Ma/CS 6c Notes

Alexander S. Kechris
with Michael A. Shulman
and Andrés E. Caicedo

April 2, 2008

Contents

1 Propositional Logic **1**

1.1 Introduction 1

1.2 Syntax of propositional logic 4

1.2.A The Language of Propositional Logic 4

1.2.B Induction on Formulas 6

1.2.C Unique Readability 7

1.2.D Recursive Definitions 8

1.2.E Recognizing Formulas 11

1.3 Polish and Reverse Polish notation 13

1.3.A Polish Notation 13

1.3.B Reverse Polish notation 16

1.4 Abbreviations 21

1.5 Semantics of Propositional Logic 21

1.5.A Valuations 21

1.5.B Models and Tautologies 25

1.5.C Logical Implication and Equivalence 28

1.6 Truth Functions 31

1.6.A Truth Functions 31

1.6.B Completeness of Binary Connectives 37

1.6.C Normal Forms 38

1.7 König's Lemma and Applications 41

1.7.A Graphs and Trees 41

1.7.B König's Lemma 43

1.7.C Domino Tilings 45

1.7.D Compactness of $[0, 1]$ 49

1.8 The Compactness Theorem 50

1.8.A The Compactness Theorem 50

1.8.B A Proof of König's Lemma 51
1.8.C Partial Orders 52
1.9 Ramsey Theory 55
1.9.A Ramsey Theorem, infinite version 55
1.9.B Ramsey Theorem, finite version 56
1.9.C A few other applications of compactness 57
1.10 The Resolution Method 58
1.11 A Hilbert-Type Proof System 66
1.11.A Formal Proofs 67
1.11.B Soundness 69
1.11.C Completeness 70
1.11.D General Completeness 73

2 First-Order Logic 75
2.1 Structures 75
2.1.A Relations and Functions 75
2.1.B Structures 77
2.1.C Introduction to First-Order Languages 80
2.2 Syntax of First-Order Logic 81
2.2.A Symbols 81
2.2.B Terms 82
2.2.C Well-Formed Formulas 84
2.2.D Scope and Free Variables 87
2.3 Semantics of First-Order Logic 88
2.3.A Structures and Interpretations 88
2.3.B Models and Validities 90
2.4 Definability in a Structure 96
2.4.A First-Order Definability 96
2.4.B Isomorphisms 100
2.5 Prenex Normal Forms and Games 103
2.5.A Prenex Normal Forms 103
2.5.B Games and Strategies 107
2.6 Theories 110
2.7 A Proof System for First-Order Logic 115
2.7.A Formal Proofs 115
2.7.B Examples of Formal Proofs 118
2.7.C Metatheorems 119
2.8 The Gödel Completeness Theorem 123

2.9 The Compactness Theorem 131
2.9.A Finiteness and Infinity 132
2.9.B Non-standard Models of Arithmetic 133

3 Computability and Complexity 139
3.1 Introduction 139
3.2 Decision Problems 140
3.3 Turing Machines 142
3.4 The Church-Turing Thesis 145
3.4.A Register Machines 145
3.4.B The Church-Turing Thesis 148
3.5 Universal Machines 149
3.6 The Halting Problem 150
3.7 Undecidability of the Validity Problem 151
3.8 The Hilbert Tenth Problem 156
3.9 Decidable Problems 156
3.10 The Class $\mathcal{P}$ 158
3.11 The Class $\mathcal{NP}$ and the $\mathcal{P} = \mathcal{NP}$ Problem 160
3.12 $\mathcal{NP}$-Complete Problems 161

A Tautologies and equivalences 167
A.1 Tautologies 167
A.2 Equivalences 169

B Validities in First Order Logic 171

Chapter 1

Propositional Logic

1.1 Introduction

A *proposition* is a statement which is either true or false.

Examples 1.1.1
"There are infinitely many primes"; "$5 > 3$"; and "14 is a square number" are propositions. A statement like "x is odd," however, is *not* a proposition (but it becomes one when x is substituted by a particular number).

A *propositional connective* is a way of combining propositions so that the truth or falsity of the compound proposition depends only on the truth or falsity of the components. The most common connectives are:

connective	*symbol*
not (negation)	$\neg$
and (conjunction)	$\wedge$
or (disjunction)	$\vee$
implies (implication)	$\Rightarrow$
iff (equivalence)	$\Leftrightarrow$

Notice that each connective has a symbol which is traditionally used to represent it. This way we can distinguish between the desired precise mathematical meaning of the connective (e.g. $\neg$) and the possibly vague or ambiguous meaning of an english word (e.g. "implies"). The intended mathematical meanings of the connectives are as follows:

- "Not" ($\neg$) is the only common *unary connective*: it applies to only one proposition, and negates its truth value. The others are *binary connectives* that apply to two propositions:
- A proposition combined with "and" ($\wedge$) is true only if *both* of its components are true, while...
- A proposition combined with "or" ($\vee$) is true whenever *either or both* of its components are true. That is, $\vee$ is the so-called "inclusive or".
- The "iff" (short for for "if and only if") or equivalence connective ($\Leftrightarrow$) is true whenever both of its components have the same truth value: either both true or both false.
- The "implication" connective is the only common binary connective which is not symmetric: $p \Rightarrow q$ is not the same as $q \Rightarrow p$. $p \Rightarrow q$ is intended to convey the meaning that whenever p is true, q must also be. So if p is true, $p \Rightarrow q$ is true if and only if q is also true. If, on the other hand, p is false, we say that $p \Rightarrow q$ is "vacuously" true. The statement that $p \Rightarrow q$ does not carry any "causal" connotation, unlike the English phrases we use (for lack of anything better) to pronounce it (such as "p implies q" or "if p, then q").

We can summarize the meanings of these connectives using *truth tables*:

p	$\neg p$
T	F
F	T

p	q	$p \wedge q$	$p \vee q$	$p \Rightarrow q$	$p \Leftrightarrow q$
T	T	T	T	T	T
T	F	F	T	F	F
F	T	F	T	T	F
F	F	F	F	T	T

Examples 1.1.2
"$\neg 2 = 2$" is false;
"$25 = 5^2 \wedge 6 > 5$" is true;
"5 is even $\vee$ 16 is a square" is true;
"5 is odd $\vee$ 16 is a square" is true;

"(the square of an odd number is odd)$\Rightarrow$(25 is odd)" is true;
"(the square of an odd number is even)$\Rightarrow$(25 is odd)" is true;
"(the square of an odd number is odd)$\Rightarrow$(10 is odd)"is false.

There are many different ways to express a compound statement, and corresponding to these there are different ways to combine propositions using these connectives which are *equivalent*: their truth values are the same for any truth values of the original propositions. Here are some examples:

(i) The statements "The cafeteria either has no pasta, or it has no sauce" and "The cafeteria doesn't have both pasta and sauce" are two ways to say the same thing. If $p =$ "The cafeteria has pasta" and $q =$ "The cafeteria has sauce", then this means that

$$(\neg p) \vee (\neg q) \quad \text{and} \quad \neg(p \wedge q)$$

are equivalent. This is one of what are known as *De Morgan's Laws*.

(ii) The *law of the double negative* says that $\neg\neg p$ and p are equivalent. That is, if something is not false, it is true, and vice versa.

(iii) The statements "If it rained this morning, the grass is wet" and "If the grass is dry, it didn't rain this morning" are two more ways to say the same thing. Now if $p =$ "It rained this morning" and $q =$ "The grass is wet", then this means that

$$(p \Rightarrow q) \quad \text{and} \quad (\neg q) \Rightarrow (\neg p)$$

are equivalent. The latter statement is called the *contrapositive* of the first. Note that because of the law of the double negative, the first statement is also equivalent to the contrapositive of the second.

(iv) Yet another way to say the same thing is "Either it did not rain this morning, or the grass is wet." That is, another proposition equivalent to those above is

$$(\neg p) \vee q.$$

One consequence of this is that we could, if we wanted to, do without the $\Rightarrow$ connective altogether. We will see more about this in section 1.6.B.

(v) Two more related statements which are *not* equivalent to $p \Rightarrow q$ are the *converse*, $q \Rightarrow p$ ("If the grass is wet, it rained this morning"), and the *inverse*, $(\neg p) \Rightarrow (\neg q)$ ("If it didn't rain this morning, the grass isn't wet"). There might, for example, be sprinklers that keep the grass wet even when it doesn't rain. The converse and inverse are contrapositives of each other, however, so they have the same truth value.

In section 1.5 we will revisit this notion of "equivalence" in a formal context.

1.2 Syntax of propositional logic

In order to rigorously prove results about the structure and truth of propositions and their connectives, we must set up a mathematical structure for them. We do this with a *formal language*, the language of *propositional logic*, which we will work with for the rest of the chapter.

1.2.A The Language of Propositional Logic

Definition 1.2.1 A *formal language* consists of a set of symbols together with a set of rules for forming "grammatically correct" strings of symbols in this language.

Definition 1.2.2 In the *language of propositional logic* we have the following list of symbols:

(i) *propositional variables*: this is an infinite list $p_1, p_2, p_2, \ldots$ of symbols. We often use $p, q, r, \ldots$ to denote propositional variables.

(ii) *symbols for the (common) propositional connectives*: $\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow$.

(iii) *parentheses*: (,).

Definition 1.2.3 A *string* or *word* in a formal language is any finite sequence of the symbols in the language. We include in this the *empty string* containing no symbols at all.

Example 1.2.4 The following are examples of strings in the language of propositional logic:

$$p\vee,\ pq\Rightarrow,\ pqr,\ (p\wedge q),\ p)$$

Definition 1.2.5 If $S = s_1 \dots s_n$ is a string with n symbols, we call n the *length* of S and denote it by $|S|$. The length of the empty string is 0.

We will next specify the rules for forming "grammatically correct" strings in propositional logic, which we call *well-formed formulas* (*wff*s) or just *formulas*.

Definition 1.2.6 (Well-Formed Formulas)

(i) Every propositional variable p is a wff.

(ii) If A is a wff, so is $\neg A$.

(iii) If A, B are formulas, so are $(A \wedge B)$, $(A \vee B)$, $(A \Rightarrow B)$, and $(A \Leftrightarrow B)$.

Thus a string A is a wff exactly when there is a finite sequence

$$A_1, \dots, A_n$$

(called a *parsing sequence*) such that $A_n = A$ and for each $1 \le i \le n$, A_i is either (1) a propositional variable, (2) for some $j < i$, $A_i = \neg A_j$, or (3) for some $j, k < i$, $A_i = (A_j * A_k)$, where $*$ is one of $\wedge, \vee, \Rightarrow, \Leftrightarrow$.

Examples 1.2.7

(i) $(p \Rightarrow (q \Rightarrow r))$ is a wff with parsing sequence

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

(Also note that

$$q,\ r,\ (q \Rightarrow r),\ p,\ (p \Rightarrow (q \Rightarrow r))$$

is another parsing sequence, so parsing sequences are not unique.)

(ii) $(\neg(p \wedge q) \Leftrightarrow (\neg p \vee \neg q))$ is a wff with parsing sequence

$$p,\ q,\ (p \wedge q),\ \neg(p \wedge q),\ \neg p,\ \neg q,\ (\neg p \vee \neg q),\ (\neg(p \wedge q) \Leftrightarrow (\neg p \vee \neg q)).$$

(iii) $p \Rightarrow qr$, $)p \Leftrightarrow$, $\vee(q\neg)$, and $p \wedge q \vee r$ are not wff, but to prove this is rather more tricky. We will see some ways to do this in section 1.2.B.

Remark. We have not yet assigned any "meaning" to any of the symbols in our formal language. While we intend to interpret symbols such as $\vee$ and $\Rightarrow$ eventually in a way analogous to the propositional connectives "or" and "implies" in section 1.1, at present they are simply symbols that we are manipulating formally.

1.2.B Induction on Formulas

We can prove properties of formulas by induction on the length of a formula. Suppose $\Phi(A)$ is a property of a formula A. For example, $\Phi(A)$ could mean "A has the same number of left and right parentheses." If $\Phi(A)$ holds for all formulas of length 1 (i.e. the propositional variables) and whenever $\Phi(A)$ holds for all formulas A of length $\leq n$, then it holds for all formulas of length $n+1$, we may conclude, by the principle of mathematical induction, that $\Phi(A)$ holds for *all* formulas.

However, because of the way formulas are defined, it is most useful to use another form of induction, sometimes called *induction on the construction of wff* (or just *induction on formulas*):

Let $\Phi(A)$ be a property of formulas A. Suppose:

(i) *Basis of the induction*: $\Phi(A)$ holds, when A is a propositional variable.

(ii) *Induction step*: If $\Phi(A), \Phi(B)$ hold for two formulas A, B, then

$$\Phi(\neg A), \Phi((A * B))$$

hold as well, where $*$ is one of the binary connectives $\wedge, \vee, \Rightarrow, \Leftrightarrow$.

Then $\Phi(A)$ holds for *all* formulas A. We can prove the validity of this procedure using standard mathematical induction, and the existence of a parsing sequence for any wff.

Example 1.2.8 One can easily prove, using this form of induction, that every formula A has the same number of left and right parentheses. This result can be used to show that certain strings, for example $)p \Leftrightarrow$, are not wffs, because they have unequal numbers of left and right parentheses.

Example 1.2.9 If $s_1 s_2 \ldots s_n$ is any string, then an *initial segment* of it is a string $s_1 \ldots s_m$, where $0 \leq m \leq n$ (when $m = 0$, this means the empty string). If $m < n$ we call this a *proper* initial segment.

Again we can easily prove by induction on the construction of wff that a nonempty proper initial segment of a formula either consists of a string of $\neg$'s or else has more left than right parentheses. (To see some examples, consider $\neg\neg(p \Rightarrow q), ((p \Rightarrow (q \wedge p)) \Rightarrow \neg p)$.) Another easy induction shows that no formula can be empty or just a string of $\neg$'s, so using also the previous example, *no proper initial segment of a wff is a wff.*

This can also be used to show certain strings (for example, $p \Rightarrow qr$) are not wffs, as they contain proper initial segments that are wffs (in the example, p). Similar methods can be used for other non-wffs.

1.2.C Unique Readability

While parsing sequences are not unique, as mentioned in examples 1.2.7, there is a sense in which both alternative parsing sequences offered are "the same": they break the wff down in the same way, to the same components, albeit in a different order. Using induction on formulas, we can prove that this will always be the case. This is called *unique readability.*

Theorem 1.2.10 (Unique Readability Theorem) *Every wff is in exactly one of the forms:*

(i) p *(i.e., a propositional variable);*

(ii) $\neg A$ *for* A *a uniquely determined wff;*

(iii) $(A * B)$*, for uniquely determined wff's* A, B *and uniquely determined* $* \in \{\wedge, \vee, \Rightarrow, \Leftrightarrow\}$.

Proof. By induction on the construction of formulas, it is straightforward that any formula must be of one of these forms. To prove uniqueness first notice that no wff can be in more than one of the forms (i), (ii), (iii). And obviously no two propositional variables are the same, and if $\neg A = \neg B$, then $A = B$.

So it is enough to show that in case (iii), if $(A * B) = (C \circ D)$, for $*, \circ \in \{\wedge, \vee, \Rightarrow, \Leftrightarrow\}$, then $A = C, B = D$ and $* = \circ$. First notice that, by eliminating the left parenthesis, we have that $A * B) = C \circ D)$. From this it follows that either A is an initial segment of C or vice versa. If $A \neq C$, then one of A, C is a proper initial segment of the other, which is impossible, as they are both formulas. So we must have $A = C$. Thus $*B) = \circ D)$ and so $* = \circ$ and $B = D$. $\dashv$

Definition 1.2.11 In form (ii) we call $\neg$ the *main connective* and in (iii) we call $*$ the *main connective*. (Thus the main connective is the one applied last.)

Example 1.2.12 In $((p \Rightarrow (q \Rightarrow p)) \Rightarrow q)$ the main connective is the third "$\Rightarrow$" (from left to right).

Given a formula A, it might be hard to recognize immediately which is the main connective. Consider for example the following formula:

$$\begin{aligned}&(((((p \vee \neg q) \vee \neg r) \wedge ((\neg p \vee (s \vee \neg r)) \wedge ((q \vee s) \vee p)))\wedge \\ &\qquad (((\neg q \vee \neg r) \vee s) \wedge (s \vee (\neg r \vee q)))) \wedge ((p \vee (\neg r \vee s)) \wedge (s \vee \neg r)))\end{aligned}$$

(it turns out to be the 5th "$\wedge$"). There is however an easy algorithm for determining the main connective: In the formula $(A*B)$ the main connective $*$ is the first binary connective (from left-to-right) such that in the string preceding it the number of left parentheses is exactly one more than the number of right parentheses.

1.2.D Recursive Definitions

Often we will want to define a function or characteristic of wffs, and the most natural way to do it is to break down the formula into simpler formulas one step at a time. To make this rigorous, we can use the unique readability theorem to justify the following principle of *definition by recursion on the construction of formulas*:

Let X be a set and

$$\begin{aligned} f &: \{p_1, p_2, \dots\} \to X \\ f_\neg &: X \to X \\ f_\wedge, f_\vee, f_\Rightarrow, f_\Leftrightarrow &: X^2 \to X \end{aligned}$$

be given functions. Then there is a unique function g from the set of all formulas into X such that

$$\begin{aligned} g(p) &= f(p) \\ g(\neg A) &= f_\neg(g(A)) \\ g((A * B)) &= f_*(g(A), g(B)), \quad * \in \{\wedge, \vee, \Rightarrow, \Leftrightarrow\}. \end{aligned}$$

Without unique readability, the existence and uniqueness of g would be called into question: if there were two different ways of breaking down a formula, then the corresponding compositions of the fs might yield different values of g for the same formula. But now that we know unique readibility, g is well-defined and unique.

Examples 1.2.13

(i) Let

$$\begin{aligned} L(p) &= 1 \\ L(\neg A) &= L(A) + 1 \\ L((A * B)) &= L(A) + L(B) + 3. \end{aligned}$$

Then it is easy to see that $L(A) = |A| =$ the length of A.

(ii) Let

$$\begin{aligned} C(P) &= 0 \\ C(\neg A) &= C(A) + 1 \\ C((A * B)) &= C(A) + C(B) + 1. \end{aligned}$$

Then $C(A) =$ the number of connectives in A.

(iii) Let p be a propositional variable and P a formula. We define for each formula A the formula $g(A) = A[p/P]$ as follows:

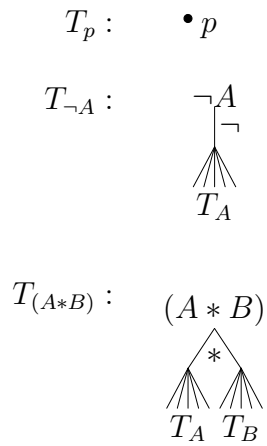
$$\begin{aligned} g(p) &= P, \\ g(q) &= q \quad \text{if } q \neq p, \\ g(\neg A) &= \neg g(A) \\ g((A * B)) &= (g(A) * g(B)). \end{aligned}$$

Then it is easy to see that $A[p/P]$ is the formula obtained by substituting every occurence of p in A by P.

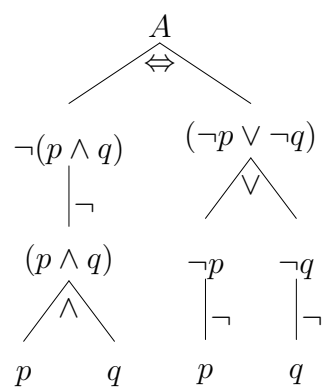
For example, if $A = (\neg p \Rightarrow (q \Rightarrow p))$ and $P = (r \vee q)$, then

$$A[p/P] = (\neg(r \vee q) \Rightarrow (q \Rightarrow (r \vee q))).$$

(iv) The *parse tree* T_A of a formula A is defined recursively as follows:



For example, if $A = (\neg(p \wedge q) \Leftrightarrow (\neg p \vee \neg q))$, then its parse tree is



The parse tree, starting now from the bottom up, gives us various ways of constructing a parsing sequence for the formula; for example:

$$p,\ q,\ (p \wedge q),\ \neg(p \wedge q),\ \neg p,\ \neg q,\ (\neg p \vee \neg q),\ (\neg(p \wedge q) \Leftrightarrow (\neg p \vee \neg q)).$$

By unique readability, all parsing sequences can be obtained in this manner. Thus the parse tree extracts the "common part" out of each parsing sequence.

(v) The formulas occurring in the parse tree of a formula A are called the *subformulas* of A. These are the formulas that arise in the construction of A. If we let $\mathrm{subf}(A)$ denote the set of subformulas of A, then subf can be defined by the following recursion:

$$\begin{aligned}\mathrm{subf}(p) &= \{p\}\\ \mathrm{subf}(\neg A) &= \mathrm{subf}(A) \cup \{\neg A\}\\ \mathrm{subf}\big((A * B)\big) &= \mathrm{subf}(A) \cup \mathrm{subf}(B) \cup \{(A * B)\}.\end{aligned}$$

1.2.E Recognizing Formulas

We will next discuss an algorithm for verifying that a given string is a formula.

Algorithm 1.2.14 Given a string S, let x be a propositional variable not occurring in S and let $S(x)$ the string we obtain from S by substituting every propositional variable by x.

For any string T containing only the variable x, let T' be the string obtained from T as follows: If T contains no *substring* (i.e. a string contained in T as a consecutive sequence of symbols) of the form $\neg x, (x \wedge x), (x \vee x), (x \Rightarrow x), (x \Leftrightarrow x)$, then $T' = T$. Otherwise substitute the leftmost occurence of any one of these forms by x, to obtain T'. Notice that if $T \neq T'$, then $|T'| < |T|$.

Now starting with an arbitrary string S, form first $S(x)$, and then define recursively $S_0 = S(x),\ S_{n+1} = S'_n$. This process stops when we hit n_0 for which $S_{n_0+1} = S'_{n_0} = S_{n_0}$, i.e., S_n has no substring of the above form. This must necessarily happen since otherwise we would have $|S_0| > |S_1| > \ldots$ (ad infinitum), which is impossible.

We now claim that if $S_{n_0} = x$, then S is a formula, otherwise it is not.

Examples 1.2.15

(i) $((p \Rightarrow (q \vee \neg p) \Rightarrow (r \vee (\neg s \Rightarrow t)))) = S$

$$\begin{array}{rcl}((x \Rightarrow (x \vee \neg x) \Rightarrow (x \vee (\neg x \Rightarrow x)))) & = & S(x) = S_0\\ ((x \Rightarrow (x \vee x) \Rightarrow (x \vee (\neg x \Rightarrow x)))) & = & S_1\\ ((x \Rightarrow x \Rightarrow (x \vee (\neg x \Rightarrow x)))) & = & S_2\\ ((x \Rightarrow x \Rightarrow (x \vee (x \Rightarrow x)))) & = & S_3\\ ((x \Rightarrow x \Rightarrow (x \vee x))) & = & S_4\\ ((x \Rightarrow x \Rightarrow x)) & = & S_5 = S_6\end{array}$$

So S is not a wff.

(ii) $((p \Rightarrow \neg q) \Rightarrow (p \vee (\neg r \wedge s))) = S$

$$\begin{array}{ccl} ((x \Rightarrow \neg x) \Rightarrow (x \vee (\neg x \wedge x))) & = & S(x) = S_0 \\ ((x \Rightarrow x) \Rightarrow (x \vee (\neg x \wedge x))) & = & S_1 \\ (x \Rightarrow (x \vee (\neg x \wedge x))) & = & S_2 \\ ((x \Rightarrow (x \vee (x \wedge x))) & = & S_3 \\ ((x \Rightarrow (x \vee x)) & = & S_4 \\ (x \Rightarrow x) & = & S_5 \\ x & = & S_6 = S_7 \end{array}$$

So S is a wff.

We will now prove the correctness of this algorithm in two parts: first, that it recognizes all wffs, and second, that it recognizes only wffs.

Part I. If $S = A$ is a wff, then this process stops at x.

Proof of part I. First notice that S is a wff iff $S(x)$ is a wff. (This can be easily proved by induction on the construction of formulas.)

So if A is a wff, so is $B = A(x)$. Thus it is enough to show that if we start with any wff B containing only the variable x and we perform this process, we will end up with x. For this it is enough to show that for any such formula B, B' is also a formula. Because then when the process $B_0 = B, B_1, \ldots, B_{n_0}$ stops, we have that B_{n_0} is a formula containing only the variable x and $B'_{n_0} = B_{n_0}$, i.e., B_{n_0} contains no substrings of the form $\neg x, (x \wedge x), (x \vee x), (x \Rightarrow x), (x \Leftrightarrow x)$, therefore it must be equal to x.

We can prove that for any formula B, B' is a formula, by induction on the construction of B. If $B = x$, then $B' = B = x$. If $B = \neg C$, then either $C = x$ and $B' = x$, or else $B' = \neg C'$, since $C \neq x$ implies that C cannot start with x. If $B = (C * D)$, then, unless $C = D = x$, in which case $B' = x$, either $B' = (C' * D)$ or $B' = (C * D')$. $\dashv$

Part II. If, starting from S, this process terminates at x, then S is a formula.

Proof of part II. Again we can assume that we start with a string S_0 containing only the variable x and show that if $S_0, S_1, \ldots, S_{n_0} = S'_{n_0} = x$, for some n_0, where $S_{n+1} = S'_n$, then S_0 is a formula. To see this notice that S_{n_0} is a formula and S_{n-1} is obtained from S_n by substituting some occurrence x by

a formula. So working backwards, and using the fact that if in a wff we substitute an occurrence of a variable by a formula we still get a formula (a fact which is easy to prove by induction), we see that $S_{n_0}, S_{n_0-1}, S_{n_0-2}, \ldots, S_1, S_0$ are all formulas. $\dashv$

1.3 Polish and Reverse Polish notation

We will now describe two different ways of writing formulas in which parentheses can be avoided. These methods are often better suited to computer processing, and many computer programs store formulas internally in one of these notations, even if they use standard notation for input and output.

1.3.A Polish Notation

We define a new formal language as follows:

(i) The symbols are the same as before, *except* that there are no parentheses, i.e.:

$$p_1, p_2, \ldots, \neg, \wedge, \vee, \Rightarrow, \Leftrightarrow$$

(ii) We have the following rules for forming grammatically correct formulas, which we now call *Polish wff* or just *P-wff* or *P-formulas*:

(a) Every propositional variable p is a P-wff;

(b) If A is a P-wff, so is $\neg A$.

(c) If A, B are P-wff, so is $*AB$, where $* \in \{\wedge, \vee, \Rightarrow, \Leftrightarrow\}$

Remark. Polish notation is named after the Polish logician Łukasiewicz who first introduced it, although he used N, K, A, C, and E in place of $\neg$, $\wedge$, $\vee$, $\Rightarrow$, and $\Leftrightarrow$ respectively.

Example 1.3.1

$$\Rightarrow\Rightarrow s\neg t\neg \vee p \wedge \neg qs$$

is a P-wff as the following parsing sequence demonstrates

q, $\neg q$, s, $\wedge\neg qs$, p, $\vee p \wedge \neg qs$, $\neg \vee p \wedge \neg qs$, t, $\neg t$, $\Rightarrow s\neg t$, $\Rightarrow\Rightarrow s\neg t\neg \vee p \wedge \neg qs$.

In ordinary notation this P-wff corresponds to the wff:

$$((s \Rightarrow \neg t) \Rightarrow \neg(p \vee (\neg q \wedge s))).$$

As with regular formulas, we have unique readability.

Theorem 1.3.2 (Unique Readability of P-wff) *Every P-wff A is in exactly one of the forms,*

(i) p

(ii) $\neg B$

(iii) $*BC$, *where* $* \in \{\wedge, \vee, \Rightarrow, \Leftrightarrow\}$, *for uniquely determined P-wff* B, C.

Proof. Clearly the only thing we have to prove is that if B, C, B', C' are P-wff and $BC = B'C'$, then $B = B', C = C'$. If $BC = B'C'$ then one of B, B' is an initial segment of the other, so it is enough to prove the following lemma:

Lemma 1.3.3 *No nonempty proper initial segment of a P-wff is a P-wff.*

Proof of Lemma 1.3.3. We define a *weight* for each symbol as follows:

$$\begin{aligned} w(p) &= 1 \\ w(\neg) &= 0 \\ w(*) &= -1 \quad \text{if } * \in \{\wedge, \vee, \Rightarrow, \Leftrightarrow\}. \end{aligned}$$

Then we define the weight of any string $S = s_1 s_2 \ldots s_n$, by

$$w(S) = \sum_{i=1}^{n} w(s_i).$$

For example, $w(\wedge \Rightarrow s\neg pq) = 1$. It is easy to show by induction on the construction of a P-wff that if A is a P-wff, then $w(A) = 1$.

We now define a *terminal segment* of a string $S = s_1 \ldots s_n$ to be any string of the form

$$s_m s_{m+1} \ldots s_n \quad (1 \leq m \leq n)$$

or the empty string. The *concatenation* of strings $S_1, \ldots, S_k$ is the string $S_1 S_2 \ldots S_k$, which consists of the symbols of S_1 followed by those of S_2, and so on.

Claim. Any nonempty terminal segment of a P-wff is a concatenation of P-wff.

Proof of Claim. Let A be a P-wff. We prove that any nonempty terminal segment of A is a concatenation of P-wff, by induction on the construction of A. If $A = p$ this is obvious. If $A = \neg B$, then any nonempty terminal segment of A is either A itself or a nonempty terminal segment of B, so it is a concatenation of P-wff, by induction hypothesis. Finally, if $A = *BC$, then any terminal segment of A is either A itself or the concatenation of a terminal segment of B with C or else a terminal segment of C, so again, by induction hypothesis, we are done.

Example. Consider the indicated terminal segment of the given P-wff:

$$\Rightarrow\Rightarrow \underbrace{s\neg t\neg \vee p \wedge \neg qs}$$

It is the concatenation of the following P-wff:

$$\underbrace{s}\ \underbrace{\neg t}\ \underbrace{\neg \vee p \wedge \neg qs}$$

We can now complete the proof of the lemma (and hence the theorem):

Suppose A is a P-wff and B is a P-wff which is a proper initial segment of A, so that $A = BC$, with C a nonempty terminal segment of A. Then we have

$$w(A) = w(B) + w(C).$$

But $w(A) = w(B) = 1$ and, as $C = C_1 \ldots C_n$ for some P-wwf $C_1, \ldots, C_n$,

$$w(C) = \sum_{i=1}^{n} w(C_i) \geq 1,$$

which is a contradiction. $\dashv$

Remark. From the preceding, it follows that any nonempty terminal segment of a P-wff is the concatenation of a *unique* sequence of P-wff.

Algorithm 1.3.4 There is a simple algorithm for verifying that a given string is a P-wff. Given a string $S = s_1 s_2 \dots s_n$ compute

$$w(s_n),\ w(s_n) + w(s_{n-1}),\ w(s_n) + w(s_{n-1}) + w(s_{n-2}), \dots,$$
$$w(s_n) + w(s_{n-1}) + \dots + w(s_1) = w(S).$$

Then S is a P-wff exactly when all these sums are ≥ 1 and $w(S) = 1$.

The correctness of this algorithm is a homework problem (Assignment #1).

1.3.B Reverse Polish notation

We can also define, in an obvious way, the *Reverse Polish notation* by defining a new formal language in which rules (b), (c) in the definition of P-wff are changed to: $A \to A\neg$ and $A, B \to AB*$. We refer to grammatically correct formulas in this formal language as *Reverse Polish wff*, *RP-wff*, or *RP-formulas*.

Example 1.3.5 $st\neg \Rightarrow pq\neg s \wedge \vee\neg \Rightarrow$ is a RP-wff with parsing sequence

s, t, $t\neg$, $st\neg \Rightarrow$, p, q, $q\neg$, $q\neg s\wedge$, $pq\neg s \wedge \vee$, $pq\neg s \wedge \vee\neg$, $st\neg \Rightarrow pq\neg s \wedge \vee\neg \Rightarrow$.

(This corresponds to example 1.3.1.)

Exactly as before, we can prove a unique readability theorem and devise a recognition algorithm, simply by reversing everything. We will not do this explicitly. Instead, we will discuss algorithms for translating formulas from Reverse Polish notation to ordinary notation and vice versa.

Algorithm 1.3.6 (Translating a RP-wff to an ordinary formula) We will describe it by an example, from which it is not hard to formulate the general algorithm:

Consider

$$st\neg \Rightarrow pq\neg s \wedge \vee\neg \Rightarrow$$

Scan from left to right until the first connective is met. Here it is $\neg$. Replace $t\neg$ by $\neg t$:

$$s\neg t \Rightarrow pq\neg s \wedge \vee\neg \Rightarrow$$

Again scan from left to right until the first unused connective is met. Here it is $\Rightarrow$. Replace $s\neg t \Rightarrow$ by $(s \Rightarrow \neg t)$.

$$(s \Rightarrow \neg t)pq\neg s \wedge \vee\neg \Rightarrow$$

Scan from left to right until the first unused connective is met. Here it is $\neg$. Replace $q\neg$ by $\neg q$:

$$(s \Rightarrow \neg t)p\neg qs \wedge \vee\neg \Rightarrow$$

Scan from left to right until the first unused connective is met. Here it is $\wedge$. Replace $\neg qs\wedge$ by $(\neg q \wedge s)$:

$$(s \Rightarrow \neg t)p(\neg q \wedge s) \vee \neg \Rightarrow$$

Scan from left to right until the first unused connective is met. Here it is $\vee$. Replace $p(\neg q \wedge s)\vee$ by $(p \vee (\neg q \wedge s))$:

$$(s \Rightarrow \neg t)(p \vee (\neg q \wedge s))\neg \Rightarrow$$

Scan from left to right until the first unused connective is met. Here it is $\neg$. Replace $(p \vee (\neg q \wedge s))\neg$ by $\neg(p \vee (\neg q \wedge s))$:

$$(s \Rightarrow \neg t)\neg(p \vee (\neg q \wedge s)) \Rightarrow$$

Scan from left to right until the first unused connective is met. Here it is $\Rightarrow$. Replace $(s \Rightarrow \neg t)\neg(p \vee (\neg q \wedge s)) \Rightarrow$ by

$$((s \Rightarrow \neg t) \Rightarrow \neg(p \vee (\neg q \wedge s))).$$

This is the final result.

If A is a RP-wff, denote by $O(A)$ the wff obtained in this fashion. Then it is not hard to see that $O(A)$ satisfies the following recursive definition:

(i) $O(p) = p$

(ii) $O(A\neg) = \neg O(A)$

(iii) $O(AB*) = (O(A) * O(B))$.

Algorithm 1.3.7 (Translating a wff to a RP-wff) The following analogous recursive definition will translate a wff A to a RP-wff $RP(A)$:

(i) $RP(p) = p$

(ii) $RP(\neg A) = RP(A)\neg$

(iii) $RP((A * B)) = RP(A)RP(B)*$.

It is not hard to see by induction that these processes are inverses of each other: i.e., for each RP-wff A, $RP(O(A)) = A$ and for each wff B $O(RP(B)) = B$.

We will now describe a way of implementing the recursive definition $A \mapsto RP(A)$ by a "stack algorithm". Visualize the given wff A as being the *input string*:

s_1	s_2	$\cdots$	s_n

We also have a *stack string*:

$\vdots$

This is initially empty and at various stages of the algorithm we will either put something on the top of the string or else remove something from the top of the string.

Finally we will have an *output string* which will eventually be $RP(A)$. In the beginning it is also empty:

		$\cdots$	

Before we describe the algorithm, let us notice that if A is a wff, say $A = s_1 s_2 \ldots s_n$, and for some $1 \leq i < n$, $s_i = \neg$ but $s_{i-1} \neq \neg$, then there is a unique wff $E = s_i \ldots s_m$, $m \leq n$, of the form $E = \neg \ldots \neg p$ or $E = \neg\neg \ldots \neg(C * D)$. This is because any wff is of the form $p, \neg \ldots \neg p$, $\neg\neg \ldots \neg(C * D)$ or $(C * D)$. Notice that in case $E = \neg\neg \ldots \neg(C * D)$, we can find the end of E by starting from the beginning $\neg$ until left and right parentheses balance.

We now describe the algorithm:

First scan A from left to right and whenever a $\neg$ is met which is not preceded by $\neg$ and it starts a wff of the form $E = \underbrace{\neg \ldots \neg}_{n}(C * D)$ or $E = \underbrace{\neg\neg \ldots \neg}_{n} p$ as above, add to the right of it n copies of a new symbol, say $\sqsupset$,

to obtain $E\underbrace{\sqsupset\sqsupset\cdots\sqsupset}_{n}$. After this was done for all such occurrences of $\neg$ in A we obtain a new string, say A' (which might contain many occurrences of this new symbol $\sqsupset$).

We now start scanning A' from left to right. If we see (do nothing. If we see p, we add p to the right of the output string. If we see $\neg$ or $* \in \{\wedge, \vee, \Rightarrow, \Leftrightarrow\}$, we add this symbol to the top of the stack. If we see) or $\sqsupset$ we remove the top symbol in the stack and add it to the right of the output string. When we have scanned all of A' this process stops and the output string is $RP(A)$.

Example 1.3.8 Consider the wff

$$((s \Rightarrow \neg t) \Rightarrow \neg(p \vee \neg(\neg q \wedge s))) = A$$

for which the RP-wff is

$$st\neg \Rightarrow pq\neg s \wedge \neg \vee \neg \Rightarrow = \ RP(A)$$

.

First we form A':

$$((s \Rightarrow \neg t \sqsupset) \Rightarrow \neg(p \vee \neg(\neg q \sqsupset \wedge s) \sqsupset) \sqsupset)$$

In the stack below, we represent top to bottom as right to left. Therefore the top of the stack, where we push and pop symbols, is on the right.

input	stack	output
(	$-$	$-$
(	$-$	$-$
s	$-$	s
$\Rightarrow$	$\Rightarrow$	s
$\neg$	$\Rightarrow \neg$	s
t	$\Rightarrow \neg$	st
$\sqsupset$	$\Rightarrow$	$st\neg$
)	$-$	$st\neg \Rightarrow$
$\Rightarrow$	$\Rightarrow$	$st\neg \Rightarrow$
$\neg$	$\Rightarrow \neg$	$st\neg \Rightarrow$

input	stack	output
$($	$\Rightarrow \neg$	$st\neg \Rightarrow$
p	$\Rightarrow \neg$	$st\neg \Rightarrow p$
$\vee$	$\Rightarrow \neg\vee$	$st\neg \Rightarrow p$
$\neg$	$\Rightarrow \neg \vee \neg$	$st\neg \Rightarrow p$
$($	$\Rightarrow \neg \vee \neg$	$st\neg \Rightarrow p$
$\neg$	$\Rightarrow \neg \vee \neg\neg$	$st\neg \Rightarrow p$
q	$\Rightarrow \neg \vee \neg\neg$	$st\neg \Rightarrow pq$
$\sqsupset$	$\Rightarrow \neg \vee \neg$	$st\neg \Rightarrow pq\neg$
$\wedge$	$\Rightarrow \neg \vee \neg\wedge$	$st\neg \Rightarrow pq\neg$
s	$\Rightarrow \neg \vee \neg\wedge$	$st\neg \Rightarrow pq\neg s$
$)$	$\Rightarrow \neg \vee \neg$	$st\neg \Rightarrow pq\neg s\wedge$
$\sqsupset$	$\Rightarrow \neg\vee$	$st\neg \Rightarrow pq\neg s \wedge \neg$
$)$	$\Rightarrow \neg$	$st\neg \Rightarrow pq\neg s \wedge \neg\vee$
$\sqsupset$	$\Rightarrow$	$st\neg \Rightarrow pq\neg s \wedge \neg \vee \neg$
$)$	$-$	$st\neg \Rightarrow pq\neg s \wedge \neg \vee \neg \Rightarrow$

We will now verify the correctness of this algorithm, by induction on the construction of formulas. We will show that if we start with A and then construct A' and apply this stack algorithm but with the stack containing some string S (not necessarily empty) and the output some string T (not necessarily empty), then we finish with the stack string S and output string $T\hat{}RP(A)$, where $\hat{}$ means concatenation.

This is obvious if $A = p$. Assume it holds for B and consider $A = \neg B$. Then notice that $A' = \neg B' \sqsupset$. Starting the algorithm from A' and stack string S and output string T, we first add $\neg$ to the top of S. Then we perform the algorithm on B' and stack string $S\neg$, output T. This finishes by producing stack string $S\neg$ and output $T\hat{}RP(B)$. Then since the last symbol of A' is $\sqsupset$, we get stack string S and output string $T\hat{}RP(B)\neg = T\hat{}RP(A)$.

Finally assume it holds for B, C and consider $A = (B * C)$. Then $A' = (B' * C')$. Starting the algorithm from A' and stack string S, output T, we first have $($, which does nothing. Then we have the algorithm on B' on stack string S, output T which produces $T\hat{}RP(B)$ and stack string S. Then $*$ is put on top of the stack and the algorithm is applied to C' with stack string $S*$ and output $T\hat{}RP(B)$ to produce stack string $S*$ and output

$T\hat{}RP(B)\hat{}RP(C)$. Finally we have) which produces stack string S and output $T\hat{}RP(B)\hat{}RP(C)* = T\hat{}RP(A)$. $\dashv$

1.4 Abbreviations

Sometimes in practice, in order to avoid complicated notation, we adopt various abbreviations in writing wff, if they don't cause any confusion.

For example, we usually omit outside parentheses: We often write $A \wedge B$ instead of $(A \wedge B)$ or $(A \wedge B) \vee C$ instead of $((A \wedge B) \vee C)$, etc.

Also we adopt an order of preference between the binary connectives, namely

$$\wedge, \vee, \Rightarrow, \Leftrightarrow$$

where connectives from left to right bind closer, i.e., $\wedge$ binds closer than $\vee, \Rightarrow, \Leftrightarrow$; $\vee$ binds closer than $\Rightarrow, \Leftrightarrow$; and $\Rightarrow$ binds closer than $\Leftrightarrow$. So if we write $p \wedge q \vee r$ we really mean $((p \wedge q) \vee r)$ and if we write $p \Rightarrow q \vee r$ we really mean $(p \Rightarrow (q \vee r))$, etc.

Finally, when we have repeated connectives, we always associate parentheses to the left, so that if we write $A \wedge B \wedge C$ we really mean $((A \wedge B) \wedge C)$, and if we write $A \Rightarrow B \Rightarrow C \Rightarrow D$ we really mean $(((A \Rightarrow B) \Rightarrow C) \Rightarrow D)$.

1.5 Semantics of Propositional Logic

We now want to ask the question, "When is a proposition true or false?" Specifically, if we know whether each atomic proposition is true or false, what is the truth value of a compound proposition? We will consider two truth values: T or 1 (true), and F or 0 (false).

1.5.A Valuations

Definition 1.5.1 A *truth assignment* or *valuation* consists of a map

$$\nu : \{p_1, p_2, \dots\} \to \{0, 1\}$$

assigning to each propositional variable a truth value.

Given such a ν we can extend it in a unique way to assign an associated truth value $\nu^*(A)$ to every wff A, by the following recursive definition:

(i) $\nu^*(p) = \nu(p)$.

(ii) $\nu^*(\neg A) = 1 - \nu^*(A)$.

(iii) For the binary connectives, the rules are:

$$\begin{aligned}
\nu^*((A \wedge B)) &= \begin{cases} 1 & \text{if } \nu^*(A) = \nu^*(B) = 1 \\ 0 & \text{otherwise} \end{cases} \\
&= \nu^*(A) \cdot \nu^*(B) \\
\nu^*((A \vee B)) &= \begin{cases} 1 & \text{if } \nu^*(A) = 1 \text{ or } \nu^*(B) = 1 \\ 0 & \text{if } \nu^*(A) = \nu^*(B) = 0 \end{cases} \\
&= 1 - (1 - \nu^*(A)) \cdot (1 - \nu^*(B)) \\
\nu^*((A \Rightarrow B)) &= \nu^*((\neg A \vee B)) \\
&= \begin{cases} 0 & \text{if } \nu^*(A) = 1,\ \nu^*(B) = 0, \\ 1 & \text{otherwise} \end{cases} \\
&= 1 - \nu^*(A) \cdot (1 - \nu^*(B)) \\
\nu^*((A \Leftrightarrow B)) &= \begin{cases} 1 & \text{if } \nu^*(A) = \nu^*(B) \\ 0 & \text{otherwise} \end{cases} \\
&= \nu^*(A) \cdot \nu^*(B) + (1 - \nu^*(A)) \cdot (1 - \nu^*(B))
\end{aligned}$$

For simplicity, we will write $\nu(A)$ instead of $\nu^*(A)$, if there is no danger of confusion.

