

Mathematical and Logical Basis of Computing: a Workbook

Howard A. Blair

Copyright 2006 – 2009

Permission is granted for printing for academic purposes.
All commercial rights reserved.

September 15, 2009

1. At *Amazon* if you search with

Epstein Carnielli

the first link returned will be to the book, *Computability* by Richard Epstein and Walter Carnielli. (cf. The course website in the Bibliography section.) The book is fully searchable. The material you will need for this section (September 15, 2009) will be found in Chapter 19, but you do not need all of it. Alternatively, you can go into the online *Wikipedia* at

http://en.wikipedia.org/wiki/Propositional_logic (Propositional Logic)

There you will find a good comprehensive introduction to propositional logic. We do not need to study propositional logic in depth, although you should be aware that there is much more in it than you might at first suppose.

2. You should also be aware that your experience in computer science has probably familiarized you already with much of what you need to know for present purposes about propositional logic and the main, classical, semantics associated with it: Boolean expressions built up purely from Boolean variables and Boolean connectives.
3. The single most important idea to “get” is that formulas in logic (i.e. propositional formulas, or Boolean expressions, in the case of propositional formulas) do not actually mean anything - until a *semantics* is specified.

4. When propositional logic is presented two main things need to be specified: the *syntax* and the *semantics*. The syntax is concerned with grammar: what is a formula of the logic, and what isn't. The semantics is concerned with what the formulas mean. To know something about logic you need to know about the grammar (syntax) of the logic, about the semantics of the logic, and how the logic relates to logic's big question: What follows from what?
5. The concept of formal proof is on the syntax side of logic. If there is a proof of something, we would like to know that what's proved is really true, or at least is true if the premises (the "givens") are true. In other words, does what we manage to prove *really* follow from what is given? This issue is that of whether the logic is *sound*. One other main issue, usually more complicated, but less important, is whether the logic is *complete*, and if not, by how much does it miss the mark? The issue of completeness of a logic is this: If the logic can express some collection of premises, and you want to know whether something φ that the logic can express does follow from the premises, then - if φ follows - there is a proof in the logic that will show you that it follows. Soundness and completeness are about the relationship between the syntax side and semantic side of the logic.
6. The first thing to do is to set up the syntax. You may want to examine the Wikipedia article on propositional logic mentioned above. Here is our set up: We start with an infinite set **Atoms**, which we call the set of *atoms*. Any member of this set - notice that we didn't actually specify what the set is - is called an *atom*. We also call a member of the set of atoms an *atomic formula*. Sometimes we call the members of the set of atoms, *Boolean variables*. We will use the terms *atom*, *atomic formula* and *Boolean variable* interchangeably. Now, we have the following *Boolean connectives*:
 - (a) true
 - (b) false
 - (c) \neg
 - (d) \rightarrow
 - (e) \leftrightarrow
 - (f) \wedge
 - (g) \vee

- (h) $|$
- (i) $— ? — : —$

The first two in the above list are 0-ary connectives. The next one is 1-ary (also called unary). The next five are 2-ary (also called binary). The last one is 3-ary (trinary). The last one is hugely important but almost never used. We also call these connectives *propositional connectives*.

7. The grammar rules for making formulas are:
 - (a) \mathbf{t} is a formula.
 - (b) \mathbf{f} is a formula.
 - (c) If A is an atom, then A is a formula.
 - (d) If A is a formula, then $\neg A$ is a formula.
 - (e) If A and B are formulas, then $(A \rightarrow B)$ is a formula.
 - (f) If A and B are formulas, then $(A \leftrightarrow B)$ is a formula.
 - (g) If A and B are formulas, then $(A \wedge B)$ is a formula.
 - (h) If A and B are formulas, then $(A \vee B)$ is a formula.
 - (i) If A and B are formulas, then $(A | B)$ is a formula.
 - (j) If A, B and C are formulas, then $(A ? B : C)$ is a formula.
8. If you are familiar with BNF-definitions, here is a similar, but more condensed, approach in BNF.
9. Let Atom range over the set of atoms. Then

```

Formula ::=  $\mathbf{t}$ 
          |  $\mathbf{f}$ 
          | Atom
          |  $\neg$  Formula
          | (Formula  $\circ$  Formula)
          | (Formula ? Formula : Formula)

```

$\circ ::= \rightarrow | \leftrightarrow | \wedge | \vee | |$

10. When we write logical formulas we will often omit parentheses when it is clear how to parse the formula.

11. That's it for now with syntax.
12. **Semantics:** Consider one of the easier, familiar propositional connectives, \wedge . What does \wedge usually mean? Well, it usually means *and*. What does that mean? First of all, the the symbol \wedge joins together two propositional formulas to make a more complicated one. So does \vee , so what is the difference? Well, $A \wedge B$ is just a different formula from $A \vee B$, so that's the difference. OK, but don't \wedge and \vee have something to do with Boolean values, and don't they get "defined by" truth tables? Yes, but what does all this cloud of vagueness mean? You might start to explain this business by saying that if you assign T to A and T to B then you must assign T to $A \wedge B$ and if you assign F either to A or to B then you must assign F to $A \wedge B$, and so on. OK, what do you mean by *assign*? And on and on and on. How are we going to make this stuff sharp and precise?
13. To start getting your mind out of this cloud of vagueness, go back and reconsider \wedge . That symbol *operates* on a pair of formulas. But the meaning of \wedge operates on a pair of Boolean values, as you know if you've ever seen truth-tables. The symbol and the symbol's *meaning* are different things. The symbol is a syntactic constructor that makes a slightly more complex formula out of two given formulas. The symbol's meaning returns a Boolean value, given a pair of Boolean-values as input.
14. A fancy symbol for "the meaning of" is $\llbracket \]$, as in $\llbracket \wedge \rrbracket$, which can be read as "the meaning of \wedge ". Is this any better? After all, what's a meaning? At least we have some notation for distinguishing between \wedge and what \wedge means - and that's some progress. But we can do a lot better than that with this idea.
15. We will write an *interpreter* - we'll call it `eval` - to evaluate the truth-values of Boolean formulas in an environment.
16. One possible move that we could make at this point is to make $\llbracket \wedge \rrbracket$ an operator on pairs of Boolean values that returns a Boolean value. We could declare

$$\llbracket \wedge \rrbracket : \{F, T\} \times \{F, T\} \longrightarrow \{F, T\}$$

which says that the meaning of \wedge is a function which takes as input a pair of Boolean values (that's the $\{F, T\} \times \{F, T\}$) and returns a Boolean value (that's the $\longrightarrow \{F, T\}$ part).

17. Now that we have declared the function $\llbracket \wedge \rrbracket$ we need to define it. This is mathematics, not a fixed programming language, so there are lots of ways we can do that. The simplest one is perhaps the one you are familiar with: truth-tables. A truth-table is just an input/output lookup table for a function:

input	output
<i>F</i> <i>F</i>	<i>F</i>
<i>F</i> <i>T</i>	<i>F</i>
<i>T</i> <i>F</i>	<i>F</i>
<i>T</i> <i>T</i>	<i>T</i>

This I/O table defines the function $\llbracket \wedge \rrbracket$ and tells you how the meaning of \wedge behaves in the familiar way.

18. The other propositional connectives are defined in exactly the same way. For example,

$$\llbracket \rightarrow \rrbracket : \{F, T\} \times \{F, T\} \longrightarrow \{F, T\}$$

where

input	output
<i>F</i> <i>F</i>	<i>T</i>
<i>F</i> <i>T</i>	<i>T</i>
<i>T</i> <i>F</i>	<i>F</i>
<i>T</i> <i>T</i>	<i>T</i>

19. $\llbracket \rightarrow \rrbracket$ often seems confusing, or at least a bit forced. The third and fourth rows of the truth seem fairly clear, but why should $p \rightarrow q$ be true, just because p is false. Is it really true that: if the moon is made of green cheese, then George W. Bush is a Liberal Democrat? Yes, according to the truth table, since the moon is not made of green cheese. (Wensleydale isn't green.) One way to convince yourself that the above truth table is correct is to ask how else could you define the meaning of \rightarrow . There are three other possibilities:

input	output
<i>F</i> <i>F</i>	<i>F</i>
<i>F</i> <i>T</i>	<i>F</i>
<i>T</i> <i>F</i>	<i>F</i>
<i>T</i> <i>T</i>	<i>T</i>

This one just above says that $\llbracket \rightarrow \rrbracket$ means just what \wedge means.

input	output
F F	F
F T	T
T F	F
T T	T

This one just above says that $p \rightarrow q$ is true exactly when q is, and p doesn't matter.

input	output
F F	T
F T	F
T F	F
T T	T

And finally this one says that $p \rightarrow q$ is true exactly when p and q have the same Boolean value, in which case there is no difference between $p \rightarrow q$ and $q \rightarrow p$.

20. So it seems that all three of the other possibilities aren't correct. This argument might not seem convincing. A hidden premise behind the argument is that the Boolean value of a compound proposition should be determined by the Boolean values of its parts, *and nothing else*. Consider the following argument:

I did not strike the match at 9:00am. Therefore, if I had struck the match at 9:00am it would have lit.

