

## Chapter 12

# Propositional Logic of Finite Domains

A googolplex is much larger than a googol, but is still finite, as the inventor of the name was quick to point out. It was first suggested that a googolplex should be 1, followed by writing zeros until you got tired. This is a description of what would happen if one actually tried to write a googolplex, but different people get tired at different times and it would never do to have Carnera a better mathematician than Dr Einstein, simply because he had more endurance. The googolplex then, is a specific finite number, with so many zeros after the 1 that the number of zeros is a googol. A googolplex is much bigger than a googol, much bigger even than a googol times a googol. A googol times a googol would be 1 with 200 zeros, whereas a googolplex is 1 with a googol of zeros. You will get some idea of the size of this very large but finite number from the fact that there would not be enough room to write it, if you went to the farthest star, touring all the nebulae and putting down zeros every inch of the way.

*Edward Kasner, James R. Newman. Mathematics and the Imagination. Penguin. 1940*

### Contents

---

<b>12.1 Syntax and Semantics</b> . . . . .	<b>178</b>
12.1.1 Motivation . . . . .	178
12.1.2 Syntax . . . . .	179
12.1.3 Semantics . . . . .	180
<b>12.2 PLFD and Propositional Logic</b> . . . . .	<b>181</b>
12.2.1 Propositional Logic as an Instance of PLFD . . . . .	181
12.2.2 Translation of PLFD to Propositional Logic . . . . .	181
<b>12.3 A Tableau System for PLFD</b> . . . . .	<b>183</b>
<b>12.4 Using Boolean Variables in PLFD</b> . . . . .	<b>186</b>

---

Exercises . . . . .	188
---------------------	-----

---

In this chapter we introduce *propositional logic of finite domains* (PLFD). This logic is convenient for specifying properties of systems whose state can be described by values of variables, so that every variable has a finite set of possible values. Propositional logic can be regarded as a special case of PLFD in which every variable ranges over the set of boolean values.

In Section 12.1 we define the syntax and semantics of PLFD. The only difference between the syntax of PLFD and propositional logic is the definition of atomic formulas. To show the similarity of PLFD and propositional logic, in Section 12.2 we give model-preserving translations between formulas of PLFD and propositional logic. Using this translation one can, in principle, work with PLFD using propositional logic. For example, to find a model of a PLFD formula, one can translate it into a propositional formula and search for models of the translated formula using any propositional logic method, such as DPLL. However, one can also modify these methods to work directly with PLFD. We illustrate this by defining a tableau system for PLFD in Section 12.3.

## 12.1 Syntax and Semantics

### 12.1.1 Motivation

Consider a microwave oven (I will actually use the one in my kitchen as an example). It has several *modes of operation*, such as micro power, convection, grill, combination, defrost etc., a *door* that can open or closed, a *temperature display*, and a *timer*. It has several buttons that can be used to control the operation of the microwave. It may be turned on or off. More complex models may include more advanced features.

If you are a designer of such a system as this microwave oven, you should write a program controlling its functioning. Your controlling program should satisfy some important conditions, for example, if the door is open, then the mode should be idle. These conditions may be related to *safety* of the device but also to other properties. To express these conditions, we can use logic.

If we try to use propositional logic for modeling the microwave, we will soon find out that it is not very natural. Consider, for example, the mode of operation. It has several possible values (grill, idle etc.), so we should be able to express properties such as that the mode is idle. If we introduce a boolean variable *idle* to express that the mode is idle and a boolean variable *grill* to express that the mode is grill, then these variables will turn out not to be independent: indeed, in our model *idle* and *grill* cannot be true at the same time. But propositional logic has interpretations in which both *idle* and *grill* are true, so we have to add axioms like  $\neg idle \vee \neg grill$  to model the microwave adequately.

It seems more natural to have a logic in which we can express properties of the microwave more directly than by introducing many boolean variables to express every possible value for the mode. Of course we do not want to have exactly the microwave logic, we

variable	domain of values
mode	{ <i>idle, micro, grill, defrost</i> }
door	{ <i>open, closed</i> }
content	{ <i>none, burger, pizza, cabbage</i> }
user	{ <i>nobody, student, veggie, mcdonald</i> }
temperature	{0, 150, 160, 170, 180, 190, 200, 210, 220, 230, 240, 250}

Figure 12.1: Variables and Domains

want to be able to describe formally various devices, from very simple ones like a vending machine, to very complex ones like a Mars rover. What is common in such devices is that their state can be characterized by values of a set of *variables*. For example, mode can be considered a variable and *idle* a possible value for this variable. Another variable for the microwave example is door whose possible values are *open* and *closed*.

Propositional logic of finite domains, or simply PLFD, is a logic in which atomic formulas express properties of the form “a variable  $x$  has a value  $v$ ”, written in the form  $x = v$ . For example, we can write  $\text{door} = \textit{open} \rightarrow \text{mode} = \textit{idle}$  to express the safety policy that the microwave should not be functioning when the door is open.

The set of possible variables and values depend on what kind of property we would like to express. For example, if we are mostly interested in how a microwave is functioning in a departmental tea room, we may introduce variables denoting users, queue, and content of the microwave.

In any case, before expressing properties of the microwave, we should fix a collection of variables describing it and possible values for each variable. For example, Figure 12.1 shows a set of variables representing both the microwave, its users and the content.

### 12.1.2 Syntax

*Propositional logic of finite domains*, or simply *PLFD* is, in fact, a *family of logics*. Each instance of this family uses a finite set  $\mathcal{X}$  of *variables* and a finite set of *domains*. Each domain is a finite non-empty set. Elements of domains are called *values*. Each variable  $x \in \mathcal{X}$  has an associated domain, denoted by  $\text{dom}(x)$ . We will say that the variable  $x$  *ranges over* the domain  $\text{dom}(x)$ , or that  $\text{dom}(x)$  is the *domain for*  $x$ .

For example, we can regard the table of Figure 12.1 as representing an instance of PLFD. The variables of this logic are shown in the left column of the table, while the domain for each variable is shown in the right column.

DEFINITION 12.1 (Formula) *Formulas* are defined inductively as follows.

- (1) If  $x$  is a variable and  $v \in \text{dom}(x)$  is a value in the domain for  $x$ , then  $x = v$  is a formula, also called *atomic formula*, or simply *atom*.

- (2) Other formulas are build from atomic formulas as in propositional logic, see Definition 3.1, using the connectives  $\top$ ,  $\perp$ ,  $\wedge$ ,  $\vee$ ,  $\neg$ ,  $\rightarrow$ , and  $\leftrightarrow$ .  $\square$

For example, assuming variables and domains as in Figure 12.1, the following is a formula of PLFD:

$$\text{mode} = \text{grill} \rightarrow \text{door} = \text{closed} \wedge \neg \text{temperature} = 0 \wedge \neg \text{user} = \text{nobody}.$$

The expression  $\text{user} = \text{none}$  is *not* a formula, since *none* is not a value in the domain for user. When we would like to be specific about which instance of PLFD we are dealing with, we will write  $PLFD(\mathcal{X}, dom)$  to refer to the instance with the set of variables  $\mathcal{X}$  and domain function  $dom$ .

