Computational Tree Logic
CTL
Logics to express the branching time properties

- CTL belongs to the family of *branching-time logics*
- CTL formulas describes properties of a *computation tree* → reason about many executions at once.
- CTL semantics is defined in terms of states.

**Difference against LTL**

- *LTL describes properties of individual executions.*
- *Its semantics is defined as a set of executions*
Computation tree

Let $K = \langle S, \ell, \rightarrow, s_0 \rangle$ be a Kripke structure defined over a set $AP$ of atomic propositions.

The computation tree of $K$ is an unfilling of $K$ defined as a tree by the tuple $\langle U, \rightarrow', u_0 \rangle$
with a labelling $l : U \rightarrow S$ such that:

- $u_0 \in U$ is the root and $l(u_0) = s_0$
- for some $u \in U$, if $l(u) = s$ and $s \rightarrow s'$ then there is $u \rightarrow' u'$ with $l(u') = s'$

Model checking CTL does not need an explicit computation tree. However, this definition serves to clarify the concepts behind CTL.
Example (continue)

Let $AP = \{r_1, r_2, a_1, a_2\}$
where $r_i$ denotes the request to the critical section by the process $i$
and $a_i$ its access.
Basic syntax of CTL

Combine temporal operators with quantification over run

Let $AP$ be a set of atomic propositions

Inductive construction:

- $a \in AP$ is a CTL formula
- then from CTL formulas $(\phi_1$ and $\phi_2)$, we can build $\neg \phi_1 \ \phi_1 \land \phi_2 \ \text{EX} \ \phi_1 \ \text{EG} \ \phi_1 \ \phi_1 \text{ EU } \phi_2$

where $X$ stands for « next », $G$ for « globally », and $U$ for « until »
Semantics of CTL

- Let $K = \langle S, \ell, \rightarrow, s_0 \rangle$ be a Kripke structure
- To each CTL formula $\phi$, we associate a subset $S_K(\phi)$ of states of $K$ satisfying $\phi$

$s \in S_K(a)$ $\iff a \in \ell(s)$

$s \in S_K(\neg \phi)$ $\iff s \notin S_K(\phi)$

$s \in S_K(\phi_1 \land \phi_2)$ $\iff s \in S_K(\phi_1) \cap S_K(\phi_2)$

$s \in S_K(\text{EX } \phi)$ $\iff \exists s \rightarrow s' \text{ with } s' \in S_K(\phi)$

$s \in S_K(\text{EG } \phi)$ $\iff \exists \text{ a run } \rho \text{ of } K \text{ with } \rho(0) = s$ and $\forall i \geq 0$, $\rho(i) \in S_K(\phi)$

$K$ satisfies a CTL formula $\phi$ iff $s_0 \in S_K(\phi)$
Syntactic CTL sugar

In practice, we will make use of the following abbreviations:

- \( \phi_1 \lor \phi_2 \equiv \neg (\neg \phi_1 \land \neg \phi_2) \)
- \( \phi_1 \implies \phi_2 \equiv \neg \phi_1 \lor \phi_2 \)
- \( \text{true} \equiv \neg a \lor a \)
- \( \text{false} \equiv \neg \text{true} \)

- \( \text{EF} \phi \equiv \text{true} \text{ EU } \phi \)
- \( \phi_1 \text{ EW } \phi_2 \equiv (\phi_1 \text{ EU } \phi_2) \lor \text{ EG } \phi_1 \)
- \( \phi_1 \text{ ER } \phi_2 \equiv \neg (\neg \phi_1 \text{ EU } \neg \phi_2) \)

- \( \text{AX} \phi \equiv \neg \text{EX} \neg \phi \)
- \( \text{AG} \phi \equiv \neg \text{EF} \neg \phi \)
- \( \text{AF} \phi \equiv \neg \text{EG} \neg \phi \)
- \( \phi_1 \text{ AW } \phi_2 \equiv \neg (\neg \phi_2 \text{ EU } \neg (\phi_1 \lor \phi_2)) \)
- \( \phi_1 \text{ AU } \phi_2 \equiv \text{AF} \phi_2 \land (\phi_1 \text{ AW } \phi_2) \)
Semantics illustration

$AX_p$
Semantics illustration

$AF^p$
Semantics illustration

$AG_p$
Semantics illustration

$q \ A U \ p$
Semantics illustration

$E G^p$
Semantics illustration

$EF^p$
Semantics illustration

\[ EX^p \]
Semantics illustration

$q \quad EU \quad p$
Solving formula with nested temporal operators

Proceed inductively
- First compute the states satisfying the innermost formulas;
- then, use the results of computed formulas to solve progressively more (complex) ones.

