1 Background

In the early 1900s, the principal efforts towards foundations for mathematical and logical reasoning—mathematical logic—focused on developing sets of axioms. Axioms are similar to rules in having meta-variables that can be instantiated (to logical formulas, for example), but differ from rules in having no premises as such. Instead, the premises are encoded as the conditions of implications. For example, the axiom

\[ A \supset (B \supset A) \]

says that if A, then: if B, then A. More clearly, it can be read “if A and B, then A.” The first occurrence of A plays the role of a premise, as does the B; the second occurrence of A plays the role of the conclusion.

Such axioms (the above is an image from Gentzen’s thesis, based on work by Hilbert and Glivenko) were not user-friendly: the process of instantiating axioms doesn’t line up well with the reasoning that mathematicians actually do.

Gentzen’s work has several advantages over axiom systems like the above.

First, rules clearly distinguish the premises from the conclusion. I think this is a relatively small advantage, because with some practice the premises are easy to see.

Second, each of Gentzen’s rules of natural deduction has a close analogue to actual mathematical reasoning, whereas axiom systems necessarily include “administrative” axioms like \( A \supset (B \supset A) \). This closeness arises by modelling assumptions; while natural deduction’s particular style of modelling assumptions is somewhat awkward, Gentzen also developed sequent calculus which models assumptions in a different way.

Third, along with rules Gentzen developed derivations. Rather than searching through a proof of \( A \land B \) for the parts contributing to A and the parts contributing to B, a derivation proving \( A \land B \)
§1 Background

clearly separates the proof of $A$ from the proof of $B$. This compositionality makes derivations easier to deal with as mathematical objects, and—from a computational standpoint—as data structures. However, a disadvantage of derivations is that they are space-inefficient. (It was once fashionable for PL researchers to include large derivations in their papers; these usually required a figure in “landscape” orientation.) Moreover, they do not resemble traditional mathematical proofs. Some descendants of natural deduction, such as Fitch systems, adopt many of Gentzen’s rules (and his rule notation) but use a line-by-line proof format rather than derivations. Such descendants seem to be a good way to teach logic, and a good way to write careful and detailed proofs, but for our purposes we need to handle proofs as objects (data structures) in their own right.

2 Introduction and elimination

A feature of natural deduction (inherited by type systems!) is that the rules can be systematically designed, or at least systematically organized. The rules for each connective, such as $\land$ or $\supset$, can be categorized as (1) rules that introduce the connective, and (2) rules that eliminate the connective. (The number of rules in each category depends on the connective; one very common connective has zero elimination rules.)

In addition, the rules of natural deduction are orthogonal: the rules to introduce and eliminate $\land$ only mention $\land$, not $\supset$ or any other connective. This makes natural deduction, and type systems based on it, easier to extend: to incorporate a connective, we only have to design its introduction and elimination rules—which may not be easy but can be done without regard for any other connectives. The rules for each connective are a kind of “module” in the rule system.
§2 Introduction and elimination

3 Natural deduction

atomic formulas $P, Q$
formulas $A, B, C ::= P$

atomic formula

$\rightarrow$ implication

$\land$ conjunction (and)

$\lor$ disjunction (or)

$\forall a : \text{nat}. A$ universal quantification

$\exists a : \text{nat}. A$ existential quantification

$A$ true $A$ is true

\[
\frac{A \text{ true} \quad B \text{ true}}{A \land B \text{ true}} \land \text{Intro} \quad \frac{A \land B \text{ true}}{A \text{ true}} \land \text{Elim1} \quad \frac{A \land B \text{ true}}{B \text{ true}} \land \text{Elim2}
\]

$\exists [A \text{ true}]$

\[
\vdots
\]

\[
\frac{B \text{ true}}{(A \rightarrow B) \text{ true}} \rightarrow \text{Intro} \quad \frac{A \rightarrow B \text{ true} \quad A \text{ true}}{B \text{ true}} \rightarrow \text{Elim}
\]

Rule $\rightarrow$Elim is modus ponens, but rule $\rightarrow$Intro requires an assumption. The assumption “floats” above the subderivation in which it is available; it is available between the floating assumption $\exists [A \text{ true}]$ to its point of introduction, marked with the superscript $x$ next to the rule name. I have highlighted the $x$ in the rule, but we may have to work without highlighting.

This notation for assumptions is somewhat unfriendly: it’s easy to lose track of the scope of the assumption. But this notation may be closer to ordinary mathematical reasoning, and it’s historically important, so we’ll keep using it for now.

Disjunction usually causes more trouble than conjunction, and this certainly holds for natural deduction. However, there is a duality between conjunction and disjunction. In Boolean logic, you can get a feeling for this by comparing the truth table for AND with the truth table for OR, after swapping “true” and “false” in one of them. In natural deduction, this opposition between conjunction and disjunction shows itself in a similarity between the elimination rules for $\land$, which are $\land$Elim1/$\land$Elim2, and the introduction rules for $\lor$:

\[
\frac{A \text{ true}}{A \lor B \text{ true}} \lor \text{Intro1} \quad \frac{B \text{ true}}{A \lor B \text{ true}} \lor \text{Intro2}
\]

Observe that $\lor$Intro1 is $\land$Elim1 turned upside down, with $\land$ changed to $\lor$.

