Propositional Logic

2

In propositional logic, as the name suggests, propositions are connected by logical operators. The statement "*the street is wet*" is a proposition, as is "*it is raining*". These two propositions can be connected to form the new proposition

if it is raining the street is wet.

Written more formally

it is raining $\Rightarrow$ *the street is wet.*

This notation has the advantage that the elemental propositions appear again in unaltered form. So that we can work with propositional logic precisely, we will begin with a definition of the set of all propositional logic formulas.

2.1 Syntax

Definition 2.1 Let $Op = \{\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow, (,)\}$ be the set of logical operators and Σ a set of symbols. The sets Op, Σ and $\{t, f\}$ are pairwise disjoint. Σ is called the *signature* and its elements are the *proposition variables*. The set of propositional logic formulas is now recursively defined:

- t and f are (atomic) formulas.
- All proposition variables, that is all elements from Σ, are (atomic) formulas.
- If A and B are formulas, then $\neg A$, (A), $A \wedge B$, $A \vee B$, $A \Rightarrow B$, $A \Leftrightarrow B$ are also formulas.

This elegant recursive definition of the set of all formulas allows us to generate infinitely many formulas. For example, given $\Sigma = \{A, B, C\}$,

$$A \wedge B, \quad A \wedge B \wedge C, \quad A \wedge A \wedge A, \quad C \wedge B \vee A, \quad (\neg A \wedge B) \Rightarrow (\neg C \vee A)$$

are formulas. $(((A)) \vee B)$ is also a syntactically correct formula.

W. Ertel, *Introduction to Artificial Intelligence*,
Undergraduate Topics in Computer Science,
DOI 10.1007/978-0-85729-299-5_2, © Springer-Verlag London Limited 2011

Definition 2.2 We read the symbols and operators in the following way:

t:	"true"	
f:	"false"	
$\neg A$:	"not A"	(negation)
$A \wedge B$:	"A and B"	(conjunction)
$A \vee B$:	"A or B"	(disjunction)
$A \Rightarrow B$:	"if A then B"	(implication (also called *material implication*))
$A \Leftrightarrow B$:	"A if and only if B"	(equivalence)

The formulas defined in this way are so far purely syntactic constructions without meaning. We are still missing the semantics.

2.2 Semantics

In propositional logic there are two truth values: t for "true" and f for "false". We begin with an example and ask ourselves whether the formula $A \wedge B$ is true. The answer is: it depends on whether the variables A and B are true. For example, if A stands for "*It is raining today*" and B for "*It is cold today*" and these are both true, then $A \wedge B$ is true. If, however, B represents "*It is hot today*" (and this is false), then $A \wedge B$ is false.

We must obviously assign truth values that reflect the state of the world to proposition variables. Therefore we define

Definition 2.3 A mapping $I : \Sigma \to \{w, f\}$, which assigns a truth value to every proposition variable, is called an *interpretation*.

Because every proposition variable can take on two truth values, every propositional logic formula with n different variables has 2^n different interpretations. We define the truth values for the basic operations by showing all possible interpretations in a *truth table* (see Table 2.1 on page 17).

The empty formula is true for all interpretations. In order to determine the truth value for complex formulas, we must also define the order of operations for logical operators. If expressions are parenthesized, the term in the parentheses is evaluated

Table 2.1 Definition of the logical operators by truth table

A	B	(A)	$\neg A$	$A \wedge B$	$A \vee B$	$A \Rightarrow B$	$A \Leftrightarrow B$
t	t	t	f	t	t	t	t
t	f	t	f	f	t	f	f
f	t	f	t	f	t	t	f
f	f	f	t	f	f	t	t

first. For unparenthesized formulas, the priorities are ordered as follows, beginning with the strongest binding: $\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow$.

To clearly differentiate between the equivalence of formulas and syntactic equivalence, we define

Definition 2.4 Two formulas F and G are called semantically equivalent if they take on the same truth value for all interpretations. We write $F \equiv G$.

Semantic equivalence serves above all to be able to use the meta-language, that is, natural language, to talk about the object language, namely logic. The statement "$A \equiv B$" conveys that the two formulas A and B are semantically equivalent. The statement "$A \Leftrightarrow B$" on the other hand is a syntactic object of the formal language of propositional logic.