Example

For $S_K(AF AG x)$ compute successively
- $S_K(x)$,
- $S_K(AG x)$, and
- $S_K(AF AG x)$
First step: compute $S_K(x)$
Second step: compute $S_K(AG\, x)$

To be kept, a state satisfying $x$ must be the root of runs satisfying $x$.

$\Rightarrow$ analyze the predecessors of states satisfying $x$.

Only 2 states remain, with a circuit between them.

$\Rightarrow$ an (infinite) run visit them.
Last step: compute $S_K(\text{AF AG } x)$

- We find the former 2 states
- and also inductively
- all predecessors that eventually reach them in all their runs.

Remove predecessors that do not have all its successors reaching them.
Examples of CTL formulae

• **Reachability**
  \[\text{AG} \neg(a_1 \land a_2)\] : It always holds that \(a_1\) and \(a_2\) do not appear together.
  *For instance, the mutex property: Two processes never enter their critical sections at the same time.*

• **Safety**
  \([\neg x] \text{AW} y\) : It always holds that \(x\) does not occur before the first occurrence of \(y\).
  However, \(y\) may not occur at all, provided \(x\) still never occurs.

• **Liveness**
  \([\neg x] \text{AU} y\) : \(x\) does not occur before the first occurrence of \(y\), and \(y\) does eventually occur.
Other CTL examples

• $AG\ AF_p$
  $p$ appears infinitely often.

• $AG(r_1 \Rightarrow AF a_1)$
  When interpreted on a mutex algorithm:
  whenever process 1 requests to enter its critical section,
  it will eventually succeed.
Expressiveness of CTL and LTL

• CTL and LTL have a large overlap, i.e. properties expressible in both logics.

Examples:
– Invariants (e.g., “p never holds.”)
  \[ \text{AG} \neg p \quad \text{or} \quad \text{G} \neg p \]
– Reactivity (“Whenever p happens, eventually q will happen.”)
  \[ \text{AG} (p \Rightarrow AF q) \quad \text{or} \quad \text{G}(p \Rightarrow Fq) \]
Difference of expressiveness between CTL and LTL

CTL allows to reason about the \textit{branching behaviour}, by considering multiple possible runs at once.

Examples:

- $f_{CTL} = AG \ EF \ p$ (“reset property”, "home state") is not expressible in LTL.

- $f_{CTL} = AF \ AX \ p$ distinguishes the following two systems, but for LTL, $f_{LTL} = FX \ p$ holds in both cases.
Difference of expressiveness between CTL and LTL (continue)

- Even though CTL considers the whole computation tree, its state-based semantics is subtly different from LTL. Thus, there are also properties expressible in LTL but not in CTL.

Examples:
\( f_{\text{LTL}} = FG \ p \) is not expressible in CTL

\begin{align*}
K & \models_{\text{LTL}} \ FG \ p \quad \text{but} \\
K & \not\models_{\text{CTL}} \ AF \ AG \ p
\end{align*}

Remember in CTL that the innermost formulas are treated first.
Difference of expressiveness between CTL and LTL (continue)

- Also, fairness conditions are not directly expressible in CTL:
  \[(GF_{p_1} \land GF_{p_2}) \Rightarrow \phi\]

  However, there is another way to extend CTL with fairness conditions (see later : CTL with fairness).

Remark: A larger logic, called CTL*, combines the expressiveness of CTL and LTL. However, we will not deal with it in this course.
Conclusion
specification CTL vs LTL

✓ The expressiveness of CTL and LTL is incomparable
✓ There is a large overlap
✓ Each logic can express properties that the other cannot

Remark: A larger logic, called CTL*, combines the expressiveness of CTL and LTL. However, we will not deal with it in this course.
CTL Model checking
Verifying CTL formulae

• We now consider the following problem:

Given an atomic proposition set $AP$, a Kripke structure $K = \langle S, \mathcal{L}, \rightarrow, s_0 \rangle$ and a CTL formula $\phi$,

Then compute $S_K(\phi)$.  

This allows to answer the question $K \models \phi$ which is equivalent to $s_0 \in S_K(\phi)$. 
Assumptions

- \( K \) does not contain **deadlocks**, i.e. states without successors.  
  --- For sake of clarity: since every state is guaranteed to have a run (usefull for the semantics of \( EG \) and \( EU \) that require the existence of an (infinite) run).

- \( \phi \) is given in the minimal syntax for CTL (i.e. using only the temporal operator \( EX \), \( EG \) and \( EU \)).  
  If \( \phi \) does not contain extended operators (e.g., AG, EF), then rewrite them first using the equivalences.
The 'bottom-up’ algorithm

The algorithm reduces $\phi$ to a formula to a single proposition, using the following steps:

1. Check whether $\phi$ is a single atomic proposition $p$.
   Then, output $S_K(\phi) = S_K(p) = \{s \in S \mid p \in \ell(s)\}$ and stop.
   Otherwise, continue at 2.