Sadly, flipping the introduction rule $\land$Intro upside down doesn’t give us a good $\lor$-elimination rule:

\[
\frac{A \lor B \text{ true}}{A \text{ true} \quad B \text{ true}} ??\lor \text{Elim}??
\]

This rule seems to have two conclusions. People sometimes write more than one conclusion as a concise notation for two rules with identical premises—we could combine $\land$Elim1 and $\land$Elim2, for example. But it’s certainly not true that, from “$A$ or $B$”, we should get $A$ and $B$. 
§3 Natural deduction

Instead, our \( \lor \)Elim rule will reason by cases. If \( A \lor B \) then either \( A \), or \( B \). As we can split a proof into cases according to a given grammar (“Case \( e = n \ldots \) Case \( e = (+ e_1 e_2) \)”), we can have subderivations with different assumptions.

\[
\begin{array}{c}
 x [A \text{ true}] \\ y [B \text{ true}] \\
 \vdots \quad \vdots \\
 A \lor B \text{ true} \\ C \text{ true} \\ C \text{ true} \\
\end{array}
\]
\[
(\lor \text{Elim}^x, y)
\]

To understand why the conclusion of \( \lor \)Elim should be \( C \text{ true} \), where \( C \) is any formula, it may help to think about case analysis in a (line-by-line) proof: we can case-analyze regardless of what our goal is. We might be trying to show that \( v = 0 \), or that \( v_1 = v_2 \), or that \( e \Downarrow v \); whatever the goal, case analysis works the same way. The only requirement is that each case must show the same goal: if we want to show \( v = 0 \) we need to show \( v = 0 \) assuming \( e = n \), and \( v = 0 \) assuming \( e = (+ e_1 e_2) \).

3.1 Example!

\[
\begin{array}{c}
 z [A_1 \lor A_2 \text{ true}] \\
 x [A_1 \text{ true}] \\
 y [A_2 \text{ true}] \\
\end{array}
\]
\[
\begin{array}{c}
 A_1 \lor A_2 \text{ true} \\
 A_1 \text{ true} \\
 A_2 \lor A_1 \text{ true} \\
 A_2 \lor A_1 \text{ true} \\
 A_2 \lor A_1 \text{ true} \\
\end{array}
\]
\[
(\lor \lor \text{Intro}^x) \\
(\lor \lor \text{Intro}^y) \\
(\lor \text{Elim}^x, y) \\
(\lor \text{Intro}^z)
\]

\[
(A_1 \lor A_2) \supset (A_2 \lor A_1) \text{ true}
\]
\[
(\supset \text{Intro}^z)
\]
§3  Natural deduction

(In 2018, this was roughly the dividing line between the January 25th and January 30th lectures.)

4  Natural deduction, extended

atomic formulas  P, Q
formulas   A, B, C ::= P  atomic formula
            | A ⊃ B  implication
            | A ∧ B  conjunction (and)
            | A ∨ B  disjunction (or)
            | ∀a : nat. A universal quantification
            | ∃a : nat. A existential quantification
            | True  truth

\[ A \text{ true} \] A is true

\[ \begin{array}{c}
A \text{ true}  \\
B \text{ true}
\end{array} \] \[ \frac{}{A \land B \text{ true}} \land \text{Intro} \]
\[ \begin{array}{c}
A \land B \text{ true}
\end{array} \] \[ \frac{}{A \text{ true} \lor B \text{ true}} \land \text{Elim1} \]
\[ \begin{array}{c}
A \land B \text{ true}
\end{array} \] \[ \frac{}{B \text{ true} \lor A \text{ true}} \land \text{Elim2} \]

\[ \begin{array}{c}
x[A \text{ true}]
\end{array} \] \[ \frac{}{(A \supset B) \text{ true}} \supset \text{Intro}^x \]
\[ \begin{array}{c}
A \supset B \text{ true}
\end{array} \] \[ \frac{}{A \text{ true} \supset B \text{ true}} \supset \text{Elim} \]

\[ \begin{array}{c}
x[A \text{ true}]
\end{array} \] \[ \frac{}{B \text{ true} \lor A \text{ true}} \lor \text{Intro1} \]
\[ \begin{array}{c}
B \text{ true}
\end{array} \] \[ \frac{}{A \lor B \text{ true} \lor B \text{ true}} \lor \text{Intro2} \]

\[ \begin{array}{c}
x[A \text{ true}]  \\
y[B \text{ true}]
\end{array} \] \[ \frac{}{A \lor B \text{ true} \lor C \text{ true} \lor C \text{ true}} \lor \text{Elim}^{x,y} \]

\[ \begin{array}{c}
A \text{ true}
\end{array} \] \[ \frac{}{A \text{ true} \lor B \text{ true}} \lor \text{TrueIntro} \]

\[ \begin{array}{c}
C \text{ true}
\end{array} \] \[ \frac{}{C \text{ true} \lor C \text{ true}} \lor \text{TrueIntro} \]

\[ \begin{array}{c}
A \land B \text{ true}
\end{array} \] \[ \frac{}{A \text{ true}} \land \text{Elim1} \]
\[ \begin{array}{c}
A \land B \text{ true}
\end{array} \] \[ \frac{}{B \text{ true}} \land \text{Elim2} \]

4.1 True

To design rules for the formula True, it may be helpful to view it as an “and” of nothing—a 0-ary conjunction. Since ∧ is a binary (2-ary) conjunction whose introduction rule ∧Intro has two premises, following that structure leads to the rule TrueIntro, which has zero premises. This seems consistent with an intuitive understanding of True: since True has no subformulas, its truth does not depend on the truth of the subformulas (unlike ∧ where A ∧ B is true only if A and B are true).