According to how many interpretations in which a formula is true, we can divide formulas into the following classes:

Definition 2.5 A formula is called
- *Satisfiable* if it is true for at least one interpretation.
- *Logically valid* or simply *valid* if it is true for all interpretations. True formulas are also called *tautologies*.
- *Unsatisfiable* if it is not true for any interpretation.

Every interpretation that satisfies a formula is called a *model* of the formula.

Clearly the negation of every generally valid formula is unsatisfiable. The negation of a satisfiable, but not generally valid formula F is satisfiable.

We are now able to create truth tables for complex formulas to ascertain their truth values. We put this into action immediately using equivalences of formulas which are important in practice.

Theorem 2.1 *The operations $\wedge, \vee$ are commutative and associative, and the following equivalences are generally valid*:

$$
\begin{array}{rcll}
\neg A \vee B & \Leftrightarrow & A \Rightarrow B & (\textit{implication}) \\
A \Rightarrow B & \Leftrightarrow & \neg B \Rightarrow \neg A & (\textit{contraposition}) \\
(A \Rightarrow B) \wedge (B \Rightarrow A) & \Leftrightarrow & (A \Leftrightarrow B) & (\textit{equivalence}) \\
\neg(A \wedge B) & \Leftrightarrow & \neg A \vee \neg B & (\textit{De Morgan's law}) \\
\neg(A \vee B) & \Leftrightarrow & \neg A \wedge \neg B & \\
A \vee (B \wedge C) & \Leftrightarrow & (A \vee B) \wedge (A \vee C) & (\textit{distributive law}) \\
A \wedge (B \vee C) & \Leftrightarrow & (A \wedge B) \vee (A \wedge C) & \\
A \vee \neg A & \Leftrightarrow & w & (\textit{tautology}) \\
A \wedge \neg A & \Leftrightarrow & f & (\textit{contradiction}) \\
A \vee f & \Leftrightarrow & A & \\
A \vee w & \Leftrightarrow & w & \\
A \wedge f & \Leftrightarrow & f & \\
A \wedge w & \Leftrightarrow & A &
\end{array}
$$

Proof To show the first equivalence, we calculate the truth table for $\neg A \vee B$ and $A \Rightarrow B$ and see that the truth values for both formulas are the same for all interpretations. The formulas are therefore equivalent, and thus all the values of the last column are "t"s.

A	B	$\neg A$	$\neg A \vee B$	$A \Rightarrow B$	$(\neg A \vee B) \Leftrightarrow (A \Rightarrow B)$
t	t	f	t	t	t
t	f	f	f	f	t
f	t	t	t	t	t
f	f	t	t	t	t

The proofs for the other equivalences are similar and are recommended as exercises for the reader (Exercise 2.2 on page 29). □

2.3 Proof Systems

In AI we are interested in taking existing knowledge and from that deriving new knowledge or answering questions. In propositional logic this means showing that a *knowledge base KB*—that is, a (possibly extensive) propositional logic formula—a formula Q[1] follows. Thus, we first define the term "entailment".

[1] Here Q stands for query.

Definition 2.6 A formula *KB entails* a formula Q (or Q *follows* from *KB*) if every model of *KB* is also a model of Q. We write $KB \models Q$.

In other words, in every interpretation in which *KB* is true, Q is also true. More succinctly, whenever *KB* is true, Q is also true. Because, for the concept of entailment, interpretations of variables are brought in, we are dealing with a semantic concept.

Every formula that is not valid chooses so to speak a subset of the set of all interpretations as its model. Tautologies such as $A \vee \neg A$, for example, do not restrict the number of satisfying interpretations because their proposition is empty. The empty formula is therefore true in all interpretations. For every tautology T then $\emptyset \models T$. Intuitively this means that tautologies are always true, without restriction of the interpretations by a formula. For short we write $\models T$. Now we show an important connection between the semantic concept of entailment and syntactic implication.

Theorem 2.2 (Deduktionstheorem)

$$A \models B \textit{ if and only if } \models A \Rightarrow B.$$

Proof Observe the truth table for implication:

A	B	$A \Rightarrow B$
t	t	t
t	f	f
f	t	t
f	f	t

An arbitrary implication $A \Rightarrow B$ is clearly always true except with the interpretation $A \mapsto t$, $B \mapsto f$. Assume that $A \models B$ holds. This means that for every interpretation that makes A true, B is also true. The critical second row of the truth table does not even apply in that case. Therefore $A \Rightarrow B$ is true, which means that $A \Rightarrow B$ is a tautology. Thus one direction of the statement has been shown.