2. Now, $\phi$ must contain some sub-formula $\alpha$ like $\neg p$, $p \land q$, $EX p$, $EG p$, or $p \ EU q$ ($p, q \in AP$)
   Compute $S_K(\alpha)$ using the algorithms on the following slides.

3. Let $p'$ be a ‘fresh’ proposition not yet in $AP$.
   - **Add** $p'$ to $AP$, setting $p' \in \ell(s) \iff \forall s \in S : s \in S_K(\alpha)$
   - **Replace** all occurrences of $\alpha$ in $\phi$ by $p'$.

4. Go back to step 1.
The easy cases of the 'bottom-up' algorithm

Case 1: \( \alpha \equiv \neg p \) (with \( p \in AP \))

\( S_K(p) \) is given by \( \{ s \in S \mid p \in \ell(s) \} \)
thus by definition :

\[ S_K(\neg p) = S \setminus S_K(p) \]

Case 2: \( \alpha \equiv p \land q \) (with \( p, q \in AP \))

\[ S_K(p \land q) = S_K(p) \cap S_K(q) \]

Case 3: \( \alpha \equiv EX \ p \) (with \( p \in AP \))

\[ S_K(EX \ p) = \text{pre}(S_K(p)) \]

\( \text{pre}(X) \) with \( X \subseteq S \),
is defined as the set :

\[ \text{pre}(X) = \{ s \in S \mid \exists t \in X : s \rightarrow t \}. \]
The $EG$ operator

Case 4: $\alpha \equiv EG\ p$ (with $p \in AP$)

$S_K(EG\ p)$ is the greatest solution (w.r.t. $\subseteq$) of the equation

$$X = S_K(p) \cap pre(X)$$

$S_K(EG\ p)$ is the fixed point of the sequence

$S, \pi(S), \pi(\pi(S)), \ldots$ where $\pi(X) = S_K(p) \cap pre(X)$
Initially, $\pi^0(S) = (all) S$
Illustration for $EG_y$

Then, $\pi^l(S) = S_K(y) \cap \text{pre}(S)$

States not satisfying $y$ are excluded
Illustration for $EG_y$

$$\pi^2(S) = S_K(y) \cap \text{pre}(\pi^1(S))$$

States having all its successors outside $\pi^l$ are excluded
The fixed point is now reached
The *EU* operator

Case 5: \( \alpha \equiv p \text{ EU } q \) (with \( p, q \in AP \))

\[ S_K(p \text{ EU } q) \] is the smallest solution (w.r.t. \( \subseteq \)) of the equation

\[ X = S_K(q) \cup (S_K(p) \cap \text{pre}(X)) \]

\( S_K(EG \ p) \) is the fixed point of the sequence

\( \emptyset, \xi(\emptyset), \xi(\xi(\emptyset)), \ldots \) where \( \xi(X) = S_K(q) \cup (S_K(p) \cap \text{pre}(X)) \)
Initially, $\xi^0(\emptyset) = \emptyset$
Illustration for $z \ EU \ y$

Then, $\xi^1(\emptyset) = S_K(y) \cup (S_K(z) \cap pre(\xi^0(\emptyset)))$

States satisfying $y$ are added
Illustration for $z \ EU \ y$

Then, $\xi^2(\emptyset) = S_K(y) \cup (S_K(z) \cap pre(\xi^1(\emptyset)))$

States satisfying $z$ and having at least a successor in $\xi^1$ are added.
Then, $\xi^3(\emptyset) = S_K(y) \cup (S_K(z) \cap \text{pre}(\xi^2(\emptyset)))$
A modeling example

• The following is a description of a microwave oven:

• The oven has the following components:
  – a switch (which is either on or off, initially off);
  – a door (which is either open or closed, initially closed);
  – a plate (which is either hot or cold, initially cold).

• The user may open or close the door.
• He may turn the switch when the oven is off.
• Turning the switch when the door is open has no effect (to prevent accidents).
• When the oven is turned on, it first warms up the plate, then cooks until the dish is ready, and then automatically turns itself off.
• Opening the door makes the heat dissipate.
Example

Behavior of the system

- off, open, cold
  - turn switch
- off, closed, cold
  - close
  - turn switch
- off, closed, hot
  - turn switch
  - done
- on, closed, cold
  - warm up
- on, closed, hot
  - cook
Example
Specification for the oven

The manufacturer wants us to check the following property:
“Whenever the user turns the switch, the plate will eventually become warm.”

