Logic and Model Checking

(draft – do not distribute without explicit permission from the authors)

Jaime Ramos  João Rasga
Departamento de Matemática, Instituto Superior Técnico, UTL, Portugal
SQIG, Instituto de Telecomunicações, Portugal

June 2011 – September 2011
Contents

I  Part 1 7

1 Preliminary concepts 9
  1.1 Propositional logic 9
  1.2 Regular languages 13
  1.3 Regular $\omega$-languages 16

2 Concurrent systems 19
  2.1 Transition systems 19
  2.2 Program graphs 27
  2.3 Parallelism and communication 32
    2.3.1 Concurrency and interleaving 32
    2.3.2 Communication by shared variables 34
    2.3.3 Handshaking 35
    2.3.4 Channels 36

3 Linear-time properties 41
  3.1 Linear-Time properties 41
  3.2 Safety properties and invariants 46
  3.3 Liveness properties 58
  3.4 Safety versus liveness properties 59

4 Regular properties 63
  4.1 Finite Automata 63
  4.2 Regular safety properties 70
    4.2.1 Verifying regular safety properties 71
  4.3 Büchi Automata 75
  4.4 $\omega$-Regular properties 88
4.4.1 Verifying $\omega$-regular properties .................................. 89

5 Linear Temporal Logic ....................................................... 107
  5.1 Linear Temporal Logic ................................................... 107
    5.1.1 Syntax ............................................................ 107
    5.1.2 Semantics ......................................................... 108
    5.1.3 Positive Normal Form ........................................... 114
  5.2 LTL Model checking ................................................... 116
  5.3 Axiomatization .......................................................... 126
    5.3.1 Decidability ...................................................... 128
    5.3.2 Completeness ..................................................... 134

6 Computation Tree Logic .................................................. 147
  6.1 Computation Tree Logic ............................................... 147
    6.1.1 Syntax ............................................................ 148
    6.1.2 Semantics ......................................................... 149
    6.1.3 Existential Normal Form ....................................... 154
    6.1.4 CLT versus LTL .................................................. 155
  6.2 Model checking ....................................................... 159
  6.3 Counterexamples and witnesses ...................................... 173
  6.4 Symbolic model-checking ............................................. 175
    6.4.1 Switching functions ............................................ 175
    6.4.2 Encoding transition systems with switching functions ...... 179
    6.4.3 Ordered Binary Decision Diagrams ............................ 184

7 Fairness ........................................................................... 193
  7.1 Fairness in linear time properties .................................. 193
  7.2 Fairness in LTL .......................................................... 195
  7.3 Fairness in CTL .......................................................... 196

8 Partial order reduction .................................................... 197

II Solutions to selected exercises ........................................ 199

1 Preliminary concepts ....................................................... 201
  1.1 Exercises ............................................................... 201
  1.2 Solutions to selected exercises ..................................... 204
## CONTENTS

2 Concurrent systems
   2.1 Exercises ............................................. 209
   2.2 Solutions to selected exercises ...................... 214

3 Linear-time properties
   3.1 Exercises ............................................. 221
   3.2 Solutions to selected exercises ...................... 224

4 Regular Properties
   4.1 Exercises ............................................. 231
   4.2 Solutions to selected exercises ...................... 235

5 Linear Temporal Logic
   5.1 Exercises ............................................. 249
   5.2 Solutions to selected exercises ...................... 253

6 Computation Tree Logic
   6.1 Exercises ............................................. 265
   6.2 Solutions to selected exercises ...................... 269
Part I

Part 1
Chapter 1

Preliminary concepts

1.1 Propositional logic

We assume fixed a set of propositional symbols \( \Xi \). The elements in \( \Xi \) are denoted by the letters \( p, q \), etc. The set of propositional formulas over \( \Xi \), denoted by \( PL \), contains the formulas of the form

\[
\varphi ::= \text{true} \mid p \mid (\neg \varphi) \mid (\varphi \land \varphi) \mid (\varphi \lor \varphi)
\]

where \( p \in \Xi \). The propositional formulas \( \text{true} \) and \( p \in \Xi \) are called atomic formulas. The intuitive meaning of the symbol \( \neg \) is negation, that is, \( \neg \varphi \) is true when \( \varphi \) is not true. The intuitive meaning of the symbol \( \land \) is conjunction, that is, \( \varphi_1 \land \varphi_2 \) is true when \( \varphi_1 \) and \( \varphi_2 \) are both true. The intuitive meaning of the symbol \( \lor \) is disjunction, that is, \( \varphi_1 \lor \varphi_2 \) is true when either \( \varphi_1 \) or \( \varphi_2 \) (or both) are true.

Other Boolean connectives can be defined by abbreviation. For instance implication (\( \rightarrow \)), equivalence (\( \leftrightarrow \)) and exclusive disjunction (\( \oplus \)) are defined as follows

\[
\begin{align*}
\text{false} & \equiv_{\text{def}} (\neg \text{true}) \\
(\varphi_1 \rightarrow \varphi_2) & \equiv_{\text{def}} ((\neg \varphi_1) \lor \varphi_2) \\
(\varphi_1 \leftrightarrow \varphi_2) & \equiv_{\text{def}} (((\varphi_1 \rightarrow \varphi_2) \land (\varphi_2 \rightarrow \varphi_1)) \\
(\varphi_1 \oplus \varphi_2) & \equiv_{\text{def}} (((\varphi_1 \land (\neg \varphi_2)) \lor ((\neg \varphi_1) \land \varphi_2))
\end{align*}
\]

When no confusion arises, we will omit some parentheses. A valuation \( v \) is a map that assigns a truth value (0 or 1) to each propositional symbol,
that is, \( v : \Xi \to \{0, 1\} \). As usual, 0 corresponds to false and 1 corresponds to true. Given a valuation \( v \) and a propositional formula \( \varphi \) we write \( v \models \varphi \) to denote that \( v \) satisfies \( \varphi \). The relation \( \models \) is the satisfaction relation, inductively defined as follows

- \( v \models \text{true} \),
- \( v \models p \) if \( v(p) = 1 \),
- \( v \models \neg \varphi \) if \( v \not\models \varphi \),
- \( v \models \varphi_1 \land \varphi_2 \) if \( v \models \varphi_1 \) and \( v \models \varphi_2 \),
- \( v \models \varphi_1 \lor \varphi_2 \) if \( v \models \varphi_1 \) or \( v \models \varphi_2 \).

If \( v \models \varphi \) we say that \( \varphi \) is satisfied by \( v \). If \( v \models \varphi \) for every valuation \( v \) then the formula \( \varphi \) is said to be valid or a tautology, and is denoted by \( \models \varphi \). If \( v \not\models \varphi \), for every valuation \( v \), then \( \varphi \) is said to be contradictory. Let \( \Psi \subseteq PL \) be a set of formulas and \( v : \Xi \to \{0, 1\} \) a valuation. We say that \( v \) satisfies \( \Psi \), written \( v \models \Psi \), if \( v \models \varphi \) for every \( \varphi \in \Psi \).

Alternatively, a valuation \( v \) can also be defined as a set of propositional symbols, that is, \( v \subseteq \Xi \). In this case, the propositional symbols in \( v \) are true and the others are false. The two definitions are equivalent and we will use them interchangeably.

**Example 1.1.** Let \( p, q \in \Xi \) and consider the valuation \( v_1 \) such that \( v_1(p) = 1 \) and \( v_1(q) = 0 \). We show that \( v_1 \models p \land (\neg q) \):

\[
v_1 \models p \land (\neg q) \text{ iff } v_1 \models p \text{ and } v_1 \not\models q
\]

\[
= v_1 \models p \text{ and } v_1 \not\models q
\]

\[
= v_1(p) = 1 \text{ and } v_1(q) \neq 1
\]

\[
= v_1(p) = 1 \text{ and } v_1(q) = 0.
\]

Since \( v_1(p) = 1 \) and \( v_1(q) = 0 \) we can conclude that \( v_1 \models p \land (\neg q) \).

Consider now the formula \( \neg(p \land (\neg p)) \). We show that this formula is valid. Let \( v \) be any valuation. Then,

\[
v \models \neg(p \land (\neg p)) \text{ iff } v \not\models p \land (\neg p)
\]

\[
= v \not\models p \text{ or } v \not\models (\neg p)
\]

\[
= v(p) \neq 1 \text{ or } v \models p
\]

\[
= v(p) = 0 \text{ or } v(p) = 1.
\]
1.1. PROPOSITIONAL LOGIC

Clearly, any valuation fulfills \( v(p) = 0 \) or \( v(p) = 1 \) and, consequently, satisfies the formula. Hence, the formula is valid.

**Lemma 1.2.** Let \( \varphi \in PL \). Then, \( \varphi \) is valid if and only if \( \neg \varphi \) is not satisfiable.

The proof of this lemma is left as an exercise.

A literal is either an atomic formula or the negation of an atomic formula. A formula is said to be in the *positive normal form* (PNF) if it is either a literal, or a conjunction or disjunction of PNF formulas. This means that negation only appears in literals.

**Proposition 1.3.** Let \( \varphi \) be a propositional formula. Then, there is a formula \( \tilde{\varphi} \) in positive normal form such that \( \models \varphi \leftrightarrow \tilde{\varphi} \).

**Proof.** Consider the following function \( \text{conv} \) that converts any formula in a PNF formula:

- \( \text{conv}(\text{true}) = \text{true} \)
- \( \text{conv}(p) = p \)
- \( \text{conv}(\varphi_1 \land \varphi_2) = \text{conv}(\varphi_1) \land \text{conv}(\varphi_2) \),
- \( \text{conv}(\varphi_1 \lor \varphi_2) = \text{conv}(\varphi_1) \lor \text{conv}(\varphi_2) \),
- \( \text{conv}(\neg \text{true}) = \neg \text{true} \)
- \( \text{conv}(\neg p) = \neg p \)
- \( \text{conv}(\neg \neg \varphi) = \text{conv}(\varphi) \)
- \( \text{conv}(\neg(\varphi_1 \land \varphi_2)) = \text{conv}(\neg \varphi_1) \lor \text{conv}(\neg \varphi_2) \),
- \( \text{conv}(\neg(\varphi_1 \lor \varphi_2)) = \text{conv}(\neg \varphi_1) \land \text{conv}(\neg \varphi_2) \).

We start by proving that

\( \text{conv}(\varphi) \) is a PNF formula. \( \text{(†)} \)

The proof follows by induction on the structure of \( \varphi \). The base corresponds to proving that if \( \varphi \) is either \text{true} or \( p \in \Xi \) then \( \text{conv}(\varphi) \) is a PNF formula.
CHAPTER 1. PRELIMINARY CONCEPTS

But this is straightforward, given that in this case $\text{conv}(\varphi) = \varphi$ and $\varphi$ is a PNF formula.

Next we prove the induction step. If $\varphi$ is $\varphi_1 \land \varphi_2$ or $\varphi_1 \lor \varphi_2$ the result immediately follows by the induction hypothesis. So, consider the case that $\varphi$ is $\neg \psi$. We prove by induction on the structure of $\psi$ that

$$\text{conv}(\neg \psi) \text{ is a PNF formula.} \quad (\dagger\dagger)$$

The base of induction for $(\dagger\dagger)$ corresponds to proving the assertion $\psi$ is an atomic formula. It follows as in the previous case and is straightforward.

Next we prove the induction step for $(\dagger\dagger)$. If $\psi$ is $\neg \psi_1$ then $\text{conv}(\neg \psi) = \text{conv}(\neg \neg \psi_1) = \text{conv}(\psi_1)$. By the induction hypothesis for $(\dagger)$, $\text{conv}(\psi_1)$ is a PNF formula. If $\psi$ is $\psi_1 \land \psi_2$ then $\text{conv}(\neg \psi) = \text{conv}(\neg(\psi_1 \land \psi_2)) = \text{conv}(\neg \psi_1) \lor \text{conv}(\neg \psi_2)$. By the induction hypothesis for $(\dagger\dagger)$, we have that both $\text{conv}(\neg \psi_1)$ and $\text{conv}(\neg \psi_2)$ are PNF formulas and, consequently, so is $\text{conv}(\neg \psi_1) \lor \text{conv}(\neg \psi_2)$. The proof for the disjunction is similar.

We leave as an exercise to prove that $\models \varphi \iff \text{conv}(\varphi)$. Then, we just need to choose $\tilde{\varphi}$ to be $\text{conv}(\varphi)$.

There are two particular cases of PNF formulas. A formula is said to be in the conjunctive normal form if it is a conjunction of disjunctions of literals, that is, if it is of the form

$$\bigwedge_{i \in I} \bigvee_{j \in J_i} l_{i,j}$$

where $I$ is a finite set, each set $J_i$, for $i \in I$, is also a finite set, and $l_{i,j}$ for $i \in I$ and $j \in J_i$ is a literal. As usual, we define $\bigwedge_{i \in \emptyset} \varphi_i \equiv_{\text{def}} \text{true}$ and $\bigvee_{i \in \emptyset} \varphi_i \equiv_{\text{def}} \text{false}$.

The formula $(p_1 \lor \neg p_1) \land p_2 \land (\neg p_1 \lor p_3)$ is a CNF formula while $(\neg (p_1 \lor \neg p_1)) \land p_2$ is not.

**Proposition 1.4.** Let $\varphi$ be a propositional formula. Then, there is a formula $\tilde{\varphi}$ in conjunctive normal form such that $\models \varphi \iff \tilde{\varphi}$.

A formula is said to be in the disjunctive normal form if it is a disjunction of conjunctions of literals, that is, if it is of the form

$$\bigvee_{i \in I} \bigwedge_{j \in J_i} l_{i,j}$$
where \( I \) is a finite set, each set \( J_i \), for \( i \in I \), is also a finite set, and \( l_{i,j} \) for \( i \in I \) and \( j \in J_i \) is a literal.

**Proposition 1.5.** Let \( \varphi \) be a propositional formula. Then, there is a formula \( \widetilde{\varphi} \) in disjunctive normal form such that \( \models \varphi \iff \widetilde{\varphi} \).

## 1.2 Regular languages

In this section, we introduce the basic concepts of regular languages. We assume fixed a nonempty and finite set \( \Sigma \), called the alphabet. The elements of \( \Sigma \) are called symbols or letters.

**Definition 1.6.** Let \( \Sigma \) be an alphabet. A finite word \( w \) over \( \Sigma \) is a finite, possibly empty, sequence \( v_1 v_2 \ldots v_n \) where \( n \in \mathbb{N} \) and each \( v_i \in \Sigma \), for \( i = 1, \ldots, n \). The set of all finite words over \( \Sigma \) is denoted by \( \Sigma^* \).

The finite word corresponding to the empty sequence is denoted by \( \varepsilon \) and is called the **empty word**. The set of all finite nonempty words is \( \Sigma^* \setminus \{\varepsilon\} \) and is denoted by \( \Sigma^+ \).

The **length** of a word is the number of symbols that appear in the underlying sequence. Hence, the length of the finite word \( w = v_1 v_2 \ldots v_n \) is \( n \). In particular, the length of the empty word is 0.

**Definition 1.7.** A **language** \( \mathcal{L} \) over \( \Sigma \) is a set of finite words over \( \Sigma \), that is, \( \mathcal{L} \subseteq \Sigma^* \).

Two straightforward examples of languages are the empty language \( \emptyset \) and the full language \( \Sigma^* \).

**Definition 1.8.** Let \( w = v_1 v_2 \ldots v_n \) be a finite word over \( \Sigma \). A **prefix** of \( w \) is a finite word \( v_1 v_2 \ldots v_i \) for some \( i \) with \( 0 \leq i \leq n \). A **suffix** of \( w \) is a finite word \( v_i v_{i+1} \ldots v_n \) for some \( i \) with \( 1 \leq i \leq n + 1 \). A **subword** of \( w \) is a prefix of a suffix of \( w \).

Observe that \( \varepsilon \) is a prefix of any finite word and is also a suffix of every finite word.

**Definition 1.9.** Let \( w = v_1 \ldots v_n \) and \( w' = v'_1 \ldots v'_m \) be finite words over \( \Sigma \). The **concatenation** of \( w \) and \( w' \) is the finite word \( w.w' = v_1 \ldots v_n v'_1 \ldots v'_m \).
From this definition, it follows that if \( w = \varepsilon \) then \( w.w' = w' \) and, similarly, if \( w' = \varepsilon \) then \( w.w' = w \).

Given a finite word \( w \in \Sigma^* \), we denote by \( w^n \) the concatenation of \( w \) with itself \( n \) times, that is,

\[
w^n = w \ldots w
\]

that is inductively defined as follows

- \( w^0 = \varepsilon \)
- \( w^{n+1} = w^n.w \)

It is not very difficult to see that, in particular, \( w^1 = w \).

The concatenation of words extends to languages in a straightforward way.

**Definition 1.10.** Let \( L \) and \( L' \) be two languages. The **concatenation** of \( L \) and \( L' \) is the language \( L.L' = \{w.w' \mid w \in L, w' \in L'\} \).

As in the case of words, \( L^n \) denotes the concatenation of \( L \) with itself \( n \) times and is defined as expected.

**Definition 1.11.** Let \( L \subseteq \Sigma^* \) be a language. Then \( L^* \) is the language \( \bigcup_{i \in \mathbb{N}} L^i \) and \( L^+ \) is the language \( \bigcup_{i \in \mathbb{N}} L^i \).

The operation \( * \) on languages is called **finite repetition** or **Kleene star**. Consider, for example, the language composed of a single word \( w \). Then \( \{w\}^* = \{\varepsilon, w, w^2, w^3, \ldots\} \), that is, \( \{w\}^* \) is the set composed by all finite repetitions of \( w \). We will often abuse notation and write \( w^* \) for \( \{w\}^* \). The difference between \( L^* \) and \( L^+ \) is that \( L^+ \) does not include the empty word. Hence, \( L^+ = L^* \setminus \{\varepsilon\} \), provided that \( \varepsilon \notin L \).

**Regular languages** can be defined in several ways. Herein, we focus on regular expressions. Later we will show that finite automata can also be used to define the same class of languages.

The language of **regular expressions** over \( \Sigma \), denoted by \( \text{RE}_\Sigma \) contains the expressions of the form

\[
E ::= \emptyset \mid \varepsilon \mid v \mid (E)^* \mid (E + E) \mid (E.E)
\]
where \( v \in \Sigma \). When no confusion arises, we will drop the subscript from \( RE_\Sigma \) and simply write \( RE \).

Each regular expression over an alphabet denotes a language over that same alphabet.

**Definition 1.12.** Let \( E \in RE_\Sigma \) be a regular expression over \( \Sigma \). The language denoted by \( E \) is defined as follows:

- \( \mathcal{L}(\emptyset) = \emptyset \)
- \( \mathcal{L}(\varepsilon) = \{\varepsilon\} \)
- \( \mathcal{L}(v) = \{v\} \)
- \( \mathcal{L}(E^*) = \mathcal{L}(E)^* \)
- \( \mathcal{L}(E + E') = \mathcal{L}(E) \cup \mathcal{L}(E') \)
- \( \mathcal{L}(E.E') = \mathcal{L}(E).\mathcal{L}(E') \)

Two regular expressions \( E_1 \) and \( E_2 \) are *equivalent* if \( \mathcal{L}(E_1) = \mathcal{L}(E_2) \). A language \( \mathcal{L} \subseteq \Sigma^* \) is called *regular* if there is some regular expression \( E \) over \( \Sigma \) such that \( \mathcal{L} = \mathcal{L}(E) \).

In the sequel we will often omit parentheses and the concatenation symbol.

**Example 1.13.** Consider the alphabet \( \Sigma = \{0, 1\} \).

- The regular expression \( 01 + 10 \) denotes the language composed of the sequences \( 01 \) and \( 10 \), that is, \( \mathcal{L}(01 + 10) = \{01, 10\} \).

- The regular expression \( (0 + 1)^*1 \) denotes the language consisting of all sequences of 0’s and 1’s that end in 1.

Regular expressions have some important closure properties. They are, in particular, closed under union, intersection and complementation. Union closure is an immediate consequence of the definition of regular expressions and regular languages: if \( \mathcal{L}_1 \) and \( \mathcal{L}_2 \) are regular languages, then there are regular expressions \( E_1 \) and \( E_2 \) such that \( \mathcal{L}_1 = \mathcal{L}(E_1) \) and \( \mathcal{L}_2 = \mathcal{L}(E_2) \). Consequently, \( \mathcal{L}_1 \cup \mathcal{L}_2 \) is also a regular language because \( \mathcal{L}_1 \cup \mathcal{L}_2 = \mathcal{L}(E_1 + E_2) \). The proof of the other properties is postponed until Chapter 4, when we study the relationship between regular languages and finite automata.
1.3 Regular $\omega$-languages

In this section we present regular $\omega$-languages, that is, regular languages over infinite words.

**Definition 1.14.** A infinite word $\sigma$ over $\Sigma$ is an infinite sequence $\sigma = v_1 v_2 \ldots$ where each $v_i \in \Sigma$, for $i \in \mathbb{N}$. The set of all infinite words over $\Sigma$ is denoted by $\Sigma^\omega$.

The length of an infinite word is always $\omega$.

**Definition 1.15.** An $\omega$-language $L_\omega$ over $\Sigma$ is a set of infinite words over $\Sigma$, that is, $L_\omega \subseteq \Sigma^\omega$.

**Definition 1.16.** Let $\sigma = v_1 v_2 \ldots$ be an infinite word over $\Sigma$. A prefix of $\sigma$ is either a finite word $v_1 v_2 \ldots v_i$ for some $i \in \mathbb{N}$, or $\sigma$ itself. A suffix of $\sigma$ is an infinite word $v_i v_{i+1} \ldots$ for some $i \in \mathbb{N}$. A subword of $\sigma$ is a prefix of a suffix of $\sigma$.

Observe that $\varepsilon$ is a prefix of any infinite word but is not the suffix of any infinite word.

**Definition 1.17.** Let $w = v_1 \ldots v_n$ be a finite word over $\Sigma$ and $\sigma = v'_1 v'_2 \ldots$ be an infinite word over $\Sigma$. The concatenation of $w$ and $\sigma$ is the infinite word $w.\sigma = v_1 \ldots v_n v'_1 v'_2 \ldots$.

As in the finite case, this concatenation operator extends to languages.

**Definition 1.18.** Let $L$ be a language and $L_\omega$ be an $\omega$-language. The concatenation of $L$ and $L_\omega$ is the language $L.L_\omega = \{w.\sigma \mid w \in L, \sigma \in L_\omega\}$.

Given a finite word $w \neq \varepsilon$ we define $w^\omega$ to be the infinite word obtained by infinite repetitions of the word $w$, that is,

$$w^\omega = w.w.\ldots$$

Adapting this to the case of the empty word $\varepsilon$, we have that $\varepsilon^\omega = \varepsilon$. Note that for the case of infinite words we have $\sigma^\omega = \sigma$.

**Definition 1.19.** Let $L \subseteq \Sigma^*$. Then, $L^\omega$ is the set of words in $\Sigma^* \cup \Sigma^\omega$ defined by $\{w_1.w_2.w_3\ldots \mid w_i \in L, i \in \mathbb{N}\}$.
Observe that if \( \varepsilon \notin L \) then \( L^\omega \) is an \( \omega \)-language, that is, if \( L \subseteq \Sigma^+ \) then \( L^\omega \subseteq \Sigma^\omega \).

Like for regular languages, \( \omega \)-regular languages can also be defined in several ways. Herein, we focus on \( \omega \)-regular expressions. In Chapter 4 we will study an alternative characterization of \( \omega \)-regular languages using Büchi automata.

**Definition 1.20.** An \( \omega \)-regular expression over \( \Sigma \) is an expression of the form

\[
E_1.F_1^\omega + \cdots + E_n.F_n^\omega
\]

for \( n \geq 1 \), and \( E_1, \ldots, E_n, F_1, \ldots, F_n \in \text{RE}_\Sigma \) such that \( \varepsilon \notin L(F_i) \), for all \( 1 \leq i \leq n \).

Each \( \omega \)-regular expression denotes an \( \omega \)-language.

**Definition 1.21.** Let \( Z = E_1.F_1^\omega + \cdots + E_n.F_n^\omega \) be an \( \omega \)-regular expression over \( \Sigma \). The language denoted by \( Z \) is

\[
L_\omega(Z) = L(E_1).L(F_1)^\omega \cup \cdots \cup L(E_n).L(F_n)^\omega.
\]

Two \( \omega \)-regular expressions \( Z_1 \) and \( Z_2 \) are equivalent if \( L_\omega(Z_1) = L_\omega(Z_2) \). An \( \omega \)-language \( L_\omega \subseteq \Sigma^\omega \) is called \( \omega \)-regular if there is some \( \omega \)-regular expression \( Z \) over \( \Sigma \) such that \( L_\omega = L_\omega(Z) \).

**Example 1.22.** Consider again the alphabet \( \Sigma = \{0, 1\} \). It is not very difficult to see that the \( \omega \)-language consisting of all infinite words that contain infinitely many 0’s is \( \omega \)-regular. An example of a regular expression that denotes such a language is, for instance, \((1^*0)^\omega\).

And what can be said about the \( \omega \)-language containing all the infinite words that contain only finitely many 0’s?

We will show later that \( \omega \)-regular expressions have some closure properties: they are closed under union and intersection. They are also closed under complement but the proof of this fact is out of the scope of this monograph.
Chapter 2

Concurrent systems

2.1 Transition systems

Throughout this monograph we will use transition systems to describe the behavior of the systems that we want to model. Intuitively, a transition system is a directed graph where nodes represent the states of the system and edges represent the transitions of the system, that is, the state changes.

**Definition 2.1.** A transition system is a tuple $T = \langle S, A, \rightarrow, I, \Xi, L \rangle$ where

- $S$ is a set of *states*;
- $A$ is a set of *actions*;
- $\rightarrow \subseteq S \times A \times S$ is a transition relation;
- $I \subseteq S$ is a set of initial states;
- $\Xi$ is a set of propositional symbols;
- $L : S \rightarrow 2^\Xi$ is a labeling function.

The transition system is said *finite* when $S$, $A$ and $\Xi$ are finite.

We will write $s \xrightarrow{a} s'$ instead of $\langle s, a, s' \rangle \in \rightarrow$. $S$ is the set of states of the system that we want to model. $A$ is the set of actions that the system can perform. The relation $\rightarrow$ describes the behavior of the system, that
is, if $s \xrightarrow{a} s'$ holds then whenever the system is in state $s$ it can change to state $s'$ by performing the action $a$. For each state such that more than one transition is possible, one is selected nondeterministically and executed. The set $I$ is the set of initial states, that is, the states from which the system can start. Observe that $I$ can be empty. In this case, the system will have no behavior as no initial state can be selected. The labeling function $L$ associates to each state $s$ a set $L(s)$ of propositional symbols that hold in that state. We say that a state $s$ satisfies a propositional formula $\varphi$ over $\Xi$, written $s \models \varphi$, if $L(s) \models \varphi$. Recall from Chapter 1 that a valuation can be seen as a set of propositional symbols and so $L(s) \models \varphi$ means that the valuation $L(s)$ satisfies the formula $\varphi$. We will assume that there is a distinguished action symbol $\tau \in A$. We will use this symbol for transitions where the action label is not relevant (for instance, for representing some internal activity of the system).

Example 2.2. Consider the transition system for a simplified snack machine. In this machine, the user inserts a coin and gets as a result either a chocolate or a pack of cookies, but he doesn’t get to choose what. He just inserts the coin and collects his snack.

![Transition System Diagram]

The transition system depicted in the previous figure, $T_{sm}$, is defined as follows:

- the set of states is $S = \{\text{coin, choose, chocolate, cookies}\}$
- the set of actions is $A = \{\text{insert\_coin, get\_choc, get\_cook, } \tau\}$.
- the set of initial states is $I = \{\text{coin}\}$.

We denote an initial state by an incoming arrow. The transition relation is the one depicted in the previous diagram. For instance, we have the transition $\text{cookies } \xrightarrow{\text{get\_cook}} \text{coin}$.
from the state cookies to the state coin labelled by the action symbol get_cook. Finally, observe that at the state choose, two transitions are available, both labelled with \( \tau \). Hence, the system may evolve nondeterministically from this state to either state chocolate or state cookies. Let \( \Xi = \{ \text{coin}, \text{choose}, \text{chocolate}, \text{cookies} \} \) (other choices could have been made). The labeling function identifies each state with the corresponding propositional symbol, that is \( L(s) = \{ s \} \), for every \( s \in S \).

**Definition 2.3.** Let \( T = \langle S, A, \rightarrow, I, \Xi, L \rangle \) be a transition system. For \( s \in S \) and \( a \in A \), the set of *direct* \( a \)-successors of \( s \) is the set
\[
\text{Suc}(s,a) = \{ s' \in S \mid s \xrightarrow{a} s' \}
\]
and the set of *direct* successors of \( s \) is the set
\[
\text{Suc}(s) = \bigcup_{a \in A} \text{Suc}(s,a).
\]
For \( s \in S \) and \( a \in A \), the set of *direct* \( a \)-predecessors of \( s \) is the set
\[
\text{Pred}(s,a) = \{ s' \in S \mid s \xleftarrow{a} s' \}
\]
and the set of *direct* predecessors of \( s \) is the set
\[
\text{Pred}(s) = \bigcup_{a \in A} \text{Pred}(s,a).
\]
These functions extend in a natural way to sets of states.

**Definition 2.4.** A state \( s \) of a transition system is called *terminal* if \( \text{Suc}(s) = \emptyset \).

In general, we are interested in nondeterministic transition systems. However, sometimes it is useful to consider transition systems where the observable behavior is deterministic, for some notion of observable behavior. There are two approaches to formalize the observable behavior of a transition system: one is based on the actions and the other on the labels of the states. The action-based approach assumes that only actions are observable from the outside. The state-based approach ignores actions and assumes that only propositional symbols that hold at each state are observable from the outside. Hence, we can define two kinds of deterministic
transition systems: action-based deterministic transition systems and state-based deterministic transition systems. In the first case there is at most one transition departing from each state for each action symbol. In the second case, for each state label \( v \in 2^\Xi \) and for each state there is at most one transition departing from that state to a state labeled with \( v \).

**Definition 2.5.** Let \( T = \langle S, A, \rightarrow, I, \Xi, L \rangle \) be a transition system. \( T \) is called action-deterministic if \(|I| \leq 1\) and \(|\text{Suc}(s, a)| \leq 1\) for all \( s \in S \) and \( a \in A \). \( T \) is called \( \Xi \)-deterministic if \(|I| \leq 1\) and \(|\text{Suc}(s) \cap \{ s' \in S \mid L(s') = V \}| \leq 1\) for all \( s \in S \) and \( V \in 2^\Xi \).

Now, we present the notion of execution (also called run). An execution describes a possible behavior of the transition system.

**Definition 2.6.** Let \( T = \langle S, A, \rightarrow, I, \Xi, L \rangle \) be a transition system. A finite execution fragment \( \hat{\rho} \) of \( T \) is an alternating sequence of states and actions ending in a state \( \hat{\rho} = s_0 \ a_1 \ s_1 \ a_2 \ldots \ a_n \ s_n \) such that \( s_{i-1} \xrightarrow{a_i} s_i \) for \( i = 1, \ldots, n \) where \( n \geq 0 \) is the length of the execution fragment \( \hat{\rho} \). An infinite execution fragment \( \rho \) of \( T \) is an infinite alternating sequence of states and actions \( \rho = s_0 \ a_1 \ s_1 \ a_2 \ldots \) such that \( s_{i-1} \xrightarrow{a_i} s_i \) for \( i \geq 0 \).

A finite execution fragment \( \hat{\rho} = s_0 \ a_1 \ s_1 \ a_2 \ldots \ a_n \ s_n \) will be written as \( \hat{\rho} = s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} \ldots \xrightarrow{a_n} s_n \).

Similarly, an infinite execution fragment \( \rho = s_0 \ a_1 \ s_1 \ a_2 \ldots \) will be written as \( \rho = s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} \ldots \).

By execution fragment, we will mean either an infinite or finite execution fragment. An execution fragment is called maximal when it cannot be prolonged.

**Definition 2.7.** A maximal execution fragment is either a finite execution fragment that ends in a terminal state, or an infinite execution fragment. An initial execution fragment is an execution fragment that starts in an initial state.
Example 2.8. Recall the example of the snack machine. Consider the following examples of execution fragments

\[\rho_1 = \text{coin} \xrightarrow{\text{insert}} \text{choose} \xrightarrow{\tau} \text{chocolate} \xrightarrow{\text{get}} \text{coin} \xrightarrow{\text{insert}} \text{choose} \ldots\]

\[\rho_2 = \text{chocolate} \xrightarrow{\text{get}} \text{coin} \xrightarrow{\text{insert}} \text{choose} \xrightarrow{\tau} \text{cookies} \ldots\]

\[\hat{\rho} = \text{coin} \xrightarrow{\text{insert}} \text{choose} \xrightarrow{\tau} \text{chocolate} \xrightarrow{\text{get}} \text{coin}.\]

Execution fragments \(\rho_1\) and \(\rho_2\) are infinite, hence maximal. Execution fragment \(\hat{\rho}\) is finite and as \(\text{coin}\) is not terminal, then \(\hat{\rho}\) is not maximal. Furthermore, \(\rho_1\) and \(\hat{\rho}\) are initial.

Having these notions, we may define the notion of execution (or run).

Definition 2.9. An execution of a transition system is an initial maximal execution fragment.

Reachable states are states that can be reached by an execution fragment that starts in an initial state.

Definition 2.10. Let \(T = \langle S, A, \rightarrow, I, \Xi, L \rangle\) be a transition system. A state \(s \in S\) is called reachable in \(T\) if there is an initial finite execution fragment

\[s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} \ldots \xrightarrow{a_n} s_n\]

such that \(s = s_n\). The set of reachable states in \(T\) is denoted by \(\text{Reach}(T)\).

As was said above, when describing the behavior of a system we can adopt two approaches: either state-based or action-based. In the state-base approach only state labels are taken into consideration. In contrast, in the action-based approach, one abstracts away from states and refers only to action labels. From now on, we consider a state-based approach. This means that action labels are not relevant for what follows (actions are relevant for modeling communication and so, are not relevant in subsequent chapters). We focus on the atomic propositional symbols of the states to express the desired properties of the system.

From now on, assume given a transition system \(T\).

Definition 2.11. The state graph of \(T\), denoted by \(G(T)\) is the direct graph \(\langle V, E \rangle\) with vertices \(V = S\) and edges \(E = \{\langle s, s' \rangle \in S \times S \mid s' \in \text{Suc}(s)\}\).
Observe that the state graph of a transition system is obtained from the transition system by omitting all the state labels, all the action labels on the transitions and by ignoring whether a state is initial or not.

We denote by $\text{Suc}^*(s)$ the set of states that are reachable from $s$ in $G(T)$ and by $\text{Pred}^*(s)$ the set of states from which $s$ is reachable also in $G(T)$. Both these notions are easily generalized to sets of states. In particular, observe that $\text{Reach}(T) = \text{Suc}^*(I)$, where $I$ is the set of initial states of $T$.

We can now adapt the notion of execution fragment to graphs. In this case we drop the references to the action labels and refer to such sequences as path fragments.

**Definition 2.12.** A finite path fragment $\widehat{\pi}$ of $T$ is a finite sequence $s_1 \ldots s_n$ such that $s_i \in \text{Suc}(s_{i-1})$, for all $i = 2, \ldots, n$ and $n \geq 0$. An infinite path fragment $\pi$ is an infinite sequence $s_0 s_1 s_2 \ldots$ such that $s_i \in \text{Suc}(s_{i-1})$, for every $i > 0$.

Observe that the empty sequence (the empty path) is a path fragment and that a path fragment $\pi$ is also a path in the graph $G(T)$.

**Convention 2.13.** Let $\widehat{\pi} = s_0 s_1 \ldots s_n$ be a finite path fragment.

- the initial state of $\widehat{\pi}$ is $s_0$ and is denoted by $\text{first}(\pi)$;
- the last state of $\widehat{\pi}$ is $s_n$ and is denoted by $\text{last}(\pi)$;
- $\widehat{\pi}[j] = s_j$ denotes the $j$th state of $\widehat{\pi}$, for $0 \leq j \leq n$;
- $\widehat{\pi}[..j]$ denotes the $j$th prefix of $\pi$, that is, $\widehat{\pi}[..j] = s_0 s_1 \ldots s_j$, for $0 \leq j \leq n$;
- $\widehat{\pi}[j..]$ denotes the $j$th suffix of $\pi$, that is, $\widehat{\pi}[j..] = s_j s_{j+1} \ldots s_n$, for $0 \leq j \leq n$;
- $\text{len}(\widehat{\pi}) = n$ denotes the length of $\widehat{\pi}$.

For infinite path fragments, similar notions can be defined. Let $\pi = s_0 s_1 s_2 \ldots$ be an infinite path fragment.

- the initial state of $\pi$ is $s_0$ and is denoted by $\text{first}(\pi)$;
- $\pi[j] = s_j$ denotes the $j$th state of $\pi$, for $j \geq 0$;
2.1. TRANSITION SYSTEMS

- \( \pi[.,j] \) denotes the \( j \)th prefix of \( \pi \), that is, \( \pi[.,j] = s_0s_1 \ldots s_j \), for \( j \geq 0 \);
- \( \pi[j.,.] \) denotes the \( j \)th suffix of \( \pi \), that is, \( \pi[j.,.] = s_j s_{j+1} \ldots \), for \( j \geq 0 \);

In this case, \( \text{len}(\pi) = \infty \) and \( \text{last}(\pi) = \bot \), where \( \bot \) denotes undefined.

**Definition 2.14.** A maximal path fragment is either a finite path that ends in a terminal state, or, an infinite path fragment. An initial path fragment is a path fragment that starts in an initial state.

**Example 2.15.** Recall the transition system of Example 2.2 for a snack machine.

\[
\begin{align*}
\pi_1 &= \text{coin} \rightarrow \text{choose} \rightarrow \text{chocolate} \rightarrow \text{coin} \rightarrow \text{choose} \rightarrow \text{cookies} \ldots \\
\pi_2 &= \text{chocolate} \rightarrow \text{coin} \rightarrow \text{choose} \rightarrow \text{cookies} \ldots \\
\hat{\pi} &= \text{coin} \rightarrow \text{choose} \rightarrow \text{chocolate} \rightarrow \text{coin}.
\end{align*}
\]

It is easy to see that \( \hat{\pi} \) is a finite path fragment and that \( \pi_1 \) and \( \pi_2 \) are infinite path fragments. Furthermore, \( \pi_1 \) and \( \pi_2 \) are maximal while \( \hat{\pi} \) is not maximal, but \( \pi_1 \) and \( \hat{\pi} \) are initial and \( \pi_2 \) is not.

We denote by \( \text{Paths}(s) \) the set of all maximal path fragments \( \pi \) such that \( \text{first}(\pi) = s \), and by \( \text{Paths}_{\text{fin}}(s) \) the set of all finite path fragments \( \hat{\pi} \) such that \( \text{first}(\hat{\pi}) = s \).

**Definition 2.16.** A path of a transition system is an initial, maximal path fragment.

It is easy to see that, in Example 2.15, \( \pi_1 \) is a path. One should now confuse a path in a transition system with a path in a directed graph. Notwithstanding, a path in a transition system is always maximal whereas a path in a directed graph is not always maximal (in the graph-theoretic sense).

We denote by \( \text{Paths}(T) \) the set of all paths of \( T \) and by \( \text{Paths}_{\text{fin}}(T) \) the set of all initial, finite path fragments of \( T \).

In a nutshell, traces are sequences of sets of propositional symbols (sequences of valuations) induced by paths of a transition system. For convenience, we assume that the transition system has no terminal states, although we can cope with that by assuming that after each terminal state, there is a state with a self loop, from which no arrow departs.
Definition 2.17. Let \( T = \langle S, A, \rightarrow, I, \Xi, L \rangle \) be a transition system without terminal states. The trace of the infinite path fragment \( \pi = s_0 s_1 \ldots \) is the sequence \( \text{trace}(\pi) = L(s_0) L(s_1) \ldots \). The trace of the finite path fragment \( \hat{\pi} = s_0 s_1 \ldots s_n \) is the sequence \( \text{trace}(\hat{\pi}) = L(s_0) L(s_1) \ldots L(s_n) \).

Example 2.18. Recall Example 2.2 and Example 2.15. Then, we can define the following traces:

\[
\begin{align*}
\text{trace}(\pi_1) &= \{\text{coin}\} \{\text{choose}\} \{\text{chocolate}\} \{\text{coin}\} \{\text{choose}\} \{\text{cookies}\} \ldots \\
\text{trace}(\pi_2) &= \{\text{chocolate}\} \{\text{coin}\} \{\text{choose}\} \{\text{cookies}\} \ldots \\
\text{trace}(\hat{\pi}) &= \{\text{coin}\} \{\text{choose}\} \{\text{chocolate}\} \{\text{coin}\}.
\end{align*}
\]

In the context of a transition system without terminal states, the trace of a path fragment is a sequence of valuations that label the corresponding states of the underlying path fragment. We can extend traces to a set \( \Pi \) of infinite path fragments:

\[
\text{trace}(\Pi) = \{ \text{trace}(\pi) \mid \pi \in \Pi \}.
\]

We can also consider the traces of a state. A trace of a state \( s \) is the trace of an infinite path fragment \( \pi \) with \( \text{first}(\pi) = s \). We denote by \( \text{Traces}(s) \) the set of all traces of \( s \), that is,

\[
\text{Traces}(s) = \text{trace}(\text{Paths}(s)).
\]

The set of traces of the transition system \( T \) without terminal states, denoted by \( \text{Traces}(T) \), is the set of traces of all initial states, that is,

\[
\text{Traces}(T) = \bigcup_{s \in I} \text{Traces}(s).
\]

In a similar way, we can define the above notions for the finite case.

\[
\text{Traces}_{\text{fin}}(s) = \text{trace}(\text{Paths}_{\text{fin}}(s))
\]

and

\[
\text{Traces}_{\text{fin}}(T) = \bigcup_{s \in I} \text{Traces}_{\text{fin}}(s).
\]

Observe that \( \text{Paths}_{\text{fin}}(T) = \text{pref}(\text{Paths}(T)) \) and, consequently, \( \text{Traces}_{\text{fin}}(T) = \text{pref}(\text{Traces}(T)) \).
Example 2.19. We have that \( \text{trace}(\pi_1) \in \text{Traces}(\text{coin}) \) and \( \text{trace}(\pi_2) \in \text{Traces}(\text{chocolate}) \). Furthermore, \( \text{trace}(\pi_1) \in \text{Traces}(T_{sm}) \). As for the finite path \( \hat{\pi} \), we have that \( \text{trace}(\hat{\pi}) \in \text{Traces}_{\text{fin}}(\text{coin}) \) and \( \text{trace}(\hat{\pi}) \in \text{Traces}_{\text{fin}}(T_{sm}) \).

In the sequel it will be useful to consider traces over a subset of propositional symbols. Let \( T \) be a transition system with set of propositional symbols \( \Xi \) and let \( \Xi' \) be a set of propositional symbols such that \( \Xi' \subseteq \Xi \). Given a path \( \pi = s_0 s_1 s_2 \ldots \) of \( T \) we denote by \( \text{trace}_{\Xi'}(\pi) \) the trace of \( \pi \) restricted to \( \Xi' \), that is,

\[
\text{trace}_{\Xi'}(\pi) = (L(s_0) \cap \Xi') (L(s_1) \cap \Xi') (L(s_2) \cap \Xi') \ldots
\]

This notation applies also to finite paths. We can extend this notation to the set of traces of \( T \) as follows:

\[
\text{Traces}_{\Xi'}(T) = \text{trace}_{\Xi'}(\text{Paths}(T)).
\]

When the set of propositional symbols is clear from context it may be omitted.

### 2.2 Program graphs

We now illustrate the use of transition systems in some more elaborate examples. In particular, we will introduce the notion of program graph to model these examples. And we will show how to extract a transition system from a program graph.

Example 2.20. Recall the snack machine of Example 2.2. Consider an extension of this machine, where the number of chocolates and cookies is counted. If the machine is empty, i.e. there are no more chocolates nor cookies, the machine returns the coin. If only one kind of item is available, then an item of that kind is returned. Additionally, there is an extra action for refilling the machine with cookies and chocolates.

In this case, while some actions are still unconditional, others depend on the state of the machine. For instance, to get a chocolate, there must be some chocolates in the machine. Intuitively, we will want to write

\[
\text{choose } \text{choch} > 0 : \text{get \_ choc} \quad \mapsto \quad \text{coin}
\]
For simplicity, we consider only two states, coin and choose, omitting the internal action $\tau$. We will have additional actions ret_coin, for returning the coin when the machine is empty, and refill, for refilling the machine.

In this case, we will need two variables for counting the number of chocolates and the number of cookies.

Assume fixed a set of typed variables $\text{Var}$, meaning that associated with each variable we have a type, for instance boolean, integer or char. The type of a variable $x$ in $\text{Var}$ is called the domain $\text{dom}(x)$ of that variable. For now, we do not restrict the sets $\text{dom}(x)$ and so, for each $x$ in $\text{Var}$, $\text{dom}(x)$ is an arbitrary, possibly infinite, set.

A Boolean condition over $\text{Var}$ is a propositional formula over the set of propositional symbols defined as follows

$$\Xi_{\text{Var}} = \{\langle x_1, \ldots, x_n \rangle \in D \mid x_i \in \text{Var}, x_i \neq x_j, D \subseteq \text{dom}(x_1) \times \cdots \times \text{dom}(x_n)\}.$$ 

For example, consider the following Boolean condition

$$(x_1 > x_2) \land (y = \text{green})$$

where $x_1, x_2$ are integer variables and $\text{dom}(y) = \{\text{green, yellow, red}\}$. Note that we are simplifying the notation by writing $x_1 > x_2$ instead of

$$\langle x_1, x_2 \rangle \in \left\{\langle n_1, n_2 \rangle \in \mathbb{N}^2 \mid n_1 > n_2\right\}.$$ 

The set of Boolean conditions over $\text{Var}$ is denoted by $\text{Cond}(\text{Var})$. From now on, we refer to a Boolean condition simply as condition.

A variable assignment on $\text{Var}$ is a map that assigns to each variable $x$ a value in $\text{dom}(x)$. We denote by $\text{Asg}(\text{Var})$ the set of all variable assignments. Let $\eta \in \text{Asg}(\text{Var})$ and $g \in \text{Cond}(\text{Var})$. The satisfaction of $g$ by $\eta$, written $\eta \models g$, is inductively defined as follows:

- $\eta \models \text{true}$,
- $\eta \models \langle x_1, \ldots, x_n \rangle \in D$ if $\langle \eta(x_1), \ldots, \eta(x_n) \rangle \in D$,
- $\eta \models \neg g$ if $\eta \not\models g$,
- $\eta \models g_1 \land g_2$ if $\eta \models g_1$ and $\eta \models g_2$. 

2.2. PROGRAM GRAPHS

- \( \eta \models g_1 \lor g_2 \) if \( \eta \models g_1 \) or \( \eta \models g_2 \).

Recall the condition \((x_1 > x_2) \land (y = \text{green})\) and consider the variable assignments \(\eta_1\) and \(\eta_2\) such that \(\eta_1(x_1) = 1, \eta_1(x_2) = 0, \eta_1(y) = \text{green},\) and \(\eta_2(x_1) = 1, \eta_2(x_2) = 0, \eta_2(y) = \text{yellow}\). For this condition, it is not very difficult to see that \(\eta_1 \models (x_1 > x_2) \land (y = \text{green})\) but \(\eta_2 \not\models (x_1 > x_2) \land (y = \text{green})\), given that \(\eta_2 \not\models y = \text{green}\).

The effect of an action is defined over variable assignments, that is, it takes a variable assignment \(\eta\) and returns a variable assignment \(\eta'\) that is obtained from \(\eta\) by changing the values of some variables to encompass the effects of the action:

\[
\text{Effect}: \text{Act} \times \text{Asg}(\text{Var}) \rightarrow \text{Asg}(\text{Var})
\]

For instance, consider the action \(x := x + 1\). This action increments the value of the variable \(x\). Then, the effect of this action on an assignment \(\eta\), \(\text{Effect}(x := x + 1, \eta) = \eta'\), is such that

\[
\eta'(x) = \eta(x) + 1 \quad \text{and} \quad \eta'(y) = \eta(y), \quad \text{for } y \neq x.
\]

In the sequel, given an assignment \(\eta\) we denote by \(\eta[x := \text{expr}]\) the assignment \(\eta'\) such that \(\eta'(y) = \eta(y)\), for \(y \neq x\), and \(\eta'(x) = \eta(\text{expr})\), where \(\eta(\text{expr})\) denotes the result of evaluating the expression \(\text{expr}\) with the assignment \(\eta\). For instance, in the previous example we could have written \(\text{Effect}(x := x + 1, \eta) = \eta[x := x + 1]\).

**Definition 2.21.** A program graph \(\text{PG}\) over \(\text{Var}\) is a tuple

\[\langle \text{Loc}, \text{Act}, \text{Effect}, \rightarrow, \text{Loc}_0, g_0 \rangle \]

where

- \(\text{Loc}\) is a set of locations,
- \(\text{Act}\) is a set of actions,
- \(\text{Effect}: \text{Act} \times \text{Asg}(\text{Var}) \rightarrow \text{Asg}(\text{Var})\) is the effect function,
- \(\rightarrow \subseteq \text{Loc} \times \text{Cond}(\text{Var}) \times \text{Act} \times \text{Loc}\) is the conditional transition relation,
- \(\text{Loc}_0 \subseteq \text{Loc}\) is the set of initial locations,
- \(g_0 \in \text{Cond}(\text{Var})\) is the initial condition.
CHAPTER 2. CONCURRENT SYSTEMS

Notation: \( l \xrightarrow{g,a} l' \) is used instead of \( \langle l, g, a, l' \rangle \in \leftrightarrow \). Condition \( g \) is called the \textit{guard} of the conditional transition. When the guard is a tautology we omit it from the conditional transition.

**Example 2.22.** We can define the program graph for snack machine described in Example 2.20. The set of variables is \( \text{Var} = \{\text{nchoc}, \text{ncook}\} \), where both variables have the domain \( \{0, 1, \ldots, \text{max}\} \). Then, \[ PG_{sm} = \langle \text{Loc}, \text{Act}, \text{Effect}, \leftrightarrow, \text{Loc}_0, g_0 \rangle \]
is such that

- \( \text{Loc} = \{\text{coin}, \text{choose}\} \);
- \( \text{Act} = \{\text{insert\_coin, get\_choc, get\_cook, ret\_coin, refill}\} \)
- \( \text{Effect} \) is defined as follows
  - \( \text{Effect}(\text{insert\_coin}, \eta) = \eta \),
  - \( \text{Effect}(\text{get\_choc}, \eta) = \eta[\text{nchoc} := \text{nchoc} - 1] \),
  - \( \text{Effect}(\text{get\_cook}, \eta) = \eta[\text{ncook} := \text{ncook} - 1] \),
  - \( \text{Effect}(\text{ret\_coin}, \eta) = \eta \)
  - \( \text{Effect}(\text{refill}, \eta) = \eta[\text{nchoc} := \text{max}, \text{ncook} := \text{max}] \)
- \( \text{Loc}_0 = \{\text{coin}\} \),
- \( g_0 = (\text{nchoc} = \text{max} \land \text{ncook} = \text{max}) \).

The initial condition states that the machine starts completely filled. The conditional transition relation contains the following transitions:

- \( \text{coin} \xrightarrow{\text{true}\_\text{insert\_coin}} \text{choose} \)
- \( \text{coin} \xrightarrow{\text{true}\_\text{refill}} \text{coin} \)
- \( \text{choose} \xrightarrow{\text{nchoc} > 0\_\text{get\_choc}} \text{coin} \)
- \( \text{choose} \xrightarrow{\text{ncook} > 0\_\text{get\_cook}} \text{coin} \)
- \( \text{coin} \xrightarrow{\text{coin\_nchoc} = 0\_\text{\&\text{ncook} = 0}\_\text{ret\_coin}} \text{choose} \)

The first transition states that at state \text{coin} the user can always insert a coin. Also at that state, the machine can always be refilled. The user will only get an item if such an item is available. This is expressed by the transitions in the second line. The last transition states that if the machine is empty then the coin is returned to the user.
2.2. PROGRAM GRAPHS

The behavior of a “program” in a location depends on the current variable assignment being considered. In a given location \( l \in \text{Loc} \) all the conditional transitions that start in \( l \) and for which the guard is satisfied by the current variable assignment, \( g^a_l : l \xrightarrow{\eta} l' \), are considered and a non-deterministic choice is made between them. The execution of the chosen action \( a \) then changes the current variable assignment according to what is specified in \( \text{Effect}(a, \cdot) \) and then, the system changes to location \( l' \). If there is no conditional transition in the above conditions the system stops.

Each program graph can be seen as a transition system. Each state of the transition system will consist of a control component, that is, a location \( l \in \text{Loc} \) of the program graph together with a variable assignment \( \eta \in \text{Asg}(\text{Var}) \). Hence, each state of the transition system will be a pair \( \langle l, \eta \rangle \). In particular, initial states will be all the initial locations paired with all the variable assignments that satisfy the initial condition \( g_0 \). Propositional symbols will contain the Boolean conditions for the variables as well as the locations so that we are able to state where the control component of the program is at. For instance, in the previous example, we may write the following condition

\[(\text{nchoc} > 0) \land \text{coin}\]

The labeling function associates with each state \( \langle l, \eta \rangle \) the label \( l \) itself together with all the conditions on variables that hold for the variable assignment \( \eta \). Finally, the transition relation is defined as follows: for each conditional transition \( l \xrightarrow{g^a_l} l' \) in the program graph and variable assignment \( \eta \) satisfying condition \( g \) there is a transition in the transition system from state \( \langle l, \eta \rangle \) to state \( \langle l', \text{Effect}(a, \eta) \rangle \) labelled with \( a \). The set of actions for the transition system is the same as the set of actions of the program graph.

**Definition 2.23.** Let \( PG = \langle \text{Loc}, \text{Act}, \text{Effect}, \rightarrow, \text{Loc}_0, g_0 \rangle \) be a program graph. The **transition system of** \( PG \) is

\[ T(PG) = \langle S, \text{Act}, \rightarrow, I, \Xi, L \rangle \]

where

- \( S = \text{Loc} \times \text{Asg}(\text{Var}) \),
- \( \rightarrow \subseteq S \times \text{Act} \times S \) is defined by:

\[
\frac{l \xrightarrow{g^a_l} l' \quad \eta \models g}{\langle l, \eta \rangle \xrightarrow{a} \langle l', \text{Effect}(a, \eta) \rangle}
\]
CHAPTER 2. CONCURRENT SYSTEMS

• \( I = \{ \langle l, \eta \rangle \mid l \in \text{Loc}_0 \text{ and } \eta \vdash g_0 \} \),

• \( \Xi = \text{Loc} \cup \text{Cond}(\text{Var}) \),

• \( L(\langle l, \eta \rangle) = \{ l \} \cup \{ g \in \text{Cond}(\text{Var}) \mid \eta \vdash g \} \).

2.3 Parallelism and communication

So far we have only considered sequential systems. In general, a complex system consists of several sequential systems composed in parallel (henceforth we will sometimes call these sequential components processes). We will present several constructions for parallel composition of transition systems. These constructions go from the very simple composition mechanism, where no communication between the systems takes place to the more complex mechanisms where messages can be transferred, either synchronously or asynchronously. Given transition systems \( T_1, \ldots, T_n \) describing the sequential components of the system (the processes) that we want to specify, our goal is to define an operator \( \parallel \) such that

\[ T = T_1 \parallel T_2 \parallel \cdots \parallel T_n \]

is a transition system for the parallel composition of transition systems \( T_1 \) through \( T_n \). Here, we are assuming that the operator \( \parallel \) is commutative and associative. But the nature of this operator will depend on the kind of communication that is being assumed. In some cases, as we will see, the operator will fail to be associative.

2.3.1 Concurrency and interleaving

Interleaving is a simple, yet widely used, mechanism for parallel composition of systems. In this case, the state of the global system is composed of the states of each of the component systems. Actions of the components are then interleaved. This means that at each point, one of the component systems is chosen nondeterministically and one of its actions is taken. Consider a system composed of two (nonterminating) “processes” \( P_1 \) and \( P_2 \), acting independent of each other. We list below some possible sequences where
the actions of the two processes are interleaved:

\[
P_1 P_2 P_1 P_2 P_1 P_2 \ldots
\]
\[
P_2 P_2 P_3 P_1 P_2 P_1 \ldots
\]
\[
P_1 P_1 P_1 P_1 P_1 \ldots
\]

Observe that in the last sequence no action from process \( P_2 \) is taken. Later, we will study mechanisms for preventing this unfair treatment of a process. For now, all the above sequences are acceptable interleaving sequences. We will denote the interleaving of transition systems by \( || \). It is assumed that no communication occurs between the processes.

**Definition 2.24.** Let \( T_1 = \langle S_1, A_1, \rightarrow_1, I_1, \Xi_1, L_1 \rangle \) and \( T_2 = \langle S_2, A_2, \rightarrow_2, I_2, \Xi_2, L_2 \rangle \) be transition systems. The transition system \( T_1 || T_2 \) is the tuple

\[
\langle S_1 \times S_2, A_1 \cup A_2, \rightarrow, I_1 \times I_2, \Xi_1 \cup \Xi_2, L \rangle
\]

where

- the transition relation \( \rightarrow \) is defined by:

\[
\begin{align*}
s_1 a_1 & \rightarrow_1 s'_1 \\
\langle s_1, s_2 \rangle & \rightarrow \langle s'_1, s_2 \rangle \\
s_2 a_2 & \rightarrow_2 s'_2 \\
\langle s_1, s_2 \rangle & \rightarrow \langle s_1, s'_2 \rangle
\end{align*}
\]

- the labeling function \( L : S_1 \times S_2 \rightarrow 2^{\Xi_1 \cup \Xi_2} \) is such that

\[
L(\langle s_1, s_2 \rangle) = L_1(s_1) \cup L_2(s_2).
\]

The states of the transition system \( T_1 || T_2 \) are pairs of states \( \langle s_1, s_2 \rangle \), where \( s_1 \) is a state of \( T_1 \) and \( s_2 \) is a state of \( T_2 \). In particular, the initial states of this transition system are all the pairs of initial states from the components. The transitions from a global state \( \langle s_1, s_2 \rangle \) are the transitions of \( T_1 \) that depart from \( s_1 \) together with the transitions of \( T_2 \) that depart from \( s_2 \), with the proviso that transitions from one component do not affect the state of the other component. This means that a transition \( s_1 a \rightarrow s'_1 \) from \( T_1 \) will induce a transition from a global state \( \langle s_1, s_2 \rangle \) to the global state \( \langle s'_1, s_2 \rangle \) where the state of \( T_1 \) is changed according to the local transition and the state of \( T_2 \) remains unchanged. The same applies to the transitions
CHAPTER 2. CONCURRENT SYSTEMS

from $T_2$. The labeling function associates to each global state the labels of both local states.

Note that this construction can be applied to program graphs, that is, to their underlying transition systems, as long as there are no shared variables.

It is straightforward to see that this operator on transition systems is commutative and associative, that is,

$$T_1 \parallel T_2 = T_2 \parallel T_1$$

and

$$T_1 \parallel (T_2 \parallel T_3) = (T_1 \parallel T_2) \parallel T_3$$

for every transition systems $T_1, T_2, T_3$.$^1$

2.3.2 Communication by shared variables

Consider the following example

$$x := 2x \parallel x := x + 1$$

If we consider the interleaving of the transition systems associated with the program graphs of the two systems starting in a state $\langle \langle l_1, \eta \rangle, \langle l_2, \eta \rangle \rangle$ with $\eta(x) = 3$, we end up with an inconsistent state $\langle \langle l_1, \eta_1 \rangle, \langle l_2, \eta_2 \rangle \rangle$ such that $\eta_1(x) = 6$ and $\eta_2(x) = 4$ which should not be permitted, given that we want the same (shared) variable to have only one value in each state of the composed system.

In order to cope with parallel composition of programs with shared variables, we need to consider an interleaving operator at the level of program graphs. Then, we can determine the associated transition system. In general, we will have $T(PG_1 \parallel PG_2) \neq T(PG_1) \parallel T(PG_2)$.

**Definition 2.25.** Let $PG_1 = \langle Loc_1, Act_1, Effect_1, \leftrightarrow_1, Loc_0, g_0 \rangle$ be a program graph over $Var_1$ and $PG_2 = \langle Loc_2, Act_2, Effect_2, \leftrightarrow_2, Loc_0, g_0 \rangle$ be a program graph over $Var_2$. The program graph $PG_1 \parallel PG_2$ over $Var_1 \cup Var_2$ is the tuple $^2$

$$\langle Loc_1 \times Loc_2, Act_1 \uplus Act_2, Effect, \leftrightarrow, Loc_0 \times Loc_0, g_0 \wedge g_0 \rangle$$

where

$^1$When we write $T = T'$ we mean that the two transition systems are the same, apart from state names.

$^2$We denote by $A \uplus B$ the disjoint union of $A$ and $B$. 
2.3. PARALLELISM AND COMMUNICATION

• the conditional transition relation $\rightarrow$ is defined by:

$$
\begin{align*}
& l_1 \xrightarrow{g^a} l_1' \\
& \langle l_1, l_2 \rangle \xrightarrow{g^a} \langle l_1', l_2 \rangle \\
& l_2 \xrightarrow{g^a} l_2' \\
& \langle l_1, l_2 \rangle \xrightarrow{g^a} \langle l_1, l_2' \rangle
\end{align*}
$$

• for every $a \in \text{Act}_i$, with $i = 1, 2$, and $x \in \text{Var}_1 \cup \text{Var}_2$

$$
\text{Effect}(a, \eta)(x) = \begin{cases} 
\text{Effect}_i(a, \eta|_{\text{Var}_i})(x) & \text{if } x \in \text{Var}_i \\
\eta(x) & \text{otherwise}
\end{cases}
$$

In the previous definition, given a variable assignment $\eta \in \text{Asg}(\text{Var}_1 \cup \text{Var}_2)$ we denote by $\eta|_{\text{Var}_i}$ the restriction of $\eta$ to the set of variables $\text{Var}_i$. Clearly, $\eta|_{\text{Var}_i} \in \text{Asg}(\text{Var}_i)$.

The program graphs $PG_1$ and $PG_2$ share the global variables $\text{Var}_1 \cap \text{Var}_2$ (also called shared variables). The variables in $\text{Var}_1 \setminus \text{Var}_2$ are the local variables of $PG_1$ and $\text{Var}_2 \setminus \text{Var}_1$ are the local variables of $PG_2$.

This distinction between local and global variables produces a distinction between actions of the program graph $PG_1 \parallel PG_2$. We will call critical to the actions that access (read or write) global variables and we will call noncritical to all the other actions.

2.3.3 Handshaking

In this section we introduce another mechanism for parallel composition of processes called handshaking. In this case processes that want to interact have to synchronize, that is, processes can only interact if they participate in the same action at the same time.

A set of handshake actions $H$ is fixed with $\tau \notin H$. If both processes are ready to execute the same handshake action, then message passing can take place. All other actions outside $H$ are independent and can be executed autonomously in an interleaved fashion.

**Definition 2.26.** Let $T_1 = \langle S_1, A_1, \rightarrow_1, I_1, \Xi_1, L_1 \rangle$ and $T_2 = \langle S_2, A_2, \rightarrow_2, I_2, \Xi_2, L_2 \rangle$ be transition systems and $H \subseteq A_1 \cap A_2$ with $\tau \notin H$. The transition system $T_1 \parallel_H T_2$ is the tuple

$$
\langle S_1 \times S_2, A_1 \cup A_2, \rightarrow, I_1 \times I_2, \Xi_1 \cup \Xi_2, L \rangle
$$

where
• the transition relation $\rightarrow$ is defined, for $a \not\in H$, by
\[
\begin{align*}
&s_1 \xrightarrow{a} s'_1 & & s_2 \xrightarrow{a} s'_2 \\
&\langle s_1, s_2 \rangle \xrightarrow{a} \langle s'_1, s_2 \rangle & & \langle s_1, s_2 \rangle \xrightarrow{a} \langle s_1, s'_2 \rangle
\end{align*}
\]
otherwise it is defined by
\[
\begin{align*}
&s_1 \xrightarrow{a} s'_1 & & s_2 \xrightarrow{a} s'_2 \\
&\langle s_1, s_2 \rangle \xrightarrow{a} \langle s'_1, s'_2 \rangle
\end{align*}
\]
• the labeling function $L : S_1 \times S_2 \rightarrow 2^{\Xi_1 \cup \Xi_2}$ is such that
\[
L(\langle s_1, s_2 \rangle) = L_1(s_1) \cup L_2(s_2).
\]
When $H = A_1 \cap A_2$ then we write $T_1 \parallel H T_2$ for $T_1 \parallel H T_2$. Furthermore, observe that if $H = \emptyset$ then
\[
T_1 \parallel \emptyset T_2 = T_1 \parallel T_2.
\]
The operator $\parallel H$ is commutative but, in general, is not associative. In general, if $H \neq H'$ then, we do not necessarily have $T_1 \parallel H (T_2 \parallel H' T_3) = (T_1 \parallel H T_2) \parallel H' T_3$. However, for a fixed set $H$ of handshake actions over which all processes synchronize the operator is associative, that is, $T_1 \parallel H (T_2 \parallel H T_3) = (T_1 \parallel H T_2) \parallel H T_3$. In this case, we may write $T_1 \parallel H T_2 \parallel H T_3$.

### 2.3.4 Channels

In this section we introduce channel systems. A channel system is a parallel system where processes communicate via channels. A channel is a buffer that is used to pass messages between processes. There are two types of channels: asynchronous channels and synchronous channels. An asynchronous channel behaves like a first-in first-out buffer that contains messages. Processes using these channels can write messages in a channel and read messages from a channel. The writing and reading of messages is asynchronous. A synchronous channel has no buffer capacity. In this case, when a message is written in the channel by a process it is read simultaneously by another process (hence the synchronous classification).

A channel system consists on $n$ processes $P_1, \ldots, P_n$. Each of these processes is specified by a program graph $PG_i$ that is extended with communication actions:
• $c!v$ transmit the value $v$ along channel $c$

• $c?x$ receive a message via the channel $c$ and assign it to variable $x$.

If $c$ is an asynchronous channel then the action $c!v$ corresponds to putting the value $v$ at the end of the channel $c$. The action $c?x$ corresponds to retrieving the first message from channel $c$ and assigning it to variable $x$, provided that the channel is not empty. It is assumed implicitly that variable $x$ is of the same type as the messages in the channel.

An asynchronous channel $c$ has a finite or infinite capacity that is the maximum number of messages that the channel can store before they are read and a type (domain) specifying the type of messages it can store. The capacity of a channel $c$, denoted by $\text{cap}(c)$, is a value in $\mathbb{N}_0 \cup \{\infty\}$. The domain of $c$ is denoted by $\text{dom}(c)$. If $\text{cap}(c) \in \mathbb{N}$ then $c$ has finite capacity. If $\text{cap}(c) = \infty$ then $c$ has infinite capacity. For a synchronous channel $c$, its capacity is 0, that is $\text{cap}(c) = 0$. Finally, observe that we can only write a message in a channel with finite capacity if the channel is not full, that is, if the number of stored messages is less than the capacity of the channel.

A channel assignment $\xi$ is a map from every channel $c \in \text{Chan}$ to a sequence $\xi(c) \in \text{dom}(c)^*$ such that the length of the sequence cannot exceed the capacity of $c$. The set $\text{Asg}(\text{Chan})$ denotes the set of all channel assignments. In particular, we denote by $\xi_0$ the channel assignment that maps every channel $c$ to the empty sequence $\varepsilon$, that is, $\xi_0(c) = \varepsilon$, for every $c \in \text{Chan}$.

We assume fixed a finite set $\text{Chan}$ of channel identifiers. We define the set of communication actions over $\text{Var}$ as follows

\[
\text{Comm}(\text{Var}) = \{c!v \mid c \in \text{Chan}, v \in \text{dom}(c)\} \\
\cup \ \{c?x \mid c \in \text{Chan}, x \in \text{Var}, \text{dom}(x) \supseteq \text{dom}(c)\}.
\]

**Definition 2.27.** A program graph over $(\text{Var}, \text{Chan})$ is a tuple

\[
\langle \text{Loc}, \text{Act}, \text{Effect}, \Rightarrow, \text{Loc}_0, g_0 \rangle
\]

where

• $\text{Loc}$ is a set of locations,

• $\text{Act}$ is a set of actions,
CHAPTER 2. CONCURRENT SYSTEMS

• $\mathcal{L} \subseteq \text{Loc} \times \text{Cond}(\text{Var}) \times (\text{Act} \cup \text{Comm}(\text{Var})) \times \text{Loc}$ is the conditional transition relation,

• $\text{Loc}_0 \subseteq \text{Loc}$ is the set of initial locations,

• $g_0 \in \text{Cond}(\text{Var})$ is the initial condition.

A channel system $\mathcal{CS}$ over $\langle \text{Var}, \text{Chan} \rangle$ is a family $\{\mathcal{P}G_i\}_{i \leq n}$ (for some $n \in \mathbb{N}$) such that each $\mathcal{P}G_i$ is a program graph over $\langle \text{Var}_i, \text{Chan} \rangle$ and $\text{Var} = \bigcup_{i \leq n} \text{Var}_i$.

We denote a channel system $\mathcal{CS}$ by

$$[\mathcal{P}G_1 | \cdots | \mathcal{P}G_n].$$

The transition relation $\mathcal{L}$ for a program graph over $\langle \text{Var}, \text{Chan} \rangle$ consists of two types of conditional transitions: conditional transitions for actions and conditional transitions for communication actions. This last type of conditional transitions can be of the form $l^g:v \mathcal{L} l'$ or of the form $l^g:c\mathcal{L} l'$.

For the sake of simplicity, in the sequel, we assume that $g$ holds.

If $\text{cap}(c) = 0$ then $c$ is a synchronous channel. In this case for a process $P_i$ to send a message $v$ through channel $c$ means that some other process $P_j$ must be ready to receive that message synchronously, that is, for process $P_i$ to perform the transition $l^i:c\mathcal{L} v \mathcal{L} l'_i$ then $P_j$ must be available to perform the transition $l^j:cx \mathcal{L} l'_j$

and both actions must be performed simultaneously. The effect of these transitions will be the assignment of the message $v$ to the variable $x$.

If $\text{cap}(c) > 0$ then $c$ is an asynchronous channel. A process $P_i$ can send a message $v$ through this channel, that is, the process can perform the transition $l^i:c\mathcal{L} v \mathcal{L} l'_i$ if and only if the channel is not full, that is, if the number of messages currently stored in $c$ is less than $\text{cap}(c)$. In this case, the message will be
stored in \( c \), according to a first-in, first-out discipline. Conversely, a process \( P_j \) can read a message from \( c \), that is, the process can perform the transition
\[
l_j \xrightarrow{c \ni x} l_j'
\]
if and only if the channel is not empty. In this case, the first message stored in \( c \) is assigned to the variable \( x \).

