LOGIC
for Computer Scientists

Sreejith A. V.
IIT Goa

CONTENTS

PREFACE

The preface tries to answer the question, "Why Logic?"

Every civilization in the world developed complex languages in which they can express facts, feelings etc. These *natural* languages also give freedom to make ambiguous statements. Here is an example.

> *Noam Chomsky: "The fish is ready to eat!"*

The natural understanding is that there is a cooked fish on the table which can now be eaten. The statement could also mean that the fish is hungry and wants to eat. Languages also allow for multiple meanings to a word. This can also lead to ambiguous statements. Someone asking to pick up the mouse, could mean picking up a computer mouse or a real mouse.

Mathematicians realized that one should develop a language for mathematics which does not allow one to make ambiguous statements. There should not be any doubt on the meaning mathematicians give to statements and theorems. In short, there should not be freedom to make ambiguous statements. The *formal* languages were developed to express properties unambiguously. Programming languages like C, C++ fall in this class of languages. In this book, we will see a few others like propositional logic, first order logic etc.

Amartya Sen has a book titled "Argumentative Indian" which looks at Indian history of verbal fights. This, though is not a unique Indian trait. Wherever there were humans they argued. They also thought about what is a correct argument. The

ancient Greeks, Indian, Chinese civilization all contemplated on "how to argue?" Let us start with a statement credited to Socrates.

> *Socrates: "All men are mortal. I am a man. Therefore, I am mortal."*

Intuitively, we all agree to this reasoning of Socrates. Let us look at another argument mentioned in Nyayasutra.

> *Person A: "If there is fire, then there is smoke. There is no smoke on the hill. Therefore there is no fire."*

Compare the above statement with the following wrong argument.

> *Person B: "If there is fire, then there is smoke. There is smoke on the hill. Therefore there is fire."*

Both the arguments says that, "If there is fire, then there will be smoke". Person A, sees no smoke on the hill. He can therefore come to the conclusion that there is no fire. Because, if he assumes there was fire in the hill, it would have created smoke. On the other hand, the statement of Person B is wrong, since the smoke he sees could have been created because of some other reason other than fire.

In the last century, mathematicians observed that there is another fundamental problem with natural languages (other than the freedom to make ambiguous statements). These languages have the ability to make statements which talk about itself. This can lead to Paradoxes. Let us look at a few examples to illustrate this.

> *Barber's Paradox: There is a village with a barber. He shaves every man in the village who does not shave himself.*

The paradox (attributed to Russell) leads to a contradiction when one asks the question, Does the barber shave himself? If he shaves himself, then he shaves someone who shaves himself (a contradiction). On the other hand, if he does not shave himself, he is not shaving a man who does not shave himself (another contradiction). Either way, it leads to a contradiction. The only conclusion we can make of this is, there is no such barber in any village. Russell, being an atheist, took this argument to troll believers

> *Russell: "Can God create a planet, so heavy that he/she cannot lift it?"*

A positive or negative answer to the above question, leads to the conclusion that there are things which God cannot do. The statements we saw above are *self-referential*. That is, those statements talk about itself. But neither of the above statements are a problem. The first statement leads to a contradiction on the existence of a barber who shaves himself, the second on the notion that God can do anything. The following paradox looks correct but is not a well-defined sentence.

> *Berry's Paradox: Let S be the set of natural numbers that can be described by a sentence of less than 100 letters.*

For example, S contains the number "one lakh", "twenty multiplied with one lakh" etc. Clearly S is a finite set, since atmost there S can contain 27^{99} numbers (the 26 letters of the english alphabet and the blank space). Now consider the following number x which is the "smallest positive number which cannot be described by

less than 100 letters". Observe that x is in S because its definition has less than 100 letters. On the other hand, x is not in S because it cannot be described in less than 100 letters. Where are we wrong? The problem is with the definition of S. We allowed for self-reference.

Berry's paradox and similar others lead to a serious introspection on how to define sets in the last century. This resulted in a proper study of set theory and logic. Many remarkable ideas came through. The most prominent among them are the Incompleteness results of Gödel.

To summarize our discussion on the need of formal languages. As we observed above, natural languages can make ambiguous statements and imprecise arguments. We therefore look at Logic for the following two reasons.

1. Write statements unambiguously.

2. Reason correctly.

We are interested in creating a language in which mathematical proofs can be written. The language should be simple enough so that vague and ambiguous statements cannot be written. On the other hand, it should be powerful enough for proofs to be written. We will build a language in which all of mathematics can be embedded.

PART I

PROPOSITIONAL LOGIC

CHAPTER 1

SYNTAX AND SEMANTICS

1.1 Syntax: the rules

We begin with propositional logic, the foundation on which other logics are built on.

We say that *true* and *false* are *boolean values*. They will be denoted by the symbols T and F respectively. *Declarative sentences* are statements which can be assigned either true or false. For example,

1. Gravity attracts light.
2. 179179 is a composite number.
3. Sun rises in the east and Japan is in Europe
4. Kattapa killed Bahubali.
5. P = NP
6. John Snow's father is Ned Stark

Since the number 179179 is divisible by 7,11,13 and 179, statement (1) is true. On the other hand statement (2) is false since Japan is not in Europe. Statement (3) is true. We do not yet know whether Statement (4) is true or false but the statement as such can be assigned true or false. Statement (5) is true or false depending on till which episode you have watched game of thrones.

The following statements are not declarative

1. What is the time?
2. Submit your assignments today.
3. Teek Hai!

Some declarative statements can be thought of as *atomic* . That is, those statements cannot be further split into logical sentences. In the above example Statement (2) is not atomic because it can be split into sentences "Sun rises in the east" and "Japan is in Europe".

As mathematicians we are not interested in identifying whether declarative sentences are true or not in reality. That is, we are not interested in finding out if "Gravity attracts light". We leave this to physicists. Rather we are interested in only what we can infer from the fact that such a statement is true or false. For example, consider the following statement.

"If it rains in Delhi today, you will fail in logic course."

This is a weird statement but the statement is a declarative statement. That is, it can either be true or false. To summarize the point being made. We are not really interested in the weirdness or the reality of the real world. We are only interested in statements which can be assigned boolean values.

Hence we can replace atomic sentences by symbols called *propositions* (or propositional variables) We will usually denote propositions by $p, q, r, \ldots$ or $p_1, p_2, \ldots$. Propositions can be combined using the following *logical connectives* or *logical symbols* .

$$(,), \neg, \wedge, \vee, \Longrightarrow$$

By repeatedly using logical connectives we can make complex declarative statements. A *formula* is a word over the propositions and logical symbols. For example, $p \Longrightarrow, (p \wedge q)$ are formulas. Consider the following sentence.

If x is prime and $x \neq 2$, then x is odd

The two declarative sentence"x is prime" and "$x \neq 2$" implies the third "x is odd". In other words this statement can be wrriten as follows

("x is prime" $\wedge$ "$x \neq 2$") $\Longrightarrow$ "x is odd"

If the propositions p, q, and r stands for "x is prime", "$x \neq 2$" and "x is odd", then the statement can be written in propositional logic as

$$((p \wedge q) \Longrightarrow r)$$

This way we can build complex logical statements. This logic build using the propositions and the logical symbols will be called *propositional logic*. We denote propositional formulas by greek letters like $\alpha, \beta, \gamma, \ldots$ etc or $\alpha_1, \alpha_2, \ldots$ etc. Just like natural

languages have rules to build meaningful sentences, to write complex statements in propositional logic, one needs to follow a set of rules. These rules are called the *syntax* of propositional logic. A formula which is build by using these rules is called a well-formed formula (wff). To formally define wffs we need to first understand *inductive definition* .

Definition 1.1 (inductive definition) *There are three requirements for an inductive definition.*

1. *A universal set, denoted by U.*
2. *A core set (usually finite) denoted by C such that $C \subseteq U$.*
3. *A finite set of functions (denoted by $\mathcal{F} = \{f_1, f_2, \dots, f_k\}$) such that for all $f \in \mathcal{F}$, f is a function from tuples of U to U. That is $f : U \times U \times \cdots \times U \longrightarrow U$.*

We say that the set inductively defined by C and $\mathcal{F}$ is the smallest *set which contains the core set C and is closed under all the functions $f \in \mathcal{F}$. This set will be denoted as* [1]*:*

$$\mathcal{T}(U, C, \mathbb{F}) = \{\textit{the smallest set which contains } C \textit{ and is closed under all } f \in \mathcal{F}\}$$

Note on inductive defintion: For sets X and U, we say that $X \subseteq U$ is closed under a function $f : U^n \longrightarrow U$, if for every $a_1, a_2, \dots, a_n \in X$, we have that $f(a_1, a_2, \dots, a_n) \in X$.

Let us look at an example for inductive definition.

EXAMPLE 1.1

We want to define the set of all your blood relatives. We can inductively define this using $\{you\}$ as the core set and the functions, $\mathcal{F} = \{$*son of*, *daughter of*, *first child of*, and *immediate younger sibling of*$\}$. That is,

$$\text{your relatives} = \mathcal{T}(\text{Humans}, \{\text{you}\}, \mathcal{F})$$

This lemma is important for the understanding of inductively defined sets.

Lemma 1.1 *For any $U, C, \mathcal{F}$, we have that*

$$\mathcal{T}(U, C, \mathcal{F}) = \bigcap \{X \subseteq U \mid C \subseteq X \textit{ and } X \textit{ is closed under } f \in \mathcal{F}\}$$

We can now define well formed formulas inductively (also see Figure 1.1).

Definition 1.2 (well formed formulas) *A* well-formed formula *(wff) is inductively defined as the smallest set of formulas which contains*

[1] the reason for choosing this notation will be clear when we talk about natural deduction

$$\frac{\alpha \qquad \beta}{(\alpha \wedge \beta)}\ \wedge \qquad\qquad \frac{\alpha \qquad \beta}{(\alpha \vee \beta)}\ \vee$$

$$\frac{\alpha \qquad \beta}{(\alpha \Longrightarrow \beta)}\ \Rightarrow$$

$$\frac{\alpha}{(\neg\alpha)}\ \neg$$

Figure 1.1 Pictorial representation for the operators $\neg, \vee, \wedge$ and $\Longrightarrow$

1. *the propositions and*
2. *If α and β are wffs then $(\neg\alpha), (\alpha \vee \beta), (\alpha \wedge \beta), (\alpha \Longrightarrow \beta)$ are wffs.*

In other words

$$wff = \mathcal{T}(all\ formulas, propositions, \{\neg, \vee, \wedge, \Longrightarrow\})$$

Here are some examples of wffs: $(\neg p_1), (p_1 \Longrightarrow (p_2 \wedge p_3)), (p_1 \vee (p_2 \wedge p_3))$.

Definition 1.3 *A* construction sequence *for a wff α is a finite sequence $\alpha_1, \alpha_2, \ldots, \alpha_n = \alpha$ such that each α_i is either a proposition or is a result of applying one of the rules of wff to α_j, α_k where $j, k < i$.*

Here is an example.

EXAMPLE 1.2

A construction sequence for the formula $(p_1 \Longrightarrow (p_2 \wedge p_3))$ is

$$p_2, p_3, (p_2 \wedge p_3), p_1, (p_1 \Longrightarrow (p_2 \wedge p_3))$$

Here is another sequence for the same formula.

$$p_1, p_2, p_3, (p_2 \wedge p_3), (p_1 \Longrightarrow (p_2 \wedge p_3))$$

Thus a wff can have multiple construction sequences. The following claim follows from the definition of wff.

Claim 1.2 *Every wff formula α has a construction sequence.*

Thus a "proof" of a formula α being a wff is a construction sequence for it. Another interesting question is, what is the proof of a formula α not being a wff? The idea is to identify a property which is satisfied by all wffs and not satisfied by α. But, how do you show that a property is satisfied by all wffs. The answer is *structural induction* .

Claim 1.3 *The formula* $(p_1 \wedge p_2$ *is not a wff.*

Proof : We show that all wffs satisfy the following property.

Induction hypothesis: For any wff α, the number of left brackets is equal to the number of right brackets.

The proof is by structural induction. It is easy to note that the hypothesis is true for the base case (which is a proposition). So, let us assume that the hypothesis is true for α and β. We need to show that the hypothesis holds for $(\neg\alpha)$, $(\alpha \vee \beta)$, $(\alpha \wedge \beta)$ and $(\alpha \implies \beta)$. It is easy to observe that this is indeed the case. Therefore the number of left brackets in a wff is equal to the number of right brackets.

Since $(p_1 \wedge p_2$ does not satisfy this property, it is not a wff. ■

Structural Induction: This is equivalent to ordinary induction. But this form of induction is more suited to inductively defined sets. In other words, if you want to show that a property holds for all elements $x \in \mathcal{T}(U, C, \mathcal{F})$, a property holds for x, you need to do the following.

1. Show that the property holds for all elements $x \in C$ and
2. Show that if $a_1, a_2, \ldots, a_n$ satisfies this property then for all $f \in \mathcal{F}$, the element $f(a_1, a_2, \ldots, a_n)$ also satisfies this property.

The following algorithm applies the above property inductively to check if a given formula is a wff or not.

Problems

1.1 Can you give an inductive definition of

1. The set of all positive integers.
2. The set of all integers.
3. The set of all even numbers.

1.2 Prove the following using structural induction.

1. The number of left brackets is greater than the number of right brackets for any strict prefix of a wff.

Algorithm 1 Check if a formula is a wff or not

Input A formula α (word over symbols Σ)
Output: YES if α is a wff, otherwise NO

```
1:  function IS-WFF(α)
2:      if (α is a proposition) then return YES                    ▷ α is a wff
3:      if (α does not have equal number of opening and closing brackets) then
4:          return NO                                              ▷ α is not a wff
5:      end if
6:      if (α is of the form (¬β)) then return IS-WFF(β)
7:      Let ~∈ {∧, ∨, ⟹}.
8:      if α is (β ~ γ) and β has equal number of opening and closing brackets then
9:          if ((IS-WFF(β) = YES) and (IS-WFF(γ) = YES) then return YES
10:     end if
11:     return NO                                                  ▷ All other case α is not a wff
12: end function
```

1.3 Suppose α is a wff which doesn't use any negation symbol, show that the length of α is odd.

1.4 Show that all the wffs have balanced parenthesis.

1.5 Let S be a set of all subformulas of α. Prove that $|S| \leq |\alpha|$. (here, $|\alpha|$ denotes the length of the formula α).

1.6 Define recursively the following notions about propositional formulas.

1. $Atoms(\alpha)$ is the set of all propositions occurring in α.
2. $SF(\alpha)$ is the set of all sub formulas of α.
3. $|\alpha|$ denotes the length of the formula.

1.2 Semantics: the meaning

We will denote true by T and false by F. The logical connectives we will be interested in and their *semantics* follow

Negation For any sentence, its negation is the opposite of it. For example, the negation of "Kattapa killed Bahubali" is "Kattapa did not kill Bahubali". Similarly negation of "P=NP" is "P $\neq$ NP". The negation of a propositional symbol, p is denoted as $\neg p$ (and called negation of p). If the p is true then $\neg p$ is false. On the other hand, if p is false, $\neg p$ is true. The following *truth table* summarizes the semantics of negation.

p	$\neg p$
T	F
F	T

Table 1.1 Truth table for negation

Conjunction stands for *and*. When it is used to connect two declarative sentences it means that both the sentences are true. For example, "2 is a prime number" and "2 is an even number". We use the symbol $\wedge$ to denote and. The following truth table captures the meaning of $\wedge$.

p	q	$p \wedge q$
T	T	T
T	F	F
F	T	F
F	F	F

Table 1.2 Truth table for conjunction

Disjunction stands for *or*. Disjunctions are slightly different from the or we use in English (or most natural languages). Assume that I made the following announcement in class.

> *"Tomorrow there will be a lecture or an exam."*

It is natural for you to assume that there will either be a lecture tomorrow and no exam, or there will be an exam and no lecture. That is, you would never think of the possibility of both a lecture and an exam going to be held tomorrow. But, or used in a mathematical sentence, can mean both of them happening. This is the difference of disjunction in or in logic and natural language. We have a name for the or in natural language. We call it xor and is denoted by the symbol $\oplus$. See Exercise 1.4. Coming back to disjunction in logic, the symbol for or is $\vee$ and its semantics is given by the following truth table.

p	q	$p \vee q$
T	T	T
T	F	T
F	T	T
F	F	F

Table 1.3 Truth table for disjunction

The semantics for xor is given below.

p	q	$p \oplus q$
T	T	F
T	F	T
F	T	T
F	F	F

Table 1.4 Truth table for xor

Implication is used to state a necessary condition. It is denoted by the symbol $\Longrightarrow$. The statment

If x is prime, then $x \neq 4$

is an example of an implication. Typically it is written in the form "If p then q" and in propositional logic as $p \Longrightarrow q$. Its truth table is given in Table 1.5.

p	q	$p \Longrightarrow q$
T	T	T
T	F	F
F	T	T
F	F	T

Table 1.5 Truth table for implication

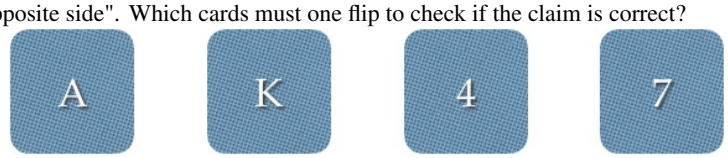
We will explain the truth table with an example. Consider the statement

"If you work hard, you get good grades".

