Chapter 14

Linear Temporal Logic LTL

Contents

In this chapter we define a logic that can be used for expressing temporal properties of concurrent systems. This logic extends propositional logic by several *temporal operators*. Such logics are generally known as *temporal logics*. There are several temporal logics used in model checkers, each one has its own collection of temporal operators. In Chapter 16 we will consider two other temporal logics: CTL^* and CTL.

14.1 Kripke Structures

In this section we introduce a notion of *Kripke* structure named after their inventor [Kripke 1963]. that is very similar but not identical to the notion of transition system. Transition systems are convenient for modeling systems while Kripke structures are more convenient for defining temporal logics. We will show that every transition system can be considered as a Kripke structure.

In the definition we assume a *fixed instance* $PLFD(\mathcal{X}, dom)$ *of PLFD*. Denote the set of all interpretations for this instance of PLFD by $\mathbb{I}$.

DEFINITION 14.1 (Kripke Structure) A *Kripke structure* is a tuple $K = (S, In, T, L)$, where

(1) S is a finite non-empty set, called the set of *states* of K.

(2) $In \subseteq S$ is a non-empty set of states, called the set of *initial states* of K.

(3) $T \subseteq S \times S$ is a set of pairs of states, called the *transition relation* of K.

(4) $L : S \to \mathbb{I}$ is a function, called the *labeling function* of K. ❏

Note that the definition of Kripke structure is implicitly parametrized by an instance $PLFD(\mathcal{X}, dom)$ of PLFD. When we want to mention this instance explicitly, we will speak about a *Kripke structure over* $PLFD(\mathcal{X}, dom)$. In literature on model checking (e.g. [Clarke, Grumberg and Peled 1999, Clarke and Schlingloff 2001]) it is usually assumed that all variables are boolean. In this case the instance of PLFD is simply identified by its set of variables $\mathcal{X}$ and we speak about a *Kripke structure over* $\mathcal{X}$.

Every finite-state transition system $\mathbb{S} = (\mathcal{X}, D, dom, I, T)$ can be made into a Kripke structure $K = (S, In, T', L)$ as follows.

(1) The instance of PLFD is $PLFD(\mathcal{X}, dom)$, as in the symbolic representation of transition systems.

(2) The set of states of K is the set of all states of $\mathbb{S}$, that is, $S = \mathbb{I}$, where $\mathbb{I}$ is the set of all interpretations for this instance of PLFD.

(3) K and $\mathbb{S}$ have the same initial states, that is $In = I$.

(4) The transition relation of K is the transition relation of $\mathbb{S}$, that is, $T' = Tr_{\mathbb{S}} = \bigcup_{t \in T} t$.

(5) The labeling function L simply maps every state s of $\mathbb{S}$ into itself: $L(s) \overset{\text{def}}{=} s$ (remember that states in transition systems are interpretations). This can be reformulate as follows:

$$L(s)(x) = s(x).$$

Though the two notions of transition system and Kripke structure are very similar, there are some differences between them. In a transition system the set of states is the set of *all* interpretations of $PLFD(\mathcal{X}, dom)$, in a Kripke structure sets are *mapped* into such instances. This has two consequences.

(1) In a transition system every interpretation is a state. For a Kripke structure there may be an interpretation I such that there is no state s with the property $L(s) = I$.

(2) In a Kripke structure there may be two *different* states s_1, s_2 labeled by the same interpretation, that is $L(s_1) = L(s_2)$. In a transition systems there is only one state correponding to a single interpretation.

We will illustrate both differences after introducing a notion of *state transition graph*.

DEFINITION 14.2 (State Transition Graph) Let $K = (S, In, T, L)$ be a Kripke structure. Its *state transition graph* is a directed graph whose set of nodes is S and set of arcs is T. Additionally, each node s in the graph is labeled by the interpretation $L(s)$. ❑

Consider an example.

EXAMPLE 14.3 Consider an instance of PLFD with one boolean variable x and the following Kripke structure K over $\{x\}$. It has two states s_1, s_2, where s_1 is the initial state, the transition relation T is $\{(s_1, s_1), (s_1, s_2), (s_2, s_1)\}$, the labelling function maps s_1 into $\{x \mapsto 1\}$ and s_2 into $\{x \mapsto 0\}$. In other words, we have $x = 1$ at s_1 and $x = 0$ at s_2. The state transition graph of this Kripke structure is

$$\begin{tikzcd} s_1 : x = 1 \arrow[loop left] \arrow[r, bend left] & s_2 : x = 0 \arrow[l, bend left] \end{tikzcd}$$

❑

As usual, for a boolean variable x we will write x instead of $x = 1$ and $\neg x$ for $x = 0$. Using this convention, we can draw the state transition graph of the Kripke structure of this example as

$$\begin{tikzcd} s_1 : x \arrow[loop left] \arrow[r, bend left] & s_2 : \neg x \arrow[l, bend left] \end{tikzcd}$$

Consider a more lengthy example.

