FIRST PhD Autumn School

on Modal Logic

IT University of Copenhagen, Denmark

November 10-11 2009

Tutorial notes on

Introduction to Temporal Logics for Specification and Verification

Valentin Goranko

Technical University of Denmark

Contents

Introduction

Temporal reasoning stems from philosophical analysis of time and temporality, initiated in the Antiquity by Diodorus Chronos and Aristotle, but first formalized in logical systems by Arthur Prior in his famous book "Past, Present and Future" [Pri67].

On the other hand, temporal aspects and phenomena are pervasive in computer and information systems, for instance: scheduling of the execution of programs by an operating system; specification and verification of concurrent and reactive systems; in particular, synchronization of concurrent processes; real-time processes and systems; hardware architecture and verification; temporal databases, etc. Many of these are related to specification and verification of properties of *transition systems* and *computations* in them.

In the seminal paper [Pnu77a] Amir Pnueli proposed the use of temporal logic for specification and verification of important properties of reactive and concurrent systems, such as safety, liveness, fairness, etc. Since then temporal logics have found numerous other applications in computer science and artificial intelligence, but the one proposed by Pnueli, and further developed by him and Manna in [MP79], [MP81], [MP92], [MP95] has been the most popular and successful so far. The reason for that success is that temporal logics provide a very natural logical framework for formal specification and verification of properties of transition systems, as these logics are syntactically simple and elegant, have natural semantics, strong expressive power, and good computational behavior. Depending on the type of systems and properties to specify and verify, two major families of temporal logics have been developed: *linear-time and branching-time logics*. Major applications of temporal logics are the formal *logical derivation* of properties of systems expressed by temporal formulae, from their specifications formalized as a system of temporal axioms, and the *satisfiability testing* of temporal formulae, which corresponds to formal verification of the realizability of system specifications expressed by these formulae.

Two major developments, both starting in 1980's, contributed strongly to the popularity and success of temporal logics in computer science. The first one is the advancement of *model checking* as method for formal verification by Clarke and Emerson in [CE81], followed by [CES83], [CES86], and independently by Queile and Sifakis in [QS82]. The second one is the emergence of *automata-based methods* for verification, advocated in a series of papers by Sistla, Vardi and Wolper [SVW87], [VW86], and further developed in [VW94], [KVW00], etc.

Both the method of model checking and the automata-based techniques for verification work extremely well for properties specified in temporal logics, and that has boosted the importance and popularity of temporal logics in the field of formal verification enormously. For instance, model checking of a property of a given transition system corresponds to checking whether the temporal formula expressing that property is true in the abstract model representing the system. This is particularly important when the property represents some unwanted behaviour which should never occur in the system. On the other hand, it turns out that satisfiability testing and model checking of temporal formulae can be uniformly reduced to testing language non-emptiness and language inclusion of automata on infinite words or trees associated with the formula (in the case of satisfiability testing), respectively the formula and the model (in the case of model checking).

In parallel with automata-based techniques, efficient and intuitively appealing *tableau-based methods* for satisfiability testing and model checking of temporal formulae have been developed since the early 1980s, by Wolper [Wol83], [Wol85] (for the linear-time logic LTL), Ben-Ari, Manna, Pnueli [BAPM81] (for the branching-time logic UB) and Emerson, Halpern in [EH82],[EH85] (for the branching-time logic CTL).

These notes introduce and discuss abstract transitions systems and computations in them, and then the standard linear and branching time temporal logics and illustrate how they can be used to specify properties of transition systems.

Chapter 1

Transition systems and computations. The basic modal logic for transition systems TL.

Transition systems consist of states and transitions between them. They are used to model and implement sequential and concurrent processes, which can be autonomous, reactive, or interactive. The states in a transition system can be thought of as program states, control states, configuration states, memory registers, etc. The actions can represent program instructions or even whole programs, autonomous processes, agents' actions, etc.

Physical examples of transition systems include clocks, vending machines, payphones, semaphores, lifts, etc. Abstract examples of transition systems include finite state machines (in particular, finite automata), as well as configuration graphs of various other abstract computing devices, such as pushdown automata, Turing machines, Petri nets, counter systems, timed automata, etc. In this course we will adopt an abstract view on transition systems.

1.1 Labeled transition systems

Transitions can be effected by different actions or processes, so we often distinguish different types of transitions, by assigning different labels to them. Thus, generally, we consider *labeled* transition systems. Here is the formal definition.

Definition 1.1.1. [**Labeled transition systems**]

A *labeled transition system (LTS)* is a structure

$$\mathcal{T} = \left\langle S, \left\{ \overset{a}{\longmapsto} \right\}_{a \in A} \right\rangle$$

consisting of:

- ⋆ a non-empty set S of *states*;
- ⋆ a non-empty set A (called the *signature* of $\mathcal{T}$) of *transitions*; each of them acts, possibly non-deterministically, on states and produce *successor states*;

⋆ a binary *transition relation* $\overset{a}{\longmapsto} \subseteq S \times S$ associated with every action $a \in A$.

  We write $s \overset{a}{\longmapsto} t$ to indicate that the action a can transform the state s into the state t and say that s is an *a-predecessor* of t, while t is an *a-successor* of s.

∇

In addition, sometimes an *initial state* (or, a set of initial states) is specified (see below).

When referring to transition systems where states have specific structure, such as states of pushdown automata, Turing machines, counter automata, Petri nets, etc., we often use the term '*configuration*' as a synonym of 'state'. Also, depending on the context, sometimes we talk about '*control states*' or '*locations*' instead of 'states', and about *actions*, or *processes*, instead of 'transitions'.

When a labelled transition system involves only one type of action, we call it a *(mono)-transition system.* Then we omit the label and typically denoted it by (S, R).

With every transition system $\mathcal{T} = \left\langle S, \left\{ \overset{a}{\longmapsto} \right\}_{a \in A} \right\rangle$ we can associate the *union transition*

$$R_{\mathcal{T}} = \bigcup \{ \overset{a}{\longmapsto} \mid a \in A \}.$$

Thus, for any states $s, t \in S$, $sR_{\mathcal{T}}t$ holds iff there is at least one transition that relates s to t.

Most of the concepts and results discussed here do not depend essentially on the specific signature of the transition system $\mathcal{T}$. So, often the labels can simply be ignored and we can assume that $\mathcal{T}$ is identified with the mono-transition system $\langle S, R_{\mathcal{T}} \rangle$. That is why, we will usually only consider the case of mono-transition systems and, when the signature is of no importance, we will often talk about *transition systems* simpliciter, abbreviated 'TS'.

Definition 1.1.2. [**Rooted transition systems**] A *rooted (or, initialized) transition system (RTS)* is a pair $(\mathcal{T}, r)$ where $\mathcal{T}$ is a TS and r is a distinguished state in $\mathcal{T}$, called the *root.* ∇

Intuitively, r is the initial state from which all computations which we consider begin.

A state may have various properties: it can be initial, terminal, accepting, deadlock, safe or unsafe, etc. We describe there properties of states by formulae of a suitable state-description language; on propositional level these can be indicated by special *atomic propositions.* We will call the set of such propositions that are declared true at a given state the *description* of that state. A transition system where every state is assigned such description will be called an *interpreted transition system.* Here is the formal definition.

Definition 1.1.3. [**Interpreted transition system**] *Interpreted transition system* (ITS) is a pair $\mathcal{M} = \langle \mathcal{T}, L \rangle$ where $\mathcal{T}$ is a transition system of some signature A, PROP is a fixed set of *atomic propositions*, and $L : S \to \mathbf{2}^{\text{PROP}}$ is a *state description* which assigns to every state s the set of atomic propositions true at s. ∇

The *type* of the ITS $\mathcal{M}$ is the pair (A, PROP). The type of $\mathcal{M}$ is finite if each of A and PROP is finite. Unless otherwise specified, we will only consider interpreted transition systems of finite types.

Rooted interpreted transition system (RITS) is defined accordingly. We will be denoting such systems by $(\mathcal{M}, r)$, or, with a slight abuse of notation, by $(\mathcal{T}, L, r)$.

NB. In terms of modal and temporal logics, transition systems are simply Kripke frames, and interpreted transition systems are Kripke models. However, we prefer to use the term transition system, to emphasize on the fact that this course is about applying temporal logics to reasoning about transition systems, not the other way around.

In logical terminology, the state description is usually called a *truth assignment.* Instead of truth assignments, *valuations* are often used in classical modal and temporal logics. A valuation $V : \mathrm{PROP} \to \mathbf{2}^S$ assigns to each atomic proposition from PROP the set of states where it is true. Clearly, the two formalisms are inter-definable and we will make use of both, for different purposes. For instance, we will use valuations in the context of *global model-checking*, where with every formula of the logic we associate (and compute) the set of states in the given transition system where that formula is true.

1.2 Paths and computations in transition systems

Definition 1.2.1. [**Paths in transition systems**] A *path (run, execution)* in a transition system $\mathcal{T}$ is a (finite or infinite) sequence of states and actions which transform every state into its successor:

$$s_0 \xmapsto{a_0} s_1 \xmapsto{a_1} s_2 \ldots$$

The path is said to be rooted at s_0. A path π consisting of n transitions is said to have *length* n, denoted $|\pi| = n$.

A path in a transition system is *maximal* if it is either infinite, or is finite and ends in a *deadlock state*, i.e., a state with no successors. Thus, in a transition system without deadlock states all paths are maximal. ∇

We sometimes use the terms '*run*' or '*execution*' as synonyms of 'path', for instance when referring to some special cases of transition systems, such as automata, Petri nets, etc.

Definition 1.2.2. [**Reachable states**] A state t is *reachable* from a state s in a transition system $\mathcal{T}$ if there is a path in $\mathcal{T}$ leading from s to t.

The set of all states in $\mathcal{T}$ reachable from s will be denoted by $post^*_{\mathcal{T}}(s)$. ∇

In a mono-transition system a path is simply a sequence of states $s_0, s_1, s_2, \ldots$, every one of which is related to its successor by the transition relation. Formally, a path in such a transition system can be defined as a mapping $\pi : \mathbb{N} \to S$, so we will often denote the successive states of a path π by $\pi(0), \pi(1), \pi(2), \ldots$.

Definition 1.2.3. A path in a transition system is a *cycle* if its first and its last state coincide; in particular, if it is a *loop*: $s \xmapsto{a} s$. ∇

Definition 1.2.4. A transition system is:

⋆ *acyclic* if it does not contain cycles;

⋆ *forest-like* if it is acyclic and every state has at most one predecessor state.

- ⋆ *tree-like* if it is a forest in which exactly one state, called *the root*, has no predecessor states.

∇

Definition 1.2.5. [**Computations**] A *computation*, or *trace*, in an interpreted transition system $(\mathcal{T}, L)$ is a (finite or infinite) sequence of state descriptions and respective actions along a path:

$$L(s_0) \xmapsto{a_0} L(s_1) \xmapsto{a_1} L(s_2) \ldots$$

∇

Thus, a computation, intuitively, is the *observable effect* (the 'trace') of a path in a transition system. It can be regarded as a record of all successive intermediate results of the computing process. The idea is that the information encoded by the state descriptions includes all that is essential in the computation, including the values of all important variables. That is why, sometimes we use the term '*trace*' as a synonym of 'computation'.

If a path or a computation is finite, it is also called *terminating*. For technical convenience we can always append to any finite path an infinite repetition of an *idle (terminating) state* and thus consider every path and computation infinite. Thus, hereafter, unless otherwise specified, we only consider the case when the (union) transition relation R is *serial*, or *total*, i.e., every state has at least one R-successor.

When the actions are not important, we will omit them from the description of paths and computations and will represent them simply as $s_0, s_1, s_2, \ldots$ and respectively $L(s_0), L(s_1), L(s_2), \ldots$.

Sometimes, when we are not interested in the specific path generating a given computation (or, when the transition system has only one path, as in $\langle \mathbb{N}, \mathsf{succ} \rangle$ where succ is the successor function on $\mathbb{N}$) we can bypass the notion of a path altogether and introduce the notion of *abstract computation* being simply a mapping $\sigma : \mathbb{N} \to 2^S$. Accordingly, we will denote the successive labels in a computation σ by $\sigma(0), \sigma(1), \sigma(2), \ldots$.

1.3 Unfoldings of transition systems. Execution trees.

The paths in a transition system are only present there implicitly. They can be made explicit by unfolding the transition system into a forest-like one, where paths coincide with branches.

Definition 1.3.1. [**Unfolding of a labelled transition system**] The unfolding of the labelled transition system $\mathcal{T} = \left\langle S, \left\{ \xmapsto{a} \right\}_{a \in A} \right\rangle$ is again a labelled transition system $\widehat{\mathcal{T}} = \left\langle \widehat{S}, \left\{ \xRightarrow{a} \right\}_{a \in A} \right\rangle$ where:

- ⋆ $\widehat{S}$ consists of all *finite paths* in $\mathcal{T}$, including all single states s, regarded as paths $\widehat{s}$ of length 0.

  The last state of a finite path π will be denoted by $l(\pi)$.
- ⋆ $\pi \xRightarrow{a} \pi'$ holds if π' is a one-step extension of π along the transition a, i.e., $\pi = \pi(0) \xmapsto{a_0} \pi(1) \xmapsto{a_1} \cdots \pi(n)$ and $\pi' = \pi(0) \xmapsto{a_0} \pi(1) \xmapsto{a_1} \cdots \pi(n) \xmapsto{a} \pi(n+1)$.

∇

Definition 1.3.2. [**Unfolding of a rooted transition system**] The unfolding of the rooted (labelled) transition system $(\mathcal{T}, r)$ is the rooted (labelled) transition system $(\widehat{\mathcal{T}}[\widehat{r}], \widehat{r})$ where $\widehat{r}$ is the single-state path beginning at r.

It is also called the *unfolding of $\mathcal{T}$ from the state r.* ∇

Definition 1.3.3. [**Unfolding of an interpreted transition system**] The unfolding of the interpreted transition system $\mathcal{M} = (\mathcal{T}, L)$ is the interpreted transition system $\widehat{\mathcal{M}} = (\widehat{\mathcal{T}}, \widehat{L})$, where $\widehat{L}(\pi) := L(l(\pi))$ for every $\pi \in \widehat{\mathcal{T}}$.

Likewise, the unfolding of the rooted interpreted transition system $(\mathcal{M}, r)$ is the rooted interpreted transition system $(\widehat{\mathcal{M}}[\widehat{r}], \widehat{r})$. ∇

Note that every unfolding of a transition system is a forest-like transition system, and every unfolding from a state is a tree-like rooted transition system.

