Chapter 13
Temporal Logic: Formulas, Models, Tableaux

Temporal logic is a formal system for reasoning about time. Temporal logic has found extensive application in computer science, because the behavior of both hardware and software is a function of time. This section will follow the same approach that we used for other logics: we define the syntax of formulas and their interpretations and then describe the construction of semantic tableaux for deciding satisfiability.

Unlike propositional and first-order logics whose variants have little theoretical or practical significance, there are many temporal logics that are quite different from each other. A survey of this flexibility is presented in Sect. 13.3, but you can skim it and go directly to Sect. 13.4 that presents the logic we focus on: *linear temporal logic*.

13.1 Introduction

Example 13.1 Here are some examples of specifications that use temporal concepts (italicized):

- *After* the reset-line of a flip-flop is asserted, the zero-line is asserted. The output lines *maintain* their values *until* the set-line is asserted; *then* they are complemented.
- If a request is made to print a file, *eventually* the file will be printed.
- The operating system will *never* deadlock.

The temporal aspects of these specification can be expressed in first-order logic using quantified variables for points in time:

DOI 10.1007/978-1-4471-4129-7_13, © Springer-Verlag London 2012

$$\forall t_1 (reset(t_1) \rightarrow \exists t_2 (t_2 \geq t_1 \land zero(t_2))),$$

$$\begin{aligned}&\forall t_1 \exists n (output(t_1) = n \land \\ &\qquad \exists t_2 (t_2 \geq t_1 \land set(t_2) \land output(t_2 + 1) = 1 - n \land \\ &\qquad\qquad \forall t_3 (t_1 \leq t_3 < t_2 \rightarrow output(t_3) = n))),\end{aligned}$$

$$\forall t_1 (RequestPrint(t_1) \rightarrow \exists t_2 (t_2 \geq t_1 \land PrintedAt(t_2))),$$

$$\forall t \neg deadlocked(t).$$

■

The use of explicit variables for points of time is awkward, especially since the specifications do not actually refer to concrete values of time. 'Eventually' simply means at any later time; the specification does not require that the file be printed within one minute or ten minutes. Temporal logic introduces new operators that enable abstract temporal relations like 'eventually' to be expressed directly within the logic.

Temporal logics are related to formal systems called *modal logics*. Modal logics express the distinction between what is *necessarily* true and what is *possibly* true. For example, the statement *'7 is a prime number'* is necessarily true because—given the definitions of the concepts in the statement—the statement is true always and everywhere. In contrast, the statement *the head of state of this country is a king* is possibly true, because its truth changes from place to place and from time to time. Temporal logic and modal logic are related because 'always' is similar to 'necessarily' and 'eventually' to 'possibly'.

Although temporal and modal logics first appeared in Greek philosophy, their vague concepts proved difficult to formalize and an acceptable formal semantics for modal logic was first given by Saul Kripke in 1959. In 1977, Amir Pnueli showed that temporal logic can specify properties of concurrent programs and that Kripke's semantics could be adapted to develop a formal theory of the temporal logic of programs. In this chapter and the next one we present the theory of linear temporal logic. Chapter 16 shows how the logic can be used for the specification of correctness properties of concurrent programs and for the verification of these properties. In that chapter, we will describe another temporal logic called computational tree logic that is also widely used in computer science.

13.2 Syntax and Semantics

13.2.1 Syntax

The initial presentation of the syntax and semantics of temporal logic will follow that used for general modal logics. We do this so that the presentation will be useful for readers who have a broader interest in modal logic and so that temporal logic can be seen within this wider context. Later, we specialize the presentation to a specific temporal logic that is used for the specification and verification of programs.

Definition 13.2 The syntax of *propositional temporal logic (PTL)* is defined like the syntax of propositional logic (Definition 2.1), except for the addition of two additional unary operators:

- $\Box$, read *always*,
- $\Diamond$, read *eventually*. ■

The discussion of syntax in Sect. 2.1 is extended appropriately: formulas of PTL are trees so they are unambiguous and various conventions are used to write the formulas as linear text. In particular, the two unary temporal logic operators have the same precedence as negation.

Example 13.3 The following are syntactically correct formulas in PTL:

$$p \wedge q, \quad \Box p, \quad \Diamond(p \wedge q) \rightarrow \Diamond p, \quad \Box\Box p \leftrightarrow \Box p, \quad \Diamond\Box p \leftrightarrow \Box\Diamond p, \quad \neg\Diamond p \wedge \Box\neg q.$$

The formula $\neg\Diamond p \;\wedge\; \Box\neg q$ is not ambiguous because the temporal operators and negation have higher precedence than the conjunction operator. The formula can be written $(\neg\Diamond p) \wedge (\Box\neg q)$ to distinguish it from $\neg(\Diamond p \wedge \Box\neg q)$. ■

13.2.2 Semantics

Informally, $\Box$ is a universal operator meaning 'for *any* time t in the future', while $\Diamond$ is an existential operator meaning 'for *some* time t in the future'. Two of the formulas from Example 13.1 can be written as follows in PTL:

$$\Box(\mathit{reset} \rightarrow \Diamond \mathit{zero}), \qquad \Box\neg\,\mathit{deadlocked}.$$

Interpretations of PTL formulas are based upon state transition diagrams. The intuitive meaning is that each state represents a *world* and a formula can have different truth values in different worlds. The transitions represent changes from one world to another.

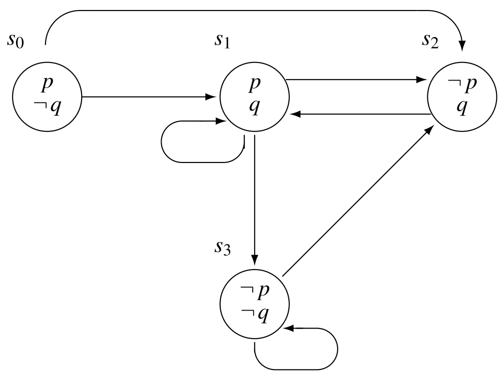


Fig. 13.1 State transition diagram

Definition 13.4 A *state transition diagram* is a directed graph. The nodes are *states* and the edges are *transitions*. Each state is labeled with a set of propositional literals such that clashing literals do not appear in any state. ■

Example 13.5 Figure 13.1 shows a state transition diagram where states are circles labeled with literals and transitions are arrows. ■

In modal logic, *necessarily* means in *all* (reachable) worlds, whereas *possibly* means in *some* (reachable) world. If a formula is possibly true, it can be true in some worlds and false in another.

Example 13.6 Consider the formula A = *the head of state of this country is a king*. The formula is possibly true but not necessarily true. If the possible worlds are the different countries, then at the present time A is true in Spain, false in Denmark (because the head of state is a queen) and false in France (which does not have a royal house). Even in a single country, the truth of A can change over time if a king is succeeded by a queen or if a monarchy becomes a republic. ■

Temporal logic is similar to modal logic except that the states are considered to specify what is true at a particular point of time and the transitions define the passage of time.

Example 13.7 Consider the formula A = *it is raining in London today*. On the day that this is being written, A is false. Let us consider each day as a state and the transitions to be the passage of time from one day to the next. Even in London $\Box A$ (meaning *every day, it rains in London)* is not true, but $\Diamond A$ (meaning *eventually, London will have a rainy day*) is certainly true. ■

We are now ready to define the semantics of PTL. An interpretation is a state transition diagram and the truth value of a formula is computed using the assignments to atomic propositions in each state and their usual meaning of the propositional operators. A formula that contains a temporal operator is interpreted using the transitions between the states.

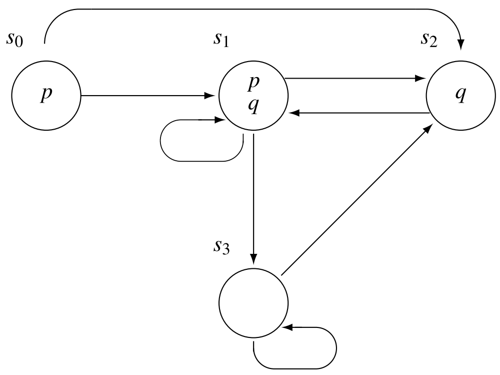


Fig. 13.2 Alternate representation of the state transition diagram in Fig. 13.1

Definition 13.8 An *interpretation* $\mathscr{I}$ for a formula A in PTL is a pair $(\mathscr{S}, \rho)$, where $\mathscr{S} = \{s_1, \ldots, s_n\}$ is a set of states each of which is an assignment of truth values to the atomic propositions in A, $s_i : \mathscr{P} \rightarrow \{T, F\}$, and ρ is a binary relation on the states, $\rho \subseteq S \times S$. ■

When displaying an interpretation graphically, the states are usually labeled only with the atomic propositions that are assigned T (Fig. 13.2). If an atom is not shown in the label of a state, it is assumed to be assigned F. Since it is clear how to transform one representation to the other, we will use whichever one is convenient.

A binary relation can be considered to be a mapping from a state to a set of states $\rho : S \rightarrow 2^S$, so the relational notation $(s_1, s_2) \in \rho$ will usually be written functionally as $s_2 \in \rho(s_1)$.

Example 13.9 In Fig. 13.2:

$$\begin{array}{ll}
s_0(p) = T, & s_0(q) = F, \\
s_1(p) = T, & s_1(q) = T, \\
s_2(p) = F, & s_2(q) = T, \\
s_3(p) = F, & s_3(q) = F.
\end{array}$$

$$\begin{aligned}
\rho(s_0) &= \{s_1, s_2\}, \\
\rho(s_1) &= \{s_1, s_2, s_3\}, \\
\rho(s_2) &= \{s_1\}, \\
\rho(s_3) &= \{s_2, s_3\}.
\end{aligned}$$

■

Definition 13.10 Let A be a formula in PTL. $v_{\mathscr{I},s}(A)$, the *truth value of A in s*, is defined by structural induction as follows:

- If A is $p \in \mathscr{P}$, then $v_{\mathscr{I},s}(A) = s(p)$.
- If A is $\neg A'$ then $v_{\mathscr{I},s}(A) = T$ iff $v_{\mathscr{I},s}(A') = F$.
- If A is $A' \vee A''$ then $v_{\mathscr{I},s}(A) = T$ iff $v_{\mathscr{I},s}(A') = T$ or $v_{\mathscr{I},s}(A'') = T$, and similarly for the other Boolean operators.
- If A is $\Box A'$ then $v_{\mathscr{I},s}(A) = T$ iff $v_{\mathscr{I},s'}(A') = T$ for *all* states $s' \in \rho(s)$.
- If A is $\Diamond A'$ then $v_{\mathscr{I},s}(A) = T$ iff $v_{\mathscr{I},s'}(A') = T$ for *some* state $s' \in \rho(s)$.

