Lecture Notes on Formal Proof

15-316: Software Foundations of Security & Privacy
Matt Fredrikson

Lecture 2
January 20, 2026

1 Introduction

We begin by talking about safety, and in particular one important class of safety properties: memory safety. Mistakes that programmers make having to do with memory safety have been responsible for a lot of security vulnerabilities over the years, and we will discuss some of the different types of memory safety errors that lead to vulnerabilities in common programs.

Our goal is to write code that does not have these vulnerabilities. There are many ways that have been proposed to avoid memory safety errors: type systems that ensure that memory is always used correctly, tools that check for memory safety errors at compile time, runtime systems that detect and prevent errors, and many others at various parts of the software and hardware stack. To understand how these work, and what their limitations and tradeoffs are, we need to understand how memory safety materializes in programs at the level of precise semantics.

In the second part of this lecture, we will go into detail to understand the semantics of programs and how they relate to proving things about how programs will behave when run. We won't yet define exactly what memory safety means at this level, but we will have laid the essential groundwork to do this, as well as to start reasoning about the safety of programs in more systematic ways.

2 Review: Program Semantics

In the previous lecture we described the smantics of a small imperative programming language. For the sake of simplicity we assumed that variables range of the integers $\mathbb{Z}$. Arithmetic expressions, denoted by placeholder e, were given by a simple grammar.

$$\text{Arithmetic Expressions} \quad e \quad ::= \quad c \mid x \mid e_1 + e_2 \mid e_1 - e_2 \mid e_1 * e_2 \mid \ldots$$

Their meaning is determined with respect to a *state* (ω, μ, ν) that assigns integers to variables. We write $\omega(x) = c$ if ω maps x to c. Then we saw how to define the value of expression e in state ω, written as $\omega[\![e]\!] = c$, in a straightforward fashion.

$$\begin{array}{lcl}
\omega[\![c]\!] & = & c \\
\omega[\![x]\!] & = & \omega(x) \\
\omega[\![e_1 + e_2]\!] & = & \omega[\![e_1]\!] + \omega[\![e_2]\!] \\
\omega[\![e_1 - e_2]\!] & = & \omega[\![e_1]\!] - \omega[\![e_2]\!] \\
\ldots & &
\end{array}$$

The meaning of a program is a *relation* between the *prestate* and *poststate* of its execution, and we write

$$\omega[\![\alpha]\!]\nu$$

if the meaning of the program α relates prestate ω to poststate ν. With these conventions in place, defining the semantics of assignment, sequential composition, and conditional statements is a matter being unambiguous about which pre and post states are related.

$$\begin{array}{lll}
\omega[\![x := e]\!]\nu & \text{iff} & \omega[x \mapsto c] = \nu \text{ where } \omega[\![e]\!] = c \\
\omega[\![\alpha \,;\, \beta]\!]\nu & \text{iff} & \omega[\![\alpha]\!]\mu \text{ and } \mu[\![\beta]\!]\nu \text{ for some state } \mu \\
\omega[\![\textbf{if } P \textbf{ then } \alpha \textbf{ else } \beta]\!]\nu & \text{iff} & \omega \models P \text{ and } \omega[\![\alpha]\!]\nu \quad \text{or} \\
 & & \omega \not\models P \text{ and } \omega[\![\beta]\!]\nu
\end{array}$$

Loops take a bit more care. There are several ways we might approach loops, because there are several possibilities about what can happen when going to execute a loop: the guard might be immediately false; or the body of the loop itself might not terminate; or the guard might never become false. Note that in all of these cases but the first, the loop doesn't terminate, so there is no post-state to relate to the pre state in question.

A concise way to navigate all this is to define a second relation between pre and post states, which is defined only for loops, and which is indexed on the number of iterations that the loop executes for.

$$\begin{array}{lll}
\omega[\![\textbf{while } P\ \alpha]\!]\nu & \text{iff} & \omega[\![\textbf{while } P\ \alpha]\!]^n\nu \quad \text{for some } n \in \mathbb{N} \\
 & & \\
\omega[\![\textbf{while } P\ \alpha]\!]^{n+1}\nu & \text{iff} & \omega \models P \text{ and } \omega[\![\alpha]\!]\mu \text{ and } \mu[\![\textbf{while } P\ \alpha]\!]^n\nu \\
\omega[\![\textbf{while } P\ \alpha]\!]^0\nu & \text{iff} & \omega \not\models P \text{ and } \omega = \nu
\end{array}$$

Although we might not ourselves know for a given program what precise value of n will satisfy this definition, or even whether such an an n exists, it does not matter for our purposes here. What matters is that the definition matches our understanding of how an imperative loop should behave: if a loop terminates in some final state ν, then there must be some number of iterations that can reach ν from the initial state ω, with $n - 1$ intermediate states related by the body α.