Also, note that $\pi_0 \overset{a_0}{\Longrightarrow} \pi_1 \overset{a_1}{\Longrightarrow} \pi_2 \cdots$ is a path in $\widehat{\mathcal{T}}$ iff $l(\pi_0) \xmapsto{a_0} l(\pi_1) \xmapsto{a_1} l(\pi_2) \cdots$ is a path in $\mathcal{T}$. Consequently, $\widehat{L}(\pi_0) \overset{a_0}{\Longrightarrow} \widehat{L}(\pi_1) \overset{a_1}{\Longrightarrow} \widehat{L}(\pi_2) \cdots$ is a computation in $\widehat{\mathcal{M}}$ iff $L(l(\pi_0)) \xmapsto{a_0} L(l(\pi_1)) \xmapsto{a_1} L(l(\pi_2)) \cdots$ is (the same) computation in $\mathcal{M}$.

Thus, paths and computations in $\mathcal{M}$ and in $\widehat{\mathcal{M}}$ are in one-to-one correspondence: the last state of a path in $\widehat{\mathcal{M}}$ is actually the corresponding path in $\mathcal{M}$. This correspondence naturally maps $\widehat{\mathcal{M}}$ onto $\mathcal{M}$ and the graph of that mapping defines a behavioral equivalence between $\mathcal{M}$ and $\widehat{\mathcal{M}}$, called *bisimulation* (see next lecture).

Since the paths and computations in a rooted ITS $(\mathcal{M}, r)$ are explicitly represented by the branches of the trees in the unfolding $(\widehat{\mathcal{M}}, \widehat{r})$, the latter is also called the *computation tree* of $(\mathcal{M}, r)$, also denoted by $\mathsf{Tr}(\mathcal{M}, r)$.

1.4 Important properties of transitions systems

There are several types of properties of transitions systems that are of great practical importance are invoke the need for their formal specification and verification. They are all related to reachability or non-reachability of desired or unwanted states from a given (e.g., the initial) state or set of states in the system. We will briefly discuss and illustrate these types of properties.

1.4.1 Local properties

Local properties of transition systems are those that refer to immediate successors or predecessors of the current state. Generally, a local property has the form:

"Some/every immediate successor/predecessor of the current state satisfies the property φ",

where φ is a property of states. Some examples:

- ⋆ "The process τ will be enabled at the next state, no matter how the system evolves."
- ⋆ "When the elevator reaches the top floor, it will start moving down."

⋆ "If the process A is currently enabled, the scheduller must have disabled the process B at the previous state."

⋆ "If train is entering the tunnel now, the semaphore at the other end must have been red at the previous moment."

Typical examples of local properties are the *pre-conditions* and *post-conditions* associated with program instructions.

Local properties can be iterated a fixed number of times, e.g., referring to some/all states in 2nd, 3rd, etc. n transitions from the current state, *but not indefinitely.* Thus, local properties cannot refer to states which are reachable by *any* finite path, i.e., they *cannot express reachability.*

1.4.2 Invariance and safety properties

Invariance properties describe what must *always hold* throughout the computation, while *safety properties* describe what must *never happen* during the computation. Examples:

⋆ **Partial correctness**: "If a pre-condition P holds at the input of the program, then whenever it terminates (if it does at all) a post-condition Q will hold at the output."

Also:

⋆ " Not more than one process will be in its critical section at any moment of time."

⋆ " A resource will never be used by two or more processes simultaneously."

⋆ "No deadlock will ever occur".

More practical examples:

⋆ "The traffic lights will never light green in both directions",

⋆ "A train will not pass a red semaphore";

⋆ "The reactor will not overheat", etc.

1.4.3 Eventualities and liveness properties

Eventuality and *liveness properties* describe *what must eventually happen* during the computation. Examples:

⋆ **Total correctness**: "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."

Also:

⋆ "If the train has entered the tunnel, it will eventually leave it."

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

⋆ "If a message is sent, it will eventually be delivered." etc.

A typical example of a bad eventuality property is 'deadlock': when the system reaches a state from which it can make no further transition.

1.4.4 Fairness properties

Fairness properties reflect the idea that *all processes must be treated 'fairly' by the operating system (scheduler, etc.)*. There is a whole variety of fairness properties and a lot of literature devoted to them (see [Fra86]). They express important requirements in *concurrent systems*, i.e. systems whereby several processes sharing resources are run concurrently by an operating system which is to schedule their execution in a 'fair' way. 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 it is competing with the other processes for resources, but a *fair scheduling* would mean that if *the process is persistent for long enough then eventually its request will be granted.*

Examples:

- ⋆ **Weak fairness:** "Every continuous request will be eventually granted."
- ⋆ **Strong fairness:** "If a request is repeated infinitely often then it is eventually granted."
- ⋆ **Impartiality:** "Every process will be scheduled infinitely often."

In [Eme90] Emerson identifies fairness as the link between concurrency and non-determinism:

concurrency = non-determinism + fairness.

1.4.5 Precedence properties

Often a specification of a system involves requirements regarding the precedence of events, such as: "The event φ will occur before the event ψ (which may or may not occur at all.

Examples:

- ⋆ "If the train has entered the tunnel, it must leave it before any other train has entered."
- ⋆ "Before the traffic light turns green in a given direction, it must have turned red in the intersecting road"

1.5 The basic modal logic for transition systems TL

Recall that interpreted transition systems are just multimodal Kripke structures, so the basic multimodal logic is the simplest natural logical language to specify *local* properties of transition systems. Here we will present and discuss that logic, as a language for specification of such properties. To reflect on that perspective, and in order to comply with the notation for the more expressive logics to be considered further, we will use a less common notation for the basic modal operators, which can be easily translated to standard modal logic.

1.5.1 Syntax

More precisely, with every type $\tau = (A, \mathrm{PROP})$ of interpreted transition systems we associate a multimodal language $\mathrm{TL} = \mathrm{TL}_\tau$ with a set of atomic propositions PROP and a family of modal operators $(\exists \mathsf{X}_a)_{a \in A}$, each representing a transition $a \in A$.

The inductive definition of formulae is:

$$\varphi = p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \exists \mathsf{X}_a \varphi$$

The other logical connectives: $\top, \bot$, $\Rightarrow$, $\vee$, $\Leftrightarrow$ are defined as usual; besides, the *dual* of each modal operator $\exists\mathsf{X}_a$ is defined as $\forall\mathsf{X}_a := \neg\exists\mathsf{X}_a\neg$.

A *constant formula* is a formula containing no atomic propositions.

The fragment of the logic $\mathrm{TL}_\mathcal{T}$ consisting only of constant formulae is known as Hennessy-Milner logic [HM80], here denoted as $\mathrm{HML}_\mathcal{T}$.

Definition 1.5.1. The *modal depth* $\mathrm{md}(\varphi)$ of a formula φ is the greatest number of nested occurrences of modal operators in it, defined recursively as follows:

- ⋆ $\mathrm{md}(\bot) = \mathrm{md}(p) = 0$;
- ⋆ $\mathrm{md}(\varphi_1 \to \varphi_2) = \max(\mathrm{md}(\varphi_1), \mathrm{md}(\varphi_2))$;
- ⋆ $\mathrm{md}(\exists\mathsf{X}_a\varphi) = \mathrm{md}(\varphi) + 1$.

The fragment TL^n comprises all formulae of TL with modal depth $\leq n$.

∇

1.5.2 Semantics

The semantics of $\mathrm{TL}_\mathcal{T}$ is the standard Kripke semantics in interpreted transition systems. The basic semantic notion of *truth of a formula at a state s of an interpreted transition system* $\mathcal{M} = (S, \left\{\xmapsto{a}\right\}_{a\in A}, L)$ is defined inductively as follows.

- ⋆ $\mathcal{M}, s \models p$ iff $p \in L(s)$;
- ⋆ $\mathcal{M}, s \models \neg\varphi$ iff $\mathcal{M}, s \not\models \varphi$;
- ⋆ $\mathcal{M}, \pi \models \varphi \wedge \psi$ if $\mathcal{M}, \pi \models \varphi$ and $\mathcal{M}, \pi \models \psi$;
- ⋆ $\mathcal{M}, s \models \exists\mathsf{X}_a\varphi$ if $\mathcal{M}, t \models \varphi$ for some $t \in S$ such that $s \xmapsto{a} t$.

The derived truth definition of $\forall\mathsf{X}_a$ becomes:

- ⋆ $\mathcal{M}, s \models \forall\mathsf{X}_a\varphi$ if $\mathcal{M}, t \models \varphi$ for every $t \in S$ such that $s \xmapsto{a} t$.

Some terminology:

Definition 1.5.2. [**Truth, validity, satisfiability**] Given an its $\mathcal{M}$, state r in $\mathcal{M}$, and a formula φ of TL, we say that φ is:

- ⋆ *satisfied at r in $\mathcal{M}$* if $\mathcal{M}, r \models \varphi$.

  We then also say that $(\mathcal{M}, r)$ *is a model of* φ.

⋆ *satisfiable in* $\mathcal{M}$ if $\mathcal{M}, s \models \varphi$ for some state $s \in \mathcal{M}$.

⋆ *satisfiable*, if it is satisfied in some ITS.

⋆ *valid in* $\mathcal{M}$, denoted $\mathcal{M} \models \varphi$, if $\mathcal{M}, s \models \varphi$ for every state $s \in \mathcal{M}$.
We then also say that $\mathcal{M}$ *is a model of* φ.

⋆ *valid in a class of ITS* $\mathcal{C}$, denoted $\mathcal{C} \models \varphi$, if it is valid in every ITS in $\mathcal{C}$.

⋆ *valid in a state* s *of a transition system* $\mathcal{T}$, denoted $\mathcal{T}, s \models \varphi$, if $\mathcal{M}, s \models \varphi$ for every ITS over $\mathcal{T}$;

⋆ *valid in a transition system* $\mathcal{T}$, denoted $\mathcal{T} \models \varphi$, if it is valid in every ITS over $\mathcal{T}$.

⋆ *valid*, denoted $\models \varphi$, if it is valid in every ITS.

∇

Definition 1.5.3. [**Extension of a formula**] The *extension* of a formula φ in an ITS $\mathcal{M}$ is the set of states in $\mathcal{M}$ satisfying the formula:

$$[\![\varphi]\!]_{\mathcal{M}} := \{s \mid \mathcal{M}, s \models \varphi\}.$$

∇

Proposition 1.5.1. The extension of a formula $[\![\varphi]\!]_{\mathcal{M}}$ can be computed inductively on the construction of φ:

⋆ $\|p\|_{\mathcal{M}} = \{s \mid p \in L(s)\}$;

⋆ $\|\bot\|_{\mathcal{M}} = \emptyset$;

⋆ $\|\varphi_1 \to \varphi_2\|_{\mathcal{M}} = (S \setminus \|\varphi_1\|_{\mathcal{M}}) \cup \|\varphi_2\|_{\mathcal{M}}$;

⋆ $\|\exists \mathsf{X}_a \varphi\|_{\mathcal{M}} = pre(\|\varphi\|_{\mathcal{M}}) = \{s \mid R_a(s) \cap \|\varphi\|_{\mathcal{M}} \neq \emptyset\}$.

1.6 Some notes

In the temporal logics framework the structure of actions and transitions is usually hidden, and this is one of the abstractions of this formalization of the notion of computation. In this framework, an action is just a blackbox, and all that matters is how it transforms states, i.e., the transition relation it generates. Of course, one can take a different approach by considering the computations from viewpoint of the *internal structure* of the actions or programs: these can be built from 'atomic' actions or programs using some action/program constructs, such as *composition, conditional branching, iteration*, etc. Transition systems with appropriately structured set of actions provide a convenient formalism for specifying operational semantics of programming languages (see e.g. [Plo81] and [Sti92]) and are typically used in modelling sequential programs in logical languages such as the *propositional dynamic logic* PDL [HKT00] and various logics of processes (see [KT90]).

Another essential (implicit) assumption is that the future of a computation only depends on the current state, but not on its past. This is the main reason why temporal models of computations usually make use only of the future fragments of temporal logics. However, temporal logics involving past operators, have been studied, too. Also, memory-based transition systems can be modelled in this framework, by using the ‘unfolding’ construction, described further.

Finally, transition systems can be finite or infinite, and much of our treatment here will apply to either. However, for some specific topics, e.g. model-checking we will assume the models to be finite.

Chapter 2

The linear-time temporal logic LTL

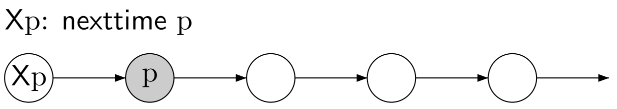
The linear-time temporal logics are intended to reason about linear models; in our case these are single computations. The most popular linear-time temporal logic LTL was first considered in the form presented here in [GPSS80], based on the early works [Kam68, Pnu77b]. Indeed, the strict until operator, that can express all temporal operators in LTL, was first proposed in [Kam68] while temporal logics were proposed as a framework for formal verification of programs in [Pnu77b]. Notably, the next-time operator was introduced in [MP79] in order to define LTL restricted to the next-time and sometime operators (see also a similar language in [Pnu79]).

Nowadays, LTL is one of the most used logical formalisms to specify the behaviours of computer systems in view of formal verification. It has also been the basis for numerous specification languages, such as PSL [EF06], and it is used as a specification language in various tools such as SPIN [Hol97] and SMV [McM93].

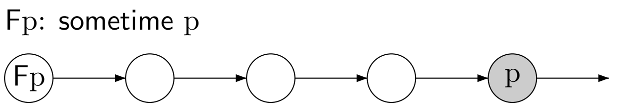
2.1 LTL intuitively

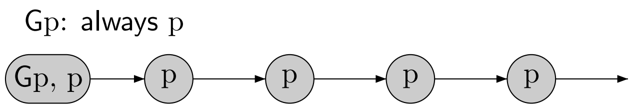
The linear-time temporal logic LTL usually involves the temporal operators X ("neXt time"), F ("sometimes in the Future"), G ("always in the future"), and U ("Until") besides the classical propositional connectives $\neg$ (negation), $\vee$ (disjunction), $\wedge$ (conjunction) and $\Rightarrow$ (material implication). The intuitive meaning of the temporal operators:

Nexttime. Whereas φ states a property of the current state, $\mathsf{X}\varphi$ states that the next state satisfies φ. Thus, $\varphi \vee \mathsf{X}\varphi$ states that φ is satisfied now or in the next state.



Sometime and **Always**. Fp claims that some future (or possibly, the current) state satisfies φ without specifying which, while $\mathsf{G}\varphi$ claims that all the future states (including the current one) satisfy φ, i.e., that "φ will always be true".



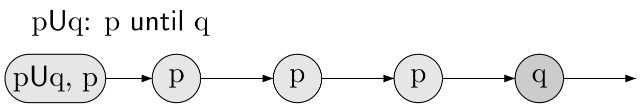


By way of example, the expression $\texttt{alert} \Rightarrow \mathsf{F}\ \texttt{halt}$ means that if the system currently is in a state of alert, then it will sometime later be in a halt state.

The operator G is the *dual* of F: whatever the formula φ may be, if φ is always satisfied, then it is not true that $\neg\varphi$ will ever be satisfied, and conversely. Hence $\mathsf{G}\varphi$ and $\neg\mathsf{F}\neg\varphi$ are equivalent.