The notation $s \models_{\mathscr{I}} A$ is used for $v_{\mathscr{I},s}(A) = T$. When $\mathscr{I}$ is clear from the context, it can be omitted $s \models A$ iff $v_s(A) = T$. ■

Example 13.11 Let us compute the truth value of the formula $\Box p \vee \Box q$ for each state s in Fig. 13.2.

- $\rho(s_0) = \{s_1, s_2\}$. Since $s_1 \models q$ and $s_2 \models q$, it follows that $s_0 \models \Box q$. By the semantics of $\vee$, $s_0 \models \Box p \vee \Box q$.
- $s_3 \in \rho(s_1)$, but $s_3 \not\models p$ and $s_3 \not\models q$, so $s_1 \not\models \Box p$ and $s_1 \not\models \Box q$. Therefore, $s_1 \not\models \Box p \vee \Box q$.
- $\rho(s_2) = \{s_1\}$. Since $s_1 \models p$, we have $s_2 \models \Box p$ and $s_2 \models \Box p \vee \Box q$.
- $s_3 \in \rho(s_3)$. $s_3 \not\models \Box p \vee \Box q$ by the same argument used for s_1. ■

13.2.3 Satisfiability and Validity

The definition of semantic properties in PTL is more complex than it is in propositional or first-order logic, because an interpretation consists of both states and truth values.

Definition 13.12 Let A be a formula in PTL.

- A is *satisfiable* iff there is an interpretation $\mathscr{I} = (\mathscr{S}, \rho)$ for A and a state $s \in \mathscr{S}$ such that $s \models_{\mathscr{I}} A$.
- A is *valid* iff for all interpretations $\mathscr{I} = (\mathscr{S}, \rho)$ for A and for all states $s \in \mathscr{S}$, $s \models_{\mathscr{I}} A$. Notation: $\models A$. ■

Example 13.13 The analysis we did for the formula $A = \Box p \vee \Box q$ in Example 13.11 shows that A is satisfiable because $s_0 \models_{\mathscr{I}} A$ or because $s_2 \models_{\mathscr{I}} A$. The formulas A is not valid because $s_1 \not\models_{\mathscr{I}} A$ or because $s_3 \not\models_{\mathscr{I}} A$. ■

We leave it as an exercise to show that any valid formula of propositional logic is a valid formula of PTL, as is any *substitution instance* of a valid propositional formula obtained by substituting PTL formulas uniformly for propositional letters. For example, $\Box p \rightarrow (\Box q \rightarrow \Box p)$ is valid since it is a substitution instance of the valid propositional formula $A \rightarrow (B \rightarrow A)$.

There are other formulas of PTL that are valid because of properties of temporal logic and not as instances of propositional validities. We will prove the validity of two formulas directly from the semantic definition. The first establishes a duality between $\Box$ and $\Diamond$, and the second is the distribution of $\Box$ over $\rightarrow$, similar to the distribution of $\forall$ over $\rightarrow$.

Theorem 13.14 (Duality) $\models \Box p \leftrightarrow \neg \Diamond \neg p$.

Proof Let $\mathscr{I} = (\mathscr{S}, \rho)$ be an arbitrary interpretation for the formula and let s be an arbitrary state in $\mathscr{S}$. Assume that $s \models \Box p$, and suppose that $s \models \Diamond \neg p$. Then there exists a state $s' \in \rho(s)$ such that $s' \models \neg p$. Since $s \models \Box p$, for all states $t \in \rho(s)$, $t \models p$, in particular, $s' \models p$, contradicting $s' \models \neg p$. Therefore, $s \models \neg \Diamond \neg p$. Since $\mathscr{I}$ and s were arbitrary we have proved that $\models \Box p \rightarrow \neg \Diamond \neg p$. We leave the converse as an exercise. ■

Theorem 13.15 $\models \Box(p \rightarrow q) \rightarrow (\Box p \rightarrow \Box q)$.

Proof Suppose, to the contrary, that there is an interpretation $\mathscr{I} = (S, \rho)$ and a state $s \in S$, such that $s \models \Box(p \rightarrow q)$ and $s \models \Box p$, but $s \models \neg \Box q$. By Theorem 13.14, $s \models \neg \Box q$ is equivalent to $s \models \Diamond \neg q$, so there exists a state $s' \in \rho(s)$ such that $s' \models \neg q$. By the first two assumptions, $s' \models p \rightarrow q$ and $s' \models p$, which imply $s' \models q$, a contradiction. ■

13.3 Models of Time

In modal and temporal logics, different logics can be obtained by placing restrictions on the transition relation. In this section, we discuss the various restrictions, leading up to the ones that are appropriate for the temporal logics used in computer science. For each restriction on the transition relation, we give a formula that characterizes interpretations with that restriction. Proofs of the characterizations are given in a separate subsection.

Reflexivity

Definition 13.16 An interpretation $\mathscr{I} = (\mathscr{S}, \rho)$ is *reflexive* iff ρ is a reflexive relation: for all $s \in \mathscr{S}$, $(s, s) \in \rho$, or $s \in \rho(s)$ in functional notation. ■

Consider the formula $\Diamond$*running*, whose intuitive meaning is *eventually the program is in the state 'running'*. Obviously, if a program is running *now*, then there is an reachable state (namely, *now*) in which the program is running. Thus it is reasonable to require that interpretations for properties of programs be reflexive.

Theorem 13.17 *An interpretation with a reflexive relation is characterized by the formula* $\Box A \rightarrow A$ (*or, by duality, by the formula* $A \rightarrow \Diamond A$).

Transitivity

Definition 13.18 An interpretation $\mathscr{I} = (\mathscr{S}, \rho)$ is *transitive* iff ρ is a transitive relation: for all $s_1, s_2, s_3 \in \mathscr{S}$, $s_2 \in \rho(s_1) \wedge s_3 \in \rho(s_2) \rightarrow s_3 \in \rho(s_1)$. ■

It is natural to require that interpretations be transitive. Consider a situation where we have proved that $s_1 \models \Diamond running$ because $s_2 \models running$ for $s_2 \in \rho(s_1)$, and, furthermore, we have proved $s_2 \models \Diamond running$ because $s_3 \models running$ for $s_3 \in \rho(s_2)$. It would be very strange if $s_3 \notin \rho(s_1)$ and could not be used to prove $s_1 \models \Diamond running$.

Theorem 13.19 *An interpretation with a transitive relation is characterized by the formula* $\Box A \rightarrow \Box\Box A$ (*or by the formula* $\Diamond\Diamond A \rightarrow \Diamond A$).

Example 13.20 In Fig. 13.2, ρ is not transitive since $s_1 \in \rho(s_2)$ and $s_3 \in \rho(s_1)$ but $s_3 \notin \rho(s_2)$. This leads to the anomalous situation where $s_2 \models \Box p$ but $s_2 \not\models \Box\Box p$. ■

Corollary 13.21 *In an interpretation that both is reflexive and transitive*, $\models \Box A \leftrightarrow \Box\Box A$ *and* $\models \Diamond A \leftrightarrow \Diamond\Diamond A$.

Linearity

Definition 13.22 An interpretation $\mathscr{I} = (\mathscr{S}, \rho)$ is *linear* if ρ is a function, that is, for all $s \in \mathscr{S}$, there is at most one $s' \in \mathscr{S}$ such that $s' \in \rho(s)$.

It might appear that a linear temporal logic would be limited to expressing properties of sequential programs and could not express properties of concurrent programs, where each state can have several possible successors depending on the interleaving of the statements of the processes. However, linear temporal logic is successful precisely in the context of concurrent programs because there is an implicit universal quantification in the definitions.

Suppose we want to prove that a program satisfies a correctness property expressed as a temporal logic formula like $A = \Box\Diamond running$: in any state, the execution will eventually reach a state in which the computation is *running*. The program will be correct if this formula is true in *every* possible execution of the program obtained by interleaving the instructions of its processes. Each interleaving can be considered as a single linear interpretation, so if we prove $\models_{\mathscr{I}} A$ for an arbitrary linear interpretation $\mathscr{I}$, then the correctness property holds for the program.

Discreteness

Although the passage of time is often considered to be continuous and expressible by real numbers, the execution of a program is considered to be a sequence of *discrete* steps, where each step consists of the execution of a single instruction of the CPU. Thus it makes sense to express the concept of the *next* instant in time. To express discrete steps in temporal logic, an additional operator is added.

Definition 13.23 The unary operator $\bigcirc$ is called *next*. ■

The definition of the truth value of a formula is extended as expected:

Definition 13.24 If A is $\bigcirc A'$ then $v_{\mathscr{I},s}(A) = T$ iff $v_{\mathscr{I},s'}(A') = T$ for some $s' \in \rho(s)$. ■

The next operator is self-dual in a linear interpretation.

Theorem 13.25 *A linear interpretation whose relation ρ is a function is characterized by the formula $\bigcirc A \leftrightarrow \neg \bigcirc \neg A$.*

The operator $\bigcirc$ plays a crucial role in the theory of temporal logic and in algorithms for deciding properties like satisfiability, but it is rarely used to express properties of programs. In a concurrent program, not much can be said about what happens *next* since we don't know which operation will be executed in the next step. Furthermore, we want a correctness statement to hold regardless of how the interleaving selects a *next* operation. Therefore, properties are almost invariably expressed in terms of *always* and *eventually*, not in terms of *next*.

*13.3.1 Proofs of the Correspondences **

The following definition enables us to talk about the structure (the states and transitions) of an entire class of interpretations while abstracting away from the assignment to atomic propositions in each state. A frame is obtained from an interpretation by ignoring the assignments in the states; conversely, a interpretation is obtained from a frame by associating an assignment with each state.

