

Linear Temporal Logic with Spin

Alexander Kurz

October 9, 2025

Contents

1	Propositional Logic	1
1.1	Truth Tables	1
1.2	An algorithm for satisfiability: semantic tableaux	5
2	Temporal Logic	7
2.1	(Linear) Temporal Logic	7
2.2	Transition Systems	9
2.3	Transition System Semantics of LTL, Some Remarks on Spin	9
2.4	Semantic Tableaux for LTL	10
3	LTL Model Checking with Semantic Tableaux	12
A	Predicate Logic	14
A.1	Introduction	14
A.2	Definitions	14
B	Logic for Transition Systems (Modal Logic)	15
C	Questions and Exercises	17

1 Propositional Logic

1.1 Truth Tables

Notation: We write $\mathbb{2}$ for the two-element set $\{0, 1\}$. We call the elements of $\mathbb{2}$ truth values and refer to 0 as ‘false’ and to 1 as ‘true’.

The formulas of propositional logic (PL) are given wrt¹ a set \mathbb{P} of *atomic propositions*, sometimes also called *propositional variables*, according to the context-free grammar²

$$\text{Fma} ::= p \mid \perp \mid \top \mid \neg \text{Fma} \mid \text{Fma} \vee \text{Fma} \mid \text{Fma} \wedge \text{Fma} \mid \text{Fma} \rightarrow \text{Fma} \mid \text{Fma} \leftrightarrow \text{Fma} \quad ^3$$

¹with respect to

²http://en.wikipedia.org/wiki/Context-free_grammar, http://en.wikipedia.org/wiki/Backus-Naur_Form

³The connectives should be pronounced as false, true, not, or, and, implies, if and only if. The connectives are named: false/bottom, verum/top, negation, disjunction, conjunction, implication, biimplication (or logical equivalence).

where p takes values in \mathbb{P} . We also say that this grammar describes the *syntax* of propositional logic. To save brackets we use the convention that \neg binds stronger than \wedge, \vee which bind stronger than \rightarrow which binds stronger than \leftrightarrow .⁴

Remark on Notation: Fma is a so-called non-terminal symbol of the context-free grammar describing all propositional formulas. We use φ to denote a propositional formula. We use p, q, r to denote atomic propositions.

Example 1. Intuitively, propositional logic is the logic of finite states, or, in other words, the logic to describe static properties of a system with finite memory. Consider, eg

$$\mathbb{P} = \{\text{red}, \text{yellow}, \text{green}\}.$$

Then we may use propositional formulas to specify static properties of traffic lights such as

- $\text{red} \rightarrow \neg \text{green}$
- $\text{green} \rightarrow \neg \text{red}$
- $\neg \text{red} \vee \neg \text{green}$

(One thing propositional logic cannot do, is to specify dynamic, or *temporal*, properties, such as: whenever the traffic light is red then *eventually* it will become green. We will come back to this when we discuss temporal logic.)

A good exercise at this point is to write out a full specification of the allowed states of a traffic light. And then to ask: How we can validate this specification?

This exercise will lead to some interesting questions about specifications. For example, when are two different specifications equivalent?

These questions lead us to the notion of model of a specification.⁵ Intuitively, a model of a specification describes a situation in which the specification is true. For example, the models of $\text{red} \rightarrow \neg \text{green}$ are all combinations a traffic light could show as long as red and green are not both on. For example

$$\begin{aligned} &(\text{red} \mapsto 1, \text{yellow} \mapsto 1, \text{green} \mapsto 0) \\ &(\text{red} \mapsto 0, \text{yellow} \mapsto 1, \text{green} \mapsto 1) \\ &(\text{red} \mapsto 0, \text{yellow} \mapsto 0, \text{green} \mapsto 0) \end{aligned}$$

are all models of $\text{red} \rightarrow \neg \text{green}$, but

$$\begin{aligned} &(\text{red} \mapsto 1, \text{yellow} \mapsto 1, \text{green} \mapsto 1) \\ &(\text{red} \mapsto 0, \text{yellow} \mapsto 0, \text{green} \mapsto 1) \end{aligned}$$

are not. After this informal explanation, let us give a formal definition. We first define the notion of model in general and come back to the notion of a model of a specification later.

⁴For example, $\neg(\varphi \vee \psi) \leftrightarrow \neg\varphi \wedge \neg\psi$ abbreviates $(\neg(\varphi \vee \psi)) \leftrightarrow ((\neg\varphi) \wedge (\neg\psi))$.

⁵Here we use the word model as a technical term in the sense of logic as made precise below. It is important to understand this technical meaning and not to confuse it with the meaning this word has in physics and engineering, or also in some areas of software engineering (model driven architectures, metamodels, ...)

Definition 2. Given the set of atomic propositions \mathbb{P} , a *model* is map, sometimes called a valuation, $v : \mathbb{P} \rightarrow \mathbb{2}$.

The reason, the notion of a model is defined as it is above, is that a model is exactly what is needed in order to assign a truth value to a formula. For example, to say that

$$p \rightarrow q$$

is true we need to know what p and q is ... actually, it is enough to know what the truth values of p and q are ... and that is exactly what a model tells us. I assume everybody knows how to evaluate propositional formulas, so let us go directly to a formal description of the algorithm for evaluating propositional formulas given truth values of the atomic propositions.

v is extended from atomic propositions to formulas using the *truth tables* corresponding to the connectives $\perp, \top, \neg, \vee, \wedge, \rightarrow, \leftrightarrow$. If we write 0, 1, $\neg, \wedge, \vee, \rightarrow, \leftrightarrow$, for the respective truth tables⁶ then

$$\begin{aligned} \llbracket p \rrbracket_v &= v(p) \\ \llbracket \perp \rrbracket_v &= 0 \\ \llbracket \top \rrbracket_v &= 1 \\ \llbracket \varphi \wedge \psi \rrbracket_v &= \llbracket \varphi \rrbracket_v \wedge \llbracket \psi \rrbracket_v \\ \llbracket \varphi \vee \psi \rrbracket_v &= \llbracket \varphi \rrbracket_v \vee \llbracket \psi \rrbracket_v \\ \llbracket \varphi \rightarrow \psi \rrbracket_v &= \llbracket \varphi \rrbracket_v \rightarrow \llbracket \psi \rrbracket_v \\ \llbracket \varphi \leftrightarrow \psi \rrbracket_v &= \llbracket \varphi \rrbracket_v \leftrightarrow \llbracket \psi \rrbracket_v \end{aligned}$$

