Chapter 3

Temporal logics

3.1 Branching time temporal logics

In many applications of temporal logics in computer science the temporal logic models are based on transition systems, best understood as arbitrary Kripke structures, instead of linearly ordered structures like in the linear temporal logic.

An arbitrary Kripke model naturally represents not just one linear ordering of worlds, but a tree, representing a high number of linear orderings corresponding to the paths in the tree. In this setting it is natural to extend the linear logic language further to be able to express properties of such trees. This leads to branching time temporal logics, of which we discuss the computation tree logic CTL\textsuperscript{*}, and its sublogics LTL and CTL [Emerson, 1990].

3.1.1 The language of the computation tree logic CTL\textsuperscript{*}

1. Every atomic proposition $p \in P$ is a formula.

2. If $\varphi$ and $\psi$ are formulae, then so are $\neg \varphi$, $\varphi \lor \psi$, $\varphi \land \psi$, $\varphi \rightarrow \psi$, $F\varphi$, $G\varphi$, $X\varphi$, $\varphi U\psi$, $A\varphi$, $E\varphi$.

Figure 3.1: A transition system (a Kripke model) and a corresponding computation tree.
The semantics of the operators $F$, $G$, $X$ and $U$ is known from the linear time temporal logics. The path quantifiers quantify over all of the possibly infinite number of computation paths in a transition system. Formulae $E\varphi$ with existential path quantification say that $\varphi$ must be true on at least one computation path, and formulae $A\varphi$ say that $\varphi$ must be true on all computation paths through the current state.

Example 3.1 CTL* formulae with path quantification are useful in expressing many correctness properties programs, controllers or other systems might have to fulfill.

- $AG\neg$-failure: A failure state is never reached.
- $AG(EF\text{restart})$: From every state of the system it is possible to reach the restart state. That is, the system cannot get stuck anywhere.
- $AG(\text{request} \rightarrow AF\text{ack})$: Any request will be eventually acknowledged.

3.1.2 The semantics of computation tree logic CTL*

For given sets $P$ of atomic propositions a model in the computation tree logic is a tuple $\mathcal{M} = \langle S, R, V \rangle$ where $S$ is a set of states (possible worlds), $R$ is an accessibility relation so that for every $s \in S$ there is $t \in S$ such that $sRt$, and $V : S \times P \to \{0, 1\}$ is a valuation that assigns truth values to atomic propositions in every state. Analogously to modal logics defined earlier we can talk about the frame $\mathcal{F} = \langle S, R \rangle$.

The truth of CTL* formulae are in general defined over a computation path, an infinite sequence of states in a model. However, for state formulae that only have $E$ and $A$ as their outermost modal operators the truth can also be defined with respect to a state.

Definition 3.2 A computation path is an infinite sequence of states $\lambda = s_0s_1s_2\ldots$ such that $s_iRs_{i+1}$ for each $i \geq 0$. For a path $\lambda = s_0s_1s_2\ldots$, $\lambda[i]$ denotes the state $s_i$. For a path $\lambda$, $\lambda^i$ denotes the $i$th suffix of a path $\lambda$, that is, $\lambda^i$ is defined by $\lambda^i[j] := \lambda[i + j]$ for all $j \geq 0$.

We often write briefly “path” instead of “computation path”.

The truth-definition of formulae in states $s \in S$ or over computation paths $\lambda$ is as follows.

1. $\mathcal{M} \models_s p$ iff $V(s, p) = 1$
2. $\mathcal{M} \models_s \neg\varphi$ iff $\mathcal{M} \not\models_s \varphi$
3. $\mathcal{M} \models_s \varphi \lor \psi$ iff $\mathcal{M} \models_s \varphi$ or $\mathcal{M} \models_s \psi$
4. $\mathcal{M} \models_s A\varphi$ iff $\mathcal{M} \models_{\lambda} \varphi$ for every path $\lambda$ such that $\lambda[0] = s$
5. $\mathcal{M} \models_s E\varphi$ iff $\mathcal{M} \models_{\lambda} \varphi$ for some path $\lambda$ such that $\lambda[0] = s$
6. $\mathcal{M} \models_{\lambda} \varphi$ iff $\mathcal{M} \models_{\lambda[0]} \varphi$ for state formulae $\varphi$
7. $\mathcal{M} \models_{\lambda} \neg\varphi$ iff $\mathcal{M} \not\models_{\lambda} \varphi$
8. $M \models \varphi \lor \psi \iff M \models \varphi$ or $M \models \varphi$

9. $M \models \lambda G \varphi \iff M \models \lambda \varphi$ for every $i \geq 0$

10. $M \models \lambda F \varphi \iff M \models \lambda \varphi$ for some $i \geq 0$

11. $M \models \lambda X \varphi \iff M \models \lambda \varphi$

12. $M \models \lambda \varphi U \psi \iff$ there is $i \geq 0$ such that $M \models \lambda \psi$ and $M \models \lambda \varphi$ for every $j \in \{0, \ldots, i - 1\}$.

