12th EUROPEAN SUMMER SCHOOL IN LOGIC, LANGUAGE AND INFORMATION

6-18 August 2000, Birmingham, UK

TEMPORAL LOGICS OF COMPUTATIONS

Introductory course

Val Goranko

Department of Mathematics, Rand Afrikaans University

vfg@na.rau.ac.za, http://general.rau.ac.za/maths/goranko/

FOREWORD[1]

This course is an introduction to temporal logics in computer science, and in particular to systems and approaches used for specification and verification of properties of concurrent and reactive systems.

The notes begin with a brief journey through classical temporal logic (chapter 2), added more as a useful background than as an essential part of this course, en route to its core: temporal logics of computations, where we introduce the basic systems of linear and branching time logics and mu-calculus, and will discuss their semantics, expressiveness, axiomatizations, complexity, comparisons with each other, and relationships with automata which provide efficient decision procedures and methods for model checking.

This is an introductory course. The reader is expected to know just a little, and that would include a good grasp of propositional logic and a familiarity with the basics of first-order logic. Some exposure to modal logic would make it easier to understand how temporal logics work, and some experience with computing would provide you with the basic intuition on computations and will help you make good sense of its formal models. If you have some knowledge on formal verification and on complexity, this would be an advantage.

The notes are peppered with exercises, most of which fairly standard, but there are also a few more challenging, which are indicated with a *. This ranking is, of course, according to my judgement, which need not reflect the reader's abilities. Do try at least some of these. No better way to learn than 'hands on'.

I hope you will enjoy reading these notes. If you have any comments I would appreciate much receiving them at vfg@na.rau.ac.za.

Val Goranko

[1]These notes were prepared while I was on a sabbatical leave to the University of Saarbrücken.. I would like to acknowledge the financial support from the National Research Foundation of South Africa and the SASOL Research Fund of the Faculty of Natural Sciences at Rand Afrikaans University, as well as the kind hospitality of Patrick Blackburn and the Department of Computer Linguistics at the University of Saarbrücken. Finally, I thank Patrick Blackburn for his comments on the notes.

Contents

1 Time, temporal logics, and models of computation

This course has two major aspect: *temporal logics* and *models of computations*. In this chapter we offer a brief semi-formal introduction to both.

1.1 Reasoning about time: introductory remarks

1.1.1 Time over time

Time is timeless. The story about time, too.

The logical discussion of time and temporality goes back at least to ancient Greece, probably farther. Here are some historical landmarks:

- **Aristotle:** the "There will be a sea battle tomorrow" argument.
- The Stoic school. **Diodorus Chronus:**
  possible: *"what is or will be";*
  necessary: *"what is and will always be".*
- medieval ages:**William from Ockham;**
- modern approaches:
  - *classical:* **Bertrand Russell, Willard Quine;**
  - *modal:* **Arthur Prior**;
  - *temporality and tenses in natural language:* **Reichenbach.**
- *temporal logic in computer science:* beginning with **Amir Pnueli's** 1977 seminal paper [Pnu77] arguing that temporal logic can be used to reason about processes and computations, especially in concurrent and reactive systems.

1.1.2 Temporal logic vs classical logic.

Classical logic describes a *snapshot of the universe.*

Temporal logic pictures *a movie of the universe.*

1.1.3 Variety of temporal ontologies

A fundamental philosophical question on the topic is about the ontological nature and structure of the flow of time. This question presents distinct alternatives in several dimensions which have been subject of a lively debate since antiquity, and the answers (or choices) to these alternatives are crucial for the general development of the philosophy of nature. Here are some of these alternatives, put mathematically.

Is time:

- *discrete or continuous?*
- *instant-based or interval-based?*
- *bounded or unbounded?*
- *beginningless? endless?*
- *(or, is it not circular?)*
- *linear or branching? forward? backward?*
- *relative or absolute?*
- etc...

Here we shall not make any ontological commitments regarding the nature of time, but only technical assumptions suitable for the intended applications.

In order to reason about time in a formal language, we need to make at least some basic choices on these alternatives, regarding the *primitive time entities and the basic relations between them* which we want to discuss. The two traditional types of models of time are *point-based* and *interval based.*

The traditional logical language is *first-order logic.* It is a generic, universal language, suitable for both approaches.

1.1.4 Reasoning about time with first-order logic: point-based models

In the point-based models of time the *primitive temporal entities* are **instants** and the *basic relationships* between them are **equality** and **precedence.** Thus, the flow of time is represented as a set of instants with a binary relation of precedence on it: $\langle T, < \rangle$. The equality is implicitly assumed.

Here are some first-order sentences expressing basic properties which the flow of time may (or may not) have:

- *irreflexivity:* $\forall x \neg x < x$;
- *transitivity:* $\forall x \forall y \forall z (x < y \wedge y < z \rightarrow x < z)$;
- *linearity:* $\forall x \forall y (x = y \vee x < y \vee y < x)$;
- *density: between every two precedence-related instants there is another instant:* $\forall x \forall y (x < y \rightarrow \exists z (x < z \wedge z < y))$;
- *no end:* $\forall x \exists y (x < y)$;
- *every instant has a successor:* $\forall x \exists y (x < y \wedge \forall z (x < z \rightarrow y \leq z))$;

- *forward discreteness: every instant with a successor has an immediate successor:* $\forall x \exists y(x < y \rightarrow \exists y(x < y \wedge \forall z(x < z \rightarrow y \leq z)))$;
- *backward discreteness:* **exercise**.

Questions:

- Does *endlessness* imply *unboundedness*?
- If every *instant has an immediate successor,* does *discreteness* follow? (I.e., does backward discreteness not follow from forward discreteness?)

In order to talk about past and future events in first-order logic we need a reference to *now.* Let $N(x)$ mean 'x is the current instant' (i.e., 'it is x o'clock'). Here are some first-order statements about *truths in time:*

- "*Sometime in the future it will be the case that* φ": $N(x) \rightarrow \exists y(x < y \wedge \varphi(y))$;
- " *It has always been the case that* φ": $N(x) \rightarrow \forall y(y < x \rightarrow \varphi(y))$;
- "ψ *will hold until* φ *occurs*": $N(x) \rightarrow \exists y(x < y \wedge \varphi(y) \wedge \forall z(x < z \wedge z < y \rightarrow \psi(z)))$;
- Assuming the flow of time is like the natural numbers: φ *occurs infinitely often:* $\forall x \exists y(x < y \wedge \varphi(y))$;
- *From some instant onwards,* φ *will hold forever:* **exercise**.

First-order logic is not expressively omnipotent, though. Some natural and important properties of flows of time cannot be expressed with first-order sentences. For example, one cannot say in first-order logic that *a linear time flow is well-ordered,* meaning that every descending sequence of instants must be finite. Neither can *Dedekind completeness* be expressed: every non-empty and bounded above set of instants has a least upper bound. For these, one needs a *monadic second order-logic* where quantification over sets is allowed. As we will see further, however, these properties can be expressed in *propositional temporal logic*, too.

1.1.5 Reasoning about time with first-order logic: interval-based models

Now the primitive temporal entities are **intervals.**

In this model there are many more natural relationship, but one can select a few basic ones: **equality, disjoint precedence, inclusion, overlap.**

The flow of time now has a richer structure: $\langle T, \prec, \sqsubseteq, O \rangle$, usually presumed linear.

- *reflexivity of* $\sqsubseteq$ *:*$\forall x x \sqsubseteq x$;

- *anti-symmetry of* $\sqsubseteq$: $\forall x \forall y (x \sqsubseteq y \wedge y \sqsubseteq x \rightarrow x = y)$;
- *symmetry of* O : $\forall x \forall y (xOy \rightarrow yOx)$
- *downward monotonicity of precedence w.r.t. inclusion:* $\forall x \forall y \forall z (x \prec y \wedge z \sqsubseteq x \rightarrow z \prec y)$;
- *atomicity of* $\sqsubseteq$ *:* $\forall x \exists y (y \sqsubseteq x \wedge \forall z (z \sqsubseteq y \rightarrow z = y))$;

Question: Is this true or false: $\forall x \forall y \forall z \forall u (x \prec y \wedge y \prec z \wedge xOu \wedge zOu \rightarrow y \sqsubseteq u)$?

There have been various proposals for *interval temporal logics* and this approach is the more prominent in *artificial intelligence* (e.g. Allen's logic of planning [All84], and Kowalski & Sergot's calculus of events [KS86], or Halpern and Shoham's interval calculus [HS86]) but also in some applications to computer science such as *real-time logics* and *hardware verification* (e.g. [HMM83] and [CHR91]).

Clearly, the point-base and interval-based approaches are closely related. Intervals and most of their relationships can be defined in the point-based language, but that does not brand the interval-based approach as 'redundant'. Conversely, in many cases the instants can be recovered from the interval structure by means of a 'Dedekind cuts' type of construction, but now they are high-level objects rather than primitive entities. Finally, both instants and intervals can co-exist in two-sorted models.

1.1.6 Our framework for temporal logics

In this course we will work in the most traditional and proven useful framework for temporal logics. In particular,

- The models of time which we will discuss in here are **instant-based.** Again, this is not an ontological choice, but rather a technical convention suitable for the intended applications.
- For several reasons, some of which will transpire during the course, *not first-order but* **modal logic** is by far the more popular logical framework for developing temporal logics in general, and in particular for applications in computer science.
- More specifically in this course we will only discuss **propositional** temporal logics. They all have their first-order extensions with their own technical issues, but the temporal aspects which are in the focus of the course, are fully represented on propositional level, while the added complications of the first-order machinery are avoided.

References for this section*:* There is ample philosophical and formal-logical literature on time and reasoning about time. A recommended book is [Ben93]. Also, see the surveys on temporal reasoning in [AGM92].

1.2 Preliminaries on modal logic

Here we provide a very basic background on modal logic, just to get started.

1.2.1 Modes of truth and the variety of modal logics.

There are various *modes of truth* and they determine various *modal operators* on propositions, and respectively various kinds of *modal logics.* Some of them:

- *aletic:* φ is necessarily true: $\Box\varphi$ and φ is possibly true: $\Diamond\varphi$;
- *epistemic:* the agent A knows φ : $K_A\varphi$;
- *deontic:* it ought to be (it is obligatory) that φ;
- *temporal:* it will always be the case that φ (**and more, coming soon...)**

But also:

- in *propositional dynamic logic (PDL):*
  $[\alpha]\varphi$: after every possible execution of the program α, φ will be true;
- in the *logic of provability in Peano arithmetic:*
  $\Box_{PA}\varphi$: φ is provable in Peano arithmetic.

1.2.2 Syntax and Kripke semantics

Formally, the **syntax** of propositional modal logic is defined as follows: the language consists of:

$\bot, \rightarrow$ (or any other functionally complete set of classical propositional connectives), a *modal operator* $\Box$, and a set of *atomic propositions* $\mathbf{ATOM}= \{p_0, p_1, ...\}$. Then the set of formulae is recursively defined by:

$$\varphi = p \mid \bot \mid \varphi_1 \rightarrow \varphi_2 \mid \Box\varphi$$

The *dual operator* of $\Box$ is denoted by $\Diamond$, i.e. $\Diamond\varphi = \neg\Box\neg\varphi$.

In the early sixties S. Kripke introduced a *relational semantics* for modal logic based on the notion of *Kripke model:* a triple $\langle W, R, V\rangle$ where W is a non-empty set of *possible worlds;* $R \subseteq W^2$ is an *accessibility relation* between possible worlds, and V : $\mathbf{ATOM}\rightarrow \mathbf{2}^W$ is a *valuation* which assigns to every atomic proposition the set of possible worlds where it is true. The truth at every possible world is subject to the laws of classical propositional logic, and the truth of $\Box\varphi$ is determined according to Leibniz's idea that something is *necessarily true if it is true in every world possible with respect to the actual world.*

Kripke semantics is defined as follows. The basic concept is **truth of a formula** φ **at a possible world** w **of a Kripke model** $M = \langle W, R, V \rangle$, denoted by $M, x \vDash \varphi$, recursively defined by the clauses:

- $M, x \vDash p$ iff $x \in V(p)$;
- $M, x \nvDash \bot$**;**
- $M, x \vDash \varphi_1 \to \varphi_2$ if $M, x \vDash \varphi_1$ implies $M, x \vDash \varphi_2$;
- $M, x \vDash \Box\varphi$ if $M, y \vDash \varphi$ for **every** $y \in W$ such that xRy.

Exercise: show that the respective clause for $\Diamond\varphi$ is:

- $M, x \vDash \Diamond\varphi$ if $M, y \vDash \varphi$ for **some** $y \in W$ such that xRy.

A modal formula is called:

- **valid in the Kripke model** M**,** denoted $M \vDash \varphi$, if *it is true at every possible world of* M;
- **(Kripke) valid,** denoted $\vDash \varphi$, if *it is valid in every Kripke model.*

1.2.3 The basic normal modal logic K

The set of valid modal formulae is axiomatized by the *minimal normal modal logic* **K** which extends the classical propositional logic **CL** with the **axiom schema**

(K): $\Box(\varphi \to \psi) \to (\Box\varphi \to \Box\psi)$

and the modal **rule of inference** *Necessitation:*

(NEC): If $\vdash \varphi$ then $\vdash \Box\varphi$

added to the classical rule *Modus Ponens:* if $\vdash \varphi$ and $\vdash \varphi \to \psi$ then $\vdash \psi$.

A modal formula φ is called a **theorem** of **K**, denoted $\vdash \varphi$, if it is derivable from the axioms of **K** using the rules of inference.

***Theorem* (weak completeness):** *A modal formula* φ *is valid iff it is a theorem of* **K**, *i.e.:*

$$\vDash \varphi \text{ iff } \vdash \varphi.$$

Completeness results and related issues will be discussed in detail in the chapter on classical temporal logics.

1.2.4 Multimodal logics

Often **multi-modal** languages are used, containing a whole set of modalities $\{[a]\}_{a\in A}$. The set of formulae extends in the obvious way:

$$\varphi = p \mid \bot \mid \varphi_1 \to \varphi_2 \mid [a]\varphi.$$

Likewise the semantics: a Kripke frame for such a language is a structure $\langle W, \{R_a\}_{a\in A}\rangle$ which contains an accessibility relation for each modality. The notions of model, validity and satisfiability are generalized in a straightforward way (will be done later). The complete axiomatization of the multimodal **K** is obtained by simply taking the axiom schema *(K)* and the inference rule *(NEC)* for each modality.

Usually, the modalities in multimodal logics interact, and their respective accessibility relations are related. A typical example is *temporal logic* regarded as a bimodal logic. Furthermore, sometimes the set of modalities in a multimodal logic has its *internal structure*, a typical example being *propositional dynamic logic* **PDL,** where they form a Kleene algebra of regular expressions. By the way, PDL is a logic to reason about programs and computations, too, but this is another story...

References: [Ben83] and [HC96] are classical references on modal logic, and [BdRVar] is a very comprehensive and state-of-the-art one.

1.3 Reasoning about time with modal logic

Temporal logic can be regarded as modal logic with modalities expressing temporal aspects of truth, and the possible worlds being instants of time. The major difference in the style of expression between first-order and modal logic is this: first-order logic takes an *external* (the point of reference is outside of the model) *and global* (in terms of quantification) *viewpoint* on the model about which it reasons, while the viewpoint of modal logic is *internal and local:* the point of reference is the *current world* (in this case, *instant*) and the quantification is relative to it.

1.3.1 Prior's classical temporal operators:

In [Pri67] Prior laid the foundations of the modal approach to temporal logic, by introducing the basic temporal operators as modalities, interpreted in a Kripke-style semantics.

Here are the traditional temporal modalities a la Prior:

The **future** operators:

- $\mathbf{F}\varphi$: "Sometime in the future it will be the case that φ"
  Compare: $\exists y(x < y \wedge \varphi(y))$
- $\mathbf{G}\varphi$;"Always in the future it will be the case that φ"
  Compare: $\forall y(x < y \to \varphi(y))$.

Note: $\mathbf{G}\varphi \equiv \neg\mathbf{F}\neg\varphi$ (or, that's how it looks...)

Likewise, the **past** operators:

- $\mathbf{P}\varphi$: "Sometime in the past φ was the case", or "It has been the case that φ";
- $\mathbf{H}\varphi$: "Always in the past φ was the case", or "It has always been the case that φ";
- Also: sometime $\mathbf{E}\varphi$ / always $\mathbf{A}\varphi$;

Soon we shall extend Prior's arsenal with more operators.

1.3.2 Combining temporal operators. Expressing tenses in natural language.

Some examples:

- $\mathbf{P}\varphi$ corresponds to present perfect (**has been),** rather than to past simple (**was).**
- **had been: PP**φ,
- **will have been; FP**φ
- **PF**φ (**would be)**
- **PFP**φ (**would have been);**
- **FPF**φ ??

Prior's operators are not suitable for expressing *progressive tenses* , though. Seemingly, interval-based logic is a more natural framework for that.

There have been numerous studies of temporal aspects of natural language (see Steedman's survey article in [BtM97]). One of the first analytic approaches to that matter was Reichenbach's *theory of tenses* (see [Rei47]) associating 3 time points with each tense: *utterance, reference*, and *action*. With that theory Reichenbach characterized most of the tenses in natural language.

1.4 Temporal models of computations

1.4.1 Aspects of temporality in computer science

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

- specification and verification of concurrent and reactive systems;
- scheduling of the execution of programs by an operating system;

- synchronization of concurrent processes;
- temporal databases;
- real-time processes and systems;
- hardware verification.

1.4.2 Expressing properties of computations with temporal logic

- **Invariance,** or **safety properties:** *what must not happen throughout the computation (i.e. 'bad things') will not happen.* Examples:
  - **Partial correctness properties**: " *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*".

  For 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.
- **Eventuality,** or **liveness properties:** *what has to happen during the computation (i.e. 'good things') will happen.* Examples:
  - **Total correctness properties:** " *If a pre-condition P holds at the input of the program, then it will terminate and a post-condition Q will hold at the output.*" Also:
  - " *If the train has entered the tunnel, it will eventually leave it (before any other train has entered it).*"
  - " *Once a printing job is activated, eventually it will be completed.*"
  - " *If a message is sent, eventually it will be delivered.*" etc.
- **Fairness properties: "** *All processes will 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 (incl. an entire book: [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 enough then eventually its request will be granted.*

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

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

Examples of fairness will be discussed later.

References: [MP92] is a comprehensive reference on the use of temporal logic for specification of concurrent and reactive systems, and [MP95] is its continuation showing how that can be used to guarantee safety of such systems. In any event, for and in-depth investigation on the topic, it is worth visiting Manna's and Pnueli'ss webpages at http://theory.stanford.edu/~zm/ and http://www.wisdom.weizmann.ac.il/home/amir/public_html/.

1.4.3 Labelled transition systems and runs.

A widely used formalism for representing concurrent processes is **(multiprocess) transition system (TS)** $\mathcal{T} = \left\langle S, \left\{ \overset{a}{\longmapsto} \right\}_{a \in \mathbf{A}} \right\rangle$ consisting of:

- a non-empty set **S** of **states;**
- a non-empty set **A** of **actions (transitions, processes)** which act, possibly non-deterministically, on the set of states.

  With every action $a \in \mathbf{A}$ there is a **transition relation** $\overset{a}{\longmapsto} \subseteq \mathbf{S} \times \mathbf{S}$ associated.

  The intuition: $s \overset{a}{\longmapsto} t$ holds if the action a can transform the state s into the state t.

  In addition, a set $\mathbf{S}_0$ of **initial states** can be specified.

Examples:

- programs or processes run concurrently on the same computing device;
- finite automata;
- Petri nets;
- rewriting systems;
- models for process algebras;
- various pieces of hardware;
- vending machines (Stirling's example);
- last but not least: *(multimodal) Kripke frames;*

Labelled transition system: a transition system $\mathcal{T}$ together with a **labelling** $L : S \to 2^{\mathbf{AP}}$, where **AP** is a fixed set of **atomic propositions** and every state s is assigned as a label the set of atomic propositions true at s.

Instead of labelling, *valuations* are usually used in classical modal and temporal logic. A valuation $V : \mathbf{AP} \rightarrow \mathbf{2}^S$ assigns to each member of a fixed set of atomic propositions **AP** the set of states where it is true. Clearly, the two formalisms are equivalent. When discussing mu-calculus we will make use of both.

The states in a transition system can be thought of as program states, or configuration states of a machine, or memory registers of a computer, etc. The actions can represent program commands, or concurrent processes, or agents' actions, or ... simply periods of time over which the states are transformed.

Furthermore, the set of actions can have an internal structure, e.g. be endowed with some *action constructs such as composition, conditional branching, loops* etc. Transitions systems with appropriately structured set of actions provide a convenient formalism for specifying operational semantics of programming languages (see e.g. [Plo81] and [Sti92]).

Definition: **A run (trace, history, execution)** in a transition system $\langle \mathbf{S}, \mathbf{A} \rangle$ is a sequence of states and actions which transform every state into its successor: $s_0 \stackrel{a_0}{\longmapsto} s_1 \stackrel{a_1}{\longmapsto} s_2 ...$ If a set of initial states is specified, then the initial state s_0 of the run must belong to it. Often the notion of a run is used as a synonym of **computation.** We will reserve the latter term to refer to the *observable effect of the run in a labelled transition system:* $L(s_0), L(s_1), L(s_2), ...$. The idea is that the atomic propositions (or what is hidden in them) describe fully the status of the computation, incl. the values of all important variables etc.

If the run is finite, it is also called **terminating**. Since we will be discussing mainly non-terminating computations, for technical convenience we can add an *idle state* (or, *terminating state*) and consider every run an infinite sequence.

For the semantics of temporal logics we will consider the simple case of a *monotransition system* $\langle S, R \rangle$ i.e. with only one type of actions with a transition relation R, which will usually be assumed *serial,* i.e. every state has at least one R-successor. Note that this is nothing else, but a Kripke frame.

In such transition systems a run is simply an infinite sequence of states $s_0, s_1, s_2, ...$, every one of which related to its successor by the transition relation. Formally, a run is a mapping $\sigma : \mathbf{N} \rightarrow S$, but instead of $\sigma(n)$ we usually write the explicit member of the sequence s_n.

This being the case, one can bypass the notion of a run altogether, and introduce the notion of *abstract computation,* being simply a labelling $\sigma : \mathbf{N} \rightarrow \mathbf{2}^{AP}$. Indeed, this is the case when the transition system is $\langle \mathbf{N}, \mathbf{succ} \rangle$ where **succ** is the successor function on **N**. As we will see later, this is a general enough representation.

It should be noted that a number of abstractions and idealizations are made in this formalization of the notion of computation. In particular, in this framework we are not interested in the details or the structure of the program which performs the computation. For us it is just a blackbox, and it only matters how it transforms states. Of course, one can take a different approach by considering the computations from viewpoint of the *internal structure* of the programs. This is the approach of *logics of programs* such as the *propositional dynamic logic PDL,* or various logics of processes (see [KT90]).

Another important 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 compu-

tations usually make use only of the future fragments of temporal logics.

References: For a detailed discussion on transition systems in the framework of temporal logic and survey of results see [Sti92].Note: Stirling uses the term "labelled" differently: by labels he means the names of the actions.

1.5 The basic modal (temporal) logic of transition systems

Transition systems are just multimodal Kripke frames $\mathcal{T} = \left\langle S, \left\{ \overset{a}{\longmapsto} \right\}_{a \in \mathbf{A}} \right\rangle$. With every set of actions (transitions) $\mathbf{A}$ we can associate a corresponding temporal language. Following the classical framework, it would have a pair of future-past modalities for each transition a. Usually, however, the applications of temporal logic to computer science use only the *future fragments* of the temporal languages, though there have been a number of arguments and proposals to add the past operators to temporal logics of computations.

Here we shall introduce the 'future' fragment of the basic temporal logic for transition systems. Thus, the language $\mathfrak{L}_{\mathbf{A}}$ corresponding to a set of actions $\mathbf{A}$ is a propositional *multimodal language* where every transition $a \in \mathbf{A}$ is represented by a modality $[a]$. The inductive definition of formulae is:

$$\varphi = p \mid \bot \mid \varphi_1 \to \varphi_2 \mid [a]\varphi$$

Besides all Boolean connectives, for each modality $[a]$ its dual is defined: $\langle a \rangle \varphi = \neg[a]\neg\varphi$.

The models are just labelled transition systems (LTS). The definition basic semantic notion is a **truth of a formula φ at a state t of an LTS** $M = \langle S, R, L \rangle$ follows the standard Kripke semantics but using *labellings* instead of valuations (essentially a matter of tradition):

- $M, t \vDash p$ iff $p \in L(t)$;
- $M, t \nvDash \bot$**;**
- $M, t \vDash \varphi_1 \to \varphi_2$ if $M, t \vDash \varphi_1$ implies $M, t \vDash \varphi_2$;
- $M, t \vDash [a]\varphi$ if $M, s \vDash \varphi$ for every $s \in S$ such that $t \overset{a}{\longmapsto} s$.

The basic modal logic $\mathbf{K}_A$ axiomatizing the semantics above is simply the multimodal version of K, i.e. it extends the classical propositional logic with the axiom schema:

(K_a):$[a](\varphi \to \psi) \to ([a]\varphi \to [a]\psi)$,

and the rules of necessitation *(NEC_a):* if $\vdash \varphi$ then $\vdash [a]\varphi$,

for each $a \in \mathbf{A}$.