**Definition 2.28.** Let \( CS = [PG_1 | \cdots | PG_n] \) be a channel system over \( \langle \text{Var}, \text{Chan} \rangle \) with \( PG_i = \langle \text{Loc}_i, \text{Act}_i, \text{Effect}_i, \xrightarrow{i}, \text{Loc}_0, g_0 \rangle \), for \( i = 1, \ldots, n \). The transition system for \( CS \) is
\[
T(CS) = (S, A, \rightarrow, I, \Xi, L)
\]
where
- \( S = (\text{Loc}_1 \times \cdots \times \text{Loc}_n) \times \text{Asg(Var)} \times \text{Asg(Chan)} \),
- \( A = \text{Act}_1 \uplus \cdots \uplus \text{Act}_n \),
- \( \rightarrow \) is defined by the following rules:
  - for every \( a \in \text{Act}_i \)
    \[
    \langle l_1, \ldots, l_i, \ldots, l_n, \eta, \xi \rangle \xrightarrow{a} \langle l_1, \ldots, l'_i, \ldots, l_n, \text{Effect}(a, \eta), \xi \rangle
    \]
    if \( c \in \text{Chan} \) and \( \text{cap}(c) > 0 \) then
    \[
    \langle l_1, \ldots, l_i, \ldots, l_n, \eta, \xi \rangle \xrightarrow{\xi(a,c)} \langle l_1, \ldots, l'_i, \ldots, l_n, \eta[x := u_1], \xi[c := u_2 \ldots u_k] \rangle
    \]
    and
    \[
    \langle l_1, \ldots, l_i, \ldots, l_n, \eta, \xi \rangle \xrightarrow{\xi(a,c)} \langle l_1, \ldots, l'_i, \ldots, l_n, \eta[x := u_1 \ldots u_k] \rangle
    \]
    if \( c \in \text{Chan} \) and \( \text{cap}(c) = 0 \) then, for \( i \neq j \),
    \[
    \langle l_1, \ldots, l_i, \ldots, l_j \ldots, l_n, \eta, \xi \rangle \xrightarrow{\gamma_1} \langle l_1, \ldots, l'_i, \ldots, l'_j, \ldots, l_n, \eta[x := u], \xi \rangle
    \]
    and
    \[
    \langle l_1, \ldots, l_i, \ldots, l_j \ldots, l_n, \eta, \xi \rangle \xrightarrow{\gamma_2} \langle l_1, \ldots, l'_i, \ldots, l'_j, \ldots, l_n, \eta[x := u], \xi \rangle
    \]
\( I = \{ \langle l_1, \ldots, l_n, \eta, \xi_0 \rangle \mid l_i \in \text{Loc}_i \text{ and } \eta \downarrow_{\text{Var}_i} \vdash g_0, \text{ for } i = 1, \ldots, n \} \),

\( \Xi = \text{Loc}_1 \uplus \cdots \uplus \text{Loc}_n \uplus \text{Cond}(\text{Var}) \),

\( L(\langle l_1, \ldots, l_n, \eta, \xi \rangle) = \{ l_1, \ldots, l_n \} \cup \{ g \in \text{Cond}(\text{Var}) \mid \eta \vdash g \} \).
Chapter 3

Linear-time properties

So far we have seen how to model a system by a transition system. In addition, we will also need to consider a property (or properties) that we want the system to satisfy. Our goal is to check if the transition system verifies this property. In this chapter we discuss the classes of properties that we will be interested in verifying. To do so we can adopt either an action-based approach or a state-based approach. In an action-based approach we abstract away from states and consider only the action labels of the transitions. In a state-based approach, we abstract away from action labels and consider only the labels of the states. Most of the results herein can be presented in either of the approaches. In this chapter we consider a state-based approach and we will use the propositional symbols of the states to formulate the system properties. Hence, we assume fixed a set of propositional symbols $\Xi$.

3.1 Linear-Time properties

A linear-time property is a requirement on traces of a transition system and can be understood as a requirement over all words over $2^\Xi$. It specifies the set of traces that we want the transition system to exhibit in order to conveniently model our system.

Definition 3.1. A linear-time property (LT property) over $\Xi$ is a subset of $(2^\Xi)^\omega$.
A linear time property is just an $\omega$-language over the alphabet $2^\Xi$. Infinite words are enough for our purposes since we consider only transition systems without terminal states.

**Definition 3.2.** Let $P$ be an LT property over $\Xi$ and $T$ a transition system also over $\Xi$ without terminal states. Then, $T$ satisfies $P$, written $T \models P$, if $\text{Traces}(T) \subseteq P$. Let $\pi \in \text{Paths}(T)$. Then, $\pi$ satisfies $P$, written $\pi \models P$, if $\text{trace}(\pi) \in P$. Let $s$ be a state of $T$. Then $s$ satisfies $P$, written $s \models P$, if $\text{Traces}(s) \subseteq P$.

A transition system $T$ satisfies $P$ if all its traces are in $P$, that is, all the admissible behavior of the transition system are in $P$.

**Example 3.3.** Consider two traffic lights described by the following transition systems, with $\Xi_1 = \{\text{red}_1, \text{green}_1\}$ and $\Xi_2 = \{\text{red}_2, \text{green}_2\}$:

\begin{align*}
T_1: & \quad \begin{array}{c}
\text{red}_1 \\
\text{green}_1
\end{array} \\
\text{a} & \quad \text{a}
\end{align*}

\begin{align*}
T_2: & \quad \begin{array}{c}
\text{green}_2 \\
\text{red}_2
\end{array} \\
\text{a} & \quad \text{a}
\end{align*}

Let $H = \{a\}$ be the set of handshake actions and let $T = T_1 \parallel_H T_2$. Then $T$ is the transition system

\begin{align*}
\begin{array}{c}
\langle \text{red}_1, \text{green}_2 \rangle \\
\text{a} & \quad \text{a}
\end{array} \\
\langle \text{green}_1, \text{red}_2 \rangle
\end{align*}

Consider the LT property $P_1$ over $\Xi_1 \cup \Xi_2$ that states that “The first traffic light is infinitely often green”. This LT property corresponds to the set of infinite words over $2^{\Xi_1 \cup \Xi_2}$ of the form $v_0 v_1 v_2 \ldots$ such that $\text{green}_1 \in v_i$
3.1. LINEAR-TIME PROPERTIES

for infinitely many values of $i$. Examples of words in $P_1$ are

$$\{\text{red}_1, \text{green}_2\} \{\text{green}_1, \text{red}_2\} \{\text{red}_1, \text{green}_2\} \{\text{green}_1, \text{red}_2\} \ldots$$

$$\{\text{green}_1\} \{\text{green}_1\} \{\text{green}_1\} \{\text{green}_1\} \{\text{green}_1\} \ldots$$

$$\{\text{red}_1, \text{green}_1, \text{green}_2\} \{\text{green}_2\} \{\text{red}_1, \text{green}_1, \text{green}_2\} \{\text{green}_2\} \ldots$$

$$\emptyset \{\text{green}_1\} \emptyset \{\text{green}_1\} \emptyset \{\text{green}_1\} \emptyset \{\text{green}_1\} \ldots$$

On the other hand, infinite words of the form

$$\{\text{red}_1, \text{green}_2\} \{\text{green}_1, \text{red}_2\} \{\text{red}_1, \text{green}_2\} \{\text{red}_1, \text{green}_2\} \{\text{red}_1, \text{green}_2\} \ldots$$

$$\{\text{green}_1\} \{\text{green}_1\} \emptyset \emptyset \emptyset \ldots$$

$$\{\text{red}_1\} \{\text{red}_2\} \{\text{red}_1\} \{\text{red}_2\} \{\text{red}_1\} \{\text{red}_2\} \{\text{red}_1\} \{\text{red}_2\} \ldots$$

are not in $P_1$.

Consider now the LT property $P_2$ stating that “the traffic lights are never both green at the same time”. This LT property corresponds to the set of infinite words over $2^\Xi_1 \cup \Xi_2$ of the form $v_0 v_1 v_2 \ldots$ such that either $\text{green}_1 \notin v_i$ or $\text{green}_2 \notin v_i$ for every $i \in \mathbb{N}_0$. Examples of words in $P_2$ are

$$\{\text{red}_1, \text{green}_2\} \{\text{green}_1, \text{red}_2\} \{\text{red}_1, \text{green}_2\} \{\text{green}_1, \text{red}_2\} \ldots$$

$$\{\text{green}_1\} \{\text{green}_1\} \{\text{green}_1\} \{\text{green}_1\} \{\text{green}_1\} \ldots$$

$$\{\text{red}_1, \text{green}_1\} \{\text{red}_2, \text{green}_2\} \{\text{red}_1, \text{green}_1\} \{\text{red}_2, \text{green}_2\} \ldots$$

$$\emptyset \{\text{green}_1\} \emptyset \{\text{green}_2\} \emptyset \{\text{green}_1\} \emptyset \{\text{green}_2\} \ldots$$

$$\{\text{red}_1\} \{\text{red}_2\} \{\text{red}_1\} \{\text{red}_2\} \{\text{red}_1\} \{\text{red}_2\} \{\text{red}_1\} \{\text{red}_2\} \ldots$$

Whereas, the following infinite words are not in $P_2$:

$$\{\text{red}_1, \text{green}_2\} \{\text{green}_1, \text{green}_2\} \{\text{red}_1, \text{green}_2\} \{\text{green}_1, \text{green}_2\} \{\text{red}_1, \text{green}_2\} \ldots$$

$$\{\text{green}_1\} \{\text{green}_1, \text{green}_2\} \emptyset \emptyset \emptyset \ldots$$

$$\{\text{red}_1, \text{green}_1, \text{green}_2\} \{\text{green}_2\} \{\text{red}_1, \text{green}_1, \text{green}_2\} \{\text{green}_2\} \ldots$$

We leave it as an exercise to check that $T$ satisfies both $P_1$ and $P_2$. 
When two transition systems $T_1$ and $T_2$ exhibit the same behavior, that is, have the same set of traces it is natural to expect that they satisfy the same LT properties. Indeed, assume that $T_1$ and $T_2$ have the same traces and let $P$ be some arbitrary LT property. If $T_1 \vDash P$ then $\text{Traces}(T_1) \subseteq P$ which implies that $\text{Traces}(T_2) \subseteq P$, because $\text{Traces}(T_1) = \text{Traces}(T_2)$. Consequently, $T_2 \vDash P$. Conversely, if $T_1 \nvDash P$ then there is some trace $\sigma \in \text{Traces}(T_1)$ such that $\sigma \notin P$. Again, as $\text{Traces}(T_1) = \text{Traces}(T_2)$, $\sigma$ is also a trace of $T_2$ and, consequently, $T_2 \nvDash P$.

We start by considering trace inclusion. In software design, this is important because if $\text{Traces}(T_1) \subseteq \text{Traces}(T_2)$ this means that $T_1$ is a correct implementation of $T_2$. We can look at $T_2$ as the more abstract model and we can look at $T_1$ as one particular implementation. The inclusion means that $T_1$ cannot exhibit any behavior that is not present in $T_2$. For instance, in the example of the snack machine, a possible implementation of this machine could deliver alternately chocolates and cookies. The set of traces of this implementation is (strictly) included in the more abstract description of the machine given above.

The next result relates trace inclusion and LT properties. It states that all the properties satisfied by the more abstract model are also satisfied by any correct implementation, as it would be expected.

**Proposition 3.4.** Let $T_1$ and $T_2$ be two transition systems without terminal states and with the same set of propositional symbols $\Xi$. Then, the following statements are equivalent:

(a) $\text{Traces}(T_1) \subseteq \text{Traces}(T_2)$

(b) if $T_2 \vDash P$ then $T_1 \vDash P$, for any LT property $P$ over $\Xi$.

*Proof. (a $\rightarrow$ b): Assume that $\text{Traces}(T_1) \subseteq \text{Traces}(T_2)$ and that $T_2 \vDash P$, for some LT property $P$. Then, $\text{Traces}(T_2) \subseteq P$. Hence, $\text{Traces}(T_1) \subseteq P$ and, consequently, $T_1 \vDash P$.

(b $\rightarrow$ a): Assume that, for any LT property $P$, if $T_2 \vDash P$ then $T_1 \vDash P$. Let $P = \text{Traces}(T_2)$. Clearly $P$ is an LT property and $T_2 \vDash P$ because $\text{Traces}(T_2) \subseteq \text{Traces}(T_2) = P$. Then, $T_1 \vDash P$, that is, $\text{Traces}(T_1) \subseteq P = \text{Traces}(T_2)$ and the result follows. □*

Transition systems are said to be trace-equivalent if they have the same set of traces.
3.1. **LINEAR-TIME PROPERTIES**

Figure 3.1: Two snack machines.

**Definition 3.5.** Let $T_1$ and $T_2$ be two transition systems with sets of propositional symbols $\Xi_1$ and $\Xi_2$, respectively, and let $\Xi$ be a set of propositional symbols such that $\Xi \subseteq \Xi_1$ and $\Xi \subseteq \Xi_2$. Then, $T_1$ and $T_2$ are said to be trace-equivalent with respect to $\Xi$ if \( \text{Traces}_\Xi(T_1) = \text{Traces}_\Xi(T_2) \).

Proposition 3.4 implies that two trace-equivalent transition systems satisfy exactly the same LT properties, as it had already been motivated.

**Proposition 3.6.** Let $T_1$ and $T_2$ be two transition systems without terminal states and with the same set of propositional symbols $\Xi$. Then, the following statements are equivalent:

(a) \( \text{Traces}(T_1) = \text{Traces}(T_2) \)

(b) $T_1$ and $T_2$ satisfy the same LT properties over $\Xi$.

**Example 3.7.** Consider the two transition systems in Figure 3.1 that model a snack machine, and where the observable actions have been omitted for simplicity. We leave it as an exercise to show that the two machines are trace equivalent with respect to $\Xi = \{\text{coin}, \text{chocolate}, \text{cookies}\}$. 
3.2 Safety properties and invariants

Intuitively, a safety property states that something bad never happens. For instance, the LT property $P_2$ described in Example 3.3 is an example of a safety property. It states that the bad thing (having both traffic lights simultaneously green) never happens.

**Definition 3.8.** Let $\varphi$ be a propositional formula over $\Xi$. An LT property $P_{\varphi}^\text{inv}$ over $\Xi$ is an invariant for $\varphi$ if for every $\sigma \in P_{\varphi}^\text{inv}$, we have that $\sigma[i] \models \varphi$, for every $i \geq 0$. In this case, $\varphi$ is called an invariant condition (or state condition of) $P_{\varphi}^\text{inv}$.

Observe that $T \models P_{\varphi}^\text{inv}$ if and only if $\text{trace}(\pi) \in P_{\varphi}^\text{inv}$, for every $\pi \in \text{Paths}(T)$ if and only if $L(s) \models \varphi$, for every state $s$ in some path of $T$ if and only if $L(s) \models \varphi$, for every state $s \in \text{Reach}(T)$.

This means that if $T \models P_{\varphi}^\text{inv}$ then $\varphi$ must hold in every initial state and must be “preserved” by all transitions in the reachable fragment of the transition system.

Recall the safety property $P_2$ of Example 3.3. This property can be described as an invariant by the propositional formula

$$\varphi \equiv \neg (\text{green}_1 \land \text{green}_2)$$

or, equivalently, by

$$\varphi \equiv \neg \text{green}_1 \lor \neg \text{green}_2.$$

Hence, $P_2$ is just $P_{\varphi}^\text{inv}$.

Next, we focus on how to check if a transition system $T$ satisfies an invariant property $P_{\varphi}^\text{inv}$. To this end, we will use a slightly modified standard graph transversal algorithm to search the reachable states of the transition system for a state that does not satisfy $\varphi$. These algorithms can be used because the transition system is finite.

The algorithm in Figure 3.2 performs a depth-first search (DFS) on the state graph $G(T)$ searching for a bad state. It starts from the initial states and for each of these states visits all reachable states that have not yet been visited looking for a state where $\varphi$ does not hold. The algorithm stores in
set $R$ all the nodes that have already been visited and it terminates when all reachable nodes from $T$ have been visited. The stack $U$ maintains all the nodes that have yet to be visited, provided that they are not in $R$. This algorithm can be slightly improved by terminating the search once a bad state is found, that is, once a state where $\varphi$ does not hold is found.

Algorithm in Figure 3.3 is an adaptation of the previous, but instead of just returning \textit{true} or \textit{false}, in case a bad state is found it returns an initial path fragment leading to that state. We take advantage of the depth-first search stack $U$. When a state $s$ is found that violates $\varphi$, the (reversed) content of $U$ is a path fragment leading to that state.

\textbf{Example 3.9.} Recall the two traffic lights from Example 3.3 and let $T'$ be the transition system obtained by the interleaving of $T_1$ and $T_2$, that is, $T' = T_1 \parallel T_2$. The resulting transition system is depicted in Figure 3.4. Observe that the action labels were dropped, as it was discussed at the beginning of the chapter. Furthermore, as usual, we identify states with the corresponding labels.

We want to check if the transition system $T'$ satisfies $P_2 (=P_{\varphi_{\text{inv}}})$ from Example 3.3, that is, we want to check $T' \models P_{\varphi_{\text{inv}}}$ with $\varphi \equiv \neg \text{green}_1 \lor \neg \text{green}_2$.

First, we illustrate the algorithm in Figure 3.2. To make the description of the content of the variables easier we will write $g_i$ for $\text{green}_i$ and $r_i$ for $\text{red}_i$, respectively, with $i = 1, 2$. Initially, $R$ is set to $\emptyset$, $U$ is set to \textit{new} and $b$ is set to \textit{true}. Since there is only one initial node $\{r_1, g_2\}$, procedure \textit{visit} is only called once, for this node. Consequently, we focus on the call visit($\{r_1, g_2\}$).

After the first two assignments, the content of the variables is

$U = \{r_1, g_2\}$ \hspace{1cm} $R = \{\{r_1, g_2\}\}$ \hspace{1cm} $b = \text{true}$

The first step of the loop is executed by letting $s' = \text{top}(U) = \{r_1, g_2\}$. Looking at the transition system we observe that $\text{Suc}(\{r_1, g_2\}) = \{\{r_1, r_2\}, \{g_1, g_2\}\}$. Since $\text{Suc}(\{r_1, g_2\}) \subseteq R$ does not hold we choose one of the elements in $\text{Suc}(\{r_1, g_2\}) \setminus R$ and add it to $R$ and push it onto $U$. In this case we can choose either one of the states and we choose $s'' = \{r_1, r_2\}$. Consequently, after the execution of this step, the content of the variables
Input: finite transition system $T$ and propositional formula $\varphi$

Output: true if every reachable state of $T$ satisfies $\varphi$, false otherwise

set of states $R := \emptyset$;
stack of states $U := \text{new}$;
bool $b := \text{true}$;

forall $s \in I$ do
    if $s \not\in R$ then
        visit($s$);
    fi
od
return $b$

proc visit (state $s$)
    push($s, U$);
    $R := R \cup \{s\}$;
    repeat
        $s' := \text{top}(U)$;
        if $\text{Suc}(s') \subseteq R$ then
            pop($U$);
            $b := b \land (s' \models \varphi)$;
        else
            let $s'' \in \text{Suc}(s') \setminus R$
            push($s'', U$);
            $R := R \cup \{s''\}$;
        fi
    until $\text{empty}(U)$
endproc

Figure 3.2: DFS algorithm for invariant checking
Input: finite transition system $T$ and propositional formula $\varphi$
Output: true if every reachable state of $T$ satisfies $\varphi$, false otherwise, with a counterexample

set of states $R := \emptyset$
stack of states $U := \text{new}$
bool $b := \text{true}$

while $((I \setminus R) \neq \emptyset \land b)$ do
  let $s \in I \setminus R$
  visit($s$);
  od
if $b$ then
  return "yes"
else
  return ("no",reverse($U$))
fi

proc visit (state $s$)
  push($s$, $U$);
  $R := R \cup \{s\}$;
  repeat
    $s' := \text{top}(U)$;
    if $\text{Suc}(s') \subseteq R$ then
      pop($U$);
      $b := b \land (s' \models \varphi)$;
    else
      let $s'' \in \text{Suc}(s') \setminus R$
      push($s''$, $U$);
      $R := R \cup \{s''\}$;
    fi
  until $\text{empty}(U) \lor \neg b$
endproc

Figure 3.3: DFS algorithm for invariant checking revised
is
\[ U = \begin{cases} \{r_1, r_2\} \\ \{r_1, g_2\} \end{cases}, \quad R = \{\{r_1, g_2\}, \{r_1, r_2\}\} \quad b = true \]

The second step of the loop is similar. In this case the top of the stack is
\[ s' = \{r_1, r_2\} \] and \( \text{Suc}(\{r_1, r_2\}) = \{\{r_1, g_2\}, \{g_1, r_2\}\} \) and, once again, the condition \( \text{Suc}(s') \subseteq R \) does not hold. Then, the choice of \( s'' \) has to be \( \{g_1, r_2\} \) since \( \text{Suc}(\{r_1, r_2\}) \setminus R = \{\{g_1, r_2\}\} \). After the execution of the
second step, the content of the variables is

\[ U = \begin{cases} \{g_1, r_2\} \\ \{r_1, r_2\} \\ \{r_1, g_2\} \end{cases}, \quad R = \{\{r_1, g_2\}, \{r_1, r_2\}, \{g_1, r_2\}\} \quad b = true \]

In the third step of the loop, we have \( s' = \{g_1, r_2\} \) and after the execution of this step, the content of the variables is

\[ U = \begin{cases} \{g_1, g_2\} \\ \{g_1, r_2\} \\ \{r_1, r_2\} \\ \{r_1, g_2\} \end{cases}, \quad R = \{\{r_1, g_2\}, \{r_1, r_2\}, \{g_1, r_2\}, \{g_1, g_2\}\} \quad b = true \]

In the fourth step, \( s' = \{g_1, g_2\} \) and condition \( \text{Suc}(s') \subseteq R \) holds (and it will hold henceforth). The first element of \( U \) is removed and as \( s' \not \models \varphi \) (for obvious reasons) \( b \) is set to \( false \). Hence, after the fourth step, the content
of the variables is

\[ U = \{\{g_1, r_2\}, \{r_1, r_2\}, \{g_1, r_2\}, \{g_1, g_2\}\} \]

\[ R = \{\{r_1, g_2\}, \{r_1, r_2\}, \{g_1, r_2\}, \{g_1, g_2\}\} \]

\[ b = false \]

Our search could stop here, but the algorithm continues the search until the stack is empty. In any case, the end result is false. Observe that once \( b \) is set to false it will remain false until the end of the execution.

If we use the algorithm in Figure 3.3 the execution is similar up to the fourth step. After this step the search stops and the algorithm returns false as well as the path fragment corresponding to the contents of \( U \) reversed, that is, it will return the path fragment

\[ \{r_1, g_2\} \{r_1, r_2\} \{g_1, r_2\} \]

Observe that this path fragment does not include the last state of the transition system, where the property is violated. We leave it as an exercise to modify the algorithm so that, in case of failure, the path fragment includes the last state.

It is clear from observing the transition system that there is a simpler counterexample path fragment for \( \varphi \). Is this algorithm able to find that path fragment?

We now look into safety properties. Intuitively, a safety property \( P \) is an LT property such that any infinite word not in \( P \) must contain a bad prefix, where the bad thing has happened. Hence, no infinite word starting with this prefix is in \( P \).

**Definition 3.10.** An LT property \( P_{safe} \) over \( \Xi \) is called a safety property if for all words \( \sigma \in (2^\Xi)^\omega \setminus P_{safe} \) there exists a finite prefix \( \hat{\sigma} \) of \( \sigma \) such that

\[ P_{safe} \cap \{\sigma' \in (2^\Xi)^\omega \mid \hat{\sigma} \text{ is a finite prefix of } \sigma'\} = \emptyset. \]

Each such word \( \hat{\sigma} \) is called a bad prefix for \( P_{safe} \). A minimal bad prefix for \( P_{safe} \) is a bad prefix \( \hat{\sigma} \) for \( P_{safe} \) for which no proper prefix of \( \hat{\sigma} \) is a bad prefix for \( P_{safe} \). The set of all bad prefixes for \( P_{safe} \) is denoted by \( \text{BadPref}(P_{safe}) \), and the set of minimal bad prefixes for \( P_{safe} \) is denoted by \( \text{MinBadPref}(P_{safe}) \).
A minimal bad prefix is a bad prefix of minimal length. What can be said about a safety property \( P \) such that \( \varepsilon \in \text{BadPref}(P) \)?

**Lemma 3.11.** Let \( T \) be a transition system without terminal states and \( P_{\text{safe}} \) a safety property. Then, the following two conditions are equivalent:

1. \( T \models P_{\text{safe}} \),
2. \( \text{Traces}_{\text{fin}}(T) \cap \text{BadPref}(P_{\text{safe}}) = \emptyset \).

**Proof.**

(1 \( \rightarrow \) 2) We prove the result by contraposition. Assume that \( \text{Traces}_{\text{fin}}(T) \cap \text{BadPref}(P_{\text{safe}}) \neq \emptyset \) and let \( \hat{\sigma} \in \text{Traces}_{\text{fin}}(T) \cap \text{BadPref}(P_{\text{safe}}) \). Given that \( T \) has no terminal states then the finite trace \( \hat{\sigma} = v_0 v_1 \ldots v_n \) can be extended to an infinite trace \( \sigma = v_0 v_1 \ldots v_n v_{n+1} \ldots \in \text{Traces}(T) \). But, by definition of safety property, \( \sigma \not\in P_{\text{safe}} \) and, consequently, \( T \not\models P_{\text{safe}} \).

(2 \( \rightarrow \) 1) Once again, we prove this result by contraposition. Assume that \( T \not\models P_{\text{safe}} \). Then, \( \text{Traces}(T) \not\subseteq P_{\text{safe}} \), that is, there is some trace \( \sigma \in \text{Traces}(T) \) such that \( \sigma \not\in P_{\text{safe}} \). By definition of safety property, \( \sigma \) starts with some bad prefix \( \hat{\sigma} \), that is, \( \hat{\sigma} \in \text{BadPref}(P_{\text{safe}}) \). On the other hand, it is also easy to see that \( \hat{\sigma} \in \text{Traces}_{\text{fin}}(T) \) and so \( \text{Traces}_{\text{fin}}(T) \cap \text{BadPref}(P_{\text{safe}}) \neq \emptyset \).

\( \square \)

**Example 3.12.** Recall the vending machine from Example 2.2. A reasonable requirement for this machine is that the number of inserted coins is always at least the number delivered snacks (cookies or chocolates). That is, we want to prevent behaviors where more snacks are delivered than the number of inserted coins. Let \( \Xi = \{\text{pay}, \text{snack}\} \) and consider a different labeling function \( L \):

- \( L(\text{coin}) = \emptyset \)
- \( L(\text{choose}) = \{\text{pay}\} \)
- \( L(\text{chocolate}) = L(\text{cookies}) = \{\text{snack}\} \)

Then, the desired safety property is the set of all infinite words \( v_0 v_1 v_2 \ldots \) over \( 2^\Xi \) such that for every \( i \in \mathbb{N} \) the condition

\[
|\{j \mid j \leq i \text{ and pay} \in v_j\}| \geq |\{j \mid j \leq i \text{ and snack} \in v_j\}|
\]
3.2. SAFETY PROPERTIES AND INVARIANTS

holds. This an example of a safety property that is not an invariant. Examples of bad prefixes for this property are the following finite words

\[
\emptyset \{\text{snack}\}
\emptyset \{\text{pay}\} \{\text{snack}\} \{\text{snack}\}
\emptyset \{\text{pay}\} \{\text{snack}\} \{\text{snack}\} \emptyset \{\text{pay}\}
\]

The first two are minimal bad prefixes. The third one is not minimal since, in particular, the second word is a prefix of this one.

We leave it as an exercise to check that the snack machine of Example 2.2 satisfies this safety property.

There is an alternative characterization of safety, in terms of their closure.

**Definition 3.13.** Let \(\sigma \in (2^{\Xi})^\omega\) be a word and let \(\text{pref}(\sigma)\) denote the set of finite prefixes of \(\sigma\), i.e.,

\[
\text{pref}(\sigma) = \{\tilde{\sigma} \in (2^{\Xi})^* \mid \tilde{\sigma} \text{ is a finite prefix of } \sigma\}.
\]

Given an LT property \(P\) over \(\Xi\) then,

\[
\text{pref}(P) = \bigcup_{\sigma \in P} \text{pref}(\sigma).
\]

The closure of \(P\) is defined by

\[
closure(P) = \{\sigma \in (2^{\Xi})^\omega \mid \text{pref}(\sigma) \subseteq \text{pref}(P)\}.
\]

If \(\sigma = v_0 v_1 v_2 \ldots\) then \(\text{pref}(\sigma)\) is the infinite set of finite words containing \(\varepsilon, v_0, v_0 v_1, v_0 v_1 v_2, \ldots\). The closure of \(P\) is the set of infinite words whose finite prefixes are also prefixes of \(P\). Observe that \(P \subseteq closure(P)\) for all LT properties. The following result will be useful later.

**Lemma 3.14.** Let \(P, P'\) be LT properties. Then,

1. if \(P \subseteq P'\) then \(\text{closure}(P) \subseteq \text{closure}(P')\),

2. \(\text{closure}(P) \cup \text{closure}(P') = \text{closure}(P \cup P')\).
Proof. 
1. If $P \subseteq P'$ then clearly $\text{pref}(P) \subseteq \text{pref}(P')$. Consequently, for each $\sigma \in (2^\Xi)^\omega$, if $\text{pref}(\sigma) \subseteq \text{pref}(P)$ then $\text{pref}(\sigma) \subseteq \text{pref}(P')$. Hence, $\text{closure}(P) \subseteq \text{closure}(P')$.

2. As $P \subseteq P \cup P'$ then, by 1, it follows that $\text{closure}(P) \subseteq \text{closure}(P \cup P')$. By a similar argument, $\text{closure}(P') \subseteq \text{closure}(P \cup P')$. Hence, $\text{closure}(P) \cup \text{closure}(P') \subseteq \text{closure}(P \cup P')$.

For the converse, assume that $\sigma \in \text{closure}(P \cup P')$. Then, $\text{pref}(\sigma) \subseteq \text{pref}(P \cup P')$. As $\text{pref}(P \cup P') = \text{pref}(P) \cup \text{pref}(P')$ then any finite prefix of $\sigma$ is either in $\text{pref}(P)$ or in $\text{pref}(P')$ or in both. Furthermore, as $\sigma$ is an infinite word it has infinitely many finite prefixes which means that infinitely many of these prefixes belong to $\text{pref}(P)$ or to $\text{pref}(P')$ or both. Without loss of generality assume that $\text{pref}(\sigma) \cap \text{pref}(P)$ is infinite. This implies that $\text{pref}(\sigma) \subseteq \text{pref}(P)$ as we show now: assume, by contradiction, that there is $\hat{\sigma} \in \text{pref}(\sigma) \setminus \text{pref}(P)$ and that the length of $\hat{\sigma}$ is $k$. As $\text{pref}(\sigma) \cap \text{pref}(P)$ is infinite there is exists $\hat{\sigma}' \in \text{pref}(\sigma) \cap \text{pref}(P)$ with length greater than $k$. But, if $\hat{\sigma}' \in \text{pref}(P)$ then there exists $\sigma' \in P$ such that $\hat{\sigma}' \in \text{pref}(\sigma')$. Observe that as $\sigma' \in P$ then $\text{pref}(\sigma') \subseteq \text{pref}(P)$. Finally, as $\hat{\sigma}$ and $\hat{\sigma}'$ are both prefixes of $\sigma$ and the length of $\hat{\sigma}'$ is greater than the length of $\hat{\sigma}$ then $\hat{\sigma}$ must be a prefix of $\hat{\sigma}'$ and, consequently, must also be a prefix of $\sigma'$.

Lemma 3.15. Let $P$ be an LT property over $\Xi$. Then, $P$ is a safety property if and only if $\text{closure}(P) = P$.

Proof. $(\rightarrow)$ Assume that $P$ is a safety property. We only need to show that $\text{closure}(P) \subseteq P$. Assume that there is some $\sigma \in \text{closure}(P)$ such that $\sigma \notin P$. Then, there is some prefix $\hat{\sigma} = v_0 v_1 \ldots v_n$ of $\sigma$ such that $\hat{\sigma} \in \text{BadPref}(P)$. Since $\sigma \in \text{closure}(P)$ then, by definition of closure, $\text{pref}(\sigma) \subseteq \text{pref}(P)$. And, as $\hat{\sigma} \in \text{pref}(\sigma)$ then $\hat{\sigma} \in \text{pref}(P)$. Then, there is a word $\sigma' \in P$ such that $\hat{\sigma} \in \text{pref}(\sigma')$. But this means that there is a word $\sigma' \in P$ that has a bad prefix $\hat{\sigma}$, contradicting the fact that $P$ is a safety property.

$(\leftarrow)$ Assume now that $\text{closure}(P) = P$ and let $\sigma \in (2^\Xi)^\omega \setminus P$. We need to show that $\sigma$ starts with a bad prefix. Given that $\sigma \notin P$ then $\sigma \notin \text{closure}(P)$.
3.2. SAFETY PROPERTIES AND INVARIANTS

Hence, \( \text{pref}(\sigma) \not\subseteq \text{pref}(P) \), that is, there is a finite prefix \( \hat{\sigma} \) of \( \sigma \) not in \( \text{pref}(P) \). Hence, by definition of \( \text{pref}(P) \), for every word \( \sigma' \) such that \( \hat{\sigma} \) is a prefix of \( \sigma' \), it cannot belong to \( P \) because otherwise \( \hat{\sigma} \) would belong to \( \text{pref}(P) \). Hence \( \hat{\sigma} \) is a bad prefix for \( P \) and, by definition, \( P \) is a safety property.

The following two lemmas will be useful below.

**Lemma 3.16.** Let \( T \) be a transition system. Then, \( T \) satisfies the safety property \( \text{closure}(\text{Traces}(T)) \), that is, \( T \models \text{closure}(\text{Traces}(T)) \).

**Lemma 3.17.** Let \( P \) be an LT property. Then, \( \text{pref}(\text{closure}(P)) = \text{pref}(P) \).

The proofs of these results are left as exercises.

In the previous section, we saw that there is a relationship between trace inclusion and satisfaction of LT properties. However, this result was only established for infinite traces. We now show that we can establish a similar relationship between transition systems and satisfaction of safety properties, for finite traces.

**Proposition 3.18.** Let \( T_1 \) and \( T_2 \) be two transition systems without terminal states and with the same set of propositional symbols \( \Xi \). Then, the following statements are equivalent:

(a) \( \text{Traces}_{\text{fin}}(T_1) \subseteq \text{Traces}_{\text{fin}}(T_2) \)

(b) if \( T_2 \models P_{\text{safe}} \) then \( T_1 \models P_{\text{safe}} \), for any safety property \( P_{\text{safe}} \) over \( \Xi \).

*Proof. (a \( \rightarrow \) b)*: Assume that \( \text{Traces}_{\text{fin}}(T_1) \subseteq \text{Traces}_{\text{fin}}(T_2) \) and let \( P_{\text{safe}} \) be a safety property over \( \Xi \) such that \( T_2 \models P_{\text{safe}} \). This implies, by Lemma 3.11, that \( \text{Traces}_{\text{fin}}(T_2) \cap \text{BadPref}(P_{\text{safe}}) = \emptyset \). As \( \text{Traces}_{\text{fin}}(T_1) \subseteq \text{Traces}_{\text{fin}}(T_2) \) then \( \text{Traces}_{\text{fin}}(T_1) \cap \text{BadPref}(P_{\text{safe}}) = \emptyset \) and using again Lemma 3.11 we have that \( T_1 \models P_{\text{safe}} \).

(b \( \rightarrow \) a): Assume that (b) holds and let \( P_{\text{safe}} = \text{closure}(\text{Traces}(T_2)) \). \( P_{\text{safe}} \) is clearly a safety property, by Lemma 3.15, and moreover, by Lemma 3.16, we also know that \( T_2 \models P_{\text{safe}} \). Consequently, we have that \( T_1 \models P_{\text{safe}} \), that is

\[ \text{Traces}(T_1) \subseteq \text{closure}(\text{Traces}(T_2)) \].
This implies that
\[ \text{pref}(\text{Traces}(T_1)) \subseteq \text{pref}(\text{closure}(\text{Traces}(T_2))). \]

By Lemma 3.17, since \( \text{Traces}(T_2) \) is an LT property, we know that
\[ \text{pref}(\text{closure}(\text{Traces}(T_2))) = \text{pref}(\text{Traces}(T_2)). \]

Hence, as \( \text{Traces}_{\text{fin}}(T_1) = \text{pref}(\text{Traces}(T_1)) \) and \( \text{Traces}_{\text{fin}}(T_2) = \text{pref}(\text{Traces}(T_2)) \)

it follows that
\[ \text{Traces}_{\text{fin}}(T_1) \subseteq \text{Traces}_{\text{fin}}(T_2). \]

An immediate corollary of this result is the following.

**Proposition 3.19.** Let \( T_1 \) and \( T_2 \) be two transition systems without terminal states and with the same set of propositional symbols \( \Xi \). Then, the following statements are equivalent:

(a) \( \text{Traces}_{\text{fin}}(T_1) = \text{Traces}_{\text{fin}}(T_2) \)

(b) \( T_2 \models P_{\text{safe}} \) if and only if \( T_1 \models P_{\text{safe}} \), for any safety property \( P_{\text{safe}} \) over \( \Xi \).

Note that as we are considering transition systems without terminal states there is a strong relationship between trace inclusion and finite trace inclusion. For \( T_1 \) finite and \( T_2 \) without terminal states trace inclusion and finite trace inclusion coincide. This relationship is expressed by the following result.

**Proposition 3.20.** Let \( T_1 \) and \( T_2 \) be two transition systems with the same set of propositional symbols \( \Xi \) such that \( T_1 \) has no terminal states and \( T_2 \) is finite. Then, the following statements are equivalent:

(a) \( \text{Traces}(T_1) \subseteq \text{Traces}(T_2) \)

(b) \( \text{Traces}_{\text{fin}}(T_1) \subseteq \text{Traces}_{\text{fin}}(T_2) \).

**Proof.** Given that \( \text{Traces}_{\text{fin}}(T) = \text{pref}(\text{Traces}(T)) \) it is straightforward to see that \( \text{Traces}(T_1) \subseteq \text{Traces}(T_2) \) implies \( \text{Traces}_{\text{fin}}(T_1) \subseteq \text{Traces}_{\text{fin}}(T_2) \). To prove the converse assume that \( \text{Traces}_{\text{fin}}(T_1) \subseteq \text{Traces}_{\text{fin}}(T_2) \) and let \( \sigma \in \text{Traces}(T_1) \). As \( T_1 \) has no terminal states, the trace \( \sigma \) must be infinite, that
3.2. SAFETY PROPERTIES AND INVARIANTS

is, \( \sigma = v_0 v_1 v_2 \ldots \). We need to prove that there exists an infinite path \( \pi = s_0 s_1 s_2 \ldots \) in \( T_2 \) such that \( \text{trace}(\pi) = \sigma \).

Each finite prefix \( \hat{\sigma}^k \) of \( \sigma \) of length \( k \in \mathbb{N}_0 \) is in \( \text{Traces}_{T_0}(T_1) \) and consequently is also in \( \text{Traces}_{T_0}(T_2) \). Then, for each of these finite traces \( \hat{\sigma}^k \) there is a finite path \( \pi^k \) in \( T_2 \) such that \( \text{trace}(\pi^k) = \hat{\sigma}^k \). But although \( \hat{\sigma}^k \) is a prefix \( \hat{\sigma}^{k+1} \), it is not guaranteed that \( \pi^k \) is a prefix of \( \pi^{k+1} \) and so, in general, we cannot use this sequence \( \pi^0, \pi^1, \pi^2, \ldots \) of finite paths in \( T_2 \) to induce an infinite path \( \pi \) in \( T_2 \) such that \( \text{trace}(\pi) = \sigma \). But, as \( T_2 \) is finite, we are able to extract from \( \pi^0, \pi^1, \pi^2, \ldots \) an infinite subsequence \( \pi^{m_0}, \pi^{m_1}, \pi^{m_2}, \ldots \) such that \( \pi^{m_{i-1}} \) and \( \pi^{m_i} \) agree on the first \( i \) states, as we show now: for each \( n \in \mathbb{N} \), let \( \pi^{m_n} = s_0^{m_n} s_1^{m_n} s_2^{m_n} \ldots s_n^{m_n} \). Then \( \pi^{m_0} \) and \( \pi^{m_1} \) agree on the first state, i.e., \( s_0^{m_0} = s_0^{m_1} \), \( \pi^{m_1} \) and \( \pi^{m_2} \) agree on the first two states, i.e., \( s_0^{m_1} = s_0^{m_2} \) and \( s_1^{m_1} = s_1^{m_2} \), and so we also have \( s_0^{m_1} = s_0^{m_2} = s_0^{m_3} \). Let \( s_0 \) be \( s_0^{m_0} \), \( s_1 \) be \( s_1^{m_1} \), and so. Then,

\[
\begin{align*}
\pi^{m_0} &= s_0 \ldots \\
\pi^{m_1} &= s_0 s_1 \ldots \\
\pi^{m_2} &= s_0 s_1 s_2 \ldots
\end{align*}
\]

Clearly, the path \( \pi = s_0 s_1 s_2 \ldots \) is a path in \( T_2 \) and furthermore \( \text{trace}(\pi) = \sigma \). Hence, we can conclude that \( \sigma \in \text{Traces}(T_2) \).

All that remains to prove is that such a subsequence \( \pi^{m_0}, \pi^{m_1}, \pi^{m_2}, \ldots \) of finite paths does indeed exists. Let \( I_0, I_1, I_2, \ldots \) be an infinite sequence of infinite sets of natural numbers such that \( I_n \subseteq \{ m \in \mathbb{N}_0 \mid m \geq n \} \) and \( s_0, s_1, s_2, \ldots \) a sequence of states in \( T_2 \) such that for every \( n \in \mathbb{N}_0 \) the following hold:

(a) if \( n \geq 1 \) then \( I_{n-1} \supseteq I_n \),

(b) \( s_0 s_1 \ldots s_n \) is an initial, finite path fragment in \( T_2 \),

(c) for every \( m \in I_n \) it holds that \( s_0 s_1 \ldots s_n = s_0^m s_1^m \ldots s_n^m \).

Intuitively, each set \( I_n \) contains the indexes of the path fragments \( \pi^m \) of the initial sequence of finite paths \( \pi^0, \pi^1, \pi^2, \ldots \) that agree on the first \( n \) states. The proof that such sequence of sets and states can be defined proceeds by induction on \( n \in \mathbb{N}_0 \).

Base (\( n = 0 \)): Recall that \( T_2 \) is finite and, consequently, has a finite set of initial states. Consider now the set of all initial states from all the finite
paths in the sequence $\pi^0, \pi^1, \pi^2, \ldots$, that is, consider the set $\{s_0^m \mid m \in \mathbb{N}_0\}$. This set is clearly finite. Hence, there must be an initial state in $T_2$ that appears repeated in an infinite number of paths from the sequence $\pi^0, \pi^1, \pi^2, \ldots$. We choose $s_0$ to be that initial state and we choose $I_0$ to be the set of indexes of the paths in $\pi^0, \pi^1, \pi^2, \ldots$ that have $s_0$ as the first state. Clearly $s_0$ and $I_0$ fulfill the desired conditions.

Step $(n+1)$: Assume that the states $s_0, s_1, \ldots, s_n$ and the sets $I_0, I_1, \ldots, I_n$ have already been defined. As $T_2$ is finite then $\text{Suc}(s_n)$ is a finite set. But we know that $I_n$ is infinite and that for every $m \in I_n$, $s_n = s_m^n$. Hence, among all the finite paths in $\pi^0, \pi^1, \pi^2, \ldots$ whose indexes are in $I_n$ there must exist an infinite number of these whose state $n+1$ is the same because $\text{Suc}(s_n)$ is finite. Let $s_{n+1}$ be one of these states and let $I_{n+1}$ be the set of indexes of the finite paths $\pi^0, \pi^1, \pi^2, \ldots$ with indexes already in $I_n$ that have $s_{n+1}$ as successor of $s_n$. The three conditions follow by construction.

3.3 Liveness properties

Safety properties express that “something bad never happens”. For instance, two traffic lights at a street junction are not green at the same time. An algorithm can fulfill a safety property simply by doing nothing. As a complement to safety properties there are properties that require some progress from the system. These are called liveness properties. However, while safety properties are violated in finite time (by a finite run), liveness properties are violated in infinite time (infinite runs).

A liveness property is an LT property such that no finite prefix is ruled out, that is, that every finite prefix can be extended in such a way that the resulting infinite trace satisfies the liveness property being considered.

**Definition 3.21.** An LT property $P_{\text{live}}$ over $\Sigma$ is called a liveness property if $\text{pref}(P_{\text{live}}) = (2^\Sigma)^*$.

This means that a liveness property $P_{\text{live}}$ is an LT property such that every finite word can be extended to an infinite word that is in $P$, that is, for every finite word $w \in (2^\Sigma)^*$ there is an infinite word $\sigma \in (2^\Sigma)^\omega$ such that $w.\sigma \in P_{\text{live}}$.

**Example 3.22.** Recall Example 3.3. It is reasonable to require that each traffic light is green infinitely often. That is, we are interested in the set $P$
of all infinite words $v_0 v_1 v_2 \ldots$ such that

$$(\forall k \in \mathbb{N}_0 \exists j \geq k.\text{green}_1 \in v_j) \land (\forall k \in \mathbb{N}_0 \exists j \geq k.\text{green}_2 \in v_j).$$

It is not very difficult to see that that $P$ is a liveness property. Given any finite word over $2^\Xi$ it is always possible to extend it to an infinite word satisfying the above condition. Furthermore, it is also very easy to see that the transition system $T = T_1 \parallel_H T_2$ defined in Example 3.3 satisfies this liveness property.

**Example 3.23.** Recall Example 2.2. Consider a liveness property similar to the one in the previous example, where we require that chocolates and cookies are distributed infinitely often. That is, in this case we are interested in the liveness property $P_{\text{sm}}^{\text{live}}$

$$(\forall k \in \mathbb{N}_0 \exists j \geq k.\text{chocolate} \in v_j) \land (\forall k \in \mathbb{N}_0 \exists j \geq k.\text{cookies} \in v_j).$$

In this case, it is easy to see that the transition system $T_{\text{sm}}$ does not satisfy this property. In fact, consider the following trace of $T_{\text{sm}}$

$$\{\text{coin}\} \{\text{choose}\} \{\text{chocolate}\} \{\text{coin}\} \{\text{choose}\} \{\text{chocolate}\} \ldots$$

This trace is not in $P_{\text{sm}}^{\text{live}}$ and so $T_{\text{sm}} \not\models P_{\text{sm}}^{\text{live}}$.

### 3.4 Safety versus liveness properties

To end this chapter, we study the relationship between safety and liveness properties. In particular, we answer the following questions:

- Are safety properties and liveness properties disjoint?
- Is any LT property a safety or a liveness property?

The answer to the first question is that the only common property is the full language $(2^\Xi)\omega$, that is, the nonrestrictive property. The answer to the second question is that for every LT property $P$ there is an equivalent property $P'$ that is a combination of a safety property and a liveness property.
Lemma 3.24. The only LT property over $\Xi$ that is both a safety and a liveness property is $(2^\Xi)^\omega$.

Proof. Assume that $P$ is a liveness property. Then $\text{pref}(P) = (2^\Xi)^*$. This implies that $\text{closure}(P) = (2^\Xi)^\omega$. Now, if $P$ is also a safety property, then $\text{closure}(P) = P$, that is, $P = (2^\Xi)^\omega$. \qed

We now present the decomposition theorem that answers the second question asked at the beginning of this section.

Theorem 3.25. Let $P$ be an LT property. Then, there exists a safety property $P_{\text{safe}}$ and a liveness property $P_{\text{live}}$ such that

$$P = P_{\text{safe}} \cap P_{\text{live}}.$$ 

Proof. Let $P$ be an LT property over $\Xi$. It was already shown that $P \subseteq \text{closure}(P)$. Hence, $P = \text{closure}(P) \cap P$ which can be rewritten as

$$P = \text{closure}(P) \cap (P \cup ((2^\Xi)^\omega \setminus \text{closure}(P))).$$

Let $P_{\text{safe}} = \text{closure}(P)$ and $P_{\text{live}} = P \cup ((2^\Xi)^\omega \setminus \text{closure}(P))$. The fact that $P_{\text{safe}}$ is a safety property is an immediate consequence of Lemma 3.15. It remains to be proved is that $P_{\text{live}}$ is indeed a liveness property. To do so we need to prove that $\text{pref}(P_{\text{live}}) = (2^\Xi)^*$. It not very difficult to see that this is equivalent to $\text{closure}(P_{\text{live}}) = (2^\Xi)^\omega$. Furthermore, it is always the case that $\text{closure}(P_{\text{live}}) \subseteq (2^\Xi)^\omega$ and so we only need to prove that $(2^\Xi)^\omega \subseteq \text{closure}(P_{\text{live}})$. As

$$\text{closure}(P_{\text{live}}) = \text{closure}(P \cup ((2^\Xi)^\omega \setminus \text{closure}(P))),$$

by Lemma 3.14, we have that

$$\text{closure}(P_{\text{live}}) = \text{closure}(P) \cup \text{closure}((2^\Xi)^\omega \setminus \text{closure}(P)).$$

Furthermore, we also know that $P' \subseteq \text{closure}(P')$ holds for any LT property $P'$ which implies that

$$(2^\Xi)^\omega \setminus \text{closure}(P) \subseteq \text{closure}((2^\Xi)^\omega \setminus \text{closure}(P))$$

must hold. This implies that

$$\text{closure}(P) \cup ((2^\Xi)^\omega \setminus \text{closure}(P)) \subseteq \text{closure}(P) \cup \text{closure}((2^\Xi)^\omega \setminus \text{closure}(P)).$$
and consequently

\[(2^\Xi)\omega \subseteq \text{closure}(P_{\text{live}}).\]

The following result shows that the decomposition described in the proof of the previous theorem is the sharpest decomposition, that is, \(P_{\text{safe}} = \text{closure}(P)\) is the strongest safety property and \(P_{\text{live}} = P \cup ((2^\Xi)\omega \setminus \text{closure}(P))\) is the weakest liveness property that can serve for a decomposition of \(P\).

**Theorem 3.26.** Let \(P\) be an LT property and \(P = P_{\text{safe}} \cap P_{\text{live}}\) where \(P_{\text{safe}}\) is a safety property and \(P_{\text{live}}\) is a liveness property. Then

1. \(\text{closure}(P) \subseteq P_{\text{safe}}\);
2. \(P_{\text{live}} \subseteq P \cup ((2^\Xi)\omega \setminus \text{closure}(P))\).

The proof of this result is left as an exercise.
Chapter 4

Regular properties

In this chapter we address the problem of verifying whether a certain class of safety properties, liveness properties, and some other linear-time properties is satisfied by a system. We start by considering regular safety properties, that is, safety properties whose set of bad prefixes is a regular language. Then, we generalize the construction to \( \omega \)-regular properties. These include regular safety properties but also regular liveness properties.

As we saw in Chapter 1, a regular language is denoted by a regular expression. However, regular expressions are not very useful from an operational perspective. To this end, finite automata are usually used as an operational semantics for regular languages. The class of regular safety properties will be defined using finite automata to characterize the set of bad prefixes.

To address the notion of \( \omega \)-regular properties we need to consider \( \omega \)-regular languages. Recall that these are denoted by \( \omega \)-regular expressions. And, like regular expressions, these are not very useful from an operational perspective. In this case, we will use Büchi automata to characterize \( \omega \)-regular languages.

4.1 Finite Automata

Definition 4.1. A nondeterministic finite automaton (NFA) is a tuple \( \mathcal{A} = \langle Q, \Sigma, \delta, Q_0, F \rangle \) where:

- \( Q \) is a finite set of states;
• $\Sigma$ is a nonempty finite set (alphabet);
• $\delta : Q \times \Sigma \rightarrow 2^Q$ (the transition function);
• $Q_0 \subseteq Q$ is a set of initial states;
• $F \subseteq Q$ is a set of final states.

The size of $A$, denoted by $|A|$ is the number of states and transitions in $A$, that is,
$$|A| = |Q| + \sum_{q \in Q} \sum_{v \in \sigma} |\delta(q, v)|.$$ 

The elements in $\Sigma$ are the symbols on which the automaton is defined. The set $Q_0$ is a (possibly empty) set of initial states from which the automaton may start. The set $F$ is the (possibly empty) set of final states, also called accept states.

**Example 4.2.** Consider the NFA depicted in Figure 4.1. In this case, $A = \langle Q, \Sigma, \delta, Q_0, F \rangle$ is such that

- $Q = \{q_0, q_1, q_2\}$,
- $\Sigma = \{0, 1\}$,
- $Q_0 = \{q_0\}$,
- $F = \{q_2\}$,
- $\delta$ is defined by

  $$\begin{align*}
  \delta(q_0, 0) &= \emptyset & \delta(q_0, 1) &= \{q_1\} \\
  \delta(q_1, 0) &= \{q_1, q_2\} & \delta(q_1, 1) &= \{q_1\} \\
  \delta(q_2, 0) &= \emptyset & \delta(q_2, 1) &= \emptyset
  \end{align*}$$

Initial states are marked with an incoming arrow without source and accept states are drawn with a double circle.

Given an NFA $A$ we can extend the transition function to words of $\Sigma^*$. We define $\delta^*$ inductively.

**Definition 4.3.** Let $A = \langle Q, \Sigma, \delta, Q_0, F \rangle$ be an NFA. The function $\delta^* : Q \times \Sigma^* \rightarrow 2^Q$ is inductively defined as follows:
4.1. FINITE AUTOMATA

Figure 4.1: An example of an NFA.

- \( \delta^*(q, \varepsilon) = \{ q \} \);
- \( \delta^*(q, vw) = \bigcup_{q' \in \delta(q,v)} \delta^*(q', w) \), for \( v \in \Sigma \) and \( w \in \Sigma^* \).

The intuitive behavior of an NFA is as follows. Initially the automaton is given an input word \( w \in \Sigma^* \). For each symbol in the word, the automaton moves to another state, according to the transition function \( \delta \). That is, if the automaton is at a certain state \( q \in Q \) and the next symbol in \( w \) is \( v \) then the next state is chosen from \( \delta(q, v) \). If more than one state is available at \( \delta(q, v) \) then the choice of the next state is made nondeterministically. This process is repeated until the entire word is analyzed or until a state is reached where no transition for the current symbol is available. In this case, the automaton halts and the input word is rejected. If the entire word has been analyzed then the automaton halts. If the current state is an accept state then the word is accepted, otherwise it is rejected.

The possible behaviors of an NFA for a given input word are defined by the notion of run. For a given input word there might be several possible behaviors (runs), some of which may be accepting while others may be rejecting. A word is accepted by an NFA if there is at least one accepting run.

**Definition 4.4.** Let \( A = \langle Q, \Sigma, \delta, Q_0, F \rangle \) be an NFA. The accepted language of \( A \) is the set

\[ L(A) = \{ w \in \Sigma^* \mid \delta^*(q_0, w) \cap F \neq \emptyset \text{ for some } q_0 \in Q_0 \}. \]

**Definition 4.5.** Let \( A = \langle Q, \Sigma, \delta, Q_0, F \rangle \) be an NFA and \( w = v_1 \ldots v_n \in \Sigma^* \). A run for \( w \) in \( A \) is a finite sequence of states \( q_0 \ldots q_n \) such that

- \( q_0 \in Q_0 \);
• \( q_{i+1} \in \delta(q_i, v_{i+1}) \), for every \( 0 \leq i < n \).

The run is accepting if \( q_n \in F \).

**Lemma 4.6.** Let \( A = \langle Q, \Sigma, \delta, Q_0, F \rangle \) be an NFA. Then,

\[ \mathcal{L}(A) = \{ w \in \Sigma^* \mid \text{there exists an accepting run for } w \text{ in } A \} \]

**Example 4.7.** Recall the NFA of Example 4.2. Examples of runs are

- \( q_0 \) for the empty word \( \varepsilon \),
- \( q_0 q_1 q_1 \) for the words 10 and 11,
- \( q_0 q_1 q_1 q_1 \) for the words 100, 101, 110 and 111,
- \( q_0 q_1 q_2 \) for the word 10.

An accepting run is a run that ends in an accepting state, in this case, \( q_2 \). From the runs above, the only accepting run is \( q_0 q_1 q_2 \) for 10. Observe, in particular, that the word 10 has two possible runs: \( q_0 q_1 q_1 \) and \( q_0 q_1 q_2 \). The first is not accepting while the second is. Consequently, the word 10 is accepted by the automaton, that is, it belongs to the accepted language of \( A \), written \( 10 \in \mathcal{L}(A) \). Conversely, the word 11 is not accepted by \( A \) because the only run for this word is \( q_0 q_1 q_1 \) and this run is not accepting. Finally, observe that there is no run for the word 00.

It is not very difficult to see that the accepted language of \( A \) is the set of all finite words over \( \{0, 1\} \) that start with a 1 and end with a 0, that is, the language denoted by the regular expression \( 1(0 + 1)^*0 \).

It can be shown that the language accepted by an NFA is a regular language, that is, given an NFA \( A \) it is always possible to generate a regular expression \( E_A \) such that \( \mathcal{L}(A) = \mathcal{L}(E_A) \). Conversely, every regular language is accepted by some NFA, that is, given a regular expression \( E \) it is always possible to construct an NFA \( A_E \) such that \( \mathcal{L}(E) = \mathcal{L}(A_E) \). We state without proof the following results.

**Proposition 4.8.** Let \( \mathcal{L} \) be a regular language over \( \Sigma \). Then, there is an NFA \( A \) with alphabet \( \Sigma \) such that \( \mathcal{L} = \mathcal{L}(A) \).

**Proposition 4.9.** Let \( A \) be an NFA with alphabet \( \Sigma \). Then \( \mathcal{L}(A) \) is a regular language over \( \Sigma \).
4.1. FINITE AUTOMATA

Definition 4.10. Two NFAs $\mathcal{A}$ and $\mathcal{A}'$ with the same alphabet are called equivalent if $L(\mathcal{A}) = L(\mathcal{A}')$.

An important problem in automata theory, called the emptiness problem, is to decide if, given an NFA $\mathcal{A}$, its accepted language is empty, that is, decide if $L(\mathcal{A}) = \emptyset$. It is not very difficult to see that $L(\mathcal{A})$ is nonempty if there is at least one run that ends in some final state. Hence, the emptiness problem is equivalent to finding a final state $q \in F$ that is reachable from an initial state $q_0 \in Q_0$. This can be easily checked using a depth first search traversal that encounters all reachable states and checks if any of them is final.

Definition 4.11. Let $\mathcal{A} = \langle Q, \Sigma, \delta, Q_0, F \rangle$ be an NFA and $q \in Q$. The set of reachable states from $q$ is

$$Reach(q) = \bigcup_{w \in \Sigma^*} \delta^*(q, w).$$

Theorem 4.12. Given an NFA $\mathcal{A} = \langle Q, \Sigma, \delta, Q_0, F \rangle$, there exists $q_0 \in Q_0$ and $q \in F$ such that $q \in Reach(q_0)$ if and only if $L(\mathcal{A}) \neq \emptyset$.

As it was mentioned in Chapter 1, regular languages enjoy some interesting closure properties. Namely, they are closed for union, intersection and complementation. The closure under union was already discussed in Chapter 1. Herein, we focus on closure under intersection and complementation, that is, we show that if $L_1$ and $L_2$ are regular languages then so are $L_1 \cap L_2$ and $\overline{L}_1 = \Sigma^* \setminus L_1$.

We start by sketching a proof for the closure under intersection. To this end, we construct the synchronous product of NFAs. The idea is to let the two automata run the same input word simultaneously. If either one of them stops before the end of the word then the word is rejected. Otherwise, the word is only accepted if both automata accept it, that is, if both automata reach a final state.

Definition 4.13. The product automaton of NFAs $\mathcal{A}_1$ and $\mathcal{A}_2$ with the same alphabet is the NFA

$$\mathcal{A}_1 \otimes \mathcal{A}_2 = \langle Q_1 \times Q_2, \Sigma, \delta, Q_{01} \times Q_{02}, F_1 \times F_2 \rangle$$

where $\delta : (Q_1 \times Q_2) \times \Sigma \rightarrow 2^{Q_1 \times Q_2}$ is such that

$$\delta((q_1, q_2), v) = \delta_1(q_1, v) \times \delta_2(q_2, v)$$
for every \( q_1 \in Q_1, q_2 \in Q_2 \) and \( v \in \Sigma \).

It is not very difficult to see that \( \mathcal{L}(A_1 \otimes A_2) = \mathcal{L}(A_1) \cap \mathcal{L}(A_2) \). The interested reader is asked to prove such a result in Exercise 4.4.

**Example 4.14.** Let \( A_1 \) be the NFA of Example 4.2 and let \( A_2 \) be the NFA defined in Figure 4.2. Recall that \( A_1 \) accepts all the sequences of 0s and 1s that start with 1 and end with 0. It is not very difficult to see that \( A_2 \) accepts only the sequences 10 and 101.

![An NFA that accepts the sequences 10 and 101.](image)

The relevant part of the NFA \( A_1 \otimes A_2 \) is depicted in Figure 4.3. For simplicity, we only present the reachable states (the complete NFA will have 12 states, most of which are not reachable). Observe that the only initial state is \( \langle q_0, q'_0 \rangle \) and that the only final states are \( \langle q_2, q'_2 \rangle \) and \( \langle q_2, q'_3 \rangle \), but this last state is not reachable. The relevant part of the transition function for this automaton is defined as follows:

\[
\begin{align*}
\delta(\langle q_0, q'_0 \rangle, 0) &= \delta_1(q_0, 0) \times \delta_2(q'_0, 0) = \emptyset \\
\delta(\langle q_0, q'_0 \rangle, 1) &= \delta_1(q_0, 1) \times \delta_2(q'_0, 1) = \{q_1\} \times \{q'_1\} = \{\langle q_1, q'_1 \rangle\} \\
\delta(\langle q_1, q'_1 \rangle, 0) &= \delta_1(q_1, 0) \times \delta_2(q'_1, 0) = \{q_1, q_2\} \times \{q'_2\} = \{\langle q_1, q'_2 \rangle, \langle q_2, q'_2 \rangle\} \\
\delta(\langle q_1, q'_1 \rangle, 1) &= \delta_1(q_1, 1) \times \delta_2(q'_1, 1) = \{q_1\} \times \emptyset = \emptyset \\
\delta(\langle q_1, q'_2 \rangle, 0) &= \delta_1(q_1, 0) \times \delta_2(q'_2, 0) = \{q_1, q_2\} \times \emptyset = \emptyset \\
\delta(\langle q_1, q'_2 \rangle, 1) &= \delta_1(q_1, 1) \times \delta_2(q'_2, 1) = \{q_1\} \times \{q'_3\} = \{\langle q_1, q'_3 \rangle\} \\
\delta(\langle q_1, q'_3 \rangle, 0) &= \delta_1(q_1, 0) \times \delta_2(q'_3, 0) = \{q_1, q_2\} \times \emptyset = \emptyset \\
\delta(\langle q_1, q'_3 \rangle, 1) &= \delta_1(q_1, 1) \times \delta_2(q'_3, 1) = \{q_1\} \times \emptyset = \emptyset \\
\delta(\langle q_2, q'_2 \rangle, 0) &= \delta_1(q_2, 0) \times \delta_2(q'_2, 0) = \emptyset \times \emptyset = \emptyset \\
\delta(\langle q_2, q'_2 \rangle, 1) &= \delta_1(q_2, 1) \times \delta_2(q'_2, 1) = \emptyset \times \{q'_3\} = \emptyset
\end{align*}
\]

Analyzing this automaton, we conclude that the only accepted word is the sequence 10, exactly the only word in \( \mathcal{L}(A_1) \cap \mathcal{L}(A_2) \).
4.1. FINITE AUTOMATA

We now consider the complementation operator for regular languages. Let \( \mathcal{L} \) be a regular language. We know from Proposition 4.9 that there is an NFA \( A \) such that \( \mathcal{L} = \mathcal{L}(A) \). We want to construct an NFA \( A' \) that accepts the complement language \( \Sigma^* \setminus \mathcal{L} \). One might be led to thinking that swapping final states with non-final states would be enough. This is in fact the main idea behind the construction, but some care must be taken. Consider, for instance, the NFA of Example 4.2. If we simply swapped final states with non-final states then, in the new automaton, \( q_0 \) and \( q_1 \) would be final states. Consequently, words like 11 and 101, that are not accepted in the original automaton would be part of the accepting language of the new automaton. However, words like 10 that are in the original language would still be accepted by the new automaton because, for instance, the run \( q_0 q_1 q_1 \) is a run for 10 and is now accepting because \( q_1 \) is now a final state. Hence, before we proceed, we need to consider deterministic finite automata (DFA).

**Definition 4.15.** An NFA \( A = \langle Q, \Sigma, \delta, Q_0, F \rangle \) is called **deterministic** if \( |Q_0| \leq 1 \) and \( |\delta(q, v)| \leq 1 \), for every \( q \in Q \) and \( v \in \Sigma \).

**Definition 4.16.** A DFA \( A = \langle Q, \Sigma, \delta, Q_0, F \rangle \) is called **total** if \( |Q_0| = 1 \) and \( |\delta(q, v)| = 1 \), for every \( q \in Q \) and \( v \in \Sigma \).

Observe that DFAs (and total DFAs) are particular cases of NFAs and, consequently, the accepted language of a DFA is a regular language. It is straightforward to prove that total DFAs have exactly one run for each word.
in $\Sigma^*$. In Exercise 4.5, the reader is asked to prove that given an NFA $A$ there exists a DFA $D_A$ that accepts exactly the same language.

Hence, complementing a total DFA is now very simple: we just swap final states with non-final states. The complete procedure to show the complement of a regular language $L$ is still a regular language is straightforward. Given that $L$ is a regular language we know, by Proposition 4.9 that there is an NFA $A$ such that $L = \mathcal{L}(A)$. Exercise 4.5 provides a total DFA $D$ that accepts the same language as $A$, that is, $L(D) = \mathcal{L}(A) = L$. Swapping the final states with the non-final states of $D$ we obtain a total DFA $\overline{D}$ that accepts the complement language of $D$, that is, $L(\overline{D}) = L(D) = \overline{L}$, and this is clearly a regular language.

### 4.2 Regular safety properties

In this section we focus on the verification of regular safety properties. These are safety properties whose set of bad prefixes is a regular language and, consequently, is accepted by some NFA. The main result of this section establishes that checking a regular safety property on a finite transition system can be reduced to invariant checking on another transition system (obtained from the product of the original transition system and the automaton that recognizes the bad prefixes of the property).

**Definition 4.17.** A safety property $P_{\text{safe}}$ over $\Xi$ is called regular if its set of bad prefixes constitutes a regular language over $2^\Xi$.

For instance, if $P_{\text{safe}}$ is a regular safety property over $\Xi = \{p, q\}$ then the set of bad prefixes contains words over the alphabet $2^\Xi = \{\emptyset, \{p\}, \{q\}, \{p, q\}\}$.

Observe that, in particular, every invariant is a regular safety property. Let $P_{\text{inv}}^{\varphi}$ be an invariant property with invariant condition $\varphi$. Recall that, in order for $P_{\text{inv}}^{\varphi}$ to be satisfied by a transition system then $\varphi$ must hold in every reachable state of the transition system. This means that the set of bad prefixes contains all the sequences of the form $v_0 v_1 v_2 \ldots v_n$ such that $v_i \not\models \varphi$ for some $0 \leq i \leq n$. This set is a regular language since it can be denoted by $\varphi^* (\neg \varphi) \text{true}^*$. In the previous expression we are overloading notation. When we write $\varphi^*$ we mean all the finite words $v_0 v_1 v_2 \ldots v_n$ over $2^\Xi$ such every $v_i \models \varphi$, for
4.2. REGULAR SAFETY PROPERTIES

Figure 4.4: An NFA accepting the bad prefixes of $P_{\text{inv}}^\varphi$.

$i = 0, \ldots, n$. Similarly, when we write $\neg \varphi$ we mean all the sets $v \in 2^\Xi$ such that $v \models \neg \varphi$. Finally, by $true^*$ we mean all the finite sequences over $2^\Xi$.

For instance, if $\Xi = \{p, q\}$ and $\varphi = p \land (\neg q)$ then $\varphi$ stands for $\{p\}$, $\neg \varphi$ stands for $\emptyset + \{q\} + \{p, q\}$, and $true$ stands for $\emptyset + \{p\} + \{q\} + \{p, q\}$. The set $BadPref(P_{\text{inv}}^\varphi)$ is denoted by the regular expression:

$$
\bigl(\{p\}^* \{\emptyset + \{q\} + \{p, q\}\}\bigl(\emptyset + \{p\} + \{q\} + \{p, q\}\bigr)^* \neg \varphi \bigl) \bigl(\emptyset + \{q\} + \{p, q\}\bigr)^* true^*
$$

This set can also be defined as the accepted language of the NFA in Figure 4.4.

The set of minimal bad prefixes is also a regular language. It is characterized by the regular expression $\varphi^*(\neg \varphi)$ and is recognized by the NFA in Figure 4.4 by omitting the self-loop in the state $q_1$.

**Lemma 4.18.** Let $P_{\text{safe}}$ be a safety property. Then, $P_{\text{safe}}$ is a regular safety property if and only if the set of minimal bad prefixes for $P_{\text{safe}}$ is regular.

The proof of this lemma is left as an exercise.

### 4.2.1 Verifying regular safety properties

Let $P_{\text{safe}}$ be a regular safety property and $A$ an automaton for recognizing the set of bad prefixes of $P_{\text{safe}}$, that is, $L(A) = BadPref(P_{\text{safe}})$ and $\varepsilon \notin BadPref(P_{\text{safe}})$. Observe that if $\varepsilon \in BadPref(P_{\text{safe}})$ then $P_{\text{safe}} = \emptyset$. In addition, let $T$ be a finite transition system without terminal states (over $\Xi$). We want to verify if $T \models P_{\text{safe}}$. Using Lemma 3.11

$$T \models P_{\text{safe}} \text{ if and only if } Traces_{\text{fin}}(T) \cap BadPref(P_{\text{safe}}) = \emptyset$$

$$\text{ if and only if } Traces_{\text{fin}}(T) \cap L(A) = \emptyset.$$
So, to check if $T \models P_{\text{safe}}$ it is enough to verify if $\text{Traces}_{\text{fin}}(T) \cap L(A) = \emptyset$. We adopt a similar strategy as the one used for checking if the languages of two automata $A_1$ and $A_2$ intersect:

$$L(A_1) \cap L(A_2) = \emptyset \text{ if and only if } L(A_1 \otimes A_2) = \emptyset.$$ 

In this case, we need to build the product of a transition system and an automaton.

**Definition 4.19.** Let $T = \langle S, A, \rightarrow, I, \Xi, L \rangle$ be a finite transition system without terminal states and $A = \langle Q, 2^\Xi, \delta, Q_0, F \rangle$ an NFA with $Q_0 \cap F = \emptyset$. The *product transition system* $T \otimes A$ is

$$\langle S \times Q, A, \rightarrow', I', \Xi', L' \rangle$$

where:

- $\rightarrow'$ is the smallest relation defined by the rule

$$s \xrightarrow{a} t \quad q \in \delta(p, L(t))$$

$$\langle s, p \rangle \xrightarrow{a} \langle t, q \rangle$$

- $I' = \{\langle s_0, q \rangle \mid s_0 \in I, \exists q_0 \in Q_0. q \in \delta(q_0, L(s_0))\}$,

- $\Xi' = Q$,

- $L' : S \times Q \rightarrow 2^Q$ is defined by $L'((s, q)) = \{q\}$.

A path fragment $\hat{\pi} = s_0 s_1 \ldots s_n$ in $T$ can be extended to the path fragment

$$\langle s_0, q_1 \rangle \langle s_1, q_2 \rangle \ldots \langle s_n, q_{n+1} \rangle$$
in $T \otimes A$, if the sequence $q_0, q_1, \ldots, q_{n+1}$ is a run in $A$ for

$$L(s_0) L(s_1) \ldots L(s_n)$$

for some initial state $q_0$ of $A$, which is exactly $\text{trace}(\hat{\pi})$.

We now show that the verification of a regular safety property $P_{\text{safe}}$ by a transition system can be reduced to checking an invariant on the product $T \otimes A$, where $A$ is the NFA accepting the language $\text{BadPref}(P_{\text{safe}})$. To
4.2. REGULAR SAFETY PROPERTIES

this end, let $P_{\text{inv}(A)}$ be the invariant property over $\Xi' = Q$ induced by the propositional formula

$$\bigwedge_{q \in F} \neg q$$

In the sequel, we will denote this formula by $\neg F$. Observe that $\neg F$ only holds in non accepting states.

**Theorem 4.20.** Let $T$ be a transition system without terminal states over $\Xi$, $P_{\text{safe}}$ a regular safety property over $\Xi$ and let $A$ be an NFA with alphabet $2^\Xi$ such that $Q_0 \cap F = \emptyset$ and $L(A) = \text{BadPref}(P_{\text{safe}})$. Then, the following statements are equivalent:

1. $T \models P_{\text{safe}}$
2. $\text{Traces}_{\text{fin}}(T) \cap L(A) = \emptyset$
3. $T \otimes A \models P_{\text{inv}(A)}$.

**Proof.** Let $T = \langle S, A, \rightarrow, I, \Xi, L \rangle$ and $A = \langle Q, 2^\Xi, \delta, Q_0, F \rangle$.

The equivalence between statement (1) and statement (2) is an immediate consequence of Lemma 3.11.