Now assume that $A \Rightarrow B$ holds. Thus the critical second row of the truth table is also locked out. Every model of A is then also a model of B. Then $A \models B$ holds. □

If we wish to show that *KB* entails Q, we can also demonstrate by means of the truth table method that $KB \Rightarrow Q$ is a tautology. Thus we have our first *proof system* for propositional logic, which is easily automated. The disadvantage of this method is the very long computation time in the worst case. Specifically, in the worst case with n proposition variables, for all 2^n interpretations of the variables the formula $KB \Rightarrow Q$ must be evaluated. The computation time grows therefore exponentially

with the number of variables. Therefore this process is unusable for large variable counts, at least in the worst case.

If a formula KB entails a formula Q, then by the deduction theorem $KB \Rightarrow Q$ is a tautology. Therefore the negation $\neg(KB \Rightarrow Q)$ is unsatisfiable. We have

$$\neg(KB \Rightarrow Q) \equiv \neg(\neg KB \vee Q) \equiv KB \wedge \neg Q.$$

Therefore $KB \wedge \neg Q$ is also satisfiable. We formulate this simple, but important consequence of the deduction theorem as a theorem.

Theorem 2.3 (Proof by contradiction) *$KB \models Q$ if and only if $KB \wedge \neg Q$ is unsatisfiable.*

To show that the query Q follows from the knowledge base KB, we can also add the negated query $\neg Q$ to the knowledge base and derive a contradiction. Because of the equivalence $A \wedge \neg A \Leftrightarrow f$ from Theorem 2.1 on page 18 we know that a contradiction is unsatisfiable. Therefore, Q has been proved. This procedure, which is frequently used in mathematics, is also used in various automatic proof calculi such as the resolution calculus and in the processing of PROLOG programs.

One way of avoiding having to test all interpretations with the truth table method is the syntactic manipulation of the formulas KB and Q by application of inference rules with the goal of greatly simplifying them, such that in the end we can instantly see that $KB \models Q$. We call this syntactic process *derivation* and write $KB \vdash Q$. Such syntactic proof systems are called *calculi*. To ensure that a calculus does not generate errors, we define two fundamental properties of calculi.

Definition 2.7 A calculus is called *sound* if every derived proposition follows semantically. That is, if it holds for formulas KB and Q that

$$\text{if} \quad KB \vdash Q \quad \text{then} \quad KB \models Q.$$

A calculus is called *complete* if all semantic consequences can be derived. That is, for formulas KB and Q the following it holds:

$$\text{if} \quad KB \models Q \quad \text{then} \quad KB \vdash Q.$$

The soundness of a calculus ensures that all derived formulas are in fact semantic consequences of the knowledge base. The calculus does not produce any "false consequences". The completeness of a calculus, on the other hand, ensures that the calculus does not overlook anything. A complete calculus always finds a proof if the formula to be proved follows from the knowledge base. If a calculus is sound

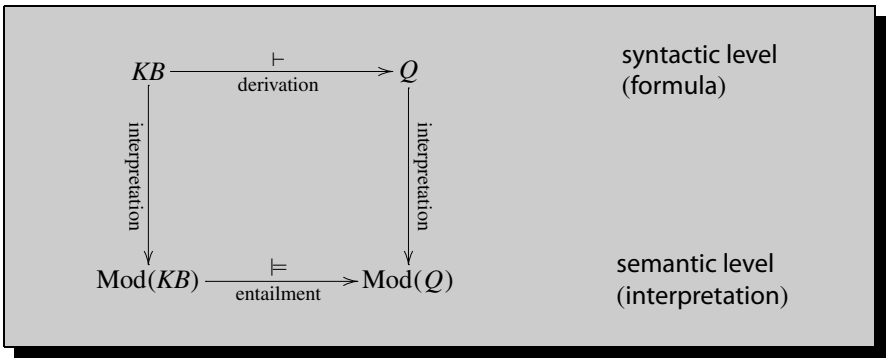

Fig. 2.1 Syntactic derivation and semantic entailment. Mod(X) represents the set of models of a formula X

and complete, then syntactic derivation and semantic entailment are two equivalent relations (see Fig. 2.1).

To keep automatic proof systems as simple as possible, these are usually made to operate on formulas in conjunctive normal form.

