# Temporal logics
# Course 2-8
# MPRI 2005/06

October 29, 2005

# Contents

# Chapter 1

# Introduction to formal languages for specifications

**Formal specification languages** This course is dedicated to temporal logics for formal specification and verification of programs. This means to specify by means of temporal formulae the properties which a program must satisfy, and given a program to verify formally that the specification are met. Unlike natural languages, formal languages have a non ambiguous semantics and are helpful to make proofs. Specifications can describe the good behaviors of a system as well as the bad ones. In this chapter, we briefly present some models of time and primitive temporal operators used in this course.

A great part of this chapter is freely inspired from [Gor00, BBF+01]. We invite the interested reader to consult the original documents for much more details and developments.

## 1.1 Reasoning about time: introductory remarks

This section is strongly inspired from [Gor00, Sect. 1.1].

**Temporal logic in computer science** Logical discussions of time and temporality goes back to ancient Greece. A modern approach using modalities (as in temporal logic) is due to A. Prior [Pri67]. A modality is a syntactic object (a term) that modifies the relationships between a predicate and a subject. Here are examples of modalities:

- It is necessary that P holds,
- I know that P holds,
- Tomorrow P holds,
- It is an obligation that P holds,

Temporal logic in computer science has it beginning with A. Pnueli's seminal paper [Pnu77] arguing that temporal logic can be used to reason about processes and computations. This is one of the most successful applications of logic to computer science.

**Nature of time** A fundamental question is about the ontological nature and structure of the flow of time. Numerous distinct alternatives exist (on-going debate since antiquity) and the answers to these alternatives are important for the development of the philosophy of nature. However, the different aspects of

temporality in computer science lead to its own choices as will be presented in the forthcoming developments. Here are some alternatives about the nature of time: is time

- linear or branching?
- instant-based or interval-based?
- discrete or continuous?
- infinite? transfinite?
- circular?
- beginningless? endless? ...

**Two traditional approaches** In order to reason about time in a formal language we need to make assumptions about

- the primitive time entities and,
- the basic relations between them.

The two traditional types of models of time are point-based and interval-based. The traditional logical language is first-order logic that can be viewed as the universal logic to be used in all fields of application. By contrast, temporal logics are adapted to models of computations and to specific applications.

**Point-based approach** In the point-based models of time the primitive temporal entities are instants and the basic relationships between them are equality and precedence. A flow of time is represented by $\langle T, < \rangle$ (equality is implicit). Here are some first-order sentences expressing standard properties about the flow of time:

**irreflexivity:** $\forall\ x\ \neg(x < x)$,

**transitivity:** $\forall\ x, y, z\ (x < y \wedge y < z) \Rightarrow x < z$,

**linearity:** $\forall\ x, y\ (x = y \vee x < y \vee y < x)$,

**no end:** $\forall\ x\ \exists y\ (x < y)$,

**density:** between every two precedence-related instants there is another instant: $\forall\ x, y\ (x < y \Rightarrow \exists z\ (x < z \wedge z < y))$.

In order to talk about past and future events in first-order logic, we need a reference to "now". To do so, let $N(x)$ mean "$x$ is the current instant". Here are some statements:

- "It is always the case that $\phi$":
  $N(x) \Rightarrow \forall y\ (y < x \Rightarrow \phi(y))$.
- "Sometime in the future it will be the case that $\phi$":
  $N(x) \Rightarrow \exists y\ y < x \wedge \phi(y)$.
- "$\phi$ occurs infinitely often" (some hypotheses on $\langle T, <\rangle$ are required):
  $\forall x\ \exists y\ x < y \wedge \phi(y)$.

**First-order logic is not enough** First-order logic cannot express everything. For instance, it cannot express that a linear time flow is well-ordered. A totally ordered set $\langle X, \leq\rangle$ is well-ordered $\stackrel{\text{def}}{\Leftrightarrow}$ every non-empty set of $X$ has a least element. $\langle \mathbb{N}, <\rangle$ is well-ordered, $\langle \mathbb{Z}, <\rangle$ is not. For this, one needs monadic second-order logic, where quantification over sets is allowed. Another standard property that is not first-order definable is to require that a relation $R$ is the transitive closure of another relation $R'$.

**Interval-based approach** In the interval-based models of time, the primitive temporal entities are interval and the basic relationships are equality, disjoint precedence, inclusion and overlap. A flow of time is represented as a structure of the form

$$\langle T, \prec, \sqsubseteq, O \rangle,$$

(equality is implicit). Here are some first-order sentences expressing standard properties:

**reflexivity of $\sqsubseteq$:** $\forall\ x\ x \sqsubseteq x$,

**anti-symmetry of $\sqsubseteq$:** $\forall\ x, y\ (x \sqsubseteq y \wedge y \sqsubseteq x \Rightarrow x = y)$,

**symmetry of $O$:** $\forall\ x, y\ xOy \Rightarrow yOx$,

**atomicity of $\sqsubseteq$:** $\forall\ x\ \exists\ y\ (y \sqsubseteq x \wedge \forall\ z\ (z \sqsubseteq y \Rightarrow z = y))$.

There exist proposal for interval temporal logics in artificial intelligence (Allen's logic [All83]) and in applications to computer science such as real-time logics (see e.g. [Koy90]).

**Current framework** In this course we will work in the most traditional useful framework for temporal logics: the models of times are instant-based, (propositional) temporal logic is the core logical framework (no first-order logic).

## 1.2 Aspects of temporality in computer science

This section is strongly inspired from [Gor00, Sect. 1.4].

Here are examples of the various ways in which reasoning about time and dealing with temporal phenomena is relevant and useful in computer science:

- specification and verification of concurrent and reactive systems,
- scheduling of the execution of programs by an operating system,
- synchronization of concurrent processes,
- temporal databases,
- real-time processes and systems.

Temporal properties of systems can be classified in many ways but some classes appear to be recurrent and meaningful when checking properties:

**Invariance, or safety properties:** what must not happen throughout the computation will not happen. Examples:

- A train will not pass a red semaphore.
- The reactor will not overheat.
- Partial correctness properties: if a pre-condition $P$ holds at the input of the program, then whenever it terminates and a post-condition $Q$ will hold at the output.
- The counter $x$ never takes the zero value.
- A nonce never takes twice the same value.

**Eventuality, or liveness properties:** what has to happen during the computation will happen. Examples:

- Once a printing job is activated, eventually it will be completed.

- If a message is sent, eventually it will be delivered.
- Total correctness properties: if a pre-condition $P$ holds at the input of the program, then it will terminate and a post-condition $Q$ will hold at the output.

**Fairness properties:** all processes will be treated fairly by the operating system (scheduler, ...). A typical situation: a process is enabled for the next step of its execution, and sends a request for scheduling. It may or may not be immediately scheduled for execution, because of competition with other processes. A fair scheduling would mean that if the process is persistent enough then eventually its request will be granted.

**Transition systems** A widely used formalism for representing concurrent processes is transition system $\langle S, (\xrightarrow{a})_{a \in Act} \rangle$ where

- $S$ is a non-empty set of states,
- $Act$ is a non-empty set of actions,
- $\xrightarrow{a}$ is a binary relation in $S \times S$.

Examples:

- programs or processes run concurrently on the same computing device,
- finite automata,
- coffee machines,
- Kripke[1] frames (used in modal logic).

[1]Saul Kripke is a logician that has been immensely influential in modal logic. The minimal modal logic K was named after Kripke because of his contributions to semantics of modal logics.

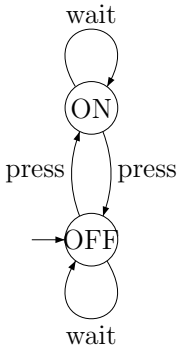



Figure 1.1: Labelled transition system

A labelled transition system is a transition system augmented with a labelling: $S \to \mathcal{P}(\text{PROP})$ where PROP is a set of propositional variables (atomic properties). A run is a sequence of the form

$$s_0 \stackrel{a_0}{\to} s_1 \stackrel{a_1}{\to} s_2 \stackrel{a_2}{\to} \ldots$$

which can be viewed as a finite or infinite path. In addition we may require that $s_0$ belongs to a set of initial states (for instance those in which all the counters are equal to zero). A run is also called a computation or an execution.

Here are executions of the labelled transition system represented in Figure 1.2:

- OFF $\stackrel{wait}{\longrightarrow}$ OFF $\stackrel{wait}{\longrightarrow}$ OFF $\stackrel{wait}{\longrightarrow}$ OFF $\stackrel{wait}{\longrightarrow} \ldots$,
- OFF $\stackrel{wait}{\longrightarrow}$ OFF $\stackrel{press}{\longrightarrow}$ ON $\stackrel{press}{\longrightarrow}$ OFF $\stackrel{wait}{\longrightarrow} \ldots$,
- OFF $\stackrel{press}{\longrightarrow}$ ON $\stackrel{press}{\longrightarrow}$ OFF $\stackrel{press}{\longrightarrow}$ ON $\stackrel{press}{\longrightarrow} \ldots$,

**Abstractions** Number of abstractions and idealizations are made in this formalization of the notion of computation. For instance,

in this framework we are not interested in the details or the structure of the program which performs the computation. A different approach consists in considering also the internal structure of the programs. This is the approach followed by the logics of programs such as the propositional dynamic logic PDL, or various logics of processes (see e.g. [HKT00]). These logics are central from an historical viewpoint however in practice, they are seldom used as formal specification languages.

## 1.3 The language of temporal logic

This section is partly inspired from [BBF+01, Sect. 2.1].

**Desirable properties of formal languages** Temporal logic are formal specification languages that share the following properties:

- to have an underlying flow of time in its models,
- to define mathematically the correctness of programs/systems,
- to express properties without ambiguity,
- to make formal proofs.

**Temporal logic well-suited for reactive systems** Temporal logic is specifically tailored for statements and reasoning which involve the notion of order in time. For instance, compared with the mathematical formulas, temporal logic notation is clearer and simpler. Temporal logic states properties of executions from labelled transition systems. Reactive systems can be modeled by such transition systems especially since they do not necessarily

compute values but maintain some interaction between different components. For instance, they include computer operating systems and network communication protocols.

**Naive first-order approach** First-order logic (with many sorts) can be easily used as a temporal logic assuming that the properties of the flow of time are definable in first-order logic. Indeed, each atomic formula of the form $P(t_1, \ldots, t_n)$ where the $t_i$'s are terms and $P$ is an $n$-ary predicate symbol, can be transformed into $P(t, t_1, \ldots, t_n)$ where the first argument refers to absolute time. The individual $a$ always satisfies the property $P$ can be written as $\forall t \in T\ P(t, a)$. However, first-order logic as a temporal logic suffers several drawbacks:

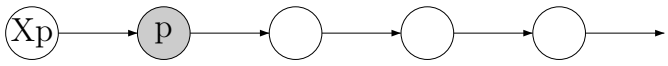
- satisfiability for first-order logic is undecidable,
- one may not need full expressive of the first-order logic but only a few temporal macros (for instance just the most common adverbs making reference to time, i.e. "always", "sometime", "never", ...),
- second-order properties are also helpful.

**Temporal operators: next, sometime and until** The temporal combinators allows one to speak about the sequencing of the states along an execution, rather than about the states taken individually. The simplest temporal combinators are X, F and G. Below, we shall freely use the Boolean operators $\neg$ (negation), $\vee$ (disjunction), $\wedge$ (conjunction) and $\Rightarrow$ (implication).

- Whereas $\phi$ states a property of the current state, X$\phi$ states that the next state (X for "next") satisfies $\phi$. For example,

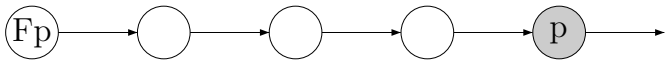
$\phi \vee \mathrm{X}\phi$ states that $\phi$ is satisfied now or in the next state.

Xp: next-time p




- Fp announces a future state (F for "future") satisfies $\phi$ without specifying which state, and G$\phi$ that all the future states satisfy $\phi$. These two combinators can be read informally as "$\phi$ will hold some day" and "$\phi$ will always be". Foundations of the modal approach to temporal logic are set in [Pri67] in which are introduced the combinators F and G.

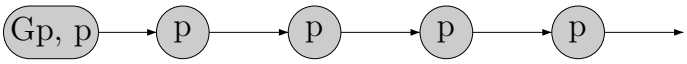
  Fp: sometimes p




**First-order equivalence** It is worth comparing G$\phi$ with $\forall\, y\ (x < y \Rightarrow \phi(y))$ and F$\phi$ with $\exists\ y\ x < y \wedge \phi(y)$. These two above-mentioned correspondences are the basis for translating into first-order logic the logic with temporal operators G and F (even with their past-time counterparts).

**Duality** The operator G is the dual of F: whatever the formula $\phi$ may be, if $\phi$ is always satisfied, then it is not true that $\neg\phi$ will some day be satisfied. Hence G$\phi$ and $\neg\mathrm{F}\neg\phi$ are equivalent.

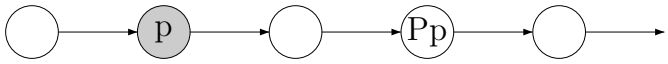
Gp: always p



**Example** By way of example, the expression `alert` $\Rightarrow$ F `halt` means that if we (currently) are in a state of alert, then we will (later) be in a halt state.

**Past-time operators** Likewise, the past operators P ("sometime in the past") and H ("always in the past") are also introduced in [Pri67].

Pp: sometime in the past p




**Quantitative aspects** Quantitative refinements for such operators are numerous. For instance, $F_{]1,2[}\phi$ would mean that sometime in the future $\phi$ is satisfied within the time-interval $]1, 2[$.

- It is the ability to arbitrarily nest the various temporal combinators which gives temporal logic its power and its strength. The example G(`alert` $\Rightarrow$ F `halt`) has an F in the scope of the G, to be compared with G(`halt` $\Rightarrow$ P `alert`).

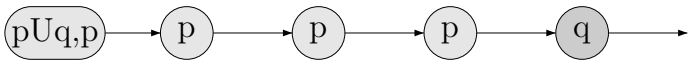
  Starting from simpler formulae, temporal combinators produce new formulae whose meaning derives from the meaning of its components.

  The nesting of F and G is very often used to express repetition properties. An ubiquitous property is to state that a property occurs infinitely often which can be written as: GF$\phi$. We use $F^{\infty}\phi \stackrel{\text{def}}{=} GF\phi$.

  The dual "almost always" temporal operator is defined by $G^{\infty}\phi \stackrel{\text{def}}{=} FG\phi$.

- The U combinator (for "until") is richer and more complicated than the combinator F. $\phi_1 \mathrm{U} \phi_2$ states that $\phi_1$ is verified until $\phi_2$ is verified. More precisely: $\phi_2$ will be verified some day, and $\phi_1$ will hold in the meantime.

  pUq: p until q

  **Example** The example G(`alert` ⇒ F `halt`) can be completed with the statement that "starting from a state of alert, the alarm remains activated until the halt state is eventually reached":

  $$\mathrm{G}(\texttt{alert} \Rightarrow (\texttt{alarm}\ \mathrm{U}\ \texttt{halt})).$$

  **Sometime operator** The F combinator is a special case of U: F$\phi$ and `true`U$\phi_2$ are equivalent.

  **Weak until** There exists also a "weak until", denoted W. The statement $\phi_1 \mathrm{W} \phi_2$ still expresses "$\phi_1 \mathrm{U} \phi_2$", but without the inexorable occurrence of $\phi_2$ and if $\phi_2$ never occurs, then $\phi_1$ remains true until the end. So, $\phi_1 \mathrm{W} \phi_2$ is equivalent to $\mathrm{G}\phi_1 \vee (\phi_1 \mathrm{U} \phi_2)$.

Other examples of expressions:

**(safety)** $\mathrm{G}\neg(x = 0)$,

**(liveness)** $\mathrm{G}(\mathrm{p} \Rightarrow \mathrm{Fq})$,

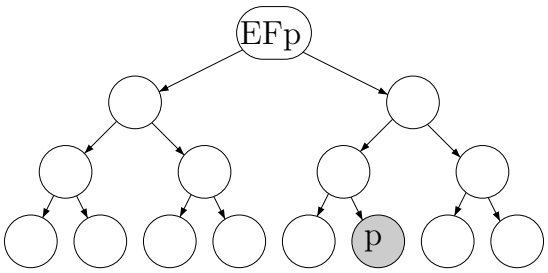
**(total correctness)** $(\mathit{init} \wedge \mathrm{p}) \Rightarrow \mathrm{F}(\mathit{end} \wedge \mathrm{q})$,

**(strong fairness)** $\mathrm{F}^{\infty}\mathit{enabled} \Rightarrow \mathrm{F}^{\infty}\mathit{executed}$.

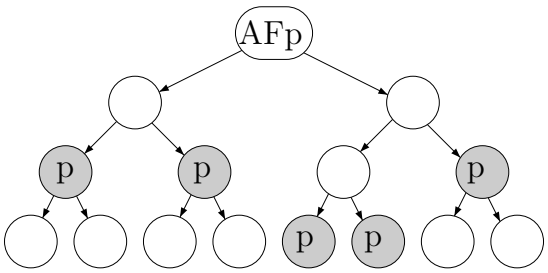
**Quantification over executions** The language introduced so far can only state properties along one execution. There remains to express the tree aspect of the behavior: many futures are possible starting from a given state. Special purpose quantifiers, A (compare with $\forall$ in first-order logic) and E (compare with $\exists$ in first-order logic), allow one to quantifier over the set of executions. These are called path quantifiers.

