Chapter 0

Boolean expressions and Theorems

In Logic—the "practice", as opposed to the (meta)theory—the primary job is to *calculate*[†] "theorems". Theorems are some very special Boolean expressions—or *well-formed-formulas*[‡]—as logicians like to call them.

So before we learn how to calculate formulas *that are theorems*, it is prudent to learn how to calculate formulas in general.

Now the term to "calculate formulas" (or theorems) is deliberately chosen to emphasize the fact that for both formulas in general, and theorems in particular, there is a *methodology of calculation*—or *calculus*, for each—which allows us to *generate* or *derive* (e.g., write on a piece of paper) all formulas *correctly* and also all theorems, *correctly.*[§]

Indeed, a computer program can be written instructing a computer to generate and print all well-formed-formulas in a systematic way (again, assuming the computer would be operating for ever on this task, and that sufficient paper would be available to print all these formulas on . . .). A similar program can be written to instruct a computer to generate all theorems of Propositional Logic and all the theorems of Predicate Logic.

1. *Formula calculus*

So how does one calculate formulas?

We start with the basic building blocks, which *collectively* form what is called the *alphabet* (for formulas). Namely,

[†]This is why Logic is often referred to as a "calculus", be it of "Propositional" or "Predicate" type. Still, *most* logicians say that they *deduce*, or *derive*, or *prove* theorems, obviously preferring any of the latter three verbs to "calculate".

[‡]Or just "formulas".

[§]This we can do *in principle only*, for it requires that we have infinite time at our disposal: Theorems, and *a fortiori* formulas, form an infinite set.

A1. *Symbols* for *variables*, the "Boolean variables". These are p, q, r, with or without primes or subscripts (i.e., p', q_{13}, r'''_{51} are also symbols for variables).

A2. Two *symbols* for Boolean constants, namely *true* and *false*.

A3. Brackets, namely, (and).

A4. "Boolean connectives", namely, the symbols listed below, separated by commas

$$\neg, \wedge, \vee, \Rightarrow, \equiv$$

(1) Even though I say very emphatically that p, q, r, etc., and also *true* and *false* are just *symbols*,†—the former for variables, the latter for constants—yet, I will stop using the qualification "symbols", and just say "variables" and "constants". This entails an agreement: I always *mean* "symbols", I just don't say it.

(2) Each of *true* and *false* are "single" symbols—*not* words of symbols (made of simpler symbols such as "t, r, u, e, f, a, l, s"). Some logicians emphasize this—and the "meaningless" character of these symbols—by using, instead of the symbols *true* and *false*, the symbols $\top$ and $\bot$ respectively.

Note. In class it is hard to write italics or boldface type on the blackboard, so I write these symbols with underlining instead, as <u>true</u> and <u>false</u> respectively.

1.1 Definition. (Strings or **expressions)** An *ordered* sequence of symbols from an alphabet, written adjacent to each other *without* any visible separators (space, comma, or the like) is called a *string* (also "word" or *expression*) *over* the alphabet. E.g., *aabba* is a string of symbols over the alphabet $\{a, b, c, 0, 1, 2, 3\}$ (note that you don't have to use all the alphabet symbols in any given string, and, moreover, repetitions are allowed). "Ordered" means that $aab \neq aba$.

We denote arbitrary strings over the alphabet **A1.**–**A4.** by "string variables", i.e., names that stand for arbitrary‡ or specific§ strings.

These string variables—by agreement—will be denoted by upper case letters A, B, C, D, E with or without primes or subscripts. In particular, since Boolean expressions (and theorems) are strings, this naming is valid for this special case too. □

We are ready to define

1.2 Definition. (Formula-calculation or **formula parse)**

A *formula-calculation* (or formula-parse) is any finite (ordered) sequence of strings that we may write respecting the following three requirements:

1. At any step we may write any symbol from **A1.** or **A2.**

2. At any step we may write the string $(\neg A)$, *provided* we have already written the string A

†More emphatic logicians say "*meaningless* symbols".
‡E.g., "let A be any string".
§E.g., "let A stand for $(\neg(p \wedge q))$".

3. At any step we may write any of the strings $(A \wedge B)$, $(A \vee B)$, $(A \Rightarrow B)$, $(A \equiv B)$, *provided* we have already written the strings A and B.

□

1.3 Example. In the *first* step of any formula calculation only the requirement (1) of Definition 1.2 is applicable, since the other two require the existence of prior steps. Thus in the first step we may *only* write a variable or a constant. In all other steps, *all* the requirements (1)–(3) are applicable.