Definition 2.8 A formula is in *conjunctive normal form* (*CNF*) if and only if it consists of a *conjunction*

$$K_1 \wedge K_2 \wedge \cdots \wedge K_m$$

of clauses. A clause K_i consists of a *disjunction*

$$(L_{i1} \vee L_{i2} \vee \cdots \vee L_{in_i})$$

of literals. Finally, a *literal* is a variable (positive literal) or a negated variable (negative literal).

The formula $(A \vee B \vee \neg C) \wedge (A \vee B) \wedge (\neg B \vee \neg C)$ is in conjunctive normal form. The conjunctive normal form does not place a restriction on the set of formulas because:

Theorem 2.4 *Every propositional logic formula can be transformed into an equivalent conjunctive normal form.*

Example 2.1 We put $A \vee B \Rightarrow C \wedge D$ into conjunctive normal form by using the equivalences from Theorem 2.1 on page 18:

$$
\begin{aligned}
& A \vee B \Rightarrow C \wedge D \\
& \quad \equiv \neg(A \vee B) \vee (C \wedge D) && \text{(implication)} \\
& \quad \equiv (\neg A \wedge \neg B) \vee (C \wedge D) && \text{(de Morgan)} \\
& \quad \equiv (\neg A \vee (C \wedge D)) \wedge (\neg B \vee (C \wedge D)) && \text{(distributive law)} \\
& \quad \equiv ((\neg A \vee C) \wedge (\neg A \vee D)) \wedge ((\neg B \vee C) \wedge (\neg B \vee D)) && \text{(distributive law)} \\
& \quad \equiv (\neg A \vee C) \wedge (\neg A \vee D) \wedge (\neg B \vee C) \wedge (\neg B \vee D) && \text{(associative law)}
\end{aligned}
$$

We are now only missing a calculus for syntactic proof of propositional logic formulas. We start with the *modus ponens*, a simple, intuitive rule of inference, which, from the validity of A and $A \Rightarrow B$, allows the derivation of B. We write this formally as

$$\frac{A, \quad A \Rightarrow B}{B}.$$

This notation means that we can derive the formula(s) below the line from the comma-separated formulas above the line. Modus ponens as a rule by itself, while sound, is not complete. If we add additional rules we can create a complete calculus, which, however, we do not wish to consider here. Instead we will investigate the *resolution rule*

$$\frac{A \vee B, \quad \neg B \vee C}{A \vee C} \tag{2.1}$$

as an alternative. The derived clause is called *resolvent*. Through a simple transformation we obtain the equivalent form

$$\frac{A \vee B, \quad B \Rightarrow C}{A \vee C}.$$

If we set A to f, we see that the resolution rule is a generalization of the modus ponens. The resolution rule is equally usable if C is missing or if A and C are missing. In the latter case the empty clause can be derived from the contradiction $B \wedge \neg B$ (Exercise 2.7 on page 30).

2.4 Resolution

We now generalize the resolution rule again by allowing clauses with an arbitrary number of literals. With the literals $A_1, \ldots, A_m, B, C_1, \ldots, C_n$ the *general resolution rule* reads

$$\frac{(A_1 \vee \cdots \vee A_m \vee B), \quad (\neg B \vee C_1 \vee \cdots \vee C_n)}{(A_1 \vee \cdots \vee A_m \vee C_1 \vee \cdots \vee C_n)}. \tag{2.2}$$

We call the literals B and $\neg B$ complementary. The resolution rule deletes a pair of complementary literals from the two clauses and combines the rest of the literals into a new clause.

To prove that from a knowledge base KB, a query Q follows, we carry out a proof by contradiction. Following Theorem 2.3 on page 20 we must show that a contradiction can be derived from $KB \wedge \neg Q$. In formulas in conjunctive normal form, a contradiction appears in the form of two clauses (A) and $(\neg A)$, which lead to the empty clause as their resolvent. The following theorem ensures us that this process really works as desired.

For the calculus to be complete, we need a small addition, as shown by the following example. Let the formula $(A \vee A)$ be given as our knowledge base. To show by the resolution rule that from there we can derive $(A \wedge A)$, we must show that the empty clause can be derived from $(A \vee A) \wedge (\neg A \vee \neg A)$. With the resolution rule alone, this is impossible. With *factorization*, which allows deletion of copies of literals from clauses, this problem is eliminated. In the example, a double application of factorization leads to $(A) \wedge (\neg A)$, and a resolution step to the empty clause.