Usually, the additional requirement of **seriality** (in the computer science literature often called *totality*) is imposed. It has two versions:

- **strong:** every state must have a successor along *each* transition. This is characterized by the additional axiom schema *(D_a):* $\langle a \rangle \top$ for each $a \in \mathbf{A}$.

- **weak:** every state must have a successor along *some* transition. This can only be characterized axiomatically for transition systems with *finitely many transitions* $a_1, ..., a_n$ by the axiom *(D):* $\langle a_1 \rangle \top \vee ... \vee \langle a_n \rangle \top$.

The resulting logics are *canonical,* hence strongly complete, and they have the finite model property, hence are decidable with the same complexity as **K** : PSPACE. The proofs are straightforward generalizations of those for **K**.

We presented this logic here not because it is particularly interesting, but because it is the *basic* temporal logic of transition systems, upon which other temporal logics, incl. μ-calculus, are built.

2 A survey of classical temporal logic

This is a brief survey of most of the traditional topics and systems of temporal logic in the classical Priorian framework. There are numerous technical and philosophical studies of these logics. For more details on results mentioned here, further references, and much more see [Bur84], [BS84], [GHR94], [GHR95], [Gol92], [BdRVar].

2.1 Prior's modal framework for temporal logic

2.1.1 Syntax and semantics

The *basic temporal language* $\mathfrak{L}_t$ is a propositional bimodal language, consisting of: the Booleans $\bot, \rightarrow$, the temporal operators G, H, and a set of *atomic propositions* $\mathbf{AP} = \{p_0, p_1, ...\}$. The choice of Booleans is just for technical convenience; any other functionally complete system of Boolean connectives will serve the same purpose.

The set of formulae is recursively defined by:

$$\varphi = p \mid \bot \mid \varphi_1 \rightarrow \varphi_2 \mid G\varphi \mid H\varphi$$

The other classical connectives $\neg, \vee, \wedge, \leftrightarrow$ are defined as usual:

$\neg\varphi := \varphi \rightarrow \bot, \varphi \vee \psi := \neg\varphi \rightarrow \psi, \varphi \wedge \psi := \neg(\varphi \rightarrow \neg\psi), \varphi \leftrightarrow \psi := (\varphi \rightarrow \psi) \wedge (\psi \rightarrow \varphi)$.

Also:

$$\top := \neg\bot, \ F\varphi := \neg G\neg\varphi, P\varphi := \neg H\neg\varphi.$$

Besides, we denote $A\varphi = H\varphi \wedge \varphi \wedge G\varphi$, and dually $E\varphi = P\varphi \vee \varphi \vee F\varphi$. On linearly ordered flows of time these operators mean respectively always and sometime.

We now introduce **Kripke semantics** for $\mathfrak{L}_t$:

Temporal frame is a pair $\mathbf{T} = \langle T, \prec \rangle$ where T is a non-empty set of **time instants** , and $\prec$ is a **precedence relation** on T , about which no assumptions are made so far.

A **valuation** in $\mathbf{T}$ is a mapping $V : \mathbf{AP} \rightarrow \mathbf{2}^T$ which associates with every atomic proposition the set of time instants where it is true. Sometimes, instead of a valuation, a **labelling** $L : T \rightarrow \mathbf{2}^{\mathbf{AP}}$ is used, which acts dually: to every time instant t from $\mathbf{T}$ it assigns the set of atomic propositions which are true at t. While discussing classical temporal logic we will use valuations.

A **temporal model** is a triple $\langle T, \prec, V \rangle$ where $\langle T, \prec \rangle$ is a temporal frame and V is a valuation in it.

Truth of a formula φ at an instant t in a temporal model $M = \langle T, \prec, V \rangle$, denoted by $M, t \vDash \varphi$, is defined as in modal logic, assuming that $\prec$ *is the accessibility relation associated with* G *, and its converse* $\succ$ *is associated with* H:

- $M, t \vDash p$ iff $t \in V(p)$;
- $M, t \nvDash \bot$**;**
- $M, t \vDash \varphi_1 \rightarrow \varphi_2$ if $M, t \vDash \varphi_1$ implies $M, t \vDash \varphi_2$;
- $M, t \vDash G\varphi$ if $M, s \vDash \varphi$ for every $s \in T$ such that $t \prec s$.
- $M, t \vDash H\varphi$ if $M, s \vDash \varphi$ for every $s \in T$ such that $t \succ s$, i.e. $s \prec t$.

Exercise*:* Show that the respective clauses for F and P are:

- $M, t \vDash F\varphi$ if $M, s \vDash \varphi$ for some $t \in T$ such that $t \prec s$.
- $M, t \vDash H\varphi$ if $M, s \vDash \varphi$ for some $s \in T$ such that $s \prec t$.

A temporal formula φ is called:

- **valid in the temporal model** M, denoted $M \vDash \varphi$, if φ is true at every instant of M;
- **valid in the temporal frame** $\mathbf{T}$, denoted $\mathbf{T} \vDash \varphi$, if φ is valid in every model on $\mathbf{T}$(i.e. under every valuation in $\mathbf{T}$);
- **valid,** denoted $\vDash \varphi$, if it is valid in every temporal frame;
- **satisfiable in a model** M, if it is true at some instant of M;
- **satisfiable** if it is satisfiable in some temporal model M; otherwise it is **unsatisfiable.**

A fundamental relationship:

φ *is valid iff* $\neg\varphi$ *is unsatisfiable;*

respectively,

φ *is satisfiable iff* $\neg\varphi$ *is not valid.*

A stronger semantic notion is **logical consequence.** A temporal formula φ **follows logically from a set of formulae** Γ, resp. Γ **implies logically** φ, denoted $\Gamma \vDash \varphi$, if φ is true at every instant of a temporal model at which all formulae from Γ are true.

In particular, $\emptyset \vDash \varphi$ iff φ is valid.

Further, a **set of temporal formulae** Γ **is satisfiable** if there is an instant in a temporal model where *each* formula from Γ is true, i.e. all formulae from Γ are *simultaneously satisfiable.*

Again, fundamental relationships:

$$\Gamma \text{ is satisfiable iff } \Gamma \nvDash \bot$$

$$\Gamma \cup \{\varphi\} \text{ is satisfiable iff } \Gamma \nvDash \neg\varphi$$

Exercise*:* check these.

2.1.2 Standard translation of the semantics of temporal logic into classical logic

Let L_1 be a first-order language with $=$, a binary predicate symbol R , and a denumerable set of unary predicate symbols $\mathfrak{P} = \{P_0, P_1, ...\}$. If the symbols from $\mathfrak{P}$ are regarded as predicate variables, that renders a *second-order* language L_2.

The **standard translation** (see [Ben84]) ST of $\mathfrak{L}_t$ into L_1 :

- $ST(p_i) = P_i(x)$;
- $ST(\bot) = \bot$;
- $ST(\varphi_1 \to \varphi_2) = ST(\varphi_1) \to ST(\varphi_2)$;
- $ST(G\varphi) = \forall y(xRy \to ST(\varphi)[y/x])$ where y is a fresh variable;
- $ST(H\varphi) = \forall y(yRx \to ST(\varphi)[y/x])$ where y is a fresh variable;

Exercise*:* Show that $ST(F\varphi) = \exists y(xRy \wedge ST(\varphi)[y/x])$, and write down the clause for $ST(P\varphi)$.

Examples: $ST(Gp_1 \to FHp_2) = \forall y(xRy \to P_1y) \to \exists y(xRy \wedge \forall z(zRy \to P_1z))$;

$ST(Gp \to GGp) = \forall y(xRy \to Py) \to \forall y(xRy \to \forall z(yRz \to Pz))$.

***Exercise**:* show that the latter formula is equivalent to *transitivity* of the relation R.

Now, every temporal model $M = \langle T, \prec, V\rangle$ can be regarded as an L_1-model by interpreting R as $\prec$ and each P_i as $V(p_i)$. Respectively, every temporal frame can be regarded as an L_2-model.

$\forall \overline{P}\Phi$ denotes the universal closure of an L_2 -formula Φ over all predicate variables occurring in it.

To distinguish from temporal truth/validity $\vDash$, we shall denote classical truth/validity by $\Vdash$.

Proposition: *For every temporal model* $M = \langle T, \prec, V\rangle$, $t \in T$, and a temporal formula φ :

$$M, t \vDash \varphi \text{ iff } M \Vdash ST(\varphi)[x := t],$$

$$M \vDash \varphi \text{ iff } M \Vdash \forall x ST(\varphi),$$

$$\langle T, \prec\rangle \vDash \varphi \text{ iff } \langle T, \prec\rangle \Vdash \forall \overline{P} \forall x ST(\varphi),$$

$$\vDash \varphi \text{ iff } \Vdash \forall x(\varphi) \text{ iff } \Vdash \forall \overline{P} \forall x ST(\varphi).$$

Thus: *validity of a temporal formula in a model is a first-order notion, while validity in a frame is a second-order notion.*

This standard translation is at the heart of the *correspondence theory* between modal and classical logic, which offers a systematic treatment of various aspects on temporal logic such as expressiveness, definability, model theory with the tools and techniques of classical logic. For more detail see [Ben83], [Ben84], and [BdRVar].

2.2 The basic temporal logic $\mathbf{K}_t$: a crash course

We begin with what can be regarded as the *minimal temporal logic* to which reasonable Kripke semantics can be ascribed, with no assumptions on the flow of time.

2.2.1 Axiomatic system for $\mathbf{K}_t$

The **minimal temporal logic $\mathbf{K}_t$** axiomatizing all valid temporal formulae extends the classical propositional logic **CL** with the **axioms schemata**

(K_G): $G(\varphi \to \psi) \to (G\varphi \to G\psi)$,

(K_H): $H(\varphi \to \psi) \to (H\varphi \to H\psi)$,

(GP): $\varphi \to GP\varphi$,

(HF): $\varphi \to HF\varphi$.

and the **rules of inference** (where $\vdash$ means ' **derivable** ', as usual):

(NEC$_G$): If $\vdash \varphi$ then $\vdash G\varphi$,

(NEC$_H$): If $\vdash \varphi$ then $\vdash H\varphi$,

added to the classical rule *Modus Ponens:* if $\vdash \varphi$ and $\vdash \varphi \to \psi$ then $\vdash \psi$.

Question: What do the axioms (GP) and (HF) say?

Some basic definitions related to derivability:

- A formula φ is a **theorem of** $\mathbf{K}_t$ if it is derivable from the axiomatic system above.
- A formula φ is **consistent** if $\nvdash \neg\varphi$, otherwise φ is **inconsistent**.
- A formula φ **follows deductively (in** $\mathbf{K}_t$**) from a set of formulae** Γ, resp. Γ **deductively implies (in** $\mathbf{K}_t$**)** φ, denoted $\Gamma \vdash_{\mathbf{K}_t} \varphi$ (or just $\Gamma \vdash \varphi$) if φ can be derived from all theorems of $\mathbf{K}_t$ and the formulae from Γ by applying the rule *Modus Ponens only.* In particular, $\emptyset \vdash \varphi$ iff $\vdash \varphi$.

A set of formulae Γ **is consistent (in** $\mathbf{K}_t$) if $\bot$ does not follow deductively from Γ; otherwise Γ **is inconsistent.**

The fundamental relationships between consistency and derivability:

$$\Gamma \cup \{\varphi\} \text{ is consistent iff } \Gamma \nvdash \neg\varphi$$

respectively,

$$\Gamma \vdash \varphi \text{ iff } \Gamma \cup \{\neg\varphi\} \text{ is inconsistent.}$$

2.2.2 Yet, what is temporal logic?

We will be talking about various temporal logics here. Before going further let us discuss briefly what we mean by a *temporal logic.* This question does not have a precise answer, and actually different people have quite different ideas about *what logic is at all.* Still, there are two traditional ways of specifying a logic: **semantic** and **deductive.** For both of them, first of all, a formal *logical language* must be fixed, with a precisely defined *syntax* and notion of *formula.*

The *semantic approach* is based on formal **logical semantics** associated with the language, i.e. notions of *semantic structure (model),* and *truth/validity of formulae in models.* Then, given a class of semantic structures $\mathfrak{C}$, the set of formulae true/valid in each structure from $\mathfrak{C}$ is called **the logic of the class** $\mathfrak{C}$. In our case, given a class of temporal frames $\mathfrak{C}$ (being those models of flows of time which satisfy the adopted ontological commitments or

technical assumptions) the set of temporal formulae valid in each frame of $\mathfrak{C}$ will be called **the temporal logic of the class** $\mathfrak{C}$.

A more general semantic approach is to identify a logic with the *logical consequence relation* determined by the semantics.

The *syntactic approach* is based on a **deductive system** specified on the set of formulae from the language, which is a set of formal *rules of inference/derivation of (sets of) formulae from (sets of) formulae.* A most common way of specifying a deductive system, particularly convenient for analyzing its metalogical properties is the *axiomatic style,* also known as Hilbert-style. Axiomatic systems, however are usually not suitable for *practical derivations* because of their non-structurality. Other, practically more suitable types of deductive systems are *sequent calculus, natural deduction, semantic tableau.*

In this course we will deal mainly with axiomatic systems, but will also discuss semantic tableaux, if time permits.

Once a deductive system is specified the corresponding logic is usually associated with its *set of theorems* i.e. derivable formulae. A more general approach is to identify the logic of the deductive system with the *deductive consequence relation* generated by it.

Thus, the two ways of specifying a logical system reflect the two major aspects of logic: *syntax and semantics.*

In a typical situations, both are present. The semantics gives the meaning of the logical formalism and the underlying notions of truth and validity, while the deductive system provides a formal method for generation of these validities. Then a major task is *to establish a perfect match between syntax and semantics, by showing that 'provable' and 'valid' coincide.* This is the essence of the **completeness theorem.**

2.2.3 Completeness of $\mathbf{K}_t$

The completeness theorem comes in two flavours: *weak* and *strong*, and each one can be formulated in two styles: *in terms of validity/provability and in terms of satisfiability/consistency.*

Weak completeness theorem, version I*: A modal formula* φ *is valid iff it is a theorem of* $\mathbf{K}_t$, *i.e.:*

$$\vDash \varphi \text{ iff } \vdash \varphi.$$

A note on the terminology: the direction from right to left is called *soundness*, or *correctness,* while the direction from left to right is often called itself *completeness;* while the equivalence is sometimes referred to as *adequacy.* Here by completeness we will mean the equivalence.

Weak completeness theorem, version II*: A modal formula* φ *is satisfiable iff it is consistent.*

Question: Which direction of this equivalence corresponds to soundness?

Exercise: Show the (weak) *soundness of* $\mathbf{K}_t$. Hint: induction on the derivation. Two key facts are needed. First, that *every instance of an axiom schema above is valid,* and second, that *the rules of inference preserve validity.*

Exercise: Derive versions I and II of the weak completeness theorem from each other.

Strong completeness theorem, version I*: A formula* φ *follows logically from a set of formulae* Γ *iff it follows deductively from* Γ *, i.e:*

$$\Gamma \vDash \varphi \text{ iff } \Gamma \vdash \varphi.$$

Strong completeness theorem, version II*: A set of formulae* Γ *is satisfiable iff it is consistent.*

Exercises: Derive versions I and II of the strong completeness theorem from each other.

Exercises: These below are essentially exercises in classical propositional logic. If you have already gone through that stuff, just ignore it. Otherwise you are advised to do them, although they do not have any direct bearing on the rest of the course.

- (*Deduction theorem)* $\Gamma \cup \{\varphi\} \vdash \psi$ iff $\Gamma \vdash \varphi \rightarrow \psi$. One direction is straightforward. For the other, use induction on the assumed derivation. Assume all classical propositional tautologies readily derived.
- Prove that $\varphi_1, ..., \varphi_n \vdash \psi$ iff $\vdash (\varphi_1 \wedge ... \wedge \varphi_n) \rightarrow \psi$. In particular, a set of formulae Γ is inconsistent iff $\vdash \neg(\varphi_1 \wedge ... \wedge \varphi_n)$ for some $\varphi_1, ..., \varphi_n \in \Gamma$.
- Prove the strong soundness: if $\Gamma \vdash \varphi$ then $\Gamma \vDash \varphi$.

Sketch of the proof of the strong completeness theorem, version II: Suppose we have already proved the soundness. For the other direction, given a consistent set Γ we want to construct a temporal model in which Γ is satisfiable. We will do more: *will construct* ***one*** *model that satisfies all consistent sets of formulae.* That will be the so called *canonical model.*

To construct it we shall build up on the idea of the completeness proof for **CL:** the worlds (instants) of that model will be *maximal consistent sets of formulae*, i.e. consistent sets which cannot be extended consistently. Here are some fundamental properties of maximal consistent sets (MCS):

A consistent set of formulae Γ is a MCS iff:

- for every formula φ, either $\varphi \in \Gamma$ or $\neg\varphi \in \Gamma$, iff
- for every formula φ, $\Gamma \cup \{\varphi\}$ is consistent iff $\varphi \in \Gamma$.

Also, if Γ is a MCS then:

- $\varphi \wedge \psi \in \Gamma$ iff $\varphi \in \Gamma$ and $\psi \in \Gamma$;

- $\varphi \vee \psi \in \Gamma$ iff $\varphi \in \Gamma$ or $\psi \in \Gamma$;

***Exercise**:* prove these.

The intuition behind MCSs is simple but fundamental. Take any instant in a temporal model. The set of formulae true at that instant is a MCS. Thus, MCSs serve as *complete descriptions of (what is true at) instants of a model.* This being the case, one can *identify* an instant with all that is true at that instant, i.e. with the MCS of formulae true there. We now take up this idea and construct the canonical model to consist of the set T^c of *all MCSs.* The leading intuition in this construction will be:

***The truth lemma**: A formula φ is true at an instant t of the canonical model exactly when $\varphi \in t$.*

That equivalence we can simply postulate for the atomic propositions by defining the *canonical valuation* V^c appropriately:

$$V^c(p) := \{t \in T^c \mid p \in t\}\,.$$

Due to the properties of MCSs mentioned above, this equivalence readily propagates through all Boolean connectives. Thus, we are just one step away from proving the truth lemma by induction on φ : we still need ensure that the equivalence propagates through the *temporal operators* . So far we are staying classical, i.e. *within* a MCS, but the temporal operators take us across the model, following the *precedence relation* between instants, which we are still to define. We should do that with the truth lemma in mind. A quick reflection on a real situation will help us. Take a model and two instants, s and t, such that $s \prec t$. Then, by definition *whenever a formula ψ is true at t, $F\psi$ must be true at s.* This can serve as a *definition* of the precedence relation in the canonical model:

$$s \prec^c t \text{ iff for every formula } \psi \in t, F\psi \in s.$$

***Exercise**:* Show that $s \prec^c t$ iff for every formula ψ, if $G\psi \in s$ then $\psi \in t$.

With a little technical work, using the axioms, one can show that the past relationships are as they should be.

***Exercise**:* Show that $s \prec^c t$ iff

for every formula ψ, if $\psi \in s$ then $P\psi \in t$, iff

for every formula ψ, if $H\psi \in t$ then $s \in t$

With this, the construction of the canonical model is completed and the truth lemma is easily established by induction.

***Exercise**:* Do the induction in detail.

To finish the completeness proof, one last fundamental fact is needed:

***Lindenbaum's lemma**: Every consistent set of formulae can be extended to a MCS.* To prove this, simply arrange all formulae of the language (countably many) in a sequence, pick

them one by one, and add them to the set if and only if that will preserve consistency. The eventual result will be a MCS. Indeed, inconsistency is a *finitary notion: if the resulting set is inconsistent, that should occur at some step of the construction*, which is prevented.

***Exercise**:* Work out the details of the proof of Lindenbaum's lemma.

Now, that we have this lemma at hand, we are ready to complete the proof: take a consistent set Γ and extend it to an MCS t. It is an instant of the canonical model $M^c = \langle T^c, \prec^c, V^c \rangle$. Then, by the truth lemma, $M^c, t \vDash \Gamma$.

QED.

Remark: This canonical model construction is *universal:* it can be applied to prove a general completeness result for any axiomatic extension of $\mathbf{K}_t$. This completeness, however, is interesting only when *the canonical frame belongs to the class of frames which is to be axiomatized by the logic.* Logics for which this is the case are called *canonical.* We will see many examples of canonical logics later, but the most interesting ones for this course, are *not canonical,* alas.

***References**:* For many more examples and details on completeness results in modal and temporal logic, see [Gol92], [HC96], [Pop92], [BdRVar].

2.2.4 On decidability and decision procedures. Finite model property and filtration for $\mathbf{K}_t$.

A logic is **decidable** if there is an algorithm (**decision procedure)** which decides whether a formula given as an input is a theorem of that logic or not.

Once we have a complete axiomatic system for the logic, half of this job is done: run the deductive machinery in a systematic way to produce all theorems, and wait to see the formula in the output. If *it is* a theorem, you will see it, sooner or later. The problem is: if it is not a theorem, then you will wait forever for nothing. So, we need a procedure that can tell when a formula is not a theorem. By the completeness theorem, that means the formula is not valid, i.e. the negation of that formula must be satisfiable. Checking validity or satisfiability however, is an infinite task again, because there are infinitely many models out there to check. Sometimes, there is nothing that can be done indeed, because the logic may *not* be decidable, as it is the case with first-order logic (Church's theorem).

In modal and temporal logic, however, decidability is often the case, and this is one of the reasons for them to be preferred to classical logic. And usually (but not always!), when the logic is decidable, it is because it has **the finite model property:** *every formula satisfiable in a model for the logic is satisfiable in a **finite** model for that logic.*

How does that help? Usually, given a finite temporal model it can be checked if it is a model for the logic in question. This is always the case when the logic is *finitely axiomatizable*, but not only. In this case, one can arranges all finite models (countably many) in a sequence, in some orderly fashion, and check them one by one: first if they are models for the logic, and if so, then if they satisfy the formula in question. Now, if the logic *has* the finite model property, and the formula is satisfiable, this will become known in finitely many steps.

Thus, running the two procedures in parallel: deriving all theorems, and checking satisfiability in all finite models, *provides a decision procedure for every complete and finitely axiomatizable logic with the finite model property.*

It should be noted immediately that the procedure described above is usually practically unfeasible and gives a much higher complexity than the best possible. However, sometimes proving finite model property is the only way of showing that a logic is decidable in principle, so we will describe briefly how that can be done using the most typical method for modal and temporal logics, viz. **filtration.**

Recall, the task is to show that every formula φ which is satisfiable in a model of the logic in question, in our case being $\mathbf{K}_t$, is satisfiable in a *finite model* of that logic. Clearly, the satisfiability of φ only depends on the truth of the (finitely many!) subformulae of φ across the model. Thus,, two instants in a model, which satisfy the same subformulae of φ are *indistinguishable* from the viewpoint of φ. Sometimes it is convenient to extend somewhat the set of subformulae to a still finite set of formulae called the (Fisher-Ladner) *closure of* φ, denoted by $cl(\varphi)$. (Fisher and Ladner proved finite model property via filtration of the Propositional Dynamic Logic PDL using such a closure.)

Thus, φ partitions the model into *finitely many equivalence classes* of instants satisfying the same subsets of $cl(\varphi)$ in each class. This is the underlying idea of the *filtration method: to collapse the infinite model to its finite quotient with respect to* φ. This method is not specified yet, until the *precedence relation* on the clusters is defined. For this definition there is a degree of freedom, which allows for adjusting the filtration in order to preserve the desired properties of the original model, such as transitivity, linearity, etc. However, there are two conditions which must be satisfied by the inherited precedence relation in the filtered model in order to preserve the satisfiability of φ, and they give respectively the lower and the upper bound for that relation.

Here are the formal details of the method of filtration in temporal logic.

Take any temporal model $M = \langle T, \prec, V\rangle$ and a *finite* set of formulae Γ.

Define an equivalence relation $\sim_\Gamma$ on M as follows:

$$s \sim_\Gamma t \text{ iff for every } \psi \in \Gamma, M, s \vDash \psi \Leftrightarrow M, t \vDash \psi.$$

Let s_Γ be the equivalence class of s with respect to $\sim_\Gamma$ and $T_\Gamma = \{s_\Gamma \mid s \in T\}$. Note that T_Γ is finite.

Now let $\prec_\Gamma$ be any relation on T_Γ satisfying the following conditions:

(MIN) For every $s, t \in T$, if $s \prec t$ then $s_\Gamma \prec_\Gamma t_\Gamma$.

(MAX) For every $s_\Gamma, t_\Gamma \in T_\Gamma$, if $s_\Gamma \prec_\Gamma t_\Gamma$ then:
for every $G\psi \in \Gamma$, if $M, s \vDash G\psi$ then $M, t \vDash \psi$, and likewise,
for every $H\psi \in \Gamma$, if $M, t \vDash H\psi$ then $M, s \vDash \psi$.

Finally, we define a *valuation* V_Γ in T_Γ as inherited from V: $V_\Gamma(p) = \{s_\Gamma \mid s \in V\}$.

The resulting model $M_\Gamma = \langle T_\Gamma, \prec_\Gamma, V_\Gamma \rangle$ is called a *filtered model* or a *collapse* of M *over* Γ.

