Temporal logics for specification and verification

Valentin Goranko
Technical University of Denmark
Email: vfgo@imm.dtu.dk

June 2009

The course will comprise ten 40-min lectures, delivered 2 per day, with a 5-10 min break in between.

Prerequisites This is an introductory course and the participants are only expected to have some background in classical and modal logic. In addition, some knowledge of computability and complexity, finite automata, and a general idea of formal specification and verification would be an advantage.

Course notes content These course notes contain parts of draft chapters from a book in preparation on "Temporal logics in Computer Science" by Stéphane Demri and myself. Not all material included in the notes will be presented in the course. Some topics (e.g. tableaux-based methods), that are less covered in the literature, and treated in more details in these notes. Because the last topic, on automata-based methods for verification of temporal logics, is extensively covered in the literature, we have only given some references to recommended readings on it.

Additional course material More course material, including updated notes, exercises, slides, and references will be placed on the course webpage, which will be announced on ESSLLI'2009 website.

Tentative contents and schedule

NB: Depending on the actual composition and background of the audience, last minute changes of this schedule are possible.

Day 1:

Lecture 1: Introduction. Transition systems and computations. Important properties of transition systems: safety, eventualities, fairness, reachability. Formal specification and verification of properties of transition systems. The basic temporal logic for transition systems.

Lecture 2: Behavioral equivalences between transition systems. Bisimulations and bisimulation games. Bisimulation invariance.

Day 2:

Lecture 3: The linear time temporal logic LTL: syntax, semantics, expressiveness.

Lecture 4: Satisfiability and validity testing and model-checking of LTL formulae.

Day 3:

Lecture 5: Branching time temporal logics. The temporal logic of reachability TLR. The computation tree logics CTL* and CTL.

Lecture 6: Satisfiability and validity testing and model-checking of branching-time logics.

Day 4:

Lecture 7: A generic tableau method for testing satisfiability and model checking of temporal formulae.
Testing satisfiability and model checking of LTL using tableaux.

Lecture 8: Testing satisfiability and model checking of TLR and CTL using tableaux.

Day 5:

Lecture 9: Automata-based methods for satisfiability testing and model-checking of LTL.

Lecture 10: Automata-based methods for satisfiability testing and model-checking of branching-time logics.

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).

Automata-based and tableau-based methods are intimately related, while each of them has pros and cons compared to the other. Because of the conceptual elegance and technical power and convenience, the automata-based methods have been favoured by the researchers in the area and most of the tools that have been implemented are based on automata. On the other hand, tableau-based methods for model checking and satisfiability checking of temporal logics are less developed and tested for industry applications, but are more natural and intuitive from logical perspective, easier for execution by humans, and (arguably) potentially more flexible and practically efficient, if suitably optimized.

In this course we will discuss abstract transitions systems and computations in them, will introduce and study the standard linear and branching time temporal logics, will illustrate how they can be used to specify properties of transition systems, and will present tableaux-based and automata-based methods for verification of temporal formulae.

1 Lecture 1: Transition systems and computations. The basic temporal 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. [**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 $\mathcal{T} = \left\langle S, \left\{ \stackrel{a}{\longmapsto} \right\}_{a \in A} \right\rangle$ we can associate the *union transition*

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

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 1.2. [**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 1.3. [**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 $L : S \to \mathbf{2}^{\mathrm{PROP}}$ 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 1.4. [**Paths in transition systems**] A *path* in a transition system $\mathcal{T}$ is a (finite or infinite) sequence of states and actions which transform every state into its successor:

$$s_0 \stackrel{a_0}{\longmapsto} s_1 \stackrel{a_1}{\longmapsto} s_2 \dots$$

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 1.5. [**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 $s_0, s_1, s_2, \dots$, 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 $\pi(0), \pi(1), \pi(2), \dots$.

Definition 1.6. A path in a transition system is a *cycle* if its first and its last state coincide; in particular, if it is a *loop*: $s \stackrel{a}{\longmapsto} s$. ∇

Definition 1.7. A transition system is:

- $\star$ *acyclic* if it does not contain cycles;
- $\star$ *forest-like* if it is acyclic and every state has at most one predecessor state.
- $\star$ *tree-like* if it is a forest in which exactly one state, called *the root*, has no predecessor states.

∇

Definition 1.8. [**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:

$$L(s_0) \stackrel{a_0}{\longmapsto} L(s_1) \stackrel{a_1}{\longmapsto} L(s_2) \dots$$

∇

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

- $\star$ $\widehat{S}$ 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 $l(\pi)$.

- $\star$ $\pi \stackrel{a}{\Longrightarrow} \pi'$ holds if π' is a one-step extension of π along the transition a, i.e., $\pi = \pi(0) \stackrel{a_0}{\longmapsto} \pi(1) \stackrel{a_1}{\longmapsto} \cdots \pi(n)$ and $\pi' = \pi(0) \stackrel{a_0}{\longmapsto} \pi(1) \stackrel{a_1}{\longmapsto} \cdots \pi(n) \stackrel{a}{\longmapsto} \pi(n+1)$.

∇

Definition 1.10. [**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 1.11. [**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 $\pi_0 \stackrel{a_0}{\Longrightarrow} \pi_1 \stackrel{a_1}{\Longrightarrow} \pi_2 \cdots$ is a path in $\widehat{\mathcal{T}}$ iff $l(\pi_0) \stackrel{a_0}{\longmapsto} l(\pi_1) \stackrel{a_1}{\longmapsto} l(\pi_2) \cdots$ is a path in $\mathcal{T}$. Consequently, $\widehat{L}(\pi_0) \stackrel{a_0}{\Longrightarrow} \widehat{L}(\pi_1) \stackrel{a_1}{\Longrightarrow} \widehat{L}(\pi_2) \cdots$ is a computation in $\widehat{\mathcal{M}}$ iff $L(l(\pi_0)) \stackrel{a_0}{\longmapsto} L(l(\pi_1)) \stackrel{a_1}{\longmapsto} L(l(\pi_2)) \cdots$ 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:

$$concurrency = non\text{-}determinism + fairness.$$

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 temporal 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 $(\mathsf{EX}_a)_{a\in A}$, each representing a transition $a \in A$.

The inductive definition of formulae is:

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

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

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

The fragment of the logic TL_τ consisting only of constant formulae is known as Hennessy-Milner logic [HM80], here denoted as HML_τ.

Definition 1.12. 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:

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

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

∇

1.5.2 Semantics

The semantics of TL_τ 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* $\mathcal{M} = (S, \left\{\overset{a}{\longmapsto}\right\}_{a\in A}, L)$ is defined inductively as follows.

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

The derived truth definition of AX_a becomes:

- $\star$ $\mathcal{M}, s \models \mathsf{AX}_a\varphi$ if $\mathcal{M}, t \models \varphi$ for every $t \in S$ such that $s \overset{a}{\longmapsto} t$.

Some terminology:

Definition 1.13. [**Truth, validity, satisfiability**] Given an its $\mathcal{M}$, state r in $\mathcal{M}$, and a formula φ of TL, we say that φ is:

- $\star$ *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*.
- $\star$ *satisfiable in $\mathcal{M}$* if $\mathcal{M}, s \models \varphi$ for some state $s \in \mathcal{M}$.
- $\star$ *satisfiable*, if it is satisfied in some ITS.
- $\star$ *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*.
- $\star$ *valid in a class of ITS $\mathcal{C}$*, denoted $\mathcal{C} \models \varphi$, if it is valid in every ITS in $\mathcal{C}$.
- $\star$ *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 $\mathcal{T}$;
- $\star$ *valid in a transition system $\mathcal{T}$*, denoted $\mathcal{T} \models \varphi$, if it is valid in every ITS over $\mathcal{T}$.
- $\star$ *valid*, denoted $\models \varphi$, if it is valid in every ITS.

∇

Definition 1.14. [**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 1.1. The extension of a formula $\|\varphi\|_{\mathcal{M}}$ can be computed inductively on the construction of φ:

- $\|p\|_{\mathcal{M}} = \{s \mid p \in L(s)\}$;
- $\|\bot\|_{\mathcal{M}} = \emptyset$;
- $\|\varphi_1 \to \varphi_2\|_{\mathcal{M}} = (S \setminus \|\varphi_1\|_{\mathcal{M}}) \cup \|\varphi_2\|_{\mathcal{M}}$;
- $\|\,\mathsf{EX}_a\varphi\|_{\mathcal{M}} = pre(\|\varphi\|_{\mathcal{M}}) = \{s \mid R_a(s) \cap \|\varphi\|_{\mathcal{M}} \neq \emptyset\}$.

1.5.3 Standard translation of TL into $\mathbf{FO}_2$

With every type $\tau = (A, \mathrm{PROP})$ of interpreted transition systems, where $\mathrm{PROP} = \{p_0, p_1, \ldots\}$, we associate a relational first-order language FO_τ having a family of unary predicates $\{P_i\}_{p_i \in \mathrm{PROP}}$, a family of binary predicates $\{R_a\}_{a \in A}$, and a set of first-order variables $\mathrm{VAR} = \{x_0, x_1, \ldots\}$. We regard the models of FO_τ as interpreted transition systems in a natural sense: the interpretations of the binary predicates are the transition systems, and the interpretations of the unary predicates define a valuation: $V(p_i) := P_i$, and hence a state description function $L(s) := \{\, p_i \mid\ s \in P_i \,\}$[1].

Thus, we now have two notions of truth and validity in an ITS: the modal and the first-order. Wherever necessary, we will highlight the distinction by writing $\models_{\mathrm{FO}}$ to explicitly appeal to first-order semantics. In fact, truth and validity of a TL-formula in an ITS are *first-order notions*, in the following sense. The formulae of TL_τ are translated into FO_τ by means of the following *standard translation* [Ben83], parameterized with the variables from VAR:

- $\mathrm{ST}(p_i; x_j) := P_i x_j$ for every $p_i \in \mathrm{PROP}$;
- $\mathrm{ST}(\bot; x_j) := \bot$;
- $\mathrm{ST}(\varphi_1 \wedge \varphi_2; x_j) := \mathrm{ST}(\varphi_1; x_j) \wedge \mathrm{ST}(\varphi_2; x_j)$;
- $\mathrm{ST}(\mathsf{EX}_a\varphi; x_j) := \exists y(x_j R_a y \wedge \mathrm{ST}(\varphi; y))$, where y is the first variable in VAR not occurring in $\mathrm{ST}(\varphi; x_j)$.

Note that only x_j is free in $\mathrm{ST}(\varphi; x_j)$. Furthermore, it suffices to use only *two variables*, x_0 and x_1 (free or bound) in an alternating fashion in the standard translation, by amending the choice of y in the clause for $\mathrm{ST}(\mathsf{EX}_a\varphi; x_j)$ as follows: y is chosen as the first variable in $\mathrm{VAR} \setminus \{x_j\}$. This yields a translation into the *two-variable fragment* FO^2 of first-order logic. We also note, that the standard translation of any modal formula falls into the *guarded fragment* of first-order logic; for more details see e.g., [BdRV01b].

The standard translation is semantically faithful in the following sense.

Proposition 1.2. For every rooted ITS $(\mathcal{M}, s)$ and $\varphi \in \mathrm{TL}$:

$$\mathcal{M}, s \models \varphi \ \text{ iff } \ \mathcal{M} \models_{\mathrm{FO}} \mathrm{ST}(\varphi; x_0)[x_0 := s].$$

[1] Note that we use the same notation for the transition relations in ITS and for the binary predicates in the associated first-order structure; this should causes no confusion.

Proof: Exercise. QED

Thus, with a slight abuse of terminology, we can say that every formula φ of TL is logically equivalent to its standard translation $\mathrm{ST}(\varphi; x)$. Then, a natural questions arise: *which formulae $\gamma(x)$ of FO_τ are logically equivalent, in the same sense, to modal formulae from TL?* We will address that question in the next section.

While, by means of the standard translation, the semantics and validity for modal formulae in ITS are essentially first-order, validity of a modal formula in a transition system becomes a *monadic second-order* property. Indeed, paraphrasing the definition in terms of the standard translation, a modal formula φ is valid in a transition system $\mathcal{T}$ iff its standard translation is true in that transition system under *every* interpretation of the unary predicates occurring in it.

Proposition 1.3. For every pointed TS $(\mathcal{T}, s)$ and $\varphi \in \mathrm{TL}$ with atomic propositions among $p_0, \ldots, p_n$:

$$\mathcal{T}, s \models \varphi \text{ iff } \mathcal{T} \models \forall P_0 \ldots \forall P_n \mathrm{ST}(\varphi; x_0)[x_0 := s].$$

Consequently, $\mathcal{T} \models \varphi$ iff $\mathcal{T} \models \forall P_0 \ldots \forall P_n \forall x_0 \mathrm{ST}(\varphi; x_0)$.

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.

2 Lecture 2: Behavioral equivalences between transition systems. Bisimulations and bisimulation games. Bisimulation invariance.

Transition systems are abstract models of the behavior of real systems with respect to transitions between states. It is therefore very important to have a precise notion of *equivalent behavior* of transition systems, so in this lecture we address the question:

When are two transition systems to be considered behaviorally equivalent?

This question does not have a unique answer, as the notion of equivalence depends on the behavioral features of transition systems that are considered of importance for the real systems they model. Such features may involve local behaviour (pre- and post-conditions), generated paths and computations, as well as safety, liveness, fairness, etc. types of reachability properties. Accordingly, a variety of natural notions of behavioral equivalence arise and we will discuss and characterize some of them.

Perhaps the most important behavioral equivalence between two models of computation is the one that guarantees that any computational step performed in one model can be 'simulated' in the other, and vice versa. This idea is at the heart of the notion of *bisimulation*, introduced in theory of processes by Park [Par81], and independently in modal and temporal logics by van Benthem (see [Ben84]) under the name '*zig-zag relation*'.

Here we will define and discuss a variety of notions of bisimulations between transition systems, and later will relate them to the temporal logics introduced there and will establish their logical characterizations.

2.1 Bisimulations

Definition 2.1. [**Bisimulation between transition systems**]

Let $\mathcal{T}_1 = \left\langle S_1, \left(\overset{a}{\longmapsto}_1\right)_{a\in A}\right\rangle$ and $\mathcal{T}_2 = \left\langle S_2, \left(\overset{a}{\longmapsto}_2\right)_{a\in A}\right\rangle$ be two labelled transition systems of the same type. A non-empty relation $\beta \subseteq S_1 \times S_2$ is a *bisimulation between* $\mathcal{T}_1$ *and* $\mathcal{T}_2$, denoted $\mathcal{T}_1 \overset{\beta}{\rightleftarrows} \mathcal{T}_2$, if it satisfies the following conditions for every pair of states (s_1, s_2) such that $s_1 \beta s_2$ and a transition label $a \in A$:

Forth: If $s_1 \overset{a}{\longmapsto}_1 t_1$ for some $t_1 \in S_1$, then there is $t_2 \in S_2$ such that $t_1 \beta t_2$ and $s_2 \overset{a}{\longmapsto}_1 t_2$.

Back: Conversely, if $s_2 \overset{a}{\longmapsto}_1 t_2$ for some $t_2 \in S_2$, then there is $t_1 \in S_1$ such that $t_1 \beta t_2$ and $s_1 \overset{a}{\longmapsto}_1 t_1$.

If such bisimulation exists between $\mathcal{T}_1$ and $\mathcal{T}_2$ such that every state in $\mathcal{T}_1$ is linked to some state of $\mathcal{T}_2$ and vice versa, we say that $\mathcal{T}_1$ and $\mathcal{T}_2$ are *(globally) bisimulation equivalent*, or just *(globally) bisimilar*[2], denoted $\mathcal{T}_1 \rightleftarrows \mathcal{T}_2$. ∇

Definition 2.2. [**Bisimulation between rooted transition systems**]

[2]The word 'bisimilar' is actually not a derivative of 'bisimulation', but it is so close and convenient, that nobody objects, so we'll keep using it. However, note that some texts use both terms 'bisimulation' and 'bisimilarity' with somewhat different meanings.

A *bisimulation between rooted transition systems* $(\mathcal{T}_1, r_1)$ and $(\mathcal{T}_2, r_2)$ is a bisimulation β between $\mathcal{T}_1$ and $\mathcal{T}_2$ such that $r_1 \beta r_2$. We denote this by $(\mathcal{T}_1, r_1) \overset{\beta}{\rightleftarrows} (\mathcal{T}_2, r_2)$ and say that $(\mathcal{T}_1, r_1)$ and $(\mathcal{T}_2, r_2)$ are *(locally) bisimulation equivalent*, or *(locally) bisimilar*. ∇

Note that a (local) bisimulation between rooted transition systems $(\mathcal{T}_1, r_1)$ and $(\mathcal{T}_2, r_2)$ is only guaranteed to relate the states of the generated subsystems $\mathcal{T}_1[r_1]$ and $\mathcal{T}_2[r_2]$.

Definition 2.3. [**Bisimulation between interpreted transition systems**]
A *bisimulation between interpreted transition systems* $\mathcal{M}_1 = (\mathcal{T}_1, L_1)$ and $\mathcal{M}_2 = (\mathcal{T}_2, L_2)$ is a bisimulation β between $\mathcal{T}_1$ and $\mathcal{T}_2$ satisfying the following additional condition:

[Atom equivalence]
For every pair $s_1 \beta s_2$ and $p \in \mathrm{PROP}$: $p \in L_1(s_1)$ iff $p \in L_2(s_2)$.
Atom equivalence of s_1 and s_2 will be denoted by $s_1 \simeq s_2$.

That is, β-related states must satisfy the same atomic propositions. ∇

If β is a bisimulation between $\mathcal{M}_1$ and $\mathcal{M}_2$ we denote that by $\mathcal{M}_1 \overset{\beta}{\rightleftarrows} \mathcal{M}_2$. If every state in $\mathcal{M}_1$ is β-related to some state of $\mathcal{M}_2$ and vice versa, we say that $\mathcal{M}_1$ and $\mathcal{M}_2$ are *(globally) bisimulation equivalent*, or *(globally) bisimilar*. If there is a global bisimulation between $\mathcal{M}_1$ and $\mathcal{M}_2$ we denote that by $\mathcal{M}_1 \rightleftarrows \mathcal{M}_2$.

Bisimulation between rooted interpreted transition systems is defined by straightforward combination of the two definitions above.

2.2 Bounded and finite bisimulations

Sometimes, we are only interested in behavioral equivalence of two rooted transition systems up to a given distance from the roots, i.e., up to a given number of transition steps. This leads to the weaker notion of *bounded bisimulation*. We will only give the definition for simple transition systems; the generalization to labeled transition systems is straightforward.

Definition 2.4. [**Bounded bisimulation**] Let $\mathcal{T}_1 = \left\langle S_1, \left(\overset{a}{\longmapsto}_1\right)_{a \in A} \right\rangle$ and $\mathcal{T}_2 = \left\langle S_2, \left(\overset{a}{\longmapsto}_2\right)_{a \in A} \right\rangle$ be two labelled transition systems of the same type.

We define the property of a relation $\beta \subseteq S_1 \times S_2$ to be a *k-bisimulation between the rooted transition systems* $(\mathcal{T}_1, r_1)$ *and* $(\mathcal{T}_2, r_2)$, denoted $(\mathcal{T}_1, r_1) \overset{\beta}{\rightleftarrows}_k (\mathcal{T}_2, r_2)$, inductively on $k \in \mathbb{N}$ as follows:

($\mathbf{B_0}$) $(\mathcal{T}_1, r_1) \overset{\beta}{\rightleftarrows}_0 (\mathcal{T}_2, r_2)$ iff $r_1 \beta r_2$.

($\mathbf{B_{k+1}}$) $(\mathcal{T}_1, r_1) \overset{\beta}{\rightleftarrows}_{k+1} (\mathcal{T}_2, r_2)$ iff for every $a \in A$:

Forth: If $r_1 \overset{a}{\longmapsto}_1 t_1$ for some $t_1 \in S_1$, then there is $t_2 \in S_2$ such that $r_2 \overset{a}{\longmapsto}_1 t_2$ and $(\mathcal{T}_1, t_1) \overset{\beta}{\rightleftarrows}_k (\mathcal{T}_2, t_2)$;

Back: Conversely, if $r_2 \overset{a}{\longmapsto}_1 t_2$ for some $t_2 \in S_2$ then there is $t_1 \in S_1$ such that $r_1 \overset{a}{\longmapsto}_1 t_1$ and $(\mathcal{T}_1, t_1) \overset{\beta}{\rightleftarrows}_k (\mathcal{T}_2, t_2)$.

∇

Clearly, every k-bisimulation between rooted transition systems is an m-bisimulation between them, for every $m \leq k$.

Definition 2.5. [**Finite bisimulation**] A relation β is a *finite bisimulation between the rooted transition systems* $(\mathcal{T}_1, r_1)$ *and* $(\mathcal{T}_2, r_2)$, denoted $(\mathcal{T}_1, r_1) \overset{\beta}{\rightleftarrows}_f (\mathcal{T}_2, r_2)$, if $(\mathcal{T}_1, r_1) \overset{\beta}{\rightleftarrows}_k (\mathcal{T}_2, r_2)$ for every $k \in \mathbb{N}$. ∇

If there is a k-bisimulation between the rooted transition systems $(\mathcal{T}_1, r_1)$ and $(\mathcal{T}_2, r_2)$, we denote that by $(\mathcal{T}_1, r_1) \rightleftarrows_k (\mathcal{T}_2, r_2)$; likewise for finite bisimulations.

Clearly, every bisimulation between the rooted transition systems is a finite bisimulation between them. The converse, however, is not always true.

Bounded and finite bisimulations between rooted interpreted transition systems are defined likewise, by adding the atom equivalence condition to the definition of 0-bisimulation.

2.3 Paths and computations in bisimilar transition systems

As discussed, bisimulation equivalence implies behavioral equivalence in a strong sense, which we will make more precise here.

The following is an immediate consequence from the definition:

Proposition 2.1. Let $(\mathcal{T}_1, r_1) \overset{\beta}{\rightleftarrows} (\mathcal{T}_2, r_2)$. Then for every path $r_1 = r_1^0 \overset{a_0}{\longmapsto} r_1^1 \overset{a_1}{\longmapsto} r_1^2 \dots$ in $\mathcal{T}_1$ there is a 'β-similar' path $r_2 = r_2^0 \overset{a_0}{\longmapsto} r_2^1 \overset{a_1}{\longmapsto} r_2^2 \dots$ in $\mathcal{T}_2$ such that $r_1^k \beta r_2^k$ for every $k \in \mathbb{N}$, and vice versa, for every path $r_2 = r_2^0 \overset{a_0}{\longmapsto} r_2^1 \overset{a_1}{\longmapsto} r_2^2 \dots$ in $\mathcal{T}_2$ there is a 'β-similar' path $r_1 = r_1^0 \overset{a_0}{\longmapsto} r_1^1 \overset{a_1}{\longmapsto} r_1^2 \dots$ in $\mathcal{T}_1$ such that $r_1^k \beta r_2^k$ for every $k \in \mathbb{N}$.

An analogous result holds for computations in bisimilar rooted interpreted transition systems.

Proposition 2.2. Let $(\mathcal{M}_1, r_1) \overset{\beta}{\rightleftarrows} (\mathcal{M}_2, r_2)$. Then every computation $l_0 \overset{a_0}{\longmapsto} l_1 \overset{a_1}{\longmapsto} l_2 \dots$ generated by a path in $\mathcal{M}_1$ starting at r_1 is also generated by a β-bisimilar path in $\mathcal{M}_2$ starting at r_2 and vice versa.

Thus, the roots of locally bisimilar rooted interpreted transition systems initiate the same sets of computations. Respectively, globally bisimilar interpreted transition systems generate the same sets of computations.

2.4 Bounded morphisms

Bounded morphisms are important particular case of bisimulations, where the bisimulation relation is a function from one ITS to the other.

Definition 2.6. Let $\mathcal{M}_1$ and $\mathcal{M}_2$ be two ITS of the same type with domains S_1 and S_2 respectively. A mapping $\rho\colon S_1 \to S_2$ is a *bounded morphism* from $\mathcal{M}_1$ to $\mathcal{M}_2$, denoted $\rho\colon \mathcal{M}_1 \overset{\rightleftarrows}{\longrightarrow} \mathcal{M}_2$, if its graph is a bisimulation between $\mathcal{M}_1$ and $\mathcal{M}_2$.

If ρ is onto, then $\mathcal{M}_2$ is called a *bounded morphic image of* $\mathcal{M}_1$. ∇

Thus, a bounded morphism ρ associates with each $s \in S_1$ a unique bisimilar state $\rho(s) \in S_2$. The bisimulation conditions for a bounded morphism between two ITS respectively become:

Atomic correspondence: $L_1(s) = L_2(\rho(s))$ for every $s \in S_1$.

Forth: For every $s, t \in S_1$ and $a \in A$, if $s \stackrel{a}{\longmapsto}_1 t$ then $\rho(s) \stackrel{a}{\longmapsto}_2 \rho(t)$.

Back: For every $s \in S_1$ and $a \in A$, if $\rho(s) \stackrel{a}{\longmapsto}_2 u$ for some $u \in S_2$, then $u = \rho(t)$ for some $t \in S_1$ such that $s \stackrel{a}{\longmapsto}_1 t$.

An important case of bounded morphism is given by the unfolding construction.

Proposition 2.3. [**Unfoldings are bounded morphisms**]
Let $\mathcal{M} = \left\langle S, \left\{\stackrel{a}{\longmapsto}\right\}_{a\in A}, L\right\rangle$ be an interpreted transition system.

Then the mapping $l : \widehat{S} \to S$ associating to every path π its last state $l(\pi)$ is a bounded morphism from $\widehat{\mathcal{M}}$ onto $\mathcal{M}$.

Proof: Exercise. QED

2.5 Generated and rooted substructures

Generated and rooted subsystems are another important case of bounded morphisms

If $R \subseteq W^2$ is any binary relation over W, and $W' \subseteq W$, we write $R \restriction W'$ for the restriction of R to W', $R \restriction W' = R \cap (W' \times W')$. Similarly for a mapping f with domain W, $f \restriction W'$ stands for its restriction to W'.

Definition 2.7. [**Generated subsystems**] Let $\mathcal{T} = \left\langle S, \left\{\stackrel{a}{\longmapsto}\right\}_{a\in A}\right\rangle$ be a TS, $\mathcal{M} = \langle \mathcal{T}, L\rangle$ be an ITS, and let $S' \subseteq S$.