These rules can be also represented by the following *truth tables*:

A	B	$\neg A$	$A \wedge B$	$A \vee B$	$A \Rightarrow B$	$A \Leftrightarrow B$
0	0	1	0	0	1	1
0	1	1	0	1	1	0
1	0	0	0	1	0	0
1	1	0	1	1	1	1

Note the similarity with the truth tables presented in section 1.1. We are now formalizing our intuitive sense of what the propositional connectives should "mean."

Definition 1.5.2 For each wff A, its *support*, $\mathrm{supp}(A)$, is the set of propositional variables occurring in it.

Example 1.5.3

$$\mathrm{supp}(\underbrace{((p \wedge q) \Rightarrow (\neg r \vee p))}_{A}) = \{p, q, r\}. \qquad (*)$$

Thus we have, by way of recursive definition,

$$\begin{aligned} \mathrm{supp}(p) &= \{p\} \\ \mathrm{supp}(\neg A) &= \mathrm{supp}(A) \\ \mathrm{supp}((A * B)) &= \mathrm{supp}(A) \cup \mathrm{supp}(B). \end{aligned}$$

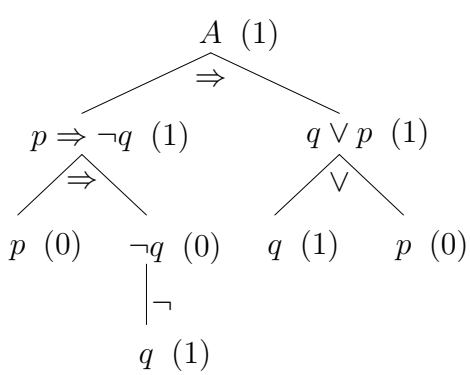
The following is easy to check:

Proposition 1.5.4 *If* $\mathrm{supp}(A) = \{q_1, \ldots, q_n\}$ *and* ν *is a valuation, then* $\nu(A)$ *depends only on* $\nu(q_1), \ldots, \nu(q_n)$*; i.e., if* μ *is a valuation, and* $\mu(q_i) = \nu(q_i)$ *for* $i = 1, \ldots, n$*, then* $\mu(A) = \nu(A)$*.*

For instance, in example 1.5.3 above, $\nu(A)$ depends only on $\nu(p)$, $\nu(q)$, and $\nu(r)$.

We can visualize the calculation of $\nu(A)$ in terms of the parse tree T_A as follows, where we illustrate the general idea by an example.

Example 1.5.5 Consider $A = ((p \Rightarrow \neg q) \Rightarrow (q \vee p))$ and its parse tree. Let $\nu(p) = 0,\ \nu(q) = 1$. Then we can find the truth value of each node of the tree working upwards. So $\nu(A) = 1$.



We also have the following algorithm for evaluating $\nu(A)$, given the valuation ν to all the propositional variables in the support of A.

Algorithm 1.5.6 First replace each propositional variable p in A by its truth value (according to ν), $\nu(p)$. Then scanning from left to right find the first occurence of a string of the form $\neg i,\ (i \wedge j),\ (i \vee j),\ (i \Rightarrow j),\ (i \Leftrightarrow j)$, where $i, j \in \{0, 1\}$, and replace it by its value according to the truth table (e.g., replace $\neg 0$ by 1, $(0 \wedge 1)$ by 0, etc.). Repeat the process until the value $\nu(A)$ is obtained.

Example 1.5.7 A, ν as in example 1.5.5. Then we first obtain $((0 \Rightarrow \neg 1) \Rightarrow (1 \vee 0))$. Next we have successively $((0 \Rightarrow 0) \Rightarrow (1 \vee 0)),\ (1 \Rightarrow (1 \vee 0)),\ (1 \Rightarrow 1),\ 1 = \nu(A)$.

It is not hard to prove by induction on A that this algorithm is correct, i.e., produces always $\nu(A)$.

Given a valuation ν we can of course define in a similar way the truth value $\nu(A)$ for any P-wff or RP-wff A. We will now describe a stack algorithm for calculating the truth value for RP-wff.

Algorithm 1.5.8 We start with a RP-wff A and a valuation ν to all the propositional variables in the support of A. The stack is empty in the beginning.

We scan A from left to right. If we see a propositional variable p, we add to the top of the stack $\nu(p)$. If we see $\neg$, we replace the top of the stack a (which is either 0 or 1) by $1 - a$. If we see $* \in \{\wedge, \vee, \Rightarrow, \Leftrightarrow\}$, we replace the top two elements of the stack b (= top), a (= next to top) by the value given by the truth table for $*$ applied to the truth values a, b (in that order). The truth value in the stack, when we finish scanning A, is $\nu(A)$.

Example 1.5.9 Let A be the RP-wff

$$pqp\neg\vee \Rightarrow rs\neg u \Rightarrow \vee \Rightarrow .$$

(in ordinary notation this is $(p \Rightarrow (q \vee \neg p)) \Rightarrow (r \vee (\neg s \Rightarrow u))$ and $\nu(p) = 1,\ \nu(q) = 1,\ \nu(r) = 0,\ \nu(s) = 1,\ \nu(u) = 1$. Then by applying the algorithm we have (once again, top to bottom on the stack is represented as right to left):

input	stack
p	1
q	11
p	111
$\neg$	110
$\vee$	11
$\Rightarrow$	1
r	10
s	101
$\neg$	100
u	1001
$\Rightarrow$	101
$\vee$	11
$\Rightarrow$	1

Again it is easy to show by induction that this algorithm is correct, i.e., it always produces $\nu(A)$. The appropriate statement we must prove, by induction on the construction of A, is that if we start this algorithm with input a RP-wff A and stack S, we end up by having stack $S\nu(A)$. (Here S is a string of 0's and 1's.)

1.5.B Models and Tautologies

Definition 1.5.10 If ν is a valuation and A a wff, we say that ν *satisfies* or *models* A if $\nu(A) = 1$.

That is, ν models A if A is true under the truth values that ν assigns to the propositional variables. We use the notation

$$\nu \models A$$

to denote that ν models A. We also write

$$\nu \not\models A$$

if ν does not satisfy or model A, i.e., if $\nu(A) = 0$. Notice that

$$\begin{aligned}
\nu \models \neg A &\text{ iff } \nu \not\models A \\
\nu \models (A \wedge B) &\text{ iff } \nu \models A \text{ and } \nu \models B \\
\nu \models (A \vee B) &\text{ iff } \nu \models A \text{ or } \nu \models B \\
\nu \models (A \Rightarrow B) &\text{ iff either } \nu \not\models A \text{ or else } \nu \models B \\
\nu \models (A \Leftrightarrow B) &\text{ iff } (\nu \models A \text{ and } \nu \models B) \text{ or } (\nu \not\models A \text{ and } \nu \not\models B).
\end{aligned}$$

Definition 1.5.11 A wff A is a *tautology* (or *valid*) iff for *every* valuation ν we have $\nu \models A$. So A is a tautology if it is true independently of the truth assignments to its propositional variables.

Examples 1.5.12

(i) $\neg(A \wedge B) \Leftrightarrow (\neg A \vee \neg B)$ is a tautology. This is an expression of one of De Morgan's laws, which we first saw in section 1.1. The other "equivalences" we discussed there can also be expressed as tautological formulas: $A \Leftrightarrow \neg\neg A$ for the law of the double negative, and so on.

(ii) $A \vee (B \wedge C) \Leftrightarrow (A \vee B) \wedge (A \vee C)$ is a tautology. This is an equivalence we did not discuss: it says that $\vee$ *distributes over* $\wedge$. It is also true that $\wedge$ distributes over $\vee$, i.e. $A \wedge (B \vee C) \Leftrightarrow (A \wedge B) \vee (A \wedge C)$ is also a tautology.

(iii) $(p \Rightarrow q) \Rightarrow (q \Rightarrow p)$ is *not* a tautology, since it is falsified by the valuation $\nu(p) = 0$, $\nu(q) = 1$. We saw in section 1.1 that the truth of an implication is generally unrelated to the truth of its converse, but this formula claims that the converse is true whenever the original formula is true.

Definition 1.5.13 A wff A is *satisfiable* iff there is *some* valuation ν such that $\nu \models A$, i.e. A has at least one model. Otherwise, we say that A is *unsatisfiable* or *contradictory*.

Clearly A is contradictory iff $\neg A$ is a tautology, and vice versa.

Example 1.5.14 $(p \Rightarrow q)$ is satisfiable (e.g. by the valuation $\nu_1(p) = \nu_1(q) = 1$) and its negation is also satisfiable (by the valuation $\nu_2(p) = 1$, $\nu_2(q) = 0$).

To check whether a given formula is a tautology or not, we can write down its truth table and check that it gives always 1's, for any value of the propositional variables.

Example 1.5.15 Consider $(p \Rightarrow q) \Leftrightarrow (\neg q \Rightarrow \neg p)$.

p	q	$\neg p$	$\neg q$	$p \Rightarrow q$	$\neg q \Rightarrow \neg p$	$(p \Rightarrow q) \Leftrightarrow (\neg q \Rightarrow \neg p)$
0	1	1	0	1	1	1
0	0	1	1	1	1	1
1	1	0	0	1	1	1
1	0	0	1	0	0	1

This procedure requires 2^n many entries if the wff A contains n propositional variables, so it has "exponential complexity." No substantially more efficient algorithm is known. The possibility of improving the "exponential" to "polynomial" complexity is a famous open problem in theoretical computer science known as the "$\mathcal{P} = \mathcal{NP}$ problem", that we will discuss in Chapter 3.

We can extend the definition of satisfiability to sets containing more than one formula. Clearly the only interesting notion is that of *simultaneous* satisfiability; i.e. whether all formulas are satisfied by the same valuation.

Definition 1.5.16 If S is any set of wff (finite or infinite) and ν is a valuation, we say that ν *satisfies* or *models* S, in symbols

$$\nu \models S$$

if $\nu \models A$ for *every* $A \in S$, i.e., ν models *all* wff in S. If there is some valuation satisfying S we say that S is *satisfiable* (or has a *model*).

Examples 1.5.17

(i) $S = \{(p_1 \vee p_2),\ (\neg p_2 \vee \neg p_3),\ (p_3 \vee p_4),\ (\neg p_4 \vee \neg p_5), \dots\}$ is satisfiable, with model the valuation:

$$\nu(p_i) = \begin{cases} 1 & \text{if } i \text{ is odd;} \\ 0 & \text{if } i \text{ is even.} \end{cases}$$

(ii) $S = \{q \Rightarrow r,\ r \Rightarrow p,\ \neg p, q\}$ is *not* satisfiable.

(iii) If $S = \{A_1, \dots, A_n\}$ is finite, then S is satisfiable iff $A_1 \wedge \cdots \wedge A_n$ is satisfiable.

1.5.C Logical Implication and Equivalence

Definition 1.5.18 Let now S be any set of wff and A a wff. (View S as a set of hypotheses and A as a conclusion.) We say that S *(tauto)logically implies* A if every valuation that satisfies S satisfies also A (i.e., any model of S is also a model of A).

If S logically implies A, we write

$$S \models A$$

(and if it does not, $S \not\models A$). We use the same symbol as for models of a formula, because the notions are compatible. If $S = \emptyset$, we just write $\models A$. This simply means that A is a tautology (it is true in all valuations).

Examples 1.5.19

(i) (Modus Ponens) $\{A, A \Rightarrow B\} \models B$.

(ii) $\{A, A \Rightarrow (B \vee C),\ B \Rightarrow D,\ C \Rightarrow D\} \models D$.

(iii) $p \Rightarrow q \not\models q \Rightarrow p$

(iv) $\{p_1 \Rightarrow p_2,\ p_2 \Rightarrow p_3,\ p_3 \Rightarrow p_4, \dots\} \models p_1 \Rightarrow p_n$ (for any n)

(v) If $S = \{A_1, \dots, A_n\}$ is finite, then $S = \{A_1, \dots, A_n\} \models A$ iff $A_1 \wedge \cdots \wedge A_n \models A$ iff $\models (A_1 \wedge \cdots \wedge A_n) \Rightarrow A$.

(vi) $S \models (A \Rightarrow B)$ iff $S \cup \{A\} \models B$.

(vii) $S \models A$ iff $S \cup \{\neg A\}$ is *not* satisfiable.

Example 1.5.20 Consider the following argument (from Mendelson's book "Introduction to Mathematical Logic")

> If capital investment remains constant, then government spending will increase or unemployment will result. If government spending will not increase, taxes can be reduced. If taxes can be reduced and capital investment remains constant, then unemployment will not result. Hence, government spending will increase.

To see whether this conclusion logically follows from the premises, represent:

p : "capital investment remains constant"
q : "government spending will increase"
r : "unemployment will result"
s : "taxes can be reduced"

Then our premises are

$$S = \{p \Rightarrow (q \vee r),\ \neg q \Rightarrow s,\ s \wedge p \Rightarrow \neg r\}$$

Our conclusion is

$$A = q.$$

So we are asking if

$$S \models q.$$

This is false since the truth assignment $\nu(p) = 0, \nu(q) = 0, \nu(r) = 1, \nu(s) = 1$ makes S true but q false. So the argument above is not valid.

Remark. Note that if $S \subseteq \tilde{S}$ and $S \models A$, then $\tilde{S} \models A$. Also, if $S \models A$ for all $A \in S'$, and $S' \models B$, then $S \models B$.

Definition 1.5.21 Two wff A, B are called (*logically) equivalent*, in symbols

$$A \equiv B,$$

if $A \models B$ and $B \models A$, i.e. $A \Leftrightarrow B$ is a tautology.

Notice that logical equivalence is an equivalence relation, i.e.,

$$A \equiv A$$
$$A \equiv B \text{ implies } B \equiv A$$
$$A \equiv B,\ B \equiv C \text{ imply } A \equiv C.$$

We consider equivalent wff as semantically indistinguishable. However, in general equivalent formulas are different syntactically. By that we mean that they are distinct strings of symbols, e.g. $(p \Rightarrow q)$ and $(\neg q \Rightarrow \neg p)$.

Once again, we see the same equivalences that we mentioned in section 1.1, such as $A \equiv \neg\neg A$ and $\neg(A \wedge B) \equiv (\neg A \vee \neg B)$. Here are a couple more:

Examples 1.5.22

(i) $(A \Leftrightarrow B) \equiv ((A \wedge B) \vee (\neg A \wedge \neg B))$

(ii) $((A \wedge B) \Rightarrow C) \equiv ((A \Rightarrow C) \vee (B \Rightarrow C))$

Recommendation. Read Appendix A, which lists various useful tautologies and equivalences. Verify that they are indeed tautologies and equivalences. You will often need to use some of them in homework assignments.

Here are some useful facts about logical equivalence.

(i) Let A be a wff containing the propositional variables $q_1, \ldots, q_n$, and let $A_1, \ldots, A_n$ be arbitrary wff. If A is a tautology, so is

$$A[q_1/A_1, q_2/A_2, \ldots, q_n/A_n],$$

the wff obtained by substituting q_i by A_i $(1 \leq i \leq n)$ in A.

(ii) If A is a wff, B a subformula of A and B' a wff such that $B \equiv B'$, and we obtain A' from A by substituting B by B', then $A \equiv A'$. This can be easily proved by induction on the construction of A. Thus we can freely substitute equivalent formulas (as parts of bigger formulas) without affecting the truth value.

Example. $A = (\neg(p \wedge q) \Rightarrow (q \vee r)), B = \neg(p \wedge q), B' = (\neg p \vee \neg q), A' = ((\neg p \vee \neg q) \Rightarrow (q \vee r))$.

The *De Morgan Laws*, which we have seen before, are the equivalences

$$\neg(p \wedge q) \equiv (\neg p \vee \neg q)$$
$$\neg(p \vee q) \equiv (\neg p \wedge \neg q)$$

By the first fact above, we can conclude that $\neg(A \wedge B) \equiv (\neg A \vee \neg B)$, $\neg(A \vee B) \equiv (\neg A \wedge \neg B)$ for any wff A, B. The De Morgan laws also have the following generalization:

Proposition 1.5.23 (General De Morgan Laws) *Let A be a wff containing only the connectives $\neg, \wedge, \vee$. Let A^* be obtained from A by substituting each propositional variable p occurring in A by $\neg p$ and replacing $\wedge$ by $\vee$ and $\vee$ by $\wedge$. Then*

$$\neg A \equiv A^*.$$

Example 1.5.24

$$\begin{aligned}\neg((p \wedge q) \vee (\neg r \wedge p)) &\equiv ((\neg p \vee \neg q) \wedge (\neg\neg r \vee \neg p)) \\ &\equiv ((\neg p \vee \neg q) \wedge (r \vee \neg p)), \text{ since } \neg\neg r \equiv r.\end{aligned}$$

Proof. The proof of the General De Morgan's Laws will be left for Homework Assignment #2.

Augustus de Morgan was born in 1806 in Madura, India, and died in 1871 in London. De Morgan lost the sight on his right eye as a boy. His family moved to England when he was 10. At 1823, he refused to take a theological test, and as a result Trinity College (Cambridge) only gave him a BA instead of an MA. In 1828 he was appointed Professor at the new University College (London); he was the first mathematics professor of the college. De Morgan was first to formally define *mathematical induction* (1838). He was a friend of Charles Babbage, the creator of the first "difference engine" (1819–22), a predecessor of the modern computer. This led to his interest in logic as an *algebra* of identities and to his introduction of what are now known as De Morgan's laws.

"A dry dogmatic pedant I fear is Mr. De Morgan, notwithstanding his unquestioned ability." Thomas Hirst.

1.6 Truth Functions

We can consider a formula (or even an equivalence class of logically equivalent formulas) to define a function. Its inputs are the truth values of the propositional variables in its support (that is, a valuation, or at least the relevant part of a valuation), and its output is the truth value of the formula (that is, whether the valuation models the formula). We can also go the other way: starting from any such function, we will construct a propositional formula which describes it.

1.6.A Truth Functions

Definition 1.6.1 For $n \geq 1$, an *n-ary truth function* (also called a *Boolean function*) is any map

$$f : \{0,1\}^n \to \{0,1\}.$$

By convention we also have two "0-ary" truth functions, namely the constants 0 and 1.

An n-ary truth function can be represented by a *truth table* as follows:

x_1	x_2	$\cdots$	x_n	$f(x_1, \ldots, x_n)$
0	0	$\cdots$	0	$f(0, \ldots, 0)$
$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
i_1	i_2	$\cdots$	i_n	$f(i_1, \ldots, i_n)$
$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
1	1	$\cdots$	1	$f(1, \ldots, 1)$

This table has 2^n rows, allowing for all possibilities for the vector $(i_1, \ldots, i_n)$, where each i_k is 0 or 1. Since for each of these 2^n rows the values of f are arbitrary (can be either 0 or 1), altogether there are 2^{2^n} possible n-ary truth functions. We will look at some or all of these, for some small values of n.

$\mathbf{n = 0}$. As mentioned above, there are two "0-ary" truth functions: the binary constants 0 and 1.

$\mathbf{n = 1}$. There are $2^2 = 4$ *unary* truth functions:

x	$f_1(x)$	$f_2(x)$	$f_3(x)$	$f_4(x)$
0	0	0	1	1
1	0	1	0	1

So $f_1(x) = 0$ (constant), $f_4(x) = 1$ (constant), $f_2(x) = x$ (identity), $f_3(x) = 1 - x$ (corresponds to negation).

$\mathbf{n = 2}$. There are $2^{2^2} = 16$ *binary* truth functions. For some examples, notice that each binary connective (via its truth table) gives rise to a corresponding binary truth function. For example, $\wedge$ gives rise to $f(x, y) = x \wedge y = x \cdot y$ (i.e. multiplication modulo two). Conversely, we can view each binary truth function as defining a binary connective, so we have altogether 16 binary connectives. (In a similar way, we can also view n-ary truth functions as *n-ary connectives.*) We have seen the standard binary connectives: $\wedge, \vee, \Rightarrow, \Leftrightarrow$. Here are a few others that play an important role in various ways.

x	y	$x+y$
0	1	1
0	0	0
1	1	0
1	0	1

This corresponds to *binary addition*, i.e., addition in $\mathbb{Z}_2$ (the integers modulo 2). We see that $x+y = (x \wedge \neg y) \vee (y \wedge \neg x)$. This connective is called *exclusive or* and sometimes also *symmetric difference*.

x	y	$x \downarrow y$
0	0	1
0	1	0
1	0	0
1	1	0

Notice that $x \downarrow y = \neg(x \vee y)$. $\downarrow$ is called *nor*.

x	y	$x \mid y$
0	0	1
0	1	1
1	0	1
1	1	0

Notice that $x \mid y = \neg(x \wedge y)$. $\mid$ is called *nand* or *Sheffer stroke*.

Remark. Notice that some of the 16 binary connectives are degenerate, i.e., depend on only one (or none) of the arguments, e.g.,

x	y	$f(x,y) = \neg x$
0	1	1
0	0	1
1	1	0
1	0	0

In fact, every 0-ary connective (i.e. constant) gives rise to a (constant) binary connective, and every unary connective gives rise to two (degenerate) binary connectives ($f(x,y) = g(x)$ and $f(x,y) = g(y)$). (These are not all distinct, of course, since some unary connectives are themselves degenerate.) More generally, if $n > m$, then any m-ary connective gives rise to degenerate n-ary connectives.

n = 3. There are $2^{2^3} = 256$ ternary connectives. These are too many to list, but an interesting example is the "if x then y, else z" connective, whose value is y if $x = 1$ and z if $x = 0$. Another one is the majority connective:

$$\mathrm{maj}(x, y, z) = \begin{cases} 1 & \text{if the majority of } x, y, z \text{ is } 1 \\ 0 & \text{otherwise} \end{cases}$$

For each wff A we indicate the fact that A contains only propositional variables among $p_1, \ldots, p_n$ by writing $A(p_1, \ldots, p_n)$. This simply means that $\mathrm{supp}(A) \subseteq \{p_1, \ldots p_n\}$. It does not mean that A contains *all* of the propositional variables $p_1, \ldots, p_n$.

Definition 1.6.2 For each wff $A(p_1, \ldots, p_n)$, we define an n-ary truth function $f_A^n : \{0, 1\}^n \to \{0, 1\}$ by

$$f_A^n(x_1, \ldots, x_n) = \text{ (the truth value of } A \text{ given by the valuation } \nu(p_i) = x_i).$$

In other words, f_A is the truth function corresponding to the truth table of the wff A.

Examples 1.6.3

(i) For any binary connective $*$, if $A = (p_1 * p_2)$, then f_A is the truth function f_* corresponding to $*$. Similarly, if $A = \neg p_1$, then $f_A = f_\neg$, and so on.

(ii) If $A = ((p_1 \wedge p_2) \vee (\neg p_1 \wedge p_3))$, then f_A is the truth function of the "if... then... else..." connective.

Remark. Strictly speaking, each wff A gives rise to infinitely many truth functions f_A^n: one for each $n \geq n_A$, where n_A is the least number m for which $\mathrm{supp}(A) \subseteq \{p_1, \ldots, p_m\}$. This is the same situation that we face when we have a polynomial, like $x + y$, which we can view as defining a function of two variables x, y, but also as a function of three variables x, y, z, which only depends on x, y, etc. However when the n is understood or irrelevant we just write f_A.

Note that by definition

$$A \equiv B \text{ iff } f_A = f_B.$$

The main fact about truth functions is the following:

Theorem 1.6.4 *Every truth function is realized by a wff containing only* $\neg, \wedge, \vee$. *That is, if* $f : \{0,1\}^n \to \{0,1\}$ *with* $n \geq 1$, *there is a wff*

$$A(p_1, \ldots, p_n)$$

containing only $\neg, \wedge, \vee$ *such that*

$$f = f_A \ (= f_A^n).$$

Proof. If f is identically 0, take $A = (p_1 \wedge \neg p_1)$.

Otherwise, for each entry $s = (x_1, \ldots, x_n) \in \{0,1\}^n$ in the truth table of f, introduce the wff

$$A_s = \epsilon_1 p_1 \wedge \epsilon_2 p_2 \wedge \cdots \wedge \epsilon_n p_n$$

where

$$\epsilon_i = \begin{cases} \text{nothing} & \text{if } x_i = 1 \\ \neg & \text{if } x_i = 0 \end{cases}.$$

Example. $s = (1,0,1) \to A_s = p_1 \wedge \neg p_2 \wedge p_3$.

Notice that $f_{A_s}(x_1, \ldots, x_n) = 1$, but $f_{A_s}(x'_1, \ldots, x'_n) = 0$ if $(x'_1, \ldots, x'_n) \neq (x_1, \ldots, x_n)$. Enumerate in a sequence $s_1, \ldots, s_m$ $(m \leq 2^n)$ all $s \in \{0,1\}^n$ such that $f(s) = 1$ (and there is at least one such) and put

$$A = A_{s_1} \vee A_{s_2} \vee \cdots \vee A_{s_m}.$$

Then $f_A(s) = 1$ iff at least one of $f_{A_{s_i}}(s) = 1$ iff $s \in \{s_1, \ldots, s_m\}$ iff $f(s) = 1$, so $f_A = f$. $\dashv$

Example 1.6.5 Suppose that f is given by the following truth table.

x_1	x_2	x_3	$f(x_1, x_2, x_3)$
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Then f is realized by the wff

$$A = (\neg p_1 \wedge p_2 \wedge \neg p_3) \vee (p_1 \wedge \neg p_2 \wedge \neg p_3) \vee (p_1 \wedge p_2 \wedge p_3).$$

Corollary 1.6.6 *Any wff is equivalent to a wff containing only $\neg, \wedge$ and to a wff containing only $\neg, \vee$. So any truth function can be realized by a wff containing only $\neg, \wedge$ or only $\neg, \vee$.*

Proof. Given a wff $A(p_1, \ldots, p_n)$, consider the truth function $f_A : \{0,1\}^n \to \{0,1\}$ and let $A'(p_1, \ldots, p_n)$ be a wff containing only $\neg, \wedge, \vee$ such that

$$f_A = f_{A'},$$

i.e.

$$A \equiv A'.$$

We can now systematically eliminate $\vee$ by using the equivalence

$$C \vee D \equiv \neg(\neg C \wedge \neg D) \tag{1.1}$$

to obtain an equivalent formula $A'' \equiv A'$ containing only $\neg, \wedge$. Similarly using

$$C \wedge D \equiv \neg(\neg C \vee \neg D) \tag{1.2}$$

we can find an equivalent formula $A''' \equiv A'$ containing only $\neg, \vee$. $\dashv$

Remark. Another way to prove this corollary is by using the equivalences (1.1) and (1.2) above, as well as

$$(C \Rightarrow D) \equiv (\neg C \vee D)$$

$$(C \Leftrightarrow D) \equiv (\neg C \vee D) \wedge (\neg D \vee C)$$

to systematically eliminate all connectives except $\neg, \vee$ or $\neg, \wedge$.

Example 1.6.7 Suppose A is

$$((p \Rightarrow q) \wedge \neg(r \Rightarrow s)) \vee (s \vee p),$$

and we want to find an equivalent wff involving only $\neg, \wedge$. We have the following equivalent wffs, successively:

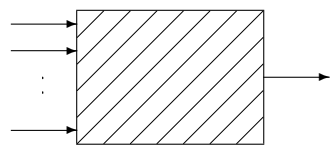
$$((p \Rightarrow q) \wedge \neg(r \Rightarrow s)) \vee (s \vee p) = A,$$

$$((\neg p \vee q) \wedge \neg(\neg r \vee s)) \vee (s \vee p),$$

$$(\neg(p \wedge \neg q) \wedge (r \wedge \neg s)) \vee \neg(\neg s \wedge \neg p),$$

$$\neg(\neg(\neg(p \wedge \neg q) \wedge (r \wedge \neg s))) \wedge (\neg s \wedge \neg p)).$$

Remark. If by a *(logic) circuit* we understand any device which accepts n binary inputs and produces a binary output, then the previous results show that any circuit can be built out of (*not*, *and*) or (*not*, *or*) gates only.



1.6.B Completeness of Binary Connectives

Definition 1.6.8 A set of connectives $C \subseteq \{\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow\}$ is called *complete* if any wff is equivalent to a wff whose only connectives are in C.

Examples 1.6.9

(i) $\{\neg, \wedge\}$, $\{\neg, \vee\}$ are complete sets, as seen in corollary 1.6.6.

(ii) $\{\neg, \Rightarrow\}$ is complete, because $(A \vee B) \equiv (\neg A \Rightarrow B)$.

(iii) $\{\neg, \Leftrightarrow\}$ is not complete: To see this notice that if $A(p_1, p_2)$ is any wff which only contains $\neg, \Leftrightarrow$ and a, b, c, d are the values of A under the four possible truth assignments to the variables p_1, p_2, then viewing a, b, c, d as members of $\mathbb{Z}_2$ we claim that

$$a + b + c + d = 0.$$

We can prove this by induction on the construction of A. If $A = p_1$ or $A = p_2$, then this is true since $a + b + c + d = 1 + 1 + 0 + 0 = 0$. If $A = \neg B$ and a, b, c, d are the values of B, then the values of A are $1 - a, 1 - b, 1 - c, 1 - d$ and their sum is

$$4 - (a + b + c + d) = 0 - 0 = 0.$$

Finally if $A = (B \Leftrightarrow C)$ and a_1, b_1, c_1, d_1 and a_2, b_2, c_2, d_2 are the corresponding values of B, C (i.e., corresponding to the same truth assignments to the variables p_1, p_2), then the truth values of A are $a_1 \Leftrightarrow a_2 = 1 - (a_1 + a_2),\ 1 - (b_1 + b_2),\ 1 - (c_1 + c_2), 1 - (d_1 + d_2)$, whose sum is

$$4 - (a_1 + b_1 + c_1 + d_1 + a_2 + b_2 + c_2 + d_2) = 0 - 0 = 0.$$

Since the values of $p_1 \wedge p_2$ are 1, 0, 0, 0 whose sum is 1, it follows that $p_1 \wedge p_2$ cannot be equivalent to any wff built using only $\neg, \Leftrightarrow$, so $\{\neg, \Leftrightarrow\}$ is not complete.

Recall that we actually have 16 binary connectives. We can introduce a symbol for each one of them (as we have already done for the most common ones) and build up formulas using them. So we can generalize the preceding definition to say that any set C of binary connectives is *complete* if any wff is equivalent to one in which the only connectives are contained in C. (In terms of truth functions this simply means that every truth function can be expressed as a composition of the truth functions contained in C. To see this notice that if $*$ is a binary connective, then $f_{A*B}(\bar{x}) = *(f_A(\bar{x}),\ f_B(\bar{x}))$.)

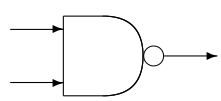
A single binary connective $*$ is complete if $C = \{*\}$ is complete, i.e., every wff is equivalent to one using only $*$. It turns out that the nand ($|$), and nor ($\downarrow$) connectives are complete. This is easily seen, because

$$\begin{aligned} \neg p &\equiv p|p \equiv p \downarrow p \\ (p \vee q) &\equiv (p|p)|(q|q) \\ (p \wedge q) &\equiv (p \downarrow p) \downarrow (q \downarrow q). \end{aligned}$$

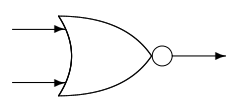
We will see in Assignment #3 that these are the *only* complete binary connectives.

Remark. In terms of circuits this implies that they can all be built using only *nor* or *nand* gates:

nand gate



nor gate



1.6.C Normal Forms

Definition 1.6.10 A wff A is in *disjunctive normal form* (*dnf*) if

$$A = A_1 \vee \cdots \vee A_n,$$

with

$$A_i = \ell_1^{(i)} \wedge \cdots \wedge \ell_{k_i}^{(i)},$$

and each $\ell_j^{(i)}$ a *literal*, i.e., p or $\neg p$ for some propositional variable p. We call A_i the *disjuncts* of A.

Examples 1.6.11

(i) $p, p \wedge q, (p \wedge q) \vee (\neg r \wedge s)$, $p \vee \neg q \vee (s \wedge \neg t \wedge u)$ are in dnf.

(ii) $p \Rightarrow q$ and $(p \vee (q \wedge r)) \wedge s$ are *not* in dnf.

Theorem 1.6.12 *For every wff A we can find a wff B in dnf such that $A \equiv B$.*

Proof. This follows from the proof of Theorem 1.6.4. If $A = A(p_1, \ldots, p_n)$ and A is contradictory, take $B = p_1 \wedge \neg p_1$. Otherwise, let

$$X = \{(\epsilon_1, \ldots, \epsilon_n) : \text{ the truth assigment } p_i \mapsto \epsilon_i \text{ satisfies } A\} \subseteq \{0,1\}^n,$$

and let

$$B = \bigvee_{(\epsilon_1,\ldots,\epsilon_n)\in X} \bigwedge_{i=1}^{n} \epsilon_i p_i,$$

where $\epsilon_i p_i = p_i$ if $\epsilon_1 = 1$ and $\neg p_i$ if $\epsilon_i = 0$. (The notation $\bigwedge_{i=1}^n$, like $\sum_{i=1}^n$, means to connect the specified values with $\wedge$, as i runs from 1 to n.) Then B is the desired formula. $\dashv$

Definition 1.6.13 A wff A is in *conjunctive normal form* (cnf) iff

$$A = A_1 \wedge \cdots \wedge A_n,$$

with

$$A_i = \ell_1^{(i)} \vee \cdots \vee \ell_{k_i}^{(i)}$$

and each $\ell_j^{(i)}$ a literal. We call A_i the *conjuncts* of A.

Example 1.6.14

$p, p \vee q, (p \vee q) \wedge (\neg r \vee s)$, $p \wedge \neg q \wedge (s \vee \neg t \vee u)$ are in cnf.

$p \Rightarrow q$, $(p \vee (q \wedge r)) \wedge s$, and $(p \wedge q) \vee r$ are *not* in cnf.

Corollary 1.6.15 *For every wff A we can find a wff B in cnf such that $A \equiv B$.*

Proof. Apply the preceding theorem to $\neg A$ to find a wff B' in dnf such that

$$\neg A \equiv B',$$

so

$$A \equiv \neg B'.$$

Now

$$B' = \bigvee_{i=1}^{n} \bigwedge_{j=1}^{k_i} \ell_j^{(i)},$$

so, by de Morgan,

$$\neg B' \equiv \bigwedge_{i=1}^{n} \bigvee_{j=1}^{k_i} \neg \ell_j^{(i)}.$$

If some $\ell_j^{(i)}$ is of the form $\neg p$ replace $\neg \ell_j^{(i)}$ by p to obtain $\tilde{\ell}_j^{(i)}$. Otherwise let $\tilde{\ell}_j^{(i)} = \ell_j^{(i)}$. Thus

$$\neg B' \equiv \bigwedge_{i=1}^{n} \bigvee_{j=1}^{k_i} \tilde{\ell}_j^{(i)} = B,$$

where each $\tilde{\ell}_j^{(i)}$ is a literal, so B is in cnf and $A \equiv B$. $\dashv$

Remark. Given a wff A in dnf it is easy to check if it is satisfiable or not: If

$$A = \bigvee_{i=1}^{n} \bigwedge_{j=1}^{k_i} \ell_j^{(i)},$$

then A is satisfiable iff one of the disjuncts is satisfiable, which is true iff there is $1 \leq i \leq n$ so that $\{\ell_j^{(i)} : 1 \leq j \leq k_i)$ does not contain both a propositional variable and its negation. Similarly given a wff A in cnf, it is easy to check if it is a tautology or not: it is a tautology iff all the conjuncts are tautologies, which happens iff for all $1 \leq i \leq n$, $\{\ell_j^{(i)} : 1 \leq i \leq k_i\}$ contains both a propositional variable and its negation.

Examples 1.6.16

(i) $(p \wedge q) \vee (p \wedge \neg r \wedge s) \vee (p \wedge s \wedge \neg p)$ is satisfiable.

(ii) $(p \vee q) \wedge (r \vee s \vee \neg r) \wedge (t \vee p \vee \neg s \vee \neg t)$ is not a tautology.

This is one reason that cnf and dnf are useful forms to be able to put a wff in. However it is not known how to transform efficiently any given wff A to an equivalent one in dnf (or cnf).

1.7 König's Lemma and Applications

We will now temporarily take a break from propositional logic and discuss a basic result in combinatorics, which we will use in the next section to prove an important result about propositional logic known as the *Compactness Theorem*.

1.7.A Graphs and Trees

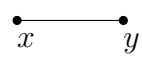
Definition 1.7.1 A (nondirected, simple) *graph* consists of a set of *vertices* V and a set $E \subseteq V^2$ of *edges* with the property that

$$(x, y) \in E \text{ iff } (y, x) \in E,$$

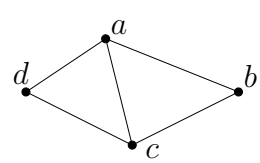
$$(x, x) \notin E.$$

If $e = (x, y) \in E$ is an edge, we say that x, y are *adjacent*.

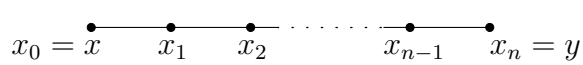
We represent a graph geometrically by drawing points for vertices and connecting adjacent vertices with lines, as shown.



Example 1.7.2 This is a graph with 4 vertices a, b, c, d and 5 edges (a, b), $(b, c), (c, d), (a, d), (a, c)$. (Formally, E has size 10.)



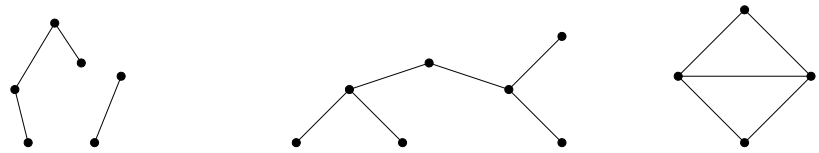
Definition 1.7.3 A *path* from $x \in V$ to $y \in V (x \neq y)$ is a finite sequence $x_0 = x, x_1, \ldots, x_n = x$ of distinct successively adjacent vertices (i.e., each (x_i, x_{i+1}) is an edge).



Definition 1.7.4 A graph is *connected* if for every $x, y \in V (x \neq y)$ there is a path from x to y.

Definition 1.7.5 A graph is a *tree* if for any $x \neq y$ there is a *unique* path from x to y.

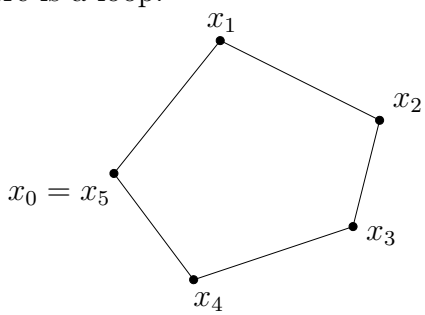
Examples 1.7.6
Here are some examples of graphs that are and are not trees.



not a tree (not connected) | tree | *not* a tree (has loops)

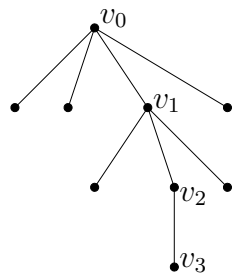
Equivalently, a connected graph is a tree exactly when it contains no *loops* (or *cycles*), where a *loop* is a sequence of successively adjacent vertices $x_0, x_1, \ldots, x_n$ with $x_0 = x_n$ and $x_0, \ldots, x_{n-1}$ distinct.

Example 1.7.7 Here is a loop:

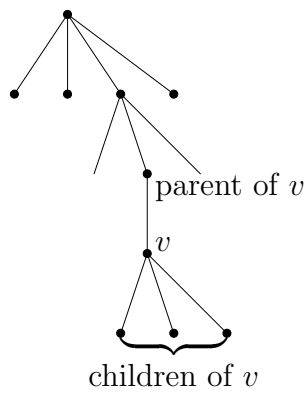


Definition 1.7.8 A *rooted* tree is a tree with a distinguished vertex called the *root*.

The root is usually denoted by v_0. For every vertex $v \neq v_0$ there is a unique path $v_0, v_1, \ldots, v_n = v$.

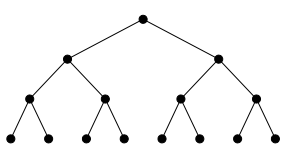


Definition 1.7.9 The *parent* of v is the vertex v_{n-1} and the *children* of v are all the vertices v' such that $v_0, v_1, \ldots, v_n, v_{n+1} = v'$ is the unique path from v_0 to v'.

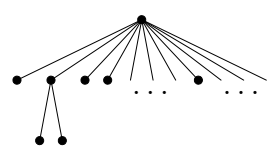


1.7.B König's Lemma

Definition 1.7.10 A tree is *finite splitting* if every vertex has only finitely many children (thus only finitely many adjacent vertices).

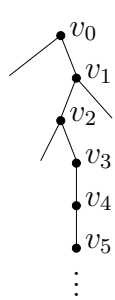


finite splitting

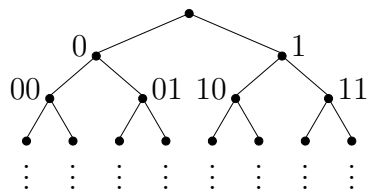


not finite splitting

Definition 1.7.11 An *infinite branch* of a rooted tree is an infinite sequence $v_0, v_1, v_2, \ldots,$ where v_{n+1} is a child of v_n.

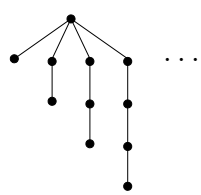


For example, if T is the infinite binary tree (in which each vertex has exactly two children), then the infinite branches correspond exactly to the infinite binary sequences $a_1, a_2, a_3, a_4, \ldots$ (each $a_i = 0$ or 1), where we interpret 0 as going left and 1 as going right.

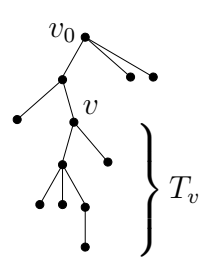


Theorem 1.7.12 (König's Lemma) *If a finite splitting tree has infinitely many vertices, then it has an infinite branch.*

Note that this fails if the tree is *not* finite splitting. Consider the following counterexample:



Proof. Let T be the given tree. For each vertex v of the tree, let T_v be the subtree of T consisting of v, the children of v, the grandchildren of v, etc., i.e., consisting of v and all its descendents, and all the edges connecting them. Denote by V the set of vertices of T, and by V_v the set of vertices of T_v.



We will use the following version of the *Pigeon Hole Principle*: If X is an infinite set and $X = X_1 \cup \cdots \cup X_n$, then some X_i, $1 \leq i \leq n$, is infinite.

Now, since T is finite splitting, v_0 has only finitely many children, say $c_1, \ldots, c_n$. Then $V = \{v_0\} \cup V_{c_1} \cup \cdots \cup V_{c_n}$, so some $V_{c_{i_1}}$ is infinite. Put $v_1 = c_{i_1}$. Let then $d_1, \ldots, d_m$ be the children of v_1. We have $V_{v_1} = \{v_1\} \cup V_{d_1} \cup \cdots \cup V_{d_m}$, so one of the V_{d_i}, say $V_{d_{i_2}}$ is infinite. Put $v_2 = d_{i_2}$, etc. Proceeding this way, we define an infinite path $v_0, v_1, v_2, \ldots$ (so that for each n, V_{v_n} is infinite). $\dashv$

1.7.C Domino Tilings

As an application of König's Lemma, we will consider the following tiling problem in the plane.

Definition 1.7.13 A *domino system* consists of a *finite* set $\mathcal{D}$ of domino types, where a *domino type* is a unit square with each side labeled.

Examples 1.7.14
Here are some example domino types:

$$\boxed{\begin{array}{ccc} & b & \\ a & & a \\ & c & \end{array}} \qquad \boxed{\begin{array}{ccc} & b & \\ a & & c \\ & d & \end{array}} \qquad \boxed{\begin{array}{ccc} & 0 & \\ 1 & & 2 \\ & 3 & \end{array}}$$

Definition 1.7.15 A *tiling* of the plane by $\mathcal{D}$ consists of a filling-in of the plane by dominoes of type in $\mathcal{D}$, so that adjacent dominoes have matching labels at the sides where they touch. (Dominoes *cannot* be rotated.)

