original knowledge base is consistent. (After all, if it is not consistent, then the fact that the query follows from it is vacuous.) The set-of-support strategy has the additional advantage of generating goal-directed proof trees that are often easy for humans to understand.

**Input resolution:** In this strategy, every resolution combines one of the input sentences (from the KB or the query) with some other sentence. The proof in Figure 9.11 on page 348 uses only input resolutions and has the characteristic shape of a single "spine" with single sentences combining onto the spine. Clearly, the space of proof trees of this shape is smaller than the space of all proof graphs. In Horn knowledge bases, Modus Ponens is a kind of input resolution strategy, because it combines an implication from the original KB with some other sentences. Thus, it is no surprise that input resolution is complete for knowledge bases that are in Horn form, but incomplete in the general case. The linear resolution strategy is a slight generalization that allows P and Q to be resolved together either if P is in the original KB or if P is an ancestor of Q in the proof tree. Linear resolution is complete.

**Subsumption:** The subsumption method eliminates all sentences that are subsumed by (that is, more specific than) an existing sentence in the KB. For example, if P(x) is in the KB, then there is no sense in adding P(A) and even less sense in adding P(A) ∨ Q(B). Subsumption helps keep the KB small and thus helps keep the search space small.

**Practical uses of resolution theorem provers**

Theorem provers can be applied to the problems involved in the synthesis and verification of both hardware and software. Thus, research is carried out in the fields of hardware design, programming languages, and software engineering—not just in AI.

In the case of hardware, the axioms describe the interactions between signals and circuit elements. (See Section 8.4.2 on page 309 for an example.) Logical reasoners designed specially for verification have been able to verify entire CPUs, including their timing properties (Sivakumar and Bickford, 1990). The AURA theorem prover has been applied to design circuits that are more compact than any previous design (Wojciechowski and Wojcik, 1983).

In the case of software, reasoning about programs is quite similar to reasoning about actions, as in Chapter 7: axioms describe the preconditions and effects of each statement. The formal synthesis of algorithms was one of the first uses of theorem provers, as outlined by Cordell Green (1969a), who built on earlier ideas by Herbert Simon (1963). The idea is to constructively prove a theorem to the effect that "there exists a program p satisfying a certain specification." Although fully automated deductive synthesis, as it is called, has not yet become feasible for general-purpose programming, hand-guided deductive synthesis has been successful in designing several novel and sophisticated algorithms. Synthesis of special-purpose programs, such as scientific computing code, is also an active area of research.

Similar techniques are now being applied to software verification by systems such as the SPIN model checker (Holzmann, 1997). For example, the Remote Agent spacecraft control program was verified before and after flight (Havelund et al., 2000). The RSA public key encryption algorithm and the Boyer–Moore string-matching algorithm have been verified this way (Boyer and Moore, 1984).
Section 9.6. Summary 357

9.6 SUMMARY

We have presented an analysis of logical inference in first-order logic and a number of algorithms for doing it.

- A first approach uses inference rules (universal instantiation and existential instantiation) to propositionalize the inference problem. Typically, this approach is slow, unless the domain is small.
- The use of unification to identify appropriate substitutions for variables eliminates the instantiation step in first-order proofs, making the process more efficient in many cases.
- A lifted version of Modus Ponens uses unification to provide a natural and powerful inference rule, generalized Modus Ponens. The forward-chaining and backward-chaining algorithms apply this rule to sets of definite clauses.
- Generalized Modus Ponens is complete for definite clauses, although the entailment problem is semidecidable. For Datalog knowledge bases consisting of function-free definite clauses, entailment is decidable.
- Forward chaining is used in deductive databases, where it can be combined with relational database operations. It is also used in production systems, which perform efficient updates with very large rule sets. Forward chaining is complete for Datalog and runs in polynomial time.
- Backward chaining is used in logic programming systems, which employ sophisticated compiler technology to provide very fast inference. Backward chaining suffers from redundant inferences and infinite loops; these can be alleviated by memoization.
- Prolog, unlike first-order logic, uses a closed world with the unique names assumption and negation as failure. These make Prolog a more practical programming language, but bring it further from pure logic.
- The generalized resolution inference rule provides a complete proof system for first-order logic, using knowledge bases in conjunctive normal form.
- Several strategies exist for reducing the search space of a resolution system without compromising completeness. One of the most important issues is dealing with equality; we showed how demodulation and paramodulation can be used.
- Efficient resolution-based theorem provers have been used to prove interesting mathematical theorems and to verify and synthesize software and hardware.

BIBLIOGRAPHICAL AND HISTORICAL NOTES

Gottlob Frege, who developed full first-order logic in 1879, based his system of inference on a collection of valid schemas plus a single inference rule, Modus Ponens. Whitehead and Russell (1910) expounded the so-called rules of passage (the actual term is from Herbrand (1930)) that are used to move quantifiers to the front of formulas. Skolem constants