate on the path level and replace the standard meaning “for all/some paths” of quantification with “for all/some fair paths”
If “fair” expresses the fairness condition on the path level, then CTL formulae that encode the intuitive meaning of $\forall (\text{fair} \rightarrow \phi)$ and $\exists (\text{fair} \land \phi)$ would be needed. However, these are not legal CTL formulas.

Moreover, this approach is not possible for CTL, based on the fact that most fairness constraints cannot be encoded into a CTL formula:

- Persistence property $\Diamond \Box a$
Therefore, in order to deal with fairness constraints in CTL, the semantics of CTL is slightly modified such that the state formulae $\forall \phi$ and $\exists \phi$ are interpreted over all fair paths rather than over all possible paths.

It is assumed that the given fixed fairness constraint is described by a formula as in LTL.
Definition 6.32. A strong CTL fairness constraint (over AP) is a term of the form:

\[ sfair = \bigwedge_{1 \leq i \leq k} (\Box \Diamond \Phi_i \rightarrow \Box \Diamond \Psi_i) \]

where \( \Phi_i \) and \( \Psi_i \) are CTL formulae over AP.

Weak and unconditional CTL fairness constraints are defined analogously by conjunctions of terms of the form \( \Diamond \Box \Phi_i \rightarrow \Box \Diamond \Psi_i \) and \( \Box \Diamond \Psi_i \), respectively.
Fairness in CTL: CTL Fairness Assumptions (Cont.)

- A **CTL Fairness Assumption** is a conjunction of strong, weak, and unconditional CTL fairness constraints.
- Note that CTL fairness assumptions are not CTL path formulae, but they can be viewed as “LTL formulae using CTL state formulae instead of atomic propositions”.
Let \( TS = (S, \text{Act}, \rightarrow, I, \text{AP}, L) \) be a transition system without terminal states, \( \pi \) be an infinite path fragment in \( TS \), and \( fair \) a fixed CTL fairness assumption.

\( \pi \models fair \) denotes that \( \pi \) satisfies the formula \( fair \), where \( \models \) should be read as the LTL semantics.
The semantics of CTL under fairness assumption \textit{fair} is identical to the semantics given earlier, except, the path quantifications range over all fair paths rather than over all paths:

$$\text{FairPaths}(s) = \{ \pi \in \text{Paths}(s) \mid \pi \models \text{fair} \}$$

And for the transition system TS:

$$\text{FairPaths}(TS) = \bigcup_{s_0 \in I} \text{FairPaths}(s_0)$$
Definition 6.33. For state formulae, the satisfaction relation $|=_{fair}$ for CTL fairness assumption $fair$ is defined as:

- $s |=_{fair} a$ iff $a \in L(s)$
- $s |=_{fair} \neg \Phi$ iff not $s |=_{fair} \Phi$
- $s |=_{fair} \Phi \land \Psi$ iff $(s |=_{fair} \Phi)$ and $(s |=_{fair} \Psi)$
- $s |=_{fair} \exists \varphi$ iff $\pi |=_{fair} \varphi$ for some $\pi \in FairPaths(s)$
- $s |=_{fair} \forall \varphi$ iff $\pi |=_{fair} \varphi$ for all $\pi \in FairPaths(s)$
Fairness in CTL: CTL Fairness Assumptions (Cont.)

For a path $\pi$ and path formulae the satisfaction relation $|=_{\text{fair}}$ for CTL fairness assumption $\text{fair}$ is defined as:

\[
\pi |=_{\text{fair}} \square \Phi \text{ iff } \pi[1] |=_{\text{fair}} \Phi \\
\pi |=_{\text{fair}} \Phi \cup \Psi \text{ iff } \exists j \geq 0. (\pi[j] |=_{\text{fair}} \Psi \land (\forall 0 \leq k < j. \pi[k] |=_{\text{fair}} \Phi))
\]

where for path $\pi = s_0 s_1 s_2 \ldots$ and integer $i \geq 0$, $\pi[i]$ denotes the $(i+1)$-th state of $\pi$, $s_i$
For CTL-state formula $\Phi$, and CTL fairness assumption $\text{fair}$, the satisfaction set $\text{Sat}_{\text{fair}}(\Phi)$ is defined by:

$$\text{Sat}_{\text{fair}}(\Phi) = \{ s \in S | s \models_{\text{fair}} \Phi \}.$$ 