Instead of formalising “meaning” as a function $\llbracket - \rrbracket_v$ from formulas and models to truth values, we can also formalise the relation between models and formulas saying when a formula holds (is true) in a model. This relation is usually denoted by \models or \Vdash and can be seen as a useful alternative, but equivalent, notation. The two notations are related via

$$\llbracket \varphi \rrbracket_v = 1 \quad \text{if and only if} \quad v \models \varphi.$$

A direct definition of \models :

$$\begin{aligned} v \models p & \quad \text{if } v(p) \\ v \models \top & \\ v \models \neg \varphi & \quad \text{if it is not the case that } v \models \varphi \\ v \models \varphi \wedge \psi & \quad \text{if } v \models \varphi \text{ and } v \models \psi \end{aligned}$$

(the other logical connectives can be defined similarly.)

⁶Eg the truth tables for ‘and’

p	q	$p \wedge q$
0	0	0
0	1	0
1	0	0
1	1	1

and implication

p	q	$p \rightarrow q$
0	0	1
0	1	1
1	0	0
1	1	1

. If you have doubts about the first two

lines of the truth-table for the implication consider $(x \text{ is even and } y \text{ is odd}) \rightarrow (x \text{ is even})$ and evaluate it for (x, y) taking the values $(3, 2)$ and $(2, 2)$.

Definition 3. • Each of the following are equivalent ways of saying $\llbracket \varphi \rrbracket_v = 1$

- $v \models \varphi$
- φ holds in v
- v satisfies φ
- φ is satisfied in v
- v is a model of φ
- φ is *satisfiable* iff there is $v : \mathbb{P} \rightarrow \mathbb{2}$ such that $v \models \varphi$.
- φ is *valid* iff for all $v : \mathbb{P} \rightarrow \mathbb{2}$ we have $v \models \varphi$.
- φ and ψ are equivalent, written $\varphi \equiv \psi$, iff for all $v : \mathbb{P} \rightarrow \mathbb{2}$ we have $\llbracket \varphi \rrbracket_v = \llbracket \psi \rrbracket_v$.

Example 4. • The following formulas are valid ⁷

$$\begin{aligned} p \rightarrow p \\ p \vee \neg p \\ p \rightarrow (q \rightarrow p) \end{aligned}$$

- The following formulas are not satisfiable

$$\begin{aligned} p \wedge \neg p \\ (p \rightarrow q) \wedge (p \rightarrow \neg q) \end{aligned}$$

The following formulas satisfiable

$$\begin{aligned} p \\ \neg p \end{aligned}$$

- The following are equivalences

$$\begin{aligned} (\varphi \leftrightarrow \psi) &\equiv (\varphi \rightarrow \psi) \wedge (\psi \rightarrow \varphi) \\ \neg(\varphi \vee \psi) &\equiv \neg\varphi \wedge \neg\psi \\ \neg(\varphi \wedge \psi) &\equiv \neg\varphi \vee \neg\psi \\ \varphi \rightarrow \psi &\equiv \neg\varphi \vee \psi \\ \neg(\varphi \rightarrow \psi) &\equiv \varphi \wedge \neg\psi \end{aligned}$$

This means that whatever formulas you plug in for φ and ψ , and however you evaluate the atomic propositions in these formulas, the left-hand side of “ \equiv ” will have the same truth value as the right-hand side. Therefore, whenever you encounter a left-hand side, you can replace it by the corresponding right-hand side (and vice versa). This will be important in the construction of semantic tableaux below. Also note that $\varphi \equiv \psi$ iff $\varphi \leftrightarrow \psi$ is valid.

⁷Valid formulas are also called tautologies. Formulas that are not satisfiable are also called contradictions.

Remark on Notation: One can think of formulas φ as programs and the function $\llbracket - \rrbracket$ as a compiler. Then $\llbracket \varphi \rrbracket$ is the compiled program which computes on input v the output $\llbracket \varphi \rrbracket_v$. This notation is used quite widely in computer science and you may meet it in other places as well.

Remark on Semantics: The notation $\llbracket - \rrbracket$ is often called the semantic brackets, because $\llbracket - \rrbracket_v$ takes a piece of syntax, namely a formula φ , and maps it to the meaning of the formula, which is the truth value $\llbracket \varphi \rrbracket_v$ in a given model v . To summarize, the *semantics* (= meaning) of a formula given a model v is its truth value, and it is computed by $\llbracket - \rrbracket_v$ using the truth tables. The *semantics* (= meaning) of a formula is the set of models satisfying the formula. Or, to say the same in software-engineering jargon, the semantics of a specification is the set of implementations satisfying the specification. This leads to:

Second Remark on Semantics: The situation in logic is simpler, but completely analogous, to what we have for programming languages. Indeed a program, like a formula, is a piece of syntax, built according to the rules of a context free grammar. If the program is written in an imperative programming language such as C or Java, then we can take the semantics of the program to be given, for example, by the compiler which determines how the program actually modifies the memory.

Question: What are the models for $p \rightarrow q$? How can you find the models of a formula φ by looking at the truth table of φ ?

1.2 An algorithm for satisfiability: semantic tableaux

We know how to prove the validity of a formula using truth tables. But truth tables grow exponentially with the number of atomic propositions involved (if n is the number of propositions then 2^n is the number of rows of the truth table). In this section we learn another method which is often in practice (but not in the worst case) more efficient and is the basis of many model-checking algorithms: semantic tableaux. In fact, we can think of the semantic tableau for a formula as a systematic search for all models that satisfy the formula. The relationship between validity and satisfiability is as follows.

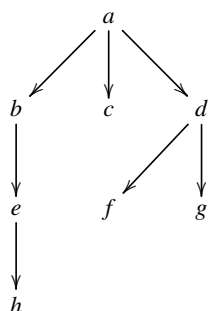
Important Fact. φ is valid iff $\neg\varphi$ is not satisfiable.

