Limitations of Logic

4

4.1 The Search Space Problem

As already mentioned in several places, in the search for a proof there are almost always many (depending on the calculus, potentially infinitely many) possibilities for the application of inference rules at every step. The result is the aforementioned explosive growth of the search space (Fig. 4.1 on page 66). In the worst case, all of these possibilities must be tried in order to find the proof, which is usually not possible in a reasonable amount of time.

If we compare automated provers or inference systems with mathematicians or human experts who have experience in special domains, we make interesting observations. For one thing, experienced mathematicians can prove theorems which are far out of reach for automated provers. On the other hand, automated provers perform tens of thousands of inferences per second. A human in contrast performs maybe one inference per second. Although human experts are much slower on the object level (that is, in carrying out inferences), they apparently solve difficult problems much faster.

There are several reasons for this. We humans use intuitive calculi that work on a higher level and often carry out many of the simple inferences of an automated prover in one step. Furthermore, we use lemmas, that is, derived true formulas that we already know and therefore do not need to re-prove them each time. Meanwhile there are also machine provers that work with such methods. But even they cannot yet compete with human experts.

A further, much more important advantage of us humans is intuition, without which we could not solve any difficult problems [PS09]. The attempt to formalize intuition causes problems. Experience in applied AI projects shows that in complex domains such as medicine (see Sect. 7.3) or mathematics, most experts are unable to formulate this intuitive meta-knowledge verbally, much less to formalize it. Therefore we cannot program this knowledge or integrate it into calculi in the form of *heuristics*. Heuristics are methods that in many cases can greatly simplify or shorten the way to the goal, but in some cases (usually rarely) can greatly lengthen

W. Ertel, *Introduction to Artificial Intelligence*, Undergraduate Topics in Computer Science, DOI 10.1007/978-3-319-58487-4_4

Fig. 4.1 Possible consequences of the explosion of a search space

the way to the goal. Heuristic search is important not only to logic, but generally to problem solving in AI and will therefore be thoroughly handled in Chap. 6.

An interesting approach, which has been pursued since about 1990, is the application of machine learning techniques to the learning of heuristics for directing the search of inference systems, which we will briefly sketch now. A resolution prover has, during the search for a proof, hundreds or more possibilities for resolution steps at each step, but only a few lead to the goal. It would be ideal if the prover could ask an oracle which two clauses it should use in the next step to quickly find the proof. There are attempts to build such proof-directing modules, which evaluate the various alternatives for the next step and then choose the alternative with the best rating. In the case of resolution, the rating of the available clauses could be computed by a function that calculates a value based on the number of positive literals, the complexity of the terms, etc., for every pair of resolvable clauses.

How can this function be implemented? Because this knowledge is "intuitive", the programmer is not familiar with it. Instead, one tries to copy nature and uses machine learning algorithms to learn from successful proofs [ESS89, SE90].

The attributes of all clause pairs participating in successful resolution steps are stored as positive, and the attributes of all unsuccessful resolutions are stored as negative. Then, using this training data and a machine learning system, a program is generated which can rate clause pairs heuristically (see Sect. 9.5).

A different, more successful approach to improving mathematical reasoning is followed with interactive systems that operate under the control of the user. Here one could name computer algebra programs such as Mathematica, Maple, or Maxima, which can automatically carry out difficult symbolic mathematical manipulations. The search for the proof, however, is left fully to the human. The aforementioned interactive prover Isabelle [NPW02] provides distinctly more support during the proof search. There are at present several projects, such as Omega [SB04] and MKM,[1] for the development of systems for supporting mathematicians during proofs.

In summary, one can say that, because of the search space problem, automated provers today can only prove relatively simple theorems in special domains with few axioms.

4.2 Decidability and Incompleteness

First-order predicate logic provides a powerful tool for the representation of knowledge and reasoning. We know that there are correct and complete calculi and theorem provers. When proving a theorem, that is, a true statement, such a prover is very helpful because, due to completeness, one knows after finite time that the statement really is true. What if the statement is not true? The completeness theorem (Theorem 3.3 on page 50) does not answer this question.[2] Specifically, there is no process that can prove or refute any formula from PL1 in finite time, for it holds that

Theorem 4.1 *The set of valid formulas in first-order predicate logic is semi-decidable.*

This theorem implies that there are programs (theorem provers) which, given a true (valid) formula as input, determine its truth in finite time. If the formula is not valid, however, it may happen that the prover never halts. (The reader may grapple with this question in Exercise 4.1 on page 73.) Propositional logic is decidable because the truth table method provides all models of a formula in finite time. Evidently predicate logic with quantifiers and nested function symbols is a language somewhat too powerful to be decidable.

[1] www.mathweb.org/mathweb/demo.html.

[2] Just this case is especially important in practice, because if I already know that a statement is true, I no longer need a prover.

On the other hand, predicate logic is not powerful enough for many purposes. One often wishes to make statements about sets of predicates or functions. This does not work in PL1 because it only knows quantifiers for variables, but not for predicates or functions.