- ⋆ The *induced subsystem* of $\mathcal{T}$ over S' is the transition system
  $\mathcal{T}' := \mathcal{T} \restriction S' = \left\langle S', \{\stackrel{a}{\longmapsto} \restriction S'\}_{a\in A}\right\rangle$.
  The subsystem relationship is denoted $\mathcal{T}' \leq \mathcal{T}$.
- ⋆ $\mathcal{T}' = \mathcal{T} \restriction S'$ is a *generated subsystem of* $\mathcal{T}$, denoted $\mathcal{T}' \trianglelefteq \mathcal{T}$, if S' is closed under all transition relations in the sense that if $s \stackrel{a}{\longmapsto}_1 t$ for $s \in S'$ then $t \in S'$.
- ⋆ The *induced subsystem of* $\mathcal{M}$ *over* S' is the ITS $\mathcal{M}' = \mathcal{M} \restriction S' = \langle \mathcal{T} \restriction S', L \restriction S'\rangle$, denoted $\mathcal{M}' \leq \mathcal{M}$. If $\mathcal{T} \restriction S' \trianglelefteq \mathcal{T}$, then $\mathcal{M}'$ is a *generated interpreted subsystem of* $\mathcal{M}$, denoted $\mathcal{M}' \trianglelefteq \mathcal{M}$.

∇

Note that if $\mathcal{M}' \trianglelefteq \mathcal{M}$ then the inclusion mapping $\rho\colon S' \to S$ is a bounded morphism.

A particularly important case of generated subsystems deals with the set of all states reachable from a given state.

Recall that $post^*_{\mathcal{T}}(s)$ denotes the set of all states in $\mathcal{T}$ reachable from the state s.

Definition 2.8. [**Rooted subsystems**] Let $\mathcal{T} = \left\langle S, \left\{ \stackrel{a}{\longmapsto} \right\}_{a \in A} \right\rangle$ be a TS, $\mathcal{M} = \langle \mathcal{T}, L \rangle$ be an ITS, and let $s \in S$.

- ⋆ The *subsystem of* $\mathcal{T}$ *rooted at* s is the TS $\mathcal{T}[s] = \mathcal{T} \restriction post^*_{\mathcal{T}}(s)$.
- ⋆ The *interpreted subsystem of* $\mathcal{M}$ *rooted at* s is the ITS $\mathcal{M}[s] = \mathcal{M} \restriction post^*_{\mathcal{T}}(s)$.
- ⋆ $\mathcal{T}$ (respectively $\mathcal{M}$) is *rooted at* s if $post^*_{\mathcal{T}}(s) = S$.

∇

Clearly, for any $s \in S$: $\mathcal{T}[s] \trianglelefteq \mathcal{T}$ and $\mathcal{M}[s] \trianglelefteq \mathcal{M}$, respectively.

2.6 Bisimulation games

Bisimulations between transition systems can be characterized in a more animated way as existence of winning strategies for one player in corresponding *bisimulation games* [Sti99], or more generally *model comparison games* (see [GO07] which exposition we follow here). We illustrate the concept in the case of bisimulations for interpreted mono-transition systems; the generalization to labelled transition systems is straightforward.

Let $\mathcal{M}_1 = (S_1, R_1, L_1)$ and $\mathcal{M}_2 = (S_2, R_2, L_2)$ be interpreted transition systems of the same type. The *bisimulation game* over $\mathcal{M}_1$ and $\mathcal{M}_2$ is played by two players **I** and **II** with one pebble in $\mathcal{M}_1$ and one in $\mathcal{M}_2$ to mark the 'current' state in each structure. A *configuration* in the game is a pair of rooted interpreted transition systems $(\mathcal{M}_1, s_1; \mathcal{M}_2, s_2)$ where the roots are the current positions of the two pebbles.

The game starts from an *initial configuration* and is played in *rounds*, each round played as follows. The first player, or *challenger*, denoted **I**, selects one of the two pebbles and moves it forward along a transition in the respective structure, to a successor state. The second player, or *defender*, denoted **II**, has to respond by similarly moving forward the pebble along a transition (with the same label) in the other structure.

Intuitively, the objective of player **I**, also known as the *challenger*, or *spoiler*, in the game is to detect and exhibit a behavioral difference between the two rooted ITS in the initial configuration by choosing a transition in one of them that cannot be simulated in the other, while the objective of player **II**, also known as the *defender*, or *duplicator*, is to defend the claim that the two rooted ITS in the initial configuration are behaviorally equivalent, so she tries to reply with a transition maintaining that equivalence for the duration of the game.

During the game, **II** loses if she cannot respond correctly to the move of **I**, or if the two pebble positions in the resulting new configuration are *not in atom-correspondent states*, i.e., these states are distinguished by at an atomic proposition. On the other hand, **I** loses during the game if he cannot make a move in the current round because both pebbles are in states without successors.

The bisimulation game can be played for a pre-determined number of rounds, or indefinitely. The *n-round bisimulation game* terminates after n rounds, or earlier if either player loses during one of these rounds. If the n-th round is completed without violating the atom equivalence in any configuration, player **II** wins the game.

We say that player **II** has a *winning strategy* in the n-round bisimulation game with a given initial configuration, if she has responses to any challenges from the first player that guarantee her to win the game, either because **I** gets stuck, or because she can respond with good moves for the duration of the game. Likewise, winning strategy of player **I** is defined.

Respectively, the *(unbounded) bisimulation game* is played until some of the players loses, otherwise forever. An infinite path of the game (which continues through an infinite sequence of rounds), played correctly according to the above rules, is won by **II**.

The notion of a winning strategy (for player **I** or **II**) is accordingly adapted for the unbounded bisimulation game.

Proposition 2.4. Every bisimulation game is determined, i.e., one of the players has a winning strategy.

Proof: See [Sti99]. QED

The intuition of player **I** challenging the claim of bisimilarity in the current configuration, while player **II** defending that claim is formalized by the following proposition.

Theorem 2.5.

1. Player **II** has a winning strategy in the n-round bisimulation game with initial configuration $(\mathcal{M}_1, s_1; \mathcal{M}_2, s_2)$ if and only if $(\mathcal{M}_1, s_1) \rightleftarrows_n (\mathcal{M}_2, s_2)$.
2. Player **II** has a winning strategy in the unbounded bisimulation game with initial configuration $(\mathcal{M}_1, s_1; \mathcal{M}_2, s_2)$ if and only if $(\mathcal{M}_1, s_1) \rightleftarrows (\mathcal{M}_2, s_2)$.

Proof: We will sketch the unbounded case, as it subsumes the bounded one. Indeed, any bisimulation $(\mathcal{M}_1, s_1) \overset{\beta}{\rightleftarrows} (\mathcal{M}_2, s_2)$ provides a non-deterministic winning strategy for **II**: she merely needs to select her responses so that the currently pebbled states remain linked by β. The atom equivalence condition on β guarantees that atom equivalence between pebbled states is maintained; the *forth* condition guarantees a matching response to challenges played by **I** in $\mathcal{M}_1$; the *back* condition similarly guarantees a matching response to challenges played in $\mathcal{M}_2$.

Conversely, the set of pairs (u, t_2) in all configurations $(\mathcal{M}_1, t_1; \mathcal{M}_2, t_2)$ from which **II** has a winning strategy, if non-empty, is a bisimulation.QED

2.7 Bisimulation invariance of TL

All results about bisimulation invariance of TL-formulae in interpreted transition systems presented here apply likewise to bisimulation invariance of formulae of Hennessy-Milner logic HML_τ in plain (non-interpreted) transition systems.

2.7.1 Invariance of TL-formulae under bisimulations

Definition 2.9. [**TL-equivalence**] Two rooted ITS $(\mathcal{M}_1, r_1)$ and $(\mathcal{M}_2, r_2)$ are *TL-equivalent*, denoted $(\mathcal{M}_1, r_1) \equiv_{\mathrm{TL}} (\mathcal{M}_2, r_2)$ if they satisfy the same TL-formulae. $(\mathcal{M}_1, r_1)$ and $(\mathcal{M}_2, r_2)$ are *TL^n-equivalent*, denoted $(\mathcal{M}_1, r_1) \equiv^n_{\mathrm{TL}} (\mathcal{M}_2, r_2)$ if they satisfy the same TL-formulae of modal depth up to n. ∇

Theorem 2.6. [**Bounded bisimulation invariance**] If $(\mathcal{M}_1, r_1)$ and $(\mathcal{M}_2, r_2)$ are rooted interpreted transition systems and $n \in \mathbb{N}$ is such that $(\mathcal{M}_1, r_1) \rightleftarrows_n (\mathcal{M}_2, r_2)$, then $(\mathcal{M}_1, r_1) \equiv^n_{\mathrm{TL}} (\mathcal{M}_2, r_2)$.

Proof: We will use the characterization (Proposition 2.5) of n-bisimulations by means of n-round bisimulation games to prove the contraposition of the claim: if $\mathcal{M}_1, r_1 \models \varphi$ and $\mathcal{M}_2, r_2 \models \neg\varphi$ for some $\varphi \in \mathrm{TL}^n$, then player **I** has a winning strategy in the n-round game on $(\mathcal{M}_1, r_1; \mathcal{M}_2, r_2)$, and therefore $(\mathcal{M}_1, r_1) \not\rightleftharpoons_n (\mathcal{M}_2, r_2)$.

This is shown by induction on the modal depth of the formula φ. If $\mathrm{md}(\varphi) = 0$, a distinction in TL^0 means violated atomic correspondence – a configuration in which player **II** has lost.

For the induction step, assume that $(\mathcal{M}_1, r_1)$ is distinguished from $(\mathcal{M}_2, r_2)$ by a formula $\varphi \in \mathrm{TL}^{n+1}$. Propositional connectives in φ can be pre-processed, so that without loss of generality φ can be assumed of the form $\mathsf{EX}\psi$ for some $\psi \in \mathrm{TL}^n$. Suppose then that for instance $\mathcal{M}_2, r_2 \models \neg\varphi$, while $\mathcal{M}_1, r_1 \models \varphi$. Suppose in that case player **I** moves the pebble in $\mathcal{M}_1$ from r_1 to some t_1, where $\mathcal{M}_1, r_1 \models \psi$. As $\mathcal{M}_2, r_2 \models \neg\,\mathsf{EX}\psi$, any available response for player **II** can only lead to a configuration $(\mathcal{M}_1, t_1; \mathcal{M}_2, t_2)$ in which $(\mathcal{M}_1, t_1)$ and $(\mathcal{M}_2, t_2)$ are distinguished by $\psi \in \mathrm{TL}^n$. Therefore, by the inductive hypothesis, player **I** has a winning strategy for the remaining n rounds of the game. QED

Corollary 2.7. [**Bisimulation invariance**]
The formulae of TL are invariant under bisimulations: if $(\mathcal{M}_1, r_1)$ and $(\mathcal{M}_2, r_2)$ are rooted interpreted transition systems, such that $(\mathcal{M}_1, r_1) \rightleftarrows (\mathcal{M}_2, r_2)$, then $(\mathcal{M}_1, r_1) \equiv_{\mathrm{TL}} (\mathcal{M}_2, r_2)$.

In particular, if $(\mathcal{M}_2, r_2)$ is a bounded morphic image of $(\mathcal{M}_1, r_1)$ then $(\mathcal{M}_1, r_1) \equiv_{\mathrm{TL}} (\mathcal{M}_2, r_2)$.

Corollary 2.8. For every constant formula $\theta \in \mathrm{TL}$ and rooted transition systems $(\mathcal{T}_1, r_1)$ and $(\mathcal{T}_2, r_2)$, if $(\mathcal{T}_1, r_1) \rightleftarrows (\mathcal{T}_2, r_2)$, then $(\mathcal{T}_1, r_1) \equiv_{\mathrm{TL}} (\mathcal{T}_2, r_2)$.

Corollary 2.9. For all interpreted transition systems $\mathcal{M}, \mathcal{M}'$ such that $\mathcal{M}' \trianglelefteq \mathcal{M}$ and for every formula φ of TL:

I. for every $s \in \mathsf{dom}(\mathcal{M}') : \mathcal{M}, r \models \varphi$ iff $\mathcal{M}', s \models \varphi$.

II. $\mathcal{M} \models \varphi$ implies $\mathcal{M}' \models \varphi$.

2.7.2 Characterization of TL by bisimulation invariance

When restricted to state properties definable in FO_2, it turns out that bisimulation invariance is not only necessary, but also a sufficient condition for definability in TL, as the following classical result shows.

Definition 2.10. [**Bisimulation invariance of a FO-formula**] A formula $\gamma(x) \in \mathrm{FO}_\tau$ of one free variable x is *bisimulation invariant* if for every ITS $(\mathcal{M}, r)$ and $\mathcal{M}', r'$, such that $(\mathcal{M}, r) \rightleftarrows (\mathcal{M}', r')$, we have that $\mathcal{M}, r \models \varphi$ iff $\mathcal{M}', r' \models \varphi$. ∇

Theorem 2.10. [**van Benthem's characterization theorem** [Ben84]]
Let $\gamma(x) \in \mathrm{FO}_\tau$. Then, the following are equivalent:

1. γ is bisimulation invariant.
2. $\gamma(x)$ is logically equivalent to a formula $\tilde{\gamma} \in \mathrm{TL}$.

2.8 Bisimulations and logical equivalence

Much of what follows here comes from [GO07]. In this section we assume that the type PROP is finite and, for technical simplicity, we consider that case of one transition relation. The generalization to arbitrary signatures is an easy exercise. In the particular case of PROP $= \emptyset$ we obtain respective results about Hennessy-Milner logic and plain (non-interpreted) transition systems.

2.8.1 Characteristic formulae

Definition 2.11. [**Characteristic formulae**] With every rooted ITS $(\mathcal{M}, r)$, where $\mathcal{M} = (S, R, L)$ and $n \in \mathbb{N}$ we associate a *characteristic formula of depth* n, $\chi^n_{[\mathcal{M},r]}$, defined inductively as follows:

- $\chi^0_{[\mathcal{M},r]} := \bigwedge\{p \mid p \in L(r)\} \wedge \bigwedge\{\neg p \mid p \notin L(r)\}$,
  where p ranges over PROP.
- $\chi^{n+1}_{[\mathcal{M},r]} := \chi^0_{[\mathcal{M},r]} \wedge \bigwedge_{rRt} \mathsf{EX}\chi^n_{[\mathcal{M},t]} \wedge \mathsf{AX} \bigvee_{rRt} \chi^n_{[\mathcal{M},t]}$.
  To be more precise, the possible repetitions of conjuncts (resp., disjuncts) in the multiple conjunction (resp., disjunction) in the formula on the right hand side above are omitted.

∇

Note that $\chi^n_{[\mathcal{M},r]} \in \mathrm{TL}^n$. Intuitively, $\chi^n_{[\mathcal{M},r]}$ combines the atomic description of r and the characteristic formulae of depth $n-1$ for all successors of r and only of them. In the long run, it describes the part of $\mathcal{M}$ seen from r within n steps – but, as we will show, only up to n-bisimulation equivalence.

It can be proved, by induction on n, simultaneous with the inductive definition above, that there are only finitely many different (up to logical equivalence) characteristic formulae of depth n, so even though a state r may have infinitely many successors, every formula $\chi^n_{[\mathcal{M},r]}$ is well-defined, i.e., finite.

Exercise 2.1. Show that $\mathcal{M}, r \models \chi^n_{[\mathcal{M},r]}$ for every $n \in \mathbb{N}$. (Hint: induction on n.)

Theorem 2.11. For every rooted ITS $(\mathcal{M}, r)$ and $(\mathcal{M}', r')$ the following are equivalent:

1. $\mathcal{M}', r' \models \chi^n_{[\mathcal{M},r]}$.
2. **II** has a winning strategy in the n-round game from $(\mathcal{M}, r; \mathcal{M}', r')$.

Proof: First we show that if $\mathcal{M}', r' \models \chi^n_{[\mathcal{M},r]}$ then player **II** has a wining strategy in the n-round game from $(\mathcal{M}, r; \mathcal{M}', r')$ by induction on n.

For $n = 0$ the claim follows by definition. Assuming it holds for n, let us look again at $\chi^{n+1}_{[\mathcal{M},r]}$ from the perspective of the game:

$$\chi^{n+1}_{[\mathcal{M},r]} = \chi^0_{[\mathcal{M},r]} \wedge \underbrace{\bigwedge_{(r,s)\in R} \mathsf{EX}\chi^n_{[\mathcal{M},s]}}_{\textit{forth}} \quad \wedge \quad \underbrace{\mathsf{AX} \bigvee_{(r,s)\in R} \chi^n_{[\mathcal{M},s]}}_{\textit{back}}.$$

The conjunct $\chi^0_{[\mathcal{M},r]}$ guarantees that the game is not lost already.

The *back*-and-*forth* conjuncts tell player **II** how to provide suitable responses in the first round to challenges from player **I** played respectively in $\mathcal{M}$ (*forth*) or in $\mathcal{M}'$ (*back*).

The *forth* part says that for all moves from r to some s in $\mathcal{M}$, $\mathcal{M}', r' \models \mathsf{EX}\chi^n_{[\mathcal{M},s]}$, and any R'-successor s' of r' such that $\mathcal{M}', s' \models \chi^n_{[\mathcal{M},s]}$ provides a response for player **II** that will allow her to succeed through another n rounds.

Similarly the *back* part says that for all moves from r' to some s' in $\mathcal{M}$ there is a R-successor s of r in $\mathcal{M}$ such that $\mathcal{M}', s' \models \chi^n_{[\mathcal{M},s]}$. That s is a response for player **II** that is good for another n rounds.

Conversely, a failure of $\mathcal{M}', r'$ to satisfy $\chi^n_{[\mathcal{M},r]}$ gives player **I** a winning strategy within n rounds, follows from Theorem 2.6 and Proposition 2.5. QED

2.8.2 Bisimulations, bisimulation games, characteristic formulae, and TL-equivalence: linking them all together

Now, we can put together bisimulations, bisimulation games, logical equivalence and characteristic formulae.

Theorem 2.12. For every rooted ITS $(\mathcal{M}, r)$ and $(\mathcal{M}', r')$ the following are equivalent:

1. $\mathcal{M}', r' \models \chi^n_{[\mathcal{M},r]}$.
2. $(\mathcal{M}, r) \equiv^n_{\mathrm{TL}} (\mathcal{M}', r')$.
3. $(\mathcal{M}, r) \overset{n}{\rightleftarrows} (\mathcal{M}', r')$.
4. Player **II** has a winning strategy in the n-round bisimulation game on $(\mathcal{M}, r; \mathcal{M}', r')$.

Proof:

By theorems 2.6, 2.11, and Proposition 2.5. QED

As corollaries we obtain a corresponding characterization of full modal equivalence, and a normal form for TL formulae.

Corollary 2.13. For every rooted ITS $(\mathcal{M}, r)$ and $(\mathcal{M}', r')$ of finite type the following are equivalent:

1. For every $n \in \mathbb{N}$, $\mathcal{M}', r' \models \chi^n_{[\mathcal{M},r]}$.
2. $(\mathcal{M}, r) \equiv_{\mathrm{TL}} (\mathcal{M}', r')$.
3. $(\mathcal{M}, r) \rightleftarrows_f (\mathcal{M}', r')$.
4. For every $n \in \mathbb{N}$ player **II** has winning strategies in the n-round bisimulation games on $(\mathcal{M}, r; \mathcal{M}', r')$.

Corollary 2.14. Over interpreted transition systems of any finite type, finite bisimulation equivalence coincides with modal equivalence.

Corollary 2.15. Any formula $\varphi \in \mathrm{TL}_n$ is logically equivalent to the disjunction $\bigvee_{\mathcal{M},r \models \varphi} \chi^n_{[\mathcal{M},r]}$.

Note that the disjunction above is finite as there are only finitely many, up to logical equivalence, such χ^n in the type of φ.

Exercise 2.2. Prove corollary 2.15.

Definition 2.12. [**Classes closed under bisimulation**] A class $\mathcal{C}$ of rooted interpreted transition systems is closed under bisimulation if, whenever $(\mathcal{M}, r) \in \mathcal{C}$ and $(\mathcal{M}, r) \leftrightarrows (\mathcal{M}', r')$ then $(\mathcal{M}', r') \in \mathcal{C}$.

Classes closed under n-bisimulations and finite bisimulations are defined likewise. ∇

Corollary 2.16. Any class $\mathcal{C}$ of rooted interpreted transition systems of a finite type that is closed under n-bisimulation is definable in TL^n by the disjunction $\bigvee_{(\mathcal{M},r)\in\mathcal{C}} \chi^n_{[\mathcal{M},r]}$.

Exercise 2.3. Prove corollary 2.16.

Proposition 2.17. Let $(\mathcal{T}_1, r_1)$ and $(\mathcal{T}_2, r_2)$ be finite rooted ITS with respectively n_1 and n_2 states, such that $(\mathcal{T}_1, r_1) \leftrightarrows_{n_1 n_2} (\mathcal{T}_2, r_2)$. Then $(\mathcal{T}_1, r_1) \leftrightarrows (\mathcal{T}_2, r_2)$.

Proof: Note that any pair of states (s_1, s_2) reachable by playing a bisimulation game between $(\mathcal{T}_1, r_1)$ and $(\mathcal{T}_2, r_2)$, is reachable within $n_1 n_2$ rounds of the game, and therefore player **II** has a suitable response to any move of player **I** from the configuration $(\mathcal{T}_1, s_1; \mathcal{T}_2, s_2)$ – that is the prescribed response by her winning strategy for the $n_1 n_2$-round bisimulation game between $(\mathcal{T}_1, r_1)$ and $(\mathcal{T}_2, r_2)$ on the first appearance of the configuration $(\mathcal{T}_1, s_1; \mathcal{T}_2, s_2)$. QED

Corollary 2.18. Let $(\mathcal{M}, r)$ and $(\mathcal{M}', r')$ be finite rooted ITS with respectively n_1 and n_2 states, for which any of the equivalent conditions in Theorem 2.12 holds for some $n \geq n_1 n_2$. Then $(\mathcal{M}, r) \leftrightarrows (\mathcal{M}', r')$.

Thus, in a finite ITS $\mathcal{M}$ every state r can be characterized up to bisimulation equivalence by the characteristic formula $\chi^n_{[\mathcal{M},r]}$ for any large enough n. We will denote by $\chi_{[\mathcal{M},r]}$ the formula for the least suitable n. Thus, we have the following.

Corollary 2.19. For every finite rooted ITS $(\mathcal{M}, r)$ and a state $s \in \mathcal{M}$ the following are equivalent:

1. $\mathcal{M}, s \models \chi_{[\mathcal{M},r]}$.
2. $(\mathcal{M}, r) \leftrightarrows (\mathcal{M}, s)$.

2.8.3 Finite versus full bisimulation

Definition 2.13. [**Finite branching TS**] A (I)TS is *finitely branching* if every state in that transition system has only finitely many immediate successors. ∇

Theorem 2.20. [Hennessy–Milner theorem] Let $\mathcal{M}$ and $\mathcal{M}'$ both be finitely branching ITS. Then $(\mathcal{M}, r) \leftrightarrows_f (\mathcal{M}', r')$ implies $(\mathcal{M}, r) \leftrightarrows (\mathcal{M}', r')$.

Proof: The argument is best given via the games. We claim that player **II** can maintain $(\mathcal{M}, r) \rightleftarrows_\omega (\mathcal{M}', r')$ indefinitely – which gives her a winning strategy for the infinite game. For instance, let player **I** play in $\mathcal{M}$ and move the pebble from r to s. Suppose that for all responses s' available to player **II** in $\mathcal{M}'$, $(\mathcal{M}, s) \not\rightleftarrows_\omega (\mathcal{M}', s')$. As there are only finitely many choices for s' due to finite branching, we can find a sufficiently large $n \in \mathbb{N}$ such that $(\mathcal{M}, s) \not\rightleftarrows_n (\mathcal{M}', s')$ for all s' with $(w', s') \in R'$. But this would imply $(\mathcal{M}, r) \not\rightleftarrows_{n+1} (\mathcal{M}', r')$, contradicting the assumption $(\mathcal{M}, r) \rightleftarrows_\omega (\mathcal{M}', r')$. QED

Corollary 2.21. Over finitely branching ITS, modal equivalence coincides with bisimulation equivalence.

We note that the finite branching assumption is essential for the result above, as the following example shows.

Example 2.1. Let $(\mathcal{T}, r)$ and $(\mathcal{T}', r')$ be tree-like TS, rooted at r and r', respectively. Let the roots have countably many distinct successors s_i, $i \geq 1$ in $\mathcal{T}$ and s'_i, $i \geq 0$ in $\mathcal{T}'$. For $i \geq 1$, we let each of s_i and s'_i be the starting point of a simple finite path of length i. We let the extra node s'_0 in $\mathcal{T}'$ be the root of a simple infinite path. Then $(\mathcal{T}, r) \not\rightleftarrows (\mathcal{T}', r')$: let player **I** move in $\mathcal{T}'$ from r' to s'_0; the second player must move to one of the s_i for $i \geq 1$ in $\mathcal{T}$; let then player **I** lead the play in $\mathcal{T}'$ along the infinite path: player **II** gets stuck and loses in round $i + 2$ when the end of the length i path from s_i has been reached. On the other hand, $(\mathcal{T}, r) \rightleftarrows_n (\mathcal{T}', r')$ for every $n \in \mathbb{N}$, since any two paths of lengths greater than or equal to n look exactly the same in an n-round game. ∇

2.9 Simulations

By removing the 'Back' condition in the definitions of bisimulation we obtain respective notions of *simulation* of one transition system by another. In particular:

Definition 2.14. [**Simulation between transition systems**]
Given two labelled transition systems $\mathcal{T}_1 = \left\langle S_1, \left(\overset{a}{\longmapsto}_1\right)_{a \in A}\right\rangle$ and $\mathcal{T}_2 = \left\langle S_2, \left(\overset{a}{\longmapsto}_2\right)_{a \in A}\right\rangle$, we say that a non-empty relation $\beta \subseteq S_1 \times S_2$ is a *simulation of* $\mathcal{T}_1$ *by* $\mathcal{T}_2$, denoted $\mathcal{T}_1 \overset{\beta}{\rightrightarrows} \mathcal{T}_2$, if it satisfies the 'Forth' condition of the definition of bisimulation for every pair of states (s_1, s_2) such that $s_1 \beta s_2$ and every transition label $a \in A$. ∇

The other notions of simulation, bounded simulation, and finite simulation, are defined likewise.

Note that if $(\mathcal{T}_1, r_1) \overset{\beta}{\rightrightarrows} (\mathcal{T}_2, r_2)$ then every path in $\mathcal{T}_1$ starting at r_1 is 'β-simulated' by a path in $\mathcal{T}_2$ starting at r_2, but not necessarily the other way around.

Consequently, if $(\mathcal{M}_1, r_1) \overset{\beta}{\rightrightarrows} (\mathcal{M}_2, r_2)$ for some interpreted transition systems $\mathcal{M}_1$ and $\mathcal{M}_2$, then every computation in $\mathcal{M}_1$ starting at r_1 is also generated in $\mathcal{M}_2$ starting at r_2.