Definition 13.26 A *frame* $\mathscr{F}$ is a pair $(\mathscr{W}, \rho)$, where $\mathscr{W}$ is a set of states and ρ is a binary relation on states. An interpretation $\mathscr{I} = (\mathscr{S}, \rho)$ is *based on a frame* $\mathscr{F} = (\mathscr{W}, \rho)$ iff there is a one-to-one mapping from $\mathscr{S}$ onto $\mathscr{W}$.

A PTL formula A *characterizes* a class of frames iff for every $\mathscr{F}_i$ in the class, the set of interpretations $\mathscr{I}$ based on $\mathscr{F}_i$ is the same as the set of interpretations in which A is true. ■

Theorems 13.17, 13.19 and 13.25 are more precisely stated as follows: the formulas $\Box A \rightarrow A$, $\Box A \rightarrow \Box\Box A$ and $\bigcirc A \leftrightarrow \neg \bigcirc \neg A$ characterize the sets of reflexive, transitive, and linear *frames*, respectively.

Proof of Theorem 13.17 Let $\mathscr{F}_i$ be a reflexive frame, let $\mathscr{I}$ be an arbitrary interpretation based on $\mathscr{F}_i$, and suppose that $\not\models_{\mathscr{I}} \Box A \rightarrow A$. Then there is a state $s \in \mathscr{S}$ such that $s \models_{\mathscr{I}} \Box A$ and $s \not\models_{\mathscr{I}} A$. By the definition of $\Box$, for *any* state $s' \in \rho(s)$, $s' \models_{\mathscr{I}} A$. By reflexivity, $s \in \rho(s)$, so $s \models_{\mathscr{I}} A$, a contradiction.

Conversely, suppose that $\mathscr{F}_i$ is not reflexive, and let $s \in \mathscr{S}$ be a state such that $s \notin \rho(s)$. If $\rho(s)$ is empty, $\Box p$ is vacuously true in s; by assigning F to $v_s(p)$, $s \not\models_{\mathscr{I}} \Box p \rightarrow p$. If $\rho(s)$ is non-empty, let $\mathscr{I}$ be an interpretation based on $\mathscr{F}_i$ such that $v_s(p) = F$ and $v_{s'}(p) = T$ for all $s' \in \rho(s)$. These assignments are well-defined since $s \notin \rho(s)$. Then $s \not\models_{\mathscr{I}} \Box p \rightarrow p$. ■

Proof of Theorem 13.19 Let $\mathscr{F}_i$ be a transitive frame, let $\mathscr{I}$ be an arbitrary interpretation based on $\mathscr{F}_i$, and suppose that $\not\models_{\mathscr{I}} \Box A \rightarrow \Box\Box A$. Then there is an $s \in \mathscr{S}$ such that $s \models_{\mathscr{I}} \Box A$ and $s \not\models_{\mathscr{I}} \Box\Box A$. From the latter formula, there must be an $s' \in \rho(s)$ such that $s' \not\models_{\mathscr{I}} \Box A$, and, then, there must be an $s'' \in \rho(s')$ be such that $s'' \not\models_{\mathscr{I}} A$. But $s \models_{\mathscr{I}} \Box A$, and by transitivity, $s'' \in \rho(s)$, so $s'' \models_{\mathscr{I}} A$, a contradiction.

Conversely, suppose that $\mathscr{F}_i$ is not transitive, and let $s, s', s'' \in \mathscr{S}$ be states such that $s' \in \rho(s)$, $s'' \in \rho(s')$, but $s'' \notin \rho(s)$. Let $\mathscr{I}$ be an interpretation based on $\mathscr{F}_i$ which assigns T to p in all states in $\rho(s)$ and F to p in s'', which is well-defined since $s'' \notin \rho(s)$. Then $s \models_{\mathscr{I}} \Box p$, but $s \not\models_{\mathscr{I}} \Box\Box p$. If there are only two states, s' need not be distinct from s. A one state frame is necessarily transitive, possibly vacuously if the relation is empty. ■

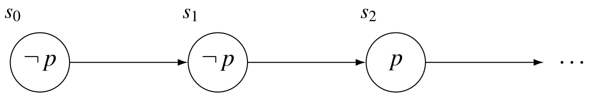
We leave the proof of Theorem 13.25 as an exercise.

13.4 Linear Temporal Logic

In the context of programs, the natural interpretations of temporal logic formulas are discrete, reflexive, transitive and linear. There is another restriction that simplifies the presentation: the transition function must be total so that each state has exactly one next state. An interpretation for a computation that terminates in state s is assumed to have a transition from s to s.

Definition 13.27 *Linear temporal logic (LTL)* is propositional temporal logic whose interpretations are limited to transitions which are discrete, reflexive, transitive, linear and total. ■

These interpretations can be represented as infinite paths:



Since there is only one transition out of each state, it need not be explicitly represented, so interpretations in LTL are defined to be paths of states:

Definition 13.28 An interpretation for an LTL formula A is a *path* of *states*:

$$\sigma = s_0, s_1, s_2, \ldots,$$

where each s_i is an assignment of truth values to the atomic propositions in A, $s_i : \mathscr{P} \rightarrow \{T, F\}$. Given σ, σ_i is the path that is the ith suffix of σ:

$$\sigma_i = s_i, s_{i+1}, s_{i+2}, \ldots.$$

$v_\sigma(A)$, the *truth value of A in σ*, is defined by structural induction:

- If A is $p \in \mathscr{P}$, then $v_\sigma(A) = s_0(p)$.
- If A is $\neg A'$ then $v_\sigma(A) = T$ iff $v_\sigma(A') = F$.
- If A is $A' \vee A''$ then $v_\sigma(A) = T$ iff $v_\sigma(A') = T$ or $v_\sigma(A'') = T$, and similarly for the other Boolean operators.
- If A is $\bigcirc A'$ then $v_\sigma(A) = T$ iff $v_{\sigma_1}(A') = T$.
- If A is $\Box A'$ then $v_\sigma(A) = T$ iff $v_{\sigma_i}(A') = T$ for *all* $i \geq 0$.
- If A is $\Diamond A'$ then $v_\sigma(A) = T$ iff $v_{\sigma_i}(A') = T$ for *some* $i \geq 0$.

If $v_\sigma(A) = T$, we write $\sigma \models A$. ■

Definition 13.29 Let A be a formula in LTL. A is *satisfiable* iff there is an interpretation σ for A such that $\sigma \models A$. A is *valid* iff for all interpretations σ for A, $\sigma \models A$. Notation: $\models A$. ■

Definition 13.30 A formula of the form $\bigcirc A$ or $\neg \bigcirc A$ is a *next* formula. A formula of the form $\Diamond A$ or $\neg \Box A$ is a *future* formula. ■

13.4.1 Equivalent Formulas in LTL

This section presents LTL formulas that are equivalent because of their temporal properties. Since any substitution instance of a formula in propositional logic is also an LTL formula, the equivalences in Sect. 2.3.3 also hold.

The equivalences are expressed in terms of an atom p but the intention is that they hold for arbitrary LTL formulas A.

The following formulas are direct consequences of our restriction of interpretations in LTL. The first three hold because interpretations are total, while the fourth holds because of linearity.

Theorem 13.31

$$\models \Box p \rightarrow \bigcirc p, \qquad \models \bigcirc p \rightarrow \Diamond p, \qquad \models \Box p \rightarrow \Diamond p, \qquad \models \bigcirc p \leftrightarrow \neg \bigcirc \neg p.$$

Inductive

The following theorem is extremely important because it provides an method for proving properties of LTL formulas inductively.

Theorem 13.32

$$\models \Box p \leftrightarrow p \wedge \bigcirc \Box p, \qquad \models \Diamond p \leftrightarrow p \vee \bigcirc \Diamond p.$$

These formulas can be easily understood by reading them in words: For a formula to be always true, p must be true today and, in addition, p must be always true tomorrow. For a formula to be true eventually, either p is true today or it must be true in some future of tomorrow.

We prove the first formula; the second follows by duality.

Proof Let σ be an arbitrary interpretation and assume that $\sigma \models \Box p$. By definition, $\sigma_i \models p$ for all $i \geq 0$; in particular, $\sigma_0 \models p$. But σ_0 is the same as σ, so $\sigma \models p$. If $\sigma \not\models \bigcirc\Box p$, then $\sigma_1 \not\models \Box p$, so for some $i \geq 1$, $\sigma_i \not\models p$, contradicting $\sigma \models \Box p$.

Conversely, assume that $\sigma \models p \wedge \bigcirc\Box p$. We prove by induction that $\sigma_i \models p \wedge \bigcirc\Box p$ for all $i \geq 0$. Since $\models A \wedge B \rightarrow A$ is a valid formula of propositional logic, we can conclude that $\sigma_i \models p$ for all $i \geq 0$, that is, $\sigma \models \Box p$.

The base case is immediate from the assumption since $\sigma_0 = \sigma$. Assume the inductive hypothesis that $\sigma_i \models p \wedge \bigcirc\Box p$. By definition of the semantics of $\bigcirc$, $\sigma_{i+1} \models \Box p$, that is, for all $j \geq i+1$, $\sigma_j \models p$, in particular $\sigma_{i+1} \models p$. Furthermore, for $j \geq i+2$, $\sigma_j \models p$, so $\sigma_{i+2} \models \Box p$ and $\sigma_{i+1} \models \bigcirc\Box p$. ■

Induction in LTL is based upon the following valid formula:

$$\models \Box(p \rightarrow \bigcirc p) \rightarrow (p \rightarrow \Box p).$$

The base case is to show that p holds in a state. The inductive assumption is p and the inductive step is to show that $p \rightarrow \bigcirc p$. When these two steps have been performed, we can conclude that $\Box p$.

Instead of proving the following equivalences semantically as in Theorem 13.32, we will prove them deductively in Chap. 14. By the soundness of the deductive system, they are valid.