The expression A$\phi$ states that all the executions out of the current state satisfy the property $\phi$. Dually, E$\phi$ states that from the current state, there exists an execution satisfying $\phi$. It is worth observing that A and E quantify over paths whereas G and F quantify over positions a long a path.

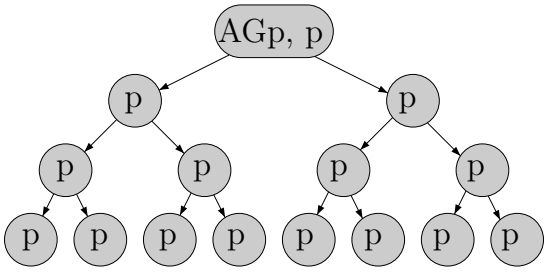
The expression EF$\phi$ states that it is possible (by following a suitable execution) to have $\phi$ some day.



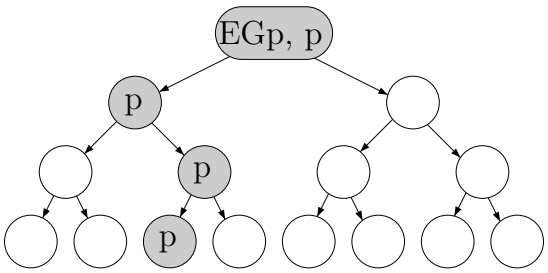

The expression AF$\phi$ states that we will necessarily have $\phi$ some day, regardless of the chosen execution.




The expression AG$\phi$ states that $\phi$ holds true in all future states including now.




The expression EG$\phi$ states that there is an execution on which $\phi$ holds always true.




"Branching-time logics" refers to logics which have this capability to freely quantify over the paths that are possible.

In forthcoming chapters, we introduce several standard temporal logics for formal specification of programs, either branching-time logics (CTL, CTL*, modal $\mu$-calculus) and linear-time temporal logics (LTL and extensions). Various relations exist between these logics, for instance in term of expressive power. We shall focus on the computational complexity of reasoning tasks for such logics, including satisfiability and model-checking.

## 1.4 Example: pay phone controller

In Figure 1.4, we describe a pay phone controller introduced in [CC00].

- $x$ is the number of quarters which have been inserted,
- $y$ measures the total communication time,
- $x'$ [resp. $y'$] is the new value of $x$ [resp. $y$].
- The controller interacts with the environment. Messages followed by a question mark are received by the controller

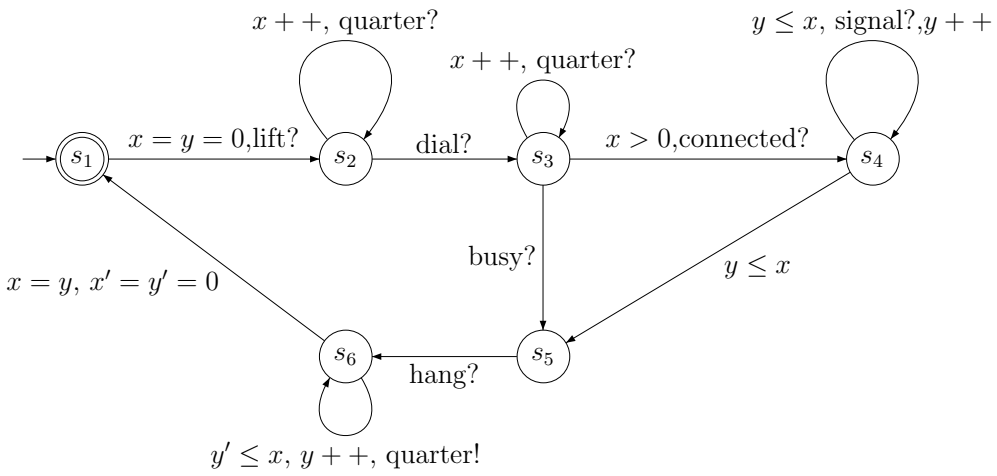



Figure 1.2: A pay phone controller

and messages followed by an exclamation mark are sent by the controller.

A configuration of the controller is a triple $\langle s, c_1, c_2 \rangle$ where $s$ is a control state and $c_1$ [resp. $c_2$] is the value of $x$ [resp. $y$]. Because of the presence of messages, queues for messages should be added (omitted here) An execution is a (possibly infinite) sequence of configurations (constrained by the transitions of the controller). So the automaton presented in Fig. 1.4 is a finite and concise representation of an infinite labeled transition system (its interpretation).

**Other properties** Here are examples of properties about the pay phone controller when evaluated at the initial configuration $\langle s_1, 0, 0 \rangle$. The careful reader will notice which ones hold true and under which assumptions.

- Communication time is never greater than the number of

inserted quarters:

$$\mathrm{A\ G}\ \neg(y > x).$$

- The number of quarters is infinitely often equal to zero:

$$\mathrm{A\ GF}\ x = 0.$$

- There is an execution of the controller such that the communication time is always equal to zero:

$$\mathrm{E\ G}\ y = 0.$$

- Whenever the communication is over, the controller regains the initial configuration:

$$\mathrm{A\ G}\ (s_5 \Rightarrow \mathrm{F}s_1).$$

- Whenever the control state $s_1$ is reached, $x = y = 0$ and conversely:

$$\mathrm{A\ G}(s_1 \Leftrightarrow (x = 0 \wedge y = 0)).$$

# Chapter 2

# Linear-time temporal logic LTL

## 2.1 Introduction

synthesis problem in [Ros91]

**Syntax** LTL ("Linear-time temporal logic") formulae are built from the following abstract grammar:

$$\phi ::= \overbrace{\mathrm{p}_i \mid \neg\phi \mid \phi \wedge \psi \mid \phi \vee \psi}^{\text{propositional calculus}} \mid \overbrace{\mathrm{X}\phi \mid \mathrm{F}\phi \mid \phi \mathrm{U} \psi}^{\text{temporal combinators}}$$

where $\mathrm{PROP} = \{\mathrm{p}_1, \mathrm{p}_2, \ldots\}$ is a countably infinite set of propositional variables. Elements of PROP are obtained by abstracting properties, for instance p may mean "$x = 0$".

The operators $\vee$ and F are definable from other operators. For instance, $\mathrm{F}\phi$ is equivalent to $\top \mathrm{U} \phi$ (with $\top = \mathrm{p} \vee \neg \mathrm{p}$).

**Abbreviations** We use the following standard abbreviations:

- $\mathrm{G}\phi$ for $\neg\mathrm{F}\neg\phi$,
- $\phi_1 \Rightarrow \phi_2$ for $\neg\phi_1 \vee \phi_2$,
- $\mathrm{F}^\infty\phi$ for $\mathrm{GF}\phi$.

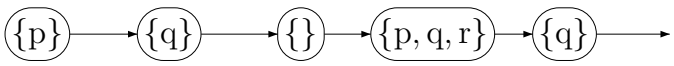
**Fragments** We write $\mathrm{LTL}_n^k(\mathrm{H}_1, \mathrm{H}_2, \ldots)$ to denote the LTL fragment restricted

- to temporal operators in $\mathrm{H}_1, \mathrm{H}_2, \ldots$,
- to formulae with temporal depth at most $k \geq 0$ and
- with at most $n \geq 1$ propositional variables.

The temporal depth of some formula $\phi$ is the maximal number of temporal operators on a branch of the formula tree for $\phi$. For instance, $\mathrm{LTL}_\omega^2(\mathrm{F})$ denotes the set of LTL formulae with temporal depth at most 2 built from the single temporal operator F.

$|\phi|$ denotes the size of the formula $\phi$ viewed as a string of characters.

A structure (or model) for LTL is an infinite sequence $\sigma : \mathbb{N} \to \mathcal{P}(\mathrm{PROP})$, i.e. an infinite word of $\mathcal{P}(\mathrm{PROP})^\omega$.




Given a structure $\sigma$, $i \in \mathbb{N}$ and a formula $\phi$, we define inductively the satisfaction relation $\models$ as follows:

- $\sigma, i \models \mathrm{p} \overset{\text{def}}{\Leftrightarrow} \mathrm{p} \in \sigma(i)$,
- $\sigma, i \models \neg\phi \overset{\text{def}}{\Leftrightarrow} \sigma, i \not\models \phi$ ($\vee$ and $\wedge$ have their usual semantics),
- $\sigma, i \models \mathrm{X}\phi \overset{\text{def}}{\Leftrightarrow} \sigma, i+1 \models \phi$,
- $\sigma, i \models \mathrm{F}\phi \overset{\text{def}}{\Leftrightarrow}$ there is $j \geq i$ such that $\sigma, j \models \phi$,
- $\sigma, i \models \phi \mathrm{U} \psi \overset{\text{def}}{\Leftrightarrow}$ there is $j \geq i$ such that $\sigma, j \models \psi$ for every $i \leq k < j$, $\sigma, k \models \phi$.

We write $\sigma \models \phi$ instead of $\sigma, 0 \models \phi$.

$\text{Models}(\phi) \stackrel{\text{def}}{=} \{\sigma \in \mathcal{P}(\text{PROP})^\omega : \sigma \models \phi\}$.

$\phi$ is LTL satisfiable $\stackrel{\text{def}}{\Leftrightarrow}$ $\text{Models}(\phi) \neq \emptyset$.

The satisfiability problem for LTL, denoted by SAT(LTL), is defined as follows:

**input:** an LTL formula $\phi$,

**sortie:** 1 if there is $\sigma$ such that $\sigma \models \phi$, 0 otherwise.

Let us now consider the model-checking problem. A Kripke structure $\mathcal{M} = \langle W, R, L \rangle$ is triple such that

- $W$ is non-empty set of states,
- $R$ is a binary relation (accessibility relation, one-step relation),
- $L$ is a labeling $L : W \to \mathcal{P}(\text{PROP})$.

$\mathcal{M}$ is simply a directed graph for which each node is labeled by a propositional interpretation (labeled transition system).

A path in $\mathcal{M}$ is a sequence $s_0 s_1 \ldots$ (finite or infinite) such that $s_i R s_{i+1}$ for every $i \geq 0$. We write $\text{Paths}(\mathcal{M}, s_0)$ to denote the set of infinite paths of $\mathcal{M}$ starting at the state $s_0$.

Alternatively, we also write $\text{Paths}(\mathcal{M}, s_0)$ to denote the set of infinite paths starting at $s_0$ on the alphabet $\mathcal{P}(\text{PROP})^\omega$. The first letter for such paths is therefore $L(s_0)$.

The model-checking problem for LTL, denoted by $\text{MC}^\exists(\text{LTL})$, is defined as follows:

**input:** an LTL formula $\phi$, a finite and total[1] Kripke structure $\mathcal{M}$ and $s_0 \in W$,

**output:** 1 if there is an infinite path starting at $s_0$ such that $\sigma \models \phi$ (written $\mathcal{M}, s_0 \models_\exists \phi$), 0 otherwise.

Without any loss of generality, in the above statement we can assume that the codomain of the labeling $L$ is restricted to the propositional variables occurring in $\phi$.

The size of $\langle W, R, L\rangle$ is defined by:

$$\mathrm{card}(W) + \mathrm{card}(R) + \Sigma_{w\in W}\mathrm{card}(L(w)).$$

It is easy to check that $\mathcal{M}, s_0 \models_\exists \phi$ iff $\mathrm{Paths}(\mathcal{M}, s_0) \cap \mathrm{Models}(\phi) \neq \emptyset$.

This definition is dual to the one also used in formal verification where a universal quantification on paths is considered (dual problem denoted by $\mathrm{MC}^\forall$(LTL) with the relation $\mathcal{M}, s_0 \models_\forall \phi$).

By using standard properties about complexity classes about deterministic Turing machines (see e.g. [Pap94, chapter 7]), one can show that $\mathrm{MC}^\forall$(LTL) is in PSPACE [resp. PSPACE-hard] iff $\mathrm{MC}^\exists$(LTL) is in PSPACE [resp. PSPACE-hard].

In Figure 2.1, we present a Kripke structure obtained from the labeled transition system described in Figure 1.2 by deleting labels on transitions. In Figure 2.1, ON and OFF are propositional variables and by abuse of language we identify them with states where they hold true respectively.

[1] $\forall x \in W, \exists y \in W,\ \langle x, y\rangle \in R$.

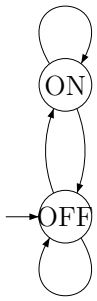



Figure 2.1: An elementary Kripke structure

We leave to the reader to check that the properties below hold true:

- $\mathcal{M}, \mathrm{ON} \models_\exists \mathrm{F}^\infty\mathrm{ON} \wedge \mathrm{F}^\infty\mathrm{OFF}$,
- $\mathcal{M}, \mathrm{ON} \models_\exists \neg\mathrm{F}^\infty\mathrm{OFF}$,
- $\mathcal{M}, \mathrm{ON} \models_\exists \mathrm{G}(\mathrm{ON} \Rightarrow \mathrm{XX\ OFF})$.

**Exercise 2.1.1**.

1. $\phi[\psi]_\rho$ be an LTL formula with subformula $\psi$ at the occurrence $\rho$. Show that $\phi[\psi]_\rho$ is satisfiable iff $\phi[\mathrm{p}]_\rho \wedge \mathrm{G}(\mathrm{p} \Leftrightarrow \psi)$ is satisfiable where p is a new propositional variable no occurring in $\phi[\psi]_\rho$.
2. Conclude that there is a logarithmic space reduction from SAT(LTL) to SAT($\mathrm{LTL}^2_\omega$).

## 2.2 Standard reductions

### 2.2.1 A few complexity results

Before stating results about LTL complexity, let us recall a few facts about complexity classes;

- co-P= P, P$\subseteq$ NP and it is open whether P$\neq$ NP.
- PSPACE= NPSPACE= co-PSPACE,
- PSPACE$\subseteq$ EXPTIME, co-EXPTIME= EXPTIME, P$\neq$ EXPTIME and it is open whether PSPACE$\neq$ EXPTIME.

More details can be found in [Pap94]. In order to show that a problem $\mathcal{P}$ is C-hard, we can either reduce every Turing machine in C to $\mathcal{P}$ (the master reduction) or reduce another problem already known to be C-hard. Similarly, in order to show that a problem is in C, we can either design an ad hoc algorithm in C or reduce $\mathcal{P}$ (in logarithmic space) to another problem already known to be in C (assuming that C contains LOGSPACE) .

**Exercise 2.2.1**. Show that SAT($\mathrm{LTL}_1$(X)) is NP-complete.

### 2.2.2 From model-checking to satisfiability

The first reduction we present is a logarithmic space reduction from $\mathrm{MC}^{\exists}$(LTL) to SAT(LTL).

**Proposition 2.2.1**. There is a logarithmic space reduction from $\mathrm{MC}^{\exists}$(LTL) to SAT(LTL).

**Proof.** The main idea of the proof is to encode a finite Kripke structure by an LTL formula [SC85, page 740].

Let $\mathcal{M} = \langle W, R, L\rangle$ a finite and total Kripke structure and $\phi$ be an LTL formula built over the propositional variables $\mathrm{p}_1, \ldots, \mathrm{p}_k$. To each state $s$ of $W$, we associate a new propositional variable $\mathrm{p}_s$ and we encode the valuation $L(s)$ by the formula $varprop_s$ below:

$$varprop_s \stackrel{\text{def}}{=} \bigwedge\{\mathrm{p}_i : 1 \leq i \leq k, \mathrm{p}_i \in L(s)\}\wedge$$

$$\bigwedge\{\neg \mathrm{p}_i : 1 \leq i \leq k, \mathrm{p}_i \notin L(s)\}.$$

For each state $s$, we encode the set of successors $R(s) = \{s' \in W : \langle s, s' \rangle \in R\}$ by the formula:

$$succ_s \stackrel{\text{def}}{=} \mathrm{X} \bigvee \{\mathrm{p}_{s'} : s' \in R(s)\}.$$

Every state $s$ in the model $\mathcal{M}$ is encoded by the formula $\phi_s \stackrel{\text{def}}{=} \mathrm{p}_s \Rightarrow (varprop_s \wedge succ_s)$.

Finally, the structure $\mathcal{M}$ is encoded by the formula

$$\phi_{\mathcal{M}} \stackrel{\text{def}}{=} \mathrm{G}(\bigwedge\{\phi_s : s \in W\} \wedge \mathrm{UNI})$$

where UNI is a propositional formula (without temporal operators) that states that a unique propositional variable from $\{\mathrm{p}_s : s \in W\}$ is satisfied at the current state.

One can show that $\mathcal{M}, s \models_\exists \phi$ (existential version) iff $\phi_{\mathcal{M}} \wedge \phi \wedge \mathrm{p}_s$ is satisfiable.

The proof of Proposition 2.2.1 uses a rather standard approach that consists in reducing a question of the form "$\mathcal{M} \models \phi$?" to satisfiability/validity of $\psi_{\mathcal{M}} \wedge \phi$ where $\psi_{\mathcal{M}}$ encodes the model $\mathcal{M}$. In order to answer this second question, deductive methods can be used.

**Exercise 2.2.2**. Complete the proof of Proposition 2.2.1.

### 2.2.3 LTL satisfiability is PSPACE-hard

The domino problem DOM can be easily reduced to SAT(LTL) as illustrated by the proof below. First we recall what is DOM.

**Domino problem** A domino game is a structure $Dom = \langle C, D, Col \rangle$ where

- $C$ is a finite set colours,
- $D$ is a finite set of dominoes,
- $Col : D \times \{left, right, up, down\} \to C$ is a map that assigns a color to each side of the dominoes.

We say that $Dom$ can pave the region $S \subseteq \mathbb{N} \times \mathbb{N}$ iff there is a map $S \to D$ that satisfies the color constraints: only domino sides with identical colors can be adjacent (no rotation of dominoes is allowed).