2.10 Trace and computational inclusions and equivalences

Sometimes we are not interested in step-by-step simulation or bisimulation, but in simulations or equivalences only involving the entire paths or computations in the respective systems. Thus, we arrive at a variety of notions of trace and computational inclusion and equivalence, of which we will only present here the more relevant ones.

Definition 2.15. [**Trace inclusion and equivalence between rooted transition systems**] Let $(\mathcal{T}_1, r_1)$ and $(\mathcal{T}_2, r_2)$ be two rooted transition systems of the same type. A non-empty relation $\beta \subseteq S_1 \times S_2$ is a *trace inclusion of* $(\mathcal{T}_1, r_1)$ *into* $(\mathcal{T}_2, r_2)$, denoted $(\mathcal{T}_1, r_1) \overset{\beta}{\rightsquigarrow} (\mathcal{T}_2, r_2)$, if for every path $r_1 = r_1^0 \overset{a_0}{\longmapsto} r_1^1 \overset{a_1}{\longmapsto} r_1^2 \ldots$ starting at r_1 in $\mathcal{T}_1$ there is a 'β-similar' path $r_2 = r_2^0 \overset{a_0}{\longmapsto} r_2^1 \overset{a_1}{\longmapsto} r_2^2 \ldots$ in $\mathcal{T}_2$, i.e., such that $r_1^k \beta r_2^k$ for every $k \in \mathbb{N}$.

If, moreover, $(\mathcal{T}_2, r_2) \overset{\beta}{\rightsquigarrow} (\mathcal{T}_1, r_1)$, then we say that β is a *trace equivalence between* $(\mathcal{T}_1, r_1)$ *and* $(\mathcal{T}_2, r_2)$, denoted $(\mathcal{T}_1, r_1) \overset{\beta}{\leftrightsquigarrow} (\mathcal{T}_2, r_2)$. ∇

Definition 2.16. [**Computational inclusion and equivalence between rooted interpreted transition systems**] Let $(\mathcal{M}_1, r_1)$ and $(\mathcal{M}_2, r_2)$ be two rooted interpreted transition systems of the same type. We say that $(\mathcal{M}_1, r_1)$ *is computationally included into* $(\mathcal{M}_2, r_2)$, denoted $(\mathcal{M}_1, r_1) \subseteq_{comp} (\mathcal{M}_2, r_2)$, if every computation in $\mathcal{M}_1$ starting at r_1 is also a computation in $\mathcal{M}_2$ starting at r_2.

If, moreover, $(\mathcal{M}_2, r_2) \subseteq_{comp} (\mathcal{M}_1, r_1)$, too, then we say that $(\mathcal{M}_1, r_1)$ *and* $(\mathcal{M}_2, r_2)$ *are computationally equivalent*, denoted $(\mathcal{M}_1, r_1) \equiv_{comp} (\mathcal{M}_2, r_2)$. ∇

Clearly, if $(\mathcal{T}_1, L_1, r_1)$ and $(\mathcal{T}_2, L_2, r_2)$ are such that $(\mathcal{T}_1, r_1) \overset{\beta}{\rightsquigarrow} (\mathcal{T}_2, r_2)$ and β satisfies the atom equivalence condition with respect to L_1 and L_2, then $(\mathcal{T}_1, L_1, r_1) \subseteq_{comp} (\mathcal{T}_2, L_2, r_2)$.

3 Lecture 3: The linear-time temporal logic LTL

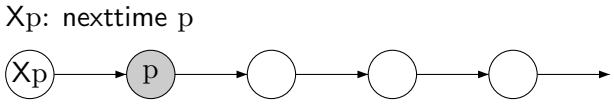
The linear-time temporal logics are intended to reason about linear models; in our case these are single computations. The most popular linear-time temporal logic LTL was first considered in the form presented here in [GPSS80], based on the early works [Kam68, Pnu77b]. Indeed, the strict until operator, 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].

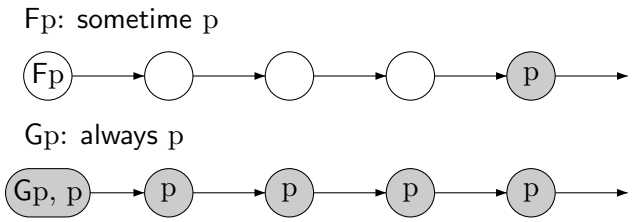
3.1 LTL intuitively

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.



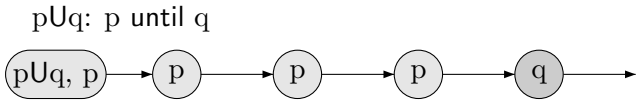
Sometime and **Always**. $\mathsf{F}p$ claims that some future (or possibly, the current) state satisfies φ without specifying which, while $\mathsf{G}\varphi$ claims that all the future states (including the current one) satisfy φ, i.e., that "φ will always be true".



By way of example, the expression `alert` $\Rightarrow \mathsf{F}$ `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.

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

3.2 Formal syntax and semantics of LTL in linear models

LTL formulae are built from the following abstract grammar:

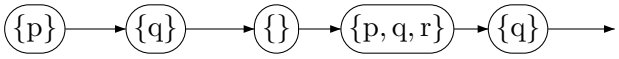
$$\varphi ::= \overbrace{\bot \mid \top \mid \mathrm{p} \mid \neg\varphi \mid \varphi \wedge \psi \mid \varphi \vee \psi}^{\text{propositional calculus}} \mid \overbrace{\mathsf{X}\varphi \mid \mathsf{F}\varphi \mid \mathsf{G}\varphi \mid \varphi\mathsf{U}\psi}^{\text{temporal extension}}$$

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 $\texttt{O}$, an $\texttt{O}$-formula is an LTL formula whose outermost connective is $\texttt{O}$, 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.

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:



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

- $\star$ always $\sigma, i \models \top$ and never $\sigma, i \models \bot$,
- $\star$ $\sigma, i \models \mathrm{p}$ iff $\mathrm{p} \in \sigma(i)$, for every $\mathrm{p} \in \mathrm{PROP}$,
- $\star$ $\sigma, i \models \neg\varphi$ iff $\sigma, i \not\models \varphi$,
- $\star$ $\sigma, i \models \varphi_1 \wedge \varphi_2$ iff $\sigma, i \models \varphi_1$ and $\sigma, i \models \varphi_2$,
- $\star$ $\sigma, i \models \varphi_1 \vee \varphi_2$ iff $\sigma, i \models \varphi_1$ or $\sigma, i \models \varphi_2$,
- $\star$ $\sigma, i \models \mathsf{X}\varphi$ iff $\sigma, i+1 \models \varphi$,
- $\star$ $\sigma, i \models \mathsf{F}\varphi$ iff there is $j \geq i$ such that $\sigma, j \models \varphi$,
- $\star$ $\sigma, i \models \mathsf{G}\varphi$ iff for all $j \geq i$, we have $\sigma, j \models \varphi$,
- $\star$ $\sigma, i \models \varphi_1 \mathsf{U} \varphi_2$ 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 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 $\mathrm{LTL}_n^k(\mathsf{O}_1, \mathsf{O}_2, \ldots)$ to denote the fragment of LTL restricted to formulae such that

- $\star$ the temporal operators are among $\mathsf{O}_1, \mathsf{O}_2, \ldots$,
- $\star$ the temporal depth is bounded by k,
- $\star$ 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. The *temporal depth* of a formula, noted $\mathrm{td}(\varphi)$ is defined as the maximal nesting of temporal operators. For instance, $\mathrm{td}((\mathsf{XX}\mathrm{p}) \vee (\mathrm{p}\mathsf{U}\neg\mathrm{q})) = 2$. So, $\mathrm{LTL}_\omega^2(\mathsf{F})$ 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).

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 that state constraints on future positions, the following holds.

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

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.

Definition 3.1. [**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.

3.3 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 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 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:

- $\star$ $Q = S$, $Q_0 = \{s\}$, $F = S$,
- $\star$ $(s, a, s') \in \delta$ 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

$$\mathsf{X}^j(\bigwedge_{\mathrm{p}\in\Gamma} \mathrm{p} \wedge \bigwedge_{\mathrm{p}\notin\Gamma} \neg\mathrm{p}))$$

We write ψ_σ to denote the formula below:

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

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

4 Lecture 4: 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. [**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 $\sigma : \mathbb{N} \to \mathcal{P}(\text{PROP})$ 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* $\sigma : \mathbb{N} \to \mathcal{P}(\text{PROP})$ 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. 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 $\Gamma_j \subseteq \text{PROP}$ 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 $\psi \in \mathit{sub}(\varphi)$, a naive reading of the above equalities implies that $T[0, \psi], \ldots, T[i+p, \psi]$ 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 $\mathrm{Traces}(\mathcal{M}, s)$ to denote the set of infinite computations whose initial state is s. Hence, $\mathrm{Traces}(\mathcal{M}, s)$ is a possibly infinite set of ω-words in $\mathcal{P}(\mathrm{PROP})^\omega$. We write $\mathrm{PROP}(\mathcal{M})$ to denote the finite set of atomic propositions occurring in the image of L. When $\mathcal{M}$ is finite, its length $|\mathcal{M}|$ 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 $\mathrm{Traces}(\mathcal{M}, s)$ satisfy a given LTL formula. More formally, the *(universal) local model-checking problem for* LTL, denoted by $\mathrm{MC}^\forall(\mathrm{LTL})$, 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 $\mathrm{PROP}(\varphi)$. The assumption that R is total is not really essential but it allows to simply some technicalities.

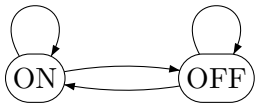
We leave to the reader the proof of the lemma below.

Lemma 4.2. $\mathcal{M}, s \models_\forall \varphi$ 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 $\mathrm{MC}^\exists(\mathrm{LTL})$. 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 4.3. $\mathcal{M}, s \models_\exists \varphi$ 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:

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

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 4.4. There is a reduction from VAL(LTL) to $\mathrm{MC}^{\forall}$(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 $\mathrm{MC}^{\forall}$(LTL). By contrast, the reduction below in the other direction will be a logarithmic space many-one reduction.

Proposition 4.5. There is a logarithmic space reduction from $\mathrm{MC}^{\forall}$(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:

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

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

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

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

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

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

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

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 4.5 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 4.6. [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 $\psi_1 \wedge \psi_2 \in cl_{\mathrm{LTL}}(\varphi)$, $\psi_1 \wedge \psi_2 \in \Gamma$ iff $\psi_1, \psi_2 \in \Gamma$,
- for every $\psi_1 \vee \psi_2 \in cl_{\mathrm{LTL}}(\varphi)$, $\psi_1 \vee \psi_2 \in \Gamma$ iff $\psi_1 \in X$ or $\psi_2 \in \Gamma$,
- for every $\neg\psi \in cl_{\mathrm{LTL}}(\varphi)$, $\neg\psi \in \Gamma$ iff $\psi \notin \Gamma$,
- for every $\psi_1 \mathsf{U} \psi_2 \in cl_{\mathrm{LTL}}(\varphi)$, $\psi_2 \in \Gamma$ or $\psi_1, \mathsf{X}(\psi_1 \mathsf{U} \psi_2) \in \Gamma$.

The proof of the following two lemmas are routine.

Lemma 4.7. Let σ be a structure, φ be an LTL formula and $i \geq 0$. Then, $cl_{\mathrm{LTL}}(\varphi, \sigma, i)$ is maximally consistent.

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

Lemma 4.8. Let σ be a structure, φ be an LTL formula and $i \geq 0$. Then, $(cl_{\mathrm{LTL}}(\varphi, \sigma, i), cl_{\mathrm{LTL}}(\varphi, \sigma, i+1))$ is one-step consistent.

Definition 4.2. [**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 $cl_{\mathrm{LTL}}(\varphi)$ 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 4.6 implies that if an LTL-formula is satisfiable then it has a small satisfiability witness. The converse is also true.

Lemma 4.9. 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 $cl_{\mathrm{LTL}}(\varphi)$ 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 4.10. [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 4.11. [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].

5 Lecture 5: 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.

5.1 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.

5.1.1 Syntax and semantics of the reachability Logic TLR

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

The *temporal logic of reachability* TLR extends TL with additional *(existential) reachability modalities* EF_a, one for each transition relation R_a, with semantics:

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

Thus, $\mathsf{EF}_a\varphi$ 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: $\mathsf{AG}_a := \neg\, \mathsf{EF}_a \neg$ and have the respective semantics:

> $\mathcal{M}, s \models \mathsf{AG}_a\varphi$ if $\mathcal{M}, t \models \varphi$ for every $t \in S$ such that $s \stackrel{a}{\longmapsto}^* t$.

Thus, $\mathsf{AG}_a\varphi$ 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.

5.1.2 Bisimulation games and bisimulation invariance of TLR-formulae

The notion of bisimulation game can be extended to *TLR-bisimulation game* by allowing player **I** to choose between two types of moves: *step-move* along single step transitions, and *path-move*, along finite paths. The former are as in the usual bisimulation games, while in

the latter, in a configuration $(\mathcal{M}_1, s_1; \mathcal{M}_2, s_2)$ player **I** is allowed to choose a finite path in $\mathcal{M}_i$ starting from s_i for $i \in \{1, 2\}$ and pick the last state t_i of that path; then player **II** must respond by choosing a finite path in the other ITS $\mathcal{M}_{3-i}$ and picking its last state s_{3-i}. The resulting configuration is $(\mathcal{M}_1, t_1; \mathcal{M}_2, t_2)$ and the rest of the game is the same.

Note that the TLR-bisimulation game are equivalent to the usual bisimulation games, in a sense that, for any given initial configuration a player **II** has a winning strategy in one iff that player has a winning strategy in the other; consequently, same applies to **I**, too. Indeed, if player **II** has a winning strategy for the TLR-bisimulation game then the same winning strategy will work for the TL-game. Conversely, if player **II** has a winning strategy for the TL-game, then any path-move of player **I** in the respective TLR-bisimulation game can be represented as a series of step-moves; all player **II** needs to do in her response is to follow step by step her winning strategy in responding to the successive step-moves.

On the other hand, it is quite easy to re-state Theorem 2.6 and Corollary **??** to apply for TLR, where the modal depth of a TLR-formula is computed by considering EX and EF as two separate modalities and replacing bisimulation game by TLR-bisimulation games:

Proposition 5.1. Let $(\mathcal{M}_1, s_1)$ and $(\mathcal{M}_2, s_2)$ be rooted interpreted transition systems.

1. If player **II** has a winning strategy for the n-round TLR-bisimulation game with initial configuration $(\mathcal{M}_1, s_1; \mathcal{M}_2, s_2)$ then $(\mathcal{M}_1, s_1) \equiv^n_{\mathrm{TLR}} (\mathcal{M}_2, s_2)$.

2. If player **II** has a winning strategy for the TLR-bisimulation game with initial configuration $(\mathcal{M}_1, s_1; \mathcal{M}_2, s_2)$ then $(\mathcal{M}_1, s_1) \equiv_{\mathrm{TLR}} (\mathcal{M}_2, s_2)$.

Proof:

1. Minor modification of the proof of Theorem 2.6.

2. Immediate from 1.

QED

Consequently, we find that the formulae of TLR are invariant under bisimulations:

Proposition 5.2. [bisimulation invariance of TLR-formulae] If $(\mathcal{M}_1, s_1)$ and $(\mathcal{M}_2, s_2)$ are rooted interpreted transition systems, such that $(\mathcal{M}_1, s_1) \rightleftarrows (\mathcal{M}_2, s_2)$, then $(\mathcal{M}_1, s_1) \equiv_{\mathrm{TLR}} (\mathcal{M}_2, s_2)$.

Proof: By Proposition 5.1, using the equivalence between TL-bisimulation games and TLR-bisimulation game. QED

5.1.3 Characterization of finite transition systems up to bisimulation equivalence with TLR

We have already seen in Lecture 3 that LTL is sufficiently expressive to define any ultimately periodic computation in a finite transition system up to isomorphism. Moreover, with LTL one can characterize any finite rooted ITS up to trace equivalence.

However, LTL cannot characterize an ITS up to bisimulation equivalence: there are easy examples of non-bisimilar ITS satisfying the same LTL-formulae. On the other hand, we will

show that TLR is expressive enough to describe every finite rooted ITS up to bisimulation equivalence[3].

First, let us recall some results from section 2.8.1:

1. With every rooted ITS $(\mathcal{M}, r)$, where $\mathcal{M} = (S, R, L)$ and $n \in \mathbb{N}$ we associate a *characteristic formula of depth n*: $\chi^n_{[\mathcal{M},r]} \in \mathrm{TL}$.

2. When $\mathcal{M}$ is finite, with every $r \in \mathcal{M}$ we associate the characteristic formula $\chi_{[\mathcal{M},r]}$ (which is $\chi^n_{[\mathcal{M},r]}$ for a large enough n) characterizing r up to bisimulation equivalence within $\mathcal{M}$.

Theorem 5.3. (Cf. [BCG88])

For every finite rooted ITS $(\mathcal{M}, r)$, where $\mathcal{M} = (S, R, L)$ there is a TLR-formula $\Phi_{[\mathcal{M},r]}$ that characterizes $(\mathcal{M}, r)$ up to bisimulation equivalence, i.e., for any rooted ITS $(\mathcal{M}', r')$, the following are equivalent:

1. $\mathcal{M}', r' \models \Phi_{[\mathcal{M},r]}$.

2. $(\mathcal{M}', r') \rightleftarrows (\mathcal{M}, r)$.

Proof: We first define for every $s \in \mathcal{M}$ the formula

$$\Xi_{[\mathcal{M},s]} := \mathsf{AG}\left(\chi_{[\mathcal{M},s]} \rightarrow \bigwedge_{sRt} \mathsf{EX}\chi_{[\mathcal{M},t]} \wedge \mathsf{AX}\bigvee_{sRt} \chi_{[\mathcal{M},t]}\right).$$

Now, we define

$$\Phi_{[\mathcal{M},r]} := \chi_{[\mathcal{M},r]} \wedge \bigwedge_{s\in S} \Xi_{[\mathcal{M},s]}.$$

The implication $2 \Rightarrow 1$ is immediate from the bisimulation invariance of TLR-formulae, Proposition 5.2.

For the implication $1 \Rightarrow 2$ it suffices to define a winning strategy for player **II** for the TL-game between $(\mathcal{M}', r')$ and $(\mathcal{M}, r)$ with initial configuration $(\mathcal{M}, r; \mathcal{M}', r')$, using the formula $\Phi_{[\mathcal{M},r]}$ as follows: at every round of the game, player **II** responds to the choice of player **I** by choosing a state satisfying the same characteristic formula $\chi_{[\mathcal{M},t]}$ as the choice of player **I**. That such strategy can be proved by induction on the number of rounds: at round 0 that follows from the fact that $\mathcal{M}', r' \models \chi_{[\mathcal{M},r]}$. Now, suppose that $\mathcal{M}', s' \models \chi_{[\mathcal{M},s]}$, i.e., i.e. the condition of the winning strategy is satisfied by the current configuration $(\mathcal{M}, s; \mathcal{M}', s')$ of round n. This, and the assumption $\mathcal{M}', r' \models \Phi_{[\mathcal{M},r]}$, imply that

(*) $\mathcal{M}', s' \models \bigwedge_{sRt} \mathsf{EX}\chi_{[\mathcal{M},t]}$, and

(**) $\mathcal{M}', s' \models \mathsf{AX}\bigvee_{sRt} \chi_{[\mathcal{M},t]}$.

Now, consider the two cases for the possible move of player **I**:

1. If player **I** chooses a successor t of s in $\mathcal{M}$, then, by (*), $\mathcal{M}', t' \models \chi_{[\mathcal{M},t]}$ for some $t' \in \mathcal{M}'$, and any such t' is a good choice of player **II**.

2. If player **I** chooses a successor t' of s' in $\mathcal{M}'$, then, by (**), $\mathcal{M}', t' \models \chi_{[\mathcal{M},t]}$ for some $t \in \mathcal{M}$, and any such t is a good choice of player **II**.

[3]This was first established in [BCG88] for CTL, but was essentially known earlier in modal logics, see e.g., [BdRV01a].

That completes the induction. QED

Corollary 5.4. Two finite rooted interpreted transition systems are locally bisimilar iff they satisfy the same TLR-formulae.

5.2 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*.

5.2.1 Language and syntax

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

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

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

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, $\mathsf{E}\varphi$ is defined as $\neg\,\mathsf{A}\,\neg\varphi$ 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 $\mathsf{A}\varphi \in$ 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.

5.2.2 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.

- $\star$ $\mathcal{M}, \pi \models p$ iff $p \in L(\pi(0))$ for $p \in$ PROP;
- $\star$ $\mathcal{M}, \pi \models \neg\varphi$ iff $\mathcal{M}, \pi \not\models \varphi$;
- $\star$ $\mathcal{M}, \pi \models \varphi \wedge \psi$ iff $\mathcal{M}, \pi \models \varphi$ and $\mathcal{M}, \pi \models \psi$;
- $\star$ $\mathcal{M}, \pi \models \mathsf{X}\varphi$ iff $\mathcal{M}, \pi_{\geq 1} \models \varphi$;
- $\star$ $\mathcal{M}, \pi \models \varphi \mathsf{U} \psi$, 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 $0 \leq i < j$.
- $\star$ $\mathcal{M}, \pi \models \mathsf{A}\,\varphi$ 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:

- $\star$ 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 $\pi(0) = s$.
- $\star$ a formula φ is *valid in* $\mathcal{M}$, denoted $\mathcal{M} \models \varphi$, if $\mathcal{M}, s \models \varphi$ for every state s in $\mathcal{M}$.
- $\star$ 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 $\mathcal{T}$.
- $\star$ a formula φ is *valid*, denoted $\models \varphi$, if it is valid in every transition system.
- $\star$ a path formula φ is *satisfiable* if it is true of some path π in some interpreted transition system $\mathcal{M}$.
- $\star$ 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, $p \Rightarrow \mathsf{A}\,p$ is valid for any $p \in$ PROP, while $Gp \Rightarrow \mathsf{A}\,Gp$ 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 5.1. 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 $(\mathsf{EX}, \mathsf{EF})$-fragment of CTL*.

5.2.3 Some useful validities in CTL*

Proposition 5.5. The following formulae are CTL*-valid:

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

Proof: Exercise. QED

5.2.4 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:*

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

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

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

⋆ *Total correctness along every possible computation:*

$$\varphi \Rightarrow \mathsf{A}\,\mathsf{F}(\mathsf{terminal} \wedge \psi).$$

⋆ and *total correctness along some possible computation:*

$$\varphi \Rightarrow \mathsf{E}\,\mathsf{F}(\mathsf{terminal} \wedge \psi).$$

⋆ *Fairness along every possible computation:*

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

etc.

5.2.5 Bisimulation Invariance of CTL*

Proposition 5.6. If $(\mathcal{M}_1, r_1) \overset{\beta}{\rightleftarrows} (\mathcal{M}_2, r_2)$, then $(\mathcal{M}_1, r_1)$ and $(\mathcal{M}_2, r_2)$ satisfy the same state formulae of CTL*.

Proof: Induction on the state formulae of CTL*, using Proposition 2.2. QED

Question: What about the converse? Read further.

A notion of bisimulation can be accordingly introduced as a relation between paths rather that states; see [Sti92] for details.

5.2.6 Addendum: generalized semantics for CTL*

The semantics of CTL* given above takes all paths in the transition system into account. This is not always necessary, and sometimes it is even *not reasonable* because some paths could be forbidden by liveness or fairness conditions imposed on the transition system. On the other hand, in order to give meaningful semantics, there ought to be *sufficiently many* available paths. The situation here is similar to Ockhamist semantics over bundled trees.

We are going to generalize the semantics of CTL* by considering models based on pairs $(\mathcal{T}, \Pi)$ where $\mathcal{T}$ is a labelled transition system and Π is a family of *recognized paths* in $\mathcal{T}$. A minimal reasonable requirement for such a family is that it must be *covering*: every state must belong to some recognized path. This, however, is not sufficient, because the truth definitions of the temporal operators invoke *suffixes* of a path, which may not be in the family. Recall, a suffix of a path π is every path $\pi_{\geq k}$ obtained from π by chopping off the first k states. Thus, we impose the additional requirement of *suffix closure:* every suffix of a path from Π must be in Π.

A pair $(\mathcal{T}, \Pi)$ where Π is a covering and suffix closed family of paths in $\mathcal{T}$ will be called a *generalized branching time structure*. An interpreted generalized branching time structure is a triple $(\mathcal{T}, \mathsf{P}(\mathcal{T}), L)$, where L is a state description in $\mathcal{T}$. The semantics of CTL* can be generalized over such structures with no complications, just by restricting the path quantifications to the family Π.

Given a transition system $\mathcal{T}$ the set of all paths on $\mathcal{T}$ will be denoted by $\mathsf{P}(\mathcal{T})$. A generalized branching time structure $(\mathcal{T}, \mathsf{P}(\mathcal{T}))$ will be called *standard* or, following [Eme83], *R-generated*.

Note that this generalized semantics is *not equivalent* to the standard one, because not every formula valid in all transition systems is valid in all generalized branching time structures. One example is $\mathsf{AX}\varphi \Rightarrow \mathsf{X}\,\mathsf{A}\,\varphi$. (See more examples in the exercises.)

Working through this example, one can see that the reason for the possible failure of that formula is that a path (falsifying φ) may belong to Π while its extension one step backwards is not in Π. Thus, another natural closure condition emerges: a family of paths Π is *prefix closed* if whenever a path π belongs to Π and $sR\pi(0)$ then the path $s.\pi$ obtained by prefixing π with s must belong to Π, too.

Given a path $\pi = \pi(0), \pi(1), ...$ we denote by $\pi_{\leq n}$ its *initial segment* $\pi(0), \pi(1), \ldots, \pi(n)$. Given two paths π and π', such that $\pi(n) = \pi'(m)$, the path $\pi_{\leq n}.\pi'_{\geq m+1}$ obtained by appending $\pi'_{\geq m+1}$ to $\pi_{\leq n}$ is called a *fusion* of π and π'.

A family of paths is called *fusion closed* if every fusion of paths from Π belongs to Π.

Proposition 5.7. Suffix closure and prefix closure together imply fusion closure.

Proof: Exercise. QED

Yet another natural closure condition, satisfied by standard models is *limit closure:* if π is a path such that for every $n \in \mathbb{N}$ there is a path ρ^n such that the fusion $\pi_{\leq n}.\rho^n_{\geq n}$ belongs to Π, then π must belong to Π, too.

Theorem 5.8. [Eme83]

1. A family of paths Π is suffix, fusion, and limit closed iff there is a transition system $\mathcal{T}$ such that $\Pi = \mathsf{P}(\mathcal{T})$.