Distributivity

The operators $\Box$ and $\bigcirc$ distribute over conjunction:

$$\models \Box(p \wedge q) \leftrightarrow (\Box p \wedge \Box q),$$
$$\models \bigcirc(p \wedge q) \leftrightarrow (\bigcirc p \wedge \bigcirc q).$$

The next operator also distributes over disjunction because it is self-dual, but $\Box$ only distributes over disjunction in one direction:

$$\models (\Box p \vee \Box q) \rightarrow \Box(p \vee q),$$
$$\models \bigcirc(p \vee q) \leftrightarrow (\bigcirc p \vee \bigcirc q).$$

By duality, there are similar formulas for $\Diamond$:

$$\models \Diamond(p \vee q) \leftrightarrow (\Diamond p \vee \Diamond q),$$
$$\models \Diamond(p \wedge q) \rightarrow (\Diamond p \wedge \Diamond q).$$

Similarly, $\Box$ and $\bigcirc$ distribute over implication in one direction, while $\bigcirc$ distributes in both directions:

$$\models \Box(p \rightarrow q) \rightarrow (\Box p \rightarrow \Box q),$$
$$\models (\Diamond p \rightarrow \Diamond q) \rightarrow \Diamond(p \rightarrow q),$$
$$\models \bigcirc(p \rightarrow q) \leftrightarrow (\bigcirc p \rightarrow \bigcirc q).$$

Example 13.33 Here is a counterexample to $\models (\Diamond p \wedge \Diamond q) \rightarrow \Diamond(p \wedge q)$:

$$\begin{tikzcd}
s_0 & s_1 & s_2 & s_3 & s_4 & \\
p \arrow[r] & q \arrow[r] & p \arrow[r] & q \arrow[r] & p \arrow[r] & \cdots
\end{tikzcd}$$

The atomic proposition p is true in even-numbered states, while q is true in odd-numbered states, but there is no state in which both are true. ■

Commutativity

The operator $\bigcirc$ commutes with $\Box$ and $\Diamond$, but $\Box$ and $\Diamond$ commute only in one direction:

$$\models \Box\bigcirc p \leftrightarrow \bigcirc\Box p,$$
$$\models \Diamond\bigcirc p \leftrightarrow \bigcirc\Diamond p,$$
$$\models \Diamond\Box p \rightarrow \Box\Diamond p.$$

Be careful to distinguish between $\Box\Diamond p$ and $\Diamond\Box p$. The formula $\Box\Diamond p$ means *infinitely often*: p is not required to hold continuously, but at any state it will hold at some future state.

$$\begin{tikzcd}
s_0 & s_1 & s_2 & s_3 & s_4 & s_5 & \\
\neg p \arrow[r] & \neg p \arrow[r] & p \arrow[r] & \neg p \arrow[r] & \neg p \arrow[r] & p \arrow[r] & \cdots
\end{tikzcd}$$

The formula $\Diamond\Box p$ means *for all but a finite number of states*: in a path $\sigma = s_0, s_1, s_2, \ldots,$ there is a natural number n such that p is true in all states in $\sigma_n = s_n, s_{n+1}, s_{n+2}, \ldots.$

$$\begin{tikzcd}
s_0 & s_1 & s_2 & s_3 & s_4 & s_5 & \\
\neg p \arrow[r] & \neg p \arrow[r] & \neg p \arrow[r] & \neg p \arrow[r] & p \arrow[r] & p \arrow[r] & \cdots
\end{tikzcd}$$

Theorem 13.34 $\models (\Diamond\Box p \wedge \Box\Diamond q) \rightarrow \Box\Diamond(p \wedge q)$.

Once p becomes always true, it will be true in the (infinite number of) states where q is true. We leave the proof as an exercise.

The diagram in Example 13.33 is also a counterexample to the formula: $\models (\Box\Diamond p \wedge \Box\Diamond q) \rightarrow \Box\Diamond(p \wedge q)$.

Collapsing

In a formula without the $\bigcirc$ operator, no more than two temporal operators need appear in a sequence. A sequence of identical operators $\Box$ or $\Diamond$ is equivalent to a single occurrence and a sequence of three non-identical operators collapses to a pair of operators:

$$\models \Box\Box p \leftrightarrow \Box p,$$
$$\models \Diamond\Diamond p \leftrightarrow \Diamond p,$$
$$\models \Box\Diamond\Box p \leftrightarrow \Diamond\Box p,$$
$$\models \Diamond\Box\Diamond p \leftrightarrow \Box\Diamond p.$$

13.5 Semantic Tableaux

The method of semantic tableaux is a decision procedure for satisfiability in LTL. The construction of a semantic tableau for a formula of LTL is more complex than that it is for a formula of propositional logic for two reasons:

First, to show that a formula in propositional logic is satisfiable, one need only find a *single* assignment to the atomic propositions that makes the formula evaluate to true. In LTL, however, there are many different assignments, one for each state. Therefore, we need to distinguish between ordinary nodes in the tableau used to decompose formulas such as $p \wedge q$ and $p \vee q$ from nodes that represent different states. For example, if $\bigcirc p$ is to be true in state s, then p must be assigned T in the state s' that follows s, but p could be assigned either T or F in s itself.

The second complication comes from future formulas like $\Diamond p$. For future formulas, it is not sufficient that they are consistent with the other subformulas; $\Diamond p$ requires that there actually exist a subsequent state where p is assigned T. This is similar to the case of $\exists x p(x)$ in first-order logic: we must demonstrate that a value a exists such that $p(a)$ is true. In first-order logic, this was simple, because we just chose new constant symbols from a countable set. In LTL, to establish the existence or non-existence of a state that *fulfills* a future formula requires an analysis of the graph of states constructed when the tableau is built.

13.5.1 The Tableau Rules for LTL

The tableau rules for LTL consist of the rules for propositional logic shown in Fig. 2.8, together with the following new rules, where next formulas are called X-formulas:

α	α_1	α_2
$\Box A$	A	$\bigcirc\Box A$
$\neg\Diamond A$	$\neg A$	$\neg\bigcirc\Diamond A$

β	β_1	β_2
$\Diamond A$	A	$\bigcirc\Diamond A$
$\neg\Box A$	$\neg A$	$\neg\bigcirc\Box A$

X	X_1
$\bigcirc A$	A
$\neg\bigcirc A$	$\neg A$

The Rules for α- and β-Formulas

The rules for the α- and β-formulas are based on Theorem 13.32:

- If $\Box A$ is true in a state s, then A is true in s *and* A must continue to be true in *all* subsequent states starting at the *next* state s'.
- If $\Diamond A$ is true in a state s, then either A is true in s *or* A will eventually become true in *some* subsequent state starting at the *next* state s'.

The Rule for X-Formulas

Consider now the tableau obtained for the formula $A = (p \vee q) \wedge \bigcirc(\neg p \wedge \neg q)$ after applying the rules for α- and β-formulas:

$$\begin{tikzcd}
 & (p \vee q) \wedge \bigcirc(\neg p \wedge \neg q) \arrow[d] & \\
 & p \vee q,\ \bigcirc(\neg p \wedge \neg q) \arrow[dl] \arrow[dr] & \\
\boxed{p,\ \bigcirc(\neg p \wedge \neg q)} & & \boxed{q,\ \bigcirc(\neg p \wedge \neg q)}
\end{tikzcd}$$

In a model σ for A, either $v_\sigma(p) = s_0(p) = T$ or $v_\sigma(q) = s_0(q) = T$, and this is expressed by the two leaf nodes that contain the atomic propositions. Since no more rules for α- and β-formulas are applicable, we have complete information on the assignment to atomic propositions in the initial state s_0. These nodes, therefore, define *states*, indicated by the frame around the node.

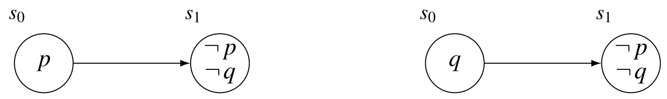
These nodes contain additional information: in order to satisfy the formula A, the formula $\bigcirc(\neg p \wedge \neg q)$ must evaluate to T in σ_0. Therefore, the formula $\neg p \wedge \neg q$ must evaluate to T in σ_1. The application of the rule for X-formulas begins the construction of the new state s_1:

$$\begin{tikzcd}
 & (p \vee q) \wedge \bigcirc(\neg p \wedge \neg q) \arrow[d] & \\
 & p \vee q,\ \bigcirc(\neg p \wedge \neg q) \arrow[dl] \arrow[dr] & \\
\boxed{p,\ \bigcirc(\neg p \wedge \neg q)} \arrow[d] & & \boxed{q,\ \bigcirc(\neg p \wedge \neg q)} \arrow[d] \\
\neg p \wedge \neg q \arrow[d] & & \neg p \wedge \neg q \arrow[d] \\
\boxed{\neg p,\ \neg q} & & \boxed{\neg p,\ \neg q}
\end{tikzcd}$$

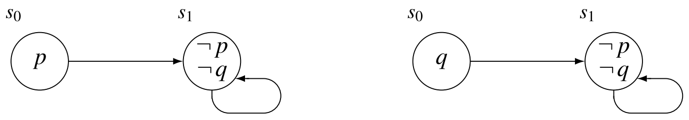
The literals in s_0 are *not* copied to the labels of the nodes created by the application of the rule for the X-formula because whatever requirements exist on the assignment in s_0 are not relevant to what happens in s_1.

On both branches, the new node is labeled by the formula $\neg p \wedge \neg q$ and an application of the rule for the propositional α-formula gives $\{\neg p, \neg q\}$ as the label of the next node. Since this node no longer contains α- or β-formulas, it defines a new state s_1.

The construction of the tableau is now complete and we have two open branches. Therefore, we can conclude that any model for A must be consistent with one of the following graphs:



This structure is not an interpretation. First, it is not total since there is no transition from s_1, but this is easily fixed by adding a self-loop to the final state:



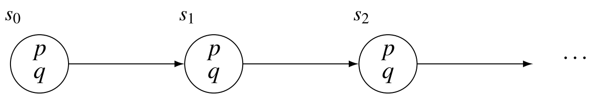
More importantly, we have not specified the value of the second literal in either of the possible states s_0. However, the structures are *Hintikka structures*, which can be extended to interpretations by specifying the values of all atoms in each state.