Example 1.7.16 Here is a tiling of the plane:

$$\begin{array}{c|c|c|c|c}
 & \cdot\,\cdot & \cdot\,\cdot & \cdot\,\cdot & \\
\hline
\vdots & \begin{array}{ccc} & b & \\ b & & a \\ & c & \end{array} & \begin{array}{ccc} & b & \\ a & & c \\ & d & \end{array} & \begin{array}{ccc} & d & \\ c & & b \\ & a & \end{array} & \vdots \\
\hline
\vdots & \begin{array}{ccc} & c & \\ c & & d \\ & b & \end{array} & \begin{array}{ccc} & d & \\ d & & b \\ & b & \end{array} & \begin{array}{ccc} & a & \\ b & & c \\ & d & \end{array} & \vdots \\
\hline
 & \cdot\,\cdot & \cdot\,\cdot & \cdot\,\cdot &
\end{array}$$

For this tiling, $\mathcal{D}$ contains at least

$$\boxed{\begin{array}{ccc} & b & \\ b & & a \\ & c & \end{array}} \quad \boxed{\begin{array}{ccc} & b & \\ a & & c \\ & d & \end{array}} \quad \boxed{\begin{array}{ccc} & d & \\ c & & b \\ & a & \end{array}} \quad \boxed{\begin{array}{ccc} & c & \\ c & & d \\ & b & \end{array}} \quad \boxed{\begin{array}{ccc} & d & \\ d & & b \\ & b & \end{array}} \quad \boxed{\begin{array}{ccc} & a & \\ b & & c \\ & d & \end{array}}$$

Problem. Given $\mathcal{D}$, can one tile the plane by $\mathcal{D}$?

Examples 1.7.17

(i) Suppose $\mathcal{D}$ consists of

$$\begin{array}{|ccc|}\hline & 1 & \\ 3 & & 1 \\ & 2 & \\ \hline\end{array}\quad \begin{array}{|ccc|}\hline & 4 & \\ 1 & & 2 \\ & 5 & \\ \hline\end{array}\quad \begin{array}{|ccc|}\hline & 7 & \\ 2 & & 3 \\ & 8 & \\ \hline\end{array}$$
$$\begin{array}{|ccc|}\hline & 2 & \\ 6 & & 4 \\ & 1 & \\ \hline\end{array}\quad \begin{array}{|ccc|}\hline & 5 & \\ 4 & & 5 \\ & 4 & \\ \hline\end{array}\quad \begin{array}{|ccc|}\hline & 8 & \\ 5 & & 6 \\ & 7 & \\ \hline\end{array}$$

Then $\mathcal{D}$ can tile the plane, by repeating the following pattern:

$$\begin{array}{|ccc|ccc|ccc|}\hline & 1 & & & 4 & & & 7 & \\ 3 & & 1 & 1 & & 2 & 2 & & 3 \\ & 2 & & & 5 & & & 8 & \\ \hline & 2 & & & 5 & & & 8 & \\ 6 & & 4 & 4 & & 5 & 5 & & 6 \\ & 1 & & & 4 & & & 7 & \\ \hline\end{array}$$

(ii) Suppose, on the other hand, that $\mathcal{D}$ consists of

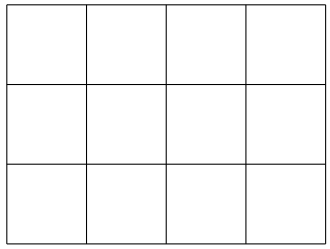
$$\begin{array}{|ccc|}\hline & 3 & \\ 1 & & 2 \\ & 1 & \\ \hline\end{array}\quad \begin{array}{|ccc|}\hline & 2 & \\ 2 & & 3 \\ & 2 & \\ \hline\end{array}\quad \begin{array}{|ccc|}\hline & 1 & \\ 3 & & 1 \\ & 2 & \\ \hline\end{array}$$

This $\mathcal{D}$ cannot tile the plane, as the following forced configuration shows:

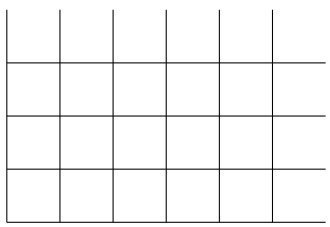
$$\begin{array}{ccc|ccc|ccc}\cline{4-6} & & & & 3 & & & & \\ & & & 1 & & 2 & & & \\ & & & & 1 & & & & \\ \hline & 2 & & & 1 & & & 3 & \\ 2 & & 3 & 3 & & 1 & 1 & & 2 \\ & 2 & & & 2 & & & 1 & \\ \hline\end{array}$$

Starting from the $\begin{array}{|ccc|}\hline & 3 & \\ 1 & & 2 \\ & 1 & \\ \hline\end{array}$ on the right, we are forced to form this configuration, which cannot be completed to a tiling. Starting from $\begin{array}{|ccc|}\hline & 1 & \\ 3 & & 1 \\ & 2 & \\ \hline\end{array}$ in the middle, we are again forced to this configuration. Finally, starting from $\begin{array}{|ccc|}\hline & 2 & \\ 2 & & 3 \\ & 2 & \\ \hline\end{array}$ on the left we are again forced to this configuration.

We can similarly define what it means to tile a rectangular finite region of the plane, like



or an infinite region, like



We just impose no restriction on the boundaries. Here is then a surprising fact:

Theorem 1.7.18 *For any given (finite) set $\mathcal{D}$ of domino types, $\mathcal{D}$ can tile the plane iff $\mathcal{D}$ can tile the upper right quadrant.*

Proof. We will actually show that the following are equivalent:

(i) $\mathcal{D}$ can tile the plane.

(ii) $\mathcal{D}$ can tile the upper right quadrant.

(iii) For each $n = 1, 2, \ldots,$ $\mathcal{D}$ can tile the $n \times n$ square.

It is clear that (i) implies (ii) implies (iii), so it is enough to show that (iii) implies (i).

So assume that for each $n = 1, 2, \ldots,$ $\mathcal{D}$ can tile the $n \times n$ square. We build a tree as follows:

The children of the root v_0 are all possible tilings of the 1×1 square, i.e., all domino types in $\mathcal{D}$. They are only finitely many. Let a be a typical one of them. Then its children are all the tilings of the 3×3 square consistent with a (viewed as being in the middle of the 3×3 square). Let b a typical one of them. Then its children are the tilings of the 5×5 square consistent with b, etc.

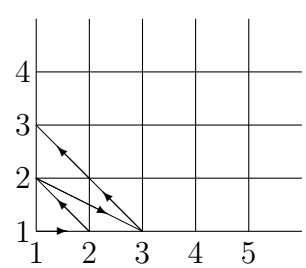
This tree is finite splitting, since for each fixed tiling of a $(2n+1)\times(2n+1)$ square there are only finitely many ways of extending it to a tiling of the $(2n+3)\times(2n+3)$ square, since $\mathcal{D}$ is finite. The tree is also infinite, since for each $n \geq 1$ there is some tiling of the $(2n-1)\times(2n-1)$ square, say u_n, and if we let $u_1, u_2, \ldots u_{n-1}$ be the tilings of the middle $1\times 1,\ 3\times 3,\ \ldots, (2n-3)\times(2n-3)$ squares contained in u_n, then $u_0, u_1, \ldots, u_n$ are all vertices in this tree, so for each n the tree has at least n vertices, i.e., it is infinite. So by König's Lemma there is an infinite branch of the tree, say $u_0, u_1, u_2, \ldots u_n, \ldots$ This gives, in an obvious way, a tiling of the plane by $\mathcal{D}$. $\dashv$

Remark. It can be proved that there is *no* algorithm to check whether a given $\mathcal{D}$ can tile the plane or not.

This has the following interesting implication. A *periodic tiling* by $\mathcal{D}$ consists of a tiling of an $(n\times m)$-rectangle, so that the top and bottom labels match and so do the right and left ones, so by repeating it we can tile the plane.

Theorem 1.7.19 *There is a domino system $\mathcal{D}$ which can tile the plane, but has no periodic tiling.*

Proof. If this fails, then for every $\mathcal{D}$ which can tile the plane, there is a periodic tiling. Then we can devise an algorithm for checking whether a given $\mathcal{D}$ can tile the plane or not, which is a contradiction. First enumerate, in some standard way, all pairs (n, m), $(n \geq 1, m \geq 1)$ (e.g. as shown in the figure), say (n_i, m_i), $i = 1, 2, \ldots$ Then generate all tilings of the $(n_1 \times m_1)$-rectangle, the $(n_2 \times m_2)$-rectangle (if any), etc.



For each fixed (n_i, m_i) this is a finite process. Stop the process when some (n_i, m_i) is found for which either there is a periodic tiling of the $(n_i \times m_i)$-rectangle, or else there is *no* tiling of the $(n_i \times m_i)$-rectangle. In the first case, $\mathcal{D}$ can tile the plane and in the second it cannot. The only thing left to prove

is that this process *terminates*, i.e. for some i this must happen. But this is the case, since either $\mathcal{D}$ can tile the plane and so by our assumption there is a periodic tiling (of some $(n_i \times m_i)$-rectangle)) or else $\mathcal{D}$ cannot tile the plane, so, by the proof of Theorem 1.6.2, $\mathcal{D}$ cannot tile the $(n_i \times m_i)$-rectangle for some $n_i = m_i$. $\dashv$

1.7.D Compactness of $[0, 1]$

As another application, we will use König's Lemma to prove that the unit interval [0,1] is *compact*. This means the following: Let $(a_i, b_i),\ i = 1, 2, \ldots$ be a sequence of open intervals such that

$$[0, 1] \subseteq \bigcup_i (a_i, b_i).$$

Then there are $i_1, \ldots, i_k$ such that

$$[0, 1] \subseteq (a_{i_1}, b_{i_1}) \cup \cdots \cup (a_{i_k}, b_{i_k}).$$

To prove this, consider the so-called *dyadic* intervals which are obtained by successively splitting [0,1] in half. They can be pictured as a binary tree.

We will prove that there is an n such that every one of the dyadic intervals at the nth level of the tree (i.e., those with denominators 2^{-n}) is contained in some (a_i, b_i) (perhaps different (a_i, b_i) for different dyadic intervals). This proves what we want.

The proof is by contradiction: If this fails, then for each n there is some dyadic interval at the nth level which is not contained in any (a_i, b_i). Consider then the subtree T of the tree of dyadic intervals, consisting of all vertices (i.e., dyadic intervals) I which are not contained in any (a_i, b_i). Then for each n, there is an $I = I_n$ at the n-th level belonging to T, and if $I_0 = [0, 1],\ I_1, \ldots, I_n$ is the unique path from the root to I_n, it is clear that $I_0 \supseteq I_1 \supseteq \cdots \supseteq I_n$, so no one of $I_0, I_1, \ldots, I_n$ are contained in any (a_i, b_i), so the tree T has at least n vertices for each n, thus it is infinite. It is clearly finite splitting. So by König's Lemma, it has an infinite branch $I_0, I_1, I_2, \ldots$ Then $I_0 \supseteq I_1 \supseteq I_2 \supseteq \ldots$ are closed intervals and the length of I_n is 2^{-n}, so there is a unique point $x \in \bigcap_n I_n$. Since $[0, 1] \subseteq \bigcup_i (a_i, b_i)$, for some i we have $x \in (a_i, b_i)$, and if n is large enough so that $\min\{x - a_i, b_i - x\} > 2^{-n}$, then $I_n \subseteq (a_i, b_i)$, a contradiction.

1.8 The Compactness Theorem

We now return to propositional logic. In this section we will prove a basic result known as the Compactness Theorem and some applications of it.

1.8.A The Compactness Theorem

The compactness theorem has two equivalent versions.

Theorem 1.8.1 (Compactness Theorem I) *Let S be any set of formulas in propositional logic. If every finite subset $S_0 \subseteq S$ is satisfiable, then S is satisfiable.*

Theorem 1.8.2 (Compactness Theorem II) *Let S be any set of formulas in propositional logic, and A any formula. Then if $S \models A$, there is a* finite *subset $S_0 \subseteq S$ such that $S_0 \models A$.*

First we show that these two forms are equivalent.

I implies II. Assume I holds *for any* S. Fix then S and A with $S \models A$. Then $S' = S \cup \{\neg A\}$ is not satisfiable, so, by applying I to S', we have a finite $S'_0 \subseteq S' = S \cup \{A\}$, so that S'_0 is not satisfiable. Say $S'_0 \subseteq S_0 \cup \{\neg A\}$, with $S_0 \subseteq S$ finite. Then $S_0 \cup \{\neg A\}$ is not satisfiable, so $S_0 \models A$.

II implies I. Say S is not satisfiable. Then $S \models \perp$, where $\perp = p \wedge \neg p$. So by II $S_0 \models \perp$ for some finite $S_0 \subseteq S$. Then S_0 is unsatisfiable.

We will now prove form I of the Compactness Theorem.

Proof of 1.8.1. We assume that every finite subset $S_0 \subseteq S$ is satisfiable. We will then build a finite splitting tree which is infinite and thus, by König's Lemma, has an infinite branch. This infinite branch will give a truth assignment satisfying S.

First let us notice that we can enumerate in a sequence $A_1, A_2, A_3, \ldots, A_n, \ldots$ all wff. So we can enumerate in a sequence

$$S = \{B_1, B_2, B_3, \ldots, B_n, \ldots\}$$

all wff in S. Next $k_1 < k_2 < k_3 < \cdots < k_n < \ldots$ are chosen so that

$$B_n = B_n(p_1, \ldots, p_{k_n})$$

i.e, all the propositional variables of B_n are among $p_1, \ldots, p_{k_n}$.

We will now build a tree as follows: Let v_0 be the root. The children of v_0 are all valuations $v_1 = (\epsilon_1, \ldots, \epsilon_{k_1}) \in \{0,1\}^{k_1}$ which satisfy B_1. Fix any such v_1, say $v_1 = (\epsilon_1, \ldots, \epsilon_{k_1})$. Its children are all valuations $v_2 = (\epsilon_1, \ldots, \epsilon_{k_1}, \epsilon_{k_1+1}, \ldots, \epsilon_{k_2}) \in \{0,1\}^{k_2}$, which agree with v_1 in their first k_1 values and also satisfy both B_1 and B_2, etc.

First we argue that this tree is infinite: Fix any $n \geq 1$. By assumption there is a valuation $v_n = \{\epsilon_1, \ldots, \epsilon_{k_n}\} \in \{0,1\}^{k_n}$ which satisfies $\{B_1, \ldots, B_n\}$. Let for $1 \leq m \leq n$, $v_m = \{\epsilon_1, \ldots, \epsilon_{k_m}\}$, i.e., the restriction of v_n to the first k_m variables. Then clearly v_m satisfies $\{B_1, \ldots, B_m\}$, so it is a vertex of our tree, and v_{m+1} is a child of v_m. So T has at least n vertices for each n, i.e., it is infinite. Clearly T is finite splitting. So, by König's Lemma, it has an infinite branch $v_0, v_1, v_2, \ldots, v_n, \ldots$ Then

$$\begin{aligned}
v_1 &= \{\epsilon_1, \ldots, \epsilon_{k_1}\} \\
v_2 &= \{\epsilon_1, \ldots, \epsilon_{k_1}, \epsilon_{k_1+1}, \ldots, \epsilon_{k_2}\} \\
&\vdots \\
v_n &= \{\epsilon_1, \ldots, \epsilon_{k_1}, \epsilon_{k_1+1}, \ldots, \epsilon_{k_2}, \epsilon_{k_2+1}, \ldots, \epsilon_{k_n}\} \\
&\vdots
\end{aligned}$$

So if $\nu = \{\epsilon_1, \epsilon_2, \ldots\}$, ν is a valuation which satisfies all the B_n, i.e., it satisfies S, and so S is satisfiable. $\dashv$

1.8.B A Proof of König's Lemma

We have proved the Compactness Theorem by using König's Lemma. On the other hand, we can also prove König's Lemma by using the Compactness Theorem as follows:

Consider a tree T with root v_0 which is infinite, but has finite splitting. Notice then that the set V of the vertices of T can be enumerated in a sequence. Introduce now a propositional variable p_v for each vertex v of T. (Our preceding remark implies that we can enumerate these variables in a sequence $p_1, p_2, \ldots$, but there is no point in doing that explicitly.) Consider now the following set S of wff, which we can view as "axioms", where the intuitive meaning of the variable p_v is that "p_v is true" iff "v is in the infinite branch we try to find".

(i) p_{v_0}

(ii) $p_v \Rightarrow \neg p_u$, if $v \neq u$ are at the same level or for some n, v is at level n, u is at level $n+1$ and u is not a child of v.

(iii) If for each n, $u_1, \ldots, u_{k_n}$ are all the vertices at level n, then we introduce the wff $p_{u_1} \vee p_{u_2} \vee \cdots \vee p_{u_{k_n}} (n = 1, 2, \ldots)$. (Here u is *at level* n if the path from v_0 to v has length n.)

Assume that this set S of wff is satisfiable, say by the valuation ν. Then consider the following set of vertices

$$v \in P \text{ iff } \nu(p_v) = 1.$$

We claim that P is an infinite branch, i.e., P contains v_0 and exactly one vertex at level $n = 1, 2, \ldots$, say v_n, so that v_{n+1} is a child of v_n. First by (i), $v_0 \in P$. By (iii), for each $n \geq 1$, there is at least one vertex v at level n with $v \in P$ and by (ii) there is exactly one, say v_n. It only remains to prove that v_{n+1} is a child of v_n. Otherwise, by (ii), $\nu(p_{v_n} \Rightarrow \neg p_{v_{n+1}}) = 1$, but also $\nu(p_{v_n}) = \nu(p_{v_n+1}) = 1$, a contradiction.

By the Compactness Theorem, it only remains to show that every finite subset S_0 of S is satisfiable. Fix such an S_0. Then for some large enough N_0, all the "axioms" (i), (ii), (iii) occurring in S_0 contain variables p_v with v at a level $n \leq N_0$. Since T is infinite, every level is nonempty, so fix a node v_{N_0} at level N_0 and look at the path $v_0, v_1, \ldots, v_{N_0-1}, v_{N_0}$ from v_0 to v_{N_0}. Consider then the valuation to all the variables p_v with v at a level $n \leq N_0$ defined as follows:

$$\nu(P_v) = 1 \text{ iff } v \text{ is one of } v_0, v_1, \ldots, v_{N_0}.$$

Then all the "axioms" (i)-(iii) belonging to S_0 are satisfied, i.e., S_0 is satisfiable, and the proof is complete. $\dashv$

1.8.C Partial Orders

We will give another application of the Compactness Theorem involving partial orders.

Definition 1.8.3 A *partial order* on a set X is a relation $x < y$ between members of X such that it satisfies the following properties:

(i) If $x < y$, then $y \not< x$ (in particular $x \not< x$),

(ii) $x < y$ and $y < z$ imply $x < z$.

Definition 1.8.4 A partial order is called a *linear order* (or *total order*) if it also satisfies:

(iii) $x < y$ or $y < x$ or $x = y$.

Examples 1.8.5

(i) Let $X = P(A) =$ set of all subsets of A, and

$$x < y \text{ iff } x \subseteq y \text{ and } x \neq y.$$

This is a partial order but is not linear, if A has more than one element.

(ii) Let $X = \mathbb{N}$ and $<$ be the usual order of the integers. Then $<$ is a linear order.

Definition 1.8.6 We say that a linear order $<$ on a set X *extends* a partial order $<'$ on X if $x <' y$ implies $x < y$.

Example 1.8.7 On $\mathbb{N}^+ = \{1, 2, 3, \dots\}$ consider the following two partial orders:

(i) $<$ is the usual order (so it is linear).

(ii) $x <' y$ iff $x \neq y$ and x divides y (this is not linear).

Then $<$ extends $<'$. If on the other hand we consider

(iii) $x <' y$ iff (x is even and y is odd) or (x, y are even and $x < y$),

then $<$ does not extend $<'$.

Theorem 1.8.8 *Let $X = \{x_1, x_2, \dots\}$ be a countable set and $<'$ a partial order on X. Then $<'$ can be extended to a linear order $<$.*

Proof. First we note the following finite version of this theorem.

Lemma 1.8.9 *Let $X = \{x_1, \dots, x_n\}$ be a finite set. Then every partial order on X can be extended to a linear order.*

This can be proved by induction, and we will give the proof at the end.

Consider now the infinite case, where $X = \{x_1, x_2, \dots\}$, $x_i \neq x_j$, if $i \neq j$. Introduce for each pair (i, j), $i = 1, 2, \dots$, $j = 1, 2, \dots$, a propositional variable $p_{i,j}$. The intuitive meaning is that "$p_{i,j}$ is true" if "$x_i < x_j$ in the linear order $<$ we try to find". Let S consist of the following "axioms":

(i) $p_{i,j} \Rightarrow \neg p_{j,i}$ for all i, j;

(ii) $p_{i,j} \wedge p_{j,k} \Rightarrow p_{i,k}$, for all i, j, k;

(iii) $p_{i,j} \vee p_{j,i}$, for all $i \neq j$;

(iv) $p_{i,j}$, whenever $x_i <' x_j$.

Suppose S is satisfiable by some valuation ν. Then define the following relation on X:

$$x_i < x_j \text{ iff } \nu(p_{i,j}) = 1.$$

By (i)-(iii), $<$ is a linear order and by (iv) it extends $<'$. To show that S is satisfiable, it is enough, by the Compactness Theorem, to show that every finite subset $S_0 \subseteq S$ is satisfiable. Fix such an S_0. Then for some large enough N_0, S_0 contains only variables $p_{i,j}$ with $i, j \leq N_0$. Let $X_0 = \{x_1, \dots, x_{N_0}\}$ and restrict the partial order $<'$ to X_0. Call it $<'_{X_0}$. By the lemma, there is a linear order $<_{X_0}$ on X_0 extending $<'_{X_0}$. Use this to define the following valuation to the variables $p_{i,j}$ for $i, j \leq N_0$:

$$\nu(p_{i,j}) = 1 \text{ iff } x_i <_{X_0} x_j.$$

Then all the axioms (i)-(iv) in S_0 are satisfied, i.e., S_0 is satisfiable.

It remains to give the proof of the lemma: Let n be the cardinality of X. We prove this by induction on n. If $n = 1$ it is obvious, as the only partial order on a set of cardinality 1 is linear. So assume X has cardinality $n + 1$. Consider a partial order $<'$ on X. Since X is finite, it has a minimal element, say x_0 (x is *minimal* if there is no $y <' x$). Let $x_1, \dots, x_n$ be the rest of the elements of X. Restrict $<'$ to $\{x_1, \dots, x_n\}$ and by induction hypothesis extend this to a linear order $\prec$ on $\{x_1, \dots, x_n\}$. Then define a linear order $<$ on $\{x_0, x_1, \dots, x_n\}$ by letting

$$x_i < x_j \text{ iff } x_i \prec x_j, \text{ for } 1 \leq i, j \leq n$$

and

$$x_0 < x_i, \text{ for } 1 \leq i \leq n$$

(i.e., the order $<$ agrees with $\prec$ on $\{x_1, \ldots, x_n\}$ and x_0 is $<$ than any element of $\{x_1, \ldots x_n\}$.) $\dashv$

1.9 Ramsey Theory

Let $\mathbb{N}$ denote the set of natural numbers, $\mathbb{N} = \{0, 1, 2, \ldots\}$. For $0 < m, l, k, n \in \mathbb{N}$, set $\mathbf{m} = \{1, 2, \ldots, m\}, \mathbf{l} = \{1, 2, \ldots, l\}$ and let $m \to (n)^k_l$ be the following assertion.

Whenever $f : [\mathbf{m}]^k \to \mathbf{l}$, there is $H \in [\mathbf{m}]^n$ homogeneous for f.

Similarly, let $\mathbb{N} \to (\mathbb{N})^k_l$ mean that for every $f : [\mathbb{N}]^k \to \mathbf{l}$ there is $H \subseteq \mathbb{N}$ infinite and homogeneous for f. Here,

- $[X]^k$ is the collection of k-sized subsets of X.
- Given $f : [X]^k \to \mathbf{l}$, $H \subseteq X$ is *homogeneous* for f iff whenever $s, t \in [H]^k$, then $f(s) = f(t)$.

The classic Ramsey Theorem was proved in 1928. Frank Plumpton Ramsey was born in 1903 in Cambridge, and died in 1930 in London as a result of an attack of jaundice. An enthusiastic logician, he considered mathematics to be part of logic. His second paper, *On a problem of formal logic* was read to the London Mathematical Society on 13 December 1928 and published posthumously in the Proceedings of the London Mathematical Society in 1930. There he proves Ramsey Theorem and uses it to deduce a result on propositional logic.

1.9.A Ramsey Theorem, infinite version

Theorem 1.9.1 *For all $0 < k, l \in \mathbb{N}$, $\mathbb{N} \to (\mathbb{N})^k_l$. In English: Given any $f : [\mathbb{N}]^k \to \mathbf{l}$ there is an infinite subset of $\mathbb{N}$ homogeneous for f.*

Proof. The proof is by induction on k. For $k = 1$ the result is obvious; it simply says that if an infinite set is partitioned into finitely many pieces, one of the pieces is itself infinite, here we are identifying a number n with the singleton $\{n\}$.

Assume we know the result for k and we are given a function $f : [\mathbb{N}]^{k+1} \to \mathbf{l}$. Clearly, if X is infinite and $h : [X]^k \to \mathbf{l}$, then there is an infinite $Y \subseteq X$ homogeneous for h. This simple observation (that we can replace $\mathbb{N}$ with any infinite set) is key to the argument.

We start by defining a decreasing sequence of infinite subsets of $\mathbb{N}$, $A_1 \supset A_2 \supset A_3 \supset \cdots$ with the property that if $a_n = \min A_n$ then $a_1 < a_2 < \cdots$.

Let $A_1 = \mathbb{N}$ (so $a_1 = 1$). In general, given A_n, define $f_n : [A_n \setminus \{a_n\}]^k \to \mathbf{l}$ by setting $f_n(s) = f(\{a_n\} \cup s)$ and use the inductive hypothesis to find $A_{n+1} \subseteq A_n \setminus \{a_n\}$ infinite and homogeneous for f_n.

Now consider the set $A = \{a_1, a_2, \dots\}$. Notice that, by construction, if $s, t \in [A]^{k+1}$ and $a_n = \min(s) = \min(t)$ then $s \setminus \{a_n\}, t \setminus \{a_n\} \in [A_{n+1}]^k$, so $f_n(s \setminus \{a_n\}) = f_n(t \setminus \{a_n\})$ (since A_{n+1} is homogeneous for f_n), i.e., $f(s) = f(t)$. This means that $f(s)$ only depends on $\min(s)$ for any $s \in [A]^{k+1}$ (usually one says that A is *min-homogeneous* for f).

Consider now the function $g : A \to \mathbf{l}$ given by $g(a) = f(s)$ where s is *any* element of $[A]^{k+1}$ with $\min(s) = a$. By the case $k = 1$, there is an infinite $B \subset A$ homogeneous for g. But then it is easy to see that B is also homogeneous for f. $\dashv$

Remark. Another proof of Ramsey theorem can be obtained from König's lemma. Instead, we use König's lemma to deduce a finite version of Ramsey's result.

1.9.B Ramsey Theorem, finite version

Theorem 1.9.2 *For all $n, k, l \in \mathbb{N}$ there is m such that $m \to (n)^k_l$. In English: Given any n, k, l there is m sufficiently large that whenever $f : [\mathbf{m}]^k \to \mathbf{l}$, there is $H \subset \mathbf{m}$, $|H| = n$ that is homogeneous for f.*

Proof. We provide a proof using König's lemma and the infinite version of Ramsey theorem. Suppose towards a contradiction that for some fixed values of n, k, l there is no m as required. This means that for every m there is a function $f : [\mathbf{m}]^k \to \mathbf{l}$ without homogeneous sets of size n. Clearly, if $m_1 < m_2$ and $f : [\mathbf{m}_2]^k \to \mathbf{l}$ has no such homogeneous sets, then f extends a function g with domain $[\mathbf{m}_1]^k$ with the same property, and any such function f is an extension of some such function g. So we can define a rooted tree $\mathcal{T}$ as follows: Any node of the tree at level m is a function $f : [\mathbf{m}]^k \to \mathbf{l}$ without homogeneous sets of size n. The children of f are all the functions

$h : [\{1, \ldots, m+1\}]^k \to \mathbf{l}$ extending f and with the same property. Clearly, $\mathcal{T}$ is finite splitting and our assumption implies that it is infinite. By König's lemma it has an infinite branch, which consists of functions with longer and longer domains that cohere. We can thus take their union and obtain a function $F : [\mathbb{N}]^k \to \mathbf{l}$. It is easy to see that F admits no homogeneous sets of size n. But this is impossible, by the infinite version of Ramsey Theorem. Contradiction. $\dashv$

Remark. This kind of argument is very powerful, but it has an important drawback. Namely, it is nonconstructive. There is no known method that allows us to extract from this proof for given n, k, l a bound on how large m must be. Different, more finitistic, proofs can be given that provide explicit bounds. The computation of actual values of m, the so-called *Ramsey numbers* (as opposed to mere upper or lower bounds) is very much an open problem.

For the application to logic that the theorem was originally conceived for, and many other applications of this method to combinatorics, we suggest to look at R. Graham, B. Rothschild, J. Spencer, **Ramsey Theory**, 2nd ed., Wiley, New-York, 1990.

In general one refers to this use of König's lemma that serves as a bridge between infinite and finite statements as a *compactness* argument.

1.9.C A few other applications of compactness

Say that a function $f : [\mathbf{m}]^k \to \{1, \ldots, m-k\}$ is *regressive* iff $f(s) < \min(s)$ for all $s \in [\mathbf{m}]^k$. Similarly we can talk of a function $f : [\mathbb{N}]^k \to \mathbb{N}$ being regressive. Say that H is *min-homogeneous* for f iff $f(s) = f(t)$ whenever $s, t \in [H]^k$ and $\min(s) = \min(t)$.

If $f : [\mathbb{N}]^k \to \mathbb{N}$ is regressive, it does not need to admit infinite homogeneous sets. For example, let $f(s) = \min(s)$. However, it does admit min-homogeneous sets.

Theorem 1.9.3 *If $X \subseteq \mathbb{N}$ is infinite and $f : [X]^k \to \mathbb{N}$ is regressive, there is an infinite subset of X min-homogeneous for f.*

The proof is an easy modification of the proof of the infinite version of Ramsey Theorem given above, and it is a useful exercise to work out its details.

Using König's lemma very much as in the proof of the finite version of Ramsey Theorem, one obtains:

Corollary 1.9.4 *For all k and n there is m such that if*

$$f : [\mathbf{m}]^k \to \{1, \ldots, m-k\}$$

is regressive, then it admits a min-homogeneous set of size n.

This was proven by Kanamori and McAloon in the late 80s. However, unlike the finite version of Ramsey Theorem, *there is no* purely finitistic proof of this result. This was also proven by Kanamori and McAloon, using methods of mathematical logic. See *On Gödel incompleteness and finite combinatorics*, Annals of Pure and Applied Logic **33 (1)** (1987), 23–41.

Suppose now that $X \subset \mathbb{N}$ and that $f : [X]^k \to \mathbf{l}$. Let H be a finite subset of X homogeneous for f. Say that H is *large-homogeneous* iff $\min(H) \leq |H|$.

Theorem 1.9.5 *For all k, l, n there is m sufficiently large that any $f : [\mathbf{m}]^k \to \mathbf{l}$ admits a large-homogeneous set of size n.*

The proof of this result can be obtained by a König's lemma argument from an appropriate infinite version, whose proof is again an easy modification of the argument we gave for the infinite version of Ramsey Theorem. Again, it is not possible to prove this result by purely finitistic methods. This was shown by Harrington and Paris in *A mathematical incompleteness in Peano Arithmetic*, in **Handbook of Mathematical Logic**, Jon Barwise ed., North-Holland, 1977.

1.10 The Resolution Method

We would like to have a mechanical procedure (algorithm) for checking whether a given set of formulas logically implies another, that is, given $A_1, \ldots, A_n, A$, whether

$$A_1, \ldots, A_n \models A.$$

We know that this happens if and only if

$$\models (A_1 \wedge \cdots \wedge A_n) \Rightarrow A$$

which happens iff

$$A_1 \wedge \cdots \wedge A_n \wedge \neg A \text{ is unsatisfiable.}$$

So it suffices to have an algorithm to check the (un)satisfiability of a single wff. The method of truth tables gives one such algorithm. We will now develop another method which is often (with various improvements) more efficient in practice.

It will be also an example of a *formal calculus*. By that we mean a set of rules for generating a sequence of strings in a language. Formal calculi usually start with a certain string or strings as given, and then allow the application of one or more "rules of production" to generate other strings.

We have already encountered some formal calculi in the form of recursive definitions. Definition 1.2.6 (Well-Formed Formulas) gives a formal calculus consisting of certain given wffs (propositional variables) and certain rules for producing new wffs (negation and adding binary connectives). We will see yet another example of a formal calculus in section 1.11.A.

Suppose A is a wff which we want to test for satisfiability. First we note that although there is no known efficient algorithm for finding a wff A' in cnf (conjunctive normal form) equivalent to A, it is not hard to show that there is an efficient algorithm for finding a wff A^* in cnf such that:

$$A \text{ is satisfiable iff } A^* \text{ is satisfiable.}$$

This will be discussed in Assignment #4, Problem 1.

So from now on we will only consider wff in cnf, and the Resolution Method applies to such formulas only. Say

$$A = (\ell_{1,1} \vee \cdots \vee \ell_{1,n_1}) \wedge \cdots \wedge (\ell_{k,1} \vee \cdots \vee \ell_{k,n_k})$$

with $\ell_{i,j}$ literals. Since order and repetition in each conjunct

$$\ell_{i,1} \vee \cdots \vee \ell_{i,n_i} \tag{$*$}$$

are irrelevant (for semantic purposes), we can replace $(*)$ by the set of literals

$$c_i = \{\ell_{i,1}, \ell_{i,2}, \ldots, \ell_{i,n_i}\}.$$

Such a set of literals is called a *clause*. It corresponds to the formula $(*)$. So the wff A above can be simply written as a set of clauses (again since the order of the conjunctions is irrelevant)

$$\begin{aligned} C &= \{c_1, \ldots, c_k\} \\ &= \{\{\ell_{i,1}, \ldots \ell_{i,n_1}\}, \ldots, \{\ell_{k,1}, \ldots, \ell_{k,n_k}\}\} \end{aligned}$$

Satisfiability of A means then simultaneous satisfiability of all of its clauses $c_1, \dots, c_k$, i.e., finding a valuation ν which makes c_i true for each i, i.e., which for each i makes some $\ell_{i,j}$ true.

Example 1.10.1

$$\begin{aligned} A &= (p_1 \vee \neg p_2) \wedge (p_3 \vee p_3) \\ c_1 &= \{p_1, \neg p_2\} \\ c_2 &= \{p_3\} \\ C &= \{\{p_1, \neg p_2\}, \{p_3\}\}. \end{aligned}$$

From now on we will deal only with a set of clauses $C = \{c_1, c_2, \dots\}$, which we would even allow to be infinite. Satisfying C means (again) that there is a valuation which satisfies all $c_1, c_2, \dots$, i.e. if $c_i = \ell_{i,1} \vee \cdots \vee \ell_{i,n_i}$, then for all i there is j so that it makes $\ell_{i,j}$ true. (Of course in the case C comes from some wff A, C is a finite set of clauses.)

Notice that if the set of clauses C_A is associated as above to A (in cnf) and C_B to B, then

$$A \wedge B \text{ is satisfiable iff } C_A \cup C_B \text{ is satisfiable.}$$

By convention we also have the *empty clause* $\Box$, which contains no literals. The empty clause is (by definition) unsatisfiable, since for a clause to be satisfied by a valuation, there has to be some literal in the clause which it makes true, but this is impossible for the empty clause, which has no literals.

For a literal u, let $\bar{u}$ denote its "conjugate", i.e.

$$\begin{aligned} \bar{u} &= \neg p, \text{ if } u = p, \\ \bar{u} &= p \text{ if } u = \neg p. \end{aligned}$$

Definition 1.10.2 Suppose now c_1, c_2, c are three clauses. We say that c is a *resolvent* of c_1, c_2 if there is a u such that $u \in c_1$, $\bar{u} \in c_2$ and

$$c = (c_1 \setminus \{u\}) \cup (c_2 \setminus \{\bar{u}\}).$$

We denote this by the diagram

$$\begin{array}{c} c \\ \diagup \quad \diagdown \\ c_1 \quad c_2 \end{array}$$

We allow here the case $c = \square$, i.e. $c_1 = \{u\},\ c_2 = \{\bar{u}\}$.

Examples 1.10.3

(i)

$$\begin{array}{c} \{p, r\} \\ \diagup \quad \diagdown \\ \{p, \neg q, r\} \quad \{q, r\} \end{array}$$

(ii)

$$\begin{array}{c} \{q, \neg q\} \\ \diagup \quad \diagdown \\ \{p, \neg q\} \quad \{\neg p, q\} \end{array} \qquad \begin{array}{c} \{p, \neg p\} \\ \diagup \quad \diagdown \\ \{p, \neg q\} \quad \{\neg p, q\} \end{array}$$

(iii)

$$\begin{array}{c} \square \\ \diagup \quad \diagdown \\ \{p\} \quad \{\neg p\} \end{array}$$

Proposition 1.10.4 *If c is a resolvent of c_1, c_2, then $\{c_1, c_2\} \models c$. (We view here c_1, c_2, c as formulas.)*

Proof. Suppose a valuation ν satisfies both c_1, c_2 and let u be the literal used in the resolution. If $\nu(u) = 1$, then since $\nu(c_2) = 1$ we clearly have $\nu(c_2 \setminus \{\bar{u}\}) = 1$ and so $\nu(c) = 1$. If $\nu(u) = 0$, then $\nu(c_1 \setminus \{u\}) = 1$, so $\nu(c) = 1$. $\dashv$

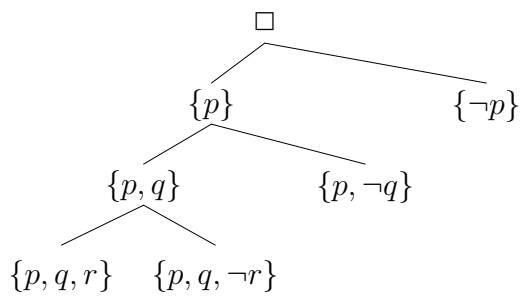
Definition 1.10.5 Let now C be a set of clauses (finite or infinite). A *proof by resolution* from C is a sequence $c_1, c_2, \ldots, c_n$ of clauses such that each c_i is either in C or else it is a resolvent of some c_j, c_k with $j, k < i$. We call c_n the *goal* or *conclusion* of the proof. If $c_n = \square$, we call this a *proof by resolution of a contradiction* from C or simply a *refutation* of C.

Example 1.10.6 Let $C = \{\{p, q, \neg r\}, \{\neg p\}, \{p, q, r\}, \{p, \neg q\}\}$. Then the following is a refutation of C:

$$\begin{array}{ll} c_1 = \{p, q, \neg r\} & (\text{in } C) \\ c_2 = \{p, q, r\} & (\text{in } C) \end{array}$$

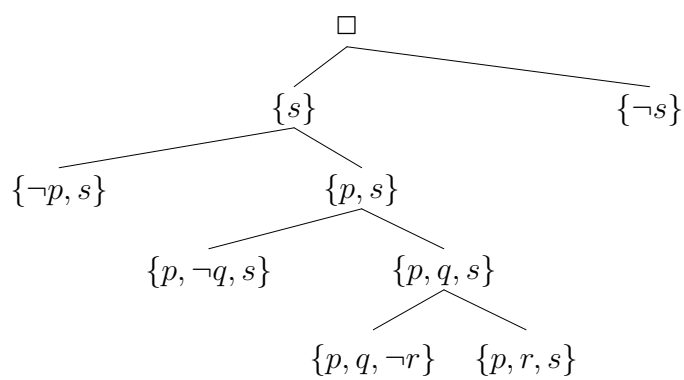
$$
\begin{array}{ll}
c_3 = \{p, q\} & \text{(resolvent of } c_1, c_2 \text{ (by } r)) \\
c_4 = \{p, \neg q\} & \text{(in } C) \\
c_5 = \{p\} & \text{(resolvent of } c_3, c_4 \text{ (by } q)) \\
c_6 = \{\neg p\} & \text{(in } C) \\
c_7 = \square & \text{(resolvent of } c_5, c_6 \text{ (by } p)).
\end{array}
$$

We can also represent this by a tree:



Terminal nodes correspond to clauses in C and each branching $\wedge$ corresponds to creating a resolvent. We call such a tree a *resolution tree*.

Example 1.10.7 Let $C = \{\{\neg p, s\}, \{p, \neg q, s\}, \{p, q, \neg r\}, \{p, r, s\}, \{\neg s\}\}$.



This can be also written as a proof as follows:

$$
\begin{array}{lcl}
c_1 & = & \{p, q, \neg r\} \\
c_2 & = & \{p, r, s\}
\end{array}
$$

$$\begin{array}{lcl} c_3 & = & \{p, q, s\} \\ c_4 & = & \{p, \neg q, s\} \\ c_5 & = & \{p, s\} \\ c_6 & = & \{\neg p, s\} \\ c_7 & = & \{s\} \\ c_8 & = & \{\neg s\} \\ c_9 & = & \Box \end{array}$$

(This proof is not unique. For example, we could move c_8 before c_3 and get another proof corresponding to the same resolution tree. The relationship between proofs by resolution and their corresponding trees is similar to that between parsing sequences and parse trees.)

The goal of proofs by resolution is to prove unsatisfiability of a set of clauses. The following theorem tells us that they achieve their goal.

Theorem 1.10.8 *Let $C = \{c_1, c_2, \dots\}$ be a set of clauses. Then C is unsatisfiable iff there is a refutation of C.*

Proof.

$\Leftarrow$*: Soundness of the proof system.*

Let $d_1, \dots, d_n$ be a proof of resolution from C. Then by Proposition 1.8.1, we can easily prove, by induction on $1 \leq i \leq n$, that

$$C \models d_i.$$

So if $d_n = \Box$, then $C \models \Box$, i.e., C is unsatisfiable.

$\Rightarrow$*: Completeness of the proof system.*

First we can assume that C has no clause c_i which contains, for some literal u, both u and $\bar{u}$ (since such a clause can be dropped from C without affecting its satisfiability).

Notation. If u is a literal, let $C(u)$ be the set of clauses resulting from C by canceling every occurrence of u within a clause of C and eliminating all clauses of C containing $\bar{u}$ (this effectively amounts to setting $u = 0$).

Example. Let $C = \{\{p, q, \neg r\}, \{p, \neg q\}, \{p, q, r\}, \{q, r\}\}$. Then

$$C(r) = \{\{p, \neg q\}, \{p, q\}, \{q\}\}$$
$$C(\bar{r}) = \{\{p, q\}, \{p, \neg q\}\}$$

Note that $u, \bar{u}$ do not occur in $C(u),\ C(\bar{u})$. Note also that if C is unsatisfiable, so are $C(u),\ C(\bar{u})$. Because if ν is a valuation satisfying $C(u)$, then, since $C(u)$ does not contain $u, \bar{u}$, we can assume that ν does not assign a value to u. Then the valuation ν' which agrees on all other variables with ν and gives $\nu(u) = 0$ satisfies C. Similarly for $C(\bar{u})$.

So assume C is unsatisfiable, in order to construct a refutation of C. By the Compactness Theorem there is a finite subset $C_0 \subseteq C$ which is unsatisfiable, so we may as well assume from the beginning that C is finite. Say that all the propositional variables occurring in clauses in C are among $p_1, \ldots, p_n$. We prove then the result by induction on n. In other words, we show that for each n, if C is a finite set of clauses containing variables among $p_1, \ldots, p_n$ and C is unsatisfiable, there is a refutation of C.

$n = 1$. In this case, we must have $C = \{\{p_1\}, \{\neg p_1\}\}$, and hence we have the refutation $\{p_1\},\ \{\neg p_1\},\ \Box$.

$n \to n+1$. Assume this has been proved for sets of clauses with variables among $\{p_1, \ldots, p_n\}$ and consider a set of clauses C with variables among $\{p_1, \ldots, p_n, p_{n+1}\}$. Let $u = p_{n+1}$.

Then $C(u), C(\bar{u})$ are also unsatisfiable and do not contain p_{n+1}, so by induction hypothesis there is a refutation $d_1, \ldots d_m, d_{m+1} = \Box$ for $C(u)$ and a refutation $e_1, \ldots, e_k, e_{k+1} = \Box$ for $C(\bar{u})$.

Consider first $d_1, \ldots, d_{m+1}$. Each clause d_i is in $C(u)$ or comes as a resolvent of two previous clauses. Define then recursively $d'_1, \ldots, d'_m, d'_{m+1}$, so that either $d'_i = d_i$ or $d'_i = d_i \cup \{u\}$.

