

Chapter 2

Talking About Graphs

In the first chapter we have seen that graphs are a tool that allows us to model some important concepts. Up to now we have talked about such graphs in natural language. In this chapter we are going to introduce a particular *formal language* in order to do that: the language of modal logics.

For many reasons it is highly desirable to use a formal language instead of plain English: on the one hand, formal languages are more constrained than natural languages, thus more able to avoid ambiguities; on the other hand, a formal language allows one to rigorously verify each property that can be expressed in it. In particular, we may use a computer in order to verify these properties in a automatic or semi-automatic manner. To do so we have to communicate with the computer in a well-defined language.

All the examples of the preceding chapter could be presented in the language of *first-order logic*: all we need for that are unary predicates for the node labels and binary predicates for the edge labels. The former have a single node as an argument, and the latter have two nodes as arguments: a source and a target node. For example, the graph of Fig. 1.3 (Sect. 1.1, page 3) can be described in first-order logic by the formula

$$\text{LightOff}(w) \wedge \text{LightOn}(u) \wedge \text{Toggle}(w, u) \wedge \text{Toggle}(u, u)$$

where w and u are names for the left and the right node of the graph. Moreover, the sentence “When the light is off then toggling the switch turns the light on” describing that graph can be written in first-order logic as

$$\forall w (\text{LightOff}(w) \rightarrow \forall u (\text{Toggle}(w, u) \rightarrow \text{LightOn}(u)))$$

where $\forall$ is the universal quantifier “for all” and $\rightarrow$ is the material implication “implies” or “if... then...” This formula may be read: “for every node w , if w is labelled *LightOff* then for every node u , there is a link from w to u labelled *Toggle* implies that u is labelled *LightOn*.”

Similarly, part of the information in Fig. 1.8 of Sect. 1.3 is formally expressed by

$$\begin{aligned} &\forall w(\text{Black}(w) \rightarrow (\forall u(R_{\text{Bob}}(w, u) \rightarrow \text{Black}(u))) \\ &\quad \wedge (\forall u(R_{\text{Ann}}(w, u) \rightarrow (\text{Black}(u) \vee \text{Red}(u)))))) \end{aligned}$$

However, in this book we shall not use the language of first-order logic. There are several reasons for that. First, if we express in the language of first-order logic statements about e.g. the agents' knowledge in the card game of Sect. 1.3 then we have to mention explicitly the nodes. This makes us write down formulas such as

$$\exists w'(R_{\text{Ann}}(w, w') \wedge \text{Red}(w')) \wedge \exists w'(R_{\text{Ann}}(w, w') \wedge \neg \text{Red}(w'))$$

in order to express that Ann does not know whether the card is red or not, and

$$\begin{aligned} &\forall w'(\text{Bob}(w, w') \rightarrow (\exists w''(\text{Ann}(w', w'') \wedge \text{Red}(w'')) \\ &\quad \wedge \exists w''(\text{Ann}(w', w'') \wedge \neg \text{Red}(w'')))) \end{aligned}$$

in order to express that Bob knows that. Such formulas are somewhat cumbersome. More importantly, the second reason for not choosing first-order logic is that automated reasoning in that logic is notoriously difficult: it is undecidable.¹

We are going to describe graphs in another, simpler language. That language has neither variables nor quantifiers: instead, there are *modal connectives*. These connectives are also called *modal operators* or *modalities*. The language is therefore called a modal language. For example, we describe the two above statements about Ann's ignorance by means of the following expressions:

$$\begin{aligned} &\neg K_{\text{Ann}} \text{Red} \wedge \neg K_{\text{Ann}} \neg \text{Red} \\ &K_{\text{Bob}}(\neg K_{\text{Ann}} \text{Red} \wedge \neg K_{\text{Ann}} \neg \text{Red}) \end{aligned}$$

This is clearly more natural, more intuitive and more concise than the above first-order logic formulas. The modal operators of these formulas are K_{Bob} and K_{Ann} . They are read “Bob knows” and “Ann knows.” More abstractly, we write expressions such as $K_I A$, read “ I knows that A ,” and $\neg K_I A$, read “ I does not know that A ,” where I is an *agent* and A is a formula. The expressions $K_I A$ and $\neg K_I A$ are themselves also formulas.

The sentence “ I does not know that A ” means that $\neg A$, i.e., the falsehood of A , is actually possible for I . In other words, it means that I can imagine a state in which $\neg A$ holds. On the contrary, “ I knows that A ” means that A holds in all states that I can imagine. The states that I can imagine are all those states that I cannot distinguish from the real state by the information that I has. In a Kripke model, the accessibility relation $R(I)$ for the modality K_I —made up by all those edges that are

¹More precisely, we here refer to the satisfiability problem of first-order logic. This is the famous Church-Turing Theorem [Kle67]. We will explain in the end of the chapter what satisfiability and decidability mean.

labelled I —models what I cannot distinguish: we have $\langle w, u \rangle \in R(I)$ if and only if I is unable to distinguish w from u .

The identification of “ I knows A ” with “ A holds in all states that I can imagine” makes explicit that the modal language involves quantification over nodes, just as the language of first-order logic. The main difference is that graph nodes do not occur explicitly in the modal language. Moreover, quantification in modal logic is always *relativised*: it is not over all nodes, but only over those that are accessible from the current node via some edge. From the point of view of first-order logic, the modal language can be identified with a *fragment* of the language of first-order logic.² In that sense, the modal language is weaker than that of first-order logic.

The modal language can actually be located halfway between the language of propositional logic and the language of first-order logic: contrarily to the language of propositional logic and just as that of first-order logic it allows for *quantification*; and contrarily to the language of first-order logic it only allows for a *restricted* form of quantification in that modal operators are relativised quantifiers.

In many cases the expressivity provided by modalities is sufficient. As far as our examples are concerned, they allow us to express things such as “The light is on and after every possible performance of the toggling action the light is off,” “There is a state that is possible for Ann where the card is red,” “All next states after red are green,” “After having opened the door, the door is opened and I know that the sun is shining,” “If I draw a card then it is possibly red,” “If I toss a coin then the result is necessarily heads or tails.” We shall see more examples of modalities in the sequel.

We start this chapter by showing how to build complex formulas from boolean and modal connectives, leading to the definition of a general language for modalities (Sect. 2.1). Then we show how we can define modal languages in LOTREC (Sect. 2.2). After that, we formally state what it means that a formula is true at a node of a graph (Sect. 2.3) and show how we can check this in LOTREC (Sect. 2.4). Finally, we formally define several reasoning problems consisting in checking the truth of formulas in graphs and classes of graphs (Sect. 2.5) and briefly mention axiomatisations (Sect. 2.6) and the complexity of satisfiability (Sect. 2.7).

2.1 The Formal Language

We are now going to define a general formal language in which we can talk about concepts such as time, knowledge, belief, and action by means of modal operators. The elements of our language are called *formulas*. These formulas talk about what is true at a given node of a graph: they are going to be either true or false there.

²That fragment is part of a larger fragment of the language of first-order logic that is called the guarded fragment [AvBN98]. Remarkably and contrarily to the satisfiability problem for formulas of the entire language of first-order logic, the satisfiability problem for formulas of that fragment is decidable.

2.1.1 Atomic Formulas

First of all, we want to be able to describe basic facts. For instance, simple sentences such as “The light is on,” “The card is red,” and “The card is black” are directly represented by expressions such as `LightOn`, `Red` and `Black`. We consider each such expression as a single symbol and suppose that it cannot be divided into smaller parts: we say that it is an *atomic formula* (also called a propositional variable). You may think of atomic formulas as the smallest sentences you can write in our formal language.

We group atomic formulas in the *set of atomic formulas* $\mathcal{P}$. The elements of $\mathcal{P}$ are the basic node labels in our graphs.

2.1.2 Boolean Connectives

Atomic formulas can be combined by *boolean connectives* in several ways, resulting in complex formulas. The main boolean connectives are: falsum $\perp$, read “false;” negation $\neg$, read “not;” conjunction $\wedge$, read “and;” and disjunction $\vee$, read “or.” These connectives allow us to write for example:

- $\neg\text{LightOn}$ for “The light is not on” or “The light is off;”
- $\text{S1Up} \wedge \text{LightOn}$ for “Switch 1 is up and the light is on;”
- $\text{S1Up} \vee \text{S2Up}$ for “Switch 1 is up or switch 2 is up.”

The formula $\neg\text{LightOn}$ is a *literal*: a literal is either an atomic formula P from $\mathcal{P}$ or the negation $\neg P$ of some P from $\mathcal{P}$.

In the above examples, boolean connectives combine atomic formulas. But we can also put together non-atomic formulas:

- $(\text{S1Up} \wedge \text{S2Up}) \wedge \text{LightOn}$ for “Both switches are up and the light is on;”
- $(\text{S1Up} \wedge \text{S2Up}) \wedge \neg\text{LightOff}$ for “Both switches are up and the light is not off;”
- $\neg(\text{LightOn} \wedge \text{LightOff})$ for “It is not the case that the light is both on and off;”
- $(\neg\text{LightOn}) \wedge \neg\text{LightOff}$ for “The light is neither on nor off;”
- $(\text{LightOn} \vee \text{LightOff}) \wedge \neg(\text{LightOn} \wedge \text{LightOff})$ for “The light is either on or off.”

Observe that we have used parentheses in order to be able to read formulas in an unambiguous way. We will say more about this in Sect. 2.1.7.

Other Boolean Connectives According to the semantics of propositional logic, $A \vee B$ has the same meaning as $\neg((\neg A) \wedge (\neg B))$. We could therefore restrict the set of boolean connectives to $\neg$ and $\wedge$.