The fundamental result (originally due to McKinsey who first applied the method in modal logic, further developed by Lemmon and Scott) is the following:

***Filtration lemma**: For every* $\psi \in \Gamma$ and $t \in T$, $M, t \vDash \psi$ iff $M_\Gamma, t_\Gamma \vDash \psi$.

The proof is by induction on ψ. For atomic propositions, this is guaranteed by the definition of V_Γ; for the Boolean connectives it is straightforward, and the steps for the temporal operators are taken care of by the conditions (MIN) and (MAX) for $\prec_\Gamma$.

***Exercise**:* Work out the details of the proof of the filtration lemma.

Before going further, and important question: *are there filtrations satisfying both (MIN) and (MAX) at all?* Yes. There is always the *minimal filtration* defined by converting (MIN) into an equivalence: $s, t \in T$, $s \prec t$ iff $s_\Gamma \prec_\Gamma t_\Gamma$. To be precise, that means $s_\Gamma \prec_\Gamma t_\Gamma$ iff $s' \prec t'$ for some $s' \sim_\Gamma s$ and $t' \sim_\Gamma t$.

Likewise, converting (MAX) into an equivalence defines *the maximal filtration:* For every $s_\Gamma, t_\Gamma \in T_\Gamma$, $s_\Gamma \prec_\Gamma t_\Gamma$ iff for every $G\psi \in \Gamma$, if $M, s \vDash G\psi$ then $M, t \vDash \psi$, and for every $H\psi \in \Gamma$, if $M, t \vDash H\psi$ then $M, s \vDash \psi$.

***Exercise**:* Verify that each of these satisfies both (MIN) and (MAX).

Clearly, every relation that is set-theoretically between the minimal and maximal filtration, is a filtration, too.

Now, let us apply the filtration lemma to $\Gamma = cl(\varphi)$, where $cl(\varphi)$ is any finite set of formulae containing φ and closed under subformulae. If $M, t \vDash \varphi$ then, by the filtration lemma $M_\Gamma, t_\Gamma \vDash \varphi$, i.e. φ is satisfied in a finite model.

QED

Thus, we have obtained the following important result:

***Theorem**:* $\mathbf{K}_t$ *has the finite model property, and hence is decidable.*

Remark: It is clear from the construction of the filtration that the size of the filtered model is exponential of the size of the formula φ. This suggests an *EXPSPACE* complexity provided by the filtration method. A more careful analysis, however, shows that *NEXPTIME* is sufficient, as it takes the right finite model can have a size exponential in the size of the formula so, once the guess is made it will take exponential time to check it.

***Remark**:* The *NEXPTIME* complexity of $\mathbf{K}_t$ provided by the filtration method is much higher than its actually complexity, which is *PSPACE.* This result can be obtained by introducing another, much more efficient and practical deduction method for temporal logic, called *semantic tableau.* For lack of time (and space) semantic tableau will not be presented here, but you can consult [BdRVar] for a general reference and [Gor99] for more details and further references.

***References**:* For more on decidability, finite model property and filtration in modal and temporal logics see [Gol92], [Pop92], [HC96], [BdRVar].

2.2.5 Temporal bisimulation

The last general topic to be discussed here is related to the *model theory* of temporal logic, and it will turn out to have an important bearing on the type of applications of temporal logic which will be discussed in this course.

When are two temporal models to be considered equivalent? A natural answer is: *when they are* ***logically equivalent, i.e.*** *satisfy the same temporal formulae.*

When are two labelled transition systems to be considered equivalent? Again, a natural answer: *when they are* ***computationally equivalent****, i.e. generate the same computations.*

As already noted, temporal models and labelled (mono)transition systems are essentially the same type of structures.

Then how different are these two criteria for equivalence? In other words, how close is the spirit of temporal logic to the models of computations provided by transition systems?

A partial answer will be suggested in this section.

The computational equivalence mentioned above essentially means that whatever computational step can be performed by one model, it can be 'simulated' by the other, and vice versa. This idea is at the heart of the notion of **bisimulation**, introduced in theory of processes by Park, and independently in modal and temporal logics by van Benthem (see [Ben84]) under the name ' *zig-zag relation*'. We shall first define the less-popular **bounded bisimulation.**

Definition*:* Let $M_1 = \langle T_1, \prec_1, V_1 \rangle$ and $M_2 = \langle T_2, \prec_2, V_2 \rangle$ be temporal models, and $x_1 \in T_1, x_2 \in T_2$. We define the property of the pairs (M_1, x_1) and (M_2, x_2) to be k-**bisimilar**, denoted $(M_1, x_1) \sim_k (M_2, x_2)$ inductively on $k \in N$ as follows:

(B1k) $(M_1, x_1) \sim_0 (M_2, x_2)$ iff (M_1, x_1) and (M_2, x_2) satisfy the same atomic propositions.

(B2k) $(M_1, x_1) \sim_{k+1} (M_2, x_2)$ if:

(forward): For every $y_1 \in M_1$ such that $x_1 \prec_1 y_1$ there exists a $y_2 \in M_2$ such that $x_2 \prec_2 y_2$ and $(M_1, y_1) \sim_k (M_2, y_2)$;

(backward): For every $y_1 \in M_1$ such that $y_1 \prec_1 x_1$ there exists a $y_2 \in M_2$ such that $y_2 \prec_2 x_2$ and $(M_1, y_1) \sim_k (M_2, y_2)$;

(Converses): Similarly for M_1 and M_2 exchanged.

Now, (M_1, x_1) and (M_2, x_2) are **finitely bisimilar,** denoted $(M_1, x_1) \sim_{fin} (M_2, x_2)$ if they are k -bisimilar for every k.

(A linguistic remark: 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.)

By **temporal rank** of a formula φ we will mean the greatest number $r(\varphi)$ of nested temporal operators in φ.

***Exercise**:* Give a precise inductive definition of $r(\varphi)$.

***Proposition**:*

1. $(M_1, x_1) \sim_k (M_2, x_2)$ iff (M_1, x_1) and (M_2, x_2) satisfy the same temporal formulae of temporal rank at most k.
2. $(M_1, x_1) \sim_{fin} (M_2, x_2)$ iff (M_1, x_1) and (M_2, x_2) satisfy the same temporal formulae.

***Exercise**:* Prove these. (By induction on k.)

In particular, for formulae with no atomic propositions, the condition (B1k) from the definition above becomes redundant.

Thus, the notion of finite bisimulation is strong enough to characterize temporal equivalence. However, it is not always strong enough to characterize computational equivalence, for it considers two states equivalent if they can simulate the *finite computations* of each other. There can be, however, *infinite computations* on which they differ.

***Exercise**:* Construct an example of two states from different transition systems, which are finitely bisimilar, yet they differ on the infinite computations beginning from them. (Hint: you can find that example in just about any text on bisimulations!)

Besides, the notion of finite bisimulation is *local,* i.e. it only applies to states in models. We would like to have a global notion, applying to models. So, a stronger notion of bisimulation is needed to characterize fully computational equivalence.

***Definition**:* Let $M_1 = \langle T_1, \prec_1, V_1 \rangle$ and $M_2 = \langle T_2, \prec_2, V_2 \rangle$ be temporal models. A relation $\beta \subseteq T_1 \times T_2$ is called a **bisimulation** between M_1 and M_2 if for every $x_1 \in T_1, x_2 \in T_2$ such that $x_1 \beta x_2$:

(B1) (M_1, x_1) and (M_2, x_2) satisfy the same atomic propositions.

(forth-future): For every $y_1 \in M_1$ such that $x_1 \prec_1 y_1$ there exists a $y_2 \in M_2$ such that $x_2 \prec_2 y_2$ and $y_1 \beta y_2$.

(back-future): Similarly for M_1 and M_2 exchanged.

(forth-past): For every $y_1 \in M_1$ such that $y_1 \prec_1 x_1$ there exists a $y_2 \in M_2$ such that $y_2 \prec_2 x_2$ and $y_1 \beta y_2$.

(back-past): Similarly for M_1 and M_2 exchanged.

***Exercise**:* Show that if β is a bisimulation between M_1 and M_2 and $x_1 \beta x_2$ then $(M_1, x_1) \sim_{fin} (M_2, x_2)$.

The example you were asked to produce above shows that not every finite bisimulation can be extended to a bisimulation.

Put simply, bisimulation is the infinitary version of finite bisimulation.

The notion of bisimulation is very important in theory of processes. It is instrumental in the model theory of temporal logic, too. Actually, it is the most general model-theoretic

construction on temporal models, and subsumes the basic constructions of *p-morphisms, generated submodels and disjoint unions* as particular cases. These will not be discussed here, but we shall give an example demonstrating how bisimulations can be used.

Example: Consider a temporal model $M = \langle T, \prec, V\rangle$ where $T = \{t_1, t_2, ...t_n, x, y\}$ and $\prec$ is defined as follows:

$t_1 \prec t_2 \prec ... \prec t_n \prec x \prec y \prec x$ hold, as well as all other relations that can be obtained from these by transitive closure, and nothing else. V is arbitrary. We shall construct a model $M' = \langle \mathbf{N}, <, V'\rangle$ which is bisimilar with M, this showing that the two models satisfy the same temporal formulae. the bisimulation β is defined as follows:

$t_k \beta k$, for $k = 1, .., n$, $x \beta t_{n+2m-1}$ and $y \beta t_{n+2m}$ for $m = 1, 2, 3, ...$

V' is defined accordingly to ensure the condition (B1): $V'(p) = \{k \mid s\beta k \text{ for some } \ s \in V(p)\}$.

Exercise: Show that β is a bisimulation between M and M'.

This construction is essentially part of a completeness proof for the linear time temporal logic **LPTL,** which will be introduced later.

One typical use of bisimulations in the theory of temporal logics is *to show that certain property of temporal models is not definable by means of a temporal formula.* The idea is simple.

For *local properties* (i.e. properties of states): if two states in models are finitely bisimilar, and one of them has the property $\mathfrak{P}$, while the other one does not, then it follows from the former proposition that $\mathfrak{P}$ is cannot definable by a temporal formula. Likewise, for *global properties* (of models), the full bisimulation can be used.

For instance, the example above shows that *irreflexivity* is not definable by a temporal formula, i.e. there is no temporal formula ψ such that ψ is true at an instant of a model iff that instant is $\prec$-irreflexive. Indeed, x and $n + 1$ are bisimilar, x is reflexive in its model, while $n + 1$ is not.

Exercise: Find a simpler example. involving a one-element and a two-element model to prove this. Also, show that *anti-symmetry* is not definable in the temporal language by an appropriate bisimulation.

Another property shown by the example above to be undefinable by a temporal formula is *finiteness.* Of course, two finite models cannot work here.

A final remark: *the notions of bisimulation are sensitive to the language.* The definitions given here correspond to the basic temporal language. For modal logic, the backwards conditions fall away. On the other hand, for extensions of the basic temporal logic with more operators, that will be introduced further, the definition notion of bisimulation needs respective adjustment. For more, see [KdR97].

With this, we complete our note on bisimulations. They will be put to work soon.

2.3 Axiomatic extensions of $\mathbf{K}_t$

2.3.1 Some important axioms

(TRAN): $G\varphi \rightarrow GG\varphi$.	(transitivity of $\prec$)
(BEG): $H\bot \vee PH\bot$.	(the time has a beginning)
(NOBEG): $P\top$.	(the time has no beginning)
(NOEND): $F\top$.	(the time has no end)
(LIN$_G$): $G(\varphi \wedge G\varphi \rightarrow \psi) \vee G(\psi \wedge G\psi \rightarrow \varphi)$	(forward linearity)
(LIN): $A\varphi \rightarrow GH\varphi \wedge HG\varphi$.	(linearity)
(DENSE): $GG\varphi \rightarrow G\varphi$	(density)
(DISCR$_G$): $F\varphi \wedge G(\varphi \rightarrow F\varphi) \rightarrow GF\varphi$	(implies discreteness backward (!))
(DISCR$_H$): $P\varphi \wedge H(\varphi \rightarrow P\varphi) \rightarrow HP\varphi$	(implies discreteness forward (!))
(WELLORD): $H(H\varphi \rightarrow \varphi) \rightarrow H\varphi$	(well ordering)
(COMPL): $A(H\varphi \rightarrow FH\varphi) \rightarrow (H\varphi \rightarrow G\varphi)$	(Dedekind completeness)

A linear ordering is called **Dedekind complete** if every non-empty and bounded above subset has a least upper bound. *Examples:* $\mathbf{N}, \mathbf{Z}, \mathbf{R}$; a *non-example:* $\mathbf{Q}$.

***Exercises**:*

1. Show for every temporal frame $\mathbf{T} = \langle T, \prec \rangle$ that:
   - $\mathbf{T}$ is transitive iff $\mathbf{T} \vDash$ *(TRAN)* iff $\mathbf{T} \vDash FF\varphi \rightarrow F\varphi$ iff $\mathbf{T} \vDash H\varphi \rightarrow HH\varphi$ iff $\mathbf{T} \vDash PP\varphi \rightarrow P\varphi$.
   - If $\mathbf{T}$ is a (strict) linear ordering then $\mathbf{T} \vDash$ *(TRAN)*$\wedge$ *(LIN);*
   - For every linear ordering $\mathbf{T}$, $\mathbf{T} \vDash$ *(DENSE)* iff $\mathbf{T}$ is dense.
   - * If $\mathbf{T} \vDash$ *(WELLORD)* then $\mathbf{T}$ is transitive.
   - * For every linear ordering $\mathbf{T}$, $\mathbf{T} \vDash$ *(WELLORD)* iff $\mathbf{T}$ is a well-ordering.
2. Show that:
   - $\langle \mathbf{N}, < \rangle \vDash$ *(DISCR$_G$).* What about *(DISCR$_H$)?*
   - $\langle \mathbf{Z}, < \rangle \vDash$ *(COMPL),* but $\langle \mathbf{Z}, < \rangle \nvDash$ *(WELLORD).*
   - * $\langle \mathbf{R}, < \rangle \vDash$ *(COMPL).*
   - * $\langle \mathbf{Q}, < \rangle \nvDash$ *(DISCR$_G$)* and $\langle \mathbf{Q}, < \rangle \nvDash$ *(COMPL)*
3. *Derive $H\varphi \rightarrow HH\varphi$ (the transitivity of H) in $\mathbf{K}_t$ + *(TRAN).*
4. *Derive $H\varphi \rightarrow HH\varphi$ in $\mathbf{K}_t$+ *(WELLORD).*

2.3.2 Temporal logics of linear flows of time

Theorem *The following axiomatic systems axiomatize completely the temporal logic of the indicated classes of frames:*

- $\mathbf{K}_t$+ *(TRAN):* transitive frames.
- $\mathbf{L}_t = \mathbf{K}_t$+ *(TRAN)*+ *(LIN):* strict linear orderings.
- $\mathbf{N}_t = \mathbf{L}_t$+ *(NOEND)*+ *(DISCR$_G$)*+ *(WELLORD):* $\langle \mathbf{N}, < \rangle$.
- $\mathbf{Z}_t = \mathbf{L}_t$+ *(NOBEG)*+ *(NOEND)*+ *(DISCR$_G$)*+ *(DISCR$_H$):* $\langle \mathbf{Z}, < \rangle$.
- $\mathbf{Q}_t = \mathbf{L}_t$+ *(NOBEG)*+ *(NOEND)*+ *(DENSE):* $\langle \mathbf{Q}, < \rangle$.
- $\mathbf{R}_t = \mathbf{L}_t$+ *(NOBEG)*+ *(NOEND)*+ *(DENSE)*+*(COMPL) :* $\langle \mathbf{R}, < \rangle$.

A brief sketch of proofs: We apply the canonical model construction. The first two logics are canonical, i.e. the canonical frame for the first one is transitive; for the second, every generated subframe of the canonical frame is a linear ordering.

For the rest, more work is needed. Next step is an appropriate filtration, and then the obtained finite model is expanded in an appropriate way to a bisimilar model on the desired frame.

For detailed proofs see [Gol92].

2.4 Adding nexttime

Consider a temporal frame $\mathbf{T} = \langle T, \prec \rangle$ and $s, t \in T$. The instant s is called an **immediate successor of** t, which we shall denote by $t \lhd s$, if s is a successor of t but not a successor to any successor of t. Formally that means:

$$t \lhd s \text{ iff } \langle T, \prec \rangle \Vdash t \prec s \wedge \neg \exists y (t \prec y \wedge y \prec s).$$

One can associate a new modal operator, ' *at every immediate successor',* or just, *nexttime* (or, *tomorrow*), denoted by X, with the relation $\lhd$, with the usual semantics:

$$M, t \vDash X\varphi \text{ iff } M, s \vDash \varphi \text{ for every } s \text{ such that } t \lhd s.$$

The dual of X will be denoted by N : $N\varphi := \neg X \neg \varphi$.

Exercise*:* Extend the standard translation ST for X.

As long as no assumptions are made on the flow of time, there is nothing to say about X, apart from the axiom K, in the basic temporal language, with no other modalities. Actually, its logic is just $\mathbf{K}$.

Question: Why?

Together with G and F, however, there are new validities.

Exercise: Show $\models G\varphi \to X(\varphi \wedge G\varphi)$.

If some assumptions on the flow of time are made, then more valid formulae emerge:

- In strict linear orderings the formula $N\varphi \to X\varphi$ saying that *every instant has at most one immediate successor* is valid.
- In forward discrete linear frames *every instant which has a successor, has a unique immediate successor:* $F\top \to (N\varphi \leftrightarrow X\varphi)$. Also, $X(\varphi \wedge G\varphi) \to G\varphi$ becomes valid, thus providing a *recursive definition* of G in terms of X (which will be of use later):

  (REC$_G$): $G\varphi \leftrightarrow X(\varphi \wedge G\varphi)$
- In *serial* forward discrete linear frames, *every instant has an immediate successor:* $N\varphi \leftrightarrow X\varphi$, i.e. X becomes *self-dual*, also expressed by

  (FUNC): $X\neg\varphi \leftrightarrow \neg X\varphi$.

  In such cases the operator X is also denoted by $\bigcirc$ (to emphasize its self-dual nature) and the unique immediate successor of any instant t is usually denoted by t'.

Actually, on linear frames the definition of an immediate successor can be re-phrased as follows;

$$t \lhd s \text{ iff } \quad \langle T, \prec\rangle \Vdash t \prec s \wedge \forall y(t \prec y \to s \preceq y)$$

(where $\preceq$ is the obvious abbreviation).

Furthermore, on $\langle \mathbf{N}, <\rangle$ one can formalize the *induction principle:*

$$\textit{(IND):}\ X\varphi \wedge G(\varphi \to X\varphi) \to G\varphi.$$

Theorem: In the temporal language with G, H and X :

- $\mathbf{L}_t(X) = \mathbf{L}_t + (K_X) +$ *(FUNC)* $+$ *(REC$_G$)* axiomatizes the temporal logic of discrete strict linear orderings;
- $\mathbf{N}_t(X) = \mathbf{N}_t + (K_X) +$ *(FUNC)* $+$ *(REC$_G$)* $+$ *(IND)* axiomatizes the temporal logic of $\langle \mathbf{N}, <\rangle$.

A set of formulae Γ is called **finitely satisfiable in a class of models $\mathfrak{C}$** if every finite subset of Γ is satisfiable in a model from $\mathfrak{C}$. A logic $\mathbf{L}$ is called **compact** if every set of formulae which is finitely satisfiable in a model of $\mathbf{L}$ is satisfiable in a model of $\mathbf{L}$.

Exercise: Show that the logic $\mathbf{N}_t(X)$ is *not compact.*

The past analogue Y of X, viz. previously, or yesterday can be defined and axiomatized likewise, see e.g. [LP00].

More ***references:*** [Gol92], [AGM+95], [GHR94].

2.5 Adding Since and Until

2.5.1 Semantics and expressiveness

The basic temporal language is not very expressive and there have been various proposals for extension of the basic language with more operators. A very successful and strongly expressive extension, particularly useful for applications to computer science, was proposed by H. Kamp in his doctoral dissertation [Kam68]. It extends the basic temporal language with the *binary temporal operators* S (Since) and U (Until) with the following semantics:

$M, t \vDash \varphi U \psi$ iff $M, s \vDash \psi$ for some $s \succeq t$ and $M, u \vDash \varphi$ for every u such that $t \preceq u \prec s$.

$M, t \vDash \varphi S \psi$ iff $M, s \vDash \psi$ for some $s \preceq t$ and $M, u \vDash \varphi$ for every u such that $t \succeq u \succ s$.

Exercise: Extend the standard translation ST for S and U.

Note that these definitions include the possibility of ψ holding at *the current instant,* and then nothing is required about φ. We shall call these the *reflexive* versions of U and S. Arguably, a more natural definition of U (resp. S) would be the one where the instant s is required to be *strictly in the future (resp. in the past) of* t. These are the *irreflexive, or strict* versions, which we shall denote by U^s and S^s.

The (strict) classical temporal operators can be expressed using these:

$F\varphi = \top U^s \varphi$, $P\varphi = \top S^s \varphi$.

Accordingly, U and S define the *reflexive versions* of F, G, P, H where the present moment is part of the future/past.

Note that U and S are definable in terms of U^s and S^s:

$$\varphi U \psi := \psi \vee \varphi U^s \psi, \varphi S \psi := \psi \vee \varphi S^s \psi.$$

Conversely, on discrete linear frames U^s can be defined in terms of U, *using* X :

$$\varphi U^s \psi = \varphi \wedge X(\varphi U \psi),$$

And likewise S^s can be defined from S using the past analogue of X.

Furthermore, U^s can be re-defined in a *very strict* form, not demanding φ to hold at the current point:

$M, t \vDash \varphi U^{vs} \psi$ iff $M, s \vDash \psi$ for some $s \succ t$ and $M, u \vDash \varphi$ for every u such that $t \prec u \prec s$.

U^s is expressible in terms of U^{vs} : $\varphi U^s \psi = \varphi \wedge \varphi U^{vs} \psi$. Moreover, X is expressible now, too: $X\varphi = \bot U^{vs} \varphi$. This formula says two things in one: that *there is an immediate successor* and that φ *holds there.* On the other hand U^{vs} is expressible in terms of U^s and X : $\varphi U^{vs} \psi = X(\psi \vee \varphi U^s \psi)$. Likewise one can introduce a very strict since S^{vs}.

When consulting the literature on U and S one should check which versions are adopted. For instance, in [GHR94] the strict versions are denoted by U and S and the reflexive ones by U^* and S^*.

Important note: Following the tradition in computer science *from now on we shall adopt the reflexive* U *and* S *as primitives and respectively the reflexive* F ,G, P, H *definable in terms of them.*

Sometimes, we shall refer to the strict versions as F^s, G^s, etc.

A number of *new operators* can be defined (at least) on discrete linear frames, as well:

- **unless (weak until)**: $\varphi W \psi = G\varphi \vee \varphi U \psi$.

  Exercises:

  1. Write down the formal semantics of W.
  2. Express F and U in terms of W.

- **before:** $\varphi B \psi$. Possible interpretations:

  (i) "φ *will occur before* ψ *which will occur, too*", or
  (ii) "φ *will occur before* ψ *which may or may not occur ever*", or
  (iii) " *if* ψ *ever occurs, then* φ *will occur before* ψ"

  Exercise: express each of these in terms of U.

Hint: One of the usual definitions of B is: $\varphi B \psi := \neg((\neg\varphi) U \psi)$. To which of the interpretations above does it correspond?

- **after:** likewise.

Exercise: Do it.

Theorem: (Kamp'68): *Every first order definable operator on the class of Dedekind complete linear ordering in expressible there in terms of* S^{vs} *and* U^{vs}.

Remark: Stavi has proposed two more operators, S' and U' which, added to S^{vs} and U^{vs} make the temporal language expressively complete on *all linear frames* (see e.g. [GHR94]). On the other hand, Gabbay has shown (*ibid)* that no finite number of new operators can make the temporal language functionally complete on *all* partial orderings.

2.5.2 Axioms for Since and Until

Burgess gives a complete axiomatic system of the Since-Until logic for *all linear flows of time* in [Bur82], further simplified by M. Xu in [Xu88]. It extends the classical propositional logic with the following axioms schemata (for the reflexive versions) and their duals, with S and U swapped:

- $G\varphi \to \varphi$,
- $G(\varphi \to \psi) \to \varphi U\chi \to \psi U\chi$,
- $G(\varphi \to \psi) \to \chi U\varphi \to \chi U\psi$,
- $\varphi \wedge \chi U\psi \to \chi U(\psi \wedge \chi S\varphi)$,
- $\varphi U\psi \to (\varphi \wedge \varphi U\psi)U\psi$,
- $\varphi U(\varphi \wedge \varphi U\psi) \to \varphi U\psi$;
- $\varphi U\psi \wedge \chi U\theta \to (\varphi \wedge \chi)U(\psi \wedge \theta) \vee (\varphi \wedge \chi)U(\psi \wedge \chi) \vee (\varphi \wedge \chi)U(\varphi \wedge \theta)$.

and the rules *(NEC_H)* and *(NEC_G).*

This axiomatization was extended by Y. Venema in [Ven93] to a complete axiomatic system for:

- *discrete linear orderings* by adding $F^s\top \to \bot U^{vs}\top$ and its dual $P^s\top \to \bot S^{vs}\top$;
- *well-orderings* by further adding $H^s\bot \vee P^sH^s\bot$ and $F^s\varphi \to (\neg\varphi)U^{vs}\varphi$,
- $\langle \mathbf{N}, < \rangle$ by $F^s\top$ to the previous system.