Here is a calculation (the comma is not part of the calculation, it just separates strings written in various steps):

$$p, \mathit{true}, (\neg \mathit{true}), q$$

Ensure that the above obeys Definition 1.2.
Here is a more "interesting" one:

$$p, q, (p \vee q), (p \wedge q), \big((p \vee q) \equiv q\big), \big((p \wedge q) \equiv p\big), \Big(\big((p \vee q) \equiv q\big) \equiv \big((p \wedge q) \equiv p\big)\Big)$$

1.4 Definition. (Boolean expressions or **WFFs)** A string A over the alphabet **A1.–A4.** will be called a Boolean expression or a *well-formed-formula* ***iff*** [†] it is a string written at some step of some formula-calculation.

The set of Boolean expressions we will denote by **WFF**. □

1.5 Remark. (1) The idea of presenting the definition of formulas as a "construction" or "calculation" goes as far back as—at least—[1, 3].

(2) We used, in the interest of user-friendliness, active or "procedural" language in Definition 1.2[‡] (i.e., that *we* may *do* this or that in each step). A mathematically more austere (hence "colder"(!)) approach that does not call upon anyone to write anything down—and does not speak of "steps"—would say *exactly the same thing as Definition 1.2* rephrased as follows:

► A *formula-calculation* (or formula-parse) is any finite (ordered) sequence of strings $A_1, A_2, \ldots, A_n$ such that—for all $i = 1, \ldots, n$—A_i is one of:

1. Any symbol from **A1.** or **A2.**
2. $(\neg A)$, *provided* A is the same string as some A_j, where $1 \leq j < i$.
3. Any of the strings $(A \wedge B)$, $(A \vee B)$, $(A \Rightarrow B)$, $(A \equiv B)$, *provided* A is the same string as some A_j, where $1 \leq j < i$ and B is the same string as some A_k, where $1 \leq k < i$ (it *is* allowed to have $j = k$, if needed). ◄

[†] If and only if.
[‡] Exactly as [3] does.

(3) There is an advantage in the procedural formulation 1.2. It makes it clear that we build formulas in stages (or steps), each stage being a calculation step.

In each step where we apply requirement (2) or (3) of 1.2 *we are building a more complex formula from simpler formulas.*

Moreover, *we are building a formula from* ***previously built*** *formulas.*

These last two remarks are at the heart of the fact that we can prove properties of formulas by induction on the number of steps (stages) it took to build them (see next section).

The concluding remark above motivates an "inductive" or "recursive" definition of formulas, which is the favourite definition in the "modern" literature, and we should become familiar with it:

1.6 Definition. (WFF) The set of well-formed-formulas is the ***smallest*** set of strings, **WFF**, that satisfies

(1) *All Boolean variables* are in **WFF**, and so are the symbols *true* and *false*.

(2) If A and B are any strings in **WFF**, so are the strings $(\neg A)$, $(A \wedge B)$, $(A \vee B)$, $(A \Rightarrow B)$, $(A \equiv B)$.

□

1.7 Remark.

(a) Why "inductive"? Because item (2) in 1.6 defines the concept "formula" in terms of (a *smaller, or earlier, instance of* [†]) "itself": It says, in essence, "… $(A \vee B)$ is a **formula**, *provided we know that* A and B are **formulas** …"

(b) Part (1) in 1.6 defines the most basic, most trivial formulas. This part constitutes what we call the "*Basis*" of the inductive definition, while part (2) is called the *inductive*, or *recursive*, part of the definition.

(c) 1.6 and 1.4 say the same thing, looking at it from opposite ends: Indeed, suppose that we want to establish that a given string D is a formula. If we are using 1.4, we will try to build D via a formula calculation, starting from "atomic" ingredients (variables and constants) and building as we go successively more complex formulas until finally, in the last step, we obtain D.

If on the other hand we are using 1.6, we are working backwards (and build a formula-calculation in reverse!). Namely, if D is not atomic, we try to

[†]Each of A and B are parts—subexpressions or substrings—of $(A \vee B)$, so they are "smaller" than the latter. They are also "earlier" in the sense that we must already *have them*—i.e., *know* that they are formulas—in order to proclaim $(A \vee B)$ a formula.

see—from its form—what was the *last* connective applied. Say, it was $\Rightarrow$, that is to say, D is $(A \Rightarrow B)$ for some strings A and B. Taking *each* of A and B in turn as (a new, smaller) "D" we repeat this process—of verifying that D is a formula. And so on. This is a terminating process since the new strings we obtain (for testing) are always smaller than the originals.

Of course, I did not *prove* here that the two definitions define the *same* set **WFF**.[†] But they do!