3 A Program Proof

Now that we have precise semantics to work with, let's apply them towards proving something about how a particular program behaves. We'll prove that the program

$$x := x + y \ ; y := x - y \ ; x := x - y$$

swaps the values of x and y. We'll approach this by working backwards, supposing that the program ends in a final state where x and y are mapped to arbitrary constants, assuming nothing else. We'll show that given the semantics from Lecture 2, any initial state related to this final state must map x and y to these constants, but swapped.

So suppose we have an arbitrary initial state ω and an arbitrary final state ν such that

$$\omega[\![x := x + y \ ; y := x - y \ ; x := x - y]\!]\nu$$

and in the final state $\nu(x) = a$ and $\nu(y) = b$ for some arbitrary constants a and b. We will show that necessarily $\omega(x) = b$ and $\omega(y) = a$.

We set up the proof:

$\omega[\![x := x + y \ ; y := x - y \ ; x := x - y]\!]\nu$ (1, assumption)
$\nu(x) = a$ and $\nu(y) = b$ (2, assumption)
...
$\omega(x) = b$ and $\omega(y) = a$ (to show)

By definition of sequential composition, assumption 1 tells us that there must be intermediate states μ_1 and μ_2 such that the three assignments relate the states in sequence.

$\omega[\![x := x + y]\!]\mu_1$ and $\mu_1[\![y := x - y \ ; x := x - y]\!]\nu$ (3, from 1 by defn. of $[\![-]\!]$)
$\mu_1[\![y := x - y]\!]\mu_2$ and $\mu_2[\![x := x - y]\!]\nu$ (4, from 3(b) by defn. of $[\![-]\!]$)

Now we start working backwards from the last assignment. By definition of assignment, $\mu_2[\![x := x - y]\!]\nu$ means that ν is obtained from μ_2 by updating x to the value of $x - y$ evaluated in μ_2.

$\nu = \mu_2[x \mapsto c_3]$ where $c_3 = \mu_2[\![x - y]\!]$ (5, from 4(b) by defn. of $[\![-]\!]$)
$\mu_2(y) = \nu(y) = b$ (6, from 2, 5 by defn. of update)
$\nu(x) = \mu_2[\![x - y]\!] = \mu_2(x) - \mu_2(y) = a$ (7, from 2, 5, 6 by defn. of state update and $[\![-]\!]$)
$\mu_2(x) = a + b$ (9, from 6 and 8)

Next we unwind the second assignment $y := x - y$. This relates μ_1 to μ_2 by updating only y, so x stays the same from μ_1 to μ_2.

$\mu_2 = \mu_1[y \mapsto c_2]$ where $c_2 = \mu_1[\![x - y]\!]$ (10, from 4(a) by defn. of $[\![-]\!]$)
$\mu_1(x) = \mu_2(x) = a + b$ (12, from 10 by defn. of state update)
$\mu_2(y) = \mu_1[\![x - y]\!] = \mu_1(x) - \mu_1(y) = b$ (13, from 12 by defn. of state update)
$\mu_1(y) = a$ (14, from 12 and 13)

Finally we unwind the first assignment $x := x + y$ to relate ω to μ_1. This assignment updates only x, so y stays the same from ω to μ_1.

$\mu_1 = \omega[x \mapsto c_1]$ where $c_1 = \omega[\![x + y]\!]$ (15, from 3(a) by defn. of $[\![-]\!]$)
$\omega(y) = \mu_1(y) = a$ (16, from 15 by defn. of state update)
$\mu_1(x) = \omega[\![x + y]\!] = \omega(x) + \omega(y) = a + b$ (17, from 15 by defn. of state update)
$\omega(x) = b$ (18, from 16 and 17)

We have now shown $\omega(x) = b$ and $\omega(y) = a$, as desired. Since a and b were arbitrary, we have shown that in every execution of this program the final value of x is the initial value of y and the final value of y is the initial value of x.

4 Formal Proof

We've now seen how defining precise semantics for our programming language makes it possible to prove conclusive facts about what will happen when a program runs. Although the property that we proved was simple, following from the simplicity of the program, it's not difficult to see how we could apply a similar process to, for example, proving that the expression used to index into an allocated array is within its bounds; or that a pointer is not null prior to dereferencing it. The semantics of the language would be more complex, as would the program in question in nearly all cases, but the underlying approach would be very similar.

But notice that there is some repetitive structure involved in the proof. We started with a straightforward assumption $x = a \land y = b$, and inferred the existence of intermediate states μ_1 and μ_2, from the semantics of sequential composition. Then for each assignment, we used the semantics of assignment to infer that,

1. The only variable to have changed was the one appearing on the left hand side, so whatever we know about the other variables in the post-state translates directly to the pre-state. We can simply carry that information forward as we work through the proof.