For the elimination rule(s), we can also argue by analogy to ∧. However, the argument feels a little different from the argument for the introduction rule.

\[ \begin{array}{c}
A \land B \text{ true}
\end{array} \] \[ \frac{}{A \text{ true} \lor B \text{ true}} \lor \text{Elim}^1 \]
\[ \begin{array}{c}
A \land B \text{ true}
\end{array} \] \[ \frac{}{B \text{ true} \lor A \text{ true}} \lor \text{Elim}^2 \]
§4 Natural deduction, extended

The trick is to find something about the above rules related to the number two. But the only thing related to two within $\land$Elim1 and $\land$Elim2 is the formula $A \land B$ itself: each rule has one premise. Instead, we must step back and observe that the number of elimination rules is two. Since $\land$ is 2-ary and True is 0-ary, this suggests that we should have zero elimination rules for True!

We can justify this by analogy to $\land$, but more intuitively: the formula $A \land B$ combines two facts ($A$ is true and $B$ is true), leading to two elimination rules, each extracting one of those two facts. But no facts are needed to justify True. Therefore, no facts can be extracted. Perhaps we could argue that True itself could be extracted:

\[
\begin{array}{c}
\text{True} \\
\text{True}
\end{array}
\]

But this rule is admissible (that is, redundant): we already have TrueIntro, which has the same conclusion. (Any rule with a premise that is identical to its conclusion is admissible.)

Allowing rules to have multiple conclusions is something that I'm trying to avoid, because it can be confusing, but allowing that leads to another argument for having no elimination rules. If we allow multiple conclusions then we can combine $\land$Elim1 and $\land$Elim2:

\[
\begin{array}{c}
A \land B \text{ true} \\
A \text{ true} \\
B \text{ true}
\end{array}
\]

Since this rule has two conclusions, TrueElim should have zero conclusions. But a rule with no conclusions isn’t a rule; even if we tried to bend the definition to allow it, it can’t conclude anything because it has no conclusion.

5 Harmony

The above arguments for designing rules for True have “intensional flavour”: we argued for our design based on existing internal features—our rules for $\land$—of the system (and then checked the resulting rules against our intuitive understanding of True).

This seems to run against Carnap’s aphorism, “In logic, there are no morals.” It suggests that we have some constraints around how the parts of the system fit together.

While I argued (in lecture, not written up here yet) by analogy to true as an identity element for $\land$, that wasn’t strictly necessary. Whether or not we believe that $\land$ comports with common usage of the word “and”, we can ask: If $\land$ is the 2-ary version of something, what is the 0-ary version? We can choose to call that 0-ary version “True”, or “tonk”, or even “False” (if we enjoy confusion).

One intensional quality-assurance tool doesn’t even need to compare the rules for different connectives: harmony checks that the introduction rules and elimination rules for a single connective match (are in harmony with) each other. Harmony has two parts:

- **Local soundness** holds when the results of applying elimination rules were already used in the introduction rule. On a high level, facts go into an introduction rule; the elimination rules should produce only those facts.

- **Local completeness** holds when the elimination rules can be used to recover all of the facts that went into the introduction.
§5 Harmony

Checking that rules satisfy these two parts of harmony gives us some protection against two possible design mistakes: neglecting to add a necessary rule, and adding a rule that is too powerful. Specifically:

- If local soundness is violated, either an elimination rule is wrong (it is producing something outside the “inputs” to the introduction rule), or we forgot an introduction rule.

- If local completeness is violated, either we forgot an elimination rule, or an introduction rule is wrong.

It’s probably easiest to grasp these ideas by considering some relatively clear mistakes in designing rules for conjunction.

\[
\begin{array}{c}
A \text{ true} \\
\hline
A \wedge B \text{ true} \\
\hline
B \text{ true} \\
\hline
A \wedge B \text{ true} \\
\hline
\wedge \text{Intro} \\
\hline
A \wedge B \text{ true} \\
\hline
\wedge \text{Elim1}
\end{array}
\]

Suppose the above two rules were our only rules for \( \wedge \). Local soundness holds, because our (only) elimination rule \( \wedge \text{Elim1} \) produces \( A \text{ true} \), which “went into” our use of \( \wedge \text{Intro} \):

\[
\begin{array}{c}
A \text{ true} \\
\hline
A \wedge B \text{ true} \\
\hline
B \text{ true} \\
\hline
A \text{ true} \\
\hline
A \wedge B \text{ true} \\
\hline
\wedge \text{Intro} \\
\hline
\wedge \text{Elim1}
\end{array}
\]

But local completeness fails, because our single elimination rule can’t recover the information \( B \text{ true} \). Checking local completeness ensures that we remember to include both elimination rules.