The statement is true if you work hard and got good grade. On the other hand it is false if you work hard and did not get good grades. The tricky question is, what if you did not work hard and got good grades. Does this violate our the statement? No, it doesnt and hence we can assign the statement to be true in this case also. Similarly the statement is true, if we did not work hard and did not get good grades.

Implication is a little tricky to understand especially for beginners. Take a look at the following puzzle.

Figure 1.2 Wason's Puzzle: "If a card has a vowel on one side then it has an even number on its opposite side". Which cards must one flip to check if the claim is correct?



PUZZLE 1.2.1

This puzzle is due to Wason. The following cards are kept on the table (see Figure 1.2.). Each card has a letter on one side and a number on the other side. We make the following claim: "If a card has a vowel on one side then it has an even number on its opposite side". Which cards must one flip to check if the claim is correct?

Answer

The answer is cards A and 7. Let us look at each of the card and check whether we need to flip or not. Card A has to be flipped, because we need to be sure that on the opposite side is a vowel. Card K need not be checked. Why? Because, it does not matter to us if, the opposite side has an even number or odd number. Similarly we do not have to flip 4 since it does not matter to us whether the opposite side was vowel or consonant. On the other hand, card 7 has to be flipped because we have to make sure it is not a vowel on the other side. Because if there was a vowel, we would have violated the claim.

We can write truth tables for wffs by inductively building the table. Consider the following formula and its truth table: $\alpha ::= \big((p \Rightarrow \neg q) \Rightarrow (q \vee \neg p)\big)$

p	q	$\neg p$	$\neg q$	$p \Rightarrow \neg q$	$q \vee \neg p$	α
T	T	F	F	F	T	T
T	F	F	T	T	F	F
F	T	T	F	T	T	T
F	F	T	T	T	T	T

Table 1.6 Truth table for $\alpha ::= \big((p \Rightarrow \neg q) \Rightarrow (q \vee \neg p)\big)$

EXAMPLE 1.3

If and only if (written also as *iff*) is used to denote a necessary and sufficient condition. In symbolic form it is denoted by $\Leftrightarrow$. Semantically $p \Leftrightarrow q$ is equivalent to $p \Rightarrow q$ and $q \Rightarrow p$. The truth table is given below.

p	q	$p \Leftrightarrow q$
T	T	T
T	F	F
F	T	F
F	F	T

Table 1.7 Truth table for if and only if

1.3 Boolean functions

A function f is called a *boolean function* if $f : \{\mathrm{T},\mathrm{F}\}^n \longrightarrow \{\mathrm{T},\mathrm{F}\}$ takes as input T or F values and outputs T or F. We can see every wff as a boolean function. What is the boolean function associated with a wff α? The truth table gives this function. For example the boolean function associated with the wff $(p \wedge q)$ is the function which takes (T, T) to T and all other inputs to F. An interesting question to ask is, can all boolean functions be captured by wffs.

Lemma 1.4 *Let $f : \{T,F\}^n \longrightarrow \{T,F\}$ be an arbitrary boolean function. Then there exists a wff α such that the boolean function associated with α is f.*

Proof : ■

We say that two formulas α and β are *equivalent* if for all assignments v from propositions in α and β to boolean values, we have that $\hat{v}(\alpha) = \hat{v}(\beta)$. In other words, the truth table is the same for both the formulas. If α and β are equivalent, we will denote it as $\alpha \equiv \beta$.

We list some very important equivalent formulas below and leave the proof of equivalence to the reader.

1. (De morgan's law) $(\alpha \wedge \beta) \equiv \neg(\neg\alpha \vee \neg\beta)$
2. (De morgan's law) $(\alpha \vee \beta) \equiv \neg(\neg\alpha \wedge \neg\beta)$
3. (double negation) $\neg(\neg\alpha) \equiv \alpha$
4. (contrapositive) $(\alpha \Longrightarrow \beta) \equiv (\neg\beta \Longrightarrow \neg\alpha)$.

Definition 1.4 (adequate functions) *A set of boolean functions is* adequate *(also called as* complete *) if compositions of these functions can express any boolean function.*

To prove that a set of functions is adequate we need to show that any truth table can be expressed using a combination of the functions. The following claim is left to the reader to prove.

Claim 1.5 *The boolean functions represented by $\{\neg, \vee, \wedge\}$ is adequate. It follows that, $\{\neg, \wedge\}$ is also adequate.*

Here is another claim which can be proved by induction.

Claim 1.6 *A set of boolean functions is adequate if and only if it can express any binary boolean function.*

We know how to show that a set is adequate. How do we show that a set is not adequate. We need to come up with a property which is not satisfied by some boolean function.

Claim 1.7 *The set* $\{\wedge, \vee, \Rightarrow\}$ *is not adequate.*

Proof: We show by that: Every function f definable by these boolean functions satisfies the condition that $f(\mathrm{T}, \mathrm{T}, \ldots, \mathrm{T}) = \mathrm{T}$. Showing this claim will prove that the set is not adequate, since the boolean function defined by $(\neg p)$ does not satisfy this condition.

The proof is by structural induction. It is clear that propositions satisfy this claim. Let us assume that α and β satisfy this claim. It follows from the semantics of $\wedge$, $\vee$ and $\Rightarrow$ that $(\alpha \wedge \beta)$, $(\alpha \vee \beta)$ and $(\alpha \Rightarrow \beta)$ also satisfies the claim. ∎

Problems

1.1 Show the following.

1. The symbols $\{\wedge, \Leftrightarrow, \oplus\}$ form a complete set.
2. No strict subset of $\{\wedge, \Leftrightarrow, \oplus\}$ form a complete set.

1.2 Check whether the following are a complete set or not.

1. $\{\Rightarrow, \neg\}$
2. $\{\vee, \wedge, \Rightarrow, \Leftrightarrow\}$
3. The NAND operator.
4. The NOR operator.

1.4 Types of formulas

1.4.1 Semantics based special formulas

Let $\mathcal{P} = \{p_1, p_2, \ldots, p_n\}$ be a set of propositions and let α be a wff over $\mathcal{P}$. Then an *assignment* $v : \mathcal{P} \longrightarrow \{\mathrm{T}, \mathrm{F}\}$ is a mapping from the propositions to T or F. Since a proposition can be assigned only one of two values, there are 2^n different assignments possible for n propositions.

EXAMPLE 1.4

Consider the formula $(p \wedge q)$. One particular assignment is p is mapped to T and q to F.

We can extend an assignment to the propositions to a function from wffs to boolean values as follows. Let $v : \mathcal{P} \longrightarrow \{\mathrm{T}, F\}$ be an assignment for the propositions $\mathcal{P}$. We extend this to $\hat{v} :$ wff $\longrightarrow \{\mathrm{T}, \mathrm{F}\}$ which is inductively defined as follows.

1. $\hat{v}(\alpha) = v(\alpha)$ if α is a proposition.
2. $\hat{v}(\neg\beta) = \mathrm{T}$ if and only if $\hat{v}(\beta) = F$
3. $\hat{v}(\alpha \wedge \beta) = \mathrm{T}$ if and only if $\hat{v}(\alpha) = \mathrm{T}$ and $\hat{v}(\beta) = \mathrm{T}$
4. $\hat{v}(\alpha \vee \beta) = \mathrm{T}$ if and only if $\hat{v}(\alpha) = \mathrm{T}$ or $\hat{v}(\beta) = \mathrm{T}$
5. $\hat{v}(\alpha \Longrightarrow \beta) = \mathrm{F}$ if and only if $\hat{v}(\alpha) = \mathrm{T}$ and $\hat{v}(\beta) = \mathrm{T}$

We say that an assignment v *satisfies* a wff α if $\hat{v}(\alpha) = \mathrm{T}$. A wff α is *satisfiable* if there exists an assignment v such that v satisfies α. If no such assignment exists, then the formula is said to be not satisfiable (also called *contradiction* or *unsatisfiable*). A formula α is valid (*tautology*) if all possible assignments satisfies α. Here are a few examples.

EXAMPLE 1.5

1. The formula $\neg p$ is satisfiable. The assignment $v(p) = \mathrm{F}$ satisfies $\neg p$.
2. The formula $p \vee \neg p$ is valid.
3. The formula $p \wedge \neg p$ is not satisfiable.

We leave the reader to prove the following claim.

Claim 1.8 *The following statements hold for all wffs α.*

1. *α is valid if and only if $\neg\alpha$ is not satisfiable.*
2. *α is not valid and satisfiable if and only if $\neg\alpha$ is not valid and satisfiable.*

The following diagram represents the various types of well formed formulas.

1.4.2 Normal Forms: Syntax based special formulas

A *literal* is a proposition or a negation of a proposition. We can represent it in grammar as follows

$$literal ::= p \mid (\neg p)$$

A *conjunctive clause* is a conjunction of literals and a *disjunctive clause* is a disjunction of literals.

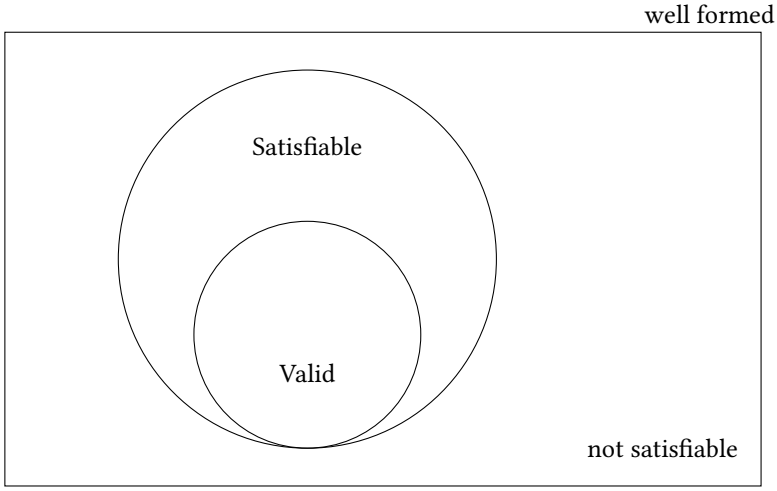


Figure 1.3 Types of wffs

Definition 1.5 (DNF) *A formula α is said to be in Disjunctive normal form (DNF) if α is disjunctions of conjunctive clauses.*

$$C ::= literal \mid (C \wedge C)$$

$$DNF ::= C \mid (C \vee C)$$

For example the formula $(p_1 \wedge \neg p_2) \vee (p_2 \wedge p_3) \vee (\neg p_1 \wedge \neg p_3)$ is in DNF.

Definition 1.6 (CNF) *A formula α is said to be in Conjunctive normal form (CNF) if α is conjunctions of disjunctive clauses.*

$$D ::= literal \mid (D \vee D)$$

$$CNF ::= D \mid (D \wedge D)$$

The formula $(p_1 \vee p_2) \wedge (\neg p_1 \vee \neg p_2)$ is in CNF. We will now look at special CNF formulas. A k-CNF formula is a conjunction of clauses with at most k-literals. For example, the following is a 2-CNF formula

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

It is also a k-CNF formula for all $k \geq 2$. On the other hand, this is not a 2-CNF formula because it has a disjunction of 3 literals.

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

The above formula though is a 3-CNF formula. The set of all 3-CNF formulas is a subset of the set of all CNF formulas.

The following fact about CNF and DNF formulas make them interesting. It says that there exist equivalent formulas in CNF and DNF for any wff.

Lemma 1.9 *For any wff α, there is a CNF formula β and a DNF formula γ such that $\alpha \equiv \beta$ and $\alpha \equiv \gamma$.*

Proof: We prove the existence of both β and γ by structural induction. The case when α is a proposition satisfies the lemma trivially. The inductive step .... (need to be filled) ■

The above lemma can be made into an algorithm to output an equivalent CNF (or DNF formula) when a wff is given as input. Exercise 1.3 asks you to give the algorithm and compute its running time. You would observe that, the worst case running time of the algorithm is exponential in the input size (2^n when n is the size of the wff). As of now, we do not know of a polynomial time running algorithm to do this. In fact, theoretical computer scientists believe this is not possible. This corresponds to the famous $P \neq NP$? problem introduced in the next chapter.

Problems

1.1 Show that the following formulas are equivalent.

1. $p \implies q$
2. $(\neg q \implies p)$
3. $\neg p \vee q$

What is the complement of $p \implies q$?

1.2 [DeMorgan's law] Show that $p \wedge q \equiv \neg(\neg p \vee \neg q)$ and $p \vee q \equiv \neg(\neg p \wedge \neg q)$

1.3 Give an algorithm which outputs the equivalent CNF and DNF formula when a wff is given as input. Also compute the worst case running time of your proposed algorithm.

1.4 The following exercise connects xor and disjunctions. Write a formula using only the symbols disjunctions and negations which is equivalent to $p \oplus q$.

1.5 Consider you are given a truth table with propositions $p_1, \ldots, p_n$. Give an algorithm which outputs a formula (using only symbols $\vee, \wedge, \neg$) having the same truth table.

These exercise (along with demorgan's law) show that any formula can be converted into an equivalent formula which uses only the symbols $\wedge$ and $\neg$.

1.6 Convert any formula into an equivalent formula which uses only symbols $\wedge$ and $\neg$.

1.7

1. Give a formula which is a tautology?

2. Give a formula which is satisfiable but not a tautology?
3. Give a formula which is a contradiction?

1.8 Prove the following statements.

1. $\Gamma \models \alpha$ iff $\Gamma \subseteq \{\neg\alpha\}$ is unsatisfiable.
2. $\Gamma \models \alpha \Longrightarrow \beta$ iff $\Gamma \subseteq \{\alpha\} \models \beta$

1.9

1. Write an algorithm to convert a formula in DNF to CNF.
2. Give a polynomial time algorithm to check whether a DNF formula is satisfiable or not.
3. Give an algorithm to check whether a CNF formula is satisfiable or not. How much time does it take?

1.10 Prove that for every formula α there exists formulas β in disjunctive normal form and γ in conjunctive normal form such that $\alpha \equiv \beta$ and $\alpha \equiv \gamma$.

1.11 Give an example of propositional formula α of size n such that converting it to a CNF will lead to exponential size formula.

1.12 Construct CNF formulas equivalent to α_1 to α_n in Table 1.8.

p	q	r	α_1	α_2	α_3	α_4
F	F	F	T	T	F	T
F	F	T	F	F	T	F
F	T	F	F	T	T	T
F	T	T	F	T	F	F
T	F	F	T	F	T	F
T	F	T	T	T	F	F
T	T	F	F	F	F	T
T	T	T	T	F	T	T

Table 1.8 Construct a CNF formula for the following truth functions.

1.13 Construct DNF formulas equivalent to α_1 to α_n in Table 1.8.

1.14 Check whether the following formulas are valid/satisfiable/unsatisfiable.

1. $((p \vee q) \Rightarrow p)$
2. $((p \wedge q) \Rightarrow p)$
3. $((p \Rightarrow q) \Rightarrow q)$
4. $((\neg(\neg p)) \Rightarrow p)$
5. $(p \Rightarrow (p \vee q))$
6. $(p \Rightarrow (p \wedge q))$
7. $(p \Rightarrow (p \Rightarrow q)) \Rightarrow (p \Rightarrow q)$
8. $((p \Rightarrow r) \Rightarrow (q \Rightarrow r)) \Rightarrow ((p \vee q) \Rightarrow r)$
9. $((p \Rightarrow r) \Rightarrow ((\neg p \Rightarrow r)) \Rightarrow r$

1.15 Prove or Disprove the following statements

1. If a formula is valid, then it is satisfiable.
2. If a formula α is unsatisfiable, then $(\neg\alpha)$ is valid.
3. If a formula is satisfiable, then it is valid.
4. If a formula is valid, then it is not unsatisfiable.
5. A formula, say α, is satisfiable, then $(\neg\alpha)$ is unsatisfiable.

1.5 Applications of propositional logic

We use propositional logic to encode different problems.