$(3 \rightarrow 1)$ Assume that $T \not\models P_{\text{safe}}$. Then, there is a trace $\sigma$ such that $\sigma \in \text{Traces}(T)$ and $\sigma \notin P_{\text{safe}}$. If $\sigma \notin P_{\text{safe}}$, by definition of safety property, there is a finite bad prefix $\hat{\sigma}$ of $\sigma$, with respect to $P_{\text{safe}}$. Then, there is a finite initial path fragment $\hat{\pi} = s_0 s_1 \ldots s_n \in \text{Paths}_{\text{fin}}(T)$ such that

$$\text{trace}(\hat{\pi}) = \hat{\sigma} = L(s_0) L(s_1) \ldots L(s_n) \in \text{BadPref}(P_{\text{safe}}).$$

Then $\hat{\sigma} = L(s_0) L(s_1) \ldots L(s_n) \in L(A)$ which means that there exists an accepting run $q_0 q_1, \ldots q_{n+1}$ in $A$ for $\hat{\sigma}$, that is,

- $q_0 \in Q_0$,
- $q_{n+1} \in F$,
- $q_{i+1} \in \delta(q_i, L(s_i))$, for $i = 0, \ldots, n$.

hence,

$$\langle s_0, q_1 \rangle \langle s_1, q_2 \rangle \ldots \langle s_n, q_{n+1} \rangle$$
CHAPTER 4. REGULAR PROPERTIES

is an initial path fragment in $T \otimes A$. Furthermore we also have that

$$\langle s_n, q_{n+1} \rangle \not\models F$$

given that $L'(\langle s_n, q_{n+1} \rangle) = \{q_{n+1}\}$ and $q_{n+1} \in F$. Consequently, $T \otimes A \not\models P_{inv(A)}$.

(2 $\rightarrow$ 3) Assume that $T \otimes A \not\models P_{inv(A)}$. Then, there is a trace $\sigma' \in Traces(T \otimes A)$ such that $\sigma' \not\in P_{inv(A)}$. If $\sigma' \not\in P_{inv(A)}$ then there is some $n \geq 0$ such that $\sigma'[n] \not\models F$, that is, $\sigma'[n] \models F$. Consider the path fragment $\hat{\sigma}' = \langle s_0, q_1 \rangle \langle s_1, q_2 \rangle \ldots \langle s_n, q_{n+1} \rangle$ such that $L'(\hat{\sigma}') = \hat{\sigma}' = \{q_1\, q_2\} \ldots \{q_{n+1}\}$. Observe that $\hat{\sigma}'[n] = \{q_{n+1}\}$ and thus, $q_{n+1} \in F$. Let $\hat{\pi}$ be the projection of $\hat{\sigma}'$ on the first component, that is, $\hat{\pi} = s_0 s_1 \ldots s_n$, which is clearly a path fragment in $T$, by definition of $T \otimes A$. Then, $\text{trace}(\hat{\pi}) \in \text{Traces}_{inv}(T)$. Observe that $\langle s_0, q_1 \rangle$ is an initial state in $T \otimes A$ and so $q_1 \in \delta(q_0, L(s_0))$. Furthermore, again by definition of $T \otimes A$, it must also be the case that $q_{i+1} \in \delta(q_i, L(s_i))$, for $i = 1, \ldots, n$. Hence, $q_0 q_1 \ldots q_{n+1}$ is an accepting run for $L(s_0) L(s_1) \ldots L(s_n)$ in $A$ which means that $\text{trace}(\hat{\pi}) = L(s_0) L(s_1) \ldots L(s_n) \in \mathcal{L}(A)$. Thus, $\text{Traces}_{inv}(T) \cap \mathcal{L}(A) \neq \emptyset$.

The theorem means that in order to check if the transition system $T$ satisfies a regular safety property $P_{safe}$ it is enough to check if no state $\langle s, q \rangle$ in $T \otimes A$ is reachable, for any final state $q \in F$. This corresponds to checking if the invariant $P_{inv(A)}$ (stating that no final state is visited) is satisfied, which can be done by the algorithm in Figure 3.3 of Chapter 3. If the invariant holds then we can conclude that $T$ satisfies the property $P_{safe}$. Otherwise, the algorithm will provide a counter-example refuting $P_{safe}$. This counter-example will be a path fragment $\langle s_0, q_1 \rangle \langle s_1, q_2 \rangle \ldots \langle s_n, q_{n+1} \rangle$ in the transition system $T \otimes A$ that leads to a final state $q_{n+1} \in F$ of $A$. If we project this path fragment on the states of $T$ we obtain a path fragment $s_0 s_1 \ldots s_n$ in $T$. Consider now the trace of this path fragment:

$$L(s_0) L(s_1) \ldots L(s_n) = v_0 v_1 \ldots v_n$$

By construction, we know that $q_1 \in \delta(q_0, L(s_0))$, $q_2 \in \delta(q_1, L(s_1))$, and so on. Consequently, $q_0 q_1 \ldots q_{n+1}$ is a run for $v_0 v_1 \ldots v_n$ in $A$ and since $q_{n+1} \in F$ this run is accepting. Hence, $v_0 v_1 \ldots v_n$ is accepted by $A$, that is, $v_0 v_1 \ldots v_n$ is a bad prefix of $P_{safe}$. Consequently, the path fragment $s_0 s_1 \ldots s_n$ is a useful error indicator since $\text{trace}(\pi) \not\in P_{safe}$ for every path $\pi \in \text{Paths}(T)$ such that $s_0 s_1 \ldots s_n \in \text{pref}(\pi)$. 
4.3 Büchi Automata

We now focus on defining automata for recognizing \( \omega \)-regular languages. Recall from Chapter 1 that an \( \omega \)-regular language \( L_\omega \) is an \( \omega \)-language that is denoted by an \( \omega \)-regular expression. The finite automata that we saw before are not adequate for recognizing this class of languages since they operate on finite words. An automaton recognizing infinite words is called an \( \omega \)-automaton. An accepting run for an \( \omega \)-automaton has to check the entire input word and, consequently, has to be infinite. This means that we need to define acceptance criteria for infinite runs. There are several kinds of \( \omega \)-automata. Herein, we consider non-deterministic Büchi automata (NBA). The definition of an NBA is similar to the definition of an NFA. The main difference resides in the definition of accepting run and of accepted language.

Corollary 4.21. Let \( T, P_{\text{safe}} \) and \( A \) be as in Theorem 4.20. Then, for each initial path fragment \( \langle s_0, q_1 \rangle \langle s_1, q_2 \rangle \cdots \langle s_n, q_{n+1} \rangle \) in \( T \otimes A \):

\[ q_1, \ldots, q_n \notin F \text{ and } q_{n+1} \in F \text{ implies } \text{trace}(s_0 s_1 \ldots s_n) \in L(A). \]
CHAPTER 4. REGULAR PROPERTIES

The intuitive meaning for an infinite run to be accepting is that it needs to visit a final state of the automaton infinitely often.

**Definition 4.22.** A nondeterministic Büchi automaton (NBA) $A$ is a tuple $\langle Q, \Sigma, \delta, Q_0, F \rangle$ where

- $Q$ is a finite set of states;
- $\Sigma$ is a nonempty finite set (alphabet);
- $\delta : Q \times \Sigma \rightarrow 2^Q$ (the transition function);
- $Q_0 \subseteq Q$ is a set of initial states;
- $F \subseteq Q$ is a set of final states.

A run for $\sigma = v_0 v_1 v_2 \cdots \in \Sigma^\omega$ in $A$ is an infinite sequence of states $q_0 q_1 q_2 \ldots$ such that

- $q_0 \in Q_0$;
- $q_{i+1} \in \delta(q_i, v_i)$, for every $i \in \mathbb{N}$.

The run is accepting if $q_i \in F$ for infinitely many indices $i \in \mathbb{N}$. The accepted language of $A$ is the set

$$L_\omega(A) = \{ \sigma \in \Sigma^\omega \mid \text{there exists an accepting run for } \sigma \text{ in } A \}.$$  

The set $F$ is called the acceptance set and its elements are also called accept states. Since the set of states $Q$ of $A$ is finite and an acceptance run for an infinite word is always infinite this means that at least one state from $Q$ has to be visited infinitely often. Then, the acceptance of a word depends on whether the set of all states that appear infinitely often in the run contains a final state or not.

The size of $A$, denoted by $|A|$ is the number of states and transitions in $A$, that is,

$$|A| = |Q| + \sum_{q \in Q} \sum_{v \in \Sigma} |\delta(q, v)|.$$
Example 4.23. Consider the NBA $A$ depicted in Figure 4.6, with alphabet $\{0, 1\}$. Let us consider some examples. The word $1^\omega$ has only one run in $A$ that is the run $q_0 q_0 q_0 \ldots$, or, in short, $q_0^\omega$. This is clearly not an accepting run and so $1^\omega \not\in L^\omega(A)$. The word $0^\omega$ has the run $q_0 q_1^\omega$ which is clearly an accepting run since the accept state $q_1$ appears infinitely often in this run. Consequently, $0^\omega \in L^\omega(A)$ holds. Examining the automaton in detail we conclude that in order for a run to be accepting it has to visit the only accept state $q_1$ infinitely often. And the only was to reach that state is by the symbol 0. Hence, the language of $A$ is composed of all the sequences of 0s and 1s that contains infinitely many 0s. Recall from Chapter 1 that this is an $\omega$-regular language that is denoted by the $\omega$-regular expression $(1^*0)\omega$.

Example 4.24. Consider a pedestrian traffic light with $\Xi = \{\text{red}, \text{green}\}$ and let $P$ be the LT property over $2^\Xi$ expressing that the green light is on infinitely often. We can adapt the previous NBA (see Figure 4.7) so that it recognizes this $\omega$-language.

The alphabet of the automaton is $2^\Xi$. Hence, each arrow is labelled with a set of propositional symbols. Recall from Section 4.2 that when we write green we mean all the sets of propositional symbols that correspond to valuations that satisfy green, that is, all the subsets of $\Xi$ that contain the
propositional symbol \( \text{green} \), and when we write \( \neg \text{green} \) we mean all the sets of propositional symbols that correspond to valuations that satisfy \( \neg \text{green} \), that is, all the subsets of \( \Xi \) that do not contain \text{green}.

Like for regular languages, NBAs can be used to characterize \( \omega \)-regular languages. We prove this result by proving that any \( \omega \)-regular language is recognized by some NBA and that the language accepted by an NBA is \( \omega \)-regular. To prove the first statement, we prove some auxiliary results that mimic the construction of \( \omega \)-regular expressions.

**Lemma 4.25.** Let \( A_1 \) and \( A_2 \) be two NBAs with the same alphabet \( \Sigma \). Then, there is an NBA \( A \) such that \( \mathcal{L}_\omega(A) = \mathcal{L}_\omega(A_1) \cup \mathcal{L}_\omega(A_2) \).

**Proof.** Let \( A_i = (Q_i, \Sigma, \delta_i, Q_{0_i}, F_i) \), with \( i = 1, 2 \), and assume without loss of generality that \( Q_1 \cap Q_2 = \emptyset \). Let \( A = (Q, \Sigma, \delta, Q_0, F) \) be the NBA defined as follows:

- \( Q = Q_1 \cup Q_2 \),
- \( \delta(q, v) = \begin{cases} 
\delta_1(q, v) & \text{if } q \in Q_1 \\
\delta_2(q, v) & \text{if } q \in Q_2 
\end{cases} \)
- \( Q_0 = Q_{0_1} \cup Q_{0_2} \),
- \( F = F_1 \cup F_2 \)

Observe that \( \delta \) is well defined because \( Q_1 \cap Q_2 = \emptyset \). It is clear that any accepting run for some word \( \sigma \) in \( A_1 \) or \( A_2 \) is also an accepting run for \( \sigma \) in \( A \). Hence, it follows \( \mathcal{L}_\omega(A_1) \cup \mathcal{L}_\omega(A_2) \subseteq \mathcal{L}_\omega(A) \).

For the converse, assume that \( \sigma \in \mathcal{L}_\omega(A) \). Then, there is an accepting run \( q_0q_1 \ldots \) for \( \sigma \) in \( A \). Given that \( Q_1 \cap Q_2 = \emptyset \) then either \( q_0 \in Q_{0_1} \) or \( q_0 \in Q_{0_2} \), but not both. If \( q_0 \in Q_{0_1} \) then \( q_0q_1 \ldots \) is an accepting run for \( \sigma \) in \( A_1 \). Otherwise, \( q_0q_1 \ldots \) is an accepting run for \( \sigma \) in \( A_2 \). Consequently, \( \mathcal{L}_\omega(A) \subseteq \mathcal{L}_\omega(A_1) \cup \mathcal{L}_\omega(A_2) \).

The details of the proof are left as an exercise. \( \square \)

We will denote this automaton by \( A_1 + A_2 \). Next, we prove that from an NFA accepting a regular language \( \mathcal{L} \) such that \( \varepsilon \notin \mathcal{L} \) we can define an NBA accepting the \( \omega \)-regular language \( \mathcal{L}_\omega \).
Lemma 4.26. Let $A_1$ be an NFA with alphabet $\Sigma$ such that $\varepsilon \not\in L(A_1)$. Then, there is an NBA $A$ such that $L_\omega(A) = L(A_1)^\omega$.

Proof. Let $A_1 = \langle Q_1, \Sigma, \delta_1, Q_{0_1}, F_1 \rangle$. Clearly, no initial state can be a final state (otherwise, $\varepsilon \in L(A_1)$). Furthermore, assume that no initial state of $A_1$ has an incoming arrow. If this is not the case, then $A_1$ can be modified to an NFA $A'_1$ as follows

- $Q'_1 = Q_1 \cup \{q_n\}$
- $\delta'_1$ is such that
  - $\delta'_1(q, a) = \delta_1(q, a)$ for $q \neq q_n$
  - $\delta'_1(q_n, a) = \bigcup_{q_0 \in Q_{0_1}} \delta_1(q_0, a)$
- $Q'_{0_1} = \{q_n\}$
- $F'_1 = F_1$

This new NFA accepts the same language as $A_1$ and has the desired property of having initial states with no incoming arrows and such that no initial state is accepting.

So, from now on assume that $A_1$ is in the above conditions. We now build an NBA $A = \langle Q, \Sigma, \delta, Q_0, F \rangle$ such that $L_\omega(A) = L(A_1)^\omega$. The basic idea is to add to this new automaton a new transition leading to the initial states of $A$ for each transition in $A_1$ that leads to a final state, so that the process of accepting a word from $L(A_1)$ may be restarted, infinitely often. To this end, we let

- $Q = Q_1$,
- $\delta(q, a) = \begin{cases} 
\delta_1(q, a) & \text{if } \delta_1(q, a) \cap F_1 = \emptyset \\
\delta_1(q, a) \cup Q_{0_1} & \text{otherwise,}
\end{cases}$
- $Q_0 = Q_{0_1}$
- $F = Q_{0_1}$

We leave as an exercise to prove that $L_\omega(A) = L(A_1)^\omega$. \qed
The third and final step of our construction is, given an NFA $A_1$ and an NBA $A_2$, construct an NBA accepting the language $L(A_1).L_\omega(A_2)$.

**Lemma 4.27.** Let $A_1$ be an NFA and $A_2$ be an NBA with alphabet $\Sigma$. Then, there is an NBA $A$ such that $L_\omega(A) = L(A_1).L_\omega(A_2)$.

**Proof.** Let $A_1 = (Q_1, \Sigma, \delta_1, Q_{01}, F_1)$ and $A_2 = (Q_2, \Sigma, \delta_2, Q_{02}, F_2)$ be such that $Q_1 \cap Q_2 = \emptyset$. Let $A = (Q, \Sigma, \delta, Q_0, F)$ be defined as follows:

- $Q = Q_1 \cup Q_2$,
- $\delta(q, a) = \begin{cases} \delta_1(q, a) & \text{if } q \in Q_1 \text{ and } \delta_1(q, a) \cap F_1 = \emptyset \\ \delta_1(q, a) \cup Q_{02} & \text{if } q \in Q_1 \text{ and } \delta_1(q, a) \cap F_1 \neq \emptyset \\ \delta_2(q, a) & \text{if } q \in Q_2,
- $Q_0 = \begin{cases} Q_{01} & \text{if } Q_{01} \cap F_1 = \emptyset \\ Q_{01} \cup Q_{02} & \text{otherwise}
- $F = F_2$.

The proof that $L_\omega(A) = L(A_1).L_\omega(A_2)$ is left as an exercise. \(\square\)

**Proposition 4.28.** Let $L_\omega$ be an $\omega$-regular language over $\Sigma$. Then, there is an NBA $A$ with alphabet $\Sigma$ such that $L_\omega = L_\omega(A)$.

**Proof.** If $L$ is an $\omega$-regular language then there is a $\omega$-regular expression $E_1.(F_1)\omega + \cdots + E_n.(F_n)\omega$ such that $L_\omega(L) = L_\omega(E_1.(F_1)\omega + \cdots + E_n.(F_n)\omega)$ for regular expressions $E_1, \ldots, E_n, F_1, \ldots, F_n$ such that $\varepsilon \notin L(F_i)$, for $i = 1, \ldots, n$. By Proposition 4.8 there exist NFAs $A_{e_1}, \ldots, A_{e_n}, A_{f_1}, \ldots, A_{f_n}$ such that $L(A_{e_i}) = L(E_i)$ and $L(A_{f_i}) = L(F_i)$, for $i = 1, \ldots, n$.

Using Lemma 4.26, there are NBAs $A'_{f_1}, \ldots, A'_{f_n}$ such that

$\begin{align*}
L_\omega(A'_{f_i}) &= L(A_{f_i})\omega = L(F_i)\omega, \text{ for } i = 1, \ldots, n.
\end{align*}$

By Lemma 4.27, there are NBAs $A_{e_{f_1}}, \ldots, A_{e_{f_n}}$ such that

$\begin{align*}
L_\omega(A_{e_{f_i}}) &= L(A_{e_i}).L(A_{f_i})\omega = L(E_i).L(F_i)\omega, \text{ for } i = 1, \ldots, n.
\end{align*}$

Finally, by Lemma 4.25, there is an NBA $A$ such that

$\begin{align*}
L_\omega(A) &= L_\omega(A_{e_{f_1}}) \cup \cdots \cup L_\omega(A_{e_{f_n}}) \\
&= L(E_1).L(F_1)\omega \cup \cdots \cup L(E_n).L(F_n)\omega \\
&= L_\omega.
\end{align*}$
We now prove the converse result, that is, that the accepted language of an NBA is an \( \omega \)-regular language.

**Proposition 4.29.** Let \( \mathcal{A} \) be an NBA with alphabet \( \Sigma \). Then, \( \mathcal{L}_\omega(\mathcal{A}) \) is an \( \omega \)-regular language over \( \Sigma \).

**Proof.** Let \( \mathcal{A} = \langle Q, \Sigma, \delta, Q_0, F \rangle \) be an NBA and for each pair of states \( q, p \in Q \) let \( \mathcal{A}_{qp} \) be the NFA \( \langle Q, \Sigma, \delta, \{q\}, \{p\} \rangle \), that is, \( \mathcal{A}_{qp} \) is the NFA recognizing the regular language consisting of all the finite words \( w \in \Sigma^* \) that have a run in \( \mathcal{A} \) leading from \( q \) to \( p \). Let us denote this language by \( \mathcal{L}_{qp} \), that is, \( \mathcal{L}_{qp} = \mathcal{L}(\mathcal{A}_{qp}) = \{ w \in \Sigma^* \mid p \in \delta^*(q, w) \} \).

Consider now an infinite word \( \sigma \in \mathcal{L}_\omega(\mathcal{A}) \). For this word there is an accepting run \( q_0 q_1 q_2 \ldots \) in \( \mathcal{A} \). By definition of accepting run, this run must contain infinite states that appear in \( F \) and since \( F \) is finite then at least one of its elements \( q_f \in F \) must appear infinitely often in the run. Hence, we may split \( \sigma \) into an infinite sequence of nonempty finite subwords \( w_0, w_1, w_2, \ldots \in \Sigma^* \) such that

- \( w_0 \in \mathcal{L}_{q_0 q_f} \),
- \( w_i \in \mathcal{L}_{q_f q_f}, \) for \( i \geq 1 \),
- \( \sigma = w_0 w_1 w_2 \ldots \)

Since none of the sequences \( w_i \), with \( i \geq 1 \), is empty then we can conclude that \( \sigma \in \mathcal{L}_{q_0 q_f} \cdot (\mathcal{L}_{q_f q_f} \setminus \{ \varepsilon \})^\omega \), with \( q_0 \in Q_0 \) and \( q_f \in F \), and consequently,

\[
\mathcal{L}_\omega(\mathcal{A}) \subseteq \bigcup_{q_0 \in Q_0, q_f \in F} \mathcal{L}_{q_0 q_f} \cdot (\mathcal{L}_{q_f q_f} \setminus \{ \varepsilon \})^\omega.
\]

Now, let \( \sigma \in \mathcal{L}_{q_0 q_f} \cdot (\mathcal{L}_{q_f q_f} \setminus \{ \varepsilon \})^\omega \), for some initial state \( q_0 \) and final state \( q_f \). It is not very difficult to see that, under these circumstances, there must exist an accepting run for \( \sigma \) in \( \mathcal{A} \). Consequently, \( \sigma \in \mathcal{L}_\omega(\mathcal{A}) \) and so,

\[
\bigcup_{q_0 \in Q_0, q_f \in F} \mathcal{L}_{q_0 q_f} \cdot (\mathcal{L}_{q_f q_f} \setminus \{ \varepsilon \})^\omega \subseteq \mathcal{L}_\omega(\mathcal{A}).
\]
Now, observe that the sets $Q_0$ and $F$ are finite and that each language $L_{q_0 q_f}$ and $L_{q_f q_f}$ is accepted by an NFA and thus is regular. Hence

$$L_\omega(A) = \bigcup_{q_0 \in Q_0, q_f \in F} L_{q_0 q_f} \cdot (L_{q_f q_f} \setminus \{\varepsilon\})^\omega$$

is an $\omega$-regular language.

From Proposition 4.28 and Proposition 4.29 we may conclude that the class of $\omega$-regular languages coincides with the class of languages accepted by NBAs.

We now address the problem of checking whether the language accepted by an NBA is empty. In this case, an analysis of the underlying graph is sufficient for our purposes, as we will see.

**Lemma 4.30.** Let $A = (Q, \Sigma, \delta, Q_0, F)$ be an NBA. Then, the following two statements are equivalent:

1. $L_\omega(A) \neq \emptyset$;

2. There exists a reachable accept state $q$ that belong to a cycle in $A$, that is,

$$\exists q_0 \in Q_0 \exists q \in F \exists w \in \Sigma^* \exists w' \in \Sigma^+ q \in \delta^*(q_0, w) \cap \delta^*(q, w').$$

**Proof.** ($1 \rightarrow 2$): If $L_\omega(A) \neq \emptyset$ then there is $\sigma = v_0 v_1 v_2 \cdots \in L_\omega(A)$. Let $q_0, q_1, q_2, \ldots$ be an accepting run in $A$ for $\sigma$. By definition of accepting run, and since $F$ is finite, there is a final state $q \in F$ that appears repeated infinitely often in the run. Let $i, j \in \mathbb{N}$ be two indices such that $0 \leq i < j$ and $q_i = q_j = q$. Now, let $w$ be the finite word $v_0 v_1 \ldots v_{i-1}$ and let $w'$ be the finite word $v_i v_{i+1} \ldots v_{j-1}$. Clearly, $q_i \in \delta^*(q_0, w)$ and $q_j \in \delta^*(q_i, w')$. Hence, as $q = q_i = q_j$ condition (2) follows.

($2 \rightarrow 1$): Assume that $q_0, q, w, w'$ are as in condition (2). It is not very difficult to see that the infinite word $\sigma = w.(w')^\omega$ has a run $q_0 \ldots q \ldots q \ldots$ that is accepting in $A$, since the final state $q$ appears infinitely often in it. Consequently, $\sigma \in L_\omega(A)$ holds and so $L_\omega(A) \neq \emptyset$. 

This lemma states that in order to check the emptiness problem for NBA we just need to investigate the underlying graph, explore all the reachable
nodes and check if any of these reachable nodes is final and belongs to a cycle.

As NBAs can be used to recognize $\omega$-regular languages, we say that two NBAs are equivalent when they recognize the same $\omega$-language.

**Definition 4.31.** Two NBAs $A_1$ and $A_2$ with the same alphabet are called equivalent if $L_\omega(A_1) = L_\omega(A_2)$, written $A_1 \equiv A_2$.

**Example 4.32.** Consider the NBA $A_1$ depicted in Figure 4.8 over the alphabet $2^{\{p_1, p_2\}}$ over the set of propositional symbols $\Xi = \{p_1, p_2\}$. Recall from Section 4.2 that when we write $p_1$ we mean all the sets of propositional symbols that correspond to valuations that satisfy $p_1$, that is, the sets $\{p_1\}, \{p_1, p_2\}$. Similarly, when we write $\neg p_1$ we mean all the sets of propositional symbols that correspond to valuations that do not satisfy $p_1$, that is, the sets $\emptyset, \{p_2\}$. Then, the language accepted by $A_1$ represents the liveness property that $p_1$ holds infinitely often and $p_2$ holds infinitely often.

Consider now the NBA $A_2$ depicted if Figure 4.9 over the same alphabet. Like in the previous case, the language accepted by this NBA represents also the liveness property that $p_1$ holds infinitely often and $p_2$ holds infinitely often. Consequently, we can conclude that $A_1 \equiv A_2$.

For technical reasons it is often useful to assume that an NBA is non-blocking, that is, that for each state and for each input symbol there is always a possible transition.
Definition 4.33. Let $\mathcal{A} = (Q, \Sigma, \delta, Q_0, F)$ be an NBA. Then $\mathcal{A}$ is called nonblocking if $\delta(q, v) \neq \emptyset$ for every $q \in Q$ and $v \in \Sigma$.

In Exercise 4.11, the reader is asked to show that given an NBA it is always possible to find an equivalent nonblocking NBA.

There are other types of $\omega$-automata that can be used as models for $\omega$-regular languages and are equally expressive as NBAs, but have more general acceptance conditions. Herein, we consider a slight variation of an NBA called generalized nondeterministic Büchi automata (GNBA). The difference between NBAs and GNBA is in the acceptance condition. In the case of a GNBA the acceptance condition is specified by a finite set $\mathcal{F}$ of (possibly empty) subsets of $Q$, and an accepting run is required to visit each of these sets infinitely often.

Definition 4.34. A generalized NBA (GNBA) is a tuple $\mathcal{G} = (Q, \Sigma, \delta, Q_0, \mathcal{F})$ where

- $Q$ is a finite set of states;
- $\Sigma$ is a nonempty finite set (alphabet);
- $\delta : Q \times \Sigma \rightarrow 2^Q$ (the transition function);
- $Q_0 \subseteq Q$ is a set of initial states;
- $\mathcal{F} \subseteq 2^Q$ whose elements are called acceptance sets.
4.3. BÜCHI AUTOMATA

A run for $\sigma = v_0v_1v_2\cdots \in \Sigma^\omega$ in $G$ is an infinite sequence of states $q_0q_1q_2\ldots$ such that

- $q_0 \in Q_0$;
- $q_{i+1} \in \delta(q_i, v_i)$, for every $i \in \mathbb{N}$.

The run is accepting if

$$\forall F \in \mathcal{F} \exists \in \mathbb{N} q_j \in F.$$ 

The accepted language of $G$ is the set

$$L_\omega(G) = \{ \sigma \in \Sigma^\omega \mid \text{there exists an accepting run for } \sigma \text{ in } G \}.$$

The set $\mathcal{F}$ of acceptance states of a GNBA may be empty. In this case, $\sigma \in L_\omega(G)$ if and only if there is an infinite run for $\sigma$ in $G$. In the case of an NBA with $F = \emptyset$ there are no accepting runs. In the case of GNAs, every infinite run in $G = \langle Q, \Sigma, \delta, Q_0, \emptyset \rangle$ is accepting.

Every GNBA $G$ is equivalent to a GNBA $G'$ having at least one acceptance set. In fact, let $G = \langle Q, \Sigma, \delta, Q_0, F \rangle$ and $G' = \langle Q, \Sigma, \delta, Q_0, \mathcal{F} \cup \{Q\} \rangle$. Then, it is not very hard to prove that $L_\omega(G) = L_\omega(G')$.

Clearly, an NBA can be seen as a GNBA with exactly one acceptance set: the set of final states of the NBA. But the converse also holds, that is, for each GNBA there is an equivalent NBA that recognizes the same $\omega$-language.

**Theorem 4.35.** For each GNBA $G = \langle Q, \Sigma, \delta, Q_0, \mathcal{F} \rangle$ there exists an NBA $A$, over the same alphabet $\Sigma$, with $L_\omega(A) = L_\omega(G)$.

**Proof.** Due to the previous remark, we may assume without loss of generality that $\mathcal{F} \neq \emptyset$. Hence, let $\mathcal{F} = \{F_1, \ldots, F_k\}$, for some $k \geq 1$. Let $A = \langle Q', \Sigma, \delta', Q'_0, F' \rangle$ be such that:

- $Q' = Q \times \{1, \ldots, k\}$,
- $Q'_0 = Q_0 \times \{1\} = \{(q_0, 1) \mid q_0 \in Q_0\}$,
- $F' = F_1 \times \{1\} = \{(q_f, 1) \mid q_f \in F_1\}$. 


• $\delta' : Q' \times \Sigma \to 2^{Q'}$ is such that

$$
\delta'(\langle q, i \rangle, v) = \begin{cases} 
\{ \langle q', i \rangle | q' \in \delta(q, v) \} & \text{if } q \notin F_i \\
\{ \langle q', i + 1 \rangle | q' \in \delta(q, v) \} & \text{otherwise.}
\end{cases}
$$

where, for simplicity of notation, we identify $k + 1$ with 1.

The idea is that there are $k$ copies of $G$. The acceptance set of the $i$th copy is connected to the corresponding states of the $(i + 1)$th copy, that is, as soon as a final state of the $i$th copy is reached, we move to the $(i + 1)$th copy. In $A$ the only acceptance set is the one corresponding to the first copy. This construction ensures that the first copy is visited infinitely often and all the other copies are also visited infinitely often. We now prove that $L_\omega(A) = L_\omega(G)$.

$L_\omega(A) \subseteq L_\omega(G)$: Let $\sigma = v_0 v_1 \cdots \in L_\omega(A)$. Then there is an accepting run for $\sigma$ in $A$ that must be of the form

$$
\langle q_0^1, 1 \rangle \langle q_1^1, 1 \rangle \cdots \langle q_{n_1}^1, 1 \rangle \langle q_2^2, 2 \rangle \langle q_2^2, 2 \rangle \cdots \langle q_{n_2}^k, 2 \rangle \cdots \langle q_{n_k}^k, k \rangle \\
\langle q_1^{k+1}, 1 \rangle \langle q_2^{k+1}, 1 \rangle \cdots \langle q_{n_{k+1}}^{k+1}, 1 \rangle \langle q_1^{k+2}, 2 \rangle \langle q_2^{k+2}, 2 \rangle \cdots \langle q_{n_{k+2}}^{k+2}, 2 \rangle \cdots
$$

Now, let $\sigma = w_1 w_2 \cdots$ where the finite segments $w_i$, for $i \in \mathbb{N}$, are defined as follows. The segment $w_1$ is the word $v_0 \cdots v_{n_1 - 1}$. We know that for each $i = 0, \ldots, n_1 - 1$, we have

$$
\langle q_i^1, 1 \rangle \in \delta'(\langle q_i^1, 1 \rangle, v_i).
$$

Consequently, it follows by construction of $\delta'$, that

$$
q_{i+1}^1 \in \delta(q_i^1, v_i).
$$

Again, by construction of $\delta'$, it must be the case $q_{n_1}^1 \in F_1$ because in our run we moved from copy 1 to copy 2. Now, let $w_2 = v_{n_1} \cdots v_{n_1 + n_2 - 1}$. First, we observe that $q_1^2 \in \delta(q_{n_1}^1, v_{n_1})$. Then, we know that for each $i = 1, \ldots, n_2 - 1$

$$
\langle q_{i+1}^2, 2 \rangle \in \delta'(\langle q_i^2, 2 \rangle, v_{n_1 + i})
$$

and, consequently,

$$
q_{i+1}^2 \in \delta(q_i^2, v_{n_1 + i}).
$$
Again, $q_{n_2}^2 \in F_2$ because we moved from the second copy to the third. Hence, the run obtained is a run in $G$ and is of the form

$$q_0 \ldots q_{n_1}^1 \ldots q_{n_2}^2 \ldots q_k^k \ldots q_{k+1}^{k+1} \ldots$$

such that $q_0 \in Q_0$, $q_1^{n_1}, q_{k+1}^{k+1} \ldots \in F_1$, $q_2^{n_2}, q_{k+2}^{k+2} \ldots \in F_2$, .... Hence, each one of the acceptance sets is visited infinitely often and consequently this is an accepting run for $\sigma$, that is, $\sigma \in L_\omega(G)$.

The proof of the converse is similar and is left as an exercise.

**Corollary 4.36.** The class of accepted languages by GNBA\-s agrees with the class of $\omega$-regular languages.

We have already proved that the class of $\omega$-regular languages is closed under union (see Lemma 4.25). This was proved using NBAs. We now prove that this class is also closed under intersection. Thanks to the previous corollary, we can use GNBA\-s to prove this result.

**Lemma 4.37.** Let $G_1$ and $G_2$ be GNBA\-s over the same alphabet. Then, there exists a GNBA $G$ with $L_\omega(G) = L_\omega(G_1) \cap L_\omega(G_2)$.

**Proof.** Let $G_1 = \langle Q_1, \Sigma, \delta_1, Q_{01}, F_1 \rangle$ and $G_2 = \langle Q_2, \Sigma, \delta_2, Q_{02}, F_2 \rangle$ and assume without loss of generality that $Q_1 \cap Q_2 = \emptyset$. Let $G$ be an GNBA

$$\langle Q_1 \times Q_2, \Sigma, \delta, Q_{01} \times Q_{02}, F \rangle$$

where

- $\delta$ is such that $\delta((q_1, q_2), a) = \delta_1(q_1, a) \times \delta_2(q_2, a)$,
- $F = \{F_1 \times F_2 \mid F_1 \in F_1\} \cup \{Q_1 \times F_2 \mid F_2 \in F_2\}$.

We now prove that $L_\omega(G) = L_\omega(G_1) \cap L_\omega(G_2)$.

$L_\omega(G) \subseteq L_\omega(G_1) \cap L_\omega(G_2)$: Let $\sigma \in L_\omega(G)$. Then there is an accepting run

$$\langle q_0^1, q_0^2 \rangle \langle q_1^1, q_1^2 \rangle \langle q_2^1, q_2^2 \rangle \ldots$$

for $\sigma$ in $G$. Then, by construction of $\delta$, we know that $q_0^1 q_1^1 q_2^1 \ldots$ is a run for $\sigma$ in $G_1$. Since $\sigma$ is accepting it must visit all the sets in $F$ infinitely often. In particular, it must visit all the sets in $\{F_1 \times Q_2 \mid F_1 \in F_1\}$ infinitely often.
Hence, \( q_0^1 q_1^1 q_2^1 \ldots \) visits all the sets in \( F_1 \) infinitely often and, consequently, is an accepting run for \( \sigma \) in \( G_1 \). By a similar argument, we can conclude that \( q_0^2 q_1^2 q_2^2 \ldots \) is an accepting run for \( \sigma \) in \( G_2 \). Thus, \( \sigma \in L_\omega(G_1) \cap L_\omega(G_2) \).

Let \( \sigma \in L_\omega(G_1) \cap L_\omega(G_2) \). Then, there is an accepting run \( q_0^0 q_1^1 q_2^1 \ldots \) for \( \sigma \) in \( G_1 \). In particular, this run visits all the sets in \( F_1 \) infinitely often. There is also an accepting run \( q_0^2 q_1^2 q_2^2 \ldots \) for \( \sigma \) in \( G_2 \) that visits all the sets in \( F_2 \) infinitely often. Then, \( \langle q_0^1, q_0^0 \rangle \langle q_1^2, q_1^1 \rangle \langle q_2^3, q_2^2 \rangle \ldots \) is a run for \( \sigma \) in \( G \). Furthermore, it visits all the sets in \( \{F_1 \times Q_2 \mid F_1 \in F_1\} \) and in \( \{Q_1 \times F_2 \mid F_2 \in F_2\} \) infinitely often. Consequently, it is an accepting run for \( \sigma \) in \( G \) and thus \( \sigma \in L_\omega(G) \).

**Corollary 4.38.** If \( L_1 \) and \( L_2 \) are \( \omega \)-regular languages over the alphabet \( \Sigma \) then so is \( L = L_1 \cap L_2 \).

### 4.4 \( \omega \)-Regular properties

An \( \omega \)-regular property is just an LT property that is simultaneously an \( \omega \)-language. In Example 4.24 we showed that the liveness property \( P_{\text{live}} \) about a traffic light expressing that the green light is on infinitely often is accepted by an NBA. From Proposition 4.29, we know that this is an \( \omega \)-regular language and, consequently, we can conclude that \( P_{\text{live}} \) is an \( \omega \)-regular property. In this section, we address the problem of verification of \( \omega \)-regular properties.

**Definition 4.39.** An LT property \( P \) is called \( \omega \)-regular if \( P \) is an \( \omega \)-regular language over the alphabet \( 2^\Xi \).

As usual, we start with a transition system \( T \) and an LT property \( P \) and we aim at verifying if \( T \) satisfies \( P \). Let \( T = \langle S, A, \rightarrow, I, \Xi, L \rangle \) be a transition system without terminal states and let \( P \) be an \( \omega \)-regular property. We want to check whether \( T \models P \) or not. The idea is to try to show that \( T \not\models P \) by providing a counterexample, that is, by trying to find a path \( \pi \) in \( T \) such that \( \text{trace}(\pi) \not\in P \). If no such path exists, then \( T \models P \). To this end, we consider an NBA \( A \) for the complement property \( \overline{P} = (2^\Xi)^\omega \setminus P \).
Thus, we have to check if \( \text{Traces}(T) \cap L_\omega(A) = \emptyset \). In fact,

\[
\text{Traces}(T) \cap L_\omega(A) \neq \emptyset \quad \text{if and only if} \quad \text{Traces}(T) \cap \overline{P} \neq \emptyset \\
\text{if and only if} \quad \text{Traces}(T) \cap ((2^\Xi)^\omega \setminus P) \neq \emptyset \\
\text{if and only if} \quad \text{Traces}(T) \not\subseteq P \\
\text{if and only if} \quad T \not\vDash P.
\]

This is similar to what was done for checking regular safety properties. Analogously, in order to check if \( \text{Traces}(T) \cap L_\omega(A) \neq \emptyset \), we start by constructing the product \( T \otimes A \), where, in this case, \( A \) is an NBA. Then, we perform a graph analysis of \( T \otimes A \) to check whether there is a path that visits an accept state of \( A \) infinitely often. If such a path exists, then we can conclude that \( T \not\vDash P \). If no such path exists, that is, if accept states can only be visited finitely many times then we can conclude that \( \text{Traces}(T) \cap L_\omega(A) = \emptyset \) and, consequently, \( T \vDash P \).

We introduce a special class of LT properties, called persistency properties, that will be used to formalize conditions stating that accept states are visited only finitely many times. Then, verifying an \( \omega \)-regular property can be reduced to the persistency checking problem, as was done when the problem of checking regular safety properties was reduced to the invariant checking problem.

Intuitively, a persistency property is a state condition \( \varphi \) that always holds after some state. Here, \( \varphi \) is a propositional formula expressing a state condition. Then, we want \( \varphi \) to hold continuously after some state or, equivalently, we want \( \neg \varphi \) to hold only finitely many times.

**Definition 4.40.** Let \( \varphi \) be some propositional formula over \( \Xi \). A persistency property induced by \( \varphi \) is an LT property \( P^\varphi_{\text{pers}} \subseteq (2^\Xi)^\omega \) such that,

\[
P^\varphi_{\text{pers}} = \{ \sigma \in (2^\Xi)^\omega \mid \text{there is } i \text{ such that } \sigma[j] \models \varphi, \text{ for every } j \geq i \}.
\]

The formula \( \varphi \) is called a persistency (or state) condition of \( P^\varphi_{\text{pers}} \).

### 4.4.1 Verifying \( \omega \)-regular properties

We are now going to show the problem of deciding whether \( \text{Traces}(T) \cap L_\omega(A) = \emptyset \) can be reduced to the problem of checking whether a certain persistency property holds in the product of \( T \) and \( A \).
**Definition 4.41.** Let $T = \langle S, A, \rightarrow, I, \Xi, L \rangle$ be a transition system without terminal states and $A = \langle Q, 2^\Xi, \delta, Q_0, F \rangle$ be a nonblocking NBA. Then, $T \otimes A$ is the following transition system:

$$T \otimes A = \langle S \times Q, A, \rightarrow', I', \Xi', L' \rangle$$

where:

- $\rightarrow'$ is the smallest relation defined by the rule
  $$\frac{s \xrightarrow{a} t \ p \in \delta(q, L(t))}{(s, q) \xrightarrow{a} (t, p)}$$

- $I' = \{ \langle s_0, q \rangle \mid s_0 \in I, \exists q_0 \in Q_0. q \in \delta(q_0, L(s_0)) \}$,

- $\Xi' = Q$,

- $L': S \times Q \rightarrow 2^Q$ is defined by $L'(\langle s, q \rangle) = \{q\}$.

Furthermore, let $P_{\text{pers}(A)}$ be the persistence property over $\Xi' = Q$ given by

“Eventually forever $\neg F$”

where $\neg F$ denotes the propositional formula $\bigwedge_{q \in F} \neg q$.

In our notation this is $P_{\neg F}^\text{pers}$.

**Theorem 4.42.** Let $T$ be a transition system without terminal states over $\Xi$ and let $P$ be an $\omega$-regular property over $\Xi$. Furthermore, let $A$ be a nonblocking NBA with alphabet $2^\Xi$ and $L_\omega(A) = (2^\Xi)^\omega \setminus P$. Then, the following statements are equivalent:

1. $T \models P$

2. $\text{Traces}(T) \cap L_\omega(A) = \emptyset$

3. $T \otimes A \models P_{\text{pers}(A)}$. 

4.4. $\omega$-REGULAR PROPERTIES

Proof. Let $T = \langle S, A, \rightarrow, I, \Xi, L \rangle$ and $A = \langle Q, 2^\Xi, \delta, Q_0, F \rangle$. The equivalence between (1) and (2) was proved in the beginning of this subsection. We now prove equivalence between (2) and (3), by proving that

$$Traces(T) \cap L_\omega(A) \neq \emptyset$$ if and only if $T \otimes A \not\models P_{\text{pers}}(A)$.

($\leftarrow$) Assume that $T \otimes A \not\models P_{\text{pers}}(A)$. Then, there is a path

$$\pi' = \langle s_0, q_1 \rangle \langle s_1, q_2 \rangle \ldots$$

in $T \otimes A$ such that

$$\pi' \not\models P_{\text{pers}}(A).$$

Recall that $trace(\pi') = L'(\langle s_0, q_1 \rangle) L'(\langle s_1, q_2 \rangle) \cdots = \{q_1\} \{q_2\} \cdots$ and let $\sigma'$ denote this trace. Given that $\pi' \not\models P_{\text{pers}}(A)$ then $\sigma' \not\models P_{\text{pers}}(A)$. By definition of persistence property, there are infinitely many indices $j$ such that $\sigma'[j] \not\models \neg F$, that is, there are infinitely many $j$ such that $\sigma'[j] = \{q_j\}$ and $q_j \in F$.

Consider now the projection of $\pi'$ to the states of $T$ which clearly yields a path $\pi = s_0 s_1 \ldots$ in $T$. Let $q_0 \in Q_0$ be such that $q_1 \in \delta(q_0, L(s_0))$. This state must exist given that $\langle s_0, q_1 \rangle$ is an initial state of $T \otimes A$. Then, the sequence $q_0 q_1 \ldots$ is a run in $A$ for the word

$$L(s_0) L(s_1) \ldots$$

by definition of $\rightarrow'$ and $Q'_0$. Observe that this word is $trace(\pi)$ so $trace(\pi) \in Traces(T)$.

In addition, since there are infinitely many accepting states in the run $q_0 q_1 \ldots$ this means that this is an accepting run. Indeed, by definition of $\rightarrow'$ and $Q'_0$ we know that if $\langle s_i, q_{i+1} \rangle \rightarrow^a \langle s_{i+1}, q_{i+2} \rangle$ then $q_{i+2} \in \delta(q_{i+1}, L(s_{i+1}))$. Furthermore, $q_1 \in \delta(q_0, L(s_0))$. Hence, $q_0 q_1 \ldots$ is an accepting run for $trace(\pi) = L(s_0) L(s_1), \ldots$, which implies that $trace(\pi) \in L_\omega(A)$.

Hence, $trace(\pi) \in Traces(T) \cap L_\omega(A)$ and so $Traces(T) \cap L_\omega(A) \neq \emptyset$.

($\rightarrow$) Assume that $Traces(T) \cap L_\omega(A) \neq \emptyset$. Then, there exists a trace $\sigma \in Traces(T) \cap L_\omega(A)$. Since $\sigma \in Traces(T)$ then there exists a path $\pi = s_0 s_1 \ldots$ such that $trace(\pi) = \sigma$. Furthermore,

$$\sigma = L(s_0) L(s_1) \ldots \in L_\omega(A).$$
Let $q_0 q_1 q_2 \ldots$ be an accepting run for $\sigma$ in $\mathcal{A}$. Then,

$$q_0 \in Q_0 \text{ and } q_{i+1} \in \delta(q_i, L(s_i)), \text{ for all } i \geq 0.$$ 

Consider now the sequence

$$\pi' = \langle s_0, q_1 \rangle \langle s_1, q_2 \rangle \langle s_2, q_3 \rangle \ldots$$

As $s_i \xrightarrow{a} s_{i+1}$, for some $a \in A$, and $q_{i+2} \in \delta(q_{i+1}, L(s_{i+1}))$, for every $i \geq 0$, we have, by definition of $\rightarrow'$, that

$$\langle s_i, q_{i+1} \rangle \xrightarrow{a} \langle s_{i+1}, q_{i+2} \rangle, \text{ for all } i \geq 0.$$ 

Furthermore, $\langle s_0, q_1 \rangle \in Q'_0$, given that $q_1 \in \delta(q_0, L(s_0))$ and $q_0 \in Q_0$. Hence, $\pi'$ is a path in $T \otimes A$. Furthermore,

$$\text{trace}(\pi') = L'(\langle s_0, q_1 \rangle) L'(\langle s_1, q_2 \rangle) L'(\langle s_2, q_3 \rangle) \ldots = \{q_1\} \{q_2\} \{q_3\} \ldots$$

Finally, as $q_0 q_1 q_2 \ldots$ is an accepting run in $\mathcal{A}$ then there are infinitely many indices $j$ such that $q_j \in F$. Hence, $\text{trace}(\pi') \not\in P_{\text{pers}(A)}$ which implies that $\pi' \not\models P_{\text{pers}(A)}$ and, consequently, $T \otimes A \not\models P_{\text{pers}(A)}$. \qed

**Example 4.43.** Let $\Xi = \{\text{red}, \text{green}\}$ and recall the liveness property $P$ of Example 4.24, expressing that a pedestrian traffic light is green infinitely often. We are going to show that the transition system $T = \langle S, A, \rightarrow, I, \Xi, L \rangle$ in Figure 4.10 specifies a traffic light that satisfies this property. As usual we assume that the labeling function of $T$ is such that $L(s) = \{s\}$ and, for simplicity, we also omit the action names.
In order to use Theorem 4.42, we need to consider an NBA for $\mathcal{P}$. Recall that $\mathcal{P}$ consists of all the infinite words $v_0 v_1 v_2 \ldots$ over $2^\Xi$ such that $\text{green} \in v_i$, for infinitely many $i$'s. Hence, $\mathcal{P}$ consists of all the infinite words $v_0 v_1 v_2 \ldots$ over $2^\Xi$ such that $\text{green} \in v_i$, for finitely many $i$'s. We can restate this by saying that after some point $\text{green}$ does not hold any more, that is, eventually forever $\neg \text{green}$. Observe that, in particular, this is a persistency property induced by $\neg \text{green}$ and so $\mathcal{P} = P_{\neg \text{green}}^{\text{Pers}}$. An NBA $\mathcal{A}$ accepting $\mathcal{P}$ is depicted in Figure 4.11.

This automaton accepts only sequences of subsets of $\Xi$ that, after some point, do not contain $\text{green}$. If at $q_1$ some set with $\text{green}$ appears then the automaton moves to the non accepting state $q_2$ where it remains forever.

We now build the transition system $T \otimes \mathcal{A} = \langle S', A, \rightarrow', I', \Xi', L' \rangle$. The set of states is

$$S' = \{ (\text{red}, q_0), (\text{red}, q_1), (\text{red}, q_2), (\text{green}, q_0), (\text{green}, q_1), (\text{green}, q_2) \}.$$ 

To define the set of initial states $I'$ observe that $I = \{ \text{red} \}$ and that $\delta(q_0, L(\text{red})) = \delta(q_0, \{ \text{red} \}) = \{ q_0, q_1 \}$. Hence,

$$I' = \{ (\text{red}, q_0), (\text{red}, q_1) \}.$$ 

The set of propositional symbols of $T \otimes \mathcal{A}$ is $Q$, that is,

$$\Xi' = \{ q_0, q_1, q_2 \},$$

and the labeling function $L'$ is such that

$$L'(s, q) = \{ q \}.$$ 

Finally, let us consider the transition relation $\rightarrow'$ for $T \otimes \mathcal{A}$. We start by considering the transitions departing from $\langle \text{red}, q_0 \rangle$. In this case, observe
that $\text{red} \rightarrow \text{green}$ holds in $T$ and that $\delta(q_0, L(\text{green})) = \delta(q_0, \{\text{green}\}) = \{q_0\}$. Hence, we have the following transition in $T \otimes A$:

$\langle \text{red}, q_0 \rangle \rightarrow' \langle \text{green}, q_0 \rangle$.

Consider now the transitions departing from $\langle \text{green}, q_0 \rangle$. In this case, we have that $\text{green} \rightarrow \text{red}$ holds in $T$ and that $\delta(q_0, L(\text{red})) = \delta(q_0, \{\text{red}\}) = \{q_0, q_1\}$. Consequently, we have the following transitions in $T \otimes A$:

$\langle \text{green}, q_0 \rangle \rightarrow' \langle \text{red}, q_0 \rangle$ and $\langle \text{green}, q_0 \rangle \rightarrow' \langle \text{red}, q_1 \rangle$.

The full transition relation $\rightarrow'$ is depicted in Figure 4.12.

We now check whether $T \Vdash P$ or not. To this end, we use Theorem 4.42, where the persistency property $P_{\text{pers}(A)}$ for $A$ is “eventually forever $\neg q_1$”, given that $q_1$ is the only final state. Observing the transition system in Figure 4.12 we may conclude that there is no state of the form $\langle \cdot, q_1 \rangle$ that is visited infinitely often and consequently the property $P_{\text{pers}(A)}$ is satisfied by $T \otimes A$, that is,

$T \otimes A \Vdash P_{\text{pers}(A)}$.

Hence, we can conclude that $T$ and $A$ have no common traces and, consequently, that $T \not\Vdash P$.

With this theorem the problem of verifying if a finite transition system satisfies a property can be reduced to checking if a transition system satisfies a persistence property. So, from now on we focus on solving the problem:

$T \not\Vdash P_{\text{pers}}^c$
for some transition system $T$, persistence property $P^\varphi_{pers}$ and propositional formula $\varphi$ over the same set of propositional symbols.

We will show that checking if $T \not\models P^\varphi_{pers}$ amounts to checking if $T$ contains a reachable state violating $\varphi$ that belongs to a cycle. The intuition behind this result is very simple. Assume that there is a reachable state $s$ such that $\varphi$ does not hold in $s$ and $s$ occurs in a cycle $s_1 \ldots s_n s$. Then, we just consider the path consisting of the initial path fragment leading to $s$ (that exists because $s \in \text{Reach}(T)$) followed by infinite copies of the path fragment $s_1 \ldots s_n$. This is clearly a path in $T$ and furthermore it visits infinitely often a state $s$ where $\varphi$ does not hold. Consequently, $T$ does not satisfy the persistency property $P^\varphi_{pers}$.

**Theorem 4.44.** Let $T$ be a transition system without terminal states over $\Xi$, $\varphi$ a propositional formula over $\Xi$, and $P^\varphi_{pers}$ a persistence property induced by $\varphi$. Then, the following assertions are equivalent:

1. $T \not\models P^\varphi_{pers}$.

2. there exists $s \in \text{Reach}(T)$ such that $s \not\models \varphi$ and $s$ occurs in a cycle in $G(T)$.

**Proof.** Let $T = (S, A, \rightarrow, I, \Xi, L)$.

(1 → 2): Assume that $T \not\models P^\varphi_{pers}$. Hence, there exists a path $\pi = s_0 s_1 s_2 \ldots \in \text{Paths}(T)$ such that $\text{trace}(\pi) \notin P^\varphi_{pers}$. Then, it must be the case that $s_i \not\models \varphi$, for infinitely many indices $i$. As the set of states of $T$ is finite then, there is at least one state $s$ such that $s \not\models \varphi$ and $s$ appears infinitely often in $\pi$. Since $s$ appears in a path in $T$ then $s \in \text{Reach}(T)$. Furthermore, given that $s$ appears infinitely often in $\pi$ then there are at least two indices $i_1$ and $i_2$, with $i_1 < i_2$ such that $s_{i_1} = s_{i_2} = s$. The path fragment $\tilde{\pi}$ of $\pi$ given by $s_{i_1} s_{i_1 + 1} \ldots s_{i_2}$ clearly defines a cycle in $G(T)$.

(2 → 1): Assume now that there is $s \in \text{Reach}(T)$ such that $s \not\models \varphi$ and $s$ occurs in a cycle in $G(T)$. If $s \in \text{Reach}(T)$ then there exists a finite initial path fragment $\tilde{\pi}_0 = s_0 s_1 \ldots s_k$ such that $s_k = s$. Since $s$ occurs in a cycle then there a finite path fragment $\tilde{\pi}_c = s_1' s_2' \ldots , s_n'$ such that $s_1' = s_n' = s$. Consider now the path $\pi$ in $T$ obtained from concatenating $\tilde{\pi}_0$ (without the last state $s_k$) with infinite copies of $\tilde{\pi}_c$ (without the last state $s_n'$), that is,

$$\pi = s_0 s_1 \ldots s_{k-1} s_1' s_2' \ldots , s_{n-1}' s_1' s_2' \ldots , s_{n-1}' \ldots .$$
Then, $\pi$ is a path in $T$. By construction, $s$ occurs infinitely often in $\pi$ and as $s \not\models \varphi$ it follows that $\text{trace}(\pi) \not\in P_{\text{pers}}$. This implies that $T \not\models P_{\text{pers}}$.

**Example 4.45.** Recall Example 4.43 where a pedestrian traffic light is specified. Recall also the persistency property $P_{\text{pers}}(A)$, defined there, stating that eventually forever $\neg q_1$ holds, where $q_1$ is the final state of $A$. Taking into account Theorem 4.44, in order for $T \otimes A \not\models P_{\text{pers}}(A)$ we would need to find a cycle containing the state $\langle \text{red}, q_1 \rangle$, given that this is the only reachable state that does not satisfy $\neg q_1$. But there is no such cycle and, consequently, we can conclude that $T \otimes A \models P_{\text{pers}}(A)$.

We now present an algorithm for persistence checking for a finite transition system. By Theorem 4.44, the problem of checking whether a finite transition system satisfies a persistency property can be reduced to the problem of detecting a cycle containing a reachable cycle with a state that does not satisfy the state condition of the persistency property.

Given a finite graph $G$ and one of its nodes $v$, we can use a DFS-based approach to check whether $v$ belongs to a cycle in $G$ or not. The idea is as follows: we begin with a depth-first search starting in $v$ and check for any visited node $w$ whether there is an edge from $w$ to $v$. If such an edge exists then there is a cycle in $G$ containing $v$, because if $w$ has been visited (in a depth-first search) then this means that there is a path from $v$ to $w$. If there is no such edge from any of the visited nodes to $v$ then we can conclude that $v$ does not belong to any cycle in $G$. This idea can be extended to checking if $G$ has a cycle containing a reachable node.

The first proposal for the algorithm is defined in Figure 4.13. It works in two phases: in the first phase, all states reachable from an initial state and that do not satisfy $\varphi$ are determined (and stored in variable $R_{\neg \varphi}$). This can be achieved by a standard depth-first search (like the algorithm for invariant checking presented in Figure 3.2). In the second phase, for each reachable state that does not satisfy $\varphi$ (that is every state in $R_{\neg \varphi}$), we check whether this state appears in a cycle in $T$ or not. This is achieved by the procedure cycle_check depicted in Figure 4.14. This procedure works as described above: we start with a depth-first search in $s$ and check for all states reachable from $s$ whether there is an edge going from that state to $s$. If such an edge is found, then a cycle containing $s$ is found and the procedure ends. If no edge is found for any state reachable from $s$ then there is no cycle in $T$ containing $s$. We use two variables: a set $R_c$ for keeping the
states that are reachable from $s$, and a stack $U_c$ for the depth-first search. The role of these variables is similar to the role of variables $R$ and $U$ in procedure $\text{visit}$.

**Example 4.46.** We illustrate the use of the algorithm to check that the transition system for the pedestrian traffic light of Example 4.43 does indeed satisfy the intended liveness property. To this end we consider the transition system $T \otimes A$ depicted in Figure 4.12 and show that it satisfies the persistence property $P^{-q_1}_{\text{pers}}$. For simplicity we re-label the states of the transition system $T \otimes A$, as depicted in Figure 4.15.

Recall from that example that the only states that satisfy $q_1$ are $s_1$ and $s_4$, that is, for $\varphi \equiv \neg q_1$ we have $s_1 \not\models \varphi$ and $s_4 \not\models \varphi$. Initially, we have $R = \emptyset$, $R_\neg \varphi = \emptyset$, and $U = \text{new}$. We do not consider the variables $R_c$ and $U_c$ for now, since they are reseted before each call to the procedure $\text{cycle\_check}$.

We start with an initial node that has not yet been visited. So, let consider $s = s_0$ and let us consider the procedure $\text{visit}$, which is very similar the procedure $\text{visit}$ presented in Figure 3.2.

After the first two assignments, the content of the variables is

\[ U = \begin{array}{c} s_0 \end{array} \quad R = \{ s_0 \} \]

The first step of the loop is executed by letting $s' = \text{top}(U) = s_0$. As $\text{Suc}(s_0) = \{ s_3 \}$ and, thus, $\text{Suc}(s_0) \subseteq R$ does not hold, we choose one of the elements in $\text{Suc}(s_0) \setminus R$ and add it to $R$ and push it onto $U$. In this case we can only choose $s_3$. Consequently, after the execution of this step, the content of the variables is

\[ U = \begin{array}{c} s_3 \\ s_0 \end{array} \quad R = \{ s_0, s_3 \} \]

The second step of the loop is similar. In this case the top of the stack is $s' = s_3$ and $\text{Suc}(s_3) = \{ s_0, s_1 \}$ and, once again, the condition $\text{Suc}(s') \subseteq R$ does not hold. In this case, the choice of $s''$ has to be $s_1$ since $\text{Suc}(s_3) \setminus R = \{ s_1 \}$. After the execution of the second step, the content of the variables is

\[ U = \begin{array}{c} s_1 \\ s_3 \\ s_0 \end{array} \quad R = \{ s_0, s_1, s_3 \} \]
Input: finite transition system $T$ without terminal states, and formula $\varphi$

Output: “yes” if $T \models P^\varphi_{\text{pers}}$, “no” otherwise

set of states $R := \emptyset$; $R_{\neg \varphi} := \emptyset$;
stack of states $U := \text{new}$;
set of states $R_c := \emptyset$;
stack of states $U_c := \text{new}$;

forall $s \in I \setminus R$ do visit($s$) od
forall $s \in R_{\neg \varphi}$ do
    $R_c := \emptyset$; $U_c := \text{new}$;
    if cycle_check($s$) then return “no”
od
return “yes”

proc visit (state $s$)
    push($s, U$);
    $R := R \cup \{s\}$;
    repeat
        $s' := \text{top}(U)$;
        if Suc($s'$) $\subseteq R$ then
            pop($U$);
            if $s' \not\models \varphi$ then $R_{\neg \varphi} := R_{\neg \varphi} \cup \{s'\}$ fi;
        else
            let $s'' \in \text{Suc}(s') \setminus R$
            push($s'', U$);
            $R := R \cup \{s''\}$;
        fi
    until empty($U$)
endproc

Figure 4.13: DFS algorithm for persistence checking
Input: finite transition system $T$ without terminal states and state $s$ such that $s \not\models \varphi$
Output: true if $s$ occurs in a cycle in $T$, false otherwise

\begin{verbatim}
proc cycle_check (state s)
    bool found := false;
    push(s, Uc);
    Rc := Rc ∪ \{s\};
    repeat
        s' := top(Uc);
        if s ∈ Suc(s') then
            found := true;
            push(s, Uc);
        else
            if Suc(s') \ Rc \̸= ∅
                let s'' ∈ Suc(s') \ Rc
                push(s'', Uc);
                Rc := Rc ∪ \{s''\};
            else
                pop(Uc);
            fi
        fi
    until empty(Uc) \∨ found
    return found
endproc
\end{verbatim}

Figure 4.14: Procedure for cycle detection
In the third step of the loop, we have $s' = s_1$ and after the execution of this step, the content of the variables is

$$U = \begin{bmatrix} s_5 \\ s_1 \\ s_3 \\ s_0 \end{bmatrix} \quad R = \{s_0, s_1, s_3, s_5\}$$

In the fourth step, we have $s' = s_5$ and after the execution of this step, the content of the variables is

$$U = \begin{bmatrix} s_2 \\ s_5 \\ s_1 \\ s_3 \\ s_0 \end{bmatrix} \quad R = \{s_0, s_1, s_2, s_3, s_5\}$$

In the fifth step, we have $s' = s_2$ and condition $\text{Suc}(s') \subseteq R$ holds (and it will hold henceforth). The first element of $U$ is removed and as $s' \models \varphi$ the set $R_{\neg \varphi}$ does not change. The content of the DFS stack continues to be analyzed until $s_1$ is reached. In this case, $s_1$ is added to $R_{\neg \varphi}$ Then, after the execution of the loop, the call to the procedure `visit` with parameter $s_0$ ends. The first cycle of the main algorithm then ends because there is no additional initial node that has not yet been visited (the other initial node is $s_1$ but $s_1 \in R$).

Next, we move to the second cycle of the main algorithm, where all the elements in $R_{\neg \varphi}$ are examined by the procedure `cycle_check`. In this case, we only have to consider $s_1$. The behavior of this procedure is similar to
the procedure visit and it will be illustrated in the next example. It checks if any of the reachable states from $s_1$ have a transition to $s_1$. Since neither $s_2$ nor $s_5$ have such a transition, the procedure ends with the answer false.

In the end, the algorithm ends with the answer “yes” meaning that the transition system $T \otimes A$ satisfies the persistence property $P_{\text{pers}}^\neg q_1$.

**Example 4.47.** Consider the transition system depicted in Figure 4.16 and let $\varphi$ be a propositional formula such that $s_0 \models \varphi$, $s_3 \models \varphi$, $s_1 \not\models \varphi$ and $s_2 \not\models \varphi$.

As in the previous example, the algorithm starts by calling the procedure visit for each initial state. In this case, there is only the state $s_0$. We omit the details of the call to procedure visit, but we assume that when choosing one of the successor states of $s_0$, we first choose $s_2$ and then $s_1$. When this call ends, the content of the variables is

$$U = \text{new} \quad R = \{s_0, s_1, s_2, s_3\} \quad R_{\neg \varphi} = \{s_1, s_2\}$$

Next, all the states in $R_{\neg \varphi}$ are examined by the procedure cycle_check. The analysis of the state $s_2$ will return the value false and we omit the details. We focus on the call to the procedure cycle_check with parameter $s = s_1$. After the first two assignments the content of the variables is:

$$U_c = s_1 \quad R_c = \{s_1\} \quad \text{found} = \text{false}$$

The first step of the loop is executed by letting $s' = \text{top}(U_c) = s_1$. As the condition $s_1 \in \text{Suc}(s_1) = \{s_3\}$ fails and as the condition $\text{Suc}(s_1) \setminus R_c \neq \emptyset$
holds, the state $s_3$ is pushed on top of the stack $U_c$ and added to $R_c$. Consequently, after the execution of this step, the content of the variables is

$$U_c = \begin{array}{c} s_3 \\ s_1 \end{array} \quad R_c = \{s_1, s_3\} \quad \text{found} = \text{false}$$

In the second step, we have $s' = s_3$. In this case, the condition $s \in \text{Suc}(s')$ holds because $s = s_1$ and $\text{Suc}(s') = \text{Suc}(s_3) = \{s_1\}$. So, the variable found is set to true and $s_1$ is pushed on top of the stack $U_c$. The content of the variables is

$$U_c = \begin{array}{c} s_1 \\ s_3 \\ s_1 \end{array} \quad R_c = \{s_1, s_3\} \quad \text{found} = \text{true}$$

A cycle containing a reachable state that does not satisfy $\varphi$ has been found and consequently the algorithm returns “no”, indicating that the persistence property $P^\varphi_{\text{pers}}$ is not satisfied by the transition system.

Observe that in the previous example, the contents of the stack $U_c$ describes a cycle in $T$ containing a state $s$ that does not satisfy $\varphi$. However, we have no information about how that state was reached.

The algorithm for persistence checking presented in Figure 4.13 can be improved. Instead of first detecting all the states that do not satisfy the formula $\varphi$ and then check if any of them is in a cycle, every time a state (whose successors have all been analyzed) that does not satisfy $\varphi$ is found it is immediately analyzed to determine if it occurs in a cycle. The improved algorithm is presented in Figure 4.17. Another improvement is that, in this case, the content of the stack $U_c$ describes a cycle for a certain state that does not satisfy $\varphi$ and the content of the stack $U$ contains a path leading to that state. Hence, a counter example for the falsification of $P^\varphi_{\text{pers}}$ can be provided.

**Example 4.48.** Consider again the transition system in Figure 4.16. In this case, we illustrate the behavior of the algorithm presented in Figure 4.17. The algorithm starts with a call to procedure reachable_cycle with parameter $s_0$. The first part of the execution of this procedure visits all reachable nodes from $s_0$ and after all the nodes have been visited, the content of the
**Input**: finite transition system $T$ without terminal states and formula $\varphi$

**Output**: “yes” if $T \models P^\varphi_{\text{pers}}$, “no” with counterexample otherwise

set of states $R := \emptyset$;
stack of states $U := \text{new}$;
set of states $R_c := \emptyset$;
stack of states $U_c := \text{new}$;
bool $\text{found} := \text{false}$;

while $(I \setminus R \neq \emptyset \land \neg \text{found})$ do
  let $s \in I \setminus R$;
  reachable_cycle($s$);
od
if $\neg \text{found}$ then
  return(“yes”)
else
  return(“no”, reverse($U_c, U$))
fi

**proc** reachable_cycle (state $s$)
  push($s, U$);
  $R := R \cup \{s\}$;
  repeat
    $s' := \text{top}(U)$;
    if $\text{Suc}(s') \setminus R \neq \emptyset$ then
      let $s'' \in \text{Suc}(s') \setminus R$;
      push($s'', U$);
      $R := R \cup \{s''\}$;
    else
      pop($U$);
      if $s' \n\varphi$ then
        $\text{found} := \text{cycle_check}(s')$;
      fi
    fi
  until $\text{empty}(U) \lor \text{found}$
endproc

Figure 4.17: Improved algorithm for persistence checking
variables is

\[ U = \begin{array}{c}
  s_1 \\
  s_3 \\
  s_2 \\
  s_0 \\
\end{array} \quad R = \{s_0, s_1, s_2, s_3\} \quad \text{found} = \text{false} \]

Then, the procedure starts looking for a cycle containing a state that does not satisfy \( \varphi \). It calls `cycle_check` with parameter \( s_1 \). The result of this call is the same as in the previous example and so, in the end, the content of the variables is

\[ U_c = \begin{array}{c}
  s_1 \\
  s_3 \\
  s_1 \\
\end{array} \quad R_c = \{s_1, s_3\} \quad \text{found} = \text{true} \]

The content of \( U \) is a path fragment leading from an initial state to a state that does not satisfy \( \varphi \) and that appears in a cycle. The path fragment that confirms this cycle is the content of the stack \( U_c \). Hence, if we join the two (and reverse the order) we get the path fragment

\[ s_0 s_2 s_3 s_1 s_3 s_1 \]

that can be used as a counter-example of the validity of \( \varphi \). Thereafter, we can conclude that the transition system does not satisfy \( P_{\varphi_{\text{pers}}} \).

Another important feature of this improved algorithm is that a state that has already been visited in a previous loop checking is no longer visited. This is achieved by not resetting the content of the variable \( R_c \) every time the procedure `cycle_check` is called. This does not affect the end result given that if we call `cycle_check` to a state \( s \) already in \( R_c \) then it is because \( s \) was a successor of some other state \( s' \) and consequently, \( s \) has already been analyzed before, as a consequence of the depth-first search nature of the algorithm. Hence, if \( s \) is in a cycle it would have been detected before.

**Theorem 4.49.** Let \( T \) be a finite transition system over \( \Xi \) without terminal states, \( \varphi \) a propositional formula over \( \Xi \). Then, the algorithm in Figure 4.17, when applied to \( T \) and \( \varphi \), returns the answer “no” if and only if \( T \not\models P_{\varphi_{\text{pers}}} \).
4.4. $\omega$-REGULAR PROPERTIES

Proof. We start by the left to right implication. So, assume that the algorithm’s answer is “no”. Then, it is because the procedure cycle_check found a state $s$ such that $s \in \text{Suc}(\text{top}(U_c))$. We state without proof that this state $s$ is always the bottom element of the stack $U_c$ during the execution of this procedure. Furthermore, if $s_1$ is immediately above $s_2$ in either of the stacks then $s_1 \in \text{Suc}(s_2)$. This follows from the construction of both stacks. Hence, the content of stack $U_c$ is a path $s \ldots s$ in the transition system.

If the procedure cycle_check was called with parameter $s$ then it is because $s$ was the top of the stack $U$ (that was in the mean time removed) and $s \not\vDash \varphi$. Furthermore, from what was said above, the contents of $U$ is a path fragment starting in an initial node $s_0$ and leading to $s$. Hence, the initial path fragment

$$s_0 \ldots s \ldots s$$

together with Theorem 4.44 proves that $T \not\vDash P^\varphi_{\text{pers}}$.

We now prove the reverse implication. Assume that $T \not\vDash P^\varphi_{\text{pers}}$. Then, by Theorem 4.44, there exists one reachable state $s$ (at least) that does not satisfy $\varphi$ and that belongs to a cycle in $T$. We are going to prove that the algorithm in Figure 4.17 finds one of such states. To this end, we start by proving that if the procedure cycle_check is invoked with parameter $s$ and $s$ is in a cycle in $T$ then, at the moment of invocation, $s$ cannot appear in the set $R_c$. To prove this, we prove that there is no cycle in $T$ containing $s$ such that some state of that cycle appears in $R_c$. We prove this statement by contradiction.

So, assume that cycle_check was invoked with parameter $s$ and assume that there is a cycle $u_0 \rightarrow u_1 \rightarrow \ldots \rightarrow u_k$ in $T$ such that

(i) $u_0 = u_k = s$;

(ii) $s \not\vDash \varphi$;

(iii) $\{u_0, u_1, \ldots, u_k\} \cap R_c \neq \emptyset$.

Let $t \in \{u_0, u_1, \ldots, u_k\} \cap R_c$, that exists because of condition (iii). Then $t$ appears in a cycle that also contains $s$. Furthermore, given that $t \in R_c$, this means that $t$ was placed in $R_c$ by a previous invocation of the procedure cycle_check when applied to some other node $u$. This means that
CHAPTER 4. REGULAR PROPERTIES

I. \( u \# \varphi \);

II. the invocation `cycle_check` with parameter \( u \) was prior to the invocation of `cycle_check` with parameter \( s \);

III. during the execution of `cycle_check` with parameter \( u \), the state \( t \) was added to \( R_c \).

From III it follows that \( t \) is reachable from \( u \). Furthermore, as \( t \) is in a cycle with \( s \) then \( s \) is also reachable from \( u \).

We now focus on the procedure `reachable_cycle`, and more precisely, on the relation between the states \( u \) and \( s \). We know that both nodes are visited by this procedure, since `cycle_check` was invoked with both states as parameters. The question is: which one was visited first?

- **\( u \) was visited prior to \( s \) during the execution of `reachable_cycle`.** This means that \( s \) was pushed on top of stack \( U \) after \( u \). But this means that `cycle_check` was invoked with parameter \( s \) before it was invoked with parameter \( u \), contradicting II.