### 3.2 Linear temporal logic LTL

The language of the linear time temporal logic LTL is the language of CTL* without the path quantifiers $A$ and $E$. This means that LTL formulae are interpreted over individual paths, and every formula is a path formula. We can define the truth of a LTL formula in a model $M \models_s \varphi$ either existentially as $M \models_s E \varphi$ or universally as $M \models_s A \varphi$.

### 3.3 Computation tree logic CTL

CTL is a simpler variant of CTL* that does not allow arbitrary combination of temporal operators. Specifically, every occurrence of operators $X$, $F$, $G$ and $U$ is preceded by the path quantifier $E$ or $A$. Essentially, we have 8 modal operators $AX$ $AF$ $AG$ $AU$ $EX$ $EF$ $EG$ $EU$. Hence every formula is a state formula and can be evaluated with respect to a state without referring to a designated computation path. This restriction on modal operators reduces the class of properties expressible in CTL.

### 3.4 Relations between CTL, LTL and CTL*

Every LTL formula and every CTL formulae has an equivalent CTL* formula, but LTL and CTL are not comparable in expressivity, that is, there are formulae in both of these logics that express properties of computation trees not expressible in the other logic.

For the LTL formula $F(p \land Xp)$ (interpreted universally with implicit prefix $A$) there is no equivalent CTL formula.

For the CTL formula $AG(EP)$ there is no equivalent LTL formula.

The conjunction $AF(p \land Xp) \land AG(EP)$ of the above two formulae is a CTL* formula that has no counterpart in LTL nor in CTL. Hence CTL* is strictly more expressive than either of the these two logics.
3.5 Computation tree logic CTL axiomatically

CTL is the smallest logic that includes all propositional tautologies, is closed under modus ponens and substitution, and includes the following axiom schemata.

1. $EF\varphi \leftrightarrow E(\top \mathbin{U} \varphi)$
2. $AG\varphi \leftrightarrow \neg EF \neg \varphi$
3. $AG\varphi \leftrightarrow A(\top \mathbin{U} \varphi)$
4. $EG\varphi \leftrightarrow \neg AF \neg \varphi$
5. $EX(\varphi \lor \psi) \leftrightarrow (EX \varphi \lor EX \psi)$
6. $AX\varphi \leftrightarrow \neg EX \neg \varphi$
7. $E(\varphi \mathbin{U} \psi) \leftrightarrow (\psi \lor (\varphi \land EX E(\varphi \mathbin{U} \psi)))$
8. $A(\varphi \mathbin{U} \psi) \leftrightarrow (\psi \lor (\varphi \land AX A(\varphi \mathbin{U} \psi)))$
9. $EX \top$
10. $AX \top$
11. $AG(\sigma \rightarrow (\neg \varphi \land EX \sigma) \rightarrow (\sigma \rightarrow \neg A(\varphi \mathbin{U} \psi)))$
12. $AG(\sigma \rightarrow (\neg \varphi \land EX \sigma) \rightarrow (\sigma \rightarrow \neg AF \psi))$
13. $AG(\sigma \rightarrow (\neg \varphi \land AX \sigma) \rightarrow (\sigma \rightarrow \neg EF \psi))$
14. $AG(\sigma \rightarrow (\varphi \rightarrow AX \sigma) \rightarrow (\sigma \rightarrow \neg E(\varphi \mathbin{U} \psi)))$
15. $AG(\varphi \rightarrow (EX \varphi \rightarrow EX \psi))$

Additionally, the following inference rule is needed.

If $\vdash_{CTL} \varphi$, then $\vdash_{CTL} GA\varphi$.

**Theorem 3.3** CTL is sound and complete with respect to the Kripke semantics for CTL∗.

3.6 Model-checking for temporal logic formulae

Temporal logics have found important applications as a specification language in computer-aided verification [Clarke et al., 1999]. Early work on theoretically well-founded approaches to verification focused on deductive verification, in which the transition systems (programs, protocols, controllers) to be verified are translated to a temporal logic formula $\Sigma$, and then the transition system is shown to satisfy a specification/property $\varphi$ by showing that $\Sigma \vdash \varphi$. This type of deductive approach to verification is restricted by the efficiency of theorem-proving procedures (and mathematicians, logicians and computer scientists proving correctness manually) to small transition
systems only, because the formulae Σ become big, and in general the runtimes of theorem-proving algorithms grow exponentially as the formulae Σ grow. The satisfiability problems of LTL, CTL and CTL* are respectively PSPACE-complete, EXPTIME-complete and 2-EXP-complete [Emerson, 1990].

An alternative and more practical approach to verification was proposed in the 1980’s [Quelle and Sifakis, 1982; Clarke et al., 1986]. Instead of representing the whole problem as a temporal logic formula, just represent the property/specification as a formula, and view the transition system as a Kripke model .M. Now the verification problem becomes a model-checking problem of testing whether .M |=s ϕ. In many cases this problem is much easier than the validity and satisfiability problems of the specification logic.