Other related results: Gabbay and Hodkinson in [GH91] axiomatize the Since-Until logic over the reals using Gabbay's *irreflexivity rule,* which has become a very handy tool for axiomatizations in modal and temporal logic; Reynolds has obtained such axiomatizations without using that rule for the reals in [Rey92] and for the integers in [Rey94]. The results of Venema mentioned above do not use that rule, either.

2.6 Hybrid temporal logics

So far we have been discussing the traditional hierarchy of temporal logics. There are numerous further developments however, which do not fit this hierarchy, and yet can be useful formalisms for applications to computer science. One of them is **hybrid logics** (not to be confused with ' *hybrid systems*') combine features of modal (temporal) and first-order logic, and actually blend together the language and metalanguage of modal logic. Hybrid languages are very expressive, while most of them are still quite tamed and the weaker ones even preserve the complexity of the basic temporal logic.

There are several syntactic mechanisms of hybrid logics which extend the classical temporal framework, and we briefly mention them approximately in order if increasing expressiveness.

- *universal modality* A , interpreted by the universal relation of the Kripke frame. The main effect of its presence in the language is that one can express global facts such as *validity in a model* locally, at an instant of that model: for any $t \in M$, $M, t \vDash A\varphi$ iff $M \vDash A\varphi$. The use of the universal modality has been studied in detail in [GP92].

- *nominals,* or *clock variables.* This is a special sort of atomic propositions which are bound to be true at *exactly one instant of the model*, i.e. one can think of a clock variable saying ' *It is t o'clock'.* The idea of nominals goes back to Prior [Pri67] and Bull, but has been systematically developed and applied in [PT91], [Bla93], [GG93].

- *satisfaction operator* $@_i$, where i is a nominal, with semantics: $M, t \vDash @_i\varphi$ iff $M, V(i) \vDash \varphi$ where $V(i)$ is the unique instant where i is true. The effect is that *the basic semantic notion of a truth at an instant of a model is now brought into the syntax.* Besides increasing the expressiveness, this also allows for an elegant development of proof systems such as sequent calculus and semantic tableau in the style of labelled deduction [Bla00]. The satisfaction operators have various precursors, but they were introduced in this form by Blackburn and Seligman, see [BS98].

- *difference modality* $D\varphi$ saying that φ *is true at some other instant.* The expressiveness of the resulting language is in a sense equivalent to the one with nominals and A [GG93] . This operator has been mentioned in various works, but the first systematic study is [dR92].

- *reference pointers*, or *binders*: $\downarrow_i$where i is a nominal. These behave syntactically as quantifiers, and the semantic effect is that $\downarrow_i$ *binds the value of the nominal* i *to the current instant of evaluation.* Formally, $M, t \vDash \downarrow_i \varphi$ iff $M_{[i \to t]}, t \vDash \varphi$ where $M_{[i \to t]}$ is the model obtained from M by re-assigning the denotation of i to be t. The temporal logics with reference pointers were introduced in [Gor94], and further studied in [Gor96], [BS98], [ABMar].

- *quantifiers over nominals*: $\forall i\varphi$, where i is a nominal. The semantics should be clear: $M, t \vDash \forall i\varphi$ iff $M_{[i \to s]}, t \vDash \varphi$ for any instant $s \in M$. These bring the full power of first-order quantification to the temporal language, while still preserving many of its propositional virtues. The idea goes back to Bull, and for demonstration of their use see [PT91].

- *propositional quantifiers:* $\forall p\varphi$ where p is an atomic proposition. This is already a second-order quantification brought into the propositional language. Towards the end of these notes we will se an evidence of its expressiveness.

Hybrid languages are quite more expressive than the purely temporal ones. Just two examples:

- *Irreflexivity* of the precedence relation, which is not expressible even with the (reflexive) U, is straightforward in a language with nominals and $@ : @_i G\neg i$.

- The operators S and U (in any variation) are definable in the hybrid language with nominals and binders: $\varphi U\psi :=\downarrow_i F(\psi \wedge H(Pi \to \varphi))$ and likewise for S.

For more on hybrid temporal logics see [BS98], [BT99] and on their complexity, see [ABMar].

2.7 Recapping: classical vs computer science oriented temporal logics

In summary, there are two different approaches to representing time and temporality and reasoning about them.

- The **classical temporal logic approach** is based on the *physical concept of time* formally represented by structures $\langle M, \prec \rangle$. These are just Kripke frames where M is a set of 'instants' constituting the *flow of time*, and $\prec$ is the *precedence relation* between instants, usually assumed transitive, actually a partial ordering, often linear, but not necessarily discrete, nor having a beginning etc. Classical temporal logics are built on these structures by interpreting the temporal language, containing the basic past and future operators, but also possibly X, U, S, and others, by means of Kripke semantics.
- The **temporal logics in computer science approach** is based on the *computational concept of time* considered as a discrete succession of states of a computational process or device, effected by a family of actions or transitions. The formal models for this concept of time are *transition systems*,which (in their simplest form) are again Kripke frames $\langle S, R \rangle$ and the semantics is Kripke semantics. The major technical difference in this approach is that R represents *immediate transitions rather than precedence between states* . In the classical framework that corresponds to R being a *successor relation* between consecutive instants in a discrete flow of time, associated with the nexttime operator X, and the precedence relation can be obtained by taking the (reflexive and) *transitive closure* of R, which interprets G and F. Furthermore, usually only the future operators are used in these logics, although there have been increasing calls for extensions with the past modalities.

3 The linear time propositional temporal logic LPTL

3.1 Syntax and semantics

As already discussed, a computation can be thought as a sequence of sets of atomic propositions describing the successive states of the execution effecting the computation. Formally, these are Kripke models over a linear flow of time represented by $\langle \mathbf{N}, < \rangle$.

Here we shall present a natural and useful temporal logic to reason about computations. It was proposed by Pnueli in his seminal paper [Pnu77], axiomatized and studied in [GPSS80] and [MP81], and since then in many more works, and has become the most well-known temporal logic used in computer science nowadays.

3.1.1 Language and syntax

The language of **LPTL** is a propositional language containing a set of atomic propositions **AP,** the Boolean constant $\bot$ and connective $\rightarrow$, and the temporal operators U and X.

The formulae of **LPTL** are defined recursively by

$$\varphi := p \mid \bot \mid \varphi_1 \rightarrow \varphi_2 \mid X\varphi \mid \varphi_1 U \varphi_2$$

The remaining Boolean connectives $\top, \vee, \wedge, \leftrightarrow$ and the operators G and F are defined as usual.

Other important definable expressions in **LPTL**:

- φ *will hold almost always:* $FG\varphi$, often denoted as G^{∞};
- φ *will hold infinitely often:* $GF\varphi$, also denoted by F^{∞}.

3.1.2 Semantics

A *linear temporal model* is a tuple $M = \langle S, R, L, \sigma \rangle$ where $\mathcal{T} = \langle S, R, L \rangle$ is a Kripke model, i.e. a labelled transition system with one transition relation R, which is assumed *serial,* and $\sigma : \mathbf{N} \rightarrow S$ is a run on $\mathcal{T}$.

The truth definition essentially follows the semantics of temporal logic as defined earlier, with the implicit assumptions that *the time flow is restricted on the run* $\sigma = s_0, s_1, s_2, ...$ *and that the formulae are evaluated at a state from the run.* More precisely, $M, i \models \varphi$ will mean that φ *is true at the state* s_i *in the model* M.

- $M, i \models p$ if $p \in L(s_i)$ for $p \in \mathbf{AP}$;
- $M, i \not\models \bot$;
- $M, i \models \varphi \rightarrow \psi$ if $M, i \models \varphi$ implies $M, i \models \psi$;
- $M, i \models X\varphi$ if $M, i+1 \models \varphi$;
- $M, i \models \varphi U \psi$, if $M, j \models \psi$ for some $j \geq i$ and $M, k \models \varphi$ for every k such that $i \leq k < j$.

Now, we say that:

- φ is **valid** in M, denoted $M \models \varphi$, if $M, 0 \models \varphi$. Then M is called a **model for** φ.
- φ is **valid**, denoted $\models \varphi$, if φ is valid in every linear temporal model.
- φ is **satisfiable** if it is true at some state of some linear temporal model.

Exercises: Show that φ is satisfiable iff it is *valid in some model* iff $\neg\varphi$ *is not valid.*

Remark: This is the *anchored version* of the semantics for **LPTL,** which deals with *initial* satisfiablility/validity. Alternatively, *global* satisfiablility / validity can be defined.

***Exercise**:* Introduce global satisfiablility/validity and investigate their relationships with the initial versions. Are these really different notions? See [Eme90], [LP00].

Formulae ψ and θ are **equivalent** if $\psi \leftrightarrow \theta$ is valid.

***Exercise**:* Check the following selection of useful equivalences in **LPTL.**

- $G^\infty \neg\varphi \equiv \neg F^\infty \varphi$;
- $G^\infty G^\infty \varphi \equiv G^\infty \varphi$;
- $F^\infty F^\infty \varphi \equiv F^\infty \varphi$;
- $F^\infty(\varphi \vee \psi) \equiv F^\infty \varphi \vee F^\infty \psi$;
- $G^\infty(\varphi \wedge \psi) \equiv G^\infty \varphi \wedge G^\infty \psi$;

For many more like these, see [Eme90].

***Exercise**:* check the validity of the following formulae:

- $G^\infty \varphi \rightarrow F^\infty \varphi$;
- $G\varphi \wedge F\psi \rightarrow \varphi U \psi$;
- $G\varphi \leftrightarrow \varphi \wedge XG\varphi$;
- $\varphi \wedge G(\varphi \rightarrow X\varphi) \rightarrow G\varphi$;
- $\varphi \wedge G(\varphi \rightarrow F\varphi) \rightarrow GF\varphi$;
- $(\varphi \rightarrow \chi)U\psi \wedge \varphi U \psi \rightarrow \chi U \psi$;
- $\varphi U \psi \leftrightarrow (\psi \vee (\varphi \wedge X(\varphi U \psi)))$;
- $\psi U(\varphi \vee \chi) \leftrightarrow (\psi U \varphi \vee \psi U \chi)$;
- $G((\psi \vee (\varphi \wedge X\theta)) \rightarrow \theta) \rightarrow (\varphi U \psi \rightarrow \theta)$.

***Exercise**:* Show that the following formulae are *not valid*, by constructing appropriate counter-models:

- $F^\infty \varphi \rightarrow G^\infty \varphi$;
- $F^\infty(\varphi \wedge \psi) \equiv F^\infty \varphi \wedge F^\infty \psi$; (But one of the implications holds. Which one?)
- $G^\infty(\varphi \vee \psi) \equiv G^\infty \varphi \vee G^\infty \psi$; (Likewise.)
- $\psi U(\varphi \rightarrow \chi) \wedge \psi U \varphi \rightarrow \psi U \chi$;
- $G(\theta \rightarrow (\psi \vee (\varphi \wedge X\theta))) \rightarrow (\theta \rightarrow \varphi U \psi)$.

3.1.3 Run-based semantics for LPTL

The semantics of **LPTL** can be redefined in a way closer to the spirit of branching time logics, by *evaluating formulae relative to runs,* rather than states as follows:

Given a run $\sigma = s_0, s_1, s_2, ..$ we define for every $n \in \mathbf{N}$, σ^n to be the **suffix run** $s_n, s_{n+1}, s_{n+2}, ...$ .Thus, $\sigma^n(m) = \sigma(n+m)$.

Now we take the run outside of the definition of a model and define **truth of a formula in a model** $M = \langle S, R, L\rangle$ **on a run** σ, denoted $M, \sigma \models^r \varphi$, recursively as follows:

- $M, \sigma \models^r p$ if $p \in L(\sigma(0))$ for $p \in \mathbf{AP}$;
- $M, \sigma \not\models^r \bot$;
- $M, \sigma \models^r \varphi \to \psi$ if $M, \sigma \models^r \varphi$ implies $M, \sigma \models^r \psi$;
- $M, \sigma \models^r X\varphi$ if $M, \sigma^1 \models^r \varphi$;
- $M, \sigma \models^r \varphi U \psi$, if $M, \sigma^j \models^r \psi$ for some $j \geq 0$ and $M, \sigma^k \models^r \varphi$ for every k such that $0 \leq k < j$.

***Exercise**:* Show that the two semantics are equivalent in the following sense:

$$\langle S, R, L, \sigma\rangle, i \models \varphi \text{ iff } \langle S, R, L\rangle, \sigma^i \models^r \varphi.$$

3.1.4 Canonical representation of runs and computations.

Two runs σ_1 and σ_2 in labelled transition systems resp. $\langle S_1, R_1, L_1\rangle$ and $\langle S_2, R_2, L_2\rangle$ will be called **computationally equivalent,** denoted $\sigma_1 \approx \sigma_2$ if they generated the same computations, i.e. $L_1(\sigma_1(n)) = L_2(\sigma_2(n))$ for every n.

***Proposition**: Let* σ_1 *and* σ_2 be *runs in labelled transition systems resp.* $\langle S_1, R_1, L_1\rangle$ *and* $\langle S_2, R_2, L_2\rangle$. *Then* $\sigma_1 \approx \sigma_2$ *iff* $\langle S_1, R_1, L_1, \sigma_1\rangle$ *and* $\langle S_2, R_2, L_2, \sigma_2\rangle$ *satisfy the same* **LPTL** *-formulae.*

Proof: ***exercise**.*

Remark: Note that in the absence of nexttime operator X, the 'if' direction of the above statement does not hold anymore. Furthermore, in such cases there is a more natural notion of equivalence, proposed by Lamport, which ignores 'stuttering' i. e. transitions where nothing happens

***Exercise**:* Give an example in a language with U but without X of two runs which are not computationally equivalent, yet satisfying the same formulae.

The semantics of **LPTL** can be given in terms of *abstract runs and computations, i.e.* on the transition system $\langle \mathbf{N}, \mathbf{succ}\rangle$.

***Proposition**: For every run σ in a labelled transition system $\langle S, R, L\rangle$ there is a labelling L_σ on $\langle \mathbf{N}, \mathbf{succ}\rangle$ such that σ is computationally equivalent to the run $0, 1, 2, ...$ in $\langle \mathbf{N}, \mathbf{succ}, L_\sigma\rangle$.*

*Proof: **exercise**.*

Thus, models for **LPTL** can be *canonically represented* as labellings on $\langle \mathbf{N}, \mathbf{succ}\rangle$ i.e. mappings ξ: $\mathbf{N} \to \mathbf{2^{AP}}$, which we will call *abstract computations.* This representation will be more convenient later when the relationship between temporal logic and automata is discussed.)

3.2 Using LPTL to express properties of computations

- **Safety:**

$$G\neg(\mathit{The\text{-}bad\text{-}thing})$$

  e.g.: "Two trains will never be in the tunnel at the same time":

$$G\neg(T1 \neq T2 \wedge \mathit{InTunnel}(T1) \wedge \mathit{InTunnel}(T2)).$$

- **Partial correctness** *of a post-condition ψ with respect to a pre-condition φ :*

$$\varphi \to G(\ \mathit{terminal} \to \psi),$$

  where *terminal* is a proposition which holds only at terminal states.

- **Liveness:**

$$F(\mathit{The\text{-}good\text{-}thing})$$

- **Total correctness** *of a post-condition ψ with respect to a pre-condition φ :*

$$\varphi \to F(\ \mathit{terminal} \wedge \psi),$$

- **Fairness (responsiveness).** Here are a few versions of fairness:

  (i) very **weak fairness**: *every continuous request (ρ) is eventually granted (τ)*:

$$G\rho \to F\tau.$$

  (ii) somewhat stronger (**justice**): *if a request (ρ) holds almost always then it is eventually granted (τ)*:

$$FG\rho \to F\tau.$$

  (iii) **strong fairness**: *if a request (ρ) is repeated infinitely often then it is eventually granted (τ)*:

$$GF\rho \to F\tau.$$

  Question: Is (ii) really stronger than (i)? is (iii) stronger than (ii)?

(iv) **impartiality: '** *Every process will be scheduled infinitely often'.*

(v) What about: *'First come, first served',* or *'Once requested, eventually granted'?* Are these reasonable notions of fairness? Why?

- **Precedence:**
  - *The event* φ *will occur before the event* ψ *, which may or may not occur at all:*

$$(\neg\psi)U(\varphi \wedge \neg\psi)$$

  - *The event* φ *will occur before the event* ψ *, which will occur, too:*

$$(\neg\psi)U(\varphi \wedge \neg\psi) \wedge F\psi$$

Exercise*:* Formalize the following specifications in **LPTL.**

- *The event* φ *will not occur before the event* ψ *which may or may not occur at all.*
- *The event* φ *will not occur before the event* ψ *which will occur.*
- *Every occurrence of the event* ψ *is preceded by an occurrence of the event* φ*, after the previous occurrence of* ψ*, if any.*

  Question: can this specification be simplified to: " *Between every two successive occurrences of the event* ψ *there is an occurrence of the event* φ*.*"?
- *Every time when a message is sent, it will not be marked as 'sent' before an acknowledgment of receipt is returned.*

Finally, a **non-example,** proposed by P. Wolper [Wol83] **:** the property " *The event* φ *will be true at least at every even step of the computation.*" is *not* expressible in **LPTL.** We will come back to this later.

References*:* There have been convincing calls see (e.g.[LPZ85]) for bringing the 'glory of the past' into **LPTL,** by adding the past operators, For axiomatization, semantic tableau, and examples of use of LPTL extended with the past operators previously and since see the recent [LP00].

For more on fairness: see [GPSS80], [Lam80], [LPS81].

For more on properties specified by temporal logic, see [Sis94] and for a detailed study of their hierarchy see [MP90] .

3.3 Axiomatic system for LPTL

Axiom schemata:

The following axiomatic system is a streamlined version of the future part of the Since-Until logic over $\langle \mathbf{N}, < \rangle$. Traditionally it is presented with a longer set of axioms (see e.g. [Sti92]), which are listed as theorems below.

(CL): *Enough propositional axiom schemata;*

(K_G: $G(\varphi \to \psi) \to (G\varphi \to G\psi)$,

(K_X): $X(\varphi \to \psi) \to (X\varphi \to X\psi)$,

(FUNC): $X\neg\varphi \leftrightarrow \neg X\varphi$,

(REC_U) $\varphi U\psi \leftrightarrow (\psi \vee (\varphi \wedge X(\varphi U\psi)))$.

(LFP_U): $G((\psi \vee (\varphi \wedge X\theta)) \to \theta) \to (\varphi U\psi \to \theta)$.

Rules:

(MP) and (NEC_G).

Note the last two axiom schemata, we'll come back to them later.

***Exercise**:*

1. Write down the recursive formulae for $F\varphi$ and $G\varphi$, following from *(REC_U).*
2. Likewise for F^s, G^s, U^s.
3. Write down recursive formulae for $\varphi W\psi$ and $\varphi B\psi$, analogous to *(REC_U).*

***Exercise**:* Show that:

1. Every axiom of **LPTL** is valid.
2. The following *induction rule* is derivable in **LPTL:**

   $$\text{If} \quad \vdash \varphi \to X\psi \text{ then} \quad \vdash \varphi \to G\psi.$$

   (Hint: use the induction principle *(IND)* listed below).
3. The axiom *(REC_U)* can be weakened to $(\psi \vee (\varphi \wedge X(\varphi U\psi))) \to \varphi U\psi$, as the other implication is derivable.
   (Hint: use *(LFP_U).*)
4. The following formulae are theorems of **LPTL:**
   - *(REC_G):* $G\varphi \leftrightarrow (\varphi \wedge XG\varphi)$ (Hint: this is a particular case of *(REC_U).*)
   - *(IND):* $\varphi \wedge G(\varphi \to X\varphi) \to G\varphi$.
     (Hint: express $G\varphi$ in terms of U and use *(LFP_U.)*
   - $\varphi U\psi \to F\psi$. (Hint: use *(LFP_U)* for $\theta = F\psi$.)
   - $G\varphi \to GG\varphi$;
   - $XG\varphi \leftrightarrow GX\varphi$;
   - $FG\varphi \to GF\varphi$

- $G\varphi \wedge F\psi \rightarrow \varphi U\psi$;
- $\varphi \wedge G(\varphi \rightarrow F\varphi) \rightarrow GF\varphi$;
- $(\varphi \rightarrow \chi)U\psi \wedge \varphi U\psi \rightarrow \chi U\psi$;
- $\varphi U\psi \rightarrow \varphi U(\varphi U\psi)$;

Theorem. *The axiomatic system for* **LPTL** *is sound and (weakly) complete i.e. a formula is* ***LPTL****-consistent iff it is satisfiable in a linear temporal model.*

Proof (sketch) There are various completeness proofs for **LPTL** in the literature, most of them based on semantic tableau constructions. See e.g. [GPSS80], [MP81], [GHR94], [LP00].

Here is a sketch of a traditional modal completeness proof, for details on which see [Gol92].

- The soundness is straightforward, as usual.
- For the completeness, take a generated canonical model satisfying given formula φ. It is a reflexive, transitive, linear, functional w.r.t. the relation R_X corresponding to X, the transitive closure of which is included in R_G etc., but it is *not* the natural numbers.
- Do *filtration* of that canonical model over the *closure of* φ (to be defined later). This is basically the set of subformulae of φ, slightly extended, but still finite.
- The result is a *finite* model (so it is not yet **N**) satisfying φ. It looks like a *balloon*, with a tail representing an initial segment of **N** and ending with a cluster of points related to each other.
- After some technical work one can show that this balloon can be *unwound into a bisimilar infinite sequence of points* i.e. a copy of **N**, ensuring that every point from the head of the balloon appears infinitely many times in the unwinding. this will take care of the satisfaction of the eventualities and, combined with the bisimulation, will guarantee satisfaction of φ. *QED.*

Corollary: **LPTL** *has the finite model property, hence it is decidable.*

The complexity of the decision procedure provided by the filtration argument is *NEXP-TIME,* but this is not the last word on the topic. The following result can be proved by an appropriate semantic tableau construction, and also by reduction to automata, which will be done later (for the upper bound), and by reduction from polynomial space bounded Turing machines (for the lower bound). See [SC85a] and [Eme90] for more details.

Theorem(Sistla and Clarke' 82) *The satisfiability problem for* **LPTL** *is PSPACE-complete.*

We shall not present semantic tableau for LPTL here, but the transformation of formulae into automata which will be discussed later will give a good idea of how to construct a tableau.

References: For axiomatization of **LPTL** with past operators and survey on decidability and completeness results on **LPTL** see [LP00]. For a semantic tableau for the subsystem with X and G only, see [BA93]. For the general case, see [Eme90], [Wol85] and for a survey on tableaux in modal and temporal logic, see [Gor99].

For the first-order extension of **LPTL** see [GW89], [Eme90].

4 Branching time temporal logics

While **LPTL** is used for reasoning about *single computations*, richer semantics and more expressive languages and logics are needed to reason about *all possible computations* of non-deterministic or concurrent processes. Since every state in a transition system has several possible successors, the structure representing all possible executions will be *tree-like* rather than linear.

Although the philosophical roots of the underlying *Ockhamist semantics* for branching-time logics go back to the Middle ages, and its use in temporal logic was already discussed in [Pri67] (see also [Zan96]) the first ideas of using branching time logics in computer science are attributed to Abrahamson, [Abr79], further developed by [BAPM81], [CE81], [EH86] etc. See the survey [Eme90] for more details and references.

4.1 Ockhamist semantics for classical temporal logic

This section describes the underlying semantics of branching time from the viewpoint of classical temporal logic. It is useful for the understanding of the semantics of branching time logics but, apart from some basic definitions related to trees, it is not necessary for the further reading.

Just like the semantics of **LPTL** is a special case of Prior's semantics over linear flows of time, the semantics of branching time logics is a particular case of *Ockhamist semantics* in temporal logic (see [Pri67], [Zan96]). Unlike the classical Prior's semantics which is essentially indifferent to the structure of the flow of time and possible futures determined by it, the Ockhamist semantics is based on the assumptions that while the past is determined and cannot be changed, *the future is non-deterministic and can take different possible courses from the present moment; however, at every moment there is one possible future which is considered* ***actual.*** Therefore, statements about future events are relative to that course of future events which is considered actual at the current moment. Thus, the phrases *'always/sometime in the future'* now have different meaning, viz: *'always/sometime in the considered actual future'.* Furthermore, that actual future is determined, i.e. linear.

Formally, this means first, that the natural flows of time for Ockhamist semantics are *tree-like*, rather than linear, and second, that the temporal formulae are now evaluated *not relative to an instant, but to a future linear branch leading from the current instant.*

Not all possible branches in a tree are needed in order to give reasonable Ockhamist semantics. It is sufficient to consider *rich enough* families of branches, which will be called a **bundles,** and the Ockhamist semantics can be defined on such bundles, considering the

branches in them as *primitive entities.*

The language of the propositional Ockhamist temporal logic contains, besides the temporal modalities F, P, also a modality $\Diamond$ for a *possible future branch passing through the current instant.* The formulae are defined accordingly:

$$\varphi = \bot \mid p \mid \varphi \to \varphi \mid P\varphi \mid F\varphi \mid \Diamond\varphi.$$

In order to introduce the semantics over bundles, we need some formal definitions.