2. Whatever we know about the left-hand variable in the post-state is a direct result of it being assigned at the point in question. We make an inference about what the value of that variable must have been in the pre-state in order for our current assumptions about the post-state to be true, and carry the resulting fact forward as a new assumption.

Through this process of consulting the semantics to make inferences that update our assumptions, we eventually reached a point where we could prove our original goal. We'll now look to make this process more systematic, so that the process of looking for proofs, and checking whether they are correct, is streamlined. In doing so, we'll briefly step back from programs and turn our focus on logical formulas.

5 Propositional Formulas

In the previous lecture, we introduced a language of logical formulas over integer variables, for use within our programs as guards on conditionals and loops. For today, we'll make our language of formulas even simpler, and assume that the variables appearing in formulas are Boolean-valued *propositional variables* $p, q, \ldots$. The rest of the logic will stay intact (conjunction, disjunction, ...), but of course we will no longer have inequality $\leq$, and we won't bother with talking about the equality of variables with $=$. This is called the Propositional Logic, as expressed in the following so-called Backus-Naur Form (BNF).

$$\text{Formulas} \quad F, G, H \quad ::= \quad p \mid F \wedge G \mid F \vee G \mid F \to G \mid \neg F \mid \top \mid \bot$$

Variables can take the value $\top$ (true) or $\bot$ (false), and the meaning of the other connectives are given by their truth tables. For example, $F \wedge G$ is true if both F and G are true, and false otherwise. $F \to G$ is true either F is false or G is true. We also refer to propositional variables as *atomic proposition* because they cannot be further decomposed. In the next lecture we will add other atomic propositions besides variables.

We say a formula F is *valid* if it is true no matter which truth values we assign to the variables contained in them. A formula F is *satisfiable* if there is some way to assign truth values to the variables so that the resulting formula is true.

6 Simple Sequents

We now want to devise a system of inference rules so we can *formally prove* that a given proposition is valid. It should be designed so that if someone doubts the validity of a proposition you can convince them by showing them the proof. Each step in the proof should be correct, and each step should be easy to check.

The design of systems of inference will occupy a lot of our time, and it is far from straightforward. There are many choices, and the same concepts may admit many different formalization, suitable for different purposes. For example, we could have very few inference rules (in propositional logic often just one!) and many axioms, usually called a Hilbert-style system. Or we could have many rules and essentially no axioms. For this lecture we choose Gentzen's *sequent calculus* [Gentzen, 1935]. It has several useful properties which we will come back to starting in Section 11. One of them is that many logics permit formulations as sequent calculi because they are constructed very systematically.

A *simple sequent* has the form $F_1, \ldots, F_n \vdash G$ where $F_1, \ldots, F_n$ are the *antecedents* and G is the *succedent*. The symbol "$\vdash$" separating them is called a *turnstile* and usually pronounced "*entails*". A sequent formalizes the state of a proof where we have the *assumptions* $F_1, \ldots, F_n$ and try to prove the *goal* G. In fact, most proof assistants like Coq or Lean will present the state of a missing proof in the form of a sequent because it reflects the way that mathematicians think about the state of an (incomplete) proof.

We often abbreviate the collection of antecedents as Γ (capital *Gamma*). Also, their order is irrelevant, so we consider, for example, p, q to be the same as q, p.

More formally, we say a sequent $\Gamma \vdash G$ is *valid* if G is true whenever all propositions in Γ are true. We often write this out as "all Γ true implies G true".

7 Right and Left Rules

A pleasant property of the sequent calculus is that the connectives are defined in isolation from each other. This makes it easy to consider subsets of the connectives or extensions with additional connectives because in many cases we don't have to change any of the existing rules.

The rules comes in two flavors. The *right rules* define how to decompose the consequent, that is, the goal proposition on the right-hand side of the turnstile. Conversely, the *left rules* define how to decompose an antecedent, that is, an assumption on the left-hand side of the turnstile.

Let's start with conjunction. How do we prove $F \wedge G$ from some assumptions Γ? Easy: we prove both F and G separately from the assumptions. Formulated as a rule:

$$\frac{\Gamma \vdash F \quad \Gamma \vdash G}{\Gamma \vdash F \wedge G} \wedge R$$

We write (and read) this rule bottom-up. Reading upwards, we see that we eliminate the conjunction from the given formulas.

The sequent below the line is called the *conclusion*, while the two sequents above the line are the *premises*. The name of the rule, $\wedge R$ (read: "and-right"), is written to the right of the line.