Instead of fewer boolean connectives we may as well use more, such as $\top$ (“True”), $\rightarrow$ (“If...then...”, or “Implies”), $\leftrightarrow$ (“Is equivalent to”), and $\oplus$ (“Either...or...”). However, they can be defined in terms of $\perp$, $\neg$, $\wedge$, and $\vee$:

- $\top$ is defined as $\neg\perp$;

- $A \rightarrow B$ is defined as $(\neg A) \vee B$;
- $A \leftrightarrow B$ is defined as $(A \rightarrow B) \wedge (B \rightarrow A)$;
- $A \oplus B$ is defined as $(A \vee B) \wedge \neg(A \wedge B)$.

We will use these abbreviations throughout the book.

One may even think of connectives involving more than two formulas, such as the ternary “Either... or... or...” However, all of them can be defined in terms of the above ones. This is the case because the set of boolean connectives is *complete*: any boolean function can be captured by a combination of $\perp$, $\neg$, $\wedge$, and $\vee$.

2.1.3 Modal Connectives

Atomic propositions and boolean connectives allow us to talk about the labels of a *single* graph node. But we also want to talk about several nodes. For instance, we may want to say that the light is off *in the current state* and on *in the state that obtains after toggling the switch*; or we may want to say that the card is red *in the actual state*, and that *there is a state that is possible for Bob where the card is not red*; etc. In order to be able to do that we introduce *modal connectives*.

The basic form of a modal connective is either $[I]$ or $\langle I \rangle$, where I is an *edge label* from some set of edge labels $\mathcal{I}$ that we suppose to be given.

When $\mathcal{I}$ is a singleton $\{I\}$, i.e., when there is just one edge label I , then we in general write $\Box$ and $\Diamond$ instead of $[I]$ and $\langle I \rangle$. In older texts, instead of $\Box$ and $\Diamond$ the reader may encounter the notation L and M . This is in particular the case when the terms of necessity and possibility are understood in the strict sense (as opposed to the large sense, encompassing in particular epistemic and deontic necessity and possibility). In this case we also talk about *alethic operators*.

The modal connective $[I]$ together with a formula A makes up a new formula $[I]A$. We read that formula “For all I -successors (of the current node), A is true” or “ A is necessary w.r.t. I ”. We therefore call $[I]$ a *necessity operator*.

The modal connective $\langle I \rangle$ together with A makes up a new formula $\langle I \rangle A$, read “For some I -successor (of the current node), A is true” or “ A is possible w.r.t. I ”. We therefore call $\langle I \rangle$ a *possibility operator*.

The readings we have given are generic and may be altered in some contexts. Here are some examples:

- When reasoning about knowledge we read for example $[\text{Ann}]\text{Red}$ as “Ann knows that the card is red” and we read $\langle \text{Ann} \rangle \text{Red}$ as “It is compatible with Ann’s knowledge that the card is red;”
- When reasoning about programs and actions we read for example $[\text{Toss}]\text{Heads}$ as “After every execution of the tossing action the coin is heads” and we read $\langle \text{Toss} \rangle \text{Heads}$ as “After some execution of the tossing action the coin is heads.”

Actually the notation $[I]$ and $\langle I \rangle$ for modal operators is also generic, and we sometimes choose more mnemonic symbols in order to improve the readability of

Table 2.1 Typical modal connectives and their reading

Formula	Typical reading	Modality
[Toss]Heads	“After <i>every</i> possible execution of the action of tossing a coin, heads will lie face up”	dynamic
(Toss)Heads	“After <i>some</i> possible execution of the action of tossing a coin, heads will lie face up”	dynamic
$K_{\text{Ann}}\text{Red}$	“Ann knows that the card is red”	epistemic
$\hat{K}_{\text{Ann}}\text{Red}$	“It is possible for Ann that the card is red”	epistemic
$B_{\text{Ann}}\text{Red}$	“Ann believes that the card is red”	doxastic
$O_I \text{ Stop}$	“It is obligatory that <i>I</i> stops”	deontic
$P_I \text{ Stop}$	“It is permitted that <i>I</i> stops”	deontic
$X \text{ Red}$	“The traffic light will be red next”	temporal
$G \text{ Red}$	“The traffic light will be red henceforth”	temporal
$F \text{ Red}$	“The traffic light will eventually be red”	temporal
$\text{Red } U \text{ Green}$	“The traffic light will be red until it turns green”	temporal

formulas with several modalities. This is particularly useful when there are modalities of different kinds, such as several agent names and several program names. For example, consider a language whose set of edge labels $\mathcal{I}$ is made up of a temporal ‘next’ label X plus a set of edge labels $\mathcal{I}_1$ denoting agents, with $X \notin \mathcal{I}_1$. Then we typically write XA instead of $[X]A$ in order to highlight that the nature of the temporal operator is different from that of the epistemic operators $[I]$ with $I \in \mathcal{I}_1$. Likewise, consider the case of edge labels that are of two different kinds: agents and actions. Let $\mathcal{I} = \mathcal{I}_1 \cup \mathcal{I}_2$ where $\mathcal{I}_1$ is the set of agents and $\mathcal{I}_2$ is the set of actions and where $\mathcal{I}_1$ and $\mathcal{I}_2$ are disjoint, i.e., $\mathcal{I}_1 \cap \mathcal{I}_2 = \emptyset$. We distinguish the two kinds of modal operators by varying notation: we write the epistemic operators K_I , one per agent $I \in \mathcal{I}_1$, and we write the dynamic operators $[a]$, one per action $a \in \mathcal{I}_2$. Even when the knowledge operator is the only modal operator then $[I]$ and $\langle I \rangle$ are typically written K_I and $\hat{K}_I$, i.e., a ‘ K ’ with a *hat*, that may be pronounced “k-hat.” Some examples of customary writings of modal operators are collected in Table 2.1.

Remark 3 Note that we did not impose any restriction on the set of edge labels $\mathcal{I}$. It may in particular be made up of formulas again. The temporal operators ‘until’ and ‘since’ are examples of operators that are indexed by formulas. The formula $[UA]B$ is read “ A until B ,” and the formula $[SA]B$ is read “ A since B .” Just as other so-called binary modal operators, such formulas are usually written $A \cup B$ and $A \text{ S } B$ instead of $[UA]B$ and $[SA]B$, as done in the last line of Table 2.1.

Together, boolean and modal connectives allow us to express a lot of things. For example, $\langle \text{Open} \rangle \top$ can be read “The action of opening the door is possible,” and $[\text{Open}] \perp$ can be read “The door cannot be opened.” Here are some more examples. The sentence “The light is off and after toggling the switch it will be on” can be

expressed by the formula

$$\neg \text{LightOn} \wedge [\text{Toggle}]\text{LightOn}$$

The sentence “Ann knows whether the card is red or not” can be written

$$K_{\text{Ann}}\text{Red} \vee K_{\text{Ann}}\neg\text{Red}$$

The sentence “Ann does not know whether the card is red, but knows that Bob does” gets

$$\neg(K_{\text{Ann}}\text{Red} \vee K_{\text{Ann}}\neg\text{Red}) \wedge K_{\text{Ann}}(K_{\text{Bob}}\text{Red} \vee K_{\text{Bob}}\neg\text{Red})$$

The sentence “Toggling the switch twice does not change the state of the light” gets³

$$(\text{Light} \rightarrow [\text{Toggle}][\text{Toggle}]\text{Light}) \wedge (\neg\text{Light} \rightarrow [\text{Toggle}][\text{Toggle}]\neg\text{Light})$$

Finally, “Next the light will be either red or green” gets⁴

$$X((\text{Red} \vee \text{Green}) \wedge \neg(\text{Red} \wedge \text{Green}))$$

Not any sequence of atomic formulas and connectives makes up a formula: the construction has to follow some rules that we are going to detail in the sequel.

2.1.4 Duality of Modal Connectives

We have seen that the modal connectives come in two kinds: necessity operators $[I]$ and possibility operators $\langle I \rangle$. Both quantify over all nodes that are accessible via I edges: the former is a ‘for all’ quantifier (just as the universal quantifier $\forall$ of first-order logic), while the latter is a ‘there is’ (just as the existential quantifier $\exists$).

We have noted in Sect. 2.1.2 that one may consider a language whose only boolean connectives are $\neg$ and $\wedge$. As to the modal connectives, it will turn out that $\langle I \rangle A$ has the same meaning as $\neg[I]\neg A$. We may therefore consider the former as an abbreviation of the latter, just as we might consider $A \vee B$ to be an abbreviation of $\neg(\neg A \wedge \neg B)$. (We are actually going to do so in Sect. 4.7 in order to shorten the presentation.) The other way round, we could as well consider $[I]A$ to be an abbreviation of $\neg\langle I \rangle\neg A$. The connectives $[I]$ and $\langle I \rangle$ are called *dual*.⁵ This is the same duality as in first-order logic, where $\forall x A$ has the same meaning as $\neg\exists x\neg A$ and $\exists x A$ has the same meaning as $\neg\forall x\neg A$.

³Remember that our primitive boolean connectives are $\perp$, $\neg$, $\wedge$, and $\vee$ and that the formula $A \rightarrow B$ abbreviates $\neg A \vee B$.

⁴Using abbreviations this could also be written as $X(\text{Red} \leftrightarrow \neg\text{Green})$ or $X(\text{Red} \oplus \text{Green})$.

⁵Note that duality is no longer the case if we replace the boolean connectives for intuitionistic connectives (cf. see Sect. 5.3): just as $\forall x$ and $\exists x$ are no longer dual in intuitionistic predicate logic, $[I]$ and $\langle I \rangle$ are no longer dual in intuitionistic modal logics: they have different meanings there.

2.1.5 The Definition of Formulas

We now give our official definition of formulas that will remain the same throughout the book.