If $d_i \in C(u)$, then it is either in C and then we put $d'_i = d_i$ or else is obtained from some $d^*_i \in C$ by dropping u, i.e., $d_i = d^*_i \setminus \{u\}$. Then put $d'_i = d^*_i$.

The other case is where for some $j, k < i$, we have that d_i is a resolvent of d_j, d_k, and thus by induction d'_j, d'_k are already defined. The variable used in this resolution is in $\{p_1, \ldots, p_n\}$, so we can use this variable to resolve from d'_j, d'_k to get d'_i.

Thus $d'_{m+1} = \Box$ or $d'_{m+1} = \{p_{n+1}\}$, and $d'_1, \ldots d'_m, d'_{m+1}$ is a proof by resolution from C. If $d'_{m+1} = \Box$ we are done, so we can assume that $d'_{m+1} = \{p_{n+1}\}$, i.e., $d'_1, \ldots, d'_m, \{p_{n+1}\}$ is a proof by resolution from C. Similarly, working with $\bar{u}$, we can define $e'_1, \ldots e'_k, e'_{k+1}$, a proof by resolution from C with $e'_{k+1} = \Box$ or $e'_{k+1} = \{\neg p_{n+1}\}$. If $e'_{k+1} = \Box$ we are done, otherwise $e'_1, \ldots, e'_k, \{\neg p_{n+1}\}$ is a proof by resolution from C. Then

$$d'_1, \ldots, d'_m, \{p_{n+1}\}, e'_1, \ldots, e'_k, \{\neg p_{n+1}\}, \Box$$

is a refutation from C. $\dashv$

Example 1.10.9

$$\begin{aligned} C &= \{\{p,q,\neg r\},\{\neg p\},\{p,q,r\},\{p,\neg q\}\} \qquad (u=r) \\ C(r) &= \{\{\neg p\},\{p,q\},\{p,\neg q\}\} \\ C(\neg r) &= \{\{p,q\},\{\neg p\},\{p,\neg q\}\} \end{aligned}$$

Refutation from $C(r)$	Proof by resolution from C	Refutation from $C(\neg r)$	Proof by resolution from C
$\{p,q\}$	$\to \{p,q,r\}$	$\{p,q\}$	$\to \{p,q,\neg r\}$
$\{p,\neg q\}$	$\to \{p,\neg q\}$	$\{p,\neg q\}$	$\to \{p,\neg q\}$
$\{p\}$	$\to \{p,r\}$	$\{p\}$	$\to \{p,\neg r\}$
$\{\neg p\}$	$\to \{\neg p\}$	$\{\neg p\}$	$\to \{\neg p\}$
$\square$	$\to \{r\}$	$\square$	$\to \{\neg r\}$

$\square$

Remark. Notice that going from n to $n+1$ variables "doubles" the length of the proof, so this gives an exponential bound for the refutation.

Remark. The method of refutation by resolution is non-deterministic–there is no unique way to arrive at it. Various strategies have been devised for implementing it.

One is by following the recursive procedure used in the proof of theorem 1.10.8. Another is by brute force. Start with a finite set of clauses C. Let $C_0 = C$. Let $C_1 = C$ together with all clauses obtained by resolving all possible pairs in $C_0, C_2 = C_1$ together with all clauses obtained by resolving all possible pairs from C_1, etc. Since any set of clauses whose variables are among $p_1, \ldots, p_n$ cannot have more than 2^{2n} elements, this will stop in at most 2^{2n} many steps. Put $C_{2^{2n}} = C^*$. If $\square \in C^*$ then we can produce a refutation proof of about that size (i.e., 2^{2n}). Otherwise, $\square \notin C^*$ and C is satisfiable.

Other strategies are more efficient in special cases, e.g., for Horn formulas (see Assignment #4, for the definition of a Horn formula.)

1.11 A Hilbert-Type Proof System

We will now describe a different formal calculus of proofs, which corresponds more directly to our intuitive concept of a "proof." That is, it will start with axioms and use rules of inference to deduce consequences, rather than dealing with abstract clause sets.

This style of arguing in formal logic, using axioms and rules, was introduced by David Hilbert and his school. Hilbert was born in Königsberg, Prussia, in 1862, and died in Göttingen, Germany, in 1943. He is one of the most important mathematicians of the late 19-th and early 20-th century. A strong advocate of *nonconstructive* methods, this probably started with his work on "invariant theory," where he proved a Basis Theorem without exhibiting an explicit basis. The paper appeared in Mathematische Annalen in 1888. The result states that in any number of variables there is a finite set of generators for the invariants of *quantics*. A quantic is a symmetric tensor of a given degree constructed from tensor powers of a (finite dimensional) vector space. Invariance is measured under invertible operators of the space in itself.

Gordon, the world expert in invariant theory, strongly opposed in vain to the publication of Hilbert's result, due to its nonconstructive nature. On the other hand, Klein said of this result that "I do not doubt this is the most important work on general algebra that the Annalen has ever published."

Hilbert's work in number theory led to his book on the theory of algebraic number fields, recently translated into English. He started his work on logic with the publication in 1899 of the *Grundlagen der Geometrie*, where he advocates the axiomatic approach to mathematics.

In 1900 he delivered the invited address "Mathematical problems" to the Second International Congress of Mathematicians in Paris, where he listed 23 problems that in his opinion were essential for the progress of mathematics. From here comes his dictum "Here is the problem, seek the solution". The inscription on his tombstone, "We must know, we will know" comes from a speech in Königsberg in 1938. His address can be found in the Bulletin of the AMS, vol 37, number 4 (2000).

His work on integral equations and calculus of variations led to functional analysis and the concept of Hilbert space. He also solved Waring's problem, a well-known problem in number theory. His work in logic resulted in the development of *proof systems* and lead to the incompleteness theorems of K. Gödel.

1.11.A Formal Proofs

It will be convenient here to consider as the basic symbols in the language of propositional logic the following:

$$\neg, \Rightarrow,), (, p_1, p_2, p_3, \ldots$$

and view $(A \wedge B), (A \vee B), (A \Leftrightarrow B)$ as abbreviations:

$$\begin{aligned}(A \wedge B) &: \neg(A \Rightarrow \neg B)\\ (A \vee B) &: (\neg A \Rightarrow B)\\ (A \Leftrightarrow B) &: \neg((A \Rightarrow B) \Rightarrow \neg(B \Rightarrow A)).\end{aligned}$$

A *(logical) axiom* is any formula of one of the following forms:

(i) $A \Rightarrow (B \Rightarrow A)$

(ii) $(A \Rightarrow (B \Rightarrow C)) \Rightarrow ((A \Rightarrow B) \Rightarrow (A \Rightarrow C))$

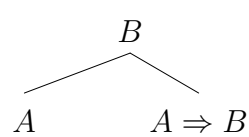
(iii) $((\neg B \Rightarrow \neg A) \Rightarrow ((\neg B \Rightarrow A) \Rightarrow B))$

(for arbitrary A, B, C). Notice that each one of these is a tautology.

We also have a *rule of inference* called *modus ponens* or MP: "From A and $A \Rightarrow B$, derive B." In formal calculi, a rule such as this for deriving one formula from others given, is sometimes written as follows:

$$\frac{A,\ A \Rightarrow B}{B}.$$

Modus ponens is short for latin "modus ponendo ponens" which means "proposing method". We can also draw this rule as a proof tree:



Notice that $A, A \Rightarrow B$ logically imply B.

Definition 1.11.1 Let S be any set (finite or infinite) of formulas. A *formal proof* from S is a finite sequence $A_1, A_2, \ldots, A_n$ of formulas such that each A_i is either a logical axiom, belongs to S or comes by applying modus ponens to some A_j, A_k with $j, k < i$ (i.e., A_k is the formula $A_j \Rightarrow A_n$). We call this sequence a *formal proof of* A_n *from* S.

Definition 1.11.2 If there is a formal proof of a formula A from S we say that A is a *formal theorem* of S and write

$$S \vdash A.$$

If $S = \emptyset$, we just write

$$\vdash A$$

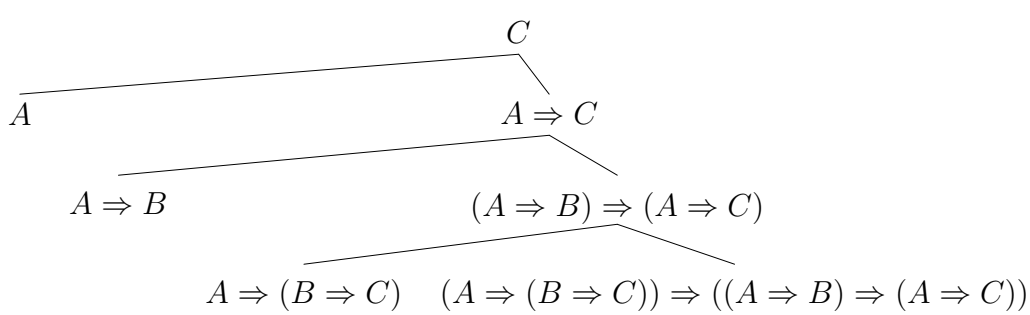
and call A a *formal theorem*.

Notice that $S \vdash A$ and $S \subseteq S'$ imply that $S' \vdash A$. Also notice that if $S' \vdash A$ and $S \vdash B$ for all $B \in S'$, then $S \vdash A$, since the formal proofs can simply be concatenated. Finally, $S \vdash A$ implies that there is *finite* $S_0 \subseteq S$ with $S_0 \vdash A$, since a formal proof can only be finitely long and hence can only use finitely many formulas of S.

Example 1.11.3 $S = \{(A \Rightarrow B),\ A \Rightarrow (B \Rightarrow C), A\}$. Here is a formal proof of C from S:

1.	$A \Rightarrow (B \Rightarrow C)$	(in S)
2.	$((A \Rightarrow (B \Rightarrow C)) \Rightarrow ((A \Rightarrow B) \Rightarrow (A \Rightarrow C)))$	(axiom (ii))
3.	$(A \Rightarrow B) \Rightarrow (A \Rightarrow C)$	(MP from 1, 2)
4.	$A \Rightarrow B$	(in S)
5.	$A \Rightarrow C$	(MP from 3, 4)
6.	A	(in S)
7.	C	(MP from 5, 6)

We can also write this as a *proof tree*:



Example 1.11.4 The following formal proof shows that $\vdash A \Rightarrow A$:

1.	$(A \Rightarrow (\underbrace{(A \Rightarrow A)}_{B} \Rightarrow \underbrace{A}_{C})) \Rightarrow ((A \Rightarrow \underbrace{(A \Rightarrow A)}_{B}) \Rightarrow (A \Rightarrow \underbrace{A}_{C}))$	(Ax. (ii))
2.	$A \Rightarrow (\underbrace{(A \Rightarrow A)}_{B} \Rightarrow A)$	(Ax. (i))
3.	$((A \Rightarrow (A \Rightarrow A)) \Rightarrow (A \Rightarrow A))$	(MP 1, 2)
4.	$A \Rightarrow (A \Rightarrow A)$	(Ax. (i))
5.	$A \Rightarrow A$	(MP 3, 4).

The main result about this formal proof system is the following:

Theorem 1.11.5 *For any set of formulas S, and any formula A:*

$$S \models A \text{ iff } S \vdash A.$$

One direction of this theorem, i.e. *the soundedness of the proof system* ($S \vdash A$ implies that $S \models A$) is easy, and we prove it first.

1.11.B Soundness

Given a property $P(A)$ of formulas, suppose we want to prove that $P(A)$ holds for all formal theorems of S, i.e., we want to show that if $S \vdash A$, then $P(A)$ holds. We can do this by the following form of induction:

Basis.

(i) Show that $P(A)$ holds, when A is a logical axiom;

(ii) Show that $P(A)$ holds when A is in S.

Induction Step. Assuming that $P(A)$, $P(A \Rightarrow B)$ hold, and show that $P(B)$ holds.

We call this *induction on proofs of A from S*.

In our case $P(A)$ is the property

$$\text{“}S \models A\text{”}.$$

The basis of the induction is clear: $S \models A$ is certainly correct if A is in S or else A is a logical axiom (since it is then a tautology). The induction step is equally simple: If $S \models A$ and $S \models A \Rightarrow B$, clearly $S \models B$.

1.11.C Completeness

To prove the hard direction of the theorem, i.e. the *completeness of the proof system*, it will be convenient to develop first some basic properties of this system. These are useful independently of this and can help, for example, in constructing formal proofs. They are indeed formal analogs of some commonly used proof techniques. They are theorems about formal proofs and theorems, and so often called *metatheorems*.

Proposition 1.11.6 (The Deduction Theorem) *If $S \cup \{A\} \vdash B$, then $S \vdash A \Rightarrow B$.*

Remark. It is easy to see that if $S \vdash A \Rightarrow B$, then $S \cup \{A\} \vdash B$.

Proof. We will show this by induction on proofs of B from $S \cup \{A\}$.

Basis.

(i) B is a logical axiom. Then clearly $S \vdash B$. But $B \Rightarrow (A \Rightarrow B)$ is a logical axiom, thus $S \vdash B \Rightarrow (A \Rightarrow B)$, so, by modus ponens, $S \vdash A \Rightarrow B$.

(ii) B is in $S \cup \{A\}$. There are two subcases:

  (a) B is in S. Then again $S \vdash B$ and the proof is completed as in (i).

  (b) $B = A$. Then $\vdash A \Rightarrow A$ by example 1.11.4, so $S \vdash A \Rightarrow A$, which is the same as $S \vdash A \Rightarrow B$

Induction step. We assume that we have shown that $S \vdash A \Rightarrow (B \Rightarrow C)$ and $S \vdash A \Rightarrow B$, and want to show that $S \vdash A \Rightarrow C$. Now $((A \Rightarrow (B \Rightarrow C)) \Rightarrow ((A \Rightarrow B) \Rightarrow (A \Rightarrow C))$ is a logical axiom, so

$$S \vdash ((A \Rightarrow (B \Rightarrow C)) \Rightarrow ((A \Rightarrow B) \Rightarrow (A \Rightarrow C)),$$

and then, by MP applied twice, we have

$$S \vdash A \Rightarrow C.$$

$\dashv$

Definition 1.11.7 We call a set of formulas S *formally inconsistent* if for some formula A,

$$S \vdash A \text{ and } S \vdash \neg A.$$

Otherwise, S is *formally consistent*.

Proposition 1.11.8 (Proof by Contradiction) *If $S \cup \{A\}$ is formally inconsistent, then $S \vdash \neg A$.*

Proof. Left for Assignment #5. $\dashv$

Proposition 1.11.9 (Proof by Contrapositive) *If $S \cup \{A\} \vdash \neg B$, then $S \cup \{B\} \vdash \neg A$.*

Proof. Left for Assignment #5. $\dashv$

We embark now on the proof of the Completeness of our proof system. We claim that it is enough to show the following:

If a set of formulas S is formally consistent, then it is satisfiable. $(*)$

To see this, assume $(*)$ has been proved. If $S \models A$, then $S \cup \{\neg A\}$ is unsatisfiable, so by $(*)$, applied to $S \cup \{\neg A\}$, $S \cup \{\neg A\}$ is formally inconsistent. Thus, by proof by contradiction, $S \vdash \neg\neg A$, so by Assignment #5, Problem 1, $S \vdash A$, which is what we wanted. So it remains to prove $(*)$.

The idea of the proof is as follows: Let us call a set of formulas $\bar{S}$ *complete* if for any formula A,

$$A \in \bar{S} \text{ or } \neg A \in \bar{S}.$$

We will first prove that if S is a given formally consistent set of formulas, then we can find a formally consistent set of formulas $\bar{S} \supseteq S$ which is also complete. Then for any propositional variable p_i we must have that *exactly* one of the following holds

$$p_i \in \bar{S} \text{ or } \neg p_i \in \bar{S}.$$

Thus we can define a valuation ν by letting

$$\nu(p_i) = \begin{cases} 1 & \text{if } p_i \in \bar{S} \\ 0 & \text{if } \neg p_i \in \bar{S} \end{cases}.$$

It will then turn out that ν is a valuation that satisfies $\bar{S}$ and so S, i.e., S is satisfiable. Let's implement this plan.

Lemma 1.11.10 *Let S be a formally consistent set of formulas. Then we can find a formally consistent and complete set of formulas $\bar{S}$ such that $S \subseteq \bar{S}$.*

Proof. First we can enumerate in a sequence $A_1, A_2, \ldots$ all formulas. Then we can define recursively a sequence

$$S_0 = S,\ S_1, S_2, \ldots$$

of formulas such that

$$S_0 \subseteq S_1 \subseteq S_2 \subseteq \ldots$$

as follows:

$$S_0 = S,$$
$$S_{n+1} = \begin{cases} S_n \cup \{A_{n+1}\} & \text{if } S \cup \{A_{n+1}\} \text{ is formally consistent} \\ S_n \cup \{\neg A_{n+1}\} & \text{otherwise.} \end{cases}$$

We claim then that each S_n is formally consistent. We do this easily by induction using the following sublemma.

Sublemma 1.11.11 *If T is a formally consistent set of formulas and A is any formula, then at least one of $T \cup \{A\},\ T \cup \{\neg A\}$ is still formally consistent.*

Proof. Otherwise both $T \cup \{A\},\ T \cup \{\neg A\}$ are formally inconsistent. So $T \vdash A$ by proof of contradiction and Assignment #5, Problem 1, and since clearly $T \vdash B$ for any $B \in T$, it follows that $T \vdash B$ for any $B \in T \cup \{A\}$. Since $T \cup \{A\}$ is formally inconsistent, this shows that T is formally inconsistent, a contradiction. $\dashv$

Now let $\bar{S} = \bigcup_{n=1}^{\infty} S_n$. We claim this works. First $\bar{S}$ is complete, since if A is any given formula, then $A = A_n$ for some n and so by construction either $A \in S_n$ or $\neg A \in S_n$, i.e., $A \in \bar{S}$ or $\neg A \in \bar{S}$. Finally $\bar{S}$ is still formally consistent, since otherwise $\bar{S} \vdash A,\ \bar{S} \vdash \neg A$ for some formula A. Then by definition of a formal proof, there is a finite subset $S^* \subseteq \bar{S}$ such that $S^* \vdash A,\ S^* \vdash \neg A$. But then for some large enough n we have $S^* \subseteq S_n$, so $S_n \vdash A,\ S_n \vdash \neg A$, and so S_n is formally inconsistent, a contradiction. $\dashv$

Lemma 1.11.12 *Let $\bar{S}$ be formally consistent and complete. Define the valuation ν by $\nu(p_i) = 1$ if $p_i \in \bar{S}$, $\nu(p_i) = 0$ if $\neg p_i \in \bar{S}$. Then for any formula A,*

$$\nu(A) = 1 \text{ iff } A \in \bar{S}.$$

Proof. Left for Assignment #5. ⊣

The combination of the two preceding lemmas finishes the proof of the completeness of the proof system. ⊣

We have already noted that if $S \vdash A$ then there is finite $S_0 \subseteq A$ with $S_0 \vdash A$. Therefore, as a corollary of the completeness of the proof system, we obtain a new proof of the Compactness Theorem for propositional logic.

1.11.D General Completeness

The Completeness Theorem for propositional logic is proved above only for formulas built using $\{\Rightarrow, \neg\}$. However, the result is valid for general formulas, provided a few logical axioms are added to the list. The proof given above can be easily adapted, and it is a useful exercise to see what additional steps are necessary. The axioms are designed to show that the formulas using the additional connectives are equivalent to formulas using only $\{\Rightarrow, \neg\}$. Here is a possible list.

(i) $(A \Rightarrow (B \Rightarrow A))$

(ii) $((A \Rightarrow (B \Rightarrow C)) \Longrightarrow ((A \Rightarrow B) \Rightarrow (A \Rightarrow C)))$

(iii) $((\neg A \Rightarrow \neg B) \Rightarrow (B \Rightarrow A))$

(iv) $((A \wedge B) \Rightarrow A)$

(v) $((A \wedge B) \Rightarrow B)$

(vi) $((A \Rightarrow B) \Longrightarrow ((A \Rightarrow C) \Rightarrow (A \Rightarrow (B \wedge C))))$

(vii) $(A \Rightarrow (A \vee B))$

(viii) $(B \Rightarrow (A \vee B))$

(ix) $((A \Rightarrow C) \Longrightarrow ((B \Rightarrow C) \Rightarrow ((A \vee B) \Rightarrow C)))$

(x) $((A \Leftrightarrow B) \Rightarrow (A \Rightarrow B))$

(xi) $((A \Leftrightarrow B) \Rightarrow (B \Rightarrow A))$

(xii) $((A \Rightarrow (B \Rightarrow C)) \Longrightarrow ((A \Rightarrow (C \Rightarrow B)) \Rightarrow (A \Rightarrow (B \Leftrightarrow C))))$

Chapter 2

First-Order Logic

2.1 Structures

2.1.A Relations and Functions

Definition 2.1.1 Let A be a *nonempty* set and let $n = 1, 2, \ldots$. An *n-ary relation* or *predicate* on A is a subset $R \subseteq A^n$, where $A^n = \{(a_1, \ldots, a_n) : a_1, \ldots, a_n \in A\}$ is the set of ordered n-tuples of elements of A. We call n the *arity* of R.

Examples 2.1.2

(i) $A = \mathbb{N}$ $(= \{0, 1, 2, \ldots\})$

$$R = \{n \in \mathbb{N} : n \text{ is even}\} \subseteq \mathbb{N} \qquad \text{(unary (1-ary))}$$
$$S = \{(m, n) \in \mathbb{N}^2 : m < n\} \subseteq \mathbb{N}^2 \qquad \text{(binary (2-ary))}$$
$$T = \{(m, n, p) \in \mathbb{N}^3 : m \equiv n \mod (p)\} \subseteq \mathbb{N}^3 \qquad \text{(ternary (3-ary))}$$

(ii) $A = \mathbb{R}$ (= the set of reals)

$$Q = \{x \in \mathbb{R} : x \text{ is rational}\} \subseteq \mathbb{R} \qquad \text{(unary)}$$
$$P = \{(x, y) \in \mathbb{R}^2 : y = x^2\} \subseteq \mathbb{R}^2 \qquad \text{(binary)}$$
$$U = \left\{ \begin{array}{l} (a_1, \ldots, a_n) \in \mathbb{R}^n : \\ a_1 x^{n-1} + a_2 x^{n-2} + \cdots + a_{n-1} x + a_n \\ \text{has } n-1 \text{ real solutions.} \end{array} \right\} \qquad (n\text{-ary})$$
$$B = \{(x, y, z) : y \text{ is between } x \text{ and } z\} \qquad \text{(ternary)}$$

(iii) $A = V$, where $G = \langle V, E\rangle$ is a graph with set of vertices V and set of edges E.

$$E = \{(x, y) \in V^2 : \ x \text{ and } y \text{ are connected by an edge}\} \qquad \text{(binary)}$$

$$F = \left\{(x, y) \in V^2 : \begin{array}{l} x \text{ and } y \text{ are connected by a path} \\ x_0 = x, x_1, \ldots, x_{n-1}, x_n = y. \end{array}\right\} \qquad \text{(binary)}$$

Remark. If $R \subseteq A^n$ is an n-ary relation, we also view R as expressing a property of n-tuples $(a_1, \ldots, a_n) \in A^n$ and we often write

$$R(a_1, \ldots, a_n),$$

instead of

$$(a_1, \ldots, a_n) \in R.$$

Thus $R(a_1, \ldots, a_n)$ means that $(a_1, \ldots, a_n)$ has property R or satisfies R. Note that a 1-ary relation is a property of single elements of A, or alternately just the subset of A containing all elements with that property.

Example 2.1.3 For R, S, T as in examples 2.1.2 (i) above:

(i) $R(n)$ means "n is even"

(ii) $S(m, n)$ means "$m < n$"

(iii) $T(m, n, p)$ means "$m \equiv n \mod (p)$

Remark. Example 2.1.3 (ii) above says that S essentially *is* the familiar "less than" relation, so we can write "$< (a, b)$" or even just "$a < b$" instead of $S(a, b)$. The same holds true for other familiar binary relations.

Definition 2.1.4 Let again A be any nonempty set. A *n-ary function* or *operation* on A is a map

$$f : A^n \to A,$$

in other words, a rule for assigning to each element $(x_1, \ldots, x_n)$ of A^n (where each $x_i \in A$) an element $f(x_1, \ldots, x_n)$ of A, called the *value* of the function at $(x_1, \ldots, x_n)$. We call n the *arity* of f.

Examples 2.1.5

(i) $A = \mathbb{N}$

$$\begin{aligned} f(n) &= n^2 && \text{(unary)} \\ g(m,n) &= m+n, && \text{(binary)} \\ a(m,n,p) &= m+n \mod (p), && \text{(ternary)} \\ q(m) &= \text{number of distinct primes dividing } m && \text{(unary)} \end{aligned}$$

(ii) $A = \mathbb{R}$

$$f(a_1,\ldots,a_n) = a_1^2 + a_2^2 + \cdots + a_n^2 \qquad (n\text{-ary})$$

(iii) $A =$ all strings of symbols in an alphabet S (which can be an arbitrary set).

$$\begin{aligned} f(s,t) &= s\hat{\ }t \text{ (the concatenation of } s \text{ and } t) && \text{(binary)} \\ g(s,t,u) &= \text{the result of substituting the leftmost} \\ &\quad \text{occurrence of } s \text{ in } u \text{ by } t\text{, if } s \text{ occurs in} \\ &\quad u\text{; otherwise, } u. && \text{(ternary)} \end{aligned}$$

By convention we allow also the case $n = 0$. Since $A^0 = \{\emptyset\}$ (that is, the only 0-tuple is the empty one), a 0-ary function f is simply an element of A, i.e. a *constant* in A. For example, π is a 0-ary function in $\mathbb{R}$.

Remark. On any set A we always have a canonical binary relation, namely "=" (equality among elements of A).

Remark. We have used the same notation $R(a_1,\ldots,a_n)$ to denote that the relation R is satisfied by the n-tuple $(a_1,\ldots,a_n)$ and $f(a_1,\ldots,a_n)$ to denote the value of the function f at the arguments $(a_1,\ldots,a_n)$. The context will tell us with which case we are dealing, so that this shouldn't cause any confusion.

2.1.B Structures

Definition 2.1.6 A *structure* consists of a nonempty set, called the *universe* of the structure, together with certain relations and functions on this set.

We write structures as

$$\mathcal{A} = \langle A; f, g, h, \ldots; R, S, T, \ldots \rangle,$$

where A is the universe, $f, g, h, \ldots$ are the functions of the structure, and $R, S, T, \ldots$ are the relations of the structure. Note that A can be finite or infinite (but not empty), and there can be any number (finite, infinite, or even zero) of functions, each with any arity $n \geq 0$, and any number (finite, infinite, or zero) of relations, each with any arity $n \geq 1$.

A structure codifies a context, of mathematical or other objects, which we study. The universe of the structure encompasses all the objects under consideration and the functions and relations of the structure are the basic or primitive operations and relations between these objects that we want to study.

Definition 2.1.7 The *signature* of $\mathcal{A}$ is

$$\langle \text{arity}(f), \text{arity}(g), \text{arity}(h), \ldots ; \text{arity}(R), \text{arity}(S), \text{arity}(T), \ldots \rangle.$$

Examples 2.1.8

(i) The *structure of (natural number) arithmetic*:

$$\mathcal{N} = \langle \mathbb{N}; 0, S, +, \cdot; < \rangle.$$

It has signature

$$\langle 0, 1, 2, 2; 2 \rangle.$$

(Here $S(n) = n + 1$ is the "successor" function.)

(ii) The *structure of the reals*:

$$\mathcal{R} = \langle \mathbb{R}; 0, 1, +, \cdot; \rangle.$$

It has signature

$$\langle 0, 0, 2, 2; \rangle.$$

This structure represents the basic algebra of real numbers. If we want to study something different, such as elementary trigonometry, a more appropriate structure might be $\langle \mathbb{R}; 0, 1, +, \cdot, \sin, \cos, \ldots \rangle$.

(iii) Let X be an arbitrary set (finite or infinite). Then

$$\langle P(X); \emptyset, \cap, \cup; \subseteq \rangle,$$

where $P(X)$ is the collection of all subsets of X, is a structure with signature

$$\langle 0, 2, 2; 2 \rangle.$$

(iv) Many algebraic structures are simply structures in the above sense, which satisfy certain properties (axioms). For example, a *group* is a structure of the form

$$\langle G; e, \cdot; \rangle$$

of signature $\langle 0, 2; \rangle$ satisfying the group laws.

(v) A *graph* is a structure of the form

$$\langle V; ; E \rangle$$

of signature $\langle ; 2 \rangle$ satisfying the usual conditions.

(vi) Let S be the set of records of employees in some company (containing, e.g., social security numbers, dates of birth, salary, etc.), let $N = \{0, 1, \ldots, n\}$ for n a large enough integer, and let

$$\mathcal{A} = \langle S \cup N; f, g, h; P, Q, R \rangle,$$

where

$$f(x) = \begin{cases} \text{the age of the employee with record } x & \text{if } x \in S \\ 0 & \text{otherwise.} \end{cases}$$

$$g(x) = \begin{cases} \text{the SSN of the employee with record } x & \text{if } x \in S \\ 0 & \text{otherwise.} \end{cases}$$

$$h(x) = \begin{cases} \text{the salary of the employee with record } x & \text{if } x \in S \\ 0 & \text{otherwise.} \end{cases}$$

$P(x)$ iff $x \in S$

$Q(x)$ iff $x \in N$

$R(x, y)$ iff x is a supervisor of y.

Then $\mathcal{A}$ is a structure of arity $\langle 1, 1, 1; 1, 1, 2 \rangle$. This example shows how we can include both the objects we want to study (here, the records) and some basic mathematical tools to use in reasoning about them (here, some natural numbers) in a single structure.

2.1.C Introduction to First-Order Languages

We now want to describe a formal language in which we can express statements about structures. We will give a series of examples showing how to develop such a language.

Example 2.1.9 Consider the statement:

"The square of an odd number is odd."

This is a statement about the structure of arithmetic,

$$\mathcal{N} = \langle \mathbb{N}; 0, S, +, \cdot; < \rangle.$$

We now introduce the following notation:

$x, y, z, \ldots$	variables representing arbitrary natural numbers;
$\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow$	the usual propositional connectives;
$(,)$	parentheses;
$\exists$ (there exists),	the *existential* and
$\forall$ (for all)	*universal* quantifier;
$=$	symbol for equality;
$0, S, +, \cdot, <$	symbols for the operations and relations of $\mathcal{N}$.

In order to write this statement in a formal language using these symbols, we must first express it in a more precise form:

"For all natural numbers x, if x is odd, then $x \cdot x$ is odd."

We must further refine this by replacing "x is odd" with

"There exists a natural number y such that $x = 2 \cdot y + 1$."

Finally, since 1 and 2 are not constants in the structure of arithmetic, we must replace them with $S(0)$ and $S(S(0))$, respectively. We can now express the above statement in a formal language as:

$$\forall x[\exists y(x = S(S(0)) \cdot y + S(0)) \Rightarrow \exists z(x \cdot x = S(S(0)) \cdot z + S(0))]$$

We often use abbreviations, however, such as

$$S(0) = 1,\ S(S(0)) = 2,\ x \cdot x = x^2,\ u \cdot v = uv,$$

Using these abbreviations, our statement becomes more readable:

$$\forall x[\exists y(x = 2y + 1) \Rightarrow \exists z(x^2 = 2z + 1)].$$

Example 2.1.10 "Every non-empty subset of $\mathbb{N}$ has a least element".

The appropriate structure here is:

$$\langle \mathbb{N} \cup P(\mathbb{N});\ N, S, \in, < \rangle,$$

where

$$P(\mathbb{N}) = \{A : A \subseteq \mathbb{N}\} = \text{ the power set of } \mathbb{N}$$

$$\begin{aligned}
&N(a) \text{ iff } a \in \mathbb{N} \\
&S(a) \text{ iff } a \in P(\mathbb{N}) \text{ (i.e., } a \subseteq \mathbb{N}) \\
&a \in b \text{ iff } a \in \mathbb{N},\ b \in P(\mathbb{N}),\ \text{and } a \text{ is an element of } b \\
&a < b \text{ iff } a, b \in \mathbb{N} \text{ and } a \text{ is smaller than } b
\end{aligned}$$

Using the same logical notation as before and symbols $N, S, \in, <$ for the relations of this structure we can express the previous statement as

$$\forall x[(S(x) \wedge \exists y(y \in x)) \Rightarrow \exists z(z \in x \wedge \forall w(w \in x \Rightarrow z = w \vee z < w))]$$

2.2 Syntax of First-Order Logic

We will now give a formal definition of the languages which were discussed by example in section 2.1.C.

2.2.A Symbols

Definition 2.2.1 A *first-order language* has the following set of symbols (alphabet):

(i) *Logical symbols*:

$x_1, x_2, x_3, \ldots$	(variables)
$\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow$	(propositional connectives)
$\exists, \forall$	(quantifiers)
$(,)$	(parentheses)
$=$	(equality symbol)

(ii) *Non-logical symbols*:

(a) For each $n \geq 0$ a set $\mathcal{F}_n$ (which may be empty) of symbols called *n-ary function symbols* (or function symbols of arity n). For $n = 0$, the symbols in $\mathcal{F}_0$ are called *constant* symbols.

(b) For each $n \geq 1$, a set $\mathcal{R}_n$ (which may be empty) of symbols called *n-ary relation symbols* (or relation symbols of arity n).

Remark. All these sets of symbols are of course assumed to be pairwise disjoint. The equality symbol $=$ is also a binary relation symbol, but plays a special role.

Remark. A first-order language is completely determined by its set $L = \bigcup_n \mathcal{F}_n \cup \bigcup_n \mathcal{R}_n$ of non-logical symbols (since the set of logical symbols is the same for all first-order languages).

Examples 2.2.2

(i) An example of a first-order language is $L = \{c, f, g; R\}$, where c is a constant symbol, f a unary function symbol, g a binary function symbol, and R a binary relation symbol.

(ii) A more meaningful example is the *language of arithmetic*:

$$L_{ar} = \{0, S, +, \cdot; <\}$$

where 0 is a constant symbol, S a unary function symbol, $+$ and $\cdot$ binary function symbols, and $<$ a binary relation symbol. This is the language that was used in example 2.1.9.

2.2.B Terms

We will now start defining the "grammatically correct" statements of a first-order language. We begin by defining strings called *terms*. Each term is intended to represent a single element of the structure. Terms are defined recursively as follows.

Definition 2.2.3 (Terms)

(i) Every variable symbol is a term.

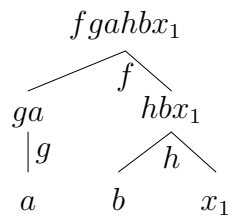
(ii) If $t_1, \ldots, t_n$ are terms and f is an n-ary function symbol, then $ft_1 \ldots t_n$ is a term. Note that if $n = 0$, i.e. f is a constant symbol, then f is a term all by itself.

Thus we use "Polish notation" to represent terms. (The weight of each n-ary function symbol is $1-n$ and that of constant symbols and variables is 1.)

Example 2.2.4 Suppose L contains the constant symbols a, b, the unary function symbol g and the binary function symbols f, h. Then the following is a term in this language

$$fgahbx_1.$$

This can be also represented by a parse tree as follows:



As in section 1.2.C, we have unique readability, in the sense that every term can be uniquely written as: x_i, a (constant symbol) or $ft_1 \dots t_n$, for uniquely determined $t_1, \dots, t_n$ terms and f an n-ary function symbol ($n \geq 1$).

To facilitate reading, we will add parentheses and adopt the following informal notation:

$$f(t_1, \dots, t_n) \text{ instead of } ft_1 \dots t_n$$

So we would write the term in the example above as

$$f(g(a), h(b, x_1)).$$

Also, for familiar binary function symbols f (like $+, \cdot$, etc) we will usually write

$$(tfs) \text{ instead of } fts,$$

and when there is no confusion we will even drop the parentheses and write tfs.

Finally, we often use $x, y, z, \dots$ for variables instead of $x_1, x_2, \dots$.

Example 2.2.5 In the language of arithmetic, the following are terms (using our informal notation, that is):

$$
\begin{array}{ll}
(x+y)\cdot z & \text{(instead of } \cdot + xyz) \\
(x\cdot x)+(y\cdot y) & \\
((x\cdot x)\cdot x + S(0)\cdot x) + S(S(0)) &
\end{array}
$$

In fact, all terms in this language are polynomials in several variables with coefficients in $\mathbb{N}$. We can view terms in arbitrary languages as "generalized polynomials."

Notation. If the variables occurring in a term t are among $x_1, \ldots, x_n$, we indicate this by writing $t(x_1, \ldots, x_n)$. Note that this does not mean that all of $x_1, \ldots, x_n$ occur in t. If $u_1, \ldots, u_n$ are terms, then

$$t[x_1/u_1, \ldots, x_n/u_n]$$

denotes the result of substituting each variable x_i in t by u_i. By induction on the construction of t, this is also a term.

Example 2.2.6 If

$$t(x,y) \text{ is } f(g(x), h(a,y)),$$

then

$$t[x/g(a), y/h(a,x))] \text{ is } f(g(g(a)), h(a, h(a,x)).$$

2.2.C Well-Formed Formulas

We can now define the statements in first-order languages, which we again call *(well-formed) formulas (wffs)*. We start with the simplest possible formulas.

Definition 2.2.7 An *atomic formula* is any string of the form

$$Rt_1 \ldots t_n$$

with R an n-ary relation symbol and $t_1, \ldots, t_n$ terms.

Example 2.2.8 Let $L = \{a, b, f, g, R\}$, a, b constant symbols, f a unary function symbol, g a binary function symbol, R a ternary relation symbol. The following are atomic formulas in this language:

$$Rfagxay, = gxyfa$$

Example 2.2.9 Let $L = \{<\}$, $<$ a binary relation symbol. Then the following is an atomic formula:

$$< xy.$$

Again we have unique readability: every atomic formula is of the form $Rt_1 \dots t_n$ for uniquely determined $R, t_1, \dots, t_n$.

And as we did with terms, we adopt various convenient informal notations. We write

$$R(t_1, \dots, t_n) \text{ instead of } Rt_1 \dots t_n.$$

For familiar binary relation symbols R (like $=, <$), we write

$$(tRs) \text{ instead of } Rts,$$

and when there is no chance of confusion we often omit the parentheses as well, writing tRs. So in example 2.2.8 we would write $R(f(a), g(x, a), y)$ and $(g(x, y) = f(a))$, and in example 2.2.9 we would write $(x < y)$ or even $x < y$.

Finally, we are ready to define:

Definition 2.2.10 (Well-Formed Formulas)

(i) Every atomic formula is a wff.

(ii) If A, B are wff, so are $\neg A, (A \wedge B), (A \vee B), (A \Rightarrow B), (A \Leftrightarrow B)$.

(iii) If A is a wff and x_i a variable, then $\exists x_i A,\ \forall x_i A$ are wff.

Examples 2.2.11

(i) Let $L = \{<\}$, $<$ a binary relation symbol. Then

$$\forall x \exists y (x < y)$$

is a wff, as is justified by the following parsing sequence:

$$(x < y),\ \exists y(x < y),\ \forall x \exists y(x < y).$$

(ii) In the same language,

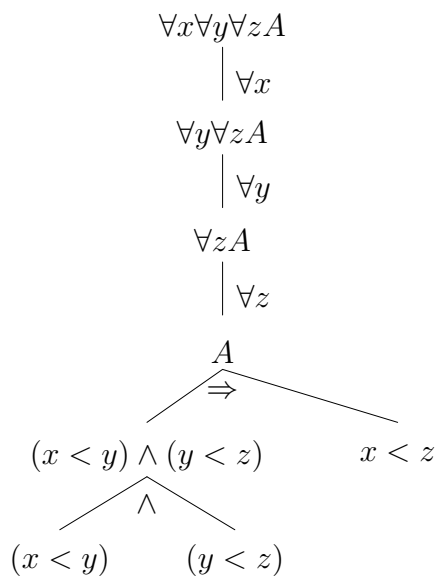
$$\forall x \forall y \forall z((x < y) \wedge (y < z) \Rightarrow (x < z))$$

is a wff, as justified by the following parsing sequence:

$$(x < y),\ (y < z),\ (x < z),\ (x < y) \wedge (y < z),$$

$$\underbrace{((x < y) \wedge (y < z) \Rightarrow (x < z))}_{A},\ \forall z A,\ \forall y \forall z A,\ \forall x \forall y \forall z A.$$

We can also represent any wff by a parse tree. For (ii) this would be



(iii) $L = \{R, Q, S\}$, with R ternary and Q, S unary relation symbols. The following is a formula:

$$\exists x(\forall y R(x, y, z) \Rightarrow (\neg Q(x) \vee R(y))).$$

Yet again we have unique readability: every formula is in exactly one of the forms:

(i) atomic;

(ii) $\neg A,\ (A \wedge B),\ (A \vee B),\ (A \Rightarrow B),\ (A \Leftrightarrow B)$ for uniquely determined A, B;

(iii) $\exists x_i A,\ \forall x_i A$ for uniquely determined x_i, A.

Definition 2.2.12 A *subformula* of a formula A is any formula occurring in the parse tree of A.

2.2.D Scope and Free Variables

Proposition 2.2.13 *Let A be a wff, Q be either $\exists$ or $\forall$, x a variable. For any occurrence of the string Qx in A we have $A = SQxBT$, where S, T are strings and B is a uniquely determined wff called the* scope *of this occurrence of Qx.*

Example 2.2.14 $P(x,y) \Rightarrow \underbrace{\forall x\,[\exists y\,\underbrace{R(x,y)}_{\text{scope}} \Rightarrow \forall x\,\underbrace{Q(x,y)}_{\text{scope}}]}_{\textit{scope}}$

Example 2.2.15 $\underbrace{\exists y\,\underbrace{\forall x\,[\underbrace{\exists y\,R(x,y)}_{\text{scope}} \Rightarrow Q(x,y)]}_{\text{scope}}}_{\text{scope}}$

Proof of proposition 2.2.13. It is easy to prove by induction on the construction of A that for any occurrence of Qx in A we have $A = SQxBT$ for some wff B and some strings S, T. Uniqueness follows from the fact that a proper nonempty initial segment of a wff is not a wff (see Assignment #6).$\dashv$

Definition 2.2.16 An occurrence of a variable x in a formula A is *bound* if it occurs in a substring of the form QxB, where B is the scope of x. Otherwise this occurrence is called *free*.

Example 2.2.17 In example 2.2.14 the first occurrences of x, y and the last occurrence of y are free. All the other occurrences are bound. In example 2.2.15, all occurrences are bound.

Definition 2.2.18 A variable x is *free* in a wff A if it has *at least one* free occurrence. It is *bound* if it has at least one bound occurence.

Note that a variable can be both free and bound. This is in contrast to a single *occurrence* of a variable, which is either one or the other.

Example 2.2.19 In example 2.2.14, x, y are both free and bound. In 2.2.15 x, y are bound but not free. In $R(x,y)$ they are free but not bound. In $\exists x R(x,y)$, x is bound and not free, and y is free but not bound.

Remark. This should be compared with a notation such as $\int(x^2+y)dx$. Here x is "bound" and y is "free."

Bound variables are sometimes called "dummy variables," since they can be replaced with a different variable (which does not appear elsewhere) without changing the meaning of the formula. That is, $\int(x^2+y)dx$ and $\int(t^2+y)dt$ are equivalent. And once we define equivalence of wffs, pairs such as $\forall x(x<y)$ and $\forall z(z<y)$ will be equivalent as well: see fact 2.3.17.

Notation. $A(x_1,\ldots,x_n)$ means that the *free variables* of A are *among* $x_1,\ldots,x_n$. So in example 2.2.14, we would write $A(x,y)$.

Definition 2.2.20 A *sentence* is a wff with no free variables.

Example 2.2.21 $\forall x\exists y(x<y)$ is a sentence, as is example 2.2.15, but $0<x$ is not, nor is example 2.2.14.

2.3 Semantics of First-Order Logic

Now we will connect the first-order languages to the structures we intended them to represent.

2.3.A Structures and Interpretations

Definition 2.3.1 Let $L=\bigcup_{n\geq 0}\mathcal{F}_n\cup\bigcup_{n\geq 1}\mathcal{R}_n$ be a first-order language. A *structure* $\mathcal{M}$ *for* L consists of:

(i) a nonempty set M (the universe of this structure)

(ii) an n-ary function $f^{\mathcal{M}}:M^n\to M$ for each $f\in\mathcal{F}_n$. (For $n=0$, i.e. $a\in\mathcal{F}_0$ is a constant symbol, $a^{\mathcal{M}}$ is simply an element of M.)