Thus, $\mathsf{G}(\texttt{alert} \Rightarrow \mathsf{F}\ \texttt{halt})$ means that whenever in the future the system is in a state of alert, it will sometime later be in a halt state.

Until. The binary operator U is richer and more complicated than the operator F. $\varphi_1\mathsf{U}\varphi_2$ states that φ_1 is true until φ_2 is true. More precisely: φ_2 will be true at some future state, and φ_1 will hold in the meantime.



The example $\mathsf{G}(\texttt{alert} \Rightarrow \mathsf{F}\ \texttt{halt})$ can be refined with the statement that “starting from a state of alert, the alarm remains activated until the halt state is eventually reached”:

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

Note that the operator F is a special case of U: $\mathsf{F}\varphi$ and $\texttt{true}\mathsf{U}\varphi$ are equivalent.

Other operators. Additional definable operators are sometimes added to LTL, e.g.:

Weak until. This is a variation of Until, denoted W. Intuitively, the statement $\varphi_1\mathsf{W}\varphi_2$ still expresses “φ_1 until φ_2”, but without the inevitable occurrence of φ_2; if φ_2 never occurs, then φ_1 must remain true forever. Thus, $\varphi_1\mathsf{W}\varphi_2$ is equivalent to $\mathsf{G}\varphi_1 \vee (\varphi_1\mathsf{U}\varphi_2)$.

Release. The “release” operator R is defined as the dual of U, i.e., $\varphi_1\mathsf{R}\varphi_2 := \neg(\neg\varphi_1\mathsf{U}\neg\varphi_2)$. The formula $\varphi_1\mathsf{R}\varphi_2$ intuitively states that the truth of φ_1 releases the constraint on the satisfaction of φ_2; more precisely, it means that either φ_1 will be true some day and φ_2 must hold between the current state and that day, or φ_2 must be true in all future states (or both).

Using LTL one can express various properties of computations, e.g.:

(safety) $\mathsf{G}(\texttt{halt} \Rightarrow \texttt{F}^{-1}\texttt{alert})$,

(liveness) $\mathsf{G}(\mathrm{p} \Rightarrow \mathsf{F}\mathrm{q})$,

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

(strong fairness) $\mathsf{GF}\ \texttt{enabled} \Rightarrow \mathsf{GF}\ \texttt{executed}$.

2.2 Formal syntax and semantics of LTL in linear models

2.2.1 Syntax and formulae of LTL

LTL formulae are built from the following formal grammar:

$$\varphi ::= \overbrace{\bot \mid \top \mid \mathrm{p} \mid \neg\varphi \mid \varphi \wedge \psi \mid \varphi \vee \psi}^{\text{propositional calculus}} \mid \overbrace{\mathsf{X}\varphi \mid \mathsf{F}\varphi \mid \mathsf{G}\varphi \mid \varphi\mathsf{U}\psi}^{\text{temporal extension}}$$

where p ranges over a countably infinite set PROP of propositional variables, obtained by abstracting properties; for instance p may mean "$x = 0$".

Given a set of temporal operators $\mathcal{O} \subseteq \{\mathsf{X}, \mathsf{F}, \mathsf{G}, \mathsf{U}\}$, we write LTL($\mathcal{O}$) to denote the restriction of LTL to formulae with temporal connectives from $\mathcal{O}$. Given a temporal operator $\mathtt{O}$, an $\mathtt{O}$-formula is an LTL formula whose outermost connective is $\mathtt{O}$, in particular it cannot be a propositional formula. We write $sub(\varphi)$ to denote the set of subformulae of the formula φ and $|\varphi|$ to denote the size of the formula φ viewed as a string of characters.

We will use the following abbreviations: $\varphi_1 \Rightarrow \varphi_2$ for $\neg\varphi_1 \vee \varphi_2$ and $\mathsf{F}^\infty\varphi$ for $\mathsf{GF}\varphi$ ("φ holds infinitely often"). Similarly, $\varphi_1\mathsf{R}\varphi_2$ is used as an abbreviation for $\neg(\neg\varphi_1\mathsf{U}\neg\varphi_2)$.

Fragments. We write $\mathrm{LTL}^k_n(\mathtt{O}_1, \mathtt{O}_2, \ldots)$ to denote the fragment of LTL restricted to formulae such that

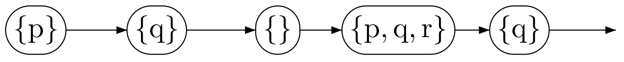
- ⋆ the temporal operators are among $\mathtt{O}_1, \mathtt{O}_2, \ldots$,
- ⋆ the temporal depth is bounded by k,
- ⋆ at most n distinct atomic propositions occur.

When n (resp. k) takes the value ω, we mean that there is no restriction on the number of propositional variables (resp. on the temporal depth). In that case, we may omit ω as well as. Just like for TL, the *modal depth* of an LTL-formula, noted md(φ) is defined as the maximal nesting of temporal operators. For instance, $\mathrm{md}((\mathsf{XX}\mathrm{p}) \vee (\mathrm{p}\mathsf{U}\neg\mathrm{q})) = 2$. So, $\mathrm{LTL}^2_\omega(\mathsf{F})$ denotes the set of LTL-formulae of temporal depth at most 2 built over the temporal operator F (no restriction on the number of propositional variables).

2.2.2 Models and truth of LTL formulae

Models for LTL are single computations, viewed as ω-sequences, and hereafter just called *linear models* or LTL-*models*.

Formally, an LTL-model is an infinite sequence $\sigma : \mathbb{N} \to \mathcal{P}(\mathrm{PROP})$, i.e., an infinite word of $(\mathcal{P}(\mathrm{PROP}))^\omega$. For example, here are the five first states of an LTL-model:


$$\begin{tikzcd}\{\mathrm{p}\} \arrow[r] & \{\mathrm{q}\} \arrow[r] & \{\} \arrow[r] & \{\mathrm{p},\mathrm{q},\mathrm{r}\} \arrow[r] & \{\mathrm{q}\} \arrow[r] & {}\end{tikzcd}$$

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

- ⋆ always $\sigma, i \models \top$ and never $\sigma, i \models \bot$,

- ⋆ $\sigma, i \models \mathrm{p}$ iff $\mathrm{p} \in \sigma(i)$, for every $\mathrm{p} \in \mathrm{PROP}$,
- ⋆ $\sigma, i \models \neg\varphi$ iff $\sigma, i \not\models \varphi$,
- ⋆ $\sigma, i \models \varphi_1 \wedge \varphi_2$ iff $\sigma, i \models \varphi_1$ and $\sigma, i \models \varphi_2$,
- ⋆ $\sigma, i \models \varphi_1 \vee \varphi_2$ iff $\sigma, i \models \varphi_1$ or $\sigma, i \models \varphi_2$,
- ⋆ $\sigma, i \models \mathsf{X}\varphi$ iff $\sigma, i+1 \models \varphi$,
- ⋆ $\sigma, i \models \mathsf{F}\varphi$ iff there is $j \geq i$ such that $\sigma, j \models \varphi$,
- ⋆ $\sigma, i \models \mathsf{G}\varphi$ iff for all $j \geq i$, we have $\sigma, j \models \varphi$,
- ⋆ $\sigma, i \models \varphi_1 \mathsf{U} \varphi_2$ iff there is $j \geq i$ such that $\sigma, j \models \varphi_2$ and $\sigma, k \models \varphi_1$ for all $i \leq k < j$.

We write $\sigma \models \varphi$ instead of $\sigma, 0 \models \varphi$. Models for a formula φ can be viewed as a language $\mathrm{Mod}(\varphi)$ over the alphabet $\mathcal{P}(\mathrm{PROP}(\varphi))$ where $\mathrm{PROP}(\varphi)$ denotes the set of propositional variables occurring in φ (these are the only relevant ones for the satisfaction of φ):

$$\mathrm{Mod}(\varphi) = \{\sigma \in (\mathcal{P}(\mathrm{PROP}(\varphi)))^\omega \mid \sigma \models \varphi\}.$$

NB. In a given temporal formula only a finite number of atomic propositions are present. That is why, some of the technical developments in the sequel assume that PROP is finite. However, when we really need an infinite amount of atomic propositions, typically to define some reductions and to establish fine-tuned complexity results, we will assume that PROP is countably infinite.

2.2.3 Equivalence, satisfiability, and validity of LTL formulae

We say that two formulae φ and ψ are *equivalent* whenever for all models σ and positions i, we have $\sigma, i \models \varphi$ if and only if $\sigma, i \models \psi$. In that case, we write $\varphi \equiv \psi$. Roughly speaking, φ and ψ state equivalent properties over the class of ω-sequences indexed by propositional valuations. Similarly, we say that φ and ψ are *initially equivalent* (noted $\varphi \equiv_0 \psi$) whenever for all models σ, we have $\sigma, 0 \models \varphi$ if and only if $\sigma, 0 \models \psi$. For instance, $\mathsf{F}\varphi$ is equivalent to $\top\mathsf{U}\varphi$. Consequently, it is clear that our set of connectives is not minimal in terms of expressive power but it provides handy notations.

Since LTL contains only temporal operators referring to the future, the following holds.

Observation: The equivalences $\equiv$ and $\equiv_0$ are identical for LTL formulae.

Definition 2.2.1. [**Satisfiability and validity**] An LTL-formula φ is *satisfiable* if $\mathrm{Mod}(\varphi)$ is non-empty. Respectively, φ is *valid* if $\neg\varphi$ is not satisfiable, i.e., $\mathrm{Mod}(\neg\varphi)$ is empty. ∇

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

Input: an LTL formula φ,

Question: Is there some LTL-model σ such that $\sigma \models \varphi$?
Equivalently, is it the case that $\mathrm{Mod}(\varphi) \neq \emptyset$?

The *validity problem* VAL(LTL) is defined similarly.

Input: an LTL formula φ,

Question: Is the case that $\sigma \models \varphi$ for all LTL-models σ?

Equivalently, is it the case that $\mathrm{Mod}(\neg\varphi) = \emptyset$?

While the LTL models are essentially linear structures, LTL formulae can naturally be interpreted over any (rooted) transition systems $(\mathcal{M}, s)$ so that φ holds true at $(\mathcal{M}, s)$ iff *for all computations* σ starting at the state s, we have $\sigma \models \varphi$. If that is the case we write $\mathcal{M}, s \models_\forall \varphi$.

An alternative notion of truth replaces the universal quantification over computations by an existential one: $\mathcal{M}, s \models_\exists \varphi$. Thus, we talk about *universal* and *existential* truth of an LTL-formula in a rooted transition system.

2.2.4 Some useful validities of LTL formulae

Proposition 2.2.1. The following LTL-formulae are valid.

- $\star$ $\mathsf{G}^\infty\varphi \to \mathsf{F}^\infty\varphi$;
- $\star$ $\mathsf{G}\varphi \wedge \mathsf{F}\psi \to \varphi\mathsf{U}\psi$;
- $\star$ $\mathsf{F}\varphi \leftrightarrow \varphi \vee \mathsf{XF}\varphi$;
- $\star$ $\mathsf{G}\varphi \leftrightarrow \varphi \wedge \mathsf{XG}\varphi$;
- $\star$ $\varphi \wedge \mathsf{G}(\varphi \to \mathsf{X}\varphi) \to \mathsf{G}\varphi$;
- $\star$ $\varphi \wedge \mathsf{G}(\varphi \to \mathsf{XF}\varphi) \to \mathsf{GF}\varphi$;
- $\star$ $(\varphi \to \chi)\mathsf{U}\psi \wedge \varphi\mathsf{U}\psi \to \chi\mathsf{U}\psi$;
- $\star$ $\varphi\mathsf{U}\psi \leftrightarrow (\psi \vee (\varphi \wedge \mathsf{X}(\varphi\mathsf{U}\psi)))$;
- $\star$ $\psi\mathsf{U}(\varphi \vee \chi) \leftrightarrow (\psi\mathsf{U}\varphi \vee \psi\mathsf{U}\chi)$;
- $\star$ $\mathsf{G}((\psi \vee (\varphi \wedge \mathsf{X}\theta)) \to \theta) \to (\varphi\mathsf{U}\psi \to \theta)$.

2.3 Addendum: On the expressiveness of LTL

A lot can be said about the expressiveness of LTL and we refer the reader to e.g., [GHR94, KM08]. In this section, we present one one result about the expressiveness of LTL, showing that this logic can capture trace equivalence between finite rooted transition systems. Clearly, when two rooted interpreted transition systems have the same computations, they satisfy the same LTL formulae. That the converse is also true is formally stated in Proposition 2.3.1 below. Moreover, this characterization can be obtained by considering ultimately periodic computations only. We write $\mathrm{Traces}^{\mathrm{UL}}(\mathcal{M}, s)$ to denote the computations in $\mathrm{Traces}(\mathcal{M}, s)$ that are ultimately periodic.

Proposition 2.3.1. Let $(\mathcal{M}, s)$ and $(\mathcal{M}', s')$ be two finite and total rooted interpreted transition systems. The following are equivalent:

(I) For every LTL formula φ, we have $\mathcal{M}, s \models_{\exists} \varphi$ iff $\mathcal{M}', s' \models_{\exists} \varphi$.

(II) $\mathrm{Traces}(\mathcal{M}, s) = \mathrm{Traces}(\mathcal{M}', s')$.

(III) $\mathrm{Traces}^{\mathrm{UL}}(\mathcal{M}, s) = \mathrm{Traces}^{\mathrm{UL}}(\mathcal{M}', s')$.

Proof: The proof uses Büchi automata on infinite words, and can be skipped by the reader not familiar with them.

(II) implies (I) is by an easy verification.

(III) implies (II) is a consequence of [CNP94]. Indeed, let $\Sigma = \mathcal{P}(\mathrm{PROP})$. Given $(\mathcal{M}, s)$, we consider the Büchi automaton $\mathcal{A}_{\mathcal{M},s} = (\Sigma, Q, Q_0, \delta, F)$ such that $\mathrm{L}(\mathcal{A}_{\mathcal{M},s}) = \mathrm{Traces}(\mathcal{M}, s)$. $\mathcal{A}_{\mathcal{M},s}$ is built as follows:

- ⋆ $Q = S$, $Q_0 = \{s\}$, $F = S$,
- ⋆ $(s, a, s') \in \delta$ iff sRs' and $L(s) = a$.