Equivalent is the following, also sometimes useful: $\neg\varphi$ is valid iff φ is not satisfiable.

A *complete semantic tableau* (for propositional logic) is a tree ⁹ where a node is a set of formulas and

⁸if and only if

⁹A typical tree looks like this



where nodes are labelled here with letters $\{a, \dots, h\}$. We say that a is the root; and b, c, d are the children or successors of a ; and h, c, f, g are the leaves; and $(a, b, e, h), (a, c), (a, d, f), (a, d, g)$ are the branches. Examples of trees abound in computer science. For example, each formula of propositional logic is a tree, in which case the labels would be logical connectives $\{\neg, \wedge, \vee, \rightarrow, \leftrightarrow\}$ for non-leaf nodes and $\mathbb{P} \cup \{\top, \perp\}$ for leaves. A semantic tableau will also be a tree, in which case each node is labelled with a set of formulas.

- Φ contains a formula $\varphi \vee \psi$ and has exactly two children given by $(\Phi \cup \{\varphi\}) \setminus \{\varphi \vee \psi\}$ and $(\Phi \cup \{\psi\}) \setminus \{\varphi \vee \psi\}$.
- Φ contains a formula $\varphi \wedge \psi$ and has exactly one child given by $(\Phi \cup \{\varphi, \psi\}) \setminus \{\varphi \wedge \psi\}$.
- Φ contains a formula $\neg\neg\varphi$ and has exactly one child given by $(\Phi \cup \{\varphi\}) \setminus \{\neg\neg\varphi\}$.

Table 1: Tableau rules

$\frac{\Gamma, \varphi \vee \psi}{\Gamma, \varphi \quad \Gamma, \psi}$	$\frac{\Gamma, \varphi \wedge \psi}{\Gamma, \varphi, \psi}$	$\frac{\Gamma, \neg\neg\varphi}{\Gamma, \varphi}$
--	---	---

Table 2: Tableau rules, short form

the leaves only contain atomic propositions or negations of atomic propositions. Moreover, after applying the equivalences of Example 4 from left to right, each non-leaf node Φ has to satisfy one of the conditions of Table 1.

Before we explain how to conclude satisfiability and validity from a complete semantic tableau, let us first define some useful notions.

Definition 5. A node in a semantic tableau is closed if it contains a contradiction, that is, if it contains a formula φ and its negation $\neg\varphi$. A leaf is called open if it is not closed. A branch in a semantic tableau is closed if it contains a closed node. A semantic tableau is closed if all branches are closed. Otherwise it is called open.

A formula φ is valid if there is some closed semantic tableau which has a root labelled by $\{\neg\varphi\}$. (*Remark:* If there is one closed semantic tableau for ψ then all complete semantic tableaux of ψ are closed. This property is important: it shows that it doesn't matter which tableau you construct.¹⁰)

A formula φ is satisfiable if there is some complete semantic tableau which has (at least) one open leaf and a root labelled by $\{\varphi\}$.

The three rules of Table 1 can be written more suggestively as in Table 2. For example, in the left-hand rule, one has a parent node consisting of a set Γ of formulas and one additional formula $\varphi \vee \psi$; this node then has two children, the left of which consists of Γ and of φ . Note that the formula $\varphi \vee \psi$ does not appear in any of the children. Also note how each of the rules in Table 2 corresponds to one of the bullet points of Table 1.

Open leaves are counter-models The method of semantic tableaux is a systematic search for *models* (that is why it is called 'semantic'). A closed leaf indicates that on this branch we cannot find a model (since a closed leaf contains a contradiction and no model can make a contradiction true). On the contrary, an open leaf represents a set of models for the formula at the root of the tableau.

¹⁰But some tableaux may be bigger than others. For example, it makes sense, in case we have a choice of several rules, to apply the non-branching rules first.

For example, if we start a tableau with $\{p \rightarrow q\}$, we obtain two leaves, namely $\{\neg p\}$ and $\{q\}$, which correspond to the three different models v for which we have $v \models p \rightarrow q$. Compare this with the truth table of Section 1.1.

Checking of the Countermodel is what is required in the typical exercises: If your tableau leads to an open leaf, then extract a model and evaluate the original formula. If the outcome is as expected, then you are done, otherwise there must be a mistake in the tableau that you then can correct.

Summary Question: Is φ valid? Start a tableau with root $\{\neg\varphi\}$. If you find an open leaf, you can stop, check your counter-example and conclude that φ is not valid. If you find no open leaf, ie all leaves are closed, then the tableau is closed and φ is valid.

2 Temporal Logic

Predicate logic is a powerful logic, but not decidable in general.¹¹ In this section, we will look at special kinds of models and logics that have a certain form of quantification but are still decidable. The general idea is, on the one hand, to look at special models (transition systems, execution sequences) and on the other hand, to only allow restricted use of quantification (eg always, sometimes).

2.1 (Linear) Temporal Logic

Linear temporal logic (LTL) is interpreted over infinite sequences, also called runs or execution sequences. ‘Linear’ emphasises that the logic speaks about sequences and not, for example, about trees. The semantics of LTL wrt sequences can be extended to a semantics wrt transition systems by considering all execution sequences generated by the transition system.

Syntax of LTL. $\text{Fma} ::= p \mid \neg\text{Fma} \mid \text{Fma} \wedge \text{Fma} \mid \bigcirc\text{Fma} \mid \Box\text{Fma} \mid \Diamond\text{Fma} \mid \text{Fma until Fma}$

Precedence rules. To save brackets, we say that the unary operators bind stronger than until which binds stronger than \wedge, \vee which bind stronger than \rightarrow which binds stronger than \leftrightarrow . For example,

$$\neg p \wedge \Diamond q \text{ until } \Box r \wedge s \rightarrow t \leftrightarrow u$$

is bracketed as

$$((\neg p \wedge ((\Diamond q) \text{ until } \Box r) \wedge s) \rightarrow t) \leftrightarrow u$$

Semantics of LTL. Let $M = (v_n)_{n \in \mathbb{N}}$ be a sequence¹² of valuations $v_n : \mathbb{P} \rightarrow \mathbb{2}$.

