Temporal logics

Master Parisien de Recherche en Informatique
Part 1 of course 2.8 (September-October 2006)

Nicolas Markey

Lab. Spécification & Vérification
CNRS & ENS Cachan

Version 1.02 – October 11, 2006

Abstract

These notes have been written as a support for the course about temporal logics taught at MPRI (Master Parisien de Recherche en Informatique) during the first semester of year 2006-2007.

They contain most of what will be studied during the course, most often with detailled proofs, and several exercises, ranked from easy ($\star$) to hard ($\star\star\star$).

Contents

Chapter 1

Introduction

Logic (from classical Greek λόγος, meaning *word* or *thought*) aims at studying statements and arguments of reasoning. According to [Sha05],

> Typically, a logic consists of a formal or informal language together with a deductive system and/or a model-theoretic semantics. The language is, or corresponds to, a part of a natural language like English or Greek. The deductive system is to capture, codify, or simply record which inferences are correct for the given language, and the semantics is to capture, codify, or record the meanings, or truth-conditions, or possible truth conditions, for at least part of the language.

Historically, the study of logic independently started in India and China, during the 6th century B.C., but has been much more developped by Greek philosophers after the 4th century B.C. However, logic was really formalized in the second half of the 19th century [Fre79], and has been much developped since then.

It has been remarked since Aristotle that classical logic cannot express subtler points such as *possibility* or *necessity*: that something is *possible* does not mean that it is true or false, but rather that it might be true or false. The *problem of the future contingents*, aiming at expressing that

> There will be a sea battle tomorrow.

is an example of a sentence that cannot be captured by classical logic. Again, while several philosophers studied the question since then, modal logics have only been formally defined in 1918 [Lew18]. Several flavours of modal logics have been developped since then: *alethic logic* is the logic to handle possibility and necessity, *deontic logic* aims at reasoning about permission and obligation, while *epistemic logic* deals with knowledge... Basically, all those logics use the same two modalities $\Diamond$ (for *possibly*) and $\Box$ (for *necessarily*), but with different sets of axioms.

In that bunch of logics, *temporal logic* is the logic to deal with the evolution of truth values with time. Temporal logic has really been studied on its own by Arthur N. Prior [Pri57, Pri67, Pri68], and by Hans W. Kamp [Kam68] in the 50's and 60's. It has then been introduced in the field of computer science by Amir Pnueli [Pnu77] as a tool to describe and reason about the behaviors of reactive systems, which has been the starting point of dazzling development of *model checking*.

These notes present temporal logics with a point of view of a computer scientist: they won't focus on axiomatizations of the logics, but rather on the problems of satisfiability and model-checking. Chapter 2 follows the history above in defining propositional logic, first- and second-order logics, modal logics, and temporal logics. Chapter 3 focuses on *linear-time* temporal logics, while Chapter 4 deals with the *branching-time* framework.

Chapter 2

From propositional logic to temporal logic

2.1 Propositional logic

If temporal logic is seen as a language for describing a movie, then propositional logic can be seen as a language for describing a picture [Gor00]. The basic blocks are *atomic propositions*, that can be combined with *Boolean operators*:

Definition 1 *The syntax of* propositional logic *over a set* AP *of atomic propositions is given by the following grammar:*

$$PL \ni \phi ::= p \mid \neg\phi \mid \phi \vee \phi \mid \phi \wedge \phi$$

where p *ranges over* AP*.*

The semantics of the Boolean operators $\neg$ (negation), $\vee$ (disjunction) and $\wedge$ (conjunction) are given as truth table displayed at table 2.1.

p	$\neg p$
false	true
true	false

p	q	$p \vee q$
false	false	false
false	true	true
true	false	true
true	true	true

p	q	$p \vee q$
false	false	false
false	true	false
true	false	false
true	true	true

Table 2.1: Truth table of the three basic Boolean operators

Example. *If* $AP = \{$ *"the sky is blue", "the grass is green"*$\}$*, then the following formula is well-formed:*

$$\neg(\text{"the sky is blue"} \vee \neg \text{"the grass is green"}).$$

It is true exactly when the sky is not blue and the grass is green.

Definition 2 *Two Boolean formulas ϕ and ψ are* equivalent *if, under each possible valuation $v\colon AP \to \{true, false\}$, the truth values of ϕ and ψ are equal. This is denoted by $\phi \equiv \psi$.*

From that semantics, it is easy to derive the following results:

Theorem 3 (De Morgan's laws)

$$\neg(\phi \wedge \psi) \equiv \neg\phi \vee \neg\psi \qquad\qquad \neg(\phi \vee \psi) \equiv \neg\phi \wedge \neg\psi$$

Several other boolean operators can be defined from the main three ones. For instance, it is usual to define the *implication*, denoted with $\Rightarrow$, as

$$p \Rightarrow q \stackrel{\text{def}}{\equiv} \neg p \vee q.$$

Similarly, *exclusive or*, written $\oplus$, is defined as

$$p \oplus q \stackrel{\text{def}}{\equiv} (p \vee q) \wedge \neg(p \wedge q).$$

We conclude this section with a famous result, due to Stephen Cook [Coo71]:

Theorem 4 *Deciding the satifiability of a formula of PL is NP-complete.*

On the other hand, evaluating the truth value of a formula given a valuation of the atomic propositions is NC^1-complete (i.e., ALOGTIME-complete) [BCGR92]. We refer to [Pap94] for more details about those complexity classes.

Exercice 2.1. ⋆ Prove Theorem 3.

Exercice 2.2. ⋆ From De Morgan's laws, we could define the syntax of PL with only negation and conjunction (or only negation and disjunction), since the third operator can be defined from the first two.

Prove that negation cannot be defined in terms of conjunction and disjunction. Is it possible to define another binary boolean operator such that negation, conjunction and disjunction can be defined using only that operator?

Exercice 2.3. ⋆⋆ Prove Theorem 4.

2.2 First- and second-order logics

First-order logic is an extension of propositional logic with quantification over a set of items X. In that setting, atomic propositions are replaced by *monadic predicates*, i.e., functions $p\colon X \to \{\text{true}, \text{false}\}$.

Definition 5 *The syntax of* first-order monadic logic *is defined as:*

$$\mathsf{FOML} \ni \phi ::= p(x) \mid \neg\phi \mid \phi \vee \psi \mid \phi \wedge \psi \mid \exists x \in X.\phi \mid \forall x \in X.\phi$$

where p ranges over the set MP of monadic predicates, x and y range over the fixed set X, with the extra requirement that any variable that appears as the argument of a predicate must be bound by an existential or universal quantifier.

The semantics is the natural one: a formula $\exists x \in X.\phi(x)$ holds if the function $\phi\colon X \to \{\text{true}, \text{false}\}$ is not always false, while $\forall x \in X.\phi(x)$ is true if ϕ always equals true. In particular:

Theorem 6

$$\neg(\exists x \in X.\phi(x)) \equiv \forall x \in X.\neg\phi(x).$$

Example. *If $X = \{sky, grass\}$ and $\mathsf{MP} = \{is\_blue, is\_green\}$, then the following is a well-formed formula:*

$$\forall x \in X.(is\_green(x) \vee is\_blue(x)).$$

In the (interesting) case when X can be ordered, first-order logic can be extended to also be able to use that order:

Definition 7 *Let $(X, \leq)$ be an ordered set. The syntax of* first-order monadic logic of order *is defined as:*

$$\mathsf{FOMLO} \ni \phi ::= p(x) \mid x \leq y \mid \neg\phi \mid \phi \vee \psi \mid \phi \wedge \psi \mid \exists x \in X.\phi \mid \forall x \in X.\phi$$

The semantics is immediately derived from that of FOML. Of course, since the order is reflexive, it is possible to test for the equality of two items of X.

Example. *If X is the set of nonnegative integers equipped with its classical order, using predicates is_odd and is_even, the following formulas hold:*

- *$\mathbb{Z}^+$ has a minimal element:*

$$\exists x \in \mathbb{Z}^+.\ \forall y \in \mathbb{Z}^+.\ x \leq y.$$

- *among two successive integers, exactly one is odd:*

$$\forall x \in \mathbb{Z}^+.\ \forall y \in \mathbb{Z}^+.\ (x \leq y) \Rightarrow \big[(\forall z \in \mathbb{Z}^+.\ (z \leq x \vee y \leq z)) \Rightarrow (is\_odd(x) \oplus is\_odd(y))\big].$$

In this case, we have a theorem similar to Theorem 4, but with a much higher complexity:

Theorem 8 *The satisfiability of a formula of FOMLO is decidable (and has non-elementary complexity).*

Note that such a result depends on the underlying order. The result above holds for classical linear orders such as $\langle \mathbb{Z}^+, \leq \rangle$ [Sto74] or $\langle \mathbb{R}^+, \leq \rangle$ [BG85].

Exercice 2.4. ⋆⋆ Over the set of nonnegative integers, using the monadic predicate is_prime, write a formula in FOMLO expressing (the conjecture) that there exists infinitely many twin primes [Wik06].

Second-order logic is an extension of first-order logic in which one can also quantify over subsets of the set X:

Definition 9 *Let $(X, \leq)$ be an ordered set. The syntax of* monadic second-order logic of order *is defined as:*

$$\mathsf{SOMLO} \ni \phi ::= p(x) \mid x \leq y \mid x \in Y \mid \neg\phi \mid \phi \vee \psi \mid \phi \wedge \psi \mid \exists x \in X.\phi \mid \forall x \in X.\phi \mid \exists Z \subseteq X.\phi \mid \forall Z \subseteq X.\phi.$$

where p ranges over the set MP of monadic predicates, x and y range over the fixed set X, Y ranges over the set of all subsets of X, with the extra requirement that any variable that appears as the argument of a predicate must be bound by an existential or universal quantifier.

Notice that we directly defined SOMLO, but obviously SOML can also be defined, as in the case of first-order logic. Again, the semantics is rather obvious, and we omit it. Also, clearly, Theorem 6 also holds for second-order quantification. Last, we have the following:

Theorem 10 *The satisfiability problem for SOMLO is decidable (and has non-elementary complexity).*

Let us insist again on the fact that this general result only holds in some (interesting) special cases: discrete and continuous linear orders [Büc60, Rab], and discrete tree orders [Rab].

Remark. *Note that it is of course possible to define higher-order logics, with quantification over sets of sets of X, and so on. In that hierarchy, propositional calculus is sometimes refered to as* zeroth-order logic.

Example. *Assume that X is a totally ordered set. FOMLO can express that X is has no minimal element:*

$$\exists x \in X.\ \forall y \in X.\ x \leq y.$$

But FOMLO cannot express that X is well-ordered, i.e., that any non-empty subset has a minimum element: this requires second-order quantification:

$$\forall Y \subseteq X.(\exists x \in Y.\ \forall y \in Y.\ x \leq y)$$

2.3 Modal logics

Roughly speaking, modal logics are extensions of propositional logic with new operators that express possibility or necessity. Basically:

Definition 11 *The syntax of modal logic is:*

$$\mathsf{ML} \ni \phi ::= p \mid \neg\phi \mid \phi \vee \phi \mid \phi \wedge \phi \mid \Diamond \phi \mid \Box \phi$$

where p ranges over the set AP of atomic propositions.

The semantics of modal logics is generally defined over Kripke structures:

Definition 12 *Let AP be a (finite) set of atomic propositions. A* Kripke structure *is a 3-tuple $\langle W, R, l\rangle$ s.t.:*

- *W is a non-empty set whose items are called* worlds*,*
- *R is a binary relation over W (often refered to as the* accessibility relation*). It is required that for any $w \in W$, there exists $w' \in W$ s.t. $(w, w') \in R$ (i.e., each world must have at least one successor).*
- *$l \colon W \to 2^{\mathsf{AP}}$ maps each world to the set of atomic propositions that hold in that world.*

A Kripke structure is in fact nothing more than a lebeled transition system, and the name is only kept in honour of Saul Kripke, who used them in order to define the semantics of modal logics.

Definition 13 *Let K be a Kripke structure, $w \in W$ be a world, and $\phi \in \mathsf{ML}$. That formula ϕ holds in the world w (denoted $w \models \phi$) is defined through the following recursive rules:*

- *$w \models p$ if, and only if, $p \in l(p)$, for any atomic proposition p,*
- *$w \models \neg\psi$ if, and only if, $w \not\models \psi$,*
- *$w \models \psi_1 \vee \psi_2$ if, and only if, $w \models \psi_1$ or $w \models \psi_2$,*
- *$w \models \psi_1 \wedge \psi_2$ if, and only if, $w \models \psi_1$ and $w \models \psi_2$,*
- *$w \models \Diamond \psi$ if, and only if, $\exists w' \in W.((w, w') \in R \wedge w' \models \psi)$,*
- *$w \models \Box \psi$ if, and only if, $\forall w' \in W.((w, w') \in R \Rightarrow w' \models \psi)$.*

This captures the intuitive meaning that something possible is true in one of the accessible worlds, while something necessary holds in any. Depending on the nature (alethic, epistemic, ...) of the modal logic under study, Kripke structures might be required to be reflexive, transitive, ...

The following result directly follows from this definition:

Theorem 14

$$\neg\Diamond \phi \equiv \Box \neg\phi.$$

2.4 Temporal logics

Temporal logic is, originally, a derivative of modal logic that focuses on the succession of events along time. With no constraint on the Kripke structure, modalities $\Diamond$ and $\Box$ have their standard meaning, concerning the set of next states. If we require that the Kripke structure be transitive (modeling the fact that "being in the future" is transitive), then

- $\Diamond\,\phi$ means that ϕ will be true some time in the future, along some evolution.
- $\Box\,\phi$ means that ϕ will always hold, at any time in the future, and along any evolution.

Still, this definition does not allow to express properties such as "ϕ is unavoidable", i.e., that ϕ will eventually occur along any evolution. In fact, temporal logic now comprises two main frameworks: the *linear-time* framework, and the *branching-time* framework. The former aims at expressing properties along each single evolution in the Kripke structure, while the latter adds quantification over the possible evolutions.

In the linear-time setting, modalities $\Diamond$ and $\Box$ are often written **F** and **G**, resp., and are read "eventually in the future" and "always in the future". It is of course possible to define extra modalities. In a broad sense, a modality is a n-ary predicate whose definition can be expressed in FOMLO. For example, extra modalities have been defined for dealing with what happened in the past: it is now standard to write them as $\mathbf{F}^{-1}$ and $\mathbf{G}^{-1}$, and to read them "sometime in the past" and "always in the past", resp. Two binary modalities are also classically used: **U**, read "until", and **S**, read "since". Those modalities have been introduced by Hans Kamp after he proved that they could not be expressed with only unary modalities [Kam68]. The formal semantical definitions, as well as many related results, will be given and explained in Chapter 3, which is entirely devoted to the linear-time framework.

The branching-time framework can be seen as an extension of linear-time temporal logics with *path quantifiers*, namely **E** ("there exists an evolution such that...") and **A** ("along any evolution, ..."). The intuition (which will be made formal in Chapter 4, which entirely focuses on branching-time temporal logics) indicates that $\mathbf{EF}\,\phi$ corresponds to $\Diamond\,\phi$, that $\mathbf{AG}\,\phi$ corresponds to $\Box\,\phi$, while $\mathbf{AF}\,\phi$ expresses that ϕ will eventually occur along any evolution.

2.5 Temporal logics in computer science

In 1977, Amir Pnueli introduced the use of (linear-time) temporal logics in computer science [Pnu77], in order to (automatically) verify that a reactive system (modeled as a Kripke structure, i.e., a labeled transition system) satisfies some properties (expressed, for instance, as temporal logic formulas). Branching-time logic have also quickly been introduced in this framework [CE81, QS82].

This technique, now called *model-checking*, has been much developped since then: efficient algorithms and powerful data-types have been introduced and

studied, and model-checking is now well-established and recognized as a powerful tool for the verification of automatic systems [CGP99, BBF+01]. The principle of model-checking is depicted on Figure 2.1.

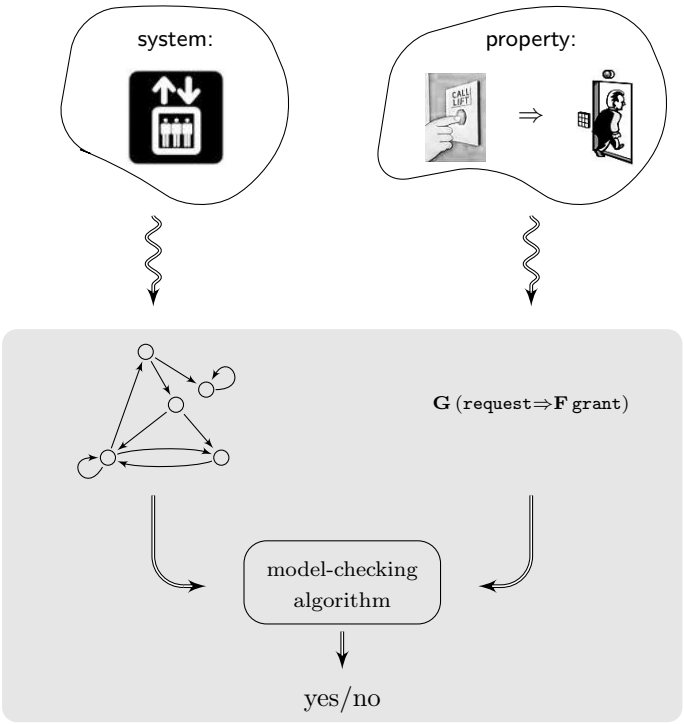


Figure 2.1: Principles of model-checking

This course is mainly devoted to the study of temporal logics for model-checking. The advantage of temporal logics in that framework is that they are expressive enough (and not too difficult to manipulate) for most applications, and have reasonably efficient verification algorithms. Satisfiability of a temporal logic formula is a connected problem, and it will also be studied in this course. Formally, those two problems are defined as follows:

Definition 15 satisfiability:

Input *a formula ϕ;*

Output *true if there exists a structure on which ϕ evaluates to true.*

model checking:

Input *a formula ϕ and a structure s;*

Output *true if ϕ holds in the structure s.*

Chapter 3

Linear-time temporal logics

In the sequel, AP represents a non-empty, finite set of atomic propositions.

3.1 Linear structures

Linear-time temporal logics are those temporal logics that express properties over each single evolution of the system. We first define formally how such an evolution is represented.

Definition 16 *A* (labeled) linear structure *is a tuple* $\mathcal{S} = \langle T, \leq, \ell \rangle$ *where:*

- $\langle T, \leq \rangle$ *is a totally-ordered infinite set with a minimal element;*
- $\ell \colon T \to 2^{\mathsf{AP}}$ *labels each point of* T *with a subset of* AP*.*

In the sequel, we will always assume that T is (isomorphic to) either the set of nonnegative integers, or the set of nonnegative reals. The set T is often refered to as the *time-line*: when $T = \mathbb{Z}^+$, we say that the time-line is discrete, while $T = \mathbb{R}^+$ corresponds to a continuous time-line.

