ConstLog: Constructive Logic

Lecture Notes on Introduction to Constructive Logic

Frank Pfenning André Platzer

Carnegie Mellon University || Karlsruhe Institute of Technology

Lecture 1

1 Introduction

Logic is the art of reasoning. Its mode of operation is the study how to draw conclusions from premises (called inference). This makes logic a fundamental part of mathematics, philosophy but also computer science, because specification with logic and verification with logical inferences play a fundamental role in the design of programs and computer hardware. Once one understands how useful and general logic is, it does not take long to realize that there are multiple logics with different purposes.

The most common logic is *classical logic* in the tradition of Gottlob Frege, Bertrand Russel and Alfred North Whitehead, which is based on the principle that every formula has a truth value of either true or false. This makes it exceedingly easy to define and understand logical operators (connectives such as $\wedge$ for and, $\vee$ for or, $\neg$ for not) as functions on these truth-values often given by truth tables. Classical logic is good for many purposes, but it also has its surprises. This lecture discusses some of them. For example, as every formula A in classical logic either has the truth value true or the truth value false, the formula $A \vee \neg A$ is always true (called the law of excluded middle). This is fine on its own but also causes paradoxes when generalized without sufficient care. What can also happen in classical logic is that something exists that satisfies a formula and yet there is no way of telling what that something specifically is.

This course studies *constructive logic*, which, despite its similarities in the logical language, is based on a very different foundation than classical logic. Constructive logic is based on the principle that truth is defined by proof[1] and that (constructive) proofs

[1] Admittedly there are many different ways of introducing both classical logic and constructive logic and either using semantic notions of truth or syntactic notions of proof as the basis. The explanation here refers to the most canonical, simple, and natural introduction of classical logic (which is based on meaning) and of constructive logic (which is based on syntactic proof).

constitute constructions. It is this constructive nature of constructive logic that gives it a very natural fit to computer science and its study of algorithms, because the concepts of constructions, algorithms, and programs ultimately differ but in name.[2]

This course will answer questions such as:

- What is the difference between a proof and a program?[3]
- Do all true formulas have witnesses? This question, of course, only makes sense for formulas that even claim existential-flavored properties to begin with.
- Is there a way to read off a program from a proof?
- How can proofs form the basis of computation? How can formulas?
- What is a type from a logical perspective?
- Is propositional constructive logic decidable? It is based on proof, not truth-tables.

2 Topics

The course is divided into four parts:

I. Proofs as Evidence for Truth || Natural Deduction

II. Proofs as Programs || Functional Programming

III. Propositions as Programs || Proof Search as Computation || Logic Programming

IV. Substructural Logics / Modal Logics / Type Systems etc.

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. While philosophical in spirit, the treatment will be 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. Typical exercises in this section test basic understanding of logical connectives and how to reason with them.

[2] As Alan J. Perlis accurately observed: "Think of all the psychic energy expended in seeking a fundamental distinction between 'algorithm' and 'program'."

[3] Understanding why a proof is like a program does not get you any closer to understanding the Mad Hatter's challenge why a raven is like a writing desk, but can be just as vexing.

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 modern mathematics are *not* constructive in this sense anymore. 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*. Typical exercises in this part may explore the connections between proofs and functional programs, and between theorem proving and programming, which are all ultimately the same in this perspective. We will also discover how $A \vee \neg A$ is a triviality in classical logic but, when it holds for a specific proposition A, a deep insight in constructive logic.

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 an exceedingly high and eloquent level of abstraction. Typical exercises in this part may focus on exploiting proof search as computation via 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 distributed 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. Time permitting, we might also explore how the basis for type systems that constructive logic provides can be generalized and made more uniform.

The core topics of this course are intuitionistic logic, natural deduction, Curry-Howard isomorphism, propositions as types, proofs as programs, formulas as programs, functional programming, logic programming, Heyting arithmetic and primitive recursion, cut elimination, connections between classical and constructive logic, inductive definitions, sequent calculus, and decidable classes. Advanced topics may include type theory, proof search, linear logic, temporal logic, modal logic.

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 mathe-

matical 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[4], but there are relatively few underlying philosophical and mathematical principles. Our second goal is to teach these underlying 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, practicing, 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 applying the material.

4 Russell's Paradox and LEM

The first shock to the working principles of classical logic came from Bertrand Russell's paradox illustrating difficulties when its *law of excluded middle* is indiscriminately applied to anything. Russel's set of all sets that do not contain themselves

$$R = \{x : x \notin x\}$$

causes a foundational paradox when asking whether R itself is in R. If it is then, by the definition of R, it is not supposed to be. If it is not, then, again by definition of R, it should have been. If by law of excluded middle either $R \in R$ or $R \notin R$ has to be the case, then, either way, $R \in R$ iff $R \notin R$, which is paradoxical. Russell's paradox can be resolved within set theory or when considering types, which are a part of the purpose of this course. After all, for the question whether x is contained in R to make sense, the type of R should denote a set of the type that x denotes, which promptly identifies the question whether $R \in R$ as neither true nor false but entirely meaningless.