### 12.1.3 Semantics

The semantics of PLFD is defined similar to that of propositional logic. We define a suitable notion of interpretation and the meaning of atomic formulas in an interpretation. The meaning of arbitrary formulas can then be defined from the meaning of atomic ones in exactly the same way as in propositional logic.

**DEFINITION 12.2 (Interpretation, Truth)** An *interpretation* for a set of variables  $X$  is a mapping  $I$  from  $X$  to the set of values such that for all  $x \in X$  we have  $I(x) \in dom(x)$ .

We will now extend interpretations to mappings from formulas to boolean values. The definition is by induction.

- (1)  $I(x = v) = 1$  if and only if  $I(x) = v$ .
- (2) If  $F$  is not atomic, then  $I(F)$  is defined as in Definition 3.2 for propositional formulas.

The definitions of *truth*, *models*, *validity*, *satisfiability*, and *equivalence* are defined exactly as in propositional logic.  $\square$

For example, given variables and their domains as in Figure 12.1, the following is an interpretation:

$$\{\text{mode} \mapsto \text{micro}, \text{door} \mapsto \text{open}, \text{content} \mapsto \text{burger}, \\ \text{user} \mapsto \text{veggie}, \text{temperature} \mapsto 0\}.$$

The formula  $\neg \text{mode} = \text{idle} \rightarrow \text{door} = \text{closed}$  is false in this interpretation.

Note that the definitions of the syntax and semantics for PLFD are *parametrized* by the domain function  $dom$ . That is, different domain functions give us different instances of PLFD. Some of the properties of PLFD essentially depend on the concrete instance. For example, the formula  $\text{door} = \text{open} \vee \text{door} = \text{closed}$  is valid in any logic in which  $dom(\text{door}) = \{\text{open}, \text{closed}\}$  but not valid if the domain for door contains more than two values.

## 12.2 PLFD and Propositional Logic

In this section show that propositional logic is, in fact, an instance of PLFD. We also give a model-preserving translation of PLFD into propositional logic.

### 12.2.1 Propositional Logic as an Instance of PLFD

Consider an instance of PLFD in which every variable  $p_i$  ranges over the domain  $\{0, 1\}$  of boolean values. Note that propositional logic and this instance of PLFD have the same set of interpretations: interpretations in both logics map variables to boolean values. In this section we will give a *model-preserving translation* of propositional logic to PLFD. This translation preserves models in the following sense: it translates any formula  $F$  of propositional logic into a formula  $F'$  of PLFD such that for every interpretation  $I$ , we have  $I \models F$  in propositional logic if and only if  $I \models F'$  in PLFD.

This translation will be defined as a *mapping*  $plfd$  from formulas of propositional logic to formulas in this instance of PLFD as follows. For every propositional formula  $F$ , the formula  $plfd(F)$  is obtained from  $F$  by replacing every propositional atomic formula  $p$  by the PLFD atomic formula  $p = 1$ . For example, the formula  $plfd(p \rightarrow \neg q)$  is  $(p = 1) \rightarrow \neg(q = 1)$

**THEOREM 12.3** *Formulas  $F$  and  $plfd(F)$  have the same models.*

**PROOF.** We prove it in the case when  $F$  is an atomic formula  $p$ , the proof for arbitrary formulas carries over by straightforward induction. Let  $I$  be a mapping from variables to boolean values. In propositional logic we have  $I \models p$  if and only if  $I(p) = 1$ . But in PLFD we have  $I \models (p = 1)$  if and only if  $I(p) = 1$ . Therefore,  $I \models p$  in propositional logic if and only if  $I \models (p = 1)$  in PLFD.  $\square$

### 12.2.2 Translation of PLFD to Propositional Logic

If we take an arbitrary instance of PLFD which has a non-boolean variable, then the set of interpretations for this instance is different from the set of interpretations for propositional logic with the same set of variables. In this case there is no model-preserving translation between the two logics. Nonetheless, one can define a translation that preserves a suitable mapping between models and use this translation for finding models and checking satisfiability and other properties of PLFD formulas.

The idea of translation is to introduce boolean variables denoting atomic formulas  $x = v$ . To this end, for every variable  $x$  and value  $v \in dom(x)$  we use a boolean variable  $x_v$ . After that, given a PLFD formula  $F$  we can replace every atom  $x = v$  in  $F$  by the propositional atom  $x_v$ , obtaining a propositional formula  $F'$ . This transformation does not directly preserve satisfiability. For example, for the microwave example the formula  $mode = idle \wedge mode = micro$  is unsatisfiable, while the corresponding propositional formula  $mode_{idle} \wedge mode_{micro}$  is satisfiable. The problem is that the propositional formula has

a model  $\{\text{mode}_{idle} \mapsto 1, \text{mode}_{micro} \mapsto 1\}$  that does not correspond to any PLFD interpretation: indeed, in every such interpretation the variable  $\text{mode}$  has exactly one value.

To provide a meaningful translation, one has to add additional propositional formulas which prevent unintended interpretations from becoming models. These formulas axiomatize properties of domains for variables, for example one can add the formula  $\neg \text{mode}_{idle} \vee \neg \text{mode}_{micro}$  to exclude models in which both  $\text{mode}_{idle}$  and  $\text{mode}_{micro}$  are true.

**DEFINITION 12.4 (Domain Axiom)** Let  $x$  be a variable and  $\text{dom}(x) = \{v_1, \dots, v_n\}$ . Then the *domain axiom* for  $x$  is the propositional formula

$$(x_{v_1} \vee \dots \vee x_{v_n}) \wedge \bigwedge_{i < j} (\neg x_{v_i} \vee \neg x_{v_j}). \quad \square$$

For instance, in the microwave example the domain axiom for the variable  $\text{mode}$  is

$$\begin{aligned} & (\text{mode}_{idle} \vee \text{mode}_{micro} \vee \text{mode}_{grill} \vee \text{mode}_{defrost}) \wedge \\ & (\neg \text{mode}_{idle} \vee \neg \text{mode}_{micro}) \wedge \\ & (\neg \text{mode}_{idle} \vee \neg \text{mode}_{grill}) \wedge \\ & (\neg \text{mode}_{idle} \vee \neg \text{mode}_{defrost}) \wedge \\ & (\neg \text{mode}_{micro} \vee \neg \text{mode}_{grill}) \wedge \\ & (\neg \text{mode}_{micro} \vee \neg \text{mode}_{defrost}) \wedge \\ & (\neg \text{mode}_{grill} \vee \neg \text{mode}_{defrost}). \end{aligned}$$

The domain axiom for the variable  $\text{temperature}$  is considerably longer and contains 144 occurrences of variables. The following lemma shows that the domain axiom indeed eliminates the unintended models.

**LEMMA 12.5** Let  $\text{dom}(x) = \{v_1, \dots, v_n\}$ . Let  $F$  be the domain axiom for  $x$  and  $I$  be an interpretation defined on the set of boolean variables  $X = \{x_{v_1}, \dots, x_{v_n}\}$ . Then  $I \models F$  if and only if exactly one of the boolean variables in  $X$  is satisfied in  $I$ .