Example. *Let* $KS = \langle W, R, l \rangle$ *be a Kripke structure, and* $w \in W$*. Informally, an evolution of* $\mathcal{K}$ *from* w *is the infinite sequence of worlds* $(w_i)_{i \in \mathbb{Z}^+}$ *that are successively traversed. In our formalism, this is represented as a linear-structure* $\langle \mathbb{Z}^+, \leq, \ell \rangle$ *s.t.* $\ell(i) = l(w_i)$*.*

Remark. *It is often handy to represent and manipulate discrete linear structures as words over the alphabet* 2^{AP}*. A linear structure* $\langle \mathbb{Z}^+, \leq, \ell \rangle$ *would then be represented as the word* $\sigma = \sigma_0 \, \sigma_1 \, \ldots$*, with* $\sigma_i = \ell(i)$*. In the sequel, we might go from one representation to the other without further notice, as both are very similar.*

Linear-time temporal logic formulas will generally be evaluated at one precise point along such a linear structure:

Definition 17 *A* pointed linear structure *is a couple* $\langle \mathcal{S}, s\rangle$ *where* $\mathcal{S} = \langle T, \leq, \ell\rangle$ *is a linear structure, and* $s \in T$*. We write* $\mathcal{M}_{lin(T)}$ *for the set of pointed linear structures built on the time-line* T*.*

A special subclass of pointed linear structures is that of initially pointed linear structures*, i.e., couples* $\langle \mathcal{S}, 0\rangle$ *where* $\mathcal{S} = \langle T, \leq, \ell\rangle$ *and* 0 *is the minimal element of* T *(refered to as the* origin *of the time-line). We write* $\mathcal{M}^0_{lin(T)}$ *for the set of initially pointed linear structures built on* T*.*

3.2 Linear-time temporal logics

We now define the formal semantics of the modalities that we already mentionned in the introduction.

Definition 18 *Let* $\langle \mathcal{S}, s\rangle \in \mathcal{M}_{lin(T)}$ *be a pointed linear structure. We define the following one-place modalities, together with their (inductive) semantics:*

- $\mathbf{F}$ *(eventually):*

$$\langle \mathcal{S}, s\rangle \models \mathbf{F}\,\phi \stackrel{def}{\Leftrightarrow} \exists t.\ (s \leq t \wedge \langle \mathcal{S}, t\rangle \models \phi).$$

- $\mathbf{G}$ *(always in the future):*

$$\langle \mathcal{S}, s\rangle \models \mathbf{G}\,\phi \stackrel{def}{\Leftrightarrow} \forall t.\ (s \leq t \Rightarrow \langle \mathcal{S}, t\rangle \models \phi).$$

- $\mathbf{X}$ *(next):*

$$\langle \mathcal{S}, s\rangle \models \mathbf{X}\,\phi \stackrel{def}{\Leftrightarrow} \exists t.\ (s < t \wedge \forall u.\ (u \leq s \vee t \leq u) \wedge \langle \mathcal{S}, t\rangle \models \phi).$$

- $\mathbf{F}^{-1}$ *(sometime in the past):*

$$\langle \mathcal{S}, s\rangle \models \mathbf{F}^{-1}\,\phi \stackrel{def}{\Leftrightarrow} \exists t.\ (t \leq s \wedge \langle \mathcal{S}, t\rangle \models \phi).$$

- $\mathbf{G}^{-1}$ *(always in the past):*

$$\langle \mathcal{S}, s\rangle \models \mathbf{G}^{-1}\,\phi \stackrel{def}{\Leftrightarrow} \forall t.\ (t \leq s \Rightarrow \langle \mathcal{S}, t\rangle \models \phi).$$

- $\mathbf{X}^{-1}$ *(previous):*

$$\langle \mathcal{S}, s\rangle \models \mathbf{X}^{-1}\,\phi \stackrel{def}{\Leftrightarrow} \exists t.\ (s < t \wedge \forall u.\ (s \leq u \vee u \leq t) \wedge \langle \mathcal{S}, t\rangle \models \phi).$$

We also define two two-place modalities:

- $\mathbf{U}$ *(until):*

$$\langle \mathcal{S}, t\rangle \models \phi\,\mathbf{U}\,\psi \stackrel{def}{\Leftrightarrow} \exists t.\ (s \leq t \wedge (\langle \mathcal{S}, t\rangle \models \psi) \wedge \forall u.\ ((s \leq u \wedge u < t) \Rightarrow \langle \mathcal{S}, u\rangle \models \phi)).$$

- $\mathbf{S}$ *(since):*

$$\langle \mathcal{S}, t\rangle \models \phi \,\mathbf{S}\, \psi \stackrel{def}{\Leftrightarrow} \exists t.\ (t \leq s \wedge (\langle \mathcal{S}, t\rangle \models \psi) \wedge \forall u.\ ((u \leq s \wedge t < u) \Rightarrow \langle \mathcal{S}, u\rangle \models \phi)).$$

Remark. *Several remarks are in order here:*

- *modalities* $\mathbf{X}$ *and* $\mathbf{X}^{-1}$ *are only interesting over discrete time-lines;*
- *one-place modalities, except* $\mathbf{X}$ *and* $\mathbf{X}^{-1}$*, can be expressed using only two-place ones;*
- *it is possible to define strict versions of the above modalities, that would not take the current position into account. Those modalities are denoted with a tilde. For instance:*
  - $\tilde{\mathbf{U}}$ *(strict until):*

$$\langle \mathcal{S}, t\rangle \models \phi \,\tilde{\mathbf{U}}\, \psi \stackrel{def}{\Leftrightarrow} \exists t.\ (s < t \wedge (\langle \mathcal{S}, t\rangle \models \psi) \wedge \forall u.\ ((s < u \wedge u < t) \Rightarrow \langle \mathcal{S}, u\rangle \models \phi)).$$

  - $\tilde{\mathbf{S}}$ *(strict since):*

$$\langle \mathcal{S}, t\rangle \models \phi \,\tilde{\mathbf{S}}\, \psi \stackrel{def}{\Leftrightarrow} \exists t.\ (t < s \wedge (\langle \mathcal{S}, t\rangle \models \psi) \wedge \forall u.\ ((u < s \wedge t < u) \Rightarrow \langle \mathcal{S}, u\rangle \models \phi)).$$

  With those definitions, all modalities (both strict and non-strict) can be defined from the above two.

Using those modalities, we can define different linear-time temporal logics:

Definition 19 *Given a set of n modalities* $\{M_1, ..., M_n\}$*, the logic* $\mathcal{L}(M_1, ..., M_n)$ *is the set of formulas built on the following grammar:*

$$\mathcal{L}(M_1, ..., M_n) \ni \phi ::= p \mid \neg\phi \mid \phi \vee \phi \mid \phi \wedge \phi \mid M_1(\phi, ..., \phi) \mid \cdots \mid M_n(\phi, ..., \phi).$$

Several logics built on this schema have a special name:

$$\mathsf{LTL} = \mathcal{L}(\mathbf{U}, \mathbf{X}) \qquad \mathsf{LTL}_s = \mathcal{L}(\tilde{\mathbf{U}})$$
$$\mathsf{LTL{+}Past} = \mathcal{L}(\mathbf{U}, \mathbf{X}, \mathbf{S}, \mathbf{X}^{-1}) \qquad \mathsf{LTL{+}Past}_s = \mathcal{L}(\tilde{\mathbf{U}}, \tilde{\mathbf{S}})$$

It might be surprising to define two flavours of LTL and LTL+Past. However, both definitions coexist in the literature: LTL_s and $\mathsf{LTL{+}Past}_s$ are the standard definitions for the general case, while LTL and LTL+Past are often prefered in the discrete-time case.

Note that, in the remarks above, we did not formally defined what we mean with "can be expressed". The formal definition relies on the notion of equivalence of two formulas:

Definition 20 *Two linear-time temporal logic formulas ϕ and ψ are* equivalent *over a set of pointed structures $\mathcal{M}$ when*

$$\forall m \in \mathcal{M}.\ (m \models \phi \ \Leftrightarrow\ m \models \psi).$$

Definition 21 *A formula ϕ can be expressed in a logic $\mathcal{L}$ over a set of structures $\mathcal{M}$ whenever there exists a formula $\psi \in \mathcal{L}$ that is equivalent to ϕ over $\mathcal{M}$.*

Exercice 3.5. ⋆ Prove all claims of the remarks following Definition 18. Prove that LTL and LTL_s are equally expressive.

Exercice 3.6. ⋆⋆⋆ This exercice aims at showing that modality $\mathbf{U}$ cannot be expressed in the logic $\mathcal{L}(\mathbf{F}, \mathbf{F}^{-1})$. We restrict here to discret-time, but a similar proof could be carried out for other time-lines.

For each positive integer i, consider the word σ^i defined with the following rules:

- for any $j \neq i$, $\sigma^i_{3j} = a$, $\sigma^i_{3j+1} = b$, $\sigma^i_{3j+2} = c$;
- $\sigma^i_{3i} = a$, $\sigma^i_{3i+1} = c$, $\sigma^i_{3i+2} = b$;

That is, σ^i almost equals $(abc)^\omega$, except that b and c are swapped in the i-th copy. Prove the following three results:

- the set of positions where formula $\mathbf{F}(c \wedge c\,\mathbf{U}\,b)$ hold along σ^i is $[0, 3i+1]$.
- if ϕ is a formula containing at most i modalities among $\mathbf{F}$, $\mathbf{G}$, $\mathbf{F}^{-1}$ and $\mathbf{G}^{-1}$ (and no other modalities), then if ϕ holds at some position $k \geq 3i$, then ϕ also holds at position $k+3$.
- formula $\mathbf{F}(c \wedge c\,\mathbf{U}\,b)$ cannot be expressed in $\mathcal{L}(\mathbf{F}, \mathbf{F}^{-1})$.

Deduce that also modality $\mathbf{X}$ cannot be defined in $\mathcal{L}(\mathbf{F}, \mathbf{F}^{-1})$.

Exercice 3.7. ⋆ A fairness property is a property stating that something occurs repeatedly, infinitely often. Prove that fairness properties can be expressed in LTL. Symmetrically, what is the meaning of formula $\mathbf{F}^{-1}\,\mathbf{G}^{-1}\,\phi$?

Before we study the complexity of satisfiability and model-checking of those logics, we have to define the notion of size of a formula. In fact, two definition coexist:

Definition 22 *The* size *of a formula $\phi \in \mathcal{L}(M_1, ..., M_n)$ is defined inductively as follows:*

$$\begin{aligned} |p| &= 1 \\ |\neg\phi_1| &= 1 + |\phi_1| \\ |\phi_1 \vee \phi_2| &= 1 + |\phi_1| + |\phi_2| \\ |\phi_1 \wedge \phi_2| &= 1 + |\phi_1| + |\phi_2| \\ |M(\phi_1, ..., \phi_n| &= 1 + |\phi_1| + \cdots + |\phi_n|\,. \end{aligned}$$

In other words, the size of a formula is the number of symbols (excluding parenthesis) needed for writing a formula. The size of a formula could also be defined as the number of nodes of the tree-representation of the formula (where, for instance, formula $M(\phi_1, ..., \phi_n)$ would correspond to one node with n successors corresponding to subformulas ϕ_1 to ϕ_n).

If, in such a tree-representation, we *merge* the states correpsonding to identical subformulas, we get a directed acyclic graph (DAG for short). The number of nodes of this DAG is another way of defining the size of the corresponding formula, called the DAG-size. There is another way of defining this measure:

Definition 23 *The* DAG-size *of a formula ϕ, written $|\phi|_{DAG}$ is the number of subformulas of ϕ, where the set of subformulas is defined inductively as follows:*

$$\begin{aligned} \mathsf{SF}p &= \{p\} \\ \mathsf{SF}\neg\phi_1 &= \{\neg\phi_1\} \cup \mathsf{SF}\phi_1 \\ \mathsf{SF}\phi_1 \vee \phi_2 &= \{\phi_1 \vee \phi_2\} \cup \mathsf{SF}\phi_1 \cup \mathsf{SF}\phi_2 \\ \mathsf{SF}\phi_1 \wedge \phi_2 &= \{\phi_1 \wedge \phi_2\} \cup \mathsf{SF}\phi_1 \cup \mathsf{SF}\phi_2 \\ \mathsf{SF}M_i(\phi_1, ..., \phi_n) &= \{M(\phi_1, ..., \phi_n)\} \cup \mathsf{SF}\phi_1 \cup \cdots \cup \mathsf{SF}\phi_n. \end{aligned}$$

Both definitions satisfy the following relation:

Theorem 24 *For any $\phi \in \mathcal{L}(M_1, ..., M_n)$, we have*

$$|\phi| \leq |\phi|_{DAG} \leq a^{|\phi|}$$

where a is the maximal arity of the modalities in $\{M_1, ..., M_n\}$.

Exercice 3.8. ⋆ Prove Theorem 24.

3.3 Expressive completeness of LTL+Past and LTL

We keep on focusing on the expressiveness of LTL and LTL+Past in this section.

Theorem 25 *Any formula in LTL or LTL+Past can be expressed in FOMLO using at most three variables.*

Exercice 3.9. ⋆ Prove Theorem 25.

In many cases, the converse fails to hold, and FOMLO, even with only three variables, contains formulas that can't be expressed in LTL or LTL+Past. However, there are some interesting special cases:

Theorem 26 ([Kam68]) *Over $\mathcal{M}_{lin(\mathbb{Z}^+)}$ and $\mathcal{M}_{lin(\mathbb{R}^+)}$, any formula of FOMLO can be expressed in LTL+Past.*

Theorem 27 ([GPSS80]) *Over $\mathcal{M}^0_{lin(\mathbb{Z}^+)}$, any formula of FOMLO can be expressed in LTL.*

Those theorems state that, in certain circumstances, LTL or LTL+Past contain all the expressive power of the formalism it is built on (namely first-order logic). This property is called *expressive completeness*.

The proofs of those results are very technical, and way beyond the scope of this course. In fact, the important point is the following corollary:

Corollary 28 *Over $\mathcal{M}^0_{lin(\mathbb{Z}^+)}$, any formula in LTL+Past can be translated in LTL.*

Exercice 3.10. ⋆⋆⋆ Prove that Theorem 27 does not hold over $\mathcal{M}_{\mathrm{lin}(\mathbb{Z}^+)}$ (i.e., over general pointed linear structure) and over $\mathcal{M}^0_{\mathrm{lin}(\mathbb{R}^+)}$. For this second part, consider the following two linear structures:

- $\mathcal{S}_1 = \langle \mathbb{R}^+, \leq, \ell_1 \rangle$ with $\ell_1(t) = \begin{cases} a & \text{if } t \in \mathbb{Z}^+ \\ \varnothing & \text{otherwise} \end{cases}$
- $\mathcal{S}_2 = \langle \mathbb{R}^+, \leq, \ell_2 \rangle$ with $\ell_2(t) = \begin{cases} a & \text{if } t \in \mathbb{Z}^+ \text{ or } 1/t \in \mathbb{Z}^+ \\ \varnothing & \text{otherwise} \end{cases}$

and prove that they satisfy exactly the same set of LTL formulas, while there exists a formula in LTL+Past that holds only of $\langle \mathcal{S}_1, 0 \rangle$.

Several other expressiveness results exist. Related to the above ones is the following:

Theorem 29 ([EVW02]) *Any formula of FOMLO involving at most two variables can be expressed in $\mathcal{L}(\tilde{\mathbf{F}}, \tilde{\mathbf{F}}^{-1})$, and conversely.*

Again, the proof of this result is rather technical, and the interested reader is invited to consult the original paper.

We conclude this section with another important expressiveness result:

Theorem 30 ([Wol83]) *Over discrete time, that "ϕ holds at any even position along a linear structure" cannot be expressed in LTL+Past.*

Exercice 3.11. ⋆⋆⋆ Explain why the tentative formulas below do not express the property of Theorem 30:

$$\phi \wedge \mathbf{G}\,(\phi \Leftrightarrow \mathbf{X}\,\neg\phi) \qquad\qquad \phi \wedge \mathbf{G}\,(\phi \Rightarrow \mathbf{X}\,\mathbf{X}\,\phi)$$

Considering the following family of words, prove Theorem 30:

$$\sigma^i = a^i \cdot b \cdot a^\omega.$$

3.4 Satisfiability of LTL is PSPACE-hard

In this and the next sections of this chapter, we only consider the case of discrete time. This section is again (slightly) related to expressiveness: we prove that LTL and LTL+Past can encode, with a polynomial-size formula, the behavior of a linear-space Turing machine.

Theorem 31 ([SC85]) *Satisfiability of an LTL formula is PSPACE-hard.*

Note that satisfiability here means the existence of a Kripke structure all of whose executions satisfy the LTL formula.

Proof. The proof is based on the following lemma:

Lemma 32 *Let T be a linear-space deterministic Turing machine. Then one can build, using logarithmic space, a polynomial-size LTL formula ϕ such that:*

T halts on the empty word if, and only if, ϕ is satisfiable.

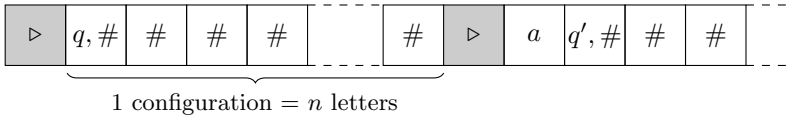
Proving this lemma obviously completes the proof of Theorem 31, since the halting problem for linear-space deterministic Turing machines is known to be PSPACE-complete.

Proof (of Lemma 32). We fix the notations: $T = \langle Q, q_i, q_f, \Sigma, \#, \delta, n\rangle$, where Q is the set of states of the Turing machine, q_i and q_f are the initial and final states, Σ is the alphabet, $\#$ is a special symbol (required not to belong to Σ) representing an empty cell, $\delta : Q \times (\Sigma \cup \{\#\}) \to Q \times \Sigma \times \{\leftarrow, \rightarrow\}$ is the transition function (remember that our machine is deterministic), and n, given in unary, is the size of the tape.

The intuition of the reduction is as follows: we build a formula that will be true only on words that represent the successive configurations of the Turing machine. The set of atomic propositions is defined as follows:

$$\mathsf{AP} = Q \cup \Sigma \cup \{\#, \triangleright\}$$

where the new symbol $\triangleright$ will be used as a configuration separator. In order to be accepted, a word will thus have to begin with the following pattern:



This example corresponds to a transition $\delta(q, \#) = (q', a, \rightarrow)$.

We have several properties to enforce:

1. the path should begin with $\triangleright$, and this symbol must repeat precisely every $n + 1$ letters: in order to express this, we first define

$$\phi_{\text{sep}} = \triangleright \wedge \bigwedge_{p \in \mathsf{AP}, p \neq \triangleright} \neg p.$$

Then

$$\phi_1 = \phi_{\text{sep}} \wedge \mathbf{G}\,(\phi_{\text{sep}} \Rightarrow \underbrace{\mathbf{X}\,(\neg\triangleright \wedge \mathbf{X}\,(\neg\triangleright \wedge \mathbf{X}\,\ldots\mathbf{X}\,(\neg\triangleright}_{n \text{ times}} \wedge \mathbf{X}\,\phi_{\text{sep}})\ldots)))$$

2. exactly one tape head (whose position is represented by writing the current state in the correspondig cell) must appear in each configuration: we first define

$$\phi_{\text{head}} = \bigvee_{q \in Q} \Big(q \wedge \bigwedge_{q' \in Q, q' \neq q} \neg q' \Big).$$

Then

$$\phi_2 = \mathbf{G}\,(\phi_{\text{sep}} \Rightarrow ((\neg\phi_{\text{head}})\,\mathbf{U}\,(\phi_{\text{head}} \wedge \mathbf{X}\,((\neg\phi_{\text{head}})\,\mathbf{U}\,\phi_{\text{sep}}))))$$

3. there is one letter (possibly #) in each cell: we first define

$$\phi_{\text{letter}} = \bigvee_{p \in \Sigma \cup \{\#\}} \Big(p \wedge \bigwedge_{p' \in \Sigma \cup \{\#\}, p' \neq p} \neg p' \Big).$$

Then:

$$\phi_3 = \mathbf{G}\,(\neg\triangleright \Rightarrow \phi_{\text{letter}}).$$

4. we have to ensure that the content of the tape is preserved from one configuration to the next one, except for the cell the tape is on:

$$\phi_4 = \mathbf{G}\,\Big(\neg\phi_{head} \Rightarrow \bigwedge_{p \in \Sigma \cup \{\#\}} (p \Rightarrow \mathbf{X}^{n+1}\,p)\Big).$$

5. if a transition can be applied from a configuration, then the next configuration must be modified accordingly:

$$\phi_5 = \bigwedge_{\delta(q,a)=(q',b,\rightarrow)} \mathbf{G}\,((q \wedge a) \Rightarrow \mathbf{X}^{n+1}\,(b \wedge \mathbf{X}\,q')).$$

Several technical details are omitted here: of course, the same kind of formula has to be written for transitions in which the tape head goes to the left, but we should also take care of the left- and right-end of the tape, as well as the case where no transition can be applied. Those details are left to the meticulous reader.