1.5.1 Reasoning

Consider the following statement by Socrates: "If I am guilty, I must be punished. I'm guilty. Therefore, I must be punished." How do we encode this reasoning in propositional logic. Let propositions p and q stand for the statements "I am guilty" and "I must be punished" respectively. Therefore the entire reasoning can be expressed by the wff

$$\alpha ::= \big((p \Rightarrow q) \wedge p) \Rightarrow q\big)$$

Consider a wrong reasoning: "If I am guilty, I must be punished. I'm not guilty. Thus I must not be punished." By the same propositions, this reasoning can be expressed as

$$\beta ::= \big((p \Rightarrow q) \wedge \neg p) \Rightarrow \neg q\big)$$

What is special about the formula α but not β. Can the wffs tell us one is correct reasoning, whereas the other is not. In fact it does. α is a valid formula, whereas β is not a valid formula. A reasoning will always lead to a valid formula.

1.5.2 Digital logic

EXAMPLE 1.6

We can view a proposition being assigned to 1 or 0 (in place of T or F). That is a proposition p can be thought of as a bit variable. Extending the idea, a valuation to a n proposition symbols can be thought of as an n-bit number. Use this view to write a formula to encode addition relation between two n-bit numbers. That is, write a formula which satisfies the following condition

$$\begin{array}{r} p_1 p_2 \cdots p_n \\ + \\ q_1 q_2 \cdots q_n \\ = \\ r_1 r_2 \ldots r_n \end{array}$$

View $p_1, \ldots, p_n, q_1, \ldots, q_n, r_1, \ldots, r_n$ as propositions.

Problems

1.1 [Pigeon hole principle] If $n + 1$ pigeons are placed on n holes, atleast one hole will have more than one pigeon. This is the pigeon hole principle. Now, consider $n + 1$ pigeons and n holes. Let proposition $p_{i,j}$ denote the fact that the i^{th} pigeon is in the j^{th} hole. Write a propositional logic formula to encode the pigeon hole principle.

1.2 [puzzles from Smullyan] Use propositional logic to answer the following questions

1. You are trapped in a room. There are two doors. Either the doors lead to an exit or to a lion (note that both leading to an exit or to a lion are also possible). In Door 1, it is written "This door is exit and other door leads to lion". In Door 2, it is written "One of the rooms lead to exit, the other to a lion". You are told that only one of the written statements are true and the other false. Which door would you choose?

2. Similar to the previous question. You are trapped in a room. There are two doors. Either the doors lead to an exit or to a lion (note that both leading to an exit or to a lion are also possible). In Door 1, it is written "Atleast one of the doors is an exit". In Door 2, it is written "There is a lion on the other door". You are told that either both are true statements or both are false. Which door would you choose?

3. One more question with same flavour. You are trapped in a room. There are two doors. Either the doors lead to an exit or to a lion (note that both leading

to an exit or to a lion are also possible). In Door 1, it is written “This door is exit or the other door leads to lion". In Door 2, it is written “The exit is the other door". The statements are either both true or both false. Which door would you choose?

1.3 [Nyayasutra] Are the following arguments correct? Write the statements using entails.

1. If there is smoke, then there is fire. There is smoke on hill. Therefore, there is fire.
2. Fire causes smoke. There is smoke on hill. Therefore, there is fire.
3. If there is smoke, then there is fire. There is no smoke. Therefore, there is no fire.

1.6 Semantic entailment

In the previous sections, we saw the semantics of formulas. The truth table of a formula tells us for what valuation makes the formula true or false. In this section, we are interested in a relationship between formulas. Let us first define *semantic entailment* in its simpler form. Consider two formulas α and β.

Definition 1.7 *We say that α semantically entails β by the following notation.*

$$\alpha \models \beta$$

It means that, for all valuations which make α true, β is also evaluated to true.

We can see semantic entailment in a non-mathematical setting as follows: In all the worlds where α is true, β is also true. Let us look at an example

$$\text{Planets have mass} \models \text{There is gravity in planets}$$

The example says that: Consider a world where planets have mass. Then planets will also show gravity. In short, “Planets have mass" semantically entails the statement “There is gravity in planets". This is a fact of the universe we live in. The above example was used to press the meaning of $\models$ and is not really a good example for mathematicians.

Mathematicians require precise definitions like definition 1.7. Below we generalise this. Let Γ be a set of formulas and β a formula. Then

$$\Gamma \models \beta$$

denotes that for all valuations which make all formulas in Γ true, we have β is true. Let us understand when Γ is finite. That is $\Gamma = \{\alpha_1, \alpha_2, \ldots, \alpha_n\}$ for some $n \in \mathbb{N}$. We take a little liberty in writing $\{\alpha_1, \alpha_2, \ldots, \alpha_n\} \models \beta$ as

$$\alpha_1, \alpha_2, \ldots, \alpha_n \models \beta$$

That is, we skip the set notation when it is clear to the reader. The following exercises show the relation between semantic entitlement in the finite case and the implication relation.

Let us go back to the case when Γ is infinite. There does not exist equivalent definitions like those in Exercise 1.3. This is because infinite implication or conjunction is not allowed in our logic.

There is one tricky case, we will elaborate on. Consider an arbitrary formula α. Then

$$\mathrm{F} \models \alpha$$

Let us go through the definition of semantic entailment. It says that for all valuations which make the left hand side (here it is F) true, we have that α is true. This is correct, since there is no valuation which makes F true. In other words, since there is no valuation which make F true, the statement is *vacuously true* . What this means is that $\mathrm{F} \models \neg\alpha$ and $\mathrm{F} \models \alpha$. In short, if falsity is true, then anything is true.

Problems

1.1 Explain what the following statement mean?
If $\mathrm{T} \vdash \beta$, then $\mathrm{T} \models \beta$.

1.2 Show that the following statements are equivalent.

1. $\alpha \models \beta$
2. $\mathrm{T} \models (\alpha \implies \beta)$

1.3 Extend the argument in the previous exercise, and show that the following statements are equivalent.

1. $\alpha_1, \alpha_2, \ldots, \alpha_n \models \beta$
2. $\mathrm{T} \models \Big(\alpha_1 \implies \big(\alpha_2 \implies \ldots (\alpha_n \implies \beta)\big)\Big)$
3. $\mathrm{T} \models (\alpha_1 \wedge \alpha_2 \wedge \ldots \alpha_n) \implies \beta$

1.7 Compactness*

We say that a set Γ (finite or infinite) of propositional formulas is satisfiable if there is a valuation which makes all formulas in Γ true. The Problem 1.1 answers when a finite set satisfiable.

The interesting question is, when is an infinite set satisfiable? The compactness theorem says that if a set is unsatisfiable, then there is a finite set which is unsatisfiable.

Theorem 1.10 (Compactness) *Γ is satisfiable if and only if for all finite subsets $Y \subseteq \Gamma$, Y is satisfiable.*

Proof: The forward direction of the proof is easy to see. Let us therefore show the other direction. We assume Γ is unsatisfiable and identify a finite set which is unsatisfiable. Let us first enumerate the propositions used by the formulas in Γ as

$$p_1, p_2, \ldots$$

We add a new proposition p_0 (this is added just for ease of explanation and is not fundamental to the proof) to this list. Now, we build an complete binary tree where every node in the tree corresponds to a valuation for some finite set of propositions. To be precise, a node at height t in the tree corresponds to a valuation for all propositions $\{p_0, p_1, \ldots, p_t\}$. In short every node at height t corresponds to a function $v : \{p_0, p_1, \ldots, p_t\} \longrightarrow \{\mathrm{T}, \mathrm{F}\}$. We will inductively define the valuation corresponding to every node. The root node (at height 0) corresponds to the function $v : \{p_0\} \longrightarrow \{\mathrm{T}, \mathrm{F}\}$, such that $v(p_0) = \mathrm{T}$. Consider an arbitrary node at height t with valuation $v : \{p_1, \ldots, p_t\} \longrightarrow \{\mathrm{T}, \mathrm{F}\}$. The valuation of its children extends v as follows. The left node corresponds to the valuation $v_l : \{p_1, \ldots, p_{t+1}\} \longrightarrow \{\mathrm{T}, \mathrm{F}\}$ where $v_l(p_{t+1}) = \mathrm{T}$ and for all other propositions p_i where $i \leq t$, $v_l(p_i) = v(p_i)$. Similarly the right node corresponds to $v_r : \{p_1, \ldots, p_{t+1}\} \longrightarrow \{\mathrm{T}, \mathrm{F}\}$ such that $v_r(p_{t+1}) = \mathrm{F}$ and for all other propositions p_i where $i \leq t$, $v_r(p_i) = v(p_i)$.

We now trim the above infinite tree as follows. Take a formula $\alpha \in \Gamma$. If a valuation v does not satisfy α, then remove all descendants of the node corresponding to v (but keep the node v). Note that, if v does not satisfy Γ then any extension of v also does not satisfy Γ. We do the above trimming for all formulas $\alpha \in \Gamma$. We claim that, the trimmed tree is a finite tree. Assume not. Then there exists an infinite path in the tree. We claim that this represents a valuation $v : \{p_1, p_2, \ldots\} \longrightarrow \{\mathrm{T}, \mathrm{F}\}$ which satisfies all formulas in Γ. Assume not. Then there exists a formula $\alpha \in \Gamma$ such that v does not satisfy α. Let α be over propositions $\{p_1, \ldots, p_t\}$. There is a valuation $v' : \{p_1, \ldots, p_t\} \longrightarrow \{\mathrm{T}, \mathrm{F}\}$ which extends to v, such that v' does not satisfy α. This leads to a contradiction. Hence the trimmed tree is a finite tree.

Let $V = \{v_1, v_2, \ldots, v_k\}$ be the set of all valuations in the leaf of the trimmed tree. Therefore any valuation v extends atleast one of the v_is in V. Now, for each $v_i \in V$ we pick one $\alpha_i \in \Gamma$ such that v_i does not satisfy α_i. Call this set $Y = \{\alpha_1, \alpha_2, \ldots, \alpha_k\}$. We claim Y is not satisfiable. Assume not. Then there exists a valuation $v : \{p_0, \ldots, p_t\} \longrightarrow \{\mathrm{T}, \mathrm{F}\}$ which satisfies Y and t is the height of the

trimmed tree. Since the tree is finite there is an extension v_i of v which does not satisfy formula $\alpha_i \in Y$. This is a contradiction, since if v_i satisfies α_i, its extension v should also. Hence, we have a finite set Y which is not satisfiable. ■

Problems

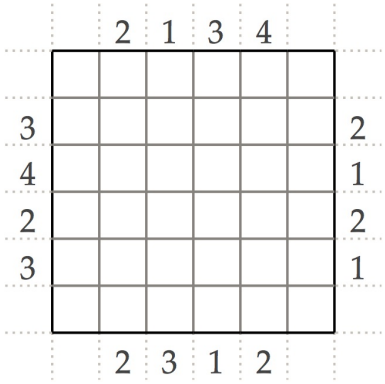
1.1 A finite set $\Gamma = \{\alpha_1, \alpha_2, \ldots, \alpha_n\}$ is satisfiable if and only if $\bigwedge_{i=1}^{n} \alpha_i$ is satisfiable.

1.8 Summary and Acknowledgements

The chapter is heavily influenced by the video lecture series of Dr. Shai Ben David [BD]. The lectures itself follow the book by Enderton [End72]. The chapter also follows the book by Huth and Ryan [HR04]. I would also like to thank Dr. Baskar Anguraj (BITS Pilani, Goa) for sharing his logic exercise questions.

1.9 Chapter exercises

1.1 [Indian Puzzle championship 2010]* Fill in the grid in such a way that every row and every column contains numbers from (1 – 5) exactly once. Some cells may remain blank. The numbers inside the grid represent the height of the building in the corresponding cell. The numbers outside the grid represent the number of buildings visible from that direction. Encode the problem in propositional logic.



1.2 Which of the following are correct?

1. $p \vee q \models p$
2. $\neg q, p \vee q \models p$
3. $(p \Longrightarrow q) \models (\neg q \Longrightarrow \neg p)$
4. $p \Longrightarrow q, s \Longrightarrow t \models (p \vee s) \Longrightarrow (q \wedge t)$
5. $(p \Longrightarrow q) \wedge (p \Longrightarrow r) \models p \Longrightarrow (q \wedge r)$
6. $p \wedge \neg p \models (r \Longrightarrow q)$

1.3 What can we say about the following?

1. T $\models \alpha$
2. T $\not\models \alpha$

1.4 [[Smu82]] Consider three persons A, B, C who need to sit in a row, but: (*a*) A does not want to sit next to C. (*b*) A does not want to sit in the left chair. (*c*) B does not want to sit to the right of C.

Write a propositional formula that is satisfiable if and only if there is a seat assignment for the three persons that satisfies all constraints. Is the formula satisfiable? If so, give an assignment. Clearly mention the meaning of each proposition.

1.5 Let α and β be arbitrary propositional formulas. Is the following correct?

$$\text{If } (\alpha \Longrightarrow \beta) \text{ then } (\alpha \Longrightarrow (\neg\beta))$$

If yes, argue why? Otherwise, give an example when this is wrong.

1.6 [Smullyan [Smu85]] In an island every inhabitant is either type T and makes only true statements, or type F and makes only false statements. Mr. Holmes hears gold is buried in the island. He goes there, meets an inhabitant and asks him, "Is there gold in this land?" The inhabitant replies, "If I am of type T, then there is gold here." Answer the following?

(a) What is the inhabitant's type?

(b) Is gold buried in this island?

1.7 [Logicians in the coffee bar] Three logicians walk in to a coffee bar, and is subsequently greeted by the waiter who asks, "Would all of you like to drink coffee?". The replies of the logicians are given below,
Logician 1 : "I don't know"
Logician 2 : "I don't know"
Logician 3 : "Yes. Bring coffee for all of us"
Provide an explanation for the responses of the logicians to the waiter's question.

1.8 Write an algorithm which outputs the number of satisfying assignments of a propositional formula. You can assume the input formula to be given in a form you want (either as a string, or a parse tree etc).

1.9 Assume $(\alpha \Longrightarrow \beta)$ is a tautology. Moreover α and β do not share a common atomic proposition. Show that either α is unsatisfiable or β is a tautology (or both). Show that the assumption about not sharing atomic propositions is necessary.

1.10 Let β be a formula over propositions $Q = \{q_1, \ldots, q_n\}$. β is neither a tautology nor a contradiction. Let α be an arbitrary formula over propositions $P = \{p_1, \ldots, p_n\}$ where $P \cap Q = \emptyset$. Consider another formula ψ, got by replacing every occurrence of p_1 in α by β. Use mathematical induction to prove.

$$\alpha \text{ is satisfiable if and only if } \psi \text{ is satisfiable}$$

1.11 Write the set of all subformulas of the following wff s.

1. $(((p1 \Rightarrow p2) \iff (p1 \Rightarrow p3)) \Rightarrow p3)$
2. $(((p1 \wedge p2) \Rightarrow p3) \iff ((p1 \Rightarrow p2) \vee (p1 \Rightarrow p3)))$

1.12 Write derivation trees and derivation sequences for the following wff s.

1. $(((p1 \Rightarrow p2) \iff (p1 \Rightarrow p3)) \Rightarrow p3)$
2. $(((p1 \wedge p2) \Rightarrow p3) \iff ((p1 \Rightarrow p2) \vee (p1 \Rightarrow p3)))$

1.13 Check whether the valuation v satisfies the wff s given below. $v(p_1)$ = T, $v(p_2)$ = F , and $v(p_3)$ = T.

1. $((p_1 \Rightarrow p_2) \Rightarrow (\neg p_1))$
2. $(((p1 \Rightarrow p2) \wedge (p1 \Rightarrow p3)) \iff (p1 \Rightarrow (p_2 \vee p_3)))$

1.14 Let α be a wff, c be the number of places at which binary connectives occur in α and s be the number of places at which atomic propositions occur in α. (For example, if α is $(p_1 \Rightarrow (p_2 \Rightarrow (\neg p_1)))$ then $c = 2$ and $s = 3$). Show by using mathematical induction $s = c + 1$.

1.15 Prove (or disprove)

1. If T $\models p$ and T $\models (p \Rightarrow q)$, then T $\models q$.
2. If $V \models p$ and $V \models (p \Rightarrow q)$, then $V \models q$.
3. If α and $(\alpha \Rightarrow \beta)$ are satisfiable, then β is satisfiable.

1.16 Given n construct a set of formulas Γ_n of size n such that Γ_n is not satisfiable, but every proper subsetof Γ_n is satisfiable.

1.17 Prove or Disprove the following statements. Given that Γ_1, Γ_2 are sets of well formed formulas.

1. $\Gamma_1 \subseteq \Gamma_2, Mod(\Gamma_2) \subseteq Mod(\Gamma_1)$
2. $\Gamma_1 \subseteq \Gamma_2, Mod(\Gamma_1) \subseteq Mod(\Gamma_2)$
3. If $\Gamma \models \alpha$, then $Mod(\Gamma) \subseteq Mod(\alpha)$

1.18 Show that a valuation v satisfies the following formula iff $v(p_i) = \mathrm{F}$ for an even number of i's, $1 \leq i \leq n$.

$$(\ldots(p_1 \iff p_2) \iff p_3) \iff \ldots) \iff p_n)$$

1.19 [Relevance Lemma] Let v_1 and v_2 are two valuations such that $v_1(p) = v_2(p)$, for all propositions $p \in Atoms(\alpha)$ for some formula α. Prove that $v_1 \models \alpha$ iff $v_2 \models \alpha$.