The temporal logics CTL*, CTL and LTL are the most important logics used in verification. In this section we present algorithms for the respective model-checking problems and discuss their computational complexity.

3.6.1 A model-checking algorithm for CTL

Of the three logics CTL has the simplest model-checking algorithms. This stems from the fact that the linear temporal operators X, F, G and U never appear without a preceding path quantifier A or E. Hence all subformulae of the specification ϕ can be evaluated in a state, and not along a computation path, which makes it unnecessary to consider any computation paths explicitly. This makes the algorithms rather simple in comparison to algorithms for LTL and CTL* model-checking, as will become obvious later.

Next we give a basic model-checking algorithm for CTL. The idea of the algorithm is to label the states in a Kripke-model with those subformulae of the formula ϕ to be model-checked that are true in that state. Because computation paths need not be considered, this labeling of states suffices for determining the truth of ϕ. The only small complication is caused by subformulae with universal quantification over all future states. If such a subformula EGψ or AGψ is true, then ψ must be true in an infinite number of future states.

For simplifying the presentation, we note that we need to consider only the operators EX, EG and EU because other operators can be reduced to these by the following equivalences.

1. EFϕ ↔ E(⊤Uϕ)
2. AFϕ ↔ ¬EG¬ϕ
3. AGϕ ↔ ¬E(⊤U¬ϕ)
4. AXϕ ↔ ¬EX¬ϕ
5. A(ϕUψ) ↔ ¬E(¬ψU(¬ϕ ∧ ¬ψ)) ∧ ¬EG¬ψ

Similarly, we assume that the formulas only contain the Boolean connectives ¬ and ∨.

Now, our objective is to determine whether .M |=s ϕ for a given model .M = ⟨S, R, V⟩, given state s ∈ S, and a given CTL formula ϕ.
procedure \( \text{EU}(\alpha, \beta) \)

\[
T := \{ s \in S | \beta \in \text{label}(s) \};
\]

for each \( s \in T \) do

\[
\text{label}(s) := \text{label}(s) \cup \{ E(\alpha U \beta) \};
\]

while \( T \neq \emptyset \)

\[
take \text{any } s \in T;
\]

\[
T := T \setminus \{ s \};
\]

for each \( t \) such that \( t R s \)

if \( \alpha \in \text{label}(t) \) and \( E(\alpha U \beta) \notin \text{label}(t) \) then

\[
\text{label}(t) := \text{label}(t) \cup \{ E(\alpha U \beta) \};
\]

\[
T := T \cup \{ t \};
\]

end if

end for

end while

Figure 3.2: Algorithm for labeling states with formulae \( E(\alpha U \beta) \)

The algorithm labels the states \( s \in S \) with subformulae of \( \varphi \) that are true in \( s \). The labeling starts from atomic formulae and then continues inductively to longer and longer formulae until \( \varphi \). At this point the truth-values of \( \varphi \) and of all its subformulae are known in every state in the model.

Initialize \( \text{label}(s) = \emptyset \) for every state \( s \in S \).

Base case \( i = 1 \): For every state \( s \in S \) and every atomic proposition \( p \) add \( p \) to \( \text{label}(s) \) if \( s \models p \).

Inductive case \( i \geq 1 \): Let \( \psi \) be a subformula of \( \varphi \) of length \( i \) and \( s \in S \) one of the states. The proper subformulae of \( \psi \) are strictly shorter than \( \psi \), and therefore they have been handled earlier.

If \( \psi = \alpha \lor \beta \) for some \( \alpha \) and \( \beta \), then add \( \alpha \lor \beta \) to \( \text{label}(s) \) if \( \alpha \in \text{label}(s) \) or \( \beta \in \text{label}(s) \).

If \( \psi = \neg \alpha \), then add \( \neg \alpha \) to \( \text{label}(s) \) if \( \alpha \notin \text{label}(s) \).

If \( \psi = \text{EX} \alpha \), then add \( \text{EX} \alpha \) to \( \text{label}(s) \) if \( \alpha \in \text{label}(s) \) for some \( s' \) such that \( s R s' \).

The algorithm in Figure 3.2 handles the case \( \psi = E(\alpha U \beta) \). First, identify the set \( T \) of all those states in which the formula \( \beta \) is true. Then \( E(\alpha U \beta) \) is immediately true in every such \( s \in T \). Second, identify those states from which a state in \( T \) is reached with one step and in which \( \alpha \) is true. Also in all these states \( E(\alpha U \beta) \) is true. Repeatedly go backwards to further states in which \( \alpha \) is true, until no more such state can be found. Then we have identified all states in which \( E(\alpha U \beta) \) is true.