EXAMPLE 14.4 Take a transition system $\mathbb{S}$ for a simplified model of the vending machine. The coin slot contains at most two coins, there are no student customers, and the machine serves only coffee. The set $\mathcal{X}$ contains the following variables.

(1) A boolean variable storage, which is true if the storage is non-empty, i.e., there is coffee in the storage.

(2) A boolean variable dispenser, which is true if the dispenser is non-empty, i.e., if there coffee in the dispenser.

(3) A variable coins with the domain $0, 1, 2$ denoting the number of coins in the slot.

(4) A boolean variable customer, which is true if a customer is present.

The transitions are the same as for the vending machine example, except that there is no transition $Dispense\_beer$. The state transition graph for this example is shown in Figure 14.1. ❑

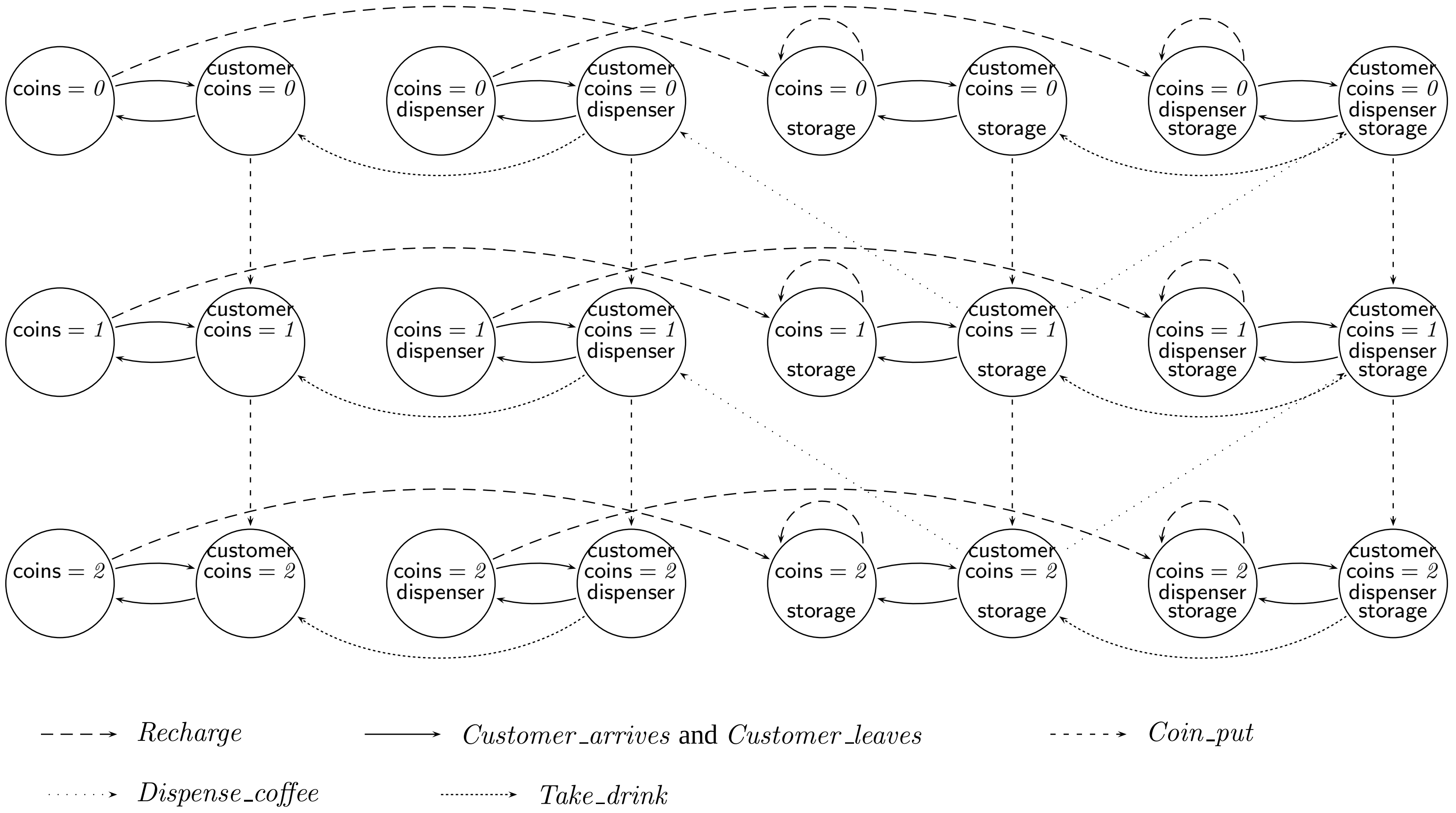

coins = 0
customer coins = 0
coins = 0 dispenser
customer coins = 0 dispenser
coins = 0 storage
customer coins = 0 storage
coins = 0 dispenser storage
customer coins = 0 dispenser storage
coins = 1
customer coins = 1
coins = 1 dispenser
customer coins = 1 dispenser
coins = 1 storage
customer coins = 1 storage
coins = 1 dispenser storage
customer coins = 1 dispenser storage
coins = 2
customer coins = 2
coins = 2 dispenser
customer coins = 2 dispenser
coins = 2 storage
customer coins = 2 storage
coins = 2 dispenser storage
customer coins = 2 dispenser storage
Recharge
Customer_arrives and Customer_leaves
Coin_put
Dispense_coffee
Take_drink

Figure 14.1: State transition graph for the simplified vending machine model

14.2 Computation Trees

The set of all possible behaviors of a Kripke structure or a transition system can be defined through a notion of *computation tree*.