We also have to consider how we *use an assumption* $F \wedge G$. The insight is that if we know $F \wedge G$ is true, then F and G must both be true. As a rule (again, read it from the bottom up):

$$\frac{\Gamma, F, G \vdash H}{\Gamma, F \wedge G \vdash H} \wedge L$$

For this rule we implicitly apply our convention that the order of the antecedents does not matter. So $F \wedge G$ doesn't actually have to be the last antecedent, it just has to be one of them.

We are almost ready to do our first formal proof, that is, derivation. But we have to be able to complete a partial proof and declare victory. Recall that $\Gamma \vdash F$ is valid (by definition) if whenever all formulas in Γ are true then F is true. This justifies the following *rule of identity*:

$$\frac{}{\Gamma, F \vdash F} \mathsf{id}$$

It may look strange at first because it is a rule with no premises. As we construct our sequent calculus proofs bottom-up, applications of the identity rule sit at the top (which are the leaves, if we think of the proof as a tree).

Let's try to prove the sequent $p \wedge q \vdash q \wedge p$. It should be intuitively clear that this is valid, so we should be able to prove it. We construct the derivation bottom-up, starting with

$$\begin{array}{c} \vdots \\ p \wedge q \vdash q \wedge p \end{array}$$

There are two options: we could apply either the right rule of the left rule for conjunction. Let's use the left rule. Our as yet incomplete derivation now looks like this:

$$\dfrac{\begin{array}{c} \vdots \\ p, q \vdash q \wedge p \end{array}}{p \wedge q \vdash q \wedge p}\ \wedge L$$

At this point we can only apply the right rule for conjunction.

$$\dfrac{\dfrac{\begin{array}{c} \vdots \\ p, q \vdash q \end{array} \quad \begin{array}{c} \vdots \\ p, q \vdash p \end{array}}{p, q \vdash q \wedge p}\ \wedge R}{p \wedge q \vdash q \wedge p}\ \wedge L$$

We recognize both unproved goals as instances of the identity rule and finish our derivation.

$$\dfrac{\dfrac{\dfrac{}{p, q \vdash q}\ \mathsf{id} \quad \dfrac{}{p, q \vdash p}\ \mathsf{id}}{p, q \vdash q \wedge p}\ \wedge R}{p \wedge q \vdash q \wedge p}\ \wedge L$$

8 Implication

We move on to implication $F \to G$. How do we prove an implication? We assume F and then prove G under this additional assumption.

$$\dfrac{\Gamma, F \vdash G}{\Gamma \vdash F \to G}\ {\to}R$$

Remember to read the rule bottom-up.

In order to understand the left rule, we have to think about how we use an assumption $F \to G$. Here is a first attempt: if we also know F we are permitted to use G.

$$\dfrac{\Gamma, F, G \vdash H}{\Gamma, F, F \to G \vdash H}\ {\to}L?$$

While this rule is sound (see Section 12) it is insufficient. Here is an example:

$$\begin{array}{c}\vdots \\ (p \to p) \to q \vdash q\end{array}$$

Is this sequent valid? This is easy to check by a truth table argument. Assume that $(p \to p) \to q$. We have to show that q is true. Since $p \to p$ is always true, so we know that q must also be true.

But in the system so far, no rule can be applied! Since the succedent is a variable and the only antecedent is an implication, the only rule that could apply is a left rule for implication. But it doesn't, since we do not have $p \to p$ as one of our antecedents.

The fix here is not at all obvious. We reinterpret uses of an implication as follows: if we know $F \to G$ and we can prove F then we know G. By "know" here we mean that it is one of our assumptions. Writing this out in the form of a sequent rule:

$$\frac{\Gamma \vdash F \quad \Gamma, G \vdash H}{\Gamma, F \to G \vdash H} \to L$$

As noted by a student in lecture, there seems to be a "temporal" dependency: we only have license to assume G if we have first proved F. But during bottom-up proof construction we could also proceed with the second premise first. That is, we could "check" (by proving) that G entails H and only if that's the case, do we bother to (try to) prove F.

Now we can prove the problematic example from above. As usual, we proceed bottom up, but we only show the final derivation.

$$\dfrac{\dfrac{\dfrac{}{p \vdash p}\,\mathsf{id}}{\cdot \vdash p \to p} \to R \qquad \dfrac{}{q \vdash q}\,\mathsf{id}}{(p \to p) \to q \vdash q} \to L$$

An excellent question from lecture: *"What is the difference between $F \to G$ and $F \vdash G$?"* Indeed, $F \to G$ is valid as a formula (that is, always true) if and only if $F \vdash G$ is valid as a sequent. Let's think about how you might go about building a prover for the validity of formulas. You are given a *formula* $F \to G$ and you want to construct a derivation of the sequent $\cdot \vdash F \to G$. Here we use "$\cdot$" to emphasize that there is an empty collection of antecedents. We break this down and now to have to build a derivation of $F \vdash G$, where you have a singleton list F as an antecedent and G as a conclusion. So $F \vdash G$ is the result of decomposing the implication, now exposing the top-level logical connectives of F and G for further (bottom-up) application of inference rules. In contrast, $F \to G$ is just a single thing: a formula that's an implication.