The domino problem DOM is defined as follows and is known to be PSPACE-complete.

**input:** a domino game $Dom$ with two colors $c_1$ and $c_2$ and $n \in \mathbb{N} \setminus \{0\}$ (in unary),

**output:** 1, if $Dom$ can pave $\{1, \ldots, n\} \times \{1, \ldots, m\}$ for some $m \geq 1$ where the first domino on the first line has color $c_1$ on the bottom and the last domino on the last line has color $c_2$ on the top.

**Exercise 2.2.3**. Show that DOM is in PSPACE by using the fact that PSPACE = NPSPACE.

**Proposition 2.2.2**. SAT(LTL) is PSPACE-hard.

**Proof.** We present a reduction from DOM to SAT($\mathrm{LTL}^2(\mathrm{F}, \mathrm{X})$), i.e. to formulae of temporal depth at most 2.

Let $Dom = \langle \{0, 1\}, \{d_1, \ldots, d_k\}, Col \rangle$ be a domino game with $k$ dominoes and $n \geq 1$.

The propositional variables below are used in the reduction:

- $lv$: true if the state encodes a line of $n$ dominoes,
- for all $1 \leq i \leq n$ and $1 \leq j \leq k$, we introduce the variable $\boxed{\text{j}}_i$ ("the $i$th position of the line is occupied by the domino $d_j$"),
- for every $1 \leq i \leq n$, we introduce the variables $up_{i,0}$, $down_{i,0}$, $left_{i,0}$, $right_{i,0}$, $up_{i,1}$, $down_{i,1}$, $left_{i,1}$, $right_{i,1}$. For instance, $down_{2,1}$ holds true in a state satisfying $lv$ iff the colour of the 2nd domino is 1.

On each line where $lv$ holds true, every position in $\{1, \ldots, n\}$ is occupied by a unique domino:

$$\mathrm{G}(lv \Rightarrow \bigwedge_{i=1}^{n}(\bigvee_{j=1}^{k}(\boxed{\text{j}}_i \wedge \bigwedge_{j'=1, j' \neq j}^{k} \neg \boxed{\text{j'}}_i)))$$

The propositional variables related to colors are compatible with the definition of dominoes:

$$\mathrm{G}(lv \Rightarrow \bigwedge_{i=1}^{n}\bigwedge_{j=1}^{k} \boxed{\text{j}}_i \implies$$

$$\bigwedge_{side \in \{up, down, right, left\}} side_{i,Col(d_j,side)} \wedge \neg side_{i,1-Col(d_j,side)})$$

We recall that the map $Col$ is of the form $Col : \{d_1, \ldots, d_k\} \times \{up, down, right, left\} \to \{0, 1\}$.

Color constraints on adjacent dominoes are the following:

$$\overbrace{\mathrm{G}(lv \Rightarrow \bigwedge_{i=1}^{n-1}(\bigwedge_{c \in \{0,1\}} right_{i,c} \Rightarrow left_{i+1,c}))}^{\text{horizontal constraints}} \wedge$$

$$\overbrace{\bigwedge_{i=1}^{n-1} \mathrm{G}((lv \wedge \mathrm{X}\ lv) \Rightarrow \bigwedge_{c\in\{0,1\}} up_{i,c} \Rightarrow \mathrm{X}\ down_{i,c}))}^{\text{vertical constraints}}.$$

Initial and final conditions as well as finiteness of the region are taken into account by the formula below:

$$lv \wedge \mathrm{G}(\neg lv \Rightarrow \mathrm{X}\ \neg lv) \wedge down_{1,0} \wedge \mathrm{F}(lv \wedge \mathrm{X}\ \neg lv \wedge up_{n,1})$$

We can show that the conjunction of above-mentioned formulae is LTL satisfiable iff *Dom* can pave $\{1,\ldots,n\} \times \{1,\ldots,m\}$ for some $m \geq 1$ where the first domino on the first line has color 0 on the bottom and the last domino on the last line has color 1 on the top.

The proof of Proposition 2.2.2 is from [Har85].

### 2.2.4 PSPACE-hardness of LTL model-checking

**QBF** We recall the definition of the problem QBF ("Quantified Boolean Formula") that is PSPACE-complete [SM73].

**input:** $\phi = \mathcal{Q}_1\ \mathrm{p}_1\ \ldots\ \mathcal{Q}_n\ \mathrm{p}_n\ \bigwedge_{i=1}^{m} \bigvee_{j=1}^{k_i} l_{i,j}$ where

- $\mathcal{Q}_1,\ldots,\mathcal{Q}_n \in \{\exists,\forall\}$,
- $\mathrm{p}_1,\ldots,\mathrm{p}_{k'}$ are propositional variables,
- the $l_{i,j}$'s are literals built over $\mathrm{p}_1,\ldots,\mathrm{p}_n$.

**output:** 1 if $\phi$ is satisfiable, 0 otherwise.

For instance, $\forall$ p $\forall$ q p $\Leftrightarrow$ q is not QBF satisfiable whereas $\exists$ p $\exists$ q p $\Leftrightarrow$ q is.

Let $\phi$ be an instance of QBF. $\phi$ is of the form

$$\mathcal{Q}_1 \mathrm{p}_1 \ldots \mathcal{Q}_n \mathrm{p}_n \overbrace{\bigwedge_{i=1}^{m} \bigvee_{j=1}^{k_i} l_{i,j}}^{\phi_0}$$

where $\{\mathcal{Q}_1, \ldots, \mathcal{Q}_n\} \subseteq \{\forall, \exists\}$, $\phi_0$ is a propositional formula in conjunctive normal form.

Each literal $l_{i,j}$ is a propositional variable $\mathrm{p}_{r(i,j)}$ or its negation $\neg \mathrm{p}_{r(i,j)}$ from the set $X = \{\mathrm{p}_1, \ldots, \mathrm{p}_n\}$.

One can easily show that $\phi$ is satisfiable iff there is a non-empty set $\mathcal{V}$ made of propositional interpretations of the form $v : X \to \{\top, \bot\}$ such that:

**(soundness)** for every $v \in \mathcal{V}$, $v \models \phi_0$ in the propositional sense,

**(closure)** for all $v \in \mathcal{V}$ and $1 \leq r \leq n$ such that $\mathcal{Q}_r = \forall$, there is $v' \in \mathcal{V}$ verifying

1. $v'(\mathrm{p}_r) \neq v(\mathrm{p}_r)$ and,
2. for every $1 \leq r' < r$, $v'(\mathrm{p}_{r'}) = v(\mathrm{p}_{r'})$.

Indeed, $\exists\ \mathrm{p}\ \psi$ is equivalent to $\psi[\mathrm{p} \leftarrow \bot] \vee \psi[\mathrm{p} \leftarrow \top]$ and $\forall\ \mathrm{p}\ \psi$ is equivalent to $\psi[\mathrm{p} \leftarrow \bot] \wedge \psi[\mathrm{p} \leftarrow \top]$.

**Exercise 2.2.4**. Compute a set $\mathcal{V}$ of interpretations that witnesses the satisfaction of $\forall\ \mathrm{p}_1\ \forall\ \mathrm{p}_2\ \exists\ \mathrm{p}_3\ \neg \mathrm{p}_3 \Leftrightarrow (\mathrm{p}_1 \wedge \mathrm{p}_2)$.

We associate to $\phi$ a Kripke structure $\mathcal{M}_\phi$ defined in Figure 2.2 by using propositional variables from the set below:

$$\{A_0, A_1, \ldots, B_0, B_1, \ldots \mathrm{p}_1^T, \ldots, L_1^1, \ldots\}.$$

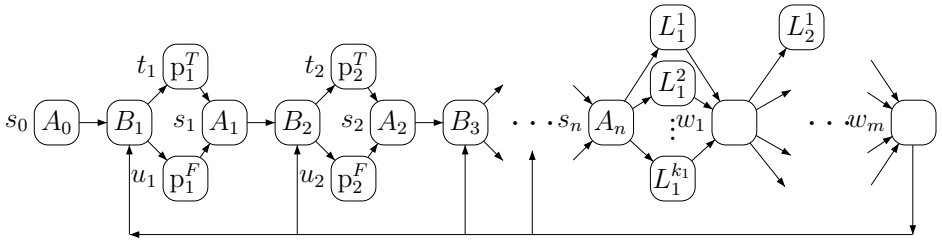



Figure 2.2: Model $\mathcal{M}_\phi$ associated to $\phi \equiv \mathcal{Q}_1 \mathrm{p}_1 \ldots \mathcal{Q}_n \mathrm{p}_n \bigwedge_{i=1}^{m} \bigvee_{j=1}^{k_i} l_{i,j}$

Each variable in $\{A_0, A_1, \ldots, B_0, B_1, \ldots \mathrm{p}_1^T, \ldots, L_1^1, \ldots\}$ holds true in a unique state of the structure $\mathcal{M}_\phi$.

Assume that $\sigma$ is an infinite path starting at $s_0$. Between $s_0$ et $s_n$, $\sigma$ encodes a propositional interpretation for variables in $X = \{\mathrm{p}_1, \ldots, \mathrm{p}_n\}$. The path $\sigma$ has to pass then via the control state $w_m$ and then returns to a control state labeled by some $B_r$ $(1 \leq r \leq n)$ where new values for interpreting $\mathrm{p}_r, \ldots, \mathrm{p}_n$ will be provided.

For each position of $\sigma$ between $s_n$ et $w_m$, we introduce the notion of current interpretation that assigns $\top$ or $\bot$ to all the propositional variables $\mathrm{p}_r$ depending on the last visited node among $u_r$ and $t_r$.

We write $\mathcal{V}(\sigma)$ to denote the set of current interpretations generated by the path $\sigma$.

Let $1 \leq r \leq n$ be such that $\mathcal{Q}_r = \forall$ and assume that when $\sigma$ visits the node $s_{r-1}$ then it visits the nodes $t_r$ and $u_r$ before possibly visiting again $s_{r-1}$. This property can be expressed in LTL(U) as follows:

$$\psi_r \stackrel{\text{def}}{=} \mathrm{G}(A_{r-1} \Rightarrow (\neg B_{r-1} \mathrm{U} \mathrm{p}_r^T) \wedge (\neg B_{r-1} \mathrm{U} \mathrm{p}_r^F))).$$

Let $\psi_{\text{closure}} \stackrel{\text{def}}{=} \bigwedge\{\psi_r \mid \mathcal{Q}_r = \forall\}$: if $\sigma$ satisfies $\psi_{\text{closure}}$, then $\mathcal{V}(\sigma)$ satisfies the condition (closure).

When $\sigma$ visits a node labeled by $L_i^j$, we say that it agrees with the current interpretation $v$ whenever $v \models l_{i,j}$.

This can be written in LTL(U) by using the fact the value of $\mathrm{p}_r$ with the current interpretation cannot be changed without having visited the node labeled by $B_r$. For every $1 \leq i \leq m$ ($m$ is the number of clauses), for every $1 \leq j \leq k_i$, we introduce the formula $\psi_{i,j}$ as follows:

$$\psi_{i,j} \stackrel{\text{def}}{=} \begin{cases} \mathrm{G}[\mathrm{p}_r^F \Rightarrow \mathrm{G}\neg L_i^j \vee \neg L_i^j \mathrm{U} B_r] & \text{if } l_{i,j} = \mathrm{p}_r, \\ \mathrm{G}[\mathrm{p}_r^T \Rightarrow \mathrm{G}\neg L_i^j \vee \neg L_i^j \mathrm{U} B_r] & \text{if } l_{i,j} = \neg \mathrm{p}_r. \end{cases}$$

Let $\psi_{\text{soundness}} \stackrel{\text{def}}{=} \bigwedge_{i=1}^{m} \bigwedge_{j=1}^{k_i} \psi_{i,j}$: if $\sigma$ satisfies $\psi_{\text{soundness}}$, then $\mathcal{V}(\sigma)$ verifies the condition (soundness).

**Lemma 2.2.3**. $\mathcal{M}_\phi, s_0 \models_\exists \psi_{\text{closure}} \wedge \psi_{\text{soundness}}$ iff $\phi$ is satisfiable.

**Proof.** If $\sigma \models \psi_{\text{closure}} \wedge \psi_{\text{soundness}}$, then the set $\mathcal{V}(\sigma)$ is non-empty, closed and correct. Hence, $\phi$ is satisfiable.

Conversely, suppose that $\phi$ is satisfiable. So there is a set of propositional interpretation $\mathcal{V}$ verifying the conditions (closure) and (soundness).

From $\mathcal{V}$, we can construct an infinite path $\sigma$ starting at $s_0$ such that $\mathcal{V}(\sigma) = \mathcal{V}$ and $\sigma \models \psi_{\text{closure}} \wedge \psi_{\text{soundness}}$: from a lexicographical enumeration of $\mathcal{V}$, $\sigma$ can be easily built so that

$\sigma \models \psi_{\text{closure}}$. This enumeration corresponds to the enumeration by increasing ordering of the natural numbers (encoded in binary) obtained from the interpretations for $\mathrm{p}_1, \ldots, \mathrm{p}_n$.

Then, to guarantee that $\sigma \models \psi_{\text{soundness}}$, each node visited between $s_n$ and the closest neighbor of the form $w_m$, $\sigma$ visits only the states labeled by $L_i^j$ satisfied by the current interpretation $v$, which is always possible because $v \models \phi_0$.

$\psi_{\text{closure}} \wedge \psi_{\text{soundness}}$ belongs to $\mathrm{LTL}^2(\mathrm{U})$ (by using the definition of G in terms of U).

As $\mathcal{M}_\phi$ and $\psi_{\text{closure}} \wedge \psi_{\text{soundness}}$ can be computed by using logarithmic space only, we can deduce:

**Proposition 2.2.4**. $\mathrm{MC}^{\exists}(\mathrm{LTL}^2(\mathrm{U}))$ is PSPACE-hard.

**Corollary 2.2.5**. $\mathrm{MC}^{\forall}(\mathrm{LTL})$ is PSPACE-hard.

An analogous construction leads to the PSPACE-hardness of $\mathrm{LTL}(\mathrm{F}, \mathrm{X})$.

## 2.3 Translation into nondeterministic Büchi automata

### 2.3.1 Basics on Büchi automata

A finite-state automata $\mathcal{A}$ is a structure $\mathcal{A} = \langle \Sigma, S, S_0, \rho, F \rangle$ such that

- $\Sigma$ is a finite set of symbols (the alphabet),
- $S$ is a finite set of states,

- $S_0 \subseteq S$ is the set of initial states,
- $\rho : S \times \Sigma \to \mathcal{P}(S)$ is the transition function,
- $F \subseteq S$ is a set of final (or accepting) states.

$\mathcal{A}$ is said to be deterministic iff $\text{card}(S_0) = 1$ and for all $s \in S$ and $a \in \Sigma$, $\text{card}(\rho(s, a)) \leq 1$.

A run is an infinite sequence $s_0 \stackrel{a_0}{\to} s_1 \stackrel{a_1}{\to} s_2 \ldots$ such that for every $i \geq 0$, $s_{i+1} \in \rho(s_i, a_i)$ and $s_0 \in S_0$.

Given a run $c$, we write $inf(c)$ to denote the set of states that occur infinitely often in $c$.

An accepting run $c$ (with Büchi accepting condition) verifies $inf(c) \cap F \neq \emptyset$.

The word $\sigma \in \Sigma^\omega$ is accepted by the accepting run $c = s_0 \stackrel{a_0}{\to} s_1 \stackrel{a_1}{\to} s_2 \ldots$ whenever $\sigma = a_0 a_1 a_2 \ldots$. The word $\sigma$ is also said to be accepted by the automaton $\mathcal{A}$ and we write $\text{L}(\mathcal{A})$ to denote the language of infinite words accepted by the Büchi automaton $\mathcal{A}$.

**Exercise 2.3.1**. Construct Büchi automata over the alphabet $\{a, b\}$ that accept the set of infinite words

- with at least one occurrence of $a$,
- with exactly one occurrence of $a$,
- in which the letters $a$ and $b$ are repeated infinitely often,
- with a finite and even number of $a$.

Given an LTL formula $\phi$, the set Models($\phi$) is a language of infinite words over the alphabet $\mathcal{P}$(PROP). We shall show that there is a Büchi automaton $\mathcal{A}_\phi$ such that $\mathrm{L}(\mathcal{A}_\phi) = \mathrm{Models}(\phi)$.

A generalized Büchi automaton is a structure

$$\mathcal{A} = \langle \Sigma, S, S_0, \rho, \{F_1, \ldots, F_k\} \rangle$$

such that $F_1, \ldots, F_k \subseteq S$. We keep the same notion of run but an accepting run $c$ (with generalized Büchi accepting condition) verifies that $inf(c) \cap F_i \neq \emptyset$ for every $F_i$.

There exists other accepting conditions but we do not need them here.

**Lemma 2.3.1**. Let $\mathcal{A} = \langle \Sigma, S, S_0, \rho, \{F_1, \ldots, F_k\} \rangle$ be a generalized Büchi automaton. There exists a (standard) Büchi automaton $\mathcal{A}^b = \langle \Sigma, S^b, S_0^b, \rho^b, F^b \rangle$ such that $\mathrm{L}(\mathcal{A}^b) = \mathrm{L}(\mathcal{A})$.

Actually, $\mathcal{A}^b$ can be computed in logarithmic space in $\mathcal{A}$.

**Proof.** Let $\mathcal{A} = \langle \Sigma, S, S_0, \rho, \{F_1, \ldots, F_k\} \rangle$ be a generalized Büchi automaton. The idea of the proof consists in defining $\mathcal{A}^b$ from $k$ copies of $\mathcal{A}$ and to simulate the generalized accepting condition by passing from one copy to another.