(iii) an m-ary relation $R^{\mathcal{M}}\subseteq M^m$ for each $R\in\mathcal{R}_m$.

We write this as

$$\mathcal{M}=\langle M,\{f^{\mathcal{M}}\}_{f\in\bigcup_{n\geq 0}\mathcal{F}_n},\{R^{\mathcal{M}}\}_{R\in\bigcup_{n\geq 1}\mathcal{R}_n}\rangle,$$

and call $f^{\mathcal{M}}$ the *interpretation* of f and $R^{\mathcal{M}}$ the *interpretation* of R in $\mathcal{M}$.

Example 2.3.2 If

$$L = \{ \quad \overbrace{a, f, g}^{\text{function symbols}} \quad ; \quad \overbrace{S, R}^{\text{relation symbols}} \quad \}$$

(a is a constant; f unary; g binary; S binary, R ternary), then a structure for L has the form

$$\mathcal{M} = \langle M; a^{\mathcal{M}}, f^{\mathcal{M}}, g^{\mathcal{M}}; S^{\mathcal{M}}, R^{\mathcal{M}} \rangle,$$

with $a^{\mathcal{M}} \in M,\ f^{\mathcal{M}} : M \to M, g^{\mathcal{M}} : M^2 \to M,\ S^{\mathcal{M}} \subseteq M^2,\ R^{\mathcal{M}} \subseteq M^3$.

Example 2.3.3 Let $L_{ar} = \{0, S, +, \cdot, <\}$ be the language of arithmetic. Then

$$\mathcal{N} = \langle N; 0, S, +, \cdot; < \rangle$$

is a structure for L_{ar} called the *standard structure* of this language. Another structure for this language is

$$\mathcal{A} = \langle A; 0^{\mathcal{A}}, S^{\mathcal{A}}, +^{\mathcal{A}}, \cdot^{\mathcal{A}}; <^{\mathcal{A}} \rangle,$$

where $A = \mathbb{R},\ 0^{\mathcal{A}} = \pi, S^{\mathcal{A}}(a) = e^a, \mathcal{A} +^{\mathcal{A}} b = a + b,\ a \cdot^{\mathcal{A}} b = a \cdot b,\ a <^{\mathcal{A}} b$ iff $b = \cos(a)$.

Given a structure $\mathcal{M} = \langle M, \dots \rangle$ for L, we can assign to each term $t(x_1, \dots, x_n)$ of L an n-ary function $t^{\mathcal{M}} : M^n \to M$ by recursion as follows:

(i) If t is a variable x_i, then $i \leq n$, and we let $t^{\mathcal{M}}(a_1, \dots, a_n) = a_i$.

(ii) If t is a constant c, then $t^{\mathcal{M}}(a_1, \dots, a_n) = c^{\mathcal{M}}$.

(iii) If $t = f(t_1, \dots, t_k)$, then

$$t^{\mathcal{M}}(a_1, \dots, a_n) = f^{\mathcal{M}}(t_1^{\mathcal{M}}(a_1, \dots, a_n), \dots, t_k^{\mathcal{M}}(a_1, \dots, a_n)).$$

Definition 2.3.4 We call $t^{\mathcal{M}}(a_1, \dots, a_n)$ the *evaluation* or *interpretation* of t at $a_1, \dots, a_n$ in $\mathcal{M}$.

Remark. If $t = t()$ has no variables, then $t^{\mathcal{M}}$ is simply an element of $\mathcal{M}$.

Remark. If $t = t(x_1, \dots, x_n)$, then also $t = t(x_1, \dots, x_n, x_{n+1}, \dots, x_k)$ for any $k > n$ and strictly speaking we have an infinite list of functions $t_n^{\mathcal{M}}$, one for each n such that all the variables of t are among $x_1, \dots, x_n$. Notice however that for $n < k$, $t_n^{\mathcal{M}}(a_1, \dots, a_n) = t_k^{\mathcal{M}}(a_1, \dots, a_n, a_{n+1}, \dots, a_k)$.

Example 2.3.5 Let $L_{ar} = \{0, S, +, \cdot, <\}$ and $\mathcal{N} = \langle \mathbb{N}; 0, S, +, \cdot; < \rangle$ its standard structure. Then if

$$t(x, y) = (x \cdot x + y) + S(S(0)),$$

we have

$$t^{\mathcal{N}} : \mathbb{N}^2 \to \mathbb{N},$$
$$t^{\mathcal{N}}(a, b) = a^2 + b + 2.$$

2.3.B Models and Validities

Now let L be a first-order language, A a wff in L, $\mathcal{M}$ a structure for L, and consider an assignment $x_i \mapsto a_i$ which associates with each variable x_i an element a_i of M (the universe of $\mathcal{M}$).

Definition 2.3.6 We say that A is *true* in $\mathcal{M}$ under this assignment, which we write

$$\mathcal{M}, a_1, a_2, \ldots \models A,$$

if the following recursive definition is satisfied:

(i) If A is atomic, say $A = R(t_1, \ldots, t_k)$, with $t_i = t_i(x_1, \ldots, x_n)$, then

$$\mathcal{M}, a_1, a_2, \ldots \models A \quad \text{iff} \quad R^{\mathcal{M}}(t_1^{\mathcal{M}}(a_1, \ldots, a_n), \ldots, t_k^{\mathcal{M}}(a_1, \ldots, a_n)).$$

Here we understand that if R is the equality symbol $=$, then $=^{\mathcal{M}}$ is the equality on M, so

$$\mathcal{M}, a_1, a_2, \ldots \models t_1 = t_2 \quad \text{iff} \quad t_1^{\mathcal{M}}(a_1, \ldots, a_n) = t_2^{\mathcal{M}}(a_1, \ldots, a_n).$$

(ii) $\mathcal{M}, a_1, a_2, \ldots \models \neg A$ iff it is not the case that $\mathcal{M}, a_1, a_2, \ldots \models A$ (i.e. $\mathcal{M}, a_1, a_2, \cdots \not\models A$)

$\mathcal{M}, a_1, a_2 \ldots \models A \wedge B$ iff $\mathcal{M}, a_1, a_2, \ldots \models A$ and $\mathcal{M}, a_1, a_2, \ldots \models B$

$\mathcal{M}, a_1, a_2 \ldots \models A \vee B$ iff $\mathcal{M}, a_1, a_2, \ldots \models A$ or $\mathcal{M}, a_1, a_2, \ldots \models B$

$\mathcal{M}, a_1, a_2 \ldots \models A \Rightarrow B$ iff $\mathcal{M}, a_1, a_2, \cdots \not\models A$ or $\mathcal{M}, a_1, a_2, \ldots \models B$

$\mathcal{M}, a_1, a_2 \ldots \models A \Leftrightarrow B$ iff either $\mathcal{M}, a_1, a_2 \ldots \models A, B$ or $\mathcal{M}, a_1, a_2 \cdots \not\models A, B$

(iii) $\mathcal{M}, a_1, a_2, \ldots, a_i, \ldots \models \exists x_i A$ iff for *some* $b_i \in M$,

$$\mathcal{M}, a_1, a_2, \ldots, a_{i-1}, b_i, a_{i+1}, \ldots \models A;$$

$\mathcal{M}, a_1, a_2, \ldots, a_i, \ldots \models \forall x_i A$ iff for *all* $b_i \in M$,

$$\mathcal{M}, a_1, a_2, \ldots, a_{i-1}, b_i, a_{i+1}, \ldots \models A.$$

It is easy to show by induction on the construction of A, that if

$$A(x_1, \ldots, x_n)$$

is given and a_i, b_i $(i = 1, 2, \ldots) \in M$ are such that $a_i = b_i$ for all $i \leq n$, then

$$\mathcal{M}, a_1, a_2, \ldots \models A \text{ iff } \mathcal{M}, b_1, b_2, \ldots \models A,$$

so we abbreviate

$$\mathcal{M}, a_1, a_2, \ldots \models A$$

by

$$\mathcal{M} \models A[a_1, \ldots, a_n]$$

or if we need to be more explicit

$$\mathcal{M} \models A[x_1 \mapsto a_1, x_2 \mapsto a_2, \ldots, x_n \mapsto a_n].$$

We read this as: $\mathcal{M}$ makes $A[a_1, \ldots, a_n]$ true or A is *satisfied* in $\mathcal{M}, a_1, \ldots, a_n$.

Definition 2.3.7 In particular, if $A = A()$ is a sentence (has no free variables), we simply write

$$\mathcal{M} \models A,$$

and say that $\mathcal{M}$ *satisfies* A or A *is true in* $\mathcal{M}$. We also say that $\mathcal{M}$ *is a model of* A.

Example 2.3.8 Let $L = \{<\}$ ($<$ a binary relation symbol) and $\mathcal{M} = \langle \mathbb{N}, < \rangle$, and consider the following wffs:

(i) $A : x < y$. Then we have

$$\begin{aligned} &\mathcal{M} \models x < y[2, 3] && \text{(i.e. } 2 < 3) \\ &\mathcal{M} \models \neg x < y[2, 1] && \text{(i.e. } 2 \not< 1). \end{aligned}$$

(ii) $B : \exists x \forall y (x < y \vee x = y)$, a sentence. Then

$$\mathcal{M} \models \exists x \forall y (x < y \vee x = y). \tag{2.1}$$

By definition this means that there is $a \in \mathbb{N}$ so that $\mathcal{M} \models \forall y (x < y \vee x = y)[x \mapsto a]$, and again, by definition, this means that for all $b \in \mathbb{N}$,

$$\mathcal{M} \models (x < y \vee x = y)[x \mapsto a,\ y \mapsto b].$$

so equation (2.1) means that for some $a \in \mathbb{N}$, and all $b \in \mathbb{N}$, $a < b$ or $a = b$, i.e. $a \leq b$. Clearly $a = 0$ works.

If on the other hand we take $\mathcal{S} = \langle \mathbb{Z}, < \rangle$, a different model for L, then

$$\mathcal{S} \models \neg \exists x \forall y (x < y \vee x = y).$$

Example 2.3.9 Let $L_{ar} = \{0, S, +, \cdot, <\}$ be the language of arithmetic and $\mathcal{N} = \langle \mathbb{N}, 0, S, +, \cdot, <\}$ its standard structure. Then the following are true:

$$\mathcal{N} \models \forall x \forall y (x + S(y) = S(x + y))$$
$$\mathcal{N} \models \forall x \exists y_1 \exists y_2 \exists y_3 \exists y_4 (x = y_1 \cdot y_1 + y_2 \cdot y_2 + y_2 \cdot y_3 + y_4 \cdot y_4)$$

(parentheses omitted by association to the left.) The latter is because every natural number is the sum of four squares—*Lagrange's Theorem.*

$$\mathcal{N} \models \neg \forall x \exists y \exists z (x = y \cdot y + z \cdot z)$$

(since, e.g., 7 is not the sum of two squares).

$$\mathcal{N} \not\models \exists y (x = y \cdot y) \text{ [5]}.$$

However, take now the following structure for L_{ar}:

$$\mathcal{A} = \langle \mathbb{R}, 0, x \mapsto e^x, +, \cdot, < \rangle.$$

Then

$$\mathcal{A} \not\models \forall x \forall y (x + S(y) = S(x + y))$$

(because $a + e^b = e^{a+b}$ is not always true), and

$$\mathcal{A} \models \forall x \forall y (S(x + y) = S(x) \cdot S(y))$$

(but $\mathcal{N} \not\models \forall x \forall y (S(x + y) = S(x) \cdot S(y))$).

Definition 2.3.10 If S is a set of sentences, then a structure $\mathcal{M}$ *satisfies* or *models* S, in symbols

$$\mathcal{M} \models S,$$

if $\mathcal{M} \models A$ for all $A \in S$.

Definition 2.3.11 If S is a set of sentences and A is a sentence, then S *logically implies* A, in symbols

$$S \models A$$

if every model $\mathcal{M}$ of S is also a model of A.

Definition 2.3.12 If $S = \emptyset$ we simply write

$$\models A,$$

instead of $\emptyset \models A$, and we say that

$$A \text{ is } \textit{(logically) valid}$$

This simply means that A is true in every structure $\mathcal{M}$ of the language L.

Definition 2.3.13 Two sentences A, B are *logically equivalent* if $\{A\} \models B$ and $\{B\} \models A$, or $\models A \Leftrightarrow B$. We then write $A \equiv B$.

Definition 2.3.14 A set of sentences S is *satisfiable* if it has a model, i.e. there is a structure $\mathcal{M}$ with $\mathcal{M} \models S$.

These notions can be also extended to arbitrary formulas (not necessarily sentences). A set of formulas S *logically* implies a formula A, in symbols $S \models A$, if for any structure $\mathcal{M}$ and each assignment $x_i \mapsto a_i \in M$ (= the universe of $\mathcal{M}$), if all the formulas in S are true, so is A. A formula A is *valid* iff $\models A$. If $A = A(x_1, \ldots, x_n)$, then it is easy to see that A is valid iff $\forall x_1 \forall x_2 \ldots \forall x_n A$ is valid. (We call $\forall x_1 \forall x_2 \ldots \forall x_n A$ the *universal closure* of A.) Again A, B are *equivalent*, in symbols $A \equiv B$, if $\models A \Leftrightarrow B$. Finally, a set of formulas S is *satisfiable* if there is a structure $\mathcal{M}$ *and an assignment* $x_i \mapsto a_i$ in $\mathcal{M}$ satisfying all the formulas in S.

Examples 2.3.15

(i) An *instance of a tautology* is any formula in first-order logic obtained from a tautology in propositional logic by substituting each propositional variable p_i by a formula A_i.

For instance,

$$A \vee \neg A$$
$$\neg(A \wedge B) \Leftrightarrow (\neg A \vee \neg B)$$

are instances of tautologies. Clearly, every instance of a tautology is valid.

(ii) A list of useful valid formulas is given in Appendix B.

(iii) In general, $\not\models \forall y \exists x A \Rightarrow \exists x \forall y A$.

For example, take $L = \{R\}$, R a binary relation symbol, and take A to be $R(x, y)$. Then we claim that

$$\not\models \forall y \exists x R(x, y) \Rightarrow \exists x \forall y R(x, y).$$

To see this we must find a structure $\mathcal{M}$ for L which fails to satisfy this sentence. Take

$$\mathcal{M} = \langle \mathbb{Z}, R^{\mathcal{M}} \rangle$$

where $R^{\mathcal{M}} = \leq$, the usual (non-strict) ordering on $\mathbb{Z}$. Then $\mathcal{M} \models \forall y \exists x R(x, y)$ (i.e. for any $b \in \mathbb{Z}$ there is $a \in \mathbb{Z}$ with $a \leq b$: that is, given any integer, there is an integer less than or equal to it). On the other hand $\mathcal{M} \not\models \exists x \forall y R(x, y)$ (i.e. it is not true that there is some $a \in \mathbb{Z}$ such that for any $b \in \mathbb{Z}$, $a \leq b$: that is, there is no least integer).

As another counterexample, take $L = \emptyset$ and let A be the formula $x = y$. If $\mathcal{M} = \langle M \rangle$ is any structure, with M having more than one element, then $\mathcal{M}$ does not satisfy this sentence.

(iv) In general, $\not\models (\forall x A \Rightarrow \forall x B) \Rightarrow \forall x (A \Rightarrow B)$.

Let $L = \{P, Q\}$, P, Q unary relation symbols. Let A be $P(x)$, B be $Q(x)$. Then $\not\models (\forall x P(x) \Rightarrow \forall x Q(x)) \Rightarrow \forall x (P(x) \Rightarrow Q(x))$. To see this, consider the structure $\mathcal{M} = \langle \mathbb{N}, P^{\mathcal{M}}, Q^{\mathcal{M}} \rangle$, where $P^{\mathcal{M}}(n)$ iff n is even, $Q^{\mathcal{M}}(n)$ iff n is odd. Then $\mathcal{M} \models \forall x P(x) \Rightarrow \forall x Q(x)$, simply because $\mathcal{M} \not\models \forall x P(x)$, but $\mathcal{M} \not\models \forall x (P(x) \Rightarrow Q(x))$ (because this would mean that $P^{\mathcal{M}} \subseteq Q^{\mathcal{M}}$).

(v) If A is a formula, B a subformula of A, $B' \equiv B$, and A' is obtained from A by substituting B by B' (not necessarily in all occurrences of B in A), then $A' \equiv A$. (This can be easily proved by induction on the construction of A.)

If A is a formula, x is a variable and t is a term, we denote by $A[x/t]$ the formula obtained by substituting every *free* occurrence of x in A by t.

Example 2.3.16

$$\begin{aligned} A &: \exists x R(x,y) \Rightarrow P(x,z) \\ t &: g(x,y) \\ A[x/t] &: \exists x R(x,y) \Rightarrow P(g(x,y),z). \end{aligned}$$

Fact 2.3.17 (Alphabetic change of bound variables) *If A is a formula, x is a variable and y is a variable* not occurring *in A, then*

$$\begin{aligned} \forall x A &\equiv \forall y A[x/y] \\ \exists x A &\equiv \exists y A[x/y]. \end{aligned}$$

Remark. The requirement that y not occur in A is crucial, otherwise the fact is not true. For example, we have

$$\forall x R(x,y) \equiv \forall z R(z,y)$$

but

$$\forall x R(x,y) \not\equiv \forall y R(y,y),$$

since in the latter case, y already occurs in the formula, so that after substitution the meaning becomes different. We will discuss this more in section 2.7.A.

Fact 2.3.18 *Every formula is equivalent to one using only $\neg, \wedge, \exists$ or only $\neg, \wedge, \forall$ (more generally only $S \cup \{\exists\}$ or only $S \cup \{\forall\}$, for any complete set of connectives S in propositional logic).*

This is because for any formula A, we have $\neg\forall x A \equiv \exists x \neg A$ (i.e. A is not true for all x iff it fails for a specific x). Equivalently, we also have $\forall x \neg A \equiv \neg\exists x A$ (i.e. A is false for all x iff it is not true for any x). Using these facts and corollary 1.6.6, we can eliminate all but the desired connectives and quantifier.

2.4 Definability in a Structure

When choosing a language and a structure, we must keep in mind what objects (functions, relations, constants) of the universe we want to be able to discuss in the language. Clearly we can discuss any function or relation for which the language has a symbol, but there are others: for example, we have seen that any term defines a function as well. Now we ask the question: which functions and relations in a structure can be defined and discussed in a given first-order language?

2.4.A First-Order Definability

First, we must say what it means to define something. Let L be a first-order language and $\mathcal{M} = \langle M, \dots \rangle$ a structure for L.

If $A(x_1, \dots, x_n)$ is a formula of L, the *graph* of A in $\mathcal{M}$ is the n-ary relation $GR_A^{\mathcal{M}} \subseteq M^n$ defined by

$$GR_A^{\mathcal{M}}(a_1, \dots, a_n) \text{ iff } \mathcal{M} \models A[a_1, \dots, a_n].$$

Example 2.4.1 Let $L = \{+, \cdot, <\}$ and $\mathcal{M} = \langle \mathbb{R}, +, \cdot, < \}$, and consider the following formulas, with their graphs.

$$A_0 : x = y$$
$$GR_{A_0}^{\mathcal{M}} = \{(a, b) : a = b\}$$
$$(GR_{A_0}^{\mathcal{M}}(a, b) \text{ iff } a = b)$$

$$A_1 : x < y$$
$$GR_{A_1}^{\mathcal{M}} = \{(a, b) : a < b\}$$
$$(GR_{A_1}^{\mathcal{M}}(a, b) \text{ iff } a < b)$$

$$A_2(x, y) : \exists z(z \cdot z = z \wedge \neg z + z = z \wedge x \cdot x + y \cdot y = z)$$
$$GR_{A_2}^{\mathcal{M}}(a, b) \text{ iff } a^2 + b^2 = 1$$

$$A_3(x, y) : \exists z(z \cdot z = z \wedge \neg z + z = z \wedge x \cdot x + y \cdot y < z)$$
$$GR_{A_3}^{\mathcal{M}}(a, b) \text{ iff } a^2 + b^2 < 1$$

$$A_4(x,y,z) : \exists w(w+w=w \wedge x+y+z=w)$$
$$GR^{\mathcal{M}}_{A_4}(a,b,c) \text{ iff } a+b+c=0$$

$$A_5(x,y) : A_3 \wedge (x \cdot x + y \cdot y < x + x)$$
$$GR^{\mathcal{M}}_{A_5}(a,b) \text{ iff } a^2+b^2<1 \text{ and } (a-1)^2+b^2<1$$

Example 2.4.2 Let $L = L_{ar} = \{0, S, +, \cdot, <\}$ and $\mathcal{N} = \langle \mathbb{N}, 0, S, +, \cdot, <\}$, and consider the formulas:

$$A(x) : \exists y(x = y \cdot y)$$
$$GR^{\mathcal{N}}_{A}(a) \text{ iff } a \text{ is a square.}$$

$$B(x,y) : \exists w \exists v[S(S(w)) = x \wedge S(S(v)) = y] \wedge$$
$$\forall z[\exists p(z \cdot p = x) \wedge \exists q(z \cdot q = y) \Rightarrow z = S(0))]$$

$GR^{\mathcal{N}}_{B}(a,b)$ iff $a, b \geq 2$ and a, b are prime to each other (i.e., $\gcd(a,b) = 1$).

It can be shown that there is a formula $P(x,y)$ in L_{ar} such that $GR^{\mathcal{N}}_{P}(a,b)$ iff a is the bth prime number, and also a formula $R(x,y,z)$ such that

$$GR^{\mathcal{N}}_{R}(a,b,c) \text{ iff } a^b = c.$$

This follows from a clever idea due to K. Gödel that allows us to *code* in an appropriate, first-order definable sense, finite sequences of numbers by numbers.

Definition 2.4.3 A relation $R \subseteq M^n$ is *first-order definable* if there is a formula $A(x_1, \ldots, x_n)$ such that $R = GR^{\mathcal{M}}_{A}$, i.e., for any $a_1, \ldots, a_n \in M$

$$R(a_1, \ldots, a_n) \text{ iff } \mathcal{M} \models A[a_1, \ldots, a_n].$$

A function $f : M^n \to M$ is *first-order definable* if its graph

$$\text{graph}(f) \subseteq M^{n+1}$$

is first-order definable, i.e. there is a formula $A(x_1, \ldots, x_n, x_{n+1})$ such that for all $a_1, \ldots, a_n, a_{n+1}$ in M,

$$f(a_1, \ldots, a_n) = a_{n+1} \text{ iff } M \models A[a_1, \ldots, a_n, a_{n+1}].$$

(If $n = 0$, an element $a \in A$ is *first-order definable* if there is a formula $A(x)$ such that a is the unique element of M with

$$\mathcal{M} \models A[a].)$$

Examples 2.4.4

(i) If f is a function symbol in L, then $f^{\mathcal{M}}$ is definable by the formula

$$f(x_1, \ldots, x_n) = x_{n+1}.$$

If R is a relation symbol in L, then $R^{\mathcal{M}}$ is definable by the formula

$$R(x_1, \ldots, x_n).$$

If t is a term, say t is $t(x_1, \ldots, x_n)$, then $t^{\mathcal{M}}$ is definable by the formula

$$t = x_{n+1}.$$

(ii) The definable relations in $\mathcal{M}$ form a *Boolean algebra*, i.e., are closed under $\sim, \cup, \cap$, where if $R \subseteq M^n$,

$$\sim R \subseteq M^n \text{ and } \sim R = M^n \setminus R,$$

and if $R, S \subseteq M^n$, then $R \cap S,\ R \cup S$ have their usual meaning. This is clear, since if R is defined by A, then $\sim R$ is defined by $\neg A$ and if R, S are defined by B, C, resp., then $R \cap S,\ R \cup S$ are defined by $B \wedge C,\ B \vee C$, resp.

(iii) The definable relations in $\mathcal{M}$ are closed under *projection*. If $R \subseteq M^{n+1}$ is defined by $A(x_1, \ldots x_n, x_{n+1})$, then $\text{proj}(R) = \{(a_1, \ldots, a_n)$: for some $a_{n+1},\ R(a_1, \ldots, a_{n+1})\}$ is defined by $\exists x_{n+1} A(x_1, \ldots, x_n, x_{n+1})$.

Example 2.4.5 Let $L = \{+, \cdot, <\}$ and $\mathcal{M} = \langle \mathbb{R}, +, \cdot, < \}$.

First we check, by induction, that every integer $n \in \mathbb{Z}$ is definable. The most important integers, of course, are 0 and 1:

$$\begin{aligned} 0 &: x + x = x && (A_0(x)) \\ 1 &: x \cdot x = x \wedge \neg x + x = x && (A_1(x)) \end{aligned}$$

Now assume a formula $A_n(x)$ defines n (in $\mathcal{M}$). Then the formula $\exists y \exists z (x = y + z \wedge A_n(y) \wedge A_1(z))$ $(A_{n+1}(x))$ defines $n + 1$. So by induction, any $n \geq 0$ is definable by a formula $A_n(x)$. Consider now the formula

$$A_{-n}(x) : \exists y \exists z (A_0(z) \wedge A_n(y) \wedge x + y = z)$$

This defines $-n$. So every integer can be defined.

One can then show that every rational number is definable, and (though this is a bit trickier) so is every algebraic number (solution of polynomial with integer coefficients).

Using this, it follows that all finite unions of intervals with algebraic endpoints are definable. It turns out that this is it, i.e. every definable unary relation (set) is a finite union of intervals with algebraic endpoints.

For binary, ternary, or higher order relations things are more complicated, but it turns out that every definable relation is a Boolean combination of relations that can be defined using polynomial equations or inequalities with integer coefficients, as for example:

$$(x \cdot x + y \cdot y < z \wedge x \cdot x \cdot x = y \cdot y + z) \vee x \cdot y + y \cdot z < x \cdot y \cdot z.$$

Every polynomial function with integer coefficients is definable (but there are more complex definable functions than polynomials). For example, the polynomial function $f(a, b, c) = 2a^2 - b + c^2$ is definable by the formula

$$A(x, y, z, w) : \exists p \exists q (A_2(p) \wedge A_{-1}(q) \wedge w = p \cdot x \cdot x + q \cdot y + z \cdot z).$$

Another formula that defines the graph of f is

$$B(x, y, z, w) : y + w = x \cdot x + x \cdot x + z \cdot z.$$

Example 2.4.6 Let $L = L_{ar} = \{0, S, +, \cdot, <\}$ and $\mathcal{N} = \langle \mathbb{N}, 0, S, +, \cdot\ < \rangle$. Again any polynomial function with coefficients in $\mathbb{N}$ is definable, but it turns out that much more complicated functions are definable, e.g. m^n, $i \mapsto p_i =$ the ith prime, etc. In fact, every standard function one studies in number theory is definable. (This is by no means obvious—try to define the exponential function, as an example.) Similarly all the ordinary relations, e.g. "n is prime", are definable. In fact "n is prime" is definable by the following formula:

$$A(x) : S(0) < x \wedge \forall y \forall z (y \cdot z = x \Rightarrow y = S(0) \vee z = S(0)).$$

Remark. Assuming that the language L has only countably many symbols (i.e. the non-logical symbols in L can be enumerated in a sequence), there are only countably many definable relations and functions in each given structure $\mathcal{M}$ for L. Since in any structure whose universe is infinite, there are uncountably many possible relations and functions, the definable ones form a very small subset of these. But so far, we have not seen even one example of a *non*-definable relation.

Definition 2.4.7 Let L be a first-order language and $\mathcal{M} = \langle M, \dots \rangle$ a structure for L. A relation $R \subseteq M^n$ is *definable with parameters* if there is a formula $A(x_1, \dots, x_n, x_{n+1}, \dots, x_{n+m})$, $m \geq 0$, and fixed $p_1, \dots, p_m \in \mathcal{M}$ (the *parameters*), such that

$$R(a_1, \dots, a_n) \text{ iff } \mathcal{M} \models A[a_1, \dots, a_n, p_1, \dots, p_m].$$

Example 2.4.8 Let $L = \{+, \cdot, <\}$ and $\mathcal{M} = \langle \mathbb{R}, +, \cdot, < \rangle$. Consider an interval $(b, c) \subseteq \mathbb{R}$. For arbitary b, c this may not be definable, but it is always definable with parameters:

$$A(x, y, z) : y < x \wedge x < z$$
$$a \in (b, c) \text{ iff } \mathcal{M} \models A[a, \underbrace{b, c}_{\text{parameters}}]$$

Similarly, a function $f : M^n \to M$ is *definable with parameters* whenever its graph is definable with parameters. Notice also that every element of M is definable with parameters; one parameter suffices (that element).

Example 2.4.9 In example 2.4.5, *every* polynomial function is definable with parameters.

2.4.B Isomorphisms

We will next discuss a method for showing that certain relations are *not* definable.

Definition 2.4.10 Let L be a first order language and $\mathcal{M}_1 = \langle M_1, \dots \rangle$ and $\mathcal{M}_2 = \langle M_2, \dots \rangle$ be two structures of L. An *isomorphism* of $\mathcal{M}_1$ with $\mathcal{M}_2$ is a map $\pi : M_1 \to M_2$ which is one-to-one and onto (i.e. a bijection of M_1, M_2) and such that for each n-ary function symbol f in L,

$$\pi(f^{\mathcal{M}_1}(a_1, \dots, a_n)) = f^{\mathcal{M}_2}(\pi(a_1), \dots, \pi(a_n)),$$

for all $a_1, \dots, a_n \in M_1$, and for any m-ary relation symbol R in L,

$$R^{\mathcal{M}_1}(a_1, \dots, a_m) \text{ iff } R^{\mathcal{M}_2}(\pi(a_1), \dots, \pi(a_m)),$$

for all $a_1, \dots, a_m \in M_1$.

We denote this by

$$\pi : \mathcal{M}_1 \cong \mathcal{M}_2$$

Clearly also

$$\pi^{-1} : \mathcal{M}_2 \cong \mathcal{M}_1.$$

Example 2.4.11 Let $L = \{f; R\}$ with f a binary function symbol and R a binary relation symbol, and consider the structures

$$\begin{aligned} \mathcal{M}_1 &= \langle \mathbb{R}^+, \cdot, < \rangle \quad (\text{i.e. } \cdot = f^{\mathcal{M}_1},\ < = R^{\mathcal{M}_1}) \\ \mathcal{M}_2 &= \langle \mathbb{R}, +, < \rangle \quad (\text{i.e., } + = f^{\mathcal{M}_2},\ < = R^{\mathcal{M}_2}) \end{aligned}$$

where $\mathbb{R}^+ = \{\alpha \in \mathbb{R} : \alpha > 0\}$. Let $\pi : \mathbb{R}^+ \to \mathbb{R}$ be given by $\pi(a) = \ln a$. Then $\pi : \mathcal{M}_1 \cong \mathcal{M}_2$ and $\pi^{-1}(a) = e^a$.

We now have the following basic fact.

Theorem 2.4.12 *If $\pi : \mathcal{M}_1 \cong \mathcal{M}_2$ and $A(x_1, \ldots, x_n)$ is any formula, then for any $a_1, \ldots, a_n \in M_1$,*

$$\mathcal{M}_1 \models A[a_1, \ldots, a_n] \text{ iff } \mathcal{M}_2 \models A[\pi(a_1), \ldots, \pi(a_n)].$$

Proof. First show by induction on the construction of the term t, that if t is $t(x_1, \ldots, x_n)$, then

$$\pi(t^{\mathcal{M}_1}(a_1, \ldots, a_n)) = t^{\mathcal{M}_2}(\pi(a_1), \ldots, \pi(a_n))$$

for any $a_1, \ldots, a_n \in M_1$. Then the theorem can be proved by induction on the construction of A. $\dashv$

Definition 2.4.13 If $\mathcal{M}_1 = \mathcal{M}_2 = \mathcal{M}$, we call any isomorphism $\pi : \mathcal{M} \cong \mathcal{M}$ an *automorphism* of $\mathcal{M}$.

Example 2.4.14 If $\mathcal{M} = \langle \mathbb{Q}, +, < \rangle$, then $\pi(a) = 3a$ is an automorphism of $\mathcal{M}$.

In particular, if π is any automorphism of $\mathcal{M}$, the preceding theorem says that for any formula $A(x_1, \ldots, x_n)$ and any $a_1, \ldots, a_n \in M$

$$\mathcal{M} \models A[a_1, \ldots, a_n] \quad \text{iff} \quad \mathcal{M} \models A[\pi(a_1), \ldots, \pi(a_n)]$$

i.e. every definable relation is invariant under automorphisms.

Similarly, any definable relation with parameters $p_1, \ldots, p_m$ is invariant under any automorphism π that fixes the parameters, i.e. $\pi(p_i) = p_i$, $i = 1, \ldots, m$.

We can use this fact to prove that certain relations are not definable: just exhibit an automorphism of the structure under which they are not invariant. Here are some examples.

Application. Consider the language $L = \emptyset$ with no nonlogical symbols and any structure $\mathcal{M} = \langle M \rangle$ for it. What are the definable subsets (i.e. unary relations) on M?

Clearly $\emptyset, M$ are definable (by $x \neq x$, $x = x$, resp.). We claim that these are the only definable subsets of M. Let A be a subset of M, $A \neq \emptyset$, $A \neq M$, and let $a \in A$, $b \notin A$. Let $\pi : M \to M$ be the bijection such that $\pi(a) = b$, $\pi(b) = a$ and $\pi(c) = c$ if $c \notin \{a, b\}$. Then π is an automorphism of M but A is not invariant under π, so A is not definable.

What are the definable with parameters subsets of M? Clearly any finite subset $\{a_1, \ldots, a_n\} \subseteq \mathcal{M}$ is definable by the formula

$$x = x_1 \vee \cdots \vee x = x_n$$

and parameters $a_1, \ldots, a_n$ (for $x_1, \ldots, x_n$). But then every co-finite (complement of finite) subset of M is definable with parameters. We claim these are the only ones. Let $A \subseteq M$ be neither finite or co-finite. Assume it is definable with parameters $p_1, \ldots, p_n$. Since A is neither finite nor co-finite, there is some element of A, say a, distinct from all $p_1, \ldots, p_n$ and some element of $M \setminus A$, say b, distinct from $p_1, \ldots, p_n$. Let $\pi(p_i) = p_i$, $\pi(a) = b$, $\pi(b) = a$, $\pi(c) = c$ if $c \notin \{p_1, \ldots, p_n, a, b\}$. Then π is an automorphism of $\mathcal{M}$ that fixes the parameters, but does not leave A invariant, a contradiction.

Application. $\mathbb{N}$ is not definable in

$$\mathcal{M} = \langle \mathbb{R}, 0, 1, \cdot, < \rangle$$

Indeed, $\pi(a) = a^3$, is an automorphism of $\mathcal{M}$ but it does not leave $\mathbb{N}$ invariant, i.e., $a \in \mathbb{N}$ iff $\pi(a) \in \mathbb{N}$ clearly fails. (It turns out that $\mathbb{N}$ is not even definable in $\langle \mathbb{R}, 0, 1, +, \cdot, < \}$, even though each integer is.)

Remark. This method of automorphisms does not always apply. For example, there are structures which have no automorphisms except the identity

(these are called *rigid*). An example is

$$\langle \mathbb{N}, 0, S \rangle.$$

Another is

$$\langle \mathbb{R}, 0, 1, +, \cdot, < \rangle$$

So although, for example, it turns out that $\cdot$ is not definable in $\langle \mathbb{N}, 0, S, +, < \rangle$, this cannot be shown by the automorphism method.

Remark. By the way, $+$ is definable in $\langle \mathbb{N}, 0, S, \cdot \rangle$ since:

$$a + b = c \text{ iff } [(a+1)(c+1)+1][b(c+1)+1] = (c+1)^2[(a+1)b+1]+1$$
$$\text{i.e. } S(S(a) \cdot S(c)) \cdot S(b \cdot S(c)) = S((S(c) \cdot S(c)) \cdot S(S(a) \cdot b))$$

2.5 Prenex Normal Forms and Games

We will now discuss a "normal form" for first-order wffs which makes many proofs easier, and follow it with an application to game theory.

2.5.A Prenex Normal Forms

Definition 2.5.1 A formula A (in a fixed language L) is in *prenex normal form* (pnf) if A has the form

$$Q_1 y_1 Q_2 y_2 \ldots Q_n y_n B,$$

where y_i are distinct variables, each Q_i is either $\exists$ or $\forall$ and B is *quantifier-free* (has no quantifers), i.e. it is built from atomic formulas using propositional connectives only.

We call $Q_1 y_1 Q_2 y_2 \ldots Q_n y_n$ the *prefix* of A and B the *matrix* of A.

Example 2.5.2

$$\overbrace{\forall x \forall y \exists z \exists w}^{\text{prefix}} \overbrace{[P(x,z) \Rightarrow \neg Q(y,w)]}^{\text{matrix}}$$
$$\text{(empty prefix)} \ \underbrace{P(x,y) \vee S(f(x), h(y))}_{\text{matrix}}$$

are in pnf, but

$$\neg \exists x [P(x) \wedge \exists y (Q(x,y) \vee \neg \exists z R(x,z))]$$

is *not* in pnf.

Theorem 2.5.3 *For each formula A, we can find a formula A^* in pnf, logically equivalent to A, i.e., $A \equiv A^*$.*

Proof. We will use the following equivalences. If $Q = \forall$ (resp. $\exists$), let $\check{Q} = \exists$ (resp. $\forall$); we call $\check{Q}$ the *dual* of Q.

(a) $\neg QxA \equiv \check{Q}x\neg A$

(b) $A \vee (QxB) \equiv Qx(A \vee B)$, if x if not free in A

(c) $A \wedge (QxB) \equiv Qx(A \wedge B)$, if x is not free in A

(d) $(A \Rightarrow QxB) \equiv Qx(A \Rightarrow B)$, if x is not free in A

(e) $(QxA \Rightarrow B) \equiv \check{Q}x(A \Rightarrow B)$, if x is not free in B.

(f) $Q_1y_1Q_2y_2 \ldots Q_ny_nA \equiv Q_1y_1'Q_2y_2' \ldots Q_ny_n'A[y_1/y_1', \ldots, y_n/y_n']$, where A is quantifier-free and $y_1' \ldots y_n'$ are any distinct variables not appearing in $Q_1y_1Q_2y_2 \ldots Q_ny_nA$ (it is assumed that $y_1, \ldots, y_n$ are distinct here.)

(g) If in $Q_1y_1Q_2y_2 \ldots Q_ny_nA$, A quantifier free, a variable y_i appears more than once and Q_my_i $(m \leq n)$ is the rightmost occurrence of y_i in the prefix, then if we eliminate all $Q_{m'}y_i$ for $m' < m$, we obtain a formula B equivalent to $Q_1y_1 \ldots Q_ny_nA$.

We now show how to construct A^* by recursion.

- If A is atomic, we simply take $A^* = A$.
- Assume A^*, B^* have been constructed, and consider $\neg A,\ A \wedge B,\ A \vee B,\ A \Rightarrow B,\ A \Leftrightarrow B$.

  We can eliminate the last case, $A \Leftrightarrow B$, simply by replacing it by $(A \Rightarrow B) \wedge (B \Rightarrow A)$. We do this because there is no simple equivalence corresponding to a–e for $\Leftrightarrow$, so it would be rather complicated to consider separately. (We could also eliminate $\Rightarrow$ and one of $\wedge$ and $\vee$ as well, but this does not seem to offer any great advantage.) We will construct, therefore, $(\neg A)^*$, $(A \wedge B)^*$, $(A \vee B)^*$, and $(A \Rightarrow B)^*$.

Let

$$\begin{aligned} A^* &: Q_1 y_1 \dots Q_n y_n C \\ B^* &: Q'_1 z_1 \dots Q'_m z_m D. \end{aligned}$$

Then, by applying (a) n times we see that

$$\begin{aligned} \neg A &\equiv \neg A^* \\ &\equiv \check{Q}_1 y_1 \check{Q}_2 y_2 \dots \check{Q}_n y_n \neg C \end{aligned}$$

so $(\neg A)^* = \check{Q}_1 y_1 \dots \check{Q}_n y_n \neg C$.

Also

$$\begin{aligned} A \wedge B &\equiv A^* \wedge B^* \\ &\equiv Q_1 y_1 \dots Q_n y_n C \wedge Q'_1 z_1 \dots Q'_m z_m D. \end{aligned}$$

Using (f) we have that

$$Q'_1 z_1 \dots Q'_m z_m D \equiv Q'_1 u_1 \dots Q'_m u_m D[z_1/u_1, \dots z_m/u_1]$$

where $u_1, \dots, u_m$ are variables distinct from all of the variables in $Q_1 y_1 \dots Q_m y_m C$ and $Q'_1 z_1 \dots Q'_m z_m D$. Thus

$$\begin{aligned} A \wedge B &\equiv Q_1 y_1 \dots Q_n y_n C \wedge Q'_1 u_1 \dots Q'_m u_m D[z_1/u_1 \dots z_m/u_m] \\ &\equiv Q'_1 u_1 \dots Q'_m u_m (Q_1 y_1 \dots Q_n y_n C \wedge D[z_1/u_1, \dots, z_m/u_m]) \end{aligned}$$

by repeatedly using (c). Now let $v_1, \dots, v_n$ be distinct variables not appearing in $Q_1 y_1 \dots Q_n y_n C$ and $D[z_1/u_1, \dots, z_m/u_m]$. Then by using (f) and (c) (and the fact that $P \wedge Q \equiv Q \wedge P$) we have

$$\begin{aligned} A \wedge B &\equiv Q'_1 u_1 \dots Q'_m u_m (Q_1 v_1 \dots Q_n v_n C[y_1/v_1, \dots, y_n/v_n] \wedge \\ &\qquad D[z_1/u_1, \dots, z_m/u_m]) \\ &\equiv Q'_1 u_1 \dots Q'_m u_m Q_1 v_1 \dots Q_n v_n (C[y_1/v_1, \dots, y_n/v_n] \wedge \\ &\qquad D[z_1/u_1, \dots, z_m/u_m]), \end{aligned}$$

and this last formula is by definition $(A \wedge B)^*$.

The cases of $(A \vee B)^*$ and $(A \Rightarrow B)^*$ are similar, using (b)-(e).

- Finally, assume A^* is defined and consider $\exists x A$ and $\forall x A$. If $A^* = Q_1 y_1 \ldots Q_n y_n C$, then

$$\begin{aligned}\exists x A &\equiv \exists x A^* \\ &\equiv \exists x Q_1 y_1 \ldots Q_n y_n C.\end{aligned}$$

  If x is different from all $y_1, \ldots, y_n$ we simply take

$$(\exists x A)^* = \exists x Q_1 y_1 \ldots Q_n y_n C.$$

  Otherwise, using (g), we take

$$(\exists x A)^* = Q_1 y_1 \ldots Q_n y_n C$$

  The case of $(\forall x A)^*$ is similar. $\dashv$

Example 2.5.4 Let A be the formula:

$$\neg\exists x(P(x) \Rightarrow \exists y(Q(z,y) \vee \neg\exists z R(x,z))).$$

The following sequence of steps transforms it in pnf.

$$\begin{gathered}\neg\exists x(P(x) \Rightarrow \exists y(Q(z,y) \vee \neg\exists z R(x,z))) \\ \neg\exists x(P(x) \Rightarrow \exists y(Q(z,y) \vee \forall z \neg R(x,z))) \\ \neg\exists x(P(x) \Rightarrow \exists y \forall z'(Q(z,y) \vee \neg R(x,z'))) \\ \neg\exists x \exists y \forall z'(P(x) \Rightarrow (Q(z,y) \vee \neg R(x,z'))) \\ \forall x \forall y \exists z' \neg(P(x) \Rightarrow (Q(z,y) \vee \neg R(x,z')))\end{gathered}$$

Remark. Of course the matrix of a formula is pnf can always be replaced, up to $\equiv$, with one which is in disjunctive or conjunctive normal form (on atomic formulas).

If $A = Q_1 y_1 \ldots Q_n y_n B$, is a formula in pnf we can group together consecutive similar quantifiers and write it in one of the forms

$$\exists \bar{z_1} \forall \bar{z_2} \cdots Q \bar{z_n} B \quad \text{or} \quad \forall \bar{z_1} \exists \bar{z_2} \ldots Q \bar{z_n} B,$$

where $\bar{z}_i = z^i_1, \ldots, z^i_{k_i}$ is a string of variables and $Q\bar{z}_i$ is an abbreviation for $Q z^i_1 Q z^i_2 \ldots Q z^i_{k_i}$ (where Q is $\exists$ or $\forall$).