**PROOF.** It is easy to see that the models of  $x_{v_1} \vee \dots \vee x_{v_n}$  are the interpretations satisfying *at least* one variable in  $X$ . The models of  $\neg x_{v_i} \vee \neg x_{v_j}$  are the interpretations satisfying *at most* one variable among  $x_{v_i}, \neg x_{v_j}$ . Therefore, the models of the domain axiom are the interpretations satisfying *exactly* one variable in  $X$ .  $\square$

Now we will define formally in what respect an interpretation for PLFD corresponds to a model of the domain axioms for this instance of PLFD. To this end we introduce a mapping from interpretations for PLFD to propositional interpretations, denoted by  $\text{prop}$ . Given an interpretation  $I$  for PLFD, define  $\text{prop}(I)$  as follows:

$$\text{prop}(I)(x_v) = 1 \stackrel{\text{def}}{=} I(x) = v.$$

In other words,  $x_v$  is true in  $\text{prop}(I)$  if and only if the value of  $x$  in  $I$  is  $v$ . We also define a mapping from formulas of PLFD to propositional formulas, also denoted by  $\text{prop}$ , as

follows. Let  $F$  be a formula of PLFD with variables  $x_1, \dots, x_n$  and  $F_1, \dots, F_n$  be the domain axioms for  $x_1, \dots, x_n$ . Let a propositional formula  $F'$  be obtained from  $F$  by replacing every atom of the form  $x = v$  by the propositional atom  $x_v$ . Then  $\text{prop}(F)$  is defined to be the formula  $F_1 \wedge \dots \wedge F_n \wedge F'$ .

**THEOREM 12.6** *Let  $F$  be a formula of PLFD. Then*

- (1) *For every interpretation  $I$ ,  $I \models F$  if and only if  $\text{prop}(I) \models \text{prop}(F)$ .*
- (2) *Each model of  $\text{prop}(F)$  has the form  $\text{prop}(I)$  for some interpretation  $I$  of PLFD.*

**PROOF.** The first property is proved by induction on  $F$ . When  $F$  is an atomic formula  $x = v$ , the claim is immediate by the definition of  $\text{prop}(I)$ . For non-atomic formulas the induction arguments are straightforward.

Let us prove the second property. Let  $I'$  be a model of  $\text{prop}(F)$ . Then it is also a model of the domain axioms for all variables of the logic. Define an interpretation  $I$  of PLFD as follows. For every variable  $x$ , let  $I(x) \stackrel{\text{def}}{=} v$ , where  $v$  is such that  $I' \models x_v$ . By Lemma 12.5, for every variable  $x$  the interpretation  $I'$  satisfies exactly one atom of the form  $x_v$ , so  $I$  is well-defined. It is not hard to argue that  $\text{prop}(I) = I'$ .  $\square$

The translations from propositional logic to PLFD and back allow to shift forth and back between the two logics. PLFD is convenient for *specifying* or *modeling* systems, while propositional logic is simpler for defining reasoning algorithms. Therefore, in the future we will sometimes use PLFD for modeling systems but switch to propositional logic when it comes to reasoning about their properties.

### 12.3 A Tableau System for PLFD

We are interested in checking satisfiability, validity, and equivalence for PLFD formulas. This can be done by defining a suitable proof system for PLFD. In this section we will define a tableau system for this logic which can be obtained by extending the tableau proof system for propositional logic to PLFD. To define the system, we will first introduce a different kind of atomic formulas.

Let  $x$  be a variable and  $D \subseteq \text{dom}(x)$ . Then we consider expressions  $x \in D$  as atomic formulas with the following straightforward semantics: for an interpretation  $I$  we have  $I \models x \in D$  if and only if  $I(x) \in D$ . It is not hard to argue that any formula  $x = v$  is equivalent to the formula  $x \in \{v\}$ , and any formula  $x \in \{v_1, \dots, v_n\}$  is equivalent to  $x = v_1 \vee \dots \vee x = v_n$ , so the logics with atoms of the form  $x = a$  and atoms of the form  $x \in D$  have the same expressive power.

The *tableau system for PLFD* operates on signed formulas and uses the same branch expansion rules as in the case of propositional logic, see Figure 9.2 on page 120. In addition to these rules, it also uses several rules specific for the treatment of atomic formulas  $x \in D$ . Before defining the new rules, let us use the following convention: a signed formula ( $x \in$

$$\boxed{\begin{array}{l} x \notin D \rightsquigarrow x \in \text{dom}(x) \setminus D \\ x \in D_1, x \in D_2 \rightsquigarrow x \in D_1 \cap D_2 \end{array}}$$

Figure 12.2: Additional tableau rules for PLFD

$D) = 1$  will simply be denoted by  $x \in D$ , likewise, a signed formula  $(x \in D) = 0$  will be denoted by  $x \notin D$ .

The additional tableau rules are given in Figure 12.2. Note that the second rule requires two formulas  $x \in D_1$  and  $x \in D_2$  to be on the same branch.

A branch in a tableau is called *closed* if it contains a signed formula of one of the following kinds:  $\top = 0$ ,  $\perp = 1$ , or  $x \in \{\}$ .

EXAMPLE 12.7 Let us show that the following formula  $F$  is valid in the PLFD with variables and domains as in Figure 12.1:

$$\begin{aligned} & ((\text{user} = \text{mcdonald} \rightarrow \text{content} = \text{none} \vee \text{content} = \text{burger}) \wedge \\ & (\text{user} = \text{veggie} \rightarrow \text{content} = \text{none} \vee \text{content} = \text{cabbage}) \wedge \\ & (\text{user} = \text{nobody} \rightarrow \text{content} = \text{none})) \rightarrow \\ & (\text{content} = \text{pizza} \rightarrow \text{user} = \text{student}). \end{aligned}$$

To this end we show that the signed formula  $F = 0$  is unsatisfiable. To apply the tableau algorithm for this signed formula we replace all atoms of the form  $x = v$  by atoms  $x \in \{v\}$  and then try to build a tableau for it. A part of a semantic tableau for this signed formula is shown in Figure 12.3. All branches of the full tableau are closed, hence the signed formula  $F = 0$  is unsatisfiable, so  $F$  is valid.  $\square$

THEOREM 12.8 (Soundness and Completeness) *Let  $T$  be a tableau resulting from a terminated game for a signed formula  $\gamma$ . Then  $\gamma$  is satisfiable if and only if all branches in  $T$  are closed.*

PROOF. We will use the arguments used in the proof of Theorem 9.17. Namely, we prove that every tableau inference is invertible and that a tableau to which no rule is applicable is satisfiable if and only if it is non-empty.

Invertibility of the tableau rules used in propositional logic is established in Lemma 9.16, so it is enough to establish invertibility of the rules of Figure 12.2 and the branch closure rules. Invertibility of all of these rules is obvious, for example, it is obvious that  $I \models x \in D_1$  and  $I \models x \in D_2$  if and only if  $I \models x \in D_1 \cap D_2$ .