1. $S^b \stackrel{\text{def}}{=} S \times \{1, \ldots k\}$,
2. $S_0^b \stackrel{\text{def}}{=} S_0 \times \{1\}$,
3. $F^b \stackrel{\text{def}}{=} F_1 \times \{1\}$,
4. $\rho^b(\langle s, i \rangle, a)$ is defined as the union of the two following sets

(a) $\{\langle s', i\rangle : s' \in \rho(s, a),\ s \notin F_i\}$ (stay in the same copy if no final state in $F_i$ is reached),

(b) $\{\langle s', (i \ mod \ k) + 1\rangle : s' \in \rho(s, a),\ s \in F_i\}$ (go to the next copy if a final state in $F_i$ is reached).

The class of languages accepted by Büchi automata admits various characterizations:

- $\omega$-regular expressions,
- monadic second-order logic over $\langle \omega, < \rangle$,
- LTL augmented with fixed-point operators (see e.g. [Var88]),
- LTL augmented with automata-based temporal operators, see e.g. [Wol83],
- LTL with second-order quantification.

Languages accepted by Büchi automata form the class of $\omega$-regular languages.

**Proposition 2.3.2**. The family of $\omega$-regular languages is closed by intersection, union and complementation.

The proof for union is similar to the proof for standard finite-state automata, the proof for intersection uses an idea similar to the proof of Lemma 2.3.1. By contrast, the closure by complementation is much more difficult to show.

It is also well-known that every regular language can be accepted by a deterministic finite-state automaton. This property does not hold anymore for $\omega$-regular languages.

**Proposition 2.3.3**. There exists an $\omega$-regular language that is not accepted by any deterministic Büchi automata.

We present below the standard proof.

**Proof.** $(a+b)^* \cdot a^\omega$ is such a language. The proof is *ad absurdum*. Suppose there is a deterministic Büchi automaton $\mathcal{A} = \langle \{a,b\}, S, \{s_0\}, \rho, F \rangle$ such that $\mathrm{L}(\mathcal{A}) = (a+b)^* \cdot a^\omega$. The transition function can be extended to a partial function $\rho : S \times \{a,b\}^* \to S$ since $\mathcal{A}$ is deterministic.

The infinite word $\sigma_0 = a^\omega$ is accepted by $\mathcal{A}$ and therefore there is finite prefix $u_0$ of $\sigma_0$ such that $\rho(s_0, u_0) \in F$. The infinite word $\sigma_1 = u_0 \cdot b \cdot a^\omega$ is also accepted by $\mathcal{A}$ and therefore there is a finite prefix $u_0 \cdot b \cdot u_1$ of $\sigma_1$ such that $\rho(s_0, u_0 \cdot b \cdot u_1) \in F$.

By generalizing the process, we can find finite words $u_i$ such that $\rho(s_0, u_0 \cdot b \cdot u_1 \cdot b \ldots b \cdot u_i) \in F$. Since $S$ is finite, there is $0 \leq i < j$ such that $\rho(s_0, u_0 \cdot b \cdot u_1 \cdot b \ldots b \cdot u_i) = \rho(s_0, u_0 \cdot b \cdot u_1 \cdot b \ldots b \cdot u_i \cdot b \ldots b \cdot u_j)$. Consequently, the word

$$u_0 \cdot b \cdot u_1 \cdot b \ldots b \cdot u_i \cdot (b \ldots b \cdot u_j)^\omega$$

is accepted by $\mathcal{A}$, which is in contradiction with the fact that $\mathrm{L}(\mathcal{A}) = (a+b)^* \cdot a^\omega$.

The emptiness problem for Büchi automata is defined as follows:

**input:** a Büchi automaton $\mathcal{A}$,

**output:** 1, if $\mathrm{L}(\mathcal{A}) \neq \emptyset$, 0 otherwise.

In order to characterize the computational complexity of the emptiness problem, we can use the lemma below.

**Lemma 2.3.4**. Let $\mathcal{A} = \langle \Sigma, S, S_0, \rho, F \rangle$ be a Büchi automaton. $\mathrm{L}(\mathcal{A}) \neq \emptyset$ iff there is an infinite path in the graph $\langle S, \{\langle s, s' \rangle : \exists a \text{ s.t. } s' \in \rho(s, a)\}\rangle$ that starts at a state in $S_0$ and visits infinitely often states in $F$.

**Proposition 2.3.5**. [VW94] The emptiness problem for Büchi automata is NLOGSPACE-complete.

**Proof.** NLOGSPACE-hardness can be shown by reducing GAP ("Graph Accessibility Problem"). In order to establish the complexity upper bound, we present a nondeterministic algorithm that uses only logarithmic space in the size of $\mathcal{A} = \langle \Sigma, S, S_0, \rho, F \rangle$. By the way, showing that GAP is in NLOGSPACE can be done in a similar fashion.

1. Guess $s_0 \in S_0$ and $s_f \in F$.
2. $i := 0$; $s := s_0$ (current state);
3. While $s \neq s_f$ and $i < |S|$ do
   (a) Guess $s'$ such that $s \xrightarrow{a} s'$ for some $a \in \Sigma$;
   (b) $i := i + 1$; $s := s'$.
4. If $s \neq s_f$, then abort otherwise
   (a) $i := 0$;
   (b) While $i := 0$ or ($s \neq s_f$ and $i < |S|$) ) do
      i. Guess $s'$ such that $s \xrightarrow{a} s'$ for some $a \in \Sigma$;
      ii. $i := i + 1$; $s := s'$.
   (c) If $s = s_f$, then accept, otherwise abort.

The algorithm requires only $\mathcal{O}(log(|S|))$ bits to encode the states $s_0, s_f, s, s'$ and the counter $i$. In the above non-deterministic algorithm, "guess $x$ such that $P(x)$ holds" is equivalent to "Guess $x$; if not $P(x)$ then abort".

By contrast, the universality problem for Büchi automata is PSPACE-complete.

### 2.3.2 Automata-based approach for LTL

We present below the automata-based approach for solving LTL problems from [VW94]. The method has been used successfully to translate other logics into automata, including branching-time logics. Satisfiability and model-checking problems are reduced to the emptiness problem for Büchi automata. For instance, the main point is to transform an LTL formula $\phi$ into a Büchi automaton $\mathcal{A}_\phi$ such that $\text{Models}(\phi) = \text{L}(\mathcal{A}_\phi)$.

Figure 2.3 presents a Büchi automaton $\mathcal{A}$ such that $\text{L}(\mathcal{A}) = \text{Models}(\text{pUq})$ with $\Sigma = \{\{\}, \{\text{p}\}, \{\text{q}\}, \{\text{p}, \text{q}\}\}$.

Similarly, Figure 2.4 presents a Büchi automaton $\mathcal{B}$ such that $\text{L}(\mathcal{B}) = \text{Models}(\neg(\text{pUq}))$. Observe that $\text{L}(\mathcal{B}) = \Sigma^\omega \setminus \text{L}(\mathcal{A})$ and $\neg(\text{pUq})$ is equivalent to $\text{G}\neg\text{q} \vee (\neg\text{qU}(\neg\text{p} \wedge \neg\text{q}))$.

We wish to define in a systematic and optimal way $\mathcal{A}_\phi$ from $\phi$. Not only this approach allows to obtain optimal complexity bounds but if the construction of automata is optimized, it can also provide efficient algorithm.

Let $\mathcal{M} = \langle W, R, L \rangle$ be a finite and total Kripke structure and $s_0 \in W$. It is easy to turn $\mathcal{M}$ into a Büchi automaton recognizing the set of paths $\text{Paths}(\mathcal{M}, s_0)$. We define $\mathcal{A}_{\mathcal{M},s_0}$

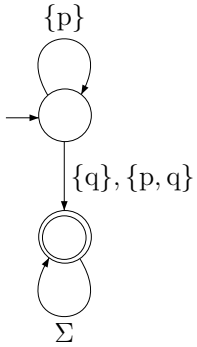


Figure 2.3: An automaton for pUq models

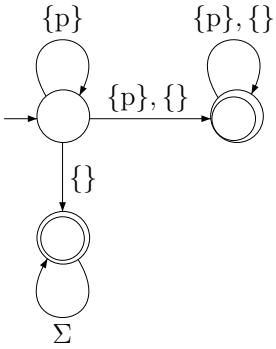


Figure 2.4: An automaton for ¬(pUq) models

such that $\mathrm{L}(\mathcal{A}_{\mathcal{M},s_0}) = \mathrm{Paths}(\mathcal{M}, s_0)$:

$$\mathcal{A}_{\mathcal{M},s_0} = \langle \mathcal{P}(\mathrm{PROP}), W, \{s_0\}, \rho, W \rangle$$

where $\rho(s,a) \stackrel{\text{def}}{=} \{s' : \langle s, s' \rangle \in R,\ a = L(s)\}$ for all $s \in W$ and $a \subseteq \mathrm{PROP}$.

**Exercise 2.3.2**. How much space do we need to compute $\mathcal{A}_{\mathcal{M},s_0}$?

The rest of this section is dedicated to show the following result.

**Proposition 2.3.6**. [VW94] For every LTL formula $\phi$, there is a Büchi automaton $\mathcal{A}_\phi$ such that

1. $\mathrm{L}(\mathcal{A}_\phi) = \mathrm{Models}(\phi)$,
2. $|\mathcal{A}_\phi|$ is in $2^{\mathcal{O}(|\phi|)}$ and,
3. $\mathcal{A}_\phi$ can be effectively computed in polynomial space in $|\phi|$.

Proposition 2.3.6 should be understood when the set of propositional variables PROP is restricted to variables occurring in $\phi$.

**Exercise 2.3.3**. By using Proposition 2.3.2, show by structural induction that there is a Büchi automaton $\mathcal{A}_\phi$ for $\phi \in \mathrm{LTL}(\mathrm{F}, \mathrm{X})$ such that $\mathrm{Models}(\phi) = \mathrm{L}(\mathcal{A}_\phi)$. Evaluate the size of $\mathcal{A}_\phi$ with respect to the size of $\phi$.

**Definition 2.3.1**. Let $\phi$ be an LTL formula. The closure of $\phi$, denoted by $\mathrm{cl}(\phi)$ is the smallest set

- containing the subformulae of $\phi$,
- closed under negation (we identify $\neg\neg\psi$ with $\psi$),
- if $\varphi_1 \mathrm{U} \varphi_2 \in \mathrm{cl}(\phi)$, then $\mathrm{X}(\varphi_1 \mathrm{U} \varphi_2) \in \mathrm{cl}(\phi)$.

$\nabla$

The closure set contains all the formulae we need to consider to check satisfiability and the cardinal of $\mathrm{cl}(\phi)$ is linear in the size of $\phi$.

**Definition 2.3.2**. Let $\phi$ be an LTL formula and $\sigma$ be an LTL model. The sequence generated from $\phi$ and $\sigma$ is an infinite sequence over $\mathcal{P}(\mathrm{cl}(\phi))^\omega$ defined as follows:

$$satseq(\sigma, \phi) = satseq(\sigma, \phi, 0)satseq(\sigma, \phi, 1)\ldots,$$

such that for every $i \geq 0$,

$$satseq(\sigma, \phi, i) \stackrel{\text{def}}{=} \{\psi \in \mathrm{cl}(\phi) : \sigma, i \models \psi\}.$$

$\nabla$

Each $\omega$-sequence $satseq(\sigma, \phi)$ (with profile $\mathbb{N} \to \mathcal{P}(\mathrm{cl}(\phi))$) satisfying $\sigma \models \phi$ verifies the properties below ($i \in \mathbb{N}$):

1. for every $\mathrm{p} \in \mathrm{cl}(\phi)$, $\mathrm{p} \in satseq(\sigma, \phi, i)$ iff $\mathrm{p} \in \sigma(i)$,
2. $\phi \in satseq(\sigma, \phi, 0)$,
3. for every $\psi \wedge \psi' \in \mathrm{cl}(\phi)$, $\psi \wedge \psi' \in satseq(\sigma, \phi, i)$ iff $\psi \in satseq(\sigma, \phi, i)$ and $\psi' \in satseq(\sigma, \phi, i)$,
4. for every $\neg\psi \in \mathrm{cl}(\phi)$, $\neg\psi \in satseq(\sigma, \phi, i)$ iff $\psi \notin satseq(\sigma, \phi, i)$,
5. for every $\mathrm{X}\psi \in \mathrm{cl}(\phi)$, $\mathrm{X}\psi \in satseq(\sigma, \phi, i)$ iff $\psi \in satseq(\sigma, \phi, i+1)$,
6. for every $\psi\mathrm{U}\psi' \in \mathrm{cl}(\phi)$, $\psi\mathrm{U}\psi' \in satseq(\sigma, \phi, i)$ iff either $\psi' \in satseq(\sigma, \phi, i)$ or ($\psi \in satseq(\sigma, \phi, i)$ and $\mathrm{X}(\psi\mathrm{U}\psi') \in satseq(\sigma, \phi, i)$)
7. for every $\psi\mathrm{U}\psi' \in \mathrm{cl}(\phi)$, if $\psi\mathrm{U}\psi' \in satseq(\sigma, \phi, i)$, then there is $j \geq i$ such that $\psi' \in satseq(\sigma, \phi, j)$.

Properties 1.-4. and 6. involve a single set $satseq(\sigma, \phi, i)$ whereas property 5. involves two sets $satseq(\sigma, \phi, i)$ and $satseq(\sigma, \phi, i+1)$. These are local properties. By contrast, property 7. is non local since there is no bound on the natural number $j$.

Hintikka[2] sequences are at first glance slight abstractions of sequences of the form $satseq(\sigma, \phi)$. In order to characterize Models($\phi$), we consider the set $\{satseq(\sigma, \phi) : \sigma, 0 \models \phi\}$ and then project.

**Definition 2.3.3**. An Hintikka sequence for $\sigma$ and $\phi$ is an infinite word $\alpha_0\alpha_1 \ldots \in \mathcal{P}(clo(\phi))^\omega$ satisfying the conditions below ($i \in \mathbb{N}$):

**(HIN1)** for every p $\in$ cl($\phi$), p $\in \alpha_i$ iff p $\in \sigma(i)$,

**(HIN2)** $\phi \in \alpha_0$,

**(HIN3)** for every $\psi \wedge \psi' \in$ cl($\phi$), $\psi \wedge \psi' \in \alpha_i$ iff $\psi \in \alpha_i$ and $\psi' \in \alpha_i$,

**(HIN4)** for every $\neg\psi \in$ cl($\phi$), $\neg\psi \in \alpha_i$ iff $\psi \notin \alpha_i$,

**(HIN5)** for every X$\psi \in$ cl($\phi$), X$\psi \in \alpha_i$ iff $\psi \in \alpha_{i+1}$,

**(HIN6)** for every $\psi$U$\psi' \in$ cl($\phi$), $\psi$U$\psi' \in \alpha_i$ iff either $\psi' \in \alpha_i$ or ($\psi \in \alpha_i$ and X($\psi$U$\psi') \in \alpha_i$),

**(HIN7)** for every $\psi$U$\psi' \in$ cl($\phi$), if $\psi$U$\psi' \in \alpha_i$, then there is $j \geq i$ such that $\psi' \in \alpha_j$.

$\nabla$

**Exercise 2.3.4**. Show that any $\alpha_0\alpha_1 \ldots \in \mathcal{P}(clo(\phi))^\omega$ satisfying (HIN2)–(HIN7) defines an unique LTL model $\sigma$ such that $\alpha_0\alpha_1 \ldots$ is an Hintikka sequence for $\sigma$ and $\phi$.

A sequence $\alpha_0\alpha_1 \ldots \in \mathcal{P}(clo(\phi))^\omega$ is an Hintikka sequence for

[2]Jaakko Hintikka is a logician regarded as the founder of formal epistemic logic and an early pioner in the semantics of modal logic.

$\phi \overset{\text{def}}{\Leftrightarrow}$ it satisfies (HIN2)-(HIN7).

Proposition 2.3.7 below states the adequacy between the sequences of the form $satseq(\sigma, \phi)$ defined semantically and Hintikka sequences that are defined syntactically. This is an auxiliary result that is central to understand the forthcoming automaton construction.

**Proposition 2.3.7**. For all formula $\phi$, $\sigma \in \text{Models}(\phi)$ and $\alpha_0\alpha_1 \ldots \in \mathcal{P}(clo(\phi))^\omega$, $\alpha_0\alpha_1 \ldots$ is an Hintikka sequence for $\phi$ and $\sigma$ iff $\alpha_0\alpha_1 \ldots = satseq(\sigma, \phi)$ and $\phi \in \alpha_0$.

Consequently, $\text{Models}(\phi)$ is non-empty iff $\phi$ has at least one Hintikka sequence. More precisely, the set of Hintikka sequences for $\phi$ induces the set of models for $\phi$ (forget the formulae that are not propositional variables): there is an bijection between the set of models for $\phi$ and the set of Hintikka sequences for $\phi$. This means in particular that $\phi$ is satisfiable iff there is an Hintikka sequence for $\phi$.

**Proof.** It is easy to show that $\alpha_0\alpha_1 \ldots = satseq(\sigma, \phi)$ and $\phi \in \alpha_0$ imply that $\alpha_0\alpha_1 \ldots$ is an Hintikka sequence for $\phi$.