We suppose given a countable set of node labels $\mathcal{P}$ and a countable set of edge labels $\mathcal{I}$. In logic the former are also called *atomic formulas* (or propositional variables), while the latter are sometimes called modalities; we here call them indexes because they can be viewed as indexes of the modal operators $\Box$ and $\Diamond$. Note that we do not suppose that $\mathcal{P}$ and $\mathcal{I}$ are disjoint: they may have symbols in common.

The set of formulas is then built from these sets by means of the logical operators.

Definition 4 (Formula) Let $\mathcal{P}$ and $\mathcal{I}$ be countable sets. The set of formulas $\mathcal{F}or$ is the smallest set such that:

1. $\mathcal{P} \subseteq \mathcal{F}or$;
2. $\perp \in \mathcal{F}or$;
3. if $A \in \mathcal{F}or$ then $\neg A \in \mathcal{F}or$;
4. if $A, B \in \mathcal{F}or$ then $(A \wedge B) \in \mathcal{F}or$;
5. if $A, B \in \mathcal{F}or$ then $(A \vee B) \in \mathcal{F}or$;
6. if $A \in \mathcal{F}or$ and $I \in \mathcal{I}$ then $[I]A \in \mathcal{F}or$;
7. if $A \in \mathcal{F}or$ and $I \in \mathcal{I}$ then $\langle I \rangle A \in \mathcal{F}or$.

The first clause $\mathcal{P} \subseteq \mathcal{F}or$ means that each atomic formula is a formula. For instance, *Black*, *LightOn* and *Red* are formulas. The second clause says that if we prefix a formula by means of the negation connective $\neg$ then we obtain a formula. The third and fourth clauses say that if we connect a formula with another formula by means of the connectives $\wedge$ or $\vee$ then we obtain a formula. For instance, since *LightOn* and *Red* are formulas, $(\text{LightOn} \wedge \text{Red})$ is a formula too. The last two clauses say that if A is a formula and I is an index then $[I]A$ and $\langle I \rangle A$ are formulas. For instance $(\text{LightOn} \wedge \text{Red})$ being a formula and *Open* being an index, $[\text{Open}](\text{LightOn} \wedge \text{Red})$ is a formula.

When $\mathcal{I}$ is empty then our language is simply that of propositional logic. When $\mathcal{I}$ is a singleton then we call the language *monomodal*. If $\mathcal{I}$ contains more than one element then we call the language *multimodal*.

Remark 4 As we have mentioned in Sects. 2.1.2 and 2.1.4, we might either consider more connectives, or fewer. Actually LOTREC is very flexible and allows for languages that are built from *any* set of connectives. We might for instance define a boolean language whose only connectives are $\top$ ('true') and $\oplus$ ('exclusive or').

Remark 5 As we have said in Remark 3, there are modal connectives having two formulas as arguments, such as the temporal operators U ('until') and S ('since'). In order to fit in the above definition, we have to replace the usual notation AUB for the 'until' operator by $[UA]B$ (but we note already that LOTREC is more flexible than that and allows us to display this in the standard way). These operators fit in the

above schema because we imposed no constraints on the set of edge labels $\mathcal{I}$; it may in particular contain formulas. However, when the indexes of $\mathcal{I}$ are not all atomic but have a structure then their rigorous definition requires an inductive construction by means of index connectives. If moreover the indexes may contain formulas (as is the case of the $\mathbf{U}$ operator of **LTL**) then one needs a definition of formulas and indexes by mutual induction. We do not go into the details of such definitions here and rather refer the reader to the formal definition of the language of **LTL** in Sect. 7.1.

Another example of a complex index structure is provided by the programs of propositional dynamic logic **PDL**, where complex programs may in particular be sequential and nondeterministic composition of programs as well as tests of the truth of a formula: the program ‘ $A?$ ’ succeeds if A is true and fails otherwise. Just as for **LTL**, the rigorous definition of the language requires an inductive construction (see Sect. 7.2).

The Connectives of Description Logics The connectives of description logics are written in a way that differs from what we have seen up to now. While the negation symbol takes the same form, conjunction and disjunction are not written $\wedge$ and $\vee$ but $\sqcap$ and $\sqcup$, and modal operators are not written $[I]$ and $\langle I \rangle$ but $\forall I$ and $\exists I$. Moreover, there is a convention to present the logics using A for atomic formulas, C for formulas (“concepts”), and R for indexes (“relations”). The formula (“concept”) $\forall R.C$ reads “every object that is related to the actual object (the object currently under consideration) by R has property C ,” and $\exists R.C$ reads “there is an object that is related to the actual object by R and that has property C .” For example,

$$\text{female} \sqcap \forall \text{ParentOf.Male}$$

says that the object under consideration is female and that all her children are male.

Description logics also have some more constructions allowing one to state that a node has some (possibly complex) label and that some concept is included in another concept. Some extensions also allow for world names in the language. We do not state them here and restrict our attention to those mirroring the modal language. We just remark that names for worlds are also encountered in hybrid logics [AtC06] which we introduce in Sect. 4.8.

2.1.6 Analysing a Formula: Subformulas, Formula Length, Arity

The formulas from the set $\mathcal{P}$ are called *atomic* because they cannot be decomposed into smaller formulas. In contrast, $(\text{Red} \wedge \text{LightOn})$ is not atomic but complex and can be decomposed into Red and LightOn . We say that Red and LightOn are *subformulas* of $(\text{Red} \wedge \text{LightOn})$. By convention, $(\text{Red} \wedge \text{LightOn})$ is also a subformula of itself. The formula $\mathbf{K}_{\text{Ann}}(\text{Red} \vee \text{Black})$ consists of the connective $\mathbf{K}_{\text{Ann}}$ governing the subformula $(\text{Red} \vee \text{Black})$; one also says that the latter is in the scope of the modal connective $\mathbf{K}_{\text{Ann}}$. The subformula $(\text{Red} \vee \text{Black})$ has two other subformulas: Red

and Black, combined by the connective $\vee$. All of these formulas are subformulas of $K_{\text{Ann}}(\text{Red} \vee \text{Black})$.

To compute all the subformulas of a given formula we need an inductive definition. Here it is.

Definition 5 (Subformulas) The set of subformulas of a given formula A —denoted by $\text{SF}(A)$ —is inductively defined as:

$$\begin{aligned}
 \text{SF}(P) &= \{P\}, \quad \text{for } P \in \mathcal{P} \\
 \text{SF}(\perp) &= \{\perp\} \\
 \text{SF}(\neg B) &= \{\neg B\} \cup \text{SF}(B) \\
 \text{SF}(B \wedge C) &= \{B \wedge C\} \cup \text{SF}(B) \cup \text{SF}(C) \\
 \text{SF}(B \vee C) &= \{B \vee C\} \cup \text{SF}(B) \cup \text{SF}(C) \\
 \text{SF}([I]B) &= \{[I]B\} \cup \text{SF}(B) \\
 \text{SF}(\langle I \rangle B) &= \{\langle I \rangle B\} \cup \text{SF}(B)
 \end{aligned}$$

For example:

$$\begin{aligned}
 \text{SF}(\langle I \rangle \neg P) &= \{\langle I \rangle \neg P\} \cup \text{SF}(\neg P) \\
 &= \{\langle I \rangle \neg P\} \cup \{\neg P\} \cup \text{SF}(P) \\
 &= \{\langle I \rangle \neg P, \neg P, P\}
 \end{aligned}$$

Remark 6 Connectives that are defined as abbreviations such as material implication $\rightarrow$ have to be decomposed according to their definition. For example:

$$\begin{aligned}
 \text{SF}(P \rightarrow Q) &= \text{SF}(\neg P \vee Q) \\
 &= \{\neg P \vee Q, \neg P, Q, P\}
 \end{aligned}$$

Remark 7 The definition of SF can be extended in the obvious way if we want to adopt other primitive connectives. For example, if we add material implication $\rightarrow$ as a primitive then

$$\text{SF}(B \rightarrow C) = \{B \rightarrow C\} \cup \text{SF}(B) \cup \text{SF}(C)$$

and consequently $\text{SF}(P \rightarrow Q)$ then becomes the set $\{P \rightarrow Q, P, Q\}$.

Remark 8 In the case of richer modal languages where the argument I of the modal operator $[I]$ is a formula or contains a formula the above definition has to be modified in order to also extract the subformulas contained in I .

Definition 6 (Length of a formula) Given a formula A , the length of A , denoted by $\text{length}(A)$, is inductively defined as:

$$\begin{aligned}
 \text{length}(P) &= 1, \quad \text{for } P \in \mathcal{P} \\
 \text{length}(\perp) &= 1 \\
 \text{length}(\neg B) &= 1 + \text{length}(B) \\
 \text{length}(B \wedge C) &= 1 + \text{length}(B) + \text{length}(C) \\
 \text{length}(B \vee C) &= 1 + \text{length}(B) + \text{length}(C) \\
 \text{length}([I]B) &= 1 + \text{length}(B) \\
 \text{length}(\langle I \rangle B) &= 1 + \text{length}(B)
 \end{aligned}$$

Again, the definition may have to be by mutual induction if the set of edge labels has a richer structure.

The number of subformulas of a given formula A is bounded by the length of A : we have $\text{card}(\text{SF}(A)) \leq \text{length}(A)$. (This is proved by induction on the structure of A .)