On the other hand, suppose we have both elimination rules but forget one of the premises of \( \wedge \text{Intro} \).

\[
\begin{array}{c}
A \text{ true} \\
\hline
A \wedge B \text{ true} \\
\hline
\wedge \text{Intro??} \\
\hline
A \wedge B \text{ true} \\
\hline
\wedge \text{Elim1} \\
\hline
A \text{ true} \\
\hline
A \wedge B \text{ true} \\
\hline
\wedge \text{Elim2}
\end{array}
\]

Checking local soundness will reveal the problem:

\[
\begin{array}{c}
A \text{ true} \\
\hline
A \wedge B \text{ true} \\
\hline
\wedge \text{Intro??} \\
\hline
B \text{ true} \\
\hline
\wedge \text{Elim2}
\end{array}
\]

The rule \( \wedge \text{Elim2} \) derives \( B \text{ true} \), but \( B \text{ true} \) didn’t go into our (wrong) introduction rule \( \wedge \text{Intro??} \), so \( \wedge \text{Elim2} \) is locally unsound with respect to \( \wedge \text{Intro??} \).

Local soundness and local completeness are not quite the same as soundness and completeness between different systems, but they are similar in that they depend on keeping something in a “fixed position” and comparing other stuff to the fixed thing: Ordinary soundness takes some system as ground truth, and checks that another system stays within that ground truth; ordinary completeness asks whether another system covers everything within that ground truth. Local soundness keeps the introduction rules stationary, and ensures that the elimination rules stay “within the scope” of the introduction rules. Local completeness also keeps the introduction rules stationary, and checks that the elimination rules can recover all of the information used by the introduction rules.
§5 Harmony

Let’s check that our rules for $\supset$ satisfy local soundness and local completeness.

\[
\begin{array}{c}
\text{x \{A true\}} \\
\vdots \\
B \text{true} \\
(A \supset B) \text{true} \\
\end{array} \quad \begin{array}{c}
\supset\text{Intro}^x \\
A \supset B \text{true} \\
A \text{true} \\
\end{array} \quad \begin{array}{c}
\supset\text{Elim} \\
B \text{true} \\
\end{array}
\]

5.1 Local soundness for $\supset$

For each elimination rule for $\supset$, we ask if that rule is locally sound. We have one elimination rule for $\supset$; is it locally sound?

\[
\begin{array}{c}
\text{x \{A true\}} \\
\vdots \\
B \text{true} \\
(A \supset B) \text{true} \\
\end{array} \quad \begin{array}{c}
\supset\text{Intro}^x \\
A \text{true} \\
\end{array} \quad \begin{array}{c}
\supset\text{Elim} \\
B \text{true} \\
\end{array}
\]

This is a little more complicated than $\land$, because $\supset\text{Elim}$ has a second premise A true. We can apply $\supset\text{Elim}$ only when A true. So the question becomes: did the information

“assuming A true [the other premise of $\supset\text{Elim}$], it holds that B true [the conclusion of $\supset\text{Elim}$]”

go into the application of $\supset\text{Intro}$? Yes, because the (only) premise of $\supset\text{Intro}$ derived B true under the assumption A true.

5.2 Local completeness for $\supset$

\[
\begin{array}{c}
\text{x \{A true\}} \\
\vdots \\
B \text{true} \\
(A \supset B) \text{true} \\
\end{array} \quad \begin{array}{c}
\supset\text{Intro}^x \\
A \text{true} \\
\end{array} \quad \begin{array}{c}
\supset\text{Elim} \\
B \text{true} \\
\end{array}
\]

For local completeness, we ask whether all the information going into $\supset\text{Intro}$ can be recovered using one of the elimination rules for $\supset$. Since there is only one elimination rule, we ask whether the information going into $\supset\text{Intro}$ can be recovered using $\supset\text{Elim}$. That information was

“assuming A true [the assumption within the premise of $\supset\text{Intro}$], it holds that B true [the premise of $\supset\text{Intro}$].”

(This is the same piece of information that we used in local soundness, but only because our rules really do satisfy local soundness and local completeness!) The argument for local completeness goes like this:

1. Assume we have a derivation of A $\supset$ B true whose concluding rule is $\supset\text{Intro}$.
2. Assume A true.
§5 Harmony

3. Our goal is to derive $B \text{ true}$.

4. Applying rule $\supset\text{Elim}$ to $A \supset B \text{ true}$ and $A \text{ true}$ gives $B \text{ true}$.

Note that if we had forgotten to write $\supset\text{Elim}$, we could not take the last step of this argument.

5.3 Local soundness for True

For each elimination rule for True, we ask if that rule is locally sound. We have no elimination rules for True, so there is nothing to check.

5.4 Local completeness for True

For local completeness, we ask whether all the information going into TrueIntro can be recovered using one of the elimination rules for True. But TrueIntro has no premises, so no information went into it. So there is nothing to check.