- A **tree** is a strict partial ordering $\langle M, < \rangle$, such that every **node** $x \in M$ has a linear set of $<$ -predecessors.. The least element of a tree, if it exists, is called the **root**.
- A **path** in a tree $T = (M, <)$ is a maximal linearly ordered set of nodes.
- A **branch** in T is any set of nodes of the type $\{y | y \in \pi \text{ and } x \leq y\}$ for fixed path π and $x \in \pi$. The least node x of a branch b is the **initial node** of b, denoted by $I(b)$, and b is said to be **stemming from** x.
- Let b and c be branches. If they have the same initial node, that will be denoted by $b \nabla c$. If $b \subseteq c$, then b is called a **sub-branch** of c, denoted $c \trianglelefteq b$, resp. $b \trianglerighteq c$, and c is an **extension** (NB: towards the root!) of b. Note that $\trianglelefteq$ is a partial ordering on branches.

The set of all branches in a tree T will be denoted by $\mathbf{B}(T)$.

- A **bundled tree** (see [Bur79]) is a pair $(T, \mathcal{B})$ where T is a tree and $\mathcal{B}$ is a non-empty set of branches (called a **bundle)** in T, closed under sub-branches and extensions and such that every node of T belongs to some branch from $\mathcal{B}$. A **complete bundled tree** is a bundled tree of the type $(T, \mathbf{B}(T))$. Thus, a bundle consists of all branches along paths covering the nodes of the tree.

A **valuation** on a bundle tree $\mathcal{T} = (T, \mathcal{B})$ is any function $V : (BN \cup PV) \to \mathcal{P}(\mathcal{B})$ such that for every $b \in BN$, $V(b)$ is a branch from $\mathcal{B}$. A **model** is a pair $(\mathcal{T}, V)$ where $\mathcal{T}$ is a bundle tree and V is a valuation in $\mathcal{T}$. A model on a complete tree will be called a *complete model.*

Now, the definition of **(Ockhamist) truth of a formula in a model** $\mathcal{M} = (M, <)$ **relative to a branch** b **in that model** is in traditional Kripke style:

- $\mathcal{M}, b \models p$ iff $b \in V(p)$, for any $p \in PV$;
- $\mathcal{M}, b \not\models \bot$;
- $\mathcal{M}, b \models \phi \to \psi$ iff $\mathcal{M}, b \models \phi$ implies $\mathcal{M}, b \models \psi$;
- $\mathcal{M}, b \models F\phi$ iff $\mathcal{M}, c \models \phi$ for some $c \trianglerighteq b$;

- $\mathcal{M}, b \models P\phi$ iff $\mathcal{M}, c \models \phi$ for some $c \trianglelefteq b$;
- $\mathcal{M}, b \models \Diamond\phi$ iff $\mathcal{M}, c \models \phi$ for some $c \nabla b$.

Bundled trees are technically equivalent to *Ockhamist frames* studied by Zanardo in [Zan96], which can be alternatively used to introduce Ockhamist semantics.

Complete finitary axiomatizations have been obtained for the general bundled tree semantics, see Zanardo's paper for results and further references on the logic for the basic language, and [BG99] for the Ockhamist temporal logic extended with a modality for an alternative branch and a special sort of 'fan-variables' representing all branches stemming from one instant.

Despite the seeming simplicity of the Ockhamist semantics on complete bundled trees, however, no complete axiomatization for it has been found so far. This situation is similar to the problem of finitary axiomatization of CTL*, the branching time logic coming up now.

4.2 The full computation tree logic CTL*

The logic **CTL*** was introduced by Emerson and Halpern in 1983 (see [EH86]) as an extension of the previously studied *computation tree logic* **CTL** (to be defined later) and a unifying approach to linear and branching time logics.

4.2.1 Language and syntax

The language of **CTL*** extends the one of **LPTL** with *path quantifiers:*

- $\forall\varphi$, meaning *"φ is true on **every** execution passing through the current state",* and
- its dual $\exists\varphi$, meaning *"φ is true on **some** execution passing through the current state",*

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

$$\varphi := p \mid \bot \mid \varphi_1 \to \varphi_2 \mid X\varphi \mid \varphi_1 U \varphi_2 \mid \forall\varphi$$

The operators $\varphi \wedge \psi$, $\varphi \vee \psi$, $\varphi \leftrightarrow \psi$, $F\varphi$, $G\varphi$ are definable as usual, as well as the 'macros' G^∞ and F^∞. Also defined is $\exists\varphi := \neg\forall\neg\varphi$.

For some purposes it is convenient to distinguish two types of **CTL***−formulae: **state formulae,** that are evaluated relative to states, and **path formulae**, evaluated relative to runs. The sets *SFOR* of state formulae and *PFOR* of path formulae are defined by mutual induction as follows.

SFOR:

- All atomic propositions and $\perp$ are in *SFOR.*
- If $\varphi, \psi \in$ *SFOR* then $\varphi \rightarrow \psi \in$ *SFOR.*
- If $\varphi \in$ *PFOR,* then $\forall\varphi \in$ *SFOR.*

PFOR:

- Every state formula is a path formula.
- If $\varphi, \psi \in$ *PFOR* then $\varphi \rightarrow \psi$, $X\varphi$, $\varphi U \psi \in$ *PFOR.*

4.2.2 Semantics

The models of **CTL*** are labelled transition systems. We shall consider the case of a mono-transition system $\langle S, R\rangle$; the generalization to arbitrary transition systems is straightforward. The basic semantic notion is **truth relative to a run in a model**. If $M = \langle S, R, L\rangle$ is an LTS and σ is a run in M, then $M, \sigma \models \varphi$ will mean that φ *is true on the run* σ *in the model* M. The inductive definition naturally extends the truth definition for **LPTL**-formulae:

- $M, \sigma \models p$ if $p \in L(\sigma(0))$ for $p \in$**AP**;
- $M, \sigma \not\models \perp$;
- $M, \sigma \models \varphi \rightarrow \psi$ if $M, \sigma \models \varphi$ implies $M, \sigma \models \psi$;
- $M, \sigma \models X\varphi$ if $M, \sigma^1 \models \varphi$;
- $M, \sigma \models \varphi U \psi$, if $M, \sigma^j \models \psi$ for some $j \geq 0$ and $M, \sigma^k \models \varphi$ for every k such that $0 \leq k < j$.
- $M, \sigma \models \forall\varphi$ if $M, \tau \models \varphi$ for every run τ in M with the same initial state as σ.

Now, we say that:

- φ is **valid** in M, denoted $M \models \varphi$, if $M, \sigma \models \varphi$ for every run σ in M.
- φ is **valid** in a transition system $\mathfrak{T}$, denoted $\mathfrak{T} \models \varphi$ if it is valid in every model over that structure.
- φ is **valid,** denoted $\models \varphi$, if it is valid in every transition system.
- φ is **satisfiable** if it is true on some run σ of some model M. Then (M, σ) is called a **model of** φ.

Remark: Validity in **CTL***, unlike in most traditional logics, *is not closed under uniform substitutions:* $p \to \forall p$, for $p \in \mathbf{AT}$, is valid, while $Gp \to \forall Gp$ is not.

If state and path formulae are distinguished, then the notion of truth (resp. at states and on runs) can be defined by simultaneous induction on state and path formulae.

For state formulae:

(S1) $M, s \models p$ if $p \in L(s)$ for $p \in \mathbf{AP}$;

(S2) $M, s \not\models \bot$.

(S3) $M, s \models \varphi \to \psi$ if $M, s \models \varphi$ implies $M, s \models \psi$;

(S4) $M, s \models \forall\varphi$ if $M, \tau \models \varphi$ for every run τ in M with an initial state s.

For path formulae:

(P1) $M, \sigma \models \varphi$ if $M, \sigma(0) \models \varphi$ for any state formula φ.

(P2) $M, \sigma \models \varphi \to \psi$ if $M, \sigma \models \varphi$ implies $M, \sigma \models \psi$;

(P3) $M, \sigma \models X\varphi$ if $M, \sigma^1 \models \varphi$;

(P4) $M, \sigma \models \varphi U \psi$, if $M, \sigma^j \models \psi$ for some $j \geq 0$ and $M, \sigma^k \models \varphi$ for every k such that $0 \leq k < j$.

Validity of path formulae is defined as before. Besides, one can define *validity of state formulae,* often arguably considered as the primary notion of validity in **CTL***: a *state formula* φ *is valid (resp. satisfiable) in a model* M if φ is true at every (resp. some) state of that model. The other notions of validity for state formulae are defined like for path formulae.

Remark: **LPTL** *can be regarded as the fragment of* **CTL*** *where only pure path formulae are considered.* More precisely, every **LPTL** formula is at the same time a path formula in **CTL***. Conversely, for every *purely path-formula* φ of **CTL***, i.e. one with no path quantifiers, is an **LPTL-**formula, too.

4.2.3 Some useful validities

Exercise*:* Show that the following formulae are **CTL** *-valid:

- $\forall\varphi$ for every **LPTL**-valid formula φ;
- $\forall\varphi \to \varphi$; (NB: φ can be a path or a state formula.)
- $\forall X\varphi \to X\forall\varphi$;
- $\forall G\varphi \to G\forall\varphi$;

- $\forall G\exists F\varphi \rightarrow \exists GF\varphi$; (Burgess's formula)
- $\forall(\varphi \rightarrow \psi) \rightarrow (\forall\varphi \rightarrow \forall\psi)$;
- $\forall G(\varphi \rightarrow \exists X\varphi) \rightarrow (\varphi \rightarrow \exists G\varphi)$;
- $\forall G(\forall\varphi \rightarrow \exists XF\forall\varphi) \rightarrow (\forall\varphi \rightarrow \exists GF\forall\varphi)$;
- * $\forall G(\exists\varphi \rightarrow \exists X((\exists\psi U\exists\theta))) \rightarrow (\exists\varphi \rightarrow \exists G((\exists\psi U\exists\theta)))$ (Reynold's limit closure formula).

***Exercise**:* Show that the following formulae are *not* **CTL***-valid by constructing appropriate counter-models.

- $G\forall\varphi \rightarrow \forall G\varphi$;
- $\forall FG\varphi \rightarrow \forall F\forall G\varphi$;
- $\forall G\exists F\varphi \rightarrow \forall GF\varphi$;
- $\forall G(\varphi \rightarrow \exists XF\varphi) \rightarrow (\varphi \rightarrow \exists G\varphi)$.

***Exercise**:* Show that the following pairs of **CTL***-formulae are equivalent, i.e. true on the same pairs (M, σ).

- $\forall G\varphi$ and $\neg\exists F\neg\varphi$;
- $\exists G\varphi$ and $\neg\forall F\neg\varphi$;
- $\forall GF\varphi$ and $\forall G\forall F\varphi$;
- $\exists FG\varphi$ and $\exists F\exists G\varphi$.

***Exercise**:* Show that the following pairs of **CTL***-formulae are *not equivalent*, by constructing appropriate counter-models. Determine which implications are valid.

- $\exists GF\varphi$ and $\exists G\forall F\varphi$;
- $\exists GF\varphi$ and $\exists G\exists F\varphi$;
- $\forall FG\varphi$ and $\forall F\exists G\varphi$;
- $\exists FG\varphi$ and $\exists F\forall G\varphi$;

We will defer the question about axiomatization of the validities of **CTL*** until later.

4.2.4 Expressing properties with CTL*

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

- *partial correctness along every possible computation:*

$$\varphi \rightarrow \forall G(terminal \rightarrow \psi).$$

- *partial correctness along some possible computation:*

$$\varphi \rightarrow \exists G(terminal \rightarrow \psi).$$

- likewise, *total correctness along every possible computation:*

$$\varphi \rightarrow \forall F(terminal \wedge \psi).$$

- and *total correctness along some possible computation:*

$$\varphi \rightarrow \exists F(terminal \wedge \psi).$$

- *fairness along every possible computation:*

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

etc.

4.3 More on the semantics of CTL*

In this section we discuss two important issues on the semantics of **CTL*** :

- that it can be naturally generalized at the expense of losing some validities which might, arguably, be too strong sometimes.
- that the models of **CTL*** can be transformed into equivalent models of some canonical form, which will be important when the relation with automata is discussed.

The content of this section is not needed for further reading on **CTL,** though**.**

4.3.1 Generalized semantics for CTL*

In the semantics of **CTL*** given above all runs in a transition system were considered. This is not always necessary, and sometimes it is *not reasonable* because some runs 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* runs available. The situation here is similar to Ockhamist semantics over bundled trees.

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

Pairs $(\mathfrak{T}, \mathfrak{R})$ where $\mathfrak{R}$ is covering and suffix closed will be called **general branching time models.** The semantics of **CTL*** can be generalized over such models with no complications, just by restricting the path quantifications to the family $\mathfrak{R}$.

Given a transition system $\mathfrak{T}$ the set of all runs on $\mathfrak{T}$ will be denoted by $\mathfrak{R}_{\mathfrak{T}}$.

A general model $(\mathfrak{T}, \mathfrak{R}_{\mathfrak{T}})$ will be called a **standard model** or, following [Eme83], ***R*-generated model.**

Note that this generalized semantics is *not equivalent* to the standard one, i.e. *not every formula valid in all computation trees is valid in all branching time structures.*

Exercise: Show that each of the following formulae which are valid in every standard model of **CTL*** fails in some general branching time model.

1. $\forall X\varphi \to X\forall\varphi$;
2. $\forall G\exists Fp \to \forall GFp$;
3. $\forall G(\varphi \to \exists X\varphi) \to (\varphi \to \exists G\varphi)$.

Working through the first example, one can see that the reason for the possible failure is that *a run (falsifying* φ *) may belong to* $\mathfrak{R}$ *while its extension one step backwards is not in* $\mathfrak{R}$. Thus, another natural closure condition emerges; a family of runs $\mathfrak{R}$ is **prefix closed** *if whenever a run* σ *belongs to* $\mathfrak{R}$ *and* $sR\sigma(0)$ *then the run* $s.\sigma$ *obtained by prefixing* σ *with* s *must belong to* $\mathfrak{R}$ *, too.*

Given a run $\sigma = \sigma(0), \sigma(1), ...$ we denote by $(\sigma \mid n)$ its **initial segment** $\sigma(0), \sigma(1), ...\sigma(n)$.

Given two runs σ and τ, such that $\sigma(n) = \tau(m)$, the run $(\sigma \mid n).\tau^{m+1}$ obtained by appending τ^{m+1} to $(\sigma \mid n)$ is called a **fusion** of σ and τ.

A family of runs is called **fusion closed** if *every fusion of runs from* $\mathfrak{R}$ *belongs to* $\mathfrak{R}$.

Exercise: Show that suffix closure and prefix closure together imply fusion closure.

(There seems to be some confusion on that in [Sti92].)

Exercise: Show that the first formula from the previous exercise is valid in the class of all prefix closed (hence on all fusion closed) general models, while the other two are not. (Hint: read further.)

Yet another natural closure condition, satisfied by standard models is **limit closure:** *if* σ *is a run such that for every* $n \in \mathbf{N}$ *there is a run* τ_n *such that the fusion* $(\sigma \mid n).\tau_n$ *belongs to* $\mathfrak{R}$, *then* σ *must belong to* $\mathfrak{R}$.

As Emerson has shown, these are all closure conditions needed to ensure that a model is R-generated.

***Theorem**:* [Eme83]

1. *A family of runs* $\mathfrak{R}$ *is suffix, fusion, and limit closed iff there is a transition system* $\mathfrak{T}$ *such that* $\mathfrak{R} = \mathfrak{R}_{\mathfrak{T}}$.
2. *A general branching time model* $(\mathfrak{T}, \mathfrak{R})$ *is* R *-generated iff it is prefix and limit closed.*

***Exercise**:* Prove these.

For more on this see [Sti92] and for discussion of the standard vs general semantics.

From the viewpoint of Ockhamist branching time logics (complete trees vs bundled trees), see [Tho84] and [Zan96].

4.3.2 Unwinding of a transition system. Abstract representation of models for branching time logics

The branching time models on transition systems can be represented in a canonical, tree-like form, in which all runs are explicit.

***Definition**:* **Unwinding of the labelled transition system** $\mathcal{T} = \left\langle S, \left\{ \overset{a}{\longmapsto} \right\}_{a \in \mathbf{A}}, L \right\rangle$ is again a labelled transition system $\widehat{\mathcal{T}} = \left\langle \widehat{S}, \left\{ \overset{a}{\Longrightarrow} \right\}_{a \in \mathbf{A}}, \widehat{L} \right\rangle$ where:

- $\widehat{S}$ consists of all *finite paths* in $\mathcal{T}$ (incl. single states, regarded as paths of length 0).
- $\mathbf{s} \overset{a}{\Longrightarrow} \mathbf{t}$ holds if $\mathbf{t}$ is a one-step extension of $\mathbf{s}$ along the transition a, i.e.
  $\mathbf{t} = s_0 \overset{a_0}{\longmapsto} s_1 \overset{a_1}{\longmapsto} ...s_n \overset{a}{\longmapsto} s_{n+1}$, where $\mathbf{s} = s_0 \overset{a_0}{\longmapsto} s_1 \overset{a_1}{\longmapsto} ...s_n$.
  The last state of $\mathbf{s}$ will be denoted by $l(\mathbf{s})$.
- $\widehat{L}(\mathbf{s}) = L(l(\mathbf{s}))$.

Note the following:

$$\mathbf{s}_0 \overset{a_0}{\Longrightarrow} \mathbf{s}_1 \overset{a_1}{\Longrightarrow} \mathbf{s}_2 ...\textit{is a run in } \widehat{\mathcal{T}} \text{ iff } l(\mathbf{s}_0) \overset{a_0}{\longmapsto} l(\mathbf{s}_1) \overset{a_1}{\longmapsto} l(\mathbf{s}_2) ...\textit{is a run in } \mathcal{T}.$$

Recall that by a *computation* in a labelled transition system we mean the sequence of labels of the states in some run. A computation can be represented abstractly as a mapping $\xi : \mathbf{N} \to \mathbf{2}^{AP}$.

We are recasting here the notion of bisimulation introduced earlier for temporal frames.

***Definition**:* Let $M_1 = \langle S_1, R_1, L_1 \rangle$ and $M_2 = \langle S_2, R_2, L_2 \rangle$ be labelled transition systems. A non-empty relation $\beta \subseteq S_1 \times S_2$ is called a **bisimulation** between M_1 and M_2 if for every $s_1 \in S_1, s_2 \in S_2$ such that $s_1 \beta s_2$:

(AT) (M_1, s_1) and (M_2, s_2) satisfy the same atomic propositions.

(forth): For every $t_1 \in M_1$ such that $s_1 R_1 t_1$ there exists a $t_2 \in M_2$ such that $s_2 R_2 t_2$ and $t_1 \beta t_2$.

(back): For every $t_2 \in M_2$ such that $s_2 R_2 t_2$ there exists a $t_1 \in M_1$ such that $s_1 R_1 t_1$ and $t_1 \beta t_2$.

***Proposition**: If β is a bisimulation between M_1 and M_2 then for every $s_1 \in S_1, s_2 \in S_2$ such that $s_1 \beta s_2$ the following hold:*

1. *For every run σ_1 in M_1 such that $\sigma_1(0) = s_1$ there is a computationally equivalent run σ_2 in M_2 such that $\sigma_2(0) = s_2$ and vice versa.*
2. *(M_1, s_1) and (M_2, s_2) satisfy the same state formulae of $\mathbf{CTL}^*$.*

Proof: ***exercise**.*

Question: What about the converses?

***Proposition**: The mapping $l : S \to \widehat{S}$ is a bisimulation between $\mathcal{T}$ and $\widehat{\mathcal{T}}$.*

Proof: ***exercise**.*

A notion of bisimulation can be accordingly introduced as a relation between *runs rather that states.* See [Sti92] for some details.

Thus, we see that the semantics of $\mathbf{CTL}^*$ (standard and general) can be restricted to models on trees. In fact, it turns out that a simple type of trees is sufficient.

A tree will be called an **ω-tree** if every path in that tree has the order type ω of the natural numbers. Note that every unwinding of a transition system with a serial transition relation is a disjoint union of ω-trees.

A **branching factor** of an ω-tree is the supremum of the cardinalities of all sets of successors of nodes in the tree. For those unfamiliar with cardinals, for example the branching factor in a finitely branching tree is the greatest number of successors of a node in the tree, if such a number exists, otherwise it is ω.

For any cardinal κ, finite or infinite, an ω-tree is called **κ-branching** if every node has κ successors. We shall call these kinds of trees **homogeneous. (**Again, for those unfamiliar with cardinals, think that κ is a natural number of ω.)

Clearly, every κ-branching 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 can be labelled with the set of all finite strings on $[k] = \{0, .., k-1\}$, as follows: the root is labelled by the empty string ε and the successors of the node labelled by ς are $\varsigma 0, ..., \varsigma(k-1)$.

We shall call this labelling *the canonical representation* of the κ-branching ω-tree, denoted by $[\kappa]^*$. Note that the runs on $[\kappa]^*$ can be represented as mappings $f : \omega \to \kappa$, by assigning to every node from the run the last term of the string representing it.

Respectively, a **computation** *on a* κ *-branching tree is a labelling* $[\kappa]^* \to \mathbf{2^{AP}}$.

Proposition*: For every branching time model on an* ω *-tree with a branching factor* κ *there is a bisimilar model on* $[\kappa]^*$.

Sketch of proof: starting from the root, level by level, add as many as necessary copies of existing successors together with the subtrees rooted at them, to make the tree k-branching.

Exercise*:* Complete the details.

Corollary*: Every* $\mathbf{CTL}^*$ *state formula* φ *satisfiable in an* ω *-tree with a branching factor* κ *is satisfiable in a model on* $[\kappa]^*$.

Sketch of proof: First, unwind the model, and take the generated submodel rooted at the state satisfying φ. That is an ω -tree still satisfying φ due to the bisimulation with the original model. Now apply the previous proposition.

Exercise*:* Complete the details.

Proposition*: Every* $\mathbf{CTL}^*-$ *satisfiable state formula* φ *is satisfiable in a* k *-branching* ω *-tree, for* $k \leq m+1$ *where* m *is the number of path quantifiers occurring in* φ.

Sketch of proof: Induction on φ. Take an ω-tree satisfying φ and, starting from the root, level by level prune all spurious successors. For a proof see [Wol95], though some important details are missing there.

Furthermore, $\mathbf{CTL}^*$ can be interpreted on bounded branching ω^+-trees, i.e. trees in which every path has the order type of $\omega + 1$, where the last point can be regarded as marking the 'end' of the ω-path. This allows for reduction of the second-order quantification over paths to a first-order quantification in a more expressive *hybrid* language with reference pointers. It has been done in [Gor00].

4.4 The computation tree logic CTL

The *computation tree logic* **CTL** is a precursor of $\mathbf{CTL}^*$ and its syntactically restricted fragment. It was introduced by Clarke and Emerson in [CE81]. Although it is not as strongly expressive as $\mathbf{CTL}^*$, it is often a better choice for practical applications because of its lower complexity. On the other hand, **CTL** is an extension of the very similar branching time logic **UB** which does not contain U (only X and G) was proposed at about the same time by [BAPM81].

The language and syntax of **CTL** are essentially the same as those of $\mathbf{CTL}^*$, but there is a

restriction on the formation of the formulae: *there are only state formulae; the path formulae $X\varphi$ and $\varphi U\psi$ must be immediately quantified by path quantifiers*. Thus, for instance $\forall GF\varphi$ or $\exists(F\varphi \wedge \chi U\psi)$ are not **CTL** -formulae. (But, allowing Boolean combinations of path formulae within the scope of a path quantifier essentially does not extend the expressiveness of the language because all these can be translated into **CTL**; see [Dam94b] for details.) That requires both path quantifiers to be present in the language and the recursive definition of **CTL** -formulae is:

$$\varphi := p \mid \bot \mid \varphi_1 \to \varphi_2 \mid \forall X\varphi \mid \forall(\varphi_1 U \varphi_2) \mid \exists(\varphi_1 U \varphi_2)$$

Note that $\exists X\varphi$ is definable as $\neg\forall X\neg\varphi$, while $\forall(\varphi_1 U\varphi_2)$ and $\exists(\varphi_1 U\varphi_2)$ are not interdefinable.

Also definable are: $\forall F\varphi := \forall(\top U\varphi)$, $\exists F\varphi := \exists(\top U\varphi)$, $\forall G\varphi := \neg\exists F\neg\varphi$, $\exists G\varphi := \neg\forall F\neg\varphi$.

The semantics of **CTL** is the same as **CTL***.

A few words on the expressiveness of **CTL.** What concerns expressing invariance and eventuality properties, **CTL** is essentially as good as **CTL***: note that the examples of expressing partial and total correctness in **CTL*** are actually **CTL** formulae. However, **CTL** is not suitable for expressing *fairness properties* where G^∞ and F^∞ are essentially used. For instance, the example of fairness along every possible computation, expressible in **CTL*** is beyond the expressiveness of **CTL,** as it is certainly different from what seems to be the closest translation in **CTL:**

$$\forall G\forall F(\text{resource requested}) \to \forall F(\text{resource granted})$$

Question: Why are these different? Which is stronger?