Arity of Connectives The number of expressions that are combined by a given connective is called its *arity*. For example, the arity of $\wedge$ is 2 and the arity of $\neg$ is 1. As $[I]A$ is built from two arguments (I and A) the arity of $[.]$ is 2. A connective of arity 1 is called *unary* and a connective of arity 2 is called *binary*.⁶ We give the arity of some standard connectives in Table 2.2 below.⁷

2.1.7 Parenthesis Conventions

The formula $K_{\text{Ann}}(\text{Red} \rightarrow \text{Red})$ can be read just in one way: “Ann *knows* that if the card is red then it is red.” Parentheses are important: suppose we omit them and just write $K_{\text{Ann}}\text{Red} \rightarrow \text{Red}$. This expression can be parenthesised in two different ways: in the way we gave above and as $(K_{\text{Ann}}(\text{Red}) \rightarrow \text{Red})$, read “If Ann knows that the card is red then it is red.” The point is that according to the first reading K_{Ann} is the main connective, whereas according to the second reading $\rightarrow$ is the main connective.

⁶The notion of arity in logic is the same as in arithmetic, where each operator has a fixed number of arguments; for example the arity of $+$ is two, and the arity of $\sqrt{}$ is 1 (if it is the quadratic root; else it is 2).

⁷We note that in modal logic it is usual to only count the argument of $[.]$ if $[.]$ contains a formula, as is the case in linear-time temporal logic **LTL**. When the set of edge labels $\mathcal{Act}$ is a singleton I then it is usually considered that $[.]$ is a 0-ary operator. When the set of edge labels $\mathcal{Act}$ is made up of atomic labels then it is usual to say that $[.]$ is unary.

However, some of the parentheses can be omitted if we adopt the following conventions.

1. The outer parentheses are dropped. For example $(A \wedge (B \vee C))$ is written $A \wedge (B \vee C)$.
2. Since in propositional logic $A \wedge (B \wedge C)$ has the same meaning as $(A \wedge B) \wedge C$, we may drop the parentheses and just write $A \wedge B \wedge C$. This generalizes to an arbitrary number of conjuncts; e.g. $A_1 \wedge ((A_2 \wedge (A_3 \wedge A_4)) \wedge A_5)$ is written $A_1 \wedge A_2 \wedge A_3 \wedge A_4 \wedge A_5$. Similarly, we drop all parentheses inside a sequence of $\vee$ and write e.g. $A \vee B \vee C$ instead of $(A \vee B) \vee C$.
3. We may specify an order of strength between the connectives. First, it is usually considered that unary connectives bind stronger than binary connectives. For example $\neg A \wedge A$ is identified with $(\neg A) \wedge A$ and $\langle I \rangle A \wedge A$ is identified with $(\langle I \rangle A) \wedge A$. Second, it is commonly considered that $\wedge$ binds stronger than $\vee$: for example, $A \wedge B \vee C$ is identified with $(A \wedge B) \vee C$. Third, concerning the boolean connectives that are abbreviations it is commonly considered that $\vee$ binds stronger than $\rightarrow$ and that $\rightarrow$ binds stronger than $\leftrightarrow$.

These conventions do not allow us to entirely drop parentheses; for example it fails to disambiguate $A_1 \rightarrow A_2 \rightarrow A_3$. We might go further and stipulate that such sequences of $\rightarrow$ associate to the right; then $A_1 \rightarrow A_2 \rightarrow A_3$ is identified with $A_1 \rightarrow (A_2 \rightarrow A_3)$. However, a general problem with such conventions is that when formulas get bigger then it requires more and more cognitive efforts to identify the main connective.

2.2 Syntax Declaration in LoTREC

In the preceding section we have learned how to define a formal language on paper, starting from a basic set of node labels $\mathcal{P}$, a basic set of edge labels $\mathcal{L}$, and logical connectives. We now show how to define such a language in LoTREC, so that we can use the formulas as graph labels and indexes as edge labels.

We have already announced in Sect. 2.1.5 that the connectives $\neg$, $\wedge$, $\vee$, $[.]$, and $\langle.\rangle$ are a particular choice: LoTREC is more general and allows for *any* set of connectives. Moreover, LoTREC allows us to write the modal operators not just in the generic forms $[I]$ and $\langle I \rangle$, but as any expression, as soon as it has been declared as such in the definition of the language.

We have also said in Sect. 2.1.5 that the set of edge labels might have some structure and that edge labels may as well be formulas. In order to provide the most general framework, LoTREC adopts an even more general stance and does not distinguish between node labels and edge labels: the general principle is that the set of complex labels $\mathbf{L}$ is built by means of connectives from a set of *set of atomic LoTREC labels* $\mathbf{L}_0$. We use *expression* as a general term for a—possibly complex—label.

In Sect. 1.8 we have introduced the typewriter font convention: whenever you encounter typewriter font then you know that we are talking about the language of

LoTREC, i.e., things that are either implemented or implementable in LoTREC. This extends to atomic and complex labels. It will also apply to things such as tableau rules and strategies that we will introduce later on.

2.2.1 Prefix Notation

In our official definition of formulas we used parentheses systematically. There is however another solution: we may write down a formula in prefix form, also called Polish notation. It consists in writing it down starting by the main connective of the formula. Then we successively write down each subformula, also in prefix form. For example, the formula $(K_{\text{Ann}}\text{Red}) \wedge \text{Red}$ is written in prefix form as $\wedge K_{\text{Ann}} \text{Red Red}$. In contrast, the formula $K_{\text{Ann}}(\text{Red} \wedge \text{Red})$ is written $K_{\text{Ann}} \wedge \text{Red Red}$.

Observe that there is no ambiguity in the prefix notation under the following conditions: connectives are identified as such, their arity is fixed, and there cannot be two connectives with the same name.

While prefix notation was popular in logic books in the past, nowadays presentations use infix notation and parentheses. In LoTREC we however chose to use prefix notation because this allows us to do without parsing mechanisms. For example, in order to enter the formula $P \wedge \neg Q$ the LoTREC user has to type and P not Q. This may burden the users at first, especially new users. However, we are confident that the LoTREC user will soon be familiar with it. Moreover, formulas written in prefix notation can be displayed in infix notation on the screen. This is explained in Sect. 2.2.5.

2.2.2 Defining Atomic Labels

By convention the elements of LoTREC's set of atomic labels L_0 are *words starting with a capital letter*. Examples are Red, Black, LightOn, LightOff, Open, Toggle, Ann, Bob, I, P, Q, J, etc.

2.2.3 Defining Connectives

The LoTREC user can define his own connectives at will. We are therefore not limited to the connectives $\perp$, $\neg$, $\wedge$, $\vee$, $[\cdot]$, and $\langle \cdot \rangle$. There is however one connective that is predefined: the 0-ary falsum connective $\perp$, written False in LoTREC and displayed FALSE. This is because graph nodes containing that connective have a particular status when we try to build a model: such graphs cannot be transformed into a legal model of the logic, and LoTREC signals this by colouring nodes that are labelled FALSE.

Table 2.2 Definitions of the standard connectives in LoTREC

Connective	Name	Arity	Displayed as	Priority	Associative
$\perp$	False	0	FALSE		
$\neg$	not	1	$\sim$ _	5	yes
$\wedge$	and	2	(_ & _)	3	yes
$\vee$	or	2	(_ $\vee$ _)	2	yes
[.]	neci	2	[_] _	4	yes
<.>	posi	2	< _ > _	4	yes

In order to define a connective the user has to enter the following information:

- Its *name*, which is any word *starting with a small letter*;
- Its *arity*, which is a natural number whose default value is 1;
- The way it is *displayed* on screen;
- Its *priority* in order to define the relative strength of binding, which is a natural number whose default value is 0;
- Whether parentheses can be dropped when the connective occurs twice in a row, the default being that they cannot. In the case of binary connectives this is the same as declaring whether it is *associative* or not.

In textbooks and articles, parentheses in iterated unary connectives are suppressed. For example, one writes $\neg\neg P$ and $[I][J]P$ instead of $\neg(\neg P)$ and $[I]([J]P)$. We stress that LoTREC does not do so by itself: the user has to declare explicitly that parentheses should not be displayed by ticking the ‘associative’ box in the LoTREC interface; otherwise, LoTREC will display them.

Table 2.2 contains the definitions of some standard connectives. We use the names *not*, *and*, and *or* for the boolean connectives and the names *neci* and *posi* for the generic, indexed modal connectives. (Priorities and associativity of the connective *False* may have given arbitrary values because it has no arguments.)

Beyond the two generic modal connectives [.] and <.>, other connectives can be defined in LoTREC, such as the epistemic connective *knows* and the temporal connectives *next*, *henceforth*, *eventually*, *until*, and *since*. The temporal connectives *next*, *henceforth*, and *eventually* have arity 1, while *until* and *since* have arity 2. The epistemic connective *knows* typically has arity 2, taking as argument an agent and a formula; however, when only a single agent is considered then the agent argument is often dropped and *knows* typically has arity 1. Some examples are collected in Table 2.3. Table 2.4 lists the description logic connectives that we have introduced in Sect. 2.1.5.

The information about display, priority, and associativity is used by LoTREC in order to drop parentheses when displaying formulas. In Sect. 2.2.5 we explain how this works in detail.

Table 2.3 Definitions of supplementary connectives in LoTREC

Connective	Name	Arity	Displayed as	Priority	Associative
$\rightarrow$	imp	2	($_ \rightarrow _$)	2	no
$\leftrightarrow$	eqv	2	($_ \leftrightarrow _$)	1	no
$\square$	nec	1	[$_$]	4	yes
$\diamond$	pos	1	$\langle _ \rangle$	4	yes
X	next	1	X $_$	4	yes
K	knows	2	K($_$) $_$	4	yes
$\hat{K}$	knowshat	2	$K^{\wedge}(_) _$	4	yes
B	bel	2	B($_$) $_$	4	yes
U	until	2	($_ U _$)	4	no

Table 2.4 Definitions of description logic connectives in LoTREC

Connective	Name	Arity	Displayed as	Priority	Associative
$\perp$	empty	0	FALSE		
$\neg$	dnot	1	$\sim _$	4	yes
$\sqcap$	dand	2	($_ \& _$)	2	no
$\sqcup$	dor	2	($_ \vee _$)	2	no
$\forall$	dforall	2	A $_ . _$	4	yes
$\exists$	dexists	2	E $_ . _$	4	yes

2.2.4 Complex Labels in LoTREC

The set of labels L of LoTREC's language is the smallest set such that:

- every element of L_0 is a label;
- if c is a connective of arity n and $A_1, \dots, A_n$ are labels of L_0 then $c \ A_1 \ \dots \ A_n$ is a label.