- $M, n \models p$ if $v_n(p) = 1$,
- $M, n \models \neg\varphi$ if not $M, n \models \varphi$,
- $M, n \models \varphi \wedge \psi$ if $M, n \models \varphi$ and $M, n \models \psi$,
- $M, n \models \bigcirc\varphi$ if $M, n + 1 \models \varphi$,

¹¹Knowledge of predicate logic is not required for these notes. But because of its importance in general and for a deeper understanding of model checking, we recall the basic definitions in an appendix.

¹² $\mathbb{N} = \{0, 1, 2 \dots\}$ denotes the set of natural numbers; $(v_n)_{n \in \mathbb{N}}$, or (v_n) , is short for $(v_0, v_1, v_2 \dots)$.

- $M, n \models \Box\varphi$ if $M, m \models \varphi$ for all $m \geq n$,
- $M, n \models \Diamond\varphi$ if $M, m \models \varphi$ for some $m \geq n$,
- $M, n \models \varphi \text{ until } \psi$ if $M, m \models \psi$ for some $m > n$ and $M, k \models \varphi$ for all k with $n < k < m$.

$M \models \varphi$ if $M, n \models \varphi$ for all $n \in \mathbb{N}$.

$M \models \Phi$ and $\Phi \models \varphi$ and $\models \varphi$ (' φ is valid') as before.

Terminology. Read

- $\bigcirc\varphi$ next φ ,
- $\Box\varphi$ always φ ,
- $\Diamond\varphi$ sometimes φ , or eventually φ ,
- $\varphi \text{ until } \psi$ φ until ψ .

Remark. [Only for readers of Appendix B] A model for LTL is a model for ML where we take \mathbb{N} as the carrier set and two relations. The first relation is $\{(n, n+1) \mid n \in \mathbb{N}\}$ and interprets the \bigcirc .¹³ The second relation is \leq and interprets \Box and \Diamond .

Some important formulae.

$\Box\Diamond p$	infinitely often p
$\Diamond\Box p$	eventually p will always be true
$\neg(\neg p \text{ until } q)$	p before q (see below)
$\Box(p \rightarrow \Diamond q)$	progress: each 'request' p gets 'acknowledgement' q
$\neg\Diamond\Box p$	weak fairness: not forever 'blocked at p '
$\Box\Diamond\neg p$	the same
$\Diamond\Box p \rightarrow \Box\Diamond q$	weak fairness: 'eventually always p (eg enabled) only if infinitely often q (eg executed)'
$\Box\Diamond p \rightarrow \Box\Diamond q$	strong fairness: 'infinitely often p (eg enabled) only if infinitely often q (eg executed)'

Show that $\neg(\neg p \text{ until } q)$ means: whenever in the future q holds, then p must have happened before q .

Fairness: Note that $\neg\Diamond\Box p$ is equivalent to $\Diamond\Box p \rightarrow \perp$, so it is a special case of the more elaborate $\Diamond\Box p \rightarrow \Box\Diamond q$. A typical example of the latter could be: Always, if a process keeps waiting to enter and the door remains open, then the process will (be scheduled to) enter eventually.¹⁴ ¹⁵ This is weaker than $\Box\Diamond p \rightarrow \Box\Diamond q$, an example of which could be: If a process is waiting to enter and the door will become open infinitely many times, then the process will (be scheduled to) enter eventually.¹⁶ ¹⁷

¹³The box and the diamond for this relation are the same as each state n has a unique successor; we therefore only need one operator and write it as \bigcirc .

¹⁴Think eg of an elevator which opens its door and doesn't go away. If the scheduler is weakly fair, the process will eventually enter.

¹⁵You might want to show that $\Box(\Box p \rightarrow \Diamond q) \leftrightarrow \Diamond\Box p \rightarrow \Box\Diamond q$.

¹⁶Think eg of an elevator which opens its door, then may go away again, but keeps coming back. If the scheduler is strongly fair, the process will eventually enter.

¹⁷You might want to show that $(\Box\Diamond p \rightarrow \Box\Diamond q) \leftrightarrow \Box(\Box\Diamond p \rightarrow \Diamond q)$ or also $(\neg\Diamond\Box p \rightarrow \Box\Diamond q) \leftrightarrow \Box(\neg\Diamond\Box p \rightarrow \Diamond q)$.

2.2 Transition Systems

A *transition system* $(X, (R_i)_{i \in I}, \nu)$, or *Kripke model*, consists of

- a set X of 'states', also called the carrier of the model,
- a collection of relations $(R_i)_{i \in I} \subseteq X \times X$,
- a valuation $\nu : X \times \mathbb{P} \rightarrow \mathbb{2}$ of atomic propositions \mathbb{P} .

Read $xR_i y$ as y is an i -successor of x .

Why do we have many relations R_i ? Think of the elements $i \in I$ as actions, or as the letters of the input alphabet of an automaton: then in a given state x , for each action $i \in I$, we can have different successors y , ie y such that $xR_i y$.

To give semantics to distributed processes, we will only need one relation and then we write simply (X, R, ν) .

2.3 Transition System Semantics of LTL, Some Remarks on Spin

Let $M = (X, R, \nu)$ be a transition system and $x \in X$. We call x the *initial state* of M . For LTL-formulae φ , we define

$$M, x \models \varphi \text{ if } \sigma, 0 \models \varphi \text{ for all sequences } \sigma \text{ through } M \text{ starting at } x.$$

Remark. This is the semantics used by the SPIN model checker: A Promela program defines a transition system M together with an initial state x ; checking a formula φ against a Promela model is the same as asking the question whether

$$M, x \models \varphi.$$

In case that $M, x \not\models \varphi$, the model checker will produce a counter example to φ , that is, a particular sequence σ such that

$$\sigma, 0 \not\models \varphi,$$

or, which is the same,

$$\sigma, 0 \models \neg \varphi,$$

Note that, to check φ using Spin, one has to use the negation $!(\varphi)$ as in `spin -f '!(\varphi)'`. Spin tries to find a counterexample to φ by finding an example of a sequence satisfying $\neg \varphi$.

Remark on Interleaving Semantics. The way SPIN assigns a transition system to a Promela program is an example of interleaving semantics: No two independent processes are allowed to move at the same time. This assumption sounds unrealistic, but it doesn't matter.