4.4.1 Complete axiomatic system for CTL.

The first complete axiomatic system for **CTL** was proposed by Emerson and Halpern in [EH82], see also the journal version [EH85].

Here we give a streamlined version of the axiomatic system for **CTL** presented in [Eme90], from which all original axioms are easily derivable.

Axiom schemata:

- *Enough classical tautologies;*
- *(K_X):* $\forall X(\varphi \to \psi) \to (\forall X\varphi \to \forall X\psi)$,
- *(D_X):* $\exists X\top$,
- *($REC_{\exists U}$):* $\exists(\varphi U\psi) \leftrightarrow (\psi \vee (\varphi \wedge \exists X\exists(\varphi U\psi)))$,
- *($REC_{\forall U}$):* $\forall(\varphi U\psi) \leftrightarrow (\psi \vee (\varphi \wedge \forall X\forall(\varphi U\psi)))$,

- *($LFP_{\exists U}$):* $\forall G((\psi \vee (\varphi \wedge \exists X\theta)) \rightarrow \theta) \rightarrow (\exists(\varphi U\psi) \rightarrow \theta)$,
- *($LFP_{\forall U}$):* $\forall G((\psi \vee (\varphi \wedge \forall X\theta)) \rightarrow \theta) \rightarrow (\forall(\varphi U\psi) \rightarrow \theta)$,

Rules:

- Modus ponens *(MP),*
- *($NEC_{\forall G}$):* $\vdash \varphi$ implies $\vdash \forall G\varphi$,

***Theorem**: **CTL** is complete.*

The completeness can be proved by constructing a semantic tableau for any satisfiable formula. For details see [EH85] or [Eme90] . Alternatively, for a modal model-theoretic proof, see [Gol92].

Exercise:* Goldblatt in [Gol92] has replaced the axioms *($LFP_{\exists U}$)* and *($LFP_{\forall U}$)* with the corresponding induction rules:

- *($\exists$ -Ind):* $\vdash \psi \vee (\varphi \wedge \exists X\theta) \rightarrow \theta$ implies $\vdash \exists(\varphi U\psi) \rightarrow \theta$,
- *($\forall$ -Ind):* $\vdash \psi \vee (\varphi \wedge \forall X\theta) \rightarrow \theta$ implies $\vdash \forall(\varphi U\psi) \rightarrow \theta$,

Clearly, the rules are derivable from the axioms. Show that the axioms can be derived from the rules.

4.5 Decidability and complexity of CTL and CTL*

***Theorem**: (Small model theorem for* **CTL,** see [Eme90]*) Let a* **CTL-** *formula φ have a length n. Then the following are equivalent:*

(i) *φ is satisfiable.*

(ii) *φ has an infinite tree-like model with branching factor bounded by $O(n)$.*

(iii) *φ has a finite model of size $\leq \exp(n)$.*

Idea of proof: (i)⇒(ii) is proved by unwinding the satisfying model and then chopping out spurious branches by leaving at most n successors for every node, going bottom up level per level.

(ii)⇒(iii) is proved by filtration of the trees, which produces a 'pseudo-model' which is then unwound appropriately into a finite model.

***Theorem**: The satisfiability problem for* **CTL** *is EXPTIME-complete.*

The upper bound is provided by the tableau construction mentioned above. The matching lower bound is by reduction from alternating polynomial space bounded Turing machines, similar to the proof for PDL in [KT90].

While there are completeness results (see [Sti92]) for **CTL** * with respect to the generalized semantics introduced above, until recently no finitary complete axiomatization of **CTL*** with respect to the standard semantics was known. M. Reynolds announced in [Rey98] such an axiomatization using a rather complex additional rule in the style of Gabbay's irreflexivity rule, the idea of which is based the relation between CTL* and Muller automata, used in the proof of completeness. In a very recent draft [Rey00] he announced that such a rule can be omitted if the past operators are added to the language.

Theorem*: The satisfiability problem for* **CTL** * *is EXPEXPTIME complete.*

Emerson and Sistla produced a double exponential time algorithm for deciding satisfiability for CTL* in [ES84] by an elaborated reduction to non-emptiness of automata on infinite trees. A matching lower bound can be obtained by reduction from alternating exponential space bounded Turing machines.

References*:* For more on variations of branching time logics and a comprehensive comparison of their expressiveness, see [Eme90].

In the last chapter of these notes we will come back to the question of expressiveness of branching time logics, by comparing it with some monadic second-order theories.

5 Temporal logics and automata

There is an intimate relationship between linear and branching time temporal logics on one hand, and a variety of notions of automata on **infinite words** (for linear temporal logic) and **infinite trees** (for branching time logics) on the other. Put in a nutshell, these automata can be considered as running on models for temporal logic, and furthermore there are efficient algorithms which for every formula of the logic produce an automaton which accepts precisely the models of that formula. Thus, *checking satisfiability of a formula from the temporal logic under consideration is reduced to checking non-emptiness of the language accepted by the corresponding automaton.* Moreover, the algorithms checking non-emptiness of automata are *constructive,* i.e. if the language is non-empty, i.e. if the formula is satisfiable, then the algorithm produces a model for it. Thus, apart from providing *efficient decision procedures* for various temporal logics, the automata serve as *model generators* for these logics.

This relationship has turned out very fruitful for the development of both areas, and instrumental for the successful use of temporal logics in computer science. Here we only give a brief outline of that topic. There is ample literature on the topic. For more details, some recommended texts are [VW86], [Tho90], [Wol95], and the online set of notes [Var].

5.1 Preliminaries: finite state automata

Definition: A **finite state automaton** is a 5-tuple $\mathbf{A} = \langle \Sigma, S, \rho, S_0, F \rangle$ where:

- Σ is a finite set of symbols called **alphabet**;
- S is a finite **set of states**;
- $\rho : S \times \Sigma \to \mathbf{2}^S$ is a **transition function** which determines the possible transitions from a state upon reading a symbol from the alphabet.
- $S_0 \subseteq S$ is a non-empty set of **initial states;**
- F is a set of **accepting states.**

The traditional finite state automata work on **finite words (strings) over the alphabet** Σ. The set of all words over Σ, incl. the empty word Λ, is denoted by Σ^*. Starting from an initial state the automaton reads a word symbol after symbol and, being at a state s after reading the next symbol a, the automaton makes a non-deterministic transition to a state from $\rho(s, a)$. Every sequence of states obtained this way is called an **execution (**or, **run**) of the automaton on the word. If the last state of *some* execution on a word is an accepting state, the automaton **accepts** the word. The set of all words from Σ^* accepted by the automaton $\mathbf{A}$ is called **the language of** $\mathbf{A}$, denoted by $L(\mathbf{A})$.

If every $\rho(s, a)$ contains no more than one state, the automaton is **deterministic,** otherwise it is **non-deterministic.**

To represent languages (usually infinite) accepted by automata, we use **regular expressions** obtained by applying the **regular operations** on sets of words**: concatenation** indicated by juxtaposition, **union** $\cup$ and **finite repetition (iteration)** $*$. For example ab^* denotes the language (set of words) on the alphabet $\{a, b\}$ consisting of all words beginning with an a followed by finitely many b's; $(ab)^* \cup (ba)^*$ is the language of all words of even **length** (number of symbols) consisting of alternating letters a and b.

Languages which can be defined by means of regular expressions are called **regular languages.**

Theorem*:*

1. A language of finite words is regular iff it is accepted by a finite state automaton.

2. The family of regular languages is closed under all Boolean operations.

The proof of this classical result can be found in any textbook on finite automata, e.g. the classical [HU79]. We only remark here that closedness under complementation is proved by *determinization* of the automaton (using *Rabin-Scott set construction)* i.e. by showing that every automaton accepts the same language as some deterministic automaton.

5.2 Büchi automata on infinite words

The simplest automaton on infinite words is the (**non-deterministic) Büchi automaton on infinite words,** also known as **Büchi sequential automaton.** It is defined just like the standard finite state automaton, but the difference in its operation is in the **acceptance condition.** The notion of **execution (run)** of a Büchi automaton over an infinite word is defined as before, but now it is an *infinite* sequence of states, and *an infinite word is accepted by a Büchi automaton if some execution on that word visits an accepting state infinitely many times.* Since the automaton has only finitely many states, this condition can be put in a stronger form: the word is accepted if *some accepting state* is visited infinitely many times during some execution.

Thus, with every Büchi automaton **A** we associate its language $L(A)$ consisting of the infinite words accepted by the automaton.

Example: The automaton on infinite words $\mathbf{A} = \langle \{a, b\}, \{s_0, s_1\}, \rho, \{s_0\}, \{s_1\} \rangle$, where $\rho(s_0, a) = \{s_0\}$, $\rho(s_0, b) = \{s_1\}$, $\rho(s_1, a) = \{s_0\}$, $\rho(s_1, b) = \{s_1\}$ accepts those words in which b occurs infinitely often.

Exercise: Construct automata on infinite words in $\{a, b\}$ which accept the following languages:

- the set of words which contain a at least once;
- the set of words which contain a exactly once;
- the set of words which eventually contain only a.
- the set of words which contain at least one a between every occurrences of b.

To represent languages accepted by Büchi automata, we use $\omega-$ **regular expressions** obtained by applying the regular operations together with the **infinite repetition** operator ω. For example $(b^*a)^\omega$ denotes the language of infinite words on the alphabet $\{a, b\}$ consisting of all words containing infinitely many a's. An $\omega-$regular expression is **star-free** if it does not use $*$.

The set of all infinite words on Σ is denoted by Σ^ω.

A language $L \subseteq \Sigma^\omega$ is (star-free) $\omega-$ **regular** if it can be defined by means of a (star-free) $\omega-$regular expression.

Theorem: 1. A language is $\omega-$ regular iff it is accepted by a Büchi automaton.

2. The family of $\omega-$ regular languages is closed under all Boolean operations.

The proof of this result is much more involved than the one for finite words automata. In particular, for the closedness under intersection, *generalized* Büchi automata are introduced, which have the same accepting power as the standard ones. These automata have *a set of sets of accepting states,* and the accepting condition is that a state from *each* of these sets must occur infinitely often in the run. Furthermore, complementation cannot be

proved only by determinization anymore, because *not every* Büchi automaton can be determinized. Actually, deterministic Büchi automata define a *proper* subclass of the $\omega-$regular languages. Instead, *Müller* automata are used which demand as an accepting condition the set of infinitely visited runs to be one of the designated accepting sets of states.

For more details, see [Tho90].

Historical remark: Büchi introduced his automata with a purely theoretical purpose, viz. to prove decidability of the monadic second-order theory $S1S$ of the natural numbers with the successor function only (to be introduced later).

5.3 Automata on infinite words and linear time temporal logic

Recall that a canonical representation of an **LPTL**-model is simply a mapping σ: $\mathbf{N} \to \mathbf{2^{AP}}$, i.e. an *infinite word of sets of atomic propositions.* Thus, *Büchi automata can be considered as running over* **LPTL** *-models*, and this is the core of the relationship we are about to describe. The a main outcome of this relationship is:

Theorem*: For every* **LPTL***- formula* φ *there is a Büchi automaton* $\mathbf{A}_\varphi$ *on the set of states* $\mathbf{2^{AP}}$ *which accepts precisely the models of* φ.

Sketch of proof: Right at the beginning we can restrict the set of atomic propositions **AP** to those occurring in φ.

A *naive approach:* build the automaton inductively on the structure of φ. Yes, it can be done, but the price to pay (i.e. the complexity) can be too high as every step with U involves and exponential blowup of the size.

A more intelligent approach is to build the automaton from components taking care of the main tasks in the model-checking. Here is an outline of the construction given in [Wol95].

First, we need the syntactic notion of a **closure of a formula** φ**,** denoted $cl(\varphi)$. This is a finite set of formulae defined in accordance with the language. For **LPTL** formulae $cl(\varphi)$ is obtained as follows: we take all subformulae of φ and their negations, and then delete the double negations, if any, occurring in the obtained formulae. For example,

$$cl(\neg Xp \to \top U \neg q) = \{\neg Xp, Xp, p, \neg p, \top U \neg q, \neg(\top U \neg q), \top, \bot, \neg q, q\}\,.$$

Note that $\mid cl(\varphi) \mid \leq 2 \mid \varphi \mid$, where $\mid \varphi \mid$ is the number of symbols in φ. Clearly, only the truth of the formulae from $cl(\varphi)$ across the model is relevant to the validity of φ at that model, and therefore these, and only these, must be under control when building or verifying a model of φ.

Further, we introduce the notion of **eventuality:** this is every formula of the type $\chi U \psi$, in particular $F\psi$. The satisfaction of an eventuality at a state demands that something (viz. ψ) *must hold at a later, but not specified state.* Thus, the satisfaction of an eventuality can be postponed *indefinitely,* which implies the risk of it being postponed *forever,* i.e. never satisfied. Therefore, special care must be taken of the satisfaction of all eventualities required for the satisfaction of the original formula at the initial state.

Now, the idea of the Büchi automaton $\mathbf{A}_\varphi$ is that while running over a model it checks

meticulously if the model satisfies φ at its initial state. An **LPTL-**model, consisting of a run in an LTS is canonically represented as a sequence of labels of the successive states of the run. The label of a state, however, will consist not only of atomic propositions, but of *all formulae from* $cl(\varphi)$ *which are regarded true at that state.* In particular, to claim that φ is satisfied by the run, it must belong to the label of the initial state. Now, the automaton must verify that the word which it is reading is *indeed a truthful representation of a genuine model for* φ . Three requirements must be checked:

1. *the set of formulae regarded as true at any state must be propositionally consistent;*
2. *every state must satisfy what is prescribed to it by its predecessor, e.g. every* ψ *such that* $X\psi$ *is true at the predecessor;*
3. *all eventualities required by any state must be eventually satisfied.* (That is, all promises made throughout the model must be eventually fulfilled.)

The automaton $\mathbf{A}_\varphi$ will be composed out of two Büchi automata taking care respectively of the 2nd and 3rd requirement, while the first one will be satisfied by the construction. These two component automata are:

- the **local automaton** $\mathbf{A}^l_\varphi$, and
- the **eventuality automaton** $\mathbf{A}^e_\varphi$.

The local automaton is defined as $\mathbf{A}^l_\varphi = \langle \mathbf{2}^{cl(\varphi)}, N_l, \rho_l, N_\varphi, N_l \rangle$ where:

- N_l consists of all *maximal propositionally consistent* subsets of $cl(\varphi)$, i.e. the propositionally consistent subsets $\mathbf{s}$ satisfying the additional conditions:

  for every $\psi \in cl(\varphi)$ not beginning with a negation, either $\psi \in \mathbf{s}$ or $\neg\psi \in \mathbf{s}$.

  Exercise*:* Show that every set $\mathbf{s}$ from N_l also satisfies the following:

  - $\bot \notin \mathbf{s}$;
  - for every $\psi \to \chi \in cl(\varphi)$, $\psi \to \chi \in \mathbf{s}$ iff either $\neg\psi \in \mathbf{s}$ or $\chi \in \mathbf{s}$.
  - for every $\chi U \psi \in cl(\varphi)$, if $\chi U \psi \in \mathbf{s}$ then $\psi \in \mathbf{s}$ or $\chi \in \mathbf{s}$.

- ρ_l is defined as follows: $\mathbf{t} \in \rho_l(\mathbf{s}, \mathbf{a})$ iff $\mathbf{s} = \mathbf{a}$ and $\mathbf{t}$ satisfies the *local conditions:*

  - for every $X\psi \in cl(\varphi)$, if $X\psi \in \mathbf{s}$ then $\psi \in \mathbf{t}$;
  - for every $\chi U \psi \in cl(\varphi)$, if $\chi U \psi \in \mathbf{s}$ and $\psi \notin \mathbf{s}$ then $\chi U \psi \in \mathbf{t}$.

- N_φ consist of those sets $\mathbf{s} \in N_l$ which contain φ.

Note that every state is accepting, so as long as the automaton can run forever it accepts the model.

The eventuality automaton is defined as $\mathbf{A}^e_\varphi = \langle \mathbf{2}^{cl(\varphi)}, \mathbf{2}^{ev(\varphi)}, \rho_e, \{\varnothing\}, \{\varnothing\} \rangle$ where:

- $ev(\varphi)$ is the subsets of eventualities in $cl(\varphi)$;
- ρ_e is defined as follows: $\mathbf{t} \in \rho_l(\mathbf{s}, \mathbf{a})$ iff:

  (i) $\mathbf{s} = \varnothing$ and $\mathbf{t}$ consists of those formulae $\chi U \psi \in \mathbf{a}$ such that $\psi \notin \mathbf{a}$, or

  (ii) $\mathbf{s} \neq \varnothing$ and $\mathbf{t}$ consists of those formulae $\chi U \psi \in \mathbf{s}$ such that $\psi \notin \mathbf{a}$.

Some intuition: at the beginning there are no promises (eventualities to be satisfied). As the execution goes, they pile up in the current state (consisting of the 'imminent tasks'), and meanwhile some of them get satisfied. When the current state is emptied, then the automaton looks at the model to see what eventualities are still to be satisfied. Note that it is *not necessary to check that at every step, but only periodically*, because all unsatisfied yet eventualities will be carried forward (*propagated)* by the local automaton. The model is accepted if the empty set of eventualities is visited infinitely often, i.e. they all get eventually satisfied.

Once constructed, the two automata $\mathbf{A}^l_\varphi$ and $\mathbf{A}^e_\varphi$ are combined into one accepting the intersection of their languages, i.e. those models which satisfy all conditions necessary for the satisfaction of φ at their initial state. Finally, the obtained automaton, which is defined on the alphabet $\mathbf{2}^{cl(\varphi)}$ is transformed to one defined on $\mathbf{2}^{\mathbf{AP}}$. Besides the alphabet, only the transition relation ρ changes into ρ' defined as follows: $\mathbf{t} \in \rho'(\mathbf{s}, \mathbf{a})$ iff there is $\mathbf{b} \in \mathbf{2}^{cl(\varphi)}$ such that $\mathbf{a} = \mathbf{b} \cap \mathbf{AP}$ and $\mathbf{t} \in \rho(\mathbf{s}, \mathbf{b})$.

This completes the sketch. For more details see e.g. [Wol95], [Var].

Now, the upshot: *in order to check whether a* **LPTL** *formula is satisfiable, it suffices to build its Büchi automaton* $\mathbf{A}_\varphi$ *and check if it accepts any model, i.e. if its language is* **non-empty.**

Theorem*: The language accepted by a Büchi automaton is non-empty precisely when there is an accepting state which is reachable from an initial state and from itself.*

Proof: ***Exercise****.*

This non-emptiness check can be done in time linear on the size of the automaton (see[EL86]). Moreover, the complexity of the whole construction is PSPACE (see [VW94] for an upper bound and [SC85b] for a lower bound) Thus:

Theorem*:* Satisfiability/validit *y in* **LPTL** *is PSPACE complete.*

Reference*:* For a comprehensive account on the topic see [Var94].

5.4 Büchi automata on infinite trees

Büchi automata can be generalized to accept branching time models over k-branching trees, i.e. labelled k-branching trees. The corresponding k **-ary Büchi tree automaton** is defined, like the sequential Büchi automaton, as a tuple $\mathbf{A} = \langle \Sigma, S, \rho, S_0, F \rangle$ with the only difference being that the transition function ρ determines for every pair of state and a symbol from Σ a set of k *-tuples* of successors, i.e. $\rho : S \times \Sigma \to \mathbf{2}^{S^k}$.

A **run** of a tree automaton over a *labelled* k-branching tree $T : [k]^* \to \Sigma$ is again a labelled k-branching tree $\Theta : [k]^* \to S$ where $\Theta(\varepsilon) \in S_0$ and for every $\varsigma \in [k]^*$, $(\varsigma 0, \varsigma 1, ..., \varsigma(k-1)) \in \rho(\Theta(\varsigma), L(\varsigma))$.

Intuitively, the run of a tree automaton $\mathbf{A}$ on a tree T produces a labelling of $[k]^*$ with the states of $\mathbf{A}$ according to the transition relation of $\mathbf{A}$.

To define the accepting runs we consider the *paths* (i.e. maximal runs) in the tree Θ. Formally, these are the infinite runs in Θ starting from the root. Now, *the run* Θ *of* $\mathbf{A}$ *over* T is **accepting** *if on every path in* Θ *some accepting state occurs infinitely often.*

The automaton $\mathbf{A}$ **accepts** the labelled tree T if *there is an accepting run of* $\mathbf{A}$ *over* T. The set of all labelled trees accepted by the automaton $\mathbf{A}$ is called **the language of** $\mathbf{A}$, denoted by $L(\mathbf{A})$.

Example: The automaton on binary trees $\mathbf{A} = \langle \{a, b\}, \{s_0, s_1\}, \rho, \{s_0\}, \{s_1\} \rangle$, where $\rho(s_0, a) = \{(s_0, s_0)\}$, $\rho(s_0, b) = \{(s_1, s_1)\}$, $\rho(s_1, a) = \{(s_0, s_0)\}$, $\rho(s_1, b) = \{(s_1, s_1)\}$ accepts those labelled binary trees on every path of which b occurs infinitely often.

Exercise: Construct automata on binary trees labelled by $\{a, b\}$ which accept the following languages:

- the set of trees on which a occurs at least once;
- the set of trees on which a occurs at least once on every path;
- the set of trees on which some path eventually contains only a.

5.5 Automata on infinite trees and CTL

The idea is the same as for **LPTL:**

Theorem: *(*[VW86]*) For every* **CTL** *-formula* φ and a natural number k there is a Büchi tree automaton $\mathbf{A}_\varphi$ which accepts precisely those k-branching models which satisfy φ.

Sketch of proof: The construction extends the construction for sequential automata outlined above. We sketch it briefly, following again [Wol95]. The automaton $\mathbf{A}_\varphi$ is built again over the alphabet $\mathbf{2}^{cl(\varphi)}$ but now it is composed from *three automata:* **local, existential eventuality (**taking care of the eventualities of the type $\exists \chi U \psi$) and **universal eventuality** (for those of the type $\forall \chi U \psi$) **.** The construction of the local automaton is pretty much similar to the one for the sequential automata.

Exercise: Specify the requirements which the local automaton must check, and then construct it.

Likewise, the universal eventualities automaton is essentially the same as for sequential eventuality automaton, but now it runs along all paths of the tree.

Exercise: Construct the universal eventualities automaton.

As for the existential eventualities automaton the situation is similar, but now the different

eventualities can be satisfied on *different paths of the tree*, so one has to ensure that the construction indeed checks that all of them eventually get satisfied.

Exercise: Ditto for the existential eventualities automaton.

Finally, all three automata are combined into one which accepts the intersection of their languages. It is then reduced to one on the alphabet $\mathbf{2^{AP}}$. For more details, see [Wol95].

Thus, the satisfiability problem for **CTL** is reduced to checking non-emptiness of Büchi tree automata, which is known to be *LOGSPACE* complete for *PTIME* ([VW86]). The construction of the automaton corresponding to the formula takes *EXPTIME,* which is the complexity of satisfiability for **CTL.**

For much more on the topic, and a state-of-the-art-picture (both of the topic and the person), check out Moshe Vardi's website at http://www.cs.rice.edu/~vardi/.

6 Introduction to μ-calculus

The propositional μ-calculus ([Pra81],[Koz83]) can be introduced as an extension of the temporal logic for transition systems with operators for *a least fixpoint (μ) and a greatest fixpoint (ν).* It turns out to be an extremely powerful formalism for specification and verification of temporal properties, in which most of the known temporal logics can be translated.

6.1 Monotonic operators and fixed points

Let W be a set. An operator $\Gamma : \mathbf{2}^W \to \mathbf{2}^W$ is **monotonic,** if $Y_1 \subseteq Y_2$ implies $\Gamma(Y_1) \subseteq \Gamma(Y_2)$.

A set $Y \subseteq W$ is a

- **fixed point (fixpoint)** of Γ if $\Gamma(Y) = Y$;
- **pre-fixpoint** of Γ if $\Gamma(Y) \subseteq Y$;
- **post-fixpoint** of Γ if $\Gamma(Y) \supseteq Y$;
- **least fixpoint (least pre-fixpoint)** if Y is a fixpoint (pre-fixpoint) and for every fixpoint (pre-fixpoint) Z, $Y \subseteq Z$. Clearly, if Γ has a least fixpoint, it is unique.
- **greatest fixpoint (greatest post-fixpoint)** if Y is a fixpoint (post-fixpoint) and for every fixpoint (post-fixpoint) Z, $Y \supseteq Z$. Again if Γ has a greatest fixpoint, it is unique.

If Γ has a least fixpoint, it is denoted by $\mu\Gamma$. Likewise, the greatest fixpoint of Γ, if it exists, is denoted by $\nu\Gamma$.

Theorem: (Knaster-Tarski):

1. Every monotonic operator $\Gamma : \mathbf{2}^W \to \mathbf{2}^W$ has a least fixpoint and a greatest fixpoint.
2. $\mu\Gamma = \bigcap \{Z \subseteq W \mid \Gamma(Z) = Z\} = \bigcap \{Z \subseteq W \mid \Gamma(Z) \subseteq Z\}$,
   $\nu\Gamma = \bigcup \{Z \subseteq W \mid \Gamma(Z) = Z\} = \bigcup \{Z \subseteq W \mid \Gamma(Z) \supseteq Z\}$.
3. $\mu\Gamma = \bigcup_{\alpha \leq \|W\|} \Gamma^\alpha(\varnothing)$,
   $\nu\Gamma = \bigcap_{\alpha \leq \|W\|} \Gamma_\alpha(W)$.