Some example formulas built from the connectives of Tables 2.2 and 2.3 are given in Table 2.5.

Remember that there is no ambiguity in the prefix notation in particular because connectives are identified by their initial small letter. We can therefore safely write $c \ A_1 \ \dots \ A_n$ without parentheses, instead of $c \ (A_1, \dots, A_n)$.

The following LoTREC rule extracts the subformulas of any formula of the form $A \wedge B$ and adds them to the node containing it.

Rule BreakupConjunctions

hasElement node and variable A variable B

add node variable A

add node variable B

Table 2.5 Infix notation vs. LoTREC's prefix notation: examples

Formula on paper	LoTREC syntax
$P \vee (Q \wedge R)$	or P and Q R
$(P \vee Q) \wedge R$	and or P Q R
$P \wedge \neg Q$	and P not Q
$K_I(P \wedge \neg Q)$	knows I and P not Q
$K_I P \wedge \neg Q$	and knows I P not Q
$K_{Ann}(Red \vee Black)$	knows Ann or Red Black

It can be implemented in LoTREC in the same way as we did for the model building rule `BuildExampleGraph` in Sect. 1.7 of Chap. 1 (page 16): add the rule `BreakupConjunctions` in the rule tab and edit the default strategy in the strategy tab such that it calls `BreakupConjunctions`. Observe that in that rule `A` and `B` are variables for formulas because they are prefixed by the LoTREC keyword variable. In contrast, the atomic labels in the formulas to be decomposed are constants (starting with a capital letter).

Exercise 16 Using `hasElement` conditions and add actions, write LoTREC rules which decompose formulas of the form $\neg A$, $A \rightarrow B$, and $[I]A$ into subformulas, one per connective. Distinguish the case where the implication connective is primitive from the case where it is an abbreviation.

2.2.5 Displaying Formulas

LoTREC allows the user to specify how formulas are displayed on the screen. He may in particular choose to display formulas in the familiar infix notation.

The way a given n -ary connective is displayed can be defined by a string of arbitrary characters with exactly n underscores ‘`_`’.

Table 2.2 contains the standard display definitions for some connectives. Accordingly, the LoTREC formula `or Red Black` is displayed as `Red  $\vee$  Black` and the LoTREC formula `knows Ann Red` is displayed as `K (Ann) Red`.

Now we turn to the way LoTREC allows us to economise on some parentheses when displaying formulas.

As we have mentioned in Sect. 2.1.7, it is a common convention in logic that unary connectives bind stronger than conjunction, allowing one to write for example $\neg A \wedge B$ instead of $(\neg A) \wedge B$. This is declared to LoTREC by giving to the connective `not` a higher priority than to `and`. The priorities of some standard connectives are in Table 2.2. Note that no such information is required for zero-ary connectives such as $\perp$ (its LoTREC priority may be set to any value). Consider for example the formula $K_I P \wedge Q \rightarrow R$, written in LoTREC's prefix notation as `imp and knows I P Q R`. When given the priorities of Table 2.2, LoTREC displays this as `K (I) P & Q -> R`. The fact that the negation `not` has priority higher

Table 2.6 Infix notation, LoTREC’s prefix notation, and LoTREC’s display: examples

Formula on paper	LoTREC syntax	LoTREC display
$P \vee (Q \wedge R)$	or P and Q R	$P \vee Q \ \& \ R$
$(P \vee Q) \wedge R$	and or P Q R	$(P \vee Q) \ \& \ R$
$K_I(P \wedge Q)$	knows I and P Q	$K(I) \ (P \text{ and } Q)$
$K_I P \wedge Q$	and knows I P Q	$K(I) \ P \text{ and } Q$
$K_{\text{Ann}}(\text{Red} \vee \text{Black})$	knows Ann or Red Black	$K(\text{Ann}) \ (\text{Red or Black})$

than for example the modal operator $[\]$ makes that LoTREC displays the formula $\Box \neg P$ as $[\] \ \sim P$ on the one hand, and displays the formula $\neg \Box P$ as $\sim ([\] \ P)$ on the other.

Finally, if we stipulate that a connective is associative then parentheses of sequences of the same connective are dropped. For example, the standard conjunction and disjunction connectives are both associative, while the implication connective is not. In the LoTREC interface we have to tick the respective fields in the definition of these connectives. For example, under the declarations of Table 2.2, the formulas $\text{and and } P \ Q \ R$ and $\text{and } P \text{ and } Q \ R$ are displayed in the same way, namely as $P \ \& \ Q \ \& \ R$. Unary connectives should be declared to be associative (even if this is a bit at odds with the usual sense of the term): we can drop parentheses from double negation $\neg(\neg A)$ because there is only one way of parenthesising $\neg\neg A$.

The way some of the example formulas of Table 2.5 are displayed is given in Table 2.6.

2.3 Truth Conditions

In Chap. 1 we modelled various situations with Kripke models. In this section we formally define how formulas are evaluated in a possible world of a Kripke model as defined in Sect. 1.6.

Remember that the set of connectives is $\neg, \wedge, \vee, [I]$, and $\langle I \rangle$.

Given a Kripke model M , a state w of M , and a formula A , we want to define what it means that A is true at w . For instance, suppose M models the functioning of a coffee machine and w represents the state “Your coin has been inserted.” How can we evaluate whether the formula $[\text{selectCoffee}]\text{Coffee}$ (“After having selected a coffee, I will have a coffee”) is true?

Formally we write $M, w \Vdash A$ for “The formula A is true in the world w of the model M ” and define the relation $\Vdash$ as follows.

Definition 7 (Truth conditions) The forcing relation $\Vdash$ between models, worlds, and formulas is inductively defined to be the smallest relation such that:

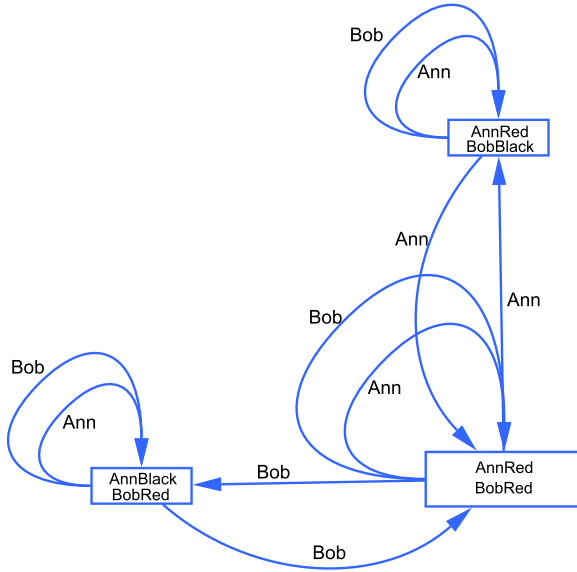


Fig. 2.1 Ann and Bob playing cards (possible world at top is w_0)

- $$\begin{aligned}
 M, w \models P & \quad \text{iff } w \in V(P), \text{ for } P \in \mathcal{P}; \\
 M, w \not\models \perp; \\
 M, w \models \neg A & \quad \text{iff } M, w \not\models A; \\
 M, w \models A \wedge B & \quad \text{iff } M, w \models A \text{ and } M, w \models B; \\
 M, w \models A \vee B & \quad \text{iff } M, w \models A \text{ or } M, w \models B; \\
 M, w \models [I]A & \quad \text{iff for every world } u, \text{ if } (w, u) \in R(I) \text{ then } M, u \models A; \\
 M, w \models \langle I \rangle A & \quad \text{iff there is a world } u \text{ such that } (w, u) \in R(I) \text{ and } M, u \models A.
 \end{aligned}$$

According to this definition, an atomic formula P is true in a world w of a model M iff the valuation of P contains w . A formula of the form $\neg A$ is true in w iff the formula A is false in w . A formula $A \wedge B$ is true in w iff both A and B are true in w , and $A \vee B$ is true in w iff A or B are true in w . Finally, the last two items say that formula $[I]A$ is true in w iff A is true in *all* the possible worlds u that are accessible from w by the relation $R(I)$; and formula $\langle I \rangle A$ is true in w iff A is true in *some* possible world u that is accessible from w by the relation $R(I)$.

Let us go back to Fig. 1.16 that describes a model M that we repeat as Fig. 2.1. Remember that the possible world at the top of the figure is called w_0 . We can now state the following:

- $M, w_0 \models \neg \text{BobRed}$;
- $M, w_0 \models \text{AnnRed} \wedge \text{BobBlack}$;

- $M, w_0 \Vdash \neg(\text{AnnRed} \wedge \text{BobRed})$;
- $M, w_0 \Vdash [\text{Ann}]\text{AnnRed}$;
- $M, w_0 \Vdash \langle \text{Ann} \rangle \neg \text{BobRed}$; this is the case because there is one world accessible by $R(\text{Ann})$ from w_0 (namely w_0 itself) where BobRed is false.

Observe that the boolean connectives $\neg$, $\wedge$, and $\vee$ are *truth-functional*: the truth value of a negation $\neg A$ is a function of the truth value of A , and the truth values of $A \wedge B$ and $A \vee B$ are functions of the truth value of their constituents A and B . In contrast, modal connectives are *not* truth-functional: the truth value of $[I]A$ and $\langle I \rangle A$ does not depend on the truth value of A .