Bibliography

[BA12] Mordechai Ben-Ari. *Mathematical Logic for Computer Science, 3rd Edition.* Springer, 2012.

[BD] Shai Ben-David. Logic for CS, https://www.youtube.com/watch?v=kceafjjtjyu.

[CLRS01] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms.* MIT Press, Cambridge, MA, 2001.

[End72] H. B. Enderton. *A Mathematical Introduction to Logic.* Academic Press, 1972.

[HR04] Michael Huth and Mark Ryan. *Logic in Computer Science.* Cambridge, second edition, 2004.

[MS11] Madhavan Mukund and S. P. Suresh. *An Introduction to Logic.* https://www.cmi.ac.in/ madhavan/papers/pdf/logic-aug2011.pdf, 2011.

[MU05] Michael Mitzenmacher and Eli Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis.* Cambridge University Press, 2005.

[RN95] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach.* Prentice Hall, 1995.

[Sch89] Uwe Schöning. *Logic for Computer Scientists.* Birkhauser, Boston, 1989.

[Smu82] Raymond Smullyan. *The Lady or the Tiger? and other Logic Puzzles.* Oxford, 1982.

[Smu85] Raymond Smullyan. *To Mock a Mockingbird.* Knopf, 1985.

CHAPTER 2

THE COMPUTATIONAL COMPLEXITY OF SATISFIABILITY

2.1 P vs NP

Definition 2.1 (Decision problem) *A problem is a function, $f : U \longrightarrow V$ which takes an input an element from a set U and outputs an element from a set V. A decision problem is a function where $V = \{\text{YES}, \text{NO}\}$. In other words the output of a decision problem is* YES *or* NO.

Some examples of decision problems are

1. SAT: Check if a given wff is satisfiable or not.
2. Primality: Check whether a positive integer is prime or not.
3. Searching: Check if an integer is in a list of integers or not.

The following are not decision problems although they can be converted into decision problems.

1. Sorting
2. Linear programming

SAT

Input: A wff α.

Output: YES if α is satisfiable, otherwise NO.

In a decision problem, an input is called an Yes instance if the problem outputs Yes on this input. An instance is a No instance if it is not an Yes instance (that is, the problem outputs No in this instance). Any problem can be modified into a decision problem as follows. Ask whether the i^{th} output bit is a 1 or a 0? For example, here is the sorting problem converted into a decision problem.

Decision version of Sorting

Input: A list of numbers, and an index i.

Output: YES if the i^{th} bit in the sorted list of number is 1. Otherwise output NO.

Thus we can talk about any problem as a decision problem. Note that an algorithm to answer the decision version can be used to answer the non-decision version.

Definition 2.2 (P - polynomial time problem) *The number of steps in a* polynomial time algorithm *is polynomial in the size of the input. Polynomial time problems (or commonly called as P) is the set of all problems which can be solved by a polynomial time algorithm.*

For example, Primality, Searching, Sorting etc are in P. Let us consider a problem in propositional logic. In the DNF SAT problem, one needs to check if a DNF formula is satisfiable or not. This is a special case of the SAT problem.

DNF SAT

Input: A DNF formula α.

Output: YES if α is satisfiable, otherwise NO.

This problem can be solved in polynomial time. Let α be a DNF formula. The polynomial time algorithm takes each conjunctive clause of α and checks if it can be satisfiable or not. A conjunctive clause cannot be satisfied if and only if the clause contains both a proposition and its negation (see Exercise **??**). If atleast one of the conjunctive clause is satisfiable then α is satisfiable. If no conjunctive clause is satisfiable, then α is not satisfiable.

What about checking if a wff α is satisfiable or not. This problem is famously called SAT. For a formula α with n propositions the truth table has 2^n number of rows. Therefore building the truth table and checking each row is a trivial way to check for satisfiability. This algorithm though takes exponential time, since the number of rows in the truth table is 2^n. The interesting question is, does there exist a faster algorithm. Or even better, does there exist a polynomial time algorithm.

That is, an algorithm whose number of steps is $O(n^c)$ for some constant c. It turns out, we do not yet know the answer to this question. Most computer scientists think there is no polynomial time algorithm for SAT. This is the biggest open problem in computer science called

$$\text{P} = \text{NP?}$$

Let us now understand the class of problems called NP. Given a wff α and an assignment v to the proposition, we can check in polynomial time if v satisfies α. It is easy to store the assignment v. It takes only $O(n)$ size if n is the number of propositions. We can think of v as a "proof" that α is satisfiable, since if you want to argue to someone that α is satisfiable you only need to give the assignment v. This proof is usually called the indexitcertificate. Thus, if α is satisfiable, there is a polynomial sized certificate of its satisfiability (namely the assignment v). In other words, the Yes instance has a short certificate. The second point to consider is that, there is a polynomial time machine which given the proof and the formula checks if the assignment satisfies it or not. The class NP consists of all problems which show similar behaviour. Surprisingly there are many problems which has a short proof for the Yes instance
We will now define the class of non-deterministic polynomial time problems (NP).

Definition 2.3 (NP - non deterministic polynomial time problems) *A problem is in NP, if there exists a polynomial time algorithm $\mathcal{A}$ and a polynomial $p : \mathbb{N} \longrightarrow \mathbb{N}$ such that*

1. *For all Yes instance x, there exists a certificate y where $|y| \leq p(|x|)$ (that is, size of y is polynomial in the size of x) such that $\mathcal{A}(x, y)$ outputs Yes.*
2. *For all No instance x, $\mathcal{A}(x, y)$ = No for all certificates y where $|y| \leq p(|x|)$.*

It follows from our discussion that

Theorem 2.1 *SAT is in NP.*

Let us look at a few more examples. The problem of Hamiltonian cycle is in NP, since the certificate is the Hamiltonian cycle. Similarly k-clique problem, 3-colorability, subset sum problem are all in NP. Here is a claim which holds because all polynomial time problems has an empty certificate.

Claim 2.2 *If a problem is in P, then it is in NP. That is,*

$$P \subseteq NP$$

Consider the composite problem. It is clearly in NP, since you can have a polynomial size certificate: a factor of x which is greater than 1 and less than x.

COMPOSITE

Input: A positive integer x

Output: YES if x is a composite number. No otherwise.

What about the complement problem? That is the problem of whether a number is prime or not. Clearly the no instance has a short certificate (namely a factor which divides the number). Let us formally define the complement problem.

Definition 2.4 *For a decision problem X, the complement decision problem (denoted by $\overline{X}$) is the problem where*

$$x \text{ is an yes instance of } X \text{ if and only if } x \text{ is a No instance of } \overline{X}$$

The complement problem for Composite is Primes. This leads to another complexity class co-NP.

Definition 2.5 (co-NP) *The class co-NP consists of all problems whose complement is in NP.*

Examples of co-NP problems are precisely the complements of NP problems. Here is an interesting problem. We leave the reader to identify why this problem is a complement of an NP problem.

VALIDITY

Input:	A well formed formula α
Output:	YES if α is a valid formula. No otherwise.
Complexity class:	co-NP.
Certificate for No instance:	An assignment which does not satisfy α.

Clearly co-NP problems has polynomial sized certificates for the No instances. Here is an alternate definition of co-NP.

Theorem 2.3 *If a problem is in co-NP, then there exists a polynomial time algorithm $\mathcal{A}$ and a polynomial $p : \mathbb{N} \longrightarrow \mathbb{N}$ such that*

1. *For all No instance x, there exists a certificate y where $|y| \leq p(|x|)$ (that is, size of y is polynomial in the size of x) such that $\mathcal{A}(x, y)$ outputs No.*
2. *For all Yes instance x, $\mathcal{A}(x, y)$ = Yes for all certificates y where $|y| \leq p(|x|)$.*

So, we know that Primes is in co-NP. Surprisingly, Primes is also in NP due to some properties of numbers. Few years back, AKS showed that Primes is in P.

PRIMES

Input:	A positive integer x
Output:	YES if x is a prime number. No otherwise.
Complexity class:	in P (by the AKS theorem).

The factorization problem is in both NP and co-NP. We do not yet know if the problem is in P. In fact we do not believe it to be in P. The factorization problem is

a foundational to cryptography. A polynomial time algorithm to do factorization can lead to breaking all cryptographic protocols.

FACTORIZATION	
Input:	Positive integer x and y
Output:	YES if there is a number z where $1 < z \leq y$ such that z divides x. Otherwise output No.
Certificate for Yes instance:	A factor z.
Certificate for No instance:	Prime factorization of x.
Complexity:	NP ∩ co-NP.

Yet another problem which is in NP ∩ co-NP is the Parity games problem. The problem is believed to be in P but we do not have a proof yet. Before we end this subsection, we leave the reader to prove the following claims.

Claim 2.4

$$P \subseteq \text{co-}NP$$

Claim 2.5 *If P=NP, then NP=co-NP. Contra-positively, if NP ≠ co-NP, then P ≠ NP.*

The biggest open problem in computer science is the following conjecture.

Conjecture 2.6

$$P \neq NP$$

2.2 SAT: Hardest among NP

In the last section, we saw that satisfiability of DNF can be done in polynomial time. What about CNF? The problem of satisfiability of CNF formulas is called CNF SAT. Can we solve this also in polynomial time?

Consider the following claim (Algorithm 2) which takes as input a wff α and outputs a CNF formula β such that α is satisfiable if and only if β is satisfiable. Note that the algorithm runs in polynomial time. What is the importance of this? It shows that, if we have a polynomial time algorithm to solve CNF SAT, then there is a polynomial time algorithm to solve SAT.

Theorem 2.7 *There is a polynomial time algorithm $\mathcal{A}$ which takes as input wffs and outputs CNF formulas such that the output formula is satisfiable if and only if the input formula is satisfiable.*

Proof: We will give a reduction from SAT to CNF SAT. Let α be a propositional formula. Our aim is to give a CNF formula $\hat{\alpha}$ such that α is satisfiable if and only if $\hat{\alpha}$ is satisfiable. We first replace subformulas of the from $(\beta \implies \gamma)$ in α by $(\neg\beta \vee \gamma)$. The second step is to push the negations to the propositions using De-Morgan's law. That is subformulas of type $\neg(\gamma \vee \beta)$ is replaced by $\neg\gamma \wedge \neg\beta$. Similarly, $\neg(\gamma \wedge \beta)$

is replaced by $\neg\gamma \vee \neg\beta$. This is done inductively until all negations apply to propositions. We now need to convert the formula into a conjunction of disjunctions. To convert subformulas of the form $(\beta \wedge \gamma) \vee \psi$ we introduce a new proposition p (which is not present in the formula). We then replace $(\beta \wedge \gamma) \vee \psi$ by $(\psi \vee p) \wedge (\beta \vee \neg p) \wedge (\gamma \vee \neg p)$. It is easy to see that the two formulas are equivalent with respect to satisfiability. Inductively applying this translation will give us a CNF formula. ∎

Algorithm 2 Convert wffs to CNF formula

Input A wff α
Output: A CNF formula β such that α is satisfiable if and only if β is satisfiable.

function WFFTOCNF(α)
  repeat
    $\alpha' = \alpha$.
    Apply DeMorgan's on α to push negations inside brackets.
    Apply double negation law to simply formula.
    Let α be the modified formula
  until $\alpha' = \alpha$ ▷ On exit, negations appear only before propositions in α.
  if α is a literal **then return** α
  if $\alpha ::= (\alpha_1 \wedge \alpha_2)$ **then return** (WFFTOCNF(α_1) $\wedge$ WFFTOCNF(α_2))
  if $\alpha ::= (\alpha_1 \vee \alpha_2)$ **then**
    Let WFFTOCNF(α_1) = $\bigwedge_i \beta_i$
    Let WFFTOCNF(α_2) = $\bigwedge_j \gamma_j$
    Introduce a new proposition p
    return $\left(\bigwedge_i (\beta_i \vee p) \wedge \bigwedge_j (\gamma_j \vee \neg p) \right)$
  end if
end function

What we have seen here is an example of *polynomial time reductions* . Let us make this definition formal (also see Figure 2.1).

Definition 2.6 (polynomial time reduction) *A polynomial time reduction from a problem X to a problem Y is an algorithm $\mathcal{A}$ which takes as input, instances of problem X and outputs instances of problem Y such that for all input instance x of problem X, x is an Yes instance if and only if $\mathcal{A}(x)$ is an Yes instance of problem Y.*

Here is another polynomial time reduction from CNF SAT to 3-CNF SAT.

Claim 2.8 *There is a polynomial time reduction from CNF SAT to* 3*-CNF SAT.*

Proof: We first show how to rewrite a disjunctive clause of the form $\alpha ::= (p_1 \vee p_2 \vee \cdots \vee p_k)$. Introduce new propositions $t_1, t_2, \ldots, t_{k-2}$. The translated formula will be

$$\beta ::= (p_1 \vee p_2 \vee t_1) \wedge (\neg t_1 \vee p_3 \vee t_2) \wedge \cdots \wedge (\neg t_{k-2} \vee p_{k-1} \vee p_k)$$

We leave it to the reader to show that α is satisfiable iff β is satisfiable. It is now easy to see how to translate conjunctions of disjunctions of formulas. ∎

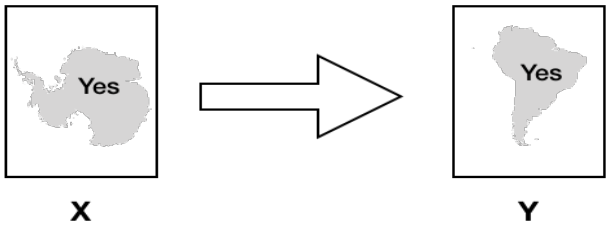


Figure 2.1 Polytime reduction from problem X to Y: The reduction takes an Yes (respectively No) instance in X to an Yes (resp. No) instance in Y.

We denote by $X \leq_p Y$ if there is a polynomial time reduction from X to Y. Intuitively we imply that the problem Y is "harder" than problem X. This is because if we can solve Y, then we can solve X. Let us write what we have learned in this notation.

$$\text{SAT} \leq_p \text{CNF SAT} \leq_p \text{3-CNF SAT}$$

Also note that 3-CNF SAT is a special case of CNF SAT which is a special case of SAT. Hence this also holds

$$\text{3-CNF SAT} \leq_p \text{CNF SAT} \leq_p \text{SAT}$$

We observed that reductions help us order problems on the basis of its hardness. What are the problems harder than all problems in NP?

Definition 2.7 (NP-hard) *A problem X is NP-hard, if there are polynomial time reductions from all NP problems to X.*

In the above definition, X is harder than all NP problems and therefore solving X can solve all NP problems. Are there problems which are NP-hard? In a beautiful result, Cook showed that there are NP-hard problems. Infact it is our favourite SAT problem.

Theorem 2.9 (Cook's) *SAT is NP-hard.*

We leave the proof of this claim. But once we know SAT is NP-hard, we have got other problems which are NP-hard. Clearly CNF-SAT is NP-hard.

Claim 2.10 *CNF-SAT is NP-hard.*

Proof: To show that CNF-SAT is NP-hard, we need to show that there are polynomial time reductions to it from all NP problems. Cook has saved us by showing that there are polynomial time reductions from all problems to SAT. So, we only need to show that there is a polynomial time reduction from SAT to CNF-SAT. We already showed this in Theorem 2.7. ∎

By a similar argument we also have that 3-CNF SAT is NP-hard. The next problem is, what are the hardest problems in NP? These are called NP-complete problems.

Definition 2.8 (NP-complete) *A problem X is called NP-complete if X satisfies both the below conditions.*

1. *X is in NP.*
2. *X is NP-hard.*

The following theorem is a consequence of our discussion till now.

Theorem 2.11 *SAT, CNF-SAT, 3-CNF SAT are NP-complete problems.*

Proof: We give the proof for SAT. A problem is NP-complete if it is both in NP and is NP-hard. It follows from Theorem 2.1 that SAT is in NP and from Theorem 2.9 that SAT is NP-hard. Therefore SAT is NP-complete. We leave it to the readers to argue why the other problems are NP-complete. ∎

Let us show a problem in graph theory which is NP-complete. The clique problem takes as input a graph G and a number k and outputs YES if and only if the graph has a k-clique (a set of k vertices such that there is an edge between all pairs of these vertices).

Theorem 2.12 *The clique problem is NP-complete.*

Proof: It is easy to see that the problem is in NP.