Technically, the term "smallest" is crucial in 1.6 and it corresponds to the similarly emphasized "iff" of 1.4. A proof that the two definitions are equivalent is beyond our syllabus. In Chapter 2 however I give a proof—solely for the benefit of the ambitious reader—in the analogous case where the *inductive* and *calculational* [‡] definitions of "theorem" rather than "formula" are compared.

In the course of a formula-calculation (1.2), some formulas we write down without looking back. (Step of type (1)). Some others we write down by combining via one of the connectives $\wedge, \vee, \Rightarrow, \equiv$ *two* formulas A and B already written, or by prefixing *one* already written formula, C, by $\neg$.

In terms of the construction by stages then, the formula built in this last stage had as *immediate predecessors* A and B in the first case, or just C in the second case.

One can put this elegantly via the following definition:

1.8 Definition. (Immediate predecessors) None among the constants *true* and *false*, and among the variables, have any immediate predecessors.

Any of the formulas $(A \wedge B), (A \vee B), (A \Rightarrow B), (A \equiv B)$ have A *and* B as immediate predecessors.

$(\neg A)$ has exactly one immediate predecessor: A.

Sometimes we use the acronym "i.s." for "immediate predecessor". □

1.9 Remark. (Priorities) In practice, too many brackets make it hard to read complicated formulas. Thus, texts (and other writings in Logic) often come up with

An agreement on how to be sloppy, and, yet, get away with it.

This agreement tells us what brackets are *redundant*—and hence can be removed—in a formula written according to Definitions 1.4 and 1.6, *while preserving its meaning*:

1. Outermost brackets are redundant.

[†] I only handwaived to that effect, arguing that for any string D in **WFF**, 1.4 builds a calculation the normal way, while 1.6 builds it backwards. I conveniently swept under the rug the case where D is *not* in **WFF**, i.e., is not correctly formed.

[‡] A theorem-calculation—unlike formula calculation—has a special name: "Proof".

2. Any other pair of brackets is redundant if its presence can be understood from the *priority*, or *precedence*, of the connectives. Higher priority connectives *bind* ***before*** lower priority ones. The order of priorities (decreasing from left to right) is:
$$\neg, \wedge, \vee, \Rightarrow, \equiv \tag{$*$}$$

3. In a situation like "$\ldots \diamond (A) \diamond \ldots$"—where $\diamond$ is any connective listed in $(*)$ above—the right $\diamond$ acts before the left. As we say, *all connectives are right associative.*

It is important to emphasize:

(a) This "agreement" results into a shorthand notation. Most of the strings depicted by this notation are ***not*** correctly written formulas, but this is fine: Our agreement allows us to decipher the shorthand and *uniquely recover* the correctly written formula we had in mind.

(b) I gave above the convention followed by 99.9% of the writings in Logic, and by almost the totality of Programming Language definitions (when it comes to "Boolean expressions" or "conditions").

The text differs in $\wedge$ versus $\vee$ (check it!) and in the associativities. It gives left associativity to some and right associativity to others.

(c) The agreement on removing brackets is a *syntactic* agreement.

In particular, right associativity says simply that, e.g., $p \vee q \vee r$ *is shorthand for* $(p \vee (q \vee r))$ *rather than* $((p \vee q) \vee r)$.

However, *no claim is either made or implied* that $(p \vee (q \vee r))$ and $((p \vee q) \vee r)$ have "different meanings". Indeed, it is trivial to check that the two evaluate identically in *every state*.

1.10 Example. p means p.

$\neg p$ means $(\neg p)$.

$p \Rightarrow q \Rightarrow r$ means $(p \Rightarrow (q \Rightarrow r))$.

If I want to simplify $((p \Rightarrow q) \Rightarrow r)$, then $(p \Rightarrow q) \Rightarrow r$ is as simple as I can get it.

$\neg p \wedge q \vee r$ is short for $(((\neg p) \wedge q) \vee r)$.

If in the previous I wanted to have $\neg$ act last, and $\vee$ to act first, then the minimal set of brackets necessary is: $\neg(p \wedge (q \vee r))$.

A connection with things to come. Any set of "rules" that tell us how to correctly write down strings constitute a so-called "grammar". *Formal language* theory studies grammars, the sets of strings they define (called "formal languages"), and the procedures (or "machines") that are appropriate to parse these strings.