Back to Duality According to our truth conditions, for every M and w we have

$$M, w \Vdash \langle I \rangle A \quad \text{iff} \quad M, w \Vdash \neg[I]\neg A$$

$$M, w \Vdash [I]A \quad \text{iff} \quad M, w \Vdash \neg\langle I \rangle \neg A$$

Therefore and as we have announced in Sect. 2.1.4, the notions of possibility and necessity are interdefinable: one may alternatively only use the necessity operator and consider the possibility operator to be an abbreviation, or the other way round. Often it is convenient to have both modal connectives as primitives of the language, which is what we do here.

2.4 How to Do Model Checking in LoTREC

We now explain how to use the procedure `Model-Checking-Multimodal` that comes with the LoTREC distribution. The procedure works for a language with modal operators $[I]$ and $\langle I \rangle$, written where I is some edge label. (If your set of indexes $\mathcal{I}$ is a singleton you might avoid typing the arguments of the modal operators and use the modal operators $[]$ and $\langle \rangle$; in that case you should apply the procedure `Model-Checking-Monomodal`.)

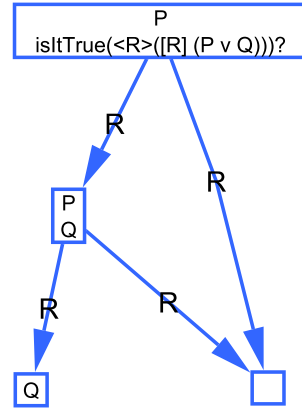
Let $M = (W, R, V)$ be a finite Kripke model and let w be a possible world from W . Let A be the formula of which we want to know whether $M, w \Vdash A$.

We open the ‘predefined logic’ `Model-Checking-Multimodal`⁸ and build the model M by means of a LoTREC rule in the same way we did in Sect. 1.7. Moreover add the formula `isItTrue(A) ?` to the root node w of M .

We illustrate model checking by an example. Let us check whether the formula $\langle R \rangle [R](P \vee Q)$ is true in the root w of the graph M depicted in Fig. 2.2, where w is the uppermost node of the graph.

⁸`Model-Checking-Multimodal` and `Model-Checking-Monomodal` are in LoTREC’s list of predefined logics, but they are not the tableaux procedure for a particular logic. (They are the only such exceptions in that list.)

Fig. 2.2 An input of the model-checking problem



We build the LoTREC graph of the model M by means of the following rule:

```

Rule BuildExampleGraph
  createNewNode w
  createNewNode u
  createNewNode v
  createNewNode x
  link w u R
  link w v R
  link v u R
  link v x R
  add w P
  add v P
  add v Q
  add x Q
  add w isItTrue pos R nec R or P Q

```

where `isItTrue` is defined in LoTREC to be a unary connective that is displayed as `isItTrue( )?`. This is not a modal operator of the logic, but an auxiliary connective that allows us to distinguish the formulas to be evaluated during the model checking process. Observe that the rule has finite length because M is finite: the set of nodes W to be created is finite and there is only a finite number of labels that have to be added to nodes and edges. Beyond the above rule `BuildExampleGraph` there are several other predefined rules in `Model-Checking-Multi-modal`. Their job is to break up the formula to be evaluated in a first pass, and then to compute the truth value of each subformula in a second pass. How this is done is explained in detail in Chap. 6.

Once the model building rule has been entered you may push the “Build Pre-models” button: LoTREC will then display the result depicted in Fig. 2.3. The result shows that $M, w \models \langle R \rangle [R] (P \vee Q)$, as indicated by the tag `[Yes]` that LoTREC has placed in w right after the input formula.

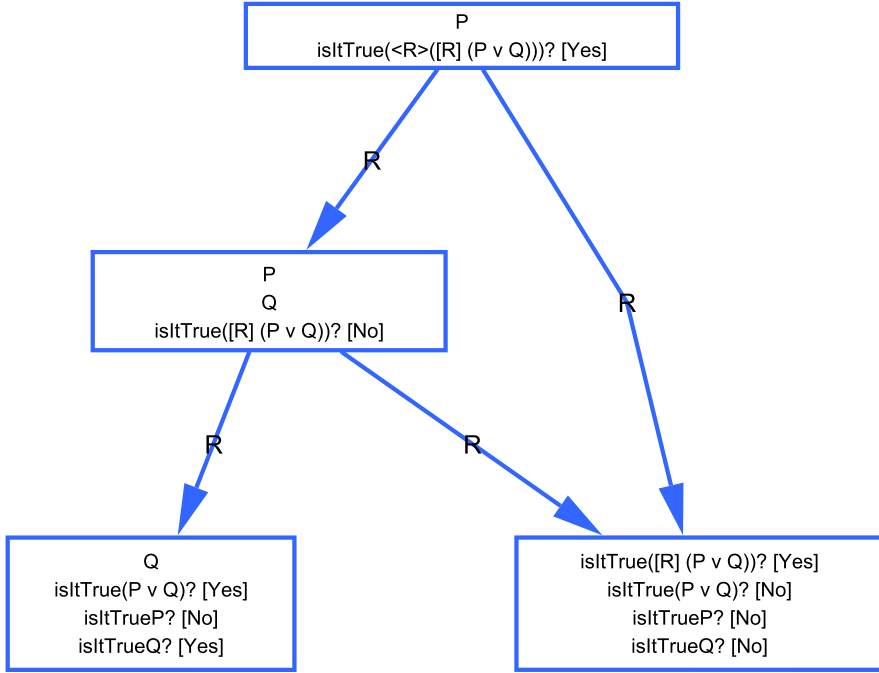


Fig. 2.3 The output of the model checking procedure

You may save the model: use “Save selected premodel” in the “Preamodels” menu. You may reload the model and check another formula by changing the `BuildExampleGraph` rule appropriately.

Exercise 17 Let M be the graph of Fig. 1.4 in Sect. 1.1. Let w_0 be the upper possible world of that graph (where the light is on and both switches are down). Check whether

$$M, w_0 \Vdash [\text{Toggle}_{S_1}] \text{LightOn}$$

Now let w_1 be the right possible world of that graph (where the light is off, the first switch is up and the second switch is down). Check whether

$$M, w_1 \Vdash [\text{Toggle}_{S_1}] \text{LightOn}$$

2.5 Modal Logics and Reasoning Problems

Various questions may be asked about formulas and their truth values in worlds of Kripke models. For instance, does the truth value of a formula in a specific world of a model change when it is considered in another world of the same model? Given

two worlds w_1 and w_2 , is there a formula distinguishing them or not? In the latter case w_1 and w_2 are *bisimilar*.⁹

Can we find a formula that is true in every world of a given model? Given a formula A , does there exist a model M and a world w of M such that $M, w \models A$? Does the answer to this question change when a specific class of models is considered? Are we able to characterise the formulas which are valid in every model, or in every model of a certain kind?

In this section, we reformulate some of these questions in form of a series of *reasoning problems*. We have already seen the first reasoning problem in the preceding Sect. 2.4.

Model Checking Given a finite model M (cf. Definition 3 in Sect. 1.6) and a world w in M , we want to check whether a formula is true in w or not. For instance, given a graph modelling a coffee machine, we would like to know whether the formula $[\text{selectCoffee}]\text{Coffee}$ is true in the “Your coin has been inserted” worlds. This reasoning problem is called *model checking*. It has the following input and output.

- Input: a model M , a world w of M , a formula A ;
- Output: “Yes” if $M, w \models A$; “No” otherwise.

Model checking is widely used in software verification [BK08]. There, the input represents the abstraction of the piece of software and a formula that expresses a property the software should verify, and the aim is to know whether the software verifies that property or not.

Validity Checking Some formulas such as $P \vee \neg P$ are clearly true in every world of every Kripke model. Such formulas are called *valid*. In contrast, the atomic formula P is not valid: it is easy to come up with a Kripke model M and a world w of M such that P is false at w . Take e.g. a model M that is made up of a single possible world w , an empty accessibility relation and an empty valuation: clearly, $M, w \not\models P$.

It is quite easy to see that $P \vee \neg P$, $[I]A \leftrightarrow \neg \langle I \rangle \neg P$, and $\langle I \rangle A \leftrightarrow \neg [I] \neg P$ are valid. Checking the validity of any formula is the following problem.

- Input: a formula A ;
- Output: “Yes” if $M, w \models A$ for every possible world w of every Kripke model M ; “No” otherwise.

⁹More generally, given two models $M_1 = \langle W_1, R_1, V_1 \rangle$ and $M_2 = \langle W_2, R_2, V_2 \rangle$, a *bisimulation* between M_1 and M_2 is a relation $Z \subseteq W_1 \times W_2$ such that

- if $\langle w_1, w_2 \rangle \in Z$ then for every atomic formula P , $w_1 \in V(P)$ iff $w_2 \in V_2(P)$;
- if $\langle w_1, w_2 \rangle \in Z$ then for every edge label $I \in \mathcal{I}$,
 - if $\langle w_1, u_1 \rangle \in R_1(I)$ then there is $u_2 \in W_2$ such that $\langle w_2, u_2 \rangle \in R_2(I)$;
 - if $\langle w_2, u_2 \rangle \in R_2(I)$ then there is $u_1 \in W_1$ such that $\langle w_1, u_1 \rangle \in R_1(I)$.

Two pointed models M_1, w_1 and M_2, w_2 are *bisimilar* if there is a bisimulation Z between M_1 and M_2 such that $\langle w_1, w_2 \rangle \in Z$.

This is the *validity problem in the class of all Kripke models*. It can also be posed w.r.t. a subset of the class of all Kripke models.