Example 2.5.5

$$\exists y_1 \exists y_2 \forall y_3 \exists y_4 \exists y_5 B(y_1, \ldots, y_5)$$

will be written as

$$\exists y_1, y_2 \forall y_3, y_4 \exists y_5 B(y_1, \ldots, y_5)$$

and

$$\forall y_1 \exists y_2 \exists y_3 \exists y_4 \forall y_5 \forall y_6 B(y_1, \ldots, y_6)$$

as

$$\forall y_1 \exists y_2, y_3, y_4 \forall y_5, y_6 B(y_1, \ldots y_6).$$

2.5.B Games and Strategies

Let $\mathcal{M}$ be a structure for the language L in which A is a sentence in prenex normal form. We will reformulate the statement $\mathcal{M} \models A$ in terms of a game. For simplicity, we will assume that A is a sentence of the form

$$A : \exists z_1 \ \forall z_2 \exists z_3 \ldots \forall z_{2n} B(z_1, \ldots, z_{2n}).$$

Obvious modifications can be made to handle the general case.

We associate with $\mathcal{M}, A$ a game $G_A^{\mathcal{M}}$ played as follows: We have two players called $\exists$ and $\forall$. $\exists$ plays first an arbitrary element $a_1 \in M$. Then $\forall$ plays an arbitrary element $a_2 \in M$. $\exists$ plays then $a_3 \in M$, $\forall$ plays $a_4 \in M$, etc., for a total of $2n$ moves. We say that $\exists$ wins this run of the game if

$$\mathcal{M} \models B[z_1 \mapsto a_1, \ldots, z_{2n} \mapsto a_{2n}]$$

and $\forall$ wins this run of the game if

$$\mathcal{M} \models \neg B[z_1 \mapsto a_1, \ldots, z_{2n} \mapsto a_{2n}].$$

$\exists$	$\forall$
a_1	
	a_2
a_3	
	a_4
$\vdots$	$\vdots$
a_{2n-1}	
	a_{2n}

It is assumed in this game that each player can see the opponent's previous moves (i.e. when $\forall$ plays a_2, he knows a_1 etc.).

Fact 2.5.6 *(i)* $\mathcal{M} \models A$ *iff* $\exists$ *has a winning strategy in* $G_A^{\mathcal{M}}$.

(ii) $\mathcal{M} \models \neg A$ *iff* $\forall$ *has a winning strategy in* $G_A^{\mathcal{M}}$.

Proof. Assume $\mathcal{M} \models A$, i.e. $\mathcal{M} \models \exists z_1 \forall z_2 \ldots \forall z_{2n} B(z_1, \ldots, z_{2n})$. Then there is $a_1 \in M$ such that

$$\mathcal{M} \models \forall z_2 \exists z_3 \ldots \forall z_{2n} B(z_1, \ldots, z_{2n})[z_1 \mapsto a_1].$$

$\exists$ starts by playing a fixed such a_1. Then for any $a_2 \in M$ that $\forall$ could play in his next move, we have

$$\mathcal{M} \models \exists z_3 \forall z_4 \ldots \forall z_{2n} B(z_1, \ldots, z_{2n})[z_1 \mapsto a_1,\ z_2 \mapsto a_2].$$

Thus $\exists$ can respond by playing some a_3 so that

$$\mathcal{M} \models \forall z_4 \ldots \forall z_{2n} B(z_1, \ldots, z_{2n})[z_1 \mapsto a_1,\ z_2 \mapsto a_2,\ z_3 \mapsto a_3],$$

and so on. By induction, if $\exists$ follows this strategy, once $a_1, a_2, \ldots, a_{2n}$ have been played, then $\mathcal{M} \models B[z_1 \mapsto a_1, \ldots, z_{2n} \mapsto a_{2n}]$, i.e. $\exists$ has won.

If on the other hand $\mathcal{M} \models \neg A$, then since

$$\begin{aligned} \neg A &= \neg \exists z_1 \forall z_2 \ldots \forall z_{2n} B(z_1, \ldots, z_{2n}) \\ &\equiv \forall z_1 \exists z_2 \ldots \exists z_{2n} \neg B(z_1, \ldots, z_{2n}), \end{aligned}$$

we have

$$\mathcal{M} \models \forall z_1 \exists z_2 \ldots \exists z_{2n} \neg B(z_1, \ldots, z_{2n}).$$

So assume now $\exists$ starts with an arbitrary $a_1 \in M$. Then we have

$$\mathcal{M} \models \exists z_2 \forall z_3 \ldots \exists z_{2n} \neg B(z_1, \ldots, z_{2n})[z_1 \mapsto a_1].$$

So $\forall$ can respond by playing some a_2 such that

$$\mathcal{M} \models \forall z_3 \exists z_4 \ldots \exists z_{2n} \neg B(z_1, \ldots, z_{2n})[z_1 \mapsto a_1,\ z_2 \mapsto a_2]$$

etc., as before. If $\forall$ follows this strategy, at the end of this run we will have $a_1, a_2, \ldots, a_{2n}$ such that

$$\mathcal{M} \models \neg B[z_1 \mapsto a_1, \ldots, z_{2n} \mapsto a_{2n}],$$

so $\forall$ has won.

Since it is clear that it cannot be that *both* $\exists$ and $\forall$ have winning strategies in $G_A^{\mathcal{M}}$ (otherwise they can play their winning strategies against each other and then they both will win, which is impossible), it follows that if $\exists$ has a winning strategy in $G_A^{\mathcal{M}}$, then we cannot have $\mathcal{M} \models \neg A$, so we must have $\mathcal{M} \models A$. Similarly, in case $\forall$ has a winning strategy, we have $\mathcal{M} \models \neg A$, so we are done. $\dashv$

Example 2.5.7 Consider the sentence A:

$$\exists x_1 \forall x_2 \exists x_3 (x_3 + x_3 = x_2 \vee x_3 + x_3 = x_2 + x_1)$$

Let $\mathcal{M} = \langle \mathbb{N}, + \rangle$. Then the game $G_A^{\mathcal{M}}$ is as follows

$\exists$	$\forall$
a_1	
	a_2
a_3	

where $\exists$ wins if $2a_3 = a_2 \vee 2a_3 = a_1 + a_2$, otherwise $\forall$ wins. Then $\exists$ has the following winning strategy (and hence $\mathcal{M} \models A$):

$\exists$ starts with $a_1 = 1$. Then after $\forall$ plays an arbitrary a_2, $\exists$ responds by playing $a_3 = \frac{a_2}{2}$, if a_2 is even, and by $a_3 = \frac{a_2+1}{2}$, if a_2 is odd.

Using this interpretation, we can give an application of first-order logic to game theory.

Definition 2.5.8 A *finite game* is determined by a set M and a relation $R \subseteq M^k$ $(k = 1, 2, \dots)$. Take for simplicity $k = 2n$ to be even. The game is played as follows: We have two players, I, II which take turns in playing $a_1, a_2, \dots, a_{2n-1}, a_{2n}$ in M. I wins this run of the game if $R(a_1, \dots, a_{2n})$. Otherwise II wins (i.e. if $R(a_1, \dots, a_{2n})$ fails).

I	II
a_1	
	a_2
a_3	
	a_4
$\vdots$	$\vdots$
a_{2n-1}	
	a_{2n}

We can use fact 2.5.6 to prove the following:

Theorem 2.5.9 (Zermelo, von Neumann) *Every finite game is* determined*, i.e. one of the two players has a winning strategy.*

Proof. Consider the language with one $2n$-ary relation symbol $\bar{R}$ and let $\mathcal{M}$ be its structure

$$\mathcal{M} = \langle M, R \rangle,$$

(i.e. $R = \bar{R}^{\mathcal{M}}$). Let A be the sentence

$$\exists x_1 \forall x_2 \ldots \exists x_{2n-1} \forall x_{2n} \bar{R}(x_1, \ldots, x_{2n}).$$

Clearly $G_A^{\mathcal{M}}$ is the above finite game, with I=$\exists$ and II=$\forall$, so by fact 2.5.6, if $\mathcal{M} \models A$, I has a winning strategy, while if $\mathcal{M} \models \neg A$, II has a winning strategy, and since either $\mathcal{M} \models A$ or $\mathcal{M} \models \neg A$, one of the two must have a winning strategy. $\dashv$

Remark. One could also define *basically finite* games. Here we have a set M and for each n a (possibly empty) relation $R_n \subseteq M^n$. Let $R = \bigcup_n R_n$. Players I and II alternate playing elements a_i of M as before. In order for the game to be basically finite, two conditions need to hold: (1) At each *move*, a player has only finitely many possibilities, and (2) the game always ends after finitely many stages, when the output $(a_1, a_2, \ldots, a_n)$ thus produced does not belong to R. The last player to have played loses. An easy application of König's lemma shows that if a game if basically finite then there is an n such that each play of the game lasts at most n moves. It then follows from the argument of the Zermelo-von Neumann theorem that basically finite games are also determined.

One can continue this line of argument and analyze infinite determined games, but we will not pursue this road here.

2.6 Theories

Let S be a set of sentences in a fixed first-order language L. We denote by $\mathrm{Con}(S)$ the set of all sentences which are logical consequences of S, i.e.

$$\mathrm{Con}(S) = \{A : S \models A, A \text{ a sentence}\}.$$

Notice that $\mathrm{Con}(\mathrm{Con}(S)) = \mathrm{Con}(S)$, i.e. $\mathrm{Con}(S)$ is closed under logical consequences.

Definition 2.6.1 A set of sentences T is called a *theory* if it is closed under logical consequences, i.e. iff $T = \mathrm{Con}(T)$.

Definition 2.6.2 If $\mathrm{Con}(S) = T$, we say that S is a *set of axioms* for the theory T. Note that in this case S and T have exactly the same models.

Example 2.6.3 Consider $L = \{<\}$, $<$ a binary relation symbol. The theory of *partial order* has as axioms:

(1) $\forall x \forall y (x < y \Rightarrow \neg y < x)$ (antisymmetry)
(2) $\forall x \forall y \forall z (x < y \wedge y < z \Rightarrow x < z)$ (transitivity)

A model $\langle M, < \rangle$ of this theory is called a *partially ordered set* (or just *partial order*). If we add the axiom:

(3) $\forall x \forall y (x < y \vee x = y \vee y < x)$ (linearity)

we obtain the theory of *linear order*, whose models are called *linearly ordered sets* (or just *linear orders*).
The theory of *dense linear order* is obtained by adding two more axioms:

(4) $\exists x \exists y (x \neq y)$
(5) $\forall x \forall y (x < y \Rightarrow \exists z (x < z \wedge z < y))$ (density)

Examples of models of this theory (i.e., *dense linear orders*) are $\langle \mathbb{Q}, < \rangle$, $\langle \mathbb{R}, < \rangle$. Examples of linear orders which are not dense are $\langle \mathbb{Z}, < \rangle$ and $\langle \mathbb{N}, < \rangle$.
Finally the theory of *dense linear orders without endpoints* is obtained by adding the axioms:

(6) $\forall x \exists y (x < y)$
(7) $\forall x \exists y (y < x)$.

Models of these are again $\langle \mathbb{R}, < \rangle$, $\langle \mathbb{Q}, < \rangle$, but not $\langle [0,1], < \rangle$.

Remark. A *well ordered set* (or *wellorder*) is a linear order $\langle M, < \rangle$ with the following additional property: Every non-empty set $S \subseteq M$ has a least element, e.g. $\langle \mathbb{N}, < \rangle$. It turns out that one cannot express this by axioms in first-order logic, i.e. there is no set of axioms S in the first-order language $\{<\}$ whose models are exactly the wellorders. This will follow from the Compactness Theorem, that we will prove later on.

Example 2.6.4 Consider $L = \{E\}$, E a binary relation symbol. The theory of *(undirected) graphs* has as axioms

(1) $\forall x \neg (xEx)$
(2) $\forall x \forall y (xEy \Rightarrow yEx)$

A model of this theory is a *graph*.

Remark. It turns out again that there is no set of axioms in first-order logic whose models are exactly the connected graphs. Similarly for trees, i.e. connected acyclic graphs.

Example 2.6.5 Let $L = \{e, \cdot\}$, e a constant symbol and $\cdot$ a binary relation symbol. The *theory of groups* has the following axioms:

(1) $\forall x \forall y \forall z (x \cdot (y \cdot z) = (x \cdot y) \cdot z)$ (associativity)
(2) $\forall x (x \cdot e = x \wedge e \cdot x = x)$ (identity)
(3) $\forall x \exists y (x \cdot y = e \wedge y \cdot x = e)$ (existence of inverses)

A model of this theory is called a *group*, e.g. $\langle \mathbb{Z}, 0, + \rangle$, or $\langle \mathbb{R}^+, 1, \cdot \rangle$, where

$$\mathbb{R}^+ = \{x \in \mathbb{R} : x > 0\}.$$

The theory of *abelian groups* has one more axiom:
(4) $\forall x \forall y (x \cdot y = y \cdot x)$ (commutativity)

Examples of abelian groups are $\langle \mathbb{Z}, 0, + \rangle$, $\langle \mathbb{Z}_m, 0, + \rangle$, etc. Examples of groups which are not abelian are $\langle \mathcal{C}([0,1]) \circ \rangle$, (functions under composition) and $\langle GL_n(\mathbb{R}), \cdot \rangle$ ($n \times n$ matrices with matrix multiplication).

Example 2.6.6 Let $L = \{0, 1, +, \cdot\}$, 0, 1 constant symbols , $+, \cdot$ binary function symbols. The theory of *commutative rings with identity* has the following axioms:

(1)-(4) Same as the four axioms in example 2.6.5 with $\cdot$ replaced by $+$ and e by 0 (i.e. the axioms for abelian groups for $0, +$)
(5) $\forall x \forall y \forall z (x \cdot (y \cdot z) = (x \cdot y) \cdot z)$ (associativity for $\cdot$)
(6) $\forall x \forall y (x \cdot y = y \cdot x)$ (commutativity for $\cdot$)
(7) $\forall x (x \cdot 1 = x)$ (identity for $\cdot$)
(8) $\forall x \forall y \forall z (x \cdot (y + z) = x \cdot y + x \cdot z)$ (distributivity)

Examples of models of this theory (i.e. *commututive rings with identity*) are $\langle \mathbb{Z}, 0, 1, +, \cdot \rangle$, $\langle \mathbb{Q}, 0, 1, +, \cdot \rangle$, $\langle \mathbb{Z}_m, 0, 1, +, \cdot \rangle$, etc.
The *theory of integral domains* is obtained by adding the axiom
(9) $\forall x \forall y (x \cdot y = 0 \Rightarrow x = 0 \vee y = 0)$ (no zero divisors)

Examples of integral domains are $\langle \mathbb{Q}, 0, 1, +, \cdot \rangle$, $\langle \mathbb{Z}, 0, 1, +, \cdot \rangle$, $\langle \mathbb{Z}_p, 0, 1, +, \cdot \rangle$ (p a prime), but not $\langle \mathbb{Z}_6, 0, 1, +, \cdot \rangle$.
The theory of *fields* is obtained by adding the axioms:
(10) $0 \neq 1$
(11) $\forall x (x \neq 0 \Rightarrow \exists y (x \cdot y = 1))$ (existence of inverses for $\cdot$)

For example, $\langle \mathbb{Q}, 0, 1, +, \cdot \rangle, \langle \mathbb{R}, 0, 1, +, \cdot \rangle$ are fields, but $\langle \mathbb{Z}, 0, 1, +, \cdot \rangle$ is not. The theory of *fields of characteristic zero* is obtained by adding the infinite list of axioms:

$(12)_2$ $1 + 1 \neq 0$
$(12)_3$ $1 + 1 + 1 \neq 0$
$\vdots$
$(12)_n$ $\underbrace{1 + 1 + \cdots + 1}_{n} \neq 0$

for each $n \geq 2$. Examples of models of this theory, i.e. fields of characteristic zero, are $\langle \mathbb{Q}, 0, 1, +, \cdot \rangle$, $\langle \mathbb{R}, 0, 1, +, \cdot \rangle$, $\langle \mathbb{C}, 0, 1, +, \cdot \rangle$ but not $\langle \mathbb{Z}_p, 0, 1, +, \cdot \rangle$, p a prime.

Remark. Notice that the theory of fields of characteristic zero has infinitely many axioms. It turns out that it is not *finitely axiomatizable*, i.e. it cannot have a finite set of axioms.

Another way a theory can arise in practice is as follows. Let L be a first-order language and $\mathcal{M}$ as structure for L. The *theory* of $\mathcal{M}$, $\mathrm{Th}(\mathcal{M})$, is defined by

$$\mathrm{Th}(\mathcal{M}) = \{A : A \text{ a sentence and } \mathcal{M} \models A\}.$$

(It is indeed easy to check that $\mathrm{Th}(\mathcal{M})$ is a theory.)

Notice that here $\mathrm{Th}(\mathcal{M})$ is not given in any meaningful sense in terms of axioms (except in the trivial way, namely taking these axioms to be the theory itself). It is often an important problem to find a reasonable set of axioms for the theory of $\mathcal{M}$, for various structures of mathematical interest. In other words, we are looking for some explicit list of sentences true about $\mathcal{M}$, so that any other one follows logically from them.

Example 2.6.7 Consider $L = \{0, 1, +, \cdot\}$ and $\mathcal{M} = \langle \mathbb{R}, 0, 1, +, \cdot \rangle$. It turns out that the following is a set of axioms for $\mathrm{Th}(\mathcal{M})$:

(1)-(11) the axioms for fields
(12) $\forall x \exists y (x = y^2 \vee x + y^2 = 0)$
(here y^2 abbreviates $y \cdot y$; similarly y^n abbreviates $\underbrace{y \cdot y \cdot \cdots \cdot y}_{n}$)

(13) $\forall x \forall y \exists z (x^2 + y^2 = z^2)$
(14) $\forall x (x^2 \neq -1)$
$(15)_n$ $\forall x_0 \ldots \forall x_n (x_n \neq 0 \Rightarrow \exists y (x_n \cdot y^n + x_{n-1} \cdot y^{n-1} + \cdots + x_1 \cdot y + x_0 = 0))$,
where $n = 1, 3, 5 \ldots$ ranges over all *odd* numbers.

Example 2.6.8 For the same L and $\mathcal{M} = \langle \mathbb{C}, 0, 1, +, \cdot \rangle$, it turns out that a set of axioms consists of the axioms for fields of characteristic 0 plus all the axioms $(15)_n$ or above but now for $n = 1, 2, 3, \ldots$ ranging over *all* numbers.

Example 2.6.9 Let now $L_{ar} = \{0, S, +, \cdot, <\}$ be the language of arithmetic and $\mathcal{N}$ be its standard structure $\langle \mathbb{N}, 0, S, +, \cdot, <\}$. Consider the following set of axioms, called the *first-order Peano axioms* (PA):

$$
\begin{array}{ll}
(1) & \forall x(S(x) \neq 0) \\
(2) & \forall x \forall y(x \neq y \Rightarrow S(x) \neq S(y)) \\
(3) & \forall x(x + 0 = x) \\
(4) & \forall x \forall y(x + S(y) = S(x + y)) \\
(5) & \forall x(x \cdot 0 = 0) \\
(6) & \forall x \forall y(x \cdot S(y) = x \cdot y + x) \\
(7)_A & \forall y_1 \cdots \forall y_n[(A(0, y_1, \ldots, y_n) \wedge \forall x(A(x, y_1, \ldots, y_n) \\
& \qquad \Rightarrow A(S(x), y_1 \ldots y_n))) \Rightarrow \forall x A(x, y_1, \ldots, y_n)]
\end{array}
$$

for any formula $A(x, y_1, \ldots, y_n)$. (So this is an infinite set of axioms.)

Practically every known fact of number theory is a logical consequence of these axioms. However this set of axioms is not sufficient as a set of axioms for $\mathrm{Th}(\mathcal{N})$, i.e., there are some sentences A true in $\mathcal{N}$ which cannot be derived logically from these axioms. In fact, *there is no way to find a reasonable set of axioms for* $\mathrm{Th}(\mathcal{N})$ (contrast this with the previous examples 2.6.7 and 2.6.8). This is a consequence of the celebrated *Gödel First Incompleteness Theorem.*

Example 2.6.10 Consider the language $L = \{\in\}$, where $\in$ is a binary relation symbol. This is called the *language of set theory* and its intended meaning is to view $\in$ as representing membership of sets and the variables are ranging over all sets. One can formulate in this language a list of axioms called the *Zermelo-Fraenkel Axioms with the Axiom of Choice* (ZFC), from which one can logically derive practically all of the present day ordinary mathematics, which, as it is well understood today, can be founded on the theory of sets. Here are a few of the ZFC axioms:

$$
\begin{array}{lll}
(1) & \forall x \forall y(x = y \Leftrightarrow \forall z(z \in x \Leftrightarrow z \in y)) & \text{(Extensionality axiom)} \\
(2) & \forall x \forall y \exists z \forall w(w \in z \Leftrightarrow w = x \vee w = y) & \text{(Pairing axiom)} \\
(3) & \forall x \exists y \forall z(z \in y \Leftrightarrow \forall t(t \in z \Rightarrow t \in x)) & \text{(Powerset axiom)} \\
\vdots & &
\end{array}
$$

Thus the theory Con(ZFC) can be viewed as a global theory encompassing most present day mathematics. (But not quite all – by the same Gödel Incompleteness Theorem no reasonable axiomatic theory can be *complete* in the sense that there will always be, for any fixed reasonable set of axioms S, a sentence A such that $S \not\models A$ and $S \not\models \neg A$, provided S is powerful enough to include some very elementary number theory. *Reasonable* here means that there is a algorithm to decide whether a sentence belongs to S or not.)

2.7 A Proof System for First-Order Logic

Just as we did for propositional logic in section 1.11, we will now discuss a proof system for first-order logic, i.e. a way to write formal proofs of statements from axioms and rules of inference. Our main goal, as before, is to prove the *Gödel Completeness Theorem*:

$$S \vdash A \text{ iff } S \models, A.$$

but we will not get to this until section 2.8.

2.7.A Formal Proofs

As in section 1.11 it will be convenient to consider as basic symbols in the language of first-order logic the following

$$\neg, \Rightarrow,), (, x_1, x_2, \ldots, =, \forall$$

and view $(A \wedge B)$ as an abbreviation of $\neg(A \Rightarrow \neg B)$, $(A \vee B)$ as an abbreviation of $(\neg A \Rightarrow B)$, $(A \Leftrightarrow B)$ as an abbreviation of $\neg((A \Rightarrow B) \Rightarrow \neg(B \Rightarrow A))$ and

$$\exists x A \text{ as an abbreviation of } \neg\forall x \neg A.$$

Definition 2.7.1 If A is a formula, then we call any formula of the form

$$\forall y_1 \forall y_2 \ldots \forall y_n A$$

a *generalization* of A.

Example 2.7.2 $\forall x \forall y (P(x) \Rightarrow \exists z Q(x, y, z))$ is a generalization of $(P(x) \Rightarrow \exists z Q(x, y, z))$.

We will now describe a Hilbert-type proof system for first-order logic (this particular system is essentially coming from H. Enderton, *A Mathematical Introduction to Logic*). At this point, we fix a first-order language L and consider only formulas in L from now on.

A *(logical) axiom* is any formula which is a *generalization* of a formula of the following form:

(a) (i) $A \Rightarrow (B \Rightarrow A)$

(ii) $((A \Rightarrow (B \Rightarrow C)) \Rightarrow ((A \Rightarrow B) \Rightarrow (A \Rightarrow C))$

(iii) $((\neg B \Rightarrow \neg A) \Rightarrow ((\neg B \Rightarrow A) \Rightarrow B))$

(b) $(\forall x(A \Rightarrow B) \Rightarrow (\forall x A \Rightarrow \forall x B))$

(c) $(A \Rightarrow \forall x A)$, provided x is not free in A

(d) $\forall x A \Rightarrow A[x/t]$, provided t is a term *substitutable* for x in A, where this proviso will be explained shortly.

(e) (i) $x = x$

(ii) $(x = y \wedge y = z) \Rightarrow (x = z)$

(iii) $x = y \Rightarrow y = x$

(iv) $(y_1 = z_1 \wedge \cdots \wedge y_n = z_n) \Rightarrow (f(y_1, \ldots, y_n) = f(z_1, \ldots, z_n))$

(v) $(y_1 = z_1 \wedge \cdots \wedge y_m = z_m) \Rightarrow (R(y_1, \ldots, y_m) \Leftrightarrow R(z_1, \ldots, z_m))$

(Here A, B, C are arbitrary formulas, $x, y, z, y_1, \ldots y_n, z_1, \ldots z_n$ arbitrary variables, f arbitrary n-ary function symbols and R arbitrary m-ary relation symbols.)

We also have a rule of inference, namely *modus ponens* (MP).

From A, $(A \Rightarrow B)$ derive B.

Definition 2.7.3 If S is any set of formulas, a *formal proof* from S is a sequence $A_1, \ldots, A_n$ of formulas such that each A_i is either a logical axiom, belongs to S or comes by applying modus ponens to some A_j, A_k with $j, k < i$. We call this sequence a *formal proof of* A_n *from* S.

If there is a formal proof of a formula A from S, we say that A is a *formal theorem* of S and write

$$S \vdash A.$$

If $S = \emptyset$, we just write

$$\vdash A$$

and call A a *formal theorem*. The same remarks as in section 1.11 apply here too. If we want to indicate the language L we are using, we explicitly write $S \vdash_L A$.

Before discussing examples of formal proofs, we have to explain the notion of substitutability. Recall that if A is a formula, x a variable and t a term, then $A[x/t]$ is the result of substituting every *free* occurrence of x in A by t. The question is: Is $\forall x A \Rightarrow A[x/t]$ logically valid?

Example 2.7.4 Consider

$$A : \exists y(x \neq y)$$
$$t : y.$$

Then $A[x/y]$ is the formula $\exists y(y \neq y)$, but

$$\forall x \exists y(x \neq y) \Rightarrow \exists y(y \neq y)$$

is clearly not valid.

The problem was that x was in the scope of $\exists y$ and we substituted x by y, which completely changed the meaning of this formula.

Definition 2.7.5 We say that t is *substitutable* for x in A if no *free* occurrence of x in A is within the scope of a quantifier $\forall z$ or $\exists z$, with z a variable occurring in t.

Examples 2.7.6

(i) x is substitutable for itself.

(ii) If t is a *closed* term, i.e., a term with no variables, then t is substitutable for any x.

(iii) If A has no quantifiers, *any* t is substitutable for x.

(iv) $f(x)$ is *not* substitutable for z in $\forall x(R(x, z) \Rightarrow \forall y Q(y))$.

(v) $h(z)$ is substitutable for x in $\forall y P(x, y)$.

So we restrict (d) only to the case t is substitutable for x in A. Then it is not hard to see that every formula of the form (d) is valid. Notice that, by example (i) above the formulas $\forall x A \Rightarrow A$ are in (d).

2.7.B Examples of Formal Proofs

Example 2.7.7 First notice that *if* $\vdash A$, *then* $\vdash \forall x A$ *for any variable* x (so $\vdash B$, where B is any generalization of A). To see this, let $A_1, \ldots, A_n = A$ be a formal proof of A. We claim that $\vdash \forall x A_i$, for $i = 1, \ldots, n$, so $\vdash \forall x A_n$, i.e., $\vdash \forall x A$. We can prove this by induction on $i = 1, \ldots, n$. If $i = 1$, then A_1 is a logical axiom and thus, by definition, so is $\forall x A_1$, so $\vdash \forall x A_1$. Assume this has been proved for all $j < i$ and consider A_i. If A_i is a logical axiom, we are done as in the case $i = 1$. Otherwise A_i comes by MP from A_j, A_k with $j, k < i$, i.e., A_k is of the form $A_j \Rightarrow A_i$. By induction hypothesis, $\vdash \forall x A_j$, $\vdash \forall x(A_j \Rightarrow A_i)$, so, since $\vdash \forall x(A_j \Rightarrow A_i) \Rightarrow (\forall x A_j \Rightarrow \forall x A_i)$ (by (b)), using MP twice, we see that $\vdash \forall x A_i$.

Example 2.7.8 If A is an instance of a tautology, then $\vdash A$, using the completeness theorem for propositional logic. So, using also example 2.7.7, we see that $\vdash \forall y_1 \ldots \forall y_n A$, for any $y_1, \ldots, y_n$ and any instance of a tautology A, i.e., any generalization of an instance of a tautology is a formal theorem, so we can use it freely in a formal proof.

Example 2.7.9 $\vdash \forall x(P(x) \Rightarrow \exists y P(y))$ or, recalling our abbreviations,

$$\vdash \forall x(P(x) \Rightarrow \neg\forall y \neg P(y)).$$

Here is why:

1. $\vdash \forall x[(\forall y \neg P(y) \Rightarrow \neg P(x)) \Rightarrow (P(x) \Rightarrow \neg\forall y \neg P(y))]$
   (This is a generalization of the instance of the tautology

   $$(A \Rightarrow \neg B) \Rightarrow (B \Rightarrow \neg A),$$

   with $A : \forall y \neg P(y),\ B : P(x)$.) For simplicity, abbreviate

   $$C : (\forall y \neg P(y) \Rightarrow \neg P(x))$$
   $$D : (P(x) \Rightarrow \neg\forall y \neg(P(y)))$$

2. $\vdash \forall x(C \Rightarrow D) \Rightarrow (\forall x C \Rightarrow \forall x D)$ (logical axiom (b))
3. $\vdash \forall x C \Rightarrow \forall x D$ (MP 1, 2)
4. $\vdash \forall x(\forall y \neg P(y) \Rightarrow \neg P(x))$ (logical axiom (d))
   (noticing that x is substitutable for y in $\neg P(y)$)
5. $\vdash \forall x D$ (MP 3, 4)
   (i.e., $\vdash \forall x(P(x) \Rightarrow \neg\forall y \neg P(y))$)

2.7.C Metatheorems

As in the case of propositional logic, we will prove a few metatheorems, which again formally correspond to common proof techniques.

Proposition 2.7.10 (Tautology Theorem) *If $S \vdash A_1, \ldots, S \vdash A_n$ and the set $\{A_1, \ldots, A_n\}$ tautologically implies B, i.e., $(A_1 \Rightarrow (A_2 \Rightarrow \ldots \Rightarrow (A_n \Rightarrow B) \ldots))$ is an instance of a tautology, then $S \vdash B$.*

Proof. We have $\vdash (A_1 \Rightarrow \ldots \Rightarrow (A_n \Rightarrow B) \ldots))$, so, since $S \vdash A_1, \ldots, S \vdash A_n$, by applying MP several times we get $S \vdash B$. $\dashv$

Proposition 2.7.11 (Deduction Theorem)

$$S \cup \{A\} \vdash B \text{ iff } S \vdash (A \Rightarrow B)$$

Proof. Exactly as in propositional logic. $\dashv$

Proposition 2.7.12 (Proof by Contradiction) *If $S \cup \{A\}$ is formally inconsistent, then $S \vdash \neg A$.*

Proof. Exactly as in propositional logic. $\dashv$

Example 2.7.13 $\vdash \exists x\,(x = x)$. We leave the verification as an exercise. It may be easier by using also some of the metatheorems that follow.

Proposition 2.7.14 (Proof by Contrapositive) *If $S \cup \{A\} \vdash \neg B$, then $S \cup \{B\} \vdash \neg A$.*

Proof. Exactly as in propositional logic. $\dashv$

Proposition 2.7.15 (Generalization Theorem) *If $S \vdash A$ and x is not free in any formula in S, then $S \vdash \forall x A$.*

Proof. By induction on proofs of A from S.

Basis.

(i) A is a logical axiom. Then $\forall x A$ is also a logical axiom, so clearly $S \vdash \forall x A$.

(ii) A is in S. Then, by hypothesis, x is not free in A. So $A \Rightarrow \forall x A$ is a logical axiom. By MP then, $S \vdash \forall x A$.

Induction Step. Assume that $S \vdash \forall x A$, $S \vdash \forall x(A \Rightarrow B)$ in order to show that $S \vdash \forall x B$. Since $\forall x(A \Rightarrow B) \Rightarrow (\forall x A \Rightarrow \forall x B)$ is a logical axiom, this follows by MP applied twice. $\dashv$

Remark. The assumption that x is not free in any formula in S is necessary, as the following example shows:

$$P(x) \vdash P(x)$$

but

$$P(x) \nvdash \forall x P(x).$$

(Otherwise, by the easy half of Gödel's Theorem—the soundedness property $S \vdash A$ implies $S \models A$,—we would have $P(x) \models \forall x P(x)$, which is easily false.)

Example 2.7.16 $\forall x \forall y A \vdash \forall y \forall x A$.

Proof. By Generalization, it is enough to show

$$\forall x \forall y A \vdash \forall x A,$$

and by Generalization once again, it is enough to show

$$\forall x \forall y A \vdash A.$$

Now

$$\vdash \forall x \forall y A \Rightarrow \forall y A$$

(logical axiom (d)) and

$$\vdash \forall y A \Rightarrow A$$

(logical axiom (d)), so by MP twice,

$$\forall x \forall y \vdash A.$$

Example 2.7.17 $\vdash \exists x \forall y A \Rightarrow \forall y \exists x A$.

Proof. By the Deduction Theorem, it is enough to show that

$$\exists x \forall y A \vdash \forall y \exists x A$$

or, by Generalization,

$$\exists x \forall y A \vdash \exists x A$$

or, recalling our abbreviations,

$$\neg \forall x \neg \forall y A \vdash \neg \forall x \neg A$$

or, by Proof by Contrapositive and the Tautology Theorem,

$$\forall x \neg A \vdash \forall x \neg \forall y A.$$

Again by Generalization enough to prove

$$\forall x \neg A \vdash \neg \forall y A$$

or by Proof by Contradiction enough to prove that

$$S = \{\forall x \neg A,\ \forall y A\}$$

is formally inconsistent.

But $\forall x \neg A \Rightarrow \neg A,\ \forall y A \Rightarrow A$ are both logical axioms (d), so by MP

$$S \vdash A,\ S \vdash \neg A$$

and we are done.

Example 2.7.18 $\vdash (A \Rightarrow \forall x B) \Leftrightarrow \forall x (A \Rightarrow B)$, provided x is not free in A.

Proof. It is enough to show (by the Tautology Theorem):

$$\vdash (A \Rightarrow \forall x B) \Rightarrow \forall x (A \Rightarrow B) \qquad (*)$$
$$\vdash \forall x (A \Rightarrow B) \Rightarrow (A \Rightarrow \forall x B) \qquad (**)$$

$(*)$: By the Deduction Theorem enough to show that

$$A \Rightarrow \forall x B \vdash \forall x (A \Rightarrow B),$$

so by Generalization enough to show that

$$A \Rightarrow \forall x B \vdash A \Rightarrow B,$$

or by the Deduction Theorem

$$A, A \Rightarrow \forall x B \vdash B$$

which is clear by MP and the fact that $\forall x B \Rightarrow B$ is a logical axiom (d).

($**$): By the Deduction Theorem, it is enough to show that

$$\forall x(A \Rightarrow B),\ A \vdash \forall x B$$

But $\forall x(A \Rightarrow B) \Rightarrow (\forall x A \Rightarrow \forall x B)$ is a logical axiom (d) and so is $A \Rightarrow \forall x A$, so this is clear by MP.

Proposition 2.7.19 (Generalization on Constants) *Let L be a first-order language, S a set of formulas in L, $A(x)$ a formula in L and c a constant symbol not in L. Then if $S \vdash_{L\cup\{c\}} A[x/c]$, we have $S \vdash_L \forall x A$.*

Proof. Since $S \vdash A[x/c]$, let $S_0 \subseteq S$ be a finite subset of S such that $S_0 \vdash A[x/c]$. Say

$$A_1, A_2, \ldots, A_k = A[x/c]$$

be a proof of $A[x/c]$ from S_0 (in $L\cup\{c\}$). Let $A'_1, \ldots, A'_k$ result by substituting c by y in $A_1, \ldots, A_k$, where y is a variable not occurring in $S_0, A_1, \ldots, A_k, A$. Then it can be easily checked by induction that $A'_1, \ldots, A'_k = A[x/y]$ is a proof from S_0 (in L). So

$$S_0 \vdash A[x/y]$$

Since y does not occur in S_0, we have by Generalization that

$$S_0 \vdash \forall y A[x/y].$$

But clearly x is substitutable for y in $A[x/y]$ and

$$A[x/y][y/x] = A,$$

so

$$\forall y A[x/y] \Rightarrow A$$

is a logical axiom, thus

$$\forall y A[x/y] \vdash A$$

(by MP), so by Generalization again

$$\forall y A[x/y] \vdash \forall x A,$$

and by MP

$$S_0 \vdash \forall x A,$$

so

$$S \vdash \forall x A.$$

$\dashv$

Example 2.7.20 $\forall x \forall y R(x, y) \vdash \forall y \forall x R(x, y)$ (a different argument).

Proof. Enough to show

$$\forall x \forall y R(x, y) \vdash \forall x R(x, d),$$

where d is a constant symbol, or again

$$\forall x \forall y R(x, y) \vdash R(c, d),$$

where c is a constant symbol different from d.

But $\forall x \forall y R(x, y) \Rightarrow \forall y R(c, y)$ is a logical axiom (d) and $\forall y R(c, y) \Rightarrow R(c, d)$ is a logical axiom (d), so by MP twice, $\forall x \forall y R(x, y) \vdash R(c, d)$.

2.8 The Gödel Completeness Theorem

Kurt Gödel was an Austrian logician, widely considered one of the most important and influential logicians of all time. He was born in 1906 in Brno, then part of the Austro-Hungarian empire, and now a Czech city. Gödel died in Princeton in 1978.

Gödel's work in mathematical logic produced three celebrated results: First, the *completeness theorem*, whose proof we present in this section: First-order logic is complete in the sense that a theory proves a formula if and only if all models of the theory are also models of the formula. Second, the *incompleteness theorems*, published in 1931, one year after finishing his doctorate at the University of Vienna. The incompleteness results state that if a theory is "understandable enough" (meaning that its axioms can be generated algorithmically) and powerful enough to prove a small fragment of Peano arithmetic, then either it is inconsistent or else there are true (arithmetic) sentences that it cannot prove. In fact, if it proves enough of Peano arithmetic, then statements about certain formulas being provable

from the theory can be represented by statement about numbers, and using this representation one can write a sentence Con that states that the theory is consistent. Then, either the theory is inconsistent, or else Con itself is not provable in the theory. This result destroyed a program initiated by Hilbert trying to show, by self-contained mathematical methods, the consistency of mathematics. Gödel's third main result is in set theory, where he introduced the universe L of *constructible sets* and proved that L is a model of set theory, together with the axiom of choice and the generalized continuum hypothesis.

Gödel also made contributions to intuitionistic logic and relativity theory. In 1933, he met Einstein while travelling in the US. The two of them became good friends and after his move to Princeton, Einstein was one of the very few people with whom Gödel kept personal contact.

He moved to the States in 1940 because his former association with Jewish members of the so called Vienna circle (an influential group of mathematicians and philosophers of which Alfred Tarski, the logician who introduced the notion of truth ($\models$), was also a member) made it difficult for him to keep his position in Vienna under the Nazi regime.

Gödel suffered from delusions that eventually developed into an acute case of paranoia. He believed conspirators tried to assassinate him with poison gas and eventually thought that they were trying to poison his food. In 1977, due to illness, his wife could no longer help him with his meals, and he refused to eat any food at all, starving himself to death.

Theorem 2.8.1 (Gödel) *Let L be a first-order language and S a set of formulas in L, A a formula in L. Then*

$$S \models A \textit{ iff } S \vdash A.$$

Proof. The direction

$$S \vdash A \text{ implies } S \models A$$

is easy and is called the *Soundedness Property* of the proof system. It follows immediately from the fact that every logical axiom is logically valid and that MP preserves validity.

We will now prove the *Completeness Property* of the proof system, i.e.,

$$S \models A \text{ implies } S \vdash A.$$

As is the case of propositional logic (section 1.11.C), it is enough to show the following:

If a set of formulas S is formally consistent, then it is satisfiable. $(*)$

(As before, we call S *formally inconsistent* if for some formula A we have $S \vdash A$ and $S \vdash \neg A$, and otherwise we call S *formally consistent*.)

For simplicity, we will give in detail the proof of $(*)$ in case S is a set of *sentences*. (The case of arbitrary formulas follows from this by a little extra argument.) So we have a formally consistent set of sentences S and we have to find a model of S. First, we need some terminology:

Definition 2.8.2 A *formal theory* is a set of sentences T closed under formal provability, i.e. $T \vdash A$ (A a sentence) implies $A \in T$.

Definition 2.8.3 A set of sentences T is *complete* if for any sentence A we have either $A \in T$ or $\neg A \in T$.

It is then clear that if T is formally consistent and complete, then for any sentence A, exactly one of A and $\neg A$ belongs to T.

Definition 2.8.4 A *Henkin theory* is a formal theory which has the following extra property:

"For any formula $A(x)$ there is a constant symbol $c = c_A$ such that $\neg\forall x A(x) \Rightarrow \neg A[x/c]$ is in T."

We call c_A a *Henkin witness* for $\neg\forall x A(x)$.

We prove two lemmas from which $(*)$ follows immediately.

Lemma 2.8.5 *Suppose L' is a first-order language, T' a formally consistent, complete Henkin theory in L'. Then T' has a model $\mathcal{M}$.*

Lemma 2.8.6 *Let L be a first-order language and S a formally consistent set of sentences in L. Then there is a first-order language $L' \supseteq L$ obtained by adding some constant symbols to L, and a formally consistent, complete, Henkin theory T' in L' such that $T' \supseteq S$.*

Proof of of Lemma 2.8.5. Let

$$A = \{t : t \text{ is a closed term in } L'\}.$$

Define an equivalence relation $\sim$ on A as follows:

$$s \sim t \text{ iff } s = t \in T'$$

($s = t$ is a sentence in L). We have first to check that this is indeed an equivalence relation:

(i) $s \sim s$.

This means that $s = s \in T'$. But $\forall x(x = x) \Rightarrow s = s$ is a logical axiom (d) and $\forall x(x = x)$ is a logical axiom (e), so, by MP, $\vdash s = s$, so $T' \vdash s = s$, thus $s = s \in T'$.

(ii) $s \sim t$ implies $t \sim s$.

We have that $s = t \in T'$. Now $\forall x \forall y(x = y \Rightarrow y = x)$ is a logical axiom (e) and $\vdash \forall x \forall y(x = y \Rightarrow y = x) \Rightarrow (s = t \Rightarrow t = s)$, so by MP, $\vdash (s = t) \Rightarrow (t = s)$ so as $s = t \in T'$, $T' \vdash t = s$, thus $t = s \in T'$ (since T' is a formal theory), i.e., $t \sim s$.

(iii) $s \sim t$ and $t \sim u$ imply $s \sim u$.

Again $\forall x \forall y \forall z((x = y \wedge y = z) \Rightarrow x = z)$ is a logical axiom (e) and $\vdash \forall x \forall y \forall x((x = y \wedge y = z) \Rightarrow x = z) \Rightarrow ((s = t \wedge t = u) \Rightarrow s = u)$ by using three times logical axiom (d) and MP, so , by MP, $\vdash (s = t \wedge t = u) \Rightarrow s = u$, so $T \vdash (s = t \wedge t = u) \Rightarrow s = u$ and since $s = t$, $t = u \in T'$ it follows that $T' \vdash s = u$, so $s = u \in T'$.

Denote by

$$[s] = \{t : t \sim s\}$$

the equivalence class of $s \in A$, and by

$$M = \{[s] : s \text{ a closed term}\}$$

the set of all equivalence classes (the quotient set of A modulo $\sim$). This will be the underlying universe of the model of T'. We will now define the interpretations of the relation and function symbols of L':

Let f be an n-ary function symbol. Define its interpretation $f^{\mathcal{M}}$ by

$$f^{\mathcal{M}}([s_1], \ldots, [s_n]) = [f(s_1, \ldots, s_n)].$$

One has to show that this is well-defined, i.e., if $s_1 \sim t_1, \ldots, s_n \sim t_n$, then

$$f(s_1, \ldots, s_n) \sim f(t_1, \ldots, t_n).$$

This follows easily from the fact that

$$\vdash (s_1 = t_1 \wedge \cdots \wedge s_n = t_n) \Rightarrow f(s_1, \ldots, s_n) = f(t_1, \ldots, t_n),$$

which can be proved by arguments similar to those before (using logical axioms (e)).