21. This is not a valid argument. What if the match was wet? Yet, by truth tables this is seen to be a valid argument. There must be something wrong with the truth table representation of it. **Challenge problem (not part of the course):** What's wrong?
22. What does the formula $p \wedge q$ mean? Remember, one of the big ideas is that formulas don't mean anything until a semantics is specified. Using our notation, the question we face is how to handle something like

$$\llbracket p \wedge q \rrbracket$$

In particular, $\llbracket p \wedge q \rrbracket$ should somehow help us with the question, is $p \wedge q$ true or false? Of course that depends on the Boolean values assigned to p and q . (Remember the question, what's an assignment?) Here's a way to deal with this.

Definition (Mathematical Logic Terminology): A *structure* for propositional logic is a function that takes an atom as input and returns a Boolean value.

Therefore a declaration for a structure ν has the form

$$\nu : \mathbf{Atoms} \longrightarrow \{F, T\}$$

The following definition repeats the definition of *structure*.

Definition (Computer Science Terminology): An *environment* for propositional logic is a function that takes an atom as input and returns a Boolean value.

23. Now, in order to press on we have to ask how we can describe the set of all structures. We need to have that description so that we can declare functions that take structures as inputs. Just for a moment consider two nonempty sets S_1 and S_2 . The set of *all* functions from S_1 to S_2 is denoted by

$$S_1 \longrightarrow S_2$$

But what is this set? To get a grip on it in terms of S_1 and S_2 consider an example involving two small finite sets. Suppose $S_1 = \{0, 1\}$ and $S_2 = \{0, 1, 2\}$. An I/O table such as

input	output
0	2
1	1

defines a function from S_1 to S_2 . So we can systematically list all of the functions from S_1 to S_2 using I/O tables. That is not the only way to describe the set $S_1 \rightarrow S_2$, but it's one way.

We get nine tables in all:

input	output
0	0
1	0

input	output
0	0
1	1

input	output
0	0
1	2

input	output
0	1
1	0

input	output
0	1
1	1

input	output
0	1
1	2

input	output
0	2
1	0

input	output
0	2
1	1

input	output
0	2
1	2

24. We can condense all nine tables into by listing the input column just once, and all output columns side by side.

input	output								
0	0	0	0	1	1	1	2	2	2
1	0	1	2	0	1	2	0	1	2

This final table exhibits the whole set $S_1 \rightarrow S_2$ in a single table.

Notice also that if we delete the input column (since that doesn't change) in each of the nine individual tables, what we end up with is the set of all possible fully filled-in instances of an array indexed by S_1 whose cells are required to hold elements from S_2 . The set of all such array instances is another way to describe $S_1 \rightarrow S_2$.

25. We have arrived at

$$\mathbf{Atoms} \rightarrow \{F, T\}$$

as the set of all structures for Propositional Logic - at least propositional logic with its usual, classical, semantics, with a fixed set **Atoms** as the set of atoms in its syntax. To make things a little more clear later on, we give this set a name: **Environments**

26. If we were to examine more deeply what we are doing with this last definition we would see that we were defining *states*. Exactly why that is would take us too far from what we are building right now, but keep in mind that propositional logic can't properly deal with state change. But there are logics, modal logics, that can deal with that in a strong sense.
27. Now we can define the meanings of formulas.
28. Intuitively, the Boolean value of $A \wedge B$ is fully determined by the Boolean values of A and B and the truth table for $\llbracket \wedge \rrbracket$. In turn, the Boolean values of A and B are determined by the parts of A and B and so on down to the atoms. The Boolean values of the atoms are given by some structure. So this gives us the way to define the meanings of all of the formulas recursively in a way that parallels the grammar that allows us to build up all of the formulas from atoms. The meaning $\llbracket \varphi \rrbracket$ of a formula φ is a function that takes an environment ν and returns a Boolean value.

$$\llbracket \varphi \rrbracket : \mathbf{Environments} \rightarrow \{F, T\}$$

We define $\llbracket \varphi \rrbracket$ recursively:

$$\begin{aligned}\llbracket \mathbf{f} \rrbracket(\nu) &= F \\ \llbracket \mathbf{t} \rrbracket(\nu) &= T \\ \llbracket A \rrbracket(\nu) &= \nu(A), \text{ for each atom } A \text{ in } \mathbf{Atoms}. \\ \llbracket \neg A \rrbracket(\nu) &= \llbracket \neg \rrbracket(\llbracket A \rrbracket(\nu))\end{aligned}$$

and, for each binary propositional connective \circ ,

$$\llbracket A \circ B \rrbracket(\nu) = \llbracket \circ \rrbracket(\llbracket A \rrbracket(\nu), \llbracket B \rrbracket(\nu))$$

29. We can now define the evaluation function Boolean formulas in environments.

$$\text{eval} : \mathbf{Formulas} \times \mathbf{Environments} \longrightarrow \{F, T\}$$

is defined by

$$\text{eval}(A, \nu) = \llbracket A \rrbracket(\nu)$$

30. We could alternatively give a recursive definition of **eval** as follows.

31.
$$\begin{aligned}\text{eval}(\mathbf{f}, \nu) &= F \\ \text{eval}(\mathbf{t}, \nu) &= T \\ \text{eval}(A, \nu) &= \nu(A), \text{ for each atom } A \text{ in } \mathbf{Atoms} \\ \text{eval}(\neg A, \nu) &= \llbracket \neg \rrbracket(\text{eval}(A, \nu))\end{aligned}$$

and, for each binary propositional connective \circ ,

$$\text{eval}(A \circ B, \nu) = \llbracket \circ \rrbracket(\text{eval}(A, \nu), \text{eval}(B, \nu))$$

32. **Task:** Using the definition just given, calculate

$$\text{eval}(p \rightarrow (\neg q \mid (r \rightarrow (p \wedge r))), \nu)$$

where

$$\nu(p) = \nu(r) = T, \text{ and } \nu(q) = F$$

Answer:

$$\begin{aligned}
& \text{eval}(p \rightarrow (\neg q \mid (r \rightarrow (p \wedge r))), \nu) \\
= & \llbracket \rightarrow \rrbracket(\text{eval}(p, \nu), \text{eval}(\neg q \mid (r \rightarrow (p \wedge r)), \nu)) \\
= & \llbracket \rightarrow \rrbracket(T, \llbracket \mid \rrbracket(\text{eval}(\neg q, \nu), \text{eval}(r \rightarrow (p \wedge r), \nu))) \\
= & \llbracket \rightarrow \rrbracket(T, \llbracket \mid \rrbracket(\llbracket \neg \rrbracket(\text{eval}(q, \nu)), \llbracket \rightarrow \rrbracket(\text{eval}(r, \nu), \text{eval}(p \wedge r, \nu)))) \\
= & \llbracket \rightarrow \rrbracket(T, \llbracket \mid \rrbracket(\llbracket \neg \rrbracket(F), \llbracket \rightarrow \rrbracket(T, \llbracket \wedge \rrbracket(\text{eval}(p, \nu), \text{eval}(r, \nu))))) \\
= & \llbracket \rightarrow \rrbracket(T, \llbracket \mid \rrbracket(T, \llbracket \rightarrow \rrbracket(T, \llbracket \wedge \rrbracket(T, T)))) \\
= & \llbracket \rightarrow \rrbracket(T, \llbracket \mid \rrbracket(T, \llbracket \rightarrow \rrbracket(T, T))) \\
= & \llbracket \rightarrow \rrbracket(T, \llbracket \mid \rrbracket(T, T)) \\
= & \llbracket \rightarrow \rrbracket(T, F) \\
= & F
\end{aligned}$$

By the way, we never defined $\llbracket \mid \rrbracket$. Here is its I/O table (i.e. truth table).

input	output
$F \quad F$	T
$F \quad T$	T
$T \quad F$	T
$T \quad T$	F

33. Example:

Assertion 1: If Alice teleports her qubit, then if Bob doesn't power up his qubit receiver, then Bob doesn't receive Alice's qubit.

Assertion 2: Bob should not buy a new qubit receiver, if Bob doesn't receive Alice's qubit and Bob is angry.

Assertion 3: If Bob gets spied on, then Bob is angry and Bob should buy a new qubit receiver.

Goal: Therefore, If Alice teleports her qubit, then Bob doesn't receive Alice's qubit and Bob doesn't get spied on.

Let A stand for "Alice teleports her qubit." Let B stand for "Bob should buy a new qubit receiver." Let R stand for "Bob receives Alice's qubit." Let P stand for "Bob powers up his qubit receiver." Let S stand for "Bob gets spied on." Let G stand for "Bob is angry".

(1) Express the assertions and goal in our notation for propositional logic.

- a1. $A \supset (P \supset \neg R)$
- a2. $(\neg R \wedge G) \supset \neg B$
- a3. $S \supset (G \wedge B)$
- g1. $A \supset (\neg R \wedge \neg S)$

(2) Does the goal follow from the assertion? (Explain.)

34. **Example:** Does the goal follow from the assertion? (Explain.)