We show that the problem is NP-hard by reducing from 3CNFSAT. Let $\Phi = (C_1 \wedge C_2 \wedge \cdots \wedge C_k)$ be a 3CNF formula. Let us denote each of the clause $C_i ::= (l_i^1 \vee l_i^2 \vee l_i^3)$. We now give a graph $G = (V, E)$ as follows. The vertices of the graph V are v_i^1, v_i^2, v_i^3 for all $i \leq k$. We now have an edge between $E(v_i^t, v_j^r)$ if

1. $i \neq j$ and
2. $v_i^t \neq \neg v_j^r$

The input to the clique problem is this graph $G = (V, E)$ and number k. It is easy to see that this translation takes polynomial time. We now show that the translation is correct. In other words: Φ is satisfiable if and only if G has a k-clique. ∎

We have argued that there are polynomial reductions from all NP problems to SAT. The next claim asks you to reduce to SAT from the Hamiltonian problem.

Claim 2.13 *Give a polynomial time algorithm which takes as input a graph $G = (V, E)$ and outputs a wff α such that the graph has a Hamiltonian cycle if and only if α is satisfiable. Does the valuation which makes α true give us the cycle? [A Hamiltonian cycle is a cycle which traverses every vertex exactly once.]*

Since SAT is the hardest problem in NP, to solve the Conjecture 2.6 of P ≠ NP is equivalent to show that there is no polynomial time algorithm for SAT. That is,

Claim 2.14 *P ≠ NP if and only if SAT is not in P.*

Proof: Clearly if SAT is not in P, then P ≠ NP. So, let us look at the other direction. Let SAT be in P. Consider an arbitrary problem X in NP. Then, there is a polynomial

time reduction from X to SAT. Since SAT is in P, we can solve X also in polynomial time. Therefore X is in P. Hence all problems in NP are in P. ■

.

This gives rise to two possibilities for NP problems as shown in Figure 2.2.

Figure 2.2 Figure 1. assumes P ≠ NP. Figure 2. assumes P = NP.

Finally we will look at the hardest co-NP hard problems.

Definition 2.9 (co-NP hard and co-NP complete) *A problem X is co-NP hard, if there are polynomial time reductions from all co-NP problems to X. A problem X is co-NP complete if X satisfies both the below conditions.*

1. *X is in co-NP.*
2. *X is co-NP hard.*

The following claim follows easily whose proof we leave it to the reader.

Claim 2.15 *A problem X is co-NP complete if and only if its complement problem $\overline{X}$ is NP complete. Therefore, Validity problem is co-NP complete.*

Here is another claim.

Claim 2.16 *If an NP problem is co-NP hard, then NP=co-NP.*

Proof: Let us assume X is an NP problem which is co-NP hard. We first show that this implies co-NP $\subseteq$ NP. Let Y be an arbitrary problem in co-NP. Since X is co-NP hard, there is a polynomial time reduction to X from Y. This implies Y is in NP.

Let us now show the other direction NP $\subseteq$ co-NP. Consider an arbitrary $Y \in$ NP. Then $\overline{Y}$ is in co-NP. From our above claim, $\overline{Y} \in$ NP. Therefore Y is in co-NP. ■

The figure 2.3 captures a possible relationship between P, NP and co-NP.

2.3 Summary and Acknowledgements

This chapter uses the same presentation as found in Introduction to Algorithms [CLRS01].

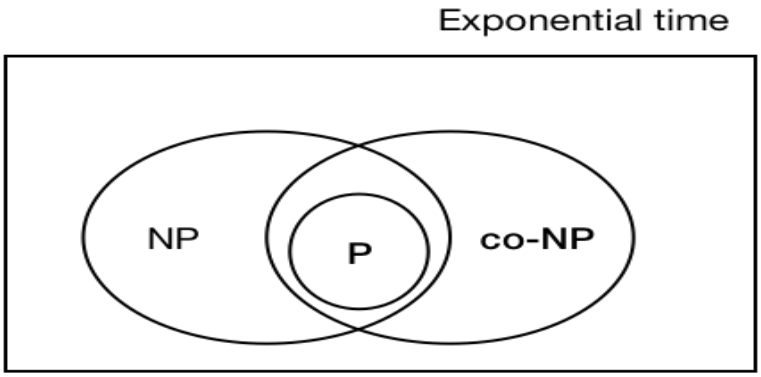


Figure 2.3 One possible relationship between P, NP and co-NP

2.4 Chapter exercises

2.1 An alternate definition of NP is the set of all problems which can be solved by a non-deterministic Turing machine running in polynomial time. Using this definition can you prove Cooks' theorem that SAT is NP-hard. Given a non-deterministic Turing Machine which runs in $poly(n)$ time (say in time n^2) and an input of size n, construct a wff (of size polynomial in n) such that the formula is satisfiable if and only if the Turing machine accepts the input.

2.2 Show that if we can reduce a problem X to an NP-complete problem, then X is in NP. Mention the short certificate in this case as well as the polynomial time algorithm.

2.3 For an undirected graph G, we say that a set of vertices $I = \{v_1, v_2, \ldots, v_k\}$ is an independent set, if there are no edges between the vertices in I. In other words, for all $v_i, v_j \in I$, the edge (v_i, v_j) does not exist. The Independent set problem is as follows.
Input: A graph G and a number k
Output: Yes, if there exists an independent set of cardinality k. No, otherwise.
Show the following

1. (2 marks) The independent set problem is in NP.

2. (4 marks) The independent set problem is NP-hard by reducing from 3-CNF SAT.

   [Hint: For a graph $G = (V, E)$, consider its complement graph $\overline{G} = (V, \overline{E})$. The vertices in G and $\overline{G}$ are the same. The edges are different though. For all vertices $v_i, v_j \in V$, we have that $(v_i, v_j) \in E$ if and only if $(v_i, v_j) \notin \overline{E}$. In other words, an edge is in one graph if and only if the corresponding edge is not present in the other graph.

The following claim is the hint (which you do not have to prove): A graph G has an independent set of size k if and only if the complement graph $\overline{G}$ has a clique of size k.]

3. (4 marks) Prove that your above reduction is correct.

2.4 Show that 3-colorability is NP-complete.

2.5 Show that k-clique problem is NP-complete.

2.6 The subset-sum problem is defined as follows: Given a set S of n positive integers and a positive integer W, determine whether there is a subset of S whose elements sum to W. There is an algorithm which solves this problem in $O(nW)$ time. Answer the following questions.

1. Does an $O(nW)$ time algorithm imply subset-sum problem is in P?
2. Show that the problem is in NP.
3. Now show that the problem is NP-hard by reducing from CNF SAT.

2.7 Write True or False for the following statements.

1. This is possible. NP \P = NP-complete (Here \denotes set substraction).
2. DNF satisfiability can be done in polynomial time.
3. Is $(p \implies q) \equiv (\neg((\neg p) \wedge q))$ (here $\equiv$ stands for equivalent).
4. This is possible. NP $\cap$ co-NP = P.
5. If we want to prove that a problem X is NP-Hard, we take a known NP-Hard problem Y and reduce Y to X.
6. This is possible. NP-complete is a subset of NP-hard.
7. This is possible. NP = co-NP but NP $\neq$ P.
8. Let X be an NP-complete problem which can be solved in worst case running time of $O(2^{\sqrt{n}})$. Then all NP problems can be solved in $O(2^{\sqrt{n}})$ time.

CHAPTER 3

SAT SOLVERS

3.1 Introduction

Consider the SAT problem (Problem 2.1). From the discussion in the previous chapter we know SAT is NP-complete. In this chapter we will look at algorithms to solve the SAT problem. Naturally, they all run in exponential time in the worst case. But, they work well in practise. A huge engineering effort has gone into making SAT solvers work in practise. In this chapter we will consider deterministic SAT solvers. This is in contrast with Randomized SAT solvers. In the later case, the algorithms use random numbers to reduce the computational speed. The trade off is in accuracy. In some situations the algorithm might say the wff is not satisfiable when in fact it can.

The deterministic algorithms we see in this chapter are: Resolution, Semantic/Analytical tableaux and DPLL.

3.2 Resolution

3.2.1 The algorithm

The *Resolution* algorithm was proposed by J. Robinson in 1965. The advantage of this method, will be clear when we do the validity problem for First order logic. Resolution is a syntactic algorithm for solving SAT. That is, the algorithm will only do symbolic manipulations. Later we will see DPLL, a semantic algorithm. In that algorithm, we will assign true or false values to propositions.

Resolution can be applied only on CNF formulas. From the previous chapter, we know that any wff can be converted to an equisatisfiable CNF formula in polynomial time. Therefore, in principle, Resolution can be used to solve satisfiability of any wff. A CNF formula α is a conjunction of disjunctions. That is $\alpha ::= \bigwedge_i \beta_i$ where β_i are disjunctions. We will call $\beta_i$as clauses (in the earlier chapter we called this as disjunctive clause). We say that a literal l appears in a clause β_i if $\beta_i ::= (\gamma_1 \vee l \vee \gamma_2)$ where γ_1 and γ_2 are clauses. Similarly, we say that a clause β appears in a wff α if $\alpha = (\beta \wedge \gamma)$ where γ is a CNF formula. The resolution algorithm works by translating one CNF formula into another. While this processing is done, we require the following simplification to be done on the formula: If β is a clause in α, then we will assume the formula $(\alpha \wedge \beta)$ to be also α. In other words, there cannot be two clauses β in any formula. Clearly this does not affect satisfiability. Similarly, we also assume that if l is a literal in a clause β, then $(\beta \vee l)$ will also be considered as β only.

For a literal l, we write

$$\bar{l} = \begin{cases} p, & \text{if } l = (\neg p) \\ (\neg p), & \text{if } l = p \end{cases}$$

A *unit clause* is a clause which has only one literal. For example the following formulas has a unit clause (namely $\neg p$) but clauses $(q \vee r)$ and $(t \vee \neg r)$ are not unit clauses.

$$(q \vee r) \wedge (\neg p) \wedge (t \vee \neg r)$$

The most important step of the resolution algorithm is the resolution step.

Resolution step: This is the most important step in the algorithm. It says that, if a wff α has clauses $(\gamma \vee l)$ and $(\beta \vee \bar{l})$, then the following clause $(\gamma \vee \beta)$ can be conjuncted with α without changing its satisfiability. This rule can increase the size of the wff. In the next section, we will see a family of wffs for which the Resolution algorithm blows up to exponential size. Note that, if the blow up is only polynomial, we will be solving SAT in polynomial time (and hence P = NP).

Consider two clauses $(\gamma \vee l)$ and $(\beta \vee \bar{l})$ where l is a literal and γ and β are clauses. We now define the operator $Resolve$ on these clauses as follows.

$$Resolve_l(\gamma \vee l, \beta \vee \bar{l}) = \gamma \vee \beta$$

$$\frac{(\gamma \vee l) \qquad (\neg l \vee \beta)}{(\gamma \wedge \beta)} \text{ Resolution on } l \qquad \frac{l \qquad (\neg l \vee \beta)}{\beta} \text{ Resolution on } l$$

$$\frac{l \qquad \neg l}{\mathrm{F}} \text{ Resolution on } l$$

Figure 3.1 Resolution step: The three ways resolution can occur.

This operation can also be applied when one of the clauses is a unit clause. That is

$$Resolve_l(\gamma \vee l, \bar{l}) = \gamma$$

Finally, we discuss the operation on two unit clauses. Consider the unit clauses p and $(\neg p)$. Since the formula which contains both these unit clauses is not satisfiable, we will define the resolution rule applied on these clauses to be F. That is,

$$Resolve_l(l, \bar{l}) ::= \mathrm{F}$$

See Figure 3.1 for the resolution step in pictorial form.

Note that a formula containing both l and $\bar{l}$ as clauses is not satisfiable. The resolution rule can states that the *Resolve* operator can be applied to any CNF formula without changing the satisfiability. The proof of the claim is left to the reader.

Claim 3.1 (Resolution Rule) *Let α be a CNF formula and l a literal. Let γ and β be two clauses in α such that l is in γ and $\bar{l}$ is in β. Then,*

$$\alpha \text{ is satisfiable iff } \big(\alpha \wedge Resolve(\gamma, \beta)\big) \text{ is satisfiable.}$$

Let us look at an example of applying the resolution rule.

EXAMPLE 3.1

Let $\alpha ::= \big((p \vee q \vee \neg r) \wedge (p \vee t \vee r)\big)$. We see that r is a literal which appears as $\neg r$ in first clause and r in second clause. Hence we can apply the resolution rule in these clauses to get the formula

$$\big((p \vee q \vee \neg r) \wedge (p \vee t \vee r) \wedge (p \vee q \vee t)\big)$$

The algorithm which implements resolution rule can be stated as follows.

The resolution algorithm (Algorithm 4) can now be stated as follows. Keep on applying the resolution rule until you cannot go further. During this process, if

Algorithm 3 Resolution rule on a proposition

Input A CNF formula α
Output: CNF formula with one less proposition than α

```
1: function ResolutionRule(α)
2:     Select a proposition p in α.
3:     repeat
4:         α' = α.
5:         Select two clauses γ and β containing p and ¬p respectively.
6:         α = (α ∧ Resolve_p(γ, β))
7:     until α' = α
8:     return α.
9: end function
```

you ever see a F in the formula, then it means the input formula is not satisfiable. If we never see F, then we can say that the formula is satisfiable. We say that the resolution algorithm is *sound* and *complete* . Soundness means that, whenever the resolution algorithm returns NO, the formula is not satisfiable. Complete means that, whenever the formula is not satisfiable, the algorithm returns NO.

Theorem 3.2 (sound and complete) *The resolution algorithm (Algorithm 4) is correct. That is, the algorithm returns NO if and only if the input formula is not satisfiable. In other words, for all formula α*

$$F \text{ is a clause in } \mathcal{T}(\text{CNF formulas}, \alpha, \{\text{ResolutionRule}\}) \text{ iff } \alpha \text{ is not satisfiable}$$

Proof : **(soundness:)** Let us now assume the algorithm returns NO after n iterations of the while loop. Let us denote by α_i, the α at the beginning of the i^{th} iteration of the loop. Therefore $\alpha_1 = \alpha$ and F is in α_n. The α's at each iteration is denoted as

$$\alpha = \alpha_1, \alpha_2, \ldots, \alpha_n$$

From claim 3.1, we have that for each $i < n$, α_{i-1} is satisfiable if and only if α_i is satisfiable. We first claim that α_n is not satisfiable. This follows from the fact that on reaching α_n, the algorithm outputs NO. This can happen only if F is a clause in α_n. Since α_n is not satisfiable, by an inductive argument using Claim 3.1 it follows that $\alpha = \alpha_1$ is not satisfiable.

(completeness:) ∎

3.2.2 Extended Resolution algorithm

The algorithm consists of two parts. As stated earlier, we modify the wff as we go along the algorithm.

Preprocessing step: In this step, we give some simple transformations of an input wff. The preprocessing step reduces the size of the formula. It also runs in polynomial time in the size of the input.

Algorithm 4 Resolution

Input A CNF formula α
Output: YES if α is satisfiable, NO otherwise.

```
1: function RESOLUTION(α)
2:     repeat
3:         if (F is in α) then return NO
4:         α′ = α.
5:         α = RESOLUTIONRULE(α).
6:     until α′ = α
7:     return YES.
8: end function
```

The following claim says that if α contains a unit clause l (but not $\neg l$) then we can get an equisatisfiable formula α' of smaller size. During these steps, we might reach a situation where all the clauses in α will be removed by the algorithm. If this happens, we show that α is satisfiable.

Claim 3.3 (unit clause elimination) *Let α be a wff such that it does not satisfy unit clause contradiction but it contains a unit clause l. Let α' got by removing from α*

1. *all clauses which contains l and*
2. *all literals ($\bar{l}$) from all clauses.*

Then, α is satisfiable if and only if (α' is satisfiable or α' is empty).

The reader can try to prove the above claim. We say that l is a pure literal in wff α if $\bar{l}$ does not appear in α. In the following formula, $q, \neg p$ and t are pure literals whereas r or $\neg r$ are not.

$$(q \vee r) \wedge (\neg p) \wedge (t \vee \neg r)$$

The next claim says that pure literals can be removed without affecting satisfiability.

Claim 3.4 (pure literal elimination) *Let l be a pure literal in a wff α. Let α' be the wff got by removing all clauses which contain l from α.*
Then, α is satisfiable if and only if (α' is satisfiable or α' is empty).

We leave the proof of the above claim as well as the one below to the reader.

Claim 3.5 (tautology elimination) *Let α contain a clause which has both literals l and $\bar{l}$. Let α' be the formula got by removing the clause from α.*
Then, α is satisfiable if and only if (α' is satisfiable or α' is empty).

We now state the extended Resolution algorithm.

We are now in a position to show that the Algorithm 5 is correct.