9 Disjunction

Disjunction throws another wrench into the works that's not as easy to repair as for implication. From the truth table definition we know that $F \vee G$ is true if either F is true or G is true. This time, we start with the left rule because it is easier. When have an assumption $F \vee G$ we can proceed in a proof by distinguishing two cases, proving our goal H in both of them. That is:

$$\frac{\Gamma, F \vdash H \quad \Gamma, G \vdash H}{\Gamma, F \vee G \vdash H} \vee L$$

From the same truth table definition, we can also conjecture that there should be *two* right rules.

$$\frac{\Gamma \vdash F}{\Gamma \vdash F \vee G} \vee R_1? \qquad \frac{\Gamma \vdash G}{\Gamma \vdash F \vee G} \vee R_2?$$

It is clear that during proof construction we now have to make a choice, which also means we may have to backtrack over this choice (consider $q \vdash p \vee q$).

What's worse, there are now some valid sequents that we cannot prove! If you weren't in lecture, or you don't remember, it is quite a brainteaser to find something valid you cannot prove. Think about it before moving on to the next page.

Consider $p \vee (p \rightarrow q)$. This is valid: if p is true then the left disjunct is true. If p is false, then the right disjunct $p \rightarrow q$ is true. Either way, the disjunction is true. But with the two right rules for disjunction, we cannot prove it. There are only two possible attempts.

$$\dfrac{\dfrac{\text{XXX}}{\cdot \vdash p}}{\cdot \vdash p \vee (p \rightarrow q)} \vee R_1 \qquad \dfrac{\dfrac{\overset{\text{XXX}}{p \vdash q}}{\cdot \vdash p \rightarrow q} \rightarrow R}{\cdot \vdash p \vee (p \rightarrow q)} \vee R_2$$

The first is not valid because the variable p could be false. The second is not valid because p could be true and q could be false.

Interestingly, even though it is incomplete for the Boolean interpretation of the formulas using two truth values, the sequent rules we have shown so far can be developed further into *intuitionistic logic*. In intuitionistic logic, proofs correspond to (functional) programs, and propositions correspond to their types. For example, an intuitionistic proof of $F \rightarrow G$ is a function of type $F \rightarrow G$ that takes a proof of F into a proof of G. You can learn more about this in *15-317 Constructive Logic* where we develop a logic and a functional programming language together in a single system.

In this course, starting in the next lecture, we instead define a small imperative programming language Tinyscript and then reason about it externally using a logic where every formula is either true or false.

10 Sequents with Multiple Succedents

Gentzen's solution to the problem with disjunction is quite clever and not obvious because it doesn't reflect the way we do mathematics. The idea is to allow sequents to have multiple succedents. We define

$$(F_1, \ldots, F_n \vdash G_1, \ldots, G_m) \quad \text{is valid}$$

if whenever all F_i are true then at least one G_j is true. Put it another way, the sequent $F_1, \ldots F_n \vdash G_1, \ldots, G_m$ is valid if and only if the formula $(F_1 \wedge \cdots \wedge F_n) \rightarrow (G_1 \vee \cdots \vee G_m)$ is valid. We generally abbreviate antecedents by Γ and succedents by Δ (capital *Delta*).

Having multiple succedents that are interpreted *disjunctively* allows us to hedge our bets with a better right rule.

$$\dfrac{\Gamma \vdash F, G, \Delta}{\Gamma \vdash F \vee G, \Delta} \vee R$$

$$\frac{}{\Gamma, F \vdash F, \Delta}\ \mathsf{id}$$

$$\frac{\Gamma \vdash F, \Delta \quad \Gamma \vdash G, \Delta}{\Gamma \vdash F \wedge G, \Delta}\ {\wedge}R \qquad \frac{\Gamma, F, G \vdash \Delta}{\Gamma, F \wedge G \vdash \Delta}\ {\wedge}L$$

$$\frac{\Gamma, F \vdash G, \Delta}{\Gamma \vdash F \to G, \Delta}\ {\to}R \qquad \frac{\Gamma \vdash F, \Delta \quad \Gamma, G \vdash \Delta}{\Gamma, F \to G \vdash \Delta}\ {\to}L$$

$$\frac{\Gamma \vdash F, G, \Delta}{\Gamma \vdash F \vee G, \Delta}\ {\vee}R \qquad \frac{\Gamma, F \vdash \Delta \quad \Gamma, G \vdash \Delta}{\Gamma, F \vee G \vdash \Delta}\ {\vee}L$$

$$\frac{\Gamma, F \vdash \Delta}{\Gamma \vdash \neg F, \Delta}\ {\neg}R \qquad \frac{\Gamma \vdash F, \Delta}{\Gamma, \neg F \vdash \Delta}\ {\neg}L$$

Figure 1: Sequent Calculus with Multiple Succedents

The price we pay is that we now have to generalize all of our other rules to allow multiple conclusions. We summarize them in Figure 1. We have added the rules for negation, to come in Section 16.