- a1. $B \supset (L \supset M)$
- a2. $(M \wedge D) \supset \neg G$
- a3. $\neg J \supset (D \wedge G)$
- g1. $B \supset (L \supset J)$

Yes:

- | | |
|--|--------------|
| a1. $B \supset (L \supset M)$ | given |
| a2. $(M \wedge D) \supset \neg G$ | given |
| a3. $\neg J \supset (D \wedge G)$ | given |
| g1. $B \supset (L \supset J)$ | given |
| a4. B | g1, if-split |
| g2. $L \supset J$ | g1, if-split |
| a5. $\text{False} \vee (\text{True} \supset (L \supset M))$ | a4,a1, AA |
| a6. $L \supset M$ | a5, rew |
| a7. L | g2, if-split |
| g3. J | g2, if-split |
| a8. $\text{False} \vee (\text{True} \supset M)$ | a7,a6, AA |
| a9. M | a8, rew |
| a10. $\text{False} \vee ((\text{True} \wedge D) \supset \neg G)$ | a9,a2, AA |
| a11. $\neg(D \wedge G)$ | a10, rew |
| a12. $(\neg J \supset \text{false}) \vee \neg \text{true}$ | a3,a11, AA |
| a13. J | a12, rew |
| g4. $\neg \text{false} \wedge \text{true}$ | a13,g3, AG |
| g5. true | g4, rew |

First Order Logic

1. **First Order Logic.** By the term *classical logic* we intuitively mean a logic with the usual two familiar Boolean values, $\{F, T\}$, in which the propositional connectives have their usual meaning that can be defined via truth-tables. Informally, *first order logic* is a classical logic extended with the quantifiers \forall and \exists and where atoms have internal structure that allow us to refer to relationships among so-called *individuals*. So our problem now is to get technical and make all of this stuff precise.
2. **The variables.** We begin by choosing an infinite (countably infinite, but the cardinality really doesn't matter) set of basic symbols, called *variables*. (These are not *Boolean variables*. There just called variables. If we want to emphasize the distinction between these variables and Boolean variables, we can call these variables *individual variables* because these variables are bound to values. Usually we don't bother with saying or writing *individual*. We will typically use symbols such as x, y, z, w, u and v , with and without subscripts, to designate variables.
3. **Terms.** We have discussed how to declare the algebraic part of a language for first-order logic, defined what the terms of this algebraic language are, and then applied these concepts in showing how the *unification algorithm* worked. To finish declaring a first-order language, we add so-called *predicate symbols*. Just as we declared function symbols for each *arity* (usually declaring that there no function symbols for most arities) we do the same for predicate symbols. There are three predicate symbols about which we have no choice.
4. **Predicate symbols.** The 0-ary predicate symbols are true and false. We met these symbols before as 0-ary propositional connectives. Their true identity could only properly be revealed with predicate symbols. The third predicate symbol about which we have no choice is the symbol, $=$. It is an arity 2 predicate symbol. All of the remaining predicate symbols are up to us to declare.
5. **The Language of Arithmetic.**

Function symbols

arity 0: : 0

	arity 1:	s
	arity 2:	+ , *
Predicate symbols:		
	arity 0:	true , false
	arity 2:	=

You might think we ought have constants for each number such as 1 for 1, and predicate symbols such as $<$, but we don't need them. 1 and 2 can for example be regarded as macros that expand to $s(0)$ and $s(s(0))$, respectively. And so on. $<$ can be regarded as a macro also, but the way to expand it awaits the next two ideas.

6. **Atomic formulas**; also called **atoms**. The definition of what a term is doesn't need to be revised. The key thing to remember about terms is that **predicate symbols never occur in terms**. Now suppose, p is a k -ary predicate symbol, where $k \geq 0$, and t_1, \dots, t_n are terms. Then $pt_1 \dots t_n$ is an *atomic formula*. As usual, we allow ourselves to apply macros, cut up, reassemble and decorate with graffiti our syntactic constructions to help make them more readable. (These modifications are also known as *syntactic sugar* because reading the expressions is "sweetened". So we would generally write $pt_1 \dots t_n$ as $p(t_1, \dots, t_n)$. More concretely, a formula such as $= +xyss0$ could be written as $(+xy, ss0)$, or as $+xy = s(s(0))$, or as $x + y = 2$.
7. **The formulas of a language for first-order logic**. First, recall the grammar for the formulas of propositional logic:

Formula ::= **t**
| **f**
| Atom
| \neg Formula
| (Formula \circ Formula)
| (Formula ? Formula : Formula)

$\circ ::= \rightarrow \mid \leftrightarrow \mid \wedge \mid \vee \mid \mid$

We modify the rules of this grammar to obtain the general form of the grammar for the formulas of a language \mathcal{L} for first-order logic. The grammar that we present below already assumes that the set of atoms of \mathcal{L} , **Atoms** $_{\mathcal{L}}$ (and therefore the set of terms **Terms** $_{\mathcal{L}}$) have been defined. We also assume that **Var** is the set of variables. Then

Formula ::= **Atoms** _{\mathcal{L}}
 | \neg Formula
 | (Formula \circ Formula)
 | (Formula ? Formula : Formula)
 | \forall **Var** Formula
 $\circ ::= \rightarrow \mid \leftrightarrow \mid \wedge \mid \vee \mid \mid$

8. Examples of formulas of the language of arithmetic.

$$(x > 1) \vee \forall y [\exists z [y * z = x] \supset (y = 1 \vee y = x)]$$

This is a formula from the language of arithmetic. To use the grammar to see that is such a formula, we proceed recursively: we first observe that there is syntactic sugar present because parentheses have been omitted. If we put them back in, we obtain

$$((x > 1) \vee \forall y [\exists z [y * z = x] \supset (y = 1 \vee y = x)])$$

This is an instance of the template

$$(\text{Formula } \circ \text{ Formula})$$

Next we need to verify that $(x > 1)$ is an instance of Formula. Again there is syntactic sugar present: the parentheses surrounding $x > 1$. Erasing these parentheses we are left with verifying that $x > 1$ is a formula of the language of arithmetic. $>$ is a macro in the sense that a formula $t_1 > t_2$ expands to $t_2 < t_1$. In turn, $<$ is a macro and $t_2 < t_1$ expands, by definition, to $\exists y [t_2 + (y + 1) = t_1]$, where y is a newly chosen variable. The symbol \exists is also a macro and a syntactic expression such as $\exists y \varphi$ expands to $\neg \forall y \neg \varphi$. 1 of course expands to $s(0)$. Thus, after macro expansion, $x > 1$ becomes

$$\neg \forall y \neg [s(0) + (y + s(0)) = x]$$

where \circ is instantiated to \vee . And so on.

9. Task: Re-express

$$(x > 1) \vee \forall y [\exists z [y * z = x] \supset (y = 1 \vee y = x)]$$

using Polish (i.e prefix, parentheses-free notation) notation.

10. Hereafter, we will freely use syntactic sugar.
11. **Semantics.** What do these formulas mean? To understand the semantics of the formulas of first-order logic we have to understand what a *relational structure* (when the context clear we just call such a thing a *structure*) is. We will now describe how to set up a structure for a language for first-order logic. We are going to build such a structure. We'll give it a name: \mathfrak{A} . This structure will be a structure for a language \mathcal{L} .
12. Part of the structure that we are building is an *algebra*. (You know what algebra is as a subject that you study in school. But if you ever wanted to know what an *algebra* is, this is it: you are about to find out.) We choose a nonempty set. This set is called the *universe* of the structure we're building. We denote the universe of \mathfrak{A} by $|\mathfrak{A}|$. When mathematicians, logicians and scientists are just dealing with the algebra part of \mathfrak{A} , they speak of the *carrier* of \mathfrak{A} . So *carrier* means the same thing as *universe* in the context of algebras. To simplify our usage of these terms we shall hereafter exclusively use the term *universe*.
13. To fully set up an algebra \mathfrak{A}_0 for the constant and function symbols of a language \mathcal{L} for FOL, we have to interpret each of these symbols on a nonempty set that will serve as the universe of the algebra. For a constant or function symbol σ we will denote the interpretation of σ by $\llbracket \sigma \rrbracket$. The *type* of $\llbracket \sigma \rrbracket$ - in a programming context, the type of $\llbracket \sigma \rrbracket$ is the meaning of a declaration. The type of $\llbracket \sigma \rrbracket$ is determined by the arity of σ - in setting up the algebra we have no choice about the type of σ . The type of $\llbracket \sigma \rrbracket$ is given by (in "mathematics notation")

$$\llbracket \sigma \rrbracket : \underbrace{|\mathfrak{A}| \times \dots \times |\mathfrak{A}|}_n \longrightarrow |\mathfrak{A}| \quad (1)$$

The corresponding declaration in a programming context looks like

$$\mathfrak{A} \sigma(\mathfrak{A}x_1, \dots, \mathfrak{A}x_n);$$

The meaning of $\mathfrak{A} \sigma(\mathfrak{A}x_1, \dots, \mathfrak{A}x_n)$; which is denoted by

$$\llbracket \mathfrak{A} \sigma(\mathfrak{A}x_1, \dots, \mathfrak{A}x_n); \rrbracket$$

The meaning of $\mathfrak{A} \sigma(\mathfrak{A}x_1, \dots, \mathfrak{A}x_n)$; could be further denoted by

$$|\mathfrak{A}| \llbracket \sigma \rrbracket (\underbrace{|\mathfrak{A}| \dots |\mathfrak{A}|}_n); \quad (2)$$

Note the exact way to translate between the two ways [displays (1) and (2) just above] of expressing the type of $\llbracket \sigma \rrbracket$: “mathematical notation” and “C-style programming notation”.