Semantics of U (Spin's until). Spin knows the temporal operators from Section 2.1, but uses a variation of until written as U . Its semantics is given by

- $\sigma, n \models pUq$ if $\begin{array}{l} \sigma, m \models q \text{ for some } m \geq n \text{ and} \\ \sigma, k \models p \text{ for all } k \text{ with } n \leq k < m. \end{array}$

$$\begin{aligned}
(\varphi \leftrightarrow \psi) &\leftrightarrow (\varphi \rightarrow \psi) \wedge (\psi \rightarrow \varphi) \\
\neg(\varphi \vee \psi) &\leftrightarrow \neg\varphi \wedge \neg\psi \\
\neg(\varphi \wedge \psi) &\leftrightarrow \neg\varphi \vee \neg\psi \\
\varphi \rightarrow \psi &\leftrightarrow \neg\varphi \vee \psi \\
\neg(\varphi \rightarrow \psi) &\leftrightarrow \varphi \wedge \neg\psi
\end{aligned}$$

Table 3: Propositional laws

U can be expressed using until: $pUq \leftrightarrow q \vee (p \wedge (p \text{ until } q))$ but not vice versa: formulae built from U are stutter invariant (see below). Accordingly, $\perp Uq$ does not express $\bigcirc q$ but we have the laws $\perp Uq \leftrightarrow q$ and $q \rightarrow (pUq)$.

until can be expressed as $(p \text{ until } q) \leftrightarrow \bigcirc(pUq)$.¹⁸

Stutter invariance An LTL-formula φ is said to be stutter-invariant if $(v_n)_{n \in \mathbb{N}} \models \varphi$ implies that $(v'_n)_{n \in \mathbb{N}} \models \varphi$ for all stuttervariants $(v'_n)_{n \in \mathbb{N}}$ of $(v_n)_{n \in \mathbb{N}}$, where $(v'_n)_{n \in \mathbb{N}}$ is called a *stuttervariant* of $(v_n)_{n \in \mathbb{N}}$ if there is a surjective function $f : \mathbb{N} \rightarrow \mathbb{N}$ such that $f(0) = 0$ and either $f(n+1) = f(n)$ or $f(n+1) = f(n) + 1$ and $v'_n = v_{f(n)}$.

2.4 Semantic Tableaux for LTL

Motivation and Explanation

LTL-tableaux are an extension of propositional tableau. Although technically more complicated, the basic ideas are not difficult. First, we need some preliminaries.

Recall that, in order to keep the number of rules small, we used certain laws to transform formulae to a form making the rules applicable. These laws are summarised in Table 3. Remember that we apply them from left to right in a tableau.

To deal with the new temporal operators, we add the laws of Table 4. (Exercise: show that these laws are valid.) They are again to be applied from left to right. Their effect is that in the leaves of a propositional tableau, there are only the following kind of formulae: atomic propositions, negations of atomic propositions, formulae of the kind $\bigcirc\varphi$.¹⁹

How do we build an LTL-tableau? Start building a propositional tableau applying the laws from Tables 3 and 4. Now, all the leaves of this tableau contain only atomic propositions or negations of atomic propositions or formulae whose outermost connective is \bigcirc .

¹⁸In Spin-LTL one writes X for \bigcirc .

¹⁹We may want to add, for the Spin-version of until,

$$\begin{aligned}
\varphi U \psi &\leftrightarrow \psi \vee (\varphi \wedge \bigcirc(\varphi U \psi)) \\
\neg(\varphi U \psi) &\leftrightarrow (\neg\psi \wedge (\neg\varphi \vee \bigcirc\neg(\varphi U \psi)))
\end{aligned}$$

$$\begin{aligned}
\neg \bigcirc \varphi &\leftrightarrow \bigcirc \neg \varphi \\
\neg \Diamond \varphi &\leftrightarrow \Box \neg \varphi \\
\neg \Box \varphi &\leftrightarrow \Diamond \neg \varphi \\
\Diamond \varphi &\leftrightarrow \varphi \vee \bigcirc \Diamond \varphi \\
\Box \varphi &\leftrightarrow \varphi \wedge \bigcirc \Box \varphi \\
\varphi \text{ until } \psi &\leftrightarrow \bigcirc(\psi \vee (\varphi \wedge (\varphi \text{ until } \psi))) \\
\neg(\varphi \text{ until } \psi) &\leftrightarrow \bigcirc(\neg\psi \wedge (\neg\varphi \vee \neg(\varphi \text{ until } \psi)))
\end{aligned}$$

Table 4: Temporal laws

These nodes are leaves in a propositional tableaux. No further propositional reasoning is possible. The leaves correspond to the states of LTL-models.

How to make a transition from one state to the next will be described now. We first need two definitions.

Definition. Let us agree to call a node labelled with a set of formulas Φ a **state**,²⁰ if Φ contains only atomic propositions or negations of atomic propositions or formulae whose outermost connective is \bigcirc ; moreover, we assume that there are no contradictions among the (negated) atomic propositions.

Definition. For a state labelled Φ define $\Phi' = \{\varphi \mid \bigcirc\varphi \in \Phi\}$.

In other words, Φ' erases the outermost \bigcirc operator from formulas in Φ .

We now come to the rule for \bigcirc . The notation Φ' has been devised in a way such that, if *now* Φ holds then at the *next* time-point Φ' must hold ... if that is not clear, pause to think about it. We therefore would like to say that any state Φ has precisely one child, namely Φ' . Unfortunately, this does not work, since the tableau for, eg, $\Box p$ would be infinite (easy exercise!). We solve this problem as follows. Given a state labelled Φ , if there is already a node labelled Φ' , then we do not create a child of Φ and instead “loop back” to the already existing node labelled by Φ' . Otherwise create a child labelled Φ' and continue as above.

Definition. A tableau is complete if there is no further rule to apply. A tableau is closed if all branches contain a contradiction, otherwise it is open.