DEFINITION 14.5 (Computation Tree, Computation) Let $K = (S, In, T, L)$ be a Kripke structure over a set of variables $\mathcal{X}$ and $s \in S$ be a state. The *computation tree for K starting at s* is the following (possibly infinite) tree.

(1) The nodes of the tree are labeled by states in S.

(2) The root of the tree is labeled by s.

(3) For every node s' in the tree, its children are exactly such nodes $s'' \in S$ that $(s', s'') \in T$.

A *computation path* for K is a sequence of nodes $s_0, s_1, \ldots$ such that

(1) for all i we have $(s_i, s_{i+1}) \in T$;

(2) if the sequence is finite, i.e., it has the form $s_1, \ldots, s_n$, then there exists no state s such that $(s_n, s) \in T$. ❏

In other words, a computation path is any maximal sequence of states through which a computation may go by applying the transitions.

It is not hard to argue that computation trees and paths have the following properties.

(1) Computation paths for a Kripke structure are exactly all branches in the computation trees for this Kripke structure.

(2) Let n be a node in a computation tree C for K labeled by s'. Then the subtree of C rooted at s' is the computation tree for K starting at s'. In other words, every subtree of a computation tree rooted at some node is itself a computation tree.

(3) For every Kripke structure K and state s there exists a unique computation tree for K starting at s, up to the order of children.

For example, a part of a computation tree for the Kripke structure of Example 14.3 is given in Figure 14.2 on the next page. Likewise, a part of a computation tree for the Kripke structure of Example 14.4 is given in Figure 14.3 on page 213. In the latter figure we label the arcs of the computation tree by the names of the corresponding transitions. The trees can be obtained by "unwinding" the corresponding state transition graphs (see, e.g., Figure 14.1 on the preceding page).

One can note that the computation tree is a convenient object for discussing possible temporal behaviors of the Kripke structure, since assertions about possible temporal behaviors can be conveniently formulated as properties of paths in the tree and states on these

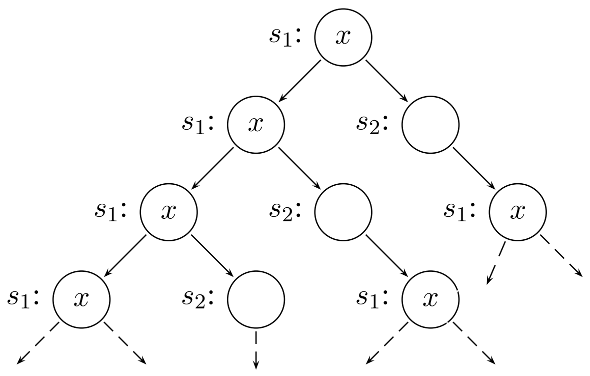


Figure 14.2: Computation tree for Example 14.3.

paths. Consider, for instance, two examples of temporal properties of the vending machine transition system:

(1) There is no state in which a professor and a student are both customers.

(2) Students never drink coffee.

To express the first property, we can say that at *all states in the tree* we have $\mathsf{customer} \neq \mathit{student} \lor \mathsf{customer} \neq \mathit{prof}$. Or, alternatively, we can say that for *every path* in the tree and *every node on this path* we have $\mathsf{customer} \neq \mathit{student} \lor \mathsf{customer} \neq \mathit{prof}$.

The second property can be reformulated in terms of paths and states as follows: *for every path* in the tree and *two consecutive states* s_1, s_2 on this path, if $s_1 \models \mathsf{customer} = \mathit{student} \land \mathsf{disp} = \mathit{coffee}$, then $s_2 \models \mathsf{disp} = \mathit{coffee}$.

14.3 LTL

In this section we introduce a logic in which one express properties of paths in a computation tree. In particular, properties such as "for some state on the path" or "for every two consecutive states" can be expressed. This logic is called *linear temporal logic*, or simply LTL

DEFINITION 14.6 (Formula of LTL) The notion of an LTL *formula* is defined inductively as follows.

(1) $\top$ and $\bot$ are formulas.

(2) Every atomic formula $x = v$ of PLFD is an (atomic) formula of LTL.

(3) If $A_1, \ldots, A_n$ are formulas, where $n \geq 2$, then $(A_1 \land \ldots \land A_n)$ and $(A_1 \lor \ldots \lor A_n)$ are formulas.