- **\( s \) was visited prior to \( u \) during the execution of `reachable_cycle`.** This means that \( s \) was pushed on top of stack \( U \) before \( u \). By II, when `cycle_check` is invoked with parameter \( u \), state \( s \) is still in the stack \( U \). This means that \( u \) is reachable from \( s \). As \( s \) is also reachable from \( u \) we can conclude that \( s \) and \( u \) are in a cycle. This means that, this cycle or another cycle containing \( u \) would have been found prior to the invocation of `cycle_check` with parameter \( s \) and procedure `reachable_cycle` would have terminated.

So, we can conclude that if \( s \) is in a cycle and the algorithm has not terminated yet, it will find a cycle containing \( s \) and, thus, the procedure will return the answer “no”.\[
\]
\[
\]
Chapter 5

Linear Temporal Logic

In this chapter we introduce propositional Linear Temporal Logic (LTL) for the specification of LT properties. We focus on future time temporal operators, although past operators could have also been considered.

5.1 Linear Temporal Logic

5.1.1 Syntax

We assume fixed a set $\Xi$ of propositional symbols. The language of LTL (over $\Xi$), denote by $L_\Xi$, contains the formulas of the form

$$\varphi ::= \text{true} \mid p \mid (\neg \varphi) \mid (\varphi \land \varphi) \mid (X \varphi) \mid (\varphi U \varphi)$$

where $p \in \Xi$. The elements of $L_\Xi$ will be denoted by the letters $\varphi, \psi, \ldots$ eventually with subscripts or superscripts. When no confusion arises, we will drop the reference to the set $\Xi$ of propositional symbols.

Intuitively, formula $(X \varphi)$, called a next formula, stands for “$\varphi$ will hold in the next instant”. Formula $(\varphi_1 U \varphi_2)$ called an until formula, stands for “either $\varphi_2$ holds now or there is an instant in the future where $\varphi_2$ will hold and until then formula $\varphi_1$ must hold”. When no confusion arises, we will omit parentheses.

Other temporal operators can also be defined. Among these, there are the sometime in the future operator ($F$), and the always in the future operator ($G$). Both these operators may have two versions, depending on whether
the present is included or not. When the present is included, the operator will have a subscript $\circ$. Furthermore, we also consider a weaker form of the until operator, where the second formula is not required to hold. In this case, the first formula must hold forever. Finally, we also introduce the operator release, that will be needed later.

\[ F_{\circ} \varphi \equiv_{\text{def}} \text{true} U \varphi \]
\[ G_{\circ} \varphi \equiv_{\text{def}} \neg F_{\circ} \neg \varphi \]
\[ F \varphi \equiv_{\text{def}} X F_{\circ} \varphi \]
\[ G \varphi \equiv_{\text{def}} \neg F \neg \varphi \]
\[ \varphi_1 W \varphi_2 \equiv_{\text{def}} (\varphi_1 U \varphi_2) \lor G_{\circ} \varphi_1 \]
\[ \varphi_1 R \varphi_2 \equiv_{\text{def}} \neg (\neg \varphi_1 U \neg \varphi_2) \]

The length of a formula $\varphi$ is the number of operators in $\varphi$.

The temporal operators can be combined to express more complex properties. For instance, as we will see later, the formula

\[ G_{\circ} F_{\circ} \varphi \]

stands for “infinitely often $\varphi$” and the formula

\[ F_{\circ} G_{\circ} \varphi \]

stands for “eventually forever $\varphi$”.

**Example 5.1.** Recall Example 3.3. Property $P_1$ specified therein states that “The first traffic light is infinitely often green”. This can be described by the LTL-formula:

\[ G_{\circ} F_{\circ} \text{green}_1. \]

Property $P_2$ states that “the traffic lights are never both green at the same time”. In this case, we can use the following LTL-formula to express this fact:

\[ G_{\circ}(\neg \text{green}_1 \lor \neg \text{green}_2). \]

### 5.1.2 Semantics

An interpretation structure for LTL will be an infinite sequence of valuations, such that each element of the sequence will determine the values of the propositional symbols at a given instant of time.
5.1. LINEAR TEMPORAL LOGIC

Definition 5.2. An interpretation for LTL is an \( \omega \)-word over \( \mathcal{2}^\Xi \).

Given \( \sigma = v_0 v_1 v_2 \ldots \) we can regard \( v_i \) as the set of propositional symbols that hold at instant \( i \). With this in mind, we can define when an interpretation satisfies an LTL formula.

Definition 5.3. Let \( \sigma \) be an interpretation and \( i \in \mathbb{N} \). The satisfaction relation for LTL is inductively defined as follows:

- \( \sigma, i \models \text{true} \)
- \( \sigma, i \models p \) if \( p \in \sigma[i] \)
- \( \sigma, i \models \neg \varphi \) if \( \sigma, i \not\models \varphi \)
- \( \sigma, i \models \varphi_1 \land \varphi_2 \) if \( \sigma, i \models \varphi_1 \) and \( \sigma, i \models \varphi_2 \)
- \( \sigma, i \models X \varphi \) if \( \sigma, i + 1 \models \varphi \)
- \( \sigma, i \models \varphi_1 \mathcal{U} \varphi_2 \) if there is \( j \geq i \) such that \( \sigma, j \models \varphi_2 \) and \( \sigma, k \models \varphi_1 \), for every \( i \leq k < j \).

The interpretation \( \sigma \) satisfies the formula \( \varphi \), written \( \sigma \models \varphi \), if \( \sigma, 0 \models \varphi \).

When \( \sigma \models \varphi \) we also say that \( \sigma \) is a model of \( \varphi \), and that \( \varphi \) is satisfied by \( \sigma \). A formula is valid if it is satisfied by all interpretations. The notion of satisfaction extends to sets of formulas in a natural way. Let \( \Gamma \subseteq \mathcal{L} \). Then, we say that \( \sigma \) satisfies \( \Gamma \), written \( \sigma \models \Gamma \), if \( \sigma \models \psi \), for every \( \psi \in \Gamma \).

Definition 5.4. Let \( \Gamma \cup \{ \varphi \} \subseteq \mathcal{L} \). We say that \( \Gamma \) entails \( \varphi \), written \( \Gamma \models \varphi \), when \( \sigma \not\models \varphi \) for every interpretation \( \sigma \) such that \( \sigma \models \Gamma \).

Given a formula \( \varphi \in \mathcal{L} \) we denote by \( \text{Mod}(\varphi) \) the set of all interpretations that satisfy \( \varphi \)

\[
\text{Mod}(\varphi) = \{ \sigma \mid \sigma \models \varphi \}.
\]

This notation extends to sets of formulas in a natural way.

Lemma 5.5. Let \( \varphi \) be a formula. Then

1. \( \text{Mod}(\neg \varphi) = (2^\Xi)^\omega \setminus \text{Mod}(\varphi) \).
Proof. Let σ be an interpretation. Then σ ∈ \text{Mod}(\neg \varphi) if and only if σ \not\models \neg \varphi \iff σ, 0 \not\models \varphi if and only if σ, 0 \not\models \varphi if and only if σ, 0 \not\models \varphi if and only if σ \not\models \varphi if and only if σ \not\models \varphi if and only if σ \not\models \varphi.

\textbf{Lemma 5.6.} The satisfaction of the abbreviations is as follows:

- σ, i \models F \varphi whenever σ, j \models \varphi for some j ≥ i
- σ, i \models G \varphi whenever σ, j \models \varphi for every j ≥ i
- σ, i \models F \varphi whenever σ, j \models \varphi for some j > i
- σ, i \models G \varphi whenever σ, j \models \varphi for every j > i
- σ, i \models \varphi_1 W \varphi_2 if either there is j ≥ i such that σ, j \models \varphi_2 and σ, k \models \varphi_1, for every i ≤ k < j, or σ, j \models \varphi_1, for every j ≥ i
- σ, i \models \varphi_1 R \varphi_2 whenever either σ, j \models \varphi_2, for all j ≥ i, or there is j ≥ i such that σ, j \models \varphi_1 and σ, k \models \varphi_2, for every i ≤ k ≤ j.

The proof of this lemma is left as an exercise.

\textbf{Example 5.7.} The following formulas are valid:

1. (X\neg \varphi) \iff (\neg X \varphi)
2. F_0(\varphi_1 \land \varphi_2) \rightarrow (F_0 \varphi_1 \land F_0 \varphi_2)
3. F_0(\varphi_1 \lor \varphi_2) \leftrightarrow (F_0 \varphi_1 \lor F_0 \varphi_2)
4. (\psi_1 U \psi_2) \leftrightarrow (\psi_2 \lor (\psi_1 \land X(\psi_1 U \psi_2)))

The following formula is not valid, but is satisfiable:

5. (F_0 p_1 \land F_0 p_2) \rightarrow F_0(p_1 \land p_2)

To prove that formula 1 is valid, we need to show that every interpretation σ satisfies (X\neg \varphi) \iff (\neg X \varphi), that is, σ, 0 \models (X\neg \varphi) \iff (\neg X \varphi), which amounts to prove that, σ, 0 \models X\neg \varphi if and only if σ, 0 \models \neg X \varphi:

\begin{align*}
\sigma, 0 \models X \neg \varphi & \iff \sigma, 1 \not\models \neg \varphi \\
& \iff \sigma, 1 \not\models \varphi \\
& \iff \sigma, 0 \not\models X \varphi \\
& \iff \sigma, 0 \not\models \neg X \varphi
\end{align*}
To prove that formula 2 is valid, assume that there is an interpretation \( \sigma \) such that \( \sigma, 0 \not\models F_0(\varphi_1 \land \varphi_2) \rightarrow (F_0 \varphi_1 \land F_0 \varphi_2) \). Then, it follows that

(a) \( \sigma, 0 \models F_0(\varphi_1 \land \varphi_2) \)

(b) \( \sigma, 0 \not\models F_0 \varphi_1 \land F_0 \varphi_2 \)

Condition (a), by Lemma 5.6, implies that there is \( j \geq 0 \) such that \( \sigma, j \models \varphi_1 \land \varphi_2 \), that is, \( \sigma, j \models \varphi_1 \) and \( \sigma, j \models \varphi_2 \). By definition of satisfaction relation, we may rephrase condition (b) as

(b) \( \sigma, 0 \not\models F_0 \varphi_1 \) or \( \sigma, 0 \not\models F_0 \varphi_2 \)

If \( \sigma, 0 \not\models F_0 \varphi_1 \) then, by Lemma 5.6, it follows that there is no \( k_1 \geq 0 \) such that \( \sigma, k_1 \models \varphi_1 \), that is, \( \sigma, k_1 \not\models \varphi_1 \) for every \( k_1 \geq 0 \). But, we know that, in particular, \( \sigma, j \models \varphi_1 \). Hence, it must be the case that \( \sigma, 0 \not\models F_0 \varphi_2 \). But, by a similar argument, this implies that \( \sigma, k_2 \not\models \varphi_2 \) for every \( k_2 \geq 0 \) which is clearly impossible, given that \( \sigma, j \models \varphi_2 \). Hence, our initial assumption leads to a contradiction and, consequently, we can conclude that there is no interpretation structure \( \sigma \) such that \( \sigma \not\models F_0(\varphi_1 \land \varphi_2) \rightarrow (F_0 \varphi_1 \land F_0 \varphi_2) \) and so the formula is valid.

The proof that formulas 3 and 4 are valid is similar to the previous and we omit it.

To prove that formula 5 is not valid we must provide an interpretation \( \sigma' \) that does not satisfy the formula, that is, \( \sigma' \not\models (F_0 p_1 \land F_0 p_2) \rightarrow (F_0 (p_1 \land p_2)) \). Hence, we must have \( \sigma', 0 \not\models (F_0 p_1 \land F_0 p_2) \rightarrow (F_0 (p_1 \land p_2)) \), i.e.

(a) \( \sigma', 0 \models F_0 p_1 \land F_0 p_2 \)

(b) \( \sigma', 0 \not\models F_0 (p_1 \land p_2) \)

Condition (a) can be rephrased as

- \( \sigma', 0 \models F_0 p_1 \) and \( \sigma', 0 \models F_0 p_2 \)

If \( \sigma', 0 \models F_0 p_1 \) then there is \( j_1 \geq 0 \) such that \( \sigma', j_1 \models p_1 \), that is, \( p_1 \in \sigma'[j_1] \). If \( \sigma', 0 \models F_0 p_2 \) then there is \( j_2 \geq 0 \) such that \( \sigma', j_2 \models p_2 \), that is, \( p_2 \in \sigma'[j_2] \). Condition (b) implies that either \( \sigma', j \not\models p_1 \) or \( \sigma', j \not\models p_2 \), for every \( j \geq 0 \), that is, there is no \( j \) such that \( p_1, p_2 \in \sigma'[j] \). Consider the interpretation \( \sigma' = v'_0 v'_1 \ldots \) such that
Clearly, choosing $j_1 = 1$ and $j_2 = 2$ condition (a) is met. In fact, $p_1 \in \sigma'[j_1] = v'_1$ and $p_2 \in \sigma'[j_2] = v'_2$. Furthermore, there is no $j$ such that $p_1, p_2 \in \sigma'[j]$ and, so, condition (b) is also met and we may conclude that the formula is not valid. To show that the formula is satisfiable, we need to find an interpretation $\sigma''$ such that $\sigma'' \models (F \circ p_1 \land F \circ p_2) \rightarrow F \circ (p_1 \land p_2)$. We leave it for the interested reader to show that the interpretation $\sigma'' = v''_0 v''_1 \ldots$ such that $v''_i = \{p_1, p_2\}$ for every $i \geq 0$ satisfies the formula.

In the sequel, the following abbreviations will be useful.

\[
\varphi \Rightarrow \psi \equiv_{\text{def}} G_0(\varphi \rightarrow \psi) \\
\varphi \Leftrightarrow \psi \equiv_{\text{def}} G_0(\varphi \leftrightarrow \psi)
\]

We now relate temporal formulas with transition systems. It is not very difficult to see that, given a transition system over a set of propositional symbols, its traces are in fact interpretations for LTL over the same set of propositional symbols. Henceforth we assume that the transition system does not have terminal states in order to guarantee that all paths and traces are infinite. For semantics purposes, it is not relevant whether the transition system is finite. However, later, when we present the model checking algorithm, the finiteness of the transition system will be required.

**Definition 5.8.** Let $T = \langle S, A, \rightarrow, I, \Xi, L \rangle$ be a transition system without terminal states and let $\varphi$ be an LTL formula over $\Xi$. Let $\pi$ be an infinite path fragment of $T$. Then, $\pi$ satisfies $\varphi$, written $\pi \models \varphi$, when trace($\pi$) $\models \varphi$. Let $s \in S$. Then, we say that the state $s$ satisfies $\varphi$, written $s \models \varphi$, if $\pi \models \varphi$, for every $\pi \in \text{Paths}(s)$. Finally, we say that the transition system $T$ satisfies $\varphi$, written $T \models \varphi$, if Traces($T$) $\subseteq$ Mod($\varphi$).

From this definition, it follows that, for every infinite path fragment $\pi$ of $T$

\[\pi \models \varphi \text{ if and only if trace}(\pi) \in \text{Mod}(\varphi).\]

It also follows

\[T \models \varphi \text{ if and only if } s_0 \models \varphi, \text{ for all } s_0 \in I.\]
We leave it for the interested reader to work out the details.

We need to be careful with negation. For a given path \( \pi \) and a formula \( \varphi \), it is always that case that \( \pi \notsat \varphi \) if and only if \( \pi \sat \neg \varphi \). This is an immediate consequence of the above observation and of Lemma 5.5. However, in the case of transition systems, the question is a bit more delicate. In general, the statements \( T \notsat \varphi \) and \( T \sat \neg \varphi \) are not equivalent. We can prove that if \( T \sat \neg \varphi \) then \( T \notsat \varphi \) but, for the converse nothing can be said, in general. Observe that

\[
T \notsat \varphi \quad \text{if and only if} \quad \text{Traces}(T) \not\subseteq \text{Mod}(\varphi)
\]

\[
\text{if and only if} \quad \text{Traces}(T) \setminus \text{Mod}(\varphi) \neq \emptyset
\]

\[
\text{if and only if} \quad \text{Traces}(T) \cap \text{Mod}(\neg \varphi) \neq \emptyset.
\]

It is not very difficult to provide an example of a transition system that satisfies neither \( \varphi \) nor \( \neg \varphi \). We only need to provide two paths \( \pi_1 \) and \( \pi_2 \) such that \( \pi_1 \sat \varphi \) and \( \pi_2 \sat \neg \varphi \). In this case, \( T \notsat \varphi \) and \( T \notsat \neg \varphi \) both hold.

**Example 5.9.** Recall the transition system \( T_1 \parallel_H T_2 \) presented in Example 3.3 and the LTL formulas representing the properties \( P_1 \) and \( P_2 \) defined in Example 5.1. It is not very difficult to see that

\[
T_1 \parallel_H T_2 \sat G_0 (\neg \text{green}_1 \lor \neg \text{green}_2)
\]

given that there is no reachable state in \( T_1 \parallel_H T_2 \) where both \( \text{green}_1 \) and \( \text{green}_2 \) hold at the same time.

We introduce the notion of equivalence of LTL formulas as well as some examples. These will be useful later.

**Definition 5.10.** Two LTL formulas \( \varphi_1 \) and \( \varphi_2 \) are said to be equivalent, denoted by \( \varphi_1 \equiv \varphi_2 \), if \( \text{Mod}(\varphi_1) = \text{Mod}(\varphi_2) \).

**Lemma 5.11.** The following hold

- \( \neg X \varphi \equiv X \neg \varphi \)
- \( \neg F_0 \varphi \equiv G_0 \neg \varphi \)
- \( \neg G_0 \varphi \equiv F_0 \neg \varphi \)
• $F_0 F_0 \varphi \equiv F_0 \varphi$
• $G_0 G_0 \varphi \equiv G_0 \varphi$
• $F_0 G_0 F_0 \varphi \equiv G_0 F_0 \varphi$
• $G_0 F_0 G_0 \varphi \equiv F_0 G_0 \varphi$
• $\varphi U \psi \equiv \psi \lor (\varphi \land X(\varphi U \psi))$
• $F_0 \varphi \equiv \varphi \lor XF_0 \varphi$
• $G_0 \varphi \equiv \varphi \land XG_0 \varphi$
• $X(\varphi U \psi) \equiv (X \varphi) U (X \psi)$
• $X(\varphi \lor \psi) \equiv (X \varphi) \lor (X \psi)$
• $X(\varphi \land \psi) \equiv (X \varphi) \land (X \psi)$

The proof of this lemma is left as an exercise. Some of these formulas play an important role. For instance, the formula

$$\varphi U \psi \equiv \psi \lor (\varphi \land X(\varphi U \psi))$$

gives a recursive characterization of the temporal operator $U$.

### 5.1.3 Positive Normal Form

Any LTL formula can be transformed into a canonical form, the *positive normal form* (PNF). In these formulas, negations can only appear adjacent to propositional symbols (See Chapter 1 for details in the case of propositional logic).

In order to transform any LTL formula into a PNF formula, for each operator in the language, we need to have a in the language a dual operator. Hence, for the constant true we need to consider the constant false, for the conjunction connective $\land$ we need to consider the disjunction connective $\lor$.

By Lemma 5.11 the operator $X$ is dual of itself. Finally, consider the $U$ operator. We start by observing that

$$\neg(\varphi U \psi) \equiv ((\varphi \land \neg \psi) \lor (\neg \varphi \land \neg \psi)) \lor G_0 (\varphi \land \neg \psi).$$

This suggests the use of the unless operator $W$ defined above. Then, $U$ and $W$ are self duals as expressed in the following lemma.
5.1. LINEAR TEMPORAL LOGIC

Lemma 5.12. Let $\varphi, \psi$ be LTL formulas. Then, the following hold

1. $\neg(\varphi U \psi) \equiv ((\varphi \land \neg \psi) W (\neg \varphi \land \neg \psi))$
2. $\neg(\varphi W \psi) \equiv ((\varphi \land \neg \psi) U (\neg \varphi \land \neg \psi))$.

The proof of this lemma is left as an exercise.

Definition 5.13. The set of LTL formulas in unless positive normal form (uPNF) is defined by

$\varphi ::= \text{true} \mid \text{false} \mid p \mid \neg p \mid \varphi \land \varphi \mid \varphi \lor \varphi \mid X \varphi \mid \varphi U \varphi \mid \varphi W \varphi$.

We can state the theorem that establishes, for any LTL-formula $\varphi$, the existence of an formula formula $\bar{\varphi}$ in uPNF.

Theorem 5.14. Let $\varphi$ be an LTL formula. Then, there exists an LTL formula $\bar{\varphi}$ in uPNF such that $\varphi \equiv \bar{\varphi}$.

Proof. The proof of this result is similar to the proof of Proposition 1.3. We only outline the conversion function for the relevant cases: The proof follows by induction on the structure of the formula $\varphi$, using the following translation:

- $\text{conv}(\neg X \varphi) = X \text{conv}(\neg \varphi)$,
- $\text{conv}(\neg(\varphi U \psi)) = \text{conv}(\varphi \land \neg \psi) W \text{conv}(\neg \varphi \land \neg \psi)$,
- $\text{conv}(\neg(\varphi W \psi)) = \text{conv}(\varphi \land \neg \psi) U \text{conv}(\neg \varphi \land \neg \psi)$
- $\text{conv}(X \varphi) = X \text{conv}(\varphi)$,
- $\text{conv}(\varphi U \psi) = \text{conv}(\varphi) U \text{conv}(\psi)$,
- $\text{conv}(\varphi W \psi) = \text{conv}(\varphi) W \text{conv}(\psi)$.

The problem with this conversion to uPNF it that it may be exponential in the size of the formula due to the translation of the until operator. To avoid this exponential explosion, we adopt the release operator as dual of the until operator.
**Definition 5.15.** The set of LTL formulas in *release positive normal form* (rPNF) is defined by

\[ \varphi ::= \text{true} \mid \text{false} \mid p \mid \neg p \mid \varphi \land \varphi \mid \varphi \lor \varphi \mid X \varphi \mid \varphi U \varphi \mid \varphi R \varphi. \]

**Theorem 5.16.** Let \( \varphi \) be an LTL formula. Then, there exists an LTL formula \( \overline{\varphi} \) in rPNF such that \( \varphi \equiv \overline{\varphi} \).

*Proof.* The proof follows by induction on the structure of the formula \( \varphi \). We consider only the relevant cases, as all the others are similar to the case of Theorem 5.14 using the following translation:

- \( \neg (\varphi U \psi) \) is translated to \( \neg (\neg \varphi R \neg \psi) \)
- \( \neg (\varphi R \psi) \) is translated to \( \neg (\neg \varphi U \neg \psi) \)

\( \square \)

### 5.2 LTL Model checking

Given a transition system \( T \) and an LTL formula \( \varphi \) that formalizes some requirement of \( T \) we want to check whether \( T \vDash \varphi \). If this fails, then an error trace must be provided.

Recall that we are assuming that the transition system is finite and has no terminal states. If \( T \vDash \varphi \) is not true then there must be a path \( \pi \) in \( T \) such that \( \pi \not\vDash \varphi \). If such a path does indeed exist then a prefix of it is returned as error trace. If no such path is found then we can conclude that \( T \vDash \varphi \). The approach to look for the path is based on the fact that is always possible to associate a nondeterministic Büchi automaton with the temporal formula \( \varphi \).

Observe that

\[
T \vDash \varphi \quad \text{if and only if} \quad \text{Traces}(T) \subseteq \text{Mod}(\varphi)
\]

\[
\text{if and only if} \quad \text{Traces}(T) \cap ((2^\mathbb{N})^\omega \setminus \text{Mod}(\varphi)) = \emptyset
\]

\[
\text{if and only if} \quad \text{Traces}(T) \cap \text{Mod}(\neg \varphi) = \emptyset.
\]

Hence, given an NBA \( \mathcal{A} \) with \( \mathcal{L}_\omega(\mathcal{A}) = \text{Mod}(\neg \varphi) \) then

\[
T \vDash \varphi \text{ if and only if } \text{Traces}(T) \cap \mathcal{L}_\omega(\mathcal{A}) = \emptyset.
\]
Then, to check it $T \models \varphi$, we start by constructing an NBA for $\neg \varphi$ and then apply the techniques presented in the previous chapter for solving the intersection problem. In Figure 5.1 we present the general algorithm just described.

**Input**: transition system $T$ without terminal states, and LTL formula $\varphi$

**Output**: “yes” if $T \models \varphi$, “no” with counterexample otherwise

1. Construct an NBA $A_{\neg \varphi}$ such that $L_\omega(A_{\neg \varphi}) = \text{Mod}(\neg \varphi)$
2. Construct the product transition system $T \otimes A_{\neg \varphi}$
3. If there is a path $\pi$ in $T \otimes A_{\neg \varphi}$ satisfying the accepting condition of $A_{\neg \varphi}$ then
   - return “no” and an expressive prefix of $\pi$
4. Else
   - return “yes”

---

Figure 5.1: Automaton-based LTL model checking

It remains to show how, given an LTL $\varphi$ formula, we can construct an NBA $A_{\varphi}$ such that $L_\omega(A_{\varphi}) = \text{Mod}(\varphi)$. To do so, we observe that $\text{Mod}(\varphi) \subseteq (2^\Sigma)^\omega$ and so, the alphabet for $A_{\varphi}$ is $\Sigma = 2^\Sigma$. Next, we must guarantee that $\text{Mod}(\varphi)$ is $\omega$-regular in order to guarantee that it can be represented by an NBA.

We start by constructing a GNBA for $\varphi$ and then apply the result of the previous chapters to transform it into an NBA. Recall the notion of GNBA $G = (Q, \Sigma, \delta, Q_0, F)$. A GNBA $G$ for which $F$ is a singleton set can be regarded as an NBA. If the set $F$ is empty then the language $L_\omega(G)$ is the set of infinite words in $(2^\Sigma)^\omega$ that have an infinite run in $G$, that is, if $F = \emptyset$ then $G$ can be viewed as an NBA for which all states are accepting.

Observe that the GNBA $G_{\varphi}$ for $\varphi$ will have as alphabet the set $2^\Sigma$. We assume that $\varphi$ contains only the operators $\land$, $\neg$, $X$ and $U$. The derived operators $\lor$, $W$, $F_0$, $G_0$ among others are going to be expressed in terms of the basic operators. Furthermore, since the special case true is trivial, we assume that $\varphi \neq \text{true}$.

So, the idea is the following: let $\sigma = v_0 v_1 v_2 \ldots \in \text{Mod}(\varphi)$. Then, we
expand each set \( v_i \subseteq \Xi \) to a set \( \bar{v}_i \) of subformulas \( \psi \) of \( \varphi \) such that
\[
\psi \in \bar{v}_i \text{ if and only if } \sigma, i \models \psi.
\]
We denote by \( \bar{\sigma} \) the infinite sequence \( \bar{v}_0 \bar{v}_1 \bar{v}_2 \ldots \). For technical reasons, we consider the set of subformulas of \( \varphi \) but also their negations. We call this set, the closure of \( \varphi \).

**Definition 5.17.** Let \( \varphi \in \mathcal{L} \). The set of subformulas of \( \varphi \), \( SF(\varphi) \), is recursively defined as follows:

- \( SF(\varphi) = \{ \varphi \} \) when \( \varphi \in \Xi \cup \{ \text{true} \} \);
- \( SF(\neg \varphi) = \{ \neg \varphi \} \cup SF(\varphi) \);
- \( SF(\varphi_1 \land \varphi_2) = \{ \varphi_1 \land \varphi_2 \} \cup SF(\varphi_1) \cup SF(\varphi_2) \);
- \( SF(X \varphi) = \{ X \varphi \} \cup SF(\varphi) \);
- \( SF(\varphi_1 \mathcal{U} \varphi_2) = \{ \varphi_1 \mathcal{U} \varphi_2 \} \cup SF(\varphi_1) \cup SF(\varphi_2) \);

**Definition 5.18.** Let \( \varphi \in \mathcal{L} \). The closure of \( \varphi \) is the set
\[
CL(\varphi) = \{ \psi \mid \psi \in SF(\varphi) \} \cup \{ \neg \psi \mid \psi \in SF(\varphi) \}.
\]

It is not very difficult to prove that \( CL(\varphi) \) is always finite. Consider the formula \( \varphi \) to be \( p_1 \mathcal{U} (\neg p_1 \land p_2) \). Then,
\[
SF(\varphi) = \{ p_1, p_2, \neg p_1, \neg p_1 \land p_2, \varphi \}.
\]
and
\[
CL(\varphi) = \{ p_1, p_2, \neg p_1, \neg p_2, \neg p_1 \land p_2, \neg (\neg p_1 \land p_2), \varphi, \neg \varphi \}.
\]
Now, consider the interpretation \( \sigma = \{ p_1 \} \{ p_1, p_2 \} \{ p_2 \} \ldots \). The set \( v_0 = \{ p_1 \} \) is extended with the formulas \( \neg p_2, \neg (\neg p_1 \land p_2), \varphi \) since all these formulas are satisfied by \( \sigma \) at 0, and all the other formulas in \( SF(\varphi) \) do not hold. Hence,
\[
\bar{v}_0 = \{ p_1, \neg p_2, \neg (\neg p_1 \land p_2), \varphi \}.
\]
Expand all the elements \( v_i \) in a similar manner to obtain the infinite sequence
\[
\bar{v}_0 \bar{v}_1 \bar{v}_2 \ldots
\]
Unfortunately, as $\sigma$ is infinite, this procedure is not effective. This example is just for illustrating how the construction of the states of $G_\varphi$ is done.

The states of the GNBA will be the sets $\overline{v}_i$. The meaning of the logical connectives and temporal operators will be encoded into these states and also in the transitions and accepting conditions of $G_\varphi$. The meaning of the logical connectives $\neg$, $\land$ and the constant true will be encoded in the states. In this case, we require the sets $\overline{v}_i$ to be consistent. The semantics of the temporal operator $X$ does not depend solely on a local state and, consequently, has to be encoded on the transition relation. Similarly, the semantics of the temporal operator $U$ also has to be encoded in the transition relation. But in this case, according to the expansion law of this operator, we need to consider the local conditions (encoded in the states), the next state conditions (encoded in the transition relation). But we also need to guarantee that the second formula of the until will hold sometime in the future. This last requirement will be encoded in the acceptance sets of the automaton.

**Definition 5.19.** Let $\overline{v} \subseteq CL(\varphi)$. Then,

- $\overline{v}$ is **consistent** with respect to propositional logic if $\text{true} \in \overline{v}$ whenever $\text{true} \in CL(\varphi)$, and if, given $\psi_1 \land \psi_2, \neg \psi \in CL(\varphi)$,
  - $\psi_1 \land \psi_2 \in \overline{v}$ if and only if $\psi_1, \psi_2 \in \overline{v}$;
  - if $\psi \in \overline{v}$ then $\neg \psi \not\in \overline{v}$;

- $\overline{v}$ is **locally consistent** with respect to the until operator if, for $\psi_1 U \psi_2 \in CL(\varphi)$, $\overline{v}$ satisfies the following conditions:
  - if $\psi_2 \in \overline{v}$ then $\psi_1 U \psi_2 \in \overline{v}$;
  - if $\psi_1 U \psi_2 \in \overline{v}$ and $\psi_2 \not\in \overline{v}$ then $\psi_1 \in \overline{v}$;

- $\overline{v}$ is **maximal** if for every $\psi \in CL(\varphi)$
  - $\psi \not\in \overline{v}$ then $\neg \psi \in \overline{v}$;

- $\overline{v}$ is **elementary** if it is consistent with respect to propositional logic, maximal and consistent with the until operator.
Example 5.20. Consider again the formula \( \varphi = p_1 \mathbin{\text{U}} \neg p_1 \land p_2 \) and recall that \( SF(\varphi) = \{ p_1, p_2, \neg p_1, \neg p_1 \land p_2, \varphi \} \) and \( CL(\varphi) = \{ p_1, p_2, \neg p_1, \neg p_1 \land p_2, \neg(\neg p_1 \land p_2), \varphi, \neg \varphi \} \).

The set
\[
\overline{v} = \{ p_1, p_2, p_1 \mathbin{\text{U}} \neg p_1 \land p_2 \} \subseteq CL(\varphi)
\]
is consistent with respect to propositional logic and is locally consistent with respect to the until operator. However, it is not maximal since for \( \neg p_1 \land p_2 \in CL(\varphi), \neg p_1 \land p_2 \not\in \overline{v} \) and \( \neg(\neg p_1 \land p_2) \not\in \overline{v} \).

The set
\[
\overline{v'} = \{ \neg p_1, \neg p_2, \neg(\neg p_1 \land p_2), p_1 \mathbin{\text{U}} \neg p_1 \land p_2 \}
\]
is consistent with respect to propositional logic and it is maximal but it is not locally consistent with respect to the until operator since \( p_1 \mathbin{\text{U}} \neg p_1 \land p_2 \not\in \overline{v'} \) and \( (\neg p_1 \land p_2) \not\in \overline{v'} \). Observe that, for any interpretation \( \sigma \) and any \( i \in \mathbb{N} \), it cannot be the case that
\[
\sigma, i \not\models p_1 \quad \text{and} \quad \sigma, i \not\models \neg p_1 \land p_2 \quad \text{and} \quad \sigma, i \models p_1 \mathbin{\text{U}} (\neg p_1 \land p_2).
\]

The set
\[
\overline{v}_1 = \{ p_1, p_2, \neg(\neg p_1 \land p_2), p_1 \mathbin{\text{U}} (\neg p_1 \land p_2) \}
\]
is elementary. We leave it as an exercise to determine all the elementary sets.

**Theorem 5.21.** Let \( \varphi \) be an LTL formula over \( \Xi \). Then, there exists a GNBA \( \mathcal{G}_\varphi \) over the alphabet \( 2^\Xi \) such that
\[
\text{Mod}(\varphi) = L_\omega(\mathcal{G}_\varphi).
\]

**Proof.** Let \( \mathcal{G}_\varphi = \langle Q, 2^\Xi, \delta, Q_0, \mathcal{F} \rangle \) be the GNBA where

- \( Q \) is the set of all elementary sets \( \overline{v} \subseteq CL(\varphi) \);
- \( Q_0 = \{ \overline{v} \in Q \mid \varphi \in \overline{v} \} \);
- \( \mathcal{F} = \{ F_{\psi_1 \cup \psi_2} \mid \psi_1 \cup \psi_2 \in CL(\varphi) \} \) where
  \[
  F_{\psi_1 \cup \psi_2} = \{ \overline{v} \in Q \mid \psi_1 \cup \psi_2 \not\in \overline{v} \text{ or } \psi_2 \not\in \overline{v} \} ;
  \]
- \( \delta : Q \times 2^\Xi \to 2^Q \) is given by:
5.2. LTL MODEL CHECKING

– if $v \neq \tau_1 \cap \Xi$ then $\delta(\tau_1, v) = \emptyset$;
– if $v = \tau_1 \cap \Xi$ then $\delta(\tau_1, v)$ is the set of all elementary sets $\tau_2$ such that:
  1. for every $X\psi \in CL(\varphi)$: $X\psi \in \tau_1$ if and only if $\psi \in \tau_2$;
  2. for every $\psi_1 \cup \psi_2 \in CL(\varphi)$: $\psi_1 \cup \psi_2 \in \tau_1$ if and only if either $\psi_2 \in \tau_1$ or $\psi_1 \in \tau_1$ and $\psi_1 \cup \psi_2 \in \tau_2$.

These conditions reflect the semantics of the $X$ operator and of the $\cup$ operator (see Example 5.7).

Next, we prove that $L_\omega(G \varphi) = Mod(\varphi)$, by proving set inclusions in both directions.

$L_\omega(G) \supseteq Mod(\varphi)$: let $\sigma = v_0 v_1 v_2 \ldots \in Mod(\varphi)$. Then $\sigma \in (2^\Xi)^\omega$ and $\sigma \models \varphi$. For each $i \in \mathbb{N}$ define:

$$v_i = \{\psi \in CL(\varphi) \mid \sigma, i \models \psi\}.$$  

We leave it as an exercise to prove that each set $v_i$ is an elementary set, that is, $v_i \in Q$. Next, we prove that $v_0 v_1 v_2 \ldots$ is a run for $\sigma$ in $G$. To this end, it is enough to show that $v_{i+1} \in \delta(v_i, \sigma_i)$, for every $i \in \mathbb{N}$:

• clearly, $v_i = v_i \cap \Xi$, by construction of $v_i$;
• let $X\psi \in CL(\varphi)$:

$$X\psi \in v_i \quad \text{iff} \quad \sigma, i \models X\psi \quad \text{(by construction of } v_i)$$
$$\quad \text{iff} \quad \sigma, i + 1 \models \psi \quad \text{(by definition of satisfaction)}$$
$$\quad \text{iff} \quad \psi \in v_{i+1} \quad \text{(by construction of } v_{i+1})$$

• let $\psi_1 \cup \psi_2 \in CL(\varphi)$:

$$\psi_1 \cup \psi_2 \in v_i \quad \text{iff} \quad \sigma, i \models \psi_1 \cup \psi_2$$
$$\quad \text{iff} \quad \sigma, i \models \psi_1, \text{ or } \sigma, i \models \psi_2, \text{ or } \sigma, i \models \psi_1 \text{ and } \sigma, i + 1 \models \psi_1 \cup \psi_2$$
$$\quad \text{iff} \quad \psi_2 \in v_{i+1}, \text{ or } \psi_1 \in v_i \text{ and } \psi_1 \cup \psi_2 \in v_{i+1}$$

Furthermore, $\sigma \models \varphi$ if and only if $\sigma, 0 \models \varphi$ if and only if $\varphi \in v_0$ if and only if $v_0 \in Q_0$. Hence, we know that $\sigma$ is a run for $\sigma$ in $G$. All that remains to prove is that $\sigma$ is an accepting run for $\sigma$ in $G$. Let $\psi_{1,1} \cup \psi_{2,1}, \ldots, \psi_{1,n} \cup \psi_{2,n}$ be all the until formulas in $CL(\varphi)$ and, for each $j = 1, \ldots, n$, let $F_j$ be
CHAPTER 5. LINEAR TEMPORAL LOGIC

$F_{\psi_{1,j} \cup \psi_{2,j}}$. We need to show that for each $F_j$ there are infinitely many $i \in \mathbb{N}$ such that $\bar{v}_i \in F_j$. Assume that this is not the case, i.e., for some $F_j$ there are only finitely many $i$ such that $\bar{v}_i \in F_j$. Hence, there is some $k$ such that $\bar{v}_{k'} \not\in F_j$, for every $k' \geq k$. In particular, $\bar{v}_k \not\in F_j$ and, by definition of $F_j$ we have that

$$\psi_{1,j} \cup \psi_{2,j} \in \bar{v}_k$$

and so, by construction of $\bar{v}_k$,

$$\sigma, k \models \psi_{1,j} \cup \psi_{2,j} \text{ and } \sigma, k \not\models \psi_{2,j}.$$

By satisfaction, there exists $k_1 \geq k$ such that $\sigma, k_1 \models \psi_{2,j}$. Observe that $k_1 > k$. Hence, $\psi_{2,j} \in \bar{v}_{k_1}$ implying that $\bar{v}_{k_1} \in F_j$, which contradicts our assumption. Thus, $\bar{v}$ is an accepting run for $\sigma$ in $G$ and, consequently, $\bar{v} \in L_\omega(G\varphi)$.

$L_\omega(G) \subseteq \text{Mod}(\varphi)$: let $\sigma = v_0 v_1 v_2 \cdots \in L_\omega(G\varphi)$. That is, there is an accepting run $\bar{v}_0 \bar{v}_1 \bar{v}_2 \cdots$ for $\sigma$. It is easy to see that $\bar{v}_i = \bar{v}_i \cap \Xi$, for every $i \in \mathbb{N}$, given that if $\bar{v}_i \neq \bar{v}_i \cap \Xi$ then $\delta(\bar{v}_i, \bar{v}_i) = \emptyset$. We need to show that $\sigma \models \varphi$, that is,

$$(\bar{v}_0 \cap \Xi)(\bar{v}_1 \cap \Xi)(\bar{v}_2 \cap \Xi)\ldots \models \varphi.$$

To this end, we prove that

$$\psi \in \bar{v}_j \text{ if and only if } \sigma, j \models \psi, \text{ for every } j \in \mathbb{N}$$

for all $\psi \in CL(\varphi)$. The proof follows by structural induction in $\psi$.

$\psi$ is true: as $\bar{v}_j$ is consistent with respect to propositional logic then $\text{true} \in \bar{v}_j$. On the other hand, $\sigma, j \models \text{true}$, by definition. Hence, the result follows.

$\psi$ is $p \in \Xi$: the result follows by construction, given that $v_j = \bar{v}_j \cap \Xi$.

$\psi$ is $\neg \psi_1$:

$$\neg \psi_1 \in \bar{v}_j \text{ iff } \psi_1 \not\in \bar{v}_j \text{ (definition of satisfaction)}$$

$$\neg \psi_1 \in \bar{v}_j \text{ iff } \sigma, j \not\models \psi_1 \text{ (induction hypothesis)}$$

$$\neg \psi_1 \in \bar{v}_j \text{ iff } \sigma, j \models \neg \psi_1 \text{ (definition of satisfaction)}$$

$\psi$ is $\psi_1 \land \psi_2$:

$$\psi_1 \land \psi_2 \in \bar{v}_j \text{ iff } \psi_1, \psi_2 \in \bar{v}_j \text{ (definition of satisfaction)}$$

$$\psi_1 \land \psi_2 \in \bar{v}_j \text{ iff } \sigma, j \models \psi_1 \text{ and } \sigma, j \models \psi_2 \text{ (induction hypothesis)}$$

$$\psi_1 \land \psi_2 \in \bar{v}_j \text{ iff } \sigma, j \models \psi_1 \land \psi_2 \text{ (definition of satisfaction)}$$
5.2. LTL MODEL CHECKING

ψ is $X \psi_1$:  

$$X \psi_1 \in \pi_j \quad \text{iff} \quad \psi_1 \in \pi_{j+1} \quad (\pi_{j+1} \in \delta(\pi_j, v_j))$$  

$$\sigma, j + 1 \Vdash \psi_1 \quad \text{(induction hypothesis)}$$  

$$\sigma, j \Vdash X \psi_1 \quad \text{(definition of satisfaction)}$$

ψ is $\psi_1 \cup \psi_2$:

$(\to)$ Assume that $\psi_1 \cup \psi_2 \in \pi_j$. Since $\pi_j$ is elementary, either $\psi_1 \in \pi_j$ or $\psi_2 \in \pi_j$. If $\psi_2 \in \pi_j$ then by induction hypothesis, it follows that $\sigma, j \Vdash \psi_2$ and, consequently, $\sigma, j \Vdash \psi_1 \cup \psi_2$. If $\psi_2 \notin \pi_j$ then $\psi_1 \in \pi_j$. By definition of accepting run, there must be at least one $\pi_k \in F_{\psi_1 \cup \psi_2}$, with $k \geq j$. Let $k_1 \geq j$ be such that $\pi_{k_1} \in F_{\psi_1 \cup \psi_2}$ and $\pi_{k'} \notin F_{\psi_1 \cup \psi_2}$, for all $k'$ such that $j \leq k' < k_1$. Then, $\psi_1 \cup \psi_2 \in \pi_{k'}$ and $\psi_2 \notin \pi_{k'}$, by definition of $F_{\psi_1 \cup \psi_2}$. As these sets are elementary then it must also be the case that $\psi_1 \in \pi_{k'}$. Furthermore, using the definition of the transition relation, if $\psi_1 \cup \psi_2, \psi_1 \in \pi_{k_1-1}$ and $\psi_2 \notin \pi_{k_1-1}$ it must be the case that $\psi_1 \cup \psi_2 \in \pi_{k_1}$ and, as $\pi_{k_1} \in F_{\psi_1 \cup \psi_2}$ it follows that $\psi_2 \in \pi_{k_1}$. Using the induction hypothesis, it follows that $\sigma, k_1 \Vdash \psi_2$ and $\sigma, k' \Vdash \psi_1$, for every $k'$ such that $j \leq k' < k_1$. Hence, using the definition of satisfaction relation, it also follows that $\sigma, j \Vdash \psi_1 \cup \psi_2$.

$(\leftarrow)$ Assume now that $\sigma, j \Vdash \psi_1 \cup \psi_2$. Then, there is some $k_1 \geq j$ such that $\sigma, k_1 \Vdash \psi_2$ and $\sigma, k' \Vdash \psi_1$, for every $k'$ such that $j \leq k' < k_1$. By induction hypothesis, it follows that $\psi_2 \in \pi_{k_1}$ and $\psi_1 \in \pi_{k'}$, for every $k'$ such that $j \leq k' < k_1$.

In order to conclude that $\psi_1 \cup \psi_2 \in \pi_j$ we prove the following result:

If $\psi_1 \cup \psi_2 \in \pi_{k_1}$ then $\psi_1 \cup \psi_2 \in \pi_{k'}$, for every $k'$ such that $j \leq k' \leq k_1$.

The proof follows by induction on $k_1$.

**Base of induction:** For $k_1 = j$, we know that $\pi_{k_1}$ is elementary and $\psi_2 \in \pi_{k_1}$. Hence $\psi_1 \cup \psi_2 \in \pi_{k_1}$ and the result follows.

**Induction step:** Assume that $k_1 > j$. As before, we know that $\psi_1 \cup \psi_2 \in \pi_{k_1}$. Furthermore, we also know that $\psi_1 \in \pi_{k_1-1}$. Hence, given that $\pi_{k_1} \in \delta(\pi_{k_1-1}, v)$, for some $v$, then it must be the case that $\psi_1 \cup \psi_2 \in \pi_{k_1-1}$. Thus, the result follows by induction hypothesis. □
Example 5.22. Consider the formula \( \varphi = Xp \). Let us construct the GNBA

\[ G_\varphi = (Q, 2^{\{p\}}, \delta, Q_0, F) \]

using the construction described in the proof of Theorem 5.21.

The closure of \( \varphi \) is

\[ CL(Xp) = \{ p, \neg p, Xp, \neg Xp \} . \]

The elementary sets are

\[ v_1 = \{ p, Xp \} \quad v_2 = \{ \neg p, Xp \} \quad v_3 = \{ p, \neg Xp \} \quad v_4 = \{ \neg p, \neg Xp \} . \]

Hence, the set of states is

\[ Q = \{ v_1, v_2, v_3, v_4 \} . \]

The set of initial states \( Q_0 \) contains all the states \( v \) such that \( Xp \in v \). Consequently,

\[ Q_0 = \{ v_1, v_2 \} . \]

Let us focus on the transition function \( \delta \). Consider the state \( v_1 \). As \( v_1 \cap \Xi = \{ p \} \) then \( \delta(v_1, \emptyset) = \emptyset \). Furthermore, \( \delta(v_1, \{ p \}) = \{ v_1, v_3 \} \) because \( Xp \in v_1 \) and \( v_1 \) and \( v_3 \) are the only states that contain \( p \). Consider now the state \( v_2 \). As \( v_2 \cap \Xi = \emptyset \) then \( \delta(v_2, \emptyset) = \emptyset \). In addition, \( \delta(v_2, \{ p \}) = \{ v_1, v_3 \} \) because \( Xp \in v_2 \) and \( v_1 \) and \( v_3 \) are the only states that contain \( p \). The remaining transitions are determined analogously. The full GNBA is depicted in Figure 5.2.

The set \( F \) is empty because \( Xp \) does not have any subformula containing the until operator. This means that every run in \( G_\varphi \) is accepting.

It is left for the interested reader to check that every infinite run in \( G_\varphi \) satisfies the formula \( Xp \).

Example 5.23. Consider the formula \( \varphi = p_1 \mathbin{\mathbf{U}} p_2 \). Let us construct the GNBA

\[ G_\varphi = (Q, 2^{\{p_1,p_2\}}, \delta, Q_0, F) \]

using again the construction described in the proof of Theorem 5.21.

The closure of \( \varphi \) is

\[ CL(\varphi) = \{ p_1, p_2, \neg p_1, \neg p_2, p_1 \mathbin{\mathbf{U}} p_2, \neg(p_1 \mathbin{\mathbf{U}} p_2) \} . \]
5.2. LTL MODEL CHECKING

The elementary sets in this case are

\[ v_1 = \{p_1, p_2, p_1 \cup p_2\} \]
\[ v_2 = \{\neg p_1, p_2, p_1 \cup p_2\} \]
\[ v_3 = \{p_1, \neg p_2, p_1 \cup p_2\} \]
\[ v_4 = \{\neg p_1, \neg p_2, \neg (p_1 \cup p_2)\} \]
\[ v_5 = \{p_1, \neg p_2, \neg (p_1 \cup p_2)\} \]

Hence, the set of states is

\[ Q = \{v_1, v_2, v_3, v_4, v_5\} \]

and the set of initial states contains all the sets \( \bar{v} \) such that \( p_1 \cup p_2 \in \bar{v} \), that is

\[ Q_0 = \{v_1, v_2, v_3\} \].

The set \( F \) contains only one set because there is only one formula containing the until operator and that is \( \varphi \). Hence, \( F = \{F_\varphi\} \) where

\[ F_\varphi = \{\bar{v} \in Q \mid \varphi \not\in \bar{v} \text{ or } p_2 \in \bar{v}\} = \{v_1, v_2, v_4, v_5\} \].

The definition of the transition function is left as an exercise.
Observe that in the previous examples both GNBA's can be easily trans-
formed into NBAs. In the case of the GNBA for the formula $Xp$ we just
need to consider the NBA with set of final states equal to the set of states
(why?). In the case of the formula $p_1 U p_2$ the resulting GNBA has only one
set of acceptance states which means that it can already be seen as an NBA.
The algorithm for converting a GNBA into an NBA depends on the number
of sets in $F$. And this number depends on the number of subformulas that
contain the temporal operator $U$.

5.3 Axiomatization

We now present a deductive system for establishing the validity of LTL for-
mulas. A tautological formula is a formula obtained from a propositional
tautology uniformly replacing each propositional symbol by a temporal for-

mula. Consider the following tautology

$$p \rightarrow (q \rightarrow p)$$

An example of a tautological formula is the formula

$$X\varphi \rightarrow (G\psi \rightarrow X\varphi)$$

where the propositional symbol $p$ was replaced by the temporal formula $X\varphi$
and the propositional symbol $q$ was replaced by the temporal formula $G\psi$.

The axioms are the following

(A0) $G_0 \varphi$, for every tautological formula $\varphi$

(A1) $(G_0 \varphi) \rightarrow \varphi$

(A2) $(X \neg \varphi) \Leftrightarrow (\neg X \varphi)$

(A3) $X(\varphi \rightarrow \psi) \Rightarrow (X \varphi \rightarrow X \psi)$

(A4) $G_0(\varphi \rightarrow \psi) \Rightarrow (G_0 \varphi \rightarrow G_0 \psi)$

(A5) $G_0 \varphi \rightarrow G_0 X \varphi$

(A6) $(\varphi \Rightarrow X \varphi) \rightarrow (\varphi \Rightarrow G_0 \varphi)$
5.3. AXIOMATIZATION

(A7) \( \varphi \cup \psi \iff \psi \vee (\varphi \land X(\varphi \cup \psi)) \)

(A8) \( \varphi \cup \psi \Rightarrow F_0 \psi \)

The rules are the following

(MP) \( \varphi, \varphi \Rightarrow \psi \vdash \psi \)

Definition 5.24. Let \( \Gamma \cup \{\varphi\} \subseteq \mathcal{L} \). We say that \( \varphi \) is derived from \( \Gamma \), \( \Gamma \vdash \varphi \) if there is a finite sequence \( \gamma_1, \ldots, \gamma_n \) of formulas such that:

- \( \gamma_n \) is \( \varphi \)
- for \( i = 1, \ldots, n \), each \( \gamma_i \) is either an axiom, or \( \gamma_i \in \Gamma \) or \( \gamma_i \) was obtained from \( \gamma_j \) and \( \gamma_k \), with \( j, k < i \), by rule (MP).

The length of the derivation is \( n \). When \( \vdash \varphi \) we say that \( \varphi \) is a theorem.

Example 5.25. We show that \( G_0 \varphi \vdash \varphi \). Consider the following derivation sequence:

1. \( G_0 \varphi \) Hyp.
2. \( G_0 \varphi \Rightarrow \varphi \) (A1)
3. \( \varphi \) MP:1,2

From now on we will call this admissible rule (PAR) (from particularization).

Next, we show that \( \varphi \Rightarrow \psi, G_0 \varphi \vdash G_0 \psi \)

1. \( \varphi \Rightarrow \psi \) Hyp.
2. \( G_0 \varphi \) Hyp.
3. \( G_0(\varphi \Rightarrow \psi) \Rightarrow (G \varphi \Rightarrow G_0 \psi) \) (A4)
4. \( G_0(G_0(\varphi \Rightarrow \psi) \Rightarrow (G \varphi \Rightarrow G_0 \psi)) \) Def. of \( \Rightarrow:3 \)
5. \( G_0(\varphi \Rightarrow \psi) \Rightarrow (G \varphi \Rightarrow G_0 \psi) \) PAR:5
6. \( G_0(\varphi \Rightarrow \psi) \) Def. of \( \Rightarrow:1 \)
7. \( G \varphi \Rightarrow G_0 \psi \) MP:6,5
8. \( G_0 \psi \) MP:7,2

From now on, we will call this admissible rule (GMP) for Generalized Modus Ponens.

Theorem 5.26 (Soundness). Let \( \varphi \) be a formula and \( \Gamma \) be a set of formulas. If \( \Gamma \vdash \varphi \) then \( \Gamma \models \varphi \).

Proof. The proof follows by induction on the length of the derivation of \( \varphi \) from \( \Gamma \). \( \square \)
5.3.1 Decidability

In this section, we prove that LTL is decidable. Explain what decidable means and what is the idea behind the decision procedure for validity (exponential time).

**Definition 5.27.** Let $\varphi \in \mathcal{L}$. The length of $\varphi$, $\|\varphi\|$, is recursively defined as follows:

- $\|\varphi\| = 1$ when $\varphi \in \Xi \cup \{\text{true}\}$;
- $\|\neg \varphi\| = 1 + \|\varphi\|$;
- $\|\varphi_1 \land \varphi_2\| = 1 + \|\varphi_1\| + \|\varphi_2\|$;
- $\|X\varphi\| = 1 + \|\varphi\|$;
- $\|\varphi_1 \mathcal{U} \varphi_2\| = 1 + \|\varphi_1\| + \|\varphi_2\|$;

Observe that $\|p_1 \land p_2\| = 3$ and $\|\text{false}\| = 2$, given that false is an abbreviation of $\neg \text{true}$ and $\|\neg \text{true}\| = 2$.

**Lemma 5.28.** The following conditions hold:

1. $\mathcal{CL}(\neg \varphi) = \{\neg \varphi, \neg \neg \varphi\} \cup \mathcal{CL}(\varphi)$;
2. $\mathcal{CL}(\varphi_1 \land \varphi_2) = \{\varphi_1 \land \varphi_2, \neg (\varphi_1 \land \varphi_2)\} \cup \mathcal{CL}(\varphi_1) \cup \mathcal{CL}(\varphi_2)$;
3. $\mathcal{CL}(X\varphi) = \{X\varphi, \neg X\varphi\} \cup \mathcal{CL}(\varphi)$;
4. $\mathcal{CL}(\varphi_1 \mathcal{U} \varphi_2) = \{\varphi_1 \mathcal{U} \varphi_2, \neg (\varphi_1 \mathcal{U} \varphi_2)\} \cup \mathcal{CL}(\varphi_1) \cup \mathcal{CL}(\varphi_2)$.

**Lemma 5.29.** For every LTL formula $\varphi$, $|\mathcal{CL}(\varphi)| \leq 2 \|\varphi\|$.

*Proof.* The proof follows by induction on the structure of $\varphi$.

$\varphi \in \Xi \cup \{\text{true}\}$: In this case, $\mathcal{SF}(\varphi) = \{\varphi\}$ and so $\mathcal{CL}(\varphi) = \{\varphi, \neg \varphi\}$. Furthermore, $\|\varphi\| = 1$ and $|\mathcal{CL}(\varphi)| = 2$ and the result follows.

$\varphi$ is $\neg \varphi_1$: By Lemma 5.28, we know that $\mathcal{CL}(\neg \varphi_1) = \{\neg \varphi_1, \neg \neg \varphi_1\} \cup \mathcal{CL}(\varphi_1)$ and, so,

$$|\mathcal{CL}(\neg \varphi_1)| = |\{\neg \varphi_1, \neg \neg \varphi_1\} \cup \mathcal{CL}(\varphi_1)| \leq 2 + |\mathcal{CL}(\varphi_1)|$$
$$\leq 2 + 2 \|\varphi_1\|$$
$$= 2(1 + \|\varphi_1\|)$$
$$= 2 \|\neg \varphi_1\|$$
where we used the induction hypothesis $|\text{CL}(\varphi_1)| \leq 2 \cdot \|\varphi_1\|$. 

$\varphi$ is $\varphi_1 \land \varphi_2$: By Lemma 5.28, in this case, we have that $\text{CL}(\varphi_1 \land \varphi_2) = \{\varphi_1 \land \varphi_2, \neg(\varphi_1 \land \varphi_2)\} \cup \text{CL}(\varphi_1) \cup \text{CL}(\varphi_2)$ and, so,

$$|\text{CL}(\varphi_1 \land \varphi_2)| = |\{\varphi_1 \land \varphi_2, \neg(\varphi_1 \land \varphi_2)\} \cup \text{CL}(\varphi_1) \cup \text{CL}(\varphi_2)|$$

$$\leq 2 + |\text{CL}(\varphi_1)| + |\text{CL}(\varphi_2)|$$

$$\leq 2 + 2 \cdot \|\varphi_1\| + 2 \cdot \|\varphi_2\|$$

$$= 2 (1 + \|\varphi_1\| + \|\varphi_2\|)$$

$$= 2 \cdot \|\varphi_1 \land \varphi_2\|$$

where, once again, we used the induction hypothesis $|\text{CL}(\varphi_1)| \leq 2 \cdot \|\varphi_1\|$ and $|\text{CL}(\varphi_2)| \leq 2 \cdot \|\varphi_2\|$.

The proof for the remaining cases is similar to the previous. $\square$

For the sake of simplicity, we identify $\neg \neg \varphi$ with $\varphi$.

**Definition 5.30.** Let $\varphi \in L$. A **molecule** for $\varphi$ is a subset $m$ of $\text{CL}(\varphi)$ such that:

- $\varphi_1 \in m$ if and only if $\neg \varphi_1 \notin m$, for every $\varphi_1 \in \text{CL}(\varphi)$;
- $\varphi_1 \land \varphi_2 \in m$ if and only if $\varphi_1, \varphi_2 \in m$, for every $\varphi_1 \land \varphi_2 \in \text{CL}(\varphi)$;
- if $\varphi_1 \lor \varphi_2 \in m$ then either $\varphi_1 \in m$ or $\varphi_2 \in m$, and if $\varphi_2 \in m$ then $\varphi_1 \lor \varphi_2 \in m$, for every $\varphi_1 \lor \varphi_2 \in \text{CL}(\varphi)$.

The set of all molecules for $\varphi$ is denoted by $M_\varphi$.

We leave to the interested reader to show that $|M_\varphi| \leq 2^2 \cdot \|\varphi\|$.

**Definition 5.31.** A **structure for the formula** $\varphi \in L$ is a direct graph $\langle M_\varphi, R_\varphi \rangle$ where $R_\varphi$ is such that $\langle m_1, m_2 \rangle \in R_\varphi$ if and only if

- $\Box \varphi_1 \in m_1$ if and only if $\varphi_1 \in m_2$, for every $\Box \varphi_1 \in \text{CL}(\varphi)$;
- if $\varphi_1 \lor \varphi_2, \neg \varphi_2 \in m_1$ then $\varphi_1 \lor \varphi_2 \in m_2$ and if $\varphi_1 \lor \varphi_2 \in m_2$ and $\varphi_1 \in m_1$ then $\varphi_1 \lor \varphi_2 \in m_1$, for every $\varphi_1 \lor \varphi_2 \in \text{CL}(\varphi)$.

When $\langle m_1, m_2 \rangle \in R_\varphi$ then we say that $m_2$ is a **successor** of $m_1$. We say that $m_2$ is a **descendant** of $m_1$ when there is a finite path (including the empty path) from $m_1$ to $m_2$. 

Lemma 5.32. Let $m_1$ and $m_2$ be molecules for $\varphi$ such that $m_2$ is a successor of $m_1$, and $X \psi \in CL(\varphi)$. Then, $\neg X \psi \in m_1$ if and only if $\neg \psi \in m_2$.

Proof. We start by observing that $X \psi \in CL(\varphi)$ implies $\psi \in CL(\varphi)$.
Assume that $\neg X \psi \in m_1$. From the definition of molecule we know that either $\psi \in m_2$ or $\neg \psi \in m_2$. If $\psi \in m_2$ then, given that $X \psi \in CL(\varphi)$, it follows, by definition of $R_\varphi$, that $X \psi \in m_1$. This, together with our initial assumption, contradicts the definition of molecule. Hence, it must be the case that $\neg \psi \in m_2$.

For the converse, assume that $\neg \psi \in m_2$. From the definition of molecule, we know that either $X \psi \in m_1$ or $\neg X \psi \in m_1$. If $X \psi \in m_1$ then, given that $X \psi \in CL(\varphi)$ and using the definition of $R_\varphi$, it follows that $\psi \in m_2$.
But this, together with the assumption, contradicts the fact that $m_2$ is a molecule. □

Lemma 5.33. Let $m_0, m_1, \ldots, m_n$ be a finite path $\langle M_\varphi, R_\varphi \rangle$. Then, if $\varphi_1 \cup \varphi_2 \in m_0$ and $\neg \varphi_2 \in m_i$, for every $i = 1, \ldots, n - 1$, then

1. $\varphi_1 \cup \varphi_2 \in m_i$, for every $i = 1, \ldots, n$;
2. $\varphi_1 \in m_i$ for every $i = 1, \ldots, n - 1$.

Proof. The proof follows by induction in the length $n$ of the path.
For $n = 0$, conditions 1 and 2 are trivially fulfilled.
Assume now that $n > 0$. We start by proving condition 1. By induction hypothesis, we know that $\varphi_1 \cup \varphi_2 \in m_i$, for every $i = 1, \ldots, n - 1$. Hence, we only need to prove that $\varphi_1 \cup \varphi_2 \in m_n$. But, by assumption, we also know that $\neg \varphi_2 \in m_{n-1}$. Hence, by definition of $R_\varphi$, it follows that $\varphi_1 \cup \varphi_2 \in m_n$.

For condition 2, we know, by induction hypothesis, that $\varphi_1 \in m_i$ for every $i = 1, \ldots, n - 2$. As $m_{n-1}$ is a molecule and $\varphi_1 \cup \varphi_2 \in m_{n-1}$. We have that, by definition, either $\varphi_1 \in m_{n-1}$ or $\varphi_2 \in m_{n-1}$. But, we also know that $\neg \varphi_2 \in m_{n-1}$ which, again by definition of molecule, implies that $\varphi_2 \notin m_{n-1}$. Consequently, $\varphi_1 \in m_{n-1}$ and condition 2 follows. □

Definition 5.34. Let $\varphi$ be a formula and $\mathcal{G}$ a structure for $\varphi$.

1. A pre-model over $\mathcal{G}$ is an infinite path on $\mathcal{G}$.
2. A pre-model for $\varphi$ is a pre-model $m_0 m_1 \ldots$ over $\mathcal{G}$ such that $\varphi \in m_0$. 
Recall that given a direct graph $\mathcal{G} = \langle V, E \rangle$ and a subset $V'$ of $V$, the subgraph of $\mathcal{G}$ specified by $V'$ is the graph $\langle V', E' \rangle$ where $E' = E \cap (V' \times V')$.

Moreover, given an infinite path $\pi$ over a graph $\mathcal{G}$, we denote by $\text{Fin}_{\mathcal{G}}(\pi)$ the subgraph of $\mathcal{G}$ specified by all the vertexes that appear infinitely many times in $\pi$.

Given an arbitrary interpretation $\sigma$ we denote by $\pi_{\sigma}$ the pre-model $m_0 m_1 m_2 \ldots$ where $m_i = \{ \psi \in \text{CL}(\phi) \mid \sigma, i \vdash \psi \}$, for $i \geq 0$. In Exercise ?? the reader is asked to show that each $m_i$ is in fact a molecule for $\phi$ and, consequently, that $\pi_{\sigma}$ is a pre-model.

Conversely, given a pre-model $\pi = m_0 m_1 m_2 \ldots$, we can define the interpretation $\sigma_{\pi} = v_0 v_1 v_2 \ldots$ where each $v_i$ is the valuation induced by the molecule $m_i$, that is, $v_i = \{ p \in \Xi \mid p \in m_i \}$, for $i \geq 0$.

In Exercise ??, the reader is asked to show that if $\sigma$ is a model of $\phi$ then $\pi_{\sigma}$ is a pre-model for $\phi$. The converse in not, in general, true. The problem may arise with delayed eventualities.

**Definition 5.35.** Let $\pi = m_0 m_1 m_2 \ldots$ be a pre-model over $\mathcal{G}$. Then, $\pi$ is a fulfilling path if, for every $i \geq 0$ and every $\varphi_1 \cup \varphi_2 \in m_i$, there exists some $j \geq i$ such that $\varphi_2 \in m_j$. Moreover, $\pi$ is a fulfilling path for $\varphi$ whenever $\pi$ is a fulfilling path and is also a pre-model for $\varphi$.

**Theorem 5.36.** Let $\varphi \in \mathcal{L}$. Then, $\varphi$ is satisfiable if and only if there exists a fulfilling path for $\varphi$ in $\langle \mathcal{M}_\varphi, R_\varphi \rangle$.

In order to determine if a formula $\varphi$ is satisfiable it is enough to find a fulfilling path for $\varphi$ in the structure for $\varphi$. This is an immediate consequence of the previous result.

Recall that a given direct graph $\mathcal{G}$ is strongly connected if it contains a path from $v$ to $v'$, for every pair of vertices $v, v'$. Consequently, there must also be a path from $v'$ to $v$. In addition, a strongly connected subgraph (scs) of $\mathcal{G}$ is a subgraph $\mathcal{C}$ of $\mathcal{G}$ such that $\mathcal{C}$ is strongly connected. A strongly connected subgraph of $\mathcal{G}$ is maximal if it is not properly contained in a larger strongly connected subgraph of $\mathcal{G}$ (mscs).

**Lemma 5.37.** Let $\pi$ be a pre-model over $\mathcal{G}$. Then, $\text{Fin}_{\mathcal{G}}(\pi)$ is a strongly connected subgraph of $\mathcal{G}$. 


Definition 5.38. Let $C$ be a strongly connected subgraph of $\langle M_\varphi, R_\varphi \rangle$. Then, $C$ is self fulfilling if for every $\varphi_1 \cup \varphi_2 \in m$ for some molecule $m$ in $C$ there exists a molecule $m'$ in $C$ such that $\varphi_2 \in m'$.

Lemma 5.39. Let $C_1$ and $C_2$ be strongly connected subgraphs of $\langle M_\varphi, R_\varphi \rangle$ with $C_1 \subseteq C_2$. If $C_1$ is self fulfilling then $C_2$ is self fulfilling.

Definition 5.40. Let $C$ be a maximal strongly connected subgraph of $\langle M_\varphi, R_\varphi \rangle$. Then, $C$ is useless if it falls into one of the following two categories:

1. $C$ is not reachable from any molecule $m$ over $\varphi$ such that $\varphi \in m$.
2. $C$ has no outgoing $R_\varphi$ edges and is not self fulfilling.

\[
\langle M_0, R_0 \rangle := \langle M_\varphi, R_\varphi \rangle
\]

\[
i := 0
\]

while $\langle M_i, R_i \rangle$ is not empty and contains some useless mscs do

\[
\text{let } C \text{ be a useless mscs in } \langle M_i, R_i \rangle
\]

\[
i := i + 1
\]

\[
M_i := M_{i-1} \setminus C
\]

\[
R_i := R_{i-1} \cap (M_i \times M_i)
\]

end

if the final structure is not empty then

success

else

failure

fi

Figure 5.3: Algorithm for deciding the validity of an LTL formula.

We denote by $\langle M^*_\varphi, R^*_\varphi \rangle$ the final structure obtained by this removal process described in Figure 5.3

Lemma 5.41. The path $\pi$ is a fulfilling path for $\varphi$ in $\langle M_i, R_i \rangle$ if and only if $\pi$ is a fulfilling path for $\varphi$ in $\langle M_{i+1}, R_{i+1} \rangle$. 
5.3. AXIOMATIZATION

Proof. Let $\pi = m_0 m_1 m_2 \ldots$ be a fulfilling path for $\varphi$ is $\langle M_i, R_i \rangle$. We know that $\langle M_i, R_i \rangle$ and $\langle M_{i+1}, R_{i+1} \rangle$ differ from a useless graph $C$. All that we need to show is that no molecule $m_i$ is in $C$. If $C$ is useless due to condition 1 then clearly no molecule $m_k$ is in $C$ because $\varphi \in m_0$ and every $m_k$ is reachable from $m_0$. If $C$ is useless due to condition 2 then $C$ has no outgoing $R_\varphi$ edges and is not self fulfilling. Assume that $m_k$ is in $C$, for some $k \geq 0$. As $C$ has no outgoing $R_\varphi$ edges then, clearly, all the molecules that occur in $\pi$ after $m_k$ must also be in $C$, i.e., all molecules $m_{k'}$ are in $C$, for $k' \geq k$. Additionally, we also know that $C$ is not self fulfilling. Hence, there is some formula $\varphi_1 \cup \varphi_2 \in m'$ for some molecule $m'$ in $C$ such that $\varphi_2 \notin m''$ for any molecule $m''$ in $C$. Given that $C$ is strongly connected then there is a finite path from $m'$ to every molecule in $C$. Hence, by Lemma 5.33, we now that formula $\varphi_1 \cup \varphi_2$ is in every molecule of $C$ and, in particular, is in $m_k$. As $\pi$ is a fulfilling path then there must be some molecule $m_j$ with $j \geq k$ such that $\varphi_2 \in m_j$ which leads to a contradiction, given that $m_j$ must be in $C$.

Theorem 5.42. A formula $\varphi$ is satisfiable if and only if the algorithm in Figure 5.3 reports success.

Proof. Let $C^* = \langle M_\varphi^*, R_\varphi^* \rangle$.
Assume that $\varphi$ is satisfiable. Then, by Theorem 5.36, there exists a fulfilling path $\pi$ for $\varphi$ in $\langle M_\varphi, R_\varphi \rangle$. Using Lemma 5.41, we may conclude that $\pi$ is also a fulfilling path for $\varphi$ in $C^*$. In this case clearly $C^*$ is not empty and, thus, the algorithm reports success.

For the converse, assume that the algorithm reports success, that is, that $C^*$ is not empty. Then, $C^*$ does not contain any useless mscs. Hence, there must exist some molecule in $m_0$ such that $\varphi \in m_0$ (due to condition 1 in the definition of useless mscs). Furthermore, there is a finite path $\hat{\pi} = m_0 \ldots m_k$ such that $m_k$ belongs to some mscs.\footnote{Elaborate on this!!!—ideia começar a construir um caminho até encontrar um nó repetido que terá que surgir, porque o grafo é finito. Os nós entre as repeticoes formam um scs. Encontrar o maximal e ver se tem alguma seta a sair dele se tiver, repetir o processo. O proximo maximal não pode ter setas para este porque nesse caso haveria um ciclo e contrariava a noção de maximal... Este processo é finito. O mscs nestas condições tem que ser self fulfilling pois caso contrário era useless o que não pode ser porque $C^*$ não mscs useless.} From this, by definition of self fulfilling scs there is a fulfilling path $\pi$ for $\varphi$ in $C^*$. Thus, by Lemma 5.41,
\( \pi \) is also a fulfilling path for \( \varphi \) in \( \langle M_\varphi, R_\varphi \rangle \) and so, by Theorem 5.36, it follows that \( \varphi \) is satisfiable. \( \square \)

### 5.3.2 Completeness

Completeness means that every valid formula is a theorem of the axiomatic system. Talk about weak completeness and why we can’t establish strong completeness.

We start by establishing a series of auxiliary results. Given a molecule \( m \) we denote by \( \hat{m} \) the formula

\[
\bigwedge_{\psi \in m} \psi.
\]

Let \( (M_0, R_0) = (M_{\neg \varphi}, R_{\neg \varphi}) \) and let

\[
(M_0, R_0) \ldots (M_i, R_i) \ldots (M_n, R_n)
\]

with \( (M_n, R_n) = (M^*, R^*) \), be the sequence of graphs generated by the algorithm in Figure 5.3.