Future Formulas

Consider the formula $A = \neg\,(\Box(p \wedge q) \rightarrow \Box p)$ which is the negation of a valid formula. Here is a semantic tableau, where (by duality) we have implicitly changed $\neg\,\Box$ to $\Diamond\neg$ for clarity:

$$\begin{tikzcd}
 & \neg\,(\Box(p \wedge q) \rightarrow \Box p) \arrow[d] & \\
 & \Box(p \wedge q),\ \Diamond\neg\, p \arrow[d] & \\
 & p \wedge q,\ \bigcirc\Box(p \wedge q),\ \Diamond\neg\, p \arrow[d] & \\
 & p,\ q,\ \bigcirc\Box(p \wedge q),\ \Diamond\neg\, p \arrow[dl] \arrow[dr] & \\
p,\ q,\ \bigcirc\Box(p \wedge q),\ \neg\, p \arrow[d, phantom, "\times"] & & \boxed{p,\ q,\ \bigcirc\Box(p \wedge q),\ \bigcirc\Diamond\neg\, p} \\
{} & &
\end{tikzcd}$$

The left-hand branch closes, while the right-hand leaf defines a state s_0 in which p and q must be true. When rule for the X-formula is applied to this node, a new node is created that is labeled by $\{\Box(p \wedge q),\ \Diamond\neg\, p\}$. But this is the same set of formulas that labels the second node in the tableau. It is clear that the continuation of the construction will create an infinite structure:

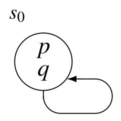


Something is wrong since A is unsatisfiable and its tableau should close!

This structure is a Hintikka structure (no node contains clashing literals and for every α-, β- and X-formula, the Hintikka conditions hold). However, the structure cannot be extended to model for A, since the future subformula $\Diamond\neg\, p$ is not *fulfilled*; that is, the structure promises to eventually produce a state in which $\neg\, p$ is true but defers forever the creation of such a state.

Finite Presentation of an Interpretation

There are only a finite number of *distinct* states in an interpretation for an LTL formula A since every state is labeled with a subset of the atomic propositions appearing in A and there are a finite number of such subsets. Therefore, although an interpretation is an infinite path, it can be *finitely presented* by reusing existing states instead of creating new ones. The infinite structure above can be finitely presented as follows:



13.5.2 Construction of Semantic Tableaux

The construction of semantic tableaux for LTL formulas and the proof of an algorithm for the decidability of satisfiability is contained in the following four subsections. First, we describe the construction of the tableau; then, we show how a Hintikka structure is defined by an open tableau; third, we extract a linear structure which can be extended to an interpretation; and finally, we show how to decide if future formulas are fulfilled.

The meaning of the following definition will become clear in the following subsection, but it is given here so that we can use it in the algorithm for constructing a tableau.

Definition 13.35 A *state node* in a tableau is a node l such that its label $U(l)$ contains only literals and next formulas, and there are no complementary pairs of literals in $U(l)$. ■

Algorithm 13.36 (Construction of a semantic tableau)
Input: An LTL formula A.
Output: A semantic tableau $\mathscr{T}$ for A.

Each node of $\mathcal{T}$ is labeled with a set of formulas. Initially, $\mathcal{T}$ consists of a single node, the root, labeled with the singleton set $\{A\}$. The tableau is built inductively as follows. Choose an unmarked leaf l labeled with a set of formulas $U(l)$ and perform one of the following steps:

- If there is a complementary pair of literals $\{p, \neg p\} \subseteq U(l)$, mark the leaf *closed* $\times$. If $U(l)$ is a set of literals but no pair is complementary, mark the leaf *open* $\odot$.
- If $U(l)$ is not a set of literals, choose $A \in U(l)$ which is an α-formula. Create a new node l' as a child of l and label l' with:

$$U(l') = (U(l) - \{A\}) \cup \{\alpha_1, \alpha_2\}.$$

  (In the case that A is $\neg\neg A_1$, there is no α_2.)
- If $U(l)$ is not a set of literals, choose $A \in U(l)$ which a β-formula. Create two new nodes l' and l'' as children of l. Label l' with:

$$U(l') = (U(l) - \{A\}) \cup \{\beta_1\},$$

  and label l'' with:

$$U(l'') = (U(l) - \{A\}) \cup \{\beta_2\}.$$

- If l is a state node (Definition 13.35) with at least one next formula, let:

$$\{\bigcirc A_1, \ldots, \bigcirc A_m, \neg \bigcirc A_{m+1}, \ldots, \neg \bigcirc A_n\}$$

  be the set of next formulas in $U(l)$. Create a new node l' as a child of l and label l' with:

$$U(l') = \{A_1, \ldots, A_m, \neg A_{m+1}, \ldots, \neg A_n\}.$$

  If $U(l') = U(l'')$ for a *state node* l'' that already exists in the tableau, do not create l'; instead connect l to l''.

The construction terminates when every leaf is marked $\times$ or $\odot$. ■

We leave it as an exercise to show that the construction always terminates.

Definition 13.37 A tableau whose construction has terminated is a *completed tableau*. A completed tableau is *closed* if all leaves are marked closed and there are no cycles. Otherwise, it is *open*. ■

Example 13.38 Here is a completed open semantic tableau with *no* leaves:

$$\begin{tikzcd}
 & & & l_0 : \Box\Diamond p \arrow[d] & & & \\
 & & & l_1 : \Diamond p,\ \bigcirc\Box\Diamond p \arrow[dll] \arrow[drr] & & & \\
 & l_2 : \boxed{p,\ \bigcirc\Box\Diamond p} \arrow[d] & & & & l_3 : \boxed{\bigcirc\Diamond p,\ \bigcirc\Box\Diamond p} \arrow[d] & \\
 & l_4 : \Box\Diamond p \arrow[d] & & & & l_5 : \Diamond p,\ \Box\Diamond p \arrow[d] & \\
 & l_6 : \Diamond p,\ \bigcirc\Box\Diamond p \arrow[dl] \arrow[dr] & & & & l_7 : \Diamond p,\ \bigcirc\Box\Diamond p \arrow[dl] \arrow[dr] & \\
\text{To } l_2 & & \text{To } l_3 & & \text{To } l_2 & & \text{To } l_3
\end{tikzcd}$$

■

13.5.3 From a Semantic Tableau to a Hintikka Structure

The next step is to construct a structure from an open tableau, to define the conditions for a structure to be a Hintikka structure and to prove that the structure resulting from the tableau satisfies those conditions. The definition of a structure is similar to the definition of an interpretation for *PTL* formulas (Definition 13.8); the difference is that the labels of a state are sets of formulas, not just sets of atomic propositions that are assigned true. To help understand the construction, you might want to refresh your memory by re-reading Sect. 2.7.2 on the definition and use of Hintikka structures in propositional logic.

Definition 13.39 A *structure* $\mathscr{H}$ for a formula A in LTL is a pair $(\mathscr{S}, \rho)$, where $\mathscr{S} = \{s_1, \ldots, s_n\}$ is a set of states each of which is labeled by a subset of formulas built from the atomic propositions in A and ρ is a binary relation on states, $\rho \subseteq S \times S$. ■

As before, functional notation may be used $s_2 \in \rho(s_1)$.

The states of the structure will be the state nodes of the tableau. However, the labels of the states must include more than the literals that label the nodes in the tableau. To obtain a Hintikka structure, the state in the structure must also include the formulas whose decomposition eventually led to each literal.

Example 13.40 In Example 13.38, state node l_2 will define a state in the structure that is labeled with p, since p must be assigned true in any interpretation containing that state. In addition, the state in the structure must also include $\Diamond p$ from l_1 (because p in l_2 resulted from the decomposition of $\Diamond p$), as well as $\Box\Diamond p$ from l_0 (because $\Diamond p$ in l_1 resulted from the decomposition of $\Box\Diamond p$). ■

The transitions in the structure are defined by paths between state nodes.

Definition 13.41 A *state path* is a path $(l_0, l_1, \ldots, l_{k-1}, l_k)$ through connected nodes in the tableau, such that l_0 is a state node or the root of the tableau, l_k is a state node, and none of $\{l_1, \ldots, l_{k-1}\}$ are state nodes. It is possible that $l_0 = l_k$ so that the set $\{l_1, \ldots, l_{k-1}\}$ is empty. ■

Given a tableau, a structure can be defined by taking the state nodes as the states and defining the transitions by the state paths. The label of a state is the union of all formulas that appear on incoming state paths (not including the first state of the path unless it is the root). The formal definition is:

Definition 13.42 Let $\mathscr{T}$ be an open tableau for an LTL formula A. The structure $\mathscr{H}$ constructed from $\mathscr{T}$ is:

- $\mathscr{S}$ is the set of state nodes.
- Let $s \in \mathscr{S}$. Then $s = l$ for some node l in the tableau. Let $\pi^i = (l_0^i, l_1^i, \ldots, l_{k_i}^i = l)$ be a state path terminating in the node l and let:

  $$U^i = U(l_1^i) \cup \cdots \cup U(l_{k_i}^i)$$

  or

  $$U^i = U(l_0^i) \cup \cdots \cup U(l_{k_i}^i)$$

  if l_0^i is the root. Label s by the set of formulas:

  $$U_i = \cup_i U^i,$$

  where the union is taken over all i such that π^i is a state path terminating in $l = s$.
- $s' \in \rho(s)$ iff there is a state path from s to s'. ■

It is possible to obtain several disconnected structures from the tableau for a formula such as $\Diamond p \vee \Diamond q$, but this is no problem as the formula can be satisfiable if and only if at least one of the structures leads to a model.

Now that we know how the structure is constructed from the tableau, it is possible to optimize Algorithm 13.36. Change:

> For a *state node* l', if $U(l') = U(l'')$ for a *state node* l'' that already exists in the tableau, do not create l'; instead connect l to l''.

so that it applies to any node l' in the tableau, not just to state nodes, *provided* that this doesn't create a cycle not containing a state node.