The algorithm in Figure 3.3 handles the case \( \psi = E G \alpha \). It is based on identifying the strongly components of a graph that is obtained from \( (S, R) \) by restricting \( S \) to the set \( S' \) of those states in which \( \alpha \) is true. The SCCs of a graph can be computed in linear time by Tarjan’s algorithm [Tarjan, 1972]. Any SCC of \( (S', R \cap (S' \times S')) \) consisting of one element with an arc to itself or more than one element contains an infinite path of states on which all \( \alpha \) is true. Hence \( E G \alpha \) is true in all such states. Further, like in the \( \psi = E(\alpha U \beta) \) case, further states in which \( E G \alpha \) is true can be found by going from these states backwards along arcs in \( R \) to states in which \( \alpha \) is true.

The data structures in the algorithm can be implemented so that its runtime is polynomial in the number of states in \( S \), in the number of arcs in \( R \), and in the size of the formula \( \varphi \) to be model-
procedure \( \text{EG}(\alpha) \)
\[
S' := \{ s \in S | \alpha \in \text{label}(s) \};
\]
\[
SCC := \{ C | C \text{ is a SCC of } (S', R \cap (S' \times S')), |C| \geq 1 \text{ or there is } s \in C \text{ with } sRs \};
\]
\[
T := \{ s \in C | C \in SCC \};
\]
for each \( s \in T \) do
\[
\text{label}(s) := \text{label}(s) \cup \{ \text{EG} \alpha \};
\]
while \( T \neq \emptyset \) do
\[
\text{take any } s \in T;
\]
\[
T := T \setminus \{ s \};
\]
for each \( t \in S' \) such that \( tRs \) do
\[
\text{if } \text{EG} \alpha \notin \text{label}(t) \text{ then}
\]
\[
\text{label}(t) := \text{label}(t) \cup \{ \text{EG} \alpha \};
\]
\[
T := T \cup \{ t \};
\]
end if
end for
end while

Figure 3.3: Algorithm for labeling states with formulae \( \text{EG} \alpha \)

checked. More precisely, the algorithm runs in \( O(|\varphi| \times (|S| + |T|)) \) time.

**Example 3.4**

### 3.7 A model-checking algorithm for LTL

The model-checking algorithms for the temporal logics LTL and CTL* with pure path formulae not directly prefixed with path quantifiers like in CTL are conceptually more complicated, and also computationally much more expensive than the algorithm we gave for CTL.

The reason for this additional complexity is that we cannot talk about the truth of formulae in a state without making a reference to a particular path in the transition system. This causes a problem because the number of different paths starting in a given state is often infinite. The number of finite paths of length \( n \) can be exponential in \( n \). At a first look this might seem to imply that we need to consider quantification over an infinite number of paths, but fortunately this turns out not to be the case. It will suffice to consider only an exponential number of different kinds of paths starting from the given state, without uniquely representing these paths in detail.

Consider a formula \( \varphi \) to be model-checked. For every state of the transition system, we need only consider all the possible values the subformulae of \( \varphi \), and formulae closely related to these subformulae, may have. For example, when we are interested in the truth of a subformula \( \psi U \chi \) in a state \( s \), we are interested in the truth of \( \psi \) and \( \chi \) in \( s \), and the truth of \( \psi \cup \chi \) in the successor states of \( s \), but it does not matter why \( \psi \cup \chi \) is true in the successor states, that is, whether \( \psi \) and \( \chi \) are true there or not.

There is a simple construction that uses the above ideas to provide a relatively efficient model-checking algorithm for LTL. When testing \( M \models_s \varphi \) by the algorithm, the runtime is exponential on the size of \( \varphi \) (because of the possibly exponential number of different kinds of paths that have
to be considered), but only polynomial in the size of $\mathcal{M}$. Because the formulae expressing useful properties of transition systems are often small, the exponentiality in the size of the formula is usually not a problem usually, and hence the algorithm is applicable to very big transition systems.

Consider LTL formulae with only the temporal operators $X$ and $U$. The operators $F$ and $G$ are reducible to these operators by the following equivalences.

1. $F\varphi \leftrightarrow (\top U \varphi)$
2. $G\varphi \leftrightarrow \neg (\top U \neg \varphi)$

Only the subformulae of the formulae $\varphi$ to be model-checked and a small number of formulae closely related to them need to be considered by the model-checking algorithm. This is the set $\text{CL}(\varphi)$.

**Definition 3.5** The closure $\text{CL}(\varphi)$ of a formula $\varphi$ is the smallest set of formulae such that

1. $\varphi \in \text{CL}(\varphi)$,
2. $\top \in \text{CL}(\varphi)$,
3. $\psi \in \text{CL}(\varphi)$ iff $\neg \psi \in \text{CL}(\varphi)$,
4. if $\psi_1 \lor \psi_2 \in \text{CL}(\varphi)$ then $\psi_1 \in \text{CL}(\varphi)$ and $\psi_2 \in \text{CL}(\varphi)$,
5. if $X \psi \in \text{CL}(\varphi)$ then $\psi \in \text{CL}(\varphi)$,
6. if $\neg X \psi \in \text{CL}(\varphi)$ then $X \neg \psi \in \text{CL}(\varphi)$,
7. if $\psi_1 U \psi_2 \in \text{CL}(\varphi)$ then $\{\psi_1, \psi_2, X(\psi_1 U \psi_2)\} \subseteq \text{CL}(\varphi)$.