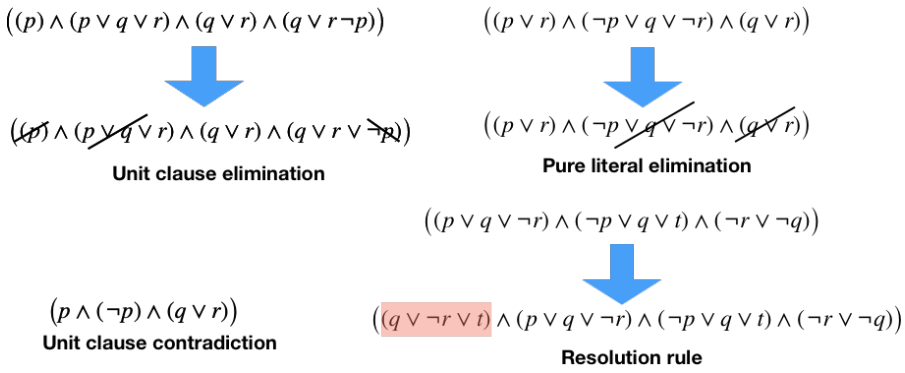


Figure 3.2 The rules of resolution algorithm

Algorithm 5 Resolution extended

Input A CNF formula α
Output: YES if α is satisfiable, NO otherwise.

```
1:  function ResolutionX(α)
2:      repeat
3:          if (F is in α) then return NO
4:          α = PureLiteralElimination(α)
5:          α = UnitClauseElimination(α)
6:          α = ResolutionRule(α)
7:          α = TautologyElimination(α)
8:          if (α is empty) then return YES
9:      until true
10:     return YES.
11: end function
```

Lemma 3.6 (Soundness) *Let α be an arbitrary wff. If Resolution(α) = No, then α is not satisfiable.*

Proof : Let us now assume the algorithm returns No after n iterations of the while loop. Let us denote by α_i, the α at the beginning of the i^{th} iteration of the loop. Therefore $\alpha_1 = \alpha$ and α_n = No. The α's at each iteration is denoted as

$$\alpha = \alpha_1, \alpha_2, \ldots, \alpha_n = \text{No}$$

From the induction statement we have that for each $i < n$, α_{i-1} is satisfiable if and only if α_i is satisfiable. We first claim that α_{n-1} is not satisfiable. This follows from claim **??** and the fact that, UnitClauseContradiction(α_{n-1}) = No. Since α_{n-1} is not satisfiable, by an inductive argument using Claim 3.1 it follows that $\alpha = \alpha_1$ is not satisfiable. ■

The next lemmas shows the other side.

Lemma 3.7 (Completeness) *Let α be an arbitrary wff. If Resolution(α) = Yes, then α is satisfiable.*

Proof: We follow the same proof as the one in Lemma 3.6.

Let us now assume the algorithm returns Yes after n iterations of the while loop. Let us denote by α_i, the α at the beginning of the i^{th} iteration of the loop. Therefore $\alpha_1 = \alpha$ and α_n = Yes. The α's at each iteration is denoted as

$$\alpha = \alpha_1, \alpha_2, \ldots, \alpha_n = \text{Yes}$$

From the induction statement we have that for each $i < n$, α_{i-1} is satisfiable if and only if α_i is satisfiable. We first claim that α_{n-1} is satisfiable. This follows from claims 3.4, 3.5 and 3.1. Since α_{n-1} is satisfiable, by an inductive argument using Claim **??** it follows that $\alpha = \alpha_1$ is satisfiable. ∎

Lemma 3.8 (Termination) *The Resolution algorithm terminates.*

Proof: We prove by induction on the number of propositions: For all wffs α, we have $\alpha \neq$ oneStepProcess(α).

Consider the base case of one proposition p_1: Then α can either be p_1, or $(p_1 \vee \neg p_1)$ or $(p_1 \wedge \neg p_1)$ or $\neg p_1$. The claim holds in all these cases.

Now let us prove the inductive step. Let us assume that the claim is true for all wffs which has atleast $n-1$ number of propositions. Consider a wff α with propositions $p_1, p_2, \ldots, p_n$. We prove this inductive step by contradiction. Let α = oneStepProcess(α). Observe that α does not satisfy unit clause contradiction. It also does not contain pure literals, unit clauses or a tautology clause. Therefore for every proposition p_i, there is a clause which contains p_i and a clause which contains $\neg p_i$ (and both cannot come in the same clause). We can therefore write α as follows

$$\left(\begin{array}{c}(\beta_1 \vee p_1) \wedge \\ (\beta_2 \vee p_1) \wedge \\ \vdots \\ (\beta_s \vee p_1)\end{array}\right) \wedge \sigma \wedge \left(\begin{array}{c}(\gamma_1 \vee \neg p_1) \wedge \\ (\gamma_2 \vee \neg p_1) \wedge \\ \vdots \\ (\gamma_t \vee \neg p_1)\end{array}\right)$$

Finally, we also have that α is closed under Resolution rule. Note that σ is a wff which has only $n-1$ propositions. Therefore by induction hypothesis, $\sigma \neq$ oneStepProcess(σ). In other words, σ satisfies atleast one of the following conditions

1. Unit clause contradiction: Then α also should satisfy this condition.

2. Contain a pure literal: If σ has a pure literal, then α also has a pure literal.

3. Contains a unit clause or tautology: Clearly this would imply α also has a unit clause or a tautology.

4. σ is not closed under Resolution rule: Again, this would imply α also doesnt satisfying the Resolution rule.

All the above cases lead to a contradiction. Hence our initial assumption is false and therefore $\alpha \neq \text{oneStepProcess}(\alpha)$.

The second part of our proof is to show by induction on the number of propositions that: For all wffs α, the algorithm terminates. The base case is easy to see. So consider an α which has propositions $p_1, \ldots, p_n$. We know by induction hypothesis that the algorithm terminates for all wffs β which contains $n-1$ propositions. From our claim above, we know that every call to the oneStepProcess subroutine changes the formula. Let

$$\alpha = \alpha_1, \alpha_2, \ldots$$

be an infinite run of the algorithm where for all $i \geq 1$, $\alpha_i \neq \alpha_{i+1}$. We now give a contradiction. There are two cases to consider.

1. Let us assume that there exists an i such that for all $j \geq i$, we have that α_j contains less than or equal to $n-1$ propositions (this could have happened because of the some literal elimination). By induction hypothesis, this means that the algorithm terminates. Therefore, there cannot be an infinite run.

2. Assume that for all j, we have α_j contains n propositions. That means, no literal was eliminated during the entire run of the algorithm. Hence only tautology elimination and the resolution rule was applied throughout. Now note that there are only 2^n clauses possible (after elimination tautology wffs). Since, the resolution step only increases the clauses and not decrease, the algorithm cannot run for more than 2^n steps. Again, we have that, there cannot be an infinite run.

Both the above two cases show that the algorithm runs only for a finite number of steps. Hence Algorithm 4 terminates. ∎

We can now use the above arguments to show that the our algorithm is correct.

Theorem 3.9 *The Resolution algorithm (Algorithm 4) is correct.*

Proof: Lemma 3.8 shows that the algorithm terminates for all inputs. It now follows from Lemma 3.6 and Lemma 3.7 that the algorithm returns Yes if and only if the input formula is satisfiable. ∎

3.2.3 Worst case running time

Todo: We show that pigeonhole principle will take worst case exponential running time.

3.2.4 Polynomial time: 2CNF SAT

See Exercise **??** for a polynomial time algorithm for 2CNF SAT.

3.2.5 Polynomial time: HornClause SAT

Horn clause formulas are special formulas for which we can check satisfiability in polynomial time. They are used in knowledge representation.

Definition 3.1 (Horn clause) *A formula is a* Horn clause *formula if it can be generated by the following grammar.*

$$\begin{aligned}\mathcal{P} &::= T \mid F \mid p \ \ \{\textit{where } p \textit{ is a proposition}\}\\ \beta &::= \mathcal{P} \mid (\mathcal{P} \wedge \beta)\\ \alpha &::= (\beta \implies \mathcal{P}) \mid (\alpha \wedge \alpha)\end{aligned}$$

Here is an example of a Horn clause formula.

$$(\mathrm{T} \implies p) \wedge ((p \wedge q) \implies r) \wedge (\mathrm{F} \implies q) \wedge (p \implies \mathrm{F})$$

Here are some examples of formulas which are not Horn clause formulas.

1. $p \implies (q \wedge r)$: Implications cannot have conjunctions on the right side.
2. $\neg p \implies q$: Negations are not allowed
3. $p \implies (q \implies r)$: Implications cannot be nested.

Here is an alternate definition of Horn Clause formulas.

Definition 3.2 (Horn clause) *A CNF formula α is called a Horn clause formula if the clauses in α contains at most one positive literal.*

Let us look at the example given above and express it in equivalent CNF form.

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

We next show that the two statements are equivalent.

Lemma 3.10 *Let α be a wff. There is a Horn clause formula β according to definition 3.1 which is equivalent to α if and only if there is a Horn clause CNF formula γ according to definition 3.2 which is equivalent to α*

Proof : We leave the proof for the reader. ∎

The polynomial time algorithm for checking satisfiability of Horn-clause formulas is given in Algorithm 6.

The correctness of the algorithm follows from the two lemmas below.

Lemma 3.11 *If the Horn-clause-SAT algorithm outputs NO, then the formula is not satisfiable.*

Algorithm 6 Horn Clause Satisfiability

Input A Horn clause formula α
Output: NO if α is unsatisfiable, otherwise a satisfying assignment

1: **function** HornClauseSAT(α)
2: Mark proposition p if (T $\Rightarrow p$) is a clause in α.
3: **while** (there exists a clause $(p_1 \wedge p_2 \wedge \cdots \wedge p_k) \Rightarrow p_{k+1}$ such that p_i is marked for all $i \leq k$ but p_{k+1} is not) **do**
4: Mark p_{k+1}
5: **if** (there exists a F which is marked) **then return** NO
6: **end while**
7: **return** assignment which maps all marked propositions to T and rest to F
8: **end function**

Proof: We first show the following loop invariant.

> *Let v be a satisfying assignment of α. Then, the marked propositions will be assigned true in v.*

The loop invariant holds before we enter the while loop in line (2), since only T is marked. So, let us assume that the loop invariant is true before it enters the while loop. We show that, the loop invariant holds after a run of the while loop. Consider that the while loop marks a proposition p_{k+1} because all propositions $p_1, \ldots, p_k$ was marked and α contains the clause $(p_1 \wedge p_2 \wedge \cdots \wedge p_k) \Rightarrow p_{k+1}$. Let v be an arbitrary assignment which makes α true. We know from the loop invariant that v assigns true to all propositions $p_1, \ldots, p_k$. Since v also satisfies the clause $(p_1 \wedge p_2 \wedge \cdots \wedge p_k) \Rightarrow p_{k+1}$ (note that α is a conjunction of such clauses), v should necessarily assign p_{k+1} to true. Therefore, the loop invariant remains to hold once we exit the while loop also.

We can now prove the lemma. Let us assume that the algorithm outputs No. Therefore, F was marked. Our loop invariant says that all valuations which satisfies α should necessarily be such that it assigns true to F. This is not possible and hence the formula is not satisfiable. ∎

Lemma 3.12 *If the Horn-Clause-SAT algorithm outputs YES, then the formula is satisfiable.*

Proof: Let α be the formula given as input to the algorithm. We first prove the following claim.

> *If p_{k+1} is not marked and there is a clause $p_1 \wedge p_2 \wedge \cdots \wedge p_k \Rightarrow p_{k+1}$ in α, then atleast one of $p_1, \ldots, p_k$ is not marked.*

Assume the above claim is false. Then all the propositions $p_1, \ldots, p_k$ are marked. But then step 2 of the algorithm will mark p_{k+1} also. This is a contradiction. Hence atleast one of $p_1, \ldots, p_k$ is not marked.

We use the above claim to prove the lemma. There can be two types of clauses, (1) either a clause has all propositions marked or (2) it has atleast one proposition not marked. So consider the latter case. A clause with atleast one proposition not

marked. From our above claim, it follows that atleast one proposition in the left hand side of the implication is not marked. The algorithm ensures this clause is satisfied, since all propositions not marked are assigned false. Now, consider the former case. A clause whose all propositions are marked. The algorithm assigns all propositions to true and hence the clause is satisfied. Therefore the clause is satisfied by the valuation given by the algorithm. ■

3.3 Analytical/Semantic Tableaux

3.4 DPLL algorithm

The algorithm takes a CNF formula and a partial assignment to the propositions as input. Initially it is called with an empty assignment. The algorithm recursively picks a proposition to be assigned a true or false value and extends the partial assignment. At any instance if the constructed assignment satisfies the formula, the algorithm immediately returns YES. On the other hand, if an assignment does not satisfy a formula, we change the assignment to a proposition.

The simplest DPLL algorithm is given in Algorithm 7.

Algorithm 7 DPLL for CNF Satisfiability

Input A CNF formula α and a partial assignment v to the propositions
Output: YES if α is satisfiable, NO otherwise.

```
1: function DPLL(α,v)
2:     if (v assigns values to all the propositions) then
3:         if (v satisfies α) then return YES
4:         else return NO
5:     end if
6:     Let p = a proposition which is not assigned a value by v
7:     if (DPLL(α,v ∪ {p = T}) = YES) then return YES
8:     return DPLL(α,v ∪ {p = F})
9: end function
```

Let us see the working of the algorithm using an example.

EXAMPLE 3.2

The tree in Figure 3.3 depicts the run of the algorithm for the CNF formula:

$$\big((p \vee q \vee r) \wedge (\neg p \vee \neg q) \wedge (\neg q \vee r) \wedge (p \vee \neg r)\big)$$

In the figure, each node represents a partial valuation.

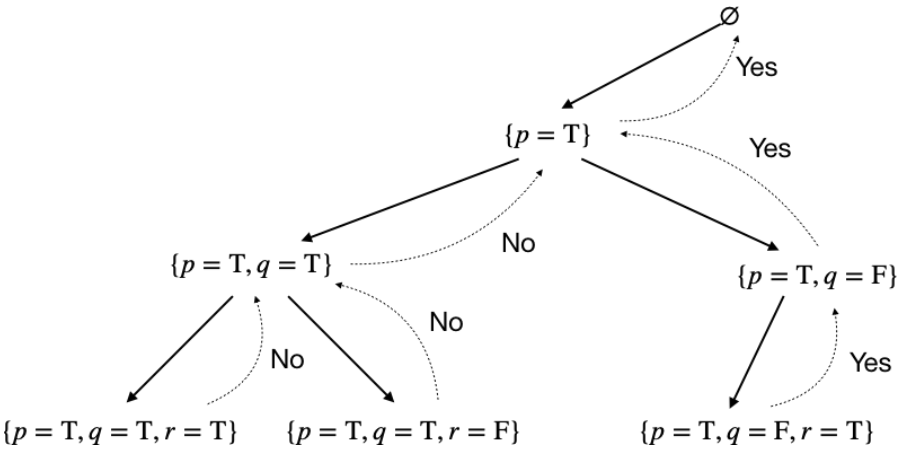


Figure 3.3 The run of DPLL algorithm for the formula $\big((p \vee q \vee r) \wedge (\neg p \vee \neg q) \wedge (\neg q \vee r) \wedge (p \vee \neg r)\big)$. A node represents the partial valuation at that point. The returns paths are labelled by YES or NO.

The algorithm can be extended by the preprocessing step we saw for Resolution.

Algorithm 8 Extended DPLL

Input A CNF formula α and a partial assignment v to the propositions
Output: YES if α is satisfiable, NO otherwise.

```
1:  function DPLLExtended(α,v)
2:      if (v assigns values to all the propositions) then
3:          if (v satisfies α) then return YES
4:          else return NO
5:      end if
6:      α =Preprocessing(α)
7:      Let p = a proposition which is not assigned a value by v
8:      if (DPLLExtended(α,v ∪ {p = T}) = YES) then return YES
9:      return DPLLExtended(α,v ∪ {p = F})
10: end function
11: function Preprocessing(α)
12:     repeat
13:         α′ = α.
14:         α =PureLiteralElimination(α).
15:         α = UnitClauseRemoval(α).
16:         α = TautologyElimination(α).
17:     until α = α′
18: end function
```

3.5 Summary and Acknowledgements

We followed the resolution algorithm from Ben-Ari [BA12] and Uwe Schöning [Sch89]. The DPLL algorithm follows the presentation in Russell and Norvig [RN95].

3.6 Chapter exercises

3.1 Let us assume you have the following programs.

1. Horn-SAT: The program on input a formula α outputs Yes if α is a satisfiable Horn clause formula. Otherwise it outputs No.
2. 2CNF-SAT: The program on input a formula α outputs Yes if α is a satisfiable 2CNF formula. Otherwise it outputs No.

Use these programs to check whether the following formulas are satisfiable or not.

(a) Let α be a conjunction of clauses (disjunction of literals) with at most one literal negated in a clause.

(b) α is generated by the following grammar. G is the start symbol and p_is are propositions.