The Büchi automaton $\mathcal{A}_{\mathcal{M}',s'}$ is built similarly. Consequently, $\mathrm{Traces}(\mathcal{M}, s)$ and $\mathrm{Traces}(\mathcal{M}', s')$ are ω-regular languages. Assume that (III) and suppose that not (II). So,

$$\mathrm{L}' = (\mathrm{Traces}(\mathcal{M}, s) \setminus \mathrm{Traces}(\mathcal{M}', s')) \cup (\mathrm{Traces}(\mathcal{M}', s') \setminus \mathrm{Traces}(\mathcal{M}, s'))$$

is nonempty and ω-regular (since ω-regular languages are closed under Boolean operations). By construction, L' has no ultimately periodic ω-word, but this is in contradiction with the fact that every nonempty ω-regular language has at least one ultimately periodic ω-word. In order to prove that (I) implies (III), we need a preliminary definition. Let $\Gamma \in \Sigma$. In the formulae below, the expression $\bigwedge_{\mathrm{PROP}} \Gamma$ is an abbreviation for

$$\bigwedge_{\mathrm{p} \in \Gamma} \mathrm{p} \wedge \bigwedge_{\mathrm{p} \in \mathrm{PROP} \setminus \Gamma} \neg \mathrm{p}$$

For any finite UP-model σ over PROP with prefix index i and period m we write ψ_σ to denote the formula below:

$$\bigwedge_{0 \leq j \leq i+m} \mathsf{X}^j (\bigwedge_{\mathrm{PROP}} \sigma(j)) \wedge \mathsf{X}^i (\bigwedge_{\Gamma \in \Sigma} \mathsf{G}((\bigwedge_{\mathrm{PROP}} \Gamma) \Rightarrow \mathsf{X}^m (\bigwedge_{\mathrm{PROP}} \Gamma))$$

Assume that (I) and not (III). Since $\mathrm{Traces}^{\mathrm{UL}}(\mathcal{M}, s) \neq \mathrm{Traces}^{\mathrm{UL}}(\mathcal{M}', s')$, say there is $\sigma \in \mathrm{Traces}^{\mathrm{UL}}(\mathcal{M}, s) \setminus \mathrm{Traces}^{\mathrm{UL}}(\mathcal{M}', s')$ is nonempty (the other case requires an analogous treatment). Consequently, $\sigma \models \psi_\sigma$ and therefore $\mathcal{M}, s \models_{\exists} \psi_\sigma$. However, $\sigma \notin \mathrm{Traces}^{\mathrm{UL}}(\mathcal{M}', s')$ and for every trace $\sigma' \in \mathrm{Traces}(\mathcal{M}', s')$, $\sigma' \neq \sigma$, whence $\sigma' \not\models \psi_\sigma$. Hence, $\mathcal{M}', s' \not\models_{\exists} \psi_\sigma$, which leads to a contradiction. QED

2.4 Exercises

1. Express in LTL the following specifications:

   (a) *"Every green light will be followed by a yellow light, which will be followed by a red light, which will be followed by a green light."*

   (b) *"Whenever the event φ occurs in the future, it will not hold at the next moment, but will occur again later in the future."*

   (c) *"A train may not enter the tunnel unless the light on the other end of the tunnel is red. "*

   (d) *"A train may only enter the tunnel after the light on the other end of the tunnel has turned red. "*

   (e) *"Whenever a train enters the tunnel, the light on the other end of the tunnel must be red and will stay red until the train has left the tunnel."*

   (f) *"The event φ may occur not more than twice in the future, at least two time units apart."*

2. Translate to English the following LTL specifications, in the most natural way you can.

   (a) $\mathsf{G}^{\infty}(q \to \mathsf{X}p)$.

   (b) $\mathsf{F}(p \wedge \mathsf{XG}\neg p)$.

   (c) $\mathsf{F}(p \wedge (q\mathsf{U}\neg p))$.

   (d) $\mathsf{F}^{\infty}(p\mathsf{U}\neg p)$.

3. Consider the following linear LTL-model:

$$\begin{tikzcd} \boxed{q,r} \arrow[r] & \boxed{p} \arrow[r] & \boxed{q} \arrow[r] & \boxed{p,q,r} \arrow[r] & \boxed{q} \arrow[r] & \boxed{p,q} \arrow[r] & \boxed{q,r} \arrow[r] & \cdots \end{tikzcd}$$

   The valuation defining the model σ:

   $V(p) = \{s_{2k+1} \mid k \in \mathbb{N}\}$,

   $V(q) = \{s_k \mid 3 \leq k \leq 100 \text{ or } k \text{ is even}\}$

   $V(r) = \{s_{3k} \mid k \in \mathbb{N}\}$.

   Determine:

   $\sigma, 0 \overset{?}{\models} \mathsf{F}(q \wedge \mathsf{XX}p)$

   $\sigma \overset{?}{\models} \mathsf{G}(r \to \mathsf{X}\neg r \wedge \mathsf{XX}\neg r)$

   $\sigma \overset{?}{\models} \mathsf{FG}\neg(q \wedge r)$

   $\sigma \overset{?}{\models} \mathsf{GF}\neg(q \wedge r)$

   $\sigma \overset{?}{\models} \mathsf{FGF}(p \wedge q \wedge r)$

   $\sigma \overset{?}{\models} \mathsf{GF}(p \wedge \mathsf{X}r)$

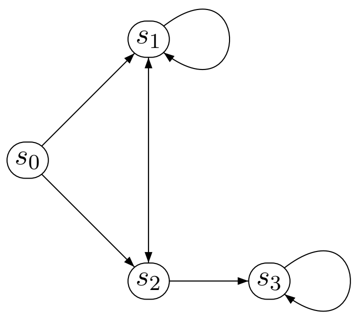
   $\sigma \overset{?}{\models} \mathsf{F}(q \;\mathsf{U}\; \neg(p \vee q \vee r))$

$\sigma \overset{?}{\models} \mathsf{GF}(r \mathbin{\mathsf{U}} (\neg p \wedge \mathsf{X}\neg r))$

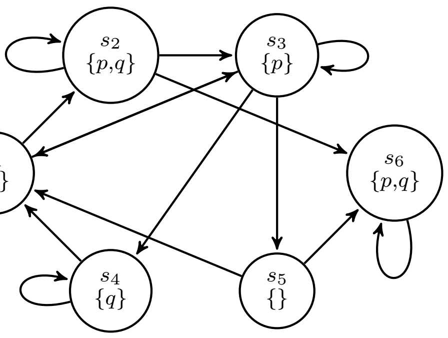
$\sigma \overset{?}{\models} \mathsf{GF}((p \wedge \neg r)\mathsf{U}\neg p)$

$\sigma \overset{?}{\models} \neg q \mathbin{\mathsf{U}} (q\mathsf{U}r)$

4. Construct the unfoldings of the following transition system from each of its states.



5. Consider the following interpreted transition system:



Check the following:

$\mathcal{M}, s_1 \overset{?}{\models}_{\exists} \mathsf{G}p$

$\mathcal{M}, s_1 \overset{?}{\models}_{\exists} \mathsf{G}(q \rightarrow \mathsf{X}p)$

$\mathcal{M}, s_1 \overset{?}{\models}_{\forall} \mathsf{G}(q \rightarrow \mathsf{X}p)$

$\mathcal{M}, s_1 \overset{?}{\models}_{\forall} \mathsf{GF}(p \vee q)$

$\mathcal{M}, s_1 \overset{?}{\models}_{\forall} \mathsf{FG}(\neg p \rightarrow \mathsf{X}q)$

$\mathcal{M}, s_1 \overset{?}{\models}_{\exists} \mathsf{X}(p \mathbin{\mathsf{U}} \mathsf{G}q)$

$\mathcal{M}, s_3 \overset{?}{\models}_{\exists} \neg q \mathbin{\mathsf{U}} (p \mathbin{\mathsf{U}} \mathsf{G}q)$

$\mathcal{M}, s_3 \overset{?}{\models}_{\exists} \mathsf{F}\varphi \leftrightarrow \varphi \vee \mathsf{XF}\varphi$

$\mathcal{M} \overset{?}{\models}_{\exists} \mathsf{FG}(p \vee q)$

$\mathcal{M} \overset{?}{\models}_{\forall} \mathsf{GF}(p \vee q)$

6. Check the validity of each of the following LTL-formulae. If not valid, give a counter-model: a linear LTL-model on which the formula is false.

   (a) $\mathsf{G}\varphi \land \mathsf{F}\psi \to \varphi \mathsf{U} \psi$

   (b) $\varphi \land \mathsf{G}(\varphi \to \mathsf{X}\varphi) \to \mathsf{G}\varphi$

   (c) $\varphi \land \mathsf{G}(\varphi \to \mathsf{XF}\varphi) \to \mathsf{GF}\varphi$

   (d) $\mathsf{G}\varphi \leftrightarrow \varphi \land \mathsf{XG}\varphi$

   (e) $\varphi \mathsf{U} \psi \leftrightarrow (\psi \lor (\varphi \land \mathsf{X}(\varphi \mathsf{U} \psi)))$

   (f) $\psi \mathsf{U}(\varphi \land \chi) \leftrightarrow (\psi \mathsf{U} \varphi \land \psi \mathsf{U} \chi)$

   (g) $\psi \mathsf{U}(\varphi \lor \chi) \leftrightarrow (\psi \mathsf{U} \varphi \lor \psi \mathsf{U} \chi)$

Chapter 3

Branching time temporal logics

While LTL is suited for reasoning about *single computations* in a transition system, more expressive languages and logics are needed to reason about the *entire* transition system. Such languages should be able to refer to, and quantify over, *all possible computations* starting from a given state.

The study and use of branching-time logics in computer science started in the late 1970's - early 1980's when several similar branching-time logical systems were proposed in [Abr79], [Lam80], [BAPM81], and [CE81]. The latter paper introduced the *computation tree logic* CTL, which subsequently emerged as the most popular and practically useful among these. Soon, a debate on the pros and cons of linear time (LTL) vs branching time (CTL) logics ensued, and in response to it the very expressive logic CTL*, encompassing both approaches, was introduced in 1983, in [EH83],[EH86]. See the survey [Eme90] and the bibliographic notes for more details and references.

3.1 The temporal logic of reachability TLR

Before discussing the more expressive and popular branching time logics CTL and CTL*, we will introduce briefly the basic *temporal logic of reachability* in transition systems, denoted as TLR. We will then build the other branching time logics on top of TLR.

3.1.1 Syntax and semantics of the reachability Logic TLR

Given a TS $\mathcal{T} = (S, \left\{\xmapsto{a}\right\}_{a\in A})$, we denote by $\xmapsto{a}^*$ the reflexive and transitive closure of the transition relation $\xmapsto{a}$, that is: $s \xmapsto{a}^* t$ iff there exists a finite path $s = s_0 \xmapsto{a} s_1 \ldots \xmapsto{a} s_n = t$, where $n \geq 0$.

The *temporal logic of reachability* TLR extends TL with additional *(existential) reachability modalities* $\exists\mathsf{F}_a$, one for each transition relation R_a, with semantics:

> $\mathcal{M}, s \models \exists\mathsf{F}_a\varphi$ if $\mathcal{M}, t \models \varphi$ for some $t \in S$ such that $s \xmapsto{a}^* t$, i.e. such that there exists a finite path $s = s_0 \xmapsto{a} s_1 \ldots \xmapsto{a} s_n = t$.

Thus, $\exists\mathsf{F}_a\varphi$ is true at the state s if φ is true at *some* state t reachable from s.

The dual, *(universal) coverability modalities*, are defined as expected: $\forall \mathsf{G}_a := \neg\exists\mathsf{F}_a\neg$ and have the respective semantics:

$$\mathcal{M}, s \models \forall\mathsf{G}_a\varphi \text{ if } \mathcal{M}, t \models \varphi \text{ for every } t \in S \text{ such that } s \overset{a}{\longmapsto}^* t.$$

Thus, $\forall\mathsf{G}_a\varphi$ is true at the state s if φ is true at *every* state t reachable from s.

All basic syntactic and semantic notions of TL extend seamlessly to TLR; we leave the details to the reader.

3.2 The full computation tree logic CTL*

As we have already noted, the logic CTL* was introduced by Emerson and Halpern in [EH83], [EH86] as an extension of, and a unifying approach to, both the linear-time logic LTL and the branching-time logic CTL. From purely logical perspective, the logic CTL* is precisely the Ockhamist logic of the class of ω-trees, i.e., of the tree-like models where every path has the order type ω of the natural numbers, while CTL is the Peircean logic on the same class of models. Arguably, CTL* is more natural in logical terms than CTL, as it fully combines the purely temporal fragment with the path quantification, without the syntactic restrictions of CTL. So, contrary to the historical development, we will begin with CTL*.

3.2.1 Language and syntax

The language of CTL* extends the one of LTL with the *path quantifier* $\forall$, thus involving formulae of the type $\forall\varphi$, meaning "φ is true on *every* computation passing through the current state".

The set of formulae of CTL* is defined recursively as follows:

$$\varphi := p \mid \neg\varphi \mid (\varphi \wedge \varphi) \mid \mathsf{X}\varphi \mid \varphi\mathsf{U}\varphi \mid \forall\varphi$$

The other logical connectives $\top, \bot, \Rightarrow, \vee, \Leftrightarrow$, and the temporal operators F and G are definable as usual, as well as the 'macros' G^∞ and F^∞. Again as usual, $\exists\varphi$ is defined as $\neg\forall\neg\varphi$ and means "φ is true on *some* computation passing through the current state". Parentheses in formulae will be omitted when no ambiguity may arise.

It is convenient to distinguish two types of CTL*-formulae: *state formulae*, that are evaluated relative to states, and *path formulae*, evaluated relative to paths. The sets StateFor of state formulae and PathFor of path formulae are defined by mutual induction as follows.

StateFor:

⋆ All atomic propositions and $\bot$ are in StateFor.

⋆ If $\varphi, \psi \in$ StateFor then $\neg\varphi, \varphi \wedge \psi \in$ StateFor.

⋆ If $\varphi \in$ PathFor, then $\forall\varphi \in$ StateFor.

PathFor:

⋆ Every state formula is a path formula: StateFor $\subset$ PathFor.

⋆ If $\varphi, \psi \in$ PathFor then $\neg\varphi, \varphi \wedge \psi, \mathsf{X}\varphi, \varphi\mathsf{U}\psi \in$ PathFor.

3.2.2 Semantics

The models of CTL* are interpreted transition systems.

Since all CTL*-formulae are path formulae, the basic semantic notion is *truth of a formula relative to a path in an interpreted transition system.* If $\mathcal{M} = \langle S, R, L\rangle$ is an interpreted transition system and π is a path in $\mathcal{M}$, then $\mathcal{M}, \pi \models \varphi$ will mean that φ *is true of the path* π *in* $\mathcal{M}$.

Notation: given a path π, we obtain the path $\pi_{\geq k}$ by chopping off the first k states of π, i.e., $\pi_{\geq k} = \pi(k), \pi(k+1), \pi(k+2), \ldots$.

The inductive definition of $\mathcal{M}, \pi \models \varphi$ naturally extends the truth definitions for LTL and TLR.