In COSC2001 one learns about formal languages. A student of COSC2001 would quickly realize that Definition 1.6 is, in effect, a definition of a grammar for the "language" (i.e., set of strings) **WFF**. He or she would utilize a neat notation,[†] such as

$$
\begin{aligned}
&E ::= A \Big| (E \wedge E) \Big| (E \vee E) \Big| (E \Rightarrow E) \Big| (E \equiv E) \Big| (\neg E)\\
&A ::= \mathit{true} \Big| \mathit{false} \Big| p \Big| q \Big| r \Big| p' \Big| \ldots
\end{aligned}
$$

to effect the definition, where E stands for (Boolean) **E**xpression, A for **A**tom, "::=" is read "is defined to be" and "$\Big|$" is read "or", separating alternatives in an "is defined to be"-list.

Thus, the first line, in English, says "A (Boolean) expression **is defined to be** an atom, **or** "(" followed by an expression, followed by "$\wedge$" followed by an expression followed by ")", **or**, etc."

The second line defines "atom" as any of the constants or the variables (note the separating "**or**"s).

2. Induction on WFF Some easy properties of WFF

Suppose now that we want to prove that every $A \in$ **WFF**[‡] has a "property" $\mathcal{P}$.

First off, we write "$\mathcal{P}(A)$ is true"—or just "$\mathcal{P}(A)$"—as shorthand for "A has property $\mathcal{P}$".

Now, since formulas A are built in stages, the sensible—indeed obvious—thing to do towards proving

$$\mathcal{P}(A), \text{ for all } A \in \mathbf{WFF} \tag{1}$$

is to prove

I1. *All* formulas that we can build in stage 1 have property $\mathcal{P}$,

I2. Property $\mathcal{P}$ is *preserved* (or, *propagates*) by the building-operations ((2) and (3) of 1.2).

In other words, *if the building blocks used in an application of (2) or (3) have property $\mathcal{P}$, then so does the string we wrote down as a result of such application.*

Instead of "$\mathcal{P}$ propagates with an application of (2) and (3)" we can say that "a formula *inherits* a property $\mathcal{P}$ if all its immediate predecessors (Definition 1.8) have $\mathcal{P}$".

[†] Known as "BNF notation", or Backus-Naur-Form notation.

[‡] "$x \in y$" is shorthand for the claim "x is a member of the set y".

Intuitively,[†] the above guarantees that no matter what stage we look at, the string produced at that stage has $\mathcal{P}$ (since the strings written in stage 1 have $\mathcal{P}$ by **I1**; the ones written in stage 2 inherit $\mathcal{P}$ from stage 1[‡]; the ones written in stage 3 inherit $\mathcal{P}$ from 1 and 2;...; the ones written in stage $n+1$ inherit $\mathcal{P}$ from the ones in stages 1 to n;...)

The procedure **I1.–I2.** for proving (1) is called *Induction on Formulas.* It is an immediate consequence of our preceding remarks that it can be rephrased:

To prove (1), prove instead,

1. All atomic Boolean expressions have $\mathcal{P}$—this step is called the *Basis.*
2. If A has $\mathcal{P}$, then so does $(\neg A)$
3. If A and B have $\mathcal{P}$, then so do $(A \wedge B)$, $(A \vee B)$, $(A \Rightarrow B)$ and $(A \equiv B)$.

The "if-part" in steps 2. and 3. above is called the *Induction Hypothesis*, or just the "I.H."

Let us apply induction on formulas to prove a few properties of formulas.

2.1 Theorem. *Every Boolean expression A has the same number of left and right brackets.*

Proof. We prove the property of A claimed in the theorem using induction on formulas, as formulated by 1.–3. above.

1. $p, q, r, \ldots$ as well as *true* and *false* have 0 left and 0 right brackets. We are OK.
2. If A has m left and m right brackets, then the property is inherited by $(\neg A)$: $m+1$ left and $m+1$ right brackets.
3. If A m left and m right brackets and B nm left and n right brackets, then each of $(A \wedge B)$, $(A \vee B)$, $(A \Rightarrow B)$ and $(A \equiv B)$ have $n+m+1$ left and $n+m+1$ right brackets. Done.

□

2.2 Corollary. *Any nonempty proper prefix of a Boolean expression A has more left than right brackets.*

Proof. Induction on A.

[†]There are infinitely many steps in the argument of this paragraph, so it *cannot* be a proof. It is just a "plausibility" argument of the "domino effect" type, such as the ones found in elementary texts in defense of induction on the integers. Essentially, **I1.–I2.** *is* induction on the integers, for stages are positive integers. Translating makes this clear: **I1.**, the "basis", says that we must verify that everything built in stage $s = 1$ has property $\mathcal{P}$. **I2.** says that *if everything we construct in stages s satisfying* $1 \le s \le n$ *has* $\mathcal{P}$ [the Induction Hypothesis or "I.H."], then so does everything constructed in stage $s = n+1$ [the induction step].