As a special case, if c is a constant symbol we have

$$c^{\mathcal{M}} = [c].$$

For further reference, note also that for any closed term s we have

$$s^{\mathcal{M}} = [s].$$

(This can be easily proved by induction on the construction of s.)

Now let R be an m-ary relation symbol. Define its interpretation $R^{\mathcal{M}}$ by

$$R^{\mathcal{M}}([s_1], \ldots, [s_m]) \text{ iff } R(s_1, \ldots, s_m) \in T'.$$

Again we have to show that this is well-defined, i.e. that $s_1 \sim t_1, \ldots, s_m \sim t_m$ imply that

$$R(s_1, \ldots, s_m) \in T' \text{ iff } R(t_1, \ldots, t_m) \in T'.$$

This follows from the fact that

$$\vdash (s_1 = t_m \wedge \cdots \wedge s_m = t_m) \Rightarrow [R(s_1, \ldots, s_m) \Leftrightarrow R(t_1, \ldots, t_m)].$$

We have now completed the description of the structure $\mathcal{M}$. It remains to show that

$$\mathcal{M} \models T'.$$

Actually we will show that for each sentence A of L' we have

$$\mathcal{M} \models A \text{ iff } A \in T'. \qquad (**)$$

For each sentence A, let

$$N(A) = \text{ the total number of } \neg, \Rightarrow, \forall \text{ in } A.$$

We will prove $(**)$ by induction on $N(A)$.

Basis. $N(A) = 0$. Then A is atomic, i.e., $s = t$ or $R(s_1, \ldots, s_m)$ for closed terms $s, t, s_1, \ldots, s_m$. For the case of $s = t$ we have

$$\begin{aligned} \mathcal{M} \models s = t \text{ iff } s^{\mathcal{M}} = t^{\mathcal{M}} &\text{ iff } [s] = [t] \\ &\text{ iff } s \sim t \text{ iff } s = t \in T'. \end{aligned}$$

For the case of $R(s_1, \ldots, s_m)$ we have

$$\begin{aligned} \mathcal{M} \models R(s_1, \ldots, s_m) &\text{ iff } R^{\mathcal{M}}(s_1^{\mathcal{M}}, \ldots, s_m^{\mathcal{M}}) \\ &\text{ iff } R^{\mathcal{M}}([s_1], \ldots, [s_m]) \\ &\text{ iff } R(s_1, \ldots, s_m) \in T'. \end{aligned}$$

Induction step. Assume $(**)$ has been proved for all sentences A' with $N(A') \leq n$. Say A is such that $N(A) = n + 1$.

Case 1. A is of the form $\neg B$. Then $N(B) = n$, so by induction hypothesis

$$\mathcal{M} \models B \text{ iff } B \in T',$$

thus

$$\begin{aligned} \mathcal{M} \models A \text{ iff } M \not\models B &\text{ iff } B \notin T' \\ &\text{ iff } \neg B \in T' \text{ (as } T' \text{ is formally consistent and complete)} \\ &\text{ iff } A \in T'. \end{aligned}$$

Case 2. A is of the form $B \Rightarrow C$. Then $N(B), N(C) \leq n$, so by induction hypothesis

$$\begin{aligned} \mathcal{M} \models B \text{ iff } B \in T' \\ \mathcal{M} \models C \text{ iff } C \in T'. \end{aligned}$$

Thus we have

$$\begin{aligned} \mathcal{M} \models A \text{ iff } \mathcal{M} \not\models B \text{ or } \mathcal{M} \models C \\ &\text{ iff } B \notin T' \text{ or } C \in T' \\ &\text{ iff } \neg B \in T' \text{ or } C \in T' \\ &\text{ iff } B \Rightarrow C \in T' \\ &\text{ iff } A \in T' \end{aligned} \tag{+}$$

To see (+), notice that if $\neg B \in T'$, then since $\neg B \Rightarrow (B \Rightarrow C)$ is an instance of a tautology, so $\vdash \neg B \Rightarrow (B \Rightarrow C)$, we have $T' \vdash B \Rightarrow C$, so $B \Rightarrow C \in T'$. Similarly if $C \in T'$, since $C \Rightarrow (B \Rightarrow C)$ is a logical axiom (a). Conversely, if $B \Rightarrow C \in T'$ but $\neg B \notin T'$ and $C \notin T'$, then $B \in T'$ and $\neg C \in T'$, so by MP $C \in T'$ and $\neg C \in T'$, which is impossible, as T' is formally consistent.

Case 3. A is of the form $\forall x B(x)$. Then $N(B) = n$, so also $N(B[x/t]) = n$ for any closed term t, thus by induction hypothesis:

$$\mathcal{M} \models B[x/t] \text{ iff } B[x/t] \in T'.$$

We will show that

$$\mathcal{M} \models \forall x B(x) \text{ iff } \forall x B(x) \in T'.$$

If $\mathcal{M} \models \forall x B(x)$, then for any closed term t, we have

$$\mathcal{M} \models B[x/t]$$

(notice that $\forall x B(x) \Rightarrow B[x/t]$ is logically valid). So $B[x/t] \in T'$, for all closed terms t. If now $\forall x B(x) \notin T$, towards a contradiction, then $\neg \forall x B(x) \in T'$, so, as T' is a Henkin theory, for some constant symbol c we have $\neg B[x/c] \in T'$, a contradiction.

Conversely, assume $\forall x B(x) \in T'$. Then as $\forall x B(x) \Rightarrow B[x/t]$ is a logical axiom (d) for every closed term t, we have $B[x/t] \in T'$, so by induction hypothesis $\mathcal{M} \models B[x/t]$, i.e., $\mathcal{M} \models B[x \mapsto t^{\mathcal{M}}]$ or $\mathcal{M} \models B[x \mapsto [t]]$. But every element of $\mathcal{M}$ has the form $[t]$ for some such t, so $\mathcal{M} \models B[x \mapsto a]$ for every $a \in M$, i.e., $\mathcal{M} \models \forall x B(x)$. Lemma 2.8.5 $\dashv$

Proof of Lemma 2.8.6. Again for simplicity in exposition, we will assume that L is countable, i.e. all non-logical symbols in L can be enumerated in a sequence. It follows that one can enumerate in a sequence all formulas of L and so all sentences of L.

The proof of Lemma 2.8.6 will be done in two steps.

1st Step. We will first define a first-order language $L' \supseteq L$ obtained by adding to L a countable set of constant symbols, and a formally consistent Henkin theory S' in L' such that $S' \supseteq S$. Notice that L' is still countable.

2nd Step. We will find a formally consistent and complete theory $T' \supseteq S'$ in the language L'. Then notice that T' is also a Henkin theory (by definition), so we are done.

The second step can be done exactly as in the case of propositional logic (see the proof of sublemma 1.11.11), so we won't repeat it.

Thus it is enough to do the 1st step: Define recursively on n, a first order language L_n and a set of sentences S_n of L_n such that

$$L = L_0 \subseteq L_1 \subseteq L_2 \subseteq \dots$$

$$S = S_0 \subseteq S_1 \subseteq S_2 \subseteq \dots$$

as follows:

$$L_0 = L,\ S_0 = S$$

Assume L_n, S_n are given. Define then L_{n+1} by adding to L_n one *new* constant symbol (not in L_n and distinct from each other)

$$c = c_{\neg\forall x A(x)}$$

for each sentence of L_n of the form $\neg\forall x A(x)$. Finally obtain S_{n+1} by adding to S_n all the sentences

$$\neg\forall x A(x) \Rightarrow \neg A[x/c],$$

where $A(x)$ is a formula of L_n and c is the constant symbol associated to $\neg\forall x A(x)$.

Put now

$$L' = \bigcup_n L_n,\ S'' = \bigcup_n S_n.$$

Clearly $L' \supseteq L$ is obtained from L by adding to L countably many constant symbols. Let S' be the formal theory generated by S'', i.e.,

$$S' = \{A \text{ a sentence of } L' : S'' \vdash A\}.$$

Clearly $S \subseteq S'' \subseteq S'$, so it is enough to show that S' is a formally consistent Henkin theory.

That it is a Henkin theory is easy: If $A(x)$ is a formula of L', then $A(x)$ is a formula of some L_n, so if $c = c_{\neg\forall x A(x)}$, then c is a constant symbol in L_{n+1}, thus in L', and

$$\neg\forall x A(x) \Rightarrow \neg A[x/c]$$

is in S_{n+1}, thus in S'.

It remains to show that S' is formally consistent or equivalently that S'' is formally consistent, and for that it is enough to show each S_n is formally

consistent. This is done by induction on n. For $n = 0$, it is given as $S_0 = S$. So assume now that S_n is formally consistent. If S_{n+1} is formally inconsistent, towards a contradiction, there are formulas $A_1(y_1), \ldots, A_k(y_k)$ of L_n and new (i.e., not in L_n) distinct constants of L_{n+1}, say $c_1, \ldots, c_k$, such that

$$S_n \cup \{\neg\forall y_i A_i(y_i) \Rightarrow \neg A_i(c_i) : 1 \leq i \leq k\}$$

is formally inconsistent, where $A_i(c_i)$ abbreviates $A_i[y_i/c_i]$. Then by proof by contradiction

$$\underbrace{S_n \cup \{\neg\forall y_i A_i(y_i) \Rightarrow \neg A_i(c_i) : 1 \leq i \leq k-1\}}_{Q} \vdash \neg(\neg\forall y_k A_k(y_k) \Rightarrow \neg A_k(c_k)),$$

so, by using tautologies,

$$Q \vdash \neg\forall y_k A_k(y_k)$$

$$Q \vdash A_k(c_k).$$

But by Generalization on Constants, since c_k does not occur in the formulas in Q, we have that

$$Q \vdash \forall y_k A_k(y_k),$$

so Q is formally inconsistent. Proceeding successively in a similar fashion we eliminate one-by-one all $\neg\forall y_i A_i(y_i) \Rightarrow \neg A_i(c_i)$ for $i = k, \ldots, 1$, so we get that S_n is formally inconsistent, a contradiction. Lemma 2.8.6 $\dashv$

2.9 The Compactness Theorem

As an immediate corollary of the Gödel Completeness Theorem for First-Order Logic, we obtain the compactness theorem for first-order logic:

Theorem 2.9.1 (The Compactness Theorem, I) *Let L be a first-order language, S a set of sentences in L and A a sentence in L. Then if $S \models A$, there is a finite subset $S_0 \subseteq S$ such that $S_0 \models A$.*

Theorem 2.9.2 (The Compactness Theorem, II) *Let L be a first-order language and S a set of sentences of L. If every finite subset $S_0 \subseteq S$ has a model, S has a model.*

We will discuss some applications of this Compactness Theorem.

2.9.A Finiteness and Infinity

Theorem 2.9.3 *Let S be a set of sentences in L. If S has arbitrarily large finite models, then S has an infinite model.*

Proof. Let λ_n be the sentence

$$\lambda_n : \exists x_1 \exists x_2 \ldots \exists x_n (\bigwedge_{i<j} x_i \neq x_j).$$

Then for any structure $\mathcal{M} = \langle M, - \rangle$,

$$\mathcal{M} \models \lambda_n \text{ iff } \mathrm{card}(M) \geq n.$$

Let $T = S \cup \{\lambda_1, \lambda_2, \ldots\}$. Our hypothesis implies that every finite subset of T has a model. So by the Compactness Theorem, T has a model $\mathcal{M}$. Then $\mathcal{M} \models S$ and $\mathcal{M} \models \lambda_n$ for each n, so $\mathcal{M}$ is infinite (i.e., M is infinite). $\dashv$

Corollary 2.9.4 *"Finiteness" cannot be expressed in first-order logic, i.e., for any first-order language L there is no set of sentences S in L such that for any structure $\mathcal{M}$ of L*

$$\mathcal{M} \models S \quad \textit{iff } \mathcal{M} \quad \textit{is finite.}$$

If $\Lambda_\infty = \{\lambda_1, \lambda_2, \ldots\}$, then clearly

$$\mathcal{M} \models \Lambda_\infty \text{ iff } \mathcal{M} \text{ is infinite,}$$

so "infinity" can be characterized by an (infinite) set of sentences in first-order logic, namely Λ_∞. Can it be characterized by a finite set of sentences, or equivalently by a single sentence A? If it could, then we would have

$$\mathcal{M} \models A \text{ iff } \mathcal{M} \text{ is infinite,}$$

so

$$\mathcal{M} \models \neg A \text{ iff } \mathcal{M} \text{ is finite,}$$

contradicting the previous corollary. It follows that if

$$T = \mathrm{Con}(\Lambda_\infty),$$

then T is *not* finitely axiomatizable.

Consider the language $L = \{<\}$. Can we find a set of sentences S in L such that

$$\mathcal{M} = \langle M, <^{\mathcal{M}} \rangle \models S \text{ iff } \mathcal{M} \text{ is a well ordering?}$$

We can see that the answer is negative as follows: Assume such an S existed, towards a contradiction. Consider the language $L' = \{<, c_0, c_1, c_2, \dots\}$ obtained by adding to L an infinite list of constant symbols $c_0, c_1, \dots$ Let

$$T = S \cup \{c_1 < c_0, c_2 < c_1, c_3 < c_2, \dots\}.$$

Then we claim that any finite subset T_0 of T has a model. This is because T_0 is contained in $S \cup \{c_1 < c_0, c_2 < c_1, \dots, c_{n_0+1} < c_{n_0}\}$ for some large enough n_0. A model of T_0 is then

$$\mathcal{M}_0 = \langle \mathbb{N}, <, c_0^{\mathcal{M}_0}, c_1^{\mathcal{M}_0}, \dots \rangle,$$

where $c_i^{\mathcal{M}} = n_0 + 1 - i$, if $i \leq n_0 + 1$ and $c_i^{\mathcal{M}} = 0$ if $i > n_0 + 1$. So, by the Compactness Theorem, T has a model

$$\mathcal{M} = \langle M, <^{\mathcal{M}}, c_0^{\mathcal{M}}, c_1^{\mathcal{M}}, \dots \rangle.$$

Then let

$$A = \{c_0^{\mathcal{M}}, c_1^{\mathcal{M}}, \dots\} \subseteq M.$$

Since $c_0^{\mathcal{M}} >^{\mathcal{M}} c_1^{\mathcal{M}} >^{\mathcal{M}} c_2^{\mathcal{M}} >^{\mathcal{M}} \dots$, clearly A has no least element (in $<^{\mathcal{M}}$), so it is not a wellordering, contradicting that

$$\langle M, <^{\mathcal{M}} \rangle \models S.$$

2.9.B Non-standard Models of Arithmetic

There are two main types of theories we have discussed. Many, such as the theories of groups, partial orders, ordered fields, and so on, are intended to have many models. The models of the theory of groups are all the objects which we call groups, and so on. From the standpoint of the first-order theory, all we can say about an arbitrary group is that it satisfies certain axioms and everything that follows from them. Of course, there are many statements independent of the axioms, and these may be true or false of individual groups. This is what makes group theory interesting.

The other type of theories are those intended (at least originally) to represent a single model. For example, the theory of arithmetic is constructed

with the structure of the natural numbers $\mathcal{N}$ in mind. A very natural question to ask is, "Do these theories also have multiple models?" For example, are there other structures which "act like $\mathcal{N}$" as far as the first-order theory can tell, but are not in fact the same? The answer is yes, and the consequences are interesting and far-reaching. We start by defining what we are looking for:

Definition 2.9.5 A *model of (complete) arithmetic* is a structure of L_{ar},

$$\mathcal{M} = \langle M, 0^{\mathcal{M}}, S^{\mathcal{M}}, +^{\mathcal{M}}, \cdot^{\mathcal{M}}, <^{\mathcal{M}} \rangle$$

which is a model of $\mathrm{Th}(\mathcal{N})$, i.e. for any sentence A in the language of arithmetic

$$\mathcal{M} \models A \text{ iff } \mathcal{N} \models A.$$

What does a general model of arithmetic look like? Well, to start with, it must contain objects $n^{\mathcal{M}}$ which "look like" the natural numbers $n \in \mathbb{N}$. For any such $n \in \mathbb{N}$, let

$$n^{\mathcal{M}} = \underline{n}^{\mathcal{M}},$$

where $\underline{n}$ is the term in L_{ar} given by:

$$\underline{n} = \underbrace{S(S(S \dots S}_{n \text{ times}}(0) \dots))$$

(so that $\underline{n}^{\mathcal{N}} = n$). Let

$$\mathbb{N}^{\mathcal{M}} = \{0^{\mathcal{M}}, 1^{\mathcal{M}}, \dots\}.$$

Then it is not hard to see that $\mathbb{N}^{\mathcal{M}}$ is closed under $S^{\mathcal{M}}, +^{\mathcal{M}}, \cdot^{\mathcal{M}}$, so it defines a structure

$$\mathcal{N}^{\mathcal{M}} = \langle \mathbb{N}^{\mathcal{M}}, 0^{\mathcal{M}}, S^{\mathcal{M}}|\mathbb{N}^{\mathcal{M}}, +^{\mathcal{M}}|\mathbb{N}^{\mathcal{M}}, \cdot^{\mathcal{M}}|\mathbb{N}^{\mathcal{M}}, <^{\mathcal{M}}|\mathbb{N}^{\mathcal{M}} \rangle,$$

and the map $n \mapsto n^{\mathcal{M}}$ is an isomorphism of $\mathcal{N}$ with $\mathcal{N}^{\mathcal{M}}$. So we can simply identify $\mathcal{N}$ with $\mathcal{N}^{\mathcal{M}}$. We call the elements $n = n^{\mathcal{M}}$ the *standard elements* of $\mathcal{M}$. So we see that any model of arithmetic must contain, at least, an embedded "standard copy" of $\mathcal{N}$.

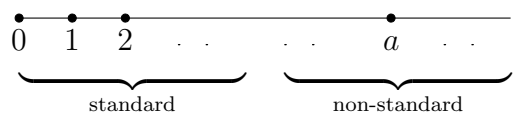
Now what if $\mathcal{M}$ has non-standard elements? What would they look like? They have to be "infinite": that is, we must have

$$n <^{\mathcal{M}} a$$

for all standard n. This is because

$$\mathcal{N} \models \forall x(x < \underline{n} \Rightarrow x = 0 \vee x = \underline{1} \vee \cdots \vee x = \underline{n-1}),$$

so $\mathcal{M} \models \forall x(x < \underline{n} \Rightarrow x = 0 \vee \cdots \vee x = \underline{n-1})$. So if $a \in \mathcal{M}$ is different than each n, then, since $<^{\mathcal{M}}$ is a linear order, if $n <^{\mathcal{M}} a$ fails, then we must have $a <^{\mathcal{M}} n$ (since $a \neq n^{\mathcal{M}}$), so $a = 0$, or $a = 1, \ldots,$ or $a = n-1$, a contradiction. So we have the picture



Do non-standard models of arithmetic exist? That is, models of arithmetic with non-standard elements (equivalently not isomorphic to $\mathcal{N}$)? The answer is yes, as we can see using the Compactness Theorem.

Consider the language

$$L' = \{0, S, +, \cdot, <\} \cup \{c\},$$

where c is a new constant symbol. Let S be the following set of sentences in L':

$$S = \mathrm{Th}(\mathcal{N}) \cup \{\underline{n} < c : n \in \mathbb{N}\}.$$

Then every finite subset S_0 of S has a model. This is because $S_0 \subseteq Th(\mathcal{N}) \cup \{\underline{n} < c : n \leq n_0\}$ for some large enough n_0. Then a model of S_0 is

$$\mathcal{N}_0 = \langle \mathbb{N}, 0, S, +, \cdot, <, c^{\mathcal{N}_0} \rangle$$

where $c^{\mathcal{N}_0} = n_0 + 1$. So, by the Compactness Theorem, S has a model

$$\mathcal{M} = \langle M, 0^{\mathcal{M}}, S^{\mathcal{M}}, +^{\mathcal{M}}, \cdot^{\mathcal{M}}, <^{\mathcal{M}}, c^{\mathcal{M}} \rangle.$$

Then if $a = c^{\mathcal{M}}$ we have

$$\underline{n}^{\mathcal{M}} = n <^{\mathcal{M}} a$$

for all $n \in \mathbb{N}$, i.e. a is non-standard.

Non-standard models such as this one can often be used in many areas of mathematics to simplify proofs and elucidate concepts more clearly. For example there are non-standard models of $\mathcal{R}$ that contain "infinitesimals"—non-standard elements which are positive yet smaller (under $<^{\mathcal{M}}$) than any

(embedded standard copy of a) positive real number—and they can be used to reformulate calculus without using limits. This field of study is called *infinitesimal analysis* or *non-standard analysis*.

Here are some basic examples. Let $A \subseteq \mathbb{R}$ and let $f : A \to \mathbb{R}$. Consider the structure

$$\mathcal{R} = (\mathbb{R}, \mathbb{N}, 0, 1, <, +, -, \times, /, f)$$

where $\mathbb{N}$, $/$ and f are seen as relations: $\mathbb{N}$ is seen as a unary relation, so $n \in \mathbb{N}$ iff $\mathbb{N}(n)$, $/$ is seen as a ternary relation, so $a/b = c$ for $b \neq 0$ iff $/(a, b, c)$ and f is seen as a binary relation, so $f(a) = b$ iff $f(a, b)$, this is simply so we can talk about f even if its domain is not all of $\mathbb{R}$, and similarly for $/$.

A *nonstandard model* of analysis is a structure

$$^*\mathcal{R} = (^*\mathbb{R}, {}^*\mathbb{N}, {}^*0, {}^*1, {}^*<, {}^*+, {}^*-, {}^*\times, {}^*/, {}^*f)$$

together with a proper elementary embedding $j : \mathbb{R} \to {}^*\mathbb{R}$. This means that j is an elementary embedding and that the image of j is not all of $^*\mathbb{R}$. That j is elementary means that for any formula $A(x_1, \ldots, x_n)$ and any reals $r_1, \ldots, r_n$,

$$\mathcal{R} \models A[x_1/r_1, \ldots, x_n/r_n]$$

if and only if

$$^*\mathcal{R} \models A[x_1/j(r_1), \ldots, x_n/j(r_n)].$$

In particular, j is 1-to-1 and $\mathcal{R}$ and $^*\mathcal{R}$ satisfy the same sentences.

Notice that $j(0) = {}^*0$ and $j(1) = {}^*1$; in general, we will write $j(r) = {}^*r$. Notice that $^*(-r) = {}^* - {}^*r$ for $r \in \mathbb{R}$ and $^*(f(r)) = {}^*f({}^*r)$ for $r \in A$.

Let $^*\mathcal{R}$ be a nonstandard model. As before, that $^*\mathcal{R}$ exists is an easy consequence of the compactness theorem. Say that $r \in {}^*\mathbb{R}$ is *finite* iff there is $n \in \mathbb{N}$ such that $^* - n\, {}^* < r\, {}^* < {}^*n$. Say that r is *infinite* otherwise. Say that r is *infinitesimal* iff for all $n \in \mathbb{N}$ such that $n > 0$, $^* - 1/n\, {}^* < r\, {}^* < {}^*1/n$. Write $r \approx s$ iff $r - s$ is infinitesimal. If $r \in {}^*\mathbb{R}, s \in \mathbb{R}$ and $r \approx {}^*s$, say that s is the *standard part* of r and write $^\circ r = s$.

The following claims are left as exercises:

(i) If $r \in {}^*\mathbb{R}$ is finite then its standard part exists, i.e., there is a unique real number s such that $r \approx s$.

(Recall that a set of reals bounded above has a least upper bound.)

(ii) There are infinite elements in ${}^*\mathbb{R}$. There are infinitesimals (other than ^{*0}). There are infinite natural numbers, i.e., infinite numbers N such that ${}^*\mathbb{N}(N)$.

(iii) The function f is continuous iff for all $x \approx y$, $x, y \in \mathrm{dom}({}^*f)$, ${}^*f(x) \approx {}^*f(y)$. Suppose that $\mathrm{dom}(f) = [0,1]$ and that f is continuous. It follows that if ${}^*0 \mathrel{{}^*\!<} r \mathrel{{}^*\!<} {}^*1$ then ${}^*f(r)$ is finite.

(iv) Let N be an infinite natural number. Then

$$\{\, i/N \colon {}^*\mathbb{N}(i) \text{ and } i \mathrel{{}^*\!\leq} N \,\}$$

is dense in $j([0,1]) = \{\, {}^*s \colon s \in [0,1] \,\}$ and for any $s \in [0,1]$ there is an i/N such that ${}^\circ(i/N) = s$. Here, ${}^*\!\leq$ means ${}^*\!<$ or $=$, and $/$ is really ${}^*/$, but we suppress the $*$ to improve readability.

(v) Suppose from now on that $A = [0,1]$, that f is continuous, and that N is an infinite natural number. The variable i will always stand for a number in ${}^*\mathbb{N}$ such that $i \mathrel{{}^*\!\leq} N$. For any such i, ${}^\circ({}^*f(i/N)) = f({}^\circ(i/N))$.

(vi) There is an i_0 such that ${}^*f(i_0/N)$ is maximum among

$$\{\, {}^*f(i/N) \colon {}^*\mathbb{N}(i) \text{ and } i^* \leq N \,\}.$$

It follows that ${}^\circ({}^*f(i_0/N))$ is the maximum of f on $[0,1]$. This shows that a continuous function on a closed interval attains its maximum (and, in particular, is bounded).

(vii) Suppose that $f(0) < 0$ and $f(1) > 0$. Then there is an i such that ${}^*f(i/N)^* \leq {}^*0^* \leq {}^*f((i^* + {}^*1)/N)$. It follows that ${}^\circ(i/N) = {}^\circ((i^* + {}^*1)/N)$ and therefore $f({}^\circ(i/N)) = 0$. This shows the intermediate value theorem.

Chapter 3

Computability and Complexity

3.1 Introduction

Consider a domain D of concrete objects, e.g., numbers, words in an alphabet, finite graphs, etc. We will refer to elements I of D as *instances*.

Definition 3.1.1 Given a set $P \subseteq D$ of instances, the *decision problem* for P is the following question:

> Is there an algorithm (*effective* or *mechanical procedure*) which will tell us, in finitely many steps, for each instance $I \in D$ whether $I \in P$ or $I \notin P$?

Example 3.1.2 Given a formula in propositional logic, is there an algorithm to determine whether it is satisfiable? One can also consider the decision problem where the formulas are now in first-order logic.

Example 3.1.3 Is there an algorithm that, given $n \in \mathbb{N}$ and

$$m \in \{0, 1, \ldots, 9\},$$

tells us whether the n-th digit in the decimal expansion of π is m?

Example 3.1.4 (The Hilbert 10th Problem) Is there an algorithm to verify whether an arbitrary polynomial (in several variables)

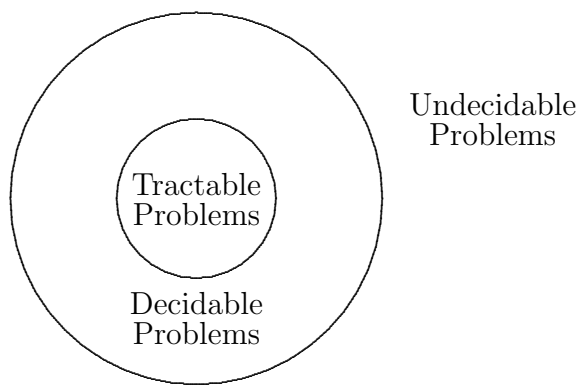
$$\sum a_{i_1 i_2 i_3 \ldots i_n} x_1^{i_1} x_2^{i_2} \ldots x_n^{i_n}$$

with integer coefficients $a_{i_1 i_2 \ldots i_n}$ has an integer solution?

Definition 3.1.5 If such an algorithm exists for the decision problem (given by) P, we will call P *decidable*. Otherwise we call it *undecidable*.

Example 3.1.6 The validity problem for formulas in propositional logic is decidable (use truth tables). The Hilbert 10th Problem is undecidable (Matyasevich, 1970).

Thus, modulo our rather vague definition of "algorithm," we have the first main classification of (decision) problems as decidable or undecidable. Our next goal will be to classify further the decidable problems according to the (time) complexity of their algorithms and distinguish between those for which an efficient (feasible) algorithm is possible, *the tractable problems*, and those for which it is not, the *intractable* problems.



In particular, this will lead to the study of an important class of problems widely believed to be intractable, but for which no proof has yet been found (this is the famous $\mathcal{P} = \mathcal{NP}$ problem).

We will now formalize these concepts and present an introduction to the study of decision problems and their complexity.

3.2 Decision Problems

We can represent concrete finite objects as words (strings) in a finite alphabet.

Definition 3.2.1 An *alphabet* is an arbitrary nonempty set, whose elements we call *symbols*.

We will be mainly dealing with finite alphabets $A = \{\sigma_1, \ldots, \sigma_k\}$.

Definition 3.2.2 A *word* (or *string*) for A is a finite sequence $a_1 a_2 \ldots a_n$, where each a_i is a symbol in A. If $w = a_1 a_2 \ldots a_n$ is a word, then $n = |w|$ is the *length* of w.

We denote by A^n the set of all words of length n from A. By convention we also have the *empty* word θ which has length 0. Thus $A^0 = \{\theta\}$. Finally we let

$$A^* = \bigcup_n A^n$$

be the set of all words from A (including θ).

Examples 3.2.3

(i) Natural numbers can be represented as words using decimal notation and the alphabet $\{0, 1, 2, \ldots, 9\}$, or in the alphabet $\{0, 1\}$ using binary notation, or even in the alphabet $\{1\}$ using unary ("tally") notation.

(ii) A finite graph $G = \langle V, E \rangle$, with set of vertices $V = \{v_1, \ldots, v_n\}$, can be represented by its adjacency matrix $A = (a_{ij}),\ 1 \leq i, j \leq n$, where $a_{ij} \in \{0, 1\}$ and $a_{ij} = 1$ iff $(a_i, a_j) \in E$. This can be in turn represented by the word

$$a_{11} a_{12} \ldots a_{1n} a_{21} \ldots a_{2n} \ldots a_{n1} \ldots a_{nn}$$

in the alphabet $\{0, 1\}$.

Definition 3.2.4 A *decision problem* consists of a finite alphabet A and a set of words $P \subseteq A^*$.

Consider a finite first order language $L = \{f_1, \ldots, f_n;\ R_1, \ldots, R_m\}$. Representing the variable x_n by the string xn, where n is written in binary, we can view every wff in L as a word in the finite alphabet $A = L \cup \{x, 0, 1, \neg, \wedge, \vee, \Rightarrow, \Leftrightarrow, \exists, \forall, (,), =\}$. We let VALIDITY_L be the decision problem corresponding to (A, P), where

$$P = \{S : S \text{ is a sentence in } L \text{ and } \models S\}.$$

3.3 Turing Machines

Now that we have precisely defined decision problems, we will go on to make precise our previously vague concept of an "algorithm." We will do this by presenting a model of computation, using a formalized representation of a computer known as a Turing machine. There are many equivalent formalizations of the notion of "computability," but Turing machines were one of the earliest to be formulated and remain one of the easiest to understand and work with.

As a motivation for the definition which is to follow, consider a running computer program. At any given time, it is in some internal state (for example, which line of the program is being executed) and has some amount of stored data (in variables in memory and on disk). With each "tick" of the computer's clock, it moves to a new state based on its previous state and the contents of its stored data (for example, performing a conditional jump), and may modify the stored data as well.

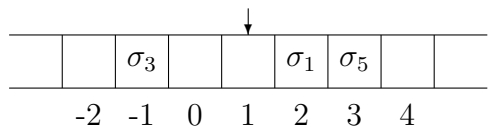
With this characterization in mind, we make the following definition.

Definition 3.3.1 A *Turing machine* M on an alphabet $A = \{\sigma_1, \sigma_2, \ldots, \sigma_k\}$ consists of the following:

(i) A finite set of states $\{Q_0, Q_1, \ldots, Q_m\}$. We call Q_0 the *start* state and Q_m the *stop* state.

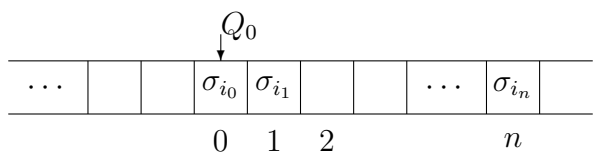
(ii) A table of $(m+1) \cdot (k+1)$ 5-tuples of the form (Q_i, a, b, s, Q_j), where $a, b \in A \cup \{*\}$. (Here $*$ is an additional symbol. whose meaning we explain later. $*$ is sometimes also denoted, for consistency, by $\sigma_0 = *$, so that $A \cup \{*\} = \{\sigma_0, \sigma_1, \ldots, \sigma_k\}$.) For each $0 \leq i \leq m$, $a \in A \cup \{*\}$, there must be exactly one entry in this table of the form (Q_i, a, b, s, Q_j). Finally we must have $s \in \{-1, 0, +1\}$.

The meaning of the set of states is clear from our discussion above, but the rest of the definition may be somewhat more obscure. Here is how to imagine the operation of a Turing machine. We have a two-way unlimited "tape" divided into squares, each of which may contain at most one symbol in A (or possibly be empty), and a read-write head that at any instant is positioned at exactly one square on the tape.



The head can move in one step along the tape at most one square at a time, right or left, it can write a symbol in the scanned square (replacing any symbol that may have been there) or erase the symbol in the scanned square.

Using this image, for any input $w \in A^*$, we can describe the computation of a Turing machine (TM) on this input w as follows: Let $w = \sigma_{i_0}\sigma_{i_1}\ldots\sigma_{i_n}$. Put this input on the tape as in the picture below and put the scanning head at the leftmost symbol in w (anywhere, if $w = \theta$) and assign to it state Q_0 (the start state):



We now follow the table of the TM by interpreting the entry

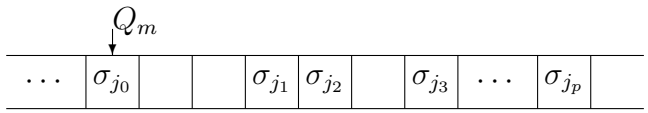
$$(Q_i, a, b, s, Q_j)$$

as expressing the instruction:

When the head is at state Q_i and scans the symbol a (if $a = *$ this means that it scans an empty square – thus $*$ is simply a symbol for the empty square), then prints the symbol b ($b = *$ means "erase what is present at the scanned square"). Then if $s = +1$, move the head one square to the right; if $s = -1$, move one square to the left; and if $s = 0$, leave the head where it is. Finally, change state to Q_j.

So in terms of our original characterization of a computer program, the symbol a represents the possible role of stored data in choosing the new state, while b is the possible change to the stored data. s is an artifact of how we have chosen to represent the stored data as an infinite tape. While it may seem like a limitation to be only able to examine and change one square of the tape at a time, this can in fact be overcome by the addition of a suitable number of intermediate internal states and transitions, and it simplifies the analysis considerably.

The computation proceeds this way step by step, starting from the input w. It terminates exactly when it reaches the stop state, Q_m. When the computation halts, whatever string of symbols from A remain on the tape to the right of the scanning head, say $v = \sigma_{j_0}\sigma_{j_1}\ldots\sigma_{j_p}$ (from left to right), we call the string v the *output* of the computation.



Note that a computation from a given input w may not terminate, because we need not ever reach the stop state Q_m.

Example 3.3.2 Consider the following TM on the alphabet $A = \{0, 1\}$:

states: Q_0, Q_1, Q_2

table: $(Q_0, a, \bar{a}, +1, Q_0)$, for $a \in A$, where $\bar{a} = 1 - a$,
$(Q_0, *, *, -1, Q_1)$
$(Q_1, b, b, -1, Q_1)$, for $b \in A$,
$(Q_1, *, *, +1, Q_2)$
$(Q_2, a, a, 0, Q_2)$, $a \in A \cup \{*\}$

On an input of, say, $w = 1011101$, the machine terminates with output 0100010. In general, on input $w \in A^*$ the machine terminates with output $\bar{w}$ which is equal to w with 0s and 1s switched. (This operation is also known as the binary "ones complement".)

Example 3.3.3 $A = \{1\}$

states: Q_0, Q_1

table: $(Q_0, 1, 1, +1, Q_0)$
$(Q_0, *, 1, +1, Q_0)$
$(Q_1, a, a, 0, Q_1)$, $a \in A \cup \{*\}$

Then no matter what the starting input is, say $w = 1\ldots1$ in A^*, this machine does not terminate: it just keeps adding 1's to the right of w forever.

Definition 3.3.4 We call a decision problem (A, P) *decidable* if there is a Turing machine M on some finite alphabet $B \supseteq A \cup \{Y, N\}$ such that for every word $w \in A^*$:

$$\begin{array}{lcl} w \in P & \Rightarrow & \text{on input } w \text{ the machine } M \text{ halts with output } Y \\ w \notin P & \Rightarrow & \text{on input } w \text{ the machine } M \text{ halts with output } N \end{array}$$

Otherwise, P is called *undecidable*.

3.4 The Church-Turing Thesis

We have now formalized the concept of decidability using the Turing machine model of computation. In this section we will discuss alternative ways to formalize this concept, and see that they are all equivalent.

3.4.A Register Machines

First we will discuss in detail one more model of computation which is closer to the operation of real-world computers: the so-called *register machines*.

Definition 3.4.1 A *register machine* N on an alphabet $A = \{\sigma_1, \ldots, \sigma_k\}$ is a program consisting of a finite list $1 : I_1, \ldots, n : I_n$ of consecutively numbered instructions, each of which is one of the following types:

ADD$_j$ Rm,	$1 \leq j \leq k$; $m = 1, 2, \ldots$
DEL Rm,	$m = 1, 2, \ldots$
CLR Rm,	$m = 1, 2, \ldots$
Rp ←Rm,	$m, p = 1, 2, \ldots$
GO TO ℓ,	$1 \leq \ell \leq n$
IF FIRST$_j$ Rm GO TO ℓ	$1 \leq j \leq k$; $1 \leq \ell \leq n$; $m = 1, 2, \ldots$
STOP	

We assume moreover that $I_i = \text{STOP}$ if and only if $i = n$.

To understand the operation of this machine, imagine an infinite list of registers $R1, R2, \ldots$, each of which is capable of storing an arbitrary word in A^*. Then the meaning of each instruction in the above list is the following:

ADD$_j$ Rm:	add σ_j to the right of (the word in) Rm
DEL Rm:	delete the leftmost symbol in Rm
CLR Rm:	erase the word in Rm
Rp ←Rm:	replace the word in Rp by the word in Rm
GO TO ℓ:	go the ℓth instruction $\ell : I_\ell$
IF FIRST$_j$ Rm GO TO ℓ:	if the word in Rm starts with σ_j, go to the ℓth instruction; otherwise go to the next instruction

STOP: halt the computation.

Using this, we can now describe the computation of the register machine (RM) N on any input $w \in A^*$: Put this input w in the first register R1 (with all other registers empty). Follow the instructions $1 : I_1, \ldots, \ell : I_\ell$ as above in order (unless of course a GO TO instruction is met). This computation terminates exactly when we reach (if ever) I_n : STOP. If $v \in A^*$ is the word in the register R1, when the computation stops, we call v the *output* of the computation. Again a computation from a given input w may not terminate. Also note that if the program for N mentions at most the registers R1, R2,..., Rm, then these are the only registers used in the computation on any input.

Example 3.4.2 Consider the following RM on the alphabet $A = \{1, \diamond\} = \{\sigma_1, \sigma_2\}$

1: IF FIRST$_1$ R1 GO TO 5
2: DEL R1
3: IF FIRST$_1$ R1 GO TO 8
4: GO TO 11
5: DEL R1
6: ADD$_1$ R2
7: GO TO 1
8: DEL R1
9: ADD$_1$ R2
10: GO TO 3
11: R1←R2
12: STOP

Then on input $\underbrace{1\ 1 \ldots 1}_{n} \diamond \underbrace{1\ 1 \ldots 1}_{m}$ the machine terminates with output $\underbrace{1\ 1 \ldots 1}_{n+m}$ $(n, m \geq 1)$. Thus it performs addition in unary notation.

Example 3.4.3 Consider the alphabet $A = \{1\}$, and the machine:

1: ADD$_1$ R1
2: GO TO 1
3: STOP

Then this RM does not terminate on any input in A^*.

Although the TM and RM models of computation are quite different, it can be shown that they are equivalent, in the sense that whatever can be computed using one of the models can be computed with the other, and vice versa. More precisely, consider two finite alphabets A, B and a *partial function* $f : A^* \to B^*$, that is a function whose domain is a subset of A^* and which takes values in B^*. (Normally, when we write $g : X \to Y$ we mean that g is a *total* function with domain *exactly* X and values in Y. We will however use this notation below even if the domain of g is only a (possibly proper) *subset* of X, but in this case we will explicitly state that g is partial.)

Definition 3.4.4 $f : A^* \to B^*$ is TM-*computable* if there is a finite alphabet $C \supseteq A \cup B$ and a Turing Machine M on C such that for each input $w \in A^*(\subseteq C^*)$, M terminates on w iff w is the domain of f (i.e. $f(w)$ is defined), and in that case the output of M on input w is $f(w)$.

We define what it means for f to be RM-computable in a similar way. We can then show

Theorem 3.4.5 *TM-computable = RM-computable.*

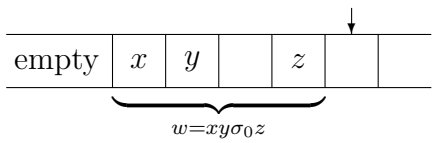
This is a programming exercise and we will not discuss it in detail here. First, we show that given a RM N on an alphabet C, one can construct a TM M on the alphabet $D = C \cup \{\diamond\}$, such that if we put $w \in C^*$ as input in N and the same input $w \in C^*$ in M, then N terminates on w iff M terminates on w, and for such w the outputs of N and M are the same.

Specifically, we assume that all the registers mentioned in N are among R1, R2,..., Rm. If R1,..., Rm, at some stage in the computation, contain the words $w_1, \ldots, w_m$, respectively, then the idea is to represent their contents by the word $w_1 \diamond w_2 \diamond \cdots \diamond w_m \diamond$ in the tape of the TM M, with the head scanning the leftmost symbol of w_1 (or the first $\diamond$ if $w_1 = \theta$). Then for each instruction of N we will introduce a block of entries in the table of M, whose effect on $w_1 \diamond w_2 \diamond \cdots \diamond w_m \diamond$ will correspond to the effect of the instruction of N on the corresponding register values. In the case of GOTO instructions this simulation will preserve the same flow of control.

Conversely, we can show that given a TM M on an alphabet C, one can construct a RM N on some alphabet $D \supseteq C$ such that again if we put $w \in C^*$ as input in M and N, then M terminates on w iff N terminates on w, and for such w the outputs of M and N on w are the same. For example, one way to do that is to take $D = \{\sigma_0, \sigma_1, \ldots, \sigma_k\} \cup \{Q_0, \ldots, Q_m\}$,

if $C = \{\sigma_1, \ldots, \sigma_k\}$ and M has set of states $\{Q_0, \ldots, Q_m\}$. Now consider a tape snapshot wQ_iav, which indicates that the head is in state Q_i, scans the symbol a (which could be the empty square, i.e. $a = \sigma_0$), and to the left of it is the word $w \in \{\sigma_0, \sigma_1, \ldots, \sigma_k\}^*$ and to the right of it the word $v \in \{\sigma_0, \sigma_1, \ldots, \sigma_k\}^*$. Notice that at each stage of the computation of the machine M there will only be finitely many non-empty squares. Here w is the word which represents from left-to-right the contents of the tape starting from the left-most non-empty square to the square just to the left of the square scanned by the head. In case every square to the left of the head is empty, $w = \theta$. Similarly for v.

Example.



We can represent this tape snapshot in the register machine N by putting w in R1, Q_ia in R2, and v in R3. Then for each entry $(Q_i, a, -, -, -)$ in the table of M we can write a set of instructions (subroutine) for N such that the effect of this entry on wQ_iav is the same as the effect of the corresponding set of instructions on the corresponding register values. In this way, we can show that the notions of TM-computability and RM-computability are equivalent.

3.4.B The Church-Turing Thesis

As mentioned earlier, many alternative models of computation $\mathcal{M}$ have been proposed over the years, and for each one we have a corresponding notion of an $\mathcal{M}$-computable function. Two common ones which we will not discuss, but which you may encounter later, are the Lambda Calculus and the theory of Recursive Functions. Another one is the theory of cellular automata, an specific example of which is Conway's Game of Life. It turns out that in all these cases, one can again prove that $\mathcal{M}$-computable = TM-computable, so all these models are, in that sense, equivalent to the Turing machine model.