6. last, we require that the tape is initially empty, and that the execution must reach the accepting location:

$$\phi_6 = \mathbf{X}\,(\#\,\mathbf{U}\,\phi_{\text{sep}}) \wedge \mathbf{F}\,q_f.$$

It is now a matter of bravery to prove that this formula satisfies the conditions of the lemma. □□

As direct corollaries, we obtain the following results:

Corollary 33 ([SC85]) *LTL model-checking in PSPACE-hard.*

Corollary 34 *Model-checking and satisfiability are PSPACE-hard for LTL+Past.*

Exercice 3.12. ⋆ Prove Corollary 33.

Exercice 3.13. ⋆⋆ Quantified boolean formula (QBF) is another instance of a PSPACE-complete problem. An instance of QBF is a formula

$$\exists p_1.\ \forall p_2.\ \ldots\ \forall p_{2n}.\ \phi(p_1, p_2, \ldots, p_{2n})$$

where $\phi(p_1, p_2, \ldots, p_n)$ is a formula of PL built on atomic propositions p_1 to p_{2n} (which can be assumed to be in conjunctive normal form). Give a direct reduction (in logarithmic space) of QBF to LTL model-checking.

3.5 LTL+Past and Büchi automata

The main goal of this section is to prove that LTL+Past satisfiability (and model-checking) are PSPACE-complete. Several proofs of this result exist, and we present below the automata-theoretic approach to LTL+Past model-checking.

3.5.1 Büchi automata

This approach involves finite-state automata on infinite words. We first define this formalism:

Definition 35 *A* finite-state automaton *is a tuple* $\mathcal{A} = \langle Q, Q_0, \Sigma, \delta \rangle$ *where:*

- Q *is a finite set of* states *(or* locations*);*
- $Q_0 \subseteq Q$ *is the set of initial locations;*
- Σ *is a finite alphabet;*
- $\delta \subseteq Q \times \Sigma \times Q$ *it the transition relation.*

Here is how an automaton "reads" words on the alphabet Σ:

Definition 36 *Let* $\mathcal{A} = \langle Q, Q_0, \Sigma, \delta \rangle$ *be an automaton, and* $\sigma \in \Sigma^\omega$ *be an infinite word on* Σ*. An* execution *of* $\mathcal{A}$ *over* σ *is an infinite sequence* $\rho = (q_i)_{i \in \mathbb{Z}^+}$ *of locations of the automaton s.t.:*

- $q_0 \in Q_0$*;*
- *for any* $i \in \mathbb{Z}^+$, $(q_i, \sigma_i, q_{i+1}) \in \delta$*.*

It remains to define when such an automaton accepts its input word. This decision is based on the set of repeated locations along the execution:

Definition 37 *Let $\rho = (q_i)_{i\in\mathbb{Z}^+}$ be an execution of an automaton $\mathcal{A} = \langle Q, Q_0, \Sigma, \delta\rangle$ over a word σ. The set of* repeated states *of the execution is defined as*

$$\mathit{Inf}(\rho) = \{q \in Q \mid \forall i \in \mathbb{Z}^+.\ \exists j \in \mathbb{Z}^+.\ (j \geq i \wedge q_j = q)\}$$

(i.e., $\mathit{Inf}(\rho)$ is the set of locations that are traversed infinitely often along ρ).

Note that, since we required that Q be finite while executions are infinite, the set of repeated states is always non-empty.

Definition 38 *A* generalized Büchi automaton *is a tuple $\mathcal{A} = \langle Q, Q_0, \Sigma, \delta, F\rangle$ where:*

- *$\langle Q, Q_0, \Sigma, \delta\rangle$ is a finite-state automaton;*
- *$F = \{F_1, ..., F_p\} \subseteq \mathcal{P}(Q)$ is a set of sets of states.*

We are now in a position to define when a word is accepted by such an automaton:

Definition 39 *Let $\mathcal{A} = \langle Q, Q_0, \Sigma, \delta, F\rangle$ be a generalized Büchi automaton, and $\sigma \in \Sigma^\omega$ be an infinite word on Σ. Then w is* accepted *by $\mathcal{A}$ if, and only if, there exists an execution $\rho = (q_i)_{i\in\mathbb{Z}^+}$ s.t.*

$$\textit{for each set } F_i \in \mathbf{F}\,, \quad \mathit{Inf}(\rho) \cap F \neq \varnothing.$$

We write $\mathcal{L}(\mathcal{A}) \subseteq \Sigma^\omega$ for the language of $\mathcal{A}$, i.e., the set of words that are accepted by $\mathcal{A}$.

Example. *Figure 3.1 is a graphical representation of a Büchi automaton over $\Sigma = \{a, b\}$ that accepts precisely the words containing an infinite number of a's. In this simple example, there is only one set of accepting sets (i.e., $F = \{\{q_a\}\}$), and that state is marked with a double line.*

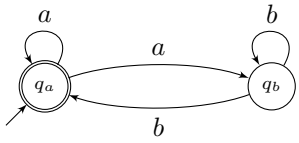


Figure 3.1: An example of a Büchi automaton

Exercice 3.14. ⋆⋆ Definition 38 defined *generalized* Büchi automata. Classical Büchi automata are a special case of generalized üchi automata where the acceptance condition is based on only one set of states: $F = \{F_1\}$. Prove that generalized Büchi automata are not more expressive than classical ones, in the sense that any generalized Büchi automaton can be transformed (using at most

logarithmic space) into a polynomial-size classical Büchi automaton accepting the same language.

Exercice 3.15. ⋆⋆ Prove that the set of languages definable by Büchi automata is closed under union and intersection. Prove that this result still holds for *deterministic* Büchi automata.

Theorem 40 *The set of languages accepted by deterministic Büchi automata is not closed under complementation.*

Proof. The proof is based on a characterization of languages recognized by deterministic Büchi automata in terms of languages recognized by deterministic automata on finite words:

Definition 41 *A finite-state automaton over finite words is a tuple* $\mathcal{A} = \langle Q, Q_0, \Sigma, \delta, F\rangle$ *where*

- $\langle Q, Q_0, \Sigma, \delta\rangle$ *is a finite-state automaton;*
- $F \subseteq Q$ *is the set of final states.*

Similarly to the case of infinite words, an execution is an automaton over a finite word $\sigma = \sigma_0 \cdots \sigma_k$ is a finite sequence $\rho = q_0 \cdots q_{k+1}$ s.t. $q_0 \in Q_0$ and $(q_i, \sigma_i, q_{i+1}) \in \delta$ for each $i \leq k$. Then σ is accepted whenever $q_{k+1} \in F$.

We prove that a language is accepted by a deterministic Büchi automaton if, and only if, it is the *limit* of a language accepted by a deterministic automaton over finite words:

Definition 42 *Let* L *be an infinite set of finite words. The limit of* L *is defined as*

$$\lim(L) = \{\sigma \in \Sigma^\omega \mid \forall i \in \mathbb{Z}^+.\ \exists j \geq i.\ \sigma_0 \cdots \sigma_j \in L\}.$$

That is, $\lim(L)$ *is the set of infinite words having infinitely many prefixes in* L*.*

Exercice 3.16. ⋆⋆ Compute the following languagues

$$\lim(\{ba \cdot b^n \mid n \in \mathbb{Z}^+\}) \qquad \lim(\{a^m \cdot b \cdot a^n \mid m, n \in \mathbb{Z}^+\})$$
$$\lim(\{(ab)^m \cdot b^n \mid m, n \in \mathbb{Z}^+\}) \qquad \lim(\{a^n b^n \mid n \in \mathbb{Z}^+\})$$

Theorem 43 *A set of infinite words* L' *is accepted by a deterministic Büchi automaton if, and only if, there exists a set of finite words* L *that is accepted by a deterministic automaton and s.t.* $L' = \lim(L)$*.*

Proof. Let $\mathcal{A} = \langle Q, \{q_0\}, \Sigma, \delta, F\rangle$ be a deterministic automaton over finite words, and $\mathcal{A}' = \langle Q, \{q_0\}, \Sigma, \delta, F\rangle$ be the same automaton, but seen as a deterministic Büchi automaton. Let $\sigma \in \Sigma^\omega$. Then if $\sigma \in \mathcal{L}(\mathcal{A}')$, the execution ρ of $\mathcal{A}'$ over σ enters infinitely many times in one of the states of F, say q_f. Each prefix of ρ is an execution of $\mathcal{A}$ over a corresponding prefix of σ. Thus, infinitely many prefixes of σ are accepted by $\mathcal{A}$. The converse implication follows the same lines. □

Theorem 44 *The set $\mathcal{L}$ of infinite words over $\Sigma = \{a, b\}$ containing only finitely many a's is not accepted by a deterministic Büchi automaton.*

Proof. Assume $\mathcal{L}$ is accepted by such an automaton. Then $\mathcal{L} = \lim(L)$, for some language L of finite words. Then $b^\omega \in \mathcal{L}$, thus $b_{n_1} \in L$ for some n_1. In the same way, $b^{n_1} \cdot a \cdot b^\omega \in \mathcal{L}$, which entails that $b^{n_1} \cdot a \cdot b^{n_2} \in L$ for some n_2. Repeating the same argument again and again, we obtain an infinite sequence $(w_i)_{i\in\mathbb{Z}}$ of words of L s.t. w_i is a prefix of w_{i+1} and w_i contains i occurrences of a. Since $\mathcal{L}$ is the limit of L, it contains a word having infinitely many a's, which is a contradiction. □

We complete the proof by remarking that the automaton of Fig. 3.1 accepts precisely the complement of the language $\mathcal{L}$ used in the above theorem. □

Exercice 3.17. ⋆⋆⋆ Prove that non-deterministic Büchi automata are closed under complement. From Theorem 40, it follows that Büchi automata are not determinizable.

3.5.2 From LTL+Past to Büchi automata

This section is entirely devoted to the proof of the following result:

Theorem 45 ([LPZ85]) *Let ϕ be a formula in LTL+Past. Then one can build a (generalized) Büchi automaton $\mathcal{A}$ over the alphabet $\Sigma = 2^{AP}$, having at most $2^{2|\phi|}$ states, and such that:*

$$\text{for any word } w \in \Sigma^\omega. \quad w \in \mathcal{L}(\mathcal{A}) \Leftrightarrow \langle w, 0\rangle \models \phi.$$

Note that we cite [LPZ85] here because it handles LTL+Past, but a the original translation of LTL to Büchi automata appeared in [WVS83].

Proof. We begin with the intuition behind this construction, and then give the technical details.

Each state of our automaton will correspond to a set of subformulas of ϕ that are or have to be fulfilled if the execution is accepting. For example, from a state corresponding to (among others) $\mathbf{X}\,\psi$, outgoing transitions will go to states corresponding to (among others) ψ. That way, the automaton will "propagate" the subformulas that still have to be fulfilled. The (generalized) Büchi condition will enforce that "until" formulas are eventually fulfilled.

We now turn to the formal proof. We first need to define the set of subformulas:

Definition 46 *Let ϕ be a formula in LTL+Past. The* closure $\mathsf{Cl}(\phi)$ *of ϕ is the smallest set of formulas containing ϕ and closed under the following rules:*

- *for any $\psi \in$ LTL+Past, $\psi \in \mathsf{Cl}(\phi)$ if, and only if, $\neg\psi \in \mathsf{Cl}(\phi)$ (where we identify $\neg\neg\psi$ with ψ);*
- *if $\psi_1 \wedge \psi_2 \in \mathsf{Cl}(\phi)$ or if $\psi_1 \vee \psi_2 \in \mathsf{Cl}(\phi)$, then both ψ_1 and ψ_2 are in $\mathsf{Cl}(\phi)$;*
- *if $\mathbf{X}\,\psi \in \mathsf{Cl}(\phi)$ or if $\mathbf{X}^{-1}\,\psi \in \mathsf{Cl}(\phi)$, then $\psi \in \mathsf{Cl}(\phi)$;*
- *if $\psi_1\,\mathbf{U}\,\psi_2 \in \mathsf{Cl}(\phi)$, then ψ_1, ψ_2 and $\mathbf{X}\,(\psi_1\,\mathbf{U}\,\psi_2)$ are in $\mathsf{Cl}(\phi)$;*
- *if $\psi_1\,\mathbf{S}\,\psi_2 \in \mathsf{Cl}(\phi)$, then ψ_1, ψ_2 and $\mathbf{X}^{-1}\,(\psi_1\,\mathbf{S}\,\psi_2)$ are in $\mathsf{Cl}(\phi)$.*

With this definition, we have the following result:

Proposition 47

$$|\mathsf{Cl}(\phi)| \leq 4\,|\phi|\,.$$

Exercice 3.18. ⋆ Prove Proposition 47.

Of course, we have to enforce that a state cannot represent both a subformula ψ and its negation. Conversely, we will also require that each state represent exactly one of ψ and $\neg\psi$:

Definition 48 *Let ϕ be a formula in LTL+Past. A subset S of $\mathsf{Cl}(\phi)$ is said to be* maximal consistent *if the following conditions are fulfilled:*

- *for any $\psi \in \mathsf{Cl}(\phi)$, $\psi \in S$ if, and only if, $\neg\psi \notin S$;*
- *for any $\psi_1 \wedge \psi_2 \in \mathsf{Cl}(\phi)$, $\psi_1 \wedge \psi_2 \in S$ if, and only if, both ψ_1 and ψ_2 are in S;*
- *for any $\psi_1 \vee \psi_2 \in \mathsf{Cl}(\phi)$, $\psi_1 \vee \psi_2 \in S$ if, and only if, at least one of ψ_1 and ψ_2 is in S;*
- *for any $\psi_1\,\mathbf{U}\,\psi_2 \in \mathsf{Cl}(\phi)$, $\psi_1\,\mathbf{U}\,\psi_2 \in S$ if, and only if, either $\psi_2 \in S$, or both ψ_1 and $\mathbf{X}\,(\psi_1\,\mathbf{U}\,\psi_2)$ are in S;*
- *for any $\psi_1\,\mathbf{S}\,\psi_2 \in \mathsf{Cl}(\phi)$, $\psi_1\,\mathbf{S}\,\psi_2 \in S$ if, and only if, either $\psi_2 \in S$, or both ψ_1 and $\mathbf{X}^{-1}\,(\psi_1\,\mathbf{S}\,\psi_2)$ are in S.*

Note that this definition does not forbid that a maximal consistent set contains both $\mathbf{X}\,p$ and $\mathbf{X}\,\neg p$.

Proposition 49 *For any LTL+Past formula ϕ, there are at most $2^{2|\phi|}$ maximal consistent subsets of $\mathsf{Cl}(\phi)$.*

This result directly follows from Proposition 47.

We now turn to the formal definition of the automaton corresponding to ϕ. As should now be clear, the set of states will precisely be the set of maximal consistent subsets of $\mathsf{Cl}(\phi)$. And if a subset contains a subformula $\mathbf{X}\,\psi$, its successors will have to contain ψ.

Definition 50 *Let ϕ be a formula in LTL+Past. The* Büchi automaton associated to ϕ *is the generalized Büchi automaton $\mathcal{A}_\phi = \langle Q, Q_0, 2^{AP}, \delta, F\rangle$ satisfying the following conditions:*

- *Q is the set of maximal consistent subsets of $\mathsf{Cl}(\phi)$;*
- *Q_0 is the subset of Q consisting of those maximal consistent subsets that contain ϕ and contain $\neg\mathbf{X}^{-1}\,\psi$ for any subformula $\mathbf{X}^{-1}\,\psi \in \mathsf{Cl}(\phi)$;*
- *$(q, \sigma, q') \in \delta$ if, and only if, the following conditions are fulfilled:*
  - *for any $p \in AP$, we have $p \in q$ if, and only if, $p \in \sigma$;*
  - *for any subformula $\mathbf{X}\,\psi \in \mathsf{Cl}(\phi)$, we have $\mathbf{X}\,\psi \in q$ if, and only if, $\psi \in q'$;*
  - *for any subformula $\mathbf{X}^{-1}\,\psi \in \mathsf{Cl}(\phi)$, we have $\psi \in q$ if, and only if, $\mathbf{X}^{-1}\,\psi \in q'$.*
- *the set of accepting sets of states is defined as follows:*

$$F = \left\{ \{q \in Q \mid \psi_2 \in q \text{ or } \neg(\psi_1\,\mathbf{U}\,\psi_2) \in q\} \;\middle|\; \psi_1\,\mathbf{U}\,\psi_2 \in \mathsf{Cl}(\phi) \right\}.$$

The size of this automaton clearly fulfills the requirements of Theorem 45. The following lemma will conclude the proof that our construction is correct:

Lemma 51 *Let $\phi \in$ LTL+Past, and $w \in (2^{AP})^\omega$. Let $\rho = (q_i)_{i\in\mathbb{Z}^+}$ be an execution of $\mathcal{A}_\phi$ on input word w, starting in some initial state q_0, and accepted by $\mathcal{A}_\phi$. Then for any $\psi \in \mathsf{Cl}(\phi)$, and for any $n \in \mathbb{Z}^+$,*

$$\psi \in q_n \quad \Leftrightarrow \quad \langle w, n\rangle \models \psi.$$

Proof. The proof is by induction of the structure of ψ:

- if ψ is an atomic proposition p, the proof is rather obvious: assuming $p \in q_n$, we get, by construction of the transitions, that, for any $(q_n, \sigma, q') \in \delta$, $p \in \sigma$. Thus $p \in w_n$ and $\langle w, n\rangle \models p$. The converse follows the same lines;
- the proofs for boolean combinators is straightforward;
- if $\psi = \mathbf{X}\,\psi_1$: by construction, $\psi \in q_n$ if, and only if, $\psi_1 \in q_{n+1}$. By induction hypothesis, this is equivalent to $\langle w, n+1\rangle \models \psi_1$, which in turns exactly means that $\langle w, n\rangle \models \psi$. The proof for $\psi = \mathbf{X}^{-1}\,\psi_1$ is similar;

- if $\psi = \psi_1 \mathbf{S} \psi_2$, we prove both implications by induction on n. First, if $\psi \in q_0$, since q_0 is an initial state, we know that is does not contain $\mathbf{X}^{-1}(\psi_1 \mathbf{S} \psi_2)$. By construction, q_0 must contain ψ_2, and by induction hypothesis, $\langle w, 0\rangle \models \psi_2$, hence $\langle w, 0\rangle \models \psi$. The converse follows the same lines.