2. A generalized branching time model $(\mathcal{T}, \Pi)$ is R-generated iff it is prefix and limit closed.

Proof: Exercise. QED

For more discussion on the standard vs generalized semantics for CTL* see [Sti92], as well as [Tho84] and [Zan96], from the viewpoint of complete trees vs bundled trees in Ockhamist branching time logics.

5.2.7 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 $\mathsf{A}\,\mathsf{GF}\varphi$ and $\mathsf{E}(\mathsf{F}\varphi \wedge \varphi\mathsf{U}\psi)$ 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, $\mathsf{A}(\varphi_1 \mathsf{U} \varphi_2)$ and $\mathsf{E}(\varphi_1 \mathsf{U} \varphi_2)$ are not inter-definable in CTL, so both path quantifiers must be present in the language. Here is the recursive definition of CTL-formulae:

$$\varphi := p \mid \bot \mid (\varphi_1 \Rightarrow \varphi_2) \mid \mathsf{AX}\varphi \mid \mathsf{A}(\varphi_1 \mathsf{U} \varphi_2) \mid \mathsf{E}(\varphi_1 \mathsf{U} \varphi_2).$$

Some definable operators in CTL:

- $\mathsf{EX}\varphi := \neg\, \mathsf{AX} \neg \varphi$,
- $\mathsf{A}\,\mathsf{F}\varphi := \mathsf{A}(\top \mathsf{U} \varphi)$,
- $\mathsf{E}\,\mathsf{F}\varphi := \mathsf{E}(\top \mathsf{U} \varphi)$,
- $\mathsf{A}\,\mathsf{G}\varphi := \neg\, \mathsf{E}\,\mathsf{F} \neg \varphi$,
- $\mathsf{E}\,\mathsf{G}\varphi := \neg\, \mathsf{A}\,\mathsf{F} \neg \varphi$.

The semantics of CTL is the same as CTL*.

Exercise 5.2. Show that $\mathsf{A}\,\mathsf{U}$ is definable in terms of E-prefixed operators.

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

5.2.8 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[4]. 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:

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

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

5.2.9 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.

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

6 Lecture 6: Model-checking and satisfiability testing of branching-time logics.

7 Model-checking of CTL*reduced 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 $\psi_1, \ldots, \psi_n$ of φ and replace them uniformly with new atomic propositions $p_1, \ldots, p_n$. Note that each ψ_i is a boolean combination of atomic propositions and formulae of the type $\mathsf{A}\,\theta$ where θ is a CTL*-formula of path quantifier rank less than the one of φ. Besides, the result of substitution of $p_1, \ldots, p_n$ respectively for $\psi_1, \ldots, \psi_n$ in φ is an LTL-formula φ'. Using the inductive argument, we can globally model check each of $\psi_1, \ldots, \psi_n$ and compute the sets of states $X_1, \ldots, X_n$ 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 $p_1, \ldots, p_n$ to be true respectively in the sets of states $X_1, \ldots, X_n$. 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].

7.1 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 CTL model-checking problem MC(CTL) can be defined as follows:

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

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

Proposition 7.1. Let $\mathcal{M} = (W, R, L)$ be a CTL model and φ be a CTL formula. Computing the set $\|\varphi\|_{\mathcal{M}} = \{s \in W : \mathcal{M}, s \models \varphi\}$ 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 a subset of $\mathcal{P}(\mathrm{PROP}(\varphi))$ 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 $\mathcal{O}(\text{card}(W) \times |\varphi|)$.

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,

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

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 $\mathcal{O}(\text{card}(W) + \text{card}(R))$.

For technical convenience, here we will assume that the primitive temporal operators in CTL are the EX-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: $\varphi_i = \mathsf{EX}\varphi_{i_1}$ 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: $\varphi_i = \mathsf{E}(\varphi_{i_1}\mathsf{U}\varphi_{i_2})$ for some $i_1, i_2 < i$.
One can show that $\mathcal{M}, s \models \varphi_i$ iff there is a R-path $s_0 \rightarrow s_1 \rightarrow \ldots \rightarrow s_n$ in $\mathcal{M}$ such that

- $\star$ $s_0 = s$,
- $\star$ $\varphi_{i_2} \in l(s_n)$ (use of induction hypothesis for the correctness),
- $\star$ 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 7.2. 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: $\varphi_i = \mathsf{EG}\varphi_{i_1}$ 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 $\text{card}(C) > 1$ or ($C = \{s'\}$ and $(s', s') \in R'$) (C non-trivial).

Suppose that $\mathcal{M}, s \models \mathsf{EG}\varphi_{i_1}$. 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 $\mathcal{O}(\text{card}(W) + \text{card}(R))$, we need to use the following result from graph theory.

Lemma 7.3. Let $G = (W, R)$ be a directed graph represented by lists of neighbours. Computing the partition of maximal SCC can be computed in time $\mathcal{O}(\text{card}(W) + \text{card}(R))$.

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 $\mathcal{O}(\text{card}(W) + \text{card}(R))$.

QED

Corollary 7.4. MC(CTL) is in PTime.

Proposition 7.5. MC(CTL) is PTime-hard.

Proof: PTime-hardness of MC(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

7.2 Tree-model property of CTL*

7.2.1 Tree unfoldings of models for CTL*

Using proposition 5.6 and 2.3 we can obtain the following (note that paths is $\mathcal{M}$ are *states* in $\widehat{\mathcal{M}}$).

Corollary 7.6. For any interpreted transition system $\mathcal{M}$ and a path $\pi \in \mathsf{P}(\mathcal{M})$, we have that:

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

7.2.2 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 7.1. [**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 7.2. [**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[5].

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 $[k] = \{0, \ldots, k-1\}$, as follows: the root is labelled by the empty string ε and the successors of a node labelled by ς are $\varsigma 0, \ldots, \varsigma(k-1)$.

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 $\omega \Rightarrow \mathbf{2}^{\mathrm{PROP}}$.

Proposition 7.7. 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

7.2.3 Satisfiability of CTL* in k-branching trees

Corollary 7.8. Every state formula φ of CTL*, satisfiable in a model with a branching factor κ is satisfiable in a model based on $[\kappa]^*$.

Theorem 7.9. [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

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

Theorem 7.10. [CE81] The satisfiability problem for CTL is ExpTime-complete.

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

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 7.11. 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

7.4 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 7.12.

1. The CTL-formula $\mathsf{A}\,\mathsf{F}\,\mathsf{A}\,\mathsf{G}p$ is not expressible in LTL.
2. The LTL-formula $\mathsf{A}\,\mathsf{FG}p$ 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 $\mathrm{LTL}(\varphi)$ obtained from φ by deleting all path quantifiers. Now, show that for every path π in a CTL*-model $\mathcal{M}$, we have that $M, \pi \models \mathrm{LTL}(\varphi)$ iff $M_\pi, \pi \vDash \varphi$, where M, π is the respective LTL-model while M_π 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].

8 Lecture 7: Tableau method for testing satisfiability and model-checking of temporal logics. Testing satisfiability and model-checking of LTL using tableaux.

The origins of the method of semantic tableau for testing satisfiability of logical formulae go back to Gentzen's work on syntactic tableau, but the method was first explicitly developed by Beth [Bet], [Bet70], and Smullyan [Smu68], and further adapted for modal logics by Fitting [Fit83]. For a survey of tableaux for modal and temporal logics see [Gor98] and for a more recent account see [Fit07]. In particular, a standard tableau for the logic TL can be extracted from any of these.

The complications in the tableaux arise when fixpoint operators are added to the language, i.e., in the cases of the linear and branching time temporal logics studied in this course, as these logics demand a special mechanism for checking realization of eventualities.

In this course I will present a uniform, *incremental tableau* building methodology for temporal logics, going back to works of Pratt [Pra79], [Pra80] (for PDL), Wolper [Wol83], [Wol85] (for LTL), Ben-Ari, Manna, Pnueli [BAPM81] (for the branching-time logic UB) and Emerson, Halpern in [EH82],[EH85] (for the branching-time logic CTL); see bibliographic notes for further details.

All these tableau-based decision procedures for testing satisfiability runs in ExpTime, which is optimal in all cases, except the case of LTL. In that case we will discuss how the procedure can be optimized to PSpace.

For convenience of the reader not previously familiar with this style of tableau-based decision procedures, we will first sketch generic tableaux constructions for testing satisfiability and for model-checking of temporal logics. Then we will present in detail the tableau constructions and proofs of soundness and completeness for the logic LTL, and in Lecture for each of the logics TLR and CTL, building on the generic common notions and results. The expositions for LTL and TLR are relatively independent, while the one for CTL refers more substantially to the TLR case. Thus, while seeing the common methodology, the reader should be able to follow each of these cases fairly independently, but that comes at the expense of some inevitable repetitions, *mutatis mutandis*, of technical details.

8.1 A generic incremental tableau method for testing satisfiability and model-checking of temporal formulae.

First, we will give an informal outline a generic incremental tableaux construction for modal and temporal logics, beginning with some preliminaries.

8.1.1 Preliminaries

The set of all subformulae of a formula η will be denoted by $sub(\eta)$; likewise, the set of all subformulae of a set of formulae Φ will be denoted by $sub(\Phi)$.

Negation normal form. A formula is in *negation normal form* (NNF) if the only occurrences of negations in it, if any, are in front of atomic propositions.

In all logics considered here, every formula can be transformed up to logical equivalence to one in NNF by systematically driving all negations inwards, using de Morgan's laws and other suitable logical equivalences.

Some tableau styles assume the input formula in NNF, as that simplifies some arguments. We will not make this assumption for the sake of efficiency: when the formulae occurring in the tableau are in NNF, patent contradictions are detected only after full decompositions of the involved formulae. Still, for some tasks it is convenient to assume that a formula is in NNF.

Types of formulae. We will distinguish 4 types of formulae:

1. α-*formulae*, of conjunctive type. Every α-formula θ is associated usually with two (but, sometimes can be more or less), possibly coinciding formulae, called the α-*components* of θ. Every α-formula is equivalent to the conjunction of its α-components.

   For instance, the α-components of $\varphi \wedge \psi$ are $\theta_1 = \varphi$ and $\theta_2 = \psi$, and the only α-component of $\neg\neg\varphi$ is $\theta_1 = \varphi$.

2. β-*formulae*, of disjunctive type. Every β-formula θ is associated usually with two (sometimes three), possibly coinciding formulae, called the β-*components* of θ. Every β-formula is equivalent to the disjunction of its β-components.

   For instance, the β-components of $\varphi \vee \psi$ are $\theta_1 = \varphi$ and $\theta_2 = \psi$.

3. *nexttime formulae*: these are formulae beginning with EX, AX (respectively X in LTL) and negations of such formulae.

4. *literals*: $\top$, $\neg\top$, atomic propositions and their negations.

The literals and the nexttime formulae are commonly called *primitive formulae*.

Closure of a formula. Often, in addition to all subformulae of a formula we have to consider their negations, as well as formulae obtained by 'unfolding' formulae from the closure, e.g., unfolding of fixed point operators prefixing subformulae of η. Thus, the notion of a *closure* of the formula η, denoted $cl(\eta)$, emerges. In all cases of logics considered here, the closure will be defined generically as follows.

Definition 8.1. [**Closure of a formula in** LTL]
The closure $cl(\eta)$ of the formula η is the least set of formulae such that:

1. $\top, \varphi \in cl(\eta)$;
2. $cl(\eta)$ is closed under taking subformulae;
3. if $\psi \in cl(\eta)$ and ψ does not begin with $\neg$ then $\neg\psi \in cl(\eta)$;
4. $cl(\eta)$ is closed under taking all components of α-formulae and β-formulae.

∇

Definition 8.2. [**Closure of a set of formulae. Closed sets**]
For any set of formulae Φ we define $cl(\Phi) := \bigcup\{\, cl(\varphi) \mid \varphi \in \Phi \,\}$.
A set of formulae Φ is *closed* if $\Phi = cl(\Phi)$. ∇

The closure of a formula (and hence, of any finite set of formulae) will always be a finite set, of cardinality linear in the length of the formula.

Consistent and fully expanded subsets of a closure

Definition 8.3. [**Patently inconsistent set**] A set of formulae is *patently inconsistent* if it contains $\bot$, or $\neg\top$, or a contradictory pair of formulae $\neg\varphi$ and φ; otherwise it is *patently consistent*. ∇

Here we give a generic definition of a fully expanded set, to be made specified in each of the particular cases of logics considered further.

Definition 8.4. [**Fully expanded set**] A set of formulae Φ is *fully expanded* iff:

1. it is *patently consistent*;
2. for every α-formula in Φ, all of its α-components are in Φ;
3. for every β-formula in Φ, at least one of its β-components is in Φ.

∇

Intuitively, a patently consistent set is fully expanded if it is closed under applications of all *local* (propositional, applying to the same state of the model) formula decomposition rules.

For some logics, specific additional conditions may be imposed on the definition of fully expanded sets.

Definition 8.5. [**Minimal full expansions**] Given a set of formulae Γ, a fully expanded set Δ is a *minimal full expansion of* Γ if $\Gamma \subseteq \Delta$ and there is no fully expanded proper subset of Δ containing Γ. ∇

Note that a set Γ may have none, or several minimal full expansions.

Note that not every fully expanded set is satisfiable. In fact, the purpose of the tableau is to determine whether there is at least one fully expanded set containing the input formula that is satisfiable.

Computing full expansions by saturation under closure operations In the tableaux constructions we will be using a rule calling the procedure FullExpansion, described below, for computing the family FE(Γ) of all full expansions of a given set of formulae Γ obtained by saturation under the closure operations corresponding to the definition of full expansion. That procedure essentially represents a propositional tableau, performing the propositional decomposition steps on the side of the main tableau procedure.

The procedure FullExpansion uses the following *set replacement operations* applied to a set of formulae Φ in a family of sets of formulae $\mathcal{F}$:

(α)**:** If $\varphi \in \Phi$ for some α-formula φ with α-components φ_1 and φ_2, replace Φ by $\Phi \cup \{\varphi_1, \varphi_2\}$.

(β)**:** If $\varphi \in \Phi$ for some β-formula φ with β-components φ_1 and φ_2, replace Φ by $\Phi \cup \{\varphi_1\}$ and $\Phi \cup \{\varphi_2\}$.

An *expansion step* consists in choosing a set Φ from the current family of sets $\mathcal{F}$, and then choosing an α- or β- formula $\varphi \in \Phi$ (if there is any) and applying the respective set replacement operation for φ to Φ, with the following proviso: if a patently inconsistent set is added to $\mathcal{F}$ as a result of such application, it is removed immediately after the replacement.

Now, given a finite set of formulae Γ, the procedure FullExpansion starts with the singleton family $\{\Gamma\}$ and checks if it is patently inconsistent. If so, it returns $\mathrm{FE}(\Gamma) = \emptyset$. Otherwise, it applies repeatedly expansion steps to the current family $\mathcal{F}$ until saturation, i.e. until no application of a set replacement operation can change $\mathcal{F}$. (The stage of saturation is guaranteed to occur because all sets of formulae produced during the procedure FullExpansion are subsets of the finite set $cl(\Gamma)$.) At that stage, the family $\mathrm{FE}(\Gamma)$ of sets of formulae is produced and returned.

We note that $\mathrm{FE}(\Gamma)$ may contain some non-minimal full expansions of Γ; examples will be given further. Nevertheless, we want to keep such non-minimal full expansions obtained by the procedure FullExpansion, too, for reasons that will become clear later.

We will call the sets in $\mathrm{FE}(\Gamma)$ the *lean full expansions* of Γ.

Proposition 8.1.

I. The family of lean full expansions $\mathrm{FE}(\Gamma)$ contains all minimal full expansions of Γ.

II. For any finite set of formulae Γ:

$$\bigwedge \Gamma \equiv \bigvee \{ \bigwedge \Delta \mid \Delta \in \mathrm{FE}(\Gamma) \}.$$

Proof:

I. Exercise.

II. Recall the requirements, that every α-formula is equivalent to the conjunction of its α-components, and every β-formula is equivalent to the disjunction of its β-components. They imply that every set replacement operation applied to a family $\mathcal{F}$ preserves the formula $\bigvee \{ \bigwedge \Delta \mid \Delta \in \mathcal{F} \}$ up to logical equivalence. At the beginning, that formula is $\bigwedge \Gamma$. QED

Exercise 8.1. Prove claim I and complete the details of the proof of claim II.

Eventualities Intuitively, *eventualities* are formulae stating that something will happen eventually in the future, but without specifying exactly when.

In particular:

- the language TL has no eventualities;
- the eventualities in TLR are the formulae of the type $\mathsf{EF}\varphi$ and $\neg\,\mathsf{AG}\varphi$;
- the eventualities in LTL are the formulae of the type $\varphi \mathsf{U} \psi$ (in particular, $\mathsf{F}\varphi$, and $\neg\mathsf{G}\varphi$);
- the eventualities in CTL are:
  - *existential eventualities*, of the type $\mathsf{E}\,\varphi \mathsf{U} \psi$ and $\neg\,\mathsf{AG}\varphi$, referring to a single path on which such eventuality must be realized.
  - *universal eventualities*: of the type $\mathsf{A}\,\varphi \mathsf{U} \psi$ and $\neg\,\mathsf{EG}\varphi$, where the eventuality must be realized on every path.

8.1.2 Hintikka structures

While the notion of *Hintikka structure* (first defined in [Pra79] by analogy with Hintikka sets for propositional logic) is not an explicit part of the tableaux construction, it is fundamental for its understanding, because the purpose of that construction is to check for existence of a Hintikka structure 'satisfying' the input formula.

Intuitively, a Hintikka structure represents a partly defined rooted ITS satisfying the input formula. It is a graph, every node of which is labeled by a set of formulae. These labels are fully expanded subsets of the closure of a designated input formula (in the satisfiability of which we are interested). All desired properties of the transition relations in a Hintikka structure are encoded by means of the labels of the states. Membership to the label of the state of a Hintikka structure simulates the notion of truth of a formula at a state of an ITS, and the labeling of states must ensure that the Hintikka structure can generate a model (or, sometimes, a *pseudo-model*) of the input formula.

Conversely, every rooted ITS uniformly generates a rooted Hintikka structure which 'satisfies' the input formula η at its root, by labelling each state with the set of all formulae from the closure of η that are true at that state. However, an essential difference between ITS and Hintikka structures is that, while an ITS determines the truth value of every formula at every state, a Hintikka structure contains only just enough information to determine the truth values of those formulae that are directly involved in the evaluation of the input formula η at the root state.

For a class of Hintikka structures to be suitable for the tableau procedure, it must be the case that every formula 'satisfiable' in such a Hintikka structure is satisfiable in an ITS, so the two notions of satisfiability are equivalent.

8.1.3 Sketch of a generic tableaux construction for testing satisfiability

The tableau procedure for a given input formula η attempts to construct a non-empty graph $\mathcal{T}^\eta$, called a *tableau*, representing in a way sufficiently many possible Hintikka structures for η. The procedure usually consists of three major phases:

1. Construction phase. In that phase a finite directed graph with labeled vertices $\mathcal{P}^\eta$, called the *pretableau* for η is produced, following prescribed *construction rules*. The set of nodes of the pretableau properly contains the set of nodes of the tableau $\mathcal{T}^\eta$ that is to be ultimately built.

   The pretableau has two types of nodes: *states* and *prestates*. The states are labeled with fully expanded subsets of $cl(\eta)$ and represent states of a Hintikka structure, while the prestates can be labeled with any subsets of $cl(\eta)$ and they play only a temporary auxiliary role.

   When there is no danger of confusion we will identify prestates or states with their labels.

2. Prestate elimination phase. The prestates are removed during this phase, using the *prestate elimination rule*, and the result is a smaller graph $\mathcal{T}_0^\eta$, called the *initial tableau for* η.

3. State elimination phase. During this phase we remove, using *state elimination rules*, all states, if any, from $\mathcal{T}_0^\eta$ that cannot be satisfied in any Hintikka structure, for one of the

following reasons: either their labels contain *unrealizable eventualities*, or some of the successor states they need for the satisfaction of nexttime formulae have been removed in the process.

The state elimination phase results in a (possibly empty) subgraph $\mathcal{T}^\eta$ of $\mathcal{T}_0^\eta$, called the *final tableau for* η.

If there is a state in the final tableau $\mathcal{T}^\eta$ containing η in its label, the tableau is declared *open* and the input formula η is pronounced satisfiable; otherwise, the tableau is declared *closed* and η is pronounced unsatisfiable.

8.1.4 Termination, soundness, and completeness of the tableaux construction

Termination Each of the phases of the above described procedure terminates:

1. The termination of the construction phase follows from the fact that there are only finitely many possible labels of states and prestates to be created in the tableau, and each of them is created at most once.

2. The prestate elimination phase terminates because the elimination of each of the finitely many prestates is a one-step act.

3. The state elimination phase terminates because there are only finitely many eventualities to be checked for realization, each in finitely many state labels, and if any round of testing realization of all eventualities at all states in which labels they occur ends without elimination of any state, the construction is completed.

Estimates for the maximal number of steps needed for the construction of the tableau will be obtained further, in all particular cases, thus yielding upper bound for the time complexity of the decision procedures.

Soundness *Soundness* of the tableau procedure means that if the input formula η is satisfiable, then the final tableau $\mathcal{T}^\eta$ is open, so the tableau procedure is guaranteed to establish the satisfiability. A generic proof of soundness consists in showing that at least one tableau state containing η will survive until the end of the procedure. Note, that if η is satisfiable, and hence propositionally consistent, there will be at least one offspring state of the initial prestate containing η. Moreover, for every rooted ITS $(\mathcal{M}, s)$ satisfying η, the set $\{\psi \in cl(\eta) \mid \mathcal{M}, s \models \psi\}$ contains at least one such state.

Thereafter, it suffices to show that *only unsatisfiable states* get eliminated in the state elimination phase. The proof of that claim is done by induction on the applications of state elimination rules.

Completeness *Completeness* of the tableau procedure means that if the input formula η is not satisfiable, then the final tableau $\mathcal{T}^\eta$ is closed, so the tableau procedure is guaranteed to establish the unsatisfiability. By contraposition, completeness means that if the final tableau is open, the input formula η is satisfiable.

A generic proof of completeness consists in proving that an open final tableau yields a Hintikka structure satisfying η, and then using the equivalence between satisfiability in Hintikka structure and in ITS. In some cases (e.g., TL, TLR) the construction of the Hintikka

structure satisfying η from an open tableau for η is straightforward – the tableau itself can be used for that purpose. In other cases (e.g., LTL, CTL) the Hintikka structure satisfying η has to be pieced together from *fragments* extracted from the tableau.

8.1.5 A generic tableaux method for model-checking

The tableau procedures for testing satisfiability can be smoothly modified to perform *local model-checking*: given a finite rooted ITS $(\mathcal{M}, r)$ and a formula η, the model-checking tableau must decide whether $\mathcal{M}, r \models \eta$.

The idea of tableau-based model-checking is simple: in a nutshell, the procedure simulates the construction of the pretableau in $\mathcal{M}$ by starting from r and advancing along the transitions in $\mathcal{M}$, while labeling the prestates and states of the tableau with the respective current states of $\mathcal{M}$. More precisely:

1. The states and prestates of the tableau are now labeled by pairs (s, Θ) where $s \in \mathcal{M}$ and $\Theta \subseteq cl(\eta)$, where in the case of states Θ is fully expanded.

2. The pretableau construction phase is similar to the one in the tableaux for satisfiability, but with modified construction rules, taking into account that the states and prestates they create are properly associated with states of $\mathcal{M}$.

3. The prestate and state elimination phases are essentially the same as in the tableaux for satisfiability.

4. When the final tableau is obtained, $\mathcal{M}, r \models \eta$ is declared true iff there is a state (r, Δ) in it such that $\eta \in \Delta$.

The soundness of the procedure follows from the following observations:

1. If a prestate in the pretableau is satisfied in some state of $\mathcal{M}$ then some offspring state of it is satisfied there, too.

2. A state (s, Δ) from the initial tableau is eliminated during the state elimination phase only if some formula from Δ is not true at s.

3. For every state (s, Δ) in the final tableau, $\mathcal{M}, s \models \Delta$ holds.

8.2 Testing satisfiability and model-checking of LTL using tableaux.

Now, we will illustrate the generic tableau construction outlined in the previous section with the logic LTL. Most, but not all, of its features are simpler than the case of branching time logics. The major distinction is that in the case of LTL we are looking for a *linear model* satisfying the input formula.

For simplicity of the exposition, we will consider LTL over a language with one transition relation; in the case of many transition relations the same procedure applies, *mutatis mutandis*, treating all temporal operators over the different transitions in the same way.

The first construction of tableau for LTL, a version of which will be presented here, was given in [Wol83], [Wol85]. It takes exponential time and space in the length of the formula, but we will discuss further how it can be optimized to work in polynomial space.

For some variations and improvements of Wolper's tableau procedure for LTL see [KMMP93] (for LTL with past) and [LP00].

8.2.1 Preliminaries

We will choose to work with a minimal language, containing $\top$, $\neg$ and $\wedge$ as Boolean connectives, and X and U as temporal operators. The other Boolean constants and connectives $\top, \bot, \vee, \Rightarrow$, as well as the operators F, G, and R will be assumed definable in a standard way. We leave it as a running exercise to the reader to extend the basic concepts and steps of the procedure outlined below to a language containing some, or all, of these connectives as primitives.

The *nexttime formulae* in LTL are $\mathsf{X}\varphi$ and $\neg\mathsf{X}\varphi$.
The α- and β-formulae in LTL and their components are:

α	α_1	α_2
$\neg\neg\varphi$	φ	φ
$\neg\mathsf{X}\varphi$	$\mathsf{X}\neg\varphi$	$\mathsf{X}\neg\varphi$
$\varphi \wedge \psi$	φ	ψ
$\neg(\varphi\mathsf{U}\psi)$	$\neg\psi$	$\neg\varphi \vee \neg\mathsf{X}(\varphi\mathsf{U}\psi)$

β	β_1	β_2
$\neg(\varphi \wedge \psi)$	$\neg\varphi$	$\neg\psi$
$\varphi\mathsf{U}\psi$	ψ	$\varphi \wedge \mathsf{X}(\varphi\mathsf{U}\psi)$

The only type of eventualities in LTL is $\varphi\mathsf{U}\psi$.

Lemma 8.2.

I. For every α-formula φ: $\varphi \equiv \alpha_1(\varphi) \wedge \alpha_2(\varphi)$.

II. For every β-formula φ: $\varphi \equiv \beta_1(\varphi) \vee \beta_2(\varphi)$.

Proof: Exercise. QED

The notions of closure of an LTL-formula, fully expanded sets, and the procedure FullExpansion computing the family of lean full expansions of a set of LTL-formulae, are defined as in Lecture 4.

Exercise 8.2. Show that $cl(\varphi)$ is finite for every $\varphi \in$ LTL.