Theorem 2.5 *The resolution calculus for the proof of unsatisfiability of formulas in conjunctive normal form is sound and complete.*

Because it is the job of the resolution calculus to derive a contradiction from $KB \wedge \neg Q$, it is very important that the knowledge base KB is *consistent*:

Definition 2.9 A formula KB is called *consistent* if it is impossible to derive from it a contradiction, that is, a formula of the form $\phi \wedge \neg\phi$.

Otherwise anything can be derived from KB (see Exercise 2.8 on page 30). This is true not only of resolution, but also for many other calculi.

Of the calculi for automated deduction, resolution plays an exceptional role. Thus we wish to work a bit more closely with it. In contrast to other calculi, resolution has only two inference rules, and it works with formulas in conjunctive normal form. This makes its implementation simpler. A further advantage compared to many calculi lies in its reduction in the number of possibilities for the application of inference rules in every step of the proof, whereby the search space is reduced and computation time decreased.

As an example, we start with a simple logic puzzle that allows the important steps of a resolution proof to be shown.

Example 2.2 Logic puzzle number 7, entitled *A charming English family*, from the German book [Ber89] reads (translated to English):

> Despite studying English for seven long years with brilliant success, I must admit that when I hear English people speaking English I'm totally perplexed. Recently, moved by noble feelings, I picked up three hitchhikers, a father, mother, and daughter, who I quickly realized were English and only spoke English. At each of the sentences that follow I wavered between two possible interpretations. They told me the following (the second possible meaning is in parentheses): The father: "We are going to Spain (we are from Newcastle)." The mother: "We are not going to Spain and are from Newcastle (we stopped in Paris and are not going to Spain)." The daughter: "We are not from Newcastle (we stopped in Paris)." What about this charming English family?

To solve this kind of problem we proceed in three steps: formalization, transformation into normal form, and proof. In many cases formalization is by far the most difficult step because it is easy to make mistakes or forget small details. (Thus practical exercise is very important. See Exercises 2.9–2.11.)

Here we use the variables S for "*We are going to Spain*", N for "*We are from Newcastle*", and P for "*We stopped in Paris*" and obtain as a formalization of the three propositions of father, mother, and daughter

$$(S \vee N) \wedge [(\neg S \wedge N) \vee (P \wedge \neg S)] \wedge (\neg N \vee P).$$

Factoring out $\neg S$ in the middle sub-formula brings the formula into CNF in one step. Numbering the clauses with subscripted indices yields

$$KB \equiv (S \vee N)_1 \wedge (\neg S)_2 \wedge (P \vee N)_3 \wedge (\neg N \vee P)_4.$$

Now we begin the resolution proof, at first still without a query Q. An expression of the form "Res(m, n): $\langle clause \rangle_k$" means that $\langle$clause$\rangle$ is obtained by resolution of clause m with clause n and is numbered k.

$$\begin{aligned} \text{Res}(1,2): &\quad (N)_5 \\ \text{Res}(3,4): &\quad (P)_6 \\ \text{Res}(1,4): &\quad (S \vee P)_7 \end{aligned}$$

We could have derived clause P also from Res$(4, 5)$ or Res$(2, 7)$. Every further resolution step would lead to the derivation of clauses that are already available. Because it does not allow the derivation of the empty clause, it has therefore been shown that the knowledge base is non-contradictory. So far we have derived N and P. To show that $\neg S$ holds, we add the clause $(S)_8$ to the set of clauses as a negated query. With the resolution step

$$\text{Res}(2,8): \quad ()_9$$

the proof is complete. Thus $\neg S \wedge N \wedge P$ holds. The "charming English family" evidently comes from Newcastle, stopped in Paris, but is not going to Spain.

Example 2.3 Logic puzzle number 28 from [Ber89], entitled *The High Jump*, reads

Three girls practice high jump for their physical education final exam. The bar is set to 1.20 meters. “I bet”, says the first girl to the second, “that I will make it over if, and only if, you don’t”. If the second girl said the same to the third, who in turn said the same to the first, would it be possible for all three to win their bets?

We show through proof by resolution that not all three can win their bets.
Formalization:

The first girl’s jump succeeds: A,	First girl’s bet: $(A \Leftrightarrow \neg B)$,
the second girl’s jump succeeds: B,	second girl’s bet: $(B \Leftrightarrow \neg C)$,
the third girl’s jump succeeds: C.	third girl’s bet: $(C \Leftrightarrow \neg A)$.