[‡]If not atomic. Atomic ones are covered by **I1**.

1. For the basis we observe that none of the atomic formulas has any nonempty proper prefixes, so we are done without lifting a finger.

2. Case of $(\neg A)$, on the I.H. that A has the property stated in the corollary. Well, let's check the nonempty proper prefixes of $(\neg A)$. These are (quotes not included, of course):

   (a) "(". OK, by inspection.

   (b) "$(\neg$". Ditto.

   (c) "$(\neg C$", where C is a nonempty proper prefix of A. By I.H., if m is the number of left and n the number of right brackets in C, then $m > n$. But the number of left brackets of "$(\neg C$" is $m + 1$. Since $m + 1 > n$, we are done.

   (d) "$(\neg A$". By 2.1, A has, say, k left and k right brackets. We are OK, since $k + 1 > k$.[†]

3. Case of $(A \circ B)$—where "$\circ$" is any of $\wedge, \vee, \Rightarrow, \equiv$—on the I.H. that A *and* B have the property stated in the corollary. Well, let's check the nonempty proper prefixes of $(A \circ B)$. These are (quotes not included, of course):

   (i) "(". OK, by inspection.

   (ii) "$(C$", where C is a nonempty proper prefix of A. By I.H., if m is the number of left and n the number of right brackets in C, then $m > n$. But the number of left brackets of "$(C$" is $m + 1$. OK.

   (iii) "$(A$". By 2.1, A has, say, k left and k right brackets. We are OK, since $k + 1 > k$.

   (iv) "$(A\circ$". The accounting exercise is exactly as in (iii). OK.

   (v) "$(A \circ C$", where C is a nonempty proper prefix of B. By 2.1, A has, say, k left and k right brackets. By I.H., C has, say, m left and r right brackets, where $m > r$. Thus, "$(A \circ C$" has $1 + k + m$ left and $k + r$ right brackets. OK!

   (vi) "$(A \circ B$". Easy.

□

The following tells us that once a formula has been written down correctly, there is a unique way to understand the order in which connectives apply.

2.3 Theorem. (Unique Readability) *For any formula A, its immediate predecessors are uniquely determined.*

[†]The I.H. was not needed in this step. Indeed, it was only used in the previous step.

Proof. Obviously, if A is atomic, then we are OK (nothing to prove, for such instances of A have *no* i.s.). Moreover, no A can be seen (written) as both atomic and non atomic.† The former do *not* start with a bracket, the latter do.‡

Suppose that A is not atomic. Is it possible ***to build*** this string in more than one ways?

Can A have ***two different sets of i.s.***, as listed below? (Below, when I say "we are OK" I mean that the answer is "no", as the theorem claims.)

1. $(\neg C)$ and $(\neg D)$? Well, if so, C is the same string as D (why?), so in this case we are OK.

2. $(\neg C)$ and $(D \circ E)$, where $\circ$ is any of $\wedge, \vee, \Rightarrow, \equiv$? Well, no (which means we are OK in this case too). Why "no"? For if $(\neg C)$ and $(D \circ E)$ are identical strings (they are, supposedly, two ways to read A, remember?), then "$\neg$" must be equal (same symbol) to the ***first symbol*** of D. Now the first symbol of D is one of "(" or an atomic symbol. None matches "$\neg$".

3. $(C \circ D)$ and $(E \diamond G)$, where $\circ$ and $\diamond$ are any of $\wedge, \vee, \Rightarrow, \equiv$ (possibly the same symbol) and either C and E are different strings, or D and G are different strings, or both? Well, nope!

   (i) If C and E are different, then, say, C is a proper prefix of E (of course, C is nonempty as well (why?)). By 2.2, C has more left brackets than right, but—being also a formula—it has the *same* number of left and right brackets (by 2.1). Impossible! The other case, E being a proper prefix of C instead, is equally impossible.

   (ii) If C and E match, then $\circ$ and $\diamond$ match. But then D and G are identical strings for the same reason as in 1. above.

   Pause. What *was* the resaon in 1. above?

Having answered "no" in all cases, we are done. $\square$

3. Inductive definitions on formulas

Now that we know (2.3) that we can decompose a formula uniquely into its constituent parts, we are comfortable with defining functions (more generally "concepts") on formulas (Boolean expressions) *by induction—or recursion—on formula structure*, or as we often say, "by induction—or recursion—on formulas".