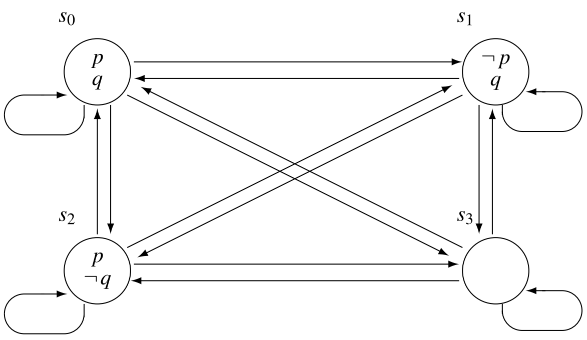
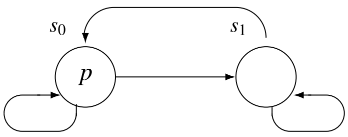


Fig. 13.3 Structure for $\Box(\Diamond(p \wedge q) \wedge \Diamond(\neg p \wedge q) \wedge \Diamond(p \wedge \neg q))$

Example 13.43 Here is an optimized tableau corresponding to the one in Example 13.38:

$$\begin{tikzcd}
& l_0 : \Box\Diamond p \arrow[d] & \\
& l_1 : \Diamond p,\ \bigcirc\Box\Diamond p \arrow[dl] \arrow[dr] & \\
l_2 : \boxed{p,\ \bigcirc\Box\Diamond p} \arrow[d] & & l_3 : \boxed{\bigcirc\Diamond p,\ \bigcirc\Box\Diamond p} \arrow[d] \\
\text{To } l_0 & & \text{To } l_1
\end{tikzcd}$$

and here is the structure constructed from this semantic tableau:



where $s_0 = l_2$ and $s_1 = l_3$. To save space, each state s_i is labeled only with the positive literals in U_i. ■

Example 13.44 Let:

$$A = \Box(\Diamond(p \wedge q) \wedge \Diamond(\neg p \wedge q) \wedge \Diamond(p \wedge \neg q)).$$

The construction of the tableau for A is left as an exercise. The structure obtained from the tableau is shown in Fig. 13.3. ■

Definition 13.45 Let $\mathscr{H} = (\mathscr{S}, \rho)$ be a structure for an LTL formula A. $\mathscr{H}$ is a *Hintikka structure for A* iff $A \in s_0$ and for all states s_i the following conditions hold for U_i, the set of formulas labeling s_i:

1. For all atomic propositions p in A, either $p \notin U_i$ or $\neg p \notin U_i$.
2. If $\alpha \in U_i$, then $\alpha_1 \in U_i$ *and* $\alpha_2 \in U_i$.
3. If $\beta \in U_i$, then $\beta_1 \in U_i$ *or* $\beta_2 \in U_i$.
4. If $X \in U_i$, then for all $s_j \in \rho(s_i)$, $X_1 \in U_j$. ■

Theorem 13.46 *Let A be an LTL formula and suppose that the tableau $\mathscr{T}$ for A is open. Then the structure $\mathscr{H}$ created as described in Definition 13.42 is a Hintikka structure for A.*

Proof The structure $\mathscr{H}$ is created from an open tableau, so condition (1) holds. Rules for α- and β-formulas are applied *before* rules for next formulas, so the union of the formulas on every incoming state path to a state node contains all the formulas required by conditions (2) and (3). When the rule for a next formula $\bigcirc A$ is applied, A will appear in the label of the next node (and similarly for $\neg \bigcirc A$), and hence in every state at the end of a state path that includes this node. ■

13.5.4 Linear Fulfilling Hintikka Structures

The construction of the tableau and the Hintikka structure is quite straightforward given the decomposition of formulas with temporal operators. Now we turn to the more difficult problem of deciding if an interpretation for an LTL formula can be extracted from a Hintikka structure. First, we need to extract a linear structure and show that it is also a Hintikka structure.

Definition 13.47 Let $\mathscr{H}$ be a Hintikka structure for an LTL formula A. $\mathscr{H}$ is a *linear* Hintikka structure iff ρ is a total function, that is, if for each s_i there is exactly one $s_j \in \rho(s_i)$. ■

Lemma 13.48 *Let $\mathscr{H}$ be a Hintikka structure for an LTL formula A and let $\mathscr{H}'$ be an infinite path through $\mathscr{H}$. Then $\mathscr{H}'$ is a linear Hintikka structure.*

Proof Clearly, $\mathscr{H}'$ is a linear structure. Conditions (1–3) of Definition 13.45 hold because they already held in $\mathscr{H}$. Let s be an arbitrary state and let U be the label of s. If a next formula $\bigcirc A'$ occurs in U, then by condition (4) of Definition 13.45, A' occurs in *all* states of $\rho(s)$, in particular, for the one chosen in the construction of $\mathscr{H}'$. ■

Next, we need to check if the linear structure fulfills all the future formulas. We define the concept of fulfilling and then show that a fulfilling Hintikka structure can be used to define a model. The algorithm for deciding if a Hintikka structure is fulfilling is somewhat complex and is left to the next subsection. To simplify the presentation, future formulas will be limited to those of the form $\Diamond A$. By duality, the same presentation is applicable to future formulas of the form $\neg \Box A$.

Recall that $\rho*$ is the transitive, reflexive closure of ρ (Definition A.21).

Definition 13.49 Let $\mathscr{H} = (\mathscr{S}, \rho)$ be a Hintikka structure. $\mathscr{H}$ is a *fulfilling* iff the following condition holds for all future formulas $\Diamond A$:

For all $s \in \mathscr{S}$, if $\Diamond A \in U_s$, then for some $s' \in \rho^*(s)$, $A \in U_{s'}$.

The state s' is said to *fulfill* $\Diamond A$. ■

Theorem 13.50 (Hintikka's Lemma for LTL) *Let $\mathscr{H} = (\mathscr{S}, \rho)$ be a linear fulfilling Hintikka structure for an LTL formula A. Then A is satisfiable.*

Proof An LTL interpretation is a path consisting of states labeled with atomic propositions (see Definition 13.28). The path is defined simply by taking the linear Hintikka structure and restricting the labels to atomic propositions. There is thus a natural mapping between states of the interpretation and states of the Hintikka structure, so for the propositional operators and next formulas, we can use the conditions on the structure to prove that A is satisfiable using structural induction.

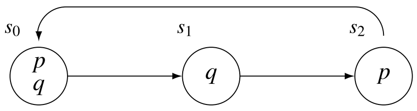
For future formulas, the satisfiability follows from the assumption that the Hintikka structure is fulfilling.

Consider now a formula of the form $\Box A \in U_{s_i}$. We must show that $v_{\sigma_j}(A) = T$ for all $j \geq i$. We generalize this for the inductive proof and show that $v_{\sigma_j}(A) = T$ and $v_{\sigma_j}(\bigcirc\Box A) = T$ for all $j \geq i$.

The base case is $j = i$. But $\Box A \in U_{s_i}$, so by Hintikka condition (2) $A \in U_{s_i}$ and $\bigcirc\Box A \in U_{s_i}$.

Let $k \geq i$ and assume the inductive hypothesis that $v_{\sigma_k}(A) = T$ and $\bigcirc\Box A \in U_{s_k}$. By Hintikka condition (4), $\Box A \in U_{s_{k+1}}$, so using Hintikka condition (2) again, $v_{\sigma_{k+1}}(A) = T$ and $\bigcirc\Box A \in U_{s_{k+1}}$. ■

Here is a finite presentation of a linear fulfilling Hintikka structure constructed from the structure in Fig. 13.3:



13.5.5 Deciding Fulfillment of Future Formulas *

The last link needed to obtain a decision procedure for satisfiability in LTL is an algorithm that takes an arbitrary Hintikka structure, and decides if it contains a path that is a linear fulfilling Hintikka structure. We begin with some definitions from graph theory. The concepts should be familiar, though it is worthwhile giving formal definitions.

Definition 13.51 A *graph* $G = (V, E)$ consists of a set of *vertices* $V = \{v_1, \ldots, v_n\}$ and a set of *edges* $E = \{e_1, \ldots, e_m\}$, which are pairs of vertices $e_k = \{v_i, v_j\} \subseteq V$.

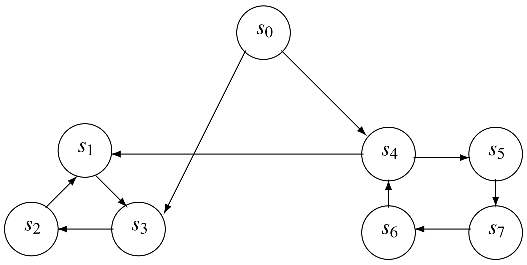


Fig. 13.4 Strongly connected components

In a *directed* graph, each edge is an ordered pair, $e_k = (v_i, v_j)$. A *path* from v to v', denoted $v \rightsquigarrow v'$, is a sequence of edges such that the second component of one edge is the first component of the next:

$$\begin{aligned} e_1 &= (v = v_{i_1}, v_{i_2}), \\ e_2 &= (v_{i_2}, v_{i_3}), \\ &\cdots \\ e_{l-1} &= (v_{i_{l-2}}, v_{i_{l-1}}), \\ e_l &= (v_{i_{l-1}}, v_{i_l} = v'). \end{aligned}$$

A subgraph $G' = (V', E')$ of a directed graph $G = (V, E)$ is a graph such that $V' \subseteq V$ and $E' \subseteq E$, provided that $e = (v_i, v_j) \in E'$ implies $\{v_i, v_j\} \subseteq V'$. ■

Definition 13.52 A *strongly connected component (SCC)* $G' = (V', E')$ in a directed graph G is a subgraph such that $v_i \rightsquigarrow v_j$ for all $\{v_i, v_j\} \subseteq V'$. A *maximal strongly connected component (MSCC)* is an SCC not properly contained in another. A *transient SCC* is an MSCC consisting of a single vertex. A *terminal SCC* is an MSCC with no outgoing edges. ■

Example 13.53 The directed graph in Fig. 13.4 contains three strongly connected components: $G_0 = \{s_0\}$, $G_1 = \{s_1, s_2, s_3\}$, $G_2 = \{s_4, s_5, s_6, s_7\}$. G_0 is transient and G_1 is terminal. ■

Definition 13.54 A directed graph G can be represented as a *component graph*, which is a directed graph whose vertices are the MSCCs of G and whose edges are edges of G pointing from a vertex of one MSCC to a vertex of another MSCC. ■

See Even, Sect. 3.4 for an algorithm that constructs the component graph of a directed graph and a proof of the following theorem.