Now, assume that $\alpha_0\alpha_1 \ldots \in \mathcal{P}(\text{cl}(\phi))^\omega$ is an Hintikka sequence for $\sigma$ and $\phi$. By (HIN2), $\phi \in \alpha_0$. We prove by structural induction on formulae that for all $\psi \in clo(\phi)$ and $i \in \mathbb{N}$, $\psi \in \alpha_i$ iff $\sigma, i \models \psi$. The base case $\psi = \text{p}$ and the cases with Boolean operators in the induction step are by an easy verification.

**Case 1:** $\psi = \text{X}\varphi$.

$\text{X}\varphi \in \alpha_i$ iff $\varphi \in \alpha_{i+1}$ (Definition 2.3.3(HIN5)) iff $\sigma, i{+}1 \models \varphi$

(IH) iff $\sigma, i \models \mathrm{X}\varphi$ (by definition of $\models$).

**Case 2:** $\psi = \varphi_1 \mathrm{U} \varphi_2$.

**Case 2.1:** $\varphi_1 \mathrm{U} \varphi_2 \in \alpha_i$.

By Definition 2.3.3(HIN6-HIN7), there is $j \geq i$ such that $\varphi_2 \in \alpha_j$ and for every $i \leq i' < j$, $\varphi_1 \in \alpha_{i'}$. If $i = j$, then the proof of the second statement is obvious.

Indeed, let $j$ be the minimal $j \geq i$ such that $\varphi_2 \in \alpha_j$. We can show that for every $i \leq i' < j$, $\varphi_1, \mathrm{X}(\varphi_1 \mathrm{U} \varphi_2) \in \alpha_{i'}$.

**Base case:** $i' = i$.

Since $\varphi_2 \notin \alpha_i$ and $\varphi_1 \mathrm{U} \varphi_2 \in \alpha_i$, by (HIN6), $\varphi_1, \mathrm{X}(\varphi_1 \mathrm{U} \varphi_2) \in \alpha_i$.

**Induction:** (IH)$'$ Suppose $\varphi_1, \mathrm{X}(\varphi_1 \mathrm{U} \varphi_2) \in \alpha_i \cap \ldots \cap \alpha_{i'-1}$. Consequently, $\varphi_1 \mathrm{U} \varphi_2 \in \alpha_{i'}$. By minimality of $j$, $\varphi_2 \notin \alpha_{i'}$ and by (HIN6), we also get that $\varphi_1, \mathrm{X}(\varphi_1 \mathrm{U} \varphi_2) \in \alpha_{i'}$.

By (IH), there is $j \geq i$ such that $\sigma, j \models \varphi_2$ and for every $i \leq i' < j$, we have $\sigma, i' \models \varphi_1$. By definition of $\models$, we get $\sigma, i \models \varphi_1 \mathrm{U} \varphi_2$.

**Case 2.2:** $\sigma, i \models \varphi_1 \mathrm{U} \varphi_2$.

By definition of $\models$, there is $j \geq i$ such that $\sigma, j \models \varphi_2$ and for every $i \leq i' < j$, $\sigma, i' \models \varphi_1$. We show that for every $i \leq i' \leq j$, $\varphi_1 \mathrm{U} \varphi_2 \in \alpha_{i'}$ (by induction on $i'$).

**Base Case:** $i' = j$.

By (IH), $\varphi_2 \in \alpha_j$ and by Definition 2.3.3(HIN6), $\varphi_1 \mathrm{U} \varphi_2 \in \alpha_j$.

**Induction:** (IH)′ Suppose that $\varphi_1 \mathrm{U} \varphi_2 \in \alpha_{i'+1} \cap \ldots \cap \alpha_j$. By (IH), $\varphi_1 \in \alpha_{i'}$ and by (IH)′, $\varphi_1 \mathrm{U} \varphi_2 \in \alpha_{i'+1}$. By satisfaction of (HIN5), $\mathrm{X}(\varphi_1 \mathrm{U} \varphi_2) \in \alpha_{i'}$ and by Definition 2.3.3(HIN6), $\varphi_1 \mathrm{U} \varphi_2 \in \alpha_{i'}$.

Let $\phi$ be an LTL formula and $\mathcal{A}_\phi^H$ be a generalized Büchi automaton (quite redundant at the moment) defined as follows:

$$\langle \mathcal{P}(\mathrm{PROP}) \times S, S, S_0, \rho, \{F_1, \ldots, F_k\} \rangle$$

with

- $S$ is the set of subsets of $\mathrm{cl}(\phi)$ that are maximally consistent at the propositional level and such that for every $\alpha \in Q$, for every $\varphi_1 \mathrm{U} \varphi_2 \in \mathrm{cl}(\phi)$, $\varphi_1 \mathrm{U} \varphi_2 \in \alpha$ iff either $\varphi_2 \in \alpha$ or ($\varphi_1 \in \alpha$ and $\mathrm{X}(\varphi_1 \mathrm{U} \varphi_2) \in \alpha$. Definition of $S$ takes care of the conditions (HIN3), (HIN4) and (HIN6).
- $S_0 \stackrel{\text{def}}{=} \{\alpha \in S : \phi \in \alpha\}$. See condition (HIN2).
- $\rho(\alpha, \langle a, \beta \rangle) \stackrel{\text{def}}{=} \{\alpha' \in S : \alpha = \beta,\ a = \mathrm{PROP} \cap \alpha,\ \alpha \rightsquigarrow \alpha'\}$ where the one-step relation $\rightsquigarrow$ is defined by: $\alpha \rightsquigarrow \alpha'$ iff for all $\mathrm{X}\psi \in \mathrm{cl}(\phi)$, $\mathrm{X}\psi \in \alpha$ iff $\psi \in \alpha'$. There is some redundancy here that will be withdrawn a bit later.
- The set of final states will take care of the condition (HIN7). If the temporal operator U does not occur in $\phi$, then $\{F_1\} \stackrel{\text{def}}{=} \{S\}$ and $k = 1$. Otherwise, suppose that $\{\psi_1 \mathrm{U} \psi'_1, \ldots, \psi_k \mathrm{U} \psi'_k\}$ is the set of formulae in $\mathrm{cl}(\phi)$ with outermost connective U. Then, for every $i \in \{1, \ldots, k\}$, $F_i \stackrel{\text{def}}{=} \{\alpha \in S :$ either $\psi_i \mathrm{U} \psi'_i \notin \alpha$ or $\psi'_i \in \alpha\}$.

Observe that the cardinal of $S$ is exponential in the size of $\phi$.

**Proposition 2.3.8**. $\langle a_0, \alpha_0 \rangle \langle a_1, \alpha_1 \rangle \ldots \in \mathrm{L}(\mathcal{A}_\phi^H)$ iff $\alpha_0 \alpha_1 \ldots$ is an Hintikka sequence for $a_0 a_1 \ldots$ and $\phi$.

**Proof.** ($\rightarrow$) Suppose that $\langle a_0, \alpha_0\rangle\langle a_1, \alpha_1\rangle \ldots \in \mathrm{L}(\mathcal{A}_\phi^H)$. So $\alpha_0 \xrightarrow{\langle a_0,\alpha_0\rangle} \alpha_1 \xrightarrow{\langle a_1,\alpha_1\rangle} \alpha_2 \ldots$ is an accepting run of $\mathcal{A}_\phi^H$ (each $a_i$ is included in PROP). We shall establish that $\alpha_0\alpha_1 \ldots$ is an Hintikka sequence for $a_0a_1 \ldots$ and $\phi$.

Condition (HIN2) in Definition 2.3.3 holds true by definition of $S_0$ and (HIN1) holds true by definition of $\rho$.

It is easy to show that conditions HIN3), (HIN4), (HIN5) and (HIN6) are satisfied by $\alpha_0\alpha_1 \ldots$. We use the very definition of the set $S$ and the symbolic successor relation $\rightsquigarrow$.

Finally, let us concentrate our efforts to check that $\alpha_0\alpha_1 \ldots$ satisfies (HIN7). Let $\psi \mathrm{U} \psi'$ be in $\alpha_i$. The proof is *ad absurdum*. Suppose that for every $j \geq i$, $\psi' \notin \alpha_j$. Since $\alpha_0\alpha_1 \ldots$ satisfies (HIN6), either $\psi' \in \alpha_i$ or ($\psi \in \alpha_i$ and $\mathrm{X}(\psi \mathrm{U} \psi') \in \alpha_i$). By satisfaction of (HIN5), $\mathrm{X}(\psi \mathrm{U} \psi') \in \alpha_i$ iff $\psi \mathrm{U} \psi' \in \alpha_{i+1}$.

As $\psi' \notin \alpha_i$, we get that $\psi \in \alpha_i$ and $\psi \mathrm{U} \psi' \in \alpha_{i+1}$. By a simple induction, we can show that for every $j \geq i$, $\psi \in \alpha_j$ and $\psi \mathrm{U} \psi' \in \alpha_j$. Consequently, for every $j \geq i$, $\alpha_j \notin \{\alpha \in S : \text{ either } \psi \mathrm{U} \psi' \notin \alpha \text{ or } \psi' \in \alpha\}$, whence $\mathcal{A}_\phi^H$ cannot accept the word $\langle a_0, \alpha_0\rangle\langle a_1, \alpha_1\rangle \ldots$. Indeed, on the run no control state from $\{\alpha \in S : \text{ either } \psi \mathrm{U} \psi' \notin \alpha \text{ or } \psi' \in \alpha\}$ is repeated infinitely often, which leads to a contradiction.
($\leftarrow$) Arguments similar to the ($\rightarrow$) direction are used.

**Exercise 2.3.5**. Complete the case ($\leftarrow$) in the proof of Proposition 2.3.8.

From $\mathcal{A}_\phi^H$, we can easily compute a generalized automaton

$\mathcal{A}_\phi$ by simplifying the alphabet:

$$\mathcal{A}_\phi = \langle \mathcal{P}(\text{PROP}), S, S_0, \rho', \{F_1, \ldots, F_k\} \rangle$$

where $\rho'(\alpha, a) \stackrel{\text{def}}{=} \{\alpha' \in S : a = \text{PROP} \cap \alpha,\ \alpha \rightsquigarrow \alpha'\}$.

**Proposition 2.3.9**. $\text{L}(\mathcal{A}_\phi) = \text{Models}(\phi)$.

**Proof.** ($\subseteq$) Let $\alpha_0 \stackrel{a_0}{\rightarrow} \alpha_1 \stackrel{a_1}{\rightarrow} \ldots$ be an accepting run of $\mathcal{A}_\phi$. By Proposition 2.3.8, $\alpha_0\alpha_1\ldots$ is an Hintikka sequence for $a_0a_1\ldots$ and $\phi$. By Proposition 2.3.7, $\alpha_0\alpha_1\ldots = \mathit{satseq}(a_0a_1\ldots, \phi)$. Consequently, $a_0a_1\ldots \models \phi$.

($\supseteq$) Let $\sigma = a_0a_1\ldots$ be a model for $\phi$, i.e. $\sigma \models \phi$. Let $\alpha_0\alpha_1\ldots$ be equal to $\mathit{satseq}(a_0a_1\ldots, \phi)$. By Proposition 2.3.7, $\alpha_0\alpha_1\ldots$ is an Hintikka sequence for $\sigma$ and $\phi$. By Proposition 2.3.8, $\alpha_0 \stackrel{a_0}{\rightarrow} \alpha_1 \stackrel{a_1}{\rightarrow} \ldots$ is an accepting run of $\mathcal{A}_\phi$.

It is time to collect the different results to establish the PSPACE upper bound.

**Proposition 2.3.10**. SAT(LTL) $\in$ PSPACE.

**Proof.** We have established that $\phi$ is LTL satisfiable iff $\text{L}(\mathcal{A}_\phi) \neq \emptyset$ where $\mathcal{A}_\phi$ is a generalized Büchi automaton. More precisely, $\text{L}(\mathcal{A}_\phi) = \text{Models}(\phi)$. From a generalized Büchi automaton it is possible to build a standard Büchi automaton that accepts the same language in LOGSPACE, see Lemma 2.3.1. Since $\mathcal{A}_\phi$ can be built in polynomial space in $|\phi|$, it is possible to compute a Büchi automaton $\mathcal{A}'_\phi$ in polynomial space such that $\phi$ is LTL satisfiable iff $\text{L}(\mathcal{A}'_\phi) \neq \emptyset$.

Additionally, the emptiness problem for Büchi automata is NLOGSPACE-complete and composing a polynomial space reduction with a

nondeterministic logarithmic space test can be done in nondeterministic polynomial space. By Savitch's theorem, NPSPACE= PSPACE and therefore checking whether $\phi$ is LTL satisfiable can be done in polynomial space in $|\phi|$.

An alternative proof that uses less structural properties about complexity classes is to test on the fly the emptiness of $\mathrm{L}(\mathcal{A}_\phi)$ which can be done by using a polynomial amount of space with a non-deterministic algorithm.

It is easy to establish that $\mathcal{M}, s_0 \models_\exists \phi$ iff $\mathrm{L}(\mathcal{A}_{\mathcal{M},s_0}) \cap \mathrm{L}(\mathcal{A}_\phi) \neq \emptyset$. Given the automaton $\mathcal{A}_\phi = \langle \mathcal{P}(\mathrm{PROP}), S, S_0, \rho, F \rangle$, the Büchi automaton

$$\mathcal{M} \mid\mid \mathcal{A}_\phi = \langle \{0\}, W \times S, \{s_0\} \times S_0, \rho', W \times F \rangle$$

with $\rho'(\langle u, s \rangle, 0) = \{\langle v, s' \rangle : \langle u, v \rangle \in R, s' \in \rho(s, L(u))\}$ satisfies $\mathrm{L}(\mathcal{M} \mid\mid \mathcal{A}_\phi) \neq \emptyset$ iff $\mathrm{L}(\mathcal{A}_{\mathcal{M},s_0}) \cap \mathrm{L}(\mathcal{A}_\phi) \neq \emptyset$.

The size of $\mathcal{M} \mid\mid \mathcal{A}_\phi$ is in $2^{\mathcal{O}(|\phi|)} \times |\mathcal{M}|$ and by building it on the fly, $\mathcal{M}, s_0 \models_\exists \phi$ can be checked in space in $\mathcal{O}(|\phi| + log|\mathcal{M}|)$ with a nondeterministic algorithm.

**Proposition 2.3.11**. $\mathrm{MC}^\exists(\mathrm{LTL}) \in$ PSPACE.

**Corollary 2.3.12**. $\mathrm{MC}^\forall(\mathrm{LTL}) \in$ PSPACE.

**Proof.** $\mathcal{M}, s_0 \models_\forall \phi$ iff not $\mathcal{M}, s_0 \models_\exists \neg\phi$. Since $\mathrm{MC}^\exists(\mathrm{LTL}) \in$ PSPACE and PSPACE= co-PSPACE, we get that $\mathcal{M}, s_0 \models_\forall \phi$ can be checked in polynomial space in $|\phi| + |\mathcal{M}|$.

**Exercise 2.3.6**. We write LTL + $\oplus$ to denote the extension of LTL by adding a single operator $\oplus$ of arity $n \geq 1$ with semantics defined as follows. The size of a formula in LTL + $\oplus$ is defined

as the size of its formula tree. Let $t$ be the mapping from the set of LTL + $\oplus$ formulae into the set of LTL formulae such that $t$ is homomorphic for $\neg$, $\wedge$, X and U, and

- $t(\oplus(\phi_1, \ldots, \phi_n)) \stackrel{\text{def}}{=} \phi_\oplus[\mathrm{p}_1 \leftarrow t(\phi_1), \ldots, \mathrm{p}_n \leftarrow t(\phi_n)]$ where $\phi_\oplus[\mathrm{p}_1 \leftarrow \phi_1, \ldots, \mathrm{p}_n \leftarrow \phi_n]$ is the formula obtained from the LTL formula $\phi_\oplus$ built over $\mathrm{p}_1, \ldots, \mathrm{p}_n$, by simultaneously replacing every occurrence of $\mathrm{p}_i$ by $\phi_i$,
- $t(\mathrm{p}) \stackrel{\text{def}}{=} \mathrm{p}$ for the propositional variable p.

By definition, $\sigma, i \models \phi$ in LTL + $\oplus$ $\stackrel{\text{def}}{\Leftrightarrow}$ $\sigma, i \models t(\phi)$ in LTL. By way of example, the "weak until" operator is usually defined by $\phi \mathrm{W} \psi \stackrel{\text{def}}{=} (\mathrm{G}\phi) \vee \phi \mathrm{U} \psi$ and the formula $\phi_\mathrm{W}$ takes the value $(\mathrm{G}\mathrm{p}_1) \vee \mathrm{p}_1 \mathrm{U} \mathrm{p}_2$.

1. From the above definition, what can you say about the complexity of model-checking and satisfiability problems for LTL + $\oplus$?

2. Define a family of formulae $(\phi_i)_{i \in \mathbb{N}}$ in LTL + W such that for every $i \geq 1$, $|\phi_i|$ is in order $i$ and $|t(\phi_i)|$ is in order $2^i$.

3. Provide a syntactic condition on $\phi_\oplus$ that guarantees that $t(\phi)$ can be computed in polynomial-time in $|\phi|$.

4. For every formula $\phi$ in LTL + $\oplus$, we write $\mathrm{cl}_\oplus(\phi)$ to denote the smallest set of formulae satisfying the clauses for defining $\mathrm{cl}(\cdot)$ in LTL plus the following one: $\oplus(\phi_1, \ldots, \phi_n) \in \mathrm{cl}_\oplus(\phi)$ implies $\phi_\oplus[\mathrm{p}_1 \leftarrow \phi_1, \ldots, \mathrm{p}_n \leftarrow \phi_n] \in \mathrm{cl}_\oplus(\phi)$. Evaluate the cardinality of $\mathrm{cl}_\oplus(\phi)$ with respect to the sizes of $\phi$ and $\phi_\oplus$.