Here is a familiar example, the definition of the state function "s" on an arbitrary formula A, once the state has been defined for all variables (below I

†So we cannot be so hopelessly confused as to think at one time that A has no i.s. and at another time that it does.

‡Easy exercise to do (by induction on formulas). Maybe I'll assign it! Prove that the first symbol of any formula A is one of (1) a variable (2) *true*, (3) *false*, (4) a left bracket.

use "p" generically, so that "$s(p)$" refers to all variables p that a state has been defined).

$$
\begin{aligned}
s(p) &= \text{ whatever we assigned, } t \text{ or } f\\
s(\mathit{true}) &= t\\
s(\mathit{false}) &= f\\
s\Big((\neg A)\Big) &= \overline{s(A)}\\
s\Big((A \wedge B)\Big) &= s(A) \cdot s(B)\\
s\Big((A \vee B)\Big) &= s(A) + s(B)\\
s\Big((A \Rightarrow B)\Big) &= s(A) \rightarrow s(B)\\
s\Big((A \equiv B)\Big) &= \Big(s(A) = s(B)\Big)
\end{aligned}
$$

Why the above works is clear at the intuitive level: Lack of ambiguity in decomposing "C" uniquely as one of $(\neg A), (A \wedge B), (A \vee B), (A \Rightarrow B), (A \equiv B)$ allows us to know how to compute ***a unique*** answer for $s(C)$. The "why", at the technical level, is beyond our reach.

Here is an example of a definition of a "concept" regarding formulas, by induction on formulas.

3.1 Definition. (Occurrence of a variable) We define "p occurs in A" and "p does not occur in A" simultaneously.

Occ1. (Atomic) p *occurs* in p. It does *not* occur in any of r, *true*, *false*—where r is a variable distinct from p.

Occ2. p occurs in $(\neg A)$ iff it occurs in A.

Occ3. p occurs in $(A \circ B)$—where $\circ$ is one of $\wedge, \vee, \Rightarrow, \equiv$ —iff it occurs in A or B or both.[†]

□

3.2 Remark. We wanted to be user-friendly (which often means "sloppy") in the first instance, and said that the above defines a "concept": Occurs, does not occur. In reality, all such "concepts" that we may define by recursion on formulas are just *functions*.

For example, the "concept" in question here is captured by the function "$occurs(p, A)$", where $occurs(p, A) = 0$ means "p occurs in A", and $occurs(p, A) = 1$ means "p does *not* occur in A".

[†]Needless to say, by the "iff", it does *not* occur exactly when it does not occur in A *and* does not occur in B.

We conclude this section with the definition of substitution in a Boolean expression, and with the proof of two important properties of substitution.

Intuitively, the symbol "$A[p := B]$" means the result of the replacement of the variable p in A—in all its occurrences—by the formula B. We may think of this operation as defining a function from formulas to strings:

Input: A, p, B, output: the string $A[p := B]$.

3.3 Definition. (Substitution in BE) *Below "=" means equality of strings.* The definition states the obvious: (1) Handles the basis cases in the trivial manner, and (2) when A is actually build from i.s., it says that we substitute into each i.s. first, and then apply the connective.

$$A[p := B] = \begin{cases} B & \text{if } A = p \\ A & \text{if } A = r \text{ (where } p \neq r\text{), or} \\ & \quad A = \textit{true}, \text{ or } A = \textit{false} \\ \Big(\neg(C[p := B])\Big) & \text{if } A = (\neg C) \\ \Big((C[p := B]) \circ (D[p := B])\Big) & \text{if } A = (C \circ D) \end{cases}$$

where $\circ$ is one of $\wedge, \vee, \Rightarrow, \equiv$.

As in [2], when we write Boolean expressions in *least parenthesized form* (i.e., sloppily), then "$[p := \dots]$" has highest priority. Thus, e.g., $A \vee B[q := E]$ means $A \vee (B[q := E])$. □

We state and prove two easy and hardly unexpected properties of substitution:

3.4 Proposition. † *For any formulas A and B and variable p, $A[p := B]$ is a (well-formed!) formula.*

Proof. Induction on A, keeping an eye on Definition 3.3.

In the *Basis* we get p or A (see 3.3!), a formula in either case.

The *induction steps*:

For $\neg$: Does A, that is, $(\neg C)$ *inherit* the property from its i.s. C? You bet. See the 3rd case in the definition above: If $C[p := B]$ is a formula, then so is $\Big(\neg(C[p := B])\Big)$ by 1.6.