Claim: the three cannot all win their bets:

$$Q \equiv \neg((A \Leftrightarrow \neg B) \wedge (B \Leftrightarrow \neg C) \wedge (C \Leftrightarrow \neg A))$$

It must now be shown by resolution that $\neg Q$ is unsatisfiable.
Transformation into CNF: First girl’s bet:

$$(A \Leftrightarrow \neg B) \equiv (A \Rightarrow \neg B) \wedge (\neg B \Rightarrow A) \equiv (\neg A \vee \neg B) \wedge (A \vee B)$$

The bets of the other two girls undergo analogous transformations, and we obtain the negated claim

$$\neg Q \equiv (\neg A \vee \neg B)_1 \wedge (A \vee B)_2 \wedge (\neg B \vee \neg C)_3 \wedge (B \vee C)_4 \wedge (\neg C \vee \neg A)_5 \\ \wedge (C \vee A)_6.$$

From there we derive the empty clause using resolution:

$$\begin{aligned} &\text{Res}(1,6): && (C \vee \neg B)_7 \\ &\text{Res}(4,7): && (C)_8 \\ &\text{Res}(2,5): && (B \vee \neg C)_9 \\ &\text{Res}(3,9): && (\neg C)_{10} \\ &\text{Res}(8,10): && () \end{aligned}$$

Thus the claim has been proved.

2.5 Horn Clauses

A clause in conjunctive normal form contains positive and negative literals and can be represented in the form

$$(\neg A_1 \vee \cdots \vee \neg A_m \vee B_1 \vee \cdots \vee B_n)$$

with the variables $A_1, \ldots, A_m$ and $B_1, \ldots, B_n$. This clause can be transformed in two simple steps into the equivalent form

$$A_1 \wedge \cdots \wedge A_m \Rightarrow B_1 \vee \cdots \vee B_n.$$

This implication contains the premise, a conjunction of variables and the conclusion, a disjunction of variables. For example, "*If the weather is nice and there is snow on the ground, I will go skiing or I will work.*" is a proposition of this form. The receiver of this message knows for certain that the sender is not going swimming. A significantly clearer statement would be "*If the weather is nice and there is snow on the ground, I will go skiing.*". The receiver now knows definitively. Thus we call clauses with at most one positive literal definite clauses. These clauses have the advantage that they only allow one conclusion and are thus distinctly simpler to interpret. Many relations can be described by clauses of this type. We therefore define

Definition 2.10 Clauses with at most one positive literal of the form

$$(\neg A_1 \vee \cdots \vee \neg A_m \vee B) \quad \text{or} \quad (\neg A_1 \vee \cdots \vee \neg A_m) \quad \text{or} \quad B$$

or (equivalently)

$$A_1 \wedge \cdots \wedge A_m \Rightarrow B \quad \text{or} \quad A_1 \wedge \cdots \wedge A_m \Rightarrow f \quad \text{or} \quad B.$$

are named *Horn clauses* (after their inventor). A clause with a single positive literal is a *fact*. In clauses with negative and one positive literal, the positive literal is called the *head*.

To better understand the representation of Horn clauses, the reader may derive them from the definitions of the equivalences we have currently been using (Exercise 2.12 on page 30).

Horn clauses are easier to handle not only in daily life, but also in formal reasoning, as we can see in the following example. Let the knowledge base consist of the following clauses (the "$\wedge$" binding the clauses is left out here and in the text that follows):

$$(nice\_weather)_1$$
$$(snowfall)_2$$
$$(snowfall \Rightarrow snow)_3$$
$$(nice\_weather \wedge snow \Rightarrow skiing)_4$$

If we now want to know whether *skiing* holds, this can easily be derived. A slightly generalized modus ponens suffices here as an inference rule:

$$\frac{A_1 \wedge \cdots \wedge A_m, \quad A_1 \wedge \cdots \wedge A_m \Rightarrow B}{B}.$$

The proof of "*skiing*" has the following form (MP($i_1, \ldots, i_k$) represents application of the modus ponens on clauses i_1 to i_k:

$$\begin{aligned} &\text{MP}(2,3): && (snow)_5 \\ &\text{MP}(1,5,4): && (skiing)_6. \end{aligned}$$

With modus ponens we obtain a complete calculus for formulas that consist of propositional logic Horn clauses. In the case of large knowledge bases, however, modus ponens can derive many unnecessary formulas if one begins with the wrong clauses. Therefore, in many cases it is better to use a calculus that starts with the query and works backward until the facts are reached. Such systems are designated *backward chaining*, in contrast to *forward chaining* systems, which start with facts and finally derive the query, as in the above example with the modus ponens.