Exercise 8.3. Give explicitly the closure conditions for fully expanded sets and the set-replacement operations of the procedure FullExpansion for the LTL-formulae $\neg\mathsf{X}\varphi$, $\varphi\mathsf{U}\psi$, $\mathsf{F}\varphi$, $\neg(\varphi\mathsf{U}\psi)$, and $\mathsf{G}\varphi$.

Note that for some sets of formulae Γ, the family $\mathrm{FE}(\Gamma)$ may contain some non-minimal full expansions of Γ. For instance, if $\Gamma = \{p\mathsf{U}q, \mathsf{X}(p\mathsf{U}q)\}$ then $\mathrm{FE}(\Gamma) = \{\{p\mathsf{U}q, \mathsf{X}(p\mathsf{U}q)\}, \{q, p\mathsf{U}q, \mathsf{X}(p\mathsf{U}q)\}\}$. In such case we want to keep such larger full expansions obtained by the procedure FE, too, for reasons that will become clear later.

8.2.2 Hintikka traces

Since we are interested in satisfiability of LTL-formulae in linear models, we will only define the notion of linear Hintikka structure, or simply Hintikka trace.

Definition 8.6. [**Hintikka trace**] Given a (usually closed) set of formulae Φ, a *Hintikka trace* (HT) for Φ is a mapping $\mathcal{H} : \mathbb{N} \to \mathcal{P}(\Phi)$ satisfying the following conditions for every $n \in \mathbb{N}$:

H1 $\mathcal{H}(n)$ is fully expanded;

H2 If $\mathsf{X}\varphi \in \mathcal{H}(n)$, then $\varphi \in \mathcal{H}(n+1)$

H3 If $\varphi \mathsf{U} \psi \in \mathcal{H}(n)$, then there exists $i \geq 0$ such that $\psi \in \mathcal{H}(n+i)$ and $\varphi \in \mathcal{H}(n+j)$ for every j such that $0 \leq j < i$.

∇

Proposition 8.3. In every Hintikka trace $\mathcal{H}$:

1. If $\neg(\varphi \mathsf{U} \psi) \in \mathcal{H}(n)$, then for every $i \in \mathbb{N}$ if $\neg\psi \in \mathcal{H}(n+i)$ then $\varphi \in \mathcal{H}(n+j)$ for some j such that $0 \leq j < i$.
2. If $\mathsf{G}\varphi \in \mathcal{H}(n)$, then $\varphi \in \mathcal{H}(n+i)$ for every $i \in \mathbb{N}$.

Proof: Exercise. QED

Definition 8.7. We say that a formula $\theta \in$ LTL is satisfiable in a Hintikka trace $\mathcal{H}$ if $\theta \in \mathcal{H}(n)$ for some $n \in \mathbb{N}$. ∇

Lemma 8.4. For any set of formulae Φ, every linear ITS $\mathcal{M} = (\mathbb{N}, L)$ generates a Hintikka trace $\mathcal{H} : \mathbb{N} \to \mathcal{P}(\Phi)$ for Φ, where $\mathcal{H}(n) = \{\varphi \in \text{LTL} \mid \mathcal{M}, n \models \varphi\}$ for every $n \in \mathbb{N}$.

Proof: Straightforward verification of H1-H3. Exercise. QED

Usually, we will be interested in Hintikka traces for sets $cl(\eta)$, where η is a formula for which we want to find a model.

Theorem 8.5. A formula $\eta \in$ LTL is satisfiable iff it is satisfiable in a Hintikka trace for $cl(\eta)$.

Proof: One direction is immediate by Lemma 8.4 for $\Phi = cl(\eta)$. For the converse, suppose $\eta \in \mathcal{H}(m)$ for some Hintikka trace $\mathcal{H} : \mathbb{N} \to \mathcal{P}(cl(\eta))$ and $m \in \mathbb{N}$. Without restriction, we can assume that $m = 0$. We define the following state description L in $\mathbb{N}$: $L(n) := \text{PROP} \cap \mathcal{H}(n)$. Let $\mathcal{M} = (\mathbb{N}, \mathsf{succ}, L)$, where succ is the successor relation in $\mathbb{N}$.

We now show by induction on $\theta \in$ LTL that for every $n \in \mathbb{N}$:

(i) if $\theta \in \mathcal{H}(n)$ then $\mathcal{M}, n \models \theta$;

(ii) if $\neg\theta \in \mathcal{H}(n)$ then $\mathcal{M}, n \models \neg\theta$.

When $\theta \in$ PROP the claim is immediate by definition of L.

Let $\theta = \neg\psi$. Then, clause (i) is part of the inductive hypothesis for ψ, while clause (ii) follows again from the inductive hypothesis for ψ and the fact that every $\mathcal{H}(n)$ is fully expanded, so if $\neg\neg\psi \in \mathcal{H}(n)$ then $\psi \in \mathcal{H}(n)$.

The case where $\theta = \theta_1 \wedge \theta_2$ is straightforward, using the inductive hypothesis and the fact that $\mathcal{H}(n)$ is fully expanded.

The case of $\theta = \mathsf{X}\varphi$ follows from the inductive hypothesis and the conditions H1 and H2 of the definition of Hintikka trace.

The case of $\theta = \varphi\mathsf{U}\psi$ follows from the inductive hypothesis, the conditions H1, and H3, and Proposition 8.3. QED

Exercise 8.4. Complete the details of the proof above.

Given an input formula $\eta \in$ LTL we are now going to build a tableau for testing the satisfiability of η. The procedure will consists of 3 phases, as described in Lecture 4.

8.2.3 The pretableau construction phase

The pretableau construction phase consists in alternating applications of two rules:

PrExp$^{\mathrm{LTL}}$: producing all *offspring states* of a given prestate;

Next$^{\mathrm{LTL}}$: producing the *successor prestate* of a given state.

The rule PrExp$^{\mathrm{LTL}}$ involves the procedure FullExpansion, described in Lecture 4, for computing the family FE(Γ) of all full expansions of a given subset Γ of $cl(\eta)$ obtained by saturation under the closure operations corresponding to the definition of full expansion.

> Rule PrExp$^{\mathrm{LTL}}$: Given a prestate Γ to which the rule has not yet been applied, do the following:
>
> 1. Compute the family FE(Γ) = $\{\Delta_1, \ldots \Delta_n\}$ of all lean full expansions of Γ and add these as (labels of) new states in the pretableau, called the *offspring states of* Γ.
> 2. For each newly introduced state Δ, create an edge $\Gamma \Longrightarrow \Delta$.
> 3. If, however, the pretableau already contains a state with label Δ then do not create a new state with that label, but create an edge $\Gamma \Longrightarrow \Delta$ to that state.

We denote the set $\{\, \Delta \mid \Gamma \Longrightarrow \Delta \,\}$ of offspring states of Γ by $\mathsf{states}(\Gamma)$.

> Rule Next$^{\mathrm{LTL}}$: Given a state with label Δ, to which the rule has not yet been applied, do the following:
>
> 1. Add a successor prestate Γ of Δ with label $\{\, \psi \mid \mathsf{X}\psi \in \Delta \,\}$.
> 2. If the label of Δ does not contain any formula of the type $\mathsf{X}\psi$ then add a successor prestate Γ of Δ with label $\{\top\}$.

3. Create an edge $\Delta \longrightarrow \Gamma$.
4. If, however, the pretableau already contains a prestate with label Γ then do not create a new prestate with that label, but extend an arrow $\Delta \longrightarrow \Gamma$ to that prestate.

The construction phase of building a pretableau for η begins with creating a single prestate $\{\eta\}$, followed by alternating applications of the rules PrExp and Next, respectively to the prestates and the states created at a previous stage of the construction. The construction phase ends when none of these rules can add any new states or prestates to the current graph. The resulting graph is the pretableau $\mathcal{P}^\eta$.

8.2.4 Prestate elimination phase

In this phase, all prestates are removed from $\mathcal{P}^\eta$, together with their incoming and outgoing arrows, by applying the following *prestate elimination rule*:

Rule PrElim$^{\mathrm{LTL}}$: For every prestate Γ in $\mathcal{P}^\eta$, do the following:

1. Remove Γ from $\mathcal{P}^\eta$;
2. If there is a state Δ in $\mathcal{P}^\eta$ with $\Delta \longrightarrow \Gamma$, then for every state $\Delta' \in \mathsf{states}(\Gamma)$, create an edge $\Delta \longrightarrow \Delta'$.

The resulting graph is called the *initial tableau* for η, denoted $\mathcal{T}_0^\eta$. The offspring states of the input prestate $\{\eta\}$ are called *input states* of $\mathcal{T}_0^\eta$.

8.2.5 State elimination phase

The purpose of the state elimination phase is to remove from the tableau all 'bad' states, the labels of which are not satisfiable in any Hintikka trace, and hence in any linear model for LTL. That will be done using the *state elimination rules* StElim1$^{\mathrm{LTL}}$ and StElim2$^{\mathrm{LTL}}$, introduced below. The state elimination phase is carried out in a sequence of stages, starting at stage 0 with the initial tableau $\mathcal{T}_0^\eta$, and eliminating at every stage n at most one state for the current tableau $\mathcal{T}_n^\eta$, by applying one of the state elimination rules, to produce the new current tableau $\mathcal{T}_{n+1}^\eta$.

One possible reason for existence of a 'bad' state in a current tableau is that it may lack a successor state. Such states can arise because some prestates may have no full expansions, and hence may yield no offspring states. Once such bad states are removed, some of their predecessor states may be left without successors, so they must be removed, too, etc., until stabilization. Formally, the elimination of states with no successors is done by applying the following rule.

Rule StElim1$^{\mathrm{LTL}}$: If a state Δ has no successor states in the current tableau, then remove Δ from the tableau.

Another reason for a state to be 'bad', i.e., for its label not to be satisfiable in any Hintikka trace, is when that label contains an eventuality formula which is not 'realized' in

any reachable (i.e., future) state. Such cases are handled by an additional state elimination rule. In order to formulate it we have to formalize the notion of *eventuality realization*.

Hereafter, by a path in the current tableau $\mathcal{T}_n^\eta$ we mean a path of successor states $\Delta_0 \longrightarrow \Delta_1 \longrightarrow \ldots$ in $\mathcal{T}_n^\eta$.

Definition 8.8. [**Realization of eventualities in** LTL] An eventuality $\xi = \varphi \mathsf{U} \psi$ is *realized* at the state Δ in $\mathcal{T}_n^\eta$ if there exists a finite path $\Delta = \Delta_0, \Delta_1, \ldots, \Delta_m$, where $m \geq 0$, in $\mathcal{T}_n^\eta$, such that $\xi, \psi \in \Delta_m$ and $\xi, \varphi \in \Delta_i$ for every $i = 0, \ldots, m-1$. We say that ξ is realized on the path $\Delta_0, \Delta_1, \ldots, \Delta_m$ and call any such path a *witness of the realization of the eventuality* ξ *in* Δ.

If $\psi \in \Delta$, then we say that ξ is *immediately realized* at the state Δ (by the singleton path Δ). ∇

There are various algorithms for testing the realization of the eventualities in tableau states. A more efficient approach is the *global* one, where that testing is done for all eventualities at all states simultaneously.

Now, we can state the second state elimination rule.

> Rule StElim2$^{\mathrm{LTL}}$: If an eventuality $\xi \in \Delta$ is not realized on any path starting from Δ in the current tableau, then remove Δ from the tableau.

The rules StElim1$^{\mathrm{LTL}}$ and StElim2$^{\mathrm{LTL}}$ are applied alternatively, in any order, which may be determined by a specific strategy. This procedure continues until reaching a stage when no further elimination of states by an application of any of the rules is possible.

8.2.6 The final tableau

When the state elimination phase reaches a stabilization stage the current tableau at that stage is the *final tableau for* η, denoted by $\mathcal{T}^\eta$, with a set of states denoted by S^η.

Definition 8.9. The final tableau $\mathcal{T}^\eta$ is *open* if $\eta \in \Delta$ for some $\Delta \in S^\eta$; otherwise, $\mathcal{T}^\eta$ is *closed*. ∇

The tableau procedure returns "not satisfiable" if the final tableau is closed; otherwise, it returns "satisfiable" and, moreover, provides sufficient information for producing a finite Hintikka trace satisfying η, as described in the completeness proof below.

8.2.7 Soundness of the tableau for LTL

Recall, that soundness of the tableau procedure means that if the input formula η is satisfiable, then the tableau $\mathcal{T}^\eta$ is open. We will establish the soundness of the tableau for LTL by proving that every state elimination rule is 'sound', in a sense that it never eliminates a state with a satisfiable label. The soundness of the overall procedure is then an immediate consequence.

Lemma 8.6. Let Γ be a prestate in $\mathcal{P}^\eta$ such that $\sigma, i \models \Gamma$ for some linear LTL-model σ and $i \in \mathbb{N}$. Then:

I. $\sigma, i \models \Delta$ for at least one $\Delta \in \mathrm{FE}(\Gamma)$.

II. Moreover, for any eventuality $\varphi \mathsf{U} \psi \in \Gamma$ such that $\sigma, i \models \psi$, the state Δ can be chosen to contain ψ.

Proof: Claim I follows from Proposition 8.1.

For claim II, let $\varphi \mathsf{U} \psi \in \Gamma$ and $\sigma, i \models \varphi$. We start the construction of the desired lean full expansion Δ of Γ by applying the rule to $\varphi \mathsf{U} \psi$ to Γ. Now, it suffices to show that we can complete that construction by applying the set replacement operations, starting with the replacement $\Gamma \cup \{\varphi\}$ of Γ until saturation, by choosing at every step to replace the current set by an extension that is still satisfied at (σ, i). This claim is proved by induction on the number of applications of expansion steps. The base of the induction, for $\Gamma \cup \{\varphi\}$, holds by assumption. The inductive step is carried out by a straightforward consideration of all cases of set replacement operations. (In case of a set replacement operation applied to a β-formula and offering a choice of two extensions, both satisfied at (σ, i), the procedure chooses any of them.) QED

Note, that the so constructed full expansion Δ need not be minimal – indeed, that happens in the example $\Gamma = \{p \mathsf{U} q, \mathsf{X}(p \mathsf{U} q)\}$ given earlier. For that reason we want to keep all lean full expansions in $\mathrm{FE}(\Gamma)$, not only the minimal ones.

Lemma 8.7. No satisfiable state $\Delta \in \mathcal{T}_n^\eta$ is removed by any application of the rules StElim1$^{\mathrm{LTL}}$ and StElim2$^{\mathrm{LTL}}$ during the state elimination phase.

Proof: It suffices to show, by induction on $n \in \mathbb{N}$, that no satisfiable state $\Delta \in \mathcal{T}_n^\eta$ is removed by an application of StElim1$^{\mathrm{LTL}}$ or StElim2$^{\mathrm{LTL}}$ to $\mathcal{T}_n^\eta$. For that purpose we will prove a somewhat stronger inductive claim, viz., that for every $n \in \mathbb{N}$, if $\Delta \in \mathcal{T}_n^\eta$ is satisfiable then:

1. There is a satisfiable state $\Theta \in \mathcal{T}_n^\eta$ such that $\Delta \longrightarrow \Theta$ in $\mathcal{T}_n^\eta$.
2. For any eventuality $\varphi \mathsf{U} \psi \in \Delta$ there is a finite path of satisfiable states in $\mathcal{T}_n^\eta$ witnessing the realization of that eventuality.
3. All satisfiable states in $\mathcal{T}_0^\eta$ are still present in $\mathcal{T}_n^\eta$.

Note that the inductive claim refers simultaneously to all satisfiable $\Delta \in \mathcal{T}_n^\eta$ and all eventualities in them.

Let $n = 0$.

Take any state $\Delta \in \mathcal{T}_0^\eta$ such that $\sigma, 0 \models \Delta$ for some linear model σ. Therefore, $\sigma, 1 \models \mathsf{X}\Delta$, where $\mathsf{X}\Delta = \{\, \psi \mid \mathsf{X}\psi \in \Delta \,\}$. Then, by Lemma 8.6, at least one lean full expansion Θ of $\mathsf{X}\Delta$ is satisfied at $(\sigma, 1)$. By the construction of the initial tableau, there is a state (with label) Θ in $\mathcal{T}_0^\eta$ such that $\Delta \longrightarrow \Theta$. Therefore, the state Δ cannot be removed from $\mathcal{T}_0^\eta$ by an application of the rule StElim1$^{\mathrm{LTL}}$.

Now, let $\varphi \mathsf{U} \psi \in \Delta$. Then, there is $i \in \mathbb{N}$ such that $\sigma, i \models \psi$ and $\sigma, j \models \varphi$ for every $0 \leq j < i$. Moreover, we can choose a minimal such i, so all intermediate states in it would satisfy $\varphi \mathsf{U} \psi$, too – shown by induction on j. By iterating the argument above, we show that the trace $\sigma(0), \ldots, \sigma(i)$ can be 'simulated' by a path $\Delta = \Delta(0), \ldots, \Delta(i)$ of states in $\mathcal{T}_0^\eta$, such that $\sigma, j \models \Delta(j)$ for $0 \leq j \leq i$, thus realizing $\varphi \mathsf{U} \psi$ at Δ in $\mathcal{T}_0^\eta$. Indeed, we show by induction on j that each intermediate state $\Delta(j)$, for $0 \leq j < i$, contains both φ and $\varphi \mathsf{U} \psi$, using the

fact that each $\Delta(j)$ is fully expanded and $\mathsf{X}(\varphi\mathsf{U}\psi)$ is a β-component of $\varphi\mathsf{U}\psi$. To show that the last state $\Delta(i)$ can be chosen to contain ψ, we use Lemma 8.6.

Thus, the state Δ cannot be removed from $\mathcal{T}_0^\eta$ by an application of the rule StElim2$^{\mathrm{LTL}}$.

Consequently, all satisfiable states in $\mathcal{T}_0^\eta$ have remained in $\mathcal{T}_1^\eta$.

Now, assuming the claim holds for all $n < m$, take any satisfiable state $\Delta \in \mathcal{T}_m^\eta$. By the argument above (for $n = 0$) there is a satisfiable state Θ in $\mathcal{T}_0^\eta$ such that $\Delta \longrightarrow \Theta$ in $\mathcal{T}_0^\eta$. By the inductive hypothesis, Θ has remained intact in $\mathcal{T}_m^\eta$. Therefore, Δ cannot be removed from $\mathcal{T}_m^\eta$ by an application of the rule StElim1$^{\mathrm{LTL}}$.

Likewise, for any $\varphi\mathsf{U}\psi \in \Delta$, the finite path of satisfiable states in $\mathcal{T}_0^\eta$ witnessing the realization of that eventuality, produced by the argument for $n = 0$, will have remained intact in $\mathcal{T}_m^\eta$, and hence $\varphi\mathsf{U}\psi$ is realized at Δ in $\mathcal{T}_m^\eta$. Therefore, Δ cannot be removed from $\mathcal{T}_m^\eta$ by an application of the rule StElim2$^{\mathrm{LTL}}$, either.

That completes the induction, and the proof. QED

Theorem 8.8. [**Soundness of the tableau for** LTL] If $\eta \in$ LTL is satisfiable in then $\mathcal{T}^\eta$ is open.

Proof: Follows immediately from Lemmas 8.6 and 8.7. QED

8.2.8 Completeness of the tableaux for LTL

To prove completeness of the tableaux for LTL it suffices to show how from the open final tableau $\mathcal{T}^\eta$ one can construct a Hintikka trace satisfying η, and then refer to Theorem 8.5. We can extract such a Hintikka trace from $\mathcal{T}^\eta$, by starting with any state Δ containing η and building up a path of successor states through $\mathcal{T}^\eta$, while ensuring that all eventualities appearing along that path eventually get realized on that path. Note, that not every path in $\mathcal{T}^\eta$ satisfies that requirement. Indeed, we have proved that every eventuality in a state Δ is realized along *some* path starting from Δ, but different eventualities may be realized along *different* paths.

Exercise 8.5. Give an example of η, where not every path in $\mathcal{T}^\eta$ is a Hintikka trace.

So, we have to guide the process of building a Hintikka trace. The idea is to build the Hintikka trace $\mathcal{H}(\eta)$ in a sequence of steps, each producing a finite *partial Hintikka trace* extending the previous one by appending a finite path realizing one *pending eventuality*, while possibly adding more pending eventualities at the end. Thus, an infinite path is produced in the limit, as a union of all partial Hintikka traces, in which there are no unrealized eventualities – and that is the required Hintikka trace.

The reason we can apply such a piecewise approach to the realization of the eventualities is that if an eventuality belongs to a state in the final tableau $\mathcal{T}^\eta$ and is not realized within some finite path in $\mathcal{T}^\eta$ starting at that state, then it gets propagated down the path, and thus its realization can be deferred. More precisely:

Proposition 8.9. If $\Delta \in \mathcal{T}^\eta$ and $\xi = \varphi\mathsf{U}\psi \in \Delta$ is not realized on a given path $\Delta = \Delta_0, \ldots, \Delta_m$ in $\mathcal{T}^\eta$, then every state on that path contains φ, $\varphi\mathsf{U}\psi$, and $\mathsf{X}(\varphi\mathsf{U}\psi)$.

Proof: Induction on i. Exercise. QED

The proposition above allows the realization of the eventuality ξ at Δ to be deferred indefinitely, i.e., throughout any finite path, and then accomplished by appending to its last state Δ' another finite path realizing ξ at Δ', and hence at Δ.

To formally organize the procedure of building the Hintikka trace we do the following:

1. Fix a list of all states in $\mathcal{T}^\eta$: $\Delta_0, \ldots, \Delta_{n-1}$, and a list of all eventualities occurring in $\mathcal{T}^\eta$: $\xi_0, \ldots, \xi_{m-1}$.

2. For every state Δ_i and an eventuality $\xi_j \in \Delta_i$ select and fix a finite path in $\mathcal{T}^\eta$, denoted $\pi(\Delta_i, \xi_j)$, witnessing the realization of ξ_j at Δ_i.

We are now ready to prove the completeness theorem.

Theorem 8.10. [**Completeness of the tableau for** LTL] For any formula $\eta \in$ LTL, if the final tableau $\mathcal{T}^\eta$ is open, then the formula is satisfiable.

Proof: We construct a sequence of partial Hintikka traces $\mathcal{H}_0(\eta), \mathcal{H}_1(\eta), \ldots$ as follows:

We start with any state $\Delta \in \mathcal{T}^\eta$ containing η. This state constitutes the singleton $\mathcal{H}_0(\eta)$, and we associate with it a list of *pending eventualities* Event_0 - these are all eventualities in Δ that are not immediately realized at Δ.

Thereafter the construction continues inductively as follows. Let $\mathcal{H}_n(\eta)$ be constructed, with a last state Δ_{i_n} and let Event_n be its list of pending eventualities, where ξ_{j_n} is the first item, if any. We then take a copy of the path $\pi(\Delta_{i_n}, \xi_{j_n})$, witnessing the realization of ξ_{j_n} at Δ_{i_n}, and append it to $\mathcal{H}_n(\eta)$ by identifying its first state with the last state of $\mathcal{H}_n(\eta)$; if Event_n is empty, we simply extend $\mathcal{H}_n(\eta)$ with any successor state of Δ_{i_n}, to keep the path going. Thus, $\mathcal{H}_{n+1}(\eta)$ is produced. The new list of pending eventualities Event_{n+1} is produced by removing ξ_{j_n} from the head of the list Event_n and appending to it a list of all eventualities in Δ_{i_n} that do not appear in Event_n and are not immediately realized at Δ_{i_n}. This completes the construction of the chain $\mathcal{H}_0(\eta), \mathcal{H}_1(\eta), \ldots$, and consequently, of $\mathcal{H}(\eta)$ as the union of that chain.

It remains to show that $\mathcal{H}(\eta)$ defined as above is indeed a Hintikka trace satisfying η. This verification is now straightforward from the construction, and is left as an exercise. QED

Corollary 8.11. The logic LTL has the small model property. (Actually, small satisfiability witness property.)

Proof:

The construction of $\mathcal{H}(\eta)$ above can be made finite by propagating the list of pending eventualities to every next state in the partial Hintikka traces produced as above and looping back the first state for which a state with the same label and same set of pending eventualities (even possibly listed in different order) appears earlier in the current trace.

A careful inspection of that construction shows that we can thus produce an ultimately periodic Hintikka trace with a number of states bounded exponentially by the size of the formula. We leave the details as an exercise. QED

8.2.9 On the complexity and optimality of the tableau for LTL

A closer look at the tableau procedure for LTL, as described above, shows that it takes exponential time in the size of the input formula η. On the other hand, as we know from [SC85], the problem of testing satisfiability of an LTL formula is PSPACE-complete, so it is clear that the tableau, as described above, uses more space resources than necessary. It can be optimized to work in (non-deterministic) PSPACE by first guessing a satisfying 'Hintikka trace' already in the pretableau, i.e., organizing a depth-first expansion of the pretableau, and then testing for its suitability by a more judicial memory use, namely only keeping in the memory the current position and the set of eventualities currently waiting to be satisfied.

8.2.10 Model-checking tableaux for LTL

In Section 8.1.5 we discussed how the generic tableau procedure for testing satisfiability can be modified to perform local model-checking. Here we will illustrate the idea by developing a model-checking tableau-based method for LTL.

Recall the general, *universal* local model-checking problem for LTL: given a finite rooted ITS $(\mathcal{M}, r)$, where $\mathcal{M} = \langle S, R, L\rangle$, and a formula $\eta \in$ LTL, decide whether $\mathcal{M}, r \models_\forall \eta$, meaning that every computation in $\mathcal{M}$ starting from r satisfies η. This problem is trivially reduced to the *existential* local model-checking problem: decide whether $\mathcal{M}, r \models_\exists \neg\eta$, meaning that *some* computation in $\mathcal{M}$ starting from r satisfies $\neg\eta$. The latter problem is closer in spirit to the satisfiability testing problem, so we will construct a model-checking tableau for it.

Given a finite rooted ITS $(\mathcal{M}, r)$, where $\mathcal{M} = \langle S, R, L\rangle$, and a formula $\eta \in$ LTL, we denote for any $s \in S$:

$$L[cl(\eta), s] := (L(s) \cup \{\, \neg p \mid p \in \text{PROP} \setminus L(s) \,\}) \cap cl(\eta).$$

Following the idea in Section 8.1.5, we first construct a modified pretableau, in which the states and prestates are now labeled by pairs (s, Θ) where $s \in S$ and $\Theta \subseteq cl(\eta)$, where in the case of states Θ is fully expanded.

The root prestate is $(r, \{\eta\} \cup L[cl(\eta), r])$.

The pretableau construction phase is similar to the one in the tableaux for satisfiability, but employs modified rules $\text{PREXP}^{\text{LTL}}$ and NEXT^{LTL}.