  Now, if the result holds up to some posittion $n-1$, assume that $\phi \in q_n$. Then, by construction of maximal consistent sets, either $\psi_2 \in q_n$, or both ψ_1 and $\mathbf{X}^{-1}(\psi_1 \mathbf{S} \psi_2)$ are in q_n. The former case is straightforward. In the latter, by construction, we get that $\psi_1 \mathbf{S} \psi_2 \in q_{n-1}$. We get that $\langle w, n\rangle \models \psi_1$ (from the induction hypothesis on ψ) and $\langle w, n-1\rangle \models \phi_1 \mathbf{S} \psi_2$ (by induction hypothesis in n). This immediately yields that $\langle w, n\rangle \models \psi_1 \mathbf{S} \psi_2$. Again, the other direction is symmetric.

- we conclude this proof with the case $\psi = \psi_1 \mathbf{U} \psi_2$. Assume $\psi \in q_n$. Since the run is accepted by the automaton, we know that one state of $\{q \in Q \mid \psi_2 \in q \text{ or } \neg(\psi_1 \mathbf{U} \psi_2) \in q\}$ will be encountered after q_n along ρ. We reason by induction on the distance d between q_n and the next such state. If $d = 0$, then $\psi_2 \in q_n$, since it can't be the case that $\neg(\psi_1 \mathbf{U} \psi_2) \in q_n$. By induction hypothesis, we get the result. Now, assuming the result holds up to distance $d-1$ with $d > 1$, suppose the distance between q_n and the next accepting state for $\psi_1 \mathbf{U} \psi_2$ is d. Then q_n is not accepting, and thus it does not contain ψ_2. It shall then contain both ψ_1 and $\mathbf{X}(\psi_1 \mathbf{U} \psi_2)$. It follows that q_{n+1} contains $\psi_1 \mathbf{U} \psi_2$. The distance between q_{n+1} and the next accepting state is then $d-1$, which entails, by indcution on d, that $\langle w, n+1\rangle \models \psi_1 \mathbf{U} \psi_2$. now, since q_n contains ψ_1, we obtains that $\langle w, n\rangle \models \psi_1 \mathbf{U} \psi_2$.

  Conversely, if $\langle w, n\rangle \models \psi_1 \mathbf{U} \psi_2$, then there exists an integer d s.t. $\langle w, n+d\rangle \models \psi_2$, and for any $i \in [0, d)$, $\langle w, n+i\rangle \models \psi_1$. By induction hypothesis, q_{n+d} contains ψ_2, thus also $\psi_1 \mathbf{U} \psi_2$, and all q_{n+i} contain ψ_1. An easy inductive argument proves that $\psi_1 \mathbf{U} \psi_2 \in q_n$, and concludes the proof. □□

Example. *Figure 3.2 displays the Büchi automaton corresponding to formula* $a \mathbf{U} (b \wedge \mathbf{X}^{-1} b)$. *the closure of this formula contains* 12 *subformulas, thus the automaton could contain up to* 2^6*, i.e.,* 64 *states. Still, consistency requirements lower this value to* 16. *For the sake of clarity, labels are not shown on transitions, since they correspond, by construction, to the set of atomic propositions labeling the source state.*

Exercice 3.19. ⋆⋆ Compute the Büchi automaton corresponding to $a \mathbf{U} (b \wedge \mathbf{X}^{-1} b)$. Can you build a smaller Büchi automaton that accepts the same language? Can you build a deterministic Büchi automaton that accepts the same language?

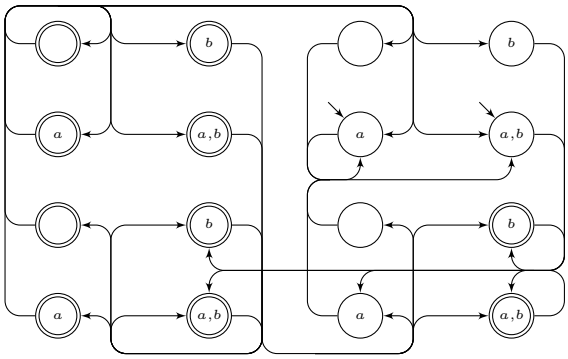


Figure 3.2: The Büchi automaton corresponding to formula $a \,\mathbf{U}\, (b \wedge \mathbf{X}^{-1}\, b)$.

3.5.3 Application to verification

The transformation above has very interesting algorithmic consequences: the satisfiability of an LTL+Past formula is equivalent to the non-emptiness of the language accepted by the associated Büchi automaton. While testing for the non-emptiness for Büchi automata is NLOGSPACE-complete (this in PTIME), the naive approach only yields an EXPTIME algorithm since the Büchi automaton associated to an LTL+Past formula has exponential size. This does not match the lower bound that we established in Theorem 31.

Still, this automata-theoretic approach can yield an optimal algorithm:

Theorem 52 *Satisfiability of an LTL+Past formula can be achieved in PSPACE.*

Proof. Instead of building the whole Büchi automaton associated to the formula ϕ under study, the technique consists in building an accepting execution of the automaton "on the fly", in a non-deterministic manner (which is fine since PSPACE = NPSPACE).

The algorithm first guesses an initial state q_0 and a set of accepting states q_1 to q_p (one state per set of accepting states of the generalized Büchi condition). It then guesses, on the fly, sequences of states of the Büchi automaton that run from q_0 to q_1, from q_1 to q_2, and so on, and from q_p to q_1. Here, "on the fly" means that the algorithm generates a set of subformulas of ϕ, check that it is maximal consistent, and that there is a transition from the previous state to this one.

Clearly, the language of the automaton is non-empty if, and only if, such choices are possible. Now, this algorithm only needs polynomial space, since it only has to store the states q_0 to q_p, plus the previous- and current state of the sequence it is guessing during the second phase, and a small amount of auxiliary informations.

This algorithm runs in space $O(|\phi|^2)$ and in time $O(|\phi|^2 \times 2^{2|\phi|})$. □

Exercice 3.20. ⋆⋆ Explain the above space- and time bounds of the algorithm for LTL+Past satisfiability.

An immediate corollary of this algorithm is the "ultimately-periodic model property" for LTL+Past:

Corollary 53 *If a formula ϕ in LTL+Past is satisfiable, then there exist two finite words $u, v \in (2^{AP})^*$ s.t. $\langle u \cdot v^\omega, 0\rangle \models \phi$ and $|u| + |v| \leq |\phi| \cdot 2^{2|\phi|}$.*

The same complexity can be achieved for model-checking:

Theorem 54 *LTL+Past model-checking is in PSPACE.*

Proof. The algorithm is very similar, but the algorithm now guesses pairs of states: one state of the Büchi automaton corresponding to the formula ϕ, and one state of the Kripke structure $\mathcal{K}$.

While one state of the Büchi automaton is stored in space $O(|\phi|)$, one state of the Kripke structure only requires space $O(\log(|\mathcal{K}|))$. This algorithm thus performs in space $O(|\phi|^2 \cdot \log(|\mathcal{K}|))$, and time $O(|\phi|^2 \times |\mathcal{K}| \times 2^{2|\phi|})$. □

Exercice 3.21. ⋆⋆ Explain the above space- and time bounds of the algorithm for LTL+Past model-checking.

Remark. *It is important to notice that, practically, the time-complexity of LTL+Past model-checking is exponential only in the size of the formula (which is generally rather small), while it is linear in the size of the Kripke structure (which is often very large). The same difference applies to space-complexity: the algorithm is (non-deterministic) logarithmic space in the size of the Kripke structure, and polynomial only in the size of the formula.*

As a corollary of these results and those of Section 3.4, we get:

Theorem 55 *Model-checking and satisfiability for LTL and LTL+Past are PSPACE-complete.*

When the full expressiveness of LTL+Past is not needed, it is possible to find sublogics that enjoy better complexity. We present one such case here.

Definition 56 *The fragment of LTL* without temporal nesting*, denoted with LTL_1, is the fragment of LTL defined by the following grammar:*

$$LTL_1 \ni \phi_m ::= \phi_p \mid \neg\phi_m \mid \phi_m \vee \phi_m \mid \phi_m \wedge \phi_m \mid \mathbf{X}\,\phi_p \mid \phi_p\,\mathbf{U}\,\phi_p$$
$$\phi_p ::= \top \mid p \mid \neg\phi_p \mid \phi_p \vee \phi_p \mid \phi_p \wedge \phi_p$$

where p ranges over AP.

We now prove that, from the ultimately-periodic-model property of LTL+Past, we can derive a small-model property, that is claimed as follows:

Proposition 57 ([DS02]) *If a formula ϕ in LTL_1 is satisfiable, then there exist two finite words $u, v \in (2^{AP})^*$ s.t. $\langle u \cdot v^\omega, 0\rangle \models \phi$ and $|u| + |v| \leq |\phi| \cdot 2\,|\phi|$.*

Proof. The proof consists in "extracting" polynomialy many witnesses from the exponential ultimately-periodic witness given by Corollary 53.

Let $\phi \in \mathsf{LTL}_1$, and $w = u \cdot v^\omega$ be an ultimately-periodic word witnessing ϕ. We select a family of witnesses as follows:

- 0 is a witness;
- if ϕ contains a subformula of the form $\mathbf{X}\,\psi$, then 1 is a witness;
- for any subformula $\psi_1\,\mathbf{U}\,\psi_2$ of ϕ, several cases may arise:
  - if $w, 0 \models \psi_1\,\mathbf{U}\,\psi_2$, then the smallest i s.t. $w, i \models \psi_2$ is a witness;
  - if $w, 0 \not\models \psi_1\,\mathbf{U}\,\psi_2$ but $w, 0 \models \mathbf{F}\,\psi_2$, then the smallest i s.t. $w, i \not\models \psi_1$ is a witness;
  - otherwise, we select no witness.

Clearly, this procedure selects at most $|\phi|$ integers, all of which are less than $|u| + |v|$. We write $\{i_0, i_1, ..., i_k\}$ for the (ordered) set of witnesses, with $i_0 = 0$. We define two words $u' = w_{i_0} w_{i_1} \cdots w_{i_p}$ and $v' = w_{i_{p+1}} w_{i_{p+2}} \cdots w_{i_k}$, with $i_p < |u| \leq i_{p+1}$. That way, u' is a subword of u, v' is a subword of v, and both subwords have size linear in the size of ϕ. It remains to show that $u' \cdot v'^\omega$ satisfies ϕ. In the sequel, we write w' for $u' \cdot v'^\omega$.

The proof is by induction on the structure of ϕ: the base cases (including the case where $\phi = \mathbf{X}\,\phi_1$) are obvious, and we focus on the case where $\phi = \phi_1\,\mathbf{U}\,\phi_2$. We prove that $w \models \phi$ iff $w' \models \phi$.

Assume $w, 0 \models \phi$. Then there exists a position i s.t. $w, i \models \phi_2$, and such that all intermediate positions satisfy ϕ_1. If we assume that i is the smallest such position, then state w_i occurs in w' as $w'_{i'}$ (and we have $w', i' \models \phi_2$ since ϕ_2 is a propositional logic formula), and all intermediate positions along w' correspond to intermediate positions along w (and each of them thus satisfy ϕ_1). As a consequence, $w', 0 \models \phi$.

Conversely, if $w, 0 \not\models \phi$, then two cases may arise: either $w, 0 \not\models \mathbf{F}\,\phi_2$, in which case it is also the case for w', or $w, 0 \models \mathbf{F}\,\phi_2$, but some earlier position not satisfying ϕ_1 occurs in w'. In both cases, $w', 0 \not\models \phi$. □

Theorem 58 *Deciding the satisfiability of a formula of LTL_1 is NP-complete.*

Proof. The algorithm consists in guessing an ultimately-periodic witness $u \cdot v^\omega$, and verifying that it really satisfies the formula. We know that the first part can be achieved in polynomial time. The second part is an easy labeling algorithm: it consists in labeling each state of the witness with the subformulas it satisfies (using the obvious fact that two positions p and $p + |v|$, with $p \geq |u|$, satisfy the same subformulas). □

Theorem 59 *Model-checking LTL_1 is NP-complete.*

Proof. The argument is similar: from a path satisfying $\phi \in \mathsf{LTL}_1$, we know that only polynomialy many witnesses are needed. Clearly, there exists a polynomial path containing those witnesses. The algorithm consists again in guessins a path and checking that it really satisfies the formula. □

3.5.4 Application to expressiveness

The automata-theoretic approach also gives a surprising way of proving the following expressiveness result:

Theorem 60 ([LMS02]) *LTL+Past can be* exponentially *more succinct than LTL.*

That is, there exists a sequence of formulas of LTL+Past whose translations in LTL are exponentially larger.

Proof. We consider the following property:

> Any two states that agree on atomic propositions p_1 to p_n also agree on proposition p_0. (3.1)

This formula can easily be expressed in LTL, e.g. by enumerating the set of valuations for p_1 to p_n:

$$\bigwedge_{a_0,a_1,\ldots,a_n \in \{\top,\bot\}} (\mathbf{F}\,(\bigwedge_{i\in[0,n]} p_i = a_i)) \Rightarrow (\mathbf{G}\,((\bigwedge_{i\in[1,n]} p_i = a_i) \Rightarrow p_0 = a_0)).$$

The set of words satisfying this property can thus be accepted by a Büchi automaton. It turns out that we have the following lemma:

Lemma 61 ([EVW02]) *Any Büchi automaton that accepts exactly the set of words satisfying Equation (3.1) has at least 2^{2^n} states.*

Proof. Let $\{a_0, a_1, ..., a_{2^n-1}\}$ be the set of letters built on atomic propositions $\{p_1, ..., p_n\}$ (note that p_0 is not included here). There are 2^n such letters.

For each subset $K \subseteq \{0, 1, ..., 2^n - 1\}$, we define a set of letters $\{b_0, ..., b_{2^n-1}\}$ as follows:

$$b_i = \begin{cases} a_i & \text{if } i \notin K \\ a_i \cup \{p_0\} & \text{otherwise.} \end{cases}$$

Then, with each subset K, we associate a finite word $w_K = b_0\, b_1 \ldots b_{2^n-1}$. Clearly, whenever $K \neq K'$, then $w_K \neq w_{K'}$. Thus, this construction defines 2^{2^n} different finite words. It is clear also that for any K, the word ${w_K}^\omega$ satisfies Equation (3.1), while the word $w_{K'} \cdot {w_K}^\omega$ does not, as soon as $K \neq K'$.

Now, let $\mathcal{A}$ be an automaton accepting exactly the words satisfying Equation (3.1). Let S_K be the set of states that can be reached from an initial state

of $\mathcal{A}$ after reading the finite word w_K. Clearly, this set is non-empty, since w_K is the prefix of an accepted word. Now, if $S_K \cap S_{K'} \neq \varnothing$ for two different sets K and K', then the word $w_{K'} \cdot {w_K}^\omega$ would be accepted by the automaton. Thus the sets S_K do not intersect, which entails that $\mathcal{A}$ has at least 2^{2^n} states. □

As a corollary of Theorem 45, any LTL+Past formula expressing the property of equation (3.1) has size at least 2^{n-1}.

We now consider a slightly different property:

> Any state that agrees with the initial state on atomic propositions p_1 to p_n also agree with the initial state on proposition p_0. (3.2)

This formula can be captured by a polynomial-size LTL+Past formula:

$$\mathbf{G}\left[\left(\bigwedge_{i\in[1,n]} (p_i \Leftrightarrow \mathbf{F}^{-1}\,\mathbf{G}^{-1}\,p_i)\right) \Rightarrow (p_0 \Leftrightarrow \mathbf{F}^{-1}\,\mathbf{G}^{-1}\,p_0)\right]. \tag{3.3}$$

It can thus also be expressed in LTL (again, possibly by enumerating the set of initial valuations). Let ϕ be an LTL formula expressing the property of Equation (3.2). Since ϕ is a pure-future formula, formula $\mathbf{G}\,\phi$ precisely expresses the property of Equation (3.1), and thus has size at least 2^{n-1}. Thus $|\phi| \geq 2^{n-1}-1$. Thus LTL can only express the LTL+Past formula (3.3) with exponential-size formulas.

Exercice 3.22. ⋆⋆ The proof above requires infinitely many atomic propositions. This is in contradiction with our original requirement that the set AP must be finite. Prove that the result still holds with only a finite number of atomic propositions.

Chapter 4

Branching-time temporal logics

4.1 Tree structures

Similarly to linear-time temporal logics, branching-time temporal logics express properties on the evolutions of a system along time. The difference is that branching-time logics consider the whole set of possible evolutions of the system, while linear-time logics only deal with one execution at a time. We begin with formally defining tree structures, which are the branching-time equivalent to linear structures (see Definition 16).

Definition 62 *A* (labeled) tree structure *is a tuple* $\mathcal{S} = \langle T, \leq, \ell \rangle$ *where:*

- $\langle T, \leq \rangle$ *is a tree-ordered (infinite) set (i.e., a partially-ordered set such that, for each* $t \in T$*, the set* $\langle \{u \in T \mid u \leq t\}, \leq \rangle$ *is totally ordered and has a least element) with the following two additional requirements:*
  - *a* branch *is a maximal totally-ordered subset of* T*. Each branch of a tree structure is required to be infinite. We write* $\mathsf{Br}(t)$ *for the set of branches of* T *containing* t*;*
  - *a* root *is a minimal element of* T*. It is required that a tree structure has finitely many roots (and, sometimes, only one);*
- $\ell \colon T \to 2^{\mathsf{AP}}$ *labels each point of* T *with a subset of* AP*.*

It should be remarked that a branch is well-ordered, and can thus be seen as a linear structure. As in the case of linear time, only two kinds of tree structures are really relevant to computer science: those in which all branches are isomorphic to $\mathbb{R}^+$, and those in which all branches are isomorphic to $\mathbb{Z}^+$. In the sequel, we will only consider discrete-time, where all branches are isomorphic to $\mathbb{Z}^+$.

Definition 63 *Let $\mathcal{K} = \langle W, R, l\rangle$ be a Kripke structure, and $w \in W$. The* computation tree *of $\mathcal{K}$ from w is the tree structure $\langle T, \leq, \ell\rangle$ where*

- *T is the set of finite words over W whose first letter is w, and s.t., for any two consecutive letters w_i and w_{i+1}, $(w_i, w_{i+1}) \in R$;*
- *$\leq$ is the lexicographic order over T;*
- *ℓ maps each word to the label of its last letter: $\ell(w_0, w_1, ..., w_n) = l(w_n)$.*

Exercice 4.23. ⋆ Prove that the computation tree of a finite-state Kripke structure is a tree structure whose branches are isomorphic to $\mathbb{Z}^+$.

Definition 64 *A* pointed tree structure *is a couple $\langle \mathcal{S}, s\rangle$ where $\mathcal{S} = \langle T, \leq, \ell\rangle$ is a tree structure, and $s \in T$. We write $\mathcal{M}_{br(T)}$ for the set of pointed tree structures built on the time-line T.*

4.2 Branching-time temporal logics

There are two ways of defining branching-time temporal logics:

- either as an extension of LTL (or LTL+Past) with *path quantifiers*;
- or as a truly modal logic, i.e., a logic built on boolean operators and a set of modalities.

We will mostly use the first definition here, and will only mention links with the second definition as exercises. Thus, given a pointed tree structure $\langle \mathcal{S}, s\rangle$, we define two *path quantifiers* as follows:

- **E** (there exists a path):

$$\langle \mathcal{S}, s\rangle \models \mathbf{E}\phi \overset{\text{def}}{\Leftrightarrow} \exists b \in \mathsf{Br}(s).\ \langle b, s\rangle \models \phi.$$

- **A** (for all paths):

$$\langle \mathcal{S}, s\rangle \models \mathbf{A}\phi \overset{\text{def}}{\Leftrightarrow} \forall b \in \mathsf{Br}(s).\ \langle b, s\rangle \models \phi.$$

This definition is rather informal for the moment, but will be make more precise at Definition 67.