Above we identify $\neg \neg \psi$ and $\psi$ so that the third rule does not cause the size of the set to become infinite.

The number of formulae in $\text{CL}(\varphi)$ is linear in the size of $\varphi$: in addition to subformulae of $\varphi$ and their negations it contains at most one additional formula for each subformula of form $\psi_1 U \psi_2$ and $\neg X \psi$.

Next we can proceed with the construction of the graph structure that represents all the different kinds of infinite paths in the Kripke model $\mathcal{M}$. This graph is obtained from $\mathcal{M}$ by taking all of its states and making several copies of them, corresponding to all the possible executions they could be a part of.

**Definition 3.6** Let $\mathcal{M} = \langle S, R, V \rangle$ be a Kripke model on which we are testing $\mathcal{M} \models_s \varphi$ for $s \in S$. Then the tableau\(^1\) of $\mathcal{M}$ is the graph $\langle S', R' \rangle$ where $S'$ consists of all pairs $\langle s, K \rangle$ such that $s \in S$ and

\(^1\)This is not related to the proof procedures with analytic/semantic tableaux, and we use the word “tableau” for historical reasons.
CHAPTER 3. TEMPORAL LOGICS

Figure 3.4: A transition system

1. $K \subseteq \text{CL}(\varphi) \cup \mathcal{P}$,
2. $p \in K$ iff $V(s, p) = 1$, for every $p \in \mathcal{P}$,
3. $\psi \in K$ iff $\neg \psi \notin K$, for every $\psi \in \text{CL}(\varphi)$,
4. $\psi_1 \lor \psi_2 \in K$ iff $\psi_1 \in K$ or $\psi_2 \in K$, for every $\psi_1 \lor \psi_2 \in \text{CL}(\varphi)$,
5. $\neg X \psi \in K$ iff $X \neg \psi \in K$, for every $\neg X \psi \in \text{CL}(\varphi)$,
6. $\psi_1 U \psi_2 \in K$ iff $\psi_2 \in K$ or $\{\psi_1, X(\psi_1 U \psi_2)\} \subseteq K$, for every $\psi_1 U \psi_2 \in \text{CL}(\varphi)$.

and $\langle s, K \rangle R' \langle s', K' \rangle$ whenever

1. $s Rs'$
2. for every $X \psi \in \text{CL}(\varphi)$, $X \psi \in K$ iff $\psi \in K'$.

Essentially, we have taken a model $\mathcal{M}$ and made all the different kinds of paths that could go through a state $s$ in $\mathcal{M}$ explicit in the elements $\langle s, K \rangle$ of the graph $\langle S', R' \rangle$. The formulae in $K$ describe the relevant properties of one particular class of paths through the state $s$, and the number of these classes is finite even though the number of all paths through $s$ may be infinite.

Example 3.7 Consider the transition system (Kripke model) $\langle S, R, V \rangle$ in Figure 3.4. We will model-check $\varphi = E(\neg p U (p \land q))$.

The unnegated formulae in $\text{CL}(\varphi)$ are

- $\top$
- $p$
- $q$
- $p \land q$
- $\neg p U (p \land q)$
- $X (\neg p U (p \land q))$

Of these, only $p$, $q$, $\alpha = \neg p U (p \land q)$ and $X \alpha$, are not truth-functionally determined by simpler formulae.

The potential nodes in the tableau $\langle S', R' \rangle$ therefore correspond to sets of formulae obtained by negating some of $\{p, q, \alpha, X \alpha\}$. Not all of these sets however occur in a node in the graph: for
example, we cannot have \( \{ p, q, \neg \alpha \} \subseteq K \) for any \( K \), nor \( \{ \neg p, \neg \alpha, X\alpha \} \subseteq K \). The nodes in \( \langle S', R' \rangle \) correspond to the following sets.

\[
\begin{align*}
\{ p, q, \alpha, X\alpha \} & \subseteq K_1 \\
\{ p, q, \alpha, \neg X\alpha \} & \subseteq K_2 \\
\{ \neg p, q, \alpha, X\alpha \} & \subseteq K_3 \\
\{ \neg p, q, \neg \alpha, \neg X\alpha \} & \subseteq K_4 \\
\{ \neg p, q, \neg \alpha, X\alpha \} & \subseteq K_5 \\
\{ \neg p, \neg q, \neg \alpha, \neg X\alpha \} & \subseteq K_6
\end{align*}
\]