Definition. Given a complete, open tableau, we extract a graph (V, E) from where the vertices V are the states of the tableau and there is an edge $(s, t) \in E$ whenever there is path (s, s_1, \dots, s_n, t) in the tableau such that the s_i are non-state nodes. We also define a mapping F from vertices to formulas. If $(\Phi, \Psi_1, \dots, \Psi_n, \Phi_2)$ are the set of formulas labelling (s, s_1, \dots, s_n, t) in the tableau, then $F(t)$ is defined as $\bigcup\{\Psi_i \mid 1 \leq i \leq n\} \cup \Phi_2$.

Extracting a model from a tableau. If the graph of a complete open tableau has a vertex that has no successor, then the path leading to that vertex is a model of the formula at the root of the tableau and the formula is therefore satisfiable. If the graph has a cycle, this cycle is a model if it is selffulfilling, defined as follows.

Definition. A cycle is **self-fulfilling** if

1. for all vertices s in the cycle and all $\Diamond\varphi \in F(s)$ there is some vertex t in the cycle such that $\varphi \in F(t)$

²⁰X-node in [1]

1. Build a complete propositional tableau for Ψ using the laws of Tables 3 and 4.
2. We continue as follows (with Φ ranging over the leaves of the propositional tableau just built):
 - If Φ contains a contradiction, then Φ has no child and the branch is closed.
 - If Φ does not contain a contradiction:
 - If Φ' appears on the path from the root to Φ , then no new child of Φ is created but one loops back to Φ' .
 - If Φ' is empty, then Φ has no child.
 - Otherwise: Φ has a child Φ' . (The notation Φ' was defined above.) Go back to 1., continuing to build a propositional tableau for $\Psi = \Phi'$.

Table 5: How to build an LTL-tableau with root Ψ

2. for all vertices s in the cycle and all $(\varphi \text{ until } \psi) \in F(s)$ there is some node t in the cycle such that $\psi \in F(t)$

Intuitively, we think of a formula $\Diamond\varphi$ as a *promise* to make φ true. Now, recall that building a tableau we are trying to construct a sequence satisfying the formulas in the tableau. In order to satisfy the promise $\Diamond\varphi$, we need that φ is true in some node of the sequence. If this happens, we call the sequence self-fulfilling.

A summary of the construction of an LTL-tableau is given in Table 5. If the tableau is not closed, we extract a graph from the tableau. Then the formula at the root of the tableau is satisfiable if the graph contains a self-fulfilling cycle.

3 LTL Model Checking with Semantic Tableaux

To check whether a property φ holds in a Kripke model M , we ‘synchronise’ the semantic tableaux method for the validity of φ with the transition system M .

Let $M = (X, R, v)$ where $v : X \times \mathbb{P} \rightarrow \mathbb{2}$ for a fixed set \mathbb{P} of atomic propositions. Also fix an initial state $x_0 \in X$. We want to build a tableau for φ and M . The nodes (x, Φ) of the tableau consist of states $x \in X$ and sets of LTL-formulae Φ .

The root is $(x_0, \{\neg\varphi\} \cup \{p \mid v(x_0)(p) = 1\} \cup \{\neg p \mid v(x_0)(p) = 0\})$.

For a non-state node Φ , the children of (x, Φ) are (x, Ψ) where Ψ is a child of Φ according to the three rules for propositional nodes from above.

For state nodes Φ , (x, Φ) is a leaf (ie, it has no child) if Φ contains a contradiction or if Φ' is empty or if the successors (x', Φ') of (x, Φ) appear on a path from the root to (x, Φ) . Otherwise the children of (x, Φ) are

$$(y, \Phi' \cup \{p \in \mathbb{P} \mid v(y, p) = 1\} \cup \{\neg p \in \mathbb{P} \mid v(y, p) = 0\})$$

for all successors y of x , that is, for all y such that xRy .

References

- [1] M. Ben-Ari. *Mathematical logic for computer science (2. ed.)*. Springer, 2001.

A Predicate Logic

This appendix contains additional material. It would fit in after propositional logic and before temporal logic, for reasons explained below.

A.1 Introduction

Propositional logic is appropriate to specify situations dealing with finite data, see eg the traffic light example from the lectures. If one wants to add dynamic features, a good choice is often to add to the propositional operators (as eg \neg, \wedge, \dots) temporal operators, which allow to specify temporal behaviour (as eg always, sometimes, until).

In this section, we briefly look at an even more powerful extension, namely extending propositional logic by quantifiers (“for all”, “there exists”). This allows us to talk about infinite structures, including data structures such as integers, lists, etc but also structures like time and hence about dynamic behaviour.

Predicate logic is powerful enough to encode (more or less) everything that one ever might want to, but this expressiveness comes at a price: Contrary to propositional logic, predicate logic is not decidable (ie, there is no algorithm that takes as input an arbitrary formula φ and decides whether $\models \varphi$ or $\not\models \varphi$; any attempt at writing such an algorithm would run into the same difficulties as we encountered with Turing’s halting problem, that is, such an algorithm would not be able to halt on all inputs). Nevertheless, predicate logic is at the basis of many important formalisms used for the specification and verification of programs and protocols and so is well worth knowing. Moreover, temporal logics such as LTL can be understood as a sophisticated way of limiting the expressive power of predicate logic just enough in order to make it decidable.

A.2 Definitions

A *first-order language* \mathcal{L} is specified by

1. a set \mathcal{F} of *function symbols* and a natural number (called *the arity of f*) for each $f \in \mathcal{F}$ (function symbols of arity 0 are called *constants*),
2. a set \mathcal{P} of *predicate symbols* and a natural number (called *the arity of p*) for each $p \in \mathcal{P}$,
3. a set Var of variables.

A *term* is either a variable or of the form $f(t_1, \dots, t_n)$ where f is a function symbol of arity n and the t_i are terms. An *atom*, or *atomic formula*, is of the form $p(t_1, \dots, t_n)$ where p is a predicate symbol of arity n and the t_i are terms or it is of the form $t_1 = t_2$. *Formulae* are now described by

$$\varphi ::= p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \forall x. \varphi \mid \exists x. \varphi$$

where p ranges over atoms and x over variables.

A *model* M (for first-order predicate logic (FOL)) consists of a non-empty set A and

1. a function $f^M : A^n \rightarrow A$ for each n -ary function symbol²¹ f ,

²¹An ‘ n -ary function symbol’ is a ‘function symbol of arity n ’.