Logic of a Class of Frames, Logical Consequence We say that a formula A is valid in a class of frames $\mathbf{C}$ if $\langle W, R, V \rangle, w \Vdash A$ for every frame $\langle W, R \rangle \in \mathbf{C}$, every valuation V over $\langle W, R \rangle$ and every w in W .

The *logic of a class of frames* $\mathbf{C}$ is the set of formulas A such that A is valid in $\mathbf{C}$. To say that A is valid in $\mathbf{C}$ is therefore the same as saying that A belongs to the logic of $\mathbf{C}$.

Given two formulas A and B and a class of models $\mathbf{C}$, we say that B is a logical consequence of A in the class of models $\mathbf{C}$, noted $A \models_{\mathbf{C}} B$, if and only if for every model $M \in \mathbf{C}$ and every possible world w of M , if $M, w \Vdash A$ then $M, w \Vdash B$.¹⁰

Exercise 18 Prove that $A \models_{\mathbf{C}} B$ if and only if $A \rightarrow B$ is valid in $\mathbf{C}$.

Satisfiability Checking The formula P is clearly satisfiable: take e.g. a single possible world w , an empty accessibility relation R , and a valuation function V such that $V(P) = \{w\}$; then $\langle \{w\}, R, V \rangle \Vdash P$. In contrast, the formula $\neg P \wedge P$ is clearly *unsatisfiable*.

More generally, the *satisfiability checking problem* is that we want to check whether for a given formula there exists or not a Kripke model M and a world w in M such that the formula is true at w . Formally it has the following description.

- Input: a formula A ;
- Output: “Yes” if there is a model M and a world w of M such that $M, w \Vdash A$; “No” otherwise.

One often only wants to check for existence of a model of a particular kind, satisfying some constraints on the accessibility relation or on the valuation function. These constraints restrict the search space to special classes of models: we are interested in validity in the class of models $\mathbf{C}$ under concern. We then talk about the *problem of satisfiability in $\mathbf{C}$* , where $\mathbf{C}$ is some special class of models. That class is part of the input of the satisfiability problem.

Suppose we have a procedure solving the satisfiability problem. If the answer for the input formula $\neg A$ is “No, there is no model M having a world w of M such that $M, w \Vdash \neg A$ ” then the validity problem for the formula A is also solved: the answer is: “Yes, A is valid.” Conversely, when the answer is “Yes, there is a model M and a world w such that $M, w \Vdash \neg A$ ” then the answer for the validity problem is “No, A is not valid.” In other words, a procedure for the satisfiability problem

¹⁰This is sometimes called the local consequence relation. The global consequence relation is defined as follows: B is a global logical consequence of A in the class of models $\mathbf{C}$ if and only if for every model $M \in \mathbf{C}$, if $M, w \Vdash A$ for every possible world w of M then $M, w \Vdash B$ for every possible world w of M . We do not consider global logical consequence in this book.

Both consequence relations can be generalised to possibly infinite sets of formulas S on the left side of the symbol of logical consequence $\models_{\mathbf{C}}$. The logic of $\mathbf{C}$ is *compact* if $S \models_{\mathbf{C}} B$ implies that there is a finite subset S' of S such that $S' \models_{\mathbf{C}} B$. All the logics of Chaps. 3, 4 and 5 are compact, while the logics **LTL** and **PDL** of Chap. 7 are not.

can be easily turned into a procedure for the validity problem. The converse is also the case. Interdefinability is moreover the case for the validity problem and for the satisfiability problem in a class of models.

Model Construction The output of the above tasks of model checking, validity checking, and satisfiability checking is just “Yes” or “No.” However, one would often like to get an *explanation* of these answers: why “Yes” and why “No”? In other words, we would like to understand why a given formula is satisfiable or not. This typically happens when students learn a logic or when researchers investigate the properties of an existing logic or design a new logic. In these cases one is interested in the following problem.

- Input: a formula A ;
- Output: a model M and a world w such that $M, w \models A$ if A is satisfiable; ‘*unsatisfiable*’ otherwise.

This is the *model construction* or *model finding problem*. A couple $\langle M, w \rangle$ such that $M, w \models A$ explains why “Yes, A is satisfiable” is the case.

Suppose now that we are checking the validity of a formula A by running a satisfiability checking procedure on $\neg A$, as explained above. When the procedure successfully returns a couple $\langle M, w \rangle$ then this means that $M, w \models \neg A$. According to the truth condition for the negation operator this means that $M, w \not\models A$. We have therefore a *counter-model* for A which explains why A is invalid.

Just as for satisfiability checking and validity checking, model construction may be restricted to models from some particular class of models.

2.6 Modal Logics and Their Axiomatisations

As we have already said, the (modal) logic of a class of frames $\mathbf{C}$ is the set of formulas A such that A is valid in $\mathbf{C}$. This set is in general infinite. The class $\mathbf{C}$ is typically infinite, too.¹¹ This contrasts with propositional logic where the number of rows of the truth table for a given formula A is bound by the length of A .

It is an old idea in logic to try to characterise the logic of a given class $\mathbf{C}$ in a finite way, viz. by means of a finite set of axioms and a finite set of inference rules. This can be traced back to Hilbert’s program for the validities of arithmetic and even to Euclid’s geometry. Once one has come up with such an axiomatisation one may prove its *theorems*: formulas that can be deduced from instances of the axiom schemas via the inference rules. That deduction is called a *proof* of that theorem. However, in order to justify such an axiomatisation two things have to be established in the first place: one has to check that every theorem is valid, and the other way round, one has to check that every valid formula is a theorem. The former is called *soundness* and the latter is called *completeness*.

¹¹More precisely, the class may not be countable. In terms of set theory, that class may not be a set.

Table 2.7 Axiomatisation of modal logic **K**

CPL	an appropriate set of axioms for classical propositional logic
M	$\Box(A \wedge B) \rightarrow (\Box A \wedge \Box B)$
C	$(\Box A \wedge \Box B) \rightarrow \Box(A \wedge B)$
N	$\Box \neg \bot$
Dual	$\Diamond A \leftrightarrow \neg \Box \neg A$
MP	from A and $A \rightarrow B$ infer B
RE	from $A \leftrightarrow B$ infer $\Box A \leftrightarrow \Box B$

Table 2.7 contains a complete axiomatisation of the modal logic **K**: the logic of all Kripke frames. The name **K** is in honour of Saul Kripke. Axiom **M** is called the axiom schema of monotony; **C** is the axiom schema for conjunction; **N** is the axiom schema of necessity; **MP** is the rule of modus ponens; **RE** is the rule for equivalence. The term *axiom schema* (instead of axiom *tout court*) highlights that the symbols A and B occurring in them have to be understood as *schematic variables*: each of these variables may be uniformly replaced by any formula, where ‘uniformly’ means that it has to be the same formula for every occurrence of the same schematic variable.¹² The result is called an *instance* of that axiom.¹³

Let us put that axiomatics to work: we are going to prove that every instance of the so-called axiom **K**: $(\Box A \wedge \Box(A \rightarrow B)) \rightarrow \Box B$ is a theorem of modal logic **K**.

1. $(\Box A \wedge \Box(A \rightarrow B)) \rightarrow \Box(A \wedge (A \rightarrow B))$ (axiom **C**)
2. $(A \wedge (A \rightarrow B)) \leftrightarrow (A \wedge B)$ (**CPL**)
3. $\Box(A \wedge (A \rightarrow B)) \leftrightarrow \Box(A \wedge B)$ (from (2) by rule **RE**)
4. $\Box(A \wedge B) \rightarrow (\Box A \wedge \Box B)$ (axiom **M**)
5. $(\Box A \wedge \Box(A \rightarrow B)) \rightarrow \Box B$ (from (1), (3), (4) by a **CPL** rule)

When we say that the last line follows “from (1), (3), (4) by a **CPL** rule” we want to say that we apply a derived inference rule of classical propositional logic. That rule takes the form “from $X_1 \rightarrow X_2$, $X_2 \leftrightarrow X_3$, and $X_3 \rightarrow (X_4 \wedge X_5)$ infer $X_1 \rightarrow X_5$.” It is applicable because formula (1) has the form $X_1 \rightarrow X_2$, formula (3) has the form $X_2 \leftrightarrow X_3$, and formula (4) has the form $X_3 \rightarrow (X_4 \wedge X_5)$. We note that in the light of our distinction between axiom and axiom schema (where the symbols A and B occurring in a schema are understood as schematic variables), the formula $(\Box A \wedge \Box(A \rightarrow B)) \rightarrow \Box B$ might perhaps better be called a theorem schema: any of its instances is also provable.

¹²We abuse our notation a bit here because we use the symbols A and B both in order to designate formulas and schematic variables. There should however never be a confusion because schematic variables only occur in axiom schemas, and such schemas will always be clearly identified as such.

¹³It is also possible to formulate axiomatisations in terms of axioms instead of axiom schemas. Such axioms have atomic formulas P and Q instead of schematic variables A and B . One then has to add the rule of uniform substitution which allows us to replace atomic formulas by other formulas.

Table 2.8 Some basic modal axioms

Axiom name	Axiom schema	Constraint
T	$\Box A \rightarrow A$	reflexivity
D	$\Box A \rightarrow \Diamond A$	seriality
Alt₁	$\Diamond A \rightarrow \Box A$	determinism
B	$A \rightarrow \Box \Diamond A$	symmetry
4	$\Box A \rightarrow \Box \Box A$	transitivity
5	$\Diamond A \rightarrow \Box \Diamond A$	euclideanity
.2 (or G)	$\Diamond \Box A \rightarrow \Box \Diamond A$	confluence
.3	$\Diamond A \wedge \Diamond B \rightarrow \Diamond(A \wedge B) \vee \Diamond(A \wedge \Diamond B) \vee \Diamond(B \wedge \Diamond A)$	right linearity

The logic of the class of reflexive frames is called **KT**. An axiomatisation of **KT** can be obtained from that of **K** by adding the axiom schema **T**: $\Box A \rightarrow A$ to it.