Theorem 13.55 *The component graph is acyclic.*

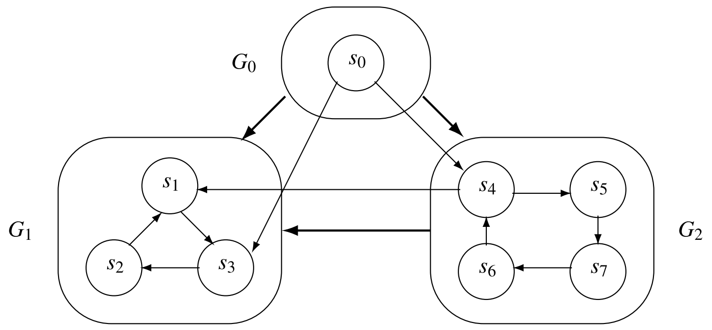


Fig. 13.5 Component graph

Example 13.56 Figure 13.5 shows the graph of Fig. 13.4 with its component graph indicated by ovals and thick arrows. ■

Suppose that we have a Hintikka structure and a future formula in a *terminal* MSCC, such as G_1 in Fig. 13.5. Then if the formula is going to be fulfilled at all, it will be fulfilled within the terminal MSCC because there are no other reachable nodes to which the fulfillment can be deferred. If a future formula is in a *non-terminal* MSCC such as G_2, it can either be fulfilled within its own MSCC, or the fulfillment can be deferred to an reachable MSCC, in this case G_1. This suggests an algorithm for checking fulfillment: start at terminal MSCCs and work backwards.

Let $\mathscr{H} = (\mathscr{S}, \rho)$ be a Hintikka structure. $\mathscr{H}$ can be considered a graph $G = (V, E)$, where V is $\mathscr{S}$ and $(s_i, s_j) \in E$ iff $s_j \in \rho(s_i)$. We simplify the notation and write $A \in v$ for $A \in U_i$ when $v = s_i$.

Definition 13.57 Let $G = (V, E)$ be a SCC of $\mathscr{H}$. G is *self-fulfilling* iff for all $v \in V$ and for all future formulas $\Diamond A \in v$, $A \in v'$ for some $v' \in V$. ■

Lemma 13.58 *Let $G = (V, E) \subseteq G' = (V', E')$ be SCCs of a Hintikka structure. If G is self-fulfilling, then so is G'.*

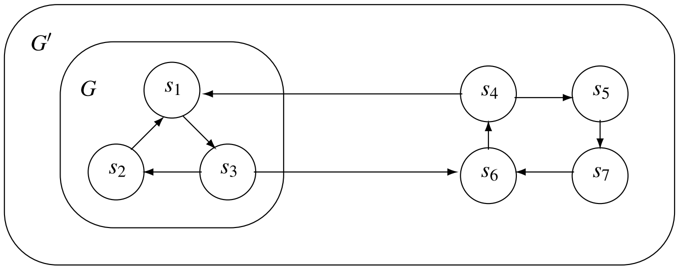


Fig. 13.6 An SCC is contained in an MSCC

Example 13.59 Let $\Diamond A$ be an arbitrary future formula that has to be fulfilled in G' in Fig. 13.6. If $\Diamond A \in s_i$ for $s_i \in G$, then by the assumption that G is self-fulfilling, $A \in s_j$ for some $s_j \in G \subset G'$ and G' is also self-fulfilling.

Suppose now that $\Diamond A \in s_7$, where $s_7 \in V' - V$. If $A \in s_7$, then s_7 itself fulfills $\Diamond A$. Otherwise, by Hintikka condition (3), $\bigcirc\Diamond A \in s_7$, so $\Diamond A \in s_6$ by Hintikka condition (4). Continuing, $A \in s_6$ or $\bigcirc\Diamond A \in s_6$; $A \in s_4$ or $\bigcirc\Diamond A \in s_4$; $A \in s_5$ or $\bigcirc\Diamond A \in s_5$. If $A \in s_j$ for one of these vertices in $V' - V$, we have the G' is self-fulfilling.

If not, then by Hintikka condition (4), $\bigcirc\Diamond A \in s_4$ implies that $\Diamond A \in s_1$, because condition (4) is a requirement on *all* immediate successors of a node. By assumption, G is self-fulfilling, so $A \in s_j$ for some $s_j \in G \subset G'$ and G' is also self-fulfilling. ■

Proof of Lemma 13.58 Let $\Diamond A$ be an arbitrary future formula in $v' \in V' - V$. By definition of a Hintikka structure, either $A \in v'$ or $\bigcirc\Diamond A \in v'$. If $A \in v'$, then A is fulfilled in G'; otherwise, $\Diamond A \in v''$ for every $v'' \in \rho(v')$. By induction on the number of vertices in $V' - V$, either A is fulfilled in $V' - V$ or $\Diamond A \in v$ for some v in V. But G is self-fulfilling, so $\Diamond A$ is fulfilled in some state $v_A \in V \subseteq V'$. Since G' is an SCC, $v' \rightsquigarrow v_A$ and A is fulfilled in G'. ■

Corollary 13.60 *Let G be a self-fulfilling SCC of a Hintikka structure. Then G can be extended to a self-fulfilling MSCC.*

Proof If G itself is not an MSCC, create a new graph G' by adding a vertex $v' \in V' - V$ and all edges (v', v) and (v, v'), where $v \in V$, provided that G' is an SCC. Continue this procedure until no new SCCs can be created. By Lemma 13.58, the SCC is self-fulfilling and by construction it is maximal. ■

Lemma 13.61 *Let $G = (V, E)$ be an MSCC of $\mathscr{H}$ and let $\Diamond A \in v \in V$ be a future formula. If G is not self-fulfilling, $\Diamond A$ can only be fulfilled by some v' in an MSCC G', such that $G \rightsquigarrow G'$ in the component graph.*

Proof Since G is not self-fulfilling, $\Diamond A$ must be fulfilled by some $v' \notin V$ such that $v \rightsquigarrow v'$. But $v' \not\rightsquigarrow v$, otherwise v' could be added to the vertices of G creating a larger SCC, contradicting the assumption that G is maximal. Therefore, $v' \in G'$ for a component $G' \neq G$. ■

This lemma directly gives the following corollary.

Corollary 13.62 *If G is a terminal MSCC and $\Diamond A \in v$ for $v \in V$, then if $\Diamond A$ cannot be fulfilled in G, it cannot be fulfilled at all.*

Algorithm 13.63 (Construction of a linear fulfilling structure)
Input: A Hintikka structure $\mathscr{H}$.
Output: A linear fulfilling Hintikka structure that is a path in $\mathscr{H}$, or a report that no such structure exists.

Construct the component graph H of $\mathscr{H}$. Since H is acyclic (Theorem 13.55), there must be a terminal MSCC G. If G is not self-fulfilling, delete G and all its incoming edges from H. Repeat until every terminal MSCC is self-fulfilling or until the component graph is empty. If every terminal MSCC is self-fulfilling, the proof of the following theorem shows how a linear fulfilling Hintikka structure can be constructed. Otherwise, if the graph is empty, the algorithm reports that no linear fulfilling Hintikka structure exists. ■

Theorem 13.64 *Algorithm 13.63 terminates with a non-empty graph iff a linear fulfilling Hintikka structure can be constructed.*

Proof Suppose that the algorithm terminates with an non-empty component graph G and let $G_1 \rightsquigarrow \cdots \rightsquigarrow G_n$ be a maximal path in G. We now define a path in $\mathscr{H}$ based upon this path in the component graph.

There must be vertices $\{v_1, \ldots, v_n\}$ in $\mathscr{H}$, such that $v_i \in G_i$, $v_{i+1} \in G_{i+1}$ and $v_i \rightsquigarrow v_{i+1}$. Furthermore, each component G_i is an SCC, so for each i there is a path $v_1^i \rightsquigarrow \cdots \rightsquigarrow v_{k_i}^i$ in $\mathscr{H}$ containing all the vertices in G_i.

Construct a path in $\mathscr{H}$ by replacing every component by a partial path and connecting them by the edges $v_i \rightsquigarrow v_{i+1}$:

- Replace a transient component by the single vertex v_1^i.
- Replace a terminal component by the closure

$$v_i \rightsquigarrow \cdots \rightsquigarrow (v_1^i \rightsquigarrow \cdots \rightsquigarrow v_{k_i}^i)^*.$$

- Replace a non-transient, non-terminal component by

$$v_i \rightsquigarrow \cdots \rightsquigarrow v_1^i \rightsquigarrow \cdots v_{k_i}^i \rightsquigarrow v_1^i \rightsquigarrow \cdots v_{k_i}^i \rightsquigarrow \cdots \rightsquigarrow v_{i+1}.$$

We leave it as an exercise to prove that this path is a fulfilling linear Hintikka structure.

Conversely, let $\mathscr{H}' = (s_1, \ldots, \ldots)$ be a fulfilling linear Hintikka structure in $\mathscr{H}$. Since $\mathscr{H}$ is finite, some suffix of $\mathscr{H}'$ must be composed of states which repeat infinitely often. These states must be contained within a self-fulfilling SCC G. By Corollary 13.60, G is contained in a self-fulfilling MSCC. ■

Example 13.65 There are two maximal paths in the component graph in Fig. 13.5: $G_0 \rightsquigarrow G_1$ and $G_0 \rightsquigarrow G_2 \rightsquigarrow G_1$. The paths constructed in the underlying graphs are:

$$s_0 \rightsquigarrow (s_3 \rightsquigarrow s_2 \rightsquigarrow s_1)^*$$

and

$$s_0 \rightsquigarrow s_4 \rightsquigarrow s_5 \rightsquigarrow s_7 \rightsquigarrow s_6 \rightsquigarrow s_4 \rightsquigarrow s_5 \rightsquigarrow s_7 \rightsquigarrow s_6 \rightsquigarrow s_4 \rightsquigarrow (s_1 \rightsquigarrow s_2 \rightsquigarrow s_3)^*,$$

respectively. ■

Theorem 13.66 *There is a decision procedure for satisfiability in LTL.*

Proof Let A be a formula in LTL. Construct a semantic tableau for A. If it closes, A is unsatisfiable. If there is an open branch, A is satisfiable. Otherwise, construct the structure from the tableau as described in Definition 13.42. By Theorem 13.46, this is a Hintikka structure. Apply Algorithm 13.63 to construct a fulfilling Hintikka structure. If the resulting graph is empty, A is unsatisfiable. Otherwise, apply the construction in Theorem 13.64 to construct a linear fulfilling Hintikka structure. By Theorem 13.50, a model can be constructed from the structure. ∎

The following corollary is obvious since the number of possible states in a structure constructed for a particular formula is finite:

Corollary 13.67 (Finite model property) *A formula in LTL is satisfiable iff it is satisfiable in a finitely-presented model.*

13.6 Binary Temporal Operators *

Consider the following correctness specification from the introduction:

The output lines *maintain* their values *until* the set-line is asserted.

