Lecture Notes on Constructive Logic: Overview

15-317: Constructive Logic
Frank Pfenning

Lecture 1
August 25, 2009

1 Introduction

According to Wikipedia, logic is the study of the principles of valid inferences and demonstration. From the breadth of this definition it is immediately clear that logic constitutes an important area in the disciplines of philosophy and mathematics. Logical tools and methods also play an essential role in the design, specification, and verification of computer hardware and software. It is these applications of logic in computer science which will be the focus of this course. In order to gain a proper understanding of logic and its relevance to computer science, we will need to draw heavily on the much older logical traditions in philosophy and mathematics. We will discuss some of the relevant history of logic and pointers to further reading throughout these notes. In this introduction, we give only a brief overview of the goal, contents, and approach of this class.

2 Topics

The course is divided into four parts:

I. Proofs as Evidence for Truth

II. Proofs as Programs

III. Proof Search as Computation

IV. Substructural and Modal Logics

Proofs are central in all parts of the course, and give it its constructive nature. In each part, we will exhibit connections between proofs and forms of computations studied in computer science. These connections will take quite different forms, which shows the richness of logic as a foundational discipline at the nexus between philosophy, mathematics, and computer science.

In Part I we establish the basic vocabulary and systematically study propositions and proofs, mostly from a philosophical perspective. The treatment will be rather formal in order to permit an easy transition into computational applications. We will also discuss some properties of the logical systems we develop and strategies for proof search. We aim at a systematic account for the usual forms of logical expression, providing us with a flexible and thorough foundation for the remainder of the course. We will also highlight the differences between constructive and non-constructive reasoning. Exercises in this section will test basic understanding of logical connectives and how to reason with them.

In Part II we focus on constructive reasoning. This means we consider only proofs that describe algorithms. This turns out to be quite natural in the framework we have established in Part I. In fact, it may be somewhat surprising that many proofs in mathematics today are *not* constructive in this sense. Concretely, we find that for a certain fragment of logic, constructive proofs correspond to functional programs and vice versa. More generally, we can extract functional programs from constructive proofs of their specifications. We often refer to constructive reasoning as *intuitionistic*, while non-constructive reasoning is *classical*. Exercises in this part explore the connections between proofs and programs, and between theorem proving and programming.

In Part III we study a different connection between logic and programs where proofs are the result of computation rather than the starting point as in Part II. This gives rise to the paradigm of *logic programming* where the process of computation is one of systematic proof search. Depending on how we search for proofs, different kinds of algorithms can be described at a very high level of abstraction. Exercises in this part focus on exploiting logic programming to implement various algorithms in concrete languages such as Prolog.

In Part IV we study logics with more general and more refined notions of truth. For example, in temporal logic we are concerned with reasoning about truth relative to time. Another example is the modal logic S_5 where we reason about truth in a collection of worlds, each of which is connected to all other worlds. Proofs in this logic can be given an interpretation as dis-

tributed computation. Similarly, *linear logic* is a substructural logic where truth is ephemeral and may change in the process of deduction. As we will see, this naturally corresponds to imperative programming.

3 Goals

There are several related goals for this course. The first is simply that we would like students to gain a good working knowledge of constructive logic and its relation to computation. This includes the translation of informally specified problems to logical language, the ability to recognize correct proofs and construct them.

The second set of goals concerns the transfer of this knowledge to other kinds of reasoning. We will try to illuminate logic and the underlying philosophical and mathematical principles from various points of view. This is important, since there are many different kinds of logics for reasoning in different domains or about different phenomena[1], but there are relatively few underlying philosophical and mathematical principles. Our second goal is to teach these principles so that students can apply them in different domains where rigorous reasoning is required.

A third set of goals relates to specific, important applications of logic in the practice of computer science. Examples are the design of type systems for programming languages, specification languages, or verification tools for various classes of systems. While we do not aim at teaching the use of particular systems or languages, students should have the basic knowledge to quickly learn them, based on the materials presented in this class.

These learning goals present different challenges for students from different disciplines. Lectures, recitations, exercises, and the study of these notes are all necessary components for reaching them. These notes do not cover all aspects of the material discussed in lecture, but provide a point of reference for definitions, theorems, and motivating examples. Recitations are intended to answer students' questions and practice problem solving skills that are critical for the homework assignments. Exercises are a combination of written homework to be handed in at lecture and theorem proving or programming problems to be submitted electronically using the software written in support of the course. A brief tutorial and manual are available with the on-line course material.

[1] for example: classical, intuitionistic, modal, second-order, temporal, belief, linear, relevance, affirmation, ...