The rule $\text{PREXP}^{\text{LTL}}$ is modified as follows:

> $\text{PREXP}^{\text{LTL}}(\mathcal{M})$: given a prestate (s, Γ) to which the rule has not yet been applied, do the following:
>
> 1. Compute the family $\text{FE}(\Gamma)$ of all lean full expansions of Γ and add a new state in the pretableau with a label (s, Δ) for each $\Delta \in \text{FE}(\Gamma)$.
> 2. For each newly introduced state (s, Δ), create an edge $(s, \Gamma) \Longrightarrow (s, \Delta)$.
> 3. If, however, the pretableau already contains a state with label (s, Δ), then do not create a new state with that label but extend an arrow $(s, \Gamma) \Longrightarrow (s, \Delta)$ to that state.

We denote the set $\{\, (s, \Delta) \mid (s, \Gamma) \Longrightarrow (s, \Delta) \,\}$ of *offspring states* of (s, Γ) by $\mathsf{states}(s, \Gamma)$.

The rule NEXT^{LTL} is now modified as follows:

Rule $\textsc{Next}^{\text{LTL}}(\mathcal{M})$: Given a state with label (s, Δ) to which the rule has not yet been applied, do the following:

if there are no R-successors of s in $\mathcal{M}$, remove the state (s, Δ) from the pretableau;

else:

1. For every $t \in S$ such that sRt, add a successor prestate of (s, Δ) with label
$$(t, \{\, \psi \mid \mathsf{X}\psi \in \Delta \,\} \cup L[cl(\eta), t]).$$
2. For each so introduced prestate (t, Γ), create an edge $(s, \Delta) \longrightarrow (t, \Gamma)$.
3. If, however, the pretableau already contains a prestate with label (t, Γ) then do not create a new prestate with that label, but extend an arrow $(s, \Delta) \longrightarrow (t, \Gamma)$ to that prestate.

Recall, that the test for patent consistency of prestates is done at the beginning of the procedure computing their lean full expansions.

Because of the finiteness of $\mathcal{M}$ the pretableau construction phase is guaranteed to terminate, producing the pretableau $\mathcal{P}^{\mathcal{M},r,\eta}$.

The prestate and state elimination phases are essentially the same as in the tableau for satisfiability.

When the final tableau $\mathcal{T}^{\mathcal{M},r,\eta}$ is obtained, $\mathcal{M}, r \models_\exists \eta$ is declared true iff there is a state (r, Δ) in $\mathcal{T}^{\mathcal{M},r,\eta}$ such that $\eta \in \Delta$.

The correctness of the procedure follows from the following theorem.

Theorem 8.12. [**Correctness of the model-checking tableau for** LTL] For any formula $\eta \in$ LTL and a finite rooted ITS $(\mathcal{M}, r)$, the following hold:

I. If a prestate (s, Γ) in the pretableau $\mathcal{P}^{\mathcal{M},r,\eta}$ is such that $\mathcal{M}, s \models_\exists \Gamma$, then it has at least one offspring state (s, Δ) in the pretableau $\mathcal{P}^{\mathcal{M},r,\eta}$ such that $\mathcal{M}, s \models_\exists \Delta$.

II. A state (s, Δ) from the initial tableau $\mathcal{T}_0^{\mathcal{M},r,\eta}$ is eliminated during the state elimination phase only if $\mathcal{M}, s \not\models_\exists \Delta$.

III. For any state (s, Δ) from the initial tableau $\mathcal{T}_0^{\mathcal{M},r,\eta}$, $\mathcal{M}, s \models_\exists \Delta$ holds iff (s, Δ) is in the final tableau $\mathcal{T}^{\mathcal{M},r,\eta}$, too.

Proof:

I. Follows from Proposition 8.1.

II. A slight modification of the proof of Lemma 8.7 as follows.

Prove by induction on n that for every $n \in \mathbb{N}$, for every $\Delta \in \mathcal{T}_n^\eta$, and for every $s \in \mathcal{M}$, if $\mathcal{M}, s \models_\exists \Delta$ for some $s \in \mathcal{M}$ then:

(a) There is a state $\Theta \in \mathcal{T}_n^\eta$ such that $\Delta \longrightarrow \Theta$ in $\mathcal{T}_n^\eta$ and $\mathcal{M}, t \models_\exists \Theta$ for some R-successor t of s.

(b) For any eventuality $\varphi \mathsf{U} \psi \in \Delta$ there is a finite path $\Delta = \Delta_0, \ldots, \Delta_m$ in $\mathcal{T}_n^\eta$ witnessing the realization of that eventuality and such that there is an R-path $s = s_0, \ldots, s_m$ in $\mathcal{M}$ such that $\mathcal{M}, s_i \models_\exists \Delta_i$ for $0 \leq i \leq m$.

(c) All states $\Phi \in \mathcal{T}_0^\eta$ such that $\mathcal{M}, t \models_\exists \Phi$ for some $t \in \mathcal{M}$ are still present in $\mathcal{T}_n^\eta$.

We leave the details to the reader.

III. One direction follows from II. For the other, suppose (s, Δ) is in the final tableau $\mathcal{T}^{\mathcal{M},r,\eta}$. By a modification of the proof of Theorem 8.10 (the completeness for the satisfiability tableau) and using the arguments from the proof of claim II, we can construct a path π in $\mathcal{M}$ starting from s and such that $\pi \models \Delta$.

Again, we leave the details to the reader.

QED

Exercise 8.6. Complete the proofs of claims II and III above.

9 Lecture 8: Testing satisfiability of branching time temporal formulae using tableau-based method

We will now adapt the generic tableau constructions presented in the previous lecture for the logic TLR, and then will extend it to CTL. The tableau for TLR subsumes one for TL, which we will indicate in passing. Unlike LTL, the tableau for TLR has to test satisfiability in an arbitrary ITS, so this will create some technical overhead compared to the construction for LTL. On the other hand, again unlike the LTL case, the construction of a satisfying Hintikka structure from an open tableau for a TLR-formula will turn out straightforward. The tableaux for LTL and TLRsupplement each other in illustrating all important issues that will apply to the tableau for the logic CTL in the next section.

9.1 Tableau for TLR

For simplicity of the exposition, we will consider TLR over a language with one transition relation; the extension to the general case of a language with many transition relations is straightforward.

9.1.1 Preliminaries

In this section, in order to illustrate the role in the construction of each logical connective, we will assume that the language of TLR contains $\top, \neg, \wedge, \vee, \Rightarrow$, EX, AX, EF, and AG as primitives connectives. Some of these can be eliminated as definable in terms of the others and the tableau construction can be seamlessly restricted. However, we will offer independent treatment for each of them, so the readers can make their favorite selection.

The *nexttime formulae* in TLR are all formulae beginning with EX or AX and their negations. The *primitive formulae* in TLR are the literals and the nexttime formulae.

The α-formulae and β-formulae in TLR and their components are given in the tables below:

α	α_1	α_2
$\neg\neg\varphi$	φ	φ
$\neg\,\mathsf{AX}\varphi$	$\mathsf{EX}\neg\varphi$	$\mathsf{EX}\neg\varphi$
$\neg\,\mathsf{EX}\varphi$	$\mathsf{AX}\neg\varphi$	$\mathsf{AX}\neg\varphi$
$\varphi \wedge \psi$	φ	ψ
$\neg(\varphi \vee \psi)$	$\neg\varphi$	$\neg\psi$
$\neg(\varphi \Rightarrow \psi)$	φ	$\neg\psi$
$\mathsf{AG}\varphi$	φ	$\mathsf{AX}\,\mathsf{AG}\varphi$
$\neg\,\mathsf{EF}\varphi$	$\neg\varphi$	$\mathsf{AX}\neg\,\mathsf{EF}\varphi$

β	β_1	β_2
$\varphi \vee \psi$	φ	ψ
$\varphi \Rightarrow \psi$	$\neg\varphi$	ψ
$\neg(\varphi \wedge \psi)$	$\neg\varphi$	$\neg\psi$
$\mathsf{EF}\varphi$	φ	$\mathsf{EX}\,\mathsf{EF}\varphi$
$\neg\,\mathsf{AG}\varphi$	$\neg\varphi$	$\mathsf{EX}\neg\,\mathsf{AG}\varphi$

Lemma 9.1.

I. For every α-formula $\varphi \in$ TLR: $\varphi \equiv \alpha_1(\varphi) \wedge \alpha_2(\varphi)$.

II. For every β-formula $\varphi \in$ TLR: $\varphi \equiv \beta_1(\varphi) \vee \beta_2(\varphi)$.

Proof: Exercise. QED

The notions of closure of a formula, fully expanded sets, and the procedure FULLEXPANSION computing the family of lean full expansions of a set of TLR-formulae, are defined as in Section 8.1.

Exercise 9.1. Show that $cl(\varphi)$ is finite for every $\varphi \in$ TLR.

Exercise 9.2. Give explicitly the closure conditions for fully expanded sets of TLR-formulae and the set-replacement operations of the procedure FULLEXPANSION for the TLR-formulae $\neg\,\mathsf{AX}\varphi$, $\mathsf{AG}\varphi$, $\mathsf{EF}\varphi$, $\neg\,\mathsf{AG}\varphi$, $\neg\,\mathsf{EF}\varphi$.

As in the case of LTL, FE(Γ) may also contain some non-minimal full expansions of Γ. For instance, if $\Gamma = \{\mathsf{EF}p, \mathsf{EX}\,\mathsf{EF}p\}$ then
FE(Γ) $= \{\{\mathsf{EF}p, \mathsf{EX}\,\mathsf{EF}p\}, \{p, \mathsf{EF}p, \mathsf{EX}\,\mathsf{EF}p\}\}$. We want to keep all full expansions obtained by the procedure FE, for reasons (same as in the case of LTL) explained later.

9.1.2 Hintikka structures

Definition 9.1. [**Hintikka structure for TLR**] Given a set of formulae $\Phi \in$ TLR, a *Hintikka structure* (HS) for Φ is a tuple $\mathcal{H} = (S, R, H)$ such that (S, R) is a TS, and $H : S \to \mathcal{P}(\Phi)$ is a labeling of the states in S with sets of formulae from Φ satisfying the following conditions for every $s \in S$:

H1 $H(s)$ is fully expanded;

H2 if $\mathsf{EX}\varphi \in H(s)$, then $\varphi \in H(s')$ for some s' such that sRs';

H3 if $\mathsf{AX}\varphi \in H(s)$, then $\varphi \in H(s')$ for every s' such that sRs';

H4 if $\mathsf{EF}\varphi \in H(s)$, then $\varphi \in H(s')$ for some s' such that sR^*s', i.e., there is an R-path connecting s with s';

H5 if $\neg\,\mathsf{AG}\varphi \in H(s)$, then $\neg\varphi \in H(s')$ for some s' such that sR^*s'.

∇

Proposition 9.2. In every Hintikka structure (S, R, H):

I. If $\mathsf{AG}\varphi \in H(s)$, then $\varphi \in H(s')$ for every s' such that sR^*s'.

II. If $\neg\,\mathsf{EF}\varphi \in H(s)$, then $\neg\varphi \in H(s')$ for every s' such that sR^*s'.

Proof: Exercise. QED

Definition 9.2. A formula η is *satisfiable* in a HS $\mathcal{H}$ for some set of formulae Φ containing η, with a labeling function H if $\eta \in H(s)$ for some state s in $\mathcal{H}$; a set of formulae $\Theta \subseteq \Phi$ is *satisfiable* in $\mathcal{H}$ if $\Theta \subseteq H(s)$ for some state s in $\mathcal{H}$. ∇

Lemma 9.3. For any set of formulae $\Phi \in$ TLR and for every ITS $\mathcal{M} = (S, R, L)$ the structure $\mathcal{H}(\mathcal{M}) = (S, R, H)$, where $H(s) = \{\varphi \in \Phi \mid \mathcal{M}, s \models \varphi\}$ for every $s \in S$, is a Hintikka structure for Φ.

Proof: Straightforward verification of H1-H5. Exercise. QED

Given a formula η for which we want to find a model, we will be interested in Hintikka structures for the set $cl(\eta)$.

Theorem 9.4. A formula η of TLR is satisfiable iff it is satisfiable in a Hintikka structure for $cl(\eta)$.

Proof: One direction is immediate by Lemma 9.3 for $\Phi = cl(\eta)$. For the converse, suppose $\eta \in H(s_0)$ for some state s_0 in some Hintikka structure $\mathcal{H} = (S, R, H)$. We define the following state description L in S: $L(s) := \text{PROP} \cap H(s)$. Let $\mathcal{M} = (S, R, L)$.

We now show by a routine induction on $\varphi \in$ TLR that for every $s \in S$:

(i) if $\varphi \in H(s)$ then $\mathcal{M}, s \models \varphi$;

(ii) if $\neg\varphi \in H(s)$ then $\mathcal{M}, s \models \neg\varphi$;

When $\varphi \in$ PROP the proof is immediate by definition of L.

Let $\varphi = \neg\psi$. Then clause (i) is part of the inductive hypothesis for ψ, while clause (ii) follows again from the inductive hypothesis for ψ and the fact that every $H(s)$ is fully expanded, so if $\neg\neg\psi \in H(s)$ then $\psi \in H(s)$.

The rest of the Boolean cases, where $\varphi = \varphi_1 \wedge \varphi_2$, $\varphi = \varphi_1 \vee \varphi_2$, and $\varphi = \varphi_1 \Rightarrow \varphi_2$ are straightforward, using the inductive hypothesis and the fact that $H(s)$ is fully expanded.

The cases of $\varphi = \mathsf{EX}\psi$, $\varphi = \mathsf{AX}\psi$, $\varphi = \mathsf{EF}\psi$, and $\varphi = \mathsf{AG}\psi$ follow from the inductive hypothesis and the conditions H1, and H2, H3, H4, and H5 respectively, together with Proposition 9.2. QED

9.1.3 Pretableau construction phase

Given an input formula $\eta \in$ TLR we are going to build a tableau for testing the satisfiability of η, following the construction outlined in Section 8.1.

The pretableau construction phase consists in alternating applications of two rules:

$\mathbf{PrExp^{TLR}}$: producing all *offspring states* of a given prestate;

$\mathbf{Next^{TLR}}$: producing all *successor prestates* of a given state.

Hereafter we will freely identify prestates or states with their labels, whenever that will not lead to confusion.

Rule PrExp$^{\mathrm{TLR}}$: Given a prestate Γ to which the rule has not yet been applied, do the following:

1. Compute the family $\mathrm{FE}(\Gamma) = \{\Delta_1, \ldots \Delta_n\}$ of all lean full expansions of Γ and add these as (labels of) new states in the pretableau, called *offspring states of* Γ.
2. For each newly introduced state Δ, create an edge $\Gamma \Longrightarrow \Delta$.
3. If, however, the pretableau already contains a state with label Δ then do not create a new state with that label, but extend an arrow $\Gamma \Longrightarrow \Delta$ to that state.

We denote the (finite) set $\{\, \Delta \mid \Gamma \Longrightarrow \Delta \,\}$ of offspring states of Γ by $\mathsf{states}(\Gamma)$.

Computing successor prestates of a state

Rule Next:$^{\mathrm{TLR}}$ Given a state with label Δ, to which the rule has not yet been applied, do the following for every $\mathsf{EX}\varphi \in \Delta$:

1. Add a successor prestate of Δ with label $\{\varphi\} \cup \{\, \psi \mid \mathsf{AX}\psi \in \Delta \,\}$.
2. If the label of Δ does not contain any formula of the type $\mathsf{EX}\varphi$ then add one successor prestate of Δ with label $\{\, \psi \mid \mathsf{AX}\psi \in \Delta \,\}$.
3. For each newly introduced prestate Γ, create an edge $\Delta \stackrel{\mathsf{EX}\varphi}{\longrightarrow} \Gamma$.
4. If, however, the pretableau already contains a prestate with label Γ then do not create a new prestate with that label, but extend an arrow $\Delta \stackrel{\mathsf{EX}\varphi}{\longrightarrow} \Gamma$ to that prestate.

Building the pretableau The construction phase of building a pretableau for η begins with creating a single prestate $\{\eta\}$, followed by alternating applications of the rules PrExp$^{\mathrm{TLR}}$ and Next$^{\mathrm{TLR}}$ respectively to the prestates and the states created at a previous stage of the construction.

The construction phase ends when none of these rules can add any new states or prestates to the current graph. The resulting graph is the pretableau $\mathcal{P}^\eta$.

Note that there are two types of branching in the pretableau: *tableau branching*, from a prestate to its offspring states, indicated by $\Longrightarrow$, and *structural branching*, from a state to its successor prestates, indicated by $\stackrel{\theta}{\longrightarrow}$ where θ is a EX-formula. The tableau branching is branching of the search tree, thus it is *disjunctive* – only one offspring state of every prestate is eventually needed to build a satisfying structure, while the structural branching is *conjunctive*, as it represents branching in the satisfying structure to be built, so all successors prestates of every state are needed in the construction.

9.1.4 Prestate elimination phase

In this phase, all prestates are removed from $\mathcal{P}^\eta$, together with their incoming and outgoing arrows, by applying the following *prestate elimination rule*:

Rule PRE:$^{\mathrm{TLR}}$ For every prestate Γ in $\mathcal{P}^\eta$, do the following:

1. Remove Γ from $\mathcal{P}^\eta$;
2. If there is a state Δ in $\mathcal{P}^\eta$ with $\Delta \xrightarrow{\theta} \Gamma$, then for every state $\Delta' \in \mathsf{states}(\Gamma)$, create an edge $\Delta \xrightarrow{\theta} \Delta'$.

The resulting graph is called the *initial tableau* for η, denoted $\mathcal{T}_0^\eta$. The offspring states of the input prestate $\{\eta\}$ are called *input states* of $\mathcal{T}_0^\eta$.

9.1.5 State elimination phase

The purpose of the state elimination phase is to remove from the tableau all 'bad' states, the labels of which are not satisfiable in any Hintikka structure, and hence in any ITS.

One possible reason for existence of such a 'bad' state (the only one in the case of TL) is that it may lack a successor state needed to satisfy some EX-formula in its label. Such states can arise because some prestates may have no full expansions, and hence may yield no offspring states. Once such a bad state is removed, their predecessor states may be left without a necessary successor, so they must be removed, too, etc., until stabilization.

Formally, the elimination of states lacking all necessary successors is done by applying the following rule.

> Rule StElim1$^{\mathrm{TLR}}$: If a state Δ contains a formula $\mathsf{EX}\psi$ and there are no outgoing arcs marked with $\mathsf{EX}\psi$ from Δ to successors states in the current tableau, then remove that state (together with all incoming and outgoing arcs) from the tableau.

Another reason for a state to be 'bad', i.e., for its label not to be satisfiable in any Hintikka structure, is when that label contains an eventuality formula which is not 'realized' in any reachable (i.e., future) state. Such cases are handled by an additional state elimination rule. In order to formulate it we have to formalize the concept of *eventuality realization*.

Recall, that the eventualities in TLR are the formulae of the type $\mathsf{EF}\varphi$ and $\neg\,\mathsf{AG}\varphi$.

Hereafter, by a path in a TLR tableau we mean a path of states with respect to the state successor relation, i.e., $\Delta_0, \Delta_1, \ldots, \Delta_m$ is a path in the current tableau if for every $0 \leq i < m$ there exists $\theta_i = \mathsf{EX}\psi_i$ such that $\Delta_i \xrightarrow{\theta_i} \Delta_{i+1}$.

Definition 9.3. [**Realization of eventualities in TLR**] An eventuality $\xi = \mathsf{EF}\varphi$ (respectively, $\xi = \neg\,\mathsf{AG}\varphi$) is *realized* at the state Δ in $\mathcal{T}_n^\eta$ if there exists a finite path $\Delta = \Delta_0, \Delta_1, \ldots, \Delta_m$ in $\mathcal{T}_n^\eta$ such that $\xi, \varphi \in \Delta_m$ (respectively, $\xi, \neg\varphi \in \Delta_m$ in the case of $\xi = \neg\,\mathsf{AG}\varphi$) and $\xi \in \Delta_i$ for every $i = 0, \ldots, m-1$.

We then say that ξ is realized on the path $\Delta_0, \Delta_1, \ldots, \Delta_m$ and call any such path a *witness of the realization of the eventuality* ξ *in* Δ.

If $\varphi \in \Delta$ in the case of $\xi = \mathsf{EF}\varphi$, or respectively, $\neg\varphi \in \Delta$ in the case of $\xi = \neg\,\mathsf{AG}\varphi$, we say that ξ is *immediately realized* at the state Δ (by the singleton path Δ). ∇

Checking for realization of a TLR-eventuality is a simple graph reachability problem.For any given eventuality it can be done simultaneously for all states in the current tableau by a standard marking procedure in time linear in the number of states in $\mathcal{T}_n^\eta$.

Now, we can formulate the second state elimination rule.

Rule StElim2$^{\mathrm{TLR}}$: If a state $\Delta \in \mathcal{T}_n^\eta$ contains in its label an eventuality that is not realized at Δ in $\mathcal{T}_n^\eta$, then remove that state from the tableau.

Using the state elimination rules StElim1$^{\mathrm{TLR}}$ and StElim2$^{\mathrm{TLR}}$, the state elimination phase is carried out in a sequence of stages, starting at stage 0 with the initial tableau $\mathcal{T}_0^\eta$, and eliminating at every stage n at most one state for the current tableau $\mathcal{T}_n^\eta$, by applying one of the state elimination rules, and producing the new current tableau $\mathcal{T}_{n+1}^\eta$. The rules StElim1$^{\mathrm{TLR}}$ and StElim2$^{\mathrm{TLR}}$ are applied alternatively, in any order, which may be determined by a specific strategy. This procedure continues until at some stage no further elimination of states by an application of any of the rules is possible.

9.1.6 The final tableau

When the state elimination phase reaches a stabilization stage the current tableau at that stage is the *final tableau for* η, denoted by $\mathcal{T}^\eta$, with a set of states denoted by S^η.

Definition 9.4. The final tableau $\mathcal{T}^\eta$ is *open* if $\eta \in \Delta$ for some $\Delta \in S^\eta$; otherwise, $\mathcal{T}^\eta$ is *closed*. ∇

The tableau procedure returns "not satisfiable" if the final tableau is closed; otherwise, it returns "satisfiable" and, moreover, provides sufficient information for producing a finite Hintikka structure satisfying η.

9.1.7 Soundness of the tableaux

Recall, that soundness of the tableau procedure means that if the input formula η is satisfiable, then the tableau $\mathcal{T}^\eta$ is open. We will establish the soundness of the tableau for TLR by proving that every state elimination rule is 'sound' in a sense that it never eliminates a state with a satisfiable label. The soundness of the overall procedure is then an immediate consequence.

Lemma 9.5. Let Γ be a prestate of $\mathcal{P}^\eta$ such that $\mathcal{M}, s \models \Gamma$ for some rooted ITS $(\mathcal{M}, s)$. Then:

I. $\mathcal{M}, s \models \Delta$ for at least one $\Delta \in \mathsf{states}(\Gamma)$.

II. Moreover, if $\mathsf{EF}\varphi \in \Gamma$ and $\mathcal{M}, s \models \varphi$, then Δ can be chosen to contain φ.

Respectively, if $\neg\,\mathsf{AG}\varphi \in \Gamma$ and $\mathcal{M}, s \models \neg\varphi$, then Δ can be chosen to contain $\neg\varphi$.

Proof: A minor modification of the proof of Lemma 8.6. QED

Note that the so constructed full expansion Δ need not be minimal – indeed, that happens in the example $\Gamma = \{\mathsf{EF}p, \mathsf{EX}\,\mathsf{EF}p\}$ given earlier. For that reason we want to keep all lean full expansions in $\mathrm{FE}(\Gamma)$, not only the minimal ones.

Lemma 9.6. No satisfiable state $\Delta \in \mathcal{T}_0^\eta$ is removed by any application of Rule StElim1$^{\mathrm{TLR}}$ or Rule StElim2$^{\mathrm{TLR}}$ during the state elimination phase.

Proof: It suffices to show, by induction on $n \in \mathbb{N}$, that no satisfiable state $\Delta \in \mathcal{T}_n^\eta$ is removed by an application of Rule StElim1$^{\mathrm{TLR}}$ or Rule StElim2$^{\mathrm{TLR}}$ to $\mathcal{T}_n^\eta$. For that purpose we will prove a somewhat stronger inductive claim, viz., that for every $n \in \mathbb{N}$, if $\Delta \in \mathcal{T}_n^\eta$ is satisfiable then:

1. For any $\mathsf{EX}\varphi \in \Delta$ there is a satisfiable state $\Theta \in \mathcal{T}_n^\eta$ such that $\Delta \xrightarrow{\mathsf{EX}\varphi} \Theta$ in $\mathcal{T}_n^\eta$.

2. For any eventuality $\mathsf{EF}\varphi \in \Delta$ (respectively, $\neg\,\mathsf{AG}\varphi \in \Delta$) there is a finite path of satisfiable states in $\mathcal{T}_n^\eta$ realizing that eventuality.

3. All satisfiable states in $\mathcal{T}_0^\eta$ are still present in $\mathcal{T}_n^\eta$.

Note that the inductive claim refers simultaneously to all satisfiable $\Delta \in \mathcal{T}_n^\eta$, all EX-formulae $\mathsf{EX}\varphi \in \Delta$, and all eventualities $\mathsf{EF}\varphi \in \Delta$ and $\neg\,\mathsf{AG}\varphi \in \Delta$.

Let $n = 0$. Take any satisfiable $\Delta \in \mathcal{T}_0^\eta$ and $\mathsf{EX}\varphi \in \Delta$. Then, $\mathcal{M}, s \models \Delta$ for some rooted ITS $(\mathcal{M}, s)$.

Recall, that all states $\Delta' \in \mathcal{T}_0^\eta$ such that $\Delta \xrightarrow{\mathsf{EX}\varphi} \Delta'$ in $\mathcal{T}_0^\eta$ are obtained as lean full expansions of the prestate $\Gamma = \{\varphi\} \cup \{\, \psi \mid \mathsf{AX}\psi \in \Delta \,\}$.

Since $\mathsf{EX}\varphi \in \Delta$, we have that $\mathcal{M}, s \models \mathsf{EX}\varphi$, hence there is an R-successor t of s in $\mathcal{M}$ such that $\mathcal{M}, t \models \varphi$.

By the truth definition for AX-formulae, it follows that $\mathcal{M}, t \models \Gamma$ because $\{\, \psi \mid \mathsf{AX}\psi \in \Delta \,\}$ is satisfied at every R-successor of s in $\mathcal{M}$.

Then, by Lemma 9.5, at least one lean full expansion Θ of Γ is satisfied by $(\mathcal{M}, t)$, and, by construction of the initial tableau, there is a state (with label) Θ in $\mathcal{T}_0^\eta$ such that $\Delta \xrightarrow{\mathsf{EX}\varphi} \Theta$. Therefore, the state Δ cannot be removed from $\mathcal{T}_0^\eta$ by an application of the rule StElim1$^{\mathrm{TLR}}$.