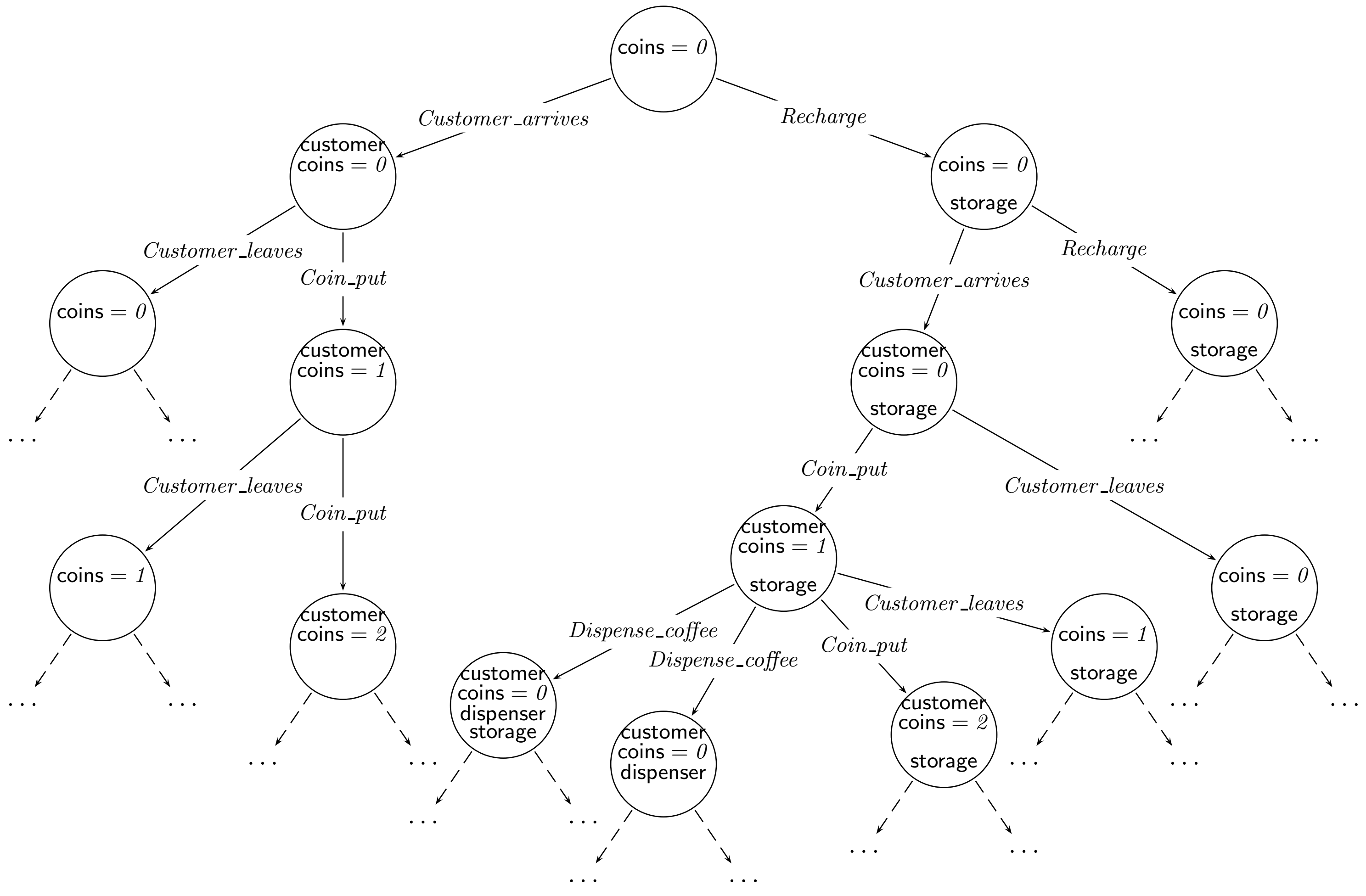


Figure 14.3: A computation tree for the Kripke structure of Example 14.4

(4) If A is a formula, then $\neg A$ is a formula.

(5) If A and B are formulas, then $(A \rightarrow B)$ and $(A \leftrightarrow B)$ are formulas.

(6) If A is a formula, then $\bigcirc A$, $\Diamond A$, and $\Box A$ are formulas.

(7) If A and B are formulas, then $A \,\mathbf{U}\, B$ and $A \,\mathbf{R}\, B$ are formulas.

The symbols $\bigcirc, \Diamond, \Box, \mathbf{U}, \mathbf{R}$ are called *temporal operators*. ❏

Sometimes we will simply refer to the formulas of LTL (and formulas of CTL* introduced in Chapter 16) as *temporal formulas*.

Before we define the semantics of LTL formulas formally, let us try to explain their meaning informally. The formulas of LTL are true or false on computation paths, that is, sequences of states $s_0, s_1, \ldots$. The formula $\Box A$ means that A is true at *all* states along the path. The formula $\Diamond A$ means that A is true at *some* state on the path. The formula $\bigcirc A$ means that A is true at the *next* state after the initial one, that is, at s_1. The formulas $A \,\mathbf{U}\, B$ and $A \,\mathbf{R}\, B$ are slightly more complex and will be explained below.