The nodes \( S' \) in the graph \( \langle S', R' \rangle \) are therefore

\[
S' = \{ \langle s_1, K_1 \rangle, \langle s_1, K_2 \rangle, \langle s_2, K_3 \rangle, \langle s_2, K_4 \rangle, \langle s_3, K_5 \rangle, \langle s_3, K_6 \rangle \}
\]

and the arcs described by \( R' \) are

\[
\begin{align*}
\langle s_1, K_1 \rangle R' & \langle s_2, K_3 \rangle \\
\langle s_1, K_2 \rangle R' & \langle s_2, K_4 \rangle \\
\langle s_3, K_5 \rangle R' & \langle s_1, K_1 \rangle \\
\langle s_3, K_5 \rangle R' & \langle s_2, K_2 \rangle \\
\langle s_2, K_3 \rangle R' & \langle s_3, K_5 \rangle \\
\langle s_2, K_4 \rangle R' & \langle s_3, K_6 \rangle \\
\langle s_3, K_5 \rangle R' & \langle s_2, K_3 \rangle \\
\langle s_3, K_6 \rangle R' & \langle s_2, K_4 \rangle
\end{align*}
\]

The graph \( \langle S', R' \rangle \) is depicted in Figure 3.5.

\[\blacksquare\]

Notice that the sets \( K \) are maximally consistent subsets of \( CL(\varphi) \cup \mathcal{P} \cup \{ \neg p | p \in \mathcal{P} \} \). For each \( \langle s, K \rangle \in S' \) we have \( \mathcal{M} \models K \cap \mathcal{P} \), that is, the atomic propositions true in \( s \) are contained in \( K \).
We also have for any formula $\psi \in \text{CL}(\varphi)$ with occurrences of the temporal operator $X$ only: $\mathcal{M} \models \lambda \psi$ for some path $\lambda$ such that $\lambda[0] = s$ if and only if $\psi \in K$ and there is a path starting from $\langle s, K \rangle$ having length at least the depth of nesting of $X$ in $\psi$.

Our goal is to have this generalized to formulae in general, that is, $\mathcal{M} \models \lambda K$ for some path $\lambda$ starting from $s$. What remains to be done is to have this for formulae with occurrences of $U$.

For formulae $\psi_1 U \psi_2$ we need to guarantee that eventually a state in which $\psi_2$ is true is reached. Put differently, we have identify infinite paths on which $\psi_1$ is always true and $\psi_1 U \psi_2$ the formula is false because $\psi_2$ will never be true. For this purpose we introduce the notion of eventuality sequences.

**Definition 3.8** An eventuality sequence $\pi$ is an infinite path in $\langle S', R' \rangle$ such that if $\psi_1 U \psi_2 \in K$ for some $\langle s, K \rangle$ on $\pi$, then there is $\langle s', K' \rangle$ on $\pi$ not before $\langle s, K \rangle$ such that $\psi_2 \in K'$.

**Lemma 3.9** Let $\mathcal{M}$ be a Kripke model, $\varphi$ a formula, and $\langle S', R' \rangle$ the tableau for $\mathcal{M}$ and $\varphi$. Then $\mathcal{M} \models s \ E \varphi$ if and only if there is an eventuality sequence starting at $\langle s, K \rangle$ such that $\varphi \in K$.

**Proof:** So assume there is an eventuality sequence $\langle s_0, K_0 \rangle, \langle s_1, K_1 \rangle, \ldots$ starting at $\langle s, K \rangle = \langle s_0, K_0 \rangle$ such that $\varphi \in K$. Let $\lambda = s_0, s_1, \ldots$, that is, a computation path in $\mathcal{M}$.

We show that for every $\psi \in \text{CL}(\varphi), \mathcal{M} \models \lambda \psi$ iff $\psi \in K_i$.

The proof is by induction on the structure of the formulae in $\text{CL}(\varphi)$.

Base case 1, $\psi = p$ for some $p \in \mathcal{P}$:

By definition of $\langle s_i, K_i \rangle$, $V(s_i, p) = 1$ iff $p \in K_i$.

Inductive case 1, $\psi = \neg \psi'$:

By the induction hypothesis $\mathcal{M} \models \lambda \psi'$ iff $\psi' \in K_i$. By definition of $\langle s_i, K_i \rangle$, now $\psi' \in K_i$ iff $\neg \psi' \notin K_i$. Hence $\mathcal{M} \models \lambda \neg \psi'$ iff $\neg \psi' \in K_i$.

Inductive case 2, $\psi = \psi_1 \lor \psi_2$:

By the induction hypothesis $\mathcal{M} \models \lambda \psi_1$ iff $\psi_1 \in K_i$. For $j \in \{1, 2\}$, $\psi' \in K_i$ iff $\psi_1 \in K_i$ or $\psi_2 \in K_i$. Hence $\psi \in K_i$ iff $\mathcal{M} \models \lambda \psi$.