Let's revisit our earlier problem to make sure we can now derive $p \vee (p \to q)$.

$$\frac{\begin{array}{c}\vdots\\ \cdot \vdash p, p \to q\end{array}}{\cdot \vdash p \vee (p \to q)}\ {\vee}R$$

At this point we can only apply one rule, the right rule for implication.

$$\frac{\dfrac{\begin{array}{c}\vdots\\ p \vdash p, q\end{array}}{\cdot \vdash p, p \to q}\ {\to}R}{\cdot \vdash p \vee (p \to q)}\ {\vee}R$$

The unproved leaf is just an instance of the identity rule and we are done.

$$\frac{\dfrac{\dfrac{}{p \vdash p, q}\ \mathsf{id}}{\cdot \vdash p, p \to q}\ {\to}R}{\cdot \vdash p \vee (p \to q)}\ {\vee}R$$

11 Properties of Inference Systems

Question: are we done? We discovered some flaws in earlier rules, so how can we know that they are all fixed? Well, we can appeal to authority (that is, Gentzen) and consider our job done. But that is unsatisfactory because we often have to think about variations (like: introducing programs and reasoning about them) and then the right inference system may not be immediately available. Also, by actually mathematically proving that our systems "works" we gain some insights that can be exploited not only for other, closely related systems but also for an implementation.

Two fundamental properties of inference system such as the one in Figure 1 are *soundness* and *completeness*. For sequents, they are:

Soundness: If we can derive $\Gamma \vdash \Delta$ then $\Gamma \vdash \Delta$ is valid.

Completeness: If $\Gamma \vdash \Delta$ is valid then we can derive $\Gamma \vdash \Delta$.

The flaws we found so far (for example, in $\rightarrow L?$ and $\vee R_1?, \vee R_2?$ were failures of completeness: there were valid sequents we could not prove. Fortunately, our generalization to multiple conclusions have led us to a system that is both sound and complete.

There are other properties of interest. For example, we can ask if a logic is *decidable*. On the sequent calculus this would mean that we can effectively either prove or refute the validity of any given sequent. Again, our sequent calculus will support this property. More properties to come.

12 Proving Soundness

We start with soundness. The way we prove it is to show that whenever all premises of a rule are valid, so is the conclusion. Since at the leaves we have no premises, the conclusion must then be valid outright. And therefore any sequent we derive in a (complete) derivation must be valid. A nice consequence of this approach is that we can consider the soundness rule by rule. If we restrict our language, soundness still holds, and if we extend it we only have to check the new rules.

Since we have to talk a lot about validity, we sometimes abbreviate

$$\text{If all } F \in \Gamma \text{ are true then some } G \in \Delta \text{ is true}$$

as

$$\text{All } \Gamma \text{ true implies some } \Delta \text{ true}$$

Identity. Since there are no premises, we need to show that $\Gamma, F \vdash F, \Delta$ is valid. This means:

$$\text{All } (\Gamma, F) \text{ true implies some } (F, \Delta) \text{ true}$$

So we assume that all (Γ, F) are true, which implies that F is true. But that means at least one of (F, Δ) is true (namely F).

Implication Right ($\rightarrow R$).

$$\frac{\Gamma, F \vdash G, \Delta}{\Gamma \vdash F \rightarrow G, \Delta} \rightarrow R$$

We set up the proof as follows.

all (Γ, F) true implies some (G, Δ) true	(validity of premise)
all Γ true	(assumption)
...	
some $(F \rightarrow G, \Delta)$ true	(to show)

We proceed with the proof (filling in the gap marked by "...") by distinguishing cases for F: either F is true or F is false.

all (Γ, F) true implies some (G, Δ) true	(validity of premise)
all Γ true	(assumption)
case: F is true	
...	
some $(F \rightarrow G, \Delta)$ true	(to show)
case: F is false	
...	
some $(F \rightarrow G, \Delta)$ true	(to show)
some $(F \rightarrow G, \Delta)$ true	(by cases on F)

Now we can fill in the gaps rather easily.