5. Build systematically a nondeterministic Büchi automaton that accepts exactly the models for $\phi$ in LTL + $\oplus$. No

formal proof is required but motivations for modifying the construction for LTL formulae are most welcomed.

6. Characterize the complexity of model-checking and satisfiability problems for LTL + $\oplus$.

## 2.4 CTL*: model-checking algorithm based on LTL

The CTL* is a temporal logic for which both LTL and CTL (see Chapter 3) can be viewed as syntactic fragments. x CTL* formulae are built from the following abstract grammar:

$$\phi \ ::= \ \mathrm{p} \ \mid \ \neg\phi \ \mid \ \phi_1 \wedge \phi_2 \ \mid \ \mathrm{E}\phi \ \mid \ \mathrm{A}\phi \ \mid \ \mathrm{X}\phi \ \mid \ \phi_1 \mathrm{U} \phi_2.$$

CTL* has the ability to quantify over paths. CTL* models are total Kripke models. Let $\sigma = s_0 s_1 \ldots$ be an infinite path in $\mathcal{M}$, $i \geq 0$ and $\phi$ be a formula. $\sigma, i \models \phi$ is defined inductively as follows:

- $\sigma, i \models \mathrm{p} \ \stackrel{\text{def}}{\Leftrightarrow} \ \mathrm{p} \in L(s_i)$,
- $\sigma, i \models \neg\phi \ \stackrel{\text{def}}{\Leftrightarrow} \ \sigma, i \not\models \phi$,
- $\sigma, i \models \phi_1 \wedge \phi_2 \ \stackrel{\text{def}}{\Leftrightarrow} \ \sigma, i \models \phi_1$ and $\sigma, i \models \phi_2$,
- $\sigma, i \models \mathrm{X}\phi \ \stackrel{\text{def}}{\Leftrightarrow} \ \sigma, i+1 \models \phi$,
- $\sigma, i \models \phi_1 \mathrm{U} \phi_2 \ \stackrel{\text{def}}{\Leftrightarrow}$ there is $k \geq 0$ such that $\sigma, i+k \models \phi_2$ and for every $0 \leq j < k$, $\sigma, i+j \models \phi_1$,
- $\sigma, i \models \mathrm{E}\phi \ \stackrel{\text{def}}{\Leftrightarrow}$ there is an infinite path $\sigma' = s'_0 s'_1 \ldots$ such that $s'_0 = s_i$ and $\sigma', 0 \models \phi$,

- $\sigma, i \models \mathrm{A}\phi \stackrel{\text{def}}{\Leftrightarrow}$ for every infinite path $\sigma' = s'_0 s'_1 \ldots$ such that $s'_0 = s_i$, we have $\sigma', 0 \models \phi$.

The model-checking problem for CTL*, denoted by MC(CTL*), is defined as follows:

**input:** a CTL* formula, a finite and total Kripke model $\mathcal{M} = \langle W, R, L \rangle$ and $s \in W$;

**output:** 1 if there is an infinite path $\sigma$ starting from $s$ such that $\sigma, 0 \models \phi$, 0 otherwise.

**Exercise 2.4.1**. Explain why MC(CTL) (see Chapter 3) and $\mathrm{MC}^{\exists}$(LTL) are subproblems of MC(CTL*).

**Proposition 2.4.1**. [EH86] MC(CTL*) is PSPACE-complete.

**Proof.** Since $\mathrm{MC}^{\exists}$(LTL) is a subproblem of MC(CTL*), the PSPACE-hardness is immediate. In order to show that MC(CTL*) is in PSPACE, we use known techniques for LTL plus renaming.

We write MC-LTL$^{\exists}(\mathcal{M}, s, \phi)$ to denote the function that returns `true` iff $\mathcal{M}, s \models_{\exists} \phi$. Similarly, we write MC-LTL$^{\forall}(\mathcal{M}, s, \phi)$ to denote the function that returns `true` iff $\mathcal{M}, s \models_{\forall} \phi$. We have seen that these functions can be computed in space $\mathcal{O}(|\phi| + log(|\mathcal{M}|))$. In order to establish the PSPACE upper bound, here is an algorithm based on formula renaming using only polynomial space:

**MC-CTL*($\mathcal{M} = \langle W, R, L \rangle, s \in W, \phi$)**

- If E, A do not occur in $\phi$, then return MC-LTL$^{\exists}(\mathcal{M}, s, \phi)$.

- Otherwise $\phi$ is of the form $\phi[\mathcal{Q}\psi]_\rho$ where $\mathrm{E}, \mathrm{A}$ do not occur in $\psi$ and $\mathcal{Q} \in \{\mathrm{E}, \mathrm{A}\}$. This means that $\psi \in$ LTL(X, U).

  Let $\mathrm{p}_{\mathcal{Q}\psi}$ be a new propositional variable. We define $L'$ an extension of $L$ for every $s' \in W$,
  - if not MC-LTL$^{\mathcal{Q}}(\mathcal{M}, s', \psi)$ then $L'(s') := L(s')$
  - otherwise $L'(s') := L(s') \cup \{\mathrm{p}_{\mathcal{Q}\psi}\}$.

  Return MC-CTL*$(\langle W, R, L'\rangle, s, \phi[\mathrm{p}_{\mathcal{Q}\ \psi}]_\rho)$.

Since in MC-CTL*$(\mathcal{M}, s, \phi)$, the recursion depth is at most $|\phi|$, we can show that MC-CTL* uses only polynomial space since MC-LTL$^{\exists}$ and MC-LTL$^{\forall}$ requires only polynomial space. The soundness of the algorithm is not very difficult to show.

The satisfiability problem for CTL* is much more complex: 2EXPTIME-complete.

# Chapter 3

# Model-checking CTL formulae

## 3.1 Definition

Computation Tree Logic CTL is a very popular fragment of CTL* that has been introduced in [EH85]. Most of its popularity is due to the fact that model-checking is tractable: bilinear in the size model and in the size of the formula. CTL formulae are defined as follows:

$$\phi \ ::= \ \mathrm{p} \ \mid \ \neg\phi \ \mid \ \phi_1 \wedge \phi_2 \ \mid \ \mathrm{EX}\phi \ \mid \ \mathrm{EG}\phi \ \mid \ \mathrm{E}(\phi_1 \mathrm{U} \phi_2).$$

Unlike CTL* formulae, in CTL formulae the occurences of LTL-like operators are restricted: X and U can only occur in the direct scope of either A (see below the abbreviations) or E.

CTL models are exactly CTL* models and the satisfaction relation is defined as follows (no current path is needed compared to CTL*):

- $\mathcal{M}, s \models \mathrm{p} \ \stackrel{\text{def}}{\Leftrightarrow} \ \mathrm{p} \in L(s)$,
- $\mathcal{M}, s \models \neg\phi \ \stackrel{\text{def}}{\Leftrightarrow} \ \mathcal{M}, s \not\models \phi$,
- $\mathcal{M}, s \models \phi_1 \wedge \phi_2 \ \stackrel{\text{def}}{\Leftrightarrow} \ \mathcal{M}, s \models \phi_1$ and $\mathcal{M}, s \models \phi_2$,
- $\mathcal{M}, s \models \mathrm{EX}\phi \ \stackrel{\text{def}}{\Leftrightarrow}$ there is $s' \in R(s)$ such that $\mathcal{M}, s' \models \phi$,

- $\mathcal{M}, s \models \mathrm{EG}\phi_1 \stackrel{\text{def}}{\Leftrightarrow}$ there is an infinite path $s_0 s_1, \ldots$ such that $s_0 = s$ and for every $k \geq 0$, $\mathcal{M}, s_k \models \phi_1$,
- $\mathcal{M}, s \models \mathrm{E}(\phi_1 \mathrm{U} \phi_2) \stackrel{\text{def}}{\Leftrightarrow}$ there is an infinite path $s_0 s_1, \ldots$ such that $s_0 = s$ and there is $k \geq 0$ verifying $\mathcal{M}, s_k \models \phi_2$ and for every $0 \leq i < k$, we have $\mathcal{M}, s_i \models \phi_1$.

Here are standard abbreviations:

1. $\mathrm{AX}\phi \stackrel{\text{def}}{=} \neg\mathrm{EX}\neg\phi$,
2. $\mathrm{EF}\phi \stackrel{\text{def}}{=} \mathrm{E}(\top \mathrm{U} \phi)$,
3. $\mathrm{AG}\phi \stackrel{\text{def}}{=} \neg\mathrm{EF}\neg\phi$,
4. $\mathrm{AF}\phi \stackrel{\text{def}}{=} \neg\mathrm{EG}\neg\phi$,
5. $\mathrm{A}(\phi_1 \mathrm{U} \phi_2) \stackrel{\text{def}}{=} \neg(\mathrm{E}(\neg\phi_2 \mathrm{U}(\neg\phi_1 \wedge \neg\phi_2)) \vee \mathrm{EG}\neg\phi_2))$.

As the above abbreviations suggest, other choices are possible to define primitive operators in CTL. However, the current choice is motivated by the simplification of forthcoming technical developments.

**Exercise 3.1.1**. Show that $\mathcal{M}, s \models \mathrm{A}(\phi_1 \mathrm{U} \phi_2)$ iff for every infinite path $s_0 s_1, \ldots$ such that $s_0 = s$, there is $k \geq 0$ such that $\mathcal{M}, s_k \models \phi_2$ and for every $0 \leq i < k$, $\mathcal{M}, s_i \models \phi_1$.

The CTL satisfiability problem is defined as a subproblem of CTL* satisfiability. Similarly, the CTL model-checking problem is defined as follows:

**input:** a CTL formula $\phi$, a finite and total Kripke structure $\mathcal{M}$ and $s_0 \in W$,

**output:** 1 if $\mathcal{M}, s_0 \models \phi$, 0 otherwise.

**Proposition 3.1.1**. SAT(CTL) is EXPTIME-complete and MC(CTL) is P-complete.

## 3.2 Expressivity

Here are some properties expressible in at least one logic among CTL, LTL, CTL*.

**(safety)** AGp (in CTL),

**(liveness)** AGp $\Rightarrow$ AFq (in CTL),

**(fairness)** A(GFp $\Rightarrow$ GFq) (in CTL* $\setminus$ CTL),

AG(EFp) (in CTL $\setminus$ LTL).

A(FGp) in (in LTL $\setminus$ CTL).

LTL and CTL are uncomparable fragments of CTL*.

## 3.3 Model checking algorithm

Popularity of CTL is mainly due to Proposition 3.3.1 below.

**Proposition 3.3.1**. Let $\mathcal{M} = \langle W, R, L\rangle$ be a CTL model and $\phi$ be a CTL formula. Computing the set $\{s \in W : \mathcal{M}, s \models \phi\}$ can be done in time $\mathcal{O}((\mathrm{card}(R) + \mathrm{card}(W)) \times |\phi|)$.

In Proposition 3.3.1 above, we assume that the codomain of $L$ is a subset of $\mathcal{P}(\mathrm{PROP}(\phi))$ where $\mathrm{PROP}(\phi)$ is the set of propositional variables occurring in $\phi$.

Consequently, MC(CTL) can be solved in polynomial-time which is a strong argument to use CTL as a formal language for specification. This should be compared with the complexity of MC(LTL) and MC(CTL*).

**Proof.** Let $\mathcal{M} = \langle W, R, L\rangle$ be a CTL model and $\phi$ be a CTL formula. The directed graph $\langle W, R\rangle$ is encoded by lists of neighbours (total size in $\mathcal{O}(\mathrm{card}(R) + \mathrm{card}(W))$) whereas the labeling

function $L$ is encoded by a vector of length $\text{card}(W)$: the $i$th element contains the indices of the propositional variables that hold true at the $i$th state of $W$ (arbitrary orderings of variables and states). The size of $L$ is in $\mathcal{O}(\text{card}(W) \times |\phi|)$.

Let $\phi_1, \ldots, \phi_k$ be the subformulae of $\phi$ ordered by increasing size. In case of conflict, we make an arbitrary choice for the formulae of identical sizes. Consequently,

- $\phi_k = \phi$,
- $\phi_1$ is a propositional variable,
- if $\phi_i$ is a strict subformula of $\phi_j$, then $i < j$,
- $k \leq |\phi|$.

For every $s \in W$, we build a set of formulae $l(s)$ such that

1. for every $i \in \{1, \ldots, k\}$, either $\phi_i \in l(s)$, or $\neg\phi_i \in l(s)$, but not both at the same time,
2. for every $\psi \in \{\phi_1, \ldots, \phi_k, \neg\phi_1, \ldots, \neg\phi_k\}$, $\psi \in l(s)$ iff $\mathcal{M}, s \models \psi$.

For all $i \in \{1, \ldots, k\}$ and $s \in W$, we insert either $\phi_i$ in $l(s)$ or $\neg\phi_i$ dans $l(s)$, following the above ordering of subformulae. Each set $l(s)$ is initialized to the empty set. We make a case analysis on the form of $\phi_i$ for every $i \in \{1, \ldots, k\}$. Each step requires time in $\mathcal{O}(\text{card}(W) + \text{card}(R))$.

**Case 1:** $\phi_i$ is a propositional variable.
For every $s \in W$, if $\phi_i \in L(s)$, then insert $\phi_i$ in $l(s)$ otherwise insert $\neg\phi_i$ in $l(s)$.

**Case 2:** $\phi_i = \neg\phi_{i_1}$ for some $i_1 < i$.
For every $s \in W$, insert $\neg\phi_i$ in $l(s)$ if $\phi_{i_1} \in l(s)$ otherwise skip ($\phi_i$ is already in $l(s)$).

**Case 3:** $\phi_i = \phi_{i_1} \wedge \phi_{i_2}$ for some $i_1, i_2 < i$.
For every $s \in W$, insert $\phi_i$ in $l(s)$ if $\{\phi_{i_1}, \phi_{i_2}\} \subseteq l(s)$ otherwise insert $\neg\phi_i$ in $l(s)$.

**Case 4:** $\phi_i = \mathrm{EX}\phi_{i_1}$ for some $i_1 < i$.
For every $s \in W$, if there is $s' \in R(s)$ such that $\phi_{i_1}$ is in $l(s')$, then insert $\phi_i$ in $l(s)$, otherwise insert $\neg\phi_i$ in $l(s)$.

**Case 5:** $\phi_i = \mathrm{E}(\phi_{i_1}\mathrm{U}\phi_{i_2})$ for some $i_1, i_2 < i$.
One can show that $\mathcal{M}, s \models \phi_i$ iff there is a $R$-path $s_0 \rightarrow s_1 \rightarrow \ldots \rightarrow s_n$ in $\mathcal{M}$ such that

- $s_0 = s$,
- $\phi_{i_2} \in l(s_n)$ (use of induction hypothesis for the correctness),
- for every $i \in \{0, \ldots, n-1\}$, $l(s_i) \cap \{\phi_{i_1}, \phi_{i_2}\} \neq \emptyset$.

For every $j \in \{1, 2\}$ we define

$$W_j \stackrel{\text{def}}{=} \{w \in W : \phi_{i_j} \in l(s)\}.$$

Let $\mathcal{M}' \stackrel{\text{def}}{=} \langle W', R', L' \rangle$ be the Kripke model such that

1. $W' \stackrel{\text{def}}{=} W_1 \cup W_2$.
2. $R' \stackrel{\text{def}}{=} R^{-1} \cap (W' \times W')$,
3. $L'$ is the restriction of $L$ to $W'$.

For every $s \in W$, if there is $s' \in W_2$ such that $s \in (R')^*(s')$ then insert $\phi_i$ in $l(s)$ otherwise insert $\neg\phi_i$ in $l(s)$. In order to show that this step requires time in $\mathcal{O}(\mathrm{card}(W) + \mathrm{card}(R))$, we use the following result from graph theory:

**Lemma 3.3.2**. Let $G = \langle W, R \rangle$ be a directed graph encoded by lists of neighbours and $X \subseteq W$. Computing the

set $\bigcup\{R^+(r) : r \in X\}$ can be done in time $\mathcal{O}(\text{card}(W) + \text{card}(R))$ where $R^+$ is the transitive closure of $R$ (smallest transitive relation containing $R$).

Observe that $\mathcal{M}'$ can be also computed in linear-time in $|\mathcal{M}|$.

**Case 6:** $\phi_i = \text{EG}\phi_{i_1}$ for some $i_1 < i$.

As in the previous case, we define

$$W' \stackrel{\text{def}}{=} \{s \in W : \phi_{i_1} \in l(s)\}.$$

Let $\mathcal{M}' = \langle W', R', L' \rangle$ be the restriction of $\mathcal{M}$ to $W'$. We show that for every $s \in W$, $\mathcal{M}, s \models \phi_i$ ssi

(I) $s \in W'$ and

(II) there is a finite path in $\mathcal{M}'$ from $s$ to a state $s'$ that belong to a non-trivial strongly connected component (SCC) for $\langle W', R' \rangle$.

A non-trivial SCC $C$ for $\langle W', R' \rangle$ is a subset of $W'$ such that

1. for all $s' \neq s'' \in C$, there is an $R'$-path from $s'$ to $s''$ and an $R'$-path from $s''$ to $s'$ ($C$ strongly connected),
2. either $\text{card}(C) > 1$ or ($C = \{s'\}$ and $\langle s', s' \rangle \in R'$) ($C$ non-trivial).