A constructive proof of a disjunction $A \vee B$ will either be a proof of A or a proof of B and not just a classical logical proof that the disjunction $A \vee B$ has to have the truth value true. This makes it impossible to generally prove the law of excluded middle in $A \vee \neg A$, because one cannot know for arbitrary propositions whether A has a proof or whether $\neg A$ has a proof even if classical logicians can write down the truth table and convince themselves that one of the two disjuncts will have the truth value true either way. When A is Fermat's last theorem, then a constructive proof of $A \vee \neg A$ is a big

[4] for example: classical, intuitionistic, modal, dynamic, second-order, higher-order, temporal, epistemic, linear, relevance, affirmation, ...

achievement that took almost 400 years of hard work while a classical proof of $A \vee \neg A$ is a trivial observation.

5 Intuitionism

We call a logic *constructive* if its proofs describe effective constructions. The emphasis here is on *effective* which is to say that the construction conveyed by a proof can actually be carried out mechanically. For the purpose of computer science, a construction is simply an algorithm, so constructive proofs describe algorithms. At first one might think that all proofs describe constructions of this form, and this was historically true for a long time. At some point in the 19th century this direct link between mathematics and computation has gotten lost. Some mathematicians starting with Brouwer objected to this and started to develop a foundations of mathematics in which *all* proofs denote effective constructions.

In order to understand this distinction better, we start with a theorem that illustrates the distinction, the so-called *Banach-Tarski Paradox*.[5]

Theorem 1. *Given a solid ball in 3-dimensional space, there exists a decomposition of the ball into a finite number of disjoint subsets, which can then be put back together in a different way to yield two identical copies of the original ball. The reassembly process involves only moving pieces around and rotating them, without changing their shape. The reconstruction can work with as few as five pieces.*

This is considered paradoxical, since we obviously cannot carry out such a decomposition in the physical reality. The intermediate pieces are in fact *non-measurable* infinite scatterings of points, motivating the development of measure theory in mathematics to distinguish measurable sets from nonmeasurable sets. The decomposition relies critically on the axiom of choice in set theory, which is highly non-constructive.

This is the kind of theorem (and proof, which we not show here but is sketched in the article) that mathematician L.E.J. Brouwer[6] might have objected to. It is meaningless with respect to our understanding of effective constructions, even if the formalities of its proof are sound. This entails a criticism of Hilbert's program, who posited that at the foundations of mathematics should be a formal system of axioms and inference rules with respect to which we can judge the correctness of mathematical arguments. Brouwer called himself an *intuitionist*, perhaps to contrast himself to Hilbert as a *formalist*.[7] Since intuitionistic logic has subsequently also been formalized (e.g., by Kolmogorov and Heyting), the modern way of framing the opposing sides are *intuitionistic logic* (or arithmetic) and *classical logic* (or arithmetic).

One of the key differences is the interpretation of the existential quantifier. In intuitionistic logic, proving $\exists x.\, A(x)$ needs a witness t and a proof of $A(t)$. In classical logic,

[5] See https://en.wikipedia.org/wiki/Banach-Tarski_paradox

[6] See https://en.wikipedia.org/wiki/L._E._J._Brouwer

[7] For more on this controversy in the foundations of mathematics, see https://en.wikipedia.org/wiki/Brouwer-Hilbert_controversy.

it is sufficient to show that $\forall x.\, \neg A(x)$ is impossible without exhibiting an actual witness for x in the proof.

As example, we consider the following theorem and proof.

Theorem 2. *There are two irrational numbers a and b such that a^b is rational.*

Proof: Consider $\sqrt{2}^{\sqrt{2}}$. There are two cases:

Case: $\sqrt{2}^{\sqrt{2}}$ is rational. Then $a = b = \sqrt{2}$ satisfies the claim.

Case: $\sqrt{2}^{\sqrt{2}}$ is irrational. Then $a = \sqrt{2}^{\sqrt{2}}$ and $b = \sqrt{2}$ satisfy the claim, since $a^b = (\sqrt{2}^{\sqrt{2}})^{\sqrt{2}} = \sqrt{2}^2 = 2$.

□

At this point, the classical mathematician is profoundly happy, since this is an extremely short and elegant proof of a prima facie nontrivial theorem. The intuitionist is profoundly unhappy, since the proof does not actually exhibit irrational witnesses a and b such that a^b is rational. They might either be $a = b = \sqrt{2}$, or they might be $a = \sqrt{2}^{\sqrt{2}}$ and $b = \sqrt{2}$, but the proof does not show which it is. Therefore an intuitionist should reject this proof.