It remains to prove that a non-empty tableau to which no rule is applicable is satisfiable. Take any such tableau and consider any branch  $B$  in it. It is not hard to argue that the branch has the form

$$\{x_1 \in D_1, \dots, x_n \in D_n\},$$

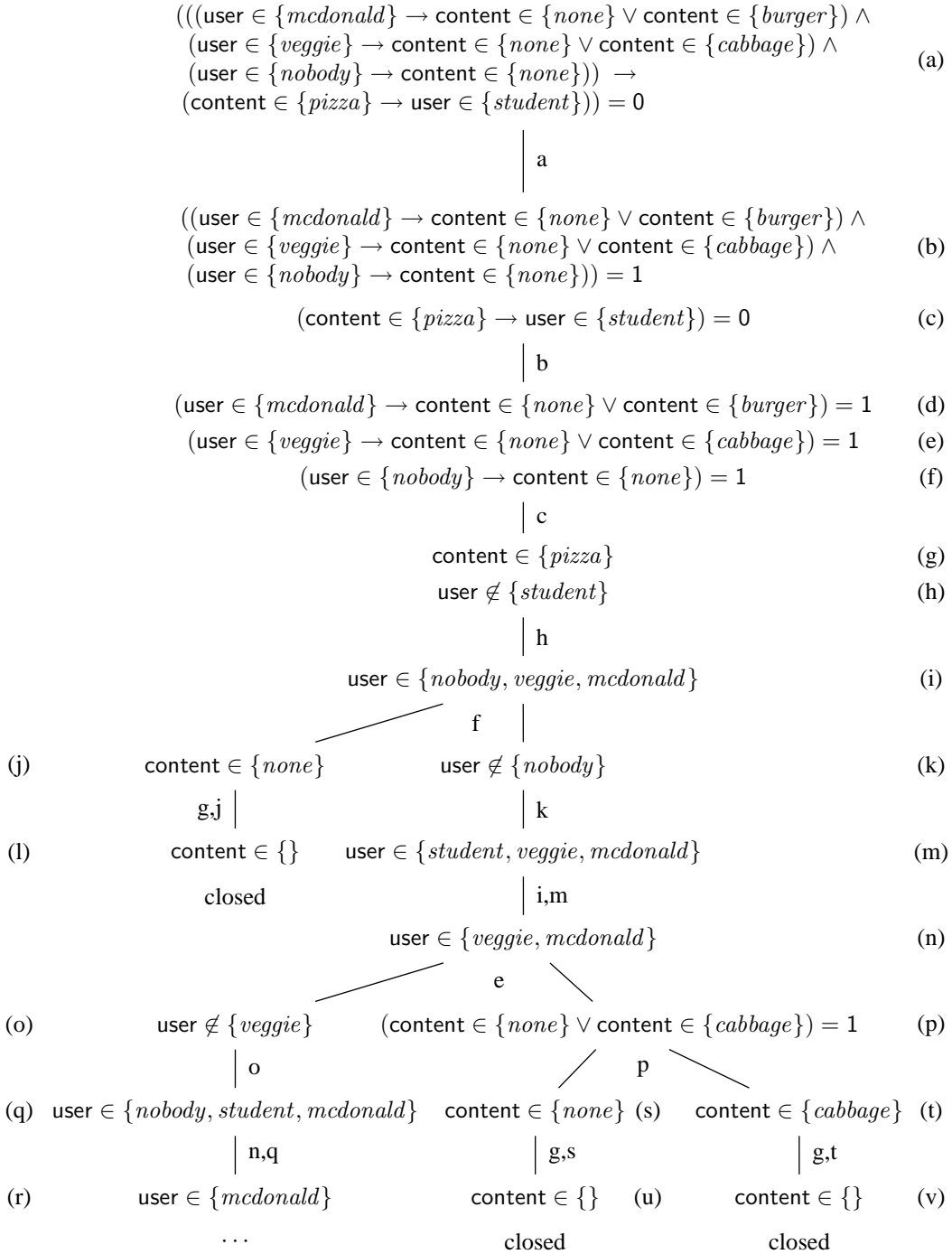


Figure 12.3: Part of a semantic tableau for the formula of Example 12.7.

where  $x_1, \dots, x_n$  are distinct variables and  $D_1, \dots, D_n$  are non-empty sets, moreover  $D_i \in \text{dom}(x_i)$  for all  $i$ . Since all  $D_i$  are non-empty, there exists an interpretation  $I$  such that  $I(x_i) = v_i$  for all  $i$ . It is not hard to argue that  $I$  is a model of this branch.  $\square$

The proof of this theorem shows how one can extract models from an open branch of a tableau. For every variable  $x$ , if the branch contains a signed formula  $x \in D$  to which no rule has been applied, then we can assign to  $x$  any value in  $D$ . If the branch contains no such formula, then we can assign to  $x$  any value in its domain. We will illustrate this in Example 12.9 below.

## 12.4 Using Boolean Variables in PLFD

When some variable  $x$  is boolean, that is, has the domain  $\{0, 1\}$  using the PLFD atomic formulas  $x = 0$  and  $x = 1$  for it may lead to unnecessarily complex formulas. To use boolean variables in PLFD one can adopt the following syntactic convention: if  $x$  is a boolean variable, we will use the propositional logic syntax for this variable. Namely, will use  $x$  as an atomic formula and write  $x$  instead of  $x = 1$  and  $\neg x$  instead of  $x = 0$ . For instance, if we change the variable `door` in the microwave example by a boolean variable `door_open` with an obvious meaning, then instead of

$$\text{mode} = \text{grill} \rightarrow \text{door\_open} = \text{false} \wedge \neg \text{temperature} = 0 \wedge \neg \text{user} = \text{nobody}.$$

we will write

$$\text{mode} = \text{grill} \rightarrow \neg \text{door\_open} \wedge \neg \text{temperature} = 0 \wedge \neg \text{user} = \text{nobody}.$$

This convention is especially convenient when there are many boolean variables and only few non-boolean ones, as it is often the case in applications. We can also modify the tableau system for PLFD to use boolean variables in the same way as they are used in the tableau system for propositional logic. That is, we use atomic formulas  $x \in D$  for every non-boolean variable  $x$  and atomic formulas  $x$  for every boolean variable  $x$ . A tableau branch is closed in one of the following three cases:

- (1) it contains the signed formula  $\top = 0$  or  $\perp = 1$ ;
- (2) it contains the signed formula  $x \in \{\}$  for a non-boolean variable  $x$ ;
- (3) it contains signed formulas  $x = 0$  and  $x = 1$  for a boolean variable  $x$ .

Let us demonstrate such a mixed tableau by a small but practical example.

**EXAMPLE 12.9** There have been a lot of discussion whether a recent war was justified. A textbook on computational logic is hardly an appropriate place for expressing an opinion on this complex issue. However, we can investigate the arguments used for justification from a logical viewpoint. The question we ask is whether it is possible to start a war against a country that is not guilty. We will use the following variables:

- (1) war: one can start a war;
- (2) guilty: the country is guilty;
- (3) has: the country has weapons of mass destruction.

As far as the author could understand, the following arguments were used for the justification. First, if a country has weapons of mass destruction, then it is guilty. Second, to start a war against the country one has to have a very good reason, possessing weapons of mass destruction is such a reason. This gives us two formulas:

$$\begin{aligned} \text{has} &\rightarrow \text{guilty}, \\ \text{war} &\rightarrow \text{has}. \end{aligned}$$

If we would like to check whether, under the above assumptions, it is possible that a war started against a country that is not guilty, then the answer is “no”. Indeed, the set of formulas

$$\begin{aligned} \text{has} &\rightarrow \text{guilty}, \\ \text{war} &\rightarrow \text{has}, \\ \text{war}, \\ \neg \text{guilty} \end{aligned}$$

is unsatisfiable.