Example. *Formula $\mathbf{E}(\mathbf{F}\,p \wedge \mathbf{F}\,q)$ means that, along some path, a state labeled with p and one labeled with q are reachable. This is of course different from $\mathbf{EF}\,(p \wedge q)$, which requires that one state labeled with both p and q is reahcable, but also from $\mathbf{EF}\,p \wedge \mathbf{EF}\,q$, which states that p is reachable in one branch and q in another (but, of course, possibly the same) branch.*

Similarly, $\mathbf{AG}\,p$ means that all reachable states are labeled with p. This is equivalent to saying that any state that is not labeled with p is not reachable.

In fact, we again have an obvious duality between our two new operators:

Proposition 65 *Both path quantifiers are dual to each other: for any* $\phi \in$ *LTL+Past,*

$$\mathbf{A}\phi \equiv \neg\,\mathbf{E}\neg\phi.$$

We now define the syntax of three important families of branching-time temporal logics:

Definition 66 *Given a set* $\{M_1, ..., M_n\}$ *of* n *first-order definable modalities, we define the following three branching-time logics:*

- *the syntax of* $\mathcal{B}(M_1, ..., M_n)$ *is given by:*

$$\begin{aligned}\mathcal{B}(M_1, ..., M_n) \ni \phi_b ::= & \ p \mid \neg\phi_b \mid \phi_b \vee \phi_b \mid \phi_b \wedge \phi_b \mid \mathbf{E}\phi_l \mid \mathbf{A}\phi_l \\ \phi_l ::= & \ M_1(\phi_b, ..., \phi_b) \mid \cdots \mid M_n(\phi_b, ..., \phi_b)\end{aligned}$$

  where p *ranges over* AP.

- *the syntax of* $\mathcal{B}^+(M_1, ..., M_n)$ *is given by:*

$$\begin{aligned}\mathcal{B}^+(M_1, ..., M_n) \ni \phi_b ::= & \ p \mid \neg\phi_b \mid \phi_b \vee \phi_b \mid \phi_b \wedge \phi_b \mid \mathbf{E}\phi_l \mid \mathbf{A}\phi_l \\ \phi_l ::= & \ \neg\phi_l \mid \phi_l \vee \phi_l \mid \phi_l \wedge \phi_l \mid \\ & M_1(\phi_b, ..., \phi_b) \mid \cdots \mid M_n(\phi_b, ..., \phi_b)\end{aligned}$$

  where p *ranges over* AP.

- *the syntax of* $\mathcal{B}^*(M_1, ..., M_n)$ *is given by:*

$$\begin{aligned}\mathcal{B}^*(M_1, ..., M_n) \ni \phi_b ::= & \ p \mid \neg\phi_b \mid \phi_b \vee \phi_b \mid \phi_b \wedge \phi_b \mid \mathbf{E}\phi_l \mid \mathbf{A}\phi_l \\ \phi_l ::= & \ \phi_b \mid \neg\phi_l \mid \phi_l \vee \phi_l \mid \phi_l \wedge \phi_l \mid \\ & M_1(\phi_l, ..., \phi_l) \mid \cdots \mid M_n(\phi_l, ..., \phi_l)\end{aligned}$$

  where p *ranges over* AP.

Formulas defined as ϕ_l *are called* path formulas*, while those defined as* ϕ_b *are* state formulas*.*

Again, for modalities $\mathbf{U}$ *and* $\mathbf{X}$*, the corresponding logics have been given special names ("*CTL*" stands for "computational tree logic"):*

$$\begin{aligned}\mathsf{CTL} &= \mathcal{B}(\mathbf{U}, \mathbf{X}) & \mathsf{CTL}_s &= \mathcal{B}(\tilde{\mathbf{U}}) \\ \mathsf{CTL}^+ &= \mathcal{B}^+(\mathbf{U}, \mathbf{X}) & \mathsf{CTL}^+_s &= \mathcal{B}^+(\tilde{\mathbf{U}}) \\ \mathsf{CTL}^* &= \mathcal{B}^*(\mathbf{U}, \mathbf{X}) & \mathsf{CTL}^*_s &= \mathcal{B}^*(\tilde{\mathbf{U}})\end{aligned}$$

Example. *In* CTL*, each modality has to be preceeded with a path quantifier:* $\mathbf{EF}\,p$ *is a* CTL *formula, but* $\mathbf{E}(\mathbf{F}\,p \wedge \mathbf{F}\,q)$ *and* $\mathbf{E}(\mathbf{F}\,\mathbf{G}\,p)$ *are not (at least, syntactically).*

Clearly enough, CTL^+ contains CTL: CTL^+ allows boolean combinations of modalities in the scope of path quantifiers. Still, modalities cannot be directly nested. Thus, $\mathbf{E}\mathbf{F}\,p$ and $\mathbf{E}(\mathbf{F}\,p \wedge \mathbf{F}\,q)$ are two formulas of CTL^+, but $\mathbf{E}(\mathbf{F}\,\mathbf{G}\,p)$ does not follow the syntax of CTL^+.

Last, it is clear again that CTL^* subsumes CTL^+: CTL^* allows the nesting of modalities. All three formulas $\mathbf{E}\mathbf{F}\,p$, $\mathbf{E}(\mathbf{F}\,p \wedge \mathbf{F}\,q)$ and $\mathbf{E}(\mathbf{F}\,\mathbf{G}\,p)$ are CTL^* formulas.

We now define the precise semantics of CTL^*. The semantics of the other logics will follow.

Definition 67 *Let $\mathcal{S} = \langle T, \leq, \ell\rangle$ be a tree structure, $s \in T$, and $b \in \mathit{Br}(s)$. Let ϕ_b, ψ_b be two state formulas for CTL^* semantics, and ϕ_l, ψ_l be two path formulas for CTL^* semantics. We recursively define two relations, denoted with $\models_b$ and $\models_l$, for the satisfaction of state- and path formulas, respectively:*

$$
\begin{array}{rcl}
\langle \mathcal{S}, s\rangle \models_b p & \Leftrightarrow & p \in \ell(t) \\
\langle \mathcal{S}, s\rangle \models_b \neg\phi_b & \Leftrightarrow & \langle \mathcal{S}, s\rangle \not\models_b \phi_b \\
\langle \mathcal{S}, s\rangle \models_b \phi_b \vee \psi_b & \Leftrightarrow & \langle \mathcal{S}, s\rangle \models_b \phi_b \text{ or } \langle \mathcal{S}, s\rangle \models_b \psi_b \\
\langle \mathcal{S}, s\rangle \models_b \phi \wedge \psi & \Leftrightarrow & \langle \mathcal{S}, s\rangle \models_b \phi_b \text{ and } \langle \mathcal{S}, s\rangle \models_b \psi_b \\
\langle \mathcal{S}, s\rangle \models_b \mathbf{E}\phi_l & \Leftrightarrow & \exists b \in \mathit{Br}(s).\ \langle \mathcal{S}, b, s\rangle \models_l \phi_l \\
\langle \mathcal{S}, s\rangle \models_b \mathbf{A}\phi_l & \Leftrightarrow & \forall b \in \mathit{Br}(s).\ \langle \mathcal{S}, b, s\rangle \models_l \phi_l \\
 & & \\
\langle \mathcal{S}, b, s\rangle \models_l \phi_b & \Leftrightarrow & \langle \mathcal{S}, s\rangle \models_b \phi_b \\
\langle \mathcal{S}, b, s\rangle \models_l \neg\phi_l & \Leftrightarrow & \langle \mathcal{S}, b, s\rangle \not\models_l \phi_l \\
\langle \mathcal{S}, b, s\rangle \models_l \phi_l \vee \psi_l & \Leftrightarrow & \langle \mathcal{S}, b, s\rangle \models_l \phi_l \text{ or } \langle \mathcal{S}, b, s\rangle \models_l \psi_l \\
\langle \mathcal{S}, b, s\rangle \models_l \phi_l \wedge \psi_l & \Leftrightarrow & \langle \mathcal{S}, b, s\rangle \models_l \phi_l \text{ and } \langle \mathcal{S}, b, s\rangle \models_l \psi_l \\
\langle \mathcal{S}, b, s\rangle \models_l \mathbf{X}\,\phi_l & \Leftrightarrow & \exists t \in b.\ t > s \wedge \forall u.\ (u \leq s \vee t \leq u) \wedge \langle \mathcal{S}, b, t\rangle \models_l \phi_l \\
\langle \mathcal{S}, b, s\rangle \models_l \phi_l\,\mathbf{U}\,\psi_l & \Leftrightarrow & \exists t \in b.\ (s \leq t \wedge (\langle \mathcal{S}, b, s\rangle \models_l \psi_l)) \wedge \\
 & & \qquad \forall u \in b.\ ((s \leq u \wedge u \leq t) \Rightarrow \langle \mathcal{S}, b, u\rangle \models_l \phi_l)
\end{array}
$$

In the sequel, we write $\models$ for $\models_b$.

Given a Kripke structure $\mathcal{K} = \langle W, R, l\rangle$, some state $w \in W$, and a CTL formula ϕ, we write $\mathcal{K}, w \models \phi$ when $\langle \mathcal{S}_{\mathcal{K},w}, w\rangle \models \phi$, where $\mathcal{S}_{\mathcal{K},w}$ is the computation tree of $\mathcal{K}$ from w.

Of course, in a similar way as for linear-time, we can define a strict version $\tilde{\mathbf{U}}$ of $\mathbf{U}$. It is also possible to define other one-place modalities, e.g. $\mathbf{F}$ and $\mathbf{G}$, from $\mathbf{U}$.

Exercice 4.24. ⋆⋆ Prove, by exhibiting a witnessing Kripke structure, that the following formulas are satisfiable:

$$\mathbf{E}(\mathbf{F}\,p \wedge \mathbf{F}\,q) \wedge \neg\,\mathbf{E}\mathbf{F}\,(p \wedge\, \mathbf{E}\mathbf{F}\,q) \qquad \mathbf{E}\mathbf{G}\,(\neg p \wedge\, \mathbf{E}\mathbf{F}\,p)$$

$$\mathbf{E}\mathbf{G}\,(\,\mathbf{E}\mathbf{F}\,p) \wedge \neg\,\mathbf{E}\mathbf{G}\,\mathbf{F}\,p \qquad \mathbf{E}\mathbf{G}\,\mathbf{F}\,p \wedge \neg\,\mathbf{E}\mathbf{G}\,(\,\mathbf{A}\mathbf{F}\,p)$$

Exercice 4.25. ⋆⋆⋆ Let $\mathcal{A} = \langle W_\mathcal{A}, R_\mathcal{A}, l_\mathcal{A}\rangle$ and $\mathcal{B} = \langle W_\mathcal{B}, R_\mathcal{B}, l_\mathcal{B}\rangle$ be two Kripke structures. A *bisimulation* on $\mathcal{A}$ and $\mathcal{B}$ is a relation $B \subseteq W_\mathcal{A} \times W_\mathcal{B}$ s.t.:

- for any $(a, b) \in B$, $l_\mathcal{A}(a) = l_\mathcal{B}(b)$;
- for any $(a, b) \in B$, for any $(a, a') \in R_\mathcal{A}$, there exists $(b, b') \in R_\mathcal{B}$ s.t. $(a', b') \in B$;
- symmetrically, for any $(a, b) \in B$, for any $(b, b') \in R_\mathcal{B}$, there exists $(a, a') \in R_\mathcal{B}$ s.t. $(a', b') \in B$;

Two states $a \in \mathcal{A}$ and $b \in \mathcal{B}$ are *bisimilar* if there exists a bisimilation B s.t. $(a, b) \in B$.

As a classical counter-example, the initial states of the Kripke structures depicted on Figure 4.1 are not bisimilar. Find a CTL formula that holds on only one of those Kripke structures.

Prove that two states are bisimilar if, and only if, they satisfy the same formulas of CTL.

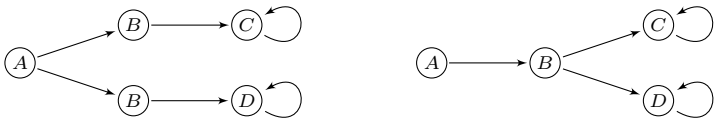


Figure 4.1: Two Kripke structures that are not bisimilar

Exercice 4.26. ⋆⋆⋆ Prove that $\mathsf{E}(p\,\mathbf{U}\,q \wedge \neg(r\,\mathbf{U}\,s))$ can be expressed in CTL over discrete time. Prove that CTL^+ is not more expressive than CTL.

Exercice 4.27. ⋆⋆ As mentionned at the beginning of this section, it is possible to define CTL in a really modal way. Define two one-place modalities **EX** and **AX**, and two two-place modalities **EU** and **AU**, and prove that CTL is equivalent to the logic defined from atomic propositions, boolean combinators, and those four modalities.

There is a characterization of CTL^* in terms of monadic second-order logic, analogous to Theorems 25, 26 and 27:

Lemma 68 *CTL^* can be expressed in SOMLO.*

Exercice 4.28. ⋆ Prove Lemma 68.

Exercice 4.29. ⋆⋆ Show that monadic second-order logic is strictly more expressive than CTL^*.

Still, there exists a fragment of monadic second-order logic that precisely characterizes CTL^*: *monadic path logic* (MPL) is the restriction of monadic second-order logic where second-order quantification is restricted to quantify

only on infinite branches, rather than on any set of states. MPL is then still more expressive than CTL* because it can express that a state has, for instance, exactly two successors labeled with p, while CTL* can only say that one or all of the successors have that property. By refining a little bit more:

Theorem 69 ([MR99]) *Any formula in MPL that is invariant under bisimulation can be expressed in CTL*.*

Bisimulation has been defined at Exercice 4.25: it allows here to consider only formulas of MPL that, roughly, will not count the number of successors of nodes.

4.3 Between CTL and CTL*

A fairness property is a property stating that something occurs infinitely often (cf. Exercise 3.7). LTL can express this class of properties: formula $\mathbf{G}\,\mathbf{F}\,p$ precisely states that p occurs infinitely often along p (along infinite paths).

Of course, CTL* can express fairness: formula $\mathbf{AG}\,\mathbf{F}\,p$ states that p occurs infinitely often along any path. Unfortunately, CTL cannot express fairness.

Exercice 4.30. ⋆⋆⋆ Using the families of Kripke structures depicted on Figure 4.2 prove that CTL cannot express fairness.

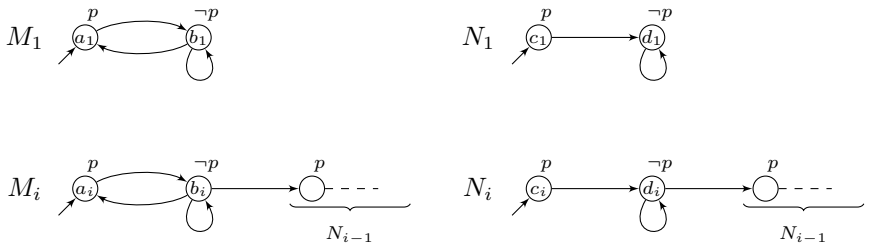


Figure 4.2: Two families of structures that CTL cannot distinguish.

Of course, CTL* can express fairness, but also much more. Intermediate logics have been defined in order to extend CTL with only fairness:

Definition 70 *Let $\mathcal{S} = \langle T, \leq, \ell\rangle$ be a tree structure, $s \in T$, and $b \in \mathsf{Br}(s)$. Let ϕ_l be a path formula for CTL* semantics. Modality $\overset{\infty}{\mathbf{F}}$ is defined as follows:*

$$\langle \mathcal{S}, b, s\rangle \models_l \overset{\infty}{\mathbf{F}}\,\phi_l \quad \Leftrightarrow \quad \forall t \in b.\ \exists u \in b.\ t \leq u \wedge \langle \mathcal{S}, b, u\rangle \models_l \phi_l$$

With this definition, formula $\mathbf{E}\overset{\infty}{\mathbf{F}}\,\phi$ is precisely equivalent to the CTL* formula $\mathbf{EG}\,\mathbf{F}\,\phi$. With this modality, the following two logics are defined:

$$\mathsf{ECTL} = \mathcal{B}(\mathbf{U}, \mathbf{X}, \overset{\infty}{\mathbf{F}}) \qquad \mathsf{ECTL}^+ = \mathcal{B}^+(\mathbf{U}, \mathbf{X}, \overset{\infty}{\mathbf{F}})$$

4.4 Model-checking branching-time logics

One important feature of CTL is that it enjoys very efficient model-checking algorithms:

Theorem 71 *CTL model-checking is PTIME-complete.*

Proof. We begin with PTIME-hardness, proving that the CIRCUIT-VALUE problem can be encoded in a CTL model-checking problem. CIRCUIT-VALUE consists in computing the value of the root of a directed acyclic graph whose nodes represent boolean connectives, and having two leaves $\top$ and $\bot$. An example of such a circuit is depicted on Figure 4.3. This problem is known to be

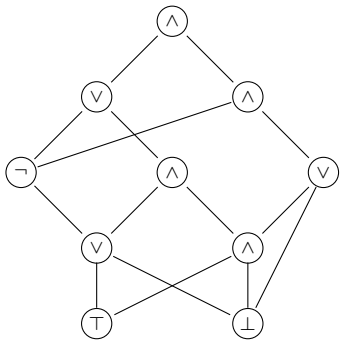


Figure 4.3: The CIRCUIT-VALUE problem

PTIME-complete [GHR95]. It is now rather easy to build a Kripke structure $\mathcal{K}$ and a CTL formula ϕ s.t. $\mathcal{K} \models \phi$ if, and only if, the circuit evaluates to true.

We now prove that CTL model-checking can be achieved in PTIME. We are given a Kripke structure $\mathcal{K} = \langle W, R, l \rangle$, one of its states $w \in W$, and a CTL formula ϕ. The principle of the algorithm is to recursively label each state of $\mathcal{K}$ with the subformulas that hold at that state. For this proof, we use the modal definition of CTL, as presented at Exercise 4.27. Moreover, it is easy to prove that the whole expressive power of CTL can be obtained from only modalities **EX**, **E U** and **AF**. We thus only describe our labeling algorithm for those modalities, plus atomic propositions and boolean operators:

- for atomic propositions: a state $q \in W$ is labeled with an atomic proposition p if, and only if, $p \in l(q)$;
- if $\phi_1 \vee \phi_2$ is a subformula of ϕ, assuming the labeling procedure has already been applied to subformulas ϕ_1 and ϕ_2, we label each state $q \in W$ with $\phi_1 \vee \phi_2$ if, and only if, it is already labeled with ϕ_1 or with ϕ_2;
- the algorithm is similar for conjunction and negation of subformulas;

- if $\mathbf{EX}\,\phi_1$ is a subformula of ϕ, a state $q \in W$ is labeled with $\mathbf{EX}\,\phi_1$ if, and only if, one of its successors by R is labeled with ϕ_1;
- if $\mathbf{E}(\phi_1\,\mathbf{U}\,\phi_2)$ is a subformula, we begin with labeling each state satisfying ϕ_2 with $\mathbf{E}(\phi_1\,\mathbf{U}\,\phi_2)$; Then, each state labeled with ϕ_1 and having one of its successors by R labeled with $\mathbf{E}(\phi_1\,\mathbf{U}\,\phi_2)$ is in turn labeled with $\mathbf{E}(\phi_1\,\mathbf{U}\,\phi_2)$. The procedure is applied until a fixpoint is reached;
- if $\mathbf{AF}\,\phi_1$ is a subformula, we begin with labeling states satisfying ϕ_1 with $\mathbf{AF}\,\phi_1$. Then, if all the successors of a state have been labeled with $\mathbf{AF}\,\phi_1$, then we also label that state with $\mathbf{AF}\,\phi_1$. This procedure is applied until a fixpoint is reached.