The transition system $TS$ satisfies CTL formula $\Phi$ under fairness assumption $\text{fair}$ if and only if $\Phi$ holds in all initial states of $TS$:

$$TS \models_{\text{fair}} \Phi \text{ if and only if } \forall s_0 \in I. \ s_0 \models_{\text{fair}} \Phi.$$
CTL Model Checking with Fairness (Basic Idea)

Algorithm 17 CTL model checking with fairness (basic idea)

Input: finite transition system $TS$, CTL formula $\Phi$ in ENF, and CTL fairness assumption $fair$ over $k$ CTL state formulae $\Phi_i$ and $\Psi_i$

Output: $TS \models_{fair} \Phi$

for all $0 < i \leq k$ do
    determine $Sat(\Phi_i)$ and $Sat(\Psi_i)$
    if $s \in Sat(\Phi_i)$ then $L(s) := L(s) \cup \{ a_i \};$ fi
    if $s \in Sat(\Psi_i)$ then $L(s) := L(s) \cup \{ b_i \};$ fi
od

compute $Sat_{fair}(\exists \Box true) = \{ s \in S \mid FairPaths(s) \neq \emptyset \}$;
forall $s \in Sat_{fair}(\exists \Box true)$ do $L(s) := L(s) \cup \{ a_{fair} \}$ od

(* compute $Sat_{fair}(\Phi)$ *)

for all $i \leq |\Phi|$ do
    for all $\Psi \in Sub(\Phi)$ with $|\Psi| = i$ do
        switch($\Psi$):
        true : $Sat_{fair}(\Psi) := S$;
        $a$ : $Sat_{fair}(\Psi) := \{ s \in S \mid a \in L(s) \}$;
        $a \land a'$ : $Sat_{fair}(\Psi) := \{ s \in S \mid a, a' \in L(s) \}$;
        $\neg a$ : $Sat_{fair}(\Psi) := \{ s \in S \mid a \notin L(s) \}$;
        $\exists \Box a$ : $Sat_{fair}(\Psi) := Sat(\exists \Box (a \land a_{fair}))$;
        $\exists(a \lor a')$ : $Sat_{fair}(\Psi) := Sat(\exists(a \lor (a' \land a_{fair})))$;
        $\exists \Box a$ : compute $Sat_{fair}(\exists \Box a)$
        end switch
    replace all occurrences of $\Psi$ in $\Phi$ by the atomic proposition $a_\Psi$;
    forall $s \in Sat_{fair}(\Psi)$ do $L(s) := L(s) \cup \{ a_\Psi \}$ od
od
return $I \subseteq Sat_{fair}(\Phi)$
Theorem 6.39. Model-Checking CTL with Fairness

The model-checking problem for CTL with fairness can be reduced to

- the model-checking problem for CTL (without fairness), and
- the problem of computing $\text{Sat}_{\text{fair}}(\exists \square a)$ for the atomic proposition $a$. 
Computation of $\text{Sat}_{\text{sfair}}(\exists\Box a)$

Algorithm 18 Computation of $\text{Sat}_{\text{sfair}}(\exists\Box a)$

Input: A finite TS without terminal states, $a \in \text{AP}$ and $\text{fair} = \bigwedge_{0 < i \leq k} \text{sfair}_i$ with $\text{sfair}_i = \Box \Diamond a_i \rightarrow \Box \Diamond b_i$

Output: $\{ s \in S \mid s \models_{\text{fair}} \exists \Box a \}$

compute the SCCs of the state graph $G[a]$ of $TS[a]$;
$T := \emptyset$;
for all nontrivial SCCs $C$ in $G[a]$ do
  (* check whether the fairness assumption $\text{sfair}$ can be realized in $C$ *)
  if $\text{CheckFair}(C, k, \text{sfair}_1, \ldots, \text{sfair}_k)$ then
    $T := T \cup C$
  fi
od
return $\{ s \in S \mid \text{Reach}_{G[a]}(s) \cap T \neq \emptyset \}$ (* e.g., backwards reachability *)
Algorithm 19 Recursive algorithm \( \text{CheckFair}(C, k, sfaire_1, \ldots, sfaire_k) \)

\[\text{Input:} \text{ nontrivial SCC } C \text{ in } G[a], \text{ and strong fairness constraints } sfaire_i = \square \diamond a_i \rightarrow \square \diamond b_i, \ i = 1, \ldots, k.\]

\[\text{Output:} \text{ true if } \bigwedge_{1 \leq i \leq k} sfaire_i \text{ is realizable in } C. \text{ Otherwise false.}\]