DEFINITION 14.7 (Semantics of LTL) Let $\pi = s_0, s_1, s_2 \ldots$ be a sequence of states and A be an LTL formula. We define the notion A *is true on* π, denoted by $\pi \models A$, by induction on A as follows. For all $i = 0, 1, \ldots$ denote by π_i the sequence of states $s_i, s_{i+1}, s_{i+2} \ldots$ (note that $\pi_0 = \pi$).

(1) $\pi \models \top$ and $\pi \not\models \bot$.

(2) $\pi \models x = v$ if $s_0 \models x = v$.

(3) $\pi \models A_1 \wedge \ldots \wedge A_n$ if for all $j = 1, \ldots, n$ we have $\pi \models A_j$;
$\pi \models A_1 \vee \ldots \vee A_n$ if for some $j = 1, \ldots, n$ we have $\pi \models A_j$.

(4) $\pi \models \neg A$ if $\pi \not\models A$.

(5) $\pi \models A \rightarrow B$ if either $\pi \not\models A$ or $\pi \models B$;
$\pi \models A \leftrightarrow B$ if either both $\pi \not\models A$ and $\pi \not\models B$ or both $\pi \models A$ and $\pi \models B$.

(6) $\pi \models \bigcirc A$ if $\pi_1 \models A$;
$\pi \models \Diamond A$ if for some $i = 0, 1, \ldots$ we have $\pi_i \models A$;
$\pi \models \Box A$ if for all $i = 0, 1, \ldots$ we have $\pi_i \models A$.

(7) $\pi \models A \,\mathbf{U}\, B$ if for some $k = 0, 1, \ldots$ we have $\pi_k \models B$ and $\pi_0 \models A, \ldots, \pi_{k-1} \models A$;
$\pi \models A \,\mathbf{R}\, B$ if for all $k \geq 0$, either $\pi_k \models B$ or there exists $j < k$ such that $\pi_j \models A$. ❏

Two LTL formulas A and B are called *equivalent*, denoted $A \equiv B$, if for every path π we have $\pi \models A$ if and only if $\pi \models B$. ❏

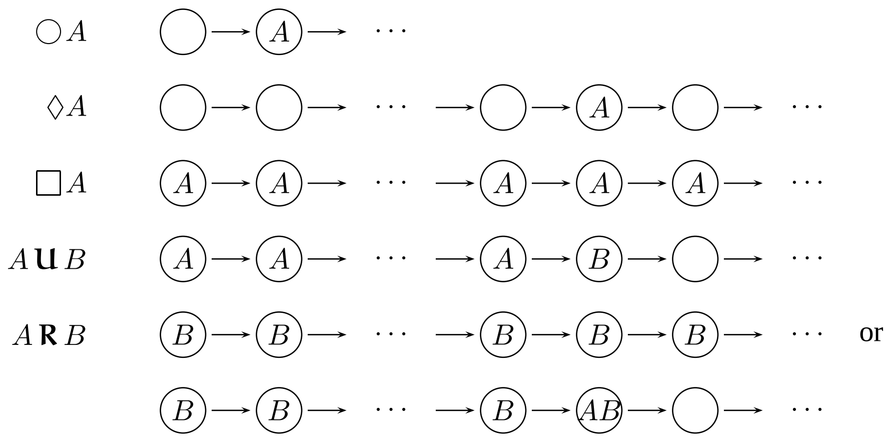


Figure 14.4: Semantics of temporal operators

When we consider a path π and paths π_i as in this definition, instead of saying that a temporal formula A is true on π_i we will sometimes say that A is true at the state s_i on the path π.

The semantics of the temporal operators of LTL is illustrated in Figure 14.4. Less formally, it can be explained as follows (we write in parentheses the name of the corresponding temporal operator).

- $\bigcirc$ (next) The formula $\bigcirc A$ holds, if A holds at the next state on the path.
- $\Diamond$ (eventually) The formula $\Diamond A$ holds, if A eventually occurs, i.e., A holds at some state on the path.
- $\Box$ (always) The formula $\Box A$ holds, if A holds globally, i.e., at every state along the path.
- $\mathbf{U}$ (until) The formula $A\,\mathbf{U}\,B$ holds, if A holds until B occurs, i.e., there is a state on the path at which B holds, and at every state before A holds.
- $\mathbf{R}$ (release) The formula $A\,\mathbf{R}\,B$ holds, if, whenever $\neg B$ occurs at a state on the path, A occurs before. Or equivalently, either B holds globally on the path, or A occurs before the first state at which B is violated.

14.4 Expressing Properties of Transition Systems in LTL

The formulas of LTL express properties of paths in computation trees, hence they are relevant to temporal properties of Kripke structures or transition systems. But we know that the set of all possible behaviors of a Kripke structure is identified by its computation tree, so what is the meaning of these formulas when we discuss properties of Kripke structures? A computation tree for a Kripke structure K can be identified with the collection of its paths. We know that these paths describe all possible computations of this Kripke structure. So if we have an LTL formula A, we can consider at least two kinds of properties of K:

(1) does A hold on *some* computation path for K from an initial state?