In the end, $\mathcal{K}, w \models \phi$ if, and only if, w has been labeled with ϕ.

That the algorithm terminates (i.e., that a fixpoint is reached in each procedure described above) is rather obvious: the number of labeled states can only increase, and it is bounded with the size of W. For each subformula, it is easy to implement the above procedures so that they execute in time $O(|W|^2)$. The global algorithm is thus in time $O(|\phi| \cdot |W|^2)$.

We now prove that this algorithm is correct. Following the algorithm, the proof is by induction on the structure of ϕ: we prove that each procedure describes above labels exactly the set of states that satisfy the corresponding formula:

- it is obvious for atomic propositions and for boolean combinations;
- for subformula $\mathbf{EX}\,\phi_1$: a state is labeled with this subformula if, and only if, it has an immediate successor satisfying ϕ_1, i.e., if, and only if, it satisfies $\mathbf{EX}\,\phi_1$;
- for subformula $\mathbf{E}(\phi_1\,\mathbf{U}\,\phi_2)$: a state is labeled with this formula either because it is labeled with ϕ_2, or because it is labeled with ϕ_1 and has a successor labeled with $\mathbf{E}(\phi_1\,\mathbf{U}\,\phi_2)$. By an obvious induction, this entails that it satisfies $\mathbf{E}(\phi_1\,\mathbf{U}\,\phi_2)$. Conversely, if a state q satisfies $\mathbf{E}(\phi_1\,\mathbf{U}\,\phi_2)$, then there is a finite path from q to a state satisfying ϕ_2 with intermediary states labeled with ϕ_1. It is then obvious that all the states of that path will eventually be labeled with $\mathbf{E}(\phi_1\,\mathbf{U}\,\phi_2)$;
- for $\mathbf{AF}\,\phi_1$: again, a state can be labeled with $\mathbf{AF}\,\phi_1$ either because it is labeled with ϕ_1, or because all of its successors are labeled with $\mathbf{AF}\,\phi_1$. An easy inductive argument proves that it then satisfies $\mathbf{AF}\,\phi_1$. Conversely, if a state q satisfies $\mathbf{AF}\,\phi_1$, then there exists a finite "distance" within which all paths issued from this state will have encountered ϕ_1 (because the Kripke structure is finite). Again, by induction on this distance, we prove that q will be labeled with $\mathbf{AF}\,\phi_1$.

□

Exercice 4.31. ⋆ Write the reduction of CIRCUIT-VALUE to CTL model-checking.

Exercice 4.32. ⋆ Using the programming language of your choice, write the complete labeling algorithms for CTL model-checking.

Exercice 4.33. ⋆ Our algorithm only involves modalities **EX**, **EU** and **AF**, as we explained that any CTL formula can be written with only those modalities. Prove this fact. From a formula $\phi \in \mathsf{CTL}$, prove that the size of the resulting formula can be exponential in the size of the original formula. Does our algorithm suffer from this exponential blowup?

Exercice 4.34. ⋆⋆ Prove that, unless NP = PTIME, CTL^+ cannot be model-checked in polynomial time.

Exercice 4.35. ⋆⋆ Prove that ECTL model-checking is PTIME-complete.

Since CTL^* contains LTL, we know that CTL^* model-checking is PSPACE-hard. It turns out that this lower bound is optimal:

Theorem 72 ([EH86]) *CTL^* model-checking is PSPACE-complete.*

Proof. The algorithm again consists in labeling states with the subformulas they satisfy. Given a formula $\mathbf{A}\phi$, we transform ϕ into a pure LTL formula be replacing its state subformulas with new, fresh atomic propositions, assuming (by induction) that the Kripke structure has already been labeled with those atomic propositions. It then suffices to apply the standard LTL model-checking algorithm, for each state of the Kripke structure, on order to find all the states that satifsy $\mathbf{A}\phi$.

Subformulas $\mathbf{E}\phi$ are handled as $\neg\mathbf{A}\neg\phi$. From the complexity of LTL model-checking, the time-complexity of the algorithm is $O(|\phi|^3 \cdot |\mathcal{K}|^2 \cdot 2^{2|\phi|})$, and space-complexity $O(|K| \cdot |\phi|^2)$. □

Remark. *As for LTL (see the remark following Theorem 54), the time-complexity is only exponential in the formula, and remains polynomial in the size of the Kripke structure. Still, the space needed for this algorithm is polynomial also in the size of the Kripke structure. In fact, compared to LTL and CTL, CTL^* is often considered to be too complex and too expressive. The logic $ECTL^+$ is often considered to be the rigth extension of CTL in terms of its expressiveness. The model-checking problem for this logic is Δ_2^P-complete (i.e., higher than NP and coNP, and below PSPACE) [LMS01].*

Using a similar argument and the result of Theorem 58, we get an upper bound for CTL^+ model-checking:

Theorem 73 *Model-checking CTL^+ is in Δ_2^P.*

The class Δ_2^{P}, also written $\mathsf{PTIME}^{\mathsf{NP}}$, is the class of problem solvable in deterministic polynomial time with an access to an NP oracle (i.e., a device that answers in one step to instances of an NP-complete problem). This class

contains NP and co-NP, and is contained in PSPACE. It is not known whether those inclusions are strict.

We now prove that this algorithm is optimal:

Theorem 74 *Model-checking CTL$^+$ is Δ_2^P-hard.*

Proof. We first have to find a starting point, i.e., a Δ_2^{P}-complete problem. We will use the following one:

SNSAT:

Input: p families of variables $Z_r = \{z_r^1, ..., z_r^m\}$, p variables x_r, p boolean formulae ϕ_r in 3-CNF, with ϕ_r involving variables in $Z_r \cup \{x_1, ..., x_{r-1}\}$.

Output: The value of z_p, defined by

$$\begin{cases} x_1 \stackrel{\mathsf{def}}{=} \exists Z_1.\ \phi_1(Z_1) \\ x_2 \stackrel{\mathsf{def}}{=} \exists Z_2.\ \phi_2(z_1, Z_2) \\ x_3 \stackrel{\mathsf{def}}{=} \exists Z_3.\ \phi_3(z_1, , z_2, Z_3) \\ \quad\quad \dots \\ x_p \stackrel{\mathsf{def}}{=} \exists Z_p.\ \phi_k(z_1, ..., z_{p-1}, Z_p) \end{cases}$$

This problem is obviously in Δ_2^{P}, and turns out to be complete for that class [LMS01, CGS02]. We prove that it can be encoded as a CTL$^+$ model-checking problem.

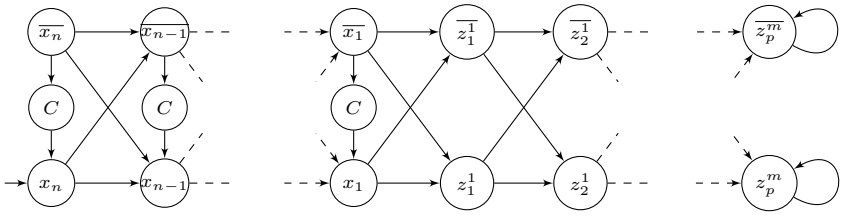


Figure 4.4: The Kripke structure used for encoding SNSAT

Let $\mathcal{I}$ be an instance of the SNSAT problem. This instance defines a unique valuation $v_{\mathcal{I}}$ of variables $x_1, ..., x_r$.

The Kripke structure used in this reduction is depicted on Figure 4.4. As is usual in this kind of reduction, a valuation of the variables will correspond to a trajectory in that structure (visiting $\overline{v}$ if, and only if, variable v is false). Nodes C will be used for ensuring that, when a variable x_i is set to false, then it is really the case that there is no valuation that fulfills ϕ_i.

We now recursively define a sequence of formulas as follows:

$$\psi_0 = \top$$
$$\psi_k = \mathbf{E}\left[\begin{array}{c} \mathbf{G}\,[(\overline{x_1} \vee \ldots \vee \overline{x_p}) \Rightarrow \mathbf{EX}\,(C \wedge \mathbf{EX}\,\neg\psi_{k-1})] \\ \wedge\ \mathbf{G}\,\neg C \wedge \bigwedge_{i=1}^{p}[(\mathbf{F}\,x_i) \Rightarrow \phi_i^{\mathbf{F}}\,(x_1, ..., x_{i-1}, Z_i)] \end{array}\right]$$

where $\phi_i^{\mathbf{F}}\,(x_1, ..., x_{i-1}, Z_i)$ is the formula ϕ_i in which each variable $v \in \{x_1, ..., x_{i-1}\} \cup Z_i$ is replaced with $\mathbf{F}\,v$.

Roughly, formula ψ_k selects a trajectory that nevers goes in a C-state, thus defining a valuation for variables in $x_1, ..., x_n$, and ensures that the valuation of the x_i's are correct. Formally:

Lemma 75 *For any $k \in \mathbb{Z}^+$ and $r \leq p$,*

(1) *if $k \geq 2r-1$, then* $v_{\mathcal{I}}(x_r) = \top$ *if, and only if,* $x_r \models \psi_k$;

(2) *if $k \geq 2r$, then* $v_{\mathcal{I}}(x_r) = \bot$ *if, and only if,* $\overline{x_r} \models \psi_k$.

Proof. The proof is by induction on k. The case when $k = 0$ holds vacuously. Assume the result holds up to some $k \geq 0$. Let $r \leq p$.

- Let w be a valuation of all the variables of $Z_1 \cup \cdots \cup Z_p \cup \{x_1, ..., x_p\}$ that coincides with $v_{\mathcal{I}}$ on $\{x_1, ..., x_p\}$, and such that $w(x_i) = \top$ iff $\phi_i(w(x_1), ..., w(x_{i-1}), w(Z_i))$.

  To w corresponds a trajectory π_w of the Kripke structure. We consider the suffix of π_w that starts in x_r or $\overline{x_r}$. We prove that if $k \geq 2r-1$ (resp. $k \geq 2r$), then π witnesses the fact that $x_r \models \psi_k$ (resp. $\overline{x_r} \models \psi_k$).

  Clearly, π satisfies $\mathbf{G}\,\neg C$. From the requirement on w, we also have that

$$\pi \models \bigwedge_{i=1}^{p}[(\mathbf{F}\,x_i) \Rightarrow \phi_i(x_1, ..., x_{i-1}, Z_i)].$$

  Now, when π visits a state $\overline{x_i}$, then $v_{\mathcal{I}}(x_i) = \bot$. If $i = r$ (in which case we are proving (2), since otherwise π cannot visit $\overline{x_i}$), then $k \geq 2r \geq 2i$; otherwise, $i < r$, and also $k \geq 2i$. Thus $k - 1 \geq 2i - 1$, and by induction hypothesis, since it is not the case that $v_{\mathcal{I}}(x_i) = \top$, we get that $x_i \not\models \psi_{k-1}$. Thus $x_i \models \mathbf{EX}\,(C \wedge \mathbf{EX}\,\neg\psi_{k-1})$.

  Conversely, assume $k \geq 2r-1$ and $x_r \models \psi_k$ (resp. $k \geq 2r$ and $\overline{x_r} \models \psi_k$). Then there is a trajectory π from x_r (resp. $\overline{x_r}$) witnessing ψ_k, thus satisfying $\mathbf{G}\,\neg C$ and defining a valuation w of variables $Z_1 \cup \cdots Z_p \cup \{x_1, ..., x_r\}$. We prove, by induction on i, that $w(x_i) = v_{\mathcal{I}}(x_i)$ for $i \leq r$.

  When $i = 1$, if $w(x_1) = \top$, then π visits x_1, and thus valuation w is a witness that $\phi_1(Z_1)$ is satisfiable. Conversely, if $w(x_1) = \bot$, then π visits $\overline{x_1}$, and we get that $x_1 \models \neg\psi_{k-1}$. We claim that $k - 1 \geq 1$: indeed, the contrary would mean that $r = 1$ and that π starts in x_1. Thus we can apply the induction hypothesis: that $x_1 \models \neg\psi_{k-1}$ implies that $v_{\mathcal{I}}(x_1) = \bot$.

  Assume $w(x_j) = v_{\mathcal{I}}(x_j)$ for any $j < i$.

– if $w(x_i) = \top$, then we know that $\phi_i(w(x_1), ..., w(x_{i-1}), Z_i)$ is true. Since $w(x_j) = v_{\mathcal{I}}(x_j)$, it follows that $\phi_i(v_\rangle(x_1), ..., v_{\mathcal{I}}(x_{i-1}), Z_i)$ is satisfiable, and that $v_{\mathcal{I}}(x_i) = \top$.

– conversely, if $w(x_i) = \bot$, then $\overline{x_i} \models \mathbf{EX}\,(C \wedge \mathbf{EX}\,\neg\psi_{k-1})$, and $x_i \not\models \psi_{k-1}$. If $i < r$, then $k - 1 \geq 2i - 1$, and the induction hypothesis yields $v_{\mathcal{I}}(x_i) = \bot$. If $i = r$, then we know that π visits $\overline{x_r}$, so that we must be in the case where $k \geq 2r$. Then $k - 1 \geq 2i - 1$, and the induction hypothesis entails that $v_{\mathcal{I}}(x_i) = \bot$. □

As an immediate consequence, we get that $v_{\mathcal{I}}(x_n) = \top$ if, and only if, $x_n \models \psi_{2n-1}$, which concludes the proof.

Corollary 76 *Model-checking CTL$^+$ is Δ_2^P-complete.*

4.5 CTL and alternating tree automata

4.5.1 Alternating tree automata

In the case of LTL and LTL+Past, satisfiability and model-checking are handled very similarly (by the automata-theoretic approach). In the case of branching-time logics, this is not the case: satisfiability is much harder. Still, satisfiability can be checked very nicely with automata theory. This requires some new stuff to be introduced.

Definition 77 *A* tree automaton *is a tuple $\mathcal{A} = \langle Q, Q_0, \Sigma, \delta\rangle$ where*

- *Q is a finite set of states;*
- *$Q_0 \subseteq Q$ is the set of initial states;*
- *Σ is a finite alphabet;*
- *$\delta \subseteq Q \times \sigma \times \{\Box, \Diamond\} \times Q$ is the transition relation.*

A tree automaton takes a tree as input:

Definition 78 *Let $\mathcal{A} = \langle Q, Q_0, \Sigma, \Delta\rangle$ be a tree automaton, and $\mathcal{T} = \langle T, \leq, \ell\rangle$ be a tree-structure (with a single root t_0) with $\ell : T \to \Sigma$. An execution of $\mathcal{A}$ on the input tree $\mathcal{T}$ is a labeled tree $\mathcal{T}_{\mathcal{A}} = \langle T, \leq, \ell'\rangle$ sith $\ell' : T \to Q$ s.t.:*

- *$\ell'(t_0) \in Q_0$;*
- *for any $t \in \mathcal{T}$, there exists $(m, q) \in \{\Box, \Diamond\} \times Q$ s.t. :*
  - *$(\ell'(t), \ell(t), m, q) \in \delta$;*
  - *if $m = \Diamond$, then there exists a successor node t' of t s.t. $\ell'(t') = q$;*
  - *if $m = \Box$, then for any immediate successor t' of t, $\ell(t') = q$.*

The acceptance condition is again given in terms of the states that repeat infinitely often along branches:

Definition 79 *A Büchi tree automaton is a tuple* $\mathcal{A} = \langle Q, Q_0, \Sigma, \delta, F\rangle$ *where*

- $\langle Q, Q_0, \Sigma, \delta\rangle$ *is a tree automaton;*
- $F \subseteq Q$ *is a set of accepting states.*

Definition 80 *A single-root tree* $\mathcal{T} = \langle T, \leq, \ell\rangle$ *is accepted by a Büchi tree automaton* $\mathcal{A} = \langle Q, Q_0, \Sigma, \delta, F\rangle$ *is there exists an execution* $\mathcal{T}_\mathcal{A} = \langle T, \leq, \ell'\rangle$ *of* $\mathcal{A}$ *over* $\mathcal{T}$ *s.t., for any branch* b *of* $\mathcal{T}_\mathcal{A}$*,*

$$\mathit{Inf}(b) \cap F \neq \varnothing$$

where $\mathit{Inf}(b) = \{q \in Q \mid \forall t \in b.\ \exists u \in b.\ t \leq u \wedge \ell'(t) = q\}$.

Example. *Let* $\mathcal{A} = \langle \{q_a, q_b\}, \{q_a\}, \{a, b\}, \delta, \{q_a\}\rangle$ *be the automaton with the following transition relation:*

$$\delta = \{(q_a, a, \square, q_a), (q_a, b, \square, q_b), (q_b, a, \square, q_a), (q_b, b, \square, q_b)\}.$$

Figure 4.5 displays an example of an input tree, and an execution tree. State q_a *is*

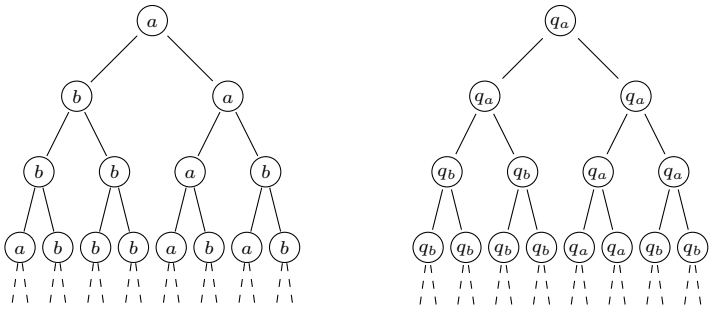


Figure 4.5: Execution of a tree automaton

entered each time an a *is read. Since that state must appear infinitely often, the intuition is that* $\mathcal{A}$ *accepts exactly those trees whose all branches contain infinitely many* a*'s. It can easily proved that this intuition is correct: this automaton accepts the set of tree structures satisfying formula* $\mathbf{A}\overset{\infty}{\mathbf{F}}\, a$.

Exercice 4.36. ⋆⋆ Let $\mathcal{A} = \langle \{q_0, q_1, q_2\}, \{q_0\}, \{a, b\}, \delta, \{q_0, q_1\}\rangle$ be the automaton with the following transition relation:

$$\delta = \{(q_0, a, \Diamond, q_1), (q_0, b, \Diamond, q_1), (q_1, a, \Diamond, q_1), (q_1, b, \Diamond, q_2),$$
$$(q_2, a, \Diamond, q_2), (q_2, b, \Diamond, q_2)\}.$$

Prove that this automaton accepts exactly the set of trees satisfying a CTL formula.

We now extend non-deterministic automata (on both words or trees) to alternating automata. The intuition is as follows: compared to a deterministic automaton, a non-deterministic one may propose several possible transitions at each step, and one of them has to be fired. This is a disjunctive choice. Symmetrically, we could imgaine an automaton in which several of the possible transitions has to be fired: that is, several executions would be launched from from one single state. Alternating automata extend non-deterministic automata with such "conjunctive choices".

Definition 81 *A* positive boolean formula *over a set AP of atomic propositions is a formula built on the ofllowing grammar:*

$$PBF(AP) \ni \phi ::= \top \mid \bot \mid p \mid \phi \vee \phi \mid \phi \wedge \phi$$

where p ranges over AP.