\[
\begin{align*}
\text{if } \forall i \in \{1, \ldots, k\}. C \cap \text{Sat}(b_j) \neq \emptyset \text{ then} \\
\quad & \text{return true} \\
\text{else} \\
\quad & \text{choose an index } j \in \{1, \ldots, k\} \text{ with } C \cap \text{Sat}(b_j) = \emptyset; \\
\quad & \text{if } C[\neg a_j] \text{ is acyclic (or empty) then} \\
\quad & \quad \text{return false} \\
\quad & \text{else} \\
\quad & \quad \text{compute the nontrivial SCCs of } C[\neg a_j]; \\
\quad & \quad \text{for all nontrivial SCC } D \text{ of } C[\neg a_j] \text{ do} \\
\quad & \quad & \text{if } \text{CheckFair}(D, k - 1, sfaire_1, \ldots, sfaire_{j-1}, sfaire_{j+1}, \ldots, sfaire_k) \text{ then} \\
\quad & \quad & \quad \text{return true} \\
\quad & \quad & \fi \\
\quad & \fi \\
\fi \\
\text{return false}
\end{align*}
\]
Theorem 6.42. Time Complexity of Verifying $\exists \Box a$ under Fairness

For transition system TS with $N$ states and $K$ transitions, and CTL strong fairness assumption fair with $k$ conjuncts, the set $\text{Sat}_{fair}(\exists \Box a)$ can be computed in $\mathcal{O}((N+K) \cdot k)$. 
Theorem 6.43. Time Complexity of CTL Model Checking with Fairness

For transition system TS with $N$ states and $K$ transitions, CTL formula $\Phi$, and CTL fairness assumption fair with $k$ conjuncts, the CTL model-checking problem $TS \models_{\text{fair}} \Phi$ can be determined in time $O((N+K) \cdot |\Phi| \cdot k)$. 
Fairness in CTL: CTL Fairness Assumption (Cont.)

- Read **Example 6.35**.

**Example 6.36.** Mutual Exclusion:
- Consider the semaphore-based solution to the two-process mutual exclusion problem. The transition system of this concurrent program is denoted $TS_{sem}$. 
The CTL formula:

\[ \Phi = (\forall \square \forall \diamond \text{crit}_1) \land (\forall \square \forall \diamond \text{crit}_2) \]

Describes the *liveness* property that both processes infinitely often have access to the critical section.

It follows that \( TS_{sem} \not\models \Phi \) (The semaphore solution does not fulfill the liveness property)
Again, consider the arbiter-based solution to the mutual exclusion problem.

The decision as to which process will acquire access to the critical section is determined by a coin flipped by the arbiter.

Consider the unconditional fairness assumption:
\[ u_{fair} = \square \diamond \text{head} \land \square \diamond \text{tail} \]
Which requires that the events “heads” and “tails” occur infinitely often with probability 1

It follows that:
\[ TS_1 \parallel \text{Arbiter} \parallel TS_2 \neq \Phi, \text{ and } TS_1 \parallel \text{Arbiter} \parallel TS_2 \neq u_{fair} \Phi \]
Syntax of CTL*

- CTL* state formulae over the set AP of atomic proposition, briefly called CTL* formulae, are formed according to the following grammar:

\[
\Phi ::= \text{true} \mid a \mid \Phi_1 \land \Phi_2 \mid \neg \Phi \mid \exists \phi
\]

where \( a \in AP \) and \( \phi \) is a path formula. The CTL* path formulae is given by the following grammar:

\[
\phi ::= \Phi \mid \phi_1 \land \phi_2 \mid \neg \phi \mid \diamond \phi \mid \phi_1 \mathcal{U} \phi_2
\]

where \( \Phi \) is a state formula, \( \phi, \phi_1, \) and \( \phi_2 \) are path formulae.
Definition 6.81. Satisfaction Relation for CTL*

Let $a \in AP$ be an atomic proposition, $TS = (S, Act, \rightarrow, I, AP, L)$ be a transition system without terminal states, state $s \in S$, $\Phi, \Psi$ be CTL* state formulae, and $\varphi, \varphi_1$ and $\varphi_2$ be CTL* path formulae. The satisfaction relation $\models$ is defined for state formulae by

\[
\begin{align*}
  s \models a & \quad \text{iff} \quad a \in L(s), \\
  s \models \neg \Phi & \quad \text{iff} \quad \text{not } s \models \Phi, \\
  s \models \Phi \land \Psi & \quad \text{iff} \quad (s \models \Phi) \text{ and } (s \models \Psi), \\
  s \models \exists \varphi & \quad \text{iff} \quad \pi \models \varphi \text{ for some } \pi \in \text{Paths}(s).
\end{align*}
\]