**Lemma 5.43.**

\[
\vdash G_\circ \left( \bigvee_{m \in M_0} \hat{m} \right)
\]

*Proof.* An immediate consequence of Exercise 1.8 using (A0). \( \square \)

**Lemma 5.44.** For every \( \psi \in \text{CL}(\varphi) \)

\[
\vdash \psi \leftrightarrow \bigvee_{m \in M_0, \psi \in m} \hat{m}
\]

*Proof.* An immediate consequence of Exercise 1.8 using (A0). \( \square \)

**Lemma 5.45.** Let \( m \in M_0 \) be some molecule. Then

\[
\vdash \hat{m} \Rightarrow X \left( \bigvee_{m' \in M_0} \hat{m}' \right)
\]
Proof. Let $\alpha$ be the formula $\bigvee_{m_1 \in M_0} \widehat{m}_1$. Consider the following derivation

1. $G_\circ \alpha$ Lemma 5.43
2. $G_\circ X \alpha$ GenX : 1
3. $\widehat{m} \Rightarrow X \alpha$ R1 : 2

Lemma 5.46. Let $m_1, m_2 \in M_0$ be molecules such that $\langle m_1, m_2 \rangle \notin R_0$. Then

$$\vdash \widehat{m}_1 \Rightarrow X(\neg \widehat{m}_2).$$

Proof. Consider the reasons for $\langle m_1, m_2 \rangle \notin R_0$:

Case 1: there is a formula $X \psi \in CL(\varphi)$ such that $X \psi \in m_1$ and $\psi \notin m_2$, that is, $\neg \psi \in m_2$:

1. $\widehat{m}_1 \Rightarrow X \psi$ A0
2. $\psi \Rightarrow \neg \psi$ A0
3. $X \psi \Rightarrow X \neg \psi$ MonX : 2
4. $\widehat{m}_1 \Rightarrow X \neg \psi$ GTrans : 1, 3
5. $\widehat{m}_2 \Rightarrow \neg \psi$ A0
6. $\neg \psi \Rightarrow \neg \widehat{m}_2$ R2 : 5
7. $X \neg \psi \Rightarrow X \neg \widehat{m}_2$ MonX : 6
8. $\widehat{m}_1 \Rightarrow X \neg \widehat{m}_2$ GTrans : 4, 7

Case 2: there is a formula $X \psi \in CL(\varphi)$ such that $X \psi \notin m_1$, that is $\neg X \psi \in m_1$, and $\psi \in m_2$:

1. $\widehat{m}_1 \Rightarrow \neg X \psi$ A0
2. $(X \neg \psi) \Leftrightarrow (X \neg \psi)$ A2
3. $(\neg X \psi) \Rightarrow (X \neg \psi)$ R3 : 2
4. $\widehat{m}_1 \Rightarrow (X \neg \psi)$ GTrans : 1, 3
5. $\widehat{m}_2 \Rightarrow \psi$ A0
6. $\neg \psi \Rightarrow \neg \widehat{m}_2$ R2 : 5
7. $X \neg \psi \Rightarrow X \neg \widehat{m}_2$ MonX : 6
8. $\widehat{m}_1 \Rightarrow X \neg \widehat{m}_2$ GTrans : 4, 7

Case 3: there is a formula $\psi_1 \cup \psi_2 \in CL(\varphi)$ such that $\psi_1 \cup \psi_2, \neg \psi_2 \in m_1$, $\psi_1 \cup \psi_2 \notin m_2$, that is, $\neg (\psi_1 \cup \psi_2) \in m_2$: 
1. \( \hat{m}_1 \Rightarrow ((\psi_1 \lor \psi_2) \land \neg \psi_2) \) \hspace{1cm} \text{A0}
2. \( (\psi_1 \lor \psi_2) \Leftrightarrow \psi_1 \land X(\psi_1 \lor \psi_2) \) \hspace{1cm} \text{A7}
3. \( (\psi_1 \lor \psi_2) \Rightarrow \psi_2 \land (\psi_1 \land X(\psi_1 \lor \psi_2)) \) \hspace{1cm} \text{R3 : 2}
4. \( ((\psi_1 \lor \psi_2) \land \neg \psi_2) \Rightarrow X(\psi_1 \lor \psi_2) \) \hspace{1cm} \text{A0+GMP : 3}
5. \( \hat{m}_1 \Rightarrow X(\psi_1 \lor \psi_2) \) \hspace{1cm} \text{GTrans : 1, 4}
6. \( \hat{m}_2 \Rightarrow \neg(\psi_1 \lor \psi_2) \) \hspace{1cm} \text{A0}
7. \( \neg \neg(\psi_1 \lor \psi_2) \Rightarrow \neg \hat{m}_2 \) \hspace{1cm} \text{R2 : 6}
8. \( (\psi_1 \lor \psi_2) \Rightarrow \neg(\psi_1 \lor \psi_2) \) \hspace{1cm} \text{A0}
9. \( (\psi_1 \lor \psi_2) \Rightarrow \neg \hat{m}_2 \) \hspace{1cm} \text{GTrans : 8, 7}
10. \( X(\psi_1 \lor \psi_2) \Rightarrow X \neg \hat{m}_2 \) \hspace{1cm} \text{MonX : 9}
11. \( \hat{m}_1 \Rightarrow X \neg \hat{m}_2 \) \hspace{1cm} \text{GTrans : 5, 10}

Case 4: there is a formula \( \psi_1 \lor \psi_2 \in CL(\varphi) \) such that \( \psi_1 \lor \psi_2 \in m_2, \psi_1 \in m_1 \) and \( \psi_1 \lor \psi_2 \notin m_1 \), that is, \( \neg(\psi_1 \lor \psi_2) \in m_1 \).

Observe that from (A7) we can prove that

\[ \vdash \neg(\psi_1 \lor \psi_2) \Rightarrow (\neg \psi_1 \lor \neg X(\psi_1 \lor \psi_2)) \]

Since \( \vdash \hat{m}_1 \Rightarrow \psi_1 \land (\neg(\psi_1 \lor \psi_2)) \) we can prove that

\[ \vdash \hat{m}_1 \Rightarrow \neg X(\psi_1 \lor \psi_2). \]

On the other hand, we also know that \( \vdash \hat{m}_2 \Rightarrow (\psi_1 \lor \psi_2) \). From this, we can conclude that

\[ \vdash \neg X(\psi_1 \lor \psi_2) \Rightarrow X \neg \hat{m}_2 \]

Therefore,

\[ \vdash \hat{m}_1 \Rightarrow X \neg \hat{m}_2. \]

\[ \square \]

**Lemma 5.47.** Let \( m \) be a molecule. Then

\[ \vdash \hat{m} \Rightarrow X(\bigvee_{m' \in M_0, (m,m') \in R_0} \hat{m}') \]

**Proof.** From Lemma 5.45, we have that

\[ \vdash \hat{m} \Rightarrow X(\bigvee_{m' \in M_0, (m,m') \in R_0} \hat{m}') \lor (\bigvee_{m' \in M_0, (m,m') \notin R_0} \hat{m}') \]
from which we can derive, using (T4) and (GTrans), that
\[ \vdash \hat{m} \Rightarrow (X \bigvee_{m' \in M_0, (m,m') \in R_0} \hat{m}') \lor (X \bigvee_{m' \in M_0, (m,m') \notin R_0} \hat{m}'). \]

Moreover, from Lemma 5.46, we know that, for every \( m' \) such that \( (m, m') \notin R_0 \) the following holds:
\[ \vdash \hat{m} \Rightarrow X \neg \hat{m}'. \]
Hence,
\[ \vdash \hat{m} \Rightarrow \bigwedge_{m' \in M_0, (m,m') \notin R_0} X \neg \hat{m}'. \]

Using (T3), (A0) and (A2), we also know that
\[ \vdash \bigwedge_{m' \in M_0, (m,m') \notin R_0} X \neg \hat{m}' \Rightarrow \neg X \bigvee_{m' \in M_0, (m,m') \notin R_0} \hat{m}'. \]
Using (GTrans) we have that
\[ \vdash \hat{m} \Rightarrow \neg X \bigvee_{m' \in M_0, (m,m') \notin R_0} \hat{m}'. \]
and, consequently,
\[ \vdash \hat{m} \Rightarrow X \bigvee_{m' \in M_0, (m,m') \in R_0} \hat{m}'. \]

Observe that for any molecule \( m \) that has no successor in \( R_0 \) then
\[ \vdash \hat{m} \Rightarrow X \text{false}. \]

We denote by \( R^* \) the reflexive and transitive closure of \( R \).

**Lemma 5.48.** Let \( m \) be a molecule. Then
\[ \vdash \hat{m} \Rightarrow G_\infty( \bigvee_{(m,m') \in R_0^*} \hat{m}'). \]
CHAPTER 5. LINEAR TEMPORAL LOGIC

Proof. Let $\alpha$ be the formula

$$\bigvee_{m' \in M_0, \langle m, m' \rangle \in R_0^*} \hat{m}' .$$

Let $m'_1, \ldots, m'_n$ be all the molecules such that $\langle m, m'_i \rangle \in R_0^*$, for $i = 1, \ldots, n$.

We start by showing that $\vdash \hat{m}'_i \Rightarrow X \alpha$. This will allow us to prove that $\vdash \alpha \Rightarrow X \alpha$, that is, if $\alpha$ holds in a state then it will also hold in the next state. Then we prove that $\hat{m}$ implies $\alpha$ which means that we will be able to conclude that, when $\hat{m}$ holds then $\alpha$ will always hold. Indeed, for each $m'_i$ we have:

1. $(\bigvee_{m'' \in M_0, \langle m'_i, m'' \rangle \in R_0} \hat{m}'') \Rightarrow \alpha \quad \text{A0}$
2. $X(\bigvee_{m'' \in M_0, \langle m'_i, m'' \rangle \in R_0} \hat{m}'') \Rightarrow X \alpha \quad \text{MonX : 1}$
3. $\hat{m}'_i \Rightarrow X(\bigvee_{m'' \in M_0, \langle m'_i, m'' \rangle \in R_0} \hat{m}'') \quad \text{Lemma 5.47}$
4. $\hat{m}'_i \Rightarrow X \alpha \quad \text{GTrans : 3, 2}$

Using rule (R4) on all the $m'_i$ we may conclude that

$$\vdash (\bigvee_{i=1}^n \hat{m}'_i) \Rightarrow X \alpha .$$

Since $\bigvee_{i=1}^n \hat{m}'_i$ is in fact $\alpha$, then

$$\vdash \alpha \Rightarrow X \alpha .$$

Finally, using (CI),

$$\vdash \alpha \Rightarrow G \alpha .$$

The result follows by (GTrans) since $R_0^*$ is reflexive, using (A0), we have

$$\vdash \hat{m} \Rightarrow \alpha .$$

Observe that for any molecule $m$ that has no descendants in $R_0$ then

$$\vdash \hat{m} \Rightarrow G_0 \text{false} .$$
Lemma 5.49.

\[ \vdash \neg \varphi \rightarrow G_0( \bigvee_{m, m' \in M_0, \langle m, m' \rangle \in R_0^* \neg \varphi \in m} \hat{m}') \].

Proof. From Lemma 5.44, we have

\[ \vdash \neg \varphi \rightarrow \bigvee_{m \in M_0, \neg \varphi \in m} \hat{m} \].

Denote by \( m_1, \ldots, m_n \) all the molecules containing \( \neg \varphi \). Then, for each \( i = 1, \ldots, n \), using Lemma 5.48, we have

\[ \vdash \hat{m}_i \Rightarrow G_0( \bigvee_{\langle m_i, m' \rangle \in R_0^*} \hat{m}') \]

and, by (PAR), we also have

\[ \vdash \hat{m}_i \rightarrow G_0( \bigvee_{\langle m_i, m' \rangle \in R_0^*} \hat{m}') \].

Using (A0) and (PAR), we can conclude that

\[ \vdash \bigvee_{i=1}^n \hat{m}_i \Rightarrow \bigvee_{i=1}^n G_0( \bigvee_{\langle m_i, m' \rangle \in R_0^*} \hat{m}') \]

which is

\[ \vdash \bigvee_{m \in M_0, \neg \varphi \in m} \bigvee_{m' \in M_0, \neg \varphi \in m} G_0( \bigvee_{\langle m_m', m' \rangle \in R_0^*} \neg \varphi \in m) \].

Hence, using (GTrans),

\[ \vdash \neg \varphi \rightarrow \bigvee_{m \in M_0, \neg \varphi \in m} G_0( \bigvee_{\langle m, m' \rangle \in R_0^*} \neg \varphi \in m) \].

On the other hand, using (T6), (A0) and (PAR)

\[ \vdash \bigvee_{m \in M_0, \neg \varphi \in m} G_0( \bigvee_{\langle m, m' \rangle \in R_0^*} \hat{m}') \rightarrow G_0( \bigvee_{m, m' \in M_0, \langle m, m' \rangle \in R_0^* \neg \varphi \in m} \hat{m}') \].

Finally, the result follows by (GTrans). \( \square \)
Lemma 5.50. For every $i = 0, \ldots, n$, and formula $\varphi$

1. For every $m \notin \mathcal{M}_i$,
   \[ \vdash \neg \varphi \rightarrow G_0 \neg \hat{m}. \]

2. For $m \in \mathcal{M}_i$,
   \[ \vdash \neg \varphi \rightarrow (\hat{m} \Rightarrow X(\bigvee_{m' \in \mathcal{M}_i, (m,m') \in R_i} \hat{m}')). \]

3. For $m \in \mathcal{M}_i$,
   \[ \vdash \neg \varphi \rightarrow (\hat{m} \Rightarrow G_0(\bigvee_{m' \in \mathcal{M}_i, (m,m') \in R^*_i} \hat{m}')). \]

4.
   \[ \vdash \neg \varphi \leftrightarrow \bigvee_{m \in \mathcal{M}_i, \neg \varphi \in m} \hat{m}. \]

5.
   \[ \vdash \neg \varphi \rightarrow G_0(\bigvee_{m, m' \in \mathcal{M}_i, (m,m') \in R^*_i, \neg \varphi \in m} \hat{m}'). \]

Proof. We prove the above properties, simultaneously, by induction on $i$.

For $i = 0$, property 1 is immediately true because $\mathcal{M}_0$ is the set of all molecules. Properties 2 and 3 are immediate consequences, by propositional reasoning, of Lemma 5.47 and Lemma 5.48. Property 4 is an instance of Lemma 5.44. Finally, property 5 is Lemma 5.49. Assume that the above properties hold for $i$. We have to prove that they also hold for $i+1$.

Property 1: Let $m$ be a molecule not in $\mathcal{M}_{i+1}$. If $m$ was in $\mathcal{M}_i$ then the result follows immediately by induction hypothesis. Otherwise, $m$ is removed when passing from $(\mathcal{M}_i, R_i)$ to $(\mathcal{M}_{i+1}, R_{i+1})$. Let $C$ be the useless mscs that is removed and $m$ is in $C$. Let us consider the two conditions for $C$ being declared useless:

(i) $C$ is not reachable from any molecule containing $\neg \varphi$:
Then, $m$ is not reachable from any molecule containing $\neg \varphi$. Denote by $\alpha$ the formula
\[
\bigvee_{m', m'' \in M_i} \langle m', m'' \rangle \in R_i^* \quad m' \in M_i \quad \neg \varphi \in m'.
\]

By propositional reasoning, we have that
\[
\vdash \alpha \rightarrow \neg \widehat{m}.
\]

\begin{align*}
1. & \quad G_o(\alpha \rightarrow \neg \widehat{m}) & \text{A0} \\
2. & \quad G_o(\alpha \rightarrow \neg \widehat{m}) \Rightarrow (G_o \alpha \rightarrow G_o \neg \widehat{m}) & \text{A4} \\
3. & \quad G_o \alpha \rightarrow G_o \neg \widehat{m} & \text{GMP : 1, 2} \\
4. & \quad \neg \varphi \rightarrow G_o \alpha & \text{Prop. 5 (induction hypothesis)} \\
5. & \quad \neg \varphi \rightarrow G_o \neg \widehat{m} & \text{A0+PAR+MP : 4, 3}
\end{align*}

This establishes Property 1 for $\langle M_{i+1}, R_{i+1} \rangle$ for this case.

(ii) $C$ has no outgoing edges and is not self fulfilling. Then, there must be some formula $\varphi_1 \cup \varphi_2$ such that every molecule $m'$ in $C$ contains $\varphi_1 \cup \varphi_2$ and $\neg \varphi_2$. Since $m$ is in $C$ then, using (A0),
\[
\vdash \widehat{m} \Rightarrow (\varphi_1 \cup \varphi_2)
\]
and
\[
\vdash \widehat{m}' \Rightarrow \neg \varphi_2.
\]

Since $C$ has no outgoing edges, any molecule $m'$ such that $\langle m, m' \rangle \in R_i^*$ is contained in $C$. Hence, from above, we have that
\[
\vdash (\bigvee_{m' \in M_i, (m, m') \in R_i^*} \widehat{m}') \Rightarrow \neg \varphi_2.
\]

Using (A4),
\[
\vdash G_o(\bigvee_{m' \in M_i, (m, m') \in R_i^*} \widehat{m}') \rightarrow G_o \neg \varphi_2.
\]

By induction hypothesis, using Property 3 and propositional reasoning,
\[
\vdash \neg \varphi \rightarrow (\widehat{m} \Rightarrow G_o \neg \varphi_2).
\]
Finally,
\[
\vdash \neg \varphi \rightarrow (\widehat{m} \Rightarrow ((\varphi_1 \cup \varphi_2) \land G_o \neg \varphi_2)).
\]
From (A8) we have
\[ \vdash (\varphi_1 U \varphi_2) \Rightarrow \neg G_0 \neg \varphi_2 \]
and so,
\[ \vdash \neg \varphi \Rightarrow (\hat{m} \Rightarrow \text{false}) \]
that is,
\[ \vdash \neg \varphi \Rightarrow G_0 \neg \hat{m}. \]
This establishes Property 1 for \( \langle M_{i+1}, R_{i+1} \rangle \) for this case.

Property 2: Let \( m \in M_{i+1} \) and let \( m_1, \ldots, m_k \) be all the molecules in \( M_i \) such that \( \langle m, m'_j \rangle \not\in R_{i+1} \), for \( j = 1, \ldots, k \). For each \( j = 1, \ldots, k \), by induction hypothesis, using Property 1, we have
\[ \vdash \neg \varphi \Rightarrow G_0 \neg \hat{m}'_j. \]
Using (A5), we have
\[ \vdash \neg \varphi \Rightarrow G_0 X \neg \hat{m}'_j. \]
Using rule (R5) and (PAR) on all the \( m_j \) we may conclude that
\[ \vdash \neg \varphi \Rightarrow \bigwedge_{j=1}^k G_0 X \neg \hat{m}'_j. \]
Then, we may apply (T7) to get
\[ \vdash \neg \varphi \Rightarrow G_0 \bigwedge_{j=1}^k X \neg \hat{m}'_j \]
and, then, by (T3)
\[ \vdash \neg \varphi \Rightarrow G_0 X \bigwedge_{j=1}^k \neg \hat{m}'_j \]
which can rephrased as
\[ \vdash \neg \varphi \Rightarrow G_0 X \bigwedge_{m' \in M'_i, \langle m, m' \rangle \in R_i \setminus R_{i+1}} \neg \hat{m}'_j. \]
Now, by induction hypothesis, we have that Property 2 holds for $\mathcal{R}_i$,

$$\vdash \neg \varphi \rightarrow (\hat{m} \Rightarrow X(\bigvee_{m' \in \mathcal{M}_i, (m, m') \in \mathcal{R}_i} \hat{m}')).$$

By propositional reasoning, we get

$$\vdash \neg \varphi \rightarrow (\hat{m} \Rightarrow X((\bigvee_{m' \in \mathcal{M}_i, (m, m') \in \mathcal{R}_i} \hat{m}')) \land (\bigwedge_{m' \in \mathcal{M}_i, (m, m') \in \mathcal{R}_i \setminus \mathcal{R}_{i+1}} \neg \hat{m}'))).$$

Again, by propositional reasoning, we derive

$$\vdash \neg \varphi \rightarrow (\hat{m} \Rightarrow X(\bigvee_{m' \in \mathcal{M}_i, (m, m') \in \mathcal{R}_{i+1}} \hat{m}')).$$

Property 3: Is similar to the proof of Lemma 5.48, but using Property 2 instead of Lemma 5.47. We leave it for the interested reader to work out the details.

Property 4: Using (A0) and (PAR), we have

$$\vdash (\bigvee_{m \in \mathcal{M}_{i+1}, \neg \varphi \notin m} \hat{m}) \rightarrow ((\bigvee_{m \in \mathcal{M}_{i+1}, \neg \varphi \notin m} \hat{m}) \lor (\bigvee_{m \in \mathcal{M}_i \setminus \mathcal{M}_{i+1}, \neg \varphi \notin m} \hat{m})).$$

But $(\bigvee_{m \in \mathcal{M}_{i+1}, \neg \varphi \notin m} \hat{m}) \lor (\bigvee_{m \in \mathcal{M}_i \setminus \mathcal{M}_{i+1}, \neg \varphi \notin m} \hat{m})$ is the same as $(\bigvee_{m \in \mathcal{M}_i, \neg \varphi \notin m} \hat{m})$.

Using the induction hypothesis, we know that Property 4 holds for $i$, that is

$$\vdash (\bigvee_{m \in \mathcal{M}_i, \neg \varphi \notin m} \hat{m}) \rightarrow \neg \varphi$$

and so, it follows

$$\vdash (\bigvee_{m \in \mathcal{M}_{i+1}, \neg \varphi \notin m} \hat{m}) \rightarrow \neg \varphi.$$  

For the converse, let $m_1, \ldots, m_k$ be all the molecules in $\mathcal{M}_i \setminus \mathcal{M}_{i+1}$ that do not contain $\neg \varphi$. Then, by Property 1, we know that, for every $j = 1, \ldots, k$:

$$\vdash \neg \varphi \rightarrow G_0 \neg \hat{m}_j$$

Hence, using (A1),

$$\vdash \neg \varphi \rightarrow \neg \hat{m}_j.$$
and, thus,
\[
\vdash \neg \varphi \rightarrow \bigwedge_{j=1}^{k} \neg \hat{m}_j
\]
which can be rephrased as
\[
\vdash \neg \varphi \rightarrow \bigwedge_{m \in M_i \setminus M_{i+1}, \neg \varphi \not\in m} \neg \hat{m}_j.
\]
Using the induction hypothesis again, we know that
\[
\vdash \neg \varphi \rightarrow \bigvee_{m \in M_i, \neg \varphi \not\in m} \hat{m}.
\]
And, thus, it is easy to conclude that
\[
\vdash \neg \varphi \rightarrow \bigvee_{m \in M_{i+1}, \neg \varphi \not\in m} \hat{m}.
\]

Property 5: Let \( m_1, \ldots, m_k \) be all the molecules in \( M_{i+1} \) such that \( \neg \varphi \in m_j \), for \( j = 1, \ldots, k \). For each of these molecules, using Property 3, we know that
\[
\vdash \neg \varphi \rightarrow G_0(\hat{m}_j \rightarrow G_0(\bigvee_{m' \in M_{i+1}, (m_j, m') \in R_{i+1}^*} \hat{m}')).
\]
Hence, using (A1), we also have
\[
\vdash \neg \varphi \rightarrow (\hat{m}_j \rightarrow G_0(\bigvee_{m' \in M_{i+1}, (m_j, m') \in R_{i+1}^*} \hat{m}')).
\]
And, thus, by propositional reasoning,
\[
\vdash \neg \varphi \rightarrow ((\bigvee_{j=1}^{k} \hat{m}_j) \rightarrow G_0(\bigvee_{m' \in M_{i+1}, (m_j, m') \in R_{i+1}^*} \hat{m}'))
\]
which can be rephrased as
\[
\vdash \neg \varphi \rightarrow ((\bigvee_{m \in M_{i+1}, \neg \varphi \not\in m} \hat{m}) \rightarrow G_0(\bigvee_{m' \in M_{i+1}, (m, m') \in R_{i+1}^*} \hat{m}')).
\]
Then, by Property 4, we can easily derive the desired conclusion.
We are now able to state our completeness result.

**Theorem 5.51.** Let $\varphi$ be a formula. If $\models \varphi$ then $\vdash \varphi$

**Proof.** If $\varphi$ is valid then $\neg \varphi$ is not satisfiable. This means that the algorithm in Figure 5.3 returns failure, that is, the structure $\langle M^*_{\neg \varphi}, R^*_{\neg \varphi} \rangle$ is empty. Using Property 5 in Lemma 5.50 we derive

$$\vdash \neg \varphi \leftrightarrow \text{false}$$

which implies that

$$\vdash \neg \neg \varphi$$

and, consequently,

$$\vdash \varphi.$$
Chapter 6

Computation Tree Logic

Computation Tree Logic (CTL) is a branching temporal logic for specifying properties of systems. In LTL the notion of time is linear in the sense that at each instant there is only one possible successor. In terms of transition systems, this means that we only consider a single path at each time when defining the satisfaction relation. But in a transition system a state might have different successors. Consequently, several paths may start in a given state. Hence, for a formula to hold at a state, it is required that it holds in every path starting from that state. This means that, in LTL, there is an underlying universal quantification on the paths of the transition system. However, it is not very easy to express properties that require some property to hold for some paths of the transition system.

In CTL, the temporal operators appear together with quantifiers over paths of the transition system. For instance, we will be able to write the formula $\text{EG} p$ stating that there is a path for which $p$ always holds. CTL is a branching time logic that is expressive enough for formulating a significant set of system properties. Furthermore, it is a logic for which efficient and simple model checking algorithms exist.

6.1 Computation Tree Logic

In this section, we introduce the syntax and semantics of CTL.
6.1.1 Syntax

We assume fixed a set \( \Xi \) of propositional symbols. The language of CTL (over \( \Xi \)), denoted by \( L_{ctl}^{\Xi} \), contains the formulas of the form

\[ \alpha ::= \text{true} \mid p \mid \neg \alpha \mid \alpha \land \alpha \mid AX \alpha \mid EX \alpha \mid A(\alpha U \alpha) \mid E(\alpha U \alpha). \]

where \( p \in \Xi \). The elements of \( L_{ctl}^{\Xi} \) are denoted by the letters \( \alpha, \beta, \ldots \) eventually with subscripts or superscripts.

Each operator in CTL is composed by a path quantifier (the existential path quantifier \( E \) or the universal path quantifier \( A \)) and by a temporal operator (the temporal operator \( X \) or the temporal operator \( U \)). Intuitively, formula \( AX \alpha \) holds in a state if \( \alpha \) holds in every successor state. Formula \( EX \alpha \) holds in a state if \( \alpha \) holds in some successor state. Formula \( A(\alpha_1 U \alpha_2) \) holds in a state if, for every path starting in that state, formula \( \alpha_1 U \alpha_2 \) holds, that is, there is a state along the path where formula \( \alpha_2 \) holds and, for every state in between formula \( \alpha_1 \) holds. Finally, formula \( E(\alpha_1 U \alpha_2) \) holds in a state if there is a path starting that state where formula \( \alpha_1 U \alpha_2 \) holds.

The usual Boolean operators are defined as in Chapter 1. We can also define other temporal operators, for each path quantifier. For instance:

\[ EF \alpha \equiv_{def} E(\text{true} U \alpha) \]
\[ AF \alpha \equiv_{def} A(\text{true} U \alpha) \]
\[ EG \alpha \equiv_{def} \neg (AF \neg \alpha) \]
\[ AG \alpha \equiv_{def} \neg (EF \neg \alpha) \]

Formula \( EF \alpha \) means that \( \alpha \) holds potentially, formula \( AF \alpha \) means that \( \alpha \) is inevitable, \( EG \alpha \) means that \( \alpha \) potentially holds always, formula \( AG \alpha \) means that \( \alpha \) holds everywhere (is invariant).

Example 6.1. Recall the example of mutual exclusion of two processes. The mutual exclusion property can be expressed by the CTL formula

\[ AG(\neg \text{crit}_1 \lor \neg \text{crit}_2) \]

If we want to express that each process enters the critical section infinitely often, we may write the CTL formula

\[ AG(EF \text{crit}_1) \land AG(EF \text{crit}_2) \]

\[ AG \text{green} \]
6.1. COMPUTATION TREE LOGIC

6.1.2 Semantics

CTL formulas are interpreted over transition systems.

Definition 6.2. Let $T = \langle S, A, \rightarrow, I, \Xi, L \rangle$ be a transition system without terminal states and $s \in S$. The satisfaction relation for CTL is inductively defined as follows:

- $T, s \models true$
- $T, s \models p$ if $p \in L(s)$
- $T, s \models \neg \alpha$ if $T, s \not\models \alpha$
- $T, s \models A\alpha$ if, for every $\pi \in \text{Paths}(s)$, $T, \pi[1] \models \alpha$
- $T, s \models E\alpha$ if, for some $\pi \in \text{Paths}(s)$, $T, \pi[1] \models \alpha$
- $T, s \models A(\alpha_1 U \alpha_2)$ if, for every $\pi \in \text{Paths}(s)$, there is some $j \geq 0$ such that $T, \pi[j] \models \alpha_2$ and $T, \pi[k] \models \alpha_1$ for every $0 \leq k < j$
- $T, s \models E(\alpha_1 U \alpha_2)$ if, for some $\pi \in \text{Paths}(s)$, there is some $j \geq 0$ such that $T, \pi[j] \models \alpha_2$ and $T, \pi[k] \models \alpha_1$ for every $0 \leq k < j$

The transition system $T$ satisfies the formula $\alpha$, denoted by $T \models \alpha$, if $T, s_0 \models \alpha$, for every $s_0 \in I$.

Recall that $\text{Paths}(s)$ denotes the set of all maximal path fragments that start in $s$.

Definition 6.3. Let $T = \langle S, A, \rightarrow, I, \Xi, L \rangle$ be a transition system without terminal states and $\alpha$ a CTL formula. The satisfaction set for $\alpha$, denoted by $\text{Sat}_T(\alpha)$, is defined as follows

$\text{Sat}_T(\alpha) = \{ s \in S \mid T, s \models \alpha \}$.

From the previous definition, it follows that, for a transition system $T = \langle S, A, \rightarrow, I, \Xi, L \rangle$ and a CTL formula $\alpha$,

$T \models \alpha$ if and only if $I \subseteq \text{Sat}_T(\alpha)$.

When no confusion arises, we will drop the subscript in $\text{Sat}_T(\alpha)$ and simply write $\text{Sat}(\alpha)$. 
Lemma 6.4. The satisfaction relation for the remaining connectives is as follows:

- $T, s \models \text{AG} \alpha$ if, for every $\pi \in \text{Paths}(s)$, $T, \pi[i] \models \alpha$, for every $i \geq 0$
- $T, s \models \text{EG} \alpha$ if, for some $\pi \in \text{Paths}(s)$, $T, \pi[i] \models \alpha$, for every $i \geq 0$
- $T, s \models \text{AF} \alpha$ if, for every $\pi \in \text{Paths}(s)$, $T, \pi[i] \models \alpha$, for some $i \geq 0$
- $T, s \models \text{EF} \alpha$ if, for some $\pi \in \text{Paths}(s)$, $T, \pi[i] \models \alpha$, for some $i \geq 0$.

Example 6.5. Consider the transition system $T$ depicted in Figure 6.1.

It is not very difficult to see that the formula $\text{EX} p$ holds in every state of $T$, since each state has at least one successor where $p$ holds. Hence $\text{Sat}_T(\text{EX} p) = S$.

The formula $\text{AX} p$ holds in the states $s_1$, $s_2$ and $s_3$ but it does not hold in state $s_0$, since $s_0$ has a successor state ($s_2$) for which $p$ does not hold. Hence, $\text{Sat}_T(\text{AX} p) = \{s_1, s_2, s_3\}$.

Consider now the formula $\text{EG} p$. If we look at state $s_0$ we can observe that there is at least one path $s_0 s_1 s_0 s_1 \ldots$ where $p$ holds in every state. Hence, we can conclude that $\text{EG} p$ holds in $s_0$. However, $\text{EG} p$ does not hold in $s_2$ because, in this case, we need to consider every path starting at $s_2$ and $p$ does not hold at $s_2$. We leave as an exercise to check that the formula also holds at the states $s_1$ and $s_3$.

The formula $\text{AG} p$ holds at state $s_3$ since the only path from $s_3$ is $(s_3)^\omega$ and $p$ holds at $s_3$. However, it does not hold at any of the other states.
because there is always a path departing from any of those states that passes through \( s_2 \), where \( p \) does not hold.

The formula \( E(p \lor q) \) holds at states \( s_1 \) and \( s_2 \) because \( q \) also holds at those states. Furthermore, it also holds at \( s_0 \) because for path \( s_0 s_1 \ldots, p \) holds at \( s_0 \) and \( q \) holds at \( s_1 \) and, consequently, \( p \lor q \) holds at \( s_0 \) along this path. However, \( E(p \lor q) \) does not hold at \( s_3 \) because the only path departing from \( s_3 \) is \((s_3)\omega\) and \( q \) does not hold at \( s_3 \).

We leave it as an exercise to check that \( A(p \lor q) \) holds in exactly the same states as \( E(p \lor q) \).

Finally, we consider the formula \( EF(EGp) \). From what was said above, \( EGp \) holds at \( s_0, s_1 \) and \( s_3 \). Hence \( EF(EGp) \) holds at any state that can reach \( s_0, s_1 \) and \( s_3 \), that is, the formula holds in every state.

**Example 6.6.** Consider the formula \( AG AF p \) and let \( T \) be a transition system without terminal states and \( s \) one of its states. We are going to show that

\[
T, s \models AG AF p \quad \text{if and only if} \quad T, \pi[i] \models p, \text{ for every } \pi \in Paths(s), \text{ and for infinitely many values of } i.
\]

\((-\rightarrow)\): Assume that \( T, s \models AG AF p \) and let \( \pi \in Paths(s) \). Then, by Lemma 6.4, we know that, in particular for \( \pi \),

\[
T, \pi[i] \models AF p
\]

for every \( i \geq 0 \). Again by Lemma 6.4, we know that for every \( \pi' \in Paths(\pi[i]) \), there is \( j \geq 0 \) such that

\[
T, \pi'[j] \models p.
\]

In particular, we know that \( \pi[i..] \in Paths(\pi[i]) \) and that \( \pi[i..][j] = \pi[i + j] \). Hence we can conclude that for each \( i \geq 0 \) there some \( j \geq 0 \) such that

\[
T, \pi[i + j] \models p.
\]

Consequently, it follows that, for every \( \pi Paths(s) \),

\[
T, \pi[i] \models p
\]

for infinitely many values of \( i \).
Let $s$ be a state such that, for every $\pi \in \text{Paths}(s)$, $T, \pi[i] \models p$ for infinitely many values of $i$. Let $\pi = s_0 s_1 \cdots \in \text{Paths}[s]$ and $i \geq 0$. Now, let $\pi' = s'_i s'_{i+1} \cdots \in \text{Paths}(\pi[i])$ be an arbitrary path. Observe that $s'_i = s_i$ and, consequently, the concatenation of $\pi[..i] = s_0 s_1 \cdots s_i$ with $\pi'[1..] = s'_{i+1} s'_{i+2} \cdots$ is a path $\pi'' = s_0 s_1 \cdots s_i s'_{i+1} s'_{i+2} \cdots$ in $\text{Paths}(s)$. Hence, by assumption, there is $j \geq i$ such that $T, \pi''[j] \models p$.

As $\pi''[j] = \pi'[j - i]$ then, it follows that

$T, \pi'[j - i] \models p$

for some $j - i \geq 0$. And as $\pi'$ was an arbitrary path in $\text{Paths}(\pi[i])$, then

$T, \pi'[i] \models \text{AF} p$.

From this last condition it is straightforward to conclude that

$T, s \models \text{AGAF} p$

since both $\pi \in \text{Paths}(s)$ and $i \geq 0$ were arbitrary.

Recall the weak version $W$ of the LTL operator $U$. In CTL we can also define two operators for the the path quantifiers as follows:

$E(\alpha W \alpha_2) \equiv \neg A((\alpha \land \neg \alpha_2) U (\neg \alpha_1 \land \neg \alpha_2))$

$A(\alpha W \alpha_2) \equiv \neg E((\alpha \land \neg \alpha_2) U (\neg \alpha_1 \land \neg \alpha_2))$

**Lemma 6.7.** Let $T$ be a transition system without terminal states and $s$ one of its states. Then

- $T, s \models A(\alpha_1 W \alpha_2)$ if, for every path $\pi \in \text{Paths}(s)$, either $T, \pi[i] \models \alpha_1$, for every $i \geq 0$, or there is $j \geq 0$ such that $T, \pi[j] \models \alpha_2$ and $T, \pi[k] \models \alpha_1$, for every $0 \leq k < j$.

- $T, s \models E(\alpha_1 W \alpha_2)$ if, there is a path $\pi \in \text{Paths}(s)$ such that either $T, \pi[i] \models \alpha_1$, for every $i \geq 0$, or there is $j \geq 0$ such that $T, \pi[j] \models \alpha_2$ and $T, \pi[k] \models \alpha_1$, for every $0 \leq k < j$. 

6.1. COMPUTATION TREE LOGIC

We look into the semantics of negation. It is clear from the definition of the satisfaction relation that \( T, s \models \neg \alpha \) if and only if \( T, s \not\models \alpha \), for any CTL formula \( \alpha \). However, in the case of transition systems this is not the case. Indeed, it is possible to find a transition system \( T \) and a formula \( \alpha \) such that \( T \not\models \alpha \) and \( T \not\models \neg \alpha \) both hold. An easy example is to consider a transition system with two initial states \( s_0 \) and \( s'_0 \) such that \( T, s_0 \models \alpha \) and \( T, s'_0 \not\models \alpha \). Since, for a transition to satisfy a formula, we require that formula to hold in every initial state then, \( T \not\models \alpha \) and \( T \not\models \neg \alpha \).

**Definition 6.8.** Two CTL formulas (over \( \Xi \)) \( \alpha_1 \) and \( \alpha_2 \) are said to be equivalent, denoted by \( \alpha_1 \equiv \alpha_2 \), if \( \text{Sat}_T(\alpha_1) = \text{Sat}_T(\alpha_2) \), for every transition system \( T \) over \( \Xi \).

In particular, it follows that \( \alpha_1 \equiv \alpha_2 \) if and only if
\[
T \models \alpha_1 \text{ if and only if } T \models \alpha_2.
\]

**Lemma 6.9.** The following hold:

- \( AX \alpha \equiv \neg EX \neg \alpha \)
- \( EX \alpha \equiv \neg AX \neg \alpha \)
- \( AF \alpha \equiv \neg EG \neg \alpha \)
- \( EF \alpha \equiv \neg AG \neg \alpha \)
- \( A(\alpha_1 U \alpha_2) \equiv \neg E(\neg \alpha_2 U (\neg \alpha_1 \land \neg \alpha_2)) \land \neg EG \neg \alpha_2 \)
- \( A(\alpha_1 U \alpha_2) \equiv \alpha_2 \lor (\alpha_1 \land AX A(\alpha_1 U \alpha_2)) \)
- \( E(\alpha_1 U \alpha_2) \equiv \alpha_2 \lor (\alpha_1 \land EX E(\alpha_1 U \alpha_2)) \)
- \( AF \alpha \equiv \alpha \lor AX AF \alpha \)
- \( EF \alpha \equiv \alpha \lor EX EF \alpha \)
- \( AG \alpha \equiv \alpha \land AX AG \alpha \)
- \( EG \alpha \equiv \alpha \land EX EG \alpha \)
- \( AG(\alpha_1 \land \alpha_2) \equiv AG \alpha_1 \land AG \alpha_2 \)
• $\text{EF}(\alpha_1 \lor \alpha_2) \equiv \text{EF} \alpha_1 \lor \text{EF} \alpha_2$

• $\text{E}(\alpha_1 \text{ W} \alpha_2) \equiv \text{E}(\alpha_1 \text{ U} \alpha_2) \lor \text{EG} \alpha_1$

However, the translation of certain LTL properties into CTL does not hold. For instance, the following equivalences do not hold in general:

• $\text{AF}(\alpha_1 \lor \alpha_2) \not\equiv \text{AF} \alpha_1 \lor \text{AF} \alpha_2$

• $\text{A}(\alpha_1 \text{ W} \alpha_2) \not\equiv \text{A}(\alpha_1 \text{ U} \alpha_2) \lor \text{AG} \alpha_1$.

### 6.1.3 Existential Normal Form

Like in the case of LTL, any CTL formula can be transformed into a canonical form. We start by considering the existential normal form (ENF). A formula in ENF can only contain existential path quantifiers.

**Definition 6.10.** The set of CTL formulas in the existential normal form is defined by

$$\alpha := \text{true} | p | \neg \alpha | \alpha \land \alpha | \text{EX} \alpha | \text{E} (\alpha \text{ U} \alpha) | (\text{EG} \alpha).$$

As expected, any CTL formula can be transformed into a formula in ENF.

**Theorem 6.11.** Let $\alpha$ be a CTL formula. Then, there exists a CTL formula $\overline{\alpha}$ in ENF such that $\alpha \equiv \overline{\alpha}$.

**Proof.** The proof is similar to the proof of Proposition 1.3. Consider the conversion function

- $\text{conv}(\text{AX} \alpha) = \neg \text{EX} \neg (\text{conv}(\alpha))$

- $\text{conv}(\text{A}(\alpha_1 \text{ U} \alpha_2)) = \neg \text{E}(\neg \text{conv}(\alpha_2) \text{ U} (\neg \text{conv}(\alpha_1) \land \neg \text{conv}(\alpha_2))) \land \neg \text{EG} \neg \text{conv}(\alpha_2)$.

Another important normal form in CTL is the positive normal form (PNF). As in the other cases, a formula is PNF if the negation occurs only applied to a propositional symbol. In this case, we need to consider $\lor$ and $\text{W}$ as primitives in the language.
**Definition 6.12.** The set of CTL formulas in positive normal form (PNF) is defined by

\[
\alpha ::= \text{true} | \text{false} | p | \neg p | \alpha \land \alpha | \alpha \lor \alpha | \text{AX} \alpha | \text{EX} \alpha | \\
\text{A}(\alpha \cup \alpha) | \text{E}(\alpha \cup \alpha) | \text{A}(\alpha \text{W} \alpha) | \text{E}(\alpha \text{W} \alpha)
\]

**Theorem 6.13.** Let \(\alpha\) be a CTL formula. Then, there exists a CTL formula \(\bar{\alpha}\) in PNF such that \(\alpha \equiv \bar{\alpha}\).

*Proof.* In this case, we consider the conversion function, were we detail only the conversion of the negation,

- \(\text{conv}(\neg \text{true}) = \text{false}\)
- \(\text{conv}(\neg p) = \neg p\)
- \(\text{conv}(\neg \neg \alpha) = \alpha\)
- \(\text{conv}(\neg (\alpha_1 \land \alpha_2)) = \neg \text{conv}(\alpha_1) \lor \neg \text{conv}(\alpha_2)\)
- \(\text{conv}(\neg \text{AX} \alpha) = \text{EX} \neg \text{conv}(\alpha)\)
- \(\text{conv}(\neg \text{EX} \alpha) = \text{AX} \neg \text{conv}(\alpha)\)
- \(\text{conv}(\text{A}(\alpha_1 \cup \alpha_2)) = \text{E}(\text{conv}(\alpha_1) \land \neg \text{conv}(\alpha_2)) \lor \neg \text{conv}(\alpha_1) \lor \neg \text{conv}(\alpha_2))\)
- \(\text{conv}(\text{E}(\alpha_1 \cup \alpha_2)) = \text{A}(\text{conv}(\alpha_1) \land \neg \text{conv}(\alpha_2)) \lor \neg \text{conv}(\alpha_1) \lor \neg \text{conv}(\alpha_2))\).

Then, we just choose \(\bar{\alpha}\) to be \(\text{conv}(\alpha)\). \(\square\)

### 6.1.4 CLT versus LTL

We now proceed to show that LTL and CTL are incomparable according to their expressive power.

**Definition 6.14.** Let \(\varphi\) be an LTL formula and \(\alpha\) be a CTL formula (both over \(\Xi\)). Then, \(\varphi\) and \(\alpha\) are said to be equivalent, denoted by \(\varphi \equiv \alpha\), if

\[
T \models \varphi \text{ if and only if } T \models \alpha
\]

for every transition system \(T\) over \(\Xi\).
Example 6.15. The LTL formula $G F p$ and the CTL formula $AF AG p$ are equivalent.

Consider the LTL formula $F p$. This formula holds in a state $s$ if all paths departing from $s$ satisfy $F p$. Hence, this LTL formula will be equivalent to the CTL formula $AF p$. The following result (that we state without proof) states that we can obtain an LTL formula from a CTL formula by dropping all path quantifiers, and these two formulas are equivalent, if the CTL formula admits an equivalent LTL formula.

Theorem 6.16. Let $\alpha$ be a CTL formula and let $\varphi$ be the LTL formula obtained from $\alpha$ by omitting all path quantifiers. Then,

$\varphi \equiv \alpha$ or there is no LTL formula that is equivalent to $\alpha$.

Lemma 6.17. The LTL formula $F G p$ and the CTL formula $AF AG p$ are not equivalent.

Proof. Consider the transition system depicted in Figure 6.2. The paths of the transition system are $\pi = s_0^\omega$ and $\pi_n = s_0^n s_1 s_2^\omega$, for $n \geq 0$, and their corresponding traces are $\text{trace}(\pi) = \{p\}^\omega$ and $\text{trace}(\pi_n) = \{p\}^n \emptyset \{p\}^\omega$, for $n \geq 0$.

In the case of the LTL formula, we have that $\text{Mod}(F_0 G_0 p)$ is the set of all interpretation structures such that there is $i \geq 0$ such that $p \in \sigma[j]$, for every $j \geq i$. Then, it is not very difficult to see that $\text{trace}(\pi) \in \text{Mod}(F_0 G_0 p)$, given that $p \in \text{trace}(\pi)[j]$ for any $j \geq 0$, and also that $\text{trace}(\pi_n) \in \text{Mod}(F_0 G_0 p)$, given that $p \in \text{trace}(\pi)[j]$ for any $j \geq n + 2$. Hence, $\text{Traces}(T) \subseteq \text{Mod}(F_0 G_0 p)$ and, consequently, $T \models F_0 G_0 p$.

Consider now the CTL formula. In this case, observe that $p \notin L(\pi_n [n + 1]) = L(s_1) = \emptyset$, for each $n \geq 0$. Consequently, it follows that $T, \pi_n[i] \not\models p$,
for some \( i \geq 0 \). And, as \( \pi_n \in \text{Paths}(s_0) \), it follows that \( T, s_0 \not\vDash AGp \). This implies that \( T, \pi[i] \not\vDash AGp \), for every \( i \geq 0 \). Hence, we can conclude that \( T, s_0 \not\vDash AF AGp \) which implies that \( T \not\vDash AF AGp \).

Consequently, we can conclude that \( F \circ G \circ p \not\equiv AF AGp \).

This lemma shows that the CTL formula \( AF AGp \) has no equivalent LTL formula. Indeed, observe that the LTL formula \( F \circ G \circ p \) was obtained from \( AF AGp \) by dropping all the path quantifiers. As these formulas are not equivalent, as we have just shown, then, by Theorem 6.16 we can conclude that \( AF AGp \) has no equivalent LTL formula.

**Theorem 6.18.**

1. There exists an LTL formula for which no equivalent CTL formula exists.

2. There exists a CTL formula for which no equivalent LTL formula exists.

**Proof.** 1. Consider the LTL formula \( F_0 G_0 p \). We are going to show that there is no CTL formula that is equivalent to \( F_0 G_0 p \). To this end, consider the following sequences of transition systems \( T_0, T_1, T_2, \ldots \) and \( T'_0, T'_1, T'_2, \ldots \) such that, for every \( n \geq 0 \),

\[
T_n = \langle S_n, \{ \tau \}, \rightarrow_n, I_n, \{ p \}, L_n \rangle \quad \text{and} \quad T'_n = \langle S'_n, \{ \tau \}, \rightarrow'_n, I'_n, \{ p \}, L'_n \rangle
\]

where

- \( S_0 = \{ s_0, t_0 \} \) and \( S'_0 = \{ s'_0, t'_0 \} \);
- \( S_n = S'_{n-1} \cup \{ s_n, t_n \} \) and \( S'_n = S'_{n-1} \cup \{ s'_n, t'_n \} \), for \( n > 0 \);
- \( I_n = \{ s_n \} \) and \( I'_n = \{ s'_n \} \), for \( n \geq 0 \);
- \( L_n(s_i) = L'_n(s'_i) = \emptyset \) and \( L_n(t_i) = L'_n(t'_i) = \{ p \} \), for \( n \geq 0 \) and \( 0 \leq i \leq n \);
- \( \rightarrow_0 \) is such that \( s_0 \rightarrow_0 t_0, t_0 \rightarrow_0 s_0 \) and \( t_0 \rightarrow_0 t_0 \);
- \( \rightarrow'_0 \) is such that \( s_0 \rightarrow'_0 t_0 \) and \( t_0 \rightarrow'_0 t_0 \);
• $\rightarrow_n$ is such that $s_n \rightarrow_n t_n$, $t_n \rightarrow_n s_n$, $t_n \rightarrow_n t_n$, $t_n \rightarrow_n s'_{n-1}$ and contains $\rightarrow'_{n-1}$, for $n > 0$;

• $\rightarrow'_{n}$ is such that $s'_n \rightarrow'_{n} t'_n$, $t'_n \rightarrow'_{n} t'_n$, $t'_n \rightarrow'_{n} s'_{n-1}$ and contains $\rightarrow'_{n-1}$, for $n > 0$;

Observe that $T_n$ is built from $T_{n-1}$ and not from $T_{n-1}$. Hence, for each $n \geq 0$, the only difference between $T_n$ and $T'_n$ is the additional transition $t_n \rightarrow_n s_n$ in $T_n$.

We start by showing that, for every $n \geq 0$, we have that $T_n \not\models F_0 G_0 p$. This is easy to establish because $T_n$ contains the initial path $(s_n t_n)^\omega$. The trace of this path is $\sigma = \emptyset \{p\} \emptyset \{p\} \ldots$ and, clearly, $\sigma \not\models F_0 G_0 p$. Hence, $\sigma \in \text{Traces}(T_n)$ and $\sigma \not\in \text{Mod}(F_0 G_0 p)$ and, thus, $T_n \not\models F_0 G_0 p$.

Next, we show that $T'_n \models F_0 G_0 p$, for every $n \geq 0$. The paths in $T'_n$ are of the form

$$\pi_i = s'_n (t'_n)^{k_n} s'_{n-1} (t'_{n-1})^{k_{n-1}} \ldots s'_i (t'_i)^{k_i}$$

for some $0 \leq i \leq n$ and $k_n, \ldots, k_{i+1} \in \mathbb{N}$. The trace of each of these paths is

$$\text{trace}(\pi_i) = \emptyset (\{p\})^{k_n} \emptyset (\{p\})^{k_{n-1}} \ldots \emptyset (\{p\})^{k_i}.$$

This trace clearly satisfies satisfies $F_0 G_0 p$. Indeed, $\text{trace}(\pi_i), k \models p$, for every $k \geq k_n + k_{n-1} + \ldots + k_{i+1} + (n - i)$. Hence, $\text{Traces}(T'_n) \subseteq \text{Mod}(F_0 G_0 p)$ and, thus, $T'_n \models F_0 G_0 p$, for every $n \geq 0$.

We state without prove that $T_n$ and $T'_n$ cannot be distinguished by any CTL formula of length less or equal to $n$. That is, for every $n \geq 0$,

$$T_n \models \alpha \text{ if and only if } T'_n \models \alpha, \text{ for every CTL formula } \alpha \text{ such that } |\alpha| \leq n.$$

We can now conclude the desired result. Assume that there is a CTL formula $\alpha$ equivalent to $F_0 G_0 p$ and let $n = |\alpha|$. Then, we know that $T_n \not\models F_0 G_0 p$ and as we are assuming that $F_0 G_0 p \equiv \alpha$ then it follows that $T_n \not\models \alpha$.

On the other hand, we also know that $T'_n \models F_0 G_0 p$ and once again, we can conclude that $T'_n \models \alpha$. Hence, we have that $T_n \not\models \alpha$ and $T'_n \models \alpha$ which is clearly impossible because $|\alpha| = n$ and so $\alpha$ cannot distinguish $T_n$ and $T'_n$.

2. In this case, as already stated, the proof is an immediate consequence of Lemma 6.17 and of Theorem 6.16. □
6.2 Model checking

We will want to check whether $T \models \alpha$ where $\alpha$ is a CTL formula. The idea is to calculate the set $\text{Sat}_T(\alpha)$ (recursively) and the check whether all initial states are in $\text{Sat}_T(\alpha)$.

Henceforth, we consider formulas to be in ENF, that is, the only temporal modalities are $\text{EX}$, $\text{E U}$ and $\text{EG}$. We need to generate the sets $\text{Sat}(\text{EX} \alpha)$, $\text{Sat}(\text{E} (\alpha \text{ U} \alpha'))$ and $\text{Sat}(\text{EG} \alpha)$ from the sets $\text{Sat}(\alpha)$ and $\text{Sat}(\alpha')$.

Let $\alpha$ be a CTL formula. Then, $\text{Sub}(\alpha)$ denotes the set of subformulas of $\alpha$ and $\text{GSub}(\alpha)$ denotes the set of maximal genuine subformulas of $\alpha$. These sets can be defined on the structure of the formula $\alpha$. The set $\text{Sub}(\alpha)$ is defined as follows:

- $\text{Sub}(\text{true}) = \{\text{true}\}$
- $\text{Sub}(p) = \{p\}$
- $\text{Sub}(\neg \alpha) = \{-\alpha\} \cup \text{Sub}(\alpha)$
- $\text{Sub}(\alpha_1 \land \alpha_2) = \{\alpha_1 \land \alpha_2\} \cup \text{Sub}(\alpha_1) \cup \text{Sub}(\alpha_2)$
- $\text{Sub}(\text{EX} \alpha) = \{\text{EX} \alpha\} \cup \text{Sub}(\alpha)$
- $\text{Sub}(\text{E} (\alpha_1 \text{ U} \alpha_2)) = \{\text{E} (\alpha_1 \text{ U} \alpha_2)\} \cup \text{Sub}(\alpha_1) \cup \text{Sub}(\alpha_2)$
- $\text{Sub}(\text{EG} \alpha) = \{\text{EG} \alpha\} \cup \text{Sub}(\alpha)$.

The set $\text{GSub}(\alpha)$ is defined as follows:

- $\text{GSub}(\text{true}) = \{\}$
- $\text{GSub}(p) = \{\}$
- $\text{GSub}(\neg \alpha) = \{\alpha\}$
- $\text{GSub}(\alpha_1 \land \alpha_2) = \{\alpha_1, \alpha_2\}$
- $\text{GSub}(\text{EX} \alpha) = \{\alpha\}$
- $\text{GSub}(\text{E} (\alpha_1 \text{ U} \alpha_2)) = \{\alpha_1, \alpha_2\}$
- $\text{GSub}(\text{EG} \alpha) = \{\alpha\}$. 
Input: transition system $T$ without terminal states and CTL formula $\alpha$

Output: “yes” if $T \models \alpha$, “no” otherwise

for all $\beta \in GSub(\alpha)$ do
  compute $\text{Sat}(\alpha)$ from $\text{Sat}(\beta)$;
end
if $I \subseteq \text{Sat}(\alpha)$ then
  return “yes”
else
  return “no”
fi

Figure 6.3: CTL model checking (general structure).

The following result gives us an inductive characterization of $\text{Sat}_T(\alpha)$, for the CTL formula $\alpha$.

**Theorem 6.19.** Let $T = \langle S, A, \rightarrow, I, L \rangle$ be a transition system without terminal states. Let $\alpha$, $\alpha_1$ and $\alpha_2$ be CTL formulas over $\Xi$. Then, the following hold:

(i) $\text{Sat}(\text{true}) = S$

(ii) $\text{Sat}(p) = \{s \in S \mid p \in L(S)\}$

(iii) $\text{Sat}(\neg \alpha) = S \setminus \text{Sat}(\alpha)$

(iv) $\text{Sat}(\alpha_1 \land \alpha_2) = \text{Sat}(\alpha_1) \cap \text{Sat}(\alpha_2)$

(v) $\text{Sat}(\text{EX} \alpha) = \{s \in S \mid \text{Suc}(s) \cap \text{Sat}(\alpha) \neq \emptyset\}$

(vi) $\text{Sat}(\text{E}(\alpha_1 \lor \alpha_2))$ is the smallest subset $C$ of $S$ such that
    - $\text{Sat}(\alpha_2) \subseteq C$
    - $s \in \text{Sat}(\alpha_1)$ and $\text{Suc}(s) \cap C \neq \emptyset$ implies that $s \in C$

(vii) $\text{Sat}(\text{EG} \alpha)$ is the largest subset $C$ of $S$ such that
    - $C \subseteq \text{Sat}(\alpha)$
(b) $s \in C$ implies that $\text{Suc}(s) \cap C \neq \emptyset$.

Proof. The proof follows by induction on the structure of the formula. Cases (i) and (ii) are straightforward from the definition of $\text{Sat}(\alpha)$ and of satisfaction relation.

(iii) 

\[
\text{Sat}(\neg \alpha) = \{ s \in S \mid T, s \notmodels \neg \alpha \} \\
= \{ s \in S \mid T, s \models \alpha \} \\
= \{ s \in S \mid s \not\in \text{Sat}(\alpha) \} \\
= S \setminus \text{Sat}(\alpha).
\]

(iv) 

\[
\text{Sat}(\alpha_1 \land \alpha_2) = \{ s \in S \mid T, s \models \alpha_1 \land \alpha_2 \} \\
= \{ s \in S \mid T, s \models \alpha_1 \text{ and } T, s \models \alpha_2 \} \\
= \{ s \in S \mid s \in \text{Sat}(\alpha_1) \text{ and } \text{Sat}(\alpha_2) \} \\
= \text{Sat}(\alpha_1) \cap \text{Sat}(\alpha_2).
\]

(v) 

\[
\text{Sat}(\text{EX} \alpha) = \{ s \in S \mid T, s \models \text{EX} \alpha \} \\
= \{ s \in S \mid T, s' \models \alpha \text{ for some } s' \in \text{Suc}(s) \} \\
= \{ s \in S \mid s' \in \text{Sat}(\alpha) \text{ for some } s' \in \text{Suc}(s) \} \\
= \{ s \in S \mid \text{Sat}(\alpha) \cap \text{Suc}(s) \neq \emptyset \}.
\]

(vi) We start by showing that the set $\text{Sat}(\text{E}(\alpha_1 \cup \alpha_2))$ satisfies conditions (a) and (b). Recall the equivalence 

\[
\text{E}(\alpha_1 \cup \alpha_2) \equiv \alpha_2 \lor (\alpha_1 \land \text{EX}(\text{E}(\alpha_1 \cup \alpha_2))).
\]

If $s \in \text{Sat}(\alpha_2)$ then $T, s \models \alpha_2$ which implies that $T, s \models \alpha_2 \lor (\alpha_1 \land \text{EX}(\text{E}(\alpha_1 \cup \alpha_2)))$ and, consequently, by the equivalence above, it follows that $s \in \text{Sat}(\text{E}(\alpha_1 \cup \alpha_2))$. Hence $\text{Sat}(\alpha_2) \subseteq \text{Sat}(\text{E}(\alpha_1 \cup \alpha_2))$, that is, $\text{Sat}(\text{E}(\alpha_1 \cup \alpha_2))$ satisfies condition (a).

For condition (b), assume that $s \in \text{Sat}(\alpha_1)$ and $\text{Suc}(s) \cap \text{Sat}(\text{E}(\alpha_1 \cup \alpha_2)) \neq \emptyset$. Then, if follows that $T, s \models \alpha_1$ and there is is $s' \in \text{Suc}(s)$ such that $T, s' \models \text{E}(\alpha_1 \cup \alpha_2)$. Hence, $T, s \models \text{EX}(\text{E}(\alpha_1 \cup \alpha_2))$ and, thus, $T, s \models \alpha_1 \land \text{EX}(\text{E}(\alpha_1 \cup \alpha_2))$. As in the previous case, we immediately conclude that $s \in \text{Sat}(\text{E}(\alpha_1 \cup \alpha_2))$ and so $\text{Sat}(\text{E}(\alpha_1 \cup \alpha_2))$ also fulfills condition (b).
Next, we prove that any set \( C \) satisfying conditions (a) and (b) necessarily contains \( \text{Sat}(E(\alpha_1 U \alpha_2)) \). Let \( s \in \text{Sat}(E(\alpha_1 U \alpha_2)) \) that is \( T, s \models E(\alpha_1 U \alpha_2) \). Then, there is \( \pi \in \text{Paths}(s) \) and \( n \geq 0 \) such that \( T, \pi[n] \models \alpha_2 \) and \( T, \pi[k] \models \alpha_1 \), for every \( 0 \leq k < n \). As \( \pi[n] \in \text{Sat}(\alpha_2) \) then, by condition (a), it follows that \( \pi[n] \in C \). Next, we prove by induction on \( n \) that if \( \pi[n] \in C \) then \( \pi[k] \in C \) for every \( 0 \leq k < n \).

**Base of induction:** For \( n = 0 \) the result follows immediately.

**Induction step:** Assume that \( n > 0 \). As was proved before, we already know that \( \pi[n] \in C \). Additionally, we also know that \( \pi[n] \in \text{Suc}(\pi[n-1]) \). Hence, \( \text{Suc}(\pi[n-1]) \cap C \neq \emptyset \). Furthermore, we also know that \( T, \pi[n-1] \models \alpha_1 \). Hence, by condition (b), it follows that \( \pi[n-1] \in C \). Hence, by induction hypothesis, it follows that \( \pi[k] \in C \) for every \( 0 \leq k < n-1 \). As \( \pi[n-1] \in C \) also holds, we conclude that \( \pi[k] \in C \) for every \( 0 \leq k < n \). In particular, we conclude that \( s = \pi[0] \in C \) and, thus, \( \text{Sat}(E(\alpha_1 U \alpha_2)) \subseteq C \).

(vii) The proof that \( \text{Sat}(E\alpha) \) satisfies conditions (a) and (b) is left as an exercise, using the equivalence

\[ E\alpha \equiv \alpha \land E\neg \alpha. \]

We prove that any set \( C \) satisfying conditions (a) and (b) is contained in \( \text{Sat}(E\alpha) \). Let \( s \in C \) and consider the path \( \pi = s_0 s_1 s_2 \ldots \) such that \( s_0 = s \) and \( s_i \in C \), for every \( i \in \mathbb{N} \). We need to show that such path does indeed exist. To this end, consider the succession \( \pi[..0], \pi[..1], \pi[..2] \ldots \) of finite prefixes of \( \pi \). We prove by induction on \( n \) that each of these prefixes is well defined.

**Base of induction:** For \( n = 0 \), \( \pi[..0] = s_0 = s \) and, by hypothesis, \( s \in C \).

**Induction step:** Assume that \( n > 0 \) and assume that \( \pi[..n-1] \) is well defined. Hence, \( \pi[n-1] \in C \). By condition (b) we know that \( \text{Suc}(\pi[n-1]) \cap C \neq \emptyset \). Hence, it follows that there is \( s_n \in \text{Suc}(\pi[n-1]) \cap C \) and consequently, \( \pi[..n] \) is also well defined.

The succession \( \pi[..0], \pi[..1], \pi[..2] \ldots \) induces a path \( \pi \in \text{Paths}(s) \) such that \( \pi[i] \in C \) for every \( i \in \mathbb{N} \). From condition (a) we know that \( C \subseteq \text{Sat}(\alpha) \). Hence, \( T, \pi[i] \models \alpha \), for every \( i \in \mathbb{N} \), which implies that \( T, s \models E\alpha \), that is, \( s \in \text{Sat}(E\alpha) \). Hence, we conclude that \( C \subseteq \text{Sat}(\alpha) \). \( \square \)

The characterization of the sets \( \text{Sat}(E(\alpha_1 U \alpha_2)) \) and \( \text{Sat}(E\alpha) \) are based on the fixed-point nature of the operators that is given by their expansion
laws.

Let us start by the set $\text{Sat}(E(\alpha_1 U \alpha_2))$. In this case, we know that the following equivalence holds

$$E(\alpha_1 U \alpha_2) \equiv \alpha_2 \lor (\alpha_1 \land \text{EX}(E(\alpha_1 U \alpha_2)))$$

We can see the CTL formula $E(\alpha_1 U \alpha_2)$ as a fixed point of the equation

$$x \equiv \alpha_2 \lor (\alpha_1 \land \text{EX}x)$$

We know that $E(\alpha_1 U \alpha_2)$ is in fact a solution of this equation, but is not necessarily the only solution. There may be other solutions, not equivalent to $E(\alpha_1 U \alpha_2)$. And in fact, there are other solutions. For instance, $E(\alpha_1 W \alpha_2)$ is also a solution for this equation. We can obtain a unique characterization of $E(\alpha_1 U \alpha_2)$ by the fact that this formula is the least solution of

$$x \equiv \alpha_2 \lor (\alpha_1 \land \text{EX}x)$$

We can defined the operator $G_\alpha : 2^S \rightarrow 2^S$ such that

$$G_\alpha(C) = \text{Sat}(\alpha) \cap \{s \in \text{Sat}(\alpha) \mid \text{Suc}(s) \cap C \neq \emptyset\}.$$
CHAPTER 6. COMPUTATION TREE LOGIC

Input: transition system $T$ without terminal states and CTL formula $\alpha$
Output: $Sat(\alpha) = \{s \in S \mid s \models \alpha\}$

\begin{verbatim}
case $\alpha$ of
  true                  : return $S$;
  $p$                   : return $\{s \in S \mid p \in L(s)\}$;
  $\neg \alpha_1$       : return $S \setminus Sat(\alpha_1)$;
  $\alpha_1 \land \alpha_2$ : return $Sat(\alpha_1) \cap Sat(\alpha_2)$;
  EX$\alpha_1$          : return $\{s \in S \mid Suc(s) \cap Sat(\alpha_1) \neq \emptyset\}$;
  E($\alpha_1$ U $\alpha_2$) : $C = Sat(\alpha_2)$;
      while $\{s \in Sat(\alpha_1) \setminus C \mid Suc(s) \cap C \neq \emptyset\} \neq \emptyset$ do
          let $s \in \{s \in Sat(\alpha_1) \setminus C \mid Suc(s) \cap C \neq \emptyset\}$;
          $C := C \cup \{s\}$;
      od
    return $C$;
  EG$\alpha_1$          : $C = Sat(\alpha_1)$;
      while $\{s \in C \mid Suc(s) \cap C = \emptyset\} \neq \emptyset$ do
          let $s \in \{s \in C \mid Suc(s) \cap C = \emptyset\}$;
          $C := C \setminus \{s\}$;
      od
    return $C$
end
\end{verbatim}

Figure 6.4: Algorithm for computing the sets $Sat$. 

6.2. MODEL CHECKING

The algorithm depicted in Figure 6.4 relies on the properties expressed in Theorem 6.19 for constructing the \( \text{Sat} \) sets. Recall that we are assuming that the formula to be checked in is ENF.

Let us come back to the computation of the set \( \text{Sat}(E(\alpha_1 \cup \alpha_2)) \). We start by observing that the operator \( U_{\alpha_1,\alpha_2} \) (over the complete lattice \( (2^S, \subseteq) \)) is continuous. Then, using Kleene's fixed point Theorem, we know that the supremum of the ascending chain

\[
\emptyset \subseteq U_{\alpha_1,\alpha_2}(\emptyset) \subseteq U_{\alpha_1,\alpha_2}(U_{\alpha_1,\alpha_2}(\emptyset)) \subseteq \ldots
\]

is the least fixed point of \( U_{\alpha_1,\alpha_2} \), that is, \( \text{Sat}(E(\alpha_1 \cup \alpha_2)) \). This means that

\[
\text{Sat}(E(\alpha_1 \cup \alpha_2)) = \bigcup_{i \geq 0} U_{\alpha_1,\alpha_2}^i(\emptyset).
\]

We proceed to analyze this chain of sets in detail. Let

\[
C_i = U_{\alpha_1,\alpha_2}^i(\emptyset), \text{ for every } i \geq 0.
\]

Then, \( C_0 = \emptyset \), \( C_1 = U_{\alpha_1,\alpha_2}(C_0) = \text{Sat}(\alpha_2) \),

\[
C_2 = U_{\alpha_1,\alpha_2}(C_1)
= C_1 \cup \{s \in \text{Sat}(\alpha_1) \mid \text{Suc}(s) \cap C_1 \neq \emptyset\}
\]

\[
C_3 = U_{\alpha_1,\alpha_2}(C_2)
= \text{Sat}(\alpha_2) \cup \{s \in \text{Sat}(\alpha_1) \mid \text{Suc}(s) \cap C_2 \neq \emptyset\}
= C_1 \cup \{s \in \text{Sat}(\alpha_1) \mid \text{Suc}(s) \cap C_1 \neq \emptyset\} \cup
\{s \in \text{Sat}(\alpha_1) \mid \text{Suc}(s) \cap C_2 \neq \emptyset\}
= C_2 \cup \{s \in \text{Sat}(\alpha_1) \mid \text{Suc}(s) \cap C_2 \neq \emptyset\}.
\]

In this last equation, we used the fact that \( C_1 \subseteq C_2 \). In general, we have for every \( i \geq 0 \)

\[
C_{i+1} = C_i \cup \{s \in \text{Sat}(\alpha_1) \mid \text{Suc}(s) \cap C_i \neq \emptyset\}.
\]

We can see each set \( C_{i+1} \) as containing all the states that can reach a state satisfying \( \alpha_2 \) in at most \( i \) steps but always through a path satisfying \( \alpha_1 \).

Given that

\[
C_0 \subseteq C_1 \subseteq C_2 \subseteq \ldots
\]

and the fact that \( S \) finite, we know that there is some \( k \) such that \( C_k = C_{k+1} \) and this will be the least fixed point of \( U_{\alpha_1,\alpha_2} \), that is, we have

\[
C_0 \subsetneq C_1 \subsetneq C_2 \subsetneq \ldots \subsetneq C_k = C_{k+1} = \ldots
\]
Input: transition system $C$ with state set $S$, and CTL formula $E(\alpha_1 \cup \alpha_2)$

Output: $\text{Sat}(E(\alpha_1 \cup \alpha_2)) = \{s \in S \mid s \models E(\alpha_1 \cup \alpha_2)\}$

\[ E := \text{Sat}(\alpha_2); \]
\[ C := E; \]

while $E \neq \emptyset$ do

let $s' \in E$;
\[ E := E \setminus \{s'\}; \]

forall $s \in \text{Pred}(s')$ do

if $s \in \text{Sat}(\alpha_1) \setminus C$ then

\[ E := E \cup \{s\}; \]
\[ C := C \cup \{s\}; \]

fi

od

od

return $C$

Figure 6.5: Algorithm for computing the set $\text{Sat}(E(\alpha_1 \cup \alpha_2))$.

Clearly, we will have $C_k = \text{Sat}(E(\alpha_1 \cup \alpha_2))$. In Figure 6.5 we present an algorithm for computing $\text{Sat}(E(\alpha_1 \cup \alpha_2))$ capitalizing on these ideas. Observe that every state that satisfies $\alpha_2$ also satisfies $E(\alpha_1 \cup \alpha_2)$. This is the set $C_1$ that we use to initialize the variable $E$. The idea now is to search backwards for states that can reach the states in this set and that simultaneously satisfy $\alpha_1$. Observe that after the first iteration we will have computed the set $C_2$. Variable $E$ keeps all the states that satisfy $E(\alpha_1 \cup \alpha_2)$ but have not yet been visited.

**Example 6.20.** Let $\Xi = \{p, q\}$ and consider the transition system depicted in Figure 6.6.

For this transition system, we have $\text{Sat}(p) = \{s_0, s_3, s_4, s_5\}$ and $\text{Sat}(q) = \{s_2, s_3\}$. Consequently, we also have $\text{Sat}(p \land q) = \text{Sat}(p) \cap \text{Sat}(q) = \{s_3\}$.

We are no interested in computing $\text{Sat}(E(p \cup (p \land q)))$. To this end, we will use the algorithm in Figure 6.5.
Initially, $E = \text{Sat}(p \land q) = \{s_3\}$ and $C = E = \{s_3\}$. Recall that the set $C$ will contain all the states that satisfy the formula. Hence, at this point, we know that $E(p \mathbin{U} (p \land q))$ holds in $s_2$.