- ⋆ $\mathcal{M}, \pi \models p$ iff $p \in L(\pi(0))$ for $p \in$ PROP;
- ⋆ $\mathcal{M}, \pi \models \neg\varphi$ iff $\mathcal{M}, \pi \not\models \varphi$;
- ⋆ $\mathcal{M}, \pi \models \varphi \wedge \psi$ iff $\mathcal{M}, \pi \models \varphi$ and $\mathcal{M}, \pi \models \psi$;
- ⋆ $\mathcal{M}, \pi \models \mathsf{X}\varphi$ iff $\mathcal{M}, \pi_{\geq 1} \models \varphi$;
- ⋆ $\mathcal{M}, \pi \models \varphi \mathsf{U} \psi$, iff $\mathcal{M}, \pi_{\geq j} \models \psi$ for some $j \geq 0$ and $\mathcal{M}, \pi_{\geq i} \models \varphi$ for every i such that $0 \leq i < j$.
- ⋆ $\mathcal{M}, \pi \models \forall\varphi$ iff $\mathcal{M}, \pi' \models \varphi$ for every path π' in $\mathcal{M}$ with the same initial state as π.

Hereafter by a CTL*-model we mean any interpreted transition system, where truth of the formulae of CTL* is defiend as above.

Now, given a CTL*-model $\mathcal{M}$ we say that:

- ⋆ a path formula φ is *true of the state* s *of* $\mathcal{M}$, denoted $\mathcal{M}, s \models \varphi$, if $\mathcal{M}, \pi \models \varphi$ for every path π in $\mathcal{M}$ such that $\pi(0) = s$.
- ⋆ a formula φ is *valid in* $\mathcal{M}$, denoted $\mathcal{M} \models \varphi$, if $\mathcal{M}, s \models \varphi$ for every state s in $\mathcal{M}$.
- ⋆ a formula φ is *valid* in a transition system $\mathcal{T}$, denoted $\mathcal{T} \models \varphi$ if it is valid in every interpreted transition system $(\mathcal{T}, L)$ over $\mathcal{T}$.
- ⋆ a formula φ is *valid*, denoted $\models \varphi$, if it is valid in every transition system.
- ⋆ a path formula φ is *satisfiable* if it is true of some path π in some interpreted transition system $\mathcal{M}$.
- ⋆ likewise, a state formula φ is *satisfiable* if it is true of some state s in some interpreted transition system $\mathcal{M}$.

NB. Note that, due to the different sorts of CTL*-formulae, unlike in most traditional logics, validity in CTL* *is not closed under uniform substitutions*. Indeed, $p \Rightarrow \forall p$ is valid for any $p \in$ PROP, while $Gp \Rightarrow \forall Gp$ is not valid.

NB. Since all formulae of CTL* are path formulae, the semantics of CTL* can be modified to evaluate every atomic proposition, and hence every formula, on a set of paths. Then, the same atomic proposition p may be true with respect to one path, while false with respect to another, starting at the same state. The resulting logic is still decidable [BHWZ04], however it becomes much less intuitive.

Exercise 3.2.1. Consider the two-sorted language for CTL*, where state and path formulae are formally distinguished, and define by simultaneous induction on state and path formulae the notions of truth respective to states and to paths.

Note that LTL can be regarded as the fragment of CTL* consisting of all *purely path formulae*, i.e., those containing no path quantifiers, while TLR is the $(\exists \mathsf{X}, \exists \mathsf{F})$-fragment of CTL*.

3.2.3 Some useful validities in CTL*

Proposition 3.2.1. The following formulae are CTL*-valid:

- $\star$ $\forall\varphi$ for every LTL-valid formula φ;
- $\star$ $\forall\varphi \Rightarrow \varphi$; (NB: φ can be a path or a state formula.)
- $\star$ $\forall(\varphi \Rightarrow \psi) \Rightarrow (\forall\varphi \Rightarrow \forall\psi)$;
- $\star$ $\forall \mathsf{X}\varphi \Rightarrow \mathsf{X}\forall\varphi$;
- $\star$ $\forall \mathsf{G}\varphi \Rightarrow \mathsf{G}\forall\varphi$;
- $\star$ $\forall \mathsf{G}\exists \mathsf{F}\varphi \Rightarrow \exists \mathsf{GF}\varphi$; (Burgess's formula)
- $\star$ $\forall \mathsf{G}(\varphi \Rightarrow \exists X\varphi) \Rightarrow (\varphi \Rightarrow \exists \mathsf{G}\varphi)$;
- $\star$ $\forall \mathsf{G}(\forall\varphi \Rightarrow \exists \mathsf{XF}\forall\varphi) \Rightarrow (\forall\varphi \Rightarrow \exists \mathsf{GF}\forall\varphi)$;
- $\star$ $\forall \mathsf{G}(\exists\varphi \Rightarrow \exists \mathsf{X}((\exists\psi \mathsf{U} \exists\theta))) \Rightarrow (\exists\varphi \Rightarrow \exists \mathsf{G}((\exists\psi \mathsf{U} \exists\theta)))$ (Reynold's limit closure formula).

Proof: Exercise. QED

3.2.4 Expressing properties of transition systems with CTL*

CTL* can be used to express various *global* properties. For instance:

- $\star$ *Partial correctness along every possible computation:*

$$\varphi \Rightarrow \forall \mathsf{G}(\mathsf{terminal} \Rightarrow \psi).$$

- $\star$ *Partial correctness along some possible computation:*

$$\varphi \Rightarrow \exists \mathsf{G}(\mathsf{terminal} \Rightarrow \psi).$$

- ⋆ *Total correctness along every possible computation:*

$$\varphi \Rightarrow \forall\mathsf{F}(\mathsf{terminal} \land \psi).$$

- ⋆ and *total correctness along some possible computation:*

$$\varphi \Rightarrow \exists\mathsf{F}(\mathsf{terminal} \land \psi).$$

- ⋆ *Fairness along every possible computation:*

$$\forall(\mathsf{GF}(\mathsf{resource\ requested}) \Rightarrow \mathsf{F}(\mathsf{resource\ granted}))$$

etc.

3.3 CTL as a fragment of CTL*

The *computation tree logic* CTL is a syntactically restricted fragment of CTL*. It was introduced before CTL* by Clarke and Emerson in [EC80] and [CE81]. It is an extension of the very similar branching time logic UB, introduced at about the same time in [BAPM81], which does not contain U, but only X and G. Although CTL is not as expressive as CTL*, it is often regarded a better choice for practical applications because of its lower computational complexity. Indeed, as we will show in next lecture, unlike CTL* model-checking of CTL is tractable.

The language and syntax of CTL are the same as those of CTL*, but there is a syntactic restriction on the formation of the CTL-formulae: the temporal operators X and U, must be immediately quantified by path quantifiers. Thus, in CTL there are *only state formulae.* For instance $\forall\mathsf{GF}\varphi$ and $\exists(\mathsf{F}\varphi \land \varphi\mathsf{U}\psi)$ are not CTL-formulae. In fact, allowing Boolean combinations of path formulae within the scope of a path quantifier does not extends essentially the expressiveness of the language, as we will show further.

Because of that syntactic restriction, $\forall(\varphi_1\mathsf{U}\varphi_2)$ and $\exists(\varphi_1\mathsf{U}\varphi_2)$ are not inter-definable in CTL, so both path quantifiers must be present in the language. Here is the recursive definition of CTL-formulae:

$$\varphi := p \mid \bot \mid (\varphi_1 \Rightarrow \varphi_2) \mid \forall\mathsf{X}\varphi \mid \forall(\varphi_1\mathsf{U}\varphi_2) \mid \exists(\varphi_1\mathsf{U}\varphi_2).$$

Some definable operators in CTL:

- ⋆ $\exists\mathsf{X}\varphi := \neg\forall\mathsf{X}\neg\varphi$,
- ⋆ $\forall\mathsf{F}\varphi := \forall(\top\mathsf{U}\varphi)$,
- ⋆ $\exists\mathsf{F}\varphi := \exists(\top\mathsf{U}\varphi)$,
- ⋆ $\forall\mathsf{G}\varphi := \neg\exists\mathsf{F}\neg\varphi$,
- ⋆ $\exists\mathsf{G}\varphi := \neg\forall\mathsf{F}\neg\varphi$.

The semantics of CTL is the same as CTL*.

Exercise 3.3.1. Show that $\forall\mathsf{U}$ is definable in terms of $\exists$-prefixed operators.

Note that the translation of TLR in CTL* in fact embeds TLR into CTL. Thus, TLR is essentially the $(\exists\mathsf{X}, \exists\mathsf{F})$-fragment of CTL. Consequently, CTL suffices to characterize any finite ITS up to bisimulation.

3.3.1 Expressing properties of transition systems with CTL

For invariance and eventuality properties, CTL is essentially as expressive as CTL*: note that the examples of CTL*-formulae expressing partial and total correctness in **??** are actually CTL-formulae.

However, CTL is not suitable for expressing *fairness properties* where G^∞ and F^∞ are essentially used[1]. For instance, the example of fairness along every possible computation, expressible in CTL* is beyond the expressiveness of CTL. It is certainly different from what seems to be the closest translation in CTL:

$$\forall\mathsf{G}\forall\mathsf{F}(\mathsf{resource\ requested}) \Rightarrow \forall\mathsf{F}(\mathsf{resource\ granted})$$

Exercise: show that these are not equivalent. Which is stronger?

3.3.2 Some variations and extensions of CTL

Numerous variations and extensions of CTL have been studied, including: CTL without nexttime operator, which is related the notion of *behavioural equivalence modulo stuttering*, see [BCG88]; CTL^2 which allows pairing of two temporal operators after a path quantifier, thus enabling expression of fairness properties, see [KG96]; CTL$^+$, which allows any boolean combinations of unnested temporal operators in the scope of a path quantifier (proved to be as expressive as CTL, but exponentially more succinct), see [LMS01]; CTL with past, see [LS00], [LMS02]; etc.

3.4 Exercises

1. Consider the two-sorted language for CTL*, where state and path formulae are formally distinguished, and define by simultaneous induction on state and path formulae the notions of truth respective to states and to paths.
2. Argue that the truth of every CTL* formula is preserved by unfoldings of rooted ITS.
3. Show that $\forall(p\mathsf{U}q)$ is definable in terms of $\exists\mathsf{X}, \exists\mathsf{G}, \exists\mathsf{U}$ and boolean connectives only.
4. Show that the CTL-formula

   $$\forall\mathsf{G}\forall\mathsf{F}(\mathsf{resource\ requested}) \rightarrow \forall\mathsf{F}(\mathsf{resource\ granted})$$

   is not equivalent to the CTL*-formula

   $$\forall(\mathsf{GF}(\mathsf{resource\ requested}) \rightarrow \mathsf{F}(\mathsf{resource\ granted}))$$

   expressing fairness along every possible computation.

   Which of these formulae is stronger?

[1]The original version of CTL introduced in [EC80] contained these operators, too.

5. Translate to English the following CTL* specifications, in the most natural way you can.

   (a) $\forall \mathsf{G}(q \to \mathsf{F}p)$.

   (b) $\forall \mathsf{F}(p \wedge \forall \mathsf{X}\mathsf{G}\neg p)$.

   (c) $\forall \mathsf{F} \forall \mathsf{G}\mathsf{F}(p \mathsf{U} \neg p)$.

6. Formalize the following specifications as state formulae in the language of CTL*. Whenever possible, produce a CTL-formula.

   (a) "On some computation starting from the current state every occurrence of p is eventually followed by an occurrence of q."

   (b) "Every state on some computation starting from the current state which satisfies p will satisfy it until q becomes false."

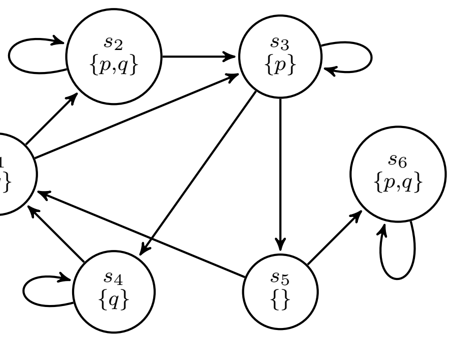
   (c) "From every state of every computation on which q is true infinitely often, a computation starts on which p will be eventually always true".

   (d) "On every computation where p is eventually always true, every state where q is false starts a computation on which p is true infinitely often."

   (e) "On every computation where p is eventually false, some state in which q is true starts no computations on which q holds immediately after every occurrence of p."

   (f) "If φ is eventually true on every computation starting from the current state, then there is a computation starting from that state and leading to a state, on every computation starting from which ψ will remain false until φ becomes true."

7. Consider the following interpreted transition system:



Check the following:

$\mathcal{M}, s_2 \overset{?}{\models} \exists \mathsf{G} \exists \mathsf{F} p$

$\mathcal{M}, s_3 \overset{?}{\models} \forall \mathsf{F} \exists \mathsf{G}(p \vee q)$

$\mathcal{M}, s_1 \overset{?}{\models} \forall \mathsf{F}(q \wedge \exists \mathsf{X} p)$

$\mathcal{M}, s_1 \overset{?}{\models} \forall \mathsf{F}(q \wedge \mathsf{X} p)$

$\mathcal{M}, s_1 \overset{?}{\models} \forall \mathsf{X} \exists (p \ \mathsf{U} \ \mathsf{G} q)$

$\mathcal{M}, s_3 \overset{?}{\models} \exists \mathsf{G}\mathsf{F}((\exists \mathsf{F} p) \mathsf{U} q)$

$$\mathcal{M} \overset{?}{\models} \forall\mathsf{G}\exists\mathsf{F}q \to \exists\mathsf{GF}q$$

$$\mathcal{M} \overset{?}{\models} \exists\mathsf{F}(\forall(q\mathsf{U}p) \wedge \neg q \wedge \exists\mathsf{X}\exists\neg(q\mathsf{U}p))$$

$$\mathcal{M} \overset{?}{\models} (q \wedge \forall\mathsf{X}\exists(q\mathsf{U}p)) \to \exists(q\mathsf{U}p).$$

8. Check the validity of each of the following CTL*-formulae. If not valid, give a counter-model: an ITS on which the formula is false.

   (a) $\forall\mathsf{G}p \to \mathsf{G}\forall p$;
   (b) $\forall\mathsf{G}(p \to \exists\mathsf{X}p) \to (p \to \exists\mathsf{G}p)$;
   (c) $\forall\mathsf{G}\varphi \leftrightarrow \varphi \wedge \forall\mathsf{X}\forall\mathsf{G}\varphi$;
   (d) $\forall\mathsf{F}\exists\mathsf{G}\varphi \to \forall\mathsf{FG}\varphi$;
   (e) $\exists(\varphi\mathsf{U}\psi) \leftrightarrow \psi \vee \exists\mathsf{X}\exists(\varphi\mathsf{U}\psi)$;
   (f) $\forall\mathsf{G}\exists\mathsf{F}\varphi \to \exists\mathsf{GF}\varphi$

9. Work out the algorithm for model checking of CTL-formulae for the cases of $[\![\forall\varphi\mathsf{U}\psi]\!]$ and $[\![\mathsf{G}^\infty]\!]$.

Chapter 4

Addendum: Model-checking and testing satisfiability of LTL formulae

4.1 LTL **model-checking**

Model-checking is about checking if a given formula is true in a given model. In order to define the *algorithmic problem* of model-checking, models must be finitely presentable objects, which leads us to the following notion.