- We first formulate the property in CTL:
  \[ AG (\text{switch} \Rightarrow \text{AF warm}) \]

- To check the property, we label the Kripke structure with the two atomic propositions: \text{warm} and \text{switch}
Example (continued)

Note: “Turning the switch” is an action, which we cannot directly model in our state-based semantics, therefore we take all the states which are the target of a switching transition.
Example (continued)

• So, we rewrite the formula:

\[ f_{CTL} = G (\text{switch } \Rightarrow \text{AF} \text{ warm}) \]

or equivalently,

\[ f_{CTL} = \neg (\text{true} \ EU (\text{switch} \land \text{EG} \ \neg \text{warm})) \]

• Checking the property yields that the formula is not true – because the system may stay forever in left most state.

  The erroneous behavior happens when
  the user keeps turning the switch while the door is open.

• A “reasonable” user should eventually realize that
  turning the switch when the door is open does no good. So,
  consider only executions that do not stay forever in this state.

  not expressible in CTL but in CTL fair ....
CTL with fairness

- Formally, we consider the problem where we are given $K$ and $\phi$ as before and additionally a fairness constraint $F \subseteq S$.

- We call a run fair (w.r.t. $F$) iff it contains infinitely many states from $F$.

- The problem is to compute $S_K(\phi)$ for the case where the operators $EG$ and $EU$ consider only fair runs (w.r.t. $F$).

- Therefore, we introduce the following modified operators:

  $$S_K(EG_f \phi) = \{ s \mid \exists \text{ a fair run } \rho \text{ of } K \text{ s.t.}$$
  $$\rho(0) = s \text{ and } \forall i \geq 0, \rho(i) \in S_K(\phi) \}$$

  $$S_K(\phi_1 EU_f \phi_2) = \{ s \mid \exists \text{ a fair run } \rho \text{ of } K \text{ s.t.}$$
  $$\rho(0) = s \text{ and } \exists i \text{ such that } \rho(i) \in S_K(\phi_2) \text{ and } \forall k < i, \rho(k) \in S_K(\phi_1) \}$$
CTL with fairness

• First, observe that fair runs have the following properties:

  1. $\rho$ is a fair run iff $\rho^i$ is fair for all $i \geq 0$.

  2. $\rho$ is a fair run iff $\rho$ has a fair suffix $\rho^i$ for some $i$.

• Using this, we can rewrite the $EU_f$ operator as follows:

  $$\phi_1 EU_f \phi_2 \equiv \phi_1 EU (\phi_2 \land EG_f true)$$

• Thus, it is enough to provide a new algorithm for $EG_f$
Strongly connected components

• First, we introduce the following definitions:

• A set of states $C \subseteq S$ is called a strongly-connected component (SCC) of $K$ iff

  for all $q, q' \subseteq C$, we have $q \rightarrow^* q'$

• An SCC $C$ is trivial iff

  $C = \{q\}$ for some $q \in S$ and $q \not\rightarrow q$.
The $EG_f$ operator

Case 6: $\alpha \equiv EG_f p$ (with $p \in AP$)

1. Compute the restriction $K_p$ of $K$ to the states in $S_K(p)$.
2. Compute the SCCs of $K_p$ (applying the algorithm of Tarjan).
3. Identify the non-trivial SCCs that contain a state from $F$.
4. $S_K(EG_f p)$ consists of states from which one of these SCCs is reachable.
Every state $s$ satisfying $\text{EG}_t p$ must reach a SCC which globally satisfies $p$ within which at least one state belongs to $F$. 
Take the fairness constraint $F$ as $S \setminus \{s_1\}$

... prevent to infinitely visit $s_1$.

$$\text{AG (switch } \Rightarrow \text{ AF}_f \text{ warm) holds}$$
**CTL Complexity vs LTL**

**CTL**  
Building of the state space

\[
O(|S|.|f|) \text{ in time} \\
\text{et } O(|f|.\log^2|S|) \text{ in space}
\]

**LTL**  
Building of a synchronized product

\[
\text{from the formula} \\
O(|S|.2^{O(|f|)}) \text{ in time} \\
\text{et } O((\log |S|+|f|)^2) \text{ in space}
\]

\[
\text{from the automaton} \\
O(|S|.|f|) \text{ in time} \\
\text{et } O(\log^2 (|A_f|.|S|)) \text{ in space}
\]

translation : space exponential order
Conclusion on CTL

- CTL (Branching time) can specify safety properties and some liveness properties.
- CTL can be efficiently implemented (linear complexity w.r.t. to the Kripke structure), provided a good management of sets of states.
- Fairness properties leads us to improve the capability of CTL model checkers (SCC searches are needed).
- CTL fair model checkers can be used to verify CTL and actually also LTL formula.

- CTL does not provide a counter example when the property does not hold. The output is the set of states that satisfy the formula (maybe huge).
- CTL model checkers cannot answer before labeling the initial state with the truth value of the formula.