As $E \neq \emptyset$, during the first iteration of the loop, we choose $s' = s_3$ and remove it from $E$. Then, we compute $\text{Pred}(s_3) = \{s_4\}$ and as $s_4 \in \text{Sat}(p) \setminus C = \{s_0, s_3, s_5\}$ then we add this state to $E$ and to $C$. Hence, after the first iteration of the loop, we have $E = \{s_4\}$ and $C = \{s_3, s_4\}$. Consequently, we know that $E(p \mathbin{U} (p \land q))$ holds in the states $s_3$ and $s_4$.

As $E \neq \emptyset$ still holds then, during the second iteration, we have $s' = s_4$ and remove it from $E$. Then, we compute $\text{Pred}(s_4) = \{s_5\}$ and as $s_5 \in \text{Sat}(p) \setminus C = \{s_0, s_5\}$ then we add this state to $E$ and to $C$. And so, after the second iteration, we have $E = \{s_5\}$ and $C = \{s_3, s_4, s_5\}$.

We proceed to the third iteration, were we set $s' = s_5$ and remove it from $E$. After computing $\text{Pred}(s_5) = \{s_3\}$ we conclude that $s_5 \notin \text{Sat}(p) \setminus C = \{s_0\}$. Consequently the sets $E$ and $C$ are not changed and the algorithm ends with $C = \{s_3, s_4, s_5\}$. This is precisely $\text{Sat}(E(p \mathbin{U} (p \land q)))$. Hence, we can conclude that $C \not\models E(p \mathbin{U} (p \land q))$ because $s_0 \notin \text{Sat}(E(p \mathbin{U} (p \land q)))$.

Next, we want to compute $\text{Sat}(E(p \mathbin{U} q))$. In this case, we just briefly sketch the main steps of the algorithm.

Initially, we have $E = \text{Sat}(q) = \{s_2, s_3\}$ and $C = E = \{s_2, s_3\}$. During the first iteration, we choose $s' = s_2$ and, at the end of this iteration, we have $E = \{s_0, s_3, s_4\}$ and $C = \{s_0, s_2, s_3, s_4\}$.

Figure 6.6: A transition system.
During the second iteration, we choose \( s' = s_0 \) and, when this iteration end, we have \( E = \{s_3, s_4\} \) and \( C = \{s_0, s_2, s_3, s_4\} \).

During the third iteration, we choose \( s' = s_3 \) and, when this iteration end, we have \( E = \{s_4\} \) and \( C = \{s_0, s_2, s_3, s_4\} \).

During the fourth iteration, we choose \( s' = s_4 \) and, when this iteration end, we have \( E = \{s_5\} \) and \( C = \{s_0, s_2, s_3, s_4, s_5\} \).

Finally, during the last iteration, we choose \( s' = s_5 \) and, when this iteration end, we have \( E = \emptyset \) and \( C = \{s_0, s_2, s_3, s_4, s_5\} \). Hence, the algorithm will return \( \text{Sat}(E(p \lor q)) = \{s_0, s_2, s_3, s_4, s_5\} \). In this case, as \( s_0 \in \text{Sat}(E(p \lor q)) \) we can conclude that \( C \models E(p \lor q) \).

Now, we consider the computation of the set \( \text{Sat}(EG\alpha) \). Again, we observe that the operator \( G_\alpha \) (over the complete lattice \( (2^S, \subseteq) \)) is continuous and therefore the infimum of the descending chain

\[
S \supseteq G_\alpha(S) \supseteq G_\alpha(G_\alpha(S)) \supseteq \ldots
\]

is the greatest fixed point of \( G_\alpha \), that is, \( \text{Sat}(EG\alpha) \). This means that

\[
\text{Sat}(EG\alpha) = \bigcap_{i \geq 0} G_\alpha^i(S).
\]

We proceed to analyze this chain in detail. In this case, let \( C_i = G_\alpha^i(S) \), for every \( i \geq 0 \). Then, \( C_0 = S \), \( C_1 = G_\alpha(S) = \text{Sat}(\alpha) \)

\[
C_2 = G_\alpha(C_1) = \text{Sat}(\alpha) \cap \{s \in \text{Sat}(\alpha) \mid \text{Suc}(s) \cap C_1 \neq \emptyset\}
\]

\[
C_3 = G_\alpha(C_2) = \text{Sat}(\alpha) \cap \{s \in \text{Sat}(\alpha) \mid \text{Suc}(s) \cap C_2 \neq \emptyset\}
\]

In this last equation, we used the fact that \( C_2 \subseteq C_1 \). In general, we have for every \( i \geq 0 \)

\[
C_{i+1} = C_i \cap \{s \in \text{Sat}(\alpha) \mid \text{Suc}(s) \cap C_i \neq \emptyset\}.
\]
Like in the case of the operator $U$, given that
\[ C_0 \supseteq C_1 \supseteq C_2 \supseteq \ldots \]
and the fact that $S$ is finite, we know that there is some $k$ such that $C_k = C_{k+1}$ and this will be the greatest fixed point of $G_\alpha$, that is, we have
\[ C_0 \supseteq C_1 \supseteq C_2 \supseteq \cdots \supseteq C_k = C_{k+1} = \ldots. \]
In this case, we will have $C_k = \text{Sat}(\text{EG} \alpha)$. We look into the construction each $C_i$. So, when constructing the set $C_2$ from the set $C_1$, we just keep the states that do not refute $\text{EG} \alpha$, that is, all the sets that have at least a successor for which $\alpha$ holds. This means that all the states that refute $\text{EG} \alpha$ in $C_1$ are discarded when defining $C_2$, that is, $C_2$ is obtained from $C_1$ by removing all the states $s \in \text{Sat}(\alpha)$ such that
\[ \text{Suc}(s) \cap C_1 = \emptyset. \]
The above sequence of sets $C_i$ can be computed by a backward search, like in the case of the until. In this case, we start with $C = \text{Sat}(\alpha)$ and $E = S \setminus \text{Sat}(\alpha)$. In general, the set $C$ contains all the states for which $\text{EG} \alpha$ has not yet been refuted. The set $E$ contains all the sets that have not yet been visited but for which it has already been determined that they do not fulfill $\text{EG} \alpha$.

During the backward search, each state $s \in C$ is removed from $C$ if it has been determined that $s$ refutes $\text{EG} \alpha$, that is, if
\[ \text{Suc}(s) \cap C = \emptyset. \]
Observe that even though $T, s \models \alpha$ all its successor states refute $\text{EG} \alpha$ (as they are not in $T$) and consequently, $T, s \not\models \text{EG} \alpha$. As these states are determined, they are added to $E$ to enable the possible removal of other states from $C$.

In order to keep track of the test $\text{Suc}(s) \cap C = \emptyset$, we use a counter for each state $s$, $\text{count}[s]$, that keeps track of the number of direct successors of $s$ that are still in $C \cup E$, that is,
\[ \text{count}[s] = |\text{Suc}(s) \cap (C \cup E)|. \]
Initially, since $C \cup E = S$ we set $\text{count}[s] = |\text{Suc}(s)|$. Then, given $s' \in E$, that is, $s'$ refute $\text{EG} \alpha$, if $s \in \text{Pred}(s')$ and $s \in C$ then this means that
$s'$ is a successor state of $s$ that has just been analyzed (and consequently removed from $E$) but is not in $C$. Hence, $s'$ was successor of $s$ and was in $C \cup E$ but will be removed from this set. Hence, in this case, $\text{count}[s]$ must be decremented. Once this number reaches 0, we know that $\text{Suc}(s) \cap (C \cup E) = \emptyset$ which implies that $\text{Suc}(s) \cap C = \emptyset$ and, consequently, $s$ must be removed from $C$. Finally, when the algorithm ends, we have $E = \emptyset$ and so $\text{count}[s] = |\text{Suc}(s) \cap C|$. It follows that any state $s \in \text{Sat}(\alpha)$ for which $\text{count}[s] > 0$ satisfies $\text{EG} \alpha$.

**Example 6.21.** Recall the transition system in Figure 6.6. We are going to build $\text{Sat}(\text{EG} p)$ using the algorithm in Figure 6.7.

Initially, we have $C = \text{Sat}(p) = \{s_0, s_3, s_4, s_5\}$ and $E = S \setminus \text{Sat}(p) = \{s_1, s_2\}$. Furthermore, we have $\text{count}[s_0] = \text{count}[s_3] = \text{count}[s_2] = 2$ and $\text{count}[s_5] = 1$. That is,

$$C = \{s_0, s_3, s_4, s_5\} \quad E = \{s_1, s_2\} \quad \text{count} = [2, 2, 2, 1].$$

We only consider the relevant states for $\text{count}$. Hence, by $[2, 2, 2, 1]$ we mean $[\text{count}[s_0], \text{count}[s_3], \text{count}[s_4], \text{count}[s_5]]$.

During the first iteration, we choose $s' = s_1$, which is removed from $E$. In this case, $\text{Pred}(s_1) = \{s_0, s_2, s_3\}$ and, of these states, only $s_0$ and $s_3$ are in $C$. Consequently, only $\text{count}[s_0]$ and $\text{count}[s_3]$ are decremented. Hence, after the first iteration, the content of the relevant variables is

$$C = \{s_0, s_3, s_4, s_5\} \quad E = \{s_2\} \quad \text{count} = [1, 1, 2, 1].$$

In the second iteration, we choose $s' = s_2$ which, like above, is removed from $E$. We compute $\text{Pred}(s_2) = \{s_0, s_1, s_4\}$ and, of these states, only $s_0$ and $s_4$ are in $C$. Consequently, $\text{count}[s_0]$ and $\text{count}[s_4]$ have to be decremented. In this case, $\text{count}[s_0]$ reaches 0 which means that we have to remove $s_0$ from $C$ and add it to $E$. After the second iteration, the content of the relevant variables is

$$C = \{s_3, s_4, s_5\} \quad E = \{s_0\} \quad \text{count} = [0, 1, 1, 1].$$

In the third iteration, we choose $s' = s_0$ and remove it from $E$. As $\text{Pred}(s_0) = \emptyset$ then there is nothing else to be done in this iteration. And, as $E = \emptyset$, the algorithm ends with $C = \{s_3, s_4, s_5\}$.

Hence $\text{Sat}(\text{EG} p) = \{s_3, s_4, s_5\}$ and we can conclude that $T \not\vDash \text{EG} p$ because $s_0 \not\in \text{Sat}(\text{EG} p)$.
6.2. MODEL CHECKING

Input: transition system $T$ with state set $S$, and CTL formula $\mathsf{EG} \alpha$

Output: $\mathsf{Sat}(\mathsf{EG} \alpha) = \{ s \in S \mid s \models \mathsf{EG} \alpha \}$

\[
E := S \setminus \mathsf{Sat}(\alpha);
C := \mathsf{Sat}(\alpha);
\text{forall } s \in \mathsf{Sat}(\alpha) \text{ do }
\hspace{1em} \text{count}[s] := |\mathsf{Suc}(s)|
\text{od}
\text{while } E \neq \emptyset \text{ do }
\hspace{1em} \text{let } s' \in E;
\hspace{1em} E := E \setminus \{s'\};
\hspace{1em} \text{forall } s \in \mathsf{Pred}(s') \text{ do }
\hspace{2em} \text{if } s \in C \text{ then }
\hspace{3em} \text{count}[s] := \text{count}[s] - 1;
\hspace{3em} \text{if } \text{count}[s] = 0 \text{ then }
\hspace{4em} C := C \setminus \{s\};
\hspace{4em} E := E \cup \{s\};
\hspace{5em} \text{fi}
\hspace{2em} \text{fi}
\text{od}
\text{od}
\text{return } C
\]

Figure 6.7: Algorithm for computing the set $\mathsf{Sat}(\mathsf{EG} \alpha)$. 
We now present an alternative algorithm for determining $\text{Sat}(\text{EG} \alpha)$, based on strongly connected components of a graph. Given a directed graph $G = \langle V, E \rangle$ and $C \subset V$. We say that $C$ is strongly connected if for every pair $v_1, v_2 \in C$, they are mutually reachable, that is, $v_1 \in \text{Suc}^*(v_2)$ and $v_2 \in \text{Suc}^*(v_1)$. A strongly connected component (SCC) of $G$ is a maximal strongly connected set of vertices. A SCC $C$ is called trivial if $C = \{v\}$ and $\langle v, v \rangle \notin E$. For determining a SCC of a graph we can use a variant of a DFS algorithm.

Assume that we are trying to compute $\text{Sat}(\text{EG} \alpha)$ and we have already determined $\text{Sat}(\alpha)$. Then, we can consider a transition system with just the states that satisfy $\alpha$. Any state that does not satisfy $\alpha$ can be safely removed because they will not satisfy $\text{EG} \alpha$. Hence, given $T = \langle S, A, \rightarrow, I, \Xi, L \rangle$ we define the transition system

$$T[\alpha] = \langle S', A, \rightarrow', I', \Xi, L' \rangle$$

where:

- $S' = S \cap \text{Sat}(\alpha)$,
- $\rightarrow' = \rightarrow \cap (S' \times A \times S')$,
- $I' = I \cap S'$,
- $L'(s) = L(s)$, for every $s \in S'$.

Next, we compute all the nontrivial SCCs in $G(T)$. All the states in such SCC satisfy $\text{EG} \alpha$. Finally, all states that can reach a state in one of such SCC are also computed because those also satisfy $\text{EG} \alpha$. Indeed, given a state $s \in S$ such that a path from $s$ to an SCC exists then (by construction of $T[\alpha]$), $\text{EG} \alpha$ holds at $s$. This can be done by a backward search.

**Example 6.22.** Recall the transition system $T$ in Figure 6.6. The transition system $T[b]$ is depicted in Figure 6.8. We have removed all the states that are not in $\text{Sat}(p)$, that is, we have remove the states $s_1$ and $s_2$, as well as the corresponding transitions.

In this case, there is only one SCC, the set $\{s_3, s_4, s_5\}$. Since $s_0$ cannot reach any of the states in this SCC then it is not considered.
6.3. COUNTEREXAMPLES AND WITNESSES

\[ s_0 \rightarrow s_3 \]

\[ s_3 \rightarrow s_4 \]

\[ s_4 \rightarrow s_5 \]

\[ \{ p, q \} \]

\[ \{ p \} \]

\[ \{ p \} \]

Figure 6.8: A transition system.

**Theorem 6.23.** Let \( T = \langle S, A, \rightarrow, I, \Xi, L \rangle \) be a transition system and \( \alpha \) a CTL formula. Then,

\[ T, s \models EG\alpha \quad \text{if and only if} \quad T, s \models \alpha \quad \text{and there a nontrivial SCC in } T[\alpha] \quad \text{reachable from } s. \]

The proof of this result is left as an exercise.

### 6.3 Counterexamples and witnesses

In LTL, when \( T \not\models \varphi \) we are able to provide a counterexample by providing a sufficiently long prefix of a path \( \pi \) that has enough information to understand why that \( \pi \) refutes \( \varphi. \) For instance, if \( T \not\models F_p \) then a counterexample for this formula would be an initial path fragment \( s_0 \ldots s_n \ldots s_n \) such that \( s_n \ldots s_n \) constitutes a cycle in \( T \) and \( p \) does not hold in any of these states.

In CTL, we have two consider two different situations. For for a formula of the form \( A \alpha \) a suffix of a path refuting \( \alpha \) is enough to conclude that \( T \not\models A \alpha. \) However, in the case of the existential quantifier, that is, in the case of a formula of the form \( E \alpha, \) if \( T \not\models E \alpha \) then all paths refute \( \alpha \) and it is unclear what a counterexample would look like. In this case, if the answer is “yes” then maybe we are interested in an initial path fragment confirming
α, while in the other case, the answer "no" is quite enough. In this case, we will call such a path fragment a witness.

We now explain how to obtain counterexamples and witnesses for the temporal operators $X$, $U$ and $G$.

Let $T = \langle S, A, \rightarrow, I, \Xi, L \rangle$ be a finite transition system without terminal states.

A counterexample for $AX\alpha$ is a pair of states $\langle s, s' \rangle$ with $s \in I$, $s' \in \text{Suc}(s)$ such that $T, s' \not\models \alpha$.

A witness for $EX\alpha$ is a pair of states $\langle s, s' \rangle$ with $s \in I$, $s' \in \text{Suc}(s)$ and $T, s' \models \alpha$.

Hence, both, witnesses and counterexamples of the next operator (either with an existential path quantifier or with a universal path quantifier) can be obtained by inspecting the successor states of the initial states.

A witness for $E(\alpha_1 U \alpha_2)$ is an initial path fragment $s_0 s_1 s_2 \ldots s_n$ such that $T, s_n \models \alpha_2$ and $T, s_i \models \alpha_1$, for every $0 \leq i < n$. Such a witness can be determined by a backward search starting on the states in $\text{Sat}(\alpha_2)$.

A counterexample for $A(\alpha_1 U \alpha_2)$ is an initial path fragment $s_0 s_1 s_2 \ldots s_n$ from which we can conclude that either $G(\alpha_1 \land \neg \alpha_2)$ or $(\alpha_1 \land \neg \alpha_2) U (\neg \alpha_1 \land \neg \alpha_2)$ holds along such a path.

In the first case, it is enough that $T, s_i \models \alpha_1 \land \neg \alpha_2$, for $0 \leq i < n$ and there is $k < n$ such that $s_k = s_n$, i.e. the path fragment contains a cycle.

In the second case, it is enough that $T, s_i \models \alpha_1 \land \neg \alpha_2$, for $0 \leq i \leq n$ and $T, s_n \models \neg \alpha_1 \land \neg \alpha_2$.

Such counterexamples can be determined by examining the graph $G = \langle S, E \rangle$ where

$$E = \{ \langle s, s' \rangle \in S \times S \mid s' \in \text{Suc}(s) \text{ and } T, s \models \alpha_1 \land \neg \alpha_2 \}.$$ 

Then, we determine the SCCs of $G$. Each path fragment in $G$ from an initial state $s_0$ that leads to a nontrivial SCC $C$ in $G$ provides a counterexample for the first case, that is, it is an initial path fragment $s_0 s_1 s_2 \ldots s_n$ such that $T, s_i \models \alpha_1 \land \neg \alpha_2$, for $0 \leq i \leq n$ and there is $k < n$ such that $s_k = s_n$.

Each path fragment $s_0 s_1 \ldots s_n$ from an initial $s_0$ state to a trivial SCC $C = \{ s' \}$ such that $s'$ has no edges departing from it (with $s_n = s'$) and $T, s' \not\models \alpha_2$ provide a counterexample for the second case. Observe that if $s_0 s_1 s_2 \ldots s_n$ is a path in $G$ then $\langle s_i, s_{i+1} \rangle \in E$ for $0 \leq i < n$ and which implies, by construction of $E$, that $T, s_i \models \alpha_1 \land \neg \alpha_2$, for $0 \leq i \leq n$. Furthermore, as $s'$ is a trivial SCC the it means that $s'$ has no successors.
6.4. Symbolic model-checking

6.4.1 Switching functions

A Boolean function of arity \( n \) is a map from \( \{0, 1\}^n \) to \( \{0, 1\} \). However, for technical reasons, it is more convenient to consider such functions as mappings from assignments to Boolean variables to \( \{0, 1\} \).

Let \( \text{Var} = \{x_1, \ldots, x_n\} \) be a set of Boolean variables. We denote by \( \text{Asg}(\text{Var}) \) the set of all assignments \( \eta : \text{Var} \to \{0, 1\} \). Note that an assignment is a function not necessarily total. We will write \([x_1 = b_1, \ldots, x_n = b_n]\) for the assignment \( \eta(x_i) = b_i \), for \( i = 1, \ldots, n \). We also write \([\vec{x} = \vec{b}]\) were \( \vec{x} = \langle x_1, \ldots, x_n \rangle \) and \( \vec{b} = \langle b_1, \ldots, b_n \rangle \).

A switching function for \( \text{Var} \) is a map \( f : \text{Asg}(\text{Var}) \to \{0, 1\} \). The switching functions for \( \text{Var} = \emptyset \) are the constants 0 and 1. We will sometimes write \( f(x_1, \ldots, x_n) \) or \( f(\vec{x}) \) to express that the underlying set of variables for the switching function \( f \) is \( \{x_1, \ldots, x_n\} \).

We can define the usual Boolean connectives over Boolean functions. Let \( f_1 \) be a Boolean function for \( \{x_1, \ldots, x_n, \ldots, x_m\} \) and \( f_2 \) be a Boolean function for \( \{x_n, \ldots, x_m, \ldots, x_k\} \) such that the variables \( x_i \) are pairwise distinct and \( 0 \leq n \leq m \leq k \). Then,

\[ \neg f_1 \text{ is the switching function for } \{x_1, \ldots, x_m\} \text{ such that} \]

\[ \neg f_1([x_1 = b_1, \ldots, x_m = b_m]) = 1 - f_1([x_1 = b_1, \ldots, x_m = b_m]); \]
• $f_1 \lor f_2$ is the switching function for $\{x_1, \ldots, x_k\}$ such that
\[
f_1 \lor f_2([x_1 = b_1, \ldots, x_k = b_k]) =
\max\{f_1([x_1 = b_1, \ldots, x_m = b_m]), f_2([x_n = b_n, \ldots, x_k = b_k])\};
\]

• $f_1 \land f_2$ is the switching function for $\{x_1, \ldots, x_k\}$ such that
\[
f_1 \land f_2([x_1 = b_1, \ldots, x_k = b_k]) =
f_1([x_1 = b_1, \ldots, x_m = b_m]), f_2([x_n = b_n, \ldots, x_k = b_k]).
\]

The switching function $\text{pr}_{x_i}$ for $\{x_1, \ldots, x_n\}$ is defined by $\text{pr}_{x_i}([x_1 = b_1, \ldots, x_n = b_n]) = b_i$, for every $i = 1, \ldots, n$, and is called the projection function for $x_i$. We will sometimes write $x_i$ for $\text{pr}_{x_i}$.

The constant switching function $f$ for $\text{Var}$ such that $f(\vec{x} = \vec{b}) = 0$ is denoted by 0 and constant switching function $f$ for $\text{Var}$ such that $f(\vec{x} = \vec{b}) = 1$ is denoted by 1.

Using this notation, we can write $x_1 \land (\neg x_2 \lor x_3)$ for the switching function $\text{pr}_{x_1}(x_1, x_2, x_3) \land (\neg \text{pr}_{x_2}(x_1, x_2, x_3) \lor \text{pr}_{x_3}(x_1, x_2, x_3))$. But it can also represent the switching function $\text{pr}_{x_1}(x_1) \land (\neg \text{pr}_{x_2}(x_1, x_2) \lor \text{pr}_{x_3}(x_2, x_3))$ and many others.

Let $f$ be a switching function for $\{y, x_1, \ldots, x_n\}$. Then, the positive cofactor of $f$ for variable $y$ is the switching function $f|_{y=1}$ for $\{y, x_1, \ldots, x_n\}$ defined by
\[
f|_{y=1}([y = b, x_1 = b_1, \ldots, x_n = b_n]) = f([y = 1, x_1 = b_1, \ldots, x_n = b_n]).
\]
The negative cofactor of $f$ for variable $y$ is the switching function $f|_{y=0}$ for $\{y, x_1, \ldots, x_n\}$ defined by
\[
f|_{y=0}([y = b, x_1 = b_1, \ldots, x_n = b_n]) = f([y = 0, x_1 = b_1, \ldots, x_n = b_n]).
\]
A variable $y$ is called essential for $f$ if $f|_{y=1} \neq f|_{y=0}$. Clearly, $y$ is not essential for $f|_{y=1}$ neither for $f|_{y=0}$.

Now, let $f$ be a switching function for $\{y_1, \ldots, y_m, x_1, \ldots, x_n\}$. Then, we write $f|_{y_1=b_1,\ldots,y_m=b_m}$ for the iterated cofactor of $f$ given by
\[
f|_{y_1=b_1,\ldots,y_m=b_m} = (\ldots (f|_{y_1=b_1})|_{y_2=b_2})|_{y_m=b_m}.
\]
The values of $f|_{y_1=b_1,...,y_m=b_m}$ are given by

$$f|_{y_1=b_1,...,y_m=b_m}([y_1 = b'_1, \ldots, y_m = b'_m, x_1 = b''_1, \ldots, x_n = b''_n]) = f([y_1 = b_1, \ldots, y_m = b_m, x_1 = b'_1, \ldots, x_n = b'_n]).$$

Observe that, as a consequence of the definition, the iterated cofactor does not depend on the order in which the cofactors for the single variables are considered.

**Example 6.24.** Consider the switching function $f(x_1, x_2, x_3)$ given by $x_1 \land (\neg x_2 \lor x_3)$. Then

$$f|_{x_1=1} = (\neg x_2 \lor x_3)$$

and

$$f|_{x_1=0} = 0$$

Hence, we can conclude that $x_1$ is essential for $f$.

Clearly, $x_2$ and $x_3$ are not essential for $pr_{x_1}(x_1, x_2, x_3)$. On the other hand, as it would be expected, $x_1$ is essential for $pr_{x_1}(x_1, x_2, x_3)$.

Consider now the switching function

$$f'(x_1, x_2) = (x_1 \land x_2) \lor (\neg x_1 \land \neg x_2).$$

In this case, $x_2$ is essential for $f'$ because

$$f'|_{x_2=0} = 0$$

and

$$f'|_{x_2=1} = x_1 \lor (\neg x_1).$$

However, $x_1$ is not essential because

$$f'|_{x_1=0} = \neg x_2$$

agrees with

$$f'|_{x_1=1} = x_2.$$

The following result will be useful later. It yields a decomposition of a switching function into its cofactors.
Lemma 6.25 (Shannon expansion). Let $f$ be a switching function for $\text{Var}$. Then, for each $x \in \text{Var}$,

$$f = (\neg x \land f|_{x=0}) \lor (x \land f|_{x=1}).$$

We can also use quantification over switching functions. Let $f$ be a switching function over $\text{Var}$ and let $x \in \text{Var}$. Then, $\exists x.f$ denotes the switching function $f|_{x=0} \lor f|_{x=1}$. Similarly, $\forall x.f$ denotes the switching function $f|_{x=0} \land f|_{x=1}$. Observe that $x$ is not essential for $\exists x.f$ nor for $\forall x.f$.

Finally, we introduce the notation for renaming variables in a switching function. In the sequel we will need to rename some variables of a switching function. Let $f(x,y) = x \land y$ be a switching function. If we want to rename some variable $x$ into variable $z$ in $f(x,y)$ this will yield the switching function $z \land y$.

Let $\vec{z} = \langle x_1, \ldots, x_n \rangle$ and $\vec{y} = \langle y_1, \ldots, y_n \rangle$ be two tuples of variables of the same length. Furthermore, let $\vec{z} = \langle z_1, \ldots, z_k \rangle$ by another tuple of variables such that $x_i$ and $y_i$ do not appear in $\vec{z}$, for $i = 1, \ldots, n$. Consider the assignment $\eta = [x_1 = b_1, \ldots, x_n = b_n] \in \text{Asg}(\{x_1, \ldots, x_n, z_1, \ldots, z_k\})$. Then, $\eta\{\vec{x} \leftarrow \vec{y}\}$ denotes the assignment in $\text{Asg}(\{y_1, \ldots, y_n, z_1, \ldots, z_k\})$ such that

$$\eta\{\vec{x} \leftarrow \vec{y}\}(y_i) = \eta(x_i), \text{ for } i = 1, \ldots, n$$

and

$$\eta\{\vec{x} \leftarrow \vec{y}\}(z_j) = \eta(z_j), \text{ for } j = 1, \ldots, k.$$ 

Given a switching function $f$ for $\{x_1, \ldots, x_n, z_1, \ldots, z_k\}$ then $f\{\vec{x} \leftarrow \vec{y}\}$ is the switching function for $\{y_1, \ldots, y_n, z_1, \ldots, z_k\}$ given by

$$f\{\vec{x} \leftarrow \vec{y}\}(\eta) = f(\eta\{\vec{y} \leftarrow \vec{x}\})$$

for every $\eta \in \text{Asg}(\{y_1, \ldots, y_n, z_1, \ldots, z_k\})$. In other words, this means that

$$f\{\vec{x} \leftarrow \vec{y}\}(\eta\{\vec{y} = \vec{b}, \vec{z} = \vec{c}\}) = f(\eta\{\vec{x} = \vec{b}, \vec{z} = \vec{c}\}).$$

For instance, given the switching function $f(x_1, x_2, x_3) = x_1 \land (\neg x_2 \lor x_3)$ we have $f\{\langle x_1, x_3 \rangle \leftarrow \langle y_1, y_3 \rangle\} = y_1 \land (\neg x_2 \lor y_3)$. When it is clear from the context which variables are being renamed, we will write $f(\vec{y}, \vec{z})$ for $f\{\vec{x} \leftarrow \vec{y}\}$. 
6.4. SYMBOLIC MODEL-CHECKING

6.4.2 Encoding transition systems with switching functions

We now proceed to encode a transition system using switching functions. As the action symbols are not relevant for our purposes we will omit them. Hence, let \( T = (S, \rightarrow, I, \Xi, \mathcal{L}) \) be a finite transition system.

Let \( n \geq \log(|S|) \). We consider an (injective) encoding function \( \text{enc} : S \to \{0,1\}^n \). Then, each state is identified by a bit vector of size \( n \). We can assume without loss of generality that \( \text{enc}(S) = \{0,1\}^n \). A subset set \( C \) of \( S \) is represented by its characteristic function \( \chi_C : \{0,1\}^n \to \{0,1\} \).

The transition relation \( \rightarrow \subseteq S \times S \) is represented by a Boolean function \( \Delta : \{0,1\}^{2n} \to \{0,1\} \) that evaluates to 1 on those pairs \( \langle s,s' \rangle \) such that \( s \rightarrow s' \).

So, we are going to use \( n \) variables \( \text{Var} = \{x_1, \ldots, x_n\} \) and we are going to identify each assignment \( [x_1 = b_1, \ldots, x_n = b_n] \in \text{Asg}(\text{Var}) \) with the unique state \( s \in S \) such that \( \text{enc}(s) = \langle b_1, \ldots, b_n \rangle \). As it was already mentioned, we are going to assume that \( S = \text{Asg}\{x_1, \ldots, x_n\} \).

Given a subset \( C \) of \( S \), its characteristic function \( \chi_C \) assigns 1 to every state \( s \in C \) and 0 to every state \( s \notin C \). Hence, \( \chi_C : \text{Asg}(\text{Var}) \to \{0,1\} \) is such that

\[
\chi_C(s) = \begin{cases} 
1 & \text{if } s \in C \\
0 & \text{otherwise.}
\end{cases}
\]

With this, we can represent the set \( \text{Sat}(p) \), for every \( p \in \Xi \), by the switching function \( f_p = \chi_{\text{Sat}(p)} \) for \( \text{Var} \). Hence, the family \( \{f_p\}_{p \in \Xi} \) yields a representation for the labeling function \( \mathcal{L} \).

The set \( I \) of initial states is encoded by its characteristic function \( \chi_I \).

Finally, we consider the representation of the transition relation \( \rightarrow \subseteq S \times S \). In this case, the idea is similar: we are going to identify \( \rightarrow \) with its characteristic function from \( S \times S \) to \( \{0,1\} \) such that we assign the value 1 to each pair \( \langle s,t \rangle \) provided that \( s \rightarrow t \). Since we have two states but only one set of variables \( \{x_1, \ldots, x_n\} \) to encode the states, we are going to use \( \bar{x} \) to encode \( s \) and a copy of \( \{x_1, \ldots, x_n\} \), that we will represent by \( \text{Var}' = \{x_1', \ldots, x_n'\} \) to encode \( t \). Then, we encode the transition relation \( \rightarrow \) by the switching function \( \Delta : \text{Asg}(\text{Var} \cup \text{Var}') \to \{0,1\} \) such that

\[
\Delta(s, t\{\bar{x} \leftarrow \bar{x}'\}) = \begin{cases} 
1 & \text{if } s \rightarrow t \\
0 & \text{otherwise.}
\end{cases}
\]
CHAPTER 6. COMPUTATION TREE LOGIC

where \( s, t \in S = \text{Asg}(\text{Var}) \) and \( t\{\vec{x} \leftarrow \vec{x}'\} \) is the variable assignment obtained from the variable assignment \( t \) by renaming each \( x_i \) by \( x'_i \), for \( i = 1, \ldots, n \).

**Example 6.26.** Consider the transition system \( T \) depicted in Figure 6.9. In this case, we are going to use two variables \( x_1 \) and \( x_2 \) to encode the states. Observe that with two variables, we can encode 4 states but we simply assume that the fourth state has no incoming and no outgoing transition and so we can just ignore it.

The encoding of the states with the adopted variables \( x_1 \) and \( x_2 \) is defined as follows:

\[
\begin{array}{c|cc}
  & x_1 & x_2 \\
\hline
s_0 & 0 & 0 \\
s_1 & 0 & 1 \\
s_2 & 1 & 0 \\
\end{array}
\]

This means that state \( s_0 \) is encoded by the tuple \( \langle 0, 0 \rangle \), i.e. by the variable assignment \( [x_1 = 0, x_2 = 0] \), the state \( s_1 \) is encoded by the tuple \( \langle 0, 1 \rangle \), i.e. by the variable assignment \( [x_1 = 0, x_2 = 1] \) and the state \( s_2 \) is encoded by the tuple \( \langle 1, 0 \rangle \), i.e. by the variable assignment \( [x_1 = 1, x_2 = 0] \).

The set of initial states \( I \) is defined by its characteristic function \( \chi_I \). In this case, \( \chi_I \) is the switching function

\[
\chi_I(x_1, x_2) = \neg x_1 \land \neg x_2.
\]

The labeling function is characterized by the switching functions \( f_p = \)
χ_{Sat(p)} and f_q = \chi_{Sat(q)}. In this case, we have Sat(p) = \{s_0, s_2\} and consequently,

\[ f_p(x_1, x_2) = (\neg x_1 \land \neg x_2) \lor ( x_1 \land \neg x_2) \]

which is equivalent to the switching function

\[ f_p(x_1, x_2) = \neg x_2. \]

Similarly, we have Sat(q) = \{s_2\} and so

\[ f_q(x_1, x_2) = \neg x_1 \land x_2. \]

Finally, we consider the transition relation $\rightarrow$. The switching function $\Delta(x_1, x_2, x'_1, x'_2)$ is defined by

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x'_1$</th>
<th>$x'_2$</th>
<th>$\Delta$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

The only rows were $\Delta$ is 1 are precisely the ones corresponding to the transitions in $T$. For instance, $\Delta([x_1 = 0, x_2 = 0, x'_1 = 0, x'_2 = 1]) = 1$ which means that there is a transition from state $\langle 0, 0 \rangle$ to state $\langle 0, 1 \rangle$. According to the encoding that we adopted for the states, this is precisely the transition $s_0 \rightarrow s_1$. 
The full switching function is
\[
\Delta(x_1, x_2, x'_1, x'_2) = (¬x_1 \wedge ¬x_2 \wedge ¬x'_1 \wedge x'_2) \lor (¬x_1 \wedge ¬x_2 \wedge x'_1 \wedge ¬x'_2) \lor (x_1 \wedge ¬x_2 \wedge ¬x'_1 \wedge x'_2)
\]
which is equivalent to the switching function
\[
\Delta(x_1, x_2, x'_1, x'_2) = (¬x_1 \wedge x'_1 \wedge ¬x'_2) \lor (¬x_2 \wedge ¬x'_1 \wedge x'_2).
\]

We now proceed to adapt the model checking algorithm for CTL to this symbolic representation of transition systems. We start by the definition of the set \(\text{Suc}(s)\) for some \(s \in S\). Let \(\Delta\) be the switching function representing the transition relation of a transition system and let \(s \in \text{Asg}(\text{Var})\) be one of its states. Then, \(\text{Suc}(s) = \{s' \in S \mid s \rightarrow s'\}\) is the set obtained from \(\Delta\) by setting \(s\) as the assignment for the variables. The result will be an assignment to the variables in \(\text{Var}'\) which we then need to convert back to an assignment in \(\text{Asg}(\text{Var})\). Hence, for each \(s \in S\), the switching function \(\chi_{\text{Suc}(s)}\) for the set \(\text{Suc}(s)\) is defined by
\[
\chi_{\text{Suc}(s)} = \Delta|_{\{\vec{x}' \leftarrow \vec{x}\}}.
\]

**Example 6.27.** Recall the transition system from Example 6.26. The successor set \(\text{Suc}(s_0) = \{s_1, s_2\}\) is obtained symbolically as follows
\[
\Delta|_{x_1=0, x_2=0}(\vec{x}' \leftarrow \vec{x}) = (x'_1 \wedge ¬x'_2) \lor (¬x'_1 \wedge x'_2)\{\vec{x}' \leftarrow \vec{x}\} = (x_1 \wedge ¬x_2) \lor (¬x_1 \wedge x_2).
\]
Recall that \(s_0 = [x_1 = 0, x_2 = 0]\). The only two states for which the previous functions yields the value 1 are \([x_1 = 0, x_2 = 1]\) and \([x_1 = 1, x_2 = 0]\), that is, the states \(s_1\) and \(s_2\).

Given the switching function \(\Delta(\vec{x}, \vec{x}')\), we briefly describe model checking algorithm for CTL.

Consider the set \(\text{Sat}(\text{EX}\alpha)\). Recall that this set contains all the states \(s\) such that \(\text{Suc}(s) \cap \text{Sat}(\alpha) \neq \emptyset\). Then, \(\chi_{\text{Sat}(\text{EX}\alpha)}\) is the switching function given by
\[
\chi_{\text{Sat}(\text{EX}\alpha)}(\vec{x}) = \exists \vec{x}' . (\Delta(\vec{x}, \vec{x}') \land \chi_{\text{Sat}(\alpha)}(\vec{x}')).
\]
Next, we consider the set \(\text{Sat}(\text{E}(\alpha_1 \cup \alpha_2))\). For convenience, let \(B_1 = \text{Sat}(\alpha_1)\) and \(B_2 = \text{Sat}(\alpha_2)\) and let \(\chi_{B_1}\) and \(\chi_{B_2}\) be the switching functions
6.4. SYMBOLIC MODEL-CHECKING

\[ f_0(\vec{x}) := \chi_{B_2}(\vec{x}); \]
\[ j := 0; \]
repeat
\[ f_{j+1}(\vec{x}) := f_j(\vec{x}) \lor (\chi_{B_1}(\vec{x}) \land \exists \vec{x}'.(\Delta(\vec{x}, \vec{x}') \land f_j(\vec{x}'))); \]
\[ j := j + 1; \]
until \( f_j(\vec{x}) = f_{j-1}(\vec{x}) \)
return \( f_j(\vec{x}) \)

Figure 6.10: Symbolic computation of \( \text{Sat}(E(\alpha_1 \cup \alpha_2)) \).

\[ f_0(\vec{x}) := \chi_{B}(\vec{x}); \]
\[ j := 0; \]
repeat
\[ f_{j+1}(\vec{x}) := f_j(\vec{x}) \land (\exists \vec{x}'.(\Delta(\vec{x}, \vec{x}') \land f_j(\vec{x}'))); \]
\[ j := j + 1; \]
until \( f_j(\vec{x}) = f_{j-1}(\vec{x}) \)
return \( f_j(\vec{x}) \)

Figure 6.11: Symbolic computation of \( \text{Sat}(EG \alpha) \).

for the corresponding characteristic maps. In this case, we construct a succession of functions \( f_i \) that capture the backward search along states satisfying \( \alpha_1 \) starting from states that satisfy \( \alpha_2 \).

Recall that we write \( f(\vec{x}') \) for \( f\{\vec{x} \leftarrow \vec{x}'\} \).

The set \( \text{Sat}(EG \alpha) \) can be computed in a similar way. In this case, we mimic the construction of the largest set \( C \subset \text{Sat}(\alpha) \) with \( \text{Suc}(s') \cap C \neq \emptyset \), for \( s' \in T \). Let \( B = \text{Sat}(\alpha) \) and \( \chi_B \) be the switching function for the corresponding characteristic map.
6.4.3 Ordered Binary Decision Diagrams

Let $f$ be a switching function over $\text{Var} = \{x_1, \ldots, x_n\}$. A binary decision tree for $f$ is a binary tree where the root node is labelled with $x_1$ and has two outgoing edges to the lower level, where both nodes are labelled with $x_2$. These two edges stand for the cases $x_1 = 0$ (represented by a dashed line) and $x_1 = 1$ (represented by a solid line). This procedure is repeated for the other variables. The leaves of the tree are labelled with 0 or 1, depending on the value of the function for the particular values that the variables have taken along the path to that leave. For instance, in Figure 6.12, we have a binary decision tree for the switching function $f(x_1, x_2) = \neg(x_1 \lor x_2)$.

Ordered Binary Decision Diagrams (OBDDs) are going to be used represent switching functions. Intuitively, an OBDD is a compressed version of a binary decision tree were all redundant nodes are eliminated. This means collapsing constant subtrees (trees where all the terminal nodes have the same value) into a single node and then identifying nodes with isomorphic subtrees.

**Example 6.28.** Recall the binary decision tree depicted in Figure 6.12. In this case, as all the terminal nodes of the left subtree are labelled with 0 (which reflects the fact that the value of the cofactor $f|_{x_1=1}$ agrees with the constant function 0), the test to variable $x_2$ in the left subtree is redundant and so this sub tree can be replaced by the terminal node 0, as depicted in Figure 6.13.
Then, we can identify all the terminal nodes with the same value, leading to the diagram depicted in Figure 6.14.

A variable ordering $\varphi$ for $\text{Var} = \{x_1, \ldots, x_n\}$ is any tuple $\varphi = \langle x_{i_1}, \ldots, x_{i_n} \rangle$ such that $i_k \neq i_m$ for $k, m = 1, \ldots, n$. We will write $<_\varphi$ for the induced total order on $\text{Var}$.

**Definition 6.29.** Let $\varphi$ be a variable ordering on $\text{Var}$. A $\varphi$-ordered binary decision diagram (\(\varphi\)-OBDD) is a tuple

$$\mathcal{B} = \langle V, V_I, V_T, \text{succ}_0, \text{succ}_1, \text{var}, \text{val}, v_0 \rangle$$

such that

- $V$ is a finite set of nodes;
- $V_I, V_T \subseteq V$ such that $V_I \cap V_T = \emptyset$ and $V_I \cup V_T = V$;
- $\text{succ}_0, \text{succ}_1 : V_I \to V$;
- $\text{var} : V_I \to \text{Var}$;
- $\text{val} : V_T \to \{0, 1\}$;
- $v_0 \in V$, the root node,

and the following conditions are met:
Figure 6.14: Binary decision diagram for \( \neg(x_1 \lor x_2) \).

- for every \( v \in V_I \) and \( w \in \{ \text{succ}_0(v), \text{succ}_1(v) \} \cap V_I \) it must be the case that \( \text{var}(v) <_\varphi \text{var}(w) \),

- for every \( v \in V \setminus \{ v_0 \} \) there is \( w \in V \) such that \( v = \text{succ}_0(w) \) or \( v = \text{succ}_1(w) \),

- there is no \( v \in V \) such that \( v_0 \in \{ \text{succ}_0(v), \text{succ}_1(v) \} \).

\( V \) is the set of all the nodes of \( \mathcal{B} \). The elements in \( V_I \) are the inner nodes and the elements in \( V_T \) are the terminal nodes. The function \( \text{succ}_0 \) assigns to each inner \( v \) node a 0-successor node \( \text{succ}_0(v) \in V \) and the function \( \text{succ}_1 \) assigns to each inner \( v \) node a 1-successor node \( \text{succ}_1(v) \in V \). The function \( \text{var} \) assigns to each inner node a label consisting of a variable. The function \( \text{val} \) assigns to each terminal node a value that is either 0 or 1. Observe that the diagrams in Figures 6.12, 6.13 and 6.14 are all \( \varphi \)-OBDDs for the variable ordering \( x_1 <_\varphi x_2 \).

Let \( \mathcal{B} = (V, V_I, V_T, \text{succ}_0, \text{succ}_1, \text{var}, \text{val}, v_0) \) be a \( \varphi \)-OBDD. Then, \( \mathcal{B} \) determines the switching function \( f_B \) for \( \text{Var} \) where \( f_B([x_1 = b_1, \ldots, x_n = b_n]) \) is the value of the terminal node that is reached when transversing the graph starting at the node \( v_0 \) and following the path according to \( [x_1 = b_1, \ldots, x_n = b_n] \), i.e., if the current node is \( v \) and \( \text{var}(v) = x_i \) then the next node is determined by \( \text{succ}_b(v) \). Formally, for each node \( v \in V \), let \( f_v : \text{Asg}(\text{Var}) \to \{ 0, 1 \} \) denote the switching function determined by that node, that is defined, for every \( \eta \in \text{Asg}(\text{Var}) \), by
6.4. SYMBOLIC MODEL-CHECKING

- if $v \in V_T$ then $f_v(\eta) = \text{val}(v)$

- if $v \in V_I$ then $f_v(\eta) = \begin{cases} f_{\text{succ}_0}(v)(\eta) & \text{if } \eta(\var(v)) = 0 \\ f_{\text{succ}_1}(v)(\eta) & \text{if } \eta(\var(v)) = 1. \end{cases}$

In particular, we have that $f_B = f_{v_0}$.

**Example 6.30.** Recall the $\varphi$-OBDD in Figure 6.13 such that nodes are labelled as follows: the root node is $v_0$, the right descendant is $v_1$ and the leaves, from left to right, are labelled by $v_2$, $v_3$ and $v_4$, respectively. Consider the variable assignment $\eta = [x_1 = 0, x_2 = 0]$. Then,

$$f_B(\eta) = f_{v_0}(\eta) = f_{\text{succ}_0}(v_0)(\eta) \quad \text{as } \var(v_0) = x_1 \text{ and } \eta(x_1) = 0$$

$$= f_{v_1}(\eta) \quad \text{as } \text{succ}_0(v_0) = v_1$$

$$= f_{\text{succ}_0}(v_1)(\eta) \quad \text{as } \var(v_1) = x_2 \text{ and } \eta(x_2) = 0$$

$$= f_{v_4}(\eta) \quad \text{as } \text{succ}_0(v_1) = v_4$$

$$= \text{val}(v_4) = 1$$

The value of $f_B$ for the remaining assignments can be determined in a similar manner.

Let $v \in V$ and consider the switching function $f_v$ determined by $v$. It is not very difficult to prove that any variable $x \in \text{Var}$ such that $x <_\varphi \var(v)$ is not essential for $f_v$, as it appears as a label of nodes that are above $v$ in the $\varphi$-OBDD. This means that, at most, the variables $y \in \text{Var}$ such that $\var(v) \leq_\varphi y$ can be essential for $f_v$. In particular, if $\var(v) = x$ then, as $x <_\varphi \var(\text{succ}_0(v))$ and $x <_\varphi \var(\text{succ}_1(v))$, it follows that

$$f_{\text{succ}_0}(v)|_{x=b} = f_{\text{succ}_0}(v) \quad \text{and} \quad f_{\text{succ}_1}(v)|_{x=b} = f_{\text{succ}_1}(v)$$

for any $b \in \{0, 1\}$.

In addition, if $\var(v) = x_i$ then

$$f_v|_{x_i=0}([x_1 = b_1, \ldots, x_i = b_i, \ldots, x_n = b_n])$$

$$= f_v([x_1 = b_1, \ldots, x_i = 0, \ldots, x_n = b_n])$$

$$= f_{\text{succ}_0}(v)([x_1 = b_1, \ldots, x_i = 0, \ldots, x_n = b_n])$$

$$= f_{\text{succ}_0}(v)|_{x_i=0}([x_1 = b_1, \ldots, x_i = b_i, \ldots, x_n = b_n])$$

$$= f_{\text{succ}_0}(v)([x_1 = b_1, \ldots, x_i = b_i, \ldots, x_n = b_n]).$$
This last equality results from the fact that $x_i$ is not essential for $f_{\text{succ}}(v)$, as established above. Hence, we conclude that $f_v|_{x_i=0} = f_{\text{succ}}(v)$. Similarly, we can conclude that $f_v|_{x_i=1} = f_{\text{succ}}(v)$. Hence, using the Shannon expansion, we can establish the following lemma.

**Lemma 6.31.** Let $B$ be a $\wp$-OBDD and $v \in V$. The switching function $f_v$ is given as follows:

- if $v \in V_T$ then $f_v$ is the constant function with value $\text{val}(v)$;
- if $v \in V_I$ and $\text{var}(v) = x$ then $f_v = (\neg x \land f_{\text{succ}}(v)) \lor (x \land f_{\text{succ}}(v))$.

Furthermore, $f_B = f_{v_0}$.

**Example 6.32.** Recall the $\wp$-OBDD in Example 6.30. Then, we have

$$f_B = f_{v_0}$$

$$= (x_1 \land f_{\text{succ}}(v_0)) \lor (\neg x_1 \land f_{\text{succ}}(v_0))$$

$$= (x_1 \land f_{v_2}) \lor (\neg x_1 \land f_{v_1})$$

$$= (x_1 \land \text{val}(v_2)) \lor (\neg x_1 \land ((x_2 \land f_{\text{succ}}(v_1)) \lor (\neg x_2 \land f_{\text{succ}}(v_1))))$$

$$= (x_1 \land 0) \lor (\neg x_1 \land ((x_2 \land f_{v_3}) \lor (\neg x_2 \land f_{v_4})))$$

$$= \neg x_1 \land (0) \lor (\neg x_2 \land 1)$$

$$= \neg x_1 \land \neg x_2$$

$$= \neg (x_1 \lor x_2).$$

Let $f$ be a switching function for $\text{Var}$ and let $\wp = \langle x_1, \ldots, x_n \rangle$ be a variable ordering for $\text{Var}$. A switching function $f'$ for $\text{Var}$ is called a $\wp$-consistent cofactor of $f$ if there is some $i \in \{0, 1, \ldots, n\}$ such that $f' = f|_{x_i=b_i}$, $i \neq 0$. When $i = 0$ we are looking at $f$ as a cofactor of itself.

**Example 6.33.** Consider the switching function $f = \neg(x_1 \lor x_2)$ over $\text{Var} = \{x_1, x_2\}$ with the variable ordering $x_1 <_\wp x_2$. Then, $f_1 = 0$, $f_2 = 1$, $f_3 = \neg x_2$ and $f_4 = f$ are the consistent cofactors of $f$. In particular, we have that

- $f_1 = 0 = f|_{x_1=1} = f|_{x_1=1,x_2=0} = f|_{x_1=1,x_2=1} = f|_{x_1=0,x_2=1}$
6.4. SYMBOLIC MODEL-CHECKING

- \( f_2 = 1 = f|_{x_1=0,x_2=0} \)
- \( f_1 = \neg x_2 = f|_{x_1=0} \)

But, for instance, \( f_5 = \neg x_1 \) is not a \( \varphi \)-consistent cofactor of \( f \).

**Lemma 6.34.** Let \( \mathcal{B} \) be a \( \varphi \)-OBDD and let \( v \in V \). Then, the switching function \( f_v \) induced by \( v \) is a \( \varphi \)-consistent cofactor of \( f_\mathcal{B} \). Conversely, given a consistent cofactor \( f' \) of \( f_\mathcal{B} \) there is a node \( v \) in \( \mathcal{B} \) such that \( f' = f_v \).

In particular, there can be more than one node representing the same consistent cofactor. If we look at the \( \varphi \)-OBDD of Example 6.30, nodes \( v_2 \) and \( v_3 \) represent the same consistent cofactor \( f_1 = 0 \). As we want avoid redundancies, we define the notion of reduced \( \varphi \)-OBDD.

**Definition 6.35.** A \( \varphi \)-OBDD \( \mathcal{B} \) is called reduced if for every two distinct nodes \( v, v' \) we have \( f_v \neq f_{v'} \).

We will write \( \varphi \)-ROBDD for a reduced \( \varphi \)-OBDD.

**Theorem 6.36.** Let \( \text{Var} \) be a set of Boolean variables and \( \varphi \) be an ordering on \( \text{Var} \). Then:
- For each switching function \( f \) for \( \text{Var} \) there exists a \( \varphi \)-ROBDD \( \mathcal{B} \) with \( f_\mathcal{B} = f \).
- Given two \( \varphi \)-ROBDDs \( \mathcal{B}_1 \) and \( \mathcal{B}_2 \) with \( f_{\mathcal{B}_1} = f_{\mathcal{B}_2} \) then \( \mathcal{B}_1 \) and \( \mathcal{B}_2 \) are isomorphic.

Let \( \mathcal{B} \) be a \( \varphi \)-OBDD. Then, the size of \( \mathcal{B} \) is the number of nodes in \( \mathcal{B} \), that is, \( \text{size}(\mathcal{B}) = |V| \).

**Corollary 6.37.** Let \( \mathcal{B} \) be a \( \varphi \)-OBDD for \( f \). Then, \( \mathcal{B} \) is reduced if and only if \( \text{size}(\mathcal{B}) \leq \text{size}(\mathcal{B}') \) for every \( \varphi \)-OBDD \( \mathcal{B}' \) for \( f \).

We now present reduction rules that allow us to transform any \( \varphi \)-OBDD into a \( \varphi \)-ROBDD for the same switching function. Hence, let \( \mathcal{B} = \langle V, V_I, V_T, \text{succ}_0, \text{succ}_1, \text{var}, \text{val}, v_0 \rangle \) be a \( \varphi \)-OBDD.

The elimination rule is as follows: if \( v \in V_I \) and \( \text{succ}_0(v) = \text{succ}_1(v) = w \) then remove \( v \) and transform all edges from a node \( u \) to \( v \) to edges from \( u \) to \( w \).
The isomorphism rule is as follows: if $v$ and $w$ are nodes in $B$ such that $v \neq w$ and either $v, w \in V_T$ with $\text{val}(v) = \text{val}(w)$, or $v, w \in V_I$ and $\text{var}(v) = \text{var}(w)$, $\text{succ}_0(v) = \text{succ}_0(w)$ and $\text{succ}_1(v) = \text{succ}_1(w)$ then remove $v$ and transform all edges from a node $u$ to $v$ to edges from $u$ to $w$.

We leave as an exercise to prove that the $\wp$-OBDD in Figure 6.14 can be obtained from the $\wp$-OBDD in Figure 6.12.

Both rules are sound in the sense that they do not affect the switching function that is being defined, that is, if $B_1$ was obtained from $B$ by some reduction rule then $B_1$ is still a $\wp$-OBDD and $f_{B_1} = f_B$. Observe that the rules just compact all the nodes that denote the same switching function, that is, the rules compact nodes $v, v'$ such that $f_v = f_{v'}$. Consider, for instance, the elimination rule. Let $v \in V$ be such that $\text{succ}_0(v) = \text{succ}_1(v) = v'$ and $\text{var}(v) = x$. Then,

$$f_v = (x \wedge f_{\text{succ}_1(v)}) \vee (\neg x \wedge f_{\text{succ}_0(v)})$$

$$= (x \wedge f_{v'}) \vee (\neg x \wedge f_{v'})$$

$$= (x \vee \neg x) \wedge f_{v'}$$

Furthermore, if $\text{succ}_0(u) = v$ then after the elimination of $v$ we will have $\text{succ}'_0(u) = v'$. Thus, the function determined by $u$ is not affected. Assume that $\text{succ}_1(u) = v$ and $\text{succ}_0(u) \neq v$. Then, after the elimination, we will have in $B_1$ $\text{succ}'_1(u) = v'$ and $\text{succ}'_0(u) = \text{succ}_0(u)$. Hence, for $x = \text{var}(u)$, we have

$$f_u = (x \wedge f_{\text{succ}_1(u)}) \vee (\neg x \wedge f_{\text{succ}_0(u)})$$

$$= (x \wedge f_{v'}) \vee (\neg x \wedge f_{\text{succ}_0(u)})$$

$$= (x \wedge f_{v'}) \vee (\neg x \wedge f_{\text{succ}_0(u)})$$

This means that any node with successor $v$ is not affected by the elimination rule, i.e. the function $f_u$ in $B$ and in $B_1$ is the same. A similar conclusion can be made for the isomorphism rule.

Next, we prove that the rule are complete.

**Theorem 6.38.** Let $B$ be a $\wp$-OBDD. Then, $B$ is reduced if and only if no reduction rule is applicable to $B$.

**Proof.** ($\rightarrow$): If we can apply a reduction rule to $B$ then this means that there are at least two nodes $v, v' \in V$ with $v \neq v'$ and such that $f_v = f_{v'}$. 

($\leftarrow$): If $B$ is reduced then there are no two nodes $v, v' \in V$ with $v \neq v'$ such that $f_v = f_{v'}$. 

Hence, no reduction rule is applicable to $B$. 

Therefore, $B$ is reduced if and only if no reduction rule is applicable to $B$. 


6.4. SYMBOLIC MODEL-CHECKING

But, then, this implies that $B$ is not reduced.

($\leftarrow$): Assume that $B$ is such that no rule is applicable. We start by considering the following sets of nodes:

- $V_i = \{ v \in V \mid x_i \leq \var(v) \} \cup V_T$, for $i = 1, \ldots, n$
- $V_{n+1} = V_T$.

That is $V_i$ is the set of nodes that are labelled with variables greater or equal to $x_i$. So, we have that $V_{n+1} \subseteq V_n \subseteq V_{n-1} \subseteq \cdots \subseteq V_1$. We now prove that

$$f_v \neq f_w,$$

for any nodes $v, w \in V_i$ such that $v \neq w$.

The proof follows by induction on $i$.

Base: The result holds for $V_{n+1}$. In this case we have that $V_{n+1} = V_T$. As we cannot apply the isomorphism rule this means that there is, at most, one terminal node labelled with 0 and one terminal node labelled with 1.

Hypothesis: The result holds for $V_i_{n+1}$.

Step: We prove the result for $V_i$. Assume that there are $v, w \in V_i$ such that $v \neq w$ and $f_v = f_w$. By the induction hypothesis, we can immediately conclude that we cannot have both $v, w \in V_{i+1}$. Hence, assume that $v \in V_i \setminus V_{i+1}$. This implies that $\var(v) = x_i$. We need to consider two cases:

- $w \in V_{i+1}$: then $w \in V_T$ or $w \in V_I$ and $\var(w) = x_j$ for some $x_j >_{\rho} x_i$.
  In either of these cases, we know that $x_i$ is not essential for $f_w$ and, as we are assuming that $f_v = f_w$, then $x_i$ is also not essential for $f_v$. Hence

$$f_v|_{x_i=0} = f_v|_{x_i=1}.$$  

But, as $f_v|_{x_i=0} = f_{\text{succ}_0(v)}$ and $f_v|_{x_i=1} = f_{\text{succ}_1(v)}$, then, we have that

$$f_{\text{succ}_0(v)} = f_{\text{succ}_1(v)}.$$  

We also know that $\var(\text{succ}_0(v)), \var(\text{succ}_1(v)) \in V_{i+1}$. Hence, by induction hypothesis, it must be the case that $\text{succ}_0(v) = \text{succ}_1(v)$. But then the elimination rule would be applicable to $v$, contradiction our initial assumption.
• $w \in V_i \setminus V_{i+1}$: then $\text{var}(w) = x_i$. As we are assuming that $f_v = f_w$, then

$$f_{\text{succ}_0(v)} = f_v|_{x_i=0} = f_w|_{x_i=0} = f_{\text{succ}_0(w)}.$$  

Furthermore, as $\text{var}(\text{succ}_0(v)), \text{var}(\text{succ}_0(w)) \in V_{i+1}$, then, by induction hypothesis, it follows that $\text{succ}_0(v) = \text{succ}_0(w)$. By a similar argument, we are also able to conclude that $\text{succ}_1(v) = \text{succ}_1(w)$. But, in this case, the isomorphism rule would be applicable to $v$, contradicting our initial assumption.

Hence, we can conclude that $f_v \neq f_w$ for any $v, w \in V_i$ such that $v \neq w$. And, as $V_1 = \text{Var}$, then we can conclude that $\mathcal{B}$ is indeed a $\varphi$-ROBDD.

\[\square\]
Chapter 7

Fairness

Fairness is an important requirement of a reactive system. Consider the example of two processes trying to access the critical section but only one of them is entering. This strategy is unfair for one of the processes because the other will have to wait infinitely long before entering the critical section.

Dar o exemplo com dois semáforos independentes.

There are several types of fairness constraints. These fairness constraints are used to rule out undesired behavior.

- Unconditional fairness – every process gets its turn infinitely often.
- Strong fairness – every process that is enabled infinitely often gets its turn infinitely often.
- Weak fairness – every process that is continuously enabled from a certain time instant gets its turn infinitely often.

By enabled, we mean that the process is ready to execute (a transition). Some more motivation

7.1 Fairness in linear time properties

Let \( T = \langle S, A, \rightarrow, I, \Xi, L \rangle \) be a transition system and let \( s \in S \). Then, we denote by \( \text{Act}(s) \) the set of actions that can be executed from state \( s \)

\[
\text{Act}(s) = \{ a \in \text{Act} \mid \exists s' \in S. s \xrightarrow{a} s' \}.
\]

193
Definition 7.1. Let $T = \langle S, A, \rightarrow, I, \Xi, L \rangle$ be a transition system without terminal states, $FA \subseteq A$, and $\rho = s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_2} \ldots$ be an infinite path fragment of $T$. Then

- $\rho$ is unconditional $FA$-fair whenever $\exists^\infty j.a_j \in FA$.
- $\rho$ is strongly $FA$-fair whenever $\exists^\infty j.\text{Act}(s_j) \cap FA \neq \emptyset$ implies $\exists^\infty j.a_j \in FA$.
- $\rho$ is weakly $FA$-fair whenever $\forall^\infty j.\text{Act}(s_j) \cap FA \neq \emptyset$ implies $\exists^\infty j.a_j \in FA$.

By $\exists^\infty j$ we mean “there are infinitely many $j$” and by $\forall^\infty j$ we mean “nearly all $j$”, in the sense of “except for finitely many $j$”.

Explain the different notions.

There is a relation between the different types of fairness:

unconditional $FA$-fairness $\implies$ strong $FA$-fairness $\implies$ weak $FA$-fairness.

But the converse does not hold, in general.

Definition 7.2. A fairness assumption for $A$ is a triple

$$\mathcal{F} = \langle \mathcal{F}_u, \mathcal{F}_s, \mathcal{F}_w \rangle$$

with $\mathcal{F}_u, \mathcal{F}_s, \mathcal{F}_w \subseteq 2^A$. An execution $\rho$ is $\mathcal{F}$-fair if

- it is unconditionally fair $FA$-fair for every $FA \in \mathcal{F}_u$;
- it is strongly fair $FA$-fair for every $FA \in \mathcal{F}_s$;
- it is weakly fair $FA$-fair for every $FA \in \mathcal{F}_w$.

The notion of $\mathcal{F}$-fair execution can be lifted to traces and paths. An infinite trace $\sigma$ is $\mathcal{F}$-fair if there is an $\mathcal{F}$-fair execution $\rho$ such that $\text{trace}(\rho) = \sigma$. $\mathcal{F}$-fair path fragments and paths are defined in a similar way.

Let $\text{FairPaths}_\mathcal{F}(s)$ denote the set of $\mathcal{F}$-fair paths of $s$ and $\text{FairPaths}_\mathcal{F}(T)$ denote the set of $\mathcal{F}$-fair paths of $T$. Additionally, let $\text{FairTraces}_\mathcal{F}(s)$ denote the set of $\mathcal{F}$-fair traces of $s$ and $\text{FairTraces}_\mathcal{F}(T)$ denote the set of $\mathcal{F}$-fair traces of $T$. 
7.2.  FAIRNESS IN LTL

Example 7.3. Exemplo...

Definition 7.4. Let $P$ be an LT property over $\Xi$ and $F$ be a fairness assumption for $A$. Transition system $T = (S, A, \rightarrow, I, \Xi, L)$ fairly satisfies $P$, written $T \vDash_F P$, if $\text{FairTraces}_F(T) \subseteq P$.

Say something about the relationship between the different notions of fairness.

Example 7.5. aaa

Definition 7.6. Let $T = (S, A, \rightarrow, I, \Xi, L)$ be a transition system and $F$ a fairness assumption for $A$. Then, $F$ is said realizable for $T$ if for every reachable state $s$, we have $\text{FairPaths}_F(s) \neq \emptyset$.

Realizable fairness assumptions are irrelevant for safety properties.

Theorem 7.7. Let $T = (S, A, \rightarrow, I, \Xi, L)$ be a transition system, $F$ a realizable fairness assumption for $T$ and $P_{\text{safe}}$ a safety property over $\Xi$. Then

$$T \vDash P_{\text{safe}} \quad \text{if and only if} \quad T \vDash_F P_{\text{safe}}$$

Proof. 

Note that this does not hold in general. It may happen, for fairness assumptions that are not realizable, that certain safety properties are violated.

7.2  Fairness in LTL

Definition 7.8. Let $\varphi$ and $\psi$ be LTL formulas over $\Xi$.

- An unconditional LTL fairness constraint is an LTL formula $ufair$ of the form
  $$G_0 F_0 \varphi.$$

- A strong LTL fairness constraint is an LTL formula $ustrong$ of the form
  $$G_0 F_0 \varphi \rightarrow G_0 F_0 \psi.$$
• A weak LTL fairness constraint is an LTL formula \( u_{\text{weak}} \) of the form
\[
F_0 \ G_0 \ \varphi \rightarrow \ G_0 \ F_0 \ \psi.
\]
An LTL fairness assumption is a conjunction of LTL fairness constraints.

In the sequel, we adopt an notation similar to the one adopted for transition systems. Let \( sfair \) be a formula
\[
\begin{align*}
\text{FairPaths}(s) &= \{ \pi \in \text{Paths}(s) \mid \pi \vDash sfair \} \\
\text{FairTraces}(s) &= \{ \text{trace}(\pi) \mid \pi \in \text{FairPaths}(s) \}
\end{align*}
\]
These extend to transition systems. Hence, we will have \( \text{FairPaths}(T) \) and \( \text{FairTraces}(T) \).

**Definition 7.9.** Let \( T = (S, A, \rightarrow, I, \Xi, L) \) be a transition system with no terminal states, \( s \in S \), \( \varphi \) be an LTL formula and \( sfair \) be a an LTL fairness assumption. Then
\[
\begin{align*}
\bullet \ s &\vDash_{\text{fair}} \varphi \quad \text{if} \quad \forall \pi \in \text{FairPaths}(s), \pi \vDash \varphi; \\
\bullet \ T &\vDash_{\text{fair}} \varphi \quad \text{if} \quad \forall \pi \in \text{FairPaths}(T), \pi \vDash \varphi.
\end{align*}
\]
The second condition could be rewritten as
\[
T \vDash_{\text{fair}} \varphi \quad \text{if} \quad \forall s_0 \in I, s_0 \vDash \varphi.
\]

**Example 7.10.** TBW

**Theorem 7.11.** Let \( T = (S, A, \rightarrow, I, \Xi, L) \) be a transition system with no terminal states, \( \varphi \) be an LTL formula and \( sfair \) be a an LTL fairness assumption. Then
\[
T \vDash_{\text{fair}} \varphi \quad \text{if and only if} \quad T \vDash (sfair \rightarrow \varphi).
\]

Say something about efficiency issues of this reduction (see footnote on page 358).

Write something about LTL model checking with fairness.

Relate the state based approach with the action based approach.

### 7.3 Fairness in CTL

An approach similar to the one adopted for LTL is not possible for CTL. We start by redefining the semantics of CTL formula.
Chapter 8

Partial order reduction

Write the motivation for partial order reduction...

In this chapter $T = \langle S, A, \rightarrow, I, \Xi, L \rangle$ is assumed to be a finite transition system without terminal states. Say something about concurrent systems... It is also assumed to be action deterministic

We denote let $\text{Act}(s) = \{ a \in A \mid \exists s'. s \xrightarrow{a} s' \}$. This set is composed of all the actions that are enabled in state $s$. As we are assuming that $T$ is action-deterministic then for each $a \in \text{Act}(s)$ there is a single state $s'$ such that $s \xrightarrow{a} s'$. We denote this successor state by $a(s)$. We can extend this notion to sequences of actions. Let $a_1 \ldots a_n$ be a sequence of actions and a state $s_0$ of $T$ such that $a_i \in \text{Act}(s_{i-1})$ and $s_i = a_i(s_{i-1})$, for $1 < i \leq n$. Then $(a_1 \ldots a_n)(s_0)$ denotes the state $s_n$, i.e. the state reached from $s_0$ by the execution of action $a_1 \ldots a_n$.

Example...

Two action are said independent if the order in which they are executed does not affect the final result. This means that if two actions $a_1$ and $a_2$ are enabled in a state $s$ then the execution of $a_1$ cannot disable $a_2$ and vice-versa. Furthermore, the sequences $a_1a_2$ and $a_2a_1$, if execute from $s$, yield the same state.

**Definition 8.1.** Let $a_1, a_2 \in \text{Act}$ with $a_1 \neq a_2$. Then, $a_1$ are $a_2$ are said to be *independent* in $T$ if, for any $s \in S$ with $a_1, a_2 \in \text{Act}(s)$, the following conditions hold:

- $a_1 \in \text{Act}(a_2(s))$ and $a_2 \in \text{Act}(a_1(s))$;
• \((a_1 a_2)(s) = (a_2 a_1)(s)\).

The actions \(a_1\) and \(a_2\) are said dependent in \(T\) if they are not independent in \(T\). Given a set \(A \subseteq \text{Act}\) and an action \(a \in \text{Act} \setminus A\), \(a_1\) is said to be independent of \(A\) in \(T\) if \(a\) and \(a'\) are independent in \(T\), for every \(a' \in A\). Otherwise, \(a\) is said dependent of \(A\).

**Example 8.2.** Parallel composition and handshaking

**Example 8.3.** Program graphs and mutual exclusion.

**Lemma 8.4.** Let \(T\) be an action-deterministic transition system, \(s\) a state in \(T\) and let

\[
s = s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} s_2 \xrightarrow{a_3} \ldots \xrightarrow{a_n} s_n
\]

be an execution fragment in \(T\) from \(s\). Let \(a \in \text{Act}\) be an action, independent of \(\{a_1, \ldots, a_n\}\). Then \(a \in \text{Act}(s_i)\) and

\[
s = s_0 \xrightarrow{a} t_0 \xrightarrow{a_1} t_1 \xrightarrow{a_2} t_2 \xrightarrow{a_3} \ldots \xrightarrow{a_n} t_n
\]

is an execution fragment in \(T\) and is such that \(t_i = a(s_i)\), for \(i = 0, \ldots, n\).

**Proof.** TBW

**Lemma 8.5.** Let \(T\) be an action-deterministic transition system, \(s\) a state in \(T\) and let

\[
s = s_0 \xrightarrow{a} t_0 \xrightarrow{a_1} t_1 \xrightarrow{a_2} t_2 \xrightarrow{a_3} \ldots
\]

be an infinite execution fragment from \(s\). Let \(a \in \text{Act}\) be an action, independent of \(\{a_1, a_2, \ldots\}\). Then, \(a \in \text{Act}(s_i)\) and

\[
s = s_0 \xrightarrow{a} t_0 \xrightarrow{a_1} t_1 \xrightarrow{a_2} t_2 \xrightarrow{a_3} \ldots
\]

is an infinite execution fragment in \(T\) and is such that \(t_i = a(s_i)\), for \(i \geq 0\).
Part II

Solutions to selected exercises
Chapter 1

Preliminary concepts

1.1 Exercises

Exercise 1.1. Let $v : \Xi \rightarrow \{0, 1\}$ be a valuation over $\Xi$. Show that

(a) $v \models \varphi_1 \rightarrow \varphi_2$ if $v \not\models \varphi_1$ or $v \models \varphi_2$.

(b) $v \models \varphi_1 \leftrightarrow \varphi_2$ if $v \models \varphi_1$ and $v \models \varphi_2$, or $v \not\models \varphi_1$ and $v \not\models \varphi_2$.

(c) $v \models \varphi_1 \oplus \varphi_2$ if $v \models \varphi_1$ and $v \not\models \varphi_2$, or $v \not\models \varphi_1$ and $v \models \varphi_2$.

Exercise 1.2. Let $\Xi$ be a set of propositional symbols. For each of the following formulas, say whether they are satisfiable, valid or contradictory:

(a) $p \rightarrow (q \rightarrow p)$

(b) $(p \land q) \rightarrow (p \rightarrow q)$

(c) $(p \rightarrow q) \rightarrow (p \land q)$

(d) $(p \land (\neg q)) \leftrightarrow ((\neg p) \lor q)$

(e) $(p \rightarrow (q \rightarrow r)) \rightarrow ((p \rightarrow q) \rightarrow (p \rightarrow r))$

Exercise 1.3. Let $v : \Xi \rightarrow \{0, 1\}$ be a valuation over $\Xi$. Show that if $v \not\models \neg \varphi$ then $v \not\models \varphi$.