Let us consider a slightly different situation, when the domain for the variable has consists of the values *yes*, *no*, and a third value, for example, *suspected*. We can reformulate our assumptions in the following way

$$\begin{aligned} \text{has} = \text{yes} &\rightarrow \text{guilty}, \\ \text{war} &\rightarrow \neg(\text{has} = \text{no}). \end{aligned}$$

In other words, if a country has weapons of mass destruction, then it is guilty, and to start a war we have to be sure that it is impossible that the country has no weapons of mass destruction.

If we ask now whether it is a possible that a war started against a country that is not guilty, then the answer is “yes”. Indeed, the set of formulas

$$\begin{aligned} \text{has} = \text{yes} &\rightarrow \text{guilty}, \\ \text{war} &\rightarrow \neg(\text{has} = \text{no}), \\ \text{war}, \\ \neg \text{guilty} \end{aligned}$$

is satisfiable.

To find a model of this set, we build a semantic tableaux, see Figure 12.4. This tableau has an open branch, which gives us the only model of this set of formulas:

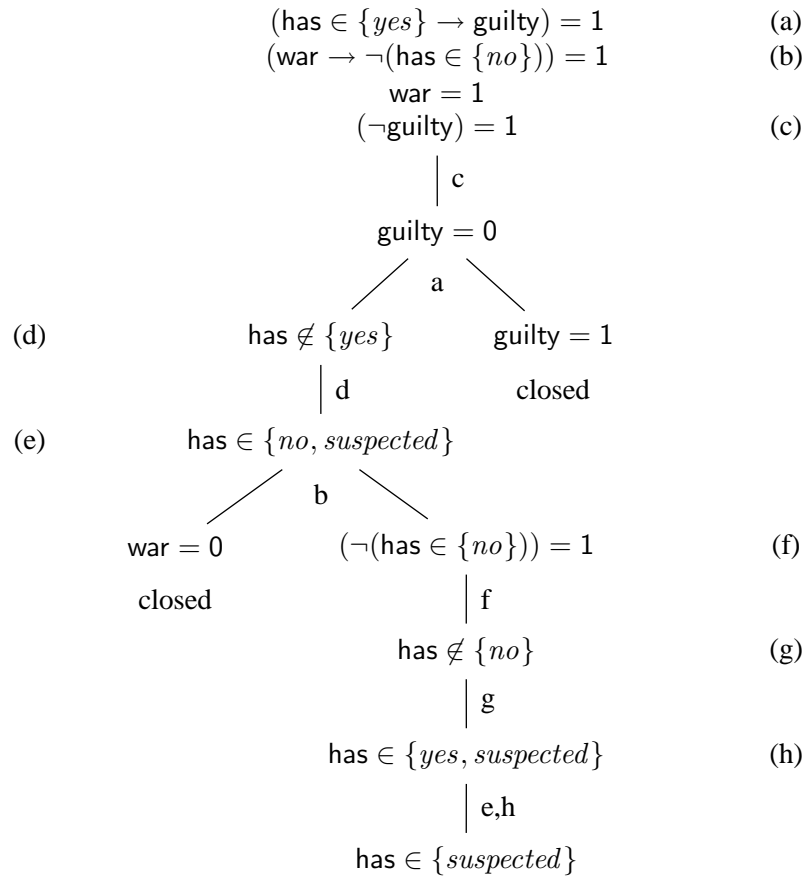


Figure 12.4: A semantic tableau for PLFD with boolean variables

$$\{\text{war} \mapsto 1, \text{guilty} \mapsto 0, \text{has} \mapsto \text{suspected}\},$$

that is, it is possible that the war started, the country is not guilty but suspected to have weapons of mass destruction. □

### Exercises

EXERCISE 12.1 Complete the tableau of Figure 12.3. □

EXERCISE 12.2 Write down the domain axiom for the variable user in the microwave example. □

EXERCISE 12.3 Is the domain axiom a formula in CNF or not? □

EXERCISE 12.4 What is the size (measured as the number of occurrences of variables) of the domain axiom for a variable whose domain contains 1000 values? □

EXERCISE 12.5 Take the domain axiom for a variable whose domain contains 1000 values and transform into CNF using the standard CNF transformation. What is the number of clauses in the resulting CNF? □

EXERCISE 12.6 Let  $m, n$  be positive integers and  $m < n$ . Express in propositional logic the following property: exactly  $m$  variables among  $x_1, \dots, x_n$  are true. □

EXERCISE 12.7 Show, using the tableau method, that the formula

$$\neg((\neg \text{content} = \text{burger} \wedge \neg \text{content} = \text{pizza}) \rightarrow \text{content} = \text{cabbage} \vee \text{content} = \text{none})$$

is unsatisfiable in the logic for the microwave example. □

EXERCISE 12.8 Find, using the tableau method, a model of the formula

$$\neg((\neg \text{content} = \text{burger} \wedge \neg \text{content} = \text{pizza}) \rightarrow \text{content} = \text{none}). \quad \square$$

EXERCISE 12.9 What is the number of different models of the formula  $\neg \text{content} = \text{none}$  in the logic for the microwave example? □

EXERCISE 12.10 Consider a formula  $x \in D$  used in semantic tableaux.

- (1) For which  $D$  is  $x \in D$  a tautology?
- (2) For which  $D$  is  $x \in D$  unsatisfiable? □

EXERCISE 12.11 Transform the propositional formula  $p_1 \rightarrow (p_2 \wedge p_3)$  into PLFD. □

EXERCISE 12.12 Transform each of the following formulas of PLFD for the microwave example into propositional logic:

$$\begin{aligned} \neg \text{mode} = \text{defrost}; \\ \text{user} = \text{nobody} \rightarrow \text{mode} = \text{idle}. \end{aligned} \quad \square$$

EXERCISE 12.13 Let  $x$  be a variable with the domain  $\{u, v, w\}$  and  $p$  be a boolean variable. Transform the following formula of PLFD into a propositional formula: □

$$\neg x = v \rightarrow x = u \wedge p = 0.$$

EXERCISE 12.14 Show that the formulas  $\text{door} = \text{closed} \rightarrow \text{mode} = \text{micro}$  and  $\neg \text{mode} = \text{idle} \rightarrow \neg \text{door} = \text{closed}$  are not equivalent in the PLFD for the microwave example by finding an interpretation in which they have different truth values. □

EXERCISE 12.15 Let  $x$  be a variable with the domain  $\{a, b, c\}$  and  $p$  be a boolean variable. Show, using the tableau method, that the following formula is unsatisfiable.

$$\neg((p \rightarrow \neg x = a) \rightarrow x = b \vee x = c \vee \neg p). \quad \square$$

EXERCISE 12.16 Take Exercise 12.15 but assume that the domain for  $x$  is  $\{a, b, c, d\}$ . Find, using the tableau method, a model of the formula of that exercise.  $\square$

# Chapter 14

## Linear Temporal Logic LTL

What then is time? If no one asks me, I know: if I wish to explain it to one that asketh, I know not.