(2) does A hold on *all* computation paths for K from an initial state?

The two properties are dual to each other. It is not hard to argue that A holds on some computation path if and only if it is not true that $\neg A$ holds on all computation paths. Vice versa, A holds on all computation paths if and only if it is not true that $\neg A$ holds on some computation path. This means that, if we can express one of the properties, we can also express the other one. For this reason, we will sometimes refer to temporal formulas as expressing properties of Kripke structures. In such cases we will try to be careful enough to explain which of the two properties we have in mind.

Reachability and safety properties. A state is called *reachable* if there is a computation path from an initial state leading to this state. Reachability is one of the most important properties of transition systems in connection with *safety properties*. Suppose that unsafe is a formula which expresses an undesirable property of a transition system. States satisfying unsafe are usually called *unsafe* or *bad*. Then the system is *safe* if one cannot reach a state at which unsafe holds. Naturally, we would like to know whether the system is safe. Reachability of a state satisfying unsafe can be expressed as the existence of a path satisfying $\Diamond\mathsf{unsafe}$. Then safety of the system can be expressed as non-reachability of a state satisfying unsafe, i.e., the property $\Box\neg\mathsf{unsafe}$. Naturally, this property must be held on *all* computation paths.

A vending machine is not an especially dangerous device, so it is not easy to invent interesting safety properties for it. One possible example of an unsafe behavior would be serving beer to a professor, which can be expressed by the formula $\mathsf{disp} = beer \wedge \mathsf{customer} = prof$. Thus, the corresponding safety property is

$$\Box(\mathsf{disp} \neq beer \vee \mathsf{customer} \neq prof).$$

Another natural example of a safety property for the vending machine transition system is that the machine is never empty, i.e., there is always either coffee or beer in the storage. It can be expressed by the following formula:

$$\Box(\mathsf{st\_coffee} \vee \mathsf{st\_beer}).$$

A bit more complex example is this: whenever there is a customer, the machine has a drink. It can be expressed by the following formula:

$$\Box(\mathsf{customer} \neq none \rightarrow \mathsf{st\_coffee} \vee \mathsf{st\_beer}).$$

Mutual exclusion. *Mutual exclusion* is usually formulated as a property of concurrent systems. It arises when two or more processes are not allowed to enter the same *critical section* of a concurrent system simultaneously. Assuming that there are two processes P_1, P_2, and that formulas $\mathsf{critical}_i$, where $i = 1, 2$ denote that P_i is in the critical section, mutual exclusion can be expressed by

$$\Box\neg(\mathsf{critical}_1 \wedge \mathsf{critical}_2).$$

Some natural mutual exclusion properties for the vending machine example are the following. First, coffee and beer cannot be in the dispenser simultaneously:

$$\Box\neg(\mathsf{disp} = \mathit{coffee} \wedge \mathsf{disp} = \mathit{beer}).$$

Likewise, we may want to express that a professor and a student cannot be customers at the same time:

$$\Box\neg(\mathsf{customer} = \mathit{student} \wedge \mathsf{customer} = \mathit{prof}),$$

Deadlock. Speaking very generally, a concurrent program is in a *deadlock* situation, when no terminal state is reached, yet no part of the program is able to proceed. A transition system or Kripke structure is said to be *deadlock-free* if no computation in it leads to a deadlock. Assuming that the set of terminal states is represented by a temporal formula $\mathsf{terminal}$, we can express deadlock-freedom by the formula

$$\Box(\bigcirc\bot \to \mathsf{terminal}).$$

This formula must be true on every path. Indeed, it is easy to see that the formula $\bigcirc\bot$ means "there is no next state", that is, no transition is possible. Likewise, we can express reachability of a deadlock state as the existence of a state with the dual property

$$\Diamond(\bigcirc\bot \wedge \neg\mathsf{terminal}).$$

Termination and finiteness. Even if we do not have a notion of a terminal state, one can define *terminal states* as those from which no transition is possible. We know that a terminal state can be represented by the formula $\bigcirc\bot$. A transition system or Kripke structure is called *terminating*, if every computation in it leads to a terminal state. Termination for a Kripke structure is equivalent to the finiteness of all computation paths, which by König's Lemma is equivalent to the finiteness of every computation tree from an initial state. But a computation path is finite if and only if it contains a deadlock state. Therefore, the following formula expresses that the computation tree is finite: $\Diamond\bigcirc\bot$ (provided that this formula holds on every path).

Fairness. Usually, we are not interested in arbitrary computations of a transition system, as we know that some computations are impossible. For example, we know that the vending machine must be recharged from time to time. This can be formulated as follows: on every computation path, the recharge transaction occurs infinitely many times. This kind of constraints imposed on the system: the system must from time to time pass through a state which satisfies some property, is called a *fairness constraint*, and computations satisfying fairness constraints are called *fair*. For the vending machine example, one can impose many natural fairness constraints. For example, we may require that the dispenser contains a drink infinitely often, that students are customers infinitely often etc. Fairness w.r.t. a property expressed by a formula A means that A holds infinitely often on all paths.