Definition 4.1.1. [**Finite transition systems**] We say that $\mathcal{M} = (S, R, L)$ is *finite* whenever S, the image of L, and each set in that image are finite sets. (When PROP is finite, the last two conditions are automatically satisfied). ∇

It is worth observing that even when a transition system $(\mathcal{M}, s)$ is finite, the set of computations starting at s may be uncountably infinite (two states and one atomic proposition suffice for that). Furthermore, in full generality, a linear model is a function $\sigma : \mathbb{N} \rightarrow \mathcal{P}(\mathrm{PROP})$ with no specific regularity, so these are infinite objects, and some of them are *essentially infinitary* (think e.g., of binary representations of irrational numbers). Still, some infinite linear models have a simple finitary presentation, which we will use to define the algorithmic problem of model-checking LTL-formulae on linear models.

Consider a finite transition system that has the shape of a lasso, i.e. consists of a finite 'tail' of linearly arranged states, followed by a 'loop', i.e. a finite cycle of states. Clearly, such a transition system generates a unique computation, which has an ultimately periodic behaviour. Formally, an *ultimately periodic model* $\sigma : \mathbb{N} \rightarrow \mathcal{P}(\mathrm{PROP})$ is a model such that there exist natural numbers i and $p > 0$ verifying for every $k \geq i$, $\sigma(k) = \sigma(k + p)$. The finite sequence $\sigma(0), \ldots, \sigma(i-1)$ is the *prefix* (possibly empty) and $\sigma(i), \ldots, \sigma(i + p - 1)$ is the *loop*. We say that σ has *prefix index* i and *period* p. Thus, an ultimately periodic model σ can be equivalently represented by the finite sequence $\Gamma_0, \ldots, \Gamma_{i+p}$ such that each $\Gamma_j = \sigma(j)$.

4.1.1 Path model-checking of LTL formulae

As will be shown in subsequent sections, model-checking problems for LTL in finite transition systems are usually intractable, since in the worst-case exponential in the size of the input formula amount of time is needed to solve them. However, in the case of finite (in the sense defined above) ultimately periodic models the complexity of model-checking is reduced considerably. Here is the corresponding *path model-checking problem* for LTL:

input: A finite ultimately periodic model σ and an LTL formula φ.

question: Does $\sigma \models \varphi$?

Proposition 4.1.1. The path model-checking problem for LTL is decidable in PTIME.

Proof: Let σ be an ultimately periodic model of prefix index i and period p encoded by the sequence $\Gamma_0, \ldots, \Gamma_{i+p}$ in which each $\Gamma_j \subseteq$ PROP for some LTL formulae φ. One can check whether $\sigma \models \varphi$ in time $\mathcal{O}((i+p) \times |\varphi|)$ by a labelling algorithm that successively marks the positions of σ between 0 and $i+p$ by subformulae of increasing size. To do so, we introduce a Boolean array T of dimension $(i+p+1) \times \text{card}(sub(\varphi))$ with the intention that $T[j,\psi] = \top$ iff $\sigma, j \models \psi$. So, $\sigma \models \varphi$ iff $T[0,\varphi] = \top$. The elements of T are computed by considering subformulae of increasing size.

- ⋆ $T[j, \mathrm{p}] = \top$ iff $\mathrm{p} \in \Gamma_j$,
- ⋆ $T[j, \neg\psi] \stackrel{\text{def}}{=} (\neg T[j,\psi])$,
- ⋆ $T[j, \psi_1 \wedge \psi_2] \stackrel{\text{def}}{=} T[j,\psi_1] \wedge T[j,\psi_2]$,
- ⋆ $T[j, \mathsf{X}\psi_1] \stackrel{\text{def}}{=} T[j+1,\psi_1]$ for $j < i+p$,
- ⋆ $T[i+p, \mathsf{X}\psi_1] \stackrel{\text{def}}{=} T[i+1,\psi_1]$,
- ⋆ $T[j, \psi_1 \mathsf{U} \psi_2]$ is equal to

$$(\bigvee_{j \leq j' \leq i+p} (T[j',\psi_2] \wedge \bigwedge_{j \leq k < j'} T[k,\psi_1])) \vee$$

$$((j \geq i) \wedge (\bigvee_{i \leq j' < j} (T[j',\psi_2] \wedge \bigwedge_{j \leq k < i+p'} T[k,\psi_1] \wedge \bigwedge_{i \leq k < j'} T[k,\psi_1])))$$

For $\psi \in sub(\varphi)$, a naive reading of the above equalities implies that $T[0,\psi]$, ..., $T[i+p,\psi]$ can be computed in time $\mathcal{O}((i+p)^2)$. However this can be refined a bit further so that $\sigma \models \varphi$ can be checked in time $\mathcal{O}((i+p) \times |\varphi|)$. QED

NB. It is currently unknown whether the path model-checking problem for LTL is PTIME-hard [DS02, Open problem 4.1]. Variants of the problem obtained by modifying the encoding of the ultimately periodic model or the specification language have been studied in [MS03].

4.1.2 The full model-checking problems for LTL

Now, let us consider the full model-checking problems for LTL. Given an interpreted transition system $\mathcal{M} = (S, R, L)$ (recall that R is always assumed total) and a state $s \in S$, we write Traces$(\mathcal{M}, s)$ to denote the set of infinite computations whose initial state is s. Hence, Traces$(\mathcal{M}, s)$ is a possibly infinite set of ω-words in $\mathcal{P}(\mathrm{PROP})^\omega$. We write PROP$(\mathcal{M})$ to denote the set of atomic propositions occurring in the image of L. When $\mathcal{M}$ is finite, its size $|\mathcal{M}|$ is defined as the sum

$$\mathrm{card}(S) + \mathrm{card}(R) + \sum_{s \in S} \mathrm{card}(L(s)).$$

The *local* model-checking problem for LTL consists in checking whether all computations in Traces$(\mathcal{M}, s)$ satisfy a given LTL formula. More formally, the *(universal) local model-checking problem for* LTL, denoted by MC$^\forall$(LTL), is defined as follows:

input: an LTL formula φ, a finite interpreted transition system $\mathcal{M}$ and a state $s \in S$,

question: Is it the case that $\mathcal{M}, s \models_\forall \varphi$?.

Without any loss of generality, in the above statement we can assume that the codomain of L is restricted to PROP(φ). The assumption that R is total is not really essential but it allows to simplify some technicalities.

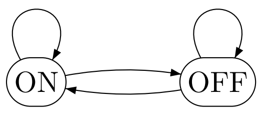
We leave to the reader the proof of the lemma below.

Lemma 4.1.2. $\mathcal{M}, s \models_\forall \varphi$ iff $\mathrm{Traces}(\mathcal{M}, s) \cap \mathrm{Mod}(\neg\varphi) = \emptyset$.

The dual, existential notion of truth of LTL-formulae leads to the *existential local model-checking*, denoted by MC$^\exists$(LTL). We write $\mathcal{M}, s \models_\exists \varphi$ if $\sigma \models \varphi$ for some $\sigma \in \mathrm{Traces}(\mathcal{M}, s)$. Likewise, the following lemma holds:

Lemma 4.1.3. $\mathcal{M}, s \models_\exists \varphi$ iff $\mathrm{Traces}(\mathcal{M}, s) \cap \mathrm{Mod}(\varphi) \neq \emptyset$.

We present below a simple interpreted transition system in which ON and OFF are propositional variables and we identify them with states where they hold respectively.



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

- $\star$ $\mathcal{M}, \mathrm{OFF} \models_\forall \mathsf{F}^\infty\mathrm{ON} \vee \mathsf{F}^\infty\mathrm{OFF}$,
- $\star$ $\mathcal{M}, \mathrm{OFF} \not\models_\forall \mathsf{F}^\infty\mathrm{ON}$ and $\mathcal{M}, \mathrm{OFF} \not\models_\forall \mathsf{F}^\infty\mathrm{OFF}$,
- $\star$ $\mathcal{M}, \mathrm{ON} \models_\exists \mathsf{F}^\infty\mathrm{ON} \wedge \mathsf{F}^\infty\mathrm{OFF}$,
- $\star$ $\mathcal{M}, \mathrm{ON} \models_\exists \neg\mathsf{F}^\infty\mathrm{OFF}$,
- $\star$ $\mathcal{M}, \mathrm{ON} \models_\exists \mathsf{G}(\mathrm{ON} \Rightarrow \mathsf{XX}\ \mathrm{OFF})$.

While the local model-checking problems concern truth of a formula at a state, the *global* model-checking problems are about computing the set of all states where the given formula is true. Formally the *(universal) global model-checking problem for* LTL, denoted by $\mathrm{GMC}^{\forall}(\mathrm{LTL})$, is defined as follows:

input: an LTL formula φ, and a finite interpreted transition system $\mathcal{M}$.

output: the set of states s such that $\mathcal{M}, s \models_{\forall} \varphi$?.

The existential global model-checking problem (which will not be discussed further) is defined likewise.

Global model-checking of an LTL-formula φ in an ITS $\mathcal{M}$ can be reduced in a straightforward way to local model-checking, by testing the truth of φ independently at each state of $\mathcal{M}$. However, this approach is apparently inefficient, because the same work would have to be done many times. A more efficient approach would be to do simultaneous model-checking of the subformulae of φ at all states of $\mathcal{M}$ and reuse the results efficiently.

For more details on LTL model-checking and satisfiability testing, see e.g., [KM08] and [BK08].

4.2 Relating model-checking and validity

Even though the decision problems related to model-checking and validity are quite different, in this section we shall provide simple reductions between them, possibly at exponential cost. In the subsequent chapters dealing with the automata-based approach, we shall see that indeed these problems can be solved uniformly.

In order to reduce validity to model-checking it is sufficient to consider interpreted transition systems whose set of traces is precisely the set of all LTL models.

Lemma 4.2.1. There is a reduction from VAL(LTL) to $\mathrm{MC}^{\forall}(\mathrm{LTL})$.

Proof: Let φ be a formula built over the propositional variables $\mathrm{p}_1, \ldots, \mathrm{p}_n$. We write $\mathcal{M}_n = (S, R, L)$ to denote the complete interpreted transition system such that $S = \mathcal{P}(\{\mathrm{p}_1, \ldots, \mathrm{p}_n\})$, $R = S \times S$ and L is the identity. Then φ is valid if and only if for all $s \in S$, we have $\mathcal{M}_n, s \models_{\forall} \varphi$. QED

Observe that the above reduction is not in logarithmic space and it is not a many-one reduction, namely in order to solve an instance of VAL(LTL) we may need to solve many instances of $\mathrm{MC}^{\forall}(\mathrm{LTL})$. By contrast, the reduction below in the other direction will be a logarithmic space many-one reduction.

Proposition 4.2.2. There is a logarithmic space reduction from $\mathrm{MC}^{\forall}(\mathrm{LTL})$ to VAL(LTL).

Proof: We follow the argument presented in [SC85, page 740]. Let $\mathcal{M} = (S, R, L)$ be a finite and total interpreted transition system and φ be an LTL formula built over the propositional variables $\mathrm{p}_1, \ldots, \mathrm{p}_k$. In this proof, we make use of the countably infinite amount of atomic propositions since to to each state s in S, we associate a new propositional variable p_s and we encode the valuation $L(s)$ by the formula AP_s below:

$$AP_s \overset{\text{def}}{=} \bigwedge\{\mathrm{p}_i : 1 \leq i \leq k, \mathrm{p}_i \in L(s)\} \wedge \bigwedge\{\neg\mathrm{p}_i : 1 \leq i \leq k, \mathrm{p}_i \notin L(s)\}.$$

For each state s, we encode the non-empty set $R(s)$ of direct successors by the formula:

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

Every state s in $\mathcal{M}$ is encoded by the formula

$$\varphi_s \overset{\text{def}}{=} AP_s \Rightarrow (AP_s \wedge Next_s).$$

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

$$\varphi_{\mathcal{M}} \overset{\text{def}}{=} \mathsf{G}(\bigwedge\{\varphi_s : s \in S\} \wedge \mathrm{UNI})$$

where UNI is a propositional formula (without temporal operators) that states that a unique atomic proposition from $\{\mathrm{p}_s : s \in S\}$ is satisfied at the current state. One can show that $\mathcal{M}, s \models_{\forall} \varphi$ iff $(\varphi_{\mathcal{M}} \wedge \mathrm{p}_s) \Rightarrow \varphi$ is valid. QED

The proof of Proposition 4.2.2 uses a rather standard approach that consists in reducing a question of the form "$\mathcal{M} \models \varphi$?" to satisfiability/validity of $\psi_{\mathcal{M}} \wedge \varphi$ where $\psi_{\mathcal{M}}$ encodes the model $\mathcal{M}$. In order to answer this second question, deductive methods can be used.

4.3 Ultimately periodic model property for LTL-formulae

In this section, we state and discuss a fundamental result about LTL that is the core of most algorithms to solve decision problems about LTL (and many of its extensions), viz., that if an LTL formula has a model then it has an ultimately periodic model. For a proof the reader is referred to [SC85].

Theorem 4.3.1. [SC85, Theorem 4.7] For every satisfiable LTL formula φ, there is an ultimately periodic structure σ such that $\sigma \models \varphi$, its period is bounded by $|\varphi| \times 2^{4|\varphi|}$ and its prefix index is bounded by $2^{4|\varphi|}$.

Existence of ultimately periodic models for satisfiable LTL formulae is a major step towards designing decision procedures for LTL decision problems. Furthermore, such models can be encoded as small sequences of sets of subformulae satisfying local conditions and fairness conditions, thus keeping the space needed for the verification of the existence of such models polynomially bounded by the size of the formula. In the rest of this section we sketch such an encoding.

Given an LTL formula φ, we write $cl_{\mathrm{LTL}}(\varphi)$ to denote the least set of formulae closed under subformulae and negations (double negations are eliminated) containing φ and such that $\psi_1 \mathsf{U} \psi_2 \in cl_{\mathrm{LTL}}(\varphi)$ implies $\mathsf{X}(\psi_1 \mathsf{U} \psi_2) \in cl_{\mathrm{LTL}}(\varphi)$. It is a routine to check that $\mathrm{card}(cl_{\mathrm{LTL}}(\varphi)) \leq 4|\varphi|$.

Given a structure σ and a formula φ, we write $cl_{\mathrm{LTL}}(\varphi, \sigma, i)$ to denote the set of subformulae of $cl_{\mathrm{LTL}}(\varphi)$ that hold true at position i, i.e. $cl_{\mathrm{LTL}}(\varphi, \sigma, i) = \{\psi \in cl_{\mathrm{LTL}}(\varphi) : \sigma, i \models \psi\}$.