5.5 Local soundness for $\lor$

For each elimination rule for $\lor$, we ask if that rule is locally sound. We have only one elimination rule for $\lor$, but it is somewhat complicated:

$$
x \llbracket A \text{ true} \rrbracket \quad y \llbracket B \text{ true} \rrbracket \\
\cdots \\
A \lor B \text{ true} \quad \lor\text{Intro} \cdots \\
C \text{ true} \quad C \text{ true} \\
\lor\text{Elim}^x,y \quad C \text{ true}
$$

Since we have more than one introduction rule for $\lor$, we don’t know which of them was used to derive $A \lor B \text{ true}$, so I have included some “...”.

For $\supset$ we assumed the other (second) premise of $\supset\text{Elim}$. So here, we assume the other (second and third) premises of $\lor\text{Elim}$:

- Second premise of $\lor\text{Elim}$: “assuming $A \text{ true}$, then $C \text{ true}$.”
- Third premise of $\lor\text{Elim}$: “assuming $B \text{ true}$, then $C \text{ true}$.”

We also assume the first premise, the derivation of $A \lor B \text{ true}$ by one of the $\lor$-introduction rules. Consider cases of which introduction rule was used. Our goal is to show $C \text{ true}$.

- Case: $\lor\text{Intro1}$ was used.
  By inversion on $\lor\text{Intro1}$, $A \text{ true}$.
  We know (“Second premise of $\lor\text{Elim}$”) that, if $A \text{ true}$, then $C \text{ true}$. Since we know $A \text{ true}$, we know $C \text{ true}$.

- Case: $\lor\text{Intro2}$ was used.
  By inversion on $\lor\text{Intro2}$, $B \text{ true}$.
  We know (“Third premise of $\lor\text{Elim}$”) that, if $B \text{ true}$, then $C \text{ true}$. Since we know $B \text{ true}$, we know $C \text{ true}$.
§5  Harmony

5.6 Local completeness for ∨

Arguing local completeness for ∨ is tricky, because ∨Elim does not literally produce the information that went into the introduction rule. Instead, it splits into two cases, each working towards a common goal C true, where C may not be the same as A or B. Instead of trying to get the literal information A true (or B true), we must consider what we could deduce if we knew that either A true holds or B true holds.

1. Assume, as usual, A ∨ B true by one of the introduction rules for ∨; this is the first premise of ∨Elim.

2. Also assume, as usual, the remaining premises of ∨Elim.

3. Suppose that C′ is a formula that can be deduced assuming A, and can be deduced assuming B. Our goal is now to use ∨Elim to derive C′ true.

4. We want to apply ∨Elim. We don’t get to choose A and B; they are determined by the pre-existing derivation of A ∨ B true. However, we can choose C.

5. Let C be C′.

6. Either ∨Intro1 was used to derive A ∨ B true, or ∨Intro2 was used to derive it.

   • Case: ∨Intro1 was used.
     By inversion on ∨Intro1, A true.
     The second premise of ∨Elim is that C′ true under the assumption A true. That is, C′ true can be deduced from A true.
   
   • Case: ∨Intro2 was used.
     By inversion on ∨Intro2, B true.
     The second premise of ∨Elim is that C′ true under the assumption B true. That is, C′ true can be deduced from B true.

7. By rule ∨Elim, C′ true.

   The business about C′ is needed to reject a possible bug in ∨Elim: an insufficiently general conclusion. Consider this specialized version of ∨Elim, which can express our example in Section 3.1 but nothing else:

   \[ \begin{array}{c}
   x \ [A \ true] \\
   \vdots \\
   A \lor B \ true \\
   y \ [B \ true] \\
   \vdots \\
   B \lor A \ true \\
   \hline \\
   B \lor A \ true \\
   \end{array} \]  

If ∨Elim-swap were our only elimination rule for ∨, step 5—Let C be C′—would work only in the special case of C′ = B ∨ A. Since ∨Elim-swap cannot derive other conclusions, including—for example—((A ∨ B) ∨ A), our argument fails, as it should: ∨Elim-swap is incomplete because it doesn’t work for most possible C′.

Note that ∨Elim-swap is locally sound: it works for only one conclusion, but for that conclusion, it does not go beyond the premise of the introduction rule.
Exercise 1. Consider the connective tonk, due to A.N. Prior (1960), who argued (tongue-in-cheek) that tonk would succeed based on its “extreme convenience”. Translated to our notation, Prior gave two rules for tonk:

\[
\frac{A \ \text{true}}{(A \ \text{tonk} \ B) \ \text{true}} \quad \text{tonkIntro} \quad \frac{(A \ \text{tonk} \ B) \ \text{true}}{B \ \text{true}} \quad \text{tonkElim}
\]

Essentially, this connective steals one of the introduction rules for $\lor$ and one of the elimination rules for $\land$.