2. a predicate (or relation) $p^M : A^n \rightarrow \mathbb{2}$ for each n -ary predicate symbol p .²²

An *environment* (or memory or look-up table or valuation or interpretation) is

3. a function $l : \text{Var} \rightarrow A$ from variables to A .

The environment l where x has been updated to a is denoted by $l[x \mapsto a]$.

The semantics $\llbracket \varphi \rrbracket_{M,l}$ of φ in a model M under environment l is now defined as follows ($\llbracket \varphi \rrbracket$ plays now the role of $v(\varphi)$ in PL).

- $\llbracket x \rrbracket_{M,l} = l(x)$,
- $\llbracket f(t_1, \dots, t_n) \rrbracket_{M,l} = f^M(\llbracket t_1 \rrbracket_{M,l}, \dots, \llbracket t_n \rrbracket_{M,l})$,
- $\llbracket p(t_1, \dots, t_n) \rrbracket_{M,l} = p^M(\llbracket t_1 \rrbracket_{M,l}, \dots, \llbracket t_n \rrbracket_{M,l})$,
- $\llbracket \neg \varphi \rrbracket_{M,l} = \neg \llbracket \varphi \rrbracket_{M,l}$,
- $\llbracket \varphi \wedge \psi \rrbracket_{M,l} = \llbracket \varphi \rrbracket_{M,l} \wedge \llbracket \psi \rrbracket_{M,l}$,
- $\llbracket \forall x. \varphi \rrbracket_{M,l} = 1$ if $\llbracket \varphi \rrbracket_{M,l[x \mapsto a]} = 1$ for all $a \in A$,
- $\llbracket \exists x. \varphi \rrbracket_{M,l} = 1$ if $\llbracket \varphi \rrbracket_{M,l[x \mapsto a]} = 1$ for some $a \in A$.

$\llbracket \varphi \rrbracket_M = 1$ if $\llbracket \varphi \rrbracket_{M,l} = 1$ for all l . $\llbracket \varphi \rrbracket = 1$ if for $\llbracket \varphi \rrbracket_M = 1$ for all M .

We also write

- $M, l \models \varphi$ for $\llbracket \varphi \rrbracket_{M,l} = 1$,
- $M \models \varphi$ for $\llbracket \varphi \rrbracket_M = 1$,
- $M \models \Phi$ if $M \models \varphi$ for all $\varphi \in \Phi$,

Validity and consequence are defined as follows.

- $\models \varphi$ (' φ is valid') if $M \models \varphi$ for all M ,
- $\Phi \models \varphi$ (' φ is a consequence of the set of formulae Φ ') if for all M it holds that $M \models \Phi$ implies $M \models \varphi$.

B Logic for Transition Systems (Modal Logic)

This section contains additional material. It would fit after the definition of a transition system. This section shows that temporal logic is part of a wider field of logic called modal logic. Modal logics come in different shapes and sizes and are used all over computer science to model very different phenomena such as knowledge, belief, obligations, etc.

Syntax of ML. $\varphi ::= p \mid \neg \varphi \mid \varphi \wedge \varphi \mid \Box_i \varphi \mid \Diamond_i \varphi$

Semantics of ML. Let $M = (X, (R_i)_{i \in I}, \nu)$ be a Kripke model and $x \in X$.

²²Note that functions $A^n \rightarrow \mathbb{2}$ are in one-to-one correspondence to subsets of A^n .

- $M, x \models p$ if $v(x, p) = 1$,
- $M, x \models \neg\varphi$ if not $M, x \models \varphi$ (notation: $M, x \not\models \varphi$),
- $M, x \models \varphi \wedge \psi$ if $M, x \models \varphi$ and $M, x \models \psi$,
- $M, x \models \Box_i \varphi$ if $M, y \models \varphi$ for all y such that $xR_i y$,²³
- $M, x \models \Diamond_i \varphi$ if $M, y \models \varphi$ for some y such that $xR_i y$.

$M \models \varphi$ if $M, x \models \varphi$ for all x .

$M \models \Phi$ if $M \models \varphi$ for all $\varphi \in \Phi$.

$\Phi \models \varphi$ if $M \models \Phi$ implies $M \models \varphi$ for all M .²⁴

$(X, (R_i)_{i \in I}) \models \varphi$ if $(X, (R_i)_{i \in I}, v) \models \varphi$ for all v .²⁵

Notation. One often writes $[i]$ for \Box_i and $\langle i \rangle$ for \Diamond_i . In many cases, there will be just one transition relation R . We then write (X, R, v) and the operators as \Box, \Diamond .

Example 1. Consider a model (\mathbb{N}, \leq, v) , which has as a carrier the natural numbers (= non-negative integers). Thinking of $x \leq y$ as y is in the future of x , we have that \Box means ‘always’ and \Diamond means ‘sometimes’.

Example 2. Consider a model (\mathbb{N}, S, v) , where \mathbb{N} is as before, but S is the relation $\{(x, x + 1) \mid x \in \mathbb{N}\}$. Thinking of \mathbb{N} again as the flow of time, we have that \Box means ‘at the next point in time’. Note that $\Box\varphi \leftrightarrow \Diamond\varphi$.

Example 3. Consider an arbitrary model (X, R, v) , where we think of X as a set of worlds (or possible scenarios) and of xRy as y being an alternative to x . Then we can read \Box as ‘necessarily’ and \Diamond as ‘possibly’. [This example is the one with which the area of modal logic originated ca hundred years ago.]

Example 4. Consider a model (X, R_i, v) , where we think of X as a set of worlds and of $xR_i y$ as: The world y looks the same as x according to the facts known to agent i . Then we can read \Box_i as ‘agent i knows’. [This example is important in the specification of multi-agent systems.]

Remark. A Kripke model can be seen as a model for FOL where the language consists of one binary predicate symbol for each $i \in I$ and one unary predicate symbol for each $p \in \mathbb{P}$. We then see that the modal language is a restriction of FOL. For example, formulae have at most one ‘free’ variable (the x in the semantics definition). Moreover, \Box and \Diamond are restricted quantifiers in that they do not quantify over all elements of the model but only over the successors of a particular element. One of the benefits from this restriction is that the logic becomes decidable (Proof: Adapt the semantic tableau method from propositional logic (we won’t do this in the course, but it is not very difficult)). There is even a stronger property: Each satisfiable formula is satisfiable in a finite model (and we have seen in the lectures that this is not true for FOL).