14. As we said above, we have no choice in determining the type of each constant and function symbol. By the way, the type of a constant symbol e is best rendered in “C-style programming notation”: $|\mathfrak{A}| \llbracket e \rrbracket ()$;
15. The set of all functions from a set A to a set B is denoted by

$$A \longrightarrow B$$

When we write

$$f : A \longrightarrow B$$

we are stating the type of function f . But, the notation for the type of f says that function f is a member of the set of all functions $A \longrightarrow B$. We could just as well have written

$$f \in (A \longrightarrow B)$$

When we fully specify a function f from set A to set B , we have to not only give the type of f , we have to choose f from $A \longrightarrow B$. That doesn’t mean we have to give a rule for somehow calculating f . So, when we are setting up an algebra, when we interpret an n -ary function symbol σ by determining the meaning of $\llbracket \sigma \rrbracket$, we just to have to choose a function from the set of functions

$$\underbrace{|\mathfrak{A}| \times \dots \times |\mathfrak{A}|}_n \longrightarrow |\mathfrak{A}|$$

That doesn’t mean we have to write down a rule for the one we choose. We only go that far if we want to *explicitly describe* an algebra. By the way, if the universe of an algebra is infinite, and the first order language that the algebra interprets has any nonconstant function symbols, then it turns out that almost none of the algebras with that universe can be explicitly described.

16. To complete the set up of an algebra there is one more issue to take note of: the equality symbol, $=$, needs an interpretation. Once again, there is no choice. The equality symbol is interpreted as the identity binary relation on $|\mathfrak{A}|$; i.e.

$$\llbracket = \rrbracket = \{(x, x) \mid x \in |\mathfrak{A}|\}$$

If you are confused about what has been said here about the equality symbol's meaning, then just take it as a given that the symbol means what you probably think it means.

17. We will now give two different algebras to interpret the algebraic part of the language of arithmetic. The first one is known as the *standard model of arithmetic*, and the second one is Raphael Robinson's *nonstandard* model of arithmetic.

Example 1: (*The standard model of arithmetic.*) Recall the algebraic part of the language of arithmetic:

Function symbols

arity 0: : 0
 arity 1: s
 arity 2: +, *

Predicate symbols:

arity 2: =

The standard model is often denoted by \mathfrak{N} ("Gothic 'N' "). The universe of \mathfrak{N} is the set of natural numbers \mathbb{N} (which includes 0.) Thus

$$|\mathfrak{N}| = \mathbb{N}$$

The type of, for example, $\llbracket + \rrbracket$ is given by

$$\llbracket + \rrbracket = |\mathfrak{N}| \times |\mathfrak{N}| \longrightarrow |\mathfrak{N}|$$

As we said above, we have no choice about the types of the symbols when we give the types. We will *identify* the constant functions with the elements in their ranges. What is at stake is that according to rules for interpreting the constant symbols, if we wanted the symbol 0 in the language of arithmetic to mean the natural number 0, we would have to say that $\llbracket 0 \rrbracket$ is the function that takes no input and returns the natural number 0. We could perhaps use a pseudo C-style declaration and definition to describe this function:

```
unsigned int 0(){
    return 0;
}
```

In mathematical notation we could use an expression from the λ -calculus (we talked about this in class, and it is a notation that you are familiar with if you know LISP or any LISP-like language, such as Scheme):

$$\llbracket 0 \rrbracket = \lambda n[0]$$

But, this is unnecessarily complicated for our purposes. Instead, it would be easier to just have the meaning of 0 be 0:

$$\llbracket 0 \rrbracket = 0$$

We can arrange for this by identifying $\lambda n[0]$ with 0. Again, you are familiar with these sorts of identifications, but you might not realize it. In matrix algebra, we often identify a 1×1 matrix $[a]$ with the number a in its entry.

The meaning of the successor symbol \mathbf{s} is given by

$$\llbracket \mathbf{s} \rrbracket = \lambda n[n + 1]$$

We could also give the meaning of \mathbf{s} by

$$\llbracket \mathbf{s} \rrbracket(n) = n + 1$$

The plus symbol on the left side of the equations in the above two displays is the symbol in the language of arithmetic - the object language. The plus symbol on the right side is our usual symbol for addition on the set of natural numbers. This symbol is a symbol in the language we are using to talk about the object language and to talk about mathematical considerations - the metalanguage. What counts as object language and as metalanguage depends on the context. A metalanguage is itself a perfectly good language, and if we wanted to talk about it in another context, it would be an object language in the other context.

For the remaining two function symbols, the meaning is given by

$$\begin{aligned} \llbracket + \rrbracket &= + \\ \llbracket * \rrbracket &= * \end{aligned}$$

Again, the symbols on the right side in the above two displays are the usual symbols for addition, and multiplication on the set of natural numbers.

Example 2: *Raphael Robinson's nonstandard model of arithmetic*. We'll use \mathfrak{Q} to denote this structure.

$$|\mathfrak{Q}| = \mathbb{N} \cup \{\alpha, \beta\}$$

where α and β are not natural numbers and are two distinct objects. (What they are is irrelevant - it is only the relationship between them and the natural numbers given by how we will interpret 0, \mathbf{s} , + and * that matters).

$$[[0]] = 0$$

We will specify the interpretations of \mathbf{s} , + and * with input/output tables. Let m and n be natural numbers. Then

[[\mathbf{s}]]:

input	output
n	$n + 1$
α	α
β	β

[[+]]:

input		output
m	n	$m + n$
m	α	β
m	β	α
α	n	α
α	α	β
α	β	α
β	n	β
β	α	β
β	β	α

In the following table, let m and n be nonzero, so that we can make 0 a special case.

[[*]]:

input	output
0	0
0	n
0	α
0	β
m	0
m	n
m	α
m	β
α	0
α	n
α	α
α	β
β	0
β	n
β	α
β	β

18. The language of arithmetic, as we have presented it, has the disadvantage that there are no predicate symbols in it other than `true`, `false` and `=`. Because there are no other predicate symbols, the examples we give of interpretations of this language can't exhibit the difference between an algebra and a structure. To remedy this deficiency, we can add a predicate symbol to the language of arithmetic: `<`. It turns out that everything we can express about arithmetic with the language of arithmetic can be said without this predicate symbol, by treating atomic formulas that use it as macros. We will show how to do this below, but right now, we want to give a couple of examples that show how to extend the algebras for the algebraic part of the language of arithmetic to relational structures that interpret the newly added predicate symbol.

19. **Example:** (*The standard model of arithmetic.*)

$$\llbracket < \rrbracket = <$$

The less-than symbol on the left is the predicate symbol in the language of arithmetic. The less-than symbol on the right denotes in the usual way the less-than relation on the set of natural numbers.

It is important for the next example to have firmly in mind that the

less-than relation on the natural numbers involves a set of pairs of natural numbers. Specifically, the less-than relation $<$ on the set of natural numbers is a triple (not a set of triples)

$$(\mathbb{N}, \mathbb{N}, \{(i, j) \mid i + k = j \text{ for some nonzero natural number } k\})$$

The first two components of the triple are the domain and codomain of the relation, and the third argument is the *extension* of the relation. We usually informally identify the relation with just its extension when the domain and codomain are clear from the context. Thus we would write (by overloading the $\llbracket \cdot \rrbracket$ notation):

$$\llbracket < \rrbracket = \{(i, j) \mid i + k = j \text{ for some nonzero natural number } k\}$$

20. **Example:** (*Raphael Robinson's nonstandard model of arithmetic.*)

Recall that the universe of this model has all of the natural numbers together with two extra individuals that we denoted by α and β . The task here is to extend the the standard less-than relation on the set of natural numbers to take account of α and β . One way to make such an extension would be have α and β not be comparable with each other nor with any natural number. But it turns out that a tremendously important property of Robinson's model would be lost if we did that. (We will look into that important property later.) What Robinson did was arrange for the extension of $\llbracket < \rrbracket$ to obey the following equation:

$$\llbracket < \rrbracket = \{(x, y) \mid x + z = y \text{ for some nonzero individual } j \text{ in } \mathbb{N} \cup \{\alpha, \beta\}\}$$

In other words,

$$\llbracket < \rrbracket = < \cup \{(n, \alpha) \mid n \in \mathbb{N}\} \cup \{(n, \beta) \mid n \in \mathbb{N}\} \cup \{(\alpha, \beta)\} \cup \{(\beta, \alpha)\}$$

Informally, the above equation implies that every natural number is less than α and less than β , and α and β are less than each other. So, the interpretation of $<$ in Robinson's nonstandard model violates antisymmetry.