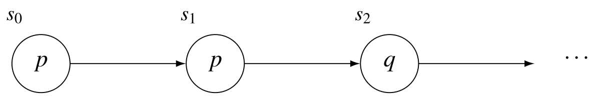
We cannot express this in LTL as defined above because we have no binary temporal operators that can connect two propositions: *unchanged-output* and *set-asserted*. To express such properties, a binary operator $\mathcal{U}$ (read *until*) can be added to LTL. Infix notation is used:

$$\textit{unchanged-output} \quad \mathcal{U} \quad \textit{set-asserted}.$$

The semantics of the operator is defined by adding the following item to Definition 13.28:

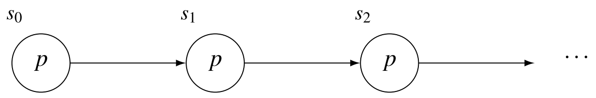
- If A is $A_1 \mathcal{U} A_2$ then $v_\sigma(A) = T$ iff $v_{\sigma_i}(A_2) = T$ for *some* $i \geq 0$ and for *all* $0 \leq k < i$, $v_{\sigma_k}(A_1) = T$.

Example 13.68 The formula $p \mathcal{U} q$ is true in the interpretation represented by the following path:


$$\begin{tikzcd}
s_0 & s_1 & s_2 & \\
p \arrow[r] & p \arrow[r] & q \arrow[r] & \cdots
\end{tikzcd}$$

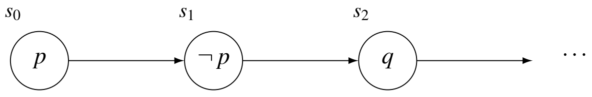
q is true at s_2 and for all previous states $\{s_0, s_1\}$, p is true.

$p \mathcal{U} q$ is not true in the following interpretation assuming that state s_2 is repeated indefinitely:


$$\begin{tikzcd}
s_0 & s_1 & s_2 & \\
p \arrow[r] & p \arrow[r] & p \arrow[r] & \cdots
\end{tikzcd}$$

The reason is that q never becomes true.

$p \mathcal{U} q$ is also not true in the following interpretation:



because p becomes false before q becomes true. ■

Defining the Existing Operators in Terms of $\mathcal{U}$

It is easy to see that:

$$\Diamond A \equiv \mathit{true} \, \mathcal{U} \, A.$$

The definition of the semantics of $\mathcal{U}$ requires that A become true eventually just as in the semantics of $\Diamond A$. The additional requirement is that *true* evaluate to T in every previous state, but that clearly holds in every interpretation.

Since binary operators are essential for expressing correctness properties, advanced presentations of LTL take $\bigcirc$ and $\mathcal{U}$ as the primitive operators of LTL and define $\Diamond$ as an abbreviation for the above formula, and then $\Box$ as an abbreviation for $\neg \Diamond \neg$.

Semantic Tableaux with $\mathcal{U}$

Constructing a semantic tableau for a formula that uses the $\mathcal{U}$ operator does not require any new concepts. The operator can be decomposed as follows:

$$A_1 \mathcal{U} A_2 \;\equiv\; A_2 \vee (A_1 \wedge \bigcirc(A_1 \mathcal{U} A_2)).$$

For $A_1 \mathcal{U} A_2$ to be true, either A_2 is true today, or we put off to tomorrow the requirement to satisfy $A_1 \mathcal{U} A_2$, while requiring that A_1 be true today. The decomposition shows that a $\mathcal{U}$-formula is a β-formula very similar to $\Diamond A$. The similarity goes deeper, because $A_1 \mathcal{U} A_2$ is a future formula and must be fulfilled by having A_2 appear in a state eventually.

The construction of semantic tableau is more efficient if operators have duals. The dual of $\mathcal{U}$ is the operator $\mathcal{R}$ (read *release*), defined as:

$$A_1 \mathcal{R} A_2 \;\equiv\; \neg (\neg A_1 \mathcal{U} \neg A_2).$$

We leave it as an exercise to write the definition of the semantics of $\mathcal{R}$.

The Weak Until Operator

Sometimes it is convenient to express precedence properties without actually requiring that something eventually occur. $\mathcal{W}$ (read *weak until*) is the same as the operator $\mathcal{U}$ except that it is not required that the second formula ever become true:

- If A is $A_1 \mathcal{W} A_2$ then $v_\sigma(A) = T$ iff: **if** $v_{\sigma_i}(A_2) = T$ for *some* $i \geq 0$, **then** for *all* $0 \leq k < i$, $v_{\sigma_k}(A_1) = T$.

Clearly, the following equivalence holds:

$$A_1 \mathcal{W} A_2 \equiv (A_1 \mathcal{U} A_2) \vee \Box A_1.$$

We leave it as an exercise to show:

$$\begin{array}{rcl} \Box A & \equiv & A \mathcal{W} \mathit{false}, \\ \neg (A_1 \mathcal{W} A_2) & \equiv & (A_1 \wedge \neg A_2) \mathcal{U} (\neg A_1 \wedge \neg A_2), \\ \neg (A_1 \mathcal{U} A_2) & \equiv & (A_1 \wedge \neg A_2) \mathcal{W} (\neg A_1 \wedge \neg A_2), \\ \neg (A_1 \mathcal{U} A_2) & \equiv & (\neg A_2) \mathcal{W} (\neg A_1 \wedge \neg A_2). \end{array}$$

13.7 Summary

Since the state of a computation changes over time, temporal logic is an appropriate formalism for expressing correctness properties of programs. The syntax of linear temporal logic (LTL) is that of propositional logic together with the unary temporal operators $\Box$, $\Diamond$, $\bigcirc$. Interpretations are infinite sequences of states, where each state assigns truth values to atomic propositions. The meaning of the temporal operators is that some property must hold in $\Box$ *all* subsequent states, in $\Diamond$ *some* subsequent state or in the $\bigcirc$ *next* state.

Satisfiability and validity of formulas in LTL are decidable. The tableau construction for propositional logic is extended so that *next formulas* (of the form $\bigcirc A$) cause new states to be generated. A open tableau defines a Hintikka structure which can be extended to a satisfying interpretation, provided that all *future formulas* (of the form $\Diamond A$ or $\neg \Box A$) are fulfilled. By constructing the component graph of strongly connected components, the fulfillment of the future formulas can be decided.

Many important correctness properties use the binary operators $\mathcal{U}$ and $\mathcal{W}$, which require that one formula hold until a second one becomes true.

13.8 Further Reading

Temporal logic (also called *tense logic*) has a long history, but it was first applied to program verification by Pnueli (1977). The definitive reference for the specification and verification of concurrent programs using temporal logic is Manna and Pnueli (1992, 1995). The third volume was never completed, but a partial draft is available (Manna and Pnueli, 1996). Modern treatments of LTL can be found in Kröger and Merz (2008, Chap. 2), and Baier and Katoen (2008, Chap. 5). The tableau method for a different version of temporal logic first appeared in Ben-Ari et al. (1983); for a modern treatment see Kröger and Merz (2008, Chap. 2).

13.9 Exercises

13.1 Prove that in LTL every substitution instance of a valid propositional formula is valid.

13.2 Prove $\models \neg\Diamond\neg p \rightarrow \Box p$ (the converse direction of Theorem 13.14).

13.3 Prove that a linear interpretation is characterized by $\bigcirc A \leftrightarrow \neg\bigcirc\neg A$ (Theorem 13.25).

13.4 * Identify the property of a reflexive relation characterized by $A \rightarrow \Box\Diamond A$. Identify the property of a reflexive relation characterized by $\Diamond A \rightarrow \Box\Diamond A$.

13.5 Show that in an interpretation with a reflexive transitive relation, any formula (without $\bigcirc$) is equivalent to one whose only temporal operators are $\Box$, $\Diamond$, $\Diamond\Box$, $\Box\Diamond$, $\Diamond\Box\Diamond$ and $\Box\Diamond\Box$. If the relation is also characterized by the formula $\Diamond A \rightarrow \Box\Diamond A$, any formula is equivalent to one with a single temporal operator.

13.6 Prove Theorem 13.34: $\models (\Diamond\Box p \wedge \Box\Diamond q) \rightarrow \Box\Diamond(p \wedge q)$.

13.7 Construct a tableau and find a model for the negation of $\Box\Diamond p \rightarrow \Diamond\Box p$.

13.8 Prove that the construction of a semantic tableau terminates.

13.9 Prove that the construction of the path in the proof of Theorem 13.64 gives a linear fulfilling Hintikka structure.

13.10 Write the definition of the semantics of the operator $\mathscr{R}$.

13.11 Prove the equivalences on $\mathscr{W}$ at the end of Sect. 13.6.

References

C. Baier and J.-P. Katoen. *Principles of Model Checking*. MIT Press, 2008.

M. Ben-Ari, Z. Manna, and A. Pnueli. The temporal logic of branching time. *Acta Informatica*, 20:207–226, 1983.

S. Even. *Graph Algorithms*. Computer Science Press, Potomac, MD, 1979.

F. Kröger and S. Merz. *Temporal Logic and State Systems*. Springer, 2008.

Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems. Vol. I: Specification*. Springer, New York, NY, 1992.

Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems. Vol. II: Safety*. Springer, New York, NY, 1995.

Z. Manna and A. Pnueli. Temporal verification of reactive systems: Progress. Draft available at `http://www.cs.stanford.edu/~zm/tvors3.html`, 1996.

A. Pnueli. The temporal logic of programs. In *18th IEEE Annual Symposium on Foundations of Computer Science*, pages 46–57, 1977.