all (Γ, F) true implies some (G, Δ) true	(validity of premise)
all Γ true	(assumption)
case: F is true	
all (Γ, F) true	(from assumption)
some (G, Δ) true	(from premise)
subcase: G is true	
$F \rightarrow G$ true	(by truth table for $\rightarrow$)
some $(F \rightarrow G, \Delta)$ true	(since $F \rightarrow G$ true)
subcase: some Δ true	
some $(F \rightarrow G, \Delta)$ true	(since some Δ true)
some $(F \rightarrow G, \Delta)$ true	(by cases on G, Δ)
case: F is false	

$F \to G$ is true (by truth table for $\to$)
some $(F \to G, \Delta)$ true (since $F \to G$ true)

some $(F \to G, \Delta)$ true (by cases on F)

This is a rather pedantic way to write down the steps, so in the next proof case we'll proceed more compactly.

Implication Left ($\to L$).

$$\frac{\Gamma \vdash F, \Delta \quad \Gamma, G \vdash \Delta}{\Gamma, F \to G \vdash \Delta} \to L$$

Then we set up:

all Γ true implies some (F, Δ) true (first premise)
all (Γ, G) true implies some Δ true (second premise)
all $(\Gamma, F \to G)$ true (assumption)
...
some Δ true (to show)

From the assumption and first premise, we know some (F, Δ) true. If Δ true we are done. Otherwise, F true and therefore (by assumption $F \to G$ true) we know G is true. By the second premise we then know Δ true.

All other rules can be proved sound in a similarly systematic manner.

Theorem 1 (Soundness of the Sequent Calculus)
If we can derive $\Gamma \vdash \Delta$ then $\Gamma \vdash \Delta$ is valid

Proof: All rules are sound. In particular, for the only leaves of the proof tree (applications of the identity rule), the conclusion is valid outright. All other rules preserve validity, and so any sequent we can derive is valid. □

It is somewhat peculiar (but common), that we motivate all the rules reading them bottom-up, but for soundness we argue with them top-down.

13 Inversion

We would like to build a theorem prover by constructing a proof in a bottom-up manner, using our rules of inference. A basic issue is that for a sequent $F_1, \ldots, F_n \vdash G_1, \ldots, G_m$ many rules may apply: left rules for formulas F_i and right rules for formulas G_j. Backtracking over all such choices quickly makes proof search infeasible.

In important class of rules are those that are *invertible*: if the conclusion is valid then are all the premises. We can always apply (bottom-up!) invertible rules in our proof search without having to backtrack: we don't lose validity.

In most inference systems we classify rules are being invertible or not. Here, remarkably, all rules are invertible! This means when a rule can be applied, it's always safe to do so.

We prove a few cases of the invertible rules. It is quite similar to the soundness proof of the rules.

Identity.

$$\frac{}{\Gamma, F \vdash F, \Delta}\ \mathsf{id}$$

The identity rule has no premises, so there is nothing to show.

Disjunction Right ($\vee R$) . Disjunction gave us problems when we conjecture two rules. Indeed, neither of them is invertible. Going to multiple conclusions solved this problem.

$$\frac{\Gamma \vdash F, G, \Delta}{\Gamma \vdash F \vee G, \Delta}\ \vee R$$

We set up:

all Γ true implies some $(F \vee G, \Delta)$ true (validity of conclusion)
all Γ true (assumption)
. . .
some (F, G, Δ) true (to show)

From the assumption and the validity of the conclusion we know that some $(F \vee G, \Delta)$ true. If some Δ is true, then also some (F, G, Δ). If $F \vee G$ is true then by the truth table, either F or G must be true. In either case, some of F, G, Δ true.

Disjunction Left ($\vee L$) .

$$\frac{\Gamma, F \vdash \Delta \quad \Gamma, G \vdash \Delta}{\Gamma, F \vee G \vdash \Delta}\ \vee L$$

We set up to show that the validity of the conclusion implies the validity of the first premise. The proof for the second premise is symmetric.

all $\Gamma, F \vee G$ true implies some Δ true (validity of conclusion)
all Γ, F true (assumption)
. . .
some Δ true (to show)

Since all Γ, F true, we have all Γ true and F true. Therefore (by truth table), $F \vee G$ true and also $\Gamma, F \vee G$. Then some Δ true by validity of the conclusion.

Theorem 2 (Invertibility)
All rules in the sequent calculus are invertible.

Proof: Case by case, as exemplified above. □

14 Termination

The sequent calculus in Figure 1 has another remarkable property: every premise of every rule is *smaller* than its conclusion. Smaller in which sense? Every rule, if read bottom-up, removes at least one of the connectives from the sequent. Therefore, if we just count the number of connectives, sequents become smaller during bottom-up proof search. Eventually we must either reach a rule with no premises, or we reach a sequent with no connectives:

$$p_1, \ldots, p_n \vdash q_1, \ldots, q_m$$

Such a sequent is valid if and only if one of the p_i is equal to one of the q_j, that is, precisely if the identity rule applies. Otherwise, we can set all p_i to true and all q_j to false and observe that the sequent is not valid.