Inductive case 3, $\psi = X \psi'$:

By the definition of $R', \langle s_i, K_i \rangle R' \langle s_{i+1}, K_{i+1} \rangle$ holds because $X \psi' \in K_i$ iff $\psi' \in K_{i+1}$. By the induction hypothesis $\psi' \in K_{i+1}$ iff $\mathcal{M} \models \lambda \psi'$. By truth-definition of $X$, $\mathcal{M} \models \lambda \psi'$. If $\mathcal{M} \models \lambda X \psi'$, hence $\mathcal{M} \models \lambda \psi'$.

Inductive case 4, $\psi = \psi_1 U \psi_2$:

Assume $\psi_1 U \psi_2 \in K_i$, either $\psi_2 \in K_i$ or $\{\psi_1, X(\psi_1 U \psi_2)\} \subseteq K_i$. If $\psi_2 \in K_i$ then by the induction hypothesis $\mathcal{M} \models \lambda \psi_2$ and hence $\mathcal{M} \models \lambda \psi_1 \psi_2$. Otherwise $\mathcal{M} \models \lambda \psi_1$ by the induction hypothesis and $\psi_1 U \psi_2 \in K_{i+1}$ by the definition of $R'$. By definition of eventuality sequences there is some $\langle s_j, K_j \rangle$ with $j \geq i$ such that $\psi_2 \in K_j$. Let $j$ be the minimal such time point. By the induction hypothesis $\mathcal{M} \models \lambda \psi_2$. An induction proof now shows that $\mathcal{M} \models \lambda \psi_1$ for all $k \in \{i, \ldots, j - 1\}$, and hence $\mathcal{M} \models \lambda \psi_1 U \psi_2$. 


Assume $\mathcal{M} \models_{\chi_i} \psi_1 U \psi_2$. Therefore $\mathcal{M} \models_{\chi_j} \psi_2$ for some $j \geq i$ and $\mathcal{M} \models_{\chi_k} \psi_1$ for all $k \in \{i, \ldots, j - 1\}$. By the induction hypothesis $\psi_1 \in K_k$ for all $k \in \{i, \ldots, j - 1\}$ and $\psi_2 \in K_j$.

Assume $\psi_1 U \psi_2 \notin K_i$. Because $\psi_1 \in K_i$, now $X\psi_1 U \psi_2 \notin K_i$, and hence $X \neg (\psi_1 U \psi_2) \in K_i$, and hence $\neg (\psi_1 U \psi_2) \in K_{i+1}$. Going inductively further we eventually get $\psi_1 U \psi_2 \notin K_j$, which contradicts $\psi_2 \in K_j$. Therefore it must be that $\psi_1 U \psi_2 \in K_j$.

This finishes the inductive case 4 and the proof of the if direction of the equivalence. The only if direction follows. So assume $\mathcal{M} \models_{s} E \varphi$. Hence there is path $\lambda = s_0, s_1, s_2, \ldots$ starting from $s = s_0$ such that $\mathcal{M} \models_{\lambda} E \varphi$.

Define $K_i = \{\psi \in \text{CL}(\varphi) | \mathcal{M} \models_{\chi_i} \psi\}$ for all $i \geq 0$. It is easy to verify that $\langle s_0, K_0 \rangle, \langle s_1, K_1 \rangle, \ldots$ is an eventuality sequence starting from $\langle s_0, K_0 \rangle$. Further, by definition of CL($\varphi$), $\varphi \in K_0$. This concludes the proof. □

Now the algorithmic problem that remains is to find eventuality sequences. Like in the CTL model-checking algorithm, also in this case the strong components are the means of identifying infinite paths having some property.

**Definition 3.10** Let $\mathcal{M}$ be a Kripke model, $\varphi$ a formula, and $\langle S', R' \rangle$ the tableau for $\mathcal{M}$ and $\varphi$.

Then a non-trivial (either $|C| > 1$ or there is $s \in C$ such that $s R' s$) strong component $C \subseteq S'$ of $\langle S', R' \rangle$ is self-fulfilling iff for every $\langle s, K \rangle \in S'$ and $\psi_1 U \psi_2 \in K$ there is $\langle s', K' \rangle \in C$ such that $\psi_2 \in K'$.

A self-fulfilling strong component in the tableau yields infinite execution paths along which until formulae are satisfied.

**Lemma 3.11** Let $\mathcal{M}$ be a Kripke model, $\varphi$ a formula, and $\langle S', R' \rangle$ the tableau for $\mathcal{M}$ and $\varphi$.

There is an eventuality sequence starting at $\langle s, K \rangle \in S'$ if and only if there is a path in $\langle S', R' \rangle$ from $\langle s, K \rangle \in S'$ to a self-fulfilling strong component.

**Proof:** Assume there is an eventuality sequence $\sigma$ starting at $\langle s, K \rangle$. Let $C'$ be the set of all $a \in S'$ that occur infinitely many times in $\sigma$. This set is a subset of a strongly connected component $C$ of $\langle S', R' \rangle$. Hence there is a path from $\langle s, K \rangle$ to a strongly connected component. It remains to show that it is self-fulfilling.