Argue that local soundness for tonk does not hold.
6 Quantifiers and falsehood

For quantifiers, we need some notation that substitutes a specific natural number for the quantified variable. Suppose we have

\( \forall a : \text{nat.} \ (\text{even}(a) \lor \text{odd}(a)) \)

and we want to know that the natural number 5 is either even or odd. We can get

\( (\text{even}(5) \lor \text{odd}(5)) \)

by looking for \( a \) (the quantified variable) throughout the body of the quantifier, and wherever we find \( a \), replacing it with 5.

\[ \forall a : \text{nat.} \ (\text{even}(a) \lor \text{odd}(a)) \]

We will write this use of substitution as

\[ [5/a](\text{even}(a) \lor \text{odd}(a)) \]

or more generally,

\[ [n/a]A = A \text{ with } n \text{ replacing each occurrence of } a \]

Substitution is a *meta-level* operation, like writing \( n_1 + n_2 \) in the rule eval-add. As with addition, it would be better to formally define what \([n/a]A\) means. However, the full definition of substitution has some “interesting” parts, which I don’t want to explain just yet.

If you’re curious about what the interesting parts might be, consider what should happen if we substitute 5 for \( b \) in the following:

\[ (b > 2) \land (\text{prime}(a) \supset (\exists b : \text{nat.} \ (b > a) \land \text{prime}(b))) \]

6.1 Mnemonic device

The PL research community has not converged on one standard notation for substitution. I have a number of reasons for preferring the notation above, which I won’t bore you with. A shortcoming of my preferred notation is that the order of \( n \) (the thing replacing the variable) and \( a \) (the variable being replaced) is not immediately clear. Here is a memory trick (more snobbily, a mnemonic device):

If we look for all occurrences of \( a \) throughout \( a \), and replace \( a \) with 5, we write that as

\[ [5/a]a = 5 \]

If we creatively reinterpret \( 5/a \) as a fraction and reinterpret substitution as multiplication, we can “cancel” the \( a \):

\[ \frac{5}{a} \cdot a = \frac{5 \cdot a}{a} = 5 \]
6 Quantifiers and falsehood

6.2 Substitution as renaming

We don’t have to substitute a constant natural number like 5. We could also substitute a variable.

\[ [a'/a] ((b > a) \land \text{prime}(b)) = (b > a') \land \text{prime}(b) \]

We could then substitute a constant for \( a' \):

\[ [2/a'] [a'/a] ((b > a) \land \text{prime}(b)) = [2/a'] ((b > a') \land \text{prime}(b)) = ((b > 2) \land \text{prime}(b)) \]

A nice feature of this notation is that such “double substitutions” have the same variable in the centre: “2 replaces \( a' \) which replaces \( a \), so 2 is replacing \( a' \)”, or (reading right to left) “replace \( a \) with \( a' \), then replace \( a' \) with 2”.
6.3 Natural deduction, extended again (2018–02–01)

atomic formulas \( P, Q \)

formulas \( A, B, C \) ::= \( P \) atomic formula
| \( A \rightarrow B \) implication
| \( A \land B \) conjunction (and)
| \( A \lor B \) disjunction (or)
| \( \forall a : \text{nat}. A \) universal quantification
| \( \exists a : \text{nat}. A \) existential quantification
| True truth
| False falsehood

\[
\begin{align*}
\text{A true} & \Rightarrow \text{A is true} \\
\vdots \\
B \text{ true} & \Rightarrow \text{B true} \\
\rightarrow \text{Intro} & \Rightarrow (A \rightarrow B) \text{ true} \\
\text{True} & \Rightarrow \text{True Intro} \\
\text{True} & \Rightarrow \text{True Elim} \\
A \text{ true} & \Rightarrow \text{A is true} \\
B \text{ true} & \Rightarrow \text{B is true} \\
\land \text{Intro} & \Rightarrow (A \land B) \text{ true} \\
A \text{ true} & \Rightarrow \text{A Elim1} \\
B \text{ true} & \Rightarrow \text{B Elim2} \\
\forall a : \text{nat}. B \text{ true} & \Rightarrow \forall \text{Intro} \\
\forall a : \text{nat}. B \text{ true} & \Rightarrow \forall \text{ Elim} \\
\rightarrow \text{Intro1} & \Rightarrow (A \lor B) \text{ true} \\
\rightarrow \text{Intro2} & \Rightarrow (A \lor B) \text{ true} \\
C \text{ true} & \Rightarrow \text{C Elim} \\
\text{False} & \Rightarrow \text{False Intro} \\
\text{False} & \Rightarrow \text{False Elim} \\
\exists a : \text{nat}. B \text{ true} & \Rightarrow \exists \text{Intro} \\
\exists a : \text{nat}. B \text{ true} & \Rightarrow \exists \text{ Elim} \\
\end{align*}
\]
6.3.1 Motivation for the new rules

Here we have added rules for $\forall$, $\exists$, and False. In class, I motivated the design through various symmetries:

- True is a 0-ary conjunction, $\land$ is a 2-ary (binary) conjunction, $\forall$ is an $\omega$-ary (infinitary) conjunction.
- False is a 0-ary disjunction, $\lor$ is a 2-ary (binary) disjunction, $\exists$ is an $\omega$-ary (infinitary) disjunction.

For $\forall$Intro, the 2 premises of $\land$Intro for the 2-ary $\land$ became “infinite premises”, one for each natural number.

\[
\begin{array}{c}
[0/a]B \text{ true} \\
[1/a]B \text{ true} \\
[2/a]B \text{ true} \\
\vdots \\
\end{array}
\]

Our actual $\forall$Intro rule represents this infinite set of premises as one premise with an assumption that $a$ is a natural number.