Suppose that $\mathcal{M}, s \models \text{EG}\phi_{i_1}$. Obviously, $s \in W'$. Hence, there is an infinite path $s_0, s_1, \ldots$ such that $s_0 = s$ and for every $j \geq 0$, $\mathcal{M}, s_j \models \phi_{i_1}$. Since the path is infinite, there is $n \geq 0$ such that for every $j \geq n$, $s_j$ occurs infinitely often in $s_n, s_{n+1}, \ldots$. The states in $s_0, \ldots, s_{n-1}$ (if $n = 0$, this is

the empty sequence), belong to $W'$.

Let $C$ be the set of states occurring in $s_n, s_{n+1}, \ldots$. We can show that $C$ is a non-trivial SCC. If $C$ is a singleton, then the proof is immediate. Otherwise, for all $s$ and $s'$ in $C$, there is an $R'$-path between $s$ and $s'$ and there is an $R'$-path between $s'$ and $s$. Indeed, $s$ and $s'$ occur infinitely often in $s_n, s_{n+1}, \ldots$.

Now, let us suppose that (I) and (II) hold true. Let $p_1$ be a finite $R'$-path between $s$ and $s'$. Let $p_2$ be a finite $R'$-path between $s'$ and the length of $s'$ is at least one (which is possible to find since $C$ is a non-trivial SCC). All the states in the path $p_1 p_2^\omega$ (i.e. $p_1 p_2 p_2 p_2 \ldots$) satisfy $\phi_{i_1}$ (use of the induction hypothesis for the correctness). Since $p_1 p_2^\omega$ is also an $R$-path starting from $s$, we get $\mathcal{M}, s \models \phi_i$.

Now we can conclude the proof. Let $C_1, \ldots, C_n$ be a partition of $W'$ such that each $C_i$ is a maximal SCC. Maximality is defined with respect to set inclusion. In order to show that each step can be computed in time $\mathcal{O}(\text{card}(W) + \text{card}(R))$, we need to use the following result from graph theory.

**Lemma 3.3.3**. Let $G = \langle W, R \rangle$ be a directed graph represented by lists of neighbours. Computing the partition of maximal SCC can be computed in time $\mathcal{O}(\text{card}(W) + \text{card}(R))$.

We advise to see [AHU74, AHU83] to find the proofs of

Lemmas 3.3.2 & 3.3.3.

We write $W''$ to denote the set of states belonging to some non-trivial maximal SCC of $\langle W', R' \rangle$. For every $s \in W$, if $s \in W'$ and there is $s' \in W''$ such that $s' \in R'^*(s)$ then insert $\phi_i$ in $l(s)$ otherwise insert $\neg\phi_i$ in $l(s)$. As in the previous case, this step can computed in time $\mathcal{O}(\text{card}(W) + \text{card}(R))$.

**Corollary 3.3.4**. MC(CTL) is in P.

P-hardness of MC(CTL) can be easily shown be reducing satisfiability for synchronized alternating monotonous Boolean circuits that is a P-complete problem, see e.g. [Sch01].

**Exercise 3.3.1**. With the current CTL definition, the operators EX, EG and EU are primitive whereas binary AU can be defined by:

$$\mathrm{A}(\phi_1 \mathrm{U} \phi_2) \stackrel{\text{def}}{=} \neg(\mathrm{E}(\neg\phi_2 \mathrm{U}(\neg\phi_1 \wedge \neg\phi_2)) \vee \mathrm{EG}\neg\phi_2)).$$

Show that MC(CTL + AU) is in P by adding one case in the proof of Proposition 3.3.1 (a naive translation would exponentially blow up the size of formulae).

# Chapter 4

# Extended LTL with finite-state automaton operators

## 4.1 Definition and expressive power

We have seen how to build a Büchi automaton $\mathcal{A}_\phi$ such that $\mathrm{L}(\mathcal{A}_\phi)$ is equal to the set of models for $\phi$ (in LTL). However, it is natural to wonder whether any Büchi automaton on the alphabet $\mathcal{P}(\mathrm{PROP}(\phi))$ corresponds to an LTL formula. For instance, is there an LTL formula, counterpart of the automaton presented in Figure 4.1?

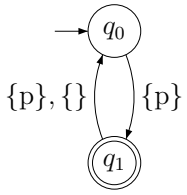



Figure 4.1: Büchi automaton over the alphabet $\{\{\mathrm{p}\}, \{\}\}$.

By Kamp's theorem [Kam68] LTL with past-time operators has the same expressive power as first-order theory of $\langle \mathbb{N}, < \rangle$. Moreover, thanks to the separation theorem, LTL and first-order logic over $\langle \mathbb{N}, < \rangle$ are initially equivalent. However, there exists quite simple properties that cannot be expressed with LTL.

**Proposition 4.1.1**. [Wol83] There is no LTL formula $\phi$ built over the unique propositional variable p such that Models($\phi$) is exactly the set of LTL models such that p holds true on every even position (on odd positions, p may hold true or not).

If we relax the condition on the number of variables (which corresponds to extend the alphabet) the following formula is then useful:

$$\mathrm{q} \wedge \mathrm{X}\neg\mathrm{q} \wedge \mathrm{G}(\mathrm{q} \Leftrightarrow \mathrm{XXq}) \wedge \mathrm{G}(\mathrm{q} \Rightarrow \mathrm{p}).$$

That is why, in [Wol83], an extension of LTL has been introduced by adding temporal operators defined from finite-state automata. Alternatively, right-linear grammars can also be used to define regular languages.

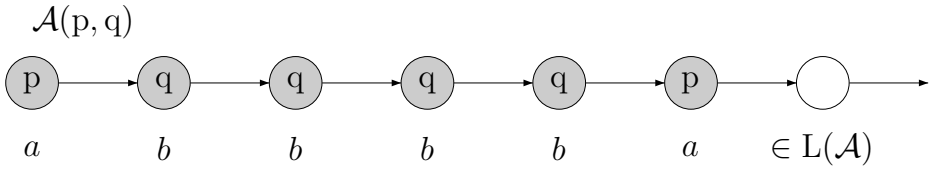
Let $\mathcal{A} = \langle \Sigma, S, S_0, \rho, F \rangle$ be a finite-state automaton with letters from $\Sigma$ linearly ordered, providing $a_1 < \ldots < a_k$. The extension $\mathrm{ETL}_f$ of LTL ("f' stands for "finite words") consists in considering formulae of the form $\mathcal{A}(\phi_1, \ldots, \phi_k)$ in a recursive fashion (and to get rid of the temporal operators X, U and F). The relation $\sigma, i \models \mathcal{A}(\phi_1, \ldots, \phi_k)$ holds true when a finite pattern induced from $\mathrm{L}(\mathcal{A})$ exists from position $i$. There is a correspondence between the letters $a_1, \ldots, a_k$ and the arguments $\phi_1, \ldots, \phi_k$. More precisely,

- $\sigma, i \models \mathcal{A}(\phi_1, \ldots, \phi_k) \stackrel{\text{def}}{\Leftrightarrow}$

– either $S_0 \cap F \neq \emptyset$ ($\epsilon \in \mathrm{L}(\mathcal{A})$),
– or there is a finite word $a_{i_1} a_{i_2} \ldots a_{i_n} \in \mathrm{L}(\mathcal{A})$ such that for every $1 \leq j \leq n$, $\sigma, i + (j-1) \models \phi_{i_j}$.

If $S_0 \cap F \neq \emptyset$, then $\mathcal{A}(\phi_1, \ldots, \phi_k)$ is equivalent to $\top$.

We show below a model for $\mathcal{A}(\mathrm{p}, \mathrm{q})$ with $\mathrm{L}(\mathcal{A}) = \{ab^i a : i \geq 0\}$ and $a < b$:




**Exercise 4.1.1**. We write $\mathrm{ETL}_r$ to denote the $\mathrm{ETL}_f$ variant that consists in considering infinite words ("r" stands for "repeating"). Typically, $\sigma, i \models \mathcal{A}(\phi_1, \ldots, \phi_k) \stackrel{\text{def}}{\Leftrightarrow}$ there is an infinite word $a_{i_1} a_{i_2} \ldots$ accepted by $\mathcal{A}$ such that for every $j \geq 1$, $\sigma, i + (j-1) \models \phi_{i_j}$. Show that there is a logarithmic space reduction from $\mathrm{SAT}(\mathrm{ETL}_f)$ to $\mathrm{SAT}(\mathrm{ETL}_r)$.

**Proposition 4.1.2**. [VW94]

**(I)** $\mathrm{MC}(\mathrm{ETL}_f)$, $\mathrm{MC}(\mathrm{ETL}_r)$, $\mathrm{SAT}(\mathrm{ETL}_f)$ and $\mathrm{SAT}(\mathrm{ETL}_r)$ are PSPACE-complete.

**(II)** $\mathrm{ETL}_f$ and $\mathrm{ETL}_r$ have the same expressive power as Büchi automata.

In particular, for any Büchi automaton $\mathcal{A}$ over the alphabet $\Sigma = \{a_1, \ldots, a_k\}$, for any map $l : \Sigma \to X$ where $X$ is a set of finite subsets of PROP, there is a formula $\phi$ in $\mathrm{ETL}_f$ built over the propositional variables in $\bigcup_i l(a_i)$ such that $\mathrm{L}(\mathcal{A}) = \mathrm{Models}(\phi)$.

**Expressive power** Hence, the class of languages defined by $\mathrm{ETL}_f$ formulae is equal to the class of languages defined by

- Büchi automata (Proposition 4.1.2),
- formulae from monadic second-order theory for $\langle \omega, < \rangle$, also known as S1S,
- $\omega$-regular expressions (or by finite union of sets $U \cdot V^\omega$ with regular $U, V \subseteq \Sigma^*$),
- formulae from LTL with second-order quantification. Given LTL models $\sigma, \sigma' : \mathbb{N} \to \mathcal{P}(\mathrm{PROP})$ and $\mathrm{p} \in \mathrm{PROP}$, we write $\sigma \approx_{\mathrm{p}} \sigma'$ to denote that for every $i \in \mathbb{N}$, $\sigma(i) \setminus \{\mathrm{p}\} = \sigma'(i) \setminus \{\mathrm{p}\}$. If $\sigma \approx_{\mathrm{p}} \sigma'$, then $\sigma$ and $\sigma'$ are equal modulo the interpretation of the propositional variable p. LTL$\forall$ (LTL with second-order quantification) is defined as an extension of LTL by allowing formulae of the form $\forall\ \mathrm{p}\ \phi$ with the following extended semantics: $\sigma, i \models \forall\ \mathrm{p}\ \phi \overset{\text{def}}{\Leftrightarrow}$ for every $\sigma'$ such that $\sigma \approx_{\mathrm{p}} \sigma'$, $\sigma', i \models \phi$. $\exists\ \mathrm{p}\ \phi$ is often used instead of $\neg\forall\ \mathrm{p}\ \neg\phi$.
- formulae from LTL with fixed-point operators [Var88].

**Conciseness** So, $\mathrm{ETL}_f$ is a powerful extension of LTL but this equivalence does not mean that all the above formalisms have the same conciseness. Actually,

- the emptiness problem for Büchi automata is NLOGSPACE-complete,
- $\mathrm{MC}^{\exists}(\mathrm{ETL}_f)$ and $\mathrm{SAT}(\mathrm{ETL}_f)$ are PSPACE-complete,
- satisfiability for LTL with fixed-point operators is PSPACE-complete [Var88],

- satisfiability for S1S is non-elementary (time complexity is not bounded by any tower of exponential of fixed height).

So, S1S is the most concise language for describing $\omega$-regular languages.

**Exercise 4.1.2**. Define a formula $\phi$ in $\mathrm{ETL}_f$ built over the unique propositional variable p such that Models($\phi$) is the set of LTL models such that p holds true on every even position.

**Exercise 4.1.3**. Define the operators U and X in $\mathrm{ETL}_f$.

## 4.2 Context-free extension

It is possible to extend the definition of $\mathrm{ETL}_f$ by replacing formulae of the form $\mathcal{A}(\phi_1, \ldots, \phi_n)$ by formulae of the form $\mathrm{L}(\phi_1, \ldots, \phi_n)$ where L is a language of finite words specified within a fixed formalism (L viewed as a set of patterns).

For a class $\mathcal{C}$ of languages, we write LTL + $\mathcal{C}$ to denote the extension of LTL to formulae of the form $\mathrm{L}(\phi_1, \ldots, \phi_n)$ for some $\mathrm{L} \in \mathcal{C}$ (this time X and U belong to LTL + $\mathcal{C}$).

Obviously, $\mathrm{ETL}_f$ is precisely LTL + REG where REG is the class of regular languages represented by finite-state automata.

$\mathrm{ETL}_f$ is decidable and it is natural to wonder whether LTL + CF where CF is the class of context-free languages (represented by context-free grammars) is decidable.

### 4.2.1 Undecidability of LTL + CF

Since numerous problems for context-free languages are undecidable, it is not very surprising to get the following result.

**Proposition 4.2.1**. SAT(LTL + CF) is undecidable.

**Proof.** It is possible to encode the undecidable equality problem between context-free grammars into SAT(LTL + CF).

Let $G_1$ and $G_2$ be two CF grammars over the terminal alphabet $\Sigma = \{a_1, \ldots, a_n\}$.

We write $G_1^+$ and $G_2^+$ the CF grammars over the terminal alphabet $\Sigma^+ = \{a_1, \ldots, a_n, a_{n+1}\}$ such that

$$\mathrm{L}(G_1^+) = \mathrm{L}(G_1) \cdot \{a_{n+1}\} \text{ et } \mathrm{L}(G_2^+) = \mathrm{L}(G_2) \cdot \{a_{n+1}\}.$$

The letter $a_{n+1}$ is simply an end marker. We have $\mathrm{L}(G_1) = \mathrm{L}(G_2)$ iff $\mathrm{L}(G_1^+) = \mathrm{L}(G_2^+)$. $G_1^+$ and $G_2^+$ can be effectively computed from $G_1$ and $G_2$, respectively.

We shall construct a formula $\phi_{G_1,G_2}$ from LTL + CF such that $\phi_{G_1,G_2}$ is satisfiable in LTL + CF iff $\mathrm{L}(G_1) \neq \mathrm{L}(G_2)$.

The formula $\phi_{G_1,G_2}$ is built from the propositional variables $\mathrm{p}_1, \ldots, \mathrm{p}_{n+1}$ and we consider the structures for which exactly one variable from $\mathrm{p}_1, \ldots, \mathrm{p}_{n+1}$ holds true at each state and, $\mathrm{p}_{n+1}$ holds true at a unique state of the model. Hence, we characterize structures that can be naturally viewed as finite words (possibly in $\mathrm{L}(G_1^+) \cup \mathrm{L}(G_2^+)$).

Let UNI be a formula expressing these properties:

$$\mathrm{UNI} \stackrel{\text{def}}{=} \mathrm{G}(\bigvee_{1\leq i\leq n+1} \mathrm{p}_i) \wedge \mathrm{G}(\bigwedge_{1\leq i\leq n+1} (\mathrm{p}_i \Rightarrow \bigwedge_{1\leq j\neq i\leq n+1} \neg \mathrm{p}_j) \wedge$$

$$((\mathrm{p}_{n+1} \wedge \mathrm{XG}\neg \mathrm{p}_{n+1}) \vee \neg \mathrm{p}_{n+1} \mathrm{U} (\mathrm{p}_{n+1} \wedge \mathrm{XG}\neg \mathrm{p}_{n+1}))$$

We can show the equivalence between the following propositions:

- $\mathrm{L}(G_1^+) \neq \mathrm{L}(G_2^+)$,
- $\mathrm{UNI} \wedge \neg(\mathrm{L}(G_1^+)(\mathrm{p}_1, \ldots, \mathrm{p}_{n+1}) \Leftrightarrow \mathrm{L}(G_2^+)(\mathrm{p}_1, \ldots, \mathrm{p}_{n+1}))$ is satisfiable.

Indeed, if $\mathrm{L}(G_1^+) \neq \mathrm{L}(G_2^+)$, say $a_{i_1} a_{i_2} \cdots a_{i_l} a_{n+1} \in \mathrm{L}(G_1^+)$ and $a_{i_1} a_{i_2} \cdots a_{i_l} a_{n+1} \notin \mathrm{L}(G_2^+)$.

Without any loss of generality, we can assume that $l \geq 1$. Let $\sigma$ be the model

$$\{\mathrm{p}_{i_1}\} \cdot \{\mathrm{p}_{i_2}\} \cdots \{\mathrm{p}_{i_l}\} \cdot \{\mathrm{p}_{n+1}\} \cdot \{\mathrm{p}_1\}^\omega.$$

We have

- $\sigma \models \mathrm{UNI}$,
- $\sigma \models \mathrm{L}(G_1^+)(\mathrm{p}_1, \ldots, \mathrm{p}_{n+1})$ and,
- $\sigma \not\models \mathrm{L}(G_2^+)(\mathrm{p}_1, \ldots, \mathrm{p}_{n+1})$ since the only finite word ending by $\{\mathrm{p}_{n+1}\}$ in $\sigma$ is $\{\mathrm{p}_{i_1}\} \cdot \{\mathrm{p}_{i_2}\} \cdots \{\mathrm{p}_{i_l}\} \cdot \{\mathrm{p}_{n+1}\}$ and $a_{i_1} a_{i_2} \cdots a_{i_l} a_{n+1} \notin \mathrm{L}(G_2^+)$.

Now suppose that $\sigma, 0 \models \mathrm{UNI} \wedge \neg(\mathrm{L}(G_1^+)(\mathrm{p}_1, \ldots, \mathrm{p}_{n+1}) \Leftrightarrow \mathrm{L}(G_2^+)(\mathrm{p}_1, \ldots, \mathrm{p}_{n+1}))$ for some model $\sigma$.