That a valuation of AP satisfies such a formula is defined in the obvious way. Given a subset $T \subseteq \mathsf{AP}$, and $\phi \in \mathsf{PBF}(\mathsf{AP})$, we will use the convenient notation $T \models \phi$ to mean that $v_T \models \phi$ where $v_T(t) = \top$ if, and only if, $t \in T$, and $v(t) = \bot$ otherwise.

In an alternating automaton, the transition function maps each pair (q, a) to a positive boolean formula on the states (roughly). For instance, if $\delta(q, a) = q_1 \wedge (q_2 \vee q_3)$, then one execution must continue in state q_1, and another one either in state q_2 or in state q_3. Of course, non-deterministic automata are a special case of alternating automata.

We now formalize this notion for tree automata:

Definition 82 *An* alternating tree automaton *is a tuple $\mathcal{A} = \langle Q, \delta_0, \Sigma, \delta \rangle$ where*

- *Q is a finite set of states;*
- *$\delta_0 \in PBF(\{\Diamond, \Box\} \times Q)$ is the initial condition;*
- *Σ is a finite alphabet;*
- *$\delta \colon Q \times \Sigma \to PBF(\{\Diamond, \Box\} \times Q)$ is the transition function.*

Definition 83 *Let $\mathcal{A} = \langle Q, Q_0, \Sigma, \delta \rangle$ be an alternating tree automaton, and $\mathcal{T} = \langle T, \leq, \ell \rangle$ be a single-root tree. An execution of $\mathcal{A}$ on input $\mathcal{T}$ is a tree $\mathcal{T}_{\mathcal{A}} = \langle T', \preccurlyeq, \ell' \rangle$ where*

- *$\ell' \colon T' \to T \times Q$ associates, with each node of T', a corresponding node in the input tree and a state of the automaton. We write ℓ'_T and ℓ'_Q for the first and second component of ℓ';*
- *the set of roots of T' satisfy the initial condition (in a sense that will be made clear later);*

- *for any state $t' \in T'$, the set of successors of t' satisfy the transition formula $\delta(\ell'_Q(t'), \ell(\ell'_T(t')))$, in a sense that is made clear below.*

We now explain what it means for a formula in $\mathsf{PBF}(\{\Diamond, \Box\} \times Q)$ to be satisfied. Let t' be a node of $\mathcal{T}_\mathcal{A}$, $t = \ell_T(t')$ be the corresponding node in T, and S be the set of successors of t in $\mathcal{T}$. Let S' be the set of successors of t' in $\mathcal{T}_\mathcal{A}$. Then:

- *$S' \models (\Diamond, q)$ if, and only if, S' contains a node labeled with (u, q) where $u \in S$;*
- *$S' \models (\Box, q)$ if, and only if, for each $u \in S$, S' contains a node labeled with (u, q).*

Boolean combinations keep their usual meaning. For the initial condition, the rules are the same, but S is made of only the root of the input tree.

Now, the acceptance condition is a Büchi condition:

Definition 84 *An* alternating Büchi tree automaton *is a tuple $\mathcal{A} = \langle Q, \delta_0, \Sigma, \delta, F\rangle$ where $\langle Q, \delta_0, \Sigma, \delta\rangle$ is an alternating tree automaton and $F \subseteq Q$ is the set of accepting states. An input tree $\mathcal{T}$ is accepted if there exists an execution tree $\mathcal{T}_\mathcal{A}$ s.t. any branch runs infinitely often in at least one state of F.*

A special subclass of alternating automata will be useful here:

Definition 85 *An alternating automaton $\mathcal{A} = \langle Q, \delta_0, \Sigma, \delta\rangle$ is* linear-weak *if there exists an order $\sqsubseteq$ on Q s.t., for any $\sigma \in \Sigma$, for any $q \in Q$, for any q' that occurs in $\delta(q, \sigma)$, $q' \sqsubseteq q$.*

4.5.2 From CTL to alternating Büchi tree automata

We can now state the important result linking CTL and linear-weak alternating Büchi tree automata:

Theorem 86 ([KVW00, Wil99]) *Given a CTL formula ϕ, one can construct, in linear time, a linear-weak alternating Büchi tree automaton $\mathcal{A}_\phi$ s.t. a single-root tree $\mathcal{T}$ is accepted by $\mathcal{A}_\phi$ if, and only if, it satisfies ϕ.*

Proof. The set of states of the automaton is the set of subformulas of ϕ that are not conjunctions or disjunctions of subformulas, and their negations. The initial condition is $\delta_0 = \phi$: it only requires to begin from states corresponding to ϕ (note that possibly no state corresponds to ϕ, if it is a boolean combination of subformulas). The alphabet is 2^{AP}.

The transition function is defined as follows:

$$\begin{aligned}
\delta(p,\sigma) &= \top \qquad \text{if, and only if, } p \in \sigma \\
\delta(p,\sigma) &= \bot \qquad \text{if, and only if, } p \notin \sigma \\
\delta(\neg p,\sigma) &= \bot \qquad \text{if, and only if, } p \in \sigma \\
\delta(\neg p,\sigma) &= \top \qquad \text{if, and only if, } p \notin \sigma \\
\delta(\,\mathbf{EX}\,\phi_1,\sigma) &= (\Diamond\,,\phi_1) \\
\delta(\,\mathbf{AX}\,\phi_1,\sigma) &= (\Box\,,\phi_1) \\
\delta(\,\mathbf{E}(\phi_1\,\mathbf{U}\,\phi_2),\sigma) &= \delta(\phi_2,\sigma) \vee (\delta(\phi_1,\sigma) \wedge (\Diamond\,,\,\mathbf{E}(\phi_1\,\mathbf{U}\,\phi_2)) \\
\delta(\,\mathbf{A}(\phi_1\,\mathbf{U}\,\phi_2),\sigma) &= \delta(\phi_2,\sigma) \vee (\delta(\phi_1,\sigma) \wedge (\Box\,,\,\mathbf{A}(\phi_1\,\mathbf{U}\,\phi_2)) \\
\delta(\neg\,\mathbf{E}(\phi_1\,\mathbf{U}\,\phi_2),\sigma) &= \delta(\neg\phi_2,\sigma) \wedge (\delta(\neg\phi_1,\sigma) \vee (\Box\,,\neg\,\mathbf{E}(\phi_1\,\mathbf{U}\,\phi_2)) \\
\delta(\neg\,\mathbf{A}(\phi_1\,\mathbf{U}\,\phi_2),\sigma) &= \delta(\neg\phi_2,\sigma) \wedge (\delta(\neg\phi_1,\sigma) \vee (\Diamond\,,\neg\,\mathbf{A}(\phi_1\,\mathbf{U}\,\phi_2))
\end{aligned}$$

It is clear enough that this automaton is linear-weak (the order being given by the "subformula" relation). Last, a state is in the accepting set if, and only if, it corresponds to a formula of the form $\neg\,\mathbf{E}(\phi_1\,\mathbf{U}\,\phi_2)$ or $\neg\,\mathbf{A}(\phi_1\,\mathbf{U}\,\phi_2)$.

We now prove that this construction is correct, i.e., that a tree is accepted by this automaton if, and only if, it satisfies ϕ. To that aim, for each subformula ψ of ϕ, we define the automaton $\mathcal{A}_\psi$ to be the same automaton as $\mathcal{A}_\phi$, but with formula ψ as initial condition. We prove that a tree $\mathcal{T} = \langle T, \leq, \ell\rangle$ is accepted by $\mathcal{A}_\psi$ if, and only if, $\langle \mathcal{T}, 0\rangle \models \psi$.

The proof is by induction on the structure of the formula:

- for atomic propositions and negations thereof, this is immediate;
- if $\psi = \psi_1 \vee \psi_2$ of $\psi = \psi_1 \wedge \psi_2$, the result follows rather easily from the induction hypothesis;
- if $\psi = \mathbf{EX}\,\psi_1$: if $\langle \mathcal{T}, 0\rangle \models \mathbf{EX}\,\psi_1$, then one successor of node 0 satisfies ψ_1, and, by induction hypothesis, the corresponding subtree is accepted by $\mathcal{A}_{\psi_1}$. It then suffices to add an extra node, labeled with $(0,\psi)$, at the root of an accepting execution tree in order to get an accepting execution tree for $\mathcal{A}_\psi$ on input $\mathcal{T}$.

  Conversely, if $\mathcal{T}$ is accepted by $\mathcal{A}_\psi$, then an accepting execution tree will start with a node labeled with $(0,\psi)$, having one single son labeled with (t,ψ_1), where t is a successor of 0. By induction, the subtree of $\mathcal{T}$ rooted at t satisfies ψ_1, which entails that $\langle \mathcal{T}, 0\rangle \models \mathbf{EX}\,\psi_1$.
- the arguments are similar for $\mathbf{AX}\,\psi_1$;
- if $\psi = \mathbf{E}(\psi_1\,\mathbf{U}\,\psi_2)$: first assume that $\langle \mathcal{T}, 0\rangle \models \mathbf{E}(\psi_1\,\mathbf{U}\,\psi_2)$. Then there exists a node t s.t. $\langle \mathcal{T}, t\rangle \models \psi_2$, while each intermediate node u_i (numbered from 0 to $n-1$ with $u_0 = 0$ and t being a successor of u_{n-1}) satisfy $\langle \mathcal{T}, u_i\rangle \models \psi_1$. By induction, there exist accepting execution trees for the corresponding subtrees in the corresponding automata. From those

trees, we build a new one as follows: from the root of the execution tree for u_0, add an extra edge to the root of the execution tree of $\mathcal{A}_{\psi_1}$ for u_1; from this node, add an extra edge to the root of the execution tree for u_2, and so on. Last, from the root of the execution tree for u_{n-1}, add an edge to to execution tree of $\mathcal{A}_{\psi_2}$ on the subtree rooted at t. We then set the labeling as follows: nodes labeled with (u_i, ψ_1) are now labeled with (u_i, ψ). It is then easy to check that this is an execution tree of $\mathcal{A}_\psi$ on input tree $\mathcal{T}$, and that it is accepting.

Conversely, assume that $\mathcal{T}$ is accepted by $\mathcal{A}_\psi$, and consider an accepting execution tree. There can't be an infinite branch whose nodes are labeled with ψ as second item, since this would contradict acceptance. Thus, there exists a finite sequence of nodes $t'_0, ..., t'_n$, with $t'_0 = 0$ and t'_{i+1} being a successor of t'_i, and labeled with (t_i, ψ) where t_{i+1} is a successor of t_i in $\mathcal{T}$. Since the transition condition must be fulfilled, it must be the case at each t'_i, $i < n$, has successors satisfying $\delta(\psi_1, \ell(t_i))$. From each corresponding subtrees, we can easily build execution trees showing that the subtree of $\mathcal{T}$ rooted at t_i is accepted by $\mathcal{A}_{\psi_1}$ and, by induction hypothesis, that $\langle \mathcal{T}, t_i \rangle \models \psi_1$. Similarly, t'_n has a set of successors satisfying $\delta(\psi_2, \ell(t_n))$, from which we can prove that $\langle \mathcal{T}, t_n \rangle \models \psi_2$. This entails that $\langle \mathcal{T}, 0 \rangle \models \psi$.

- the arguments for the other cases are very similar, and left to the intrepid reader. □

Exercice 4.37. ⋆⋆⋆ Given an alternating Büchi tree automaton $\mathcal{A}$, prove that there exists a Büchi tree automaton $\mathcal{A}'$ that accepts exactly the same set of trees, and has size at most exponential in the size of $\mathcal{A}$.

4.5.3 Application to verification

Since checking the emptiness of the language accepted by an alternating Büchi (tree) automaton is in EXPTIME, we get the following result:

Theorem 87 *CTL satisfiability is in EXPTIME.*

Exercice 4.38. ⋆⋆⋆ By encoding the behavior of a linear-space alternating Turing machine (see [Pap94]), prove that CTL satisfiability is EXPTIME-hard.

Remark. *Note that the automaton can also be used for model-checking, and yields a polynomial-time algorithm [KVW00].*

Remark. *Infinite words are a special case of infinite trees, and CTL, when interpreted on infinite words, is obviously equivalent to LTL. It follows that the construction above can be adapted to the linear-time case. A direct construction is given in [Var96]. In this case, the converse translation is also possible: linear-weak alternating Büchi automata are exactly as expressive as LTL.*

4.6 Alternating-time temporal logics

Alternating-time temporal logics are special kinds of branching-time logics that are especially powerful in order to reason about "game structures", in which several agents concurrently act upon the system. This approach is very interesting for modeling systems that are embedded in an environment (assumed to be hostile), and that we want to control in order to enforce some properties. We begin with some definitions of all those notions.

4.6.1 Game structures

Definition 88 ([AHK02]) *A* Concurrent Game Structure *(*CGS *for short) is a* 6*-tuple* $\mathcal{C} = \langle A, Q, \mathsf{AP}, \ell, c, \delta \rangle$ *s.t:*

- $A = \{A_1, ..., A_k\}$ *is a finite set of* agents *(or* players*);*
- Q *is a finite set of* locations*;*
- AP *is a finite set of* atomic propositions*, resp.;*
- $\ell \colon Q \to 2^{\mathsf{AP}}$ *is a function labeling each location by the set of atomic propositions that hold for that location;*
- $c \colon Q \times A \to \mathcal{P}(\mathbb{Z}^+) \setminus \{\varnothing\}$ *defines the (finite) set of possible moves of each agent in each location.*
- $\delta \colon Q \times {\mathbb{Z}^+}^k \to Q$*, where* $k = |A|$*, is a (partial) function defining the transition table. With each location and each set of moves of the agents, it associates the resulting location.*

The intuitive semantics is as follows: in a given state q, each agent A_i has several possible moves, represented as integers given by $c(q, A_i)$. Once all the agents have chosen a move, the transition table returns the successor state, where the execution can continue.

Definition 89 *Let* $\mathcal{C} = \langle A, Q, \mathsf{AP}, \ell, c, \delta \rangle$ *be a CGS, and* $q \in Q$*. The set of successors of* q*, denoted with* $\mathsf{succ}(q)$*, is defined as*

$$\mathsf{succ}(q) = \{q' \in Q \mid \exists a_1, ..., a_k \in \mathbb{Z}^+ \text{ s.t. } q' = \delta(q, a_1, ..., a_k) \text{ and for all } i \leq k,\ a_i \in c(q, A_i)\}.$$

Definition 90 *Let* $\mathcal{C} = \langle A, Q, \mathsf{AP}, \ell, c, \delta \rangle$ *be a CGS. A* computation *of* $\mathcal{C}$ *is an infinite sequence* $\rho = (q_i)_{i \in \mathbb{Z}^+}$ *s.t. for any* i*,* $q_{i+1} \in \mathsf{succ}(q_i)$*.*

Definition 91 *A* strategy *for player* $A_i \in A$ *is a function* $s_i \colon Q^* \to \mathbb{Z}^+$ *that maps each finite prefix of a computation to a move, with the requirement that, for any finite sequence* $(q_0, ..., q_n)$*,* $s_i(q_0, ..., q_n) \in c(q_n, A_i)$*.*

If $B \subseteq A$ *is a set of agents (also called a* coalition*), then a strategy for coalition* B *is a set of strategies, one for each agent in* B*.*

An important class of stratgies consists in those strategies that only depend on the current state, and not on the whole history:

Definition 92 *A strategy s is* memoryless *is, for any two finite sequences $(q_0, ..., q_n)$ and $(q'_0, ..., q'_m)$, if $q_n = q'_m$, then $s(q_0, ..., q_n) = s(q'_0, ..., s'_m)$.*

We should now define what it means to play according to a strategy:

Definition 93 *Let $\mathcal{C} = \langle A, Q, \mathsf{AP}, \ell, c, \delta \rangle$ be a CGS, $B \subseteq A$ be a coalition of agents, s_B be a strategy for this coalition, and $(q_0, ..., q_n)$ be a finite prefix of a computation. We write $\mathsf{succ}((q_0, ..., q_n), s_b)$ for the set of possible successors under the strategy s_B, defined formally as*

$$\begin{aligned}\mathsf{succ}((q_0, ..., q_n), s_B) = \{q' \in \mathsf{succ}(q_n) \mid \exists a_1, ..., a_k \in \mathbb{Z}^+ \text{ s.t.} \\ q' = \delta(q, a_1, ..., a_k) \text{ and} \\ \text{for all } i \leq k,\ a_i \in c(q, A_i) \text{ and} \\ \text{for all } i \leq k.\ (A_i \in B \Rightarrow a_i = s_{A_i}(q_0, ..., q_n))\}.\end{aligned}$$

A computation $\rho = (q'_i)_{i \in \mathbb{Z}^+}$ is an outcome *of the strategy s_B with history $(q_0, ..., q_n)$ if, for any i, we have*

$$q'_{i+1} \in \mathsf{succ}((q_0, ..., q_n, q'_0, ..., q'_i), s_B).$$

We write $\mathsf{Out}((q_0, ..., q_n), s_B)$ for the set of outcomes of s_B with history $(q_0, ..., q_n)$.

Example. *We consider a very simple example, with only two players, depicted on Figure 4.6. In such a representation, edges are labeled with tuples, with the following meaning: an edge (q, q') is labeled with $\langle a_1, a_2 \rangle$ when $\delta(q, a_1, a_2) = q'$. That is, this transition corresponds to move a_1 of player A_1 and to move a_2 of player A_2.*

In the example of Figure 4.6, player A_1 has only one possible move in state p, and two moves in q. Conversely, A_2 has only one move from q, and two from p. This is an example of a turn-based *CGS, where, in each state, at most one player has several moves.*

Example. *Consider the example of Figure 4.7. This example is not turn-based anymore, since both players have two possible moves in the topmost state. From that state, it is clear enough that player A_1 has no strategy to be sure to go to state p: whichever move she plays, the other player simply has to play the opposite in order to avoid state p. Conversely, player A_2 has no strategy to avoid state p: whatever she plays, the opponent simply has to play the same move in order to end up in state p.*

This property is called non-determinedness. *It must be mentionned that many cases of asynchronous games are* determined*: if a player has no strategy to achieve some goal, then the opponent has a strategy to avoid that goal. Synchronous games do not have this property.*

Figure 4.6: A simple CGS

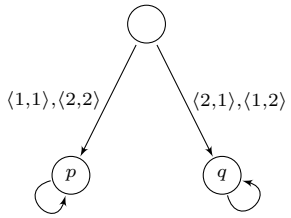


Figure 4.7: Another CGS

Remark. *It can be noticed that our definition of a CGS does not allow for non-determinism: one all the payers have chosen their move, there is exactly one possible next state. However, non-determinism can be modeled by adding one extra player, that would, in some sense, "resolve" non-determinism.*

Definition 94 *Let $n \in \mathbb{Z}^+$. A* (labeled) alternating tree structure *is a tuple $\mathcal{S} = \langle T, \leq, \ell, m\rangle$ s.t.:*

- *$\langle T, \leq, \ell\rangle$ is a tree structure;*
- *m labels each edge (t, t') with a set of n-tuples of integers, with the requirement that, for any $t \in T$, there exist sets of integers $C_1, ..., C_n$ such that the set*

$$\{\langle a_1, ..., a_n\rangle \mid \exists t' \in T.\ m(t, t') = \langle a_1, ..., a_n\rangle\}$$

is equal to the set

$$\{\langle a_1, ..., a_n\rangle \mid \forall i \leq n.\ a_i \in C_i\}.$$

Of course, this definition corresponds to computation trees of CGSs, in the following sense:

Definition 95 *Let $\mathcal{C} = \langle A, Q, \mathsf{AP}, \ell, c, \delta\rangle$ be a CGS. Let $q \in Q$, $B \subseteq A$ be a coalition of agents, and s_B be a strategy for this coalition. The* computation tree *of $\mathcal{C}$ from q under strategy s_B is the alternating tree structure $\langle T, \leq, \ell, m\rangle$ where*

- *T is the set of finite words $(q_0, q_1, ..., q_k)$ on Q s.t. $q_0 = q$ and, for any two consecutive letters q_i and q_{i+1}, $q_{i+1} \in \mathsf{succ}((q_0, ..., q_i), s_B)$;*
- *$\leq$ is the lexicographical order;*
- *$\ell(q_0, ..., q_k) = \ell(q_k)$;*
- *$m(q_i, q_{i+1})$ is the set $\{\langle a_1, ..., a_n\rangle \mid q_{i+1} = \delta(q_i, \langle a_1, ..., a_n\rangle)$ and $a_j = s_{A_j}(q_0, ..., q_i)$ if $A_j \in B\}$.*

Exercice 4.39. ⋆ Prove that the computation tree of a CGS is indeed an alternating tree structure.