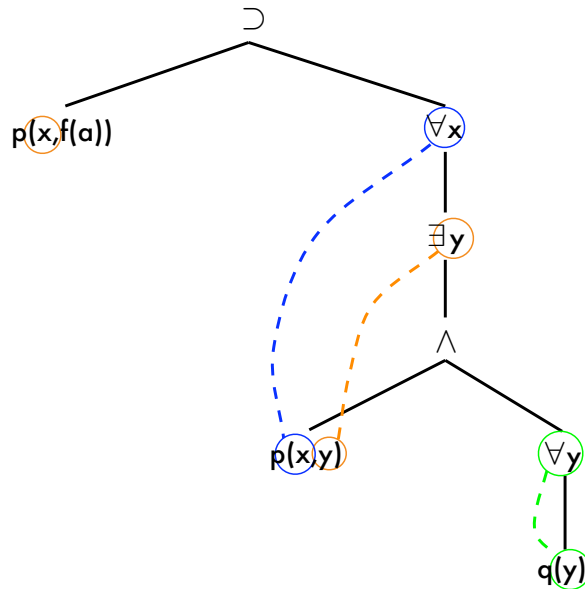
21. Now that we have some sense of how structures interpret the symbols of a language for first-order logic, we need to know how formulas are given Boolean values by a structure. To deal with that issue, we must first take up the notion of free and bound occurrences of variables. The easiest way to define the free and bound occurrences of variables

in a formula is to consider the formula's phrase-structure tree in which the leaves are atomic formulas. Every variable occurrence terminates within an atomic formula; i.e. variables appear either paired with quantifiers or within atomic formulas.

22. **Example:** Consider the formula

$$p(x, f(a)) \supset \forall x \exists y (p(x, y) \wedge \forall y q(y))$$

Going from left to right, the first occurrence of x is free in the formula. The second occurrence of x is free in the subformula $p(x, y)$, free in the subformula $(p(x, y) \wedge \forall y q(y))$, and free in the subformula $\exists y (p(x, y) \wedge \forall y q(y))$. It is bound in the subformula $\forall x \exists y (p(x, y) \wedge \forall y q(y))$ by the only occurrence of $\forall x$ and bound in the whole formula by the only occurrence of $\forall x$. The occurrence of y in $p(x, y)$ is free in $p(x, y)$, free in $(p(x, y) \wedge \forall y q(y))$ and bound in $\forall x \exists y (p(x, y) \wedge \forall y q(y))$ by the only occurrence of the quantifier $\exists y$. The occurrence of y in $q(y)$ is free in $q(y)$, bound in $\forall y q(y)$ by the only occurrence of $\forall y$, and similarly bound in all formulas containing $\forall y q(y)$ as a subformula.



23. **Expanding a formula over a structure.** We present the steps for expanding a formula over a structure. Again, let \mathcal{L} be an arbitrary first order language, and let \mathfrak{A} be an arbitrary structure for \mathcal{L} . Suppose we are given formula A to expand.

Step 1: (Preprocessing) Let x_1, \dots, x_k be the variables that have free occurrence in A . Replace A by its *universal closure* $\forall x_1 \dots \forall x_k A$. Call the resulting formula A' . If A had no free occurrences of variables, then A' is just A . The order of the quantifiers introduced in this preprocessing step does not matter.

Step 2: If A' is $\forall x B$, then replace A' by

$$\bigwedge_{i \in |\mathcal{A}|} \overline{B[x \mapsto i]}.$$

where $\overline{B[x \mapsto i]}$ is the formula that results from replacing all free occurrences of x in B by the individual i , and $\overline{B[x \mapsto i]}$ is the expansion of $B[x \mapsto i]$. (In effect we expanded the language \mathcal{L} by putting in each individual i as new constant symbol, and we will extend the structure \mathfrak{A} to a structure for this newly expanded language by interpreting each individual as itself.)

Step 3: If A' is $\exists x B$, then replace A' by

$$\bigvee_{i \in |\mathcal{A}|} \overline{B[x \mapsto i]}.$$

Step 4: If A' is a propositional combination of formulas, $B_1 \circ B_2$, then the expansion of A' is $\overline{B_1} \circ \overline{B_2}$, where $\overline{B_1}$ is the expansion of B_1 and $\overline{B_2}$ is the expansion of B_2 .

Step 5: If A' is $\neg B$, then the expansion of A' is $\neg \overline{B}$, where \overline{B} is the expansion of B .

Step 6: If A' is $B ? C : D$, then the expansion of A' is $\overline{B} ? \overline{C} : \overline{D}$, where \overline{B} is the expansion of B , \overline{C} is the expansion of C and \overline{D} is the expansion of D .

24. In effect we expanded the language \mathcal{L} to a language $\mathcal{L}(\mathfrak{A})$ by putting in each individual i in $|\mathfrak{A}|$ as new constant symbol, and we will extend the structure \mathfrak{A} to a structure for this newly expanded language by interpreting each individual as itself. Each variable-free term \mathfrak{t} of the expanded version of \mathcal{L} then denotes an individual $\llbracket \mathfrak{t} \rrbracket$ in $|\mathfrak{A}|$.
25. **Definition:** A variable-free atomic formula $\mathfrak{p}(\mathfrak{t}_1, \dots, \mathfrak{t}_n)$ has Boolean value *True* in \mathfrak{A} iff $(\llbracket \mathfrak{t}_1 \rrbracket, \dots, \mathfrak{t}_n) \in R$, where R is the n -ary relation on $|\mathfrak{A}|$ that is the interpretation of \mathfrak{p} in \mathfrak{A} . A formula A is *valid* in

structure \mathfrak{A} iff the expansion of A over $|\mathfrak{A}|$ has Boolean value *True* in \mathfrak{A} . We denote that A is valid in \mathfrak{A} by

$$\mathfrak{A} \models A$$

A formula A is said to be *logically valid* iff A is valid in \mathfrak{A} for every structure \mathfrak{A} for every language \mathcal{L} of which A is a formula. (In other words A is logically valid iff for every set of individuals, the expansion of A is a propositional tautology.) We denote that A is logically valid by

$$\models A$$

We also abbreviate the phrase *logically valid* to just *valid*.

26. **Example Problem:** Show that the formula is

$$(\forall x [p(x, x) \supset p(x, a)] \wedge \exists y p(y, y)) \supset p(a, a)$$

not valid by giving a relational structure \mathcal{A} in which the formula is false.

Answer: We guess that an interpretation with 2 individuals in its universe of discourse will provide an interpretation in which the formula evaluates to **False**. Call the two individuals 0 and 1, call the interpretation \mathcal{A} and expand the formula over the set $\{0, 1\}$.

$$\begin{aligned} & (\forall x [p(x, x) \supset p(x, a)] \wedge \exists y p(y, y)) \supset p(a, a) \\ \equiv_{\mathcal{A}} & (\bigwedge_{x \in \{0, 1\}} [p(x, x) \supset p(x, a)] \wedge \bigvee_{y \in \{0, 1\}} p(y, y)) \supset p(a, a) \\ \equiv_{\mathcal{A}} & ([p(0, 0) \supset p(0, a)] \wedge [p(1, 1) \supset p(1, a)]) \wedge [p(0, 0) \vee p(1, 1)] \supset p(a, a) \end{aligned}$$

We now try to falsify the above formula by assigning Boolean values to the atoms that occur in it. We see by Gentzen's method that if we interpret the constant symbol a as 0, and assign **true** to $p(1, 1)$, $p(1, 0)$ and $p(0, 1)$, and assign **false** to $p(0, 0)$, then the formula evaluates to **false**. This truth-value assignment tells us how to complete the specification of interpretation \mathcal{A} . The pairs in $\{0, 1\} \times \{0, 1\}$ that should be in the interpretation $p_{\mathcal{A}}$ of p are all and only the pairs (i, j) such that $p(i, j)$ evaluates to **true**. Thus,

$$p_{\mathcal{A}} = \{(0, 1), (1, 0), (1, 1)\}.$$

27. Sometimes, in order to falsify a formula, we are forced to use an infinite structure. Consider falsifying the formula

$$\forall x \forall y [f(x) = f(y) \supset x = y] \supset \forall y \exists x [f(x) = y]$$

A structure \mathcal{A} that falsifies this formula must provide an interpretation for the function symbol f . We must have

$$\mathcal{A} \models \forall x \forall y [f(x) = f(y) \supset x = y]$$

while falsifying

$$\forall y \exists x [f(x) = y]$$

The formula $\forall x \forall y [f(x) = f(y) \supset x = y]$ asserts that f is a one-to-one function. The formula $\forall y \exists x [f(x) = y]$ asserts that f is onto. Since a 1-1 function from a *finite* set S to S that is one-to-one must also be onto, for it to be true that f is one-to-one but false that f is onto, it must be that $|\mathcal{A}|$ is not finite.

28. **Example:** “There are exactly two individuals.” Consider the formula **Two**:

$$\exists x \exists y [x \neq y \wedge \forall z [z = x \vee z = y]]$$

Then

$$\mathfrak{A} \models \mathbf{Two} \text{ iff } \text{size}|\mathfrak{A}| = 2$$

29. **Practice Problem:** Give a formula **Three** analogous to **Two**.
30. Here is a simple language: it has three constant symbols 0,1 and 2, one unary function symbol h and four predicate symbols **true**, **false**, q and $=$. We set up a structure \mathcal{S} for this language as follows: Let the universe of the structure be the set $\{a, b, c\}$, where a , b and c are three distinct individuals.
31. **Practice Problem:** To finish setting up \mathcal{S} , assign interpretations to the symbols of the language so that all of the following hold: 0,1 and 2 each have distinct interpretations, h is interpreted as a one-to-one monotonic function other than the identity function and q is a partial ordering with respect to which $\llbracket 0 \rrbracket$ is the least element, but the partial ordering has no greatest element. [Note: A function $f : D \rightarrow D$, where (D, \sqsubseteq) is a partial ordering, is monotonic on (D, \sqsubseteq) iff for all $x, y \in D$, if $x \sqsubseteq y$, then $f(x) \sqsubseteq f(y)$].