By assuming that $\sigma \models \mathrm{L}(G_1^+)(\mathrm{p}_1, \ldots, \mathrm{p}_{n+1})$ and $\sigma \not\models \mathrm{L}(G_2^+)(\mathrm{p}_1, \ldots, \mathrm{p}_{n+1})$, a simple reasoning allows to deduce that $\mathrm{L}(G_1^+) \neq \mathrm{L}(G_2^+)$.

Proposition 4.2.1 is interesting, but after all, it rests on the fact that LTL + CF can easily encode equivalences between CF grammars. It does not say much about the interplay between temporal operators X and U and formulae of the form $\mathrm{L}(\phi_1, \ldots, \phi_n)$.

For instance, is there a context-free language L such that SAT(LTL + {L}) is undecidable?

Obviously, the proof of Proposition 4.2.1 cannot be adapted to this restricted language.

### 4.2.2 Undecidability of LTL + $\mathrm{L}_0$

Actually, we shall answer positively to the previous question. Let $\mathrm{L}_0$ be the language $\{a_1^k \cdot a_2 \cdot a_1^{k-1} \cdot a_3 : k \geq 1\}$.

**Exercise 4.2.1**. Define a linear grammar (at most one non-terminal symbol in right-hand sides of production rules) for $\mathrm{L}_0$.

We write $\mathrm{L}_1$ to denote the language $\{a_1^k \cdot a_2 \cdot a_1^k \cdot a_3 : k \geq 0\}$ ($\mathrm{L}_0 = \{a_1\} \cdot \mathrm{L}_1$).

**Exercise 4.2.2**. Show the validity in LTL + $\{\mathrm{L}_0, \mathrm{L}_1\}$ of the following formula:

$$\mathrm{L}_1(\mathrm{p}, \mathrm{q}, \mathrm{r}) \Leftrightarrow (\mathrm{q} \wedge \mathrm{Xr}) \vee \mathrm{L}_0(\mathrm{p}, \mathrm{q}, \mathrm{p} \wedge \mathrm{Xr}).$$

Conclude that SAT(LTL + $\{\mathrm{L}_0, \mathrm{L}_1\}$) is undecidable iff SAT(LTL + $\{\mathrm{L}_0\}$) is undecidable.

We can also show that F can be defined with $\mathrm{L}_1$: $\mathrm{F}\phi$ is equivalent to $\mathrm{L}_1(\top, \phi, \top)$. So, we shall indeed show the undecidability

of SAT(LTL(X) + $L_0$) and $MC^{\exists}$(LTL(X) + $L_0$).

**Proposition 4.2.2**. SAT(LTL + $\{L_0\}$) is undecidable.

**Proof.** The domino problem DOMREC is defined as follows and is known to be $\Sigma_1^1$-complete (see e.g. [Har85]).

**input:** a domino game $Dom$ with a distinguished color $c$.

**output:** 1, if $Dom$ can pave $\mathbb{N} \times \mathbb{N}$ where the color $c$ occurs infinitely often.

We reduce DOMREC to SAT(LTL + $\{L_0, L_1\}$). Let $Dom = \langle C, D, Col \rangle$ be a domino game with $C = \{1, \ldots, n\}$, $D = \{d_1, \ldots, d_m\}$, and $c = 1$. So we have an instance of DOMREC. We recall that the map $Col$ is of the form

$$Col : D \times \{up, down, left, right\} \to \{1, \ldots, n\}.$$

We use the following propositional variables:

- $in$ is a propositional variable that holds true when the state encodes a position in $\mathbb{N}^2$. Indeed, there are states in the model that do not correspond to positions in $\mathbb{N}^2$. In order to facilitate the presentation, we also introduce $out$ that is equivalent to the negation of $in$.
- For every $1 \leq j \leq m$, we introduce the variable $\boxed{\text{j}}$ with intended meaning that "the position in $\mathbb{N}^2$ associated to the current state is occupied by the domino type $d_j$".
- For every $1 \leq i \leq n$, we use the variables $up_i$, $down_i$, $left_i$, $right_i$. For instance, $up_1$ holds true whenever the domino on the position associated to the current state has color 1 on its top.

Every state encoding a position in $\mathbb{N}^2$ is occupied by a unique domino:

$$\mathrm{G}(in \Rightarrow \bigvee_{j=1}^{m}(\boxed{\mathrm{j}} \wedge \bigwedge_{j'=1,j'\neq j}^{m} \neg\boxed{\mathrm{j'}}))$$

Propositional variables for colours are compatible with the definition of domino types:

$$\mathrm{G}(in \Rightarrow \bigwedge_{j=1}^{m} \boxed{\mathrm{j}} \implies$$

$$\bigwedge_{side\in\{up,down,right,left\}} side_{Col(d_j,side)} \wedge \bigwedge_{1\leq j'\neq Col(d_j,side)\leq n} \neg side_{j'})$$

We write PAVE to denote the conjunction of the above formulae.

Now, we shall define the states of the model that correspond to a position in $\mathbb{N}^2$.

We write SNAKE to denote the conjunction of the following formulae:

- $\mathrm{G}(in \Leftrightarrow \neg out)$,
- $in \wedge \mathrm{X}out \wedge \mathrm{XX}in \wedge \mathrm{XXX}in \wedge \mathrm{XXXX}out$,
- $\mathrm{G}(out \Rightarrow \mathrm{XL}_1(in, out, in \wedge \mathrm{X}out))$.
  $(\mathrm{L}_1 = \{a_1^k \cdot a_2 \cdot a_1^k \cdot a_3 : k \geq 0\})$.

The only structure (built over *in* and *out*) satisfying SNAKE is:

$$\{in\} \cdot \{out\} \cdot \{in\}^2 \cdot \{out\} \cdot \{in\}^3 \cdot \{out\} \cdot \{in\}^4 \ldots$$

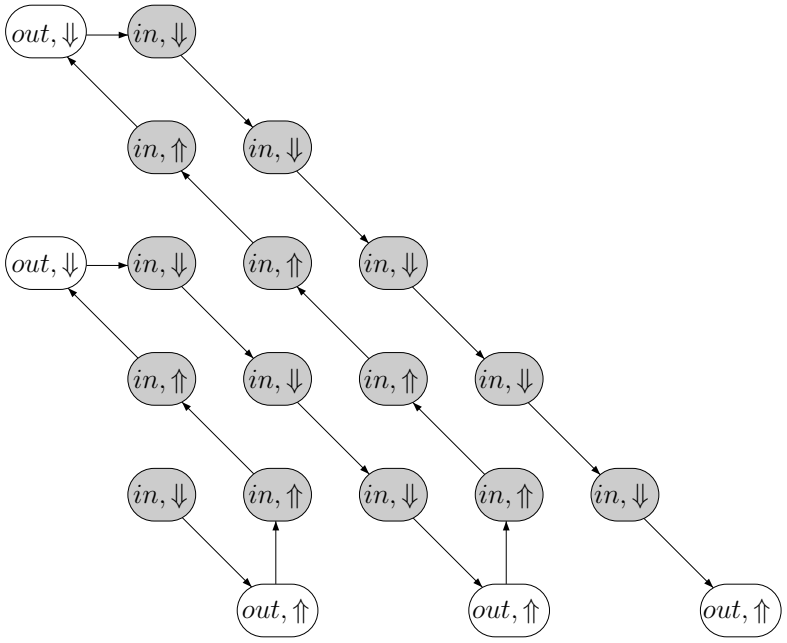



Figure 4.2: The path in $\mathbb{N}^2$

This sequence makes reference to the path presented in Figure 4.2.

The difficulty of the proof is not to design a path through $\mathbb{N}^2$ but rather to define a path on which it is easy to access to neighbours (right or top).

For every state *in*, we need to remember if it occurs in a sequence of *in* that is ascendant or descendant. Indeed, this criterion is relevant to access to (right or top) neighbours.

We introduce the variables $\Uparrow$ and $\Downarrow$. The former one $\Downarrow$ is only introduced to facilitate the presentation. We write DIRECTION to denote the conjunction of the following formulae:

- $\mathrm{G}(\Uparrow \Leftrightarrow \neg \Downarrow)$,
- $\Downarrow \wedge \mathrm{X} \Uparrow$,
- $\mathrm{G}(in \wedge \mathrm{X}in \wedge \Uparrow \Rightarrow \mathrm{X} \Uparrow)$ ("we stay on ascending chain"),

- G($in \wedge \mathrm{X}in \wedge \Downarrow \Rightarrow \mathrm{X} \Downarrow$) ("we stay on descending chain"),
- G($in \wedge \mathrm{X}out \wedge \Uparrow \Rightarrow (\mathrm{X} \Downarrow \wedge \mathrm{XX} \Downarrow)$) ("we pass from ascending to descending chain"),
- G($in \wedge \mathrm{X}out \wedge \Downarrow \Rightarrow (\mathrm{X} \Uparrow \wedge \mathrm{XX} \Uparrow)$) ("we pass from descending to ascending chain").

The only structure (built over $in$, $out$, $\Uparrow$ and $\Downarrow$) satisfying SNAKE $\wedge$ DIRECTION is:

$$\{in, \Downarrow\}\{out, \Uparrow\} \cdot \{in, \Uparrow\}^2 \cdot \{out, \Downarrow\} \cdot \{in, \Downarrow\}^3 \cdot \{out, \Uparrow\} \cdot \{in, \Uparrow\}^4 \ldots$$

This structure encode the path through $\mathbb{N}^2$ described in Figure 4.2.

The path allows to access to adjacent states as follows:

- in a state $\{in, \Uparrow\}$, we access to the right neighbour with the help of $\mathrm{L}_1$,
- in a state $\{in, \Uparrow\}$, we access to the up neighbour with the help of $\mathrm{L}_0$,
- in a state $\{in, \Downarrow\}$, we access to the right neighbour with the help of $\mathrm{L}_0$,
- in a state $\{in, \Downarrow\}$, we access to the up neighbour with the help of $\mathrm{L}_1$.

We write CONSTRAINTS to denote the conjunction of following formulae that expresses color constraints for adjacent dominoes:

- G($in \wedge \Uparrow \Rightarrow (\bigwedge_{1 \leq i \leq n} right_i \Rightarrow \mathrm{L}_1(in, out, left_i))$),

- $\mathrm{G}(in \wedge \Uparrow \Rightarrow (\bigwedge_{1 \leq i \leq n} up_i \Rightarrow \mathrm{L}_0(in, out, down_i)))$,
- $\mathrm{G}(in \wedge \Downarrow \Rightarrow (\bigwedge_{1 \leq i \leq n} right_i \Rightarrow \mathrm{L}_0(in, out, left_i)))$,
- $\mathrm{G}(in \wedge \Downarrow \Rightarrow (\bigwedge_{1 \leq i \leq n} up_i \Rightarrow \mathrm{L}_1(in, out, down_i)))$.

We write REC to denote the formula that states that the colour 1 occurs infinitely often:

$$\mathrm{GF}(in \wedge \bigvee_{side \in \{left, right, up, down\}} side_1).$$

The domino game *Dom* can pave $\mathbb{N}^2$ by repeating infinitely often the colour 1 iff

$$\mathrm{PAVE} \wedge \mathrm{SNAKE} \wedge \mathrm{DIRECTION} \wedge \mathrm{CONSTRAINTS} \wedge \mathrm{REC}$$

is satisfiable in LTL + $\{\mathrm{L}_0, \mathrm{L}_1\}$.

SAT(LTL + $\{\mathrm{L}_0\}$) is therefore $\Sigma_1^1$-hardness and is not recursively enumerable.

The proof of Proposition 4.2.2 is inspired from the undecidability of propositional dynamic logic (PDL) augmented with the CF language $\{a_1^n \cdot a_2 \cdot a_1^n : n \geq 0\}$ (see for example [HKT00, chapitre 9] for more details).

For formal verification, the following result is more interesting:

**Corollary 4.2.3**. $\mathrm{MC}^{\exists}(\mathrm{LTL(X)} + \{\mathrm{L}_0\})$ is undecidable.

**Exercise 4.2.3**. Complete the proof of Corollary 4.2.3 by reducing SAT(LTL(X) + $\{\mathrm{L}_0\}$) to $\mathrm{MC}^{\exists}$(LTL(X) + $\{\mathrm{L}_0\}$) (exponential-time reduction).

**Exercise 4.2.4**. Show that the model-checking problem for the propositional calculus augmented with the unique temporal operator $\mathrm{L}_1$ is undecidable.

**Exercise 4.2.5**. Let L be a language of finite words such that MC(LTL $+\{\mathrm{L}\}$) is undecidable and $\mathrm{L}'$ be a finite language. Show that MC(LTL $+\{(\mathrm{L} \setminus \mathrm{L}')\}$) is also undecidable.

**Exercise 4.2.6**. Let $\langle \Sigma, S, S_0, \rho, F \rangle$ be a deterministic finite-state automaton. We write $min(\mathcal{A})$ to denote the language

$$\{w \in \mathrm{L}(\mathcal{A}) : \forall v \in \mathrm{L}(\mathcal{A}),\ v \leq w \text{ implies } v = w\}$$

where $\leq$ is the prefix relation.

1. Build an automaton $\mathcal{A}'$ such that $\mathrm{L}(\mathcal{A}') = min(\mathcal{A})$, justify the correctness of the construction and provide an example of automaton $\mathcal{A}$ such that $min(\mathcal{A})$ is infinite.
2. Suppose that $|\Sigma| = n \geq 1$. Show that for all formula $\phi_1, \ldots, \phi_n \in \mathrm{ETL}_f$, $\mathcal{A}(\phi_1, \ldots, \phi_n) \Leftrightarrow \mathcal{A}'(\phi_1, \ldots, \phi_n)$ is valid formula in $\mathrm{ETL}_f$.
3. Now, suppose that $min(\mathcal{A})$ is finite. Define a translation $t$ from formulae built over the operators $\neg, \wedge$ and $\mathcal{A}$ (fragment denoted by PC + $\mathcal{A}$) into LTL(X) such that for every total and finite Kripke structure $\mathcal{M} = \langle W, R, L \rangle$ and for every $s \in W$, $\mathcal{M}, s \models_\exists \phi$ iff $\mathcal{M}, s \models_\exists t(\phi)$.
4. Evaluate the size of $t(\phi)$ in terms of the size of $\phi$ and the number of subformulae of $t(\phi)$ in terms of the number of subformulae of $\phi$.
5. Show that for every deterministic finite-state automaton $\mathcal{A}$ such that $min(\mathcal{A})$ is finite, MC(PC + $\mathcal{A}$) is in NP.

# Bibliography


[AHU74] A. Aho, J. Hopcroft, and J. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, Reading, MA, 1974.

[AHU83] A. Aho, J. Hopcroft, and J. Ullman. *Data Structures and Algorithms*. Addison-Wesley, Reading, MA, 1983.

[All83] J. Allen. Maintaining knowledge abut temporal intervals. *Communications of the ACM*, 26(11):832–843, 1983.

[BBF+01] B. Bérard, M. Bidoit, A. Finkel, F. Laroussinie, A. Petit, L. Petrucci, and Ph. Schnoebelen. *Systems and Software Verification, Model-Checking Techniques and Tools*. Springer-Verlag, 2001.

[CC00] H Comon and V. Cortier. Flatness is not a weakness. In *14 Int. Workshop Computer Science Logic*, volume 1862 of *Lecture Notes in Computer Science*, pages 262–276. Springer, 2000.

[EH85] A. Emerson and J. Halpern. Decision procedures and expressiveness in the temporal logic of branching time. *Journal of Computer and System Sciences*, 30:1–24, 1985.

[EH86] A. Emerson and J. Halpern. "sometimes" and "not never" revisited: on branching versus linear time temporal logic. *Journal of the Association for Computing Machinery*, 33:151–178, 1986.

[Gor00] V. Goranko. Temporal logics of computations, 2000. Lecture Notes from ESSLLI'2000. 83 pages.

[Har85] D. Harel. Recurring dominoes: making the highly undecidable highly understandable. *Annals of Discrete Mathematics*, 24:51–72, 1985.

[HKT00] D. Harel, D. Kozen, and J. Tiuryn. *Dynamic Logic*. MIT Press, 2000.

[Kam68] J. Kamp. *Tense Logic and the theory of linear order*. PhD thesis, UCLA, USA, 1968.

[Koy90] R. Koymans. Specifying real-time properties with metric temporal logic. *Real-Time Systems*, 2(4):255–299, 1990.

[Pap94] Ch. Papadimitriou. *Computational Complexity*. Addison-Wesley, Reading, MA, 1994.

[Pnu77] A. Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science*, pages 46–57. IEEE Computer Society Press, 1977.

[Pri67] A. Prior. *Past, Present and Future*. Oxford University Press, 1967.

[Ros91] R. Rosner. *Modular synthesis of reative systems*. PhD thesis, the Weizmann Institute of Science, Rehovot, Israel, February 1991.

[SC85] A. Sistla and E. Clarke. The complexity of propositional linear temporal logic. *Journal of the Association for Computing Machinery*, 32(3):733–749, 1985.

[Sch01] Ph. Schnoebelen. Spécification et vérification des systèmes concurrents, October 2001. Habilitation thesis.

[SM73] L. Stockmeyer and A. Meyer. Word problems requiring exponential-time. In *5th ACM Symposium on the Theory of Computing*, pages 1–9, 1973.

[Var88] M. Vardi. A temporal fixpoint calculus. In *15th Annual ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages, San Diego*, pages 250–259. ACM, 1988.

[VW94] M. Vardi and P. Wolper. Reasoning about infinite computations. *Information and Computation*, 115:1–37, 1994.

[Wol83] P. Wolper. Temporal logic can be more expressive. *Information and Control*, 56:72–99, 1983.