*St. Augustine's Confessions*

A good deal of work in the philosophy of time has been produced by people worried about Fatalism, which can be understood as the thesis that whatever will happen in the future is already unavoidable (where to say that an event is unavoidable is to say that no human is able to prevent it from occurring).

*Stanford Encyclopedia of Philosophy*

### Contents

---

<b>14.1 Computation Trees</b> . . . . .	<b>209</b>
<b>14.2 LTL</b> . . . . .	<b>211</b>
<b>14.3 Expressing Properties of Transition Systems in LTL</b> . . . . .	<b>215</b>
<b>14.4 Equivalences of Temporal Formulas</b> . . . . .	<b>219</b>
<b>Exercises</b> . . . . .	<b>220</b>

---

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 Computation Trees

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

DEFINITION 14.1 (Computation Tree, Computation) Let  $\mathbb{S} = (S, In, T, \mathcal{X}, dom, L)$  be a transition system and  $s \in S$  be a state. The *computation tree for  $\mathbb{S}$  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  $\mathbb{S}$  is a sequence of nodes  $s_0, s_1, \dots$  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  $s_1, \dots, s_n$ , then there exists no state  $s$  such that  $(s_n, s) \in T$ .  $\square$

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 transition system are exactly all branches in the computation trees for this transition system.
- (2) Let  $n$  be a node in a computation tree  $C$  for  $\mathbb{S}$  labeled by  $s'$ . Then the subtree of  $C$  rooted at  $s'$  is the computation tree for  $\mathbb{S}$  starting at  $s'$ . In other words, every subtree of a computation tree rooted at some node is itself a computation tree.
- (3) For every transition system  $\mathbb{S}$  and state  $s$  there exists a unique computation tree for  $\mathbb{S}$  starting at  $s$ , up to the order of children.

For example, a part of a computation tree for the transition system of Example 13.3 is given in Figure 14.1 on the next page. Likewise, a part of a computation tree for the transition system of Example 13.5 is given in Figure 14.2 on page 212. 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 13.2 on page 200).

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

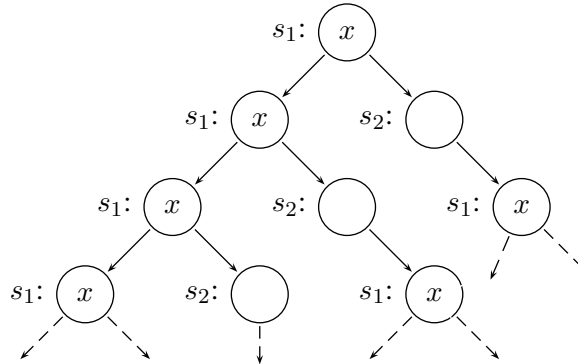


Figure 14.1: Computation tree for Example 13.3.

- (2) Students never drink coffee.

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

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  $s_1 \models \text{customer} = \text{student} \wedge \text{disp} = \text{coffee}$ , then  $s_2 \models \text{disp} = \text{coffee}$ .

## 14.2 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 14.2 (Formula of LTL)** The notion of an LTL *formula* is defined inductively as follows.

- (1)  $\top$  and  $\perp$  are formulas.
- (2) Every atomic formula  $x = v$  of PLFD is an (atomic) formula of LTL.
- (3) If  $A_1, \dots, A_n$  are formulas, where  $n \geq 2$ , then  $(A_1 \wedge \dots \wedge A_n)$  and  $(A_1 \vee \dots \vee A_n)$  are formulas.
- (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.

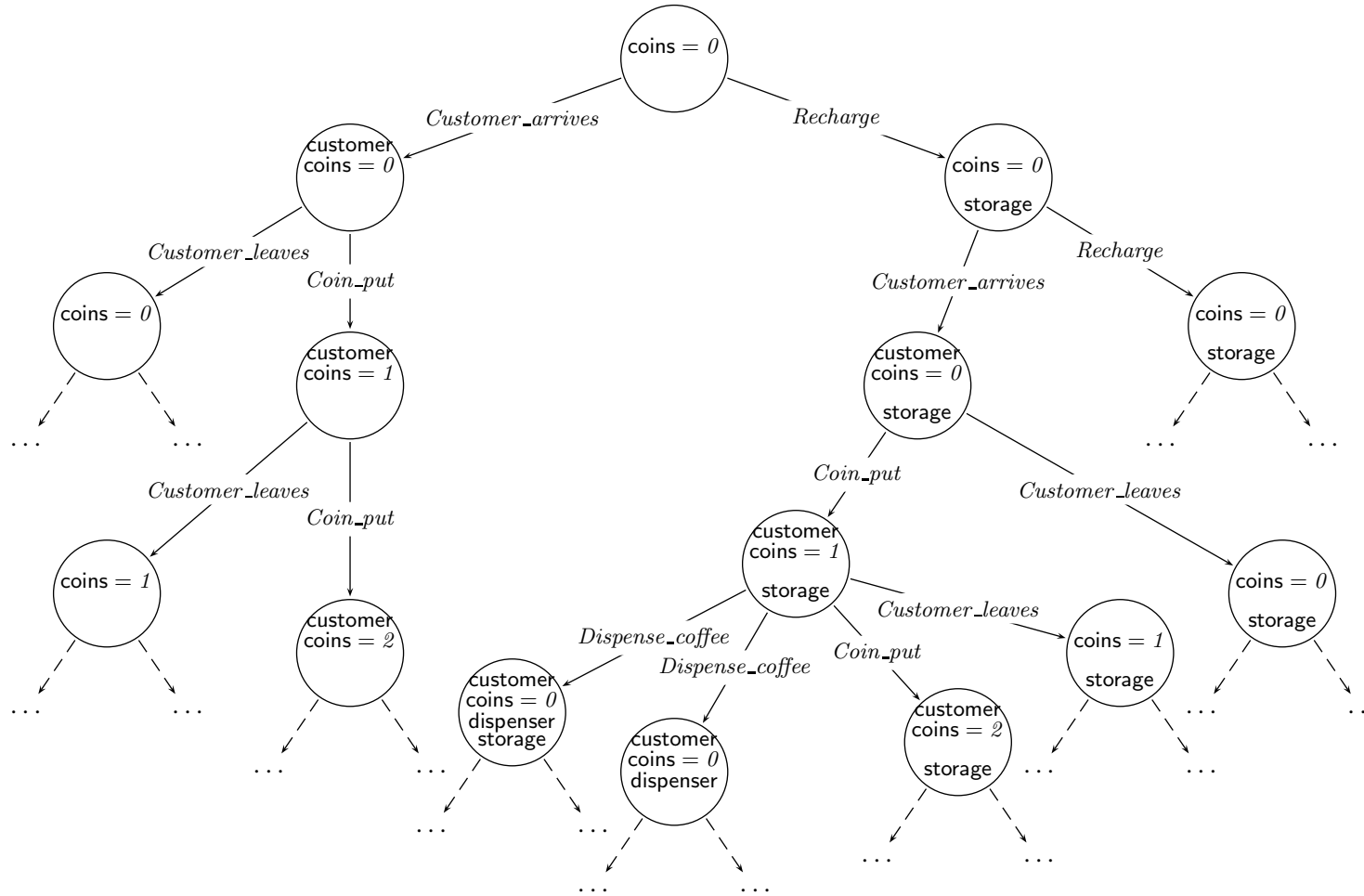


Figure 14.2: A computation tree for the transition system of Example 13.5

Connective	Name	Priority
$\top$	<i>verum</i>	
$\perp$	<i>falsum</i>	
$\neg$	<i>negation</i>	0
$\bigcirc$	<i>next</i>	0
$\square$	<i>always</i>	0
$\diamond$	<i>eventually</i>	0
$\mathbf{U}$	<i>until</i>	1
$\mathbf{R}$	<i>release</i>	1
$\wedge$	<i>conjunction</i>	2
$\vee$	<i>disjunction</i>	2
$\rightarrow$	<i>implication</i>	3
$\leftrightarrow$	<i>equivalence</i>	4