For backward chaining of Horn clauses, *SLD resolution* is used. SLD stands for "*Selection rule driven linear resolution for definite clauses*". In the above example, augmented by the negated query $(skiing \Rightarrow f)$

$$\begin{aligned} &(nice\_weather)_1 \\ &(snowfall)_2 \\ &(snowfall \Rightarrow snow)_3 \\ &(nice\_weather \wedge snow \Rightarrow skiing)_4 \\ &(skiing \Rightarrow f)_5 \end{aligned}$$

we carry out SLD resolution beginning with the resolution steps that follow from this clause

$$\begin{aligned} &\text{Res}(5,4): && (nice\_weather \wedge snow \Rightarrow f)_6 \\ &\text{Res}(6,1): && (snow \Rightarrow f)_7 \\ &\text{Res}(7,3): && (snowfall \Rightarrow f)_8 \\ &\text{Res}(8,2): && () \end{aligned}$$

and derive a contradiction with the empty clause. Here we can easily see "*linear resolution*", which means that further processing is always done on the currently derived clause. This leads to a great reduction of the search space. Furthermore, the literals of the current clause are always processed in a fixed order (for example, from right to left) ("*Selection rule driven*"). The literals of the current clause are called

subgoal. The literals of the negated query are the *goals*. The inference rule for one step reads

$$\frac{A_1 \wedge \cdots \wedge A_m \Rightarrow B_1, \quad B_1 \wedge B_2 \wedge \cdots \wedge B_n \Rightarrow f}{A_1 \wedge \cdots \wedge A_m \wedge B_2 \wedge \cdots \wedge B_n \Rightarrow f}.$$

Before application of the inference rule, $B_1, B_2, \ldots, B_n$—the current subgoals—must be proved. After the application, B_1 is replaced by the new subgoal $A_1 \wedge \cdots \wedge A_m$. To show that B_1 is true, we must now show that $A_1 \wedge \cdots \wedge A_m$ are true. This process continues until the list of subgoals of the current clauses (the so-called *goal stack*) is empty. With that, a contradiction has been found. If, for a subgoal $\neg B_i$, there is no clause with the complementary literal B_i as its clause head, the proof terminates and no contradiction can be found. The query is thus unprovable.

SLD resolution plays an important role in practice because programs in the logic programming language PROLOG consist of predicate logic Horn clauses, and their processing is achieved by means of SLD resolution (see Exercise 2.13 on page 30, or Chap. 5).

2.6 Computability and Complexity

The truth table method, as the simplest semantic proof system for propositional logic, represents an algorithm that can determine every model of any formula in finite time. Thus the sets of unsatisfiable, satisfiable, and valid formulas are decidable. The computation time of the truth table method for satisfiability grows in the worst case exponentially with the number n of variables because the truth table has 2^n rows. An optimization, the method of *semantic trees*, avoids looking at variables that do not occur in clauses, and thus saves computation time in many cases, but in the worst case it is likewise exponential.

In resolution, in the worst case the number of derived clauses grows exponentially with the number of initial clauses. To decide between the two processes, we can therefore use the rule of thumb that in the case of many clauses with few variables, the truth table method is preferable, and in the case of few clauses with many variables, resolution will probably finish faster.

The question remains: can proof in propositional logic go faster? Are there better algorithms? The answer: probably not. After all, S. Cook, the founder of complexity theory, has shown that the 3-SAT problem is NP-complete. 3-SAT is the set of all CNF formulas whose clauses have exactly three literals. Thus it is clear that there is probably (modulo the P/NP problem) no polynomial algorithm for 3-SAT, and thus probably not a general one either. For Horn clauses, however, there is an algorithm in which the computation time for testing satisfiability grows only linearly as the number of literals in the formula increases.

2.7 Applications and Limitations

Theorem provers for propositional logic are part of the developer's everyday toolset in digital technology. For example, the verification of digital circuits and the generation of test patterns for testing of microprocessors in fabrication are some of these tasks. Special proof systems that work with binary decision diagrams (BDD) () are also employed as a data structure for processing propositional logic formulas.

In AI, propositional logic is employed in simple applications. For example, simple expert systems can certainly work with propositional logic. However, the variables must all be discrete, with only a few values, and there may not be any cross-relations between variables. Complex logical connections can be expressed much more elegantly using predicate logic.