For path $\pi$, the satisfaction relation $\models$ for path formulae is defined by:

\[
\begin{align*}
  \pi \models \Phi & \quad \text{iff} \quad s_0 \models \Phi, \\
  \pi \models \varphi_1 \land \varphi_2 & \quad \text{iff} \quad \pi \models \varphi_1 \text{ and } \pi \models \varphi_2, \\
  \pi \models \neg \varphi & \quad \text{iff} \quad \pi \models \varphi, \\
  \pi \models \bigcirc \varphi & \quad \text{iff} \quad \pi[1..] \models \varphi, \\
  \pi \models \varphi_1 \mathcal{U} \varphi_2 & \quad \text{iff} \quad \exists j \geq 0. \ (\pi[j..] \models \varphi_2 \land (\forall 0 \leq k < j. \pi[k..] \models \varphi_1))
\end{align*}
\]

where for path $\pi = s_0 s_1 s_2 \ldots$ and integer $i \geq 0$, $\pi[i..]$ denotes the suffix of $\pi$ from index $i$ on.  

\[\square\]
**Definition 6.82.** CTL* Semantics for Transition Systems

For CTL*-state formula $\Phi$, the *satisfaction set* $\text{Sat}(\Phi)$ is defined by

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

The transition system $TS$ satisfies CTL* formula $\Phi$ if and only if $\Phi$ holds in all initial states of $TS$:

$$TS \models \Phi \text{ if and only if } \forall s_0 \in I. s_0 \models \Phi.$$
Theorem 6.83. Embedding of LTL in CTL*

Let $TS = (S, \text{Act}, \rightarrow, I, \text{AP}, L)$ be a transition system without terminal states. For each LTL formula $\varphi$ over AP and for each $s \in S$:

$$s \models \varphi \quad \text{if and only if} \quad s \models \forall \varphi$$

$LTL$ semantics \quad $CTL^*$ semantics

In particular, $TS \models \varphi$ (with respect to the LTL semantics) if and only if $TS \models \forall \varphi$ (with respect to the $CTL^*$ semantics)
Theorem 6.84. CTL* is More Expressive Than LTL and CTL

For the CTL* formula over \( \text{AP} = \{a, b\} \),

\[ \Phi = (\forall \Diamond \Box a) \lor (\forall \Box \exists \Diamond b), \]

there does not exist any equivalent LTL or CTL formula.
Relationship between LTL, CTL, and CTL*

Figure 6.27: Relationship between LTL, CTL, and CTL*.
Some Equivalence Laws for CTL*

\[
\begin{align*}
\text{duality laws for path quantifiers} \\
\neg \forall \varphi & \equiv \exists \neg \varphi \\
\neg \exists \varphi & \equiv \forall \neg \varphi \\
\text{distributive laws} \\
\forall(\varphi_1 \land \varphi_2) & \equiv \forall \varphi_1 \land \forall \varphi_2 \\
\exists(\varphi_1 \lor \varphi_2) & \equiv \exists \varphi_1 \lor \exists \varphi_2 \\
\text{quantifier absorption laws} \\
\forall \square \Diamond \varphi & \equiv \forall \square \forall \varphi \\
\exists \Diamond \square \varphi & \equiv \exists \Diamond \exists \square \varphi
\end{align*}
\]

Figure 6.28: Some equivalence laws for CTL*.
Maximal Proper State Subformula

Definition 6.86. Maximal Proper State Subformula

State formula $\Psi$ is a maximal proper state subformula of $\Phi$ whenever $\Psi$ is a subformula of $\Phi$ that differs from $\Phi$ and that is not contained in any other proper state subformula of $\Phi$. 
Example of CTL* Model Checking

Example 6.87. Abstract Example of CTL* Model Checking

The CTL* model-checking approach is illustrated by considering the CTL* formula:

$$\exists \varphi \text{ where } \varphi = \lozenge (\forall \square \exists \Diamond a) \land \Diamond \lozenge (\lozenge a \land \square b).$$

The maximal proper state subformulae of $\varphi$ are

$$\Phi_1 = \forall \square \exists \Diamond a \text{ and } \Phi_2 = \exists (\lozenge a \land \square b).$$

Thus:

$$\varphi = \lozenge (\forall \square \exists \Diamond a) \land \Diamond \lozenge (\lozenge a \land \square b) = \lozenge \Phi_1 \land \Diamond \Phi_2.$$
CTL* Model Checking Algorithm