Of course, CTL can be evaluated on such a computation tree, but it can also be enriched in order to take the extra labeling of the edges into account. This yields the logic ATL (for *Alternating-time Temporal Logic*):

Definition 96 *Given a set $M_1, ..., M_n$ of first-order definable modalities, the syntax of the alternating-time temporal logic $\mathcal{A}(M_1, ..., M_n)$ is given by:*

$$\mathcal{A}(M_1, ..., M_n) \ni \phi_b ::= p \mid \neg\phi_b \mid \phi_b \vee \phi_b \mid \phi_b \wedge \phi_b \mid \langle\!\langle B \rangle\!\rangle \phi_l \mid [\![B]\!] \phi_l$$
$$\phi_l ::= M_1(\phi_b, ..., \phi_b) \mid \cdots \mid M_n(\phi_b, ..., \phi_b)$$

where p ranges over AP and B ranges over the subsets of A. The logic ATL is then defined as $\mathcal{A}(\mathbf{X}, \mathbf{U})$.

The semantics is as follows:

Definition 97 *Let $\mathcal{S} = \langle T, \leq, \ell, m\rangle$ be an alternating tree structure, $t \in T$, $(t_0, ..., t_{n-1})$ be the (ordered) set of predecessors of t, and ϕ_l be a path formula of ATL. We define the semantics of ATL as an extension of the semantics of CTL* (see Definition 67) with the following two rules:*

$$\langle \mathcal{S}, t\rangle \models_b \langle\!\langle B \rangle\!\rangle \phi_l \quad \Leftrightarrow \quad \exists \text{ strategy } s_B.$$
$$\forall b \in \mathit{Br}(t).\ \text{if } b \in \mathit{Out}((t_0, ..., t_{n-1}), s_B),$$
$$\text{then } \langle \mathcal{S}, b, t\rangle \models_l \phi_l$$
$$\langle \mathcal{S}, t\rangle \models_b [\![B]\!] \phi_l \quad \Leftrightarrow \quad \forall \text{ strategy } s_B.$$
$$\exists b \in \mathit{Br}(t).\ b \in \mathit{Out}((t_0, ..., t_{n-1}), s_B) \text{ and}$$
$$\langle \mathcal{S}, b, t\rangle \not\models_l \phi_l$$

Quantifiers $\langle\!\langle B \rangle\!\rangle$ and $[\![B]\!]$ quantify over strategies: formula $\langle\!\langle B \rangle\!\rangle \phi_l$ is read "there is a strategy for coalition B to enforce ϕ_l", while $[\![B]\!] \phi_l$ is dual, and can be read "coalition B has no strategy to prevent ϕ_l from happening".

It should be remarked that ATL subsumes CTL, because both existential and universal path quantifiers can be expressed in terms of $\langle\!\langle \cdot \rangle\!\rangle$:

$$\mathbf{E}\phi_l \equiv \langle\!\langle A \rangle\!\rangle \phi_l \qquad \mathbf{A}\phi_l \equiv \langle\!\langle \varnothing \rangle\!\rangle \phi_l$$

Remark. *The original definition of ATL, which appeared in [AHK97, AHK02], is not exactly the one described above: it was a "modal" definition, where formulas are built from atomic propositions, boolean combinators, and the following three kinds of modalities: $\langle\!\langle B \rangle\!\rangle \mathbf{X}$, $\langle\!\langle B \rangle\!\rangle \mathbf{U}$ and $\langle\!\langle B \rangle\!\rangle \mathbf{G}$. However, this definition is strictly less expressive than the above one [LMO06].*

Exercice 4.40. ⋆⋆ The "weak until" modality is a relaxed version of the "until" modality, where the eventuality might not occur. It is defined as

$$\phi \,\mathbf{W}\, \psi \stackrel{\text{def}}{\equiv} \phi \,\mathbf{U}\, \psi \vee \mathbf{G}\, \phi.$$

Prove the following equivalences (over discrete time):

$$\begin{aligned}\mathbf{E}\phi\,\mathbf{W}\,\psi &\equiv \mathbf{E}\phi\,\mathbf{U}\,\psi \vee \mathbf{EG}\,\phi\\ \phi\,\mathbf{W}\,\psi &\equiv \neg(\neg\psi\,\mathbf{U}\,(\neg\phi\wedge\neg\psi))\\ \mathbf{E}\phi\,\mathbf{W}\,\psi &\equiv \neg\,\mathbf{A}(\neg\psi\,\mathbf{U}\,(\neg\phi\wedge\neg\psi))\end{aligned}$$

Give counter-examples proving that the following tentative formulas do not express $\langle\!\langle B\rangle\!\rangle\mathbf{W}$:

$$\begin{aligned}\langle\!\langle B\rangle\!\rangle\,\phi\,\mathbf{U}\,\psi \vee \langle\!\langle A\rangle\!\rangle\,\mathbf{G}\,\phi &\not\equiv \langle\!\langle B\rangle\!\rangle\,\phi\,\mathbf{W}\,\psi\\ \neg\,\langle\!\langle A\smallsetminus B\rangle\!\rangle\,(\neg\psi\,\mathbf{U}\,(\neg\phi\wedge\neg\psi)) &\not\equiv \langle\!\langle B\rangle\!\rangle\,\phi\,\mathbf{W}\,\psi\end{aligned}$$

Prove that those formulas are equivalent on turn-based CGSs.

4.6.2 ATL model-checking

This section is devoted to proving the following result:

Theorem 98 *If the number of players is fixed, then ATL model-checking is PTIME-complete.*

Remark. *It is important to insist here on the fact that the number of players must be fixed in order to achieve this complexity. The problem comes from the fact that, if each of the n players have m possible moves, the transition table given by δ has size m^n, but could be encoded is a more succinct way than just the whole explicit table. The above complexity result still holds if the number of players is a parameter of the problem provided that the transition table is given explicitely.*

Proof. Hardness in PTIME follows from that of CTL, and we only have to develop a PTIME algorithm in order to prove our statement. The algorithm is similar to that of CTL: it consists in recursively label the states of the structures with the subformulas that hold true in those states.

Let $\mathcal{C} = \langle A, Q, \mathsf{AP}, \ell, c, \delta\rangle$ be a CGS with a fixed number of players, and $\phi \in$ ATL. For each subformula ψ of ϕ, we label the states of $\mathcal{C}$ according to the following rules (omitting the trivial cases of atomic propositions and boolean combinators, for which the algorithm is similar to that of CTL):

- if $\psi = \langle\!\langle B\rangle\!\rangle\,\mathbf{X}\,\psi_1$: assuming the labeling algorithm has already been applied for ψ_1, we label a state q with ψ if, and only if, there exists a set of moves $a_{i_1}, ..., a_{i_{|B|}}$, one for each player in B, s.t.

$$\{q' \in Q \mid \exists a_{j_1}, ..., a_{j_{|A|-|B|}}.\ q' = \delta(q, a_1, ..., a_n)\} \subseteq \{q' \in Q \mid q' \text{ is labeled with } \psi_1\}.$$

  This can easily be proved in linear-time by reading the transition table.

- if $\psi = [\![B]\!]\,\mathbf{X}\,\phi_1$: we label with ψ those states from which coalition B cannot avoid going in a location labeled with ψ_1. Formally, we label a state q if, for any set of moves $a_{i_1}, ..., a_{i_{|B|}}$, we have

$$\{q' \in Q \mid \exists a_{j_1}, ..., a_{j_{|A|-|B|}}.\ q' = \delta(q, a_1, ..., a_n)\} \cap \{q' \in Q \mid q' \text{ is labeled with } \psi_1\} \neq \varnothing.$$

- if $\psi = \langle\!\langle B \rangle\!\rangle\,(\psi_1\,\mathbf{U}\,\psi_2)$: we recursively label the states of the structure: at the first stage, the states that have been labeled with ψ_2 are labeled with ψ. We then apply the following procedure until a fixpoint is reached: if a state q is labeled with ψ_1, and if there is a set of moves $a_{i_1}, ..., a_{i_{|B|}}$, one for each player in B, s.t.

$$\{q' \in Q \mid \exists a_{j_1}, ..., a_{j_{|A|-|B|}}.\ q' = \delta(q, a_1, ..., a_n)\} \subseteq \{q' \in Q \mid q' \text{ is labeled with } \psi\}$$

  then we label q with ψ.

- last, if $\psi = [\![B]\!]\,(\psi_1\,\mathbf{U}\,\psi_2)$, then we again begin with labeling with ψ all the states that are labeled with ψ_2, and then recursively label with ψ each state that is labeled with ψ_1 and satisfies the following condition:

$$\{q' \in Q \mid \exists a_{j_1}, ..., a_{j_{|A|-|B|}}.\ q' = \delta(q, a_1, ..., a_n)\} \cap \{q' \in Q \mid q' \text{ is labeled with } \psi\} \neq \varnothing.$$

  The procedure stops when a fixpoint is reached.

That the algorithm terminates in polynomial time is rather obvious: it runs in time $O(|\delta| \cdot |\phi|)$. That it is correct can be proved following the same lines as for the algorithm for CTL. □

Exercice 4.41. ⋆⋆ Prove that the algorithm is correct.

Remark. *The algorithm above only computes the set of states in which formula ϕ holds, but, for each subformula of the form $\langle\!\langle B \rangle\!\rangle\,\phi$ and for each state, a winning strategy is easily obtained from the sets of moves that witness the fact that we label this state. This entails in particular that if there exists a strategy for coalition B to enforce ϕ, with ϕ a path formula in ATL, then there exists a memoryless strategy, i.e., a strategy that only depends on the current state, and not on the whole history of the computation.*

Exercice 4.42. ⋆⋆⋆ Prove that memoryless strategies still exist if we extend ATL with the modality $\overset{\infty}{\mathbf{F}}$ (with the semantics derived from that of CTL*).

Prove that ATL$^+$, where we allow boolean combinations of modalities in the scope of strategy quantifiers, can be translated in ATL, and thus also benefits from memoryless strategies.

Prove that if we combine both extensions (modality $\overset{\infty}{\mathbf{F}}$ and boolean combinations of modalities), we lose the existence of a memoryless strategy.

Exercice 4.43. ⋆ It should be noticed that we did not specify the way the transition table should be represented. The naive representation is as a (huge) table of size $O(|Q| \times M^{|A|})$, where M is the maximal number of possible choices af any agent in any location. Still, there are more succinct representations, e.g. involving boolean formulas on the choices of the agents. Prove that, under this encoding, ATL model-checking is NP-hard.

Bibliography

[AHK97] Rajeev Alur, Thomas Henzinger, and Orna Kupferman. Alternating-time temporal logic. In *Proceedings of the 38th Annual Symposium on Foundations of Computer Science (FOCS'97)*, pages 100–109. IEEE Computer Society Press, October 1997.

[AHK02] Rajeev Alur, Thomas Henzinger, and Orna Kupferman. Alternating-time temporal logic. *Journal of the ACM*, 49(5):672–713, 2002.

[BBF+01] Béatrice Bérard, Michel Bidoit, Alain Finkel, François Laroussinie, Antoine Petit, Laure Petrucci, and Philippe Schnoebelen. *Systems and Software Verification. Model-Checking Techniques and Tools*. Springer, 2001.

[BCGR92] Samuel R. Buss, Stephen A. Cook, Arvind Gupta, and Vijaya Ramachandran. An optimal parallel algorithm for formula evaluation. *SIAM Journal on Computing*, 21(4):755–780, 1992.

[BG85] John P. Burgess and Yuri Gurevich. The decision problem for lineasr temporal logic. *Notre Dame Journal of Formal Logic*, 26(2):115–128, 1985.

[Büc60] J. Richard Büchi. Weak second-order arithmetic and finite automata. *Zeitschrift fur Mathematische Logik und Grundlagen der Mathematik*, 6:66–92, 1960.

[CE81] Edmund M. Clarke and E. Allen Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In Dexter Kozen, editor, *Proceedings of the 3rd Workshop Logics of Programs*, volume 131 of *Lecture Notes in Computer Science*, pages 52–71. Springer, 1981.

[CGP99] Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. *Model Checking*. MIT Press, 1999.

[CGS02] Marco Cadoli, Andrea Giovanardi, and Marco Schaerf. An algorithm to evaluate quantified boolean formulae. *Journal of Automated Reasoning*, 28(2):101–142, 2002.

[Coo71] Stephen A. Cook. The complexity of theorem proving procedures. In *Proceedings of the 3rd Annual ACM Symposium on the Theory of Computing (STOC'71)*, pages 151–158. ACM Press, 1971.

[DS02] Stéphane Demri and Philippe Schnoebelen. The complexity of propositional linear temporal logics in simple cases. *Information and Computation*, 174(1):84–103, 2002.

[EH86] E. Allen Emerson and Joseph Halpern. "sometimes" and "not never" revisited: on branching versus linear time temporal logic. *Journal of the ACM*, 33(1):151–178, 1986.

[EVW02] Kousha Etessami, Moshe Y. Vardi, and Thomas Wilke. First-order logic with two variables and unary temporal logic. *Information and Computation*, 179(2):279–295, 2002.

[Fre79] Gottlob Frege. *Begriffsschrift, eine der arithmetischen nachgebildete Formelsprache des reinen Denkens*. Verlag von Louis Nebert, Halle, 1879.

[GHR95] Raymond Greenlaw, H. James Hoover, and Walter L. Ruzzo. *Limits to Parallel Computation*. Oxford University Press, 1995.

[Gor00] Valentin Goranko. Temporal logics of computations. In *Proceedings of the 12th European Summer School in Logic, Language and Information (ESSLLI 2000)*, 2000.

[GPSS80] Dov M. Gabbay, Amir Pnueli, Saharon Shelah, and Jonathan Stavi. On the temporal analysis of fairness. In *Conference Record of the 7th ACM Symposium on Principles of Programming Languages (POPL'80)*, pages 163–173. ACM Press, 1980.

[Kam68] Hans W. Kamp. *Tense Logic and the Theory of Linear Order*. PhD thesis, UCLA, Los Angeles, California, USA, 1968.

[KVW00] Orna Kupferman, Moshe Y. Vardi, and Pierre Wolper. An automata-theoretic approach to branching-time model-checking. *Journal of the ACM*, 47(2):312–360, 2000.

[Lew18] Clarence Irving Lewis. *A Survey of Symbolic Logic*. University of California Press, 1918.

[LMO06] François Laroussinie, Nicolas Markey, and Ghassan Oreiby. Expressiveness and complexity of ATL. Research Report LSV-06-03, Laboratoire Spécification et Vérification, ENS Cachan, France, February 2006. 20 pages.

[LMS01] François Laroussinie, Nicolas Markey, and Philippe Schnoebelen. Model checking CTL^+ and FCTL is hard. In Furio Honsell and

Marino Miculan, editors, *Proceedings of the 4th International Conference on Foundations of Software Science and Computation Structure (FoSSaCS 2001)*, volume 2030 of *Lecture Notes in Computer Science*, pages 318–331. Springer-Verlag, 2001.

[LMS02] François Laroussinie, Nicolas Markey, and Philippe Schnoebelen. Temporal logic with forgettable past. In *Proceedings of the 17th Annual Symposium on Logic in Computer Science (LICS'02)*, pages 383–392. IEEE Computer Society Press, 2002.

[LPZ85] Orna Lichtenstein, Amir Pnueli, and Lenore D. Zuck. The glory of the past. In Rohit Parikh, editor, *Proceedings of the Conference on Logics of Programs*, volume 193 of *Lecture Notes in Computer Science*, pages 413–424. Springer-Verlag, 1985.

[MR99] Faron Moller and Alexander Rabinovich. On the expressive power of CTL*. In *Proceedings of the 14th Annual Symposium on Logic in Computer Science (LICS'99)*, pages 360–369. IEEE Computer Society Press, 1999.

[Pap94] Christos H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

[Pnu77] Amir Pnueli. The temporal logic of programs. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science (FOCS'77)*, pages 46–57. IEEE Comp. Soc. Press, 1977.

[Pri57] Arthur N. Prior. *Time and Modality*. Oxford University Press, 1957.

[Pri67] Arthur N. Prior. *Past, Present, Future*. Oxford University Press, 1967.

[Pri68] Arthur N. Prior. *Papers on Time and Tense*. Oxford University Press, 1968.

[QS82] Jean-Pierre Queille and Joseph Sifakis. Specification and verification of concurrent systems in CESAR. In Mariangiola Dezani-Ciancaglini and Ugo Montanari, editors, *Proceedings of the 5th International Symposium on Programming*, volume 137 of *Lecture Notes in Computer Science*, pages 337–351. Springer, 1982.

[Rab] Michael O. Rabin.

[SC85] A. Prasad Sistla and Edmund M. Clarke. The complexity of propositional linear temporal logics. *Journal of the ACM*, 32(3):733–749, 1985.

[Sha05] Stewart Shapiro. Classical logic. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. 2005. http://plato.stanford.edu/archives/fall2005/entries/logic-classical/.

[Sto74] Larry J. Stockmeyer. *The Complexity of Decision Problems in Automata Theory and Logic*. PhD thesis, Massachusets Institute of Technology, Cambridge, Massasuchets, USA, 1974.

[Var96] Moshe Y. Vardi. An automata-theoretic approach to linear temporal logic. In Faron Moller and Graham M. Birtwistle, editors, *Logics for Concurrency - Structure versus Automata: Proceedings 8th Banff Higher Order Workshop (Banff'94)*, volume 1043 of *Lecture Notes in Computer Science*, pages 238–266. Springer-Verlag, 1996.

[Wik06] Wikipedia. Twin prime conjecture — Wikipedia, the free encyclopedia, 2006. http://en.wikipedia.org/wiki/Twin_prime_conjecture.

[Wil99] Thomas Wilke. CTL$^+$ is exponentially more succinct than CTL. In C. Pandu Rangan, Vijayshankar Raman, and R. Ramanujam, editors, *Proceedings of the 19th Conferentce on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'99)*, volume 1738 of *Lecture Notes in Computer Science*, pages 110–121. Springer-Verlag, 1999.

[Wol83] Pierre Wolper. Temporal logic can be more expressive. *Information and Control*, 56(1-2):72–99, 1983.

[WVS83] Pierre Wolper, Moshe Y. Vardi, and A. Prasad Sistla. Reasoning about infinite computation paths. In *Proceedings of the 24th Annual Symposium on Foundations of Computer Science (FOCS'83)*, pages 185–194. IEEE Comp. Soc. Press, 1983.

Index