For $\circ$: Does A, that is, $(C \circ D)$ *inherit* the property from its *two* i.s., C and D? You bet. See the 4th case in the definition above: If $C[p := B]$ and $D[p := B]$ are formulas, then so is $\Big((C[p := B]) \circ (D[p := B])\Big)$, again by 1.6. □

† A "Proposition" is a theorem—here metatheorem—that did not quite make it to be called that. You see, people reserve the term "Theorem", or "Metatheorem", for the "important" or earth shattering stuff that we prove. All else that we prove are just "Propositions", some are "Lemmata" (singular: Lemma)—if they have just "auxiliary status", just like FORTRAN subroutines—and some are "Corollaries", i.e., "little" Theorems (or Metatheorems) that follow trivially—more or less—from earlier results.

Finally,

3.5 Proposition. *If p does* not *occur in A, then $A[p := B] = A$,* **where we mean "=" as equality of strings**—*it says that the two sides read, as strings, exactly the same.*

Proof. Again, induction on formulas A:

Basis. A can only be r ($r \neq p$), *true* or *false*, by Definition 3.1. Then, by Definition 3.3 (2nd case), $A[p := B] = A$. OK, so far.

For the *induction steps* we assume that the i.s. of A have the property (I.H.), and prove that A inherits it :

For $A = (\neg C)$: We are told that $occurs(p, (\neg C)) = 1$ (see Remark 3.2). By 3.1, $occurs(p, C) = 1$. By I.H. above, $C[p := B] = C$ and we are done by 3.3, case 3.

For $A = (C \circ D)$: We are told that $occurs(p, (C \circ D)) = 1$. By 3.1, $occurs(p, C) = 1$ *and* $occurs(p, D) = 1$. By I.H. above, $C[p := B] = C$ *and* $D[p := B] = D$, and we are done by 3.3, case 4. □

4. Theorem calculus

We are ready to calculate theorems. As in formula calculations, we need to know:

(1) What we can write down outright

(2) What we can write as a result of things that we already wrote down.

Item (1) is addressed by the *axioms.* These are some well chosen formulas that *we agree to write down at any step, without checking to see what—if anything—we wrote earlier.*

In intuitive terms—granting that the quest for theorems is the quest for "mathematical truths"—these axioms are our *initial truths*, which, for reasons of our own, take for granted ("without proof"—but I am in the danger of getting ahead of myself).

Item (2) is serviced by our "writing rules" or rules of inference. The latter are often written as fractions, like

$$\frac{P_1, P_2, \ldots, P_n}{Q} \tag{R}$$

where all of $P_1, \ldots, P_n, Q$ denote formulas.

A rule like (R) above is understood as follows: In the course of a theorem calculation we are allowed to write the formula Q at any stage, as long as all of $P_1, \ldots, P_n$ have already been written in earlier stages.

But let us not get carried away in generalities, and let us spell out the exact rules of inference that we will use in this course, at least the ones we will need to define *proof* and *theorem.*†

†We will soon learn that we can apply additional rules in a theorem calculation, that were not mentioned in the definition below. Such rules we call *derived rules*, or *secondary rules.*

4.1 Definition. (Rules of Inference) The following three are our *primary* rules of inference:[‡]

Inf1

$$\frac{A \equiv B}{C[p := A] \equiv C[p := B]}, \text{ for any } A, B, p, C \qquad (\textit{Leibniz})$$

Inf2

$$\frac{A, A \equiv B}{B}, \text{ for any } A, B \qquad (\textit{Equanimity})$$

Inf3

$$\frac{A \equiv B, B \equiv C}{A \equiv C}, \text{ for any } A, B, C \qquad (\textit{Transitivity})$$

An instance of a rule of inference is obtained by replacing all the letters A, B, C and p by specific formulas and a specific variable respectively. □

4.2 Remark. (1) The rules above apply to formulas, *not* to arbitrary strings (unlike the formula formation rules of Section 1). The capital letters then stand for *any* formula, while the lower case, "p", for *any* variable.

(2) Other than the restriction that the A, B, C are formulas, there *is NO other restriction on the letters*. Thus, any of the rules "says" that "if in earlier stage(s) the formula(s) in the numerator of a rule have already been written—emphasis: *regardless of what led to them being written*—then we can write the denominator down in the present stage."

So we have disclosed our rules, in preparation of the following definition. Unfortunately, we will not disclose our set of axioms for the following reasons.

There are two types of axioms:

(1) Those that are *common* to *ALL* mathematical applications, and hence are considered as *part of Logic*. We call them for that reason *logical axioms*.

These we *will* disclose little by little, as our story unfolds (from Ch.3, to Ch.9, to Ch.8). But we will not do so right away now, and we will not need to do so anyway.