This means we can use proof search in the sequent calculus as a very simple decision procedure. Invertibility tells us we can blindly apply left and right rules without losing validity. Then we check if the identity rule applies to the leaves.

There is a small matter of strategy: in such a prover, we should probably apply 0-premise rules (proof is done!), before 1-premise rules (a proof goal replaced by an equivalent, smaller one), before 2-premise rules (because we now have two sequents to prove). This is only a heuristic and a matter of efficiency.

While sequent calculus search represents a decision procedure for validity in propositional logic, it is not one that is commonly used for efficiency reasons. The course *15-311 Logic and Mechanized Reasoning* goes into great detail about different approaches to proving propositional formulas and how to represent their proof in practice. It also examines many applications in mathematics and a few in computer science.

A consequence of the fact that all rules are invertible is that we could restrict the identity rule to just propositional variables, as in

$$\frac{}{\Gamma, p \vdash p, \Delta}\ \mathsf{id}^*$$

It wouldn't change the sequents we can derive, it might just make a few proofs slightly longer.

15 Completeness

We often have to deal with rules of inference that, taken as a whole, are not complete. That's because formal reasoning about arithmetic is inherently incomplete, as demonstrated by Gödel [1931] and we generally include integers in programming languages. However, in the case of the sequent calculus for propositional language we do obtain completeness as a consequence of invertibility and termination.

Theorem 3 (Completeness of the Sequent Calculus)
If $\Gamma \vdash \Delta$ *is valid, then* $\Gamma \vdash \Delta$ *is derivable.*

Proof: Assume $\Gamma \vdash \Delta$ is valid. In some arbitrary order we try to construct a derivation of $\Gamma \vdash \Delta$. We can always make progress while preserving the validity of all goal sequents until we reach a sequent consisting entirely of variables. Since it must be valid, one of the antecedent must be the same as one of the succedents and the identity rule applies. Therefore, all branches in the derivation can be closed off. □

We close with word on terminology. In today's lecture, we have used the word "proof" in different ways. For one, we have used it to describe a *formal object* that we can construct, communicate, and check (whether by hand or by machine). For another, we have used it in the usual mathematical sense: a *rigorous mathematical argument* that some claim is true, but not necessarily a formal object. Adding the word "formal" in many places is awkward, so we will try to stick to the convention to say *deduction* or *derivation* for a formal object, while we continue to use the word *proof* in the usual mathematical sense.

16 Addendum: Negation

There is a few common logical connectives we didn't cover. For example, negation. Fortunately, negation is straightforward and preserves the good properties of the sequent calculus: the rules are sound and invertible, and the overall system remains complete and decidable.

$$\frac{\Gamma, F \vdash \Delta}{\Gamma \vdash \neg F, \Delta}\ \neg R \qquad \frac{\Gamma \vdash F, \Delta}{\Gamma, \neg F \vdash \Delta}\ \neg L$$

Let's prove the soundness of the right rule.

all (Γ, F) true implies some Δ true (premise)
all Γ true (assumption)
...
some $(\neg F, \Delta)$ true (to show)

We distinguish two cases. If F is true, then we know by assumption that all (Γ, F) true. Therefore by the premise, some Δ true and hence some $(\neg F, \Delta)$ true. If F is false, then $\neg F$ is true and so, again some $(\neg F, \Delta)$ true. In either case, some $(\neg F, \Delta)$ true.

The remaining case of soundness and the two cases of invertibility follow similarly. Also, the premises of the two rules are smaller than the conclusion, so the decidability result and the overall completeness theorem continue to hold.

You might wonder about some other connectives, like logical equivalence also called bi-implication, $F \leftrightarrow G$. Instead of giving new rules, we can also consider it a *notational definition* and simply say

$$F \leftrightarrow G \quad \triangleq \quad (F \to G) \wedge (G \to F)$$

We can then expand the definition and just use the rule for implication and conjunction. In Assignment 1 you will explore something related.

References

Gerhard Gentzen. Untersuchungen über das logische Schließen. *Mathematische Zeitschrift*, 39:176–210, 405–431, 1935. English translation in M. E. Szabo, editor, *The Collected Papers of Gerhard Gentzen*, pages 68–131, North-Holland, 1969.

Kurt Gödel. Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme, I. *Monatshefte für Mathematik und Physik*, 38(1):173–198, 1931. English translation in Solomon Feferman, editor, *Kurt Gödel Collected Works, Vol I*, pages 144–195, Oxford University Press, 1986.