It is accepted today that the informal notion of algorithmic computability is represented correctly by the precise notion of TM-computability (= RM-computability = ...), in a way similar to the assertion that the formal ϵ-δ definition of continuity of functions of a real variable represents accurately

the intuitive concept of continuity. This (extramathematical) assertion that "computability = TM-computability" is referred to as the *Church-Turing Thesis*.

More explicitly, consider a partial function $f : A^* \to B^*$ (A, B finite alphabets). We call it *intuitively computable* if there is an algorithm which for each input $w \in A^*$ terminates (in finitely many steps) exactly when w is in the domain of f and in that case produces the output $f(w)$. Then the Church-Turing Thesis is the assertion:

$$\text{intuitively computable} = \text{TM-computable}.$$

We will simply say *computable* instead of TM-computable from now on.

Remark. Any decision problem (A, P) may be identified with the (total) function $f : A^* \to \{Y, N\}^*$ given by $f(w) = Y$ if $w \in P$ and $f(w) = N$ if $w \notin P$. Thus (A, P) is decidable iff f is computable.

3.5 Universal Machines

So far we have been discussing machines which represent the operation of a single computer program. But what about a computer itself, i.e. a single machine which can execute any program? It turns out that our existing formalization is sufficient to deal with this more general case as well. After a bit of thought, this should come as no surprise, since after all the inside of a computer is just a program which sequentially reads and executes the instructions of other programs. We will now describe how to formalize this notion.

To start with, we can effectively code the words in an arbitrary finite alphabet $A = \{\sigma_1, \ldots, \sigma_k\}$ by using words in the binary alphabet $\{0, 1\}$. For example, if $k = 22$ we can use five letter words from $\{0, 1\}$ to represent each individual symbol in A, and thus every word of length n from A can be encoded as a word of length $5n$ in the binary alphabet. (We could even use the unary alphabet $\{1\}$ instead of the binary alphabet, but this would require exponential increase in the length of the word if A has more than one symbol. This does not affect computability issues, but it can certainly affect the complexity issues discussed later.)

So let's assume that we have fixed some simple method for encoding words in a given alphabet A by binary words and refer to the binary word

$b(w)$ encoding the word $w \in A^*$ as the *binary* code of w. It is now not hard to see that given a TM M on A we can construct a TM M^* on $\{0,1\}$ such that for each $w \in A^*$ the input-output behavior of M on w is exactly the same on the input-output behavior of M^* on $b(w)$.

Thus, without any loss of generality, we can restrict ourselves to discussing TMs on the binary alphabet. Any such machine M is simply given by a finite table of 5-tuples which again can be easily coded as a word in the binary alphabet. We call this the *binary code* of M and denote it by $b(M)$. By carrying out a somewhat complicated programming project, one can now construct a *universal Turing Machine*. This is a TM U on $\{0,1\}$ such that for any TM M on $\{0,1\}$ and any input $w = \{0,1\}^*$, if we put $b(M)*w$ (where $* =$ empty square) as input in U (so that in the beginning the head scans the first symbol of $b(M)$), then U terminates on this input iff M terminates on w, and in this case their outputs are the same. So we have:

Theorem 3.5.1 (Existence of universal TM) *We can construct a TM U on $\{0,1\}$ such that for any TM M on $\{0,1\}$ with binary code $b(M)$, the input-output behavior of U on $b(M) * w$ ($w \in \{0,1\}^*$) is the same as the input-output behavior of M on w.*

Remark. The universal TM was conceived theoretically by Turing in the 1930's. Any personal computer today is essentially a practical implementation of this idea.

3.6 The Halting Problem

We will now use theorem 3.5.1 to show that the so-called *halting problem* for TM is undecidable. This is precisely formulated as:

$$\begin{aligned} A &= \{0,1\} \\ P &= \{b(M) : M \text{ is a TM on } \{0,1\} \text{ and } M \\ &\qquad \text{terminates on the empty input}\}. \end{aligned} \tag{HALTING}$$

Theorem 3.6.1 (Turing) HALTING *is undecidable.*

Proof. Consider the universal TM U. For each $x \in \{0,1\}^*$ consider the TM U_x which on the empty input first prints $x * x$, moves the head over

the first symbol of the first x, and then follows U. The table of U_x is easily obtained from the table of U by adding some further entries depending on x.

Now, if HALTING were decidable, one could easily build a TM N_0 on $\{0,1\}$ such that for any $x \in \{0,1\}^*$, N_0 on input x would terminate with output 0 if U_x terminated on the empty input, and N_0 on input x would terminate with output 1 if U_x did not terminate on the empty input. By changing N_0 a little, we could then construct a TM M_0 on $\{0,1\}$ such that if N_0 on input x terminates with output 1, so does M_0, but if N_0 on input x terminates with output 0, then M_0 does *not* terminate in that input. To simplify the notation, let us write

$$M(w) \downarrow$$

if a TM M terminates on input w and

$$M(w) \uparrow$$

if it does not. Then we have for $x \in \{0,1\}^*$

$$\begin{aligned} M_0(x) \downarrow &\Leftrightarrow U_x(\theta) \uparrow \\ &\Leftrightarrow U(x * x) \uparrow . \end{aligned}$$

In particular, for any TM M on $\{0,1\}$,

$$\begin{aligned} M_0(b(M)) \downarrow &\Leftrightarrow U(b(M) * b(M)) \uparrow \\ &\Leftrightarrow M(b(M)) \uparrow \end{aligned}$$

Putting $M = M_0$ we get a contradiction, since it cannot be the case that both $M_0(b(M_0)) \uparrow$ and $M_0(b(M_0)) \downarrow$. $\dashv$

Similarly one can of course prove that the corresponding halting problem for RM is undecidable. More generally, given any programming language, it is impossible to design an algorithm that will decide whether an arbitrary program will terminate on a given input or not.

3.7 Undecidability of the Validity Problem

We have succeeded in showing that one problem—the halting problem—is undecidable, but the prospects are not good for applying the method we

used to other problems, since it depends on the halting problem being a sort of "meta-problem" *about* computability. Fortunately, we can leverage this success to show the undecidability of other problems, by using the method of computable reductions, as follows.

Definition 3.7.1 Suppose (A, P) and (B, Q) are two decision problems. A (total) function $f : A^* \to B^*$ is a *(computable) reduction* of P to Q if f is computable and for any $w \in A^*$,

$$w \in P \Leftrightarrow f(w) \in Q.$$

Notice that if P is reduced to Q via f and Q is decidable, so that there is a total computable function $g : B^* \to \{Y, N\}^*$ such that $v \in Q \Rightarrow g(v) = Y$, $v \notin Q \Rightarrow g(v) = N$, then $g \circ f = h$ is computable and $w \in P \Rightarrow h(w) = Y$, $w \notin P \Rightarrow h(w) = N$. Thus P is decidable as well.

So if P can be reduced to Q and Q is decidable, then P is decidable. This observation provides a general method for showing that decision problems are *un*decidable. Specifically, to show that (B, Q) is undecidable, choose an appropriate problem (A, P) that is known to be undecidable, and find a reduction of P to Q. For then if (B, Q) were decidable, so would (A, P) be, which we know is not the case. We will apply this method to show the undecidability of the validity problem for an appropriate language L.

Theorem 3.7.2 (Church) *There is a finite language L such that the decision problem* VALIDITY_L *is undecidable.*

Proof. First consider a variation of the halting problem, HALTING_U. This is simply the halting problem for the universal TM U. It consists of all $w \in \{0, 1\}^*$ for which $U(w) \downarrow$. Since for any TM M on $\{0, 1\}^*$, $M(\theta) \downarrow$ iff $U(b(M) * \theta) \downarrow$ iff $U(b(M)) \downarrow$, clearly HALTING_U is also undecidable.

Say the states of U are $\{Q_0, Q_1, \ldots, Q_m\}$. We take L to consist of the following symbols:

$$\begin{array}{rl} 0 & : \text{constant} \\ S & : \text{unary function} \\ < & : \text{binary relation} \\ H & : \text{binary relation} \\ T_*,\ T_0,\ T_1 & : \text{binary relations} \\ R_0, \ldots, R_m & : \text{unary relations.} \end{array}$$

Intuitively, we have in mind the following interpretation for these:

(i) $0, S, <$ will be the 0, successor and order on $\langle \mathbb{Z}, 0, S, < \rangle$.

(ii) Variables will vary over $\mathbb{Z}$. They will represent both the time (i.e. stage in the computation), when restricted to ≥ 0 values (so 0 will be the beginning, 1 the next step, 2 the next one, etc.), and also the location of a square on the tape.

...						...
	-2	-1	0	1	2	

(iii) $H(t, x)$ will be true iff $t, x \in \mathbb{Z}$, $t \geq 0$ and the head at time t scans the square x.

(iv) $T_*(t, x)$ will be true iff $t, x \in \mathbb{Z}$, $t \geq 0$ and the tape at time t contains $*$ (i.e. is empty) at the square x, and similarly for T_0 and T_1.

(v) $R_i(t)$ will be true iff $t \in \mathbb{Z}$, $t \geq 0$, and at time t the machine is in state Q_i.

We will now assign to each $w \in \{0, 1\}^*$ a sentence σ_w in L such that

$$U(w) \downarrow \quad \text{iff} \quad \sigma_w \text{ is valid.}$$

It will be clear from our construction that the map $w \mapsto \sigma_w$ is computable. Thus it will follow that HALTING_U can be reduced to VALIDITY_L, so VALIDITY_L must be undecidable.

We will first construct some preliminary sentences in L:

(i) Z is the conjunction of the following sentences:

$$\forall x(\neg x < x)$$
$$\forall x \forall y \forall z(x < y \wedge y < z \Rightarrow x < z)$$
$$\forall x \forall y(x < y \vee x = y \vee y < x)$$
$$\forall x \exists y(S(y) = x)$$
$$\forall x(x < S(x))$$
$$\forall x \forall y(x < y \Rightarrow y = S(x) \vee S(x) < y)$$

(expressing the usual properties of $\langle \mathbb{Z}, 0, S, < \rangle$).

(ii) NOCONFLICT is the conjunction of the following sentences:

$$\begin{gathered}
\forall t(0 = t \vee 0 < t \Rightarrow R_0(t) \vee \cdots \vee R_m(t)) \\
\forall t(0 = t \vee 0 < t \Rightarrow \neg R_i(t) \vee \neg R_{i'}(t)),\ 0 \leq i < i' \leq m \\
\forall t(0 = t \vee 0 < t \Rightarrow \exists x H(t, x)) \\
\forall t \forall x \forall x'((0 = t \vee 0 < t) \wedge x \neq x' \Rightarrow \neg H(t, x) \vee \neg H(t, x')) \\
\forall t \forall x(0 = t \vee 0 < t \Rightarrow T_*(t, x) \vee T_0(t, x) \vee T_1(t, x)) \\
\forall t \forall x(0 = t \vee 0 < t \Rightarrow (\neg T_*(t, x) \vee \neg T_0(t, x))) \\
\forall t \forall x(0 = t \vee 0 < t \Rightarrow (\neg T_0(t, x) \vee \neg T_1(t_1, x)) \\
\forall t \forall x(0 = t \vee 0 < t \Rightarrow (\neg T_1(t, x) \vee \neg T_*(t, x))
\end{gathered}$$

expressing that at each time t, M is in exactly one state, and scans exactly one square, and for each time t and each square x there is exactly one symbol on that square (possibly empty).

(iii) START_w is the conjunction of the following sentences, where $w = w_0 \dots w_{n-1}$ with $w_i \in \{0, 1\}$, and we use the abbreviation

$$\bar{i} = S(S(\dots S(0)) \dots)$$

(i times) for all $0 \leq i \in \mathbb{Z}$:

$$\begin{gathered}
R_0(0) \\
H(0, 0) \\
T_{w(i)}(0, \bar{i}),\ 0 \leq i < n \\
\forall x(\bar{n} < x \vee \bar{n} = x \Rightarrow T_*(x)) \\
\forall x(x < 0 \Rightarrow T_*(x))
\end{gathered}$$

(expressing that the machine starts in state Q_0 with the head on the 0th square and on the input w).

(iv) NONSTOP is the sentence

$$\forall t(0 = t \vee 0 < t \Rightarrow \neg R_m(t))$$

expressing that the machine does not stop.

(v) STEP is the sentence constructed as follows:

Let (Q_i, a, b, s, Q_j) be an entry in the table of U. Say $s = +1$. (Appropriate changes have to be made in the formula below in case $s = 0$ or $s = -1$.) Assign to this entry the following sentence:

$$\forall t \forall x [0 = t \vee 0 < t \Rightarrow (R_i(t) \wedge H(t,x) \wedge T_a(t,x) \Rightarrow \\ R_j(S(t)) \wedge H(S(t), S(x)) \wedge T_b(S(t), x))]$$

expressing the action of the machine following this entry. Then STEP is the conjunction of all these sentences for all entries in the table.

Now let ρ_w be the conjunction of all the sentences Z, NOCONFLICT START$_w$, NONSTOP, STEP, and put $\sigma_w = \neg\rho_w$. Then we have

$$U(w)\downarrow \quad \text{iff} \quad \sigma_w \text{ is valid.}$$

To see this, first notice that if $U(w)\uparrow$, then σ_w is not valid, i.e. ρ_w has a model. For we can take as such a model

$$\mathcal{M} = \langle \mathbb{Z}, 0, S, <, H^{\mathcal{M}}, T_*^{\mathcal{M}}, T_0^{\mathcal{M}}, T_1^{\mathcal{M}}, R_0^{\mathcal{M}}, \ldots, R_m^{\mathcal{M}} \rangle,$$

where $0, S, <$ have their usual meaning and $H^{\mathcal{M}}, \ldots, R_m^{\mathcal{M}}$ are interpreted as in (3)-(5) before.

Conversely, assume that $U(w)\downarrow$, say the computation on input w terminates at time $N \geq 0$. If σ_w failed to be valid (working towards a contradiction), ρ_w would have some model

$$\mathcal{A} = \langle A, 0^{\mathcal{A}}, S^{\mathcal{A}}, <^{\mathcal{A}}, H^{\mathcal{A}}, T_*^{\mathcal{A}}, T_0^{\mathcal{A}}, T_1^{\mathcal{A}}, R_0^{\mathcal{A}}, \ldots, R_m^{\mathcal{A}} \rangle.$$

Since $\mathcal{A} \models Z$, clearly A contains a copy of $\mathbb{Z}$, with $0^{\mathcal{A}}$ corresponding to 0, $S^{\mathcal{A}}(0^{\mathcal{A}})$ to 1, etc. Denote by $\mathbf{n}$ the element of A corresponding to $n \in \mathbb{Z}$. (In general, A contains many more other elements than these $\mathbf{n}$.) Then it is easy to see that for $t \in \mathbb{Z}$ with $t \geq 0$ and $x \in \mathbb{Z}$, $H^{\mathcal{A}}(\mathbf{t}, \mathbf{x})$ will be true iff the head at time t scans the square x, and similarly for $T_*^{\mathcal{A}}, \ldots, R_m^{\mathcal{A}}$. This can be proved, for example, by induction on t. Thus $R_m^{\mathcal{A}}(\mathbf{N})$ is true and this contradicts that $\mathcal{A} \models$ NONSTOP. $\dashv$

3.8 The Hilbert Tenth Problem

Using the method of reduction it has been also shown that the Hilbert 10th Problem (example 3.1.4) is undecidable. More precisely, consider the alphabet

$$A = \{x, 0, 1, \cdot, +, -, (,), \hat{}\}.$$

Encoding the variable x_n by $x(n$ written in binary), natural numbers by their binary notation, and using $\hat{}$ for exponentiation, any polynomial in several variables with integer coefficients can be encoded by a word in this alphabet.

Example 3.8.1 $x_1^2 - 7x_2^2 - 1$ will be encoded by the word

$$(x(1))\hat{}(10) + (-111)(x(10))\hat{}(10) + (-1).$$

Let now DIOPHANTINE EQUATIONS be the decision problem (A, P) where

$$P = \{w \in A^* : w \text{ encodes a polynomial (with integer coefficients in several variables) which has an integer solution}\}.$$

We now have

Theorem 3.8.2 (Matyasevich) DIOPHANTINE EQUATIONS *is undecidable.*

So this gives a negative answer to the Hilbert 10th Problem.

3.9 Decidable Problems

We will now discuss some examples of decidable problems. We will present them informally, by giving the instances together with the question defining the set of instances that constitutes the decision problem. It will be assumed that then these can be encoded in some reasonable way as formal decision problems (A, P) as in section 3.2.

Example 3.9.1 ELEMENTARY ALGEBRA is the following decision problem:

Instance. A sentence σ in the first-order language $L = \{0, 1, +, \cdot\}$.

Question. Is σ true in $\langle \mathbb{R}, 0, 1, +, \cdot \rangle$, i.e. is $\sigma \in \mathrm{Th}(\mathbb{R}, 0, 1, +, \cdot)$?

Tarski in 1949 has shown that ELEMENTARY ALGEBRA is decidable.

Remark. On the other hand, if we consider the corresponding problem ELEMENTARY ARITHMETIC for $\mathrm{Th}(\mathcal{N}, 0, S, +, \cdot, <)$, then Church has shown in the 1930's that it is undecidable (this also follows easily from 3.8.2).

Example 3.9.2 SATISFIABILITY is the following decision problem:

Instance. A finite set $U = \{u_1, \dots, u_k\}$ of propositional variables and $C = \{c_1, \dots, c_m\}$ a finite set of clauses $c_i = \{\ell_{i,1}, \dots, \ell_{i,n_i}\}$, where each $\ell_{i,j}$ is a literal from U, i.e. a variable in U or its negation. (As usual we write $\bar{u}_i$ instead of $\neg u_i$ in this context.)

Question. Is C satisfiable (i.e. is there a valuation $\nu : U \to \{T, F\}$ which makes every clause c_i true, recalling that the clause c_i as above represents the disjunction $\ell_{i,1} \vee \cdots \vee \ell_{i,n_i}$)?

Using truth tables it is clear that SATISFIABILITY is decidable.

Example 3.9.3 TRAVELING SALESMAN is the following decision problem:

Instance. A finite set $\{c_1, \dots, c_m\}$ of "cities", a distance function $d(c_i, c_j) \in \{1, 2, \dots\}$ for $i \neq j$, and a bound $B \in \{1, 2, \dots\}$.

Question. Is there a tour of all the cities with total length $\leq B$, i.e. an ordering $c_{\pi(1)}, c_{\pi(2)}, \dots, c_{\pi(m)}$ of $\{c_1, \dots, c_m\}$ such that

$$\sum_{i=1}^{m-1} d(c_{\pi(i)}, c_{\pi(i+1)}) + d(c_{\pi(m)}, c_{\pi(1)}) \leq B?$$

Again, by listing all possible orderings and calculating the above sum for each one of these, it is trivial to see that this problem is decidable.

Example 3.9.4 PRIMES is the following decision problem:

Instance. An integer $n \geq 2$.

Question. Is n prime?

This is also decidable as we can go through all the integers between 2 and $n - 1$ and check whether they divide n.

3.10 The Class $\mathcal{P}$

We have so far been discussing what problems are computable *in principle*. However, a question of more practical interest is the following: given a problem which *is* computable, can we compute it *efficiently*? The study of efficiency is also called *computational complexity*. We will concentrate here on time complexity (how long it takes to solve the problem), but one can also discuss, for example, space complexity (how much "memory" storage is used in solving it). A natural measure of time complexity for us is the number of steps in the computation of a TM that decides a given problem.

Since different problems require different amounts of input, in order to compare the complexities of different algorithms and problems, we must measure complexity as a function of the input size. In addition, with computer speeds increasing rapidly, small instances of a problem can usually be solved no matter how difficult the problem is. For this reason, and also to eliminate the effect of "overhead" time, we will consider only the *asymptotic* complexity as the size of the input increases. First we recall the "big-O notation," which gives a rigorous way of comparing asymptotic growth rates.

Definition 3.10.1 Given two functions $f, g : \mathbb{N} \to \mathbb{N}$ we define

$$f = O(g) \text{ iff } \exists n_0 \, \exists C \, \forall n \geq n_0 (f(n) \leq Cg(n)).$$

For example, if $p(n)$ is a polynomial, then $p(n) = O(n^d)$ for large enough d.

Definition 3.10.2 Given any $T : \mathbb{N} \to \mathbb{N}$, we let TIME($T$) consist of all decision problems (A, P) which can be decided by a TM in time t for some $t = O(T)$.

More precisely, a decision problem (A, P) is in TIME(T) if there is $t = O(T)$ and a TM M on some alphabet $B \supseteq A \cup \{Y, N\}$ such that for $w \in A^*$:

(i) $w \in P \Rightarrow$ on input w, M halts in $\leq t(|w|)$ steps with output Y.

(ii) $w \notin P \Rightarrow$ on input w, M halts in $\leq t(|w|)$ steps with output N

(here $|w|$ = length of the word w).

What sort of growth rate should we require of an algorithm to consider it manageable? After all, we must expect the time to increase somewhat with the input size. It turns out that the major distinctions in complexity are between "polynomial time" algorithms and faster-growing ones, such as exponentials.

Definition 3.10.3 A decision problem is in the class $\mathcal{P}$ (or it is *polynomially decidable*) if it is TIME(n^d) for some $d \geq 0$, i.e. it can be decided in polynomial time.

Problems in the class $\mathcal{P}$ are considered *tractable* (efficiently decidable) and the others *intractable*. It is clear that the class $\mathcal{P}$ provides an upper limit for problems that can be algorithmically solved in realistic terms. If a problem is in $\mathcal{P}$, however, it does not necessarily mean that an algorithm for it can be practically implemented, for example, it could have time complexity of the order of $n^{1,000,000}$ or of the order n^3 but with enormous coefficients. However, most natural problems that have been shown to be polynomially decidable have been found to have efficient (e.g., very low degree) algorithms. Moreover, the class of problems in $\mathcal{P}$ behaves well mathematically, and is independent of the model of computation, since any two formal models of computation can be mutually simulated within polynomial time.

Remark. Time complexity as explained here is a worst case analysis. If a problem P is intractable, then there is no polynomial time algorithm which for all n and all inputs of length n will decide P. But one can still search for approximate algorithms that work well on the average or for most practical (e.g., small) instances of the problem or give the correct answer with high probability, etc.

Example 3.10.4 ELEMENTARY ALGEBRA is intractable (Fisher-Rabin 1974). In fact it is in TIME($2^{2^{cn}}$) for some $c > 0$ but not in TIME(2^{dn}) for any $d > 0$; that is, it is *super-exponential*.

Example 3.10.5 LINEAR PROGRAMMING is the decision problem given by:

Instance. An integer matrix (v_{ij}) for $1 \leq i \leq m$ and $1 \leq j \leq n$, along with integer vectors $D = (d_i)_{i=1}^m$ and $C = (c_j)_{j=1}^n$, and an integer B.

Question. Is there a rational vector $X = (x_j)_{j=1}^n$ such that $\sum_{j=1}^n v_{ij}x_j \leq d_i$, for $1 \leq i \leq m$, and $\sum_{j=1}^n c_j x_j \geq B$?

LINEAR PROGRAMMING turns out to be in $\mathcal{P}$ (Khachian, 1979).

3.11 The Class $\mathcal{NP}$ and the $\mathcal{P} = \mathcal{NP}$ Problem

Although a large class of problems have been classified as tractable or intractable, there is a vast collection of decision problems, many of them of great practical importance, that are widely assumed to be intractable but no one until now has been able to demonstrate it. These are the so-called *$\mathcal{NP}$-complete problems*. In order to introduce these problems, we must first define the class $\mathcal{NP}$ of *non-deterministic polynomial* decision problems.

Definition 3.11.1 Let (A, P) be a decision problem. We say that (A, P) is in the class $\mathcal{NP}$ if there is a TM M on an alphabet $B \supseteq A \cup \{Y, N\}$ and a polynomial $p(n)$ such that for any $w \in A^*$:

$$w \in P \Leftrightarrow \begin{array}{l} \exists v \in B^* \text{ such that } |v| \leq p(|w|) \text{ and on input } w * v,\ M \\ \text{stops with output } Y \text{ after at most } p(|w|) \text{ many steps.} \end{array}$$

In other words, $w \in P$ iff there is a "guess" v, of length bounded by a polynomial in the length of w, such that w together with v pass a polynomial acceptance test. (This can be also viewed as a non-deterministic polynomial time algorithm.)

Example 3.11.2 SATISFIABILITY is in $\mathcal{NP}$, since if we can guess a truth assignment, we can verify that it satisfies the given set of clauses in polynomial time.

Example 3.11.3 Similarly, TRAVELING SALESMAN is in $\mathcal{NP}$, since if we guess the order of the set of cities, we can calculate whether the length of the tour is $\leq B$ in polynomial time.

In fact a vast number of problems like these, which involve some kind of search, are in $\mathcal{NP}$.

Remark. Clearly $\mathcal{P} \subseteq \mathcal{NP} \subseteq \bigcup_d \mathrm{TIME}(2^{n^d})$.

Problem. A famous problem in theoretical computer science is whether

$$\mathcal{P} = \mathcal{NP},$$

that is, whether any problem with an efficient nondeterministic algorithm also has an efficient deterministic algorithm. This is known, unsurprisingly, as the $\mathcal{P} = \mathcal{NP}$ *Problem*. Recently the Clay Mathematics Institute included the $\mathcal{P} = \mathcal{NP}$ Problem as one of its seven Millenium Prize Problems, offering \$1,000,000 for its solution! The prevailing assumption today is that $\mathcal{P} \neq \mathcal{NP}$.

3.12 $\mathcal{NP}$-Complete Problems

We can get a better understanding of the $\mathcal{P} = \mathcal{NP}$ problem by discussing the notion of an $\mathcal{NP}$-complete problem. The $\mathcal{NP}$-complete problems form sort of a "core" of the "most difficult" problems in $\mathcal{NP}$. Recall the concept of a computable reduction (definition 3.7.1). We adapt this to the setting of complexity as follows:

Definition 3.12.1 A (total) function $f : A^* \to B^*$, where A, B are finite alphabets, is *polynomial-time computable* if there is a TM M on a finite alphabet $C \supseteq A \cup B$, and a polynomial p, such that for every input $w \in A^*$, M terminates on at most $p(|w|)$ steps with output $f(w)$.

Unsurprisingly, a *polynomial-time reduction* is a computable reduction which is in addition polynomial-time computable. We now say that

Definition 3.12.2 A decision problem (B, Q) is $\mathcal{NP}$*-complete* if it is in $\mathcal{NP}$, and for every $\mathcal{NP}$ problem (A, P) there is a polynomial-time reduction of P to Q. (That is, there is a polynomial-time computable function $f : A^* \to B^*$ such that $w \in P$ if and only if $f(w) \in Q$.)

An $\mathcal{NP}$-complete problem is in some sense a hardest possible problem in $\mathcal{NP}$. For example, it is clear that if (B, Q) is in $\mathcal{P}$ and (A, P) can be polynomial-time reduced to Q, then also (A, P) is in $\mathcal{P}$. It therefore follows that if *any* $\mathcal{NP}$-complete problem is in $\mathcal{P}$, then $\mathcal{P} = \mathcal{NP}$. Thus the $\mathcal{P} = \mathcal{NP}$ question is equivalent to the question of whether any given $\mathcal{NP}$-complete problem is in $\mathcal{P}$.

It is clear from the definition that all $\mathcal{NP}$-complete problems are "equivalent" in some sense, since each one can be reduced to the other by a polynomial-time computable function. We have not yet established however that there are *any* $\mathcal{NP}$-complete problems. This was first shown by Cook and Levin in 1971.

Theorem 3.12.3 (Cook, Levin) SATISFIABILITY *is* $\mathcal{NP}$*-complete.*

Karp in 1972 has shown that TRAVELING SALESMAN and many other combinatorial problems are $\mathcal{NP}$-complete, and since that time hundreds of others have been discovered in many areas of mathematics and computer science.

We will give a proof of this theorem shortly, but first let us say a few things about the complexity of PRIMES, because it occupies a special place among "difficult" problems. Given a decision problem (A, P), its *complement* is the decision problem $(A, \sim P)$, where $\sim P = A^* \setminus P$. It is clear that a problem is decidable (or in $\mathcal{P}$) iff its complement is decidable (or in $\mathcal{P}$), but it is unknown whether a problem is in $\mathcal{NP}$ iff its complement is $\mathcal{NP}$.

We denote by "co-$\mathcal{NP}$" the class of problems whose complements are in $\mathcal{NP}$. Thus $\mathcal{P} \subseteq \mathcal{NP} \cap \text{co-}\mathcal{NP}$, but whether $\mathcal{NP} = \text{co-}\mathcal{NP}$ is unknown. Now it is easy to see that PRIMES in co-$\mathcal{NP}$. (In precisely formulating PRIMES we assume that every positive integer is represented by its binary notation. If numbers were represented in unary notation, then clearly PRIMES would be in $\mathcal{P}$.) With some work, using some elementary number theory, it can be shown that PRIMES is also in $\mathcal{NP}$. Thus

$$\text{PRIMES} \in \mathcal{NP} \cap \text{co-}\mathcal{NP}.$$

Miller in 1976 has shown that if one assumes a widely believed but still unproven hypothesis, the so-called Generalized Riemann Hypothesis, then actually PRIMES is in $\mathcal{P}$, but it is still unknown whether PRIMES is actually in $\mathcal{P}$. (*Addendum*: In 2002 Agrawal, Kayal and Saxena proved that PRIMES is indeed in $\mathcal{P}$.)

Proof of 3.12.3. The proof is a variation of that of 3.7.2.

To fix a formal encoding of the satisfiability problem, we simply view each instance as a formula in propositional logic in conjunctive normal form (cnf), and which can thus be viewed as a word in the alphabet

$$C = \{p, 0, 1, \neg, \wedge, \vee, (,)\},$$

where we write pn, with n in binary, instead of the propositional variable p_n.

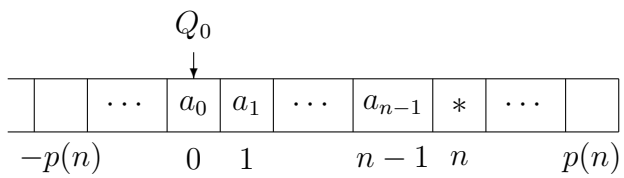
Consider now an arbitrary decision problem (A, P) which is in the class $\mathcal{NP}$. We will find a polynomial-time computable map $f : A^* \to C^*$ such that, for each $w \in A^*$, $f(w)$ is a formula in cnf and

$$w \in P \Leftrightarrow f(w) \text{ is satisfiable.}$$

By a slight reformulation of the definition of what it means to be in the class $\mathcal{NP}$, we can find an alphabet $B \supseteq A$, a polynomial $p(n) > n$ and a TM M on B with states $\{Q_0, \ldots, Q_m\}$ (Q_0 is the start and Q_m is the stop

state), such that all the entries in the table of M that begin with Q_m are of the form $(Q_m, b, b, 0, Q_m)$, and we have for any $w = a_0 a_1 \ldots a_{n-1} \in A^*$:

$w \in P$ iff there is an input



in which a_i is in square i, $0 \leq i \leq n-1$, square n is empty, as are all squares $j < 0$ and $j > p(n)$ (but some symbols in B are possibly occupying some squares between $n+1$ and $p(n)$), and on which M terminates in at most $p(n)$ many steps (and thus at each step of the computation the machine only visits squares with index $-p(n) \leq j \leq p(n)$).

Let $A = \{\sigma_1, \ldots, \sigma_K\}$, $B = \{\sigma_1, \ldots, \sigma_K, \sigma_{K+1}, \ldots, \sigma_L\}$, and finally let σ_0 denote the empty square.

Associate with M and each $n = 0, 1, 2, \ldots$ the following list of propositional variables:

(i) $R_{i,k}$, $0 \leq i \leq p(n)$, $0 \leq k \leq m$,

(ii) $H_{i,j}$, $0 \leq i \leq p(n)$, $-p(n) \leq j \leq p(n)$,

(iii) $T_{i,j,\ell}$, $0 \leq i \leq p(n)$, $-p(n) \leq j \leq p(n)$, $0 \leq \ell \leq L$.

The intended meanings of these variables are as follows:

(i) $R_{i,k}$ being true means that at time i the machine is in state Q_k.

(ii) $H_{i,j}$ being true means that at time i the head scans square j.

(iii) $T_{i,j,\ell}$ being true means that at time i the tape contains σ_ℓ at square j.

(We can of course view these propositional variables as part of our standard list $p_0, p_1, p_2, \ldots$.)

Consider now the following clauses in these variables, associated to each $w = \sigma_{k_0}\sigma_{k_1}\ldots\sigma_{k_{n-1}} \in A^*$, $1 \leq k_i \leq K$:

(a)

$$\begin{array}{ll}
R_{i,0} \vee \cdots \vee R_{i,m} & \\
\neg R_{i,k} \vee \neg R_{i,k'} & (0 \le i \le p(n),\ 0 \le k < k' \le m) \\
H_{i,-p(n)} \vee \cdots \vee H_{i,p(n)} & \\
\neg H_{i,j} \vee \neg H_{i,j'} & (0 \le i \le p(n),\ -p(n) \le j < j' \le p(n)) \\
T_{i,j,0} \vee \cdots \vee T_{i,j,L} & \\
\neg T_{i,j,\ell} \vee \neg T_{i,j,\ell'} & (0 \le i \le p(n),\ -p(n) \le j \le p(n), \\
 & \ 0 \le \ell < \ell' \le L)
\end{array}$$

(expressing that at each time i the machine is in exactly one state and scans exactly one square and for each time i and square j there is exactly one symbol on j, possibly $*$).

(b)

$$\begin{array}{ll}
R_{0,0};\ H_{0,0};\ T_{0,j,k_j} & (0 \le j \le n-1) \\
T_{0,n,0} & \\
T_{0,j,0} & (j < 0)
\end{array}$$

(these express that the machine starts at time 0 at state Q_0 with the head at the 0th square and on input as in the preceding picture)

(c)

$$R_{p(n),m}$$

(this expresses that the machine terminates at time $\le p(n)$)

(d) Let $(Q_k, \sigma_\ell, \sigma_{\ell'}, s, Q_{k'})$ be an entry in the table of M. Say $s = +1$. (Appropriate changes in the formula below have to be made if $s = 0$ or $s = -1$.) Assign to this entry the following clauses for each $0 \le i < p(n)$, $-p(n) \le j < p(n)$:

$$\begin{array}{c}
\neg R_{i,k} \vee \neg H_{i,j} \vee \neg T_{i,j,\ell} \vee R_{i+1,k'} \\
\neg R_{i,k} \vee \neg H_{i,j} \vee \neg T_{i,j,\ell} \vee H_{i+1,j+1} \\
\neg R_{i,k} \vee \neg H_{i,j} \vee \neg T_{i,j,\ell} \vee T_{i+1,j,\ell'}
\end{array}$$

(these express the action of the machine, when at time i it is at state Q_k and scans σ_ℓ on the jth square). Note here that $\neg R_{i,k} \vee \neg H_{i,j} \vee \neg T_{i,j,\ell} \vee R_{i+1,k'}$ is equivalent to $R_{i,k} \wedge H_{i,j} \wedge T_{i,j,\ell} \Rightarrow R_{i+1,k'}$.

Finally, let $f(w)$ be the conjunction of all the formulas (clauses) given in (a), (b), (c), (d). It is easy to check that f is a polynomial-time computable function. We verify that $w \in P \Leftrightarrow f(w)$ is satisfiable.

If $w \in P$, then we can find an input as in the previous picture with the properties stated there and we assign truth values to the variables according to their intended meaning in this computation associated with that input. This valuation clearly satisfies all the clauses of $f(w)$.

Conversely, assume a valuation ν satisfies all the clauses in $f(w)$. Because ν satisfies all the clauses in (a), (b), for each $-p(n) \leq j \leq p(n)$ there is a unique $\sigma_{k(j)}$ $(0 \leq k(j) \leq L)$ with $\nu(T_{0,j,k(j)}) = T$. Consider the input for M that has $\sigma_{k(j)}$ in the jth square. Because $k_j = k(j)$ for $0 \leq j \leq n-1$ and $k(n) = 0$, this is exactly an input as in the preceding picture. If we start M on that input then it is easy to see, since ν satisfies all the clauses in (a), (d), that for all $0 \leq i \leq p(n)$, $\nu(R_{i,k}) = T$ iff at time i in this computation the machine is in state Q_k, and similarly for $H_{i,j}$ and $T_{i,j,\ell}$. This is proved by induction on i. But then, since ν satisfies (c), we must have that the computation of M on that inputs stops in at most $p(n)$ steps, so $w \in P$. $\dashv$

Appendix A

A list of tautologies and equivalences in propositional logic

A.1 Tautologies

1. $((A \wedge A) \Leftrightarrow A)$
2. $((A \vee A) \Leftrightarrow A)$
3. $((A \wedge B) \Leftrightarrow (B \wedge A))$
4. $((A \vee B) \Leftrightarrow (B \vee A))$
5. $((A \wedge (B \wedge C)) \Leftrightarrow ((A \wedge B) \wedge C))$
6. $((A \vee (B \vee C)) \Leftrightarrow ((A \vee B) \vee C))$
7. $((A \wedge (B \vee C)) \Leftrightarrow ((A \wedge B) \vee (A \wedge C)))$
8. $((A \vee (B \wedge C)) \Leftrightarrow ((A \vee B) \wedge (A \vee C)))$
9. $((A \wedge (A \vee B)) \Leftrightarrow A)$
10. $((A \vee (A \wedge B)) \Leftrightarrow A)$
11. $(\neg(A \vee B) \Leftrightarrow (\neg A \wedge \neg B))$

12. $(\neg(A \wedge B) \Leftrightarrow (\neg A \vee \neg B))$
13. $((A \wedge \top) \Leftrightarrow A)$, $\top$ any tautology
14. $((A \vee \bot) \Leftrightarrow A)$, $\bot$ any contradictory wff
15. $((A \wedge \bot) \Leftrightarrow \bot)$
16. $((A \vee \top) \Leftrightarrow \top)$
17. $((A \Rightarrow B) \Leftrightarrow (\neg B \Rightarrow \neg A))$
18. $(A \vee \neg A)$
19. $(A \Rightarrow A)$
20. $(A \Leftrightarrow A)$
21. $(\neg\neg A \Leftrightarrow A)$
22. $(A \Rightarrow (A \vee B))$
23. $((A \wedge B) \Rightarrow A)$
24. $(((A \Rightarrow B) \wedge A) \Rightarrow B)$
25. $(((A \Rightarrow B) \wedge \neg B) \Rightarrow \neg A)$
26. $((\neg A \Rightarrow A) \Leftrightarrow A)$
27. $(\neg A \Rightarrow (A \Rightarrow B))$
28. $(A \vee (A \Rightarrow B))$
29. $(A \Rightarrow (B \Rightarrow A))$
30. $(((A \Rightarrow B) \wedge (B \Rightarrow C)) \Rightarrow (A \Rightarrow C))$
31. $((A \Rightarrow B) \vee (C \Rightarrow A))$
32. $((A \Rightarrow B) \vee (\neg A \Rightarrow B))$
33. $((A \Rightarrow B) \vee (A \Rightarrow \neg B))$
34. $((A \Rightarrow B) \Rightarrow ((B \Rightarrow C) \Rightarrow (A \Rightarrow C)))$

35. $(\neg A \Rightarrow (\neg B \Leftrightarrow (B \Rightarrow A)))$

36. $((A \Rightarrow B) \Rightarrow (((A \Rightarrow C) \Rightarrow B) \Rightarrow B))$

A.2 Equivalences

All the formulas in each row are equivalent to each other.

37. $(A \Rightarrow B),\ (\neg A \vee B),\ (\neg B \Rightarrow \neg A),\ ((A \wedge B) \Leftrightarrow A),\ ((A \vee B) \Leftrightarrow B)$

38. $\neg(A \Rightarrow B),\ (A \wedge \neg B)$

39. $(A \Leftrightarrow B),\ ((A \wedge B) \vee (\neg A \wedge \neg B)),\ ((\neg A \vee B) \wedge (\neg B \vee A))$

40. $(A \Leftrightarrow B),\ ((A \Rightarrow B) \wedge (B \Rightarrow A)),\ (\neg A \Leftrightarrow \neg B),\ (B \Leftrightarrow A)$

41. $(A \Leftrightarrow B),\ ((A \vee B) \Rightarrow (A \wedge B))$

42. $\neg(A \Leftrightarrow B),\ (A \Leftrightarrow \neg B),\ (\neg A \Leftrightarrow B)$

43. $A,\ \neg\neg A,\ (A \wedge A),\ (A \vee A),\ (A \vee (A \wedge B)),\ (A \wedge (A \vee B))$

44. $A,\ (\neg A \Rightarrow A),\ ((A \Rightarrow B) \Rightarrow A),\ ((B \Rightarrow A) \wedge (\neg B \Rightarrow A))$

45. $A,\ (A \wedge \top),\ (A \vee \bot),\ (A \Leftrightarrow \top),\ (\top \Rightarrow A)$

46. $\neg A,\ (A \Rightarrow \neg A),\ ((A \Rightarrow B) \wedge (A \Rightarrow \neg B))$

47. $\neg A,\ (A \Rightarrow \bot),\ (A \Leftrightarrow \bot)$

48. $\bot,\ (A \wedge \bot),\ (A \Leftrightarrow \neg A)$

49. $\top,\ (A \vee \top),\ (A \Rightarrow \top),\ (\bot \Rightarrow A)$

50. $(A \wedge B),\ (B \wedge A),\ (A \wedge (\neg A \vee B)),\ \neg(A \Rightarrow \neg B)$

51. $(A \vee B),\ (B \vee A),\ (A \vee (\neg A \wedge B)),\ (\neg A \Rightarrow B),\ ((A \Rightarrow B) \Rightarrow B)$

52. $(A \Rightarrow (B \Rightarrow C)),\ ((A \wedge B) \Rightarrow C),\ (B \Rightarrow (A \Rightarrow C)),\ ((A \Rightarrow B) \Rightarrow (A \Rightarrow C))$

53. $(A \Rightarrow (B \wedge C)),\ ((A \Rightarrow B) \wedge (A \Rightarrow C))$

54. $(A \Rightarrow (B \vee C)),\ ((A \Rightarrow B) \vee (A \Rightarrow C))$

55. $((A \wedge B) \Rightarrow C),\ ((A \Rightarrow C) \vee (B \Rightarrow C))$

56. $((A \vee B) \Rightarrow C),\ ((A \Rightarrow C) \wedge (B \Rightarrow C))$

57. $(A \Leftrightarrow (B \Leftrightarrow C)),\ ((A \Leftrightarrow B) \Leftrightarrow C)$

Appendix B

A list of validities in first order logic

1. $\forall x \forall y A \Leftrightarrow \forall y \forall x A$
2. $\exists x \exists y A \Leftrightarrow \exists y \exists x A$
3. $\forall x A \Rightarrow \exists x A$
4. $\forall x (A \wedge B) \Leftrightarrow (\forall x A) \wedge (\forall x B)$
5. $\exists x (A \vee B) \Leftrightarrow (\exists x A) \vee (\exists x B)$
6. $\neg \forall x A \Leftrightarrow \exists x \neg A$
7. $\neg \exists x B \Leftrightarrow \forall x \neg B$
8. $\forall x (A \Rightarrow B) \Rightarrow (\forall x A \Rightarrow \forall x B)$
9. $\exists x \forall y A \Rightarrow \forall y \exists x A$
10. $\forall x A \Leftrightarrow A$ } if x is not free in A.
11. $\exists x A \Leftrightarrow A$
12. $(\forall x A \Rightarrow B) \Leftrightarrow \exists x (A \Rightarrow B)$ } if x is not free in B.
13. $(\exists x A \Rightarrow B) \Leftrightarrow \forall x (A \Rightarrow B)$
14. $(A \Rightarrow \forall x B) \Leftrightarrow \forall x (A \Rightarrow B)$ } if x is not free in A.
15. $(A \Rightarrow \exists x B) \Leftrightarrow \exists x (A \Rightarrow B)$

16. $x = x$

17. $(x = y \wedge y = z) \Rightarrow (x = z)$

18. $x = y \Rightarrow y = x$

19. $(y_1 = z_1 \wedge \cdots \wedge y_n = z_n) \Rightarrow (f(y_1, \ldots, y_n) = f(z_1, \ldots, z_n))$
    (f any n-ary function symbol)

20. $(y_1 = z_1 \wedge \cdots \wedge y_m = z_m) \Rightarrow (R(y_1, \ldots, y_m) \Leftrightarrow R(z_1, \ldots, z_m))$
    (R any m-ary relation symbol)