The step which turns out to be incorrect here is to assume that there are two cases (either $\sqrt{2}^{\sqrt{2}}$ is rational or not) without knowing which of the cases hold. More generally, an intuitionist rejects the *law of excluded middle* that any proposition is either true or false (in symbols: $A \vee \neg A$). Concretely, a constructive proof of a disjunction $A \vee B$ is either a proof of A or a proof of B. So in addition to existential quantification, the intuitionistic and classical mathematician disagree on the interpretation of disjunction.

However, all is not lost! Intuitionists look at the above proof and say

> *I understand your proof, but it is for a different theorem! What you have proven is:*
>
> > **Theorem.** If $\sqrt{2}^{\sqrt{2}}$ is rational or it is not, then there are two irrational numbers a and b such that a^b is rational.

Surprisingly, as long as we stick to pure logic, or perhaps the theory of natural numbers, any classical proof can be reinterpreted as an intuitionistic proof but of a different theorem![8] This suggests that, once we accept that intuitionism can in fact also be formalized, intuitionistic and classical are no longer in conflict. Instead, intuitionistic logic is a *generalization* of classical logic in the sense that it has a constructive existential quantifier and a constructive disjunction, which is absent from classical logic. At the same time, all classical theorems and proofs can be uniformly imported into intuitionistic logic under some translation.

The intuitionistic interpretation of the proof above yields another question: what is the nature of implication? The intuitionistic interpretation of this particular example

[8] We may show this interpretation in a future lecture.

clarifies this: the proof of $A \supset B$ consists of a *function* to convert a proof of A into a proof of B. Here, this function proceeds by analyzing the proof of whether $\sqrt{2}^{\sqrt{2}}$ is rational or not. If it is rational, we return the witnesses $a = b = \sqrt{2}$, together with the proof that a^b is rational in this case (which we were in fact given). If it is irrational, we return the witnesses $a = \sqrt{2}^{\sqrt{2}}$ and $b = \sqrt{2}$, together with a (simple equational) proof that $a^b = 2$ in this case. With a suitable understanding, this function seems en route to a perfectly good program.

Through this example, we have already identified three critical intuitionistic principles and add two more (Troelstra and van Dalen):

1. An intuitionistic proof of $\exists x.\, A(x)$ exhibits a witness t and a proof of $A(t)$.
2. An intuitionistic proof of $A \vee B$ consists of either a proof of A or a proof of B.
3. An intuitionistic proof of $A \supset B$ contains a construction that transforms a proof of A into a proof of B.
4. Additionally, an intuitionistic proof of $\forall x.\, A(x)$ contains a construction that takes an arbitrary element d as argument and exhibits a proof of $A(d)$.
5. $\bot$ has no proof and an intuitionistic proof of $\neg A$ is a construction taking a (hypothetical) proof of A to a contradiction.

To achieve these, an intuitionist has to reject some classical reasoning principles or axioms. In natural deduction (as discussed in Lecture 2), this is manifest in the single *axiom of excluded middle*.

As a final example, consider the claim:

Theorem 3. *Among all fast algorithms, there is a "super NP-complete algorithm": if this algorithm's results can be checked quickly, then every algorithm's results can be checked quickly.*

In logical language, we could formalize this claim as

$$\exists x.\, (\mathsf{quickcheck}(x) \supset \forall y.\, \mathsf{quickcheck}(y))$$

where the quantifiers range over all algorithms (never mind that we're ultimately using constructive logic proofs as algorithms instead of as objects that quantifiers range over). Here is the (non-constructive!) proof

Proof: Either every algorithm's results can be checked quickly, or there is at least one algorithm s whose results cannot be checked quickly.

Case: Every algorithm's results can be checked quickly. Then any x will do[9], because the conclusion of the implication holds.

[9] Of course, there still needs to be some algorithm at all, but every student who doubts this may want to take their first semester classes again just in case.

Case: There is some algorithm s whose results cannot be checked quickly. Then this algorithm s is a super NP-complete: since $\mathsf{quickcheck}(\mathsf{s})$ is false, then the implication $\mathsf{quickcheck}(x) \supset \forall y.\, \mathsf{quickcheck}(y)$ is true.

□

This shallow proof is non-constructive (and has nothing to do with the Cook-Levin theorem that witness the SAT problem has this property among all problems with quickly checkable answers), because we use a form of the excluded middle to avoid naming a witness to the existential. Actually, as long as the domain of quantification is non-empty (which is assumed in classical logic except in free logics), this proof has nothing to do with algorithms and the quick check capabilities of their results, but the proof above applies to the logical form

$$\exists x.\, (A(x) \supset \forall y.\, A(y))$$

Intuitionistically, we cannot prove this without further assumptions about A.[10]

The next lecture looks closely at the intuitionistic meaning of the logical connectives and their proof rules, based on the interpretation we sketched in this lecture.

[10] Exercise: which intuitionistically true proposition does the proof above establish instead?