Kurt Gödel showed, shortly after his completeness theorem for PL1, that completeness is lost if we extend PL1 even minimally to construct a higher-order logic. A first-order logic can only quantify over variables. A second-order logic can also quantify over formulas of the first order, and a third-order logic can quantify over formulas of the second order. Even adding only the induction axiom for the natural numbers makes the logic incomplete. The statement "*If a predicate* $p(n)$ *holds for* n*, then* $p(n + 1)$ *also holds*", or

$$\forall p \; p(n) \;\Rightarrow\; p(n+1)$$

is a second-order proposition because it quantifies over a predicate. Gödel proved the following theorem:

Theorem 4.2 (Gödel's incompleteness theorem) *Every axiom system for the natural numbers with addition and multiplication* (*arithmetic*) *is incomplete. That is, there are true statements in arithmetic that are not provable.*

Gödel's proof works with what is called Gödelization, in which every arithmetic formula is encoded as a number. It obtains a unique Gödel number. Gödelization is now used to formulate the proposition

$$F = \text{"I am not provable."}$$

in the language of arithmetic. This formula is true for the following reason. Assume F is false. Then we can prove F and therefore show that F is not provable. This is a contradiction. Thus F is true and therefore not provable.

The deeper background of this theorem is that mathematical theories (axiom systems) and, more generally, languages become incomplete if the language becomes too powerful. A similar example is set theory. This language is so powerful that one can formulate paradoxes with it. These are statements that contradict themselves, such as the statement we already know from Example 3.7 on page 54 about the barbers who all shave those who do not shave themselves (see Exercise 4.2 on page 73). The dilemma consists therein that with languages which are powerful enough to describe mathematics and interesting applications, we are smuggling contradictions and incompletenesses through the back door. This does not mean, however, that higher-order logics are wholly unsuited for formal methods. There are certainly formal systems as well as provers for higher-order logics.

4.3 The Flying Penguin

With a simple example we will demonstrate a fundamental problem of logic and possible solution approaches. Given the statements

1. Tweety is a penguin
2. Penguins are birds
3. Birds can fly

Formalized in PL1, the knowledge base *KB* results:

$$\begin{aligned} &penguin(tweety)\\ &penguin(x) \Rightarrow bird(x)\\ &bird(x) \Rightarrow fly(x) \end{aligned}$$

From there (for example with resolution) *fly*(*tweety*) can be derived (Fig. 4.2 on page 70).[3] Evidently the formalization of the flight attributes of penguins is insufficient. We try the additional statement *Penguins cannot fly*, that is

$$penguin(x) \Rightarrow \neg fly(x)$$

From there $\neg$*fly*(*tweety*) can be derived. But *fly*(*tweety*) is still true. The knowledge base is therefore inconsistent. Here we notice an important characteristic of logic, namely monotony. Although we explicitly state that penguins cannot fly, the opposite can still be derived.

Definition 4.1 A logic is called monotonic if, for an arbitrary knowledge base *KB* and an arbitrary formula ϕ, the set of formulas derivable from *KB* is a subset of the formulas derivable from $KB \cup \phi$.

If a set of formulas is extended, then, after the extension, all previously derivable statements can still be proved, and additional statements can potentially also be proved. The set of provable statements thus grows monotonically when the set of formulas is extended. For our example this means that the extension of the knowledge base will never lead to our goal. We thus modify *KB* by replacing the obviously false statement "(*all*) *birds can fly*" with the more exact statement "(*all*) *birds except penguins can fly*" and obtain as KB_2 the following clauses:

$$\begin{aligned} &penguin(tweety)\\ &penguin(x) \Rightarrow bird(x)\\ &bird(x) \wedge \neg penguin(x) \Rightarrow fly(x)\\ &penguin(x) \Rightarrow \neg fly(x) \end{aligned}$$

[3]The formal execution of this and the following simple proof may be left to the reader (Exercise 4.3 on page 73).

Fig. 4.2 The flying penguin Tweety

Now the world is apparently in order again. We can derive $\neg fly(tweety)$, but not $fly(tweety)$, because for that we would need $\neg\ penguin(x)$, which, however, is not derivable. As long as there are only penguins in this world, peace reigns. Every normal bird, however, immediately causes problems. We wish to add the raven Abraxas (from the German book "The Little Witch") and obtain

$$\begin{aligned}
&raven(abraxas)\\
&raven(x) \Rightarrow bird(x)\\
&penguin(tweety)\\
&penguin(x) \Rightarrow bird(x)\\
&bird(x) \wedge \neg penguin(x) \Rightarrow fly(x)\\
&penguin(x) \Rightarrow \neg fly(x)
\end{aligned}$$

We cannot say anything about the flight attributes of Abraxas because we forgot to formulate that ravens are not penguins. Thus we extend KB_3 to KB_4:

$$\begin{aligned}
&raven(abraxas)\\
&raven(x) \Rightarrow bird(x)\\
&raven(x) \Rightarrow \neg pinguin(x)\\
&penguin(tweety)\\
&penguin(x) \Rightarrow bird(x)\\
&bird(x) \wedge \neg penguin(x) \Rightarrow fly(x)\\
&penguin(x) \Rightarrow \neg fly(x)
\end{aligned}$$

The fact that ravens are not penguins, which is self-evident to humans, must be explicitly added here. For the construction of a knowledge base with all 9,800 or so

types of birds worldwide, it must therefore be specified for every type of bird (except for penguins) that it is not a member of penguins. We must proceed analogously for all other exceptions such as the ostrich.