For $\forall$Elim, the 2 elimination rules of $\land$Elim for the 2-ary $\land$ became an infinite number of elimination rules.

\[
\begin{array}{c}
[\forall a : \text{nat.}\ B \text{ true}] \\
[\exists a : \text{nat.}\ B \text{ true}] \\
[\exists a : \text{nat.}\ B \text{ true}] \\
\vdots \\
\end{array}
\]

Since we cannot directly write all the rules in the infinite set \{\forallElim0, \forallElim1, \forallElim2, \forallElim3, \ldots\}, we replaced them with one rule that requires a specific $n$:

\[
\begin{array}{c}
[\forall a : \text{nat.}\ B \text{ true}] \\
[n/a]B \text{ true}
\end{array}
\]

If we choose $n = 0$, our $\forall$Elim does the same thing as $\forall$Elim0; if we choose $n = 1$, it does the same thing as $\forall$Elim1, and so forth.

Moving on to $\exists$, I waved my hands about duality between $\forall$ and $\exists$—which suggests that aspects of $\forall$Intro, should find their way into $\exists$’s elimination rule, and that aspects of $\forall$Elim should show up in $\exists$’s introduction rule. I also (perhaps more clearly) used our rules for 2-ary disjunction $\lor$ to inform the rules for the infinitary disjunction $\exists$.

Thus, the 2 introduction rules for the 2-ary $\lor$ became an infinite number of introduction rules

\[
\begin{array}{c}
[\exists a : \text{nat.}\ B \text{ true}] \\
[\exists a : \text{nat.}\ B \text{ true}] \\
[\exists a : \text{nat.}\ B \text{ true}] \\
\vdots \\
\end{array}
\]

which we coalesced into $\exists$Intro, noting that since the elimination rule for $\forall$ has a premise $n : \text{nat}$, duality between $\forall$ and $\exists$ suggests that the introduction rule for $\exists$ should also assume $a : \text{nat}$ within a premise.

Our elimination rule for $\exists$ has similar structure to our elimination rule for $\forall$. We arrived at that structure by, first, replicating the two $C \text{ true}$ premises of $\lor$ into an infinite set gives

\[
\begin{array}{c}
y_0[0/a]B \text{ true} \\
y_1[1/a]B \text{ true} \\
y_2[2/a]B \text{ true} \\
\vdots \\
\end{array}
\]

\[
\begin{array}{c}
[\exists a : \text{nat.}\ B \text{ true}] \\
C \text{ true} \\
C \text{ true} \\
C \text{ true} \\
\vdots \\
\end{array}
\]

\[
\begin{array}{c}
\exists\text{Intro}
\end{array}
\]

Second, we coalesced the infinite premises into a premise under the assumption $a : \text{nat}$.
§6 Quantifiers and falsehood

6.4 Historical notes (optional reading)

6.4.1 Assumptions and substitution

In Gentzen’s natural deduction system (“NJ”, essentially a misprint of “NI”), the assumptions $x[a: \text{nat}]$ are not written out. Moreover, Gentzen’s rules rename the variable in the quantifier (e.g. the $a$ in $\forall a : \text{nat}. B$) to a new variable $a'$: Instead of assuming $a : \text{nat}$ and $B$ true, Gentzen assumes $[a'/a]B$ true and calls the new variable $a'$ an Eigenvariable. (Eigen is German for “own”: the eigenvariable “belongs” to the particular application of the rule.)

This “extra” substitution is painless in Gentzen’s notation, because he wrote quantifiers differently: instead of $\forall a : \text{nat}. A$, with $\forall a : \text{nat}. \text{prime}(a)$ as a concrete example, he wrote (roughly) $\forall a : \text{nat}. A(a)$. Then the substitution of $a'$ for $a$ in the body $A(a)$ of the quantifier can be written as $A(a')$. If this course were entirely about natural deduction, I might have used Gentzen’s notation since it is more compact than square-bracket substitutions, but Gentzen’s notation is not suited for code in programming languages. By using square-bracket substitution, we can use the same notation consistently.

6.4.2 Judgment form

Gentzen did not write true in judgments or derivations. For example, Gentzen’s original $\land$Intro looked like this:

$$
\begin{array}{c}
\frac{A \quad B}{A \land B} \quad \land-I
\end{array}
$$

6.4.3 Included connectives

Apart from such notational differences, Gentzen’s NJ differs from our development in several ways:

- NJ does not include True. Without True, I think our FalseElim would seem less plausible.

- NJ does include negation, which still lurks on our horizon. Also, while Gentzen has our exact FalseElim, he calls it a negation elimination rule... because he does have a way to derive False through a negation-elimination rule.

6.4.4 Local soundness

For Gentzen [1934] §5.13, the introduction rules for a connective are its definition. He informally explained (without using the term) an example of local soundness (for $\lor$), noting that “we need not go into the ‘informal sense’ of the $\lor$-symbol”: local soundness is an intensional (internal) property of the rules. Later, the view that introduction rules define a connective became known as verificationism (we “verify” that $A \land B$ true is derivable by applying the introduction rule for $\land$). An opposing view, known as pragmatism, regards the elimination rules as “the” definition: the meaning of $A \land B$ true is what you can do with it.