32. Expand the following formula over the set $\{a, b, c\}$ and evaluate it on the structure you set up in the previous practice problem.

$$\exists x \forall y \neg q(x, h(x))$$

33. **Definition of Prenex Form.** A formula is in *prenex form* iff it is closed and has the form

$$Q_1 x_1 \dots Q_n x_n \varphi$$

where each Q_i is either \forall or \exists , each x_i is a variable, and φ does not contain any quantifiers. The variables x_1, \dots, x_n must be pairwise distinct, i.e. no variable can appear more than once in the sequence. Every closed formula is equivalent to some formula that is in prenex form. $Q_1 x_1 \dots Q_n x_n$ is called the *quantifier prefix* and φ is called the *matrix*.

34. To find a formula in prenex form that is equivalent to a given closed formula Ψ we have to “factor out” the quantifiers in Ψ . To do that we use *equivalences from propositional logic* together with three rules for manipulating quantifiers:

- (a) (α -conversion)

$$Qx \varphi \leftrightarrow Qy \varphi'$$

where φ' is obtained from φ by replacing each free occurrence of x in φ by y and φ can be recovered from φ' by replacing each free occurrence of y in φ' by x . (This version of the requirements for substituting variables must be satisfied is due to David Jakel.)

- (b)

$$(\psi \rightarrow Qx \varphi) \leftrightarrow Qx (\psi \rightarrow \varphi)$$

provided x does not have a free occurrence in ψ .

- (c)

$$Qx \neg \varphi \leftrightarrow \neg \bar{Q}x \varphi$$

where $\bar{\exists}$ is \forall and $\bar{\forall}$ is \exists .

35. **Problem:** Explain why an α -conversion cannot be performed on

$$\forall x [p(x) \vee \exists y q(x, y)]$$

where the variable y is substituted for the variable x .

36. **Problem:** Find a formula in prenex form that is equivalent to

$$\forall y \forall x [p(x, y) \vee (\neg \exists y q(y) \rightarrow \forall x p(y, x))]$$

37. **“Skolemizing”.** Suppose that we have a formula in prenex form. When we Skolemize we seek to eliminate all of the existential quantifiers from the quantifier prefix. Each existentially quantified variable is replaced by a new function of all the preceding universally quantified variables in the prefix, and then existential quantifiers are removed.

38. **Example of how to Skolemize.** Consider

$$\exists x \forall y \exists w \exists z \forall u \forall v \exists x' \Psi(x, y, w, z, u, v, x')$$

We replace x by a new function of all of the preceding universally quantified variables. In this case, there are no such variables, the new function is 0-ary, i.e. a new constant added to the background language. In practice, we pick a new constant symbol not previously used in the example, theory, or proof we are working on.

We will choose a to obtain

$$\forall y \exists w \exists z \forall u \forall v \exists x' \Psi(a, y, w, z, u, v, x')$$

Next, we replace w by a new function of y . Suppose that, for example, the function symbol f occurs in $\Psi(a, y, w, z, u, v, x')$ but that g does not occur. We can therefore use g as a unary (i.e. 1-ary) function symbol and replace w by $g(y)$ to obtain

$$\forall y \exists z \forall u \forall v \exists x' \Psi(a, y, g(y), z, u, v, x')$$

We next introduce another new unary function symbol, e.g. g' , and replace z by $g'(y)$ to obtain

$$\forall y \forall u \forall v \exists x' \Psi(a, y, g(y), g'(y), u, v, x')$$

Lastly, we introduce a new 3-ary function symbol, e.g. h , and replace x' by $h(y, u, v)$ to obtain

$$\forall y \forall u \forall v \Psi(a, y, g(y), g'(y), u, v, h(y, u, v))$$

Here is the example redone more concretely, where for example, $\Psi(x, y, w, z, u, v, x')$ is the formula

$$p(x, f(w)) \rightarrow (q(y, w, u, v) \vee r(x', a, y, x))$$

Notice that z doesn't happen to actually occur in $\Psi(x, y, w, z, u, v, x')$ despite what the notation appears to indicate.

We are assuming that we are now starting with

$$\exists x \forall y \exists w \exists z \forall u \forall v \exists x' [p(x, f(w)) \rightarrow (q(y, w, u, v) \vee r(x', a, y, x))]$$

This time, since the constant symbol a already occurs in the formula we are working on, we start by replacing x with a new constant, e.g. b .

After completely Skolemizing, we obtain

$$\forall y \forall u \forall v [p(b, f(g(y))) \rightarrow (q(y, g(y), u, v) \vee r(h(y, u, v), a, y, b))]$$

39. **Problem:** Verify the result of Skolemizing with which we concluded the immediately preceding example.
40. **Problem:** Skolemize this:

$$\exists x \forall y \forall z \exists w [p(a, f(y, w)) \wedge q(b, x)]$$

Notice that z does not occur in the matrix to begin with, but be careful about what the Skolemizing procedure gives you.

41. **Definition (theory):** A *theory* T is a pair consisting of a language L for FOL and a set of formulas Γ of L . L is called the *language of* T and is denoted by $L(T)$. Γ is called the set of *nonlogical axioms* of T and is denoted by $\text{NlAx}(T)$.
42. The language of theory called *Robinson arithmetic*, (denoted by \mathbf{Q}) is the language of arithmetic. The nonlogical axioms of \mathbf{Q} are:

- Q1. $s(x_1) = s(x_2) \rightarrow x_1 = x_2$
 Q2. $\neg(0 = s(x_1))$
 Q3. $\neg(x_1 = 0) \rightarrow \exists x_2 (x_1 = s(x_2))$
 Q4. $x_1 + 0 = x_1$
 Q5. $x_1 + s(x_2) = s(x_1 + x_2)$
 Q6. $x_1 * 0 = 0$
 Q7. $x_1 * s(x_2) = (x_1 * x_2) + x_1$
 Q8. $x_1 \leq x_2 \leftrightarrow \exists x_3 (x_1 + x_3 = x_2)$

43. **Problem:** Show that Q6 and Q7 are valid in Robinson's nonstandard model.
44. **Problem:** Show that $x + y = y + x$ is not valid in Robinson's nonstandard model.
45. The reason we care about Robinson Arithmetic is that it turns out that in a sense that can be made rigorous and precise, the problem of computing a computable function, defined by code written in e.g. any programming language, can be made represented as a theorem proving task in \mathcal{Q} . So, for example, any C-program with its input can be compiled to a single goal in the language of arithmetic that is to be computed by showing the resulting tableau to be valid in \mathcal{Q} . Theory \mathcal{Q} is a minimal theory with this property. In other words, all computation can be represented in \mathcal{Q} , but this is not so for any theory strictly weaker than \mathcal{Q} . This further shows that familiar laws of arithmetic such as the commutativity of addition are *not needed for computation!*
46. Nevertheless, we would still like to have a theory of arithmetic in which we are able to prove familiar laws of arithmetic such as the fact that $\forall x [0 + x = x]$ - which cannot be proved in \mathcal{Q} . This deficiency of \mathcal{Q} can be remedied by Peano Arithmetic.
47. *Unification:* We will proceed to illustrate unification by studying some examples.
48. Find all solutions of the equation

$$k(f(x, u, a), f(g(y), y, v)) = k(f(g(y), y, v), f(w, h(z), z))$$

that hold regardless of the meaning of the constant and function symbols a, f, g, h and k . (In the preceding equation, a is a constant, and x, y, z, u, v and w are variables.)

Answer: $k(f(x, u, a), f(g(y), y, v)) = k(f(g(y), y, v), f(w, h(z), z))$

$$f(x, u, a) = f(g(y), y, v)$$

$$f(g(y), y, v) = f(w, h(z), z)$$

$$x = g(y)$$

$$u = y$$

$$\begin{aligned}
a &= v \\
g(y) &= w \\
y &= h(z) \\
v &= z
\end{aligned}$$

$$\begin{aligned}
x &= g(h(z)) \\
u &= h(z) \\
v &= a \\
w &= g(h(z)) \\
y &= h(z) \\
z &= a
\end{aligned}$$

$$\begin{aligned}
x &= g(h(a)) \\
u &= h(a) \\
v &= a \\
w &= g(h(a)) \\
y &= h(a) \\
z &= a
\end{aligned}$$

$$k(f(g(h(a)), h(a), a), f(g(h(a)), h(a), a)) = k(f(g(h(a)), h(a), a), f(g(h(a)), h(a), a))$$

49. Find a most general unifier of the two terms

$$g(x, y, f(x), z, u) \text{ and } g(a, x, z, f(x), h(v, z)).$$

Answer:

$$g(x, y, f(x), z, u) = g(a, x, z, f(x), h(v, z))$$

$$\begin{aligned}
x &= a \\
y &= x \\
f(x) &= z \\
z &= f(x) \\
u &= h(v, z)
\end{aligned}$$

$$\begin{aligned}
x &= a \\
y &= a \\
f(a) &= z \\
z &= f(a)
\end{aligned}$$

$$u = h(v, z)$$

$$x = a$$

$$y = a$$

$$z = f(a)$$

$$z = f(a)$$

$$u = h(v, z)$$

$$x = a$$

$$y = a$$

$$z = f(a)$$

$$f(a) = f(a)$$

$$u = h(v, f(a))$$

$$x = a$$

$$y = a$$

$$z = f(a)$$

$$a = a$$

$$u = h(v, f(a))$$

$$x = a$$

$$y = a$$

$$z = f(a)$$

$$u = h(v, f(a))$$

check: $g(a, a, f(a), f(a), h(v, f(a))) = g(a, a, f(a), f(a), h(v, f(a)))$

50.