In these identities α is an ordinal taking all values not greater than the cardinality $\|W\|$ of W.

The meaning of Γ^α is: $\Gamma^0(Z) = Z$, $\Gamma^{\beta+1}(Z) = \Gamma(\Gamma^\beta(Z))$, and $\Gamma^\gamma(Z) = \bigcup_{\beta<\gamma} \Gamma^\beta(Z)$ for limit ordinals γ.

Respectively, the meaning of Γ_α is: $\Gamma_0(Z) = Z$, $\Gamma_{\beta+1}(Z) = \Gamma(\Gamma_\beta(Z))$, and $\Gamma_\gamma(Z) = \bigcap_{\beta<\gamma} \Gamma_\beta(Z)$ for limit ordinals γ.

***Exercise**:* The proofs of these claims are nice elementary exercises. Try them!

Remark: The last part of Knaster-Tarski's theorem actually says more than meets the eye: due to the monotonicity of Γ it is easy to see that $\Gamma^\alpha(Z) \subseteq \Gamma^\beta(Z)$ and $\Gamma_\alpha(Z) \supseteq \Gamma_\beta(Z)$ whenever $\alpha < \beta$. i e. both sequences are monotone, hence they are bound to reach their fixpoints by the κ-th iteration where $\kappa = \|W\|$. Thus, the least (resp. greatest) fixpoint can be obtained by simply applying the successive iterations of Γ, beginning with $\varnothing$ (for μ) or with S (for ν) until a fixpoint is reached. Often it is practically easier to compute or at least analyze these iterations (*unfolding*) than the intersection (resp.) union of all pre-fixpoint (resp. post-fixpoints).

6.2 Temporal formulae as operators on transition systems

Consider any temporal language where truth of a formula at a state is defined. Let $\mathfrak{T} = \langle S, R\rangle$ be a transition system. Then, given a labelling L, with every formula φ we can associate its *extensional:* the set $\|\varphi\|$ of states where φ is true. Of course, $\|\varphi\|$ depends on L, and more specifically, on the extensionals of the atomic propositions occurring in it. Thus, if p is an atomic proposition occurring in φ, then φ can be regarded as an operator $\lambda p.\varphi(p) : \mathbf{2}^S \to \mathbf{2}^S$, defined by $\lambda p.\varphi(p)(\|p\|) = \|\varphi\|$.

We shall make this precise later. For now, this suffices to give an idea of the μ-calculus.

***Exercise**:* Write down the inductive definitions of $\lambda p.\varphi(p)$ for **LPTL** and for **CTL.**

A formula φ is **positive in the propositional variable** p if every occurrence of p in φ is positive, i.e. in the scope of an even number of negations.

Proposition: If φ is positive in p then $\lambda p.\varphi(p)$ is monotone.

***Exercise**:* Prove this fact by induction on φ. Note that every time a negation is applied, the direction of the monotonicity reverses.

Remark: Actually, the converse holds, too: if $\lambda p.\varphi(p)$ is monotone then φ is positive in p.

Thus, with every formula $\varphi(p)$ monotone in p one can associate the formulae expressing *the least fixpoint* $\mu p.\varphi(p)$ and *the greatest fixpoint* $\nu p.\varphi(p)$ of $\lambda p.\varphi(p)$.

We are now ready to introduce formally μ-calculus.

6.3 μ-calculus: language and syntax

First, we will define the basic μ-calculus on transition systems and then will extend it to the language of branching time logics.

Before going further, we need to modify the language slightly, by adding a new sort of **propositional variables**. These will behave just like atomic propositions, apart from the fact that they will serve a different purpose, viz. they will only be used as *bound variables* by the fixpoint operators. They can be avoided altogether, but we will use them for technical convenience and syntactic disambiguity.

The language of μ-calculus extends the temporal language for transition systems with a countable set of propositional variables $\mathbf{PV} = \{z_0, z_1, ...\}$, (disjoint from the set of atomic propositions), and the additional operator of **least fixpoint** μ. We will consider the simplest case of a monotransition system $\langle S, R\rangle$ for which no assumptions are made. The generalization to systems with arbitrary set of transitions is straightforward. As already discussed, the modality corresponding to R is X where the meaning of $X\varphi$ is: φ *is true at all* $R-$ *successors of* s . We denote the set of R-successors of s by $R(s)$.

The inductive definition of formulae is:

$$\varphi := p \mid z \mid \bot \mid \varphi_1 \to \varphi_2 \mid X\varphi \mid \mu z.\varphi$$

where p is a meta-variable for an atomic proposition, z is one for a propositional variable, and $\mu z.\varphi$ is only allowed if z occurs positively in φ.

Again, all Boolean connectives are definable as usual. Besides, we define the **dual of** μ :

$$\nu z.\varphi := \neg\mu z.\neg\varphi[\neg z/z].$$

The operator ν is called the **greatest fixpoint.**

Some syntactic definitions a la first-order logic. A variable z is **bound** in a formula ψ if it occurs in a subformula $\mu z.\varphi$ of ψ. More precisely, such an occurrence of z is bound by the occurrence of μ at the beginning of the *smallest subformula* $\mu z.\varphi$ containing that occurrence of z. Variables which are not bound in a formula are **free** in that formula. A formula is a **sentence** if all occurrences of variables in it are bound.

6.4 Semantics of μ-calculus

The models for μ-calculus are labelled transition systems. The basic semantic notion is *truth of a formula at a state of a model* $M = \langle S, R, L\rangle$, *relative to a valuation* $V : PV \to \mathbf{2}^S$.

In order to define it we first define inductively the **extensional of a formula** φ **relative to the valuation** V, denoted $\|\varphi\|_V$. For the propositional variables it is already defined. For the other formulae:

- $\|p\|_V = \{s \mid p \in L(s)\}$ for $p \in \mathbf{AP}$;
- $\|\bot\|_V = \varnothing$;
- $\|\varphi \to \psi\|_V = (S - \|\varphi\|_V) \cup \|\psi\|_V$;
- $\|X\varphi\|_V = \{s \mid R(s) \subseteq \|\varphi\|_V\}$;
- $\|\mu z.\varphi\|_V = \bigcap \left\{ Z \subseteq S \mid \|\varphi\|_{V[z:=Z]} \subseteq Z \right\}$,

where $V[z := Z]$ is the valuation obtained from V by redefining to take a value Z at z.

Note that, by Knaster-Tarski's theorem, $\|\mu z.\varphi\|_V$, being the intersection of all pre-fixed points of the monotone operator $\lambda z.\varphi$ is the least fixpoint of that operator.

Now, we can define **truth at a state** s **of a model** M, **relative to a valuation** V:

$$M, V, s \models \varphi \text{ iff } s \in \|\varphi\|_V .$$

Exercises: Show that:

- This truth definition coincides with the usual one for the formulae with no propositional variables and μ-operators.
- $\|N\varphi\|_V = \{s \mid R(s) \cap \|\varphi\|_V \neq \varnothing\}$.
- $\|\nu z.\varphi\|_V = \bigcup \left\{ Z \subseteq S \mid Z \subseteq \|\varphi\|_{V[z:=Z]} \right\}$, i.e. $\|\nu z.\varphi\|_V$ is the greatest fixpoint of $\lambda z.\varphi$.
- The extensional of any formula φ, hence its truth in a model, *only depends on the valuation of the propositional variables which have free occurrences in* φ. In particular, the extensional of a sentence does not depend on the valuation.

Eventually, we are only interested in sentences, so the valuations play an auxilliary role in the semantics of μ-calculus.

Understanding the meaning of a formula of μ-calculus is usually not easy. Some syntactic conditions on the syntax of the formulae can be made which will not restrict the language but will make formulae easier to deal with. First, of all, by appropriate *renaming of bound variables,* just like in first-order logic, it can be ensured that different occurrences of the fixpoint operators bind different variables. Furthermore, using both μ and ν allows all negations to be driven in front of atomic propositions and variables. Thus, in sentences all negations will be applied to atomic propositions only. This is the *positive normal form* of a formula.

An **alternating depth** of a formula is the greatest number of *essential nestings of alternating fixpoint operators.* A nesting is **essential** if the variable bound by the external operator occurs free in the scope of the internal one.

All formulae considered so far have an alternating depth 1, as well as e.g. $\mu z_1.X(z_1 \vee \nu z_2.Nz_2)$. The formula $\mu z_1.X(z_1 \vee \nu z_2.(z_1 \wedge Nz_2))$ has an alternating depth 2, however.

Essentially all practically useful and comprehensible sentences have an alternating depth at most 2, but as Bradfield has proved in [Bra96], the alternating depth hierarchy of μ-calculus is strict, i.e. no fragment with bounded alternating depth is as expressive as the whole language.

6.5 Defining the temporal operators

First, let us explore the semantics of fixpoint operators with some simple examples.

For technical simplicity we shall write $X(Z)$ and $N(Z)$ instead of $\lambda z.Xz(Z)$ and $\lambda z.Nz(Z)$.

***Examples**:*

1. $\varphi = \mu z.Nz : \|\varphi\|_V = \bigcap \{Z \subseteq S \mid \{s \mid R(s) \cap Z \neq \varnothing\} \subseteq Z\}$. This seems not very easy to analyze, but becomes trivial once you notice that $\{s \mid R(s) \cap \varnothing \neq \varnothing\} \subseteq \varnothing$, hence $\varnothing$ is one of those Z occurring in the intersection. So, $\|\varphi\|_V = \varnothing$, i.e. $\mu z.Nz$ is equivalent to $\bot$.

   Often an easier way of computing fixpoints is by using the last part of Knaster-Tarski"s theorem, i.e. applying the successive iterations *(unfolding)* of the operator. In our case: $N(\varnothing) = \{s \mid R(s) \cap \varnothing \neq \varnothing\} = \varnothing$ and we have reached the fixpoint.

2. $\varphi = \nu z.Xz : X(S) = \{s \mid R(s) \subseteq S\} = S$, which is a fixpoint, hence $\nu z.Xz = \top$. Alternatively, we can use the previous example and duality: $\nu z.Xz = \neg\mu z.\neg X\neg z = \neg\mu z.Nz = \top$.

3. $\varphi = \nu z.Nz$. $N(S) = \{s \mid R(s) \cap S \neq \varnothing\} = \{s \mid R(s) \neq \varnothing\} = \|N\top\|_V$;
   Likewise, $N^2(S) = N(N(S)) = \|NN\top\|_V = \left\|N^2\top\right\|_V$, ..., $N^m(S) = \|N^m\top\|_V$.

   Note that $\|N^m\top\|_V$ consists of those states from which some run of length m starts.

   Thus, $N^\omega(S) = \bigcap_{m<\omega} N^m(S)$ is the set of those states which initiate arbitrarily long finite runs. So, $N^{\omega+1}(S)$ consists of the states which have a successor with that property (and hence have the property themselves), etc. With no assumptions on the transition system, the description of the fixpoint does not seem easy. However, assuming that it is **image finite**, i.e. every state has finitely many successors, then *König's lemma* can be applied at the ω-step, implying that $N^\omega(S)$ consists of *all states which initiate an infinite run*, which is easily seen to be a fixpoint.

   Actually, *this assumption is not necessary!* Indeed, knowing the answer, let us show that $\nu z.Nz$ is $S_\infty = \{s \mid$ there is an infinite run beginning at $s\}$ in an *arbitrary* transition system. For that, we simply apply the definition of a greatest fixpoint. Two facts must be proved:

(i) S_∞ is a fixpoint of $\lambda z.Nz$: indeed, every state in S_∞ has a successor in S_∞, so $S_\infty \subseteq N(S_\infty)$; conversely, if a state has a successor in S_∞ then it is in S_∞ itself, hence $S_\infty \supseteq N(S_\infty)$.

(ii) Every fixpoint of $\lambda z.Nz$ is included in S_∞ : Indeed, let $Y = N(Y)$ and $s \in Y$. Then $s \in N(Y)$, i.e. s has a successor s_1 in Y. Applying the same argument to s_1 we obtain a successor s_2 in Y etc. Eventually, we obtain an infinite run beginning at s, i.e. $s \in S_\infty$. Thus, $Y \subseteq S_\infty$.

4. $\varphi = \mu z.Xz = \neg\nu z.\neg X\neg z = \neg\nu z.Nz$, hence $\mu z.Xz = \{s \mid$ no infinite run begins from $s\}$.

Proposition: *On the transition system* $\langle \mathbf{N}, \mathbf{succ}\rangle$ *the following operators are definable:*

- $F\varphi := \mu z.(\varphi \vee Nz)$;
- $G\varphi := \nu z.(\varphi \wedge Xz)$;
- $\varphi U\psi := \mu z.(\psi \vee (\varphi \wedge Nz))$.

Proof sketch: We will streamline the presentation a little by doing the unfolding syntactically:

1. $(\lambda z.(\varphi \vee Nz))^m(\varnothing) = \varphi \vee N(\varphi \vee ...\ m \text{ times } ...(\varphi \vee N\bot)..)$, meaning that φ will occur in at most m steps from now. Then, $(\lambda z.(\varphi \vee Nz))^\omega(\varnothing)$ means that φ will occur in finitely many steps, i.e. ever. This is a fixpoint.
2. By duality and (1).
3. $(\lambda z.(\psi \vee (\varphi \wedge Nz))^m(\varnothing) = \psi \vee (\varphi \wedge N(...\ m \text{ times } ...\ \psi \vee (\varphi \wedge N\bot)...) = \psi \vee (\varphi \wedge N(...\ m \text{ times } ...\ \psi \vee \bot)...)$, meaning that ψ will occur in at most m steps and until then φ will be true. Therefore, $(\lambda z.(\psi \vee (\varphi \wedge Nz))^\omega(\varnothing) = \varphi U\psi$ which is a fixpoint. *QED*

Question: What do the definitions above mean on an arbitrary transition system?

Thus, we see that the *language of* **LPTL** *is definable in* μ *-calculus,* i.e. **LPTL** can be regarded as a fragment of μ-calculus on $\langle \mathbf{N}, \mathbf{succ}\rangle$, known as the *linear* μ *-calculus* $L\mu$.

Exercise: Show that $\nu z.(\varphi \wedge XXz)$ defines in $L\mu$ Wolper's operator saying that φ *is true at every even state, at least.*

Exercises: Determine the following fixpoints on: (a) $\langle \mathbf{N}, \mathbf{succ}\rangle$; (b) any serial transition system; (c) any image finite transition system, (d) any transition system.

1. $\mu z.(\varphi \wedge Xz)$, and its dual $\nu z.(\varphi \vee Nz)$;
2. $\mu z.(\varphi \vee Xz)$, and its dual $\nu z.(\varphi \wedge Nz)$;
3. $\mu z.(\varphi \wedge Nz)$, and its dual $\nu z.(\varphi \vee Xz)$;
4. $\mu z.(\psi \vee (\varphi \wedge Xz))$;
5. $\nu z.(\psi \vee (\varphi \wedge Nz))$;
6. $\nu z.(\psi \vee (\varphi \wedge Xz))$.

6.6 Embedding CTL in μ-calculus

CTL can be embedded in μ-calculus likewise, by modifying the syntax accordingly: instead of X the language now contains $\forall X$, and formulae are defined acccordingly:

$$\varphi := p \mid z \mid \bot \mid \varphi_1 \to \varphi_2 \mid \forall X\varphi \mid \mu z.\varphi$$

Again, $\exists X$ is definable as $\neg\forall X\neg$.

The semantics is essentially the same, as the definition of X above corresponds precisely to $\forall X$ on **CTL**-models:

$$\|\forall X\varphi\|_V = \{s \mid R(s) \subseteq \|\varphi\|_V\}\,.$$

All other operators of **CTL** are definable as fixpoints now.

Proposition: *The operators of* **CTL** *are definable on any transition system as follows:*

$\exists F\varphi := \mu z.(\varphi \vee \exists X z)$;

$\forall F\varphi := \mu z.(\varphi \vee \forall X z)$;

$\exists G\varphi := \nu z.(\varphi \wedge \exists X z)$;

$\forall G\varphi := \nu z.(\varphi \wedge \forall X z)$;

$\exists \varphi U \psi := \mu z.(\psi \vee (\varphi \wedge \exists X z))$;

$\forall \varphi U \psi := \mu z.(\psi \vee (\varphi \wedge \forall X z))$.

Proof: Most of these are easy adaptations of previous examples. Let us prove e.g. $\forall F\varphi := \mu z.(\varphi \vee \forall X z)$. If we start unfolding we will end up in need of image finiteness and König's lemma as before. Instead, let us apply the definition directly. First, we show that $\forall F\varphi$ is a fixpoint of $\lambda z.(\varphi \vee \forall X z)$. This amounts to checking the validity of $\forall F\varphi \leftrightarrow \varphi \vee \forall X \forall F\varphi$, which is easy. Now, we want to show that every fixpoint y of $\lambda z.(\varphi \vee \forall X z)$ contains the extensional of $\forall F\varphi$. We shall reason by contraposition and, for a change, in logical instead of set-theoretic terms. Suppose $s \notin y$, i.e. (being a bit sloppy) $s \not\models y$. Then $s \not\models \varphi \vee \forall X y$, i.e. $s \not\models \varphi$ and $s \not\models \forall X y$, so there is a successor s_1 of s such that $s_1 \not\models y$. Applying the same argument to s_1 we obtain that $s_1 \not\models \varphi$ and s_1 has a successor s_2 such that $s_2 \not\models y$, etc. Thus, eventually we obtain an infinite run $s, s_1, s_2, ...$ such that φ is false at every state of that run. Therefore, $s \not\models \forall F\varphi$.

With this, we have proved that $\forall F\varphi$ is indeed the least fixpoint of $\lambda z.(\varphi \vee \forall X z)$.

Exercise: Complete the proof for the other operators.

Thus, we see that **CTL** *can be considered as a fragment of the μ -calculus, too.*

6.7 Embedding the full CTL* and beyond

What we have seen so far is just a modest demonstration of the expressive power of μ-calculus. In fact, the full **CTL*** is embeddable into μ-calculus, as well, but the translation is far more complicated. See [EL86] and [Dam94a].

Let us give a couple of examples on expressing fairness conditions in μ -calculus which are not expressible in **CTL.** For them we will use *alternating operators:*

- $\exists F^{\infty}\varphi$, saying '$\varphi$ *holds infinitely often along some computation',* is expressed by $\nu z.\mu y.\exists X((\varphi \wedge z) \vee y)$;
- its dual $\forall G^{\infty}\varphi$, saying φ *'holds almost always along every computation'* is expressed accordingly by $\mu z.\nu y.\forall X((\varphi \vee z) \wedge y)$.

Exercise: Prove these!

We will say a bit more on the expressiveness of μ-calculus in the last chapter.

6.8 Axiomatization, completeness, decidability, complexity

The following simple axiomatization was proposed by Kozen in [Koz83]

Axiom: $\varphi(\mu z.\varphi(z)) \rightarrow \mu z.\varphi(z)$

saying that $\mu z.\varphi(z)$ is a pre-fixpoint, and

Park's rule:

$$\frac{\varphi(\theta/z) \rightarrow \theta}{\mu z.\varphi(z) \rightarrow \theta}$$

saying that $\mu z.\varphi(z)$ is a least pre-fixpoint.

Kozen proved a partial completeness result, with respect to so called *aconjunctive formulae.* The full completeness was proved only quite recently by I. Walukiewicz, see [Wal95],[Wal96].

Moreover, Kozen proved that μ *-calculus has the finite model property and is decidable with EXPTIME complete satisfiability problem.*

With this we finish our brief introduction to μ-calculus. For more, see [Koz83],[SE89], [Zuc93a],[Kai95],[AKM95],[JW96],[NW96] , [Wal95]. You can also check out Walukiewicz's webpage at http://www.brics.dk/~igw/.

7 Capita selecta

Here we mention very briefly a few more topics which could be part of the course, if the time for it were infinite.

7.1 A few words on model checking

One of the major applications of temporal logics in computer science is *temporal specification and verification of programs.* These mean: *to specify by means of temporal formulae the properties which a program must satisfy, and given a program to verify formally that the specifications are met.* The idea of formal verification goes back to the seminal work of Floyd and Hoare in the late 60s.

There are two basic approaches to verification of properties specified in temporal logic:

- *proof-theoretic approach:* the program is represented as a *theory* (set of formulae) and the verification consists in formal derivation of the specified properties. This is the approach proposed by Pnueli and Manna (see [MP92]).
- *model checking approach:* the program is represented as a finite model structure for temporal logic, i.e. a finite labelled transition system, and then the task is to check if it is a *model* for the temporal formula specifying the desired property. The idea of modal checking goes back to Clarke, Emerson and Sistla ([CES83] and the journal version [CES86]), also independently proposed by Queille and Sifakis in the early 80's.

These two approaches are quite differerent both conceptually and technically, and their pros and contras are subject to a lively debate. Major advantages of model checking to theorem proving are:

- It is fully automatized; no need for human experts to write and check long proofs.
- It does not claim generality as the theorem proving approach, but wherever it is applicable (mainly to systems representable as finite state machines) it is much more efficient.
- Low computational complexity.
- Constructiveness: gives diagnostics of what went wrong, and partly generates counter-models when the verification fails.

The main disadvantage and a limiting factor for model checking is the *state-explosion problem,* which often occurs in the construction of a transition system representing a program. But, some recent developments (e.g. partial-order and abstraction techniques, 'on-the-fly' algorithms) successfully improve the efficiency of the representation and model-checking.

Another approach is *symbolic model checking* where transition systems are encoded appropriately, e.g. as Binary Decision Diagrams (BDD's) [Bry92]; see also [BCM$^+$92].

The model checking approach has been developed extensively for linear and branching time logics, and particularly to μ-calculus (for a survey of various approaches see [EL86]). The complexity of model checking is easy (O(size of structure $\times$ size of formula)) for **CTL,** but harder (EXPTIME, PSPACE complete) for **LPTL** and **CTL***. For the full μ-calculus it is in $NP \cap co - NP$.

Model checking has also been applied successfully to *verification of probabilistic systems* [PZ93], [PZ86], [BCHG$^+$97], which is the topic of the ESSLLI'2000 course offered by Mark Ryan and Marta Kwiatkowska. For more on model checking, see their course notes, or e.g. [EL86] [SW91], [Zuc93b], [KVW94], [Zuc93a], [Var98].

7.2 Branching time vs linear time logics

The debate on the pros and contras of using linear vs branching time temporal logics has been alive and unabating since the begining of the 80's. See [Lam80] and [EH86] for the beginning of it. Of course, the linear and branching time approaches have somewaht different spheres of application, viz. linear time approach is the more natural when the properties to be checked are about *single program executions,* while the branching time approach is more natural to reason about all executions, i.e. abouth the very *structure of the program.* 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 the intended practical purposes.*

Here we discuss briefly the first three.

Regarding expressiveness, it should be intuitively clear that **LPTL** and **CTL** are *incompatible* in their expressive powers. Indeed, on one hand, **CTL** can express *global* properties, regarding *all possible computations,* while **LPTL's** claims are confined within a particular computation. On the other hand, however, **LPTL** can say much more than **CTL** about a *particular computation* and, notably, it can express fairness properties which are beyond **CTL's** expressiveness. Indeed, this intuition can be turned into a precise statement

***Proposition**:*

1. *The* **CTL** *-formula* $\forall F \forall G p$ *is not expressible in* **LPTL.**
2. *The formula* $\forall FGp$ *from the branching time translation of* **LPTL** *is not expressible in* **CTL.**

Sketch: The key to the proof of these is the observation that with every **CTL*** formula φ, one can associate the **LPTL-**formula $\varphi^{\mathbf{L}}$ obtained from φ by deleting all path quantifiers. Now, show that for every run σ in a **CTL** $^*-$model M, $\langle M, \sigma \rangle \models \varphi^{\mathbf{L}}$ iff $M_\sigma, \sigma \vDash \varphi$, where $\langle M, \sigma \rangle$ is the respective **LPTL**-model while M_σ is the **CTL**$^*-$model obtained by restricting M over the run σ. See [Dam94b] for details.

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 imposed by the complexity of **CTL** *, (deterministic *EXPEXPTIME* complete) is virually 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, viz the very low complexity of *model checking* for **CTL** (linear both in the size of the formula and in the size of the structure). On the other hand, **LPTL,** while having better than **CTL (** *PSPACE*) complexity of the satisfiability, fares worse (*EXPTIME)* 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 more on the discussion, see Moshe Vardi's paper and check out his website at

http://www.cs.rice.edu/~vardi/.

7.3 Characterizing expressiveness of temporal logics in terms of automata and monadic second order theories

As we have already seen, there is a close relationship between the temporal logics studied here and automata, and it also gives some characterization of the expressiveness of these logics in terms of the classes of languages recognizable by the respective automata (althoug the automata are, generally speaking, more expressive than the respective temporal logics). On the other hand, there is an intimate relationship between the various types of automata and fragments of *monadic second-order logics* over the types of structures which serve as inputs for these automata, and that links temporal logics to second-order logics in terms of expressiveness.