²³Other notations for $xR_i y$ are: $(x, y) \in R_i$, $R_i(x, y)$, $R_i(x, y) = 1$, $x \xrightarrow{i} y$, $x \rightarrow_i y$.

²⁴Notation: $\psi \models \varphi$ for $\{\psi\} \models \varphi$; $\models \varphi$ for $\emptyset \models \varphi$.

²⁵Terminology: $(X, (R_i)_{i \in I})$ is called a *Kripke frame* in this context.

C Questions and Exercises

Questions marked with * relate to material relevant for the exam. Questions of particular importance for the exam are marked with **.

Question 1. * A traffic light can be modelled in propositional logic by three atomic propositions red, yellow, green. Without further specification, this means that a traffic light can be in $2^3 = 8$ different states. Now suppose you want to model a crossing with 4 traffic lights (for cars) and 4 traffic lights for cyclists (the latter have only red and green). How many possible states are there for the crossing now? What if you add 8 more traffic lights (two colours each) for pedestrians? Explain the notion of ‘state explosion’.

Question 2. * For each of the following formulae determine whether they are valid and whether they are satisfiable.

$$p \rightarrow p$$

$$p \wedge \neg p$$

$$p \wedge \neg q$$

$$p \vee \neg p$$

$$p$$

$$\neg p$$

Question 3. * List all models of the following traffic light specification (r, y, g for red, yellow, green).

$$(r \vee y) \rightarrow \neg g$$

Do this using a truth table and do this again using a semantic tableau. Compare the two procedures. Add a formula that makes sure that not all colours can be dark simultaneously.

Question 4. 1. Explain in what sense propositional logic can also be defined by the smaller grammar

$$\varphi ::= p \mid \neg\varphi \mid \varphi \wedge \varphi$$

[Hint: Use the equivalences of Example 4.]

2. Similarly, show that from each of the following sets of connectives $\{\perp, \rightarrow\}$, $\{\neg, \vee\}$ all other connectives can be defined.

Question 5. ** Use semantic tableaux to check whether the following formulae are valid and, in case they are not, give a counterexample.

$$((p \wedge q) \rightarrow r) \rightarrow (p \rightarrow (q \rightarrow r))$$

$$(p \rightarrow (q \rightarrow r)) \rightarrow ((p \wedge q) \rightarrow r)$$

$$((p \wedge q) \rightarrow r) \rightarrow ((p \rightarrow q) \rightarrow r)$$

$$((p \rightarrow q) \rightarrow r) \rightarrow ((p \wedge q) \rightarrow r)$$

$$(p \wedge q \wedge (q \rightarrow (r \rightarrow p))) \rightarrow r$$

$$(r \wedge q) \rightarrow (((r \rightarrow s) \wedge q) \vee \neg(q \rightarrow s))$$

$$(p \rightarrow (q \rightarrow r)) \rightarrow ((p \rightarrow q) \rightarrow (p \rightarrow r))$$

Question 6.

1. Show that the following FOL-formulae are valid.

$$\begin{aligned}\forall x. \varphi &\rightarrow \exists x. \varphi \\ \exists y. \forall x. \varphi &\rightarrow \forall x. \exists y. \varphi \\ \forall x. (\varphi \rightarrow \psi) &\rightarrow (\forall x. \varphi \rightarrow \forall x. \psi)\end{aligned}$$

2. Give counter-examples for

$$\begin{aligned}\forall x. \exists y. \varphi &\rightarrow \exists y. \forall x. \varphi \\ (\forall x. \varphi \rightarrow \forall x. \psi) &\rightarrow \forall x. (\varphi \rightarrow \psi)\end{aligned}$$

Question 7. Why is PL decidable. Why does this argument not apply to FOL (First-order predicate logic)? Is FOL decidable?

Question 8. * Define

- $M, n \models \varphi \text{ atnext } \psi$ if $M, m \not\models \psi$ for all $m > n$ or $M, k \models \varphi$ for the smallest $k > n$ with $M, k \models \psi$.
- $M, n \models \varphi \text{ while } \psi$ if $M, m \not\models \psi$ for some $m > n$ and $M, k \models \varphi \wedge \psi$ for all k with $n < k < m$.
- $M, n \models \varphi \text{ before } \psi$ if for all $m > n$ with $M, m \models \psi$ there is k with $n < k < m$ such that $M, k \models \varphi$.

Express these operators using LTL.

Question 9. ** For each of the following LTL-formulae, either show that the formula is valid or give a counterexample.

1. $\Box \Box p \rightarrow \Box \Box p$,
2. $\Box(p \vee q) \rightarrow \Box p \vee \Box q$,
3. $\Box p \vee \Box q \rightarrow \Box(p \vee q)$,
4. $\Box \Diamond \varphi \rightarrow \Diamond \Box \varphi$,
5. $\Box(\varphi \rightarrow \Box \Diamond \psi) \wedge \Diamond \varphi \rightarrow \Diamond \psi$.
6. $\Box q \wedge (p \text{ until } q) \rightarrow \Box p$,
7. $\Box \varphi \leftrightarrow \perp \text{ until } \varphi$.
- 8.
9. $\Box(p \rightarrow \Diamond \neg p) \leftrightarrow \Box \Diamond \neg p$.

Question 10. * Further temporal laws: Show that the following are valid.

1. $\Box(p \rightarrow \bigcirc q) \wedge p \rightarrow \Box q$. [This law is a powerful induction principle if you want to prove that a formula of the kind $\Box\varphi$ is true.]
2. $(\Box\Diamond p \rightarrow \Box\Diamond q) \rightarrow \Box(\Box\Diamond p \rightarrow \Box\Diamond q)$. [This is related to fairness.]

Question 11. *

1. Give a counterexample to p until $q \rightarrow p$ before q .
2. Change the definition of the semantics of **until** and **before** in such a way that the formula above becomes valid.