Figure 14.3: Connectives and Temporal Operators

- (6) If  $A$  is a formula, then  $\bigcirc A$ ,  $\diamond A$ , and  $\square A$  are formulas.
- (7) If  $A$  and  $B$  are formulas, then  $A \mathbf{U} B$  and  $A \mathbf{R} B$  are formulas.

The symbols  $\bigcirc$ ,  $\diamond$ ,  $\square$ ,  $\mathbf{U}$ ,  $\mathbf{R}$  are called *temporal operators*.  $\square$

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

The connectives and temporal operators of LTL are summarised in the table of Figure 14.3. We will omit parentheses in the LTL formulas according to the *priorities* given in this table: horizontal lines divide connectives and operators with different priorities.

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 of computation paths, that is, sequences of states  $s_0, s_1, \dots$ . The formula  $\square A$  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 14.3 (Semantics of LTL)** Let  $\pi = s_0, s_1, s_2 \dots$  be a sequence of states and  $A$  be an LTL formula. We define the notion  *$A$  is true on  $\pi$* , denoted by  $\pi \models A$ , by induction on  $A$  as follows. For all  $i = 0, 1, \dots$  denote by  $\pi_i$  the sequence of states  $s_i, s_{i+1}, s_{i+2} \dots$  (note that  $\pi_0 = \pi$ ).

- (1)  $\pi \models \top$  and  $\pi \not\models \perp$ .
- (2)  $\pi \models x = v$  if  $s_0 \models x = v$ .

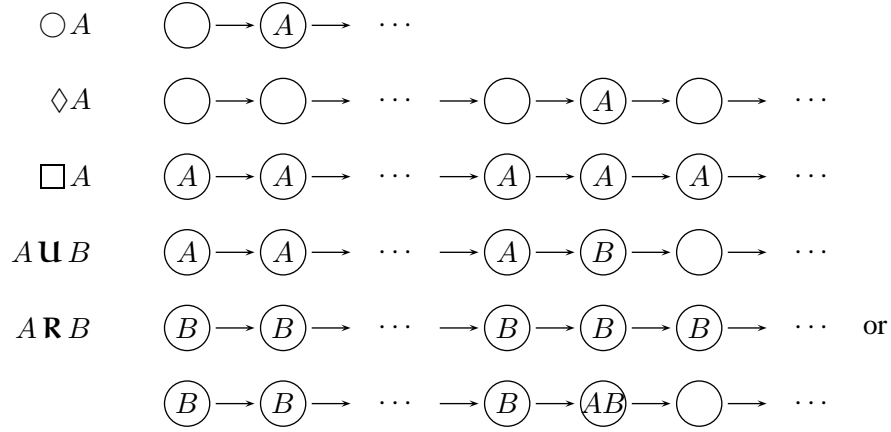


Figure 14.4: Semantics of temporal operators

- (3)  $\pi \models A_1 \wedge \dots \wedge A_n$  if for all  $j = 1, \dots, n$  we have  $\pi \models A_j$ ;  
 $\pi \models A_1 \vee \dots \vee A_n$  if for some  $j = 1, \dots, n$  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  $i = 0, 1, \dots$  we have  $\pi_i \models A$ ;  
 $\pi \models \square A$  if for all  $i = 0, 1, \dots$  we have  $\pi_i \models A$ .
- (7)  $\pi \models A \mathbf{U} B$  if for some  $k = 0, 1, \dots$  we have  $p_k \models B$  and  $p_0 \models A, \dots, p_{k-1} \models A$ ;  
 $\pi \models A \mathbf{R} B$  if for all  $k \geq 0$ , either  $\pi_k \models B$  or there exists  $j < k$  such that  $\pi_j \models A$ .  $\square$

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

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

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.

- ◇ (eventually) The formula  $\diamond A$  holds, if  $A$  eventually occurs, i.e.,  $A$  holds at some state on the path.
- (always) The formula  $\square A$  holds, if  $A$  holds globally, i.e., at every state along the path.
- 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.
- 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.

### 14.3 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 transition systems or transition systems. But we know that the set of all possible behaviors of a transition system is identified by its computation tree, so what is the meaning of these formulas when we discuss properties of transition systems? A computation tree for a transition system  $\mathbb{S}$  can be identified with the collection of its paths. We know that these paths describe all possible computations of this transition system. So if we have an LTL formula  $A$ , we can consider at least two kinds of properties of  $\mathbb{S}$ :

- (1) does  $A$  hold on *some* computation path for  $\mathbb{S}$  from an initial state?
- (2) does  $A$  hold on *all* computation paths for  $\mathbb{S}$  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 transition systems. 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  $\diamond \text{unsafe}$ . Then safety of the system can be expressed as non-reachability of a state

satisfying *unsafe*, i.e., the property  $\Box \neg \text{unsafe}$ . 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  $\text{disp} = \text{beer} \wedge \text{customer} = \text{prof}$ . Thus, the corresponding safety property is

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

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:

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

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:

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

**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  $\text{critical}_i$ , where  $i = 1, 2$  denote that  $P_i$  is in the critical section, mutual exclusion can be expressed by

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

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

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

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

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

**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 transition system 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 *terminal*, we can express deadlock-freedom by the formula

$$\Box(\bigcirc\perp \rightarrow \text{terminal}).$$

This formula must be true on every path. Indeed, it is easy to see that the formula  $\bigcirc\perp$  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

$$\Diamond(\bigcirc\perp \wedge \neg\text{terminal}).$$

**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  $\bigcirc\perp$ . A transition system or transition system is called *terminating*, if every computation in it leads to a terminal state. Termination for a transition system 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:  $\Diamond\bigcirc\perp$  (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  $\pi = s_0, s_1, \dots$ . Denote by  $\pi_j$  for  $j = 0, 1, \dots$  the path  $s_j, s_{j+1}, \dots$ . Let us prove the following property: for every  $j = 0, 1, \dots$  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  $j_0 < j_1 < j_2 < \dots$  such that  $\pi_{j_m} \models A$  for all  $m$ , so  $A$  occurs infinitely often on the path  $\pi$ .

In the proof that  $\Box\Diamond A$  expresses that  $A$  holds infinitely often we assumed that the path is infinite. When the path  $\pi$  is finite, it is not hard to argue that this property implies that  $A$  holds at the last state of  $\pi$ . We can ensure that the path is infinite by asserting  $\Box\bigcirc 1$ .