(2) Those that are *special* to particular branches of mathematics or *to particular situations/examples*—and may not "hold" outside that branch of MATH, or outside that situation/example. Certainly they will not hold in *all* mathematics. For example, a "special" axiom of "Peano Arithmetic" (on the natural numbers $\mathbb{N}$) is "$x + 1 \neq 0$". Note how this axiom is not true over all reals.

Since our job here (in MATH1090) is not to do "all mathematics", we will simply acknowledge the presence of such special axioms without saying—99.96% of the time—which ones we have in mind. Thus, this generality will allow us to explore *and practice* the reasoning that takes place in any MATH field, without specifically studying any *particular* MATH field.

[‡]Actually, it turns out that we only need **Inf1**–**Inf2** as primary, and we can demote **Inf3** to derived status.

We write Λ to denote **the** *set* of *all* logical axioms (for now, these are the ones in [2], Ch.3, with a few minor amendments).

We write Γ or Σ or Δ for **any** *set* of special (*NOT* logical) axioms. By the way, "special" axioms are called "*nonlogical*" by most logicians. We will sometimes call them assumptions.[†]

So our totality of axioms at any given point in an exposition—if we stick to the Γ notation—is $\Lambda \cup \Gamma$, i.e., the set of all formulas in Λ and all formulas in Γ, put together. Λ is fixed, but not yet spelled out. Γ changes with the context. On occasion it will be spelled out.

We are ready! (Compare with Definition 1.2.)

4.3 Definition. (Theorem calculations—or, Proofs) A *theorem-calculation*, or *proof*, is any finite (ordered) sequence of formulas that we may write respecting the following two requirements:

In any stage we may write down

Pr1 Any axiom (member of $\Lambda \cup \Gamma$)

Pr2 Any formula that appears in the denominator of *an instance* of a rule **Inf1**–**Inf3** as long as *all the formulas* in the numerator of the same instance of the (same) rule have already been written down at an earlier stage.

□

4.4 Definition. (Theorems) Any formula A that appears in a proof with nonlogical axiom set Γ is called a Γ-theorem. We write $\Gamma \vdash A$ to indicate this. We say "A is proved from Γ" or "Γ proves A".

If Γ is empty ($\Gamma = \emptyset$)—i.e., we have no special assumptions, we are just "doing Logic"—then we simply write $\vdash A$ and call A just "a theorem".

Caution! We may also do this out of laziness and call a Γ-theorem just "a theorem", if the context makes clear which $\Gamma \neq \emptyset$ we have in mind.

We say that A is *an absolute*, or *logical* theorem whenever Γ is empty. □

Note how in the symbol "$\vdash A$" we take Λ for granted and do not mention it.

So, 4.4 tels us what kind of theorems we have:

1. Anything in $\Lambda \cup \Gamma$.
2. For any formula C and variable p, $C[p := A] \equiv C[p := B]$, *provided* (we know that[†]) $A \equiv B$ is a (Γ-) theorem.
3. B (any B), *provided* (we know that) $A \equiv B$ *and* A are theorems.

[†]One mathematician calls them "temporary" assumptions.
[†]E.g., by looking back at an earlier stage.

4. $A \equiv C$ (any $A \equiv C$), *provided* (we know that) $A \equiv B$ *and* $B \equiv C$ are theorems.

Hey! The above is a recursive definition of (Γ-) theorems, and is worth recording (compare with Definition 1.6).

4.5 Definition. (Theorems—inductively) A formula E is a Γ-theorem iff E fulfills one of (**"=" below is equality of strings!**):

Th1 E is in $\Lambda \cup \Gamma$.

Th2 For some formula C and variable p, $E = \Big(C[p := A] \equiv C[p := B]\Big)$, *and* (we know that) $A \equiv B$ is a (Γ-) theorem.

Th3 (we know that) $A \equiv E$ *and* A are a theorems.

Th4 $E = \Big(A \equiv C\Big)$ *and* (we know that) $A \equiv B$ *and* $B \equiv C$ are theorems.

□

Needless to say that we can prove properties of theorems by induction on theorems, which is equivalent to doing induction on stages (compare with the case for formulas, Section 2). Our Chapter 2 ("Post's theorem and other things") looks into this issue, among other things.

Bibliography

[1] N. Bourbaki. *Éléments de Mathématique; Théorie des Ensembles.* Hermann, Paris, 1966.

[2] David Gries and Fred B. Schneider. *A Logical Approach to Discrete Math.* Springer-Verlag, New York, 1994.

[3] H. Hermes. *Introduction to Mathematical Logic.* Springer-Verlag, New York, 1973.