$P := p_1 \mid p_2 \mid \ldots \mid p_n \mid \neg p_1 \mid \neg p_2 \mid \ldots \mid \neg p_n$
$C := P \wedge C \mid P$
$G := C \mid (P \Longrightarrow G)$

3.2 Consider the following wff.

$$\Big(p \wedge \big(p \longrightarrow ((q \vee r) \wedge \neg(q \wedge r))\big) \wedge \big(p \longrightarrow ((s \vee t) \wedge \neg(s \wedge t))\big) \wedge \big(s \longrightarrow q\big) \wedge \big((\neg r) \longrightarrow t\big) \wedge \big(t \longrightarrow s\big)\Big)$$

Answer the following questions.

1. Give a CNF formula equivalent to the above formula (Let us call this CNF formula α).
2. List the clauses in α and identify the unit clauses.
3. Give a DNF formula equivalent to the complement of α.
4. Run the resolution algorithm to check if α is satisfiable or not. Mention each rule you apply and the translation which happens to the formula.

3.3 Consider the following algorithm for checking HornClauseSAT. Prove the soundness and completeness of the algorithm.

Algorithm 9 Horn Clause Satisfiability

Input A Horn clause formula α
Output: NO if α is unsatisfiable, otherwise YES

```
1: function HornClauseCNFSAT(α)
2:     repeat
3:         α′ = α
4:         α =UnitClauseElimination(α)
5:         if (p and (¬p)) are clauses in α) then return NO
6:     until (α′ = α) return YES
7: end function
```

3.4 The following questions are about the relationship between P, NP and co-NP. We only know that P is a subset of both NP and co-NP. Consider the following situations. For each of them, check whether these situations will lead to new relationships between P, NP and co-NP. You also need to explain why?

1. Assume Algorithm 4. runs in polynomial number of steps for both YES and NO instances.
2. Assume whenever Algorithm 4. returned NO, the while loop executed polynomial number of steps.
3. Assume, there is a choice of clauses for resolution such that whenever algorithm returned NO, the while loop executed only polynomial number of steps.

3.5 Answer the following questions based on 2-CNF formulas and the Resolution algorithm (Algorithm 4).

1. Let α be a CNF formula which contains n propositions and m clauses. We give α as input to Algorithm 4. Each time, the while loop is executed, the formula α changes. How many clauses can α have in the worst case? Why?

2. What is a 2-CNF formula?

3. Let α be a 2-CNF formula containing n propositions and m clauses. We give this α as input to Algorithm 4. How many clauses can α have in the worst case? Why?

4. Argue that Algorithm 4 runs in polynomial time for a 2-CNF formula. You can assume that the algorithm runs correctly for your argument.

5. The algorithm does not run in polynomial time for a 3-CNF formula. What goes wrong in this case?

CHAPTER 4

PROOF SYSTEM

Mathematical proofs typically use a set of premises along with some rules to derive a theorem. In this chapter we will look at a set of rules (called *natural deduction*) to derive a theorem from a set of axioms. For a set of wffs Γ (called premises) and a wff β, we will denote by

$$\Gamma \vdash \beta$$

if using the rules in natural deduction and starting from the premises Γ we can derive the wff β. There are two important properties the natural deduction set of rules satisfy.

Theorem 4.1 (Soundness) *If* $\Gamma \vdash \beta$, *then* $\Gamma \models \beta$.

The soundness theorem shows that statements we can prove, are all true statements assuming the premises are true. In other words, using the natural deduction proof rules, we cannot derive false theorems. Every proof system expects this property. A proof system without soundness does not make much sense.

Natural deduction also satisfy the following interesting property.

Theorem 4.2 (Completness) *If* $\Gamma \models \beta$, *then* $\Gamma \vdash \beta$.

The completeness theorem says that, all true statements in the axiom system can be proved using the natural deduction proof rules. Mathematicians are interested in completeness. It assures them that all true theorems can be proved and therefore it is worthwhile to search for proofs. Do we have completeness for every logic? Gödel showed that there is a logic which is not complete (infact the logic is embedded in set theory, making set theory not complete). That means there are true statements in the logic which cannot be proved.

If we want to say $\alpha \vdash \beta$ and $\beta \vdash \alpha$ then we use the notation $\alpha \dashv\vdash \beta$.

4.1 Natural Deduction

We will now develop the rules to derive theorems from a set of axioms. Keep in mind that a proof is a sequence of formulas each of them generated by applying some rule on the previously generated formulas. Therefore, we need to identify rules by which we can introduce the logical symbols $\{\neg, \vee, \wedge, \Longrightarrow\}$. We also need to identify rules by which each of these symbols can be eliminated. We will now list all the rules of natural deduction.

(axiom rule) If $\phi \in \Gamma$*, then* $\Gamma \vdash \phi$. The rules says that if ϕ is a premise then it can be proved from the wffs in Γ.

(and-introduction rule) If $\Gamma \vdash \alpha$ *and* $\Gamma \vdash \beta$*, then* $\Gamma \vdash (\alpha \wedge \beta)$. Let us assume that we already have a proof of α and a proof of β from the premises Γ. The and-introduction rule can now be applied to get a proof of $(\alpha \wedge \beta)$. That is, $\Gamma \vdash \alpha \wedge \beta$

As you would expect the rule holds for any set of premises. The following pictorial representation, therefore avoids the premises to represent this rule.

$$\frac{\alpha \qquad \beta}{\alpha \wedge \beta} \wedge i$$

Figure 4.1 And-Introduction ($\wedge i$)

On the top of the separating line we have α and β, the two formulas for which we already have a proof. The formula below the line is a consequent of the formulas mentioned above and applying the and-introduction rule. The rule is mentioned on the right side of the line. This pictorial representation will be used for mentioning other rules too.

Here is an example of using this rule:

EXAMPLE 4.1

Below we show that $\{A \subseteq B, A \subseteq C\} \vdash (A \subseteq B \wedge A \subseteq C)$.

1.	$A \subseteq B$	premise
2.	$A \subseteq C$	premise
3.	$(A \subseteq B \wedge A \subseteq C)$	$\wedge i$, 1,2

Figure 4.2 and introduction

(and-elimination rule) If $\Gamma \vdash (\alpha \wedge \beta)$*, then* $\Gamma \vdash \alpha$ *and* $\Gamma \vdash \beta$*.* This rule says the following. Let us assume we have a proof of $(\alpha \wedge \beta)$. What else can we infer from this? Isnt it true that if $(\alpha \wedge \beta)$ is true, then individually both of them have to be true too. This is what the and-elimination rule says. If we can prove $\alpha \wedge \beta$, then we can prove α (and similarly β).

$$\frac{\alpha \wedge \beta}{\alpha} \wedge e_1 \qquad \frac{\alpha \wedge \beta}{\beta} \wedge e_2$$

Figure 4.3 and-elimination rules

The and-elimination has two rules. One to prove the left hand side of the conjunction. The other to derive the right hand side. Why do we require two rules? Isnt only the left rule enough? Note that the natural deduction rules does not assume any property of conjunction. In other words, it is not assumed that conjunction is a commutative operation. In fact commutativity is something we can prove using the rules we have seen till now. Let us consider an example

EXAMPLE 4.2

Below we show that $\{(A \subseteq B \wedge A \subseteq C)\} \vdash (A \subseteq B)$.

1.	$(A \subseteq B \wedge A \subseteq C)$	premise
2.	$A \subseteq B$	$\wedge e_1$ 1

Figure 4.4 and elimination

The following example combines both and elimination and introduction.

EXAMPLE 4.3

We show that $\{(A \subseteq B \wedge A \subseteq C), A \subseteq D\} \vdash (A \subseteq B \wedge A \subseteq D)$.

1.	$(A \subseteq B \wedge A \subseteq C)$	premise
2.	$A \subseteq D$	premise
3.	$A \subseteq B$	$\wedge e_1$ 1
4.	$(A \subseteq B \wedge A \subseteq D)$	$\wedge i$ 3,2

Figure 4.5 Proof of $\{(A \subseteq B \wedge A \subseteq C), A \subseteq D\} \vdash (A \subseteq B \wedge A \subseteq D)$

(double-negation elimination) If $\Gamma \vdash \neg\neg\alpha$, *then* $\Gamma \vdash \alpha$. Consider the following statement

"It is not true that it is not raining."

The above statement uses two negations to say, "It is raining". This rule says that such double negations can be eliminated.

$$\frac{\neg\neg\alpha}{\alpha}\ \neg\neg e$$

Figure 4.6 Double negation elimination ($\neg\neg e$)

(double-negation introduction) If $\Gamma \vdash \alpha$, *then* $\Gamma \vdash \neg\neg\alpha$. The double negation can be introduced by the following rule

$$\frac{\alpha}{\neg\neg\alpha}\ \neg\neg i$$

Figure 4.7 Double negation introduction ($\neg\neg i$)

Let us look at an example.

EXAMPLE 4.4

We show that $\{\neg\neg(A \subseteq B \wedge A \subseteq C), A \subseteq D\} \vdash \neg\neg(A \subseteq D) \wedge (A \subseteq C)$.

1.	$\neg\neg(A \subseteq B \wedge A \subseteq C)$	premise
2.	$A \subseteq D$	premise
3.	$(A \subseteq B \wedge A \subseteq C)$	$\neg\neg e$ 1
4.	$A \subseteq C$	$\wedge e_2$ 3
5.	$\neg\neg(A \subseteq D)$	$\neg i$ 2
6.	$\neg\neg(A \subseteq D) \wedge (A \subseteq C)$	$\wedge i$ 5,4

Figure 4.8 Proof of $\{\neg\neg(A \subseteq B \wedge A \subseteq C), A \subseteq D\} \vdash \neg\neg(A \subseteq D) \wedge (A \subseteq C)$

(implication elimination) If $\Gamma \vdash \alpha$ *and* $\Gamma \vdash (\alpha \Longrightarrow \beta)$*, then* $\Gamma \vdash \beta$*.* Implication elimination is something which is very natural. The high school mathematics has lots of proofs with implication elimination without explicitly mentioning it. This rule is also called as *modus ponens* . It says that, if we have a proof for $(\alpha \Longrightarrow \beta)$ and we have a proof for α, then β can also be proved. Note that, if formulas $(\alpha \Longrightarrow \beta)$ is true and α is true, then β is true necessarily (see truth table for implication). The rule for eliminating implication is given below.

$$\frac{\alpha \qquad \alpha \Longrightarrow \beta}{\beta} \Longrightarrow e$$

Figure 4.9 Implication-elimination ($\Longrightarrow e$)

EXAMPLE 4.5

The following argument makes use of implication elimination: If there is fire, there is smoke. There is fire. Therefore there is smoke.

1.	If there is fire, there is smoke.	premise
2.	There is fire.	premise
3.	Therefore, there is smoke.	$\Longrightarrow e$ 2,1

Figure 4.10 Proof using implication elimination.

(implication introduction) If $\Gamma \cup \{\alpha\} \vdash \beta$*, then* $\Gamma \vdash (\alpha \Rightarrow \beta)$. This rule is different from the kind of rules we have seen till now. It says that, if we assume α and are able to derive β, then we should be able to prove $(\alpha \Rightarrow \beta)$. It will require some time to convince yourself that this rule is not "nonsense". We denote this using our pictorial representation as follows.

$$\frac{\begin{array}{c}\alpha \\ \vdots \\ \beta\end{array}}{\alpha \Rightarrow \beta} \Rightarrow i$$

Figure 4.11 Implication-introduction ($\Rightarrow i$)

The following example will use both implication introduction and elimination.

EXAMPLE 4.6

1.	$\big(A \subseteq B \wedge B \subseteq C\big) \Rightarrow (A \subseteq C)$	premise
2.	$A \subseteq B$	assumption
3.	$B \subseteq C$	assumption
4.	$\big(A \subseteq B \wedge B \subseteq C\big)$	$\wedge i$ 2,3
5.	$A \subseteq C$	$\Rightarrow e$ 4,1
6.	$\big((B \subseteq C) \Rightarrow (A \subseteq C)\big)$	$\Rightarrow i$ 3-5
7.	$(A \subseteq B) \Rightarrow \big((B \subseteq C) \Rightarrow (A \subseteq C)\big)$	$\Rightarrow i$ 2-6

Figure 4.12 Proof of $(\alpha \wedge \beta) \Rightarrow \gamma \vdash \alpha \Rightarrow (\beta \Rightarrow \gamma)$

Here is another example. For any formula α and any set of premises Γ, we can prove $\Gamma \vdash (\alpha \Rightarrow \alpha)$.

1.	α	assume
2.	$\alpha \Longrightarrow \alpha$	$\Longrightarrow i$ 1-1

Figure 4.13 Proof of $\{T\} \vdash (\alpha \Longrightarrow \alpha)$, for any wff α.

(disjunction introduction) If $\Gamma \vdash \alpha$, *then* $\Gamma \vdash (\alpha \vee \beta)$ *for any wff* β. Let us assume that we have a proof of α. Then clearly we have a proof of $(\alpha \vee \beta)$, no matter what β is. This is because if α is true, $(\alpha \vee \beta)$ is true for all β. The following rules introduces this disjunction symbol. Note that, since we do not know about the commutativity of disjunction we need a second rule too: *If* $\Gamma \vdash \alpha$, *then* $\Gamma \vdash (\beta \vee \alpha)$ *for any wff* β.

$$\frac{\alpha}{\alpha \vee \beta} \vee i_1 \qquad\qquad \frac{\beta}{\alpha \vee \beta} \vee i_2$$

Figure 4.14 disjunction introduction

Here is a simple example of disjunction introduction.

1.	$A \subseteq B$	premise
2.	$(A \subseteq B) \vee (A \subseteq C)$	$\vee i_1$ 1

Figure 4.15 An example of disjunction introduction

(disjunction elimination) If $\Gamma \cup \{\alpha\} \vdash \gamma$, $\Gamma \cup \{\beta\} \vdash \gamma$ *and* $\Gamma \vdash (\alpha \vee \beta)$, *then* $\Gamma \vdash \gamma$. Let us assume we have a proof of $\alpha \vee \beta$. Moreover we have a proof of γ assuming α is a premise. Similarly we have a proof of γ assuming β is a premise. We can therefore infer that γ should be provable from the original set of premises. This is what disjunction elimination helps us achieve.

$$\frac{\alpha \vee \beta \qquad \begin{array}{c}\alpha \\ \vdots \\ \gamma\end{array} \qquad \begin{array}{c}\beta \\ \vdots \\ \gamma\end{array}}{\gamma} \vee e$$

Figure 4.16 Disjunction-elimination ($\vee e$)

Let us look at an example.

EXAMPLE 4.7

In this example we show that $\{(A \subseteq B \vee A \subseteq C), B \subseteq D, (A \subseteq B \wedge B \subseteq D) \Longrightarrow A \subseteq D\} \vdash (A \subseteq D \vee A \subseteq C)$ using disjunction elimination.

1.	$(A \subseteq B \vee A \subseteq C)$	premise
2.	$B \subseteq D$	premise
3.	$(A \subseteq B \wedge B \subseteq D) \Longrightarrow A \subseteq D$	premise
4.	$A \subseteq B$	assumption
5.	$(A \subseteq B \wedge B \subseteq D)$	$\wedge i$ 4,2
6.	$A \subseteq D$	$\Longrightarrow e$ 5,3
7.	$(A \subseteq D \vee A \subseteq C)$	$\vee i_1$ 6
8.	$A \subseteq C$	assumption
9.	$(A \subseteq D \vee A \subseteq C)$	$\vee i_2$ 8
10.	$(A \subseteq D \vee A \subseteq C)$	$\vee e$ 1, 4-7, 8-9

Figure 4.17 Proof of $\{\alpha_1 \vee \alpha_2, \beta, (\alpha_1 \wedge \beta) \Longrightarrow \gamma\} \vdash \gamma \vee \alpha_2$

(contradiction introduction) If $\Gamma \vdash \alpha$ *and* $\Gamma \vdash \neg\alpha$*, then* $\Gamma \vdash \mathbf{F}$. Let us assume we are able to prove α and also prove $\neg\alpha$. Clearly there is a contradiction.

$$\frac{\alpha \qquad \neg\alpha}{\mathrm{F}}\ \mathrm{F}i$$

Figure 4.18 Contradiction introduction (Fi)

(proof by contradiction) If $\Gamma \cup \{\alpha\} \vdash \mathbf{F}$*, then* $\Gamma \vdash \neg\alpha$. We are used to proof by contradiction. This proof strategy assumes that a certain property is true and use that to prove a contradiction. Therefore, we can assume that our assumption was wrong.

$$\begin{array}{c} \alpha \\ \vdots \\ \mathrm{F} \\ \hline \neg\alpha \end{array} \mathrm{F}e$$

Figure 4.19 Proof by contradiction (Fe)

EXAMPLE 4.8

The following argument makes use of proof by contradiction: If there is fire, there is smoke. There is no smoke. Therefore, there is no fire.