We claim that fairness w.r.t. A can be expressed by the following formula: $\Box\Diamond A$. To this end, take any path $\pi = s_0, s_1, \ldots$. Denote by π_j for $j = 0, 1, \ldots$ the path $s_j, s_{j+1}, \ldots$. Let us prove the following property: for every $j = 0, 1, \ldots$ there exists $k > j$ such that $\pi_k \models A$. Indeed, since $\pi \models \Box\Diamond A$, we also have $\pi_{j+1} \models \Diamond A$. But this means that there exists $k \geq j+1$ such that $p_k \models A$. Evidently, $k > j$, so we are done. By this property, there exists $j_0 > 0$ such that $\pi_{j_0} \models A$. Again by this property, there exists also $j_1 > j_0$ such that $\pi_{j_1} \models A$. By using this argument again and again we can build an infinite sequence of numbers $j_0 < j_1 < j_2 < \ldots$ such that $\pi_{j_m} \models A$ for all m, so A occurs infinitely often on the path π.

In the proof that $\Box\Diamond A$ expresses that A holds infinitely often we assumed that the path is infinite. When the path π is finite, it is not hard to argue that this property implies that A holds at the last state of π. We can ensure that the path is infinite by asserting $\Box\bigcirc\top$. Likewise, the property $\Box\bigcirc\Diamond A$ expresses that A holds infinitely often and the path is infinite.

Responsiveness. It is often the case in concurrent systems that one process sends requests that have to be acknowledged (or responded to) by other processes. For such systems we are interested in the *responsiveness* property: whether every request is eventually acknowledged. Assuming that the request is expressed by a formula $\mathsf{request}$ and acknowledgement by a formula ack, one can express responsiveness by the formula

$$\Box(\mathsf{request} \rightarrow \bigcirc\Diamond\mathsf{ack}).$$

If we also want that $\mathsf{request}$ should remain true until it is acknowledged, responsiveness can be expressed by the formula

$$\Box(\mathsf{request} \rightarrow (\mathsf{request}\,\mathbf{U}\,\mathsf{ack})).$$

We can also require that the request formula and the acknowledgement formula be mutually exclusive, i.e., $\mathsf{request}$ should remain true until it is acknowledged, after which it immediately becomes false. This can be expressed by the formula

$$\Box(\mathsf{request} \rightarrow ((\mathsf{request} \land \neg\mathsf{ack})\,\mathbf{U}\,(\neg\mathsf{request} \land \mathsf{ack}))).$$

Alternation. To give the reader an idea of other properties expressible in LTL, consider the following example. Let $\pi = s_0, s_1, s_2, \ldots$ be a path. We claim that the formula $A \land \Box(A \leftrightarrow \neg\bigcirc A)$ expresses the following property: A is true at the even states $s_0, s_2, s_4, \ldots$ and false at the odd states s_1, s_3, s_5 on this path, as illustrated in the following picture:

$$\begin{tikzcd} A \arrow[r] & \neg A \arrow[r] & A \arrow[r] & \neg A \arrow[r] & A \arrow[r] & \neg A \arrow[r] & A \arrow[r] & \cdots \end{tikzcd}$$

We will denote subpaths of π by $\pi_0, \pi_1, \ldots$ as before. Indeed, we have $\pi_0 \models \Box(A \leftrightarrow \neg\bigcirc A)$, which implies that for all i, $\pi_i \models A$ if and only if $\pi_{i+1} \models \neg A$. Therefore, the value of A changes from any state to the next state, and so A is either true exactly at all even states or exactly at all odd states. By $\pi \models A$ means that $\pi_0 \models A$, so A is true at the even states.

Interestingly enough, the property "A is true at the even states" is not expressible by an LTL formula. This property can be illustrated by the following picture:

$$\begin{tikzcd} A \arrow[r] & ? \arrow[r] & A \arrow[r] & ? \arrow[r] & A \arrow[r] & ? \arrow[r] & A \arrow[r] & \cdots \end{tikzcd}$$

where "?" means that the value of A can be either 0 or 1. It may seem that this property is expressed by the formula $A \land \Box(A \to \bigcirc\bigcirc A)$. On the one hand, this formula implies that A is true at every even state. On the other hand, this formula is false on the following path, on which A is true at every even state:

$$\begin{tikzcd} A \arrow[r] & \mathrm{A} \arrow[r] & A \arrow[r] & \neg A \arrow[r] & A \arrow[r] & \arrow[r] & A \arrow[r] & \cdots \end{tikzcd}$$

14.5 Equivalences of Temporal Formulas

In this section we will consider several equivalences between temporal formulas. Studying these equivalences will help us to understand the LTL operators.