Exercise 1.4. Prove Lemma 1.2.
Exercise 1.5. Conclude the proof of Proposition 1.3.

Exercise 1.6. Prove Proposition 1.4.

Exercise 1.7. Prove Proposition 1.5.

Exercise 1.8. Let $P$ be a finite set of propositional symbols and $q \in \Xi$. Show that the following formulas are tautologies:

1. \[
\bigvee_{A \subseteq \Xi} \left( \bigwedge_{p \in A} p \right) \land \left( \bigwedge_{p \in \Xi \setminus A} \neg p \right)
\]

2. \[
q \leftrightarrow \bigvee_{A \subseteq \Xi, q \in A} \left( \bigwedge_{p \in A} p \right) \land \left( \bigwedge_{p \in \Xi \setminus A} \neg p \right)
\]

Exercise 1.9. Let $\Sigma$ be the set of ASCII characters. Show that the following sets are regular languages:

(a) $L_1$ is the set of all words over $\Sigma$ representing positive integers.

(b) $L_2$ is the set of all words over $\Sigma$ representing all even numbers.

(c) $L_3$ is the set of all words over $\Sigma$ representing all numbers, in decimal notation.

(d) $L_4$ is the set of all words over $\Sigma$ representing all IP addresses (in DOT-decimal notation). Recall that an IP address in DOT-decimal notation is a sequence of the form $a_1.a_2.a_3.a_4$ where each $a_i$, with $i = 1, \ldots, 4$, is an integer between 0 and 255.

Exercise 1.10. Let $\Sigma = \{0, 1\}$. Show that the following sets are regular languages:

(a) $L_1$ is the set of all words over $\Sigma$ that start in 1 and end in 0.

(b) $L_2$ is the set of all words over $\Sigma$ that start and end in 0.

(c) $L_3$ is the set of all words over $\Sigma$ that have an even number of 1’s.

Exercise 1.11. Let $\Sigma$ be an alphabet and $E$ a regular expression over $\Sigma$. Is $L(E)^+$ a regular language? Justify.
Exercise 1.12. Consider the alphabet $\Sigma = \{0, 1\}$. Show that the following sets are regular $\omega$-languages

(a) $L_\omega$ is the set of all infinite words over $\Sigma$ that contain only finitely many 0’s;

(b) $L_\omega$ is the set of all infinite words over $\Sigma$ that contain infinitely many 1’s;

(c) $L_\omega$ is the set of all infinite words over $\Sigma$ that contain the sequence 010 infinitely often;

(d) $L_\omega$ is the set of all infinite words over $\Sigma$ that contain the sequence 11 infinitely often but contain the sequence 00 only finitely often.

Exercise 1.13. Prove or disprove the following equivalences for $\omega$-regular expressions:

(a) $(E_1 + E_2).F^\omega = E_1.F^\omega + E_2.F^\omega$

(b) $E.(F_1 + F_2)^\omega = E.F_1^\omega + E.F_2^\omega$

(c) $E.(F.F^*)^\omega = E.F^\omega$

(d) $(E^*.F)^\omega = E^*.F^\omega$

where $E, E_1, E_2, F, F_1, F_2$ are regular expressions such that $\varepsilon \not\in \mathcal{L}(F), \mathcal{L}(F_1), \mathcal{L}(F_2)$. 
1.2 Solutions to selected exercises

Exercise 1.1.
(a) Recall the abbreviation for implication:

\[ \varphi_1 \rightarrow \varphi_2 \equiv_{def} (\neg \varphi_1) \lor \varphi_2. \]

Then,

\[ v \models \varphi_1 \rightarrow \varphi_2 \iff v \not\models (\neg \varphi_1) \lor \varphi_2 \]
\[ \iff v \not\models \neg \varphi_1 \text{ or } v \models \varphi_2 \]
\[ \iff v \not\models \varphi_1 \text{ or } v \models \varphi_2. \]

Exercise 1.2.
(a) Let \( v \) be a valuation and assume that \( v \not\models p \rightarrow (q \rightarrow p) \). Then,

\[ v \not\models p \rightarrow (q \rightarrow p) \iff v \not\models p \text{ and } v \not\models q \rightarrow p \]
\[ \iff v \not\models p \text{ and } v \not\models q \text{ and } v \not\models p \]
\[ \iff v(p) = 1 \text{ and } v(q) = 1 \text{ and } v(p) = 0. \]

This last condition is clearly impossible because \( v \) is a function and consequently we cannot have \( v(p) = 1 \) and \( v(p) = 0 \). Hence, there is no valuation that does not satisfy the formula and so the formula is valid.

(c) Let \( v \) be a valuation and assume that \( v \not\models (p \rightarrow q) \rightarrow (p \land q) \). Then,

\[ v \not\models (p \rightarrow q) \rightarrow (p \land q) \iff v \not\models p \rightarrow q \text{ and } v \not\models p \land q \]
\[ \iff v \not\models p \text{ or } v \not\models q, \text{ and } v \not\models p \text{ or } v \not\models q \]
\[ \iff v(p) = 0 \text{ or } v(q) = 1, \text{ and } v(p) = 0 \text{ or } v(q) = 0. \]

Choosing \( v \) such that \( v(p) = 0 \) we can conclude that \( v \not\models (p \rightarrow q) \rightarrow (p \land q) \) and so the formula is not valid. However, the formula is satisfiable. In fact, let \( v' \) be a valuation such that \( v'(p) = v'(q) = 1 \). Observe that this valuation does not meet the previous requirements. In this case, we have

\[ v' \models (p \rightarrow q) \rightarrow (p \land q) \iff v' \not\models p \rightarrow q \text{ or } v' \not\models p \land q \]
\[ \iff v' \models p \text{ and } v' \not\models q, \text{ or } v' \models p \text{ and } v' \not\models q \]
\[ \iff v'(p) = 1 \text{ and } v'(q) = 1, \text{ or } v'(p) = 1 \text{ and } v'(q) = 1. \]

The valuation \( v' \) meets the second condition and so we may conclude that \( v' \models (p \rightarrow q) \rightarrow (p \land q) \) and thus the formula is satisfiable.
Exercise 1.8.
1. Let $v$ be a valuation and, for this valuation, define the set $A_v = \{ p \in \Xi \mid v(p) = 1 \}$. From the definition of $A_v$ it follows that $v \models p$ for every $p \in A_v$ and, consequently,

$$v \models \left( \bigwedge_{p \in A_v} p \right).$$

Similarly, it also follows that $v \not\models p$ for every $p \in \Xi \setminus A_v$, that is, $v \models \neg p$ for every $p \in \Xi \setminus A_v$ and, consequently,

$$v \models \left( \bigwedge_{p \in \Xi \setminus A_v} \neg p \right).$$

Hence,

$$v \models \left( \bigwedge_{p \in A_v} p \right) \land \left( \bigwedge_{p \in \Xi \setminus A_v} \neg p \right)$$

and, as $A_v \subseteq \Xi$, then

$$v \models \bigvee_{A \subseteq \Xi} \left( \left( \bigwedge_{p \in A_v} p \right) \land \left( \bigwedge_{p \in \Xi \setminus A} \neg p \right) \right).$$

Since $v$ was arbitrary it follows that the formula is valid.

Exercise 1.9.
In order to show that a language is regular, we need to find a regular expression that denotes that language.

(a) The set of positive integers is denoted by the regular expression

$$(1 + 2 + \cdots + 9)(0 + 1 + \cdots + 9)^*$$

(d) We start by defining a regular expression $D$ for denoting the integers between 0 and 255:

$$\underbrace{(0 + \cdots + 9)}_{0.9} + \underbrace{(1 + \cdots + 9)(0 + \cdots + 9)}_{10.9} + \underbrace{1(0 + \cdots + 9)(0 + \cdots + 9)}_{100..199} + \underbrace{2(0 + \cdots + 4)(0 + \cdots + 9)}_{200..249} + \underbrace{25(0 + \cdots + 5)}_{250..255}$$
which can be simplified to

\[(\varepsilon + (1 + \cdots + 9) + 1(0 + \cdots + 9) + 2(0 + \cdots + 4))(0 + \cdots + 9) + 25(0 + \cdots + 5)].\]

Then, the regular expression for denoting the language of IP addresses in DOT-decimal notation is

\[D.D.D.D\]

where the symbol ‘.’ is the ASCII character for the decimal point and not the concatenation symbol.

**Exercise 1.12.**

(a) If there is only a finite number of 0’s then there is a finite prefix composed of 0’s and 1’s and then and infinite suffix composed only of 1’s. An \(\omega\)-regular expression that denotes such a language is, for instance,

\[(0 + 1)^* 1^\omega\]

It is not very difficult to see that this is in fact an \(\omega\)-regular expression. Furthermore, the regular expression \((0 + 1)^*\) denotes the finite prefixes containing 0’s and 1’s and \(1^\omega\) denotes the infinite word composed only of 1’s.

**Exercise 1.13.**

(a) The assertion is true. Let

\[\begin{align*}
A &= \mathcal{L}_\omega((E_1 + E_2).F^\omega) \\
   &= \mathcal{L}(E_1 + E_2).\mathcal{L}_\omega(F^\omega) \\
   &= (\mathcal{L}(E_1) \cup \mathcal{L}(E_2)).\mathcal{L}(F)^\omega.
\end{align*}\]

and

\[\begin{align*}
B &= \mathcal{L}_\omega(E_1.F^\omega + E_2.F^\omega) \\
   &= \mathcal{L}_\omega(E_1.F^\omega) \cup \mathcal{L}_\omega(E_2.F^\omega) \\
   &= \mathcal{L}(E_1).\mathcal{L}_\omega(F^\omega) \cup \mathcal{L}(E_2).\mathcal{L}_\omega(F^\omega) \\
   &= \mathcal{L}(E_1).\mathcal{L}(F)^\omega \cup \mathcal{L}(E_2).\mathcal{L}(F)^\omega.
\end{align*}\]

Let \(\sigma \in A\). Then, \(\sigma = w.\sigma'\) where \(w \in \mathcal{L}(E_1) \cup \mathcal{L}(E_2)\) and \(\sigma' \in \mathcal{L}(F)^\omega\). Thus, either \(w \in \mathcal{L}(E_1)\) or \(w \in \mathcal{L}(E_2)\) which implies that either \(w.\sigma' \in \mathcal{L}(E_1).\mathcal{L}(F)^\omega\) or \(w.\sigma' \in \mathcal{L}(E_2).\mathcal{L}(F)^\omega\). And so, \(\sigma \in B\), that is, \(A \subseteq B\). The converse is similar.
(b) The assertion is false. Observe that $L_\omega(E.(F_1 + F_2)\omega)$ contains in all the infinite words of the form $e.f^1.f^2.f^3\ldots$ such that $e \in L(E)$ and each $f^i \in L(F_1) \cup L(F_2)$, for $i = 1, \ldots$. It contains, in particular, the infinite word

$$\sigma = e.f_1.f_2.f_1.f_2, \ldots$$

for some $f_1 \in L(F_1)$ and $f_2 \in L(F_2)$. The language $L_\omega(E.F_1\omega)$ contains all the infinite words of the form $e.f^1.f^2.f^3\ldots$ such that $e \in L(E)$ and each $f^i \in L(F_1)$, for $i = 1, \ldots$. Hence, in particular, $\sigma \notin L_\omega(E.F_1\omega)$. By a similar argument, we can also conclude that $\sigma \notin L_\omega(E.F_2\omega)$. So, $L_\omega(E.(F_1 + F_2)\omega) \not\subseteq L_\omega(E.F_1\omega + E.F_2\omega)$ which implies that the two $\omega$-regular expressions are not equivalent.
Chapter 2

Concurrent systems

2.1 Exercises

Exercise 2.1. Consider a traffic light defined by the transition system depicted below.

\[ \text{Traffic Light:} \]

\[ \begin{array}{c}
\text{red} \\
\tau \\
\tau \\
\text{green}
\end{array} \]

Define the transition system for the parallel composition of two traffic lights, by interleaving.

Exercise 2.2. Let \( \text{Var}_1 \) and \( \text{Var}_2 \) be two sets of variables and \( y \) a variable with binary domain such that \( y \in \text{Var}_1 \cap \text{Var}_2 \). Consider the two program graphs \( PG_i = (Loc_i, Act_i, Effect_i, \rightarrow, Loc_0, g_0) \) over \( \text{Var}_i \), with \( i = 1, 2 \), such that

- \( \text{Loc}_i = \{ \text{noncrit}_i, \text{wait}_i, \text{crit}_i \} \);
- \( \text{Act}_i = \{ \tau, y := y - 1, y := y + 1 \} \);
- \( \text{Effect}_i \) is such that
•  $\text{Loc}_0 = \{\text{noncrit}_i\}$

•  $g_0$ is $y = 1$.

The conditional transition relation is depicted in the following diagram.

Show that when two processes represented by these two program graphs are put in parallel they cannot both be in location crit$_1$ and crit$_2$ simultaneously. Suggestion: Start by constructing the program graph $PG = PG_1 \parallel PG_2$. Then, define the transition system $T(PG)$ and show that every global state of the form $(\langle \text{crit}_1, \text{crit}_2 \rangle, \eta)$ is not reachable.

**Exercise 2.3.** Let $\text{Var}_1$ and $\text{Var}_2$ be two sets of variables, and $y_1$ and $y_2$ be two variables with integer domain such that $y_1, y_2 \in \text{Var}_1 \cap \text{Var}_2$. Consider the two program graphs $PG_i = \langle \text{Loc}_i, \text{Act}_i, \text{Effect}_i, \rightarrow, \text{Loc}_0, g_0 \rangle$ over $\text{Var}_i$, with $i = 1, 2$, such that

•  $\text{Loc}_i = \{\text{noncrit}_i, \text{wait}_i, \text{crit}_i\}$;

•  $\text{Act}_1 = \{\tau, y_1 := y_2 + 1, y_1 := 0\}$ and $\text{Act}_2 = \{\tau, y_2 := y_1 + 1, y_2 := 0\}$;

•  $\text{Effect}_i$ is such that

  - $\text{Effect}_1(y_1 := y_2 + 1, \eta) = \eta[y_1 := y_2 + 1]$
  - $\text{Effect}_1(y_1 := 0, \eta) = \eta[y_1 := 0]$
  - $\text{Effect}_2(y_2 := y_1 + 1, \eta) = \eta[y_2 := y_1 + 1]$
  - $\text{Effect}_2(y_2 := 0, \eta) = \eta[y_2 := 0]$
2.1. EXERCISES

- \( \text{Loc}_0 = \{ \text{noncrit}_i \} \)
- \( g_0 \) is \( y_1 = 0 \land y_2 = 0 \).

The conditional transition relation for \( PG_1 \) is depicted in the following diagram. For \( PG_2 \), the transition relation is similar.

(a) Define \( PG = PG_1 \parallel PG_2 \).
(b) Define the reachable part of the transition system of \( PG \) where \( y_1 \leq 2 \) and \( y_2 \leq 2 \).
(c) Describe an execution that shows that the entire transition system is infinite.
(d) Check if the transition system does indeed ensure mutual exclusion.
(e) Check if the transition system reaches a state in which both processes are mutually waiting for each other.
(f) Is it possible that a process wanting to enter its critical section has to wait indefinitely?

Exercise 2.4. Consider two processes described by the transition systems depicted in the figure below, with \( i=1,2 \).

\( T_i \):

\[ \begin{align*}
\text{nc}_i & \quad \text{rel} \\
\text{req} & \quad \text{c}_i
\end{align*} \]
(a) Define $T_1 \parallel T_2$ and show that mutual exclusion is not ensured.

(b) Consider the process $T_{Sem}$ defined by the following transition system

\[
\begin{align*}
\text{free} & \quad \text{rel} \quad \text{req} \quad \text{busy}
\end{align*}
\]

and let $H = \{\text{req, rel}\}$ be a set of handshake actions. Show that the system $(T_1 \parallel T_2) \parallel H T_{Sem}$ guarantees mutual exclusion between $T_1$ and $T_2$.

**Exercise 2.5.** Consider a street junction $J$ as depicted in the following figure. At that junction, each street is controlled by a traffic light $T_i$ whose behavior is defined as described by the transition system depicted below.

\[
\begin{align*}
S_1 & \quad J \quad S_2 \\
S_2 & \quad J \quad S_3 \\
T_i: & \quad \text{red} \\
\text{green} & \quad \text{yellow}
\end{align*}
\]

(a) Is $T_1 \parallel T_2 \parallel T_3$ a good solution for controlling the traffic at junction $J$?

(b) Define a controller $C_1$ in order to ensure that there is, at most, one green signal lamp on at each time. Start by choosing appropriate names for
2.1. EXERCISES

the actions. Then, choose an adequate communication mechanism and outline the transition system \((T_1 \parallel T_2 \parallel T_3) \parallel C_1\).

(c) Repeat the previous exercise, but defining an controller \(C_2\) that switches the green signal lamp in the following order: \(T_1, T_2, T_3, T_1, T_2, T_3, \ldots\)

**Exercise 2.6.** Let \(T_1, T_2, T_3\) be transition systems. Show the following equalities:

(a) \(T_1 \parallel T_2 = T_2 \parallel T_1\).

(b) \(T_1 \parallel (T_2 \parallel T_3) = (T_1 \parallel T_2) \parallel T_3\).
2.2 Solutions to selected exercises

Exercise 2.1.

Let $T_1 = \langle S_1, A_1, \rightarrow_1, I_1, \Xi_1, L_1 \rangle$ be the transition system for the first traffic light and $T_2 = \langle S_2, A_2, \rightarrow_2, I_2, \Xi_2, L_2 \rangle$ be the transition system for the second traffic light. These transition systems are as follows, for $i = 1, 2$:

- $S_i = \{\text{red, green}\}$;
- $A_i = \{\tau\}$;
- $\rightarrow_i$ is defined by the diagram;
- $I_i = \{\text{red}\}$;
- $\Xi_i = \{\text{red}_i, \text{green}_i\}$;
- $L_i(\text{red}) = \{\text{red}_i\}$ and $L_i(\text{green}) = \{\text{green}_i\}$.

Note that we need to distinguish the propositional symbols of $T_1$ and $T_2$, so that we can refer to situation where traffic light 1 is green or traffic light 2 is red.

Then, $T_1 \parallel T_2 = \langle S, A, \rightarrow, I, \Xi, L \rangle$ is as follows:

- $S = S_1 \times S_2 = \{\langle \text{red, red}\rangle, \langle \text{red, green}\rangle, \langle \text{green, red}\rangle, \langle \text{green, green}\rangle\}$
- $A = A_1 \cup A_2 = \{\tau\}$;
- $\rightarrow$ is such that
  - as red$\rightarrow_1$green then
    * $\langle \text{red, red}\rangle \rightarrow \langle \text{green, red}\rangle$
    * $\langle \text{red, green}\rangle \rightarrow \langle \text{green, green}\rangle$
  - as green$\rightarrow_1$red then
    * $\langle \text{green, red}\rangle \rightarrow \langle \text{red, red}\rangle$
    * $\langle \text{green, green}\rangle \rightarrow \langle \text{red, green}\rangle$
  - as red$\rightarrow_2$green then
    * $\langle \text{red, red}\rangle \rightarrow \langle \text{red, green}\rangle$
    * $\langle \text{green, red}\rangle \rightarrow \langle \text{green, green}\rangle$
2.2. SOLUTIONS TO SELECTED EXERCISES

– as $\text{green} \rightarrow_2 \text{red}$ then

\[ \ast \langle \text{red}, \text{green} \rangle \rightarrow \langle \text{red}, \text{red} \rangle \]
\[ \ast \langle \text{green}, \text{green} \rangle \rightarrow \langle \text{green}, \text{red} \rangle \]

\[ I = I_1 \times I_2 = \{ \langle \text{red}, \text{red} \rangle \}; \]

\[ \Xi = \Xi_1 \cup \Xi_2 = \{ \text{red}_1, \text{green}_1, \text{red}_2, \text{green}_2 \}; \]

\[ L \text{ is such that} \]

\begin{itemize}
  \item $L(\langle \text{red}, \text{red} \rangle) = L_1(\text{red}) \cup L_2(\text{red}) = \{ \text{red}_1, \text{red}_2 \}$,
  \item $L(\langle \text{red}, \text{green} \rangle) = L_1(\text{red}) \cup L_2(\text{green}) = \{ \text{red}_1, \text{green}_2 \}$,
  \item $L(\langle \text{green}, \text{red} \rangle) = L_1(\text{green}) \cup L_2(\text{red}) = \{ \text{green}_1, \text{red}_2 \}$,
  \item $L(\langle \text{green}, \text{green} \rangle) = L_1(\text{green}) \cup L_2(\text{green}) = \{ \text{green}_1, \text{green}_2 \}$
\end{itemize}

The transition system for $T_1 \parallel T_2$ is depicted in the following figure.

Exercise 2.2.

As suggested, we start by defining the program graph for the interleaving of $PG_1$ and $PG_2$. For the sake of simplicity, we write $c_i$ for crit$_i$, nc$_i$ for noncrit$_i$ and $w_i$ for wait$_i$. The program graph $PG = PG_1 \parallel PG_2 = (\text{Loc}, \text{Act}, \text{Effect}, \rightarrow, \text{Loc}_0, g_0)$ is defined as follows:

\begin{itemize}
  \item $\text{Loc} = \{ \langle \text{nc}_1, \text{nc}_2 \rangle, \langle \text{w}_1, \text{nc}_2 \rangle, \langle \text{c}_1, \text{nc}_2 \rangle, \langle \text{nc}_1, \text{w}_2 \rangle, \langle \text{c}_1, \text{w}_2 \rangle, \langle \text{c}_1, \text{c}_2 \rangle, \langle \text{nc}_1, \text{c}_2 \rangle, \langle \text{w}_1, \text{c}_2 \rangle, \langle \text{c}_1, \text{c}_2 \rangle \};$
  \item $\text{Act} = \text{Act}_1 \uplus \text{Act}_2 = \{ \tau_1, \tau_2, (y := y - 1)_1, (y := y + 1)_1, (y := y - 1)_2, (y := y + 1)_2 \};$
\end{itemize}
• **Effect** is such that
  - \( \text{Effect}((y := y - 1)_i, \eta) = \eta[y := y - 1]; \)
  - \( \text{Effect}((y := y + 1)_i, \eta) = \eta[y := y + 1]; \)

• \( \text{Loc}_0 = \text{Loc}_{01} \times \text{Loc}_{02} = \{\langle nc_1, nc_2 \rangle\}; \)

• \( g_0 = g_{01} \land g_{02} \) is \( y = 1. \)

where \( (a)_1 \in \text{Act}_1 \cup \text{Act}_2 \) is the action \( a \in \text{Act}_i \) in the disjoint union. The conditional transition relation is depicted in the following figure. In order to avoid further overload of the figure, we omit the subscript of the actions.

We now define the transition system of \( PG_1 \parallel PG_2: \)
\[
T(PG_1 \parallel PG_2) = (S, \text{Act}, \rightarrow, I, \Xi, L).
\]
Recall that \( S = \text{Loc} \times \text{Asg}(\text{Var}), \) which is a very large set. So, in order to get a clear picture of the transition system, we will start by considering the initial states and unfold the transitions starting from that state. The set of initial states is
2.2. SOLUTIONS TO SELECTED EXERCISES

• \( I = \{ (l, \eta) \mid l \in \text{Loc}_0 \text{ and } \eta \models g_0 \} = \{ (\langle nc_1, nc_2 \rangle, \eta) \mid \eta(y) = 1 \} \).

We will denote these states by \( \langle nc_1, nc_2, y = 1 \rangle \). The reachable part of \( T(\text{PG}_1 \parallel \text{PG}_2) \) is depicted below. For simplicity, we omit the action labels and briefly explain some of the transitions.

The transition 1 from \( \langle nc_1, nc_2, y = 1 \rangle \) to \( \langle w_1, nc_2, y = 1 \rangle \) results from the conditional transition \( \langle nc_1, nc_2 \rangle \xrightarrow{\tau} \langle w_1, nc_2 \rangle \) and the fact that any assignment satisfies \text{true}. The transition 2 from \( \langle w_1, nc_2, y = 1 \rangle \) to \( \langle c_1, nc_2, y = 0 \rangle \) results from the conditional transition \( \langle w_1, nc_2 \rangle \xrightarrow{y > 0; y := y - 1} \langle c_1, nc_2 \rangle \) and the fact that any assignment \( \eta \) such that \( \eta(y) = 1 \) satisfies the condition \( y > 0 \), that is, \( \eta \models y > 0 \). Furthermore, the effect on variable \( y \) is to decrement it. The rest of the transitions is similar.

Consider now the conditional transition \( \langle c_1, w_2 \rangle \xrightarrow{y > 0; y := y - 1} \langle c_1, c_2 \rangle \). This transition does not induce a transition departing from \( \langle c_1, w_2, y = 0 \rangle \) because, in this case, any assignment \( \eta \) such that \( \eta(y) = 0 \) does not satisfy the condition \( y > 0 \).

Hence, observing the transition system \( T(\text{PG}_1 \parallel \text{PG}_2) \) we can conclude that there is no state of the form \( \langle c_1, c_2, \eta \rangle \) that is reachable. And so, mutual exclusion is guaranteed.

Exercise 2.4.
(a) The transition system $T_1 \parallel T_2$ is

\[
\begin{array}{c}
\text{\langle nc}_1, nc_2\rangle \\
\text{rel} \quad \text{rel} \\
\text{req} \quad \text{req} \\
\text{\langle c}_1, nc_2\rangle \quad \text{\langle nc}_1, c_2\rangle \\
\text{rel} \quad \text{req} \\
\text{\langle c}_1, c_2\rangle \\
\end{array}
\]

We omit the details on the construction of this transition system as it is very similar to the one defined in Exercise 2.1. Observing the transition system, it is not very difficult to find an execution that violates mutual exclusion:

\[\langle nc_1, nc_2 \rangle \rightarrow \langle c_1, nc_2 \rangle \rightarrow \langle c_1, c_2 \rangle \ldots\]

(b) We now define $T = (T_1 \parallel T_2) \parallel H T_{Sem}$, where $T_{Sem} = \langle S_s, Act_s, \rightarrow_s, I_s, \Xi, L_s \rangle$. In this case, $T = \langle S, Act, \rightarrow, I, \Xi, L \rangle$

where

- $S = (S_1 \times S_2) \times S_s = \{\langle nc_1, nc_2, free \rangle, \langle nc_1, nc_2, busy \rangle, \langle c_1, nc_2, free \rangle, \langle c_1, nc_2, busy \rangle, \langle nc_1, c_2, free \rangle, \langle nc_1, c_2, busy \rangle, \langle c_1, c_2, free \rangle, \langle c_1, c_2, busy \rangle\}$
- $Act = \{\text{req, rel}\}$
- $\rightarrow$ is such that
  - as $\langle nc_1, nc_2 \rangle \rightarrow_{12} \langle c_1, nc_2 \rangle$ and free $\rightarrow_s \text{ busy}$ then
    \[\langle nc_1, nc_2, free \rangle \rightarrow \langle c_1, nc_2, busy \rangle\]
  - as $\langle nc_1, nc_2 \rangle \rightarrow_{12} \langle nc_1, c_2 \rangle$ and free $\rightarrow_s \text{ busy}$ then
    \[\langle nc_1, nc_2, free \rangle \rightarrow \langle nc_1, c_2, busy \rangle\]
2.2. SOLUTIONS TO SELECTED EXERCISES

– as \( \langle c_1, nc_2 \rangle \xrightarrow{rel} 12 \langle nc_1, nc_2 \rangle \) and \( busy \xrightarrow{rel} s free \) then

\[
\langle c_1, nc_2, busy \rangle \xrightarrow{rel} \langle nc_1, nc_2, free \rangle
\]

– as \( \langle nc_1, c_2 \rangle \xrightarrow{rel} 12 \langle nc_1, nc_2 \rangle \) and \( busy \xrightarrow{rel} s free \) then

\[
\langle nc_1, c_2, busy \rangle \xrightarrow{rel} \langle nc_1, nc_2, free \rangle
\]

– as \( \langle c_1, c_2 \rangle \xrightarrow{rel} 12 \langle c_1, nc_2 \rangle \) and \( busy \xrightarrow{rel} s free \) then

\[
\langle c_1, c_2, busy \rangle \xrightarrow{rel} \langle c_1, nc_2, free \rangle
\]

among others.

• the remaining components of \( T \) are as expected.

Observe that the last transition, although it is correctly define, will never occur because, as we will see, the state \( \langle c_1, c_2, busy \rangle \) is not be reachable. To this end, we build the reachable part of the transition system, by unfolding the transitions starting from the initial state. The result is the transition system

![Transition System Diagram]

It is easy to see that this solution ensures mutual exclusion.
Chapter 3

Linear-time properties

3.1 Exercises

Exercise 3.1. Let \( \varphi \) be a propositional formula over \( \Xi \) and let \( P^\varphi_{inv} \) be an invariant for \( \varphi \). Show that \( P^\varphi_{inv} \) is also a safety property. What can be said about the converse?

Exercise 3.2. Let \( \Xi = \{p, q\} \). Formulate the following properties as LT properties and characterize them as being either an invariant property, a safety property or a liveness property, or none of these

(i) \( p \) should never occur;

(ii) \( p \) should occur exactly once;

(iii) in each state of the system, either \( p \) or \( q \) should hold;

(iv) \( p \) should eventually be followed by \( q \);

(v) \( p \) and \( q \) should occur infinitely often;

(vi) \( p \) and \( q \) alternate infinitely often;

(vii) \( q \) occurs exactly three times;

(viii) \( p \) occurs only finitely often.
Exercise 3.3. Let $\Xi = \{p, q\}$ and let $P$ be the LT property composed of all infinite words $v_0 v_1 v_2 \cdots \in (2^\Xi)^\omega$ such that there exists $n \geq 0$ with $p \in v_i$ for $0 \leq i \leq n$, $v_n = \{p, q\}$ and $q \in v_j$ for infinitely many $j \geq 0$. Provide a decomposition into a safety property and a liveness property for $P$.

Exercise 3.4. Let $P_1$ and $P_2$ be LT properties. Prove or disprove the following statement: $\text{pref}(P_1) = \text{pref}(P_2)$ if and only if $\text{closure}(P_1) = \text{closure}(P_2)$.

Exercise 3.5. Let $P$ be an LT-property. Prove the following results:

(a) $P \subseteq \text{closure}(P)$;

(b) $\text{pref}(\text{closure}(P)) = \text{pref}(P)$ (Lemma 3.17);

(c) $\text{closure}(\text{closure}(P)) = \text{closure}(P)$.

Exercise 3.6. Prove Lemma 3.16.

Exercise 3.7. Let $P_{\text{live}}$ and $P'_{\text{live}}$ be two liveness properties. Prove or disprove the following claims:

(a) $P_{\text{live}} \cup P'_{\text{live}}$ is a liveness property,

(b) $P_{\text{live}} \cap P'_{\text{live}}$ is a liveness property.

Exercise 3.8. Let $P_{\text{safe}}$ and $P'_{\text{safe}}$ be two safety properties. Prove or disprove the following claims:

(a) $P_{\text{safe}} \cup P'_{\text{safe}}$ is a safety property,

(b) $P_{\text{safe}} \cap P'_{\text{safe}}$ is a safety property.


Exercise 3.10. A bank uses a non-terminating program that monitors the balance $b$ of all its customer’s accounts once per week. The account balances are characterized by the set of propositional symbols

$$\{b < 0, b = 0, b > 100\}.$$  

- Express the following statements as LT-properties:

  - the balance is negative only finitely many times,
3.1. EXERCISES

- the balance alternates between debit and credit,
- eventually, the account remains with more than 100€ credit;
- any debit that occurs is balanced within two weeks.

• Determine for each LT-property whether it is a safety property or a liveness property.
3.2 Solutions to selected exercises

Exercise 3.1.

We show that $P_{\text{inv}}^\varphi$ is a safety property. Assume that $\sigma \notin P_{\text{inv}}^\varphi$. Then, there is some $i \geq 0$ such that $\sigma[i] \not\models \varphi$. Let $\tilde{\sigma} = \sigma[..i]$. Clearly, $\tilde{\sigma}$ is a bad prefix for $P_{\text{inv}}^\varphi$, as any sequence $\sigma'$ starting with $\tilde{\sigma}$ cannot belong to $P_{\text{inv}}^\varphi$. So, $P_{\text{inv}}^\varphi$ is a safety property.

The converse does not hold in general. Consider for instance the safety property $P$ over a traffic light stating that the green light is on immediately after the red light:

$$P = \{ \sigma \in (2\{g,r\})^\omega \mid \text{if } r \in \sigma[i] \text{ then } g \in \sigma[i+1], \text{ for all } i \geq 0 \}. $$

It is not very difficult to show that $P$ is a safety property. We leave it as an exercise to prove that the set of bad prefixes for $P$ is

$$\text{BadPref}(P) = \{ \tilde{\sigma} \in (2\{g,r\})^* \mid r \in \tilde{\sigma}[i] \text{ and } g \notin \tilde{\sigma}[i+1], \text{for } 0 \leq i < \text{length}(\tilde{\sigma}) \}. $$

However, $P$ is not an invariant because there is no propositional formula that can express the intended property. Looking the set of bad prefixes we can conclude that in order to falsify the property, we need to consider two consecutive states but the invariant condition is checked only statewise.

Exercise 3.2.

(i) Let $P_i$ be the LT-property “$p$ should never occur”. Then,

$$P_i = \{ \sigma \in (2\{p,q\})^\omega \mid p \notin \sigma[i], \text{ for every } i \geq 0 \} = \{ \sigma \in (2\{p,q\})^\omega \mid \sigma[i] \models \neg p, \text{ for every } i \geq 0 \}. $$

This is an invariant with invariant condition $\varphi = \neg p$.

(ii) Let $P_{ii}$ be the LT-property “$p$ should occur exactly once”. Then,

$$P_{ii} = \{ \sigma \in (2\{p,q\})^\omega \mid |\{i \mid p \in \sigma[i]\}| = 1 \}. $$

We start by showing that $P_{ii}$ is not a safety property. There are some $\omega$-words for which we can find bad prefixes. Namely, finite sequences with more than one occurrence of $p$ are bad prefixes. However, we cannot capture the $\omega$-words were $p$ does not occur. Consider, for instance, $\sigma = \emptyset^\omega$. Clearly $\sigma \notin P_{ii}$. However, for each finite prefix $\tilde{\sigma}_k = \sigma[..k] = \emptyset^k$ of $\sigma$, there is at
least one \( \sigma' \in (2\{p,q\})^\omega \) such that \( \hat{\sigma}_k \) is its prefix and \( \sigma' \in P_{ii} \). Take, for instance, \( \sigma' = \emptyset^k \{p\} \emptyset^\omega \). Hence, we can conclude that \( P_{ii} \) is not a safety property.

Next, we show that \( P_{ii} \) is not a liveness property. For \( P_{ii} \) to be a liveness property, we need to show that \( \text{pref}(P_{ii}) = (2\{p,q\})^* \). But is not the case.

Consider, for instance, the sequence \( \hat{\sigma} = \{p\} \emptyset^\omega \). Then, for every \( \sigma \) such that \( \hat{\sigma} \in \text{pref}(\sigma) \), we know that \( \sigma \notin P_{ii} \). Hence, we can conclude that \( \hat{\sigma} \notin \text{pref}(P_{ii}) \), and so \( P_{ii} \) is not a liveness property.

We leave as exercise to show that

\[
P_1 = \{ \sigma \in (2\{p,q\})^\omega \mid |\{i \mid p \in \sigma[i]\}| \leq 1 \}
\]

stating that \( p \) should occur at most once, is a safety property, that

\[
P_2 = \{ \sigma \in (2\{p,q\})^\omega \mid |\{i \mid p \in \sigma[i]\}| \geq 1 \}
\]

stating that \( p \) should occur at least once, is a liveness property, and that

\[P_{ii} = P_1 \cap P_2.\]

(iv) Let \( P_{iv} \) be the LT-property “\( p \) should eventually be followed by \( q \)”. This is a liveness property. To this end, we need to show that \( \text{pref}(P_{ii}) = (2\{p,q\})^* \). Clearly, we have \( \text{pref}(P_{ii}) \subseteq (2\{p,q\})^* \). To prove the converse inclusion, let \( \hat{\sigma} \in (2\{p,q\})^* \) and let \( \sigma = \{q\} \emptyset^\omega \). It is not very difficult to see that \( \hat{\sigma} \sigma \in P_{iv} \) because any occurrence of \( p \) in \( \hat{\sigma} \) is followed by the occurrence of \( q \) in \( \sigma \) and afterward there are no more occurrences of \( p \). Then, \( \hat{\sigma} \in \text{pref}(P_{iv}) \), as intended.

(vi) Let \( P_{vi} \) be the LT-property “\( p \) and \( q \) alternate infinitely often”. Then,

\[
P_{vi} = \{ \sigma \in (2\{p,q\})^\omega \mid \begin{align*}
&\text{if } p \in \sigma[i] \text{ then } q \in \sigma[i+1] \text{ and } \\
&\text{if } q \in \sigma[i] \text{ then } p \in \sigma[i+1], \text{ for } i \geq 0 \}
\end{align*} \]

Some examples of \( \omega \)-words in \( P_{vi} \) are:

- \( pqqpqqqpqq\ldots \)
- \( qqqpqqqpqq\ldots \)
- \( \emptyset\ldots\emptyset pqqpqqqpqq\ldots \)
This is not a safety property because, for instance, $\emptyset^\omega \notin P_v$ and we cannot find a bad prefix for it. See the discussion in (ii). It is also not a liveness property because, for instance, $pp$ is not in $\text{pref}(P_v)$. We leave as an exercise to find the decomposition of this property into a safety and a liveness properties.

**Exercise 3.3.**

Consider the property

$$P_1 = \{ \sigma \in (2^{\{p,q\}})^\omega \mid \exists i \geq 0. (\sigma[i] = \{p,q\}) \land \forall k \leq i. p \in \sigma[k] \} \lor \forall i \geq 0. p \in \sigma[i] \}.$$ 

This property states that, for every infinite sequence $\sigma$, either there is some point where $p$ and $q$ hold and until that point $p$ must also hold or, alternatively, that $p$ must hold everywhere. We now show that $P_1$ is a safety property. To this end, assume that $\sigma \notin P_1$. Then, there must a finite sequence $\hat{\sigma} = v_0 v_1 \ldots v_n$, prefix of $\sigma$, such that $p \in v_i, q \notin v_i$, for $i = 0, \ldots, n - 1$, and $p \notin v_n$. Any infinite sequence having $\hat{\sigma}$ as prefix will not be in $P_1$. Hence $\hat{\sigma}$ is a (minimal) bad prefix. Hence, we can conclude that $P_1$ is a safety property.

Consider now

$$P_2 = \{ \sigma \in (2^{\{p,q\}})^\omega \mid (\exists i \geq 0. \sigma[i] = \{p,q\}) \land (\forall i \geq 0. \exists k \geq i. q \in \sigma[k]) \}.$$ 

This property states that $p$ and $q$ must eventually hold simultaneously and also that $q$ must occur infinitely often. This is clearly a liveness property. Let $\tilde{\sigma} \in (2^{\{p,q\}})^*$ be an arbitrary finite sequence over $2^{\{p,q\}}$. Then, it follows that $\tilde{\sigma} \{p,q\}^\omega \in P_2$ because $p$ and $q$ hold simultaneously after $\tilde{\sigma}$ and $q$ holds infinitely often. Hence, $\text{pref}(P_2) = (2^{\{p,q\}})^*$ and, so, $P_2$ is a liveness property.

To conclude, we show that the desired property $P$ is $P_1 \cap P_2$. Let $\sigma \in P_1 \cap P_2$. Then, as $\sigma \in P_2$ then it must be the case that $q$ occurs infinitely often. Furthermore, we also know that there is some $n$ such that $\sigma[n] = \{p,q\}$. Then, as $\sigma \in P_1$ then, for every $k \leq n$, we know that $p \in v_k$. Then, $\sigma \in P$. With this, we have established $P_1 \cap P_2 \subseteq P$. The converse is straightforward.
Exercise 3.4.

Let $P_1$ and $P_2$ be LT-properties.

$(\Rightarrow)$ Assume that $\text{pref}(P_1) = \text{pref}(P_2)$. Then
\[
\sigma \in \text{closure}(P_1) \quad \text{iff} \quad \text{pref}(\sigma) \subseteq \text{pref}(P_1) \\
\text{iff} \quad \text{pref}(\sigma) \subseteq \text{pref}(P_2) \\
\text{iff} \quad \sigma \in \text{closure}(P_2).
\]

Hence, if $\text{pref}(P_1) = \text{pref}(P_2)$ then $\text{closure}(P_1) = \text{closure}(P_2)$.

$(\Leftarrow)$ Assume that $\text{closure}(P_1) = \text{closure}(P_2)$ and let $\hat{\sigma} \in \text{pref}(P_1)$. Then, there $\sigma \in P_1$ such that $\hat{\sigma} \in \text{pref}(\sigma)$. Then, as $P_1 \subseteq \text{closure}(P_1)$, we have that $\sigma \in \text{closure}(P_1)$ and, by hypothesis, we also have that $\sigma \in \text{closure}(P_2)$. Then, $\text{pref}(\sigma) \subseteq \text{pref}(P_2)$ and, so, $\hat{\sigma} \in \text{pref}(P_2)$. Hence, $\text{pref}(P_1) \subseteq \text{pref}(P_2)$. The proof of the converse is similar. So, we conclude that if $\text{closure}(P_1) = \text{closure}(P_2)$ then $\text{pref}(P_1) = \text{pref}(P_2)$.

Exercise 3.5.

(a) Assume that $\sigma \in P$. Then $\text{pref}(\sigma) \subseteq \text{pref}(P)$ and, so, $\sigma \in \text{closure}(P)$.

(b) Using (a), we know that $P \subseteq \text{closure}(P)$. Hence, it follows that $\text{pref}(P) \subseteq \text{pref}(\text{closure}(P))$.

For the converse, assume that $\hat{\sigma} \in \text{pref}(\text{closure}(P))$. Then, there $\sigma \in \text{closure}(P)$ such that $\hat{\sigma} \in \text{pref}(\sigma)$. But, as $\sigma \in \text{closure}(P)$ then $\text{pref}(\sigma) \subseteq \text{pref}(P)$. Hence, it follows that $\hat{\sigma} \in \text{pref}(P)$, that is, $\text{pref}(\text{closure}(P)) \subseteq \text{pref}(P)$.

(c) Using (a), we know that $\text{closure}(P) \subseteq \text{closure}(\text{closure}(P))$. To prove the converse, assume that $\sigma \in \text{closure}(\text{closure}(P))$. Then, by definition, it follows that $\text{pref}(\sigma) \subseteq \text{pref}(\text{closure}(P))$. By (b), we know that $\text{pref}(\text{closure}(P)) = \text{pref}(P)$. So $\text{pref}(\sigma) \subseteq \text{pref}(P)$, that is, $\sigma \in \text{closure}(P)$.

Exercise 3.6.

Let $T$ be a transition system. First, we prove that $P = \text{closure}(\text{Traces}(T))$ is a safety property. In fact
\[
\text{closure}(P) = \text{closure}(\text{closure}(\text{Traces}(T))) = \text{closure}(\text{Traces}(T)) = P.
\]

Hence, by Lemma 3.15, it follows that $P$ is a safety property.
For the other condition, and as \( P' \subseteq \text{closure}(P') \), for any LT-property \( P' \), it follows that \( \text{Traces}(T) \subseteq \text{closure}(\text{Traces}(T)) \), which means that \( T \models \text{closure}(\text{Traces}(T)) \).

**Exercise 3.7.**

(a) If \( P_{\text{live}} \) and \( P'_{\text{live}} \) are liveness properties then \( P_{\text{live}} \cup P'_{\text{live}} \) is also a liveness property. This is an immediate consequence of the following result on prefixes:

\[
\text{pref}(P \cup P') = \text{pref}(P) \cup \text{pref}(P').
\]

The inclusion from right to left is immediate. So, we prove that

\[
\text{pref}(P \cup P') \subseteq \text{pref}(P) \cup \text{pref}(P').
\]

Let \( \hat{\sigma} \in \text{pref}(P \cup P') \). Then, there is \( \sigma \in P \cup P' \) such that \( \hat{\sigma} \in \text{pref}(\sigma) \). If \( \sigma \in P \) then \( \hat{\sigma} \in \text{pref}(P) \). If \( \sigma \in P' \) then \( \hat{\sigma} \in \text{pref}(P') \). Hence, \( \hat{\sigma} \in \text{pref}(P) \cup \text{pref}(P') \) and so, \( \text{pref}(P \cup P') \subseteq \text{pref}(P) \cup \text{pref}(P') \).

The, the intended result follows immediately. Recall that, as \( P_{\text{live}} \) and \( P'_{\text{live}} \) are both liveness properties, then \( \text{pref}(P_{\text{live}}) = \text{pref}(P'_{\text{live}}) = (2^\Xi)^* \).

Hence,

\[
\text{pref}(P_{\text{live}} \cup P'_{\text{live}}) = \text{pref}(P_{\text{live}}) \cup \text{pref}(P'_{\text{live}}) = (2^\Xi)^* \cup (2^\Xi)^* = (2^\Xi)^*.
\]

And so, we can conclude that \( P_{\text{live}} \cup P'_{\text{live}} \) is, indeed, a liveness property.

(b) In general, \( P_{\text{live}} \cap P'_{\text{live}} \) is not necessarily a liveness property. Let \( \Xi = \{r, g\} \) and consider

\[
P_{\text{live}} = \{\sigma \in (2^\Xi)^\omega \mid \exists i \geq 0 \forall j \geq i. \ r \in \sigma[j]\}
\]

and

\[
P'_{\text{live}} = \{\sigma \in (2^\Xi)^\omega \mid \forall i \geq 0 \exists j \geq i. \ r \notin \sigma[j]\}.
\]

\( P_{\text{live}} \) expresses the fact that, after a certain point, the red light is constantly on. \( P'_{\text{live}} \) expresses the fact that the red light is off infinitely often. We leave as an exercise to prove that \( P_{\text{live}} \) and \( P'_{\text{live}} \) are liveness properties.

Then, the result follows, as it is easy to see that

\[
P_{\text{live}} \cap P'_{\text{live}} = \emptyset.
\]
3.2. **SOLUTIONS TO SELECTED EXERCISES**

and this is not a liveness property.

**Exercise 3.8.**

(a) If $P_{\text{safe}}$ and $P'_{\text{safe}}$ are safety properties then $P_{\text{safe}} \cup P'_{\text{safe}}$ is also a safety property. By Lemma 3.15, it is enough to show that $\text{closure}(P_{\text{safe}} \cup P'_{\text{safe}}) = P_{\text{safe}} \cup P'_{\text{safe}}$. But this is an immediate consequence of Lemma 3.14. Indeed

$$\text{closure}(P_{\text{safe}} \cup P'_{\text{safe}}) = \text{closure}(P_{\text{safe}}) \cup \text{closure}(P'_{\text{safe}}) = P_{\text{safe}} \cup P'_{\text{safe}}.$$ 

The last step holds because $P_{\text{safe}}$ and $P'_{\text{safe}}$ are safety properties.

(b) If $P_{\text{safe}}$ and $P'_{\text{safe}}$ are safety properties then $P_{\text{safe}} \cap P'_{\text{safe}}$ is also a safety property. Let $\sigma \not\in P_{\text{safe}} \cap P'_{\text{safe}}$. Then, either $\sigma \not\in P_{\text{safe}}$ or $\sigma \not\in P'_{\text{safe}}$. In the first case, there a bad prefix $\tilde{\sigma}$ for $\sigma$ in $P_{\text{safe}}$ such that if, for any $\sigma'$, we have $\tilde{\sigma} \not\in \text{pref}(\sigma')$ then $\sigma' \not\in P_{\text{safe}}$ and, consequently, $\sigma' \not\in P_{\text{safe}} \cap P'_{\text{safe}}$. This means that $\tilde{\sigma}$ is also a bad prefix for $\sigma$ in $P_{\text{safe}} \cap P'_{\text{safe}}$. The case where $\sigma \not\in P'_{\text{safe}}$ is similar. Observe that $\text{BadPref}(P_{\text{safe}} \cap P'_{\text{safe}}) = \text{BadPref}(P_{\text{safe}}) \cup \text{BadPref}(P'_{\text{safe}})$.

**Exercise 3.9.**

Let $P = P_{\text{safe}} \cap P_{\text{live}}$.

1. As $P \subseteq P_{\text{safe}}$ then $\text{closure}(P) \subseteq \text{closure}(P_{\text{safe}})$. Furthermore, as $P_{\text{safe}}$ is a safety property then, by Lemma 3.15, $\text{closure}(P_{\text{safe}}) = P_{\text{safe}}$. So, $\text{closure}(P) \subseteq P_{\text{safe}}$.

2. Let $\sigma \in P_{\text{live}}$. If $\sigma \in P$ then the result follows. So, assume that $\sigma \not\in P$. We need to show that $\sigma \not\in \text{closure}(P)$ that is, we need to show that $\sigma \not\in \text{closure}(P)$. But, if $\sigma \in P_{\text{live}}$ and $\sigma \not\in P$ then $\sigma \not\in P_{\text{safe}}$, by definition of $P$. And, using the previous result we can also conclude that $\sigma \not\in \text{closure}(P)$. 

Chapter 4

Regular Properties

4.1 Exercises

Exercise 4.1. Let $\Xi = \{p, q, r\}$. Consider the following LT-properties:

(i) If $q$ holds then, afterward $q$ will hold forever or until $r$ holds.

(ii) Between any two consecutive occurrences of $p$, $q$ must always hold.

(iii) Between any two consecutive occurrences of $p$, $q$ hold more often than $r$.

(iv) $p \land \neg q$ and $\neg p \land q$ must hold in alternation forever or until $r$ holds.

For each of the previous properties $P$ decide if it is a regular safety property and, if so, define an NFA $A$ such that $\mathcal{L}(A) = \text{BadPref}(P)$.

Exercise 4.2. Prove Lemma 4.6.

Exercise 4.3. Prove Theorem 4.12.

Exercise 4.4. Let $A_1$ and $A_2$ be NFAs. Show that $\mathcal{L}(A_1 \otimes A_2) = \mathcal{L}(A_1) \cap \mathcal{L}(A_2)$.

Exercise 4.5. Let $A$ be a NFA. Show that there is a total DFA $D$ that accepts the same language as $A$, that is, $\mathcal{L}(D) = \mathcal{L}(A)$.

Hint: Consider as states of $D$ the powerset of the set of states of $A$. 

231

Exercise 4.7. Complete the proof of Lemma 4.25.

Exercise 4.8. Complete the proof of Lemma 4.26. In the construction of the NBA, the set of accept states of the NBA is defined as the set of initial states of the NFA. Could the this been defined as the set of final states of the NFA instead?

Exercise 4.9. Complete the proof of Lemma 4.27.


Exercise 4.11. Let $\mathcal{A}$ be an NBA. Show that there is a nonblocking NBA $\mathcal{A}'$ equivalent to $\mathcal{A}$, that is, such that $L_\omega(\mathcal{A}) = L_\omega(\mathcal{A}')$.

Exercise 4.12. Complete the proof of Theorem 4.35.

Exercise 4.13. Consider the following safety property $P_{\text{safe}}$ over $\{a, b\}$:

Every time $a$ holds then next $b$ has to hold until $a$ eventually holds again.

(a) Show that $P_{\text{safe}}$ is a regular safety property.

(b) Define an NFA $\mathcal{A}$ such that $L(\mathcal{A}) = \text{BadPref}(P_{\text{safe}})$.

(c) Consider the transition system $T$

```
\begin{center}
\begin{tikzpicture}
    \node[state] (s0) at (0,0) {$s_0$};
    \node[state] (s1) at (1,0) {$s_1$};
    \node[state] (s2) at (0,-1) {$s_2$};
    \node[state] (s3) at (1,-1) {$s_3$};
    \draw (s0) edge[->] node {$a$} (s1);
    \draw (s0) edge[->] node {$b$} (s2);
    \draw (s1) edge[->] node {$a,b$} (s3);
    \draw (s2) edge[->] node {$\emptyset$} (s3);
\end{tikzpicture}
\end{center}
```

Define $T \otimes \mathcal{A}$. 
(d) Check if $T \models P_{safe}$.

**Exercise 4.14.** Consider the following safety property $P_{safe}$ over $\{p, q\}$:

*It is always the case that between any two consecutive occurrences of $p$, $q$ cannot hold.*

Additionally, consider the transition system $T$ presented below:

(a) Show that $P_{safe}$ is a regular safety property.

(b) Construct an NFA $A$ such that $L(A) = \text{BadPref}(P_{safe})$.

(c) Define the reachable part of $T \otimes A$.

(d) Check if $T \models P_{safe}$. Use the transition system $T \otimes A$ to justify your answer.

**Exercise 4.15.** Let $P_{safe}$ be a safety property. Prove or disprove the following assertions:

1. If $L$ is a regular language such that
   $$\text{MinBadPref}(P_{safe}) \subseteq L \subseteq \text{BadPref}(P_{safe})$$
   then $P_{safe}$ is regular.

2. If $P_{safe}$ is regular then any language $L$ such that
   $$\text{MinBadPref}(P_{safe}) \subseteq L \subseteq \text{BadPref}(P_{safe})$$
   is regular.

**Exercise 4.16.** Consider the GNBA $G = \langle Q, \Sigma, \delta, Q_0, F \rangle$ such that
with $\mathcal{F} = \{\{q_1\}, \{q_2\}\}$. Define an equivalent NBA.

**Exercise 4.17.** Find examples of

1. safety properties that are not $\omega$-regular,

2. liveness properties that are not $\omega$-regular.

**Exercise 4.18.** Recall the transition system $T = (T_1 \parallel T_2) \parallel H_{\text{sem}}$ defined in Exercise 2.3 and let $\Xi = \{nc_1, c_1\}$. Consider the liveness property $P_{\text{live}}$ defined by

"everytime $P_1$ is in its non critical section then eventually it will enter its critical section."

(a) Define an NBA $A$ such that $L_\omega(A) = P_{\text{live}}$.

(b) Define an NBA $\overline{A}$ such that $L_\omega(\overline{A}) = (2^\Xi \setminus P_{\text{live}})$.

(c) Show that $T \not\models P_{\text{live}}$:

(i) Start by defining the accessible fragment of $T \otimes \overline{A}$.

(ii) Sketch the use of the algorithm in Figure 4.17. Present the counter-example provided by the algorithm.
4.2 Solutions to selected exercises

Exercise 4.1.

(i) $P_i$ is a regular safety property. A bad prefix for $P_i$ is a finite sequence where, at some point, $q$ holds. Afterward, $q$ keeps holding (and $r$ does not hold) until neither $q$ nor $r$ hold. Hence, the set of bad prefixes for $P_i$ is

$$BadPref(P_i) = true^* q (q \land \neg r)^* (\neg q \land \neg r) true$$

Recall that when we write $\varphi$ we mean all the valuations that satisfy $\varphi$. For instance, when we write $q$ we mean the regular expression $\{q\} + \{p, q\}$. The automaton for $BadPref(P_i)$ is

(ii) $P_{ii}$ is a regular safety property. In this case, a bad prefix is a finite sequence where there are two consecutive occurrences of $p$ and between them $q$ fails to hold at least once. Observe that, between the two occurrences of $p$, $p$ must not hold so that the two occurrences are consecutive (at least, until $q$ fails). And after the failure of $q$, anything can hold until the next occurrence of $p$. Then, the set of bad prefixes for $P_{ii}$ is

$$BadPref(P_{ii}) = true^* p (q \land \neg p)^* (\neg q \land \neg p) true^* p true^*$$

The automaton for $BadPref(P_{ii})$ is

(iii) $P_{iii}$ is not a regular safety property. In this case, a bad prefix is a finite sequence such that between two consecutive occurrences of $p$, we must find at least as many occurrences of $r$ as of $q$. But this involves counting the number of occurrences of $q$ and $r$, which is cannot be captured by a regular language. Hence, $BadPref(P_{iii})$ is not a regular language and, consequently,
$P_{iii}$ is not a regular safety property. We leave as an exercise to show that $P_{iii}$ is, nevertheless, a safety property.

**Exercise 4.5.**

Let $\mathcal{A} = \langle Q, \Sigma, \delta, Q_0, F \rangle$ be an NFA. Consider the DFA $\mathcal{D}_A = \langle Q', \Sigma, \delta', q'_0, F' \rangle$ defined as follows:

- $Q' = 2^Q$;
- $\delta'(q', v) = \bigcup_{q \in q'} \delta(q, v)$;
- $q'_0 = Q_0$;
- $F' = \{ q' \in Q' \mid q' \cap F \neq \emptyset \}$.

Observe that a DFA has, at most, one initial state that, in this case, is the set composed of all initial states of $\mathcal{A}$. A state in $\mathcal{D}_A$ is accepting if it contains at least one accepting state from $\mathcal{A}$. The transition function $\delta'$ can be extended to words of $\Sigma^*$ in a straightforward way:

- $\delta'^*(q', \epsilon) = q'$;
- $\delta'^*(q', vw) = \delta'^*(\delta'(q', v), w)$.

Before we prove that $L(\mathcal{D}_A) = L(\mathcal{A})$ we illustrate the construction with an example. Consider the following NFA over the alphabet $\Sigma = \{0, 1\}$

The language accepted by this automaton is $(0 + 1)^*1^+$. The corresponding DFA is the automaton $\mathcal{D} = \langle Q', \Sigma, \delta', q'_0, F' \rangle$ defined as follows:

- $Q' = \{ \emptyset, \{q_0\}, \{q_1\}, \{q_0, q_1\} \}$;
- $q'_0 = \{q_0\}$;
- $F' = \{\{q_1\}, \{q_0, q_1\}\}$. 
As for the transition function $\delta'$, we have, for state $\{q_0\}$

$$
\delta'(\{q_0\}, 0) = \bigcup_{q \in \{q_0\}} \delta(q, 0) \\
= \delta(q_0, 0) \\
= \{q_0\}
$$

$$
\delta'(\{q_0\}, 1) = \bigcup_{q \in \{q_0\}} \delta(q, 1) \\
= \delta(q_0, 1) \\
= \{q_0, q_1\}.
$$

For state $\{q_1\}$, we have

$$
\delta'({q_1}, 0) = \bigcup_{q \in \{q_1\}} \delta(q, 0) \\
= \delta(q_1, 0) \\
= \emptyset
$$

$$
\delta'({q_1}, 1) = \bigcup_{q \in \{q_1\}} \delta(q, 1) \\
= \delta(q_1, 1) \\
= \{q_1\}
$$

For state $\{q_0, q_1\}$, we have

$$
\delta'({q_0, q_1}, 0) = \bigcup_{q \in \{q_0, q_1\}} \delta(q, 0) \\
= \delta(q_0, 0) \cup \delta(q_1, 0) \\
= \{q_0\} \cup \emptyset \\
= \{q_0\}
$$

$$
\delta'({q_0, q_1}, 1) = \bigcup_{q \in \{q_0, q_1\}} \delta(q, 1) \\
= \delta(q_0, 1) \cup \delta(q_1, 1) \\
= \{q_0, q_1\} \cup \{q_1\} \\
= \{q_0, q_1\}
$$

For state $\emptyset$ and for any $v \in \Sigma$,

$$
\delta'({\emptyset}, v) = \emptyset
$$

A diagrammatic depiction of $\mathcal{D}$ is presented in Figure 4.1.
We now prove that $L(D_A) = L(A)$. To this end, we start by proving that

$$\delta^*(q', w) = \bigcup_{q \in q'} \delta^*(q, w) \quad (\ast)$$

for any word $w \in \Sigma^*$. The proof follows by induction on $w$.

Assume that $w = \epsilon$. Then, we have $\delta^*(q', \epsilon) = q'$. On the other hand, $\bigcup_{q \in q'} \delta^*(q, \epsilon) = \bigcup_{q \in q'} \{q\} = q'$, and so the equality holds.

Assume now that $w = vw'$. Then

$$\delta^*(q', vw') = \delta^*(\delta'(q', v), w')$$
$$= \bigcup_{q'' \in \delta'(q', v)} \delta^*(q'', w') \quad (\dagger)$$
$$= \bigcup_{q \in q'} \bigcup_{q'' \in \delta(q, v)} \delta^*(q'', w') \quad (\ddagger)$$
$$= \bigcup_{q \in q'} \delta^*(q, v w')$$

In (\dagger), we used the induction hypothesis. In (\ddagger), we used the definition of $\delta$. Observe that $q'' \in \delta'(q', v)$ if and only if $q'' \in \bigcup_{q \in q'} \delta(q, v)$.
Having established (⋆), it is straightforward to prove \( \mathcal{L}(\mathcal{D}_A) = \mathcal{L}(A) \):

\[
\begin{align*}
  w \in \mathcal{L}(\mathcal{D}_A) & \iff \delta^*(q'_0, w) = q'_f, \text{ for some } q'_f \in F' \\
  & \iff \delta^*(Q_0, w) = q'_f, \text{ for some } q'_f \in F' \\
  & \iff \bigcup_{q \in Q_0} \delta^*(q, w) = q'_f, \text{ for some } q'_f \text{ such that } q'_f \cap F \neq \emptyset \\
  & \iff \bigcup_{q \in Q_0} \delta^*(q, w) \cap F \neq \emptyset \\
  & \iff w \in \mathcal{L}(A).
\end{align*}
\]

**Exercise 4.6.**

Let \( P_{safe} \) be a safety property. First, assume that \( \text{MinBadPref}(P_{safe}) \) is regular. Then, there is an NFA \( A = (Q, 2^\Sigma, \delta, Q_0, F) \) such that \( \mathcal{L}(A) = \text{MinBadPref}(P_{safe}) \). Let \( A' = (Q, 2^\Sigma, \delta', Q_0, F) \) be the NFA obtained from \( A \) by adding a transition \( q \xrightarrow{v} q' \) for each final state \( q \in F \) and \( v \in 2^\Sigma \), that is,

\[
\delta'(q, v) = \begin{cases} 
\delta(q, v) & \text{if } q \notin F \\
\delta(q, v) \cup \{q\} & \text{if } q \in F.
\end{cases}
\]

We have to prove that \( \mathcal{L}(A') = \text{BadPref}(P_{safe}) \). Let \( w = v_1 \ldots v_n \in \mathcal{L}(A') \). Then, there is an accepting run \( q_0 \ldots q_n \) for \( w \) in \( A' \). This not necessarily a run in \( A \). But, we know that \( q_n \) is an accept state and that there might be other accept states in this run. Let \( k \) be the first index of such accept states. Observe that, in this case, \( q_0 \ldots q_k \) is a run in \( A \) and, furthermore, it is accepting for \( v_1 \ldots v_k \). This means that \( v_1 \ldots v_k \in \text{MinBadPref}(P_{safe}) \) and, consequently, \( w \in \text{BadPref}(P_{safe}) \).

Conversely, let \( w = v_1 \ldots v_n \in \text{BadPref}(P_{safe}) \). Then, there is a minimal bad prefix \( v_1 \ldots v_k \) of \( w \), and there is an accepting run \( q_0 \ldots q_k \) for \( v_1 \ldots v_k \) in \( A \). Consider now the run \( q_0 \ldots q_k \ldots q_k \) obtained from \( q_0 \ldots q_k \) by adding \( n - k \) copies of \( q_k \) at the end. As \( q_k \in F \), and by construction, this is a run for \( w \) in \( A' \). And, furthermore, it also accepting. Hence, \( w \in \mathcal{L}(A') \).

To prove the converse, assume that \( P_{safe} \) is regular, i.e., assume that \( \text{BadPref}(P_{safe}) \) is regular. Then, there is an NFA \( A \) such that \( \mathcal{L}(A) = \text{BadPref}(P_{safe}) \). Let \( \mathcal{D}_A \) be a DFA obtained from \( A \) accepting the same language (see Exercise 4.5). Finally, let \( D \) be the DFA obtained from \( \mathcal{D}_A \) by removing all outgoing edges from accept states. We prove that \( \mathcal{L}(D) = \text{MinBadPref}(P_{safe}) \). Let \( w = v_1 \ldots v_n \in \mathcal{L}(D) \). Then, there is a unique accepting run \( q_0 \ldots q_n \) for \( w \) in \( D \). We also know that this a run in \( \mathcal{D}_A \) and

---

**Exercise 4.6.**

Let \( P_{safe} \) be a safety property. First, assume that \( \text{MinBadPref}(P_{safe}) \) is regular. Then, there is an NFA \( A = (Q, 2^\Sigma, \delta, Q_0, F) \) such that \( \mathcal{L}(A) = \text{MinBadPref}(P_{safe}) \). Let \( A' = (Q, 2^\Sigma, \delta', Q_0, F) \) be the NFA obtained from \( A \) by adding a transition \( q \xrightarrow{v} q' \) for each final state \( q \in F \) and \( v \in 2^\Sigma \), that is,

\[
\delta'(q, v) = \begin{cases} 
\delta(q, v) & \text{if } q \notin F \\
\delta(q, v) \cup \{q\} & \text{if } q \in F.
\end{cases}
\]

We have to prove that \( \mathcal{L}(A') = \text{BadPref}(P_{safe}) \). Let \( w = v_1 \ldots v_n \in \mathcal{L}(A') \). Then, there is an accepting run \( q_0 \ldots q_n \) for \( w \) in \( A' \). This not necessarily a run in \( A \). But, we know that \( q_n \) is an accept state and that there might be other accept states in this run. Let \( k \) be the first index of such accept states. Observe that, in this case, \( q_0 \ldots q_k \) is a run in \( A \) and, furthermore, it is accepting for \( v_1 \ldots v_k \). This means that \( v_1 \ldots v_k \in \text{MinBadPref}(P_{safe}) \) and, consequently, \( w \in \text{BadPref}(P_{safe}) \).

Conversely, let \( w = v_1 \ldots v_n \in \text{BadPref}(P_{safe}) \). Then, there is a minimal bad prefix \( v_1 \ldots v_k \) of \( w \), and there is an accepting run \( q_0 \ldots q_k \) for \( v_1 \ldots v_k \) in \( A \). Consider now the run \( q_0 \ldots q_k \ldots q_k \) obtained from \( q_0 \ldots q_k \) by adding \( n - k \) copies of \( q_k \) at the end. As \( q_k \in F \), and by construction, this is a run for \( w \) in \( A' \). And, furthermore, it also accepting. Hence, \( w \in \mathcal{L}(A') \).

To prove the converse, assume that \( P_{safe} \) is regular, i.e., assume that \( \text{BadPref}(P_{safe}) \) is regular. Then, there is an NFA \( A \) such that \( \mathcal{L}(A) = \text{BadPref}(P_{safe}) \). Let \( \mathcal{D}_A \) be a DFA obtained from \( A \) accepting the same language (see Exercise 4.5). Finally, let \( D \) be the DFA obtained from \( \mathcal{D}_A \) by removing all outgoing edges from accept states. We prove that \( \mathcal{L}(D) = \text{MinBadPref}(P_{safe}) \). Let \( w = v_1 \ldots v_n \in \mathcal{L}(D) \). Then, there is a unique accepting run \( q_0 \ldots q_n \) for \( w \) in \( D \). We also know that this a run in \( \mathcal{D}_A \) and

---
that is run is accepting. Hence \( w \in \text{BadPref}(P_{\text{safe}}) \). Assume that it is not minimal. Then, there would be \( k < n \) such that \( v_1 \ldots v_k \) would also be a bad prefix, that is \( v_1 \ldots v_k \in L(D_A) \). And, as \( D_A \) is deterministic, the only run for \( v_1 \ldots v_k \) would be \( q_0 \ldots q_k \) and \( q_k \) would have to be an accept state. But this is impossible because, then \( q_0 \ldots q_k \ldots q_n \) could not be a run in \( D \) since, by construction, we removed all outgoing edges from accept states. Hence, \( w \) has to be a minimal bad prefix, that is \( w \in \text{MinBadPref}(P_{\text{safe}}) \).

Conversely, assume that \( w = v_1 \ldots v_n \in \text{MinBadPref}(P_{\text{safe}}) \). Then, in particular, \( w \in \text{BadPref}(P_{\text{safe}}) \) and so \( w \in L(D_A) \). Let \( q_0 \ldots q_n \) be the (unique) accepting run for \( w \) in \( D_A \). As \( w \) is a minimal bad prefix, then \( q_n \) is the only accept state in this run. This means that \( q_0 \ldots q_n \) is also a run for \( w \) in \( D \) and, furthermore, is accepting. Hence, \( w \in L(D) \).

**Exercise 4.13.**

(a) We start by showing that \( P_{\text{safe}} \) is a safety property. Assume that \( \sigma \not\in P_{\text{safe}} \). Then, there is some \( i \) such that \( a \) holds in \( \sigma[i] \) and after this occurrence of \( a \), \( b \) may hold for some time but it eventually will fail to hold at some point \( j \), before \( a \) holds again, if \( a \) ever holds again. Let \( \widehat{\sigma} = \sigma[..j] \). Then, \( \widehat{\sigma} \) is as follows:

- \( a \in \widehat{\sigma}[i] \);
- \( b \in \widehat{\sigma}[k] \) and \( a \not\in \widehat{\sigma}[k] \), for \( i < k < j \);
- \( a, b \not\in \sigma[j] \).

Clearly, any sequence \( \sigma' \) such that \( \widehat{\sigma} \) is its prefix is not in \( P_{\text{safe}} \). Hence, \( \widehat{\sigma} \) is a bad prefix for \( P_{\text{safe}} \).

Next, we show that \( \text{BadPref}(P_{\text{safe}}) \) is a regular language. But, from what said above, it is easy to see that \( \text{BadPref}(P_{\text{safe}}) \) is denoted by

\[
\text{true}^* a (\neg a \land b)^* (\neg a \land \neg b) \text{ true}^* .
\]

(b) The following NFA accepts the language \( \text{BadPref}(P_{\text{safe}}) \):

![NFA diagram](null)
(c) Let $T \otimes A = \langle S', A, \rightarrow', I', \Xi', L' \rangle$ where

- $S' = S \otimes Q = \{ \langle s_0, q_0 \rangle, \langle s_0, q_1 \rangle, \langle s_0, q_2 \rangle, \langle s_1, q_0 \rangle, \langle s_1, q_1 \rangle, \langle s_1, q_2 \rangle, \langle s_2, q_0 \rangle, \langle s_2, q_1 \rangle, \langle s_2, q_2 \rangle, \langle s_3, q_0 \rangle, \langle s_3, q_1 \rangle, \langle s_3, q_2 \rangle \}$

- $I' = \{ \langle s_0, q_0 \rangle, \langle s_0, q_1 \rangle \}$

- $\Xi' = Q = \{ q_0, q_1, q_2 \}$

- $L'((s, q)) = \{ q \}$, for each $\langle s, q \rangle \in Q'$.

Concerning the definition of $I'$, observe that, in $T$, we know that $L(s_0) = \{ a \}$, and, in $A$, we know that $q_0 \xrightarrow{\{ a \}} q_0$ and $q_0 \xrightarrow{\{ a \}} q_1$. Hence, $I' = \{ \langle s_0, q_0 \rangle, \langle s_0, q_1 \rangle \}$. Finally, we define the transition relation $\rightarrow'$:

- from $s_0 \xrightarrow{b} s_1$, $L(s_1) = \{ b \}$ and $q_0 \xrightarrow{\{ b \}} q_0$ we get

  $\langle s_0, q_0 \rangle \xrightarrow{\rightarrow'} \langle s_1, q_0 \rangle$

- from $s_1 \xrightarrow{a} s_0$, $L(s_0) = \{ a \}$, $q_0 \xrightarrow{\{ a \}} q_0$ and $q_0 \xrightarrow{\{ a \}} q_1$ we get

  $\langle s_1, q_0 \rangle \xrightarrow{\rightarrow'} \langle s_0, q_0 \rangle$ and $\langle s_1, q_0 \rangle \xrightarrow{\rightarrow'} \langle s_0, q_1 \rangle$

The full transition relation for the reachable part of $T \otimes A$ is depicted below. Note that we omit the labels of the states to avoid overloading the diagram.
(d) To check if $T \models P_{\text{safe}}$ we use Theorem 4.20 and check instead if $T \otimes A \models P_{\text{inv}(A)}$. Recall that, in this case, $P_{\text{inv}(A)} = \bigwedge_{q \in F} \neg q$ is $\neg q_2$. Consider the following path in $T \otimes A$:

$$\pi = (s_0, q_0) (s_2, q_1) (s_1, q_1) (s_3, q_2) \ldots.$$ 

The trace of $\pi$ is $\sigma$

$$\sigma = \{ q_0 \} \{ q_1 \} \{ q_1 \} \{ q_2 \} \ldots$$

Clearly, $\sigma[3] \not\models \neg q_2$ which implies that $\sigma \not\in P_{\text{inv}(A)}$ and, consequently, that $T \otimes A \not\models P_{\text{inv}(A)}$. Hence, by Theorem 4.20, we conclude that $T \not\models P_{\text{safe}}$. Additionally, using $\pi$ we can obtain a counter-example, by considering the path $\pi'$ obtained from $\pi$

$$\pi' = s_0 s_2 s_1 s_3 \ldots$$

with trace

$$\sigma' = \{ a \} \{ a, b \} \{ b \} \{ \} \ldots.$$
Observe that $\sigma[1..3]$ is a bad prefix for $P_{\text{safe}}$, as discussed in (a).