Now, let us consider the case of an eventuality $\mathsf{EF}\varphi \in \Delta$, the other case being completely analogous. Suppose $\mathcal{M}, s \models \Delta$ for some rooted ITS $(\mathcal{M}, s)$. Then, there is a finite path of states in $\mathcal{M}$ from s to a state t satisfying φ, while all intermediate states in it satisfy $\mathsf{EF}\varphi$. By iterating the argument for the case of $\mathsf{EX}\varphi \in \Delta$, we show that that path can be simulated by a path of satisfiable states in $\mathcal{T}_n^\eta$, each containing $\mathsf{EF}\varphi$ and ending in a state containing φ, thus realizing $\mathsf{EF}\varphi$ at Δ in $\mathcal{T}_0^\eta$. To show that the last state of the path can be chosen to contain φ, we use Lemma 9.5. Therefore, the state Δ cannot be removed from $\mathcal{T}_0^\eta$ by an application of the rule StElim2$^{\mathrm{TLR}}$.

Now, assuming the claim holds for all $n < m$, take a satisfiable $\Delta \in \mathcal{T}_m^\eta$. For any $\mathsf{EX}\varphi \in \Delta$, by the argument for $n = 0$ there is a satisfiable state Θ in $\mathcal{T}_0^\eta$ such that $\Delta \xrightarrow{\mathsf{EX}\varphi} \Theta$ in $\mathcal{T}_0^\eta$, and hence, by the inductive hypothesis, Θ has remained intact in $\mathcal{T}_m^\eta$. Therefore, Δ cannot be removed from $\mathcal{T}_m^\eta$ by an application of the rule StElim1$^{\mathrm{TLR}}$.

Likewise, for any $\mathsf{EF}\varphi \in \Delta$, by the argument for $n = 0$ there is a finite path of satisfiable states in $\mathcal{T}_0^\eta$ leading to a state containing φ. By the previous case and the inductive hypothesis, all states in that path have remained intact in $\mathcal{T}_m^\eta$, and hence $\mathsf{EF}\varphi$ is realized at Δ in $\mathcal{T}_m^\eta$. Therefore, Δ cannot be removed from $\mathcal{T}_m^\eta$ by an application of the rule StElim2$^{\mathrm{TLR}}$.

QED

Theorem 9.7. [**Soundness of the tableau for TLR**] If $\eta \in$ TLR is satisfiable in then $\mathcal{T}^\eta$ is open.

Proof: Follows immediately from Lemmas 9.5 and 9.6. QED

9.1.8 Completeness of the tableaux

Theorem 9.8. [**Completeness of the tableau for TLR**] For any formula $\eta \in$ TLR, if the final tableau $\mathcal{T}^\eta$ is open, then the formula is satisfiable.

Proof: It suffices to show how from the open final tableau $\mathcal{T}^\eta$ one can construct a Hintikka structure satisfying η. That construction in the case of TLR is straightforward: the final tableau $\mathcal{T}^\eta$ itself can be taken as such a Hintikka structure, with a labelling function assigning to each state its own label.

The proof that $\mathcal{T}^\eta$ is a Hintikka structure is straightforward from the construction of the tableau and left as an exercise. QED

Exercise 9.3. Prove that the final tableau $\mathcal{T}^\eta$ is a Hintikka structure.

9.1.9 On the complexity and optimality of the tableau for TLR

A closer look at the tableau procedure for TLR, as described above, shows that it takes exponential time in the size of the input formula η. Indeed, ExpTime is shown in [Spa93], using the argument for PDL from [FL79], to be a lower bound for its complexity.

Still, this tableau procedure can be optimized, both timewise and spacewise, by a more judicial strategy of search and memory use.

1. To begin with, note that there are no more tableau branching arrows in the initial tableaux – the tableau branching has become implicit there, viz.: every state Δ now has possibly several successor arrows marked with the same EX-formula θ (to all offsprings of the respective successor prestate), while only one such successor is needed for the satisfaction of θ at Δ. Thus, the branching of the tableau at Δ is implicitly represented by all possible *successor family selections* for Δ, i.e. selections of one successor state of Δ for each EX-formula θ marking at least one outgoing arrow from Δ. Consequently, a branch of the initial tableau is a subgraph of the tableau, rooted at an input state and branching at every state into a family of successors obtained by some successor family selection applied at that state.

   If the tableau is open then at least one *open branch*, containing a state with the input formula in its label. It is sufficient to select such an open branch in order to produce a satisfying Hintikka structure, and that would reduce significantly its size.

2. Various further optimizations are possible. For instance, testing for closedness of the tableau can be done after every stage of the state elimination phase, for, if any intermediate tableau $\mathcal{T}^\eta_n$ closes, the final tableau will be closed, too.

9.1.10 Model-checking tableaux for TLR

Here we will modify the tableau procedure for testing satisfiability of formulae in TLR to perform local model-checking; global model-checking would be more efficiently done by a labeling algorithm as the one for CTL in Lecture 6.

Recall the local model-checking problem: given a finite rooted ITS $(\mathcal{M}, r)$, where $\mathcal{M} = \langle S, R, L\rangle$, and a formula $\eta \in$ TLR, decide whether $\mathcal{M}, r \models \eta$.

The construction combines the satisfiability tableau for TLR and the model-checking tableau for LTL in Section 8.2.10.

Again, given a finite rooted ITS $(\mathcal{M}, r)$, where $\mathcal{M} = \langle S, R, L\rangle$, and a formula $\eta \in$ TLR, we denote for any $s \in S$:

$$L[cl(\eta), s] := (L(s) \cup \{\, \neg p \mid\ p \in \text{PROP} \setminus L(s) \,\}) \cap cl(\eta).$$

The states and prestates of the pretableau are now labeled by pairs (s, Θ) where $s \in S$ and $\Theta \subseteq cl(\eta)$, where in the case of states Θ is fully expanded.

The root prestate is $(r, \{\eta\} \cup L[cl(\eta), r])$.

The pretableau construction phase employs modified rules $\textsc{PrExp}^{\text{TLR}}$ and $\textsc{Next}^{\text{TLR}}$.

The rule $\textsc{PrExp}^{\text{TLR}}$ is modified as follows:

$\textsc{PrExp}^{\text{TLR}}(\mathcal{M})$: given a prestate (s, Γ) to which the rule has not yet been applied, do the following:

1. Compute the family $\text{FE}(\Gamma)$ of all lean full expansions of Γ and for every $\Delta \in \text{FE}(\Gamma)$ add a new state in the pretableau with a labels (s, Δ).
2. For each newly introduced state (s, Δ), create an edge $(s, \Gamma) \Longrightarrow (s, \Delta)$.
3. If, however, the pretableau already contains a state with label (s, Δ), then do not create a new state with that label but extend an arrow $(s, \Gamma) \Longrightarrow (s, \Delta)$ to that state.

We denote the set $\{\, (s, \Delta) \mid\ (s, \Gamma) \Longrightarrow (s, \Delta) \,\}$ of offspring states of (s, Γ) by $\mathsf{states}(s, \Gamma)$.

The rule $\textsc{Next}^{\text{TLR}}$ is modified as follows:

Rule $\textsc{Next}^{\text{TLR}}(\mathcal{M})$: Given a state with label (s, Δ) to which the rule has not yet been applied, do the following:

1. Choose $\mathsf{EX}\varphi \in \Delta$ to which the rule has not yet been applied and:
   (a) For every $t \in S$ such that sRt, add a successor prestate of (s, Δ) with label
   $$(t, \{\varphi\} \cup \{\, \psi \mid\ \mathsf{AX}\psi \in \Delta \,\} \cup L[cl(\eta), t]).$$
   (b) For each so introduced prestate (t, Γ), create an edge $(s, \Delta) \xrightarrow{\mathsf{EX}\varphi} (t, \Gamma)$.
   (c) If, however, the pretableau already contains a prestate with label (t, Γ) then do not create a new prestate with that label, but extend an arrow $(s, \Delta) \xrightarrow{\mathsf{EX}\varphi} (t, \Gamma)$ to that prestate.
2. If there are no R-successors of s in $\mathcal{M}$, complete the execution of the rule by removing the state (s, Δ) from the pretableau.
3. If Δ does not contain any formula of the type $\mathsf{EX}\varphi$ then add for every $t \in S$ (if any) one successor prestate of (s, Δ) with label $(t, \{\, \psi \mid\ \mathsf{AX}\psi \in \Delta \,\} \cup L[cl(\eta), t])$. Again, if such prestate exists, do not create a new one but extend an arrow $\xrightarrow{\mathsf{EX}\varphi}$ from (s, Δ) to the existing one.

Recall, that the test for patent consistency of prestates is done at the beginning of the procedure computing their lean full expansions.

Because of the finiteness of $\mathcal{M}$ the pretableau construction phase is guaranteed to terminate, producing the pretableau $\mathcal{P}^{\mathcal{M},r,\eta}$.

The prestate and state elimination phases are essentially the same as before.

When the final tableau $\mathcal{T}^{\mathcal{M},r,\eta}$ is obtained, $\mathcal{M}, r \models \eta$ is declared true iff there is a state (r, Δ) in $\mathcal{T}^{\mathcal{M},r,\eta}$ such that $\eta \in \Delta$.

The correctness of the procedure follows from the following theorem.

Theorem 9.9. [**Correctness of the model-checking tableau for TLR**] For any formula $\eta \in$ TLR and finite rooted ITS $(\mathcal{M}, r)$, the following hold:

I. If a prestate (s, Γ) in the pretableau $\mathcal{P}^{\mathcal{M},r,\eta}$ is such that $\mathcal{M}, s \models \Gamma$, then it has at least one offspring state (s, Δ) in the pretableau $\mathcal{P}^{\mathcal{M},r,\eta}$ such that $\mathcal{M}, s \models \Delta$.

II. A state (s, Δ) from the initial tableau $\mathcal{T}_0^{\mathcal{M},r,\eta}$ is eliminated during the state elimination phase only if $\mathcal{M}, s \not\models \varphi$ for some $\varphi \in \Delta$.

III. For any state (s, Δ) from the initial tableau $\mathcal{T}_0^{\mathcal{M},r,\eta}$, $\mathcal{M}, s \models \Delta$ holds iff (s, Δ) is in the final tableau $\mathcal{T}^{\mathcal{M},r,\eta}$, too.

Proof:

I. Follows from Proposition 8.1.

II. A slight modification of the proof of Lemma 9.6. Exercise.

III. We show by structural induction on formulae $\varphi \in cl(\eta)$ that for every (s, Δ) in the final tableau $\mathcal{T}^{\mathcal{M},r,\eta}$:

(a) If $\varphi \in \Delta$ then $\mathcal{M}, s \models \varphi$.
(b) If $\neg\varphi \in \Delta$ then $\mathcal{M}, s \models \neg\varphi$.

If $\varphi \in$ PROP, then the claim holds because Δ is patently consistent and $L[cl(\eta), s] \subseteq \Delta$.

The case of $\varphi = \neg\psi$ is straightforward: both claims hold by the inductive hypothesis, because if $\neg\neg\psi \in \Delta$ then $\psi \in \Delta$.

The other Boolean cases are just as easy, using the fact that Δ is fully expanded.

Let $\varphi = \mathsf{EX}\psi$. If $\varphi \in \Delta$ then, since (s, Δ) has not been removed by Rule StElim1$^{\mathrm{TLR}}$, there is $(s', \Delta') \in \mathcal{T}^{\mathcal{M},r,\eta}$ such that $(s, \Delta) \xrightarrow{\mathsf{EX}\psi} (s', \Delta')$ for some R-successor s' of s and $\psi \in \Delta'$. Then, the inductive hypothesis, $\mathcal{M}, s' \models \psi$, hence $\mathcal{M}, s \models \varphi$. Now, if $\neg\varphi \in \Delta$ then $\mathsf{AX}\neg\psi \in \Delta$ since Δ is fully expanded. Let s' be any R-successor of s in $\mathcal{M}$. Then there is $(s', \Delta') \in \mathcal{T}^{\mathcal{M},r,\eta}$ such that $(s, \Delta) \xrightarrow{\mathsf{EX}\psi} (s', \Delta')$, hence $\neg\psi \in \Delta'$, hence $\mathcal{M}, s' \models \neg\psi$ by the inductive hypothesis. Therefore $\mathcal{M}, s \models \mathsf{AX}\neg\psi$, hence $\mathcal{M}, s \models \neg\,\mathsf{EX}\psi$.

The case $\varphi = \mathsf{AX}\psi$ is completely analogous.

Let $\varphi = \mathsf{EF}\psi$. If $\varphi \in \Delta$ then, since (s, Δ) has not been removed by Rule StElim2$^{\mathrm{TLR}}$, φ is realized at (s, Δ) in $\mathcal{T}^{\mathcal{M},r,\eta}$, i.e., there is a path $(s, \Delta) = (s_0, \Delta_0), \ldots, (s_m, \Delta_m)$

in $\mathcal{T}^{\mathcal{M},r,\eta}$, such that $\psi \in \Delta_m$. Then, by the inductive hypothesis, $\mathcal{M}, s_m \models \psi$. Since $s = s_0, \ldots, s_m$ is an R-path in $\mathcal{M}$, it follows that $\mathcal{M}, s \models \varphi$.

Now, if $\neg\varphi \in \Delta$ then $\mathsf{AG}\neg\psi \in \Delta$ since Δ is fully expanded. Then for any R-path $s = s_0, \ldots, s_m$ in $\mathcal{M}$ there is a correspondent path $(s, \Delta) = (s_0, \Delta_0), \ldots, (s_m, \Delta_m)$ in $\mathcal{T}^{\mathcal{M},r,\eta}$. By the property for AG of fully expanded set and the rule $\textsc{Next}^{\mathrm{TLR}}$ we can show that $\neg\psi \in \Delta_m$, hence, by the inductive hypothesis, $\mathcal{M}, s_m \models \neg\psi$. Thus, $\mathcal{M}, s \models \mathsf{AG}\neg\psi$, hence $\mathcal{M}, s \models \neg\,\mathsf{EF}\psi$.

The case of $\varphi = \mathsf{AG}\psi$ is completely analogous.

QED

Exercise 9.4. Complete the details of the proof of claim II.

9.2 Tableau for CTL

First constructions: [BAPM81] for UB, [EH85], [EH82] for CTL.

The tableau for TLR can be transformed to a tableau for CTL with several changes described below.

9.2.1 Preliminaries

In this section we assume that the language of CTL contains the Bollean connectives $\top, \neg, \wedge, lor, \Rightarrow$ and the temporal operators EX, AX, EG, AG, EU, AU, while we will assume EF, AF, ER, AR as definable: $\mathsf{EF}\varphi := \mathsf{E}(\top\mathsf{U}\varphi)$, $\mathsf{AF}\varphi := \mathsf{A}(\top\mathsf{U}\varphi)$; $\mathsf{E}\,\varphi\mathsf{R}\psi := \neg\,\mathsf{A}(\neg\varphi\mathsf{U}\neg\psi)$, $\mathsf{A}\,\varphi\mathsf{R}\psi := \neg\,\mathsf{E}(\neg\varphi\mathsf{U}\neg\psi)$.

The reader may wish to add these operators to the language, or to remove more of those definable in terms of the others; all that follows would be modified predictably, *mutatis mutandis*.

The nexttime formulae in CTL are $\mathsf{EX}\varphi$, $\mathsf{AX}\varphi$, and their negations.

The α- and β-formulae in CTL and their components are:

α	α_1	α_2
$\neg\neg\varphi$	φ	φ
$\neg\,\mathsf{EX}\varphi$	$\mathsf{AX}\neg\varphi$	$\mathsf{AX}\neg\varphi$
$\neg\,\mathsf{AX}\varphi$	$\mathsf{EX}\neg\varphi$	$\mathsf{EX}\neg\varphi$
$\varphi \wedge \psi$	φ	ψ
$\neg(\varphi \vee \psi)$	$\neg\varphi$	$\neg\psi$
$\neg(\varphi \Rightarrow \psi)$	φ	$\neg\psi$
$\mathsf{EG}\varphi$	φ	$\mathsf{EX}\,\mathsf{EG}\varphi$
$\mathsf{AG}\varphi$	φ	$\mathsf{AX}\,\mathsf{AG}\varphi$
$\neg\,\mathsf{E}(\varphi\mathsf{U}\psi)$	$\neg\psi$	$\neg\varphi \vee \mathsf{A}\,\mathsf{X}\neg\,\mathsf{E}(\varphi\mathsf{U}\psi)$
$\neg\,\mathsf{A}(\varphi\mathsf{U}\psi)$	$\neg\psi$	$\neg\varphi \vee \mathsf{EX}\neg\,\mathsf{A}(\varphi\mathsf{U}\psi)$

β	β_1	β_2
$\varphi \vee \psi$	φ	ψ
$\varphi \Rightarrow \psi$	$\neg\varphi$	ψ
$\neg(\varphi \wedge \psi)$	$\neg\varphi$	$\neg\psi$
$\neg\,\mathsf{EG}\varphi$	$\neg\varphi$	$\mathsf{AX}\neg\,\mathsf{EG}\varphi$
$\neg\,\mathsf{AG}\varphi$	$\neg\varphi$	$\mathsf{EX}\neg\,\mathsf{AG}\varphi$
$\mathsf{E}(\varphi\mathsf{U}\psi)$	ψ	$\varphi \wedge \mathsf{EX}\,\mathsf{E}(\varphi\mathsf{U}\psi)$
$\mathsf{A}(\varphi\mathsf{U}\psi)$	ψ	$\varphi \wedge \mathsf{A}\,\mathsf{X}\,\mathsf{A}(\varphi\mathsf{U}\psi)$

Lemma 9.10.

I. For every α-formula $\varphi \in$ CTL: $\varphi \equiv \alpha_1(\varphi) \wedge \alpha_2(\varphi)$.

II. For every β-formula $\varphi \in$ CTL: $\varphi \equiv \beta_1(\varphi) \vee \beta_2(\varphi)$.

Proof: Exercise. QED

The notions of closure of a formula, fully expanded sets, and the procedure FullExpansion computing the family of lean full expansions of a set of CTL-formulae, are defined as in Section 8.1.

Exercise 9.5. Show that $cl(\varphi)$ is finite for every $\varphi \in$ CTL.

Exercise 9.6. Give explicitly the closure conditions for fully expanded sets, and the set-replacement operations of the procedure FullExpansion, for the CTL-formulae $\mathsf{EG}\varphi$, $\mathsf{AF}\varphi$, $\mathsf{E}\,\varphi\mathsf{U}\psi$, $\mathsf{A}\,\varphi\mathsf{U}\psi$, and their negations.

9.2.2 Hintikka structures for CTL

The definition and main properties of a Hintikka structure are predictable modifications of those in Section 9.1.2.

Definition 9.5. [**Hintikka structure for** CTL] Given a set of formulae $\Phi \in$ CTL, a *Hintikka structure* (HS) for Φ is a tuple $\mathcal{H} = (S, R, H)$ such that (S, R) is a TS, and $H : S \to \mathcal{P}(\Phi)$ is a labeling of the states in S with sets of formulae from Φ satisfying the following conditions for every $s \in S$:

H1 $H(s)$ is fully expanded;

H2 If $\mathsf{EX}\varphi \in H(s)$, then $\varphi \in H(s')$ for some s' such that sRs';

H3 If $\mathsf{AX}\varphi \in H(s)$, then $\varphi \in H(s')$ for every s' such that sRs';

H4 If $\neg\,\mathsf{AG}\varphi \in H(s)$, then $\neg\varphi \in H(s')$ for some s' such that sR^*s'.

H5 If $\mathsf{E}(\varphi\mathsf{U}\psi) \in H(s)$, then $\psi \in H(s')$ for some s' such that there is an R-path $s = s_0, s_1, \ldots s_n = s'$ where $\varphi \in H(s_i)$ for all $i = 0, \ldots, n-1$.

H6 If $\neg \mathsf{EG}\varphi \in H(s)$, then for every R-path $s = s_0, s_1, \ldots$ there is some $n \in \mathbb{N}$ such that $\neg\varphi \in H(s_n)$.

H7 If $\mathsf{A}(\varphi \mathsf{U} \psi) \in H(s)$, then for every R-path $s = s_0, s_1, \ldots$ there is some $n \in \mathbb{N}$ such that $\psi \in H(s_n)$ and $\varphi \in H(s_i)$ for all $i = 0, \ldots, n-1$.

∇

Proposition 9.11. In every Hintikka structure (S, R, H):

I. If $\mathsf{EG}\varphi \in H(s)$, then $\varphi \in H(s_i)$ for every state s_i on some R-path $s = s_0, s_1, \ldots$.

II. If $\mathsf{AG}\varphi \in H(s)$, then $\varphi \in H(s_i)$ for every state s_i on every R-path $s = s_0, s_1, \ldots$.

III. If $\neg\, \mathsf{E}(\varphi \mathsf{U} \psi) \in H(s)$, then for every R-path $s = s_0, s_1, \ldots$, for every $i \in \mathbb{N}$ if $\psi \in H(s_i)$ then $\neg\varphi \in H(s_j)$ for some j such that $0 \leq j < i$.

VI. If $\neg\, \mathsf{A}(\varphi \mathsf{U} \psi) \in H(s)$, then there is an R-path $s = s_0, s_1, \ldots$, such that for every $i \in \mathbb{N}$ if $\psi \in H(s_i)$ then $\neg\varphi \in H(s_j)$ for some j such that $0 \leq j < i$.

Proof: Exercise. QED

Definition 9.6. A formula $\eta \in$ CTL is *satisfiable* in a HS $\mathcal{H}$ for some set of formulae Φ containing η, with a labeling function H if $\eta \in H(s)$ for some state s in $\mathcal{H}$; a set of CTL-formulae $\Theta \subseteq \Phi$ is *satisfiable* in $\mathcal{H}$ if $\Theta \subseteq H(s)$ for some state s in $\mathcal{H}$. ∇

Lemma 9.12. For any set of formulae $\Phi \in$ CTL and for every ITS $\mathcal{M} = (S, R, L)$ the structure $\mathcal{H}(\mathcal{M}) = (S, R, H)$, where $H(s) = \{\varphi \in \Phi \mid \mathcal{M}, s \models \varphi\}$ for every $s \in S$, is a Hintikka structure for Φ.

Proof: Straightforward verification of H1-H7. Exercise. QED

Theorem 9.13. A formula η of CTL is satisfiable iff it is satisfiable in a Hintikka structure for $cl(\eta)$.

Proof: Easy modification of the proof of Theorem 9.4. Exercise. QED

9.2.3 The construction of the tableau for CTL

Given an input formula $\eta \in$ CTL we are going to build a tableau for testing the satisfiability of η, following closely the construction for TLR in Section 9.1.

The pretableau construction phase again employs essentially the same rules PrExp and Next from Section 9.1.3, modulo the accordingly modified definition of the set of lean full expansions $\mathrm{FE}(\Gamma)$, which only differs in the set replacement operations specific to the temporal operators in CTL. We leave the details to the reader.

As before, the construction phase ends when none of these rules can add any new states or prestates to the current graph. The resulting graph is the pretableau $\mathcal{P}^\eta$.

The prestate elimination phase is the same as before and it ends with an initial tableau $\mathcal{T}_0^\eta$.

The state elimination phase employs essentially the same state elimination rules StElim1 and StElim2 as for the TLRtableau, now phrased in the CTL setting:

Rule StElim1$^{\mathrm{CTL}}$: If a state Δ contains in its label a formula $\mathsf{EX}\psi$ and there are no outgoing arcs marked with $\mathsf{EX}\psi$ from Δ to successors states in the current tableau, then remove that state from the tableau.

Rule StElim2$^{\mathrm{CTL}}$: If a state $\Delta \in \mathcal{T}_n^\eta$ contains in its label an eventuality that is not realized at Δ in $\mathcal{T}_n^\eta$, then remove that state from the tableau.

Recall, that there are two types of eventualities in CTL:

- ⋆ *existential eventualities*: $\neg\,\mathsf{AG}\varphi$ and $\mathsf{E}(\varphi\mathsf{U}\psi)$.
- ⋆ *universal eventualities*: $\neg\,\mathsf{EG}\varphi$ and $\mathsf{A}(\varphi\mathsf{U}\psi)$.

Thus, the rule StElim2$^{\mathrm{CTL}}$ now refers to realization of existential or universal eventualities. Let us specify what realization of such eventualities means.

The case of existential eventualities is similar to the one in TLR:

Definition 9.7. [**Realization of existential eventualities**]

1. An existential eventuality $\xi = \mathsf{E}(\varphi\mathsf{U}\psi)$ is *realized* at the state Δ in $\mathcal{T}_n^\eta$ if there exists a finite path $\Delta = \Delta_0, \Delta_1, \ldots, \Delta_m$, where $m \geq 0$, in $\mathcal{T}_n^\eta$, such that $\xi, \psi \in \Delta_m$ and $\xi, \varphi \in \Delta_i$ for every $i = 0, \ldots, m-1$.
2. An existential eventuality $\xi = \neg\,\mathsf{AG}\varphi$ is realized at the state Δ in $\mathcal{T}_n^\eta$ if there exists a finite path $\Delta = \Delta_0, \Delta_1, \ldots, \Delta_m$ in $\mathcal{T}_n^\eta$, such that $\xi, \neg\varphi \in \Delta_m$ and $\xi \in \Delta_i$ for every $i = 0, \ldots, m-1$.

In each case, we say that ξ is realized on the path $\Delta_0, \Delta_1, \ldots, \Delta_m$ and call any such path a *path-witness of the realization of the eventuality ξ in Δ*.

If $\varphi \in \Delta$ in the case of $\xi = \mathsf{EF}\varphi$, or respectively, $\neg\varphi \in \Delta$ in the case of $\xi = \neg\,\mathsf{AG}\varphi$, we say that ξ is *immediately realized* at the state Δ (by the singleton path Δ). ∇

Again, testing for realization of existential eventualities is a simple graph reachability problem,and can be done globally for all states in the current tableau by a standard marking procedure.

The realization of universal eventualities is more subtle. In order to realize a universal eventuality $\mathsf{A}(\varphi\mathsf{U}\psi)$ at a state Δ in $\mathcal{T}_n^\eta$ it would be too much to insist that *every path* starting from Δ in $\mathsf{A}(\varphi\mathsf{U}\psi)$ must reach a ψ-state while passing through φ-states, because not all such paths are needed in any Hintikka structure that can be extracted from that tableau, but only those that belong to *some branch* of the tableau. Recall that a branch in the initial tableau $\mathcal{T}_0^\eta$ is a subgraph of the tableau, rooted at an input state and branching at every state Δ into a family of successors obtained by selecting *one successor* from every non-empty family of successor states of Δ marked with the same formula $\mathsf{EX}\theta$. Then any branch in $\mathcal{T}_n^\eta$ is obtained as the remainder of a branch in $\mathcal{T}_0^\eta$, after possibly removing some bad states from it on the way.