7.3.1 Monadic second order logics and theories.

Second-order logic is an extension of first-order logic allowing *quantification over predicates and over functions,* while in first-order logic only quantification over individuals (elements of the domain) is possible. Second-order logic is extremely expressive, to such an expent that in order to fully understand its semantics one has toadress some deep set-theoretical questions concerning the very foundations of mathematics (such as the continuum hypothesis). This, of course comes at a price: not only it is undecidable, *but it cannot be axiomatized recursively (i.e. efficiently).* That is why, it is important to look for natural tradeoffs of expressiveness for better behaved fragments.

One very natural restriction of second order logic is to allow *only quantification over unary predicates,* i.e. sets. The result is **monadic second-order logic** (MSOL). Formally, its syntax extends first-order logic with constructions $\forall P\varphi$ and $\exists P\varphi$ where P is a *unary predicate variable.* The mechanism of binding etc. of the second order quantifiers is just like in first-order logic, and the semantics of these constructions should be intuitively clear.

The MSOL is still very expressive. For instance, the *full induction axiom for natural numbers* is definable here:

$\forall P(P(0) \wedge \forall x(P(x) \rightarrow P(x+1)) \rightarrow \forall x P(x))$.

so the natural numbers can be described up to isomorphism in that logic. Furthermore, note that the *frame-validity in classical temporal logic* is expressible by a (universal) monadic second-order formula, so various non first-order definable properties of orderings such as well-foundedness and Dedekind completeness can be expressed here.

7.3.2 Büchi's and Rabin's decidability results

Given a structure, or a class of structures $\mathfrak{C}$, the **monadic second-order theory (MSOT) of $\mathfrak{C}$** is the set of *sentences* (formulae with no free individual or predicate variables) valid in $\mathfrak{C}$.

The MSO theory of the structure of natural numbers with a successor relation $\langle \mathbf{N}, \mathbf{succ} \rangle$ is one of the simplest yet interesting monadic second order theories. It is known as the **MSOT of one successor S1S,** or also called by Büchi the *sequential calculus.* Büchi studied that theory in detail, and in order to prove its *decidability*, as it turned out to be the case, he introduced and studied the Büchi automata on infinite words, which we discussed earlier.

Later on, Rabin introduced *Rabin's automata.* (Over infinite words these are similar to generalized Büchi automata, but with a set of *pairs* of accepting states: GREEN and RED; for a run to be accepting, *no* state from the RED sets and *some* state from the GREEN sets must be visited infinitely often. Over trees, likewise, along every path of the run.) Using these automata Rabin proved in [Rab69] the decidability of the **MSOT of 2 successors S2S ,** i.e. of binary trees. This is an extremely powerful result, as it turned out that many other MSO theories, such as the MSOT of n-branching trees **SnS**, the MSOT of ω-branching trees **SωS**, and the MSOT of the rational numbers etc. can be interpreted in SnS and thus proved decidable.

7.3.3 The extended temporal logic ETL and the quantified LPTL

Noting that there are simple and natural second-order properties, such as ' *true at every even state'* which are not expressible in **PLTL** Wolper argued in [Wol83] that *"temporal logic can be more expressive"*, and showed how it could be extended to capture the full expressiveness of Büchi automata, introducing his *extended temporal logic* **ETL.** The idea in a nutshell: for Büchi automaton **A** over the alphabet $\{a_1, .., a_n\}$ add to **LPTL** a new n-ary operator $\Psi_{\mathbf{A}}(p_1, .., p_n)$ with the following semantics: an infinite run σ satisfies $\Psi_{\mathbf{A}}(\varphi_1, .., \varphi_n)$ if there is a word **w** accepted by the automaton such that for every k, if $\mathbf{w}(k) = a_j$ then φ_j is true at $\sigma(k)$. Thus, in particular, the operator $E(\varphi)$ saying that 'φ is true at every even state' is $\Psi_{\mathbf{E}}(\varphi, \top)$ generated by the automaton **E** accepting the language defined by the ω -regular expression $(a_1 a_2)^\omega$.

***Exercise**:* Construct Büchi automata which define the operators X, G and U.

Remark: **CTL*** can be extended similarly, producing **ECTL***.

Another way of extending **LPTL** is by allowing **propositional quantification**, i.e. extending the language with quantifiers over propositional variables so that for every formula φ and a propositional variable p, $\exists p\varphi$ is a formula, too, with the expected semantics. Note that this is essentially a *monadic second-order quantification.* The result is dubbed **QLPTL.**

7.3.4 Some results on expressive equivalences

***Theorem**: The following formalisms have equivalent expressiveness on the structure* $\langle \mathbf{N}, < \rangle$:

- *LPTL;*
- *first-order logic;*
- *counter-free automata;*
- *star-free ω -regular expressions.*

Some of these we have already discussed, for the rest, see e.g. [Dam94b], [Tho90].

***Theorem**: The following formalisms have equivalent expressiveness on the structure* $\langle \mathbf{N}, < \rangle$:

- *ETL,*
- *S1S,*
- *Büchi automata,*
- *ω -regular expressions,*
- *linear μ -calculus,*
- **QLPTL.**

For details and proofs see [Tho90], [Wol83], [Dam94b], [Var].

***Theorem**:* (Hafer and Thomas [HT87]) *The expressivenes of* **CTL*** *is equivalent to the one of the monadic second-order logic on infinite binary trees with second-order quantifiers over infinite pahts.*

***Theorem**:* (Emerson and Jutla [EJ91]) *In terms of expressiveness μ -calculus is equivalent to alternating automata on infinite trees.*

7.4 Other topics

7.4.1 Process logics

Closely related to temporal logics of computations are various *logics of processes* proposed by Pratt [Pra79], Harel, Kozen and Parikh [HKP82], and logics of programs such as the *propositional dynamic logic PDL* and others [KT90] and some new approaches such as the *computational path logic* [HS99] combine features of dynamic and temporal logics.

7.4.2 Executable temporal logics

These are based on an *imperative view of temporal logics,* proposed by Gabbay [Gab89], and developed by him, Barringer, Fisher, Reynolds and others. The idea is that temporal logics formulae can be viewed as imperative specifications, e.g $G\varphi$ means ' *maintain* φ *true from now on',* $\varphi U\psi$ – ' *maintain* φ *true until* ψ *becomes true and ensure that* ψ *will become true' etc.* This approach has been used as a basis of *programming with temporal logic* for which the language MetateM has been developed [BFG$^+$95]. For more, see [GR95].

7.4.3 Temporalizing logical systems

Temporality is an important aspect of any dynamic system, so it is necessary to develop logical tools for enrichment of a formal system, program, protocol, database etc. with a temporal dimension in a uniform and efficient way. For an approach towards such methodology see [FG92].

7.4.4 Metric and layered temporal logics for time granular systems.

Two (related) aspects of time that are not reflected at all in the classical framework is *granularity:* the 'resolution power of the temporal qualification of a statement' [Mon96], and *measurability* i.e. its quantitative side of time, to which the qualitative Priorian framework is completely indifferent. Developing these leads to *metric and layered temporal logics,* which have various practical applications e.g. to *time-critical, real-time* etc. systems [Koy92], [Mon96]. For more on this, see the notes of ESSLLI'2000 course on the topic, given by Angelo Montanari and Alberto Policriti.

7.4.5 ...till the end of time.

Time is endless. The story about time, too...

References

[ABMar] C. Areces, P. Blackburn, and M. Marx. Hybrid logics: Characterization, interpolation and complexity. *Journal of Symbolic Logic*, to appear.

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

[AGM92] S. Abramsky, D.M. Gabbay, and T.S.E. Maibaum, editors. *Handbook of Logic in Computer Science*, volume 2. Clarendon Press, Oxford, 1992.

[AGM+95] H. Andreka, V. Goranko, S. Mikulas, I. Nemeti, and I. Sain. Effective first-order temporal logics of programs. In L. Bolc and A. Szalas, editors, *Time and Logic: A Computational Approach*, pages 51–129. Univ. College London Press, 1995.

[AKM95] S. Ambler, M. Kwiatkowska, and N. Measor. Duality and the completeness of the modal μ-calculus. *Theoretical Computer Science*, 151(1):3–27, 1995.

[All84] James F. Allen. Towards a general theory of action and time. *Artificial Intelligence*, 23:123–154, 1984.

[BA93] M. Ben-Ari. *Mathematical Logic for Computer Science*. Prentice Hall, 1993.

[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.

[BCHG+97] C. Baier, E. Clarke, V. Hartonas-Garmhausen, M. Kwiatkowska, and M. Ryan. Symbolic model checking for probabilistic processes. In P. Degano, R. Gorrieri, and A. Marchetti-Spaccamela, editors, *Automata, Languages and Programming, 24th International Colloquium*, volume 1256 of *Lecture Notes in Computer Science*, pages 430–440, Bologna, Italy, 7–11 July 1997. Springer-Verlag.

[BCM+92] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98(2):142–170, June 1992.

[BdRVar] P. Blackburn, M. de Rijke, and Y. Venema. *Modal Logic*. Cambridge Tracts on Theoretical Computer Science, to appear.

[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.

[Ben93] J. van Benthem. *The Logic of Time*. Kluwer Academic Publishers, Dordrecht, second edition, 1993.

[BFG+95] H Barringer, M Fisher, D Gabbay, G Gough, and R Owens. Metatem: An introduction. *Formal Aspects of Computing*, 7(5):533–549, 1995.

[BG99] M. Brown and V. Goranko. An extended branching-time Ockhamist temporal logic. *Journal of Logic, Language and Information*, 8(2):143–166, 1999.

[Bla93] P. Blackburn. Nominal tense logic. *Notre Dame Journal of Formal Logic*, 14:56–83, 1993.

[Bla00] P. Blackburn. Internalizing labelled deduction. *Journal of Logic and Computation*, 10(1):137–168, 2000.

[Bra96] J. Bradfield. The modal mu-calculus alternation hierarchy is strict. STACS'96, to appear in Theoretical Computer Science, 1996.

[Bry92] Randal E. Bryant. Symbolic Boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys*, 24(3):293–318, September 1992.

[BS84] R.A. Bull and K. Segerberg. Basic modal logic. In Gabbay and Guenthner [GG84], pages 1–88.

[BS98] P. Blackburn and J. Seligman. What are hybrid languages? In M. Kracht, M. de Rijke, H. Wansing, and M. Zakharyaschev, editors, *Advances in Modal Logic*, volume 1, pages 41–62. CSLI Publications, Stanford University, 1998.

[BT99] P. Blackburn and M. Tzakova. Hybrid languages and temporal logic. *Logic Journal of the IGPL*, 7:27–54, 1999.

[BtM97] J. van Benthem and A. ter Meulen, editors. *Handbook of Logic and Language*. Elsevier Science, 1997.

[Bur79] J. Burgess. Logic and time. *Journal of Symbolic Logic*, 44:556–582, 1979.

[Bur82] J. Burgess. Axioms for tense logic I: 'since' and 'until'. *Notre Dame Journal of Formal Logic*, 23:375–383, 1982.

[Bur84] J.P. Burgess. Basic tense logic. In Gabbay and Guenthner [GG84], pages 89–133.

[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.

[CHR91] Zhou Chaochen, C. A. R. Hoare, and Anders P. Ravn. A calculus of durations. *Information Processing Letters*, 40(5):269–276, 13 December 1991.

[Dam94a] M. Dam. CTL* and ECTL* as fragments of the modal μ-calculus. *Theoretical Computer Science*, 126(1):77–96, 11 April 1994.

[Dam94b] M. Dam. Temporal logic, automata and classical theories. Lecture notes for the 6th ESSLLI, Kopenhagen, available online at http://www.sics.se/~mfd/home.html, 1994.

[dR92] Maarten de Rijke. The modal logic of inequality. *The Journal of Symbolic Logic*, 57(2):566–584, June 1992.

[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.

[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.

[EJ91] E. A. Emerson and C. S. Jutla. Tree automata, mu-calculus and determinacy (extended abstract). In *32nd Annual Symposium on Foundations of Computer Science*, pages 368–377, San Juan, Puerto Rico, 1–4 1991. IEEE.

[EL86] E. Emerson and C.-L. Lei. Efficient model checking in fragments of the propositional mu-calculus (extended abstract). In *Proc. Symposium on Logic in Computer Science* [IEE86], pages 267–278.

[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] E. A. Emerson and A. P. Sistla. Deciding full branching time logic. *Information and Control*, 61(3):175–201, June 1984.

[FG92] M. Finger and D.M. Gabbay. Adding a temporal dimension to a logic system. *Journal of Logic, Language and Information*, 1:203–233, 1992.

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

[Gab89] D.M. Gabbay. The declarative past and imperative future: executable temporal logic for interactive systems. In B. Banieqbal, H. Barringer, and A. Pnueli, editors, *Proc. Colloquium on Temporal Logic in Specification*, volume 398 of *LNCS*, pages 431–448. Springer, 1989.

[GG84] D.M. Gabbay and F. Guenthner, editors. *Handbook of Philosophical Logic*, volume 2. Reidel, Dordrecht, 1984.

[GG93] G. Gargov and V. Goranko. Modal logic with names. *Journal of Philosophical Logic*, 22:607–636, 1993.

[GH91] D.M. Gabbay and I.M. Hodkinson. An axiomatization of the temporal logic with Since and Until over the real numbers. *Journal of Logic and Computation*, 1:229–259, 1991.

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

[GHR95] D.M. Gabbay, C.J. Hogger, and J.A. Robinson, editors. *Handbook of Logic in Artificial Intelligence and Logic Programming: Epistemic and Temporal Reasoning*, volume 4. Oxford University Press, Oxford, 1995.

[Gol92] R. Goldblatt. *Logics of Time and Computation*, volume 7 of *CSLI Lecture Notes*. Center for the Study of Language and Information, Stanford University, 2nd. ed. edition, 1992.

[Gor94] Valentin Goranko. Temporal logic with reference pointers. In D. Gabbay and H.-J. Ohlbach, editors, *Temporal Logic*, volume 827 of *Lecture Notes in Artificial Intelligence*, pages 133–148. Springer-Verlag, 1994.

[Gor96] V. Goranko. Hierarchies of modal and temporal logics with reference pointers. *Journal of Logic, Language and Information*, 5(1):1–24, 1996.

[Gor99] R. Goré. Tableau methods for modal and temporal logics. In M. D'Agostino, D.M. Gabbay, R. Hähnle, and J. Posegga, editors, *Handbook of Tableau Methods*. Kluwer Academic Publishers, 1999.

[Gor00] V. Goranko. Computation tree logics and temporal logics with reference pointers. *Journal of Applied Non-classical Logics*, 10(0):00–00, 2000.

[GP92] V. Goranko and S. Passy. Using the universal modality: Gains and questions. *Journal of Logic and Computation*, 2:5–30, 1992.

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

[GR95] D. Gabbay and M. Reynolds. Towards a computational treatment of time. In Gabbay et al. [GHR95], pages 351–431.

[GW89] P. Gribomont and P. Wolper. *From Modal Logic to Deductive Databases: Introducing a Logic Based Approach to Artificial Intelligence*, chapter Temporal Logic, pages 165–234. Wiley, 1989.

[HC96] G. Hughes and M. Cresswell. *A New Introduction to Modal Logic*. Routledge, London, 1996.

[HKP82] David Harel, Dexter Kozen, and Rohit Parikh. Process logic: Expressiveness, decidability, completeness. *Journal of Computer and System Sciences*, 25(2):144–170, October 1982.

[HMM83] Joseph Y. Halpern, Zohar Manna, and Ben Moszkowski. A hardware semantics based on temporal intervals. In Josep Díaz, editor, *Automata, Languages and Programming, 10th Colloquium*, volume 154 of *Lecture Notes in Computer Science*, pages 278–291, Barcelona, Spain, 18–22 July 1983. Springer-Verlag.

[HS86] J. Y. Halpern and Y. Shoham. A propositional model logic of time intervals. In *Proc. Symposium on Logic in Computer Science* [IEE86], pages 279–292.

[HS99] D. Harel and E. Singerman. Computation paths logic: An expressive, yet elementary, process logic. *Annals of Pure and Applied Logic*, 96:167–186, 1999.

[HT87] T. Hafer and W. Thomas. Computation tree logic CTL* and path quantifiers in the monadic theory of the binary tree. In Thomas Ottmann, editor, *Automata, Languages and Programming, 14th International Colloquium*, volume 267 of *Lecture Notes in Computer Science*, pages 269–279, Karlsruhe, Germany, 13–17 July 1987. Springer-Verlag.

[HU79] J. Hopcroft and J. Ullman. *Introduction to Automata theory, Languages and Computation.* Addison-Wesley, 1979.

[IEE86] IEEE Computer Society. *Proc. Symposium on Logic in Computer Science*, Cambridge, Massachusetts, 16–18 June 1986.

[JW96] David Janin and Igor Walukiewicz. On the expressive completeness of the propositional mu-calculus with respect to monadic second order logic. In Ugo Montanari and Vladimiro Sassone, editors, *CONCUR '96: Concurrency Theory, 7th International Conference*, volume 1119 of *Lecture Notes in Computer Science*, pages 263–277, Pisa, Italy, 26–29August 1996. Springer-Verlag.

[Kai95] Roope Kaivola. Axiomatising linear time mu-calculus. In Insup Lee and Scott A. Smolka, editors, *CONCUR '95: Concurrency Theory, 6th International Conference*, volume 962 of *Lecture Notes in Computer Science*, pages 423–437, Philadelphia, Pennsylvania, 21–24 August 1995. Springer-Verlag.

[Kam68] H. Kamp. *Tense Logic and the Theory of Linear Order.* PhD thesis, University of California, Los Angeles, 1968.

[KdR97] N. Kurtonina and M. de Rijke. Bisimulations for temporal logic. *Journal of Logic, Language and Information*, 6:403–425, 1997.

[Koy92] R. Koymans. *Specifying Message Passing and Time-Critical Systems with Temporal Logic.* LNCS 651, Springer-Verlag, Berlin, 1992.

[Koz83] Dexter Kozen. Results on the propositional μ-calculus. *Theoretical Computer Science*, 27(3):333–354, December 1983.

[KS86] R. Kowalski and M. Sergot. A logic-based calculus of events. *New Generation Computing*, 4:67–95, 1986.

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

[KVW94] O. Kupferman, , M. Vardi, and P. Wolper. An automata-theoretic approach to branching-time model checking. COV'94, available at http://www.cs.rice.edu/~vardi/papers/, 1994.

[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.

[LP00] O. Lichtenstein and A. Pnueli. Propositional temporal logics: Decidability and completeness. *Journal of the IGPL*, 8(1):55–85, 2000.

[LPS81] D. Lehmann, A. Pnueli, and J. Stavi. Impartiality, justice and fairness: The ethics of concurrent termination. In *ICALP'81*, LNCS, Vol. 115, pages 264–277. Springer-Verlag, 1981.

[LPZ85] O. Lichtenstein, A. Pnueli, and L. Zuck. The glory of the past. In *Brooklyn College Conference on Logics of Programs*, LNCS. Springer-Verlag, 1985.

[Mon96] A. Montanari. *Metric and Leyered Temporal Logic for Time Granularity*. PhD thesis, Institute of logic, Language and Computation, 1996.

[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.

[MP90] Z. Manna and A. Pnueli. Hierarchies of temporal properties. In *Proceedings of the 9th ACM Symposium Principles of Distributed Computing (PODC 1990)*, pages 377–408, 1990.

[MP92] Zohar Manna and Amir Pnueli. *The Temporal Logic of Reactive and Concurrent Systems, Vol. 1: Specifications*. Springer, New York, 1992.

[MP95] Zohar Manna and Amir Pnueli. *The Temporal Logic of Reactive and Concurrent Systems, Vol. 2: Safety*. Springer, New York, 1995.

[NW96] Damian Niwiński and Igor Walukiewicz. Games for the μ-calculus. *Theoretical Computer Science*, 163(1–2):99–116, 1996.

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

[Pnu77] A. Pnueli. The temporal logic of programs. In *Proc. 18th Symp. Foundations of Computer Science*, pages 46–57, 1977.

[Pop92] S. Popkorn. *First Steps in Modal Logic*. Cambridge University Press, Cambridge, 1992.

[Pra79] Vaughan R. Pratt. Process logic. In *Conference Record of the Sixth Annual ACM Symposium on Principles of Programming Languages*, pages 93–100, San Antonio, Texas, January 1979.

[Pra81] V. R. Pratt. A decidable mu-calculus: Preliminary report. In *22nd Annual Symposium on Foundations of Computer Science*, pages 421–427, Nashville, Tennessee, 28–30 October 1981. IEEE.

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

[PT91] S. Passy and T. Tinchev. An essay in combinatory dynamic logic. *Information and Computation*, 93:263–332, 1991.

[PZ86] Amir Pnueli and Lenore D. Zuck. Verification of multiprocess probabilistic protocols. *Distributed Computing*, 1(1):53–72, 1986.

[PZ93] Amir Pnueli and Lenore D. Zuck. Probabilistic verification. *Information and Computation*, 103(1):1–29, March 1993.

[Rab69] M. O. Rabin. Decidability of second-order theories and automata on infinite trees. *Transactions of the American Mathematical Society*, 141:1–35, 1969.

[Rei47] H. Reichenbach. *Elements of Symbolic logic*. Macmillan, New York, 1947.

[Rey92] M. Reynolds. An axiomatization for Until and Since over the reals without the IRR rule. *Studia Logica*, 1992.

[Rey94] M. Reynolds. Axiomatizing U and S over integer time. In D. Gabbay and H.-J. Ohlbach, editors, *Temporal Logic, First International Conference ICTL'94*, volume 827 of *Lecture Notes in Artificial Intelligence*, pages 117–132. Springer-Verlag, 1994.

[Rey98] M. Reynolds. An axiomatization of full computation tree logic. *submitted*, 1998.

[Rey00] M. Reynolds. More past glories. In *LICS'2000*. IEEE, to appear 2000.

[SC85a] A. P. Sistla and E. M. Clarke. The complexity of propositional linear temporal logics. *Journal of the ACM*, 32(3):733–749, July 1985.

[SC85b] A. P. Sistla and E. M. Clarke. The complexity of propositional linear temporal logics. *Journal of the ACM*, 32(3):733–749, July 1985.

[SE89] Robert S. Streett and E. Allen Emerson. An automata theoretic decision procedure for the propositional mu-calculus. *Information and Computation*, 81(3):249–264, June 1989.

[Sis94] A Prasad Sistla. Safety, liveness and fairness in temporal logic. *Formal Aspects of Computing*, 6(5):495–512, 1994.

[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.

[SW91] Colin Stirling and David Walker. Local model checking in the modal mu-calculus. *Theoretical Computer Science*, 89(1):161–177, 1991.

[Tho84] R. H. Thomason. Combinations of tense and modality. In Gabbay and Guenthner [GG84], pages 135–166.

[Tho90] W. Thomas. Automata on infinite objects. In Leeuwen [Lee90], pages 165–191.

[Var] M. Vardi. Automata-theoretic approach to automated verification. Lecture notes available at http://www.wisdom.weizmann.ac.il/home/vardi/public_html/av/notes/.

[Var94] M. Vardi. An automata-theoretic approach to linear temporal logic. Banff'94, available at http://www.cs.rice.edu/~vardi/papers/, 1994.

[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.

[Ven93] Y. Venema. Completeness via completeness: Since and Until. In M. de Rijke, editor, *Diamonds and Defaults*, pages 279–286, Dordrecht, 1993. Kluwer Academic Publishers.

[VW86] M. Vardi and P. Wolper. Automata-theoretic techniques for modal logics of programs. *Journal of Computer and System Science*, 32(2):183–221, April 1986.

[VW94] Moshe Y. Vardi and Pierre Wolper. Reasoning about infinite computations. *Information and Computation*, 115(1):1–37, November 1994.

[Wal95] Igor Walukiewicz. Completeness of Kozen's axiomatisation of the propositional μ-calculus. In *Proceedings, Tenth Annual IEEE Symposium on Logic in Computer Science*, pages 14–24, San Diego, California, 26–29 June 1995. IEEE Computer Society.

[Wal96] Igor Walukiewicz. A note on the completeness of Kozen's axiomatisation of the propositional μ-calculus. *The Bulletin of Symbolic Logic*, 2(3):349–366, September 1996.

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

[Wol85] Pierre Wolper. The tableau method for temporal logic: An overview. *Logique et Analyse*, 28:119–136, 1985.

[Wol95] Pierre Wolper. On the relation of programs and computations to models of temporal logic. In Leonard Bolc and Andrzej Szalas, editors, *Time and Logic, a computational approach*, chapter 3, pages 131–178. UCL Press Limited, 1995.

[Xu88] M. Xu. On some U,S-tense logics. *Journal of Philosophical Logic*, 17:181–202, 1988.

[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.

[Zuc93a] J. Zucker. Propositional mu-calculus and its use in model checking. In P. Lauer, editor, *Functional Programming, Concurrency, Simulation and Automated Reasoning*, volume 693 of *Lecture Notes in Computer Science*, pages 117–128. Springer-Verlag, 1993.

[Zuc93b] J. Zucker. Propositional temporal logics and their use in model checking. In P. Lauer, editor, *Functional Programming, Concurrency, Simulation and Automated Reasoning*, volume 693 of *Lecture Notes in Computer Science*, pages 108–116. Springer-Verlag, 1993.