Algorithm 27 CTL* model checking algorithm (basic idea)

Input: finite transition system $TS$ with initial states $I$, and CTL* formula $\Phi$

Output: $I \subseteq \text{Sat}(\Phi)$

for all $i \leq |\Phi|$ do
  for all $\Psi \in \text{Sub}(\Phi)$ with $|\Psi| = i$ do
    \textbf{switch}(\Psi):
    \begin{align*}
      \text{true} & : \text{Sat}(\Psi) := S; \\
      a & : \text{Sat}(\Psi) := \{ s \in S \mid a \in L(s) \}; \\
      a_1 \land a_2 & : \text{Sat}(\Psi) := \text{Sat}(a_1) \cap \text{Sat}(a_2); \\
      \neg a & : \text{Sat}(\Psi) := S \setminus \text{Sat}(a); \\
      \exists \varphi & : \text{determine Sat}_{LTL}(\neg \varphi) by means of an LTL model-checker; \\
      & : \text{Sat}(\Psi) := S \setminus \text{Sat}_{LTL}(\neg \varphi)
    \end{align*}
  \textbf{end switch}
  \text{AP} := \text{AP} \cup \{ a_\varphi \}; \\
  \text{replace } \Psi \text{ with } a_\varphi \\
  \text{forall } s \in \text{Sat}(\Psi) \text{ do } L(s) := L(s) \cup \{ a_\varphi \}; \text{ od}
  \text{od}
  \text{return } I \subseteq \text{Sat}(\Phi)
Time Complexity of CTL*

Theorem 6.88. Time Complexity of CTL* Model Checking

For transition system TS with N states and K transitions, and CTL* formula $\Phi$, the CTL* model-checking problem $TS \models \Phi$ can be determined in time $O((N+K) \cdot 2^{|\Phi|})$. 
### Complexity of the Model Checking and Algorithms and Satisfiability Checking

<table>
<thead>
<tr>
<th></th>
<th>CTL</th>
<th>LTL</th>
<th>CTL*</th>
</tr>
</thead>
<tbody>
<tr>
<td>model checking</td>
<td>PTIME</td>
<td>PSPACE-complete</td>
<td>PSPACE-complete</td>
</tr>
<tr>
<td>without fairness</td>
<td>$size(TS) \cdot</td>
<td>\Phi</td>
<td>$</td>
</tr>
<tr>
<td>with fairness</td>
<td>$size(TS) \cdot</td>
<td>\Phi</td>
<td>\cdot</td>
</tr>
<tr>
<td>for fixed specifications (model complexity)</td>
<td>$size(TS)$</td>
<td>$size(TS)$</td>
<td>$size(TS)$</td>
</tr>
<tr>
<td>satisfiability check</td>
<td>EXPTIME</td>
<td>PSPACE-complete</td>
<td>2EXPTIME</td>
</tr>
<tr>
<td>best known technique</td>
<td>$exp(</td>
<td>\Phi</td>
<td>)$</td>
</tr>
</tbody>
</table>

Figure 6.29: Complexity of the model-checking algorithms and satisfiability checking.
Branching Time vs. Linear Time (in a Nutshell)

<table>
<thead>
<tr>
<th>Aspect</th>
<th>Linear time</th>
<th>Branching time</th>
</tr>
</thead>
<tbody>
<tr>
<td>“behavior” in a state $s$</td>
<td>path-based: $\operatorname{trace}(s)$</td>
<td>state-based: computation tree of $s$</td>
</tr>
<tr>
<td>temporal logic</td>
<td>$\mathsf{LTL}$: path formulae $\phi$</td>
<td>$\mathsf{CTL}$: state formulae $\exists \phi$</td>
</tr>
<tr>
<td></td>
<td>$s \models \phi \iff \forall \pi \in \operatorname{Paths}(s), \pi \models \phi$</td>
<td>existential path quantification $\exists \phi$</td>
</tr>
<tr>
<td>complexity of the model checking problems</td>
<td>$\mathsf{PSPACE}$-complete</td>
<td>$\mathsf{PSPACE}$-complete</td>
</tr>
<tr>
<td></td>
<td>$O(</td>
<td>TS</td>
</tr>
<tr>
<td>implementation relation</td>
<td>trace inclusion and the like (proof is $\mathsf{PSPACE}$-complete)</td>
<td>simulation and bisimulation (proof in polynomial time)</td>
</tr>
<tr>
<td>fairness</td>
<td>no special techniques needed</td>
<td>special techniques needed</td>
</tr>
</tbody>
</table>