Exam Practice Questions WITH ANSWERS

Here are problems to practice for the final exam.

Problem 1) Find a structure that shows that $p \supset q$ is not a theorem of the propositional theory whose only nonlogical axiom is $p \supset (q \vee r)$.

Answer: The convention we adopted for our notation is that lower-case letters p , q , and r denote atoms. Thus, we can choose a structure ν such that $\nu(p) = \nu(r) = T$ and $\nu(q) = F$. Then, $\llbracket p \supset (q \vee r) \rrbracket \nu = T$, but $\llbracket p \supset q \rrbracket \nu = F$.

Problem 2) Give a formula in prenex form equivalent to

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

Answer: $\forall y \exists x \forall w \exists z [p(x, y) \supset p(w, z)].$

Problem 3a) “Prenex” and Skolemize as assertions the two formulas in the set Γ given below.

$$\{\exists y \forall x p(x, y), \neg \forall w \exists z p(w, z)\}$$

Answer: Let a and b be new constant symbols. Then the set of formulas that results from prenexing and Skolemizing as assertions the formulas in Γ is:

$$\{\forall x p(x, a), \forall z \neg p(b, z)\}$$

Problem 3b) Skolemize as goals the two formulas in the set Γ below.

$$\{\exists y \forall x p(x, y), \neg \forall w \exists z p(w, z)\}$$

Answer: Let g and h be new unary function symbols. Then the set of formulas that results from prenexing and Skolemizing as assertions the formulas in Γ is:

$$\{\exists y p(g(y), y), \exists w p(w, h(w))\}$$

Problem 3c) Consider a theory whose nonlogical axioms are the formulas in the set you gave as the answer in part a. Use tableau methods to prove that the theory is inconsistent; i.e. import the nonlogical axioms as assertions into the tableau

$g1.$ false

and prove that the resulting tableau is valid.

Answer: A formula φ is provable in a theory T (written $T \vdash \varphi$) iff the tableau

$g1.$ φ

is valid relative to theory T .

	g1. false	given
a1.	$\forall x p(x, a)$	Nonlogical axiom
a2.	$\forall z \neg p(b, z)$	Nonlogical axiom
a3.	$p(x, a)$	1, \forall elimination
a4.	$p(b, a)$	3, Substitution, $\{x := b\}$
a5.	$\neg p(b, z)$	2, \forall elimination
a6.	$\neg p(b, a)$	5, Substitution, $\{z := a\}$
a7.	false	4,6, Propositional logic

The derived tableau is valid because it has the assertion false.

Problem 4) The Boolean connective NAND is symbolized by $|$. In other words, $\text{NAND}(p,q)$ is written $p | q$.

Show how to define exclusive-or in terms of $|$.

Some helpful information and hints: Exclusive-or(P, Q) is written $P + Q$.

Note that $P | Q$ is false if P and Q are both true. Otherwise $P | Q$ is true. $P + Q$ is true if P is true, or Q is true, but P and Q are not both true.

Define \vee in terms of $|$. Define \wedge in terms of $|$. Define \neg in terms of $|$.

Answer:

$$\begin{aligned}\neg P &:= P | P \\ P \vee Q &:= (P | P) | (Q | Q) \\ P \wedge Q &:= (P | Q) | (P | Q)\end{aligned}$$

Now, since

$$P + Q \leftrightarrow ((P \vee Q) \wedge \neg(P \wedge Q))$$

we can define $P + Q$ in terms of $|$ by

$$P + Q := (((P | P) | (Q | Q)) | (P | Q)) | (((P | P) | (Q | Q)) | (P | Q))$$

Problem 5) Find a prenex form equivalent to

$$\exists x[p(x) \wedge \forall y[\forall x[q(y, x)] \supset y = x]]$$

Answer:

$$\exists x \forall y \exists x_1 [p(x) \wedge (q(y, x_1) \supset y = x)]$$

Problem 6) Skolemize as an assertion:

$$(**) \quad \forall x \exists y \forall z \exists w [p(x, f(a, w), f(x, f(z, y)))] .$$

Answer:

$$\forall x \forall z [p(x, f(a, h(x, z)), f(x, f(z, g(x))))]$$

Note: We are assuming that the function symbols g and h are new; i.e. g and h are not function symbols of the language of which $(**)$ is a formula.

Problem 7) Skolemize as an assertion:

$$\exists x \forall y \exists z \forall w [p(x, f(a, w), f(x, f(z, y)))] .$$

Answer:

$$\forall y \forall w [p(b, f(a, w), f(b, f(k(y), y)))]$$

where b is a new constant symbol, and k is a new function symbol.

Problem 8) Consider

$$g1. \exists x [p(x) \wedge \forall y [p(y) \supset q(x, y)]] \supset \exists x [p(x) \vee q(x, x)]$$

Prove that the tableau is valid by tableau methods.

Answer:

$g1. \exists x [p(x) \wedge \forall y [p(y) \supset q(x, y)]] \supset \exists x [p(x) \vee q(x, x)]$	given
a1. $\exists x [p(x) \wedge \forall y [p(y) \supset q(x, y)]]$	g1, if-split
g2. $\exists x [p(x) \vee q(x, x)]$	g1, if-split
a2. $\exists x \forall y [p(x) \wedge (p(y) \supset q(x, y))]$	a1, prenex
a3. $\forall y [p(a) \wedge (p(y) \supset q(a, y))]$	a2, Skolemization
a4. $p(a) \wedge (p(y) \supset q(a, y))$	a3, \forall elimination
g3. $p(x) \vee q(x, x)$	g2, \exists -elimination
a5. $\neg(p(x) \vee q(x, x))$	g3, duality
a6. $p(a) \wedge (p(a) \supset q(a, a))$	a4, Substitution, $\{y := a\}$
a7. $\neg(p(a) \vee q(a, a))$	a5, Substitution, $\{x := a\}$
7. false	a6,a7, Propostional logic

Problem 9) Intentionally omitted.

Problem 10) Consider

$$\text{PA} \vdash \neg(x = s(x))$$

Part a) What is the induction axiom for

$$\neg(x = s(x))$$

Answer:

$$(\neg(0 = s(0)) \wedge \forall y[\neg(y = s(y)) \supset \neg(s(y) = s(s(y)))]) \supset \neg(x = s(x)))$$

Part b) Give a formal proof that shows

$$\text{PA} \vdash \neg(x = s(x))$$

Answer:

Lemma A: (Base step:) $\text{PA} \vdash \neg(0 = s(0))$.

- | | |
|-----------------------|--------------------------------|
| 1. $\neg(0 = s(x_1))$ | Q2 |
| 2. $\neg(0 = s(0))$ | 1, Substitution $\{x_1 := 0\}$ |

This completes the proof of Lemma A.

Lemma B: (Induction Step)

$$\text{PA} \vdash \forall y[\neg(y = s(y)) \supset \neg(s(y) = s(s(y)))]$$

- | | |
|--|--|
| g1. $\forall y[\neg(y = s(y)) \supset \neg(s(y) = s(s(y)))]$ | given |
| g2. $\neg(a = s(a)) \supset \neg(s(a) = s(s(a)))$ | g1, Skolemization |
| a1. $\neg(a = s(a))$ | g2, if-split // This is the induction assumption |
| g3. $\neg(s(a) = s(s(a)))$ | g2, if-split |
| a2. $s(x_1) = s(x_2) \supset x_1 = x_2$ | Q1 |
| a3. $s(a) = s(s(a)) \supset a = s(a)$ | Substitution $\{x_1 := 0; x_2 := s(a)\}$ |
| g4. $s(a) = s(s(a)) \supset a = s(a)$ | g2, Propositional Logic |
| g5. True | a3, g4, assertion=goal |

This completes the proof of Lemma B.

g1. $\neg(x = s(x))$	given
a1. $(\neg(0 = s(0)) \wedge \forall y[\neg(y = s(y)) \supset \neg(s(y) = s(s(y)))] \supset \neg(x = s(x)))$	induction axiom
a2. $\neg(0 = s(0))$	Lemma A
a3. $\forall y[\neg(y = s(y)) \supset \neg(s(y) = s(s(y)))]$	Lemma B
a4. $\neg(x = s(x))$	a1, a3, Modus Ponens)
g2. True	a4, g1, assertion=goal