**Exercise 4.14.**

(a) We only prove that $P_{\text{safe}}$ is regular, and leave as an exercise to show that $P_{\text{safe}}$ is a safety property. We need to show that the set $\text{BadPref}(P_{\text{safe}})$ is a regular language. But is not very difficult to see that

$$\text{BadPref}(P_{\text{safe}}) = \text{true}^* p (\neg p \land \neg q)^* (\neg p \land q) \text{true}^* p \text{true}^*.$$  

(b) The following NFA $A$ accepts the language $\text{BadPref}(P_{\text{safe}})$:

(c) We present the reachable part of $T \otimes A$, but omit the details of the construction, as it is similar to previous examples.

(d) To check if $T \models P_{\text{safe}}$ we use, once again, Theorem 4.20 and check instead if $T \otimes A \models P_{\text{inv}(A)}$. In this case, $P_{\text{inv}(A)} = \bigwedge_{q \in F} \neg q$ is $\neg q_3$. It is not very difficult to see that $q_3$ does not appear in the label of any reachable state. So, $s' \models \neg q_3$, for every reachable state $s'$ in $T \otimes A$. Consequently, $T \otimes A \models P_{\text{inv}(A)}$ and we may conclude that $T \models P_{\text{safe}}$.

**Exercise 4.15.**

1. The assertion holds. Let $P_{\text{safe}}$ be a safety property and let $L$ be a regular
language such that MinBadPref(\(P_{\text{safe}}\)) \(\subseteq \mathcal{L} \subseteq \text{BadPref}(P_{\text{safe}})\). If \(\mathcal{L}\) is regular then there is an NFA \(\mathcal{A}\) such that \(\mathcal{L}(\mathcal{A}) = \mathcal{L}\). Let \(\mathcal{A}'\) be the NFA obtained from \(\mathcal{A}\) by adding a transition \(q \xrightarrow{v} q\) for each final state \(q \in F\) and \(v \in 2^\Xi\), as in Exercise 4.6. We claim that \(\mathcal{L}(\mathcal{A}') = \text{BadPref}(P_{\text{safe}})\) and, so, \(P_{\text{safe}}\) is a regular safety property. Let \(w = v_1 \ldots v_n \in \text{BadPref}(P_{\text{safe}})\). Then, there is a minimal bad prefix \(v_1 \ldots v_k \in \text{MinBadPref}(P_{\text{safe}})\). As MinBadPref(\(P_{\text{safe}}\)) \(\subseteq \mathcal{L} = \mathcal{L}(\mathcal{A})\) it follows that there is an accepting run \(q_0 \ldots q_k\) for \(v_1 \ldots v_k\) in \(\mathcal{A}\). Then, we can construct a run for \(w\) in \(\mathcal{A}'\) by adding \(n - k\) copies of \(q_k\) at the end of \(q_0 \ldots q_k\). This can be done in \(\mathcal{A}'\) because \(q_k\) is an accept state. Hence, \(w \in \mathcal{L}(\mathcal{A}')\).

Assume now that \(w = v_1 \ldots v_n \in \mathcal{L}(\mathcal{A}')\). Then, there is an accepting run \(q_0 \ldots q_n\) for \(w\) in \(\mathcal{A}'\). But, by construction of \(\mathcal{A}\), we know that the last states of this run are all copies of the same accept state \(q_k\).

\[
q_0 \xrightarrow{v_1} q_1 \xrightarrow{v_2} \ldots \xrightarrow{v_k} \underbrace{\ldots \xrightarrow{v_{k+1}} q_k}_{\text{copies of } q_k} \xrightarrow{\ldots} q_n
\]

It is not very difficult to see that \(q_0 \ldots q_k\) is an accept run for \(v_1 \ldots v_k\) in \(\mathcal{A}\). This means that \(v_1 \ldots v_k \in \mathcal{L}\) and, as \(\mathcal{L} \subseteq \text{BadPref}(P_{\text{safe}})\), it follows that \(v_1 \ldots v_k \in \text{BadPref}(P_{\text{safe}})\). Hence, it also follows that \(w \in \text{BadPref}(P_{\text{safe}})\), as \(v_1 \ldots v_k\) is a prefix of \(w\).

So, we proved that there is an NFA \(\mathcal{A}'\) such that \(\mathcal{L}(\mathcal{A}') = \text{BadPref}(P_{\text{safe}})\). So \(\text{BadPref}(P_{\text{safe}})\) is a regular language and, consequently, \(P_{\text{safe}}\) is a regular safety property.

2. The assertion is false. Consider the safety property \(P_{\text{safe}} = \emptyset\). We have that MinBadPref(\(P_{\text{safe}}\)) = \{\(\epsilon\)\} and BadPref(\(P_{\text{safe}}\)) = \((2^\Xi)^*\). So, \(P_{\text{safe}}\) is clearly regular. However, not every \(\mathcal{L}\) such that MinBadPref(\(P_{\text{safe}}\)) \(\subseteq \mathcal{L} \subseteq \text{BadPref}(P_{\text{safe}})\) is regular because this would mean that every language was a regular language.

Exercise 4.16.

We are going to the algorithm proposed in the proof of Theorem 4.35. As we have two sets \(F_1 = \{q_1\}\) and \(F_2 = \{q_2\}\) of final states in \(G\) we need two copies of \(G\) to build the equivalent NBA

\[
\mathcal{A} = \langle Q', \{0, 1\}, \delta', Q'_0, F'\rangle.
\]

The set of states \(Q'\) includes two copies of the original set. Hence,
• $Q' = \{ q_{0,1}, q_{1,1}, q_{2,1}, q_{0,2}, q_{1,2}, q_{2,2} \}$.

For convenience, we use $q_{i,1}$ and $q_{i,2}$ instead of $\langle q_i, 1 \rangle$ and $\langle q_i, 2 \rangle$, respectively, with $i = 0, 1, 2$. The set of initial states $Q'_0$ is the set of initial states of the first copy and the set of final states is the first set of final states, $F_1$. Hence,

• $Q' = \{ q_{0,1} \}$;

• $F' = \{ q_{1,1} \}$.

All that remains is to define the transition relation $\delta'$:

• as $\delta(q_0, 0) = \{ q_1 \}$ then

  - $\delta'(q_{0,1}, 0) = \{ q_{1,1} \}$ – in this case, we do not change copy because $q_0 \not\in F_1$;
  - $\delta'(q_{0,2}, 0) = \{ q_{1,2} \}$ – in this case, we do not change copy because $q_0 \not\in F_2$;

• as $\delta(q_1, 1) = \{ q_0, q_1, q_2 \}$ then

  - $\delta'(q_{1,1}, 1) = \{ q_{0,2}, q_{1,2}, q_{2,2} \}$ – in this case, we move from copy 1 to copy 2, because $q_1 \in F_1$;
  - $\delta'(q_{1,2}, 1) = \{ q_{0,2}, q_{1,2}, q_{2,2} \}$ – in this case, we do not change copy, because $q_1 \not\in F_2$;

• as $\delta(q_2, 1) = \{ q_0 \}$ then

  - $\delta'(q_{2,1}, 1) = \{ q_{0,1} \}$ – in this case, we do not change copy, because $q_2 \not\in F_1$;
  - $\delta'(q_{2,2}, 1) = \{ q_{0,1} \}$ – in this case, we move from copy 2 to copy 1, because $q_2 \in F_2$. 
Note that, in particular, $q_{2,1}$ is not reachable and, consequently, may be omitted.

**Exercise 4.18.**

(a) An NBA for $P_{\text{live}}$ is

(b) The language $(2^\Sigma)\omega \setminus P_{\text{live}}$ contains all $\omega$-words such that at some instant $nc_1$ holds and from that point on $c_1$ will never hold again. A (non-blocking) NBA $\overline{A}$ for this property is

(c.i) We present the reachable part of $T \otimes \overline{A}$. We present just the diagram for the transition relation and relevant states. The details of the construction
are similar to what was done in Exercise 4.13. In order to simplify the diagram, we are going to use the labels $s_0$, $s_1$, and $s_2$ instead of $\langle nc_1, nc_2, \text{free} \rangle$, $\langle c_1, nc_2, \text{busy} \rangle$ and $\langle nc_1, c_2, \text{busy} \rangle$, respectively.

(c.ii) By Theorem 4.42, $T \models P_{\text{live}}$ if and only if $T \otimes \mathcal{A} \models P_{\text{pers}(\mathcal{A})}$. And, by Theorem 4.44, $T \otimes \mathcal{A} \not\models P_{\text{pers}(\mathcal{A})}$ if there is a reachable state $s$ such that $s \not\models P_{\text{pers}(\mathcal{A})}$ and $s$ occurs in a cycle in $T \otimes \mathcal{A}$. In this case, the states that do not satisfy the formula of $P_{\text{pers}(\mathcal{A})}$ are $\langle s_0, q_1 \rangle$, and $\langle s_1, q_1 \rangle$. Each of these states is reachable and occurs in a cycle. For instance,

$$\pi = \langle s_0, q_0 \rangle \langle s_1, q_0 \rangle \langle s_0, q_1 \rangle \langle s_2, q_1 \rangle \langle s_0, q_1 \rangle \ldots$$

illustrates that $\langle s_0, q_1 \rangle$ is reachable and that it occurs in a cycle. Hence, $T \otimes \mathcal{A} \not\models P_{\text{pers}(\mathcal{A})}$ and, consequently, $T \not\models P_{\text{live}}$.

If we project $\pi$ in $T$, we get the path

$$\pi' = s_0 \ s_1 \ (s_0 \ s_2)^\omega$$

whose trace is

$$\text{trace}(\pi') = \{ nc_1 \} \{ c_1 \} (\{ nc_1 \} \{ nc_1 \})^\omega$$

and this trace clearly violates $P_{\text{live}}$. 
Chapter 5

Linear Temporal Logic

5.1 Exercises

Exercise 5.1. Prove Lemma 5.6.

Exercise 5.2. Check which of the following formulas are valid and/or satisfiable. Justify your answer.

1. $X G_0 \varphi \rightarrow G_0 X \varphi$
2. $X F_0 \varphi \rightarrow F_0 X \varphi$
3. $G_0 \varphi \rightarrow F_0 \varphi$
4. $F_0 \varphi \rightarrow G_0 \varphi$
5. $G_0 (X X \varphi \rightarrow X \varphi)$
6. $F_0 \varphi \rightarrow G_0 F_0 \varphi$
7. $F_0 \varphi \rightarrow F_0 F_0 \varphi$
8. $(\varphi \land G_0 (\varphi \rightarrow X \varphi)) \rightarrow G_0 \varphi$.

Exercise 5.3. Prove Lemma 5.12

Exercise 5.4. Let $\Xi = \{p, q\}$ and consider the formula

$$\varphi = (p \rightarrow X \neg q) \cup (p \land q).$$
(a) Show that \( P = \text{Mod}(\varphi) \) is a safety property.

(b) Define an NFA \( A \) such that \( L(A) = \text{BadPref}(P) \).

(c) Consider now the formula
\[
\varphi' = (p \rightarrow X\neg q) \cup (p \land q)
\]
and let \( P' = \text{Mod}(\varphi') \). Decompose \( P' \) into a safety property \( P'_{\text{safe}} \) and a liveness property \( P'_{\text{live}} \) such that
\[
P' = P'_{\text{safe}} \cap P'_{\text{live}}.
\]
Show that \( P'_{\text{safe}} \) is indeed a safety property and the \( P'_{\text{live}} \) is indeed a liveness property.

Exercise 5.5. Define an NBA for the following LTL formulas:
1. \( G_o(p \lor \neg Xq) \)
2. \( (F_0 p) \lor G_o F_0 (p \leftrightarrow q) \)
3. \( XX(p \lor F_0 G_0 q) \).

Exercise 5.6. Consider the transition system \( T \)

![Transition System Diagram]

and the LTL formulas
\[
\varphi_1 = G_o F_0 p \rightarrow G_o F_0 q \quad \text{and} \quad \varphi_2 = X(p \land Xp).
\]
For \( i = 1, 2 \), sketch the main steps of the model checking algorithm when applied to \( T \) and \( \varphi_i \). To this end, carry out the following steps.
5.1. EXERCISES

(a) Define an NBA for $\neg \varphi_i$.
(b) Define the reachable fragment of $T \otimes A_{\neg \varphi_i}$.
(c) Explain the main steps of the search algorithm when applied to $T \otimes A_{\neg \varphi_i}$.
(d) If $T \not\models \varphi_i$, present a counter-example provided by the algorithm.

Exercise 5.7. Consider the transition system $T$, with $\Xi = \{p, r\}$.

```
\begin{array}{c}
s_0\\ \{p\}\\ \rightarrow\\ \{r\}\\ s_1\\ s_2
\end{array}
```

Check if

$$T \models G_\omega (p \rightarrow F_\omega r)$$

or find a counterexample produced by the relevant model checking algorithm. You may generate only the reachable part of the product construction.

Hint: For the NBA below, it holds

$$L_\omega(A) = Mod(F_\omega (p \land G_\omega \neg r)).$$

Exercise 5.8. Let $\Xi = \{p\}$ and consider the LTL formula $\varphi = (p \land X p) \cup \neg p$.

1. Compute the elementary sets with respect to $\varphi$.
2. Construct the GNBA $G_\varphi$, describe in Theorem 5.21.
**Exercise 5.9.** Consider the formula $\varphi = p \cup (\neg p \land q)$ over $\Xi = \{p, q\}$, and let $G_\varphi$ be the GNBA constructed as described in Theorem 5.21.

(a) What are the initial states of $G_\varphi$?
(b) What are the accept states of $G_\varphi$?
(c) Provide an accepting run for $\{p\} \{p\} \{p, q\} \omega$.
(d) Explain why there are no accepting runs for the words $\{p\}^\omega$ and $\{p\} \{p\} \{p, q\}^\omega$.

(*Hint: it not necessary to depict $G_\varphi$.*)

**Exercise 5.10.** Extend the construction in the proof of Theorem 5.21 to the temporal operators $G_0$, $F_0$ and $W$. 
5.2 Solutions to selected exercises

Exercise 5.2.
1. The formula $X G_0 \varphi \to G_0 X \varphi$ is valid. Let $\sigma$ be an arbitrary interpretation and assume that
   \[ \sigma, 0 \models X G_0 \varphi. \]
   Then, by definition of the satisfaction relation,
   \[ \sigma, 1 \models G_0 \varphi \]
   which implies that
   \[ \sigma, k \models \varphi, \text{ for all } k \geq 1. \]
   Hence,
   \[ \sigma, k \models X \varphi, \text{ for all } k \geq 0 \]
   and so,
   \[ \sigma, 0 \models G_0 X \varphi. \]
   This means that
   \[ \sigma, 0 \models X G_0 \varphi \to G_0 X \varphi. \]
   Consequently, $\sigma, 0 \models X G_0 \varphi \to G_0 X \varphi$, and, as $\sigma$ is arbitrary, we may conclude that the formula is valid.

4. The formula $F_0 \varphi \to G_0 \varphi$ is not valid, in general. Let $\varphi = p$ and consider the interpretation $\sigma$ such that $\sigma(1) = \{p\}$ and $\sigma(k) = \emptyset$, for $k \neq 1$. Then
   \[ \sigma, 1 \models p \text{ implies that } \sigma, 0 \models F_0 p \]
   and
   \[ \sigma, 2 \not\models p \text{ implies that } \sigma, 0 \not\models G_0 p. \]
   Hence, $\sigma, 0 \not\models F_0 \varphi \to G_0 \varphi$. Consequently, $\sigma \not\models F_0 \varphi \to G_0 \varphi$ and the formula is not valid. However, it is satisfiable. The interpretation $\sigma'$ such that $\sigma'(k) = \{p\}$, for every $k \in \mathbb{N}$ satisfies the formula. We leave it as an exercise to check this fact.

Exercise 5.4.
(a) Let $\sigma \not\in \mathcal{P}$. Then,
   \[ \sigma, 0 \not\models (p \to X \neg q) \lor (p \land q). \]
CHAPTER 5. LINEAR TEMPORAL LOGIC

This means that there is $k \geq 0$ such that

$$\sigma, k \not\models (p \rightarrow X \neg q) \quad \text{and} \quad \sigma, k \not\models (p \land q)$$

and, for every $0 \leq j < k$,

$$\sigma, j \models (p \rightarrow X \neg q) \quad \text{and} \quad \sigma, j \not\models (p \land q).$$

Then,

$$\sigma, k \models p \quad \text{and} \quad \sigma, k + 1 \models q \quad \text{and} \quad \sigma, k \not\models q.$$  (*)

Furthermore, for every $0 \leq j < k$,

$$\text{if } \sigma, j \models p \text{ then } \sigma, j \not\models q \text{ and } \sigma, j + 1 \not\models q.$$  (‡)

Let $\hat{\sigma} = \sigma[..k]$. Clearly, any trace $\sigma'$ such that $\hat{\sigma} \in \text{pref}(\sigma')$ will not satisfy $\varphi$ and, consequently, will not be in $P$. So, $\hat{\sigma}$ is a bad prefix and $P$ is a safety property.

(b) Observe that (†) and (‡) describe the behavior of bad prefixes. It is not very difficult to see that a possible NFA for $\text{BadPref}(P)$ is

![NFA diagram]

(c) First, we observe that

$$((p \rightarrow X \neg q) \cup (p \land q)) \equiv ((p \rightarrow X \neg q) \land (p \land q)) \land F_0(p \land q).$$

Let $P'_{\text{safe}} = P = \text{Mod}(\varphi)$, that we already proved to be a safety property. Then, let $P'_{\text{live}} = \text{Mod}(\varphi'')$. It is straightforward to prove that $P'_{\text{live}}$ is, indeed, a liveness property. Finally, observe that

$$P' = P'_{\text{safe}} \cap P'_{\text{live}}.$$

To prove that $P'_{\text{live}}$ is a liveness property, consider any finite sequence $\hat{\sigma} \in (2^\{p,q\})^*$, and let $\sigma = \hat{\sigma}\{p,q\}0^\varphi$. Clearly, $\sigma, 0 \models F_0(p \land q)$, and, so,
\( \sigma \in P'_{\text{live}} \). Consequently, \( \text{pref}(P'_{\text{live}}) = (2^{\{p,q\}})^* \), and we can conclude that \( P'_{\text{live}} \) is a liveness property.

**Exercise 5.5.**

(a) Consider the formula \( G_0 (p \lor \neg X q) \). At each instant, if \( p \) holds then there is that is a good state. If \( p \) does not hold, then, we must avoid that in the next state \( q \) holds. An NBA for this formula is presented below. It should be clear why the accept states are \( q_0 \) and \( q_1 \).

![NBA for formula](image)

**Exercise 5.6.**

We start by a solution for \( \varphi_1 \).

(a) We present a nonblocking NBA \( A_{-\varphi_1} \) for \( -\varphi_1 \). In this case,

\[
-\varphi_1 \equiv - (G_0 F_0 p \rightarrow G_0 F_0 q) \\
\equiv G_0 F_0 p \land \neg G_0 F_0 q \\
\equiv G_0 F_0 p \land F_0 G_0 \neg q
\]

and the intended NBA is

![NBA for formula](image)

Note that this NBA is nonblocking, as required by the model checking algorithm.
(b) We present the reachable part of \( T \otimes A_{\neg \varphi_1} \), omitting the details of the construction that have been explained in previous exercises.

\[
\langle s_0, q_0 \rangle \langle s_1, q_0 \rangle \langle s_2, q_0 \rangle \langle s_2, q_1 \rangle \langle s_3, q_0 \rangle \langle s_3, q_1 \rangle \langle s_1, q_3 \rangle
\]

(c) The algorithm will try to verify if \( T \otimes A_{\neg \varphi_1} \models P_{\text{pers}}(A_{\neg \varphi_1}) \). The only accept state in \( A_{\neg \varphi_1} \) is \( q_1 \). Hence \( P_{\text{pers}}(A_{\neg \varphi_1}) \) is the persistence property \( P_{\text{pers}}^{\neg q_1} \) induced by the formula \( \neg q_1 \). By Theorem 4.44, \( T \otimes A_{\neg \varphi_1} \not\models P_{\text{pers}}(A_{\neg \varphi_1}) \) if there is a reachable state \( s \) such that \( s \not\models \neg q_1 \) (i.e., \( s \models q_1 \)) and that occurs in a cycle. The only states that satisfy the formula \( q_1 \) are \( \langle s_2, q_1 \rangle \) and \( \langle s_3, q_1 \rangle \). Consider the path

\[
\pi' = \langle s_0, q_0 \rangle \langle s_1, q_0 \rangle \langle s_3, q_1 \rangle \omega
\]

This path shows that state \( \langle s_3, q_1 \rangle \) is reachable and that it occurs in a cycle. Hence,

\[
T \otimes A_{\neg \varphi_1} \not\models P_{\text{pers}}(A_{\neg \varphi_1})
\]

and, by Theorem 4.42,

\[
T \not\models G_0 F_0 p \rightarrow G_0 F_0 q.
\]

Consider the projection of \( \pi' \) in \( T \), that is, consider the path

\[
\pi = s_0 s_1 s_3 \omega
\]
5.2. SOLUTIONS TO SELECTED EXERCISES

in \( T \). The trace of this path is

\[
\sigma = \{p, q\} \cup \{p\}^\omega
\]

and \( \sigma \not\models G_o F_o p \rightarrow G_o F_o q \). We leave as an exercise to show this last result.

Now, we present a solution for \( \varphi_2 \).

(a) In this case, \( \neg \varphi_2 \) is

\[
\neg \varphi_2 \equiv \neg(X(p \land Xp)) \\
\equiv X(\neg q \lor X\neg p)
\]

and a nonblocking NBA \( A_{\neg \varphi_2} \) for \( \neg \varphi_2 \) is

(b) We present the reachable part of \( T \otimes A_{\neg \varphi_2} \).
As in the previous case, the algorithm will try to verify if $T \otimes A_{\neg \varphi_2} \models P_{\text{pers}(A_{\neg \varphi_2})}$. In this case, the only accept state in $A_{\neg \varphi_2}$ is $q_2$. Hence $P_{\text{pers}(A_{\neg \varphi_2})}$ is the persistence property $P_{\neg q_2}$ induced by the formula $\neg q_2$.

By Theorem 4.44, $T \otimes A_{\neg \varphi_2} \not\models P_{\text{pers}(A_{\neg \varphi_2})}$ if there is a reachable state $s$ such that $s \not\models \neg q_2$ (i.e., $s \models q_2$) and that occurs in a cycle. The states that satisfy the formula $q_2$ are $\langle s_1, q_2 \rangle$, $\langle s_2, q_2 \rangle$ and $\langle s_3, q_2 \rangle$. Consider the path

$$\pi' = \langle s_0, q_1 \rangle \langle s_1, q_2 \rangle \langle s_3, q_2 \rangle^\omega.$$ 

This path shows that state $\langle s_3, q_2 \rangle$ is reachable and that it occurs in a cycle. Hence,

$$T \otimes A_{\neg \varphi_2} \not\models P_{\text{pers}(A_{\neg \varphi_2})}$$

and, by Theorem 4.42,

$$T \not\models X(p \land (Xp)).$$

Consider the projection of $\pi'$ in $T$, that is, consider the path

$$\pi = s_0 \ s_1 \ s_3^\omega$$

in $T$. The trace of this path is

$$\sigma = \{p, q\} \emptyset \{p\}^\omega$$

and $\sigma \not\models X(p \land (Xp))$.

**Exercise 5.7.**

Let $\varphi = G_0(p \rightarrow F_0 r)$. The first step is to construct an NBA $A_{\neg \varphi}$ for $\neg \varphi$. We observe that

$$\neg G_0(p \rightarrow F_0 r) \equiv F_0 \neg(p \rightarrow F_0 r) \equiv F_0(p \land \neg F_0 r) \equiv F_0(p \land G_0 \neg r)$$

Hence, we can use the NBA provided.

The second step consists in constructing the product $T \otimes A_{\neg \varphi}$. We present only the reachable part of this construction.
The last step is the verification of $T \models \varphi$. By Theorem 4.42, this is equivalent to verify $T \otimes A \neg \varphi \models P_{\text{pers}}(A \neg \varphi)$, where $P_{\text{pers}}(A \neg \varphi)$ is the persistence property $P_{\text{pers}}^{q_1}$ induced by the formula $\neg q_1$, because $q_1$ is the only accept state. By Theorem 4.44, $T \otimes A \neg \varphi \not\models P_{\text{pers}}(A \neg \varphi)$ if there is a reachable state $s$ such that $s \not\models \neg q_1$ (i.e., $s \models q_1$) and that occurs in a cycle. The states that satisfy the formula $q_1$ are $\langle s_0, q_1 \rangle$ and $\langle s_1, q_1 \rangle$. Consider the path $\pi' = (\langle s_0, q_1 \rangle \langle s_1, q_1 \rangle)^\omega$. This path shows that the state $\langle s_1, q_1 \rangle$ is reachable and occurs in a cycle. Hence, $T \otimes A \neg \varphi \not\models P_{\text{pers}}(A \neg \varphi)$ and so, $T \not\models G_0(p \rightarrow F_0 r)$.

To find the counter-example, we only need to project the path $\pi'$ into a path $\pi$ in $T$: $\pi = (s_0 s_1)^\omega$.

The trace $\sigma$ of this path is $\sigma = (\{p\} \emptyset)^\omega$.

And this trace clearly does not satisfy $G_0(p \rightarrow F_0 r)$.

**Exercise 5.9.**

The states of $G_\varphi$ are the elementary sets $\pi \subseteq CL(\langle \rangle \varphi)$. So, we start by defining the set of subformulas and the closure of $\varphi$:

$$SF(\varphi) = \{p, (\neg p) \land q, \neg p, q, p \cup ((\neg p) \land q)\}$$
and

\[ CL(\varphi) = \{ p, (\neg p), q, \neg q, (\neg p) \land q, \neg((\neg p) \land q), p \cup ((\neg p) \land q), \neg(p \cup ((\neg p) \land q)) \}. \]

The elementary sets of \( CL(\varphi) \) are:

- \( \overline{v}_0 = \{ p, q, \neg((\neg p) \land q), p \cup ((\neg p) \land q) \}; \)
- \( \overline{v}_1 = \{ p, q, \neg((\neg p) \land q), \neg(p \cup ((\neg p) \land q)) \}; \)
- \( \overline{v}_2 = \{ p, \neg q, \neg((\neg p) \land q), p \cup ((\neg p) \land q) \}; \)
- \( \overline{v}_3 = \{ p, \neg q, \neg((\neg p) \land q), \neg(p \cup ((\neg p) \land q)) \}; \)
- \( \overline{v}_4 = \{ \neg p, q, (\neg p) \land q, p \cup ((\neg p) \land q) \}; \)
- \( \overline{v}_5 = \{ \neg p, \neg q, \neg((\neg p) \land q), \neg(p \cup ((\neg p) \land q)) \}. \)

Let us see why some of these sets are in fact elementary. Let us consider \( \overline{v}_0 \). First, we observe that it must be consistent with respect to propositional logic. So, if \( p \in \overline{v}_0 \) and \( q \in \overline{v}_0 \) it cannot be the case that \( (\neg p) \land q \in \overline{v}_0 \) because that would imply that \( \neg p \in \overline{v}_0 \) and if \( p \in \overline{v}_0 \) then \( \neg p \notin \overline{v}_0 \). Furthermore, we also don’t have \( p \land q \in \overline{v}_0 \) because \( p \land q \notin CL(\varphi). \) So, if \( (\neg p) \land q \notin \overline{v}_0 \) and as \( \overline{v}_0 \) has to be maximal, we have \( \neg((\neg p) \land q) \in \overline{v}_0 \). Finally, \( \overline{v}_0 \) is locally consistent with respect to the until operator. In this case, as \( (\neg p) \land q \notin \overline{v}_0 \) we can choose to have \( p \cup ((\neg p) \land q) \in \overline{v}_0 \), because \( p \in \overline{v}_0 \). But, if we choose for \( p \cup ((\neg p) \land q) \) not to be in the set then we obtain \( \overline{v}_1 \).

Consider now \( \overline{v}_4 \). In this case, as \( \neg p \in \overline{v}_4 \) and \( q \in \overline{v}_4 \) it must also be the case that \( (\neg p) \land q \in \overline{v}_4 \) by local consistency with respect to propositional logic. And, as \( (\neg p) \land q \in \overline{v}_4 \) it must also be the case that \( p \cup ((\neg p) \land q) \in \overline{v}_4 \) by local consistency with respect to the until operator. Finally, let us consider \( \overline{v}_5 \). As \( \neg p \in \overline{v}_5 \) and \( \neg q \in \overline{v}_4 \) then, as before, \( \neg((\neg p) \land q) \in \overline{v}_5 \). But, in this case, it must also be the case that \( \neg(p \cup ((\neg p) \land q)) \in \overline{v}_5 \) by local consistency with respect to the until operator. If \( p \cup ((\neg p) \land q) \in \overline{v}_5 \) then, as \( (\neg p) \land q \notin \overline{v}_5 \), it must be the case that \( p \in \overline{v}_5 \), by local consistency with respect to the until operator. And this cannot be.

Let

\[ A = \{ \overline{v}_0, \overline{v}_1, \overline{v}_2, \overline{v}_3, \overline{v}_4, \overline{v}_5 \} \]
5.2. SOLUTIONS TO SELECTED EXERCISES

(a) The initial states are all the elementary sets \( \pi \in Q \) such that \( \varphi \in \pi \). So, the set \( I \) of initial states is

\[
I = \{ \pi_0, \pi_2, \pi_4 \}.
\]

(b) We only have one formula with the until operator, that is \( \varphi \). So \( F = \{ F_\varphi \} \) and

\[
F_\varphi = \{ \pi \in Q \mid \varphi \notin \pi \text{ or } (\neg p) \land q \in \pi \} = \{ \pi_1, \pi_3, \pi_4 \}.
\]

(c) Let \( \sigma = \{ p \} \{ p \} \{ p, q \} \{ q \}^\omega \). Recall that \( \delta(\pi, \nu') \neq \emptyset \) provided that the propositional symbols in \( \pi \) and \( \nu' \) are the same. In this case, we have three initial states and we want to accept \( \{ p \} \). Then, we must choose \( \pi_2 \) because this is the only initial state that has exactly the same propositional symbols as \( \{ p \} \). So, we have

\[
\pi_2 \xrightarrow{\{ p \}} \ldots
\]

By definition of \( \delta \), as \( p, p \cup ((\neg p) \land q) \in \pi_2 \), we have that

\[
\delta(\pi_2, \{ p \}) = \{ \pi \in Q \mid p \cup ((\neg p) \land q) \in \pi \} = \{ \pi_0, \pi_2, \pi_4 \}.
\]

The second symbol we want to accept is \( \{ p \} \) so, again, we must choose \( \pi_2 \). Hence, we have

\[
\pi_2 \xrightarrow{\{ p \}} \pi_2 \xrightarrow{\{ p \}} \ldots
\]

Again, \( \delta(\pi_2, \{ p \}) = \{ \pi_0, \pi_2, \pi_4 \} \). But now, the next symbol we want to accept is \( \{ p, q \} \) and from \( \{ \pi_0, \pi_2, \pi_4 \} \) the only state that has the same propositional symbols as \( \{ p, q \} \) is \( \pi_0 \). So, we get,

\[
\pi_2 \xrightarrow{\{ p \}} \pi_2 \xrightarrow{\{ p \}} \pi_0 \xrightarrow{\{ p, q \}} \ldots
\]

Once again, by definition of \( \delta \), as \( p, p \cup ((\neg p) \land q) \in \pi_0 \), we have that

\[
\delta(\pi_0, \{ p, q \}) = \{ \pi \in Q \mid p \cup ((\neg p) \land q) \in \pi \} = \{ \pi_0, \pi_2, \pi_4 \}.
\]

The next symbol we want to accept is \( \{ q \} \) so we must choose \( \pi_4 \). Hence,

\[
\pi_2 \xrightarrow{\{ p \}} \pi_2 \xrightarrow{\{ p \}} \pi_0 \xrightarrow{\{ p, q \}} \pi_4 \xrightarrow{\{ q \}} \ldots
\]
By definition of $\delta$, as $(\neg p) \land q \in v_4$, we have that $\delta(v_4, \{q\}) = Q$. But the next symbol we want to accept is again $\{q\}$ so we must choose $v_4$ again. And so, it is not very difficult to see that

$$v_2 \xrightarrow{\{p\}} v_2 \xrightarrow{\{p\}} v_0 \xrightarrow{\{p,q\}} v_4 \xrightarrow{\{q\}} v_4 \xrightarrow{\{q\}} \ldots$$

that is, $v_2v_2(v_4)^\omega$ is the only initial run for $\sigma$. Observe that this run is initial because $v_2 \in I$ and it is accepting because $v_4 \in F\varphi$ and, so, $F\varphi$ is visited infinitely often.

(d) Let $\sigma_1 = \{p\}\omega$. From what we said above, in order to accept $\{p\}$ we must choose the initial state $v_2$. Furthermore, $\delta(v_2, \{p\}) = \{v_0, v_2, v_4\}$. And, as we want to accept $\{p\}$ must choose $v_2$ again. So we have

$$v_2 \xrightarrow{\{p\}} v_2 \xrightarrow{\{p\}} v_2 \xrightarrow{\{p\}} \ldots$$

that is, the only initial run for $\sigma_1$ is $(v_2)^\omega$. This run is not accepting because $v_2 \notin F\varphi$. Let $\sigma_2 = \{p\} \{p\} \{p,q\}\omega$. In this case, the first two states have to be $v_2$, as for $\sigma_1$. Hence, $v_2 \xrightarrow{\{p\}} v_2 \xrightarrow{\{p\}} \ldots$

Then next symbol we want to accept is $\{p,q\}$, and, as $\delta(v_2, \{p\}) = \{v_0, v_2, v_4\}$, we must choose $v_0$. Thus, $v_2 \xrightarrow{\{p\}} v_2 \xrightarrow{\{p\}} v_0 \xrightarrow{\{p,q\}} \ldots$

Finally, as $\delta(v_0, \{p,q\}) = \{v_0, v_2, v_4\}$, and we want to accept $\{p,q\}$ again, we have to choose $v_0$. So, we have $v_2 \xrightarrow{\{p\}} v_2 \xrightarrow{\{p\}} v_0 \xrightarrow{\{p,q\}} v_0 \xrightarrow{\{p,q\}} \ldots$

that is, the only initial run for $\sigma_2$ is $v_2v_2(v_0)^\omega$, and this run is not accepting because $v_0 \notin F\varphi$ and $v_0$ is the only state that is visited infinitely often.

**Exercise 5.10.**

Let us consider the temporal operator $G_0$, in the context of some formula $\varphi$. 
5.2. SOLUTIONS TO SELECTED EXERCISES

In this case, we need to adapt the notion of elementary set, the transition relation and the set of accepting states. Recall the following equivalence

\[ G_\circ \psi \equiv \psi \land X G_\circ \psi. \]

The notion of elementary set needs to be adapted to cope with the local properties of the temporal operator. Hence, we say that \( \overline{\nu} \) is locally consistent with respect to the temporal operator \( G \) if, for \( G \psi \in CL(\varphi) \), \( \overline{\nu} \) satisfies the following condition

- if \( G \psi \in \overline{\nu} \) then \( \psi \in \overline{\nu} \).

Next, we need to adapt the definition of the transition relation in the definition of the GNBA \( G_\varphi \). In this case, we need to cope with the \( X \) operator that appears in the inductive definition presented above.

Thus, \( \delta : Q \times 2^\Xi \rightarrow 2^Q \) is such that for \( v \in 2^\Xi \) and \( \overline{\nu}_1 \in Q \) such that \( v = \overline{\nu}_1 \cap \Xi \) we have that \( \delta(\overline{\nu}_1, v) \) is the set of all elementary sets \( \overline{\nu}_2 \) such that, for \( G \psi \in CL(\varphi) \)

\[ G \psi \in \overline{\nu}_1 \text{ iff } G \psi \in \overline{\nu}_2. \]

Finally, we need to consider the accept states. Consider the following trace

\[ \overline{\nu}_0 \longrightarrow \overline{\nu}_1 \longrightarrow \overline{\nu}_2 \longrightarrow \overline{\nu}_3 \longrightarrow \ldots \]

such that \( \psi \in \overline{\nu}_i \), for \( i \geq 0 \). Then, this trace satisfies \( G \psi \). Hence, \( G \psi \) must be in each of these states. To this end, for each \( G \psi \in CL(\varphi) \), we consider a set of accept states (that must be included in \( F_\varphi \)):

\[ F_{G\psi} = \{ \overline{\nu} \in Q \mid G \psi \in \overline{\nu} \text{ or } \psi \notin \overline{\nu} \}. \]
Exercises 6.1. Consider the transition system $T$

-describing a traffic light where the yellow light blinks before changing to red. Verify which of the following CTL formulas hold in this transition system:

(a) $AFy$

(b) $AGy$

(c) $AGAFy$

(d) $AFg$

(e) $AGAFg$
(f) $\text{EF } g$

(g) $\text{AG EF } g$

(h) $\text{A}(b \lor \neg b)$

(i) $\text{A}(\neg b \lor b)$

**Exercise 6.2.** Let $T$ be a finite transition system without terminal states and $s$ one of its states. Prove or disprove the following assertions:

(a) If $T, s \models \text{EG } p$ then $T, s \models \text{AG } p$.

(b) If $T, s \models \text{AG } p$ then $T, s \models \text{EG } p$.

(c) If $T, s \models \text{EF } p \lor \text{EF } q$ then $T, s \models \text{EF } (p \lor q)$.

(d) If $T, s \models \text{EF } (p \lor q)$ then $T, s \models \text{EF } p \lor \text{EF } q$.

**Exercise 6.3.** Prove or disprove the following equivalences between CTL formulas:

(a) $\text{AX AF } \alpha \equiv \text{AF AX } \alpha$

(b) $\text{EX EF } \alpha \equiv \text{EF EX } \alpha$

(c) $\text{AX AG } \alpha \equiv \text{AG AX } \alpha$

(d) $\text{EX EG } \alpha \equiv \text{EG EX } \alpha$

(e) $\text{EF EG } \alpha \equiv \text{EG EG } \alpha$

**Exercise 6.4.** Prove or disprove the following equivalences between LTL formulas and CTL formulas, where $p, q \in \mathcal{E}$:

1. $G_p X p \equiv \text{AG AX } p$

2. $F_p X p \equiv \text{AF AX } p$

3. $F_p (p \lor q) \equiv (\text{AF } p) \lor (\text{AF } q)$

**Exercise 6.5.** Prove, using Theorem 6.16, that there is no LTL formula equivalent to the following CTL formulas:
6.1. EXERCISES

1. $\text{AF}(p \land \text{AX}p)$.
2. $\text{AF}(p \land \text{EX}p)$.

**Exercise 6.6.** Prove, without using Theorem 6.16, that there is no LTL formula equivalent to the CTL formula $\text{AF EX AF}p$.

**Exercise 6.7.** Consider the transition system $T$

![Transition System Diagram]

and the CTL formulas

$$\alpha_1 = \text{EF AG} r \quad \text{and} \quad \alpha_2 = \text{A} (p \mathbin{U} \text{AF} q).$$

Check if $T \models \alpha_i$, for $i = 1, 2$, using the CTL model checking algorithm.

**Exercise 6.8.** Consider the transition system $T$

![Transition System Diagram]

Check if $T_2 \models \text{A} (\text{AF} p \mathbin{U} q)$, using the CTL model checking algorithm.

**Exercise 6.9.** Show that there exist two transition systems $T$ and $T'$ without terminal states and a CTL formula $\alpha$ such that
1. $\text{Traces}(T) = \text{Traces}(T')$, and

2. $T \models \alpha$ and $T' \not\models \alpha$.

**Exercise 6.10.** Provide an algorithm for computing the sets $\text{Sat}(\cdot)$ for the remaining operators, without converting the formula into ENF.

**Exercise 6.11.** Let $\text{Var} = \{x_1, x_2, x_3, x_4, x_5, x_6\}$ and $\varphi = (x_1, x_2, x_3, x_4, x_5, x_6)$. Consider the binary function $f$ for $\text{Var}$ defined by

$$f([x_1 = b_1, x_2 = b_2, x_3 = b_3, x_4 = b_4, x_5 = b_5, x_6 = b_6]) = \begin{cases} 1 & \text{if } \sum_{i=1}^{6} b_i \geq 4 \\ 0 & \text{otherwise.} \end{cases}$$

Depicted the $\varphi$-ROBDD for $f$. 
6.2 Solutions to selected exercises

Exercise 6.1.

We start by computing the satisfaction sets for each formula. Then, $T$ satisfies a formula if the set of initial states $I = \{s_0\}$ is a subset of the satisfaction set for that formula.

(a) The formula is $AFy$. In this case we have that:

- $s_0 \models AFy$: for every $\pi \in \text{Paths}(s_0)$ we have $\pi[2] = s_2$ and $s_2 \models y$;
- $s_1 \models AFy$: for every $\pi \in \text{Paths}(s_1)$ we have $\pi[1] = s_2$ and $s_2 \models y$;
- $s_2 \models AFy$: for every $\pi \in \text{Paths}(s_2)$ we have $\pi[0] = s_2$ and $s_2 \models y$;
- $s_3 \models AFy$: for every $\pi \in \text{Paths}(s_3)$ we have $\pi[1] = s_3$ and $s_2 \models y$.

Consequently, $\text{Sat}(AFy) = S$, and as $I \subseteq S$, we conclude that $T \models AFy$.

(b) The formula is $AGy$. In this case, we have that:

- $s_0 \not\models AGy$: there is at least one path $\pi \in \text{Paths}(s_0)$ such that $\pi[0] = s_0$ and $s_0 \not\models y$;
- $s_1 \not\models AGy$: there is at least one path $\pi \in \text{Paths}(s_1)$ such that $\pi[0] = s_1$ and $s_1 \not\models y$;
- $s_2 \not\models AGy$: there is at least one path $\pi \in \text{Paths}(s_2)$ such that $\pi[1] = s_0$ and $s_0 \not\models y$;
- $s_3 \not\models AGy$: there is at least one path $\pi \in \text{Paths}(s_3)$ such that $\pi[0] = s_3$ and $s_3 \not\models y$.

Hence, $\text{Sat}(AGy) = \emptyset$, and in this case, we conclude that $T \not\models AGy$.

(c) The formula is $AGAFy$. From (a), we know that $\text{Sat}(AFy) = S$, i.e., $AFy$ holds in every state. From this, it is immediate to conclude that $\text{Sat}(AGAFy) = S$.

(d) The formula is $AFg$. In this case we have that:

- $s_0 \models AFg$: for every $\pi \in \text{Paths}(s_0)$ we have $\pi[1] = s_1$ and $s_1 \models g$;
- $s_1 \models AFg$: for every $\pi \in \text{Paths}(s_1)$ we have $\pi[0] = s_1$ and $s_1 \models g$;
• $s_2 \not\models AFg$: consider the $\pi \in Paths(s_2)$ such that $\pi = (s_2 s_3)^\omega$. As $s_2 \not\models g$ and $s_3 \not\models g$ we conclude that $s_2 \not\models AFg$;

• $s_3 \not\models AFg$: consider the $\pi \in Paths(s_3)$ such that $\pi = (s_3 s_2)^\omega$. As $s_2 \not\models g$ and $s_3 \not\models g$ we conclude that $s_2 \not\models AFg$;

Consequently, $Sat(AFg) = \{s_0, s_1\}$, and as $I \subseteq \{s_0, s_1\}$, we conclude that $T \models AFg$.

(e) The formula is $AG AFg$. Recall that, from (d), we know that $Sat(AFg) = \{s_0, s_1\}$. Observe that any path is $T$ must go through $s_2$, and $s_2 \not\models AFg$. Hence, no path will satisfy $AFg$ in every state. Hence, $Sat(AG AFg) = \emptyset$. Consequently, $T \not\models AG AFg$.

Exercise 6.2.

(a) This statement is false. Consider the transition system

```
\begin{array}{cc}
\bullet & \bullet \\
\{p\} & \emptyset \\
\end{array}
```

and consider the path $\pi = (s_0)^\omega$. Clearly, we have $\pi[i] \models p$ for every $i \geq 0$. Hence, $s_0 \models EGp$.

On the other hand, consider the path $\pi' = s_0 s_1^\omega$. It is easy to see that $\pi'[1] \not\models p$, and so, there is a path starting in $s_0$, such that $p$ is not true in every state. Consequently, $s_0 \not\models AGp$.

(b) In order for this statement to be true it is essential that $T$ does not have terminal states. In this case, we know that $Paths(s) \neq \emptyset$, for any state $s \in S$. Assume that $s \not\models EGp$. Then, for every path $\pi \in Paths(s)$ there is $i \geq 0$ such that $\pi[i] \not\models p$. Let $\pi' \in Paths(s)$ (that exists because $T$ has no terminal states). Then, $\pi'[i] \not\models p$, for some $i \geq 0$, and so, $s \not\models AGp$.

Exercise 6.3.

(a) In order to show $AX AF\alpha \equiv AF AX\alpha$ we need to show that $Sat(AX AF\alpha) = Sat(AF AX\alpha)$. However, this equality does not hold. We start by showing that $Sat(AF AX\alpha) \subseteq Sat(AX AF\alpha)$ holds. Then, we show that $Sat(AX AF\alpha) \subseteq Sat(AF AX\alpha)$ does not hold, in general.
To prove the first condition, assume that \( s_0 \not\in \text{Sat}(\text{AX AF } \alpha) \). Then, there is \( s_1 \in \text{Suc}(s_0) \) such that \( s_1 \not\models \text{AF } \alpha \). This means that there is a path \( \pi' \in \text{Paths}(s_1) \) such that \( \pi'[i] \not\models \alpha \), for every \( i \geq 0 \). For convenience, let \( \pi' = s_1 s_2 s_3 \ldots \), for which it holds

\[
 s_i \not\models \alpha \quad \text{for every } i > 0.
\]

(†)

Consider now the path \( \pi = s_0 s_1 s_2 s_3 \ldots \). This is a path in \( \text{Paths}(s_0) \) because \( \pi' \in \text{Paths}(s_1) \) and \( s_1 \in \text{Suc}(s_0) \). Furthermore, by (†), for \( i = 1 \), it follows that

\[
 s_0 \not\models \text{AX } \alpha.
\]

Similarly, for \( i = 2 \), it follows that

\[
 s_1 \not\models \text{AX } \alpha.
\]

It is not very difficult to conclude that

\[
 \pi[i] \not\models \text{AX } \alpha \quad \text{for every } i \geq 0.
\]

and, so, it follows that

\[
 s_0 \not\models \text{AF AX } \alpha
\]

that is, \( s_0 \not\in \text{Sat}(\text{AF AX } \alpha) \). Hence, \( \text{Sat}(\text{AF AX } \alpha) \subseteq \text{Sat}(\text{AX AF } \alpha) \).

To prove the converse, consider the following transition system \( T \)

\[
\begin{array}{ccc}
\{p\} & \xrightarrow{} & \{p\} \\
\downarrow & & \downarrow \\
s_0 & \xrightarrow{} & s_1 \\
\emptyset & \xrightarrow{} & \{p\} \\
s_2 & \xrightarrow{} & \{p\}
\end{array}
\]

We now show that \( T \models \text{AX AF } p \), i.e., \( T, s_0 \models \text{AX AF } p \). Observe that

- \( s_0 \models \text{AF } p \): any path \( \pi \in \text{Paths}(s_0) \) will have as first state \( \pi[0] \) the state \( s_0 \), and \( s_0 \models p \);

- \( s_1 \models \text{AF } p \): the only path in \( \text{Paths}(s_1) \) is \( \pi = s_1 s_2 \omega \), and \( \pi[1] \models p \).

Consequently, as \( \text{Suc}(s_0) = \{ s_0, s_1 \} \), we may conclude that \( s_0 \models \text{AX AF } p \).

Finally, we show that \( T \not\models \text{AF AX } p \). In this case, we know that \( s_1 \not\models p \). Consequently, \( s_0 \not\models \text{AF } p \) because \( s_1 \in \text{Suc}(s_0) \). Now, consider the path \( \pi = s_0 \omega \in \text{Paths}(s_0) \). Clearly, \( \pi[i] \not\models \text{AX } p \), for every \( i \geq 0 \). Hence, \( s_0 \not\models \text{AF AX } p \),
and so, $T \not\models \mathsf{AF} \mathsf{AX} p$.

(b) We show that $\mathsf{EX} \mathsf{EF} \alpha \equiv \mathsf{EF} \mathsf{EX} \alpha$, that is, we show that $\mathit{Sat}(\mathsf{EX} \mathsf{EF} \alpha) = \mathit{Sat}(\mathsf{EF} \mathsf{EX} \alpha)$. Let $s_0 \in \mathit{Sat}(\mathsf{EX} \mathsf{EF} \alpha)$. Then, there is $s_1 \in \mathit{Suc}(s_0)$ such that

$$s_1 \models \mathsf{EF} \alpha$$

and, so, there a path $\pi \in \mathit{Paths}(s_1)$ such that

$$\pi[i] \models \alpha \quad \text{for some } i \geq 0.$$ 

Now, let $\pi' = s_0 \pi$ that is well defined because $s_1 \in \mathit{Suc}(s_0)$ and $\pi \in \mathit{Paths}(s_1)$. Observe that $\pi'[i + 1] = \pi[i]$, for every $i \geq 0$. Then,

$$\pi'[i + 1] \models \alpha \quad \text{for some } i \geq 0.$$ 

and, as $\pi'[i + 1] \in \mathit{Suc}(\pi'[i])$, then

$$\pi'[i] \models \mathsf{EX} \alpha \quad \text{for some } i \geq 0.$$ 

which implies that

$$s_0 \models \mathsf{EF} \mathsf{EX} \alpha.$$ 

Conversely, assume that $s_0 \in \mathit{Sat}(\mathsf{EF} \mathsf{EX} \alpha)$. Then, there is a path $\pi \in \mathit{Paths}(s_0)$ such that

$$\pi[i] \models \mathsf{EX} \alpha \quad \text{for some } i \geq 0$$

Then, there is $s'_{i+1} \in \mathit{Suc}(\pi[i])$ (not necessarily $\pi[i + 1]$) such that

$$s'_{i+1} \models \alpha.$$ 

Let $\pi' \cdots \in \mathit{Paths}(s'_{i+1})$ be an arbitrary path, that exists because $T$ has no terminal states. Now, consider the path

$$\pi'' = \pi[1] \cdots \pi[i] \pi'[0] \pi'[1] \cdots$$

which is clearly a well defined path in $\mathit{Paths}(s_1)$, and as $s'_{i+1} \models \alpha$, it follows that

$$\pi''[0] \models \mathsf{EF} \alpha.$$ 

Finally, as $\pi''[0] = \pi[1] \in \mathit{Suc}(s_0)$, it also follows that

$$s_0 \models \mathsf{EX} \mathsf{EF} \alpha.$$
That is, $s_0 \in \text{Sat} (\text{EX EF } \alpha)$.

**Exercise 6.4.**
Recall that an LTL formula $\varphi$ and a CTL formula $\alpha$ are equivalent if their are satisfied by the same transition systems.

(a) We want to prove that $G \circ X p \equiv AG AX p$. Let $T$ be an arbitrary transition system. We need to show that

$$T \vDash G \circ X p \quad \text{if and only if} \quad AG AX p.$$  
Assume that

$$T \not\vDash G \circ X p.$$  
Then, there is $\sigma \in \text{Traces}(T)$ such that

$$\sigma \not\vDash G \circ X p.$$  
This means that there is $i \geq 0$ such that

$$\sigma, i \not\vDash X p$$  
that is,

$$\sigma, i + 1 \not\vDash p.$$  
Let $\pi \in \text{Paths}(T)$ be a path such that $\text{trace}(\pi) = \sigma$. Then, it follows that

$$\pi[i + 1] \not\vDash p.$$  
And, by definition of path, $\pi[i + 1] \in \text{Suc}(\pi[i])$. So

$$\pi[i] \not\vDash AX p$$  
which implies that

$$\pi[0] \not\vDash AG AX p.$$  
As $\pi[0] \in I$ (because $\pi \in \text{Paths}(T)$), then

$$T \not\vDash AG AX p.$$  
Conversely, assume that

$$T \not\vDash AG AX p.$$
Then, there is $\pi \in \text{Paths}(T)$ such that

$$\pi[i] \not\models \text{AX} p$$

So, there is $s'_{i+1} \in \text{Suc}(\pi[i])$ (not necessarily $\pi[i + 1]$) such that

$$s'_{i+1} \not\models p.$$ Let $\pi' \in \text{Paths}(s'_{i+1})$ be an arbitrary path and consider the path

$$\pi'' = \pi[0] \ldots \pi[i] \pi'[0] \pi'[1] \ldots \in \text{Paths}(s_0)$$

Observe that $\pi''[i + 1] = \pi'[0] = s'_{i+1}$, and so $\pi''[i + 1] \not\models p$. Now, let $\sigma'' = \text{trace}(\pi'')$. Clearly,

$$\sigma'', i + 1 \not\models p$$

which implies that

$$\sigma'', i \not\models \text{XP}$$

and also

$$\sigma, 0 \not\models \text{G}_0 \text{XP}.$$ Hence,

$$T \not\models \text{G}_0 \text{XP}.$$ (b) The assertion $F_0 \text{XP} \equiv \text{AF AX} p$ is false. Consider the transition system $T$ depicted below

```
s_0 \rightarrow s_1 \rightarrow s_2
\{p\} \rightarrow \emptyset \rightarrow \{p\}
```

In this transition system, we have the following paths:

$$\pi = s_0^\omega \quad \text{and} \quad \pi_k = s_0^k s_1 s_2^\omega \quad \text{for} \quad k > 0.$$ The traces of these paths are, respectively,

$$\sigma = \{p\}^\omega \quad \text{and} \quad \sigma_k = \{p\}^k \emptyset \{p\}^\omega \quad \text{for} \quad k > 0.$$ It is not very difficult to prove that $\sigma \models F_0 \text{XP}$ and $\sigma_k \models F_0 \text{XP}$, for $k > 0$. Hence, $T \models F_0 \text{XP}$. 
Now, observe that $s_1 \not\models p$ and $s_1 \in \text{Suc}(s_0)$. Hence,

$$s_0 \not\models AX p.$$ 

Then, it follows that

$$\pi[i] \not\models AX p \quad \text{for every } i \geq 0$$

which implies that, as $\pi \in \text{Paths}(s_0)$,

$$s_0 \not\models AF AX p.$$ 

Hence, $T \not\models AF AX p$, and so $F \circ X p \not\equiv AF AX p$.

(c) The assertion

$$F_c(p \lor q) \equiv (AF p) \lor (AF q)$$

is false. Consider the transition system $T$

![Diagram](image)

The only paths in this transition system are:

$$\pi_1 = s_0 s_1^\omega \quad \text{and} \quad \pi_2 = s_0 s_2^\omega.$$ 

The traces of these paths are, respectively,

$$\sigma_1 = \emptyset \{p\}^\omega \quad \text{and} \quad \sigma_2 = \emptyset \{q\}^\omega.$$ 

Again, it is easy to prove that $\sigma_1 \models F_c(p \lor q)$ and $\sigma_2 \models F_c(p \lor q)$. Hence, $T \models F_c(p \lor q)$.

However,

$$\pi_1[i] \not\models q \quad \text{for every } i \geq 0$$
so, as $\pi_1 \in \text{Paths}(s_0)$, $s_0 \not\models \text{AF } q$.

Similarly, $\pi_2[i] \not\models p$ for every $i \geq 0$

and again, as $\pi_2 \in \text{Paths}(s_0)$, $s_0 \not\models \text{AF } p$.

Then, it follows that $s_0 \not\models \text{AF } p \lor \text{AF } q$.

**Exercise 6.5.**

(a) By Theorem 6.16, either the LTL formula $F_r(p \land Xp)$ is equivalent to $\text{AF}(p \land AXp)$ or $\text{AF}(p \land AXp)$ has no LTL formula equivalent to it.

Consider the following transition system

```
\begin{align*}
\text{s}_0 & \xrightarrow{\{p\}} \{p\} \\
\{p\} & \xrightarrow{\emptyset} \emptyset \quad \{p\} \xrightarrow{\emptyset} \emptyset \\
\emptyset & \xrightarrow{\{p\}} \{p\} \quad \emptyset & \xrightarrow{\{p\}} \{p\}
\end{align*}
```

The paths of this transition system are $\pi_1 = s_0 s_1 s^\omega_2$ and $\pi_2 = s_0 s_3 s^\omega_4$.

The traces of these paths are, respectively, $\sigma_1 = \{p\} \{p\} \emptyset^\omega$ and $\sigma_2 = \{p\} \emptyset \{p\}^\omega$.

Now, observe that $\sigma_1, 1 \models p$

so $\sigma_1, 0 \models Xp$. 
Furthermore, 

\[ \sigma_1, 0 \models p. \]

So, 

\[ \sigma_1, 0 \models p \land Xp \]

which implies that 

\[ \sigma_1, 0 \models F_\sigma(p \land Xp). \]

Hence, 

\[ \sigma_1 \models F_\sigma(p \land Xp). \]

Similarly, 

\[ \sigma_2, 4 \models p \]

so 

\[ \sigma_2, 3 \models Xp \]

Additionally, 

\[ \sigma_2, 3 \models p \]

so 

\[ \sigma_2, 3 \models p \land Xp \]

which implies that 

\[ \sigma_2, 0 \models F_\sigma(p \land Xp) \]

that is, 

\[ \sigma_2 \models F_\sigma(p \land Xp). \]

Hence, we may conclude that \( T \models F_\sigma(p \land Xp). \)

In the case of the CTL formula we have that as 

\[ s_2 \not\models p \]

then 

\[ s_2 \not\models p \land AXp. \]

Additionally, as \( s_2 \in \text{Suc}(s_1) \), then 

\[ s_1 \not\models AXp \]

and so 

\[ s_1 \not\models p \land AXp. \]
Finally, as $s_3 \in \text{Suc}(s_0)$ and $s_3 \not\Vdash p$, then

$$s_0 \not\Vdash AXp$$

and so

$$s_0 \not\Vdash p \land AXp.$$  

Now, consider path $\pi_1 \in \text{Paths}(s_0)$. Clearly,

$$\pi[i] \not\Vdash p \land AXp \quad \text{for every } i \geq 0$$

so

$$s_0 \not\Vdash AF(p \land AXp).$$

Hence, $T \not\models AF(p \land AXp)$, and so $F_0(p \land Xp) \not\equiv AF(p \land AXp)$. By Theorem 6.16, the formula $AF(p \land AXp)$ has no equivalent LTL formula.

(b) Consider the formulas $AF(p \land EXp)$ and $F_0(p \land Xp)$. We show that these two formulas are not equivalent and, consequently, by Theorem 6.16, the formula $AF(p \land EXp)$ has no equivalent LTL formula.

Consider the transition system

```
```

The only paths in this transition system are:

$$\pi_1 = s_0 s_1^\omega \quad \text{and} \quad \pi_2 = s_0 s_2^\omega.$$  

The traces of these paths are, respectively,

$$\sigma_1 = \{p\}^\omega \quad \text{and} \quad \sigma_2 = \{p\}^0.$$
Observe that
\[ \sigma_2, i \not\models p \quad \text{for } i > 0 \]
and, so,
\[ \sigma_2, i \not\models p \land Xp \quad \text{for } i > 0 \]
Furthermore, as \( \sigma_2, 1 \not\models p \), then
\[ \sigma_2, 0 \not\models Xp \]
so,
\[ \sigma_2, 0 \not\models p \land Xp \]
Hence,
\[ \sigma_2, i \not\models p \land Xp \quad \text{for } i \geq 0 \]
which means that
\[ \sigma_2, 0 \not\models F_\infty(p \land Xp) \]
and, consequently,
\[ T \not\models F_\infty(p \land Xp). \]

On the other hand,
\[ s_0 \models p \]
and, as \( s_1 \in \text{Suc}(s_0) \) and \( s_1 \models p \) then
\[ s_0 \models \text{EX} p. \]
Consequently,
\[ s_0 \models p \land \text{EX} p. \]
All the paths in \( \text{Paths}(s_0) \) start in \( s_0 \). So,
\[ s_0 \models \text{AF}(p \land \text{EX} p) \]
and, as \( s_0 \) is the only initial state, it follows that
\[ T \models \text{AF}(p \land \text{EX} p). \]
So, \( F_\infty(p \land Xp) \not\equiv \text{AF}(p \land \text{EX} p) \).

**Exercise 6.6.**
Assume that there is an LTL formula \( \varphi \), that is equivalent to the CTL formula \( \text{AF} \text{EX} \text{AF} p \). Let \( T_1 \) be the transition system
We claim that $T_1 \models \text{AF EX AF } p$. Observe that
\[
s_2 \models \text{AF } p
\]
because the only path starting in $s_2$ is $s_2^\omega$. Then, as $s_2 \in \text{Suc}(s_0)$, $s_2 \in \text{Suc}(s_1)$, and $s_2 \in \text{Suc}(s_2)$, we obtain that
\[
s_0 \models \text{EX AF } p \quad s_1 \models \text{EX AF } p \quad s_2 \models \text{EX AF } p.
\]
Given that any path in $T_1$ goes through $s_0$, $s_1$ or $s_2$ we conclude that
\[
T_1 \models \text{AF EX AF } p.
\]

As we are assuming that $\varphi \equiv \text{AF EX AF } p$ then $T_1 \models \varphi$, which means that $\text{Traces}(T_1) \subseteq \text{Mod}(\varphi)$. In particular, the trace $\sigma = \{p\}^\omega$ corresponding to the path $\pi = s_0 s_1^\omega$ is in $\text{Mod}(\varphi)$.

Consider now the transition system $T_2$ defined as follows:

The only trace in $T_2$ is precisely $\sigma = \{p\}^\omega$. Hence, we know that $T_2 \not\models \varphi$. And, as we are assuming that $\varphi \equiv \text{AF EX AF } p$ then we also have $T_2 \not\models \text{AF EX AF } p$. However, this is not the case. In fact, is is easy to prove that
\[
s_1' \not\models \text{AF } p.
\]
As $s_1'$ is the only successor of $s_0'$ and of $s_1'$ then
\[
s_0' \not\models \text{EX AF } p \quad \text{and} \quad s_1' \not\models \text{EX AF } p.
\]
The only path in $T_2$ is $s'_0 s'_1 \omega$. So, we may conclude that

$$s'_0 \not\models \text{AF EX AF } p$$

and, so, $T_2 \not\models \text{AF EX AF } p$, contradicting the initial assumption. Hence, we can conclude that the CTL formula $\text{AF EX AF } p$ has no equivalent LTL formula.

**Exercise 6.7.**

Consider the formula $\alpha_1 = \text{EF AG } r$. Before we can apply the model checking algorithm for CTL we need to transform this formula into its ENF equivalent.

$$\text{EF AG } r \equiv \text{EF } \neg \text{EF } \neg r$$

$$\equiv \text{EF } \neg \text{E}(\text{true } U \neg r)$$

$$\equiv \text{E}(\text{true } U \neg \text{E}(\text{true } U \neg r))$$

We now proceed to compute the $\text{Sat}(\cdot)$ sets, working bottom up in the structure of the formulas.

$$\text{Sat}(r) = \{s_2, s_3, s_4\}.$$ 

From this, we can compute the set $\text{Sat}(\neg r) = S \setminus \text{Sat}(r)$. Hence,

$$\text{Sat}(\neg r) = \{s_0, s_1\}.$$ 

Next, we compute

$$\text{Sat}(\text{E}(\text{true } U \neg r)).$$

In this case, we start with the set

$$C = \text{Sat}(\alpha'_2)$$

$$= \text{Sat}(\neg r)$$

$$= \{s_0, s_1\}.$$ 

Then, we start adding to $C$ states in $\text{Sat}(\alpha'_1) = \text{Sat}(\text{true}) = S$ that are not already in $C$ and have a successor in $C$. In this case, we have

$$\text{Sat}(\alpha'_1) \setminus C = \{s_2, s_3, s_4\}.$$
Since none of these states has a successor in $C$, the algorithm terminates and we conclude that

$$Sat(E(true \cup \neg r)) = \{s_0, s_1\}.$$  

Then,

$$Sat(\neg E(true \cup \neg r)) = S \setminus Sat(E(true \cup \neg r))$$

$$= \{s_2, s_3, s_4\}.$$  

Finally, we compute

$$Sat(\underbrace{E(true \cup \neg E(true \cup \neg r))}_{\alpha''_1} \cup \underbrace{E(true \cup \neg r)}_{\alpha''_2})$$

We start with the set

$$C = Sat(\alpha''_2) = Sat(\neg E(true \cup \neg r)) = \{s_2, s_3, s_4\}.$$  

Then, we start adding to $C$ the states in $Sat(\alpha''_1) = Sat(true) = S$ that are not already in $C$ and that have a successor in $C$. In this case,

$$Sat(\alpha''_1) \setminus C = \{s_0, s_1\}.$$  

The only state that has a successor in $C$ is $s_1$. So we add $s_1$ to $C$. After this iteration,

$$C = \{s_1, s_2, s_3, s_4\}.$$  

We repeat the process. In this case, we have

$$Sat(\alpha''_1) \setminus C = \{s_0\}.$$  

And the state $s_0$ has a successor in $C$ so we add it to $C$. After this iteration,

$$C = \{s_0, s_1, s_2, s_3, s_4\}.$$  

As $Sat(\alpha''_1) \setminus C = \emptyset$, the algorithm ends and we conclude that

$$Sat(E(true \cup \neg E(true \cup \neg r))) = \{s_0, s_1, s_2, s_3, s_4\}.$$
As $I = \{s_0\} \subseteq Sat(E(true \cup \neg E(true \cup \neg r)))$ it follows that
\[ T \models E(true \cup \neg E(true \cup \neg r)) \]
that is,
\[ T \models EF AG r. \]

Consider now the formula $\alpha_2 = A(p \cup AF q)$. Again, before we apply the model checking algorithm for CTL, we need to transform this formula into its ENF equivalent.
\[
A(p \cup AF r) \equiv \neg E(\neg AF q \cup (\neg p \land \neg AF q)) \land \neg EG \neg AF q \\
\equiv \neg E(EG \neg q \cup (\neg p \land EG \neg q)) \land \neg EG EG \neg q
\]

The case of propositional symbols and its negations is as follows
\[
Sat(p) = \{s_0, s_1\} \\
Sat(q) = \{s_1, s_2\} \\
Sat(\neg p) = S \setminus Sat(p) \\
= \{s_2, s_3, s_4\} \\
Sat(\neg q) = S \setminus Sat(q) \\
= \{s_0, s_3, s_4\}.
\]

Next, we compute $Sat(EG \neg q)$. In this case, we start we a set
\[
C = Sat(\neg q) \\
= \{s_0, s_3, s_4\}.
\]
and successively eliminate states from $C$ that do not have at least one successor in $C$. In this case, $s_0$ has as its only successor the state $s_1$ that is not in $C$. So, we remove $s_0$ from $C$. After this iteration, we have
\[
C = \{s_3, s_4\}.
\]
Observe that both $s_3$ and $s_4$ have successors in $C$. So, the algorithm terminates and we conclude that
\[
Sat(EG \neg q) = \{s_3, s_4\}.
\]
Next, we compute
\[
\text{Sat}(\neg p \land \text{EG} \neg q) = \text{Sat}(\neg p) \cap \text{Sat}(\text{EG} \neg q) = \{s_3, s_4\}.
\]

So, we can also compute
\[
\text{Sat}(\text{E}(\text{EG} \neg q \cup (\neg p \land \text{EG} \neg q)))
\]

We start by setting a set \(C\) to
\[
\text{Sat}(\alpha'_2) = \text{Sat}(\neg p \land \text{EG} \neg q) = \{s_3, s_4\}
\]

Then, we search for states in \(\text{Sat}(\alpha'_1) = \text{Sat}(\text{EG} \neg q) = \{s_3, s_4\}\) that are not already in \(C\) and that have a successor in \(C\). In this case, all the states in \(\text{Sat}(\alpha'_1)\) are already in \(C\) so the algorithm ends, and we conclude that
\[
\text{Sat}(\text{E}(\text{EG} \neg q \cup (\neg p \land \text{EG} \neg q))) = \{s_3, s_4\}.
\]

From this, we obtain
\[
\text{Sat}(\neg \text{E}(\text{EG} \neg q \cup (\neg p \land \text{EG} \neg q))) = S \setminus \text{Sat}(\text{E}(\text{EG} \neg q \cup (\neg p \land \text{EG} \neg q))) = \{s_0, s_1, s_2\}.
\]

Next, we compute \(\text{Sat}(\text{EG} \neg q)\). We start with a set
\[
C = \text{Sat}(\text{EG} \neg q) = \{s_3, s_4\}.
\]

Then, we successively remove from \(C\) all states that don’t have, at least, one successor in \(C\). In this case, both states have a successor in \(C\) (the state itself). So the algorithm ends, and we obtain
\[
\text{Sat}(\text{EG} \neg q) = \{s_3, s_4\}.
\]

From this, we can compute
\[
\text{Sat}(\neg \text{EG} \neg q) = S \setminus \text{Sat}(\text{EG} \neg q) = \{s_0, s_1, s_2\}.
\]
Finally, we compute
\[ Sat(\neg E(EG \neg q U (\neg p \land EG \neg q)) \land \neg EG EG \neg q) \]
\[ = Sat(\neg E(EG \neg q U (\neg p \land EG \neg q))) \land Sat(\neg EG EG \neg q) \]
\[ = \{s_0, s_1, s_2\} \cap \{s_0, s_1, s_2\} \]
\[ = \{s_0, s_1, s_2\}. \]

Hence, as \( I = \{s_0\} \subseteq Sat(\neg E(EG \neg q U (\neg p \land EG \neg q)) \land \neg EG EG \neg q) \) we conclude that
\[ T \models \neg E(EG \neg q U (\neg p \land EG \neg q)) \land \neg EG EG \neg q \]
that is,
\[ T \models A(p U AF q). \]

**Exercise 6.9.**
Consider the CTL formula \( \alpha = AX EX p \) and the transition systems \( T \)

![Diagram of transition system T]

and \( T' \)

![Diagram of transition system T']


The paths in $T$ are
\[ \pi_1 = s_0 s_1 s_2^\omega \quad \text{and} \quad \pi_2 = s_0 s_1 s_3^\omega \]
with traces
\[ \text{trace}(\pi_1) = \emptyset \emptyset \{p\}^\omega \quad \text{and} \quad \text{trace}(\pi_2) = \emptyset^\omega. \]
The paths in $T'$ are
\[ \pi'_1 = s'_0 s'_1 s'_2^\omega \quad \text{and} \quad \pi'_2 = s'_0 s'_3 s'_4^\omega \]
with traces
\[ \text{trace}(\pi'_1) = \emptyset \emptyset \{p\}^\omega \quad \text{and} \quad \text{trace}(\pi'_2) = \emptyset^\omega. \]
So,
\[ \text{Traces}(T) = \text{Traces}(T'). \]

Next, we prove that $T \models \text{AX EX } p$. In this case, observe that $s_2 \models p$ and as $s_2 \in \text{Suc}(s_1)$ we conclude that $s_1 \models \text{EX } p$. Furthermore, as $s_1$ is the only successor of $s_0$ we also conclude that $s_0 \models \text{AX EX } p$. And, as $s_0$ is the only initial state of $T$, we obtain the desired result, that is,
\[ T \models \text{AX EX } p. \]

Finally, we prove that $T' \not\models \text{AX EX } p$. In this case, we observe that $s'_4 \not\models p$. As $s'_4$ is the only successor state of $s'_3$ then we conclude that $s'_3 \not\models \text{EX } p$. And, as $s'_3 \in \text{Suc}(s'_0)$, we also conclude that $s'_0 \not\models \text{AX EX } p$. Hence, as $s'_0$ is an initial state of $T'$, we have that
\[ T' \not\models \text{AX EX } p. \]