A subset $\Gamma \subseteq cl_{\mathrm{LTL}}(\varphi)$ is *maximally consistent* (with respect to φ) whenever

- ⋆ for every $\psi_1 \wedge \psi_2 \in cl_{\rm LTL}(\varphi)$, $\psi_1 \wedge \psi_2 \in \Gamma$ iff $\psi_1, \psi_2 \in \Gamma$,
- ⋆ for every $\psi_1 \vee \psi_2 \in cl_{\rm LTL}(\varphi)$, $\psi_1 \vee \psi_2 \in \Gamma$ iff $\psi_1 \in \Gamma$ or $\psi_2 \in \Gamma$,
- ⋆ for every $\neg\psi \in cl_{\rm LTL}(\varphi)$, $\neg\psi \in \Gamma$ iff $\psi \notin \Gamma$,
- ⋆ for every $\psi_1 \mathsf{U} \psi_2 \in cl_{\rm LTL}(\varphi)$, $\psi_2 \in \Gamma$ or $\psi_1, \mathsf{X}(\psi_1 \mathsf{U} \psi_2) \in \Gamma$.

The proof of the following two lemmas are routine.

Lemma 4.3.2. Let σ be a structure, φ be an LTL formula and $i \geq 0$. Then, $cl_{\rm LTL}(\varphi, \sigma, i)$ is maximally consistent.

A pair (Γ_1, Γ_2) of subsets of $cl_{\rm LTL}(\varphi)$ is *one-step consistent* (with respect to φ) iff for every $\mathsf{X}\psi \in cl_{\rm LTL}(\varphi)$, $\mathsf{X}\psi \in \Gamma_1$ iff $\psi \in \Gamma_2$.

Lemma 4.3.3. Let σ be a structure, φ be an LTL formula and $i \geq 0$. Then, $(cl_{\rm LTL}(\varphi, \sigma, i), cl_{\rm LTL}(\varphi, \sigma, i+1))$ is one-step consistent.

Definition 4.3.1. [**Small satisfiability witness**] Let φ be an LTL formula. A *small satisfiability witness* for φ is a finite sequence $\Gamma_0, \ldots, \Gamma_i, \ldots, \Gamma_{i+p}$ of subsets of $cl_{\rm LTL}(\varphi)$ such that

1. $0 \leq i \leq 2^{4|\varphi|}$ and $0 \leq p \leq |\varphi| \times 2^{4|\varphi|}$ (i is just a distinguished position)
2. each Γ_j is maximally consistent, $\Gamma_i = \Gamma_{i+p}$ and $\varphi \in \Gamma_0$,
3. for $0 \leq j < i + p$, (Γ_j, Γ_{j+1}) is one-step consistent,
4. if an U-formula $\psi_1 \mathsf{U} \psi_2$ belongs to $\bigcup_{i \leq j < i+p} \Gamma_j$ then ψ_2 belongs to $\bigcup_{i \leq j < i+p} \Gamma_j$.

∇

Condition 1 guarantees that the sequence is not too large whereas Conditions 2 and 3 ensure local consistency as well as initial and final conditions. Combining them with Condition 4 provides a fairness condition.

Theorem 4.3.1 implies that if an LTL-formula is satisfiable then it has a small satisfiability witness. The converse is also true.

Lemma 4.3.4. An LTL formula is satisfiable iff it has a small satisfiability witness.

For a proof, see [SC85].

Most methods to check satisfiability of an LTL formulae attempt to find a small satisfiability witness or a variant. For instance, a brute force algorithm consists in generating all the sequences of subsets of $cl_{\rm LTL}(\varphi)$ of length at most $2^{4|\varphi|} + |\varphi| \times 2^{4|\varphi|}$ and checking whether one of the sequences is a small satisfiability witness. This provides a double exponential-time decision procedure for LTL satisfiability. Alternatively, one can guess a sequence of length at most $2^{4|\varphi|} + |\varphi| \times 2^{4|\varphi|}$ and check on-the-fly that it is a small satisfiability witness. This can be done in nondeterministic polynomial space as shown in [SC85]. By Savitch's Theorem [Sav70], this provides a polynomial space upper bound.

Theorem 4.3.5. [SC85] LTL satisfiability is in PSPACE.

The (existential) model-checking problem for LTL can be solved in polynomial space using a similar argument. Indeed, each position of the small satisfiability witness can be augmented with a state of the transition system and the one-step consistency condition further requires that two states in successive positions belong to the accessibility relation of the transition system. The only other essential difference is that the maximal values for the prefix index and the period have an additional factor: the number of states of the transition system.

Theorem 4.3.6. [SC85] Existential model-checking problem for LTL is in PSPACE.

Since a satisfiable LTL formula may have more than one small satisfiability witness, it makes sense to develop different methods to generate as efficiently as possible at least one witness. The goal of the tableau-based approach introduced in [Wol85] consists in generating a tableau that encodes all the small satisfiability witnesses for a given formula by decomposing subformulae on demand (the fairness condition needs to be part of the final check). Similarly, in the automata-based approach [VW94], to each formula we associate a Büchi automaton that accepts exactly the models of the formula and checking the nonemptiness of this automaton amounts to find a small satisfiability witness for the formula. Hence, the methods for checking LTL satisfiability (and model-checking) adopt distinct strategies and heuristics to build such small satisfiability witnesses when they exist.

4.4 A note on extensions of LTL

Various extensions of LTL have been proposed and studied. The most notable of them include LTL with past operators [LPZ85], [LS00], [LMS02], [Mar02], Wolper's Extended Temporal Logic ETL [Wol83], [VW94], and the linear mu-calculus [Var88].

Chapter 5

Addendum: Model-checking and satisfiability testing of branching-time logics

5.1 Reducing model-checking of CTL* to model-checking of LTL

As first shown in [EL87], model-checking of CTL*-formulae can be reduced to global model-checking of LTL-formulae, inductively on the path quantifier rank (the maximal number of nested path quantifiers) of the formula as follows. First, note that the CTL*-formulae with path quantifier rank 0 are precisely the LTL-formulae. Now, given any CTL*-formula φ and a CTL*-model $\mathcal{M}$, we identify the maximal state subformulae $\psi_1, \dots, \psi_n$ of φ and replace them uniformly with new atomic propositions $p_1, \dots, p_n$. Note that each ψ_i is a boolean combination of atomic propositions and formulae of the type $\forall\theta$ where θ is a CTL*-formula of path quantifier rank less than the one of φ. Besides, the result of substitution of $p_1, \dots, p_n$ respectively for $\psi_1, \dots, \psi_n$ in φ is an LTL-formula φ'. Using the inductive argument, we can globally model check each of $\psi_1, \dots, \psi_n$ and compute the sets of states $X_1, \dots, X_n$ in the model $\mathcal{M}$ where each of them is true. Then we modify $\mathcal{M}$ to a model $\mathcal{M}'$ by assigning the atomic propositions $p_1, \dots, p_n$ to be true respectively in the sets of states $X_1, \dots, X_n$. Now, the global model-checking of φ in $\mathcal{M}$ is reduced to the global model-checking of the LTL-formula φ' in $\mathcal{M}'$. For more details on CTL* model-checking see also [BK08].

The procedure outlined above gives a PSPACE-complete method for model-checking of CTL*, which is optimal since LTL is a fragment of CTL*.

As an alternative approach, an efficient on-the-fly method for CTL* model-checking has been developed in [BCG95].

5.2 Polynomial time model-checking algorithm for CTL

The reduction method described above applies, in particular, to CTL, but it turns out that model-checking of CTL-formulae can be done much more efficiently. The model-checking problem for CTL was first discussed and solved optimally in [CE81], where a model-checking algorithm for CTL was developed that works in time linear both in the size of the model and in the size of the formula, which is a strong argument to use CTL as a formal language for specification and verification. We will present that algorithm in detail here.

Formally, the *local model-checking problem for* CTL, LMC(CTL), can be defined as follows:

input: a CTL formula φ, a finite ITS $\mathcal{M}$, and $s_0 \in \mathcal{M}$,

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

The *global model-checking problem* GMC(CTL):

input: a CTL formula φ and a finite ITS $\mathcal{M}$.

output: the set $[\![\varphi]\!]_{\mathcal{M}} = \{s \in \mathcal{M} : \mathcal{M}, s \models \varphi\}$.

Proposition 5.2.1. Let $\mathcal{M} = (W, R, L)$ be a CTL model and φ be a CTL formula. Computing $[\![\varphi]\!]_{\mathcal{M}}$ can be done in time $\mathcal{O}((\mathrm{card}(R) + \mathrm{card}(W)) \times |\varphi|)$.

In the proposition above, we assume that the codomain of L is $\mathcal{P}(\mathrm{PROP}(\varphi))$ where $\mathrm{PROP}(\varphi)$ is the set of propositional variables occurring in φ.

Proof: Let $\mathcal{M} = (W, R, L)$ be a CTL model and φ be a CTL formula. The directed graph (W, R) 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 $\mathrm{card}(W)$: the ith element contains the indices of the propositional variables that hold true at the ith state of W (arbitrary orderings of variables and states). The size of L is in $\mathcal{O}(\mathrm{card}(W) \times |\varphi|)$.

Let $\varphi_1, \ldots, \varphi_k$ be the subformulae of φ ordered by increasing size. In case of conflict, we make an arbitrary choice for the formulae of identical sizes. Consequently,

- ⋆ $\varphi_k = \varphi$,
- ⋆ φ_1 is a propositional variable,
- ⋆ if φ_i is a strict subformula of φ_j, then $i < j$,
- ⋆ $k \leq |\varphi|$.

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

1. for every $i \in \{1, \ldots, k\}$, either $\varphi_i \in l(s)$, or $\neg\varphi_i \in l(s)$, but not both at the same time,
2. for every $\psi \in \{\varphi_1, \ldots, \varphi_k, \neg\varphi_1, \ldots, \neg\varphi_k\}$, $\psi \in l(s)$ iff $\mathcal{M}, s \models \psi$.

For all $i \in \{1, \ldots, k\}$ and $s \in W$, we insert either φ_i in $l(s)$ or $\neg\varphi_i$ 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 φ_i for every $i \in \{1, \ldots, k\}$. Each step requires time in $\mathcal{O}(\mathrm{card}(W) + \mathrm{card}(R))$.

For technical convenience, here we will assume that the primitive temporal operators in CTL are the $\exists$X-prefixed ones.

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

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

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

Case 4: $\varphi_i = \exists \mathsf{X}\varphi_{i_1}$ for some $i_1 < i$.
For every $s \in W$, if there is $s' \in R(s)$ such that φ_{i_1} is in $l(s')$, then insert φ_i in $l(s)$, otherwise insert $\neg\varphi_i$ in $l(s)$.

Case 5: $\varphi_i = \exists(\varphi_{i_1} \mathsf{U} \varphi_{i_2})$ for some $i_1, i_2 < i$.
One can show that $\mathcal{M}, s \models \varphi_i$ iff there is a R-path $s_0 \to s_1 \to \ldots \to s_n$ in $\mathcal{M}$ such that

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

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

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

Let $\mathcal{M}' \stackrel{\text{def}}{=} (W', R', L')$ 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 φ_i in $l(s)$ otherwise insert $\neg\varphi_i$ in $l(s)$. In order to show that this step requires time in $\mathcal{O}(\text{card}(W) + \text{card}(R))$, we use the following result from graph theory (see e.g., [AHU74, AHU83]):

Lemma 5.2.2. Let $G = (W, R)$ 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: $\varphi_i = \exists \mathsf{G}\varphi_{i_1}$ for some $i_1 < i$.
As in the previous case, we define

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

Let $\mathcal{M}' = (W', R', L')$ be the restriction of $\mathcal{M}$ to W'. We show that for every $s \in W$, $\mathcal{M}, s \models \varphi_i$ 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 (W', R').

A non-trivial SCC C for (W', R') 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 card(C) > 1 or ($C = \{s'\}$ and $(s', s') \in R'$) (C non-trivial).

Suppose that $\mathcal{M}, s \models \exists \mathsf{G} \varphi_{i_1}$. Obviously, $s \in W'$. Hence, there is an infinite path $s_0, s_1, \ldots$ such that $s_0 = s$ and for every $j \geq 0$, $\mathcal{M}, s_j \models \varphi_{i_1}$. 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 φ_{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 $\mathcal{M}, s \models \varphi_i$.

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 $\mathcal{O}(\mathrm{card}(W) + \mathrm{card}(R))$, we need to use the following result from graph theory.

Lemma 5.2.3. Let $G = (W, R)$ be a directed graph represented by lists of neighbours. Computing the partition of maximal SCC can be computed in time $\mathcal{O}(\mathrm{card}(W) + \mathrm{card}(R))$.

See [AHU74, AHU83] for a proof.

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

QED

Corollary 5.2.4. LMC(CTL) and GMC(CTL) are in PTime.

Proposition 5.2.5. LMC(CTL) is PTime-hard.

Proof: PTime-hardness of LMC(CTL) can be shown be reducing to it satisfiability for synchronized alternating monotonous Boolean circuits, that is a PTime-complete problem, see e.g. [LMS01]. QED

5.3 Tree-model property of CTL*

5.3.1 Tree unfoldings of models for CTL*

Using proposition **??** and **??** we can obtain the following (note that paths in $\mathcal{M}$ are *states* in $\widehat{\mathcal{M}}$).

Corollary 5.3.1. For any interpreted transition system $\mathcal{M}$ and a path $\pi \in \mathcal{M}$, we have that:

1. $(\widehat{\mathcal{M}}, \pi) \equiv_{\mathrm{StateFor}} (\mathcal{M}, \pi(0))$.
2. $(\widehat{\mathcal{M}}, \widehat{\pi}) \equiv_{\mathrm{PathFor}} (\mathcal{M}, \pi)$, where $\widehat{\pi}$ is the path in $\widehat{\mathcal{M}}$ corresponding to the path π in $\mathcal{M}$.

5.3.2 Uniformly branching trees

Thus, we see that the semantics of CTL* (standard and general) can be restricted to models on tree-like transition systems. In fact, it turns out that a simple type of trees is sufficient.

Definition 5.3.1. [**ω-trees**]

A tree-like transition system is called an *ω-tree* if every maximal path in it has the order type ω of the natural numbers.

Note that every unfolding of a transition system with a serial transition relation is a disjoint union of ω-trees. ∇

Definition 5.3.2. [**Finitely branching transition systems**]

A transition system is *finitely branching* if every state in it has finitely many successors.

A *branching factor* of a finitely branching transition system $\mathcal{T}$ is the supremum of the cardinalities of all sets of successors of states in $\mathcal{T}$. If that branching factor is finite, $\mathcal{T}$ is said to be *boundedly branching.*

For any cardinal number (finite or infinite) κ, a transition system $\mathcal{T}$ is called *κ-branching* if every state has κ successors[1].

We call κ-branching trees *uniformly branching.* ∇

Clearly, every κ-branching ω-tree is unique up to isomorphism. Such a tree can be represented in a canonical way by ordering all successors of a node with the ordinals in κ and labelling every node with a finite string of such ordinals marking the path from the root to that node. For instance, the nodes of a k-branching tree are labelled with the set of all finite strings on $[k] = \{0, \dots, k-1\}$, as follows: the root is labelled by the empty string ε and the successors of a node labelled by ς are $\varsigma 0, \dots, \varsigma(k-1)$.