Thus, for testing for realization of a universal eventuality that belongs to a state Δ in $\mathcal{T}_n^\eta$ it suffices to select *one branch* in $\mathcal{T}_n^\eta$ and then test *all paths in that branch* that pass through Δ for realizing that eventuality. More precisely, consider the universal eventuality $\xi = \mathsf{A}(\varphi\mathsf{U}\psi) \in \Delta$ (respectively, $\xi = \neg\,\mathsf{EG}\varphi \in \Delta$) and let $\mathcal{B}$ be a branch in $\mathcal{T}_n^\eta$ in which every

path starting from Δ has a finite prefix $\Delta = \Delta_0, \Delta_1, \ldots, \Delta_m$ consisting of states containing ξ and realizing that eventuality, i.e., $\xi, \psi \in \Delta_m$ and $\xi, \varphi \in \Delta_i$ for every $i = 0, \ldots, m-1$ (respectively, $\xi, \neg\varphi \in \Delta_m$ and $\xi \in \Delta_i$ for every $i = 0, \ldots, m-1$ for the case of $\xi = \neg\,\mathsf{EG}\varphi$). We say that such branch *realizes* ξ *at* Δ and the subgraph of that branch consisting of the union of the shortest finite prefixes realizing ξ along all paths starting from Δ will be called a *witness of the realization of the eventuality* ξ *at* Δ (on the branch $\mathcal{B}$). Then we say that the universal eventuality ξ is realized at Δ in $\mathcal{T}_n^\eta$ if there is at least one branch in $\mathcal{T}_n^\eta$ realizing ξ at Δ.

Alternatively, realization of a universal eventuality can be defined recursively, simultaneously for all states of the tableau:

Definition 9.8. [**Realization of universal eventualities**]

1. Let $\xi = \mathsf{A}(\varphi\mathsf{U}\psi) \in cl(\eta)$. Then for any state $\Delta \in \mathcal{T}_n^\eta$:
   (a) If $\psi \in \Delta$ then ξ is realized at Δ;
   (b) If $\varphi \in \Delta$ and, for every $\mathsf{EX}\theta \in \Delta$, there is a state $\Delta' \in \mathcal{T}_n^\eta$ such that $\Delta \xrightarrow{\mathsf{EX}\theta} \Delta'$ and ξ is realized at Δ', then ξ is realized at Δ, too.
2. Let $\xi = \neg\,\mathsf{EG}\varphi \in cl(\eta)$. Then for any state $\Delta \in \mathcal{T}_n^\eta$:
   (a) If $\neg\varphi \in \Delta$ then ξ is realized at Δ;
   (b) If for every $\mathsf{EX}\theta \in \Delta$, there is a state $\Delta' \in \mathcal{T}_n^\eta$ such that $\Delta \xrightarrow{\mathsf{EX}\theta} \Delta'$ and ξ is realized at Δ', then ξ is realized at Δ, too.

If $\psi \in \Delta$ in the case of $\xi = \mathsf{A}(\varphi\mathsf{U}\psi)$, or respectively, $\neg\varphi \in \Delta$ in the case of $\xi = \neg\,\mathsf{EG}\varphi$, we say that ξ is *immediately realized* at the state Δ (by the singleton path Δ).

∇

This definition also yields a non-deterministic recursive procedure for identifying all states in the current tableau that realize a given universal eventuality.

Exercise 9.7. Develop an efficient algorithm for identifying all states in the current tableau that realize a given universal eventuality, and determine its complexity.

Now, the state elimination phase is carried exactly like in the tableau for TLR.

Clearly, all phases of the tableau construction terminate. The resulting graph $\mathcal{T}^\eta$ upon stabilization of the state elimination phase is the final tableau.

The tableau $\mathcal{T}^\eta$ is *open* if it contains a state Δ such that $\eta \in \Delta$, otherwise it is *closed*.

9.2.4 Soundness of the tableau for CTL

Lemma 9.14. Let $\eta \in$ CTL and Γ be a prestate of $\mathcal{P}^\eta$ such that $\mathcal{M}, s \models \Gamma$ for some rooted ITS $(\mathcal{M}, s)$. Then:

I. $\mathcal{M}, s \models \Delta$ for at least one $\Delta \in \mathsf{states}(\Gamma)$.

II. Moreover, if $\mathsf{E}(\varphi\mathsf{U}\psi) \in \Gamma$ or $\mathsf{A}(\varphi\mathsf{U}\psi) \in \Gamma$ and $\mathcal{M}, s \models \psi$, then Δ can be chosen to contain ψ.

Respectively, if $\neg\,\mathsf{AG}\varphi \in \Gamma$ or $\neg\,\mathsf{EG}\varphi \in \Gamma$ and $\mathcal{M}, s \models \neg\varphi$, then Δ can be chosen to contain $\neg\varphi$.

Proof: Essentially the same as the proof of Lemma 9.5. Exercise. QED

Lemma 9.15. No satisfiable state $\Delta \in \mathcal{T}_0^\eta$ is removed by any application of the rules StElim1$^{\mathrm{CTL}}$ and StElim2$^{\mathrm{CTL}}$ during the state elimination phase.

Proof: For Rule StElim1$^{\mathrm{CTL}}$, and for Rule StElim2$^{\mathrm{CTL}}$ applied to existential eventualities, the proof essentially repeats the proof of Lemma 9.6.

The argument for the case of rule StElim2$^{\mathrm{CTL}}$ applied to universal eventualities is a bit more involved. We will prove that for any satisfiable state Δ and a universal eventuality $\xi = \mathsf{A}(\varphi\mathsf{U}\psi) \in \Delta$ (respectively, $\xi = \neg\,\mathsf{EG}\varphi \in \Delta$) there is a branch $\mathcal{B}$ in the initial tableau containing a witness of the realization of that eventuality consisting of satisfiable states.

We will consider the case $\xi = \mathsf{A}(\varphi\mathsf{U}\psi)$, the other case being analogous. If $\psi \in \Delta$ then ξ is realized at Δ, and any branch $\mathcal{B}$ that contains the state Δ (there is at least one such branch) can be selected. So, let us assume that $\psi \notin \Delta$, hence $\varphi, \mathsf{AX}\,\mathsf{A}(\varphi\mathsf{U}\psi) \in \Delta$.

Let $\mathcal{M}, s \models \Delta$ for some rooted ITS $(\mathcal{M}, s)$. Then $\mathcal{M}, s \models \mathsf{A}(\varphi\mathsf{U}\psi)$ hence every path in $\mathcal{M}$ starting at s has a finite prefix $s = s_0, s_1, \ldots, s_m$ ending with a state s_m satisfying ψ, and hence ξ, while all intermediate states $s_0, s_1, \ldots, s_{m-1}$ satisfy φ and ξ. We will call such a prefix of a path π *fulfilling* ξ, and the subsystem of $\mathcal{M}$ rooted at s and consisting of the states along all fulfilling ξ prefixes of paths starting at s – a *witness of the truth of* ξ *at* s *in* $\mathcal{M}$. Hereafter we fix such a witness and denote it by $W(\mathcal{M}, s, \xi)$. Since $\mathcal{M}, s \models \mathsf{AX}\,\mathsf{A}(\varphi\mathsf{U}\psi)$, we can assume that $W(\mathcal{M}, s, \xi)$ does not consist of s alone.

Now, we will construct a branch $\mathcal{B}$ realizing ξ at Δ in $\mathcal{T}_0^\eta$ as follows. We start with any branch leading from an input state to Δ – by construction of the initial tableau there is at least one. Then, we continue from Δ constructing $\mathcal{B}$ by building a witness of the realization of ξ at Δ step by step, in accordance with $W(\mathcal{M}, s, \xi)$:

1. For every nexttime formula $\mathsf{EX}\theta \in \Delta$ there is a successor t of s satisfying θ in $\mathcal{M}$, and hence satisfying the successor prestate Γ of Δ corresponding to $\mathsf{EX}\theta$. Then t satisfies at least one of the offspring states of Γ, say $\Delta'(\mathsf{EX}\theta)$.

   Note that $t \in W(\mathcal{M}, s, \xi)$ and hence $\mathcal{M}, t \models \xi$. Furthermore, if $\mathcal{M}, t \models \psi$ then $\Delta'(\mathsf{EX}\theta)$ is chosen to contain ψ; such choice is guaranteed by Lemma 9.14.

2. Now, any family of successor states of Δ, one for every nexttime formula $\mathsf{EX}\theta \in \Delta$, selected in such a way, provides an extension of the branch $\mathcal{B}$ at Δ.

3. Thereafter the construction of $\mathcal{B}$ repeats for every descendant of Δ along $\mathcal{B}$ until reaching states containing ψ. Since the construction follows $W(\mathcal{M}, s, \xi)$, every path in $\mathcal{B}$ starting from Δ is bound to reach such a state.

   From all such states onwards the branch is extended arbitrarily.

Once the branch $\mathcal{B}$ realizing ξ at Δ in $\mathcal{T}_0^\eta$ is constructed, we show by induction on n that none of the satisfiable states in the witness of the realization of ξ at Δ on $\mathcal{B}$ is eliminated from $\mathcal{T}_n^\eta$, and hence $\mathcal{B}$ realizes ξ at Δ in $\mathcal{T}_n^\eta$ for every n.

Thus, in the long run, no satisfiable state gets eliminated in the state elimination phase. QED

Theorem 9.16. [**Soundness of the tableau for** CTL] If $\eta \in$ CTL is satisfiable in then $\mathcal{T}^\eta$ is open.

Proof: Follows immediately from Lemmas 9.14 and 9.15. QED

9.2.5 Completeness of the tableau for CTL

To prove the completeness of the tableau for CTL, we have to construct a Hintikka structure $\mathcal{H}(\eta)$ satisfying the input formula η whenever the final tableau $\mathcal{T}^\eta$ is open. However, taking the entire tableau as a Hintikka structure, as we did in the case of TLR, would not work in general here, because there may be too many paths in it and the truth of some universal eventualities may get damaged.

Selecting one open branch of $\mathcal{T}^\eta$ to produce such a Hintikka structure may work sometimes, e.g., if there is at most one universal eventuality in $cl(\eta)$. In general, however, a more modular approach is needed, where eventualities are satisfied one by one, each of them at least once after every reappearance. To that aim, the Hintikka structure $\mathcal{H}(\eta)$ is constructed from building blocks associated with pairs $\langle$ state, eventuality $\rangle$, extracted from $\mathcal{T}^\eta$ as follows.

First, we will define so called local fragments for all states in $\mathcal{T}^\eta$.

Definition 9.9. [**Local fragments**] A *local fragment* for a state $\Delta \in \mathcal{T}^\eta$ consists of Δ together with all of its successor states in any fixed branch of $\mathcal{T}^\eta$.

The state Δ is an *internal node* of the fragment, while all other states in the fragment are its *leaves*. ∇

Then, we define realization witness fragments for all states in $\mathcal{T}^\eta$ and eventualities in them.

Definition 9.10. [**Realization witness fragments for universal eventualities**]

For a universal eventuality $\xi = \mathsf{A}(\varphi \mathsf{U} \psi)$ (respectively, $\xi = \neg\,\mathsf{EG}\varphi$) a *realization witness fragment* is simply any witness of the realization of ξ in $\mathcal{T}^\eta$, as defined in Section 9.2.3.

The last states of every path in such a fragment, i.e., the states containing ψ (respectively, $\neg\varphi$ in the case of $\xi = \neg\,\mathsf{EG}\varphi$), are the *leaves* of that fragment, and all other states in it are its *interior nodes*. ∇

Definition 9.11. [**Realization witness fragments for existential eventualities**]

For an existential eventuality $\xi = \mathsf{E}(\varphi \mathsf{U} \psi)$ (respectively, $\xi = \neg\,\mathsf{AG}\varphi$), a *realization witness fragment* is a path-witness of the realization of ξ in $\mathcal{T}^\eta$, extended with local fragments for each of the intermediate states of the path.

All interior states in the path-witness are *interior nodes* of the fragment; all other nodes are the *leaves* of that fragment. ∇

Now, for every state Δ in $\mathcal{T}^\eta$ we fix a local fragment $\mathrm{LFR}(\Delta)$, and for every pair (Δ, ξ) where Δ is a state in $\mathcal{T}^\eta$ and ξ is an eventuality in Δ, we fix a realization witness fragment $\mathrm{RWF}(\Delta, \xi)$. In each case, the state Δ is called the *root of the fragment.*

Note that all these fragments may come from different branches of $\mathcal{T}^\eta$.

Then we fix a list of all states in $\mathcal{T}^\eta$: $\Delta_0, \ldots, \Delta_{n-1}$ and a list of all eventualities occurring in $\mathcal{T}^\eta$: $\xi_0, \ldots, \xi_{m-1}$, and denote the fragment corresponding to (Δ_i, ξ_j) (if there is one) by $\mathrm{FR}(i, j)$.

We will be building the Hintikka structure $\mathcal{H}(\eta)$ in a sequence of steps, producing a chain by inclusion of *partial Hintikka structures* $\mathcal{H}_0(\eta), \mathcal{H}_1(\eta), \ldots$. Each of these will be a finite graph composed of fragments and consisting of interior nodes and leaves, where all unrealized eventualities will be listed in the leaves as 'pending realization'. Each step will taking care of the satisfaction of one eventuality at one of the current leaves, by grafting at that leaf a fragment realizing that eventuality. In the process, new unrealized eventualities may occur, and their realization will be deferred to the new leaves. If no unrealized eventualities are listed at a leaf, a local fragment will be grafted at it in order to ensure totality of the transition relation. The union of the chain $\mathcal{H}_0(\eta), \mathcal{H}_1(\eta), \ldots$ will be the structure $\mathcal{H}(\eta)$, where there will be no leaves and no unrealized eventualities.

The crucial observation that makes this construction possible is that if an eventuality ξ in the root of a given fragment is not realized within that fragment, then it is passed down to the leaves of the fragment. This enables the partial Hintikka structures $\mathcal{H}_n(\eta)$ not to falsify unrealized eventualities but to only defer their realization by listing them at the leaves. More precisely:

Proposition 9.17. Let ξ be an eventuality which belongs to a state $\Delta_i \in \mathcal{T}^\eta$ but is not realized at Δ_i in the fragment $\mathrm{FR}(i, j)$. Then:

- $\star$ If $\xi = \mathsf{E}(\varphi \mathsf{U} \psi)$ then there is a path in the fragment ending in a leaf, every state in which contains $\mathsf{EX}\,\mathsf{E}(\varphi \mathsf{U} \psi)$.
- $\star$ If $\xi = \neg\,\mathsf{AG}\varphi$ then there is a path in the fragment ending in a leaf, every state in which contains $\mathsf{EX}\neg\,\mathsf{AG}\varphi$.
- $\star$ If $\xi = \mathsf{A}(\varphi \mathsf{U} \psi)$ then for every path in the fragment that does not realize ξ every state on that path contains $\mathsf{AX}\,\mathsf{A}(\varphi \mathsf{U} \psi)$; in particular, the leaves of all such paths contain $\mathsf{AX}\,\mathsf{A}(\varphi \mathsf{U} \psi)$.
- $\star$ If $\xi = \neg\,\mathsf{EG}\varphi$ is not realized at the state Δ_i in a fragment $\mathrm{FR}(i, j)$ then for every path in the fragment that does not realize ξ, every state on that path contains $\mathsf{AX}\neg\,\mathsf{EG}\varphi$; in particular, the leaves of all such paths contain $\mathsf{AX}\neg\,\mathsf{EG}\varphi$.

We are now ready to prove the completeness theorem.

Theorem 9.18. [**Completeness of the tableau for** CTL] For any formula $\eta \in \mathrm{CTL}$, if the final tableau $\mathcal{T}^\eta$ is open, then the formula is satisfiable.

Proof:

We start the construction of $\mathcal{H}(\eta)$ with any state Δ containing η. This state is the label of the root Ψ_0 of the initial partial Hintikka structure $\mathcal{H}_0(\eta)$ having one leaf Ψ_0 with a list of pending eventualities $\mathsf{Event}(\Psi_0)$ consisting of all eventualities in Δ that are not immediately realized at Δ. Thereafter the construction continues inductively as follows. Let $\mathcal{H}_n(\eta)$ be constructed and let $\mathsf{Leaves}(\mathcal{H}_n(\eta))$ be the list of all leaves in it. Let Ψ be the fist leaf in that list, labelled with a state $\Delta_i \in \mathcal{T}^\eta$, and let ξ_j be the first eventuality in the list of pending eventualities $\mathsf{Event}(\Psi)$ at that leaf, if any. We then take a copy of the fragment $\mathrm{FR}(i,j)$ and graft it to the leaf Ψ, by identifying that leaf with the root of the fragment (which has the same label Δ_i); if $\mathsf{Event}(\Psi)$ is empty, we graft a copy of the local fragment $\mathrm{LFR}(\Delta_i)$ instead. Thus, $\mathcal{H}_{n+1}(\eta)$ is produced. The new list of leaves $\mathsf{Leaves}(\mathcal{H}_{n+1}(\eta))$ is obtained by removing Ψ from the list $\mathsf{Leaves}(\mathcal{H}_n(\eta))$ and then adding to it all leaves of the newly grafted fragment. Each of the new leaves inherits the list of pending eventualities of $\mathsf{Event}(\Psi)$ from which ξ_j has been removed, to which the list of eventualities in their respective labels, that are not immediately realized, has been appended. Note that the graph $\mathcal{H}_{n+1}(\eta)$ is an extension of the graph $\mathcal{H}_n(\eta)$.

This completes the construction of the chain $\mathcal{H}_0(\eta), \mathcal{H}_1(\eta), \ldots$, and consequently, of $\mathcal{H}(\eta)$ as the union of that chain.

Now, it remains to verify that $\mathcal{H}(\eta)$ defined as above is indeed a Hintikka structure satisfying η at its root Ψ_0. But, this verification is now straightforward from the construction, and is left as an exercise. QED

Corollary 9.19. The logic CTL has the small model property.

Proof:

The construction of $\mathcal{H}(\eta)$ above can be made finite by identifying leaves with earlier introduced states with the same label and reusing identical fragments. That is, if a fragment $\mathrm{FR}(i,j)$ is to be grafted to the leaf Ψ of the current partial HS $\mathcal{H}_n(\eta)$, and that fragment has already been grafted to another node Ψ' at an earlier stage, instead of grafting a new copy of $\mathrm{FR}(i,j)$, the leaf Ψ is identified with Ψ' and all incoming arrows to Ψ are accordingly redirected to Ψ'.

A more careful inspection of the construction shows that the Hintikka structure $\mathcal{H}(\eta)$ has a size $\mathcal{O}(2^{c|\eta|})$. We leave the details as an exercise. QED

As a consequence, we can now obtain an earlier result about tree model property of CTL* for the case of CTL.

Corollary 9.20. The logic CTL has the tree model property.

Proof: The construction above can be modified by defining the fragments as finite trees. Thus every partial Hintikka structure $\mathcal{H}_n(\eta)$ will be a finite tree, and eventually $\mathcal{H}(\eta)$ will be constructed as a tree. QED

Exercise 9.8. Prove that any open branch in the final tableau $\mathcal{T}^\eta$ is a Hintikka structure satisfying η if there at most one universal eventuality in $cl(\eta)$. Then, give an example of a case with more universal eventualities in $cl(\eta)$, where this approach does not work.

9.2.6 Model-checking tableaux for CTL

Just like in the cases of LTL and TLR, the tableau procedure for testing satisfiability in CTL can be smoothly modified to perform local model-checking. It is however relatively inefficient, compared to the labeling algorithm for global model-checking of CTL-formulae presented in Lecture 6, so we will not discuss it further.

10 Lecture 9: Automata-based methods for satisfiability testing and model-checking of LTL

Recommended reading: M. Vardi, An automata-theoretic approach to linear temporal logic, [Var96]. Available online from: http://www.cs.rice.edu/ vardi/papers/banff94rj.ps.gz

Additional readings:

1. M. Vardi, Lecture notes on Automata-Theoretic Approach to Automated Verification, course given in 1999 at The Weizmann Institute.

   Available online from: http://www.cs.rice.edu/ vardi/av.html

2. M. Vardi and P. Wolper, Automata-theoretic approach to automatic program verification, [VW86].

   Available online from: http://www.cs.rice.edu/ vardi/papers/lics86.pdf.gz

3. M. Vardi, Verification of concurrent programs: the automata-theoretic framework, [Var91].

   Available online from: http://www.cs.rice.edu/ vardi/papers/lics87r2.ps.gz

4. M. Vardi and T. Wilke, Automata: From Logics to Algorithms,

   Available online from: http://www.cs.rice.edu/ vardi/papers/wal07.pdf

5. M. Vardi, Automata-theoretic techniques for temporal reasoning,

   chapter in the Handbook of Modal Logic, [Var07].

   Available online from: http://www.cs.rice.edu/ vardi/papers/mlhb06.ps.gz

6. F. Kröger and S. Merz, Temporal Logic and State Systems, EATCS Texts in Theoretical Computer Science Series, 2008.

7. C. Baier and J.-P. Katoen, Principles of Model Checking, The MIT Press, 2008.

11 Lecture 10: Automata-based methods for satisfiability testing and model-checking of branching time logics

Recommended reading: O. Bernholtz, M. Vardi, and P. Wolper, An automata-theoretic approach to branching-time model checking (Extended abstract), in: Proc. of CAV'94.
Available from: http://www.cs.rice.edu/ vardi/papers/cav94.ps.gz

Additional readings:

1. M. Vardi, Lecture notes on Automata-Theoretic Approach to Automated Verification, course given in 1999 at The Weizmann Institute
   Available online from: http://www.cs.rice.edu/ vardi/av.html
2. O. Kupferman, M. Vardi, and P. Wolper, An automata-theoretic approach to branching-time model checking, [KVW00]. Long version of the CAV'94 paper recommended above, available from: http://www.cs.rice.edu/ vardi/papers/cav94rj.ps.gz
3. M. Vardi and T. Wilke, Automata: From Logics to Algorithms,
   Available online from: http://www.cs.rice.edu/ vardi/papers/wal07.pdf
4. M. Vardi, Automata-theoretic techniques for temporal reasoning,
   chapter in the Handbook of Modal Logic, [Var07].
   Available online from: http://www.cs.rice.edu/∼vardi/papers/mlhb06.ps.gz
5. C. Baier and J.-P. Katoen, Principles of Model Checking, The MIT Press, 2008.

References

[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.

[BdRV01a] P. Blackburn, M. de Rijke, and Y. Venema. *Modal Logic*. Cambridge University Press, 2001.

[BdRV01b] Patrick Blackburn, Maarten de Rijke, and Yde Venema. *Modal Logic*. CUP, 2001.

[Ben83] J. van Benthem. *Modal Logic and Classical Logic*. Bibliopolis, Naples, 1983.

[Ben84] J. van Benthem. Correspondence theory. In Gabbay and Guenthner [GG84], pages 167–247.

[Bet] E.W. Beth. Semantic entailment and formal derivability. *Nieuwe Reeks*, 18(13):309.

[Bet70] Evert W. Beth. *Formal Methods: An introduction to symbolic logic and to the study of effective operations in arithmetic and logic*. D. Reidel Publ. Co., Dordecht, 1970.

[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.

[BvBW07] Patrick Blackburn, Johan van Bentem, and Frank Wolter, editors. *Handbook of Modal Logic*. Elsevier, 2007.

[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.

[Eme83] E. Allen Emerson. Alternative semantics for temporal logics. *Theoretical Computer Science*, 26(1–2):121–130, September 1983.

[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.

[Fit83] Melvin Fitting. *Proof Methods for Modal and Intuitionistic Logics*. D. Reidel, 1983.

[Fit07] Melvin Fitting. Modal proof theory. In Blackburn et al. [BvBW07], pages 85–138.

[FL79] Michael J. Fisher and Richard E. Ladner. Propositional dynamic logic of regualar programs. *Journal of Computer and System Sciences*, 18:194–211, 1979.

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

[GG84] D.M. Gabbay and F. Guenthner, editors. *Handbook of Philosophical Logic*, volume 2. Reidel, Dordrecht, 1984.

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

[GO07] V. Goranko and M. Otto. Model theory of modal logic. In Blackburn et al. [BvBW07], pages 255–325.

[Gor98] Rajeev Gore. Tableau methods for modal and temporal logics. In M. D'Agostino et al., editor, *Handbook of Tableau Methods*. Kluwer, 1998.

[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.

[KMMP93] Y. Kesten, Z. Manna, H. McGuire, and A. Pnueli. A decision algorithm for full propositional temporal logic. In *CAV'93*, volume 697, pages 97–109. Springer, 1993.

[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.

[LP00] O. Lichtenstein and A. Pnueli. Propositional temporal logics: Decidability and completeness. *Logic Journal of the IGPL*, 8(1):55–85, 2000.

[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.

[Par81] D. Park. Concurrency and automata on infinite sequences. In *5th GI Conference on Theoretical Computer Science, Karlsruhe, Germany*, pages 167–183. Lecture Notes in Computer Science, Vol. 104. Springer, Berlin, 1981.

[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.

[Pra79] Vaughan R. Pratt. A practical decision method for propositional dynamic logic. In *Proceedings of the 10th Annual ACM Symposium on the Theory of Computing*, pages 326–227, San Diego, California, May 1979.

[Pra80] Vaughan R. Pratt. A near optimal method for reasoning about actions. *Journal of Computer and System Sciences*, 20:231–254, 1980.

[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.

[Smu68] Raymond M. Smullyan. *First-order Logic*. Springer-Verlag, 1968.

[Spa93] E. Spaan. *Complexity of Modal Logics*. PhD thesis, ILLC, Amsterdam University, 1993.

[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.

[Sti99] C. Stirling. Bisimulation, modal logic and model checking games. *Logic Journal of the IGPL*, 7(1):103–124, 1999.

[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.

[Tho84] R. H. Thomason. Combinations of tense and modality. In Gabbay and Guenthner [GG84], pages 135–166.

[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.

[Var91] M. Y. Vardi. Verification of concurrent programs: the automata-theoretic framework. *Annals of Pure and Applied Logic*, 51(1–2):79–98, 1991.

[Var96] M. Vardi. An automata-theoretic approach to linear temporal logic. In F. Moller and G. Birtwistle, editors, *Logics of Concurrency: Structure versus Automata*, pages 238–266. Lecture Notes in Computer Science, Vol. 1043. Springer, Berlin, 1996.

[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.

[Var07] M. Vardi. Automata-theoretic techniques for temporal reasoning. In P. Blackburn, J. van Benthem, and F. Wolter, editors, *Handbook of Modal Logic*, pages 971–990. Elsevier, 2007.

[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.

[Zan96] A. Zanardo. Branching time logic with quantification over branches: the point of view of modal logic. *Journal of Symbolic Logic*, 61:1–39, 1996.