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  $\text{request}$  and acknowledgement by a formula  $\text{ack}$ , one can express responsiveness by the formula

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

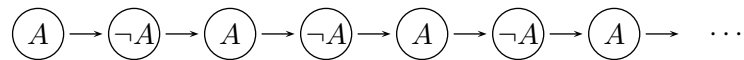
If we also want that  $\text{request}$  should remain true until it is acknowledged, responsiveness can be expressed by the formula

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

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

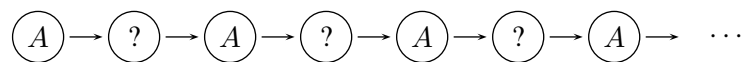
$$\Box(\text{request} \rightarrow ((\text{request} \wedge \neg\text{ack}) \mathbf{U} (\neg\text{request} \wedge \text{ack}))).$$

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

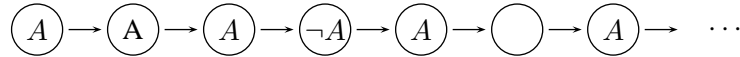


We will denote subpaths of  $\pi$  by  $\pi_0, \pi_1, \dots$  as before. Indeed, we have  $\pi_0 \models \Box(A \leftrightarrow \neg\bigcirc A)$ , which implies that for all  $i$ ,  $\pi_i \models A$  if and only if  $\pi_{i+1} \models \neg A$ . Therefore, the value of  $A$  changes from any state to 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:



where “?” means that the value of  $A$  can be either 0 or 1. It may seem that this property is expressed by the formula  $A \wedge \square(A \rightarrow \bigcirc\bigcirc A)$ . 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:



## 14.4 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.

$$\begin{aligned}\diamond A &\equiv A \vee \bigcirc \diamond A; \\ \square A &\equiv A \wedge \bigcirc \square A; \\ A \mathbf{U} B &\equiv B \vee (A \wedge \bigcirc A \mathbf{U} B); \\ A \mathbf{R} B &\equiv B \wedge (A \vee \bigcirc A \mathbf{R} B).\end{aligned}$$

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  $\square$  and a duality between  $\mathbf{U}$  and  $\mathbf{R}$ .

$$\begin{aligned}\neg \bigcirc A &\equiv \bigcirc \neg A; \\ \neg \diamond A &\equiv \square \neg A; \\ \neg \square A &\equiv \diamond \neg A; \\ \neg(A \mathbf{U} B) &\equiv \neg A \mathbf{R} \neg B; \\ \neg(A \mathbf{R} B) &\equiv \neg A \mathbf{U} \neg B.\end{aligned}$$

**Expressing operators through  $\mathbf{U}$ .** Operators  $\diamond$ ,  $\square$ , and  $\mathbf{R}$  can be expressed through  $\mathbf{U}$  as follows.

$$\begin{aligned}\diamond A &\equiv \top \mathbf{U} A; \\ \square A &\equiv \neg(\top \mathbf{U} \neg A); \\ A \mathbf{R} B &\equiv \neg(\neg A \mathbf{U} \neg B).\end{aligned}$$

This shows that LTL without the operators  $\diamond$ ,  $\square$ ,  $\mathbf{R}$  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 infinitely many times.
- (3) The recharge transaction occurs infinitely 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  $s_0, s_1, \dots$ ).

- (1) If  $A$  occurs at least twice, then  $A$  occurs infinitely often.
- (2)  $A$  holds at all states  $s_{3k}$  and does not hold at all states  $s_{3k+1}, s_{3k+2}$ , where  $k = 0, 1, \dots$
- (3) If  $A$  holds at a state  $s_i$ , then  $B$  must hold 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)  $\diamond \square A$ .
- (2)  $\square(A \rightarrow \bigcirc A)$ .
- (3)  $\neg A \mathbf{U} \square A$ .
- (4)  $A \mathbf{U} \neg A$ .
- (5)  $\diamond A \wedge \square(A \rightarrow \bigcirc A)$ . □

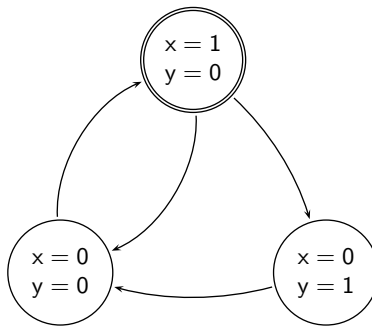
EXERCISE 14.4 Find two different formulas from Exercise 14.3 that are equivalent to each other. □

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

- (1)  $\diamond \Box A$  and  $\Box(A \rightarrow \bigcirc A)$ ;
- (2)  $\diamond \Box A$  and  $\neg A \mathbf{U} \Box A$ ;
- (3)  $\Box(A \rightarrow \bigcirc A)$  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 transition system with the following state transition graph.



Which of the following formulas are true on all paths?

- (1)  $\Box(x = 0 \vee y = 0)$ ;
- (2)  $\Box \diamond(y = 0)$ ;
- (3)  $\Box \diamond(y = 1)$ ;
- (4)  $\Box(x = 1 \rightarrow \diamond y = 1)$ .

Explain your answer. □

EXERCISE 14.8 For the transition system of Exercise 14.7, answer the following questions.

- (1) Does the formula  $x = 1$  symbolically represent the set of initial states?
- (2) Find a symbolic representation of the transition relation. □

EXERCISE 14.9 Let the formula  $x$  symbolically represent the set of initial states and the formula  $x \leftrightarrow \neg x'$  the transition relation of a transition system  $\mathbb{S}$  over  $\{x\}$ .

- (1) Draw the state transition graph of  $\mathbb{S}$ .
- (2) Draw a path for  $\mathbb{S}$ .
- (3) Which of the following formulas are true along all paths in  $\mathbb{S}$ ?
  - (a)  $\Box(x \leftrightarrow \bigcirc \neg x)$ ;
  - (b)  $\Box(x \leftrightarrow \bigcirc \bigcirc \neg x)$ ;

(c)  $\Box(x \leftrightarrow \bigcirc \bigcirc x)$ . □

**EXERCISE 14.10** Let the formula  $x \wedge y$  symbolically represent the set of initial states and the formula  $(x' \leftrightarrow \neg x) \wedge (y' \leftrightarrow (x \leftrightarrow y))$  the transition relation of a transition system  $\mathbb{S}$  over  $\{x\}$ .

- (1) Draw the state transition graph of  $\mathbb{S}$ .
- (2) Draw a path for  $\mathbb{S}$ .
- (3) Which of the following formulas are true along all paths in  $\mathbb{S}$ ?
  - (a)  $\Box(x \leftrightarrow \bigcirc \neg x)$ ;
  - (b)  $\Box(x \leftrightarrow \bigcirc \bigcirc x)$ ;
  - (c)  $\Box(y \leftrightarrow \bigcirc \bigcirc \neg y)$ .
- (4) Show that for every formula  $F$  the formula  $\Box(F \leftrightarrow \bigcirc \bigcirc \bigcirc \bigcirc F)$  holds along all paths in  $\mathbb{S}$ . □