For every object in the knowledge base, in addition to its attributes, all of the attributes it does not have must be listed.

To solve this problem, various forms of non-monotonic logic have been developed, which allow knowledge (formulas) to be removed from the knowledge base. Under the name *default logic*, logics have been developed which allow objects to be assigned attributes which are valid as long as no other rules are available. In the Tweety example, the rule *birds can fly* would be such a *default rule*. Despite great effort, these logics have at present, due to semantic and practical problems, not succeeded.

Monotony can be especially inconvenient in complex planning problems in which the world can change. If for example a blue house is painted red, then afterwards it is red. A knowledge base such as

$$\begin{aligned} &color(house, blue) \\ &paint(house, red) \\ &paint(x, y) \Rightarrow color(x, y) \end{aligned}$$

leads to the conclusion that, after painting, the house is red and blue. The problem that comes up here in planning is known as the *frame problem*. A solution for this is the situation calculus presented in Sect. 5.6.

An interesting approach for modeling problems such as the Tweety example is probability theory. The statement "all birds can fly" is false. A statement something like "almost all birds can fly" is correct. This statement becomes more exact if we give a probability for "birds can fly". This leads to *probabilistic logic*, which today represents an important sub-area of AI and an important tool for modeling uncertainty (see Chap. 7).

4.4 Modeling Uncertainty

Two-valued logic can and should only model circumstances in which there is true, false, and no other truth values. For many tasks in everyday reasoning, two-valued logic is therefore not expressive enough. The rule

$$bird(x) \Rightarrow fly(x)$$

is true for almost all birds, but for some it is false. As was already mentioned, working with probabilities allows exact formulation of uncertainty. The statement "*99% of all birds can fly*" can be formalized by the expression

$$P(bird(x) \Rightarrow fly(x)) = 0.99.$$

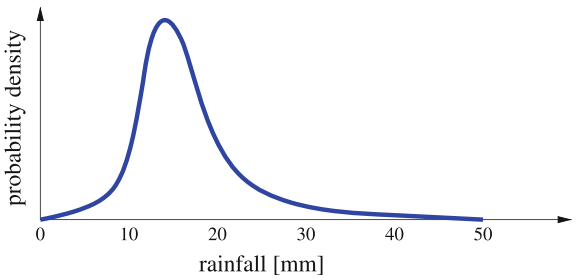

Fig. 4.3 Probability density of the continuous variable rainfall

In Chap. 7 we will see that here it is better to work with conditional probabilities such as

$$P(fly|bird) = 0.99.$$

With the help of *Bayesian networks*, complex applications with many variables can also be modeled.

A different model is needed for the statement "*The weather is nice*". Here it often makes no sense to speak in terms of true and false. The variable *weather_is_nice* should not be modeled as binary, rather continuously with values, for example, in the interval [0, 1]. *weather_is_nice* = 0.7 then means "*The weather is fairly nice*". *Fuzzy logic* was developed for this type of continuous (fuzzy) variable.

Probability theory also offers the possibility of making statements about the probability of continuous variables. A statement in the weather report such as "*There is a high probability that there will be some rain*" could for example be exactly formulated as a probability density of the form

$$P(rainfall = X) = Y$$

and represented graphically something like in Fig. 4.3.

This very general and even visualizable representation of both types of uncertainty we have discussed, together with inductive statistics and the theory of Bayesian networks, makes it possible, in principle, to answer arbitrary probabilistic queries.

Probability theory as well as fuzzy logic are not directly comparable to predicate logic because they do not allow variables or quantifiers. They can thus be seen as extensions of propositional logic as shown in the following table.

Formalism	Number of truth values	Probabilities expressible
Propositional logic	2	–
Fuzzy logic	∞	–
Discrete probabilistic logic	*n*	yes
Continuous probabilistic logic	∞	*yes*

4.5 Exercises

❄ **Exercise 4.1**

(a) With the following (false) argument, one could claim that PL1 is decidable: *We take a complete proof calculus for PL1. With it we can find a proof for any true formula in finite time. For every other formula ϕ I proceed as follows: I apply the calculus to $\neg\phi$ and show that $\neg\phi$ is true. Thus ϕ is false. Thus I can prove or refute every formula in PL1.* Find the mistake in the argument and change it so it becomes correct.

(b) Construct a decision process for the set of true and unsatisfiable formulas in PL1.

Exercise 4.2

(a) Given the statement "*There is a barber who shaves every person who does not shave himself.*" Consider whether this barber shaves himself.

(b) Let $M = \{x \mid x \notin x\}$. Describe this set and consider whether M contains itself.

Exercise 4.3 Use an automated theorem prover (for example E [Sch02]) and apply it to all five different axiomatizations of the Tweety example from Sect. 4.3. Validate the example's statements.