Unwinding properties. These equivalences relate the value of a formula at a state to its value at the next state.

$$\begin{array}{rcl} \Diamond A & \equiv & A \lor \bigcirc\Diamond A; \\ \Box A & \equiv & A \land \bigcirc\Box A; \\ A\,\mathbf{U}\,B & \equiv & B \lor (A \land \bigcirc A\,\mathbf{U}\,B); \\ A\,\mathbf{R}\,B & \equiv & B \land (A \lor \bigcirc A\,\mathbf{R}\,B). \end{array}$$

These equivalences are immediate, see Figure 14.4.

Negation of temporal operators. These equivalences express the negations of temporal operators in terms of other operators. They show that there is a duality between $\Diamond$ and $\Box$ and a duality between $\mathbf{U}$ and $\mathbf{R}$.

$$\begin{array}{rcl}
\neg\bigcirc A & \equiv & \bigcirc\neg A; \\
\neg\Diamond A & \equiv & \Box\neg A; \\
\neg\Box A & \equiv & \Diamond\neg A; \\
\neg(A\,\mathbf{U}\,B) & \equiv & \neg A\,\mathbf{R}\,\neg B; \\
\neg(A\,\mathbf{R}\,B) & \equiv & \neg A\,\mathbf{U}\,\neg B.
\end{array}$$

Expressing operators through $\mathbf{U}$. Operators $\Diamond$, $\Box$, and $\mathbf{R}$ can be expressed through $\mathbf{U}$ as follows.

$$\begin{array}{rcl}
\Diamond A & \equiv & \top\,\mathbf{U}\,A; \\
\Box A & \equiv & \neg(\top\,\mathbf{U}\,\neg A); \\
A\,\mathbf{R}\,B & \equiv & \neg(\neg A\,\mathbf{U}\,\neg B).
\end{array}$$

This shows that LTL without the operators $\Diamond$, $\Box$, $\mathbf{R}$ has the same expressive power as LTL.

Exercises

EXERCISE 14.1 Formalize the following statements about the vending machine example in LTL.

(1) If the beer storage becomes empty, it gets recharged immediately.

(2) The beer storage becomes empty infi nitely many times.

(3) The recharge transaction occurs infi nitely often.

(4) Students never leave without a drink.

(5) Professors sometimes leave with a drink left in the dispenser.

(6) If a student forgets a coin in the coin slot, she (or another student) will use this coin to get a drink before any professor can do this.

(7) If a professor forgets coins or drink at the machine, there immediately will be a student who will come to the machine.

(8) If, when a professor arrives, there is a coin in the coin slot, then he leaves without getting a drink.

(9) If a professor is currently at the machine, there will be no student at the machine for at least the next three transitions. ❏

EXERCISE 14.2 Express in LTL the following properties (we assume a path $s_0, s_1, \ldots$).

(1) If A occurs at least twice, then A occurs infi nitely often.

(2) A holds at all states s_{3k} and does not hold at all states s_{3k+1}, s_{3k+2}, where $k = 0, 1, \ldots$.

(3) If A holds at a state s_i, then B must holds at at least one of the two states just before s_i, that is s_{i-1} and s_{i-2}.

(4) A never holds at less than two consecutive states (that is, if A holds at a state s_i, it also holds either at the state s_{i+1} or at the state s_{i-1}). ❏

EXERCISE 14.3 What are the properties expressed by the following LTL formulas?

(1) $\Diamond \Box A$.

(2) $\Box(A \to \bigcirc A)$.

(3) $\neg A \,\mathbf{U}\, \Box A$.

(4) $A \,\mathbf{U} \neg A$.

(5) $\Diamond A \land \Box(A \to \bigcirc A)$. ❏

EXERCISE 14.4 Find two different formulas from Exercise 14.3 that are equivalent to each other. ❏

EXERCISE 14.5 Show that the following formulas are not equivalent by giving a path that satisfi es one of them but does not satisfy the other one:

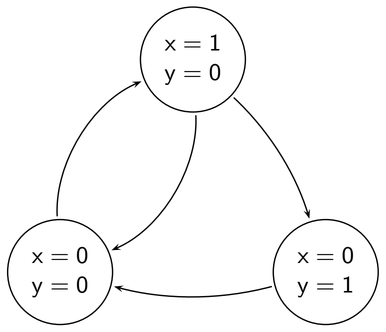
(1) $\Diamond \Box A$ and $\Box(A \to \bigcirc A)$;

(2) $\Diamond \Box A$ and $\neg A \,\mathbf{U}\, \Box A$;

(3) $\Box(A \to \bigcirc A)$ and $\neg A \,\mathbf{U}\, \Box A$. ❏

EXERCISE 14.6 Which of the formulas of Exercise 14.1 hold on every computation path from the initial state for the vending machine example? Which of them hold on some computation paths? ❏

EXERCISE 14.7 Consider a Kripke structure with the following state transition graph.



The initial state is the top one. Is the formula $\Box(\mathsf{x} = 1 \to \Diamond \mathsf{y} = 1)$ true along all paths? Explain your answer. ❏