Let us show that $P \rightarrow \Diamond P$ is a theorem of modal logic **KT**.

1. $\Box \neg A \rightarrow \neg A$ (axiom **T**)
2. $A \rightarrow \neg \Box \neg A$ (from (1) by a **CPL** rule)
3. $\neg \Box \neg A \leftrightarrow \Diamond A$ (axiom **Dual**)
4. $A \rightarrow \Diamond A$ (from (2) and (3) by a **CPL** rule)

Observe that the first line is not exactly axiom **T**, but rather an instance of it.

Actually the schema $A \rightarrow \Diamond A$ may replace axiom **T** in the axiomatisation of **KT**.

Exercise 19 Prove that $\Box P \rightarrow \Diamond P$ is a theorem of **KT**.

We are now going to present several standard modal logics. We focus on classes of frames with a single accessibility relation, and therefore on a monomodal language, i.e., a language where the set of edge labels is a singleton. We have already seen that the logic of the class of all Kripke models is called **K** and that the logic of reflexive frames is called **KT**. In Table 2.8 below we list the axioms corresponding to the semantic constraints of Table 1.1 in Sect. 1.9.

The names of the modal logic of some subset of the constraints of Table 1.1 are well-established names in the literature. They are obtained by adjoining the respective list of axiom schemas to the letter **K**. For example, **KD** is the logic of serial frames, **KB** is the logic of symmetric frames, and so on. Sometimes **KD** is just called **D**, **KT** is just called **T** and **KB** is just called **B**. One may also add more than a single axiom to **K**: the axiom of reflexivity being **T** and the axiom of symmetry being **B**, the logic of the class of reflexive and symmetric Kripke models is called **KTB**.

Observe that every reflexivity relation is also serial; therefore the logics **KT** and **KDT** are the same. We also have **KT5** = **KDT5**, etc. We moreover have **KT45** = **KTb5**. There are some logics with historic names which differ from our nomenclature and which are often used: the traditional name of **KT4** is **S4**, that of **KT45** is **S5**, that of **KT4.2** is **S4.2**, that of **KT4.3** is **S4.3**.

Up to now we have only considered logics of a single relation. As to multiple relations, the logic of the class of all Kripke models having n edge labels is designated by $\mathbf{K}_n$. So the monomodal $\mathbf{K}$ is the same as $\mathbf{K}_1$, and $\mathbf{K}_2$ is the logic of two relations which do not satisfy any constraint. That class of these models is also written $\mathbf{K} \otimes \mathbf{K}$. Such a notation allows us to denominate other logics, such as that of the class of frames with two relations one of which is reflexive: it is written $\mathbf{K} \otimes \mathbf{KT}$. We may also have constraints involving two relations. A simple example is the inclusion of one relation in another. The logic of such frames is $\mathbf{K}_2 \oplus \mathbf{Inclusion} = (\mathbf{K} \otimes \mathbf{K}) \oplus \mathbf{Inclusion}$. Instead of introducing these classes here we will define them on the fly.

Our language being enumerable, we may use a finite axiomatisation in order to automatically enumerate all the theorems. However, this does not allow us to *decide* whether a given formula A is a theorem or not: if A is a theorem then the enumeration will reach it some day; but if it is not then the enumeration procedure will run forever. It is however possible to obtain a decision procedure for $\mathbf{C}$ validity when the logic of $\mathbf{C}$ moreover has the *finite model property*, abbreviated fmp. The latter says that for every formula A , if A is satisfiable in $\mathbf{C}$ then there is a finite $M \in \mathbf{C}$ and a world w in M such that $M, w \models A$. Indeed, if the logic has the fmp then it basically suffices to enumerate in parallel the theorems of the axiomatisation on the one hand and the finite models of $\mathbf{C}$ on the other:¹⁴ if A is a theorem then the former enumeration will reach it, and if A is invalid then the latter will reach a *countermodel*. (Remember that such decision procedure only works correctly if soundness and completeness have been proved beforehand.)

Note that one may as well try to directly come up with a proof of A by fiddling around with the axiom schemas and inference rules. Actually many of the proofs that can be found in textbooks are of that kind. They require quite some creativity, and this is an enterprise that is closer to craftsmanship than to an automated procedure. We have seen in some of the above proofs that one has to guess the ‘right’ instances of the axiom schemas to start with. This is typical for *Hilbert-style axiomatisations*. These are typically axiomatisations with many axioms and few rules: modus ponens plus one or two inference rules per modal operator. In contrast, the tableaux method—as well as sequent calculi and natural deduction calculi—can be viewed as having a single axiom and many rules.

Axiomatisations such as the above for logics $\mathbf{K}$ and $\mathbf{KT}$ provide a mechanical way to enumerate theorems and seems to provide a way for designing algorithms for automated reasoning. However, no algorithm for satisfiability checking or validity checking is based on a Hilbert axiomatisation. Indeed, even if the above decision procedure works in theory, it is however very inefficient. As we will see, tableaux provide for a more efficient procedure.

¹⁴Actually we should be able to enumerate the finite models of $\mathbf{C}$. It is possible for all classes referred in this book. Nevertheless it is not always the case, for instance for this tricky example: the class of all models where the number of worlds n is such that the n th Turing machine does not halt.

2.7 A Note on Computational Complexity

The problem of satisfiability of formulas of first-order logic is undecidable. This contrasts with modal logics: as M. Vardi once formulated it, modal logic satisfiability problems are “surprisingly often decidable” [Var96].

In this section, we briefly recap the main results about the complexity of the satisfiability problem in modal logics.

The aim of computational complexity theory is to characterise the resources that are needed to solve a decision problem. A decision procedure solving the satisfiability problem needs two kinds of resources: time and memory.

The set of decision problems that can be solved by a deterministic algorithm working in time polynomial in the input is called P.

The set of decision problems that can be solved by a nondeterministic algorithm working in time polynomial in the input is called NP. Problems in the set NP can typically be solved by (nondeterministically) guessing a solution of polynomial size and then checking in polynomial time that the solution is correct. For instance, the satisfiability problem of classical propositional logic is in NP: for a given formula A , after guessing a truth-value for each atomic proposition occurring in A , we may check in polynomial time that the resulting truth-value assignment makes A true. It was proved by Stephen Cook that the satisfiability problem of classical propositional logic is among those problems of NP that are most difficult to solve: any other reasoning problem in NP can be transformed into a satisfiability problem of classical propositional logic, and so in polynomial time [Coo71]. For that reason we say that the satisfiability problem of classical propositional logic is NP-complete.

It has to be noted that while almost all researchers believe that the set of problems solvable in polynomial time is strictly included in NP, this has actually not been proven yet and remains a conjecture.

There are problems that cannot be solved in nondeterministic polynomial time and that are therefore beyond the set NP. Here are two of them:

- PSPACE is the set of decision problems that can be solved by an algorithm that only uses a polynomial amount of memory;
- EXPTIME is the set of decision problems that can be solved by an algorithm that only uses an exponential amount of time.

In the same way as for NP one can define PSPACE completeness and EXPTIME completeness. We have $P \subseteq NP \subseteq PSPACE \subseteq EXPTIME$. We have already said that it is an open problem whether the inclusion $P \subseteq NP$ is strict; the same is the case for each of the three other inclusions.

Figure 2.4 classifies the most important modal logics according to the complexity of their satisfiability problem.

2.8 Summary

In this chapter we have finished setting the stage for this book: in order to talk about graphs we have defined a general formal language having parametrised modal

NP-complete	S5, KD45, S4.3
PSPACE-complete	K, KD, KB, KT, K4, S4, LTL
EXPTIME-complete	PDL

Fig. 2.4 Complexity of the satisfiability problem of different modal logics. (The logics **LTL** and **PDL** are defined in Chap. 7)

operators. We have then shown how to define a particular formal language on our tableaux platform **LoTREC** by listing the logical connectives together with their arity and the way they are displayed. We have then given a truth condition for each of these connectives: together, they allow one to decide whether a formula is true or false in a given world of a given model. We have finally presented four important reasoning problems related to the evaluation of formulas: model checking, validity checking, satisfiability checking, and model construction. However, we have not said how these problems can be solved. This is what we are going to do from the next chapter on.

Let us already note that for the problems of validity, satisfiability, and model building the search space is clearly very large: we look for an arbitrary model M and an arbitrary world in that model where the formula is true. It is *a priori* not clear how the size of such a model and the size of the search space of all relevant models could be bounded. This contrasts with e.g. the satisfiability problem for a formula A of propositional logic where it suffices to check the relevant propositional valuations, i.e., the set of subsets of the set of atomic formulas of A : there are only exponentially many such valuations, depending on the number of atomic formulas occurring in A , i.e., depending on the cardinality of the set $\mathcal{P} \cap \text{SF}(A)$.

We have seen that a solution to the model construction problem also allows us to check satisfiability and validity, while the converse is not the case. We therefore focus in this book on a method for constructing models. We will start by a method searching for a model in the class of all Kripke models and then examine methods for various classes of models. These will be explained throughout the next chapters. We will get back to the model checking problem in Chap. 6.

Kripke's Worlds

An Introduction to Modal Logics via Tableaux

Gasquet, O.; Herzig, A.; Said, B.; Schwarzentruher, F.

2014, XV, 198 p. 73 illus., Softcover

ISBN: 978-3-7643-8503-3

A product of Birkhäuser Basel