Consider a subformula $\psi_1 U \psi_2$ of $\varphi$ such that $\psi_1 U \psi_2 \in K'$ for some $\langle s', K' \rangle \in C$. Now there is a finite path from $\langle s', K' \rangle$ to $C'$ because $C$ is a strongly connected component and $C' \subseteq C$. If $\psi_2 \in K''$ for some $\langle s'', K'' \rangle$ on that finite path, then $\psi_2 \in K'''$ for some $\langle s''', K''' \rangle \in C$.

If $\psi_2 \notin K''$ for any $\langle s'', K'' \rangle$ on that finite path, then $\psi_1 U \psi_2 \in K''$ for every $\langle s'', K'' \rangle$ on that finite path, and in particular, its last element which is in $C'$. Because $C'$ is part of an eventuality sequence, there is $\langle s'', K'' \rangle \in C'$ such that $\psi_2 \in K'''$. Hence in both cases there is $\psi_2$ in one of the nodes in $C$, and therefore $C$ is a self-fulfilling strong component.

Assume there is a path from $\langle s, K \rangle$ to a self-fulfilling strongly connected component $C$. 
For any \( \psi_1 U \psi_2 \) occurring in a node in \( C \) we can construct a path from that node to a node with \( \psi_2 \), because \( C \) is self-fulfilling. Hence we can construct an infinite cycle of nodes in \( C \) with the same property. To make this infinite cycle an eventuality sequence, we also have to show that the path from \( \langle s, K \rangle \) to \( C \) satisfies the required properties, that is, any \( \psi_1 U \psi_2 \) on that path is followed by \( \psi_2 \). By the definition of \( R' \), for every node on that path following \( \psi_1 U \psi_2 \) we either have \( \psi_2 \), or \( X(\psi_1 U \psi_2) \) and hence \( \psi_1 U \psi_2 \) in the following node. In the first case we have \( \psi_2 \) on the path, and in the second we have \( \psi_1 U \psi_2 \) in \( C \), and because \( C \) is self-fulfilling we have \( \psi_2 \) in \( C \). Hence the infinite path with the cycle is an eventuality sequence. \( \square \)

**Theorem 3.12** Let \( \mathcal{M} \) be a Kripke model, \( \varphi \) a formula, and \( \langle S', R' \rangle \) the tableau for \( \mathcal{M} \) and \( \varphi \).

Now \( \mathcal{M} \models E \varphi \) if and only if there is a path in \( \langle S', R' \rangle \) from \( \langle s, K \rangle \in S' \) (for some \( K \)) to a self-fulfilling strong component.

**Proof:** By the preceding two lemmata. \( \square \)

From the above result we can construct an algorithm for LTL model-checking that runs in \( O((|S| + |R|)2^{O(|\varphi|)}) \) time, which is polynomial in the size of the transition system but exponential in the size of the formula to be model-checked.

### 3.7.1 Fairness

Assume that we are verifying a specification of a computer system consisting of two processes \( P_1 \) and \( P_2 \) that perform their computational otherwise independently but communicate once in a while. Because there is only one CPU, there is a scheduler that decides when the computations proceed and for how long. When verifying the correctness of the system we may want not to model the functioning of the scheduler, and assume that the scheduler gives CPU to the both processes in a fair manner. This is the assumption of fairness that is used in connection with concurrent systems. A fair scheduler could for example produce computation \( P_1, P_2, P_1, P_2, P_1, \ldots \) or possibly \( P_1, P_2, P_2, P_1, P_2, P_2, P_1, P_2, \ldots \) if \( P_2 \) has a higher overhead, but definitely not \( P_1, P_1, P_1, \ldots \) ignoring \( P_2 \) completely.

When model-checking such systems the assumption that the scheduler is fair should be taken into account. If it is not, and the correct behavior of the system involves both \( P_1 \) and \( P_2 \), then it is not difficult to find behaviors \( E \varphi \) that violate the correctness properties \( \varphi \). So, we often want to make the fairness assumption when doing model-checking.

Fairness assumptions can be represented as sets of formulae \( \Phi \) with the requirement that for every \( \phi \in \Phi \), any computation path contains infinitely many states in which \( \phi \) is true.

In LTL handling fairness assumptions is easy. When model-checking \( E(F \varphi) \) (test whether there is a “bad” execution having undesirable property \( F \varphi \)), just include the fairness assumptions as \( E(\varphi \land GF \phi_1 \land \cdots \land GF \phi_n) \) where \( \Phi = \{\phi_1, \ldots, \phi_n\} \).

CTL cannot express fairness assumptions in the same way this is possible with LTL. In particular, adjoining the formulae \( GF \phi_i \) as further conjuncts is not possible because of restrictions on path
quantification. Instead, researchers have defined variants of CTL (Fair CTL) with extra machinery for expressing fairness.