# Chapter 1

# Introduction

Logic is a science studying the principles of reasoning and valid inference. Automated deduction is concerned with the mechanization of formal reasoning, following the laws of logic. The roots of the field go back to the end of the last century when Frege developed his *Begriffsschrift*[1], the first comprehensive effort to develop a formal language suitable as a foundation for mathematics. Alas, Russell discovered a paradox which showed that Frege's system was *inconsistent*, that is, the truth of every proposition can be derived in it. Russell then devised his own system based on a *type theory* and he and Whitehead demonstrated in the monumental *Principia Mathematica* how it can serve as a foundation of mathematics. Later, Hilbert developed a simpler alternative, the *predicate calculus*. Gentzen's formulation of the predicate calculus in a system of *natural deduction* provides a major milestone for the field. In natural deduction, the meaning of each logical connective is explained via inference rules, an approach later systematically refined by Martin-Löf. This is the presentation we will follow in these notes.

Gentzen's seminal work also contains an early consistency proof for a formal logical system. As a technical device he introduced the *sequent calculus* and showed that it derives the same theorems as natural deduction. The famous *Hauptsatz*[2] establishes that all proofs in the sequent calculus can be found according to a simple strategy. It is immediately evident that there are many propositions which have no proof according to this strategy, thereby guaranteeing consistency of the system.

Most search strategies employed by automated deduction systems are either directly based on or can be derived from the sequent calculus. We can broadly classify procedures as either working backwards from the proposed theorem toward the axioms, or forward from the axioms toward the theorem. Among the backward searching procedures we find tableaux, connection methods, matrix methods and some forms of resolution. Among the forward searching procedures we find classical resolution and the inverse method. The prominence of

[1] literally translated as *concept notation*

[2] literally just "main theorem", often called the *cut elimination theorem*

resolution among these methods is no accident, since Robinson's seminal paper represented a major leap forward in the state of the art. It is natural to expect that a combination of forward and backward search could improve the efficiency of theorem proving system. Such a combination, however, has been elusive up to now, due to the largely incompatible basic choices in design and implementation of the two kinds of search procedures.

In this course we study both types of procedures. We investigate high-level questions, such as how these procedures relate to the basic sequent calculus. We also consider low-level issues, such as techniques for efficient implementation of the basic inference engine.

There is one further dimension to consider: which *logic* do we reason in? In philosophy, mathematics, and computer science many different logics are of interest. For example, there are classical logic, intuitionistic logic, modal logic, relevance logic, higher-order logic, dynamic logic, temporal logic, linear logic, belief logic, and lax logic (to mention just a few). While each logic requires its own considerations, many techniques are shared. This can be attributed in part to the common root of different logics in natural deduction and the sequent calculus. Another reason is that low-level efficiency improvements are relatively independent of higher-level techniques.

For this course we chose intuitionistic logic for a variety of reasons. First, intuitionistic propositions correspond to logical specifications and proofs to functional programs, which means intuitionistic logic is of central interest in the study of programming languages. Second, intuitionistic logic is more complex than classical logic and exhibits phenomena obscured by special properties which apply only to classical logic. Third, there are relatively straightforward interpretations of classical in intuitionistic logic which permits us to study logical interpretations in connection with theorem proving procedures.

The course is centered around a project, namely the joint design and implementation of a succession of theorem provers for intuitionistic logic. We start with natural deduction, followed by a sequent calculus, and a simple tableau prover. Then we turn toward the inverse method and introduce successive refinements consisting of both high-level and low-level optimizations.[3] The implementation component is important to gain a deeper understanding of the techniques introduced in our abstract study.

The goal of the course is to give students a thorough understanding of the central techniques in automated theorem proving. Furthermore, they should understand the systematic development of these techniques and their correctness proofs, thereby enabling them to transfer methods to different logics or applications. We are less interested here in an appreciation of the pragmatics of highly efficient implementations or performance tuning.

[3]The precise order and extent of the improvements possible in a one-semester graduate course has yet to be determined.