Probabilistic logic is a very interesting and current combination of propositional logic and probabilistic computation that allows modeling of uncertain knowledge. It is handled thoroughly in Chap. 7. Fuzzy logic, which allows infinitely many truth values, is also discussed in that chapter.

2.8 Exercises

Exercise 2.1 Give a Backus–Naur form grammar for the syntax of propositional logic.

Exercise 2.2 Show that the following formulas are tautologies:
(a) $\neg(A \wedge B) \Leftrightarrow \neg A \vee \neg B$
(b) $A \Rightarrow B \Leftrightarrow \neg B \Rightarrow \neg A$
(c) $((A \Rightarrow B) \wedge (B \Rightarrow A)) \Leftrightarrow (A \Leftrightarrow B)$
(d) $(A \vee B) \wedge (\neg B \vee C) \Rightarrow (A \vee C)$

Exercise 2.3 Transform the following formulas into conjunctive normal form:
(a) $A \Leftrightarrow B$
(b) $A \wedge B \Leftrightarrow A \vee B$
(c) $A \wedge (A \Rightarrow B) \Rightarrow B$

Exercise 2.4 Check the following statements for satisfiability or validity.
(a) (play_lottery $\wedge$ six_right) $\Rightarrow$ winner
(b) (play_lottery $\wedge$ six_right $\wedge$ (six_right $\Rightarrow$ win)) $\Rightarrow$ win
(c) $\neg$($\neg$gas_in_tank $\wedge$ (gas_in_tank $\vee$ $\neg$car_starts) $\Rightarrow$ $\neg$car_starts)

Exercise 2.5 Using the programming language of your choice, program a theorem prover for propositional logic using the truth table method for formulas in conjunctive normal form. To avoid a costly syntax check of the formulas, you may represent clauses as lists or sets of literals, and the formulas as lists or sets of clauses. The program should indicate whether the formula is unsatisfiable, satisfiable, or true, and output the number of different interpretations and models.

Exercise 2.6
(a) Show that modus ponens is a valid inference rule by showing that $A \wedge (A \Rightarrow B) \models B$.
(b) Show that the resolution rule (2.1) is a valid inference rule.

❄ **Exercise 2.7** Show by application of the resolution rule that, in conjunctive normal form, the empty clause is equivalent to the false statement.

❄ **Exercise 2.8** Show that, with resolution, one can "derive" any arbitrary clause from a knowledge base that contains a contradiction.

Exercise 2.9 Formalize the following logical functions with the logical operators and show that your formula is valid. Present the result in CNF.
(a) The XOR operation (exclusive or) between two variables.
(b) The statement *at least two of the three variables* A, B, C *are true*.

❄ **Exercise 2.10** Solve the following case with the help of a resolution proof: "If the criminal had an accomplice, then he came in a car. The criminal had no accomplice and did not have the key, or he had the key and an accomplice. The criminal had the key. Did the criminal come in a car or not?"

Exercise 2.11 Show by resolution that the formula from
(a) Exercise 2.2(d) is a tautology.
(b) Exercise 2.4(c) is unsatisfiable.

Exercise 2.12 Prove the following equivalences, which are important for working with Horn clauses:
(a) $(\neg A_1 \vee \cdots \vee \neg A_m \vee B) \equiv A_1 \wedge \cdots \wedge A_m \Rightarrow B$
(b) $(\neg A_1 \vee \cdots \vee \neg A_m) \equiv A_1 \wedge \cdots \wedge A_m \Rightarrow f$
(c) $A \equiv w \Rightarrow A$

Exercise 2.13 Show by SLD resolution that the following Horn clause set is unsatisfiable.

$$(A)_1 \quad (D)_4 \quad (A \wedge D \Rightarrow G)_7$$
$$(B)_2 \quad (E)_5 \quad (C \wedge F \wedge E \Rightarrow H)_8$$
$$(C)_3 \quad (A \wedge B \wedge C \Rightarrow F)_6 \quad (H \Rightarrow f)_9$$

➳ **Exercise 2.14** In Sect. 2.6 it says: "Thus it is clear that there is probably (modulo the P/NP problem) no polynomial algorithm for 3-SAT, and thus probably not a general one either." Justify the "probably" in this sentence.

http://www.springer.com/978-0-85729-298-8

Introduction to Artificial Intelligence
Ertel, W.
2011, XII, 316 p., Softcover
ISBN: 978-0-85729-298-8