7 Negation ($\neg$)

In Boolean logic, we may treat propositions as variables that are either true or false. Up to about 1900, this conception was predominant in mathematical logic; its first main challenge came from
§7 Negation (¬)

Brouwer (who, like Gentzen, wasn't a great human being) in 1907. By the 1930s, Brouwer’s intuitionism had gained some popularity; two of the four calculi of Gentzen's thesis are intuitionistic.

I won’t cover the philosophical side of intuitionism; for that, look at the Stanford Encyclopedia’s “Intuitionistic Logic” and follow the links in its second paragraph. I will discuss a narrow technical side of intuitionism—how it affects the specific rules of natural deduction—and the broader relationship of intuitionism and computing.

A bit is a binary digit, either 0 or 1. Thinking of the truth of a logical proposition as a bit-value is well suited to digital circuits, and to applications such as SAT solving. Being clearly either 0 or 1 is the essential feature of a bit, which may suggest that the truth of a logical proposition should be similarly clear:

- Consider an integer \( m \). If \( m \) is odd, \( m \) is not even; if \( m \) is even, \( m \) is not odd. Since every integer is either odd or even, every integer is either odd or not-odd. So it seems reasonable for “oddness” to be a Boolean property, either true (odd) or false (not-odd = even).

- Consider a Turing machine, whose configuration is given by a tuple including the number of its current state. Some states are designated as final states (meaning that the machine has halted). At any step of time, the machine is either in a final state (its current state is in the set of final states) or not in a final state (its current state is not in the set of final states), so the truth value of the proposition “the current state” is either true or false.

But undecidability spoils this clarity. We can ask whether a known integer (the current state) is part of a small finite set of integers (the set of final states), and the answer will be immediate. However, not all questions have answers.

- Consider a Turing machine, whose configuration is given by a tuple including the number of its current state. If the machine is allowed to run indefinitely, will it reach a halting state?

While the Boolean formula \( \text{halts}(H) \) looks like the Boolean formula \( \text{in-a-final-state}(H) \), the truth value of \( \text{halts}(H) \) is much less clear, because this is an undecidable problem. For specific Turing machines we can give an answer. For example:

- If the initial state of \( H \) is a final state, then \( \text{halts}(H) \) is true.
- If the Turing machine has only one state, which is not a final state, then \( \text{halts}(H) \) is false.

Being able to answer specific instances of the general question does not mean the problem is decidable. In fact, we can answer the question for a large number of Turing machines, by simulating the machine for a few million steps and checking if it halts. If it halts, then \( \text{halts}(H) \) is true. If it doesn’t halt, we haven’t answered the question. For some machines that never halt, we can say definitively that they don’t halt by providing a proof: “The machine has only one state; therefore, it will always be in that state; since that state is not a final state, the machine never halts.”

In intuitionistic logics, statements of existence must be backed with a witness to the existence. If I claim that the machine halts, I must show you when it halts. If I claim that the machine doesn’t halt, I must prove that to you. A proposition is true if and only if you have a proof.

Thus, in intuitionistic logics, a statement like

\[
\text{either } \text{halts}(H) \text{ or } \neg \text{halts}(H)
\]
is interpreted as

\[ \text{either there exists a proof of } \text{halts}(H) \text{ or there exists a proof of } \neg \text{halts}(H) \]

Since the halting problem is undecidable, the statement

\[ \text{for all } H, \text{ either there exists a proof of } \text{halts}(H) \text{ or there exists a proof of } \neg \text{halts}(H) \]

is not true. (Whether we also say the statement is false is, I think, a matter of taste.) So the statement

\[ \text{for all propositions } P, \]

\[ \text{either there exists a proof of } P \text{ or there exists a proof of } \neg P \]

is false.

In natural deduction, the truth of a formula \( A \) is determined by whether there exists a derivation of \( A \) true. Since natural deduction (through the atomic formulas that I deliberately haven’t specified) can talk about pretty much anything, we should not expect (after we add negation)

\[ \text{for all formulas } A, \]

\[ \text{either there exists a natural-deduction derivation of } A \text{ true} \]

\[ \text{or there exists a natural-deduction derivation of } (\neg A) \text{ true} \]

This means we should not want a rule “LEM” (Law of the Excluded Middle):

\[
\frac{A \lor \neg A \text{ true}}{\text{LEM?}}
\]

If our mental model includes undecidable questions (and it should), including this rule would make natural deduction unsound with respect to that mental model.

Rejecting LEM does not reject negation entirely! We can still have the Law of Contradiction:

\[
\frac{A \text{ true} \quad (\neg A) \text{ true}}{\text{LoC}}
\]

In fact, LoC was Gentzen’s elimination rule for \( \neg \). What was Gentzen’s introduction rule for \( \neg \)?

\[
\frac{x[\text{A true}]}{\text{False true} \quad \neg \text{Intro}^x}
\]

Both LoC and \( \neg \text{Intro} \) are laws permitted in Brouwer’s intuitionistic mathematics.

Classical (not intuitionistic) logics do include LEM. In addition to NJ (which is essentially the same as our rules), Gentzen presented NK (for klassische), which is identical to NJ but adds the rule LEM.

### References