We call the resulting labelled tree the *canonical κ-branching ω-tree*, denoted by $[\kappa]^*$. Note that every path in $[\kappa]^*$ can be represented as a mapping $\pi : \omega \Rightarrow \kappa$, by assigning to every node from the path the last term of the string representing it. Conversely, every such mapping defines a path in $[\kappa]^*$.

Respectively, a computation in a $[\kappa]^*$ is a mapping $\omega \Rightarrow \mathbf{2}^{\mathrm{PROP}}$.

[1]Readers unfamiliar with infinite cardinals can think that κ is a natural number of ω.

Proposition 5.3.2. For every rooted interpreted transition system $(\mathcal{M}, r)$ with a branching factor κ there is a bisimilar interpreted system based on $[\kappa]^*$, relating r with ε.

Proof: *Sketch:* First, take the unfolding of $\mathcal{M}$ from r. It is an ω-tree with a branching factor κ. Then, starting from the root, level by level, add as many copies of existing successors together with the subtrees rooted at them, as necessary to make the tree exactly κ-branching. QED

5.3.3 Satisfiability of CTL* in k-branching trees

Corollary 5.3.3. Every state formula φ of CTL*, satisfiable in a model with a branching factor κ is satisfiable in a model based on $[\kappa]^*$.

Theorem 5.3.4. [ES84, Theorem 3.2] Every satisfiable state formula φ of CTL* is satisfiable in a k-branching ω-tree, for $k \leq m+1$, where m is the number of path quantifiers occurring in φ.

Proof: *Sketch:* Structural induction on φ. Take an ω-tree satisfying φ and, starting from the root, level by level prune all 'unnecessary' successors. For a proof see [ES84], though some important details are missing there. QED

5.4 Decidability and complexity of satisfiability testing for CTL* and CTL

Theorem 5.4.1. [CE81] The satisfiability problem for CTL is ExpTime-complete.

Proof: *Sketch:* The upper bound is provided by the tableau construction that will be described later. The matching lower bound is by reduction from alternating polynomial space bounded Turing machines, similar to the proof for PDL in [KT90]. QED

Theorem 5.4.2. The satisfiability problem for CTL* is 2ExpTime-complete.

Proof: *Sketch:* The complexity lower bound is established in [VS85] by reduction from alternating exponential space bounded Turing machines, whereas the upper bound is proved in [EJ00], where Emerson and Jutla produce a deterministic double exponential time algorithm for deciding satisfiability for CTL* by an elaborated reduction to non-emptiness of automata on infinite trees. QED

5.5 Linear-time vs branching-time Logics

The debate on the pros and cons of using linear vs branching time temporal logics has been alive and unabated since the beginning of the 1980's; see [Lam80] and [EH86] for the beginning of it. Of course, the linear and branching time approaches have somewhat different scopes of application, viz. linear time approach is the more natural when the properties to be checked are about a single computation in a transition system, while the branching time approach is more natural to reason about all computations, i.e., about the global structure of the transition system. Yet, for many purposes both approaches seem to compete hard, and there has been a quest for theoretical argumentation of the superiority of one approach to the other. The major types of arguments brought to the battlefield are:

- ⋆ *expressiveness*,
- ⋆ *complexity of satisfiability*,
- ⋆ *complexity of model checking*,
- ⋆ *suitability for specific practical purposes.*

Here we discuss briefly the first three.

Regarding expressiveness, it should be intuitively clear that LTL and CTL are *incompatible* in their expressive powers. Indeed, this intuition can be turned into a precise statement:

Proposition 5.5.1.

1. The CTL-formula $\forall \mathsf{F} \forall \mathsf{G} p$ is not expressible in LTL.
2. The LTL-formula $\forall \mathsf{F} \mathsf{G} p$ is not expressible in CTL.

Proof: *Sketch:* The key to the proof of these is the observation that with every CTL* formula φ, one can associate the LTL-formula LTL(φ) obtained from φ by deleting all path quantifiers. Now, show that for every path π in a CTL*-model $\mathcal{M}$, we have that $\pi \models \text{LTL}(\varphi)$ iff $\mathcal{M}_\pi, \pi \models \varphi$, where $\mathcal{M}_\pi$ is the CTL*-model obtained by restricting $\mathcal{M}$ over the path π. See [Eme90] for details. QED

Thus, so far it seems that the ideal solution is CTL* which subsumes both rivals. However, the second type of arguments comes in against this choice: the computational price demanded by the complexity of CTL* (deterministic 2ExpTime) is virtually prohibiting for practical use. Indeed, the deterministic ExpTime completeness of the complexity of satisfiability of CTL is already high enough to put many customers off, but CTL has other virtues to compensate for this, notably the bilinear complexity of model checking. On the other hand, LTL, while having better than complexity of the satisfiability (PSpace) fares worse when it comes to model checking.

All these suggest that, so far, there are no decisive theoretical arguments in favour of one approach to the other, so the practical consideration play a leading role here, and, as Vardi puts it in [Var98], "the debate might end up being decided by the market-place rather than by the research community".

For the latest on the discussion, see [Var01], [KV05] [NV07].

Bibliography

[Abr79] K. Abrahamson. Modal logic of concurrent nondeterministic programs. In *Lecture Notes in Computer Science*, volume 70, pages 21–33. Springer, 1979.

[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.

[BAPM81] M. Ben-Ari, A. Pnueli, and Z. Manna. The temporal logic of branching time. In *Proc. 8th ACM Symposium on Principles of Programming Languages, also appeared in Acta Informatica, 20(1983), 207-226*, pages 164–176, 1981.

[BCG88] Michael C. Browne, Edmund M. Clarke, and Orna Grumberg. Characterizing finite kripke structures in propositional temporal logic. *Theor. Comput. Sci.*, 59:115–131, 1988.

[BCG95] Girish Bhat, Rance Cleaveland, and Orna Grumberg. Efficient on-the-fly model checking for ctl*. In *LICS*, pages 388–397. IEEE Computer Society, 1995.

[BHWZ04] Sebastian Bauer, Ian M. Hodkinson, Frank Wolter, and Michael Zakharyaschev. On non-local propositional and weak monodic quantified ctl. *J. Log. Comput.*, 14(1):3–22, 2004.

[BK08] Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking.* The MIT Press, 2008.

[CE81] E.M. Clarke and E.A. Emerson. Design and synthesis of synchronisation skeletons using branching time temporal logic. In D. Kozen, editor, *Logics of Programs*, pages 52–71. Springer, 1981.

[CES83] E. Clarke, E. A. Emerson, and A. Sistla. Automatic verification of finite state concurrent systems using temporal logic specifications: A practical approach. In *Conference Record of the Tenth Annual ACM Symposium on Principles of Programming Languages*, pages 117–126, Austin, Texas, January 1983.

[CES86] E. Clarke, E.A. Emerson, and A.P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, April 1986.

[CNP94] H. Calbrix, M. Nivat, and A. Podelski. Ultimately periodic words of rational ω-languages. In *MFPS'93*, volume 802 of *Lecture Notes in Computer Science*, pages 554–566. Springer, 1994.

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

[EC80] E. Allen Emerson and Edmund M. Clarke. Characterizing correctness properties of parallel programs using fixpoints. In J. W. de Bakker and Jan van Leeuwen, editors, *ICALP*, volume 85 of *Lecture Notes in Computer Science*, pages 169–181. Springer, 1980.

[EF06] C. Eisner and D. Fisman. *A Practical Introduction to PSL.* Springer, 2006.

[EH82] E. A. Emerson and J. Y. Halpern. Decision procedures and expressiveness in the temporal logic of branching time. In *Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing*, pages 169–180, San Francisco, California, 5–7 May 1982.

[EH83] E. Allen Emerson and Joseph Y. Halpern. "sometimes" and "not never" revisited: On branching versus linear time. In *POPL*, pages 127–140, 1983.

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

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

[EJ00] E.A. Emerson and C.S. Jutla. The complexity of tree automata and logics of programs. *SIAM Journal of Computing*, 29(1):132–158, 2000.

[EL87] A. Emerson and C.-L. Lei. Modalities for model checking: branching time logic strikes back. *Science of Computer Programming*, 8(3):275–306, 1987.

[Eme90] E.A. Emerson. Temporal and modal logics. In Leeuwen [Lee90], pages 995–1072.

[ES84] A. Emerson and P. Sistla. Deciding full branching time logic. *Information and Control*, 61:175–201, 1984.

[Fra86] N. Francez. *Fairness.* Springer-Verlag, NY, 1986.

[GHR94] D.M. Gabbay, I. Hodkinson, and M. Reynolds. *Temporal Logic: Mathematical Foundations and Computational Aspects.* Oxford University Press, 1994.

[GPSS80] D. Gabbay, A. Pnueli, S. Shelah, and J. Stavi. On the temporal analysis of fairness. In *7th Annual ACM Symposium on Principles of Programming Languages*, pages 163–173. ACM Press, 1980.

[HKT00] David Harel, Dexter Kozen, and Jerzy Tiuryn. *Dynamic Logic.* MIT Press, 2000.

[HM80] M. Hennessy and R. Milner. On observing nondeterminism and concurrency. In *ICALP'80*, volume 85 of *Lecture Notes in Computer Science*, pages 299–309. Springer, 1980.

[Hol97] G. Holzmann. The model checker SPIN. *IEEE Transactions on Software Engineering*, 23(5):279–295, 1997.

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

[KG96] Orna Kupferman and Orna Grumberg. Buy one, get one free!!! *J. Log. Comput.*, 6(4):523–539, 1996.

[KM08] Fred Krger and Stephan Merz. *Temporal Logic and State Systems.* 2008.

[KT90] D. Kozen and J. Tiuryn. Logics of programs. In Leeuwen [Lee90], pages 789–840.

[KV05] Orna Kupferman and Moshe Y. Vardi. From linear time to branching time. *ACM Trans. Comput. Log.*, 6(2):273–294, 2005.

[KVW00] O. Kupferman, M. Vardi, and P. Wolper. An automata-theoretic approach to branching-time model checking. *Journal of the Association for Computing Machinery*, 47(2):312–360, 2000.

[Lam80] L. Lamport. Sometimes is sometimes "not never"- on the temporal logic of programs. In *Proc. of the 7th Annual ACM Symp. on Principles of Programming Languages*, pages 174–185, 1980.

[Lee90] J. van Leeuwen, editor. *Handbook of Theoretical Computer Science*, volume B: Formal Models and Semantics. Elsevier, 1990.

[LMS01] F. Laroussinie, N. Markey, and Ph. Schnoebelen. Model checking ctl$^+$ and fctl is hard. In *4th International Conference on Foundations of Software Science and Computation Structures, Berlin, Germany*, pages 318–331. volume 2030 of Lecture Notes in Computer Science, Springer-Verlag, 2001.

[LMS02] F. Laroussinie, N. Markey, and Ph. Schnoebelen. Temporal logic with forgettable past. In *LICS'02*, pages 383–392. IEEE, 2002.

[LPZ85] O. Lichtenstein, A. Pnueli, and L. Zuck. The glory of the past. In *Brooklyn College Conference on Logics of Programs*, volume 193 of *Lecture Notes in Computer Science*, pages 196–218. Springer, 1985.

[LS00] F. Laroussinie and Ph. Schnoebelen. Specification in ctl + past for verification in ctl. *Information and Computation*, 156:236–263, 2000.

[Mar02] N. Markey. Past if for free: on the complexity of verifying linear temporal properties with past. In *9th Int. Workshop on Expressiveness in Concurrency (EXPRESS'02)*, volume 68 of *Electronic Notes in Theoretical Computer Science.* Elsevier, 2002.

[McM93] K. McMillan. *Symbolic Model Checking.* Kluwer Academic Publishers, 1993.

[MP79] Z. Manna and A. Pnueli. The modal logic of programs. In *ICALP'79*, volume 71 of *Lecture Notes in Computer Science*, pages 385–409. Springer, 1979.

[MP81] Z. Manna and A. Pnueli. Verification of concurrent programs: The temporal framework. In R. Boyer and J. Moore, editors, *The Correctness Problem in Computer Science*, pages 215–273, London, 1981. Academic Press.

[MP92] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification.* Springer-Verlag, New York, 1992.

[MP95] Z. Manna and A. Pnueli. *Temporal verification of reative systems: safety.* Springer-Verlag, New York, 1995.

[MS03] N. Markey and Ph. Schnoebelen. Model checking a path. In *CONCUR'03*, volume 2761 of *Lecture Notes in Computer Science*, pages 251–261. Springer, 2003.

[NV07] Sumit Nain and Moshe Y. Vardi. Branching vs. linear time: Semantical perspective. In Kedar S. Namjoshi, Tomohiro Yoneda, Teruo Higashino, and Yoshio Okamura, editors, *ATVA*, volume 4762 of *Lecture Notes in Computer Science*, pages 19–34. Springer, 2007.

[Plo81] G. Plotkin. A structural approach to operational semantics. Tech. report daimi fn-19, Aarhus Univ., 1981.

[Pnu77a] A. Pnueli. The temporal logic of programs. In *Proc. 18th Symp. Foundations of Computer Science*, pages 46–57, 1977.

[Pnu77b] A. Pnueli. The temporal logic of programs. In *FOCS'77*, pages 46–57. IEEE Computer Society Press, 1977.

[Pnu79] A. Pnueli. The temporal semantics of concurrent programs. In *International Symposium on Semantics of Concurrent Computation 1979*, volume 70 of *Lecture Notes in Computer Science*, pages 1–20. Springer, 1979.

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

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

[Sav70] W.J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, 4(2):177–192, 1970.

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

[Sti92] C. Stirling. Modal and temporal logics. In *Handbook of Logic in Computer Science*, volume 2 (Background: Computational Structures), pages 477–563. Clarendon Press, Oxford, 1992.

[SVW87] A. Sistla, M. Vardi, and P. Wolper. The complementation problem for Büchi automata with applications to temporal logic. *Theoretical Computer Science*, 49:217–237, 1987.

[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.

[Var98] M. Vardi. Linear vs branching time: A complexity-theoretic perspective. In *Proceedings, 13th Annual IEEE Symposium on Logic in Computer Science.* IEEE Computer Society Press, 1998.

[Var01] Moshe Y. Vardi. Branching vs. linear time: Final showdown. In Tiziana Margaria and Wang Yi, editors, *TACAS*, volume 2031 of *Lecture Notes in Computer Science*, pages 1–22. Springer, 2001.

[VS85] M. Vardi and L. Stockmeyer. Improved upper and lower bounds for modal logics of programs. In *STOC'85*, pages 240–251. ACM, 1985.

[VW86] M. Vardi and P. Wolper. Automata-theoretic approach to automatic program verification. In *LICS'86*, pages 322–331. IEEE, 1986.

[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.

[Wol85] P. Wolper. The tableau method for temporal logic: An overview. *Logique et Analyse*, 110–111:119–136, 1985.