Problem 11) Find a structure in which the following equivalence is not valid:

$$((\forall x p(x)) \supset q(a)) \leftrightarrow (\forall x (p(x) \supset q(a)))$$

Answer: Let L be a language whose nonlogical symbols are the predicate symbols p , q and whose only function symbol is the constant symbol a . Let \mathcal{M} be a structure for L in which there are two individuals i and j . Let $a_{\mathcal{M}} = i$, $p_{\mathcal{M}} = \{i\}$ and $q_{\mathcal{M}} = \{j\}$. Then

$$\mathcal{M} \models \neg \forall x p(x)$$

Therefore,

$$\mathcal{M} \models \forall x p(x) \supset q(a)$$

But,

$$\mathcal{M} \models p(a)$$

and

$$\mathcal{M} \models \neg q(a)$$

Therefore,

$$\mathcal{M} \not\models p(a) \supset q(a)$$

Therefore,

$$\mathcal{M} \not\models \forall x [p(x) \supset q(x)]$$

Therefore,

$$\mathcal{M} \not\models ((\forall x p(x)) \supset q(a)) \leftrightarrow (\forall x (p(x) \supset q(a)))$$

Appendix

Valid Sentence Schemata

Basic:	$F \equiv F$	$F \vee \neg F$	$F \supset F$	$F \wedge G \supset F$	$F \supset F \vee G$		
True-False:	$true$	$\neg false$	$F \vee true$	$\neg(F \wedge false)$	$false \supset F$	$F \supset true$	$F \vee false \equiv F$
	$F \wedge true \equiv F$	$true \supset F \equiv F$	$true?F : G \equiv F$	$false?F : G \equiv G$	$(true \equiv F) \equiv F$	$(false \equiv F) \equiv \neg F$	
Commutativity:	$F \wedge G \equiv G \wedge F$	$F \vee G \equiv G \vee F$	$(F \equiv G) \equiv (G \equiv F)$				
Associativity:	$(F \wedge G) \wedge H \equiv F \wedge (G \wedge H)$	$(F \vee G) \vee H \equiv F \vee (G \vee H)$	$((F \equiv G) \equiv H) \equiv (F \equiv (G \equiv H))$				
Reduction:	$(F \wedge F) \equiv F$	$(F \vee F) \equiv F$	$(F \wedge (F \vee G)) \equiv F$	$(F \vee (F \wedge G)) \equiv F$	$F?G : G \equiv G$		
Transitivity:	$((F \supset G) \wedge (G \supset H)) \supset (F \supset H)$	$((F \equiv G) \wedge (G \equiv H)) \supset (F \equiv H)$					
Contrapositive:	$(F \supset G) \equiv (\neg G \supset \neg F)$	$(F \equiv G) \equiv (\neg F \equiv \neg G)$					
Distributivity:	$(F \wedge (G \vee H)) \equiv (F \wedge G) \vee (F \wedge H)$	$(F \vee (G \wedge H)) \equiv (F \vee G) \wedge (F \vee H)$					
	$(F \vee G) \supset H \equiv (F \supset H) \wedge (G \supset H)$	$F \supset (G \vee H) \equiv (F \supset G) \vee (F \supset H)$	$(F \wedge G) \supset H \equiv (F \supset H) \vee (G \supset H)$				
	$F \supset (G \wedge H) \equiv (F \supset G) \wedge (F \supset H)$	$(F \wedge G) \supset H \equiv F \supset (G \supset H)$					
Negation:	$\neg(\neg F) \equiv F$	$\neg(F \wedge G) \equiv \neg F \vee \neg G$	$\neg(F \vee G) \equiv \neg F \wedge \neg G$	$\neg(F \supset G) \equiv F \wedge \neg G$			
	$\neg(F?G : H) \equiv F?\neg G : \neg H$	$\neg(F \equiv G) \equiv (F \equiv \neg G)$					
Connective Elimination:	$F \supset G \equiv \neg F \vee G$	$F?G : H \equiv (F \wedge G) \vee (\neg F \wedge H)$	$F?G : H \equiv (F \supset G) \wedge (\neg F \supset H)$				
	$(F \equiv G) \equiv (F \wedge G) \vee (\neg F \wedge \neg G)$	$(F \equiv G) \equiv (F \supset G) \wedge (G \supset F)$					

True/False Simplifications

Neg:	$\neg true \Rightarrow false$	$\neg false \Rightarrow true$		
Conj:	$F \wedge true \Rightarrow F$	$true \wedge F \Rightarrow F$	$F \wedge false \Rightarrow false$	$false \wedge F \Rightarrow false$
Disj:	$F \vee true \Rightarrow true$	$true \vee F \Rightarrow true$	$F \vee false \Rightarrow F$	$false \vee F \Rightarrow F$
Impl:	$true \supset G \Rightarrow G$	$false \supset G \Rightarrow true$	$F \supset true \Rightarrow true$	$F \supset false \Rightarrow \neg F$
Equiv:	$F \equiv true \Rightarrow F$	$true \equiv F \Rightarrow F$	$F \equiv false \Rightarrow \neg F$	$false \equiv F \Rightarrow \neg F$
Cond:	$true?G : H \Rightarrow G$	$false?G : H \Rightarrow H$	$F?true : H \Rightarrow F \vee H$	$F?false : H \Rightarrow \neg F \wedge H$
	$F?G : true \Rightarrow F \supset G$	$F?G : false \Rightarrow F \wedge G$		

Other Simplifications

Neg:	$\neg(\neg F) \Rightarrow F$	$\neg(\neg F \wedge \neg G) \Rightarrow F \vee G$	$\neg(\neg F \vee \neg G) \Rightarrow F \wedge G$	$\neg(F \supset \neg G) \Rightarrow F \wedge G$
	$\neg(\neg F \equiv G) \Rightarrow F \equiv \neg G$	$\neg(F \equiv \neg G) \Rightarrow F \equiv G$	$\neg(F?G : \neg H) \Rightarrow F?G : H$	
Conj:	$F \wedge F \Rightarrow F$	$F \wedge \neg F \Rightarrow false$	$\neg F \wedge F \Rightarrow false$	
Disj:	$F \vee F \Rightarrow F$	$F \vee \neg F \Rightarrow true$	$\neg F \vee F \Rightarrow true$	
Impl:	$F \supset F \Rightarrow true$	$\neg F \supset F \Rightarrow F$	$F \supset \neg F \Rightarrow \neg F$	$\neg G \supset \neg F \Rightarrow F \supset G$
Equiv:	$F \equiv F \Rightarrow true$	$F \equiv \neg F \Rightarrow false$	$\neg F \equiv F \Rightarrow false$	
Cond:	$F?G : G \Rightarrow G$	$\neg F?G : H \Rightarrow F?H : G$		

Rewritings

Negation:	$\neg(F \wedge G) \iff \neg F \vee \neg G$	$\neg(F \vee G) \iff \neg F \wedge \neg G$	$\neg(F \supset G) \iff F \wedge \neg G$
	$\neg(F \equiv G) \iff F \equiv \neg G$	$\neg(F?G : H) \iff F?\neg G : \neg H$	
Elimination:	$F \supset G \iff \neg F \vee G$	$F \equiv G \iff (F \supset G) \wedge (G \supset F)$	$F \equiv G \iff (F \wedge G) \vee (\neg F \wedge \neg G)$
	$F?G : H \iff (F \supset G) \wedge (\neg F \supset H)$	$F?G : H \iff (F \wedge G) \vee (\neg F \wedge H)$	
Commutativity:	$F \wedge G \iff G \wedge F$	$F \vee G \iff G \vee F$	$(F \equiv G) \iff (G \equiv F)$
Associativity:	$(F \wedge G) \wedge H \iff F \wedge (G \wedge H)$	$(F \vee G) \vee H \iff F \vee (G \vee H)$	$(F \equiv G) \equiv H \iff F \equiv (G \equiv H)$
Distributivity:	$(F \wedge (G \vee H)) \iff (F \wedge G) \vee (F \wedge H)$	$F \vee (G \wedge H) \iff (F \vee G) \wedge (F \vee H)$	

Polarity: $(\neg A^{-p})^p$ | $(A^p \wedge B^p)^p$ | $(A^p \vee B^p)^p$ | $(A^{-p} \supset B^p)^p$ | $(A^\pm \equiv B^\pm)^p$

Splitting Rules:

AND-Split	OR-Split	IF-Split
a1. $A_1 \wedge A_2$	g1. $G_1 \vee G_2$	g1. $A \supset G$
a2. A_1	g2. G_1	a1. A
a3. A_2	g3. G_2	g2. G
		g1, \vee -split
		g1, \vee -split
		g1, \supset -split
		g1, \supset -split

Resolutions:

AA	GG
a1. $A_1 [P^-]$	g1. $G_1 [P^-]$
a2. $A_2 [P^+]$	g2. $G_2 [P^+]$
a3. $A_1 [false] \vee A_2 [true]$	g3. $G_1 [false] \wedge G_2 [true]$
	g1, g2, GG-res
AG	GA
a1. $A [P^-]$	a1. $A [P^+]$
g1. $G [P^+]$	g1. $G [P^-]$
g2. $\neg A [false] \wedge G [true]$	g2. $\neg A [true] \wedge G [false]$
a1, g1, AG-res	g1, a1, GA-res