1.	If there is fire, there is smoke.	premise
2.	There is no smoke.	premise
3.	There is fire.	assumption
4.	Therefore, there is smoke.	$\Longrightarrow e$ 3,1
5.	A contradiction.	Fi 2,4
6.	Therefore, there is no fire.	Fe 3-5

Figure 4.20 Natural deduction proof of modus tollens.

The above example, is also called *modus tollens* . As we have observed, modus tollens can be derived by the other rules.

$$\frac{\neg\beta \qquad \alpha \Longrightarrow \beta}{\neg\alpha} \text{ modus tollens}$$

Figure 4.21 Modus tollens rule - This is a derived rule.

We have now seen all the rules of natural deduction. See Figure 4.22. We can now look at examples. First we will look at some derived rules, like the above modus tollens rule.

The next derivation is similar to the proof by contradiction (except for the sign in α). If $\Gamma \cup \{\neg\alpha\} \vdash \mathrm{F}$, then $\Gamma \vdash \alpha$. See Figure 4.23.

Introduction	Elimination
$\dfrac{\alpha \quad \beta}{\alpha \wedge \beta}\wedge i$	$\dfrac{\alpha \wedge \beta}{\alpha}\wedge e_1 \qquad \dfrac{\alpha \wedge \beta}{\beta}\wedge e_2$
$\dfrac{\alpha}{\neg\neg\alpha}\neg\neg i$	$\dfrac{\neg\neg\alpha}{\alpha}\neg\neg e$
$\dfrac{\begin{array}{c}\alpha\\ \vdots\\ \beta\end{array}}{\alpha \Rightarrow \beta}\Rightarrow i$	$\dfrac{\alpha \quad \alpha \Rightarrow \beta}{\beta}\Rightarrow e$
$\dfrac{\alpha}{\alpha \vee \beta}\vee i_1 \qquad \dfrac{\beta}{\alpha \vee \beta}\vee i_2$	$\dfrac{\alpha \vee \beta \quad \begin{array}{c}\alpha\\ \vdots\\ \gamma\end{array} \quad \begin{array}{c}\beta\\ \vdots\\ \gamma\end{array}}{\gamma}\vee e$
	$\dfrac{F}{\alpha}Fe$
$\dfrac{\begin{array}{c}\alpha\\ \vdots\\ F\end{array}}{\neg\alpha}\neg i$	$\dfrac{\alpha \quad \neg\alpha}{F}\neg e$

Figure 4.22 The rules of Natural Deduction

$$\dfrac{\begin{array}{c}\neg\alpha\\ \vdots\\ F\end{array}}{\alpha}\neg e$$

Figure 4.23 Negation elimination

The above rule can be derived by the following natural deduction proof.

1.	$\neg\alpha$	assumption
2.	...	
3.	F	following the proof of $\Gamma \cup \{\neg\alpha\} \vdash F$
4.	$\neg\neg\alpha$	F*e* 1-3
5.	α	$\neg\neg e$ 4

Figure 4.24 Natural deduction for negation elimination.

The next derivation says that: from a contradiction we can derive any formula.

$$\frac{\mathrm{F}}{\alpha}$$

Figure 4.25 "Contradiction implies anything"

Here is a proof of the above claim.

1.	$\neg\alpha$	assumption
2.	F	premise
3.	α	$\neg e$ 1-2

Figure 4.26 Natural deduction proof of "contradiction implies anything".

We now see one very important rule called "law of excluded middle (LEM)". It says that, for any wff α, we have $\{\mathrm{T}\} \vdash (\alpha \vee \neg\alpha)$.

$$\frac{}{(\alpha \vee \neg\alpha)}\ \mathrm{LEM}$$

Figure 4.27 "Law of exclude middle (LEM)"

1.	$\neg(\alpha \vee \neg\alpha)$	assumption
2.	α	assumption
3.	$(\alpha \vee \neg\alpha)$	$\vee i_1$ 2
4.	F	Fi 1,3
5.	$\neg\alpha$	Fe 2-4
6.	$(\alpha \vee \neg\alpha)$	$\vee i_2$ 5
7.	F	Fi 1,6
8.	$(\alpha \vee \neg\alpha)$	Fe 1-7

Figure 4.28 Natural deduction proof of "law of excluded middle (LEM)".

Let us now look at some examples of natural deduction. We first show that $\neg\alpha \vdash \alpha \Longrightarrow \beta$

1.	$\neg\alpha$	premise
2.	α	assumption
3.	F	Fi 2, 1
4.	β	"false implies anything" 3
5.	$\alpha \Longrightarrow \beta$	$\Longrightarrow i$ 2-4

Figure 4.29 Proof of $\neg\alpha \vdash \alpha \Longrightarrow \beta$

1.	$\neg\alpha$	premise
2.	$\alpha \wedge \beta$	assumption
3.	α	$\wedge e_1$ 2
4.	F	Fi 3,1
5.	$\neg(\alpha \wedge \beta)$	Fe 2-4

Figure 4.30 Proof of $\neg\alpha \vdash \neg(\alpha \wedge \beta)$

Finally, we show that $\alpha \vee \beta, \alpha \vee \neg\beta \vdash \alpha$.

1.	$\alpha \vee \beta$	premise
2.	$\alpha \vee \neg\beta$	premise
3.	$\neg\alpha$	assumption
4.	α	assumption
5.	F	$\neg e$ 4,3
6.	β	Fe 5
7.	β	assumption
8.	β	$\vee e$ 1, 4-6, 7
9.	α	assumption
10.	F	$\neg e$ 9,3
11.	$\neg\beta$	Fe 10
12.	$\neg\beta$	assumption
13.	$\neg\beta$	$\vee e$ 2, 9-11,12
14.	F	$\neg e$ 8,13
15.	$\neg\neg\alpha$	$\neg i$ 3-14
16.	α	$\neg\neg e$ 15

Figure 4.31 Proof of $\alpha \vee \beta, \alpha \vee \neg\beta \vdash \alpha$

Problems

4.1 Prove the following.

1. (commutative) $\alpha \wedge \beta \vdash \beta \wedge \alpha$.

4.2 Prove the following.

1. (commutative) $\alpha \vee \beta \vdash \beta \vee \alpha$
2. (associative) $\alpha \vee (\beta \vee \gamma) \vdash (\alpha \vee \beta) \vee \gamma$
3. (distributive) $\alpha \wedge (\beta \vee \gamma) \dashv\vdash (\alpha \wedge \beta) \vee (\alpha \wedge \gamma)$.
4. $\alpha \vee (\beta \wedge \gamma) \dashv\vdash (\alpha \vee \beta) \vee (\alpha \vee \gamma)$.

4.3 [modus tollens] Show that $\alpha \Longrightarrow \beta, \neg\beta \vdash \neg\alpha$.

4.4 [LEM] Show that $\mathrm{T} \vdash \alpha \vee \neg\alpha$.

4.1.1 Soundness theorem

The proof of the theorem involves mathematical induction. If you are not used to induction, go through Chapter **??**.

Let us restate the soundness theorem first.

Theorem 4.3 (Soundness) *If* $\Gamma \vdash \psi$, *then* $\Gamma \models \psi$.

The theorem says that, all formulas proved using natural deduction are true in a world where the axioms are true. The rest of this section will be devoted to proving the soundness theorem. The proof is by mathematical induction on the length of the proof. The length of a proof is the number of steps required in a natural deduction proof. The induction hypothesis is as follows:

"For all set of formulas Γ, *if* $\Gamma \vdash \psi$ *where proof length is n, then* $\Gamma \models \psi$ *holds."*

Base Case ($n = 1$): The only proof of length 1 is as follows

$$\Gamma \vdash \alpha \ \ -\text{Axiom}$$

where $\alpha \in \Gamma$ is an axiom. From the definition of $\models$ it follows that $\Gamma \models \alpha$.

Inductive step: Let us assume that the induction hypothesis holds for all proofs of length less than or equal to n. We will show that the claim holds for proofs of length $n+1$. Consider one such proof. We will do a case analysis on the rule applied to derive ψ in the $n+1$th step of the proof.

Case $\wedge i$: The step is and-introduction. That is, we have $\Gamma \vdash \psi$ and ψ is of the form $\alpha \wedge \beta$ for some formulas α and β which were derived earlier in the proof. Hence we know that $\Gamma \vdash \alpha$ and $\Gamma \vdash \beta$. From induction hypothesis (since the proof lengths are less than $n+1$) it follows $\Gamma \models \alpha$ and $\Gamma \models \beta$. From the semantics of $\wedge$, we get $\Gamma \models \alpha \wedge \beta$.

Case $\wedge e$: That means, $\Gamma \vdash \psi$ and ψ has been derived by applying an and elimination from a formula of the form $\psi \wedge \alpha$ or $\alpha \wedge \psi$. We will assume the former (the latter will have a symmetric argument). This means, there was a step in the proof which derived $\psi \wedge \alpha$ and hence $\Gamma \vdash \psi \wedge \alpha$. Since the proof length of $\psi \wedge \alpha$ is less than $n+1$, from induction hypothesis we get $\Gamma \models \psi \wedge \alpha$. From the semantics of and ($\wedge$) it follows $\Gamma \models \psi$.

Case $\neg\neg i$: That is ψ is of the form $\neg\neg\alpha$ for an α which was derived earlier in the proof. From induction hypothesis and semantics of negation (applied twice), it follows $\Gamma \models \psi$.

Case $\neg\neg e$: We can assume ψ is got by elimination the double negation from a formula $\neg\neg\psi$ which was derived earlier in the proof. Again, applying induction hypothesis and using the semantics of negation, we get that $\Gamma \models \psi$.

Case $\Longrightarrow i$: Let us assume that $\Gamma \vdash \psi$ and ψ is of the form $\alpha \Longrightarrow \beta$ and the rule applied was implication-introduction. Therefore, after assuming α, there is a derivation of β in the proof. In other words, $\Gamma \cup \alpha \vdash \beta$. This proof is of length less than $n+1$ and hence $\Gamma \cup \alpha \models \beta$. From the semantics of implication, it follows that $\Gamma \models \alpha \Longrightarrow \beta$.

Similarly going through all the other cases will finish the proof of the soundness theorem. The reader is asked to try showing this.

Problems

4.1 Show the remaining cases, left out in Theorem 4.3.

4.1.2 Completeness theorem: Huth & Ryan

We say that a formula α is a *theorem* if α can be proved without assuming any axioms. That is, $\mathrm{T} \vdash \alpha$.

In this section we prove the completeness theorem in a weaker setting, where Γ is a finite set of formulas. The stronger result will be given later.

Theorem 4.4 (Completeness) *Let Γ be a finite set of formulas. Then, $\Gamma \models \psi$ implies $\Gamma \vdash \psi$.*

The theorem says that, if our formula is true in a world where the axioms are true, then the formula can be proved from the axioms. The rest of the section is proving the theorem. The proof strategy we follow is given in Figure 4.32.

Our first step is to reduce the completeness theorem to a simpler case.

Lemma 4.5 *If α is a tautology, then α is a theorem. That is, if $T \models \alpha$, then $T \vdash \alpha$*

Before we prove the lemma, let us show how it would imply completeness theorem. Let us assume that $\Gamma = \{\alpha_1, \ldots, \alpha_n\}$ and $\alpha_1, \alpha_2, \ldots, \alpha_n \models \psi$. From Exercise 1.3 we know this is equivalent to $\mathrm{T} \models (\alpha_1 \wedge \alpha_2 \wedge \ldots \alpha_n) \Longrightarrow \psi$. Our Lemma 4.5 shows

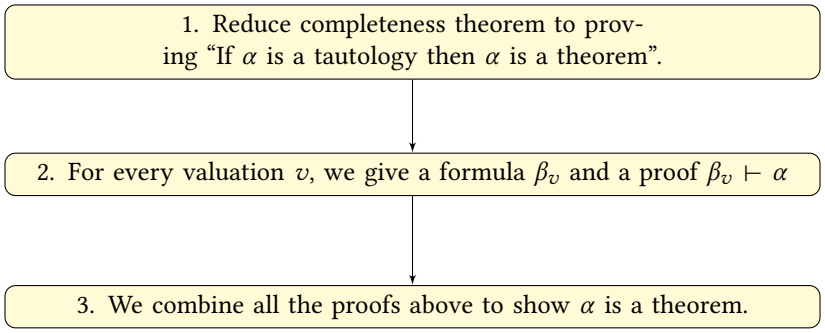


Figure 4.32 Proof of Completeness Theorem

that this is equivalent to $\mathrm{T} \vdash (\alpha_1 \wedge \alpha_2 \wedge \ldots \alpha_n) \implies \psi$. From Exercise 4.3 it follows that $\alpha_1, \alpha_2, \ldots, \alpha_n \vdash \psi$. This finishes the completeness theorem.

Now we can go the proof of Lemma 4.5.

Proof: [Proof of Lemma 4.5] Let P be the set of all propositions in α. For a particular valuation, v of P, we can define the following formula β_v

$$\beta_v = \bigwedge_{\substack{p \in P \\ v(p)=\mathrm{T}}} p \wedge \bigwedge_{\substack{p \in P \\ v(p)=\mathrm{F}}} \neg p$$

That is β_v is the conjunction of all propositions which are assigned true in the valuation along with the neg of all propositions which are assigned false. Let α be an arbitrary formula. Then the following claim holds for any valuation v because β_v is satisfied by exactly one valuation, namely v.

Claim 4.6

See exercise 4.9 for the proof of the above claim. We now prove the following for all subformulas ψ of α.

$$\begin{aligned} &\text{If } \beta_v \models \psi \text{ then } \beta_v \vdash \psi \\ &\text{If } \beta_v \not\models \psi \text{ then } \beta_v \vdash \neg\psi \end{aligned} \tag{4.1}$$

The proof is by structural induction on the parse tree of α. Let us do a case analysis of the type of node.

Case $\psi := p$: Let us first consider the case $\beta_v \models p$. Since p is a conjunct in β_v and-elimination gives us $\beta_v \vdash p$. Now if $\beta_v \not\models p$, then $\beta_v \models \neg p$, which implies $\neg p$ is a conjunct in β_v. Therefore, $\beta_v \vdash \neg p$.

Case $\psi := \gamma_1 \wedge \gamma_2$: Let us first consider the case $\beta_v \models \gamma_1 \wedge \gamma_2$.

$$\begin{aligned} &\text{Let } \beta_v \models \gamma_1 \wedge \gamma_2 \\ &\implies \beta_v \models \gamma_1 \text{ and } \beta_v \models \gamma_2 && \text{(semantics of and)} \\ &\implies \beta_v \vdash \gamma_1 \text{ and } \beta_v \vdash \gamma_2 && \text{(induction hypothesis)} \\ &\implies \beta_v \vdash \gamma_1 \wedge \gamma_2 && \text{(and-introduction)} \end{aligned}$$

Now let us prove the other if condition.

$$
\begin{aligned}
&\text{Let } \beta_v \not\models (\gamma_1 \wedge \gamma_2) \\
&\implies \beta_v \models \neg(\gamma_1 \wedge \gamma_2) && \text{(definition of semantic entailment)} \\
&\implies \beta_v \models \neg\gamma_1 \vee \neg\gamma_2 && \text{(Demorgan's law (proof in Exercise 1.2))} \\
&\implies \beta_v \not\models \gamma_1 \text{ or } \beta_v \not\models \gamma_2 && \text{(semantics of or)} \\
&\implies \beta_v \vdash \neg\gamma_1 \text{ or } \beta_v \vdash \neg\gamma_2 && \text{(induction hypothesis)} \\
&\implies \beta_v \vdash \neg\gamma_1 \vee \neg\gamma_2 && \text{(or-introduction)} \\
&\implies \beta_v \vdash \neg(\gamma_1 \wedge \gamma_2) && \text{(See Exercise 4.1 for this derivation)}
\end{aligned}
$$

We have shown both sides of equation 4.1.

Case $\psi := \gamma_1 \vee \gamma_2$: Let us first consider the case $\beta_v \models \gamma_1 \vee \gamma_2$. From the semantics of disjunction, it follows that $\beta_v \models \gamma_1$ or $\beta_v \models \gamma_2$. By induction hypothesis, we have $\beta_v \vdash \gamma_1$ or $\beta_v \vdash \gamma_2$. From or-introduction, it follows that $\beta_v \vdash \gamma_1 \vee \gamma_2$. Now let us assume $\beta_v \not\models \gamma_1 \vee \gamma_2$. From the semantics of disjunction and negation, it follows that $\beta_v \models \neg\gamma_1$ and $\beta_v \models \neg\gamma_2$. By induction hypothesis, we have $\beta_v \vdash \neg\gamma_1$ and $\beta_v \vdash \neg\gamma_2$. Now exercise 4.1 gives us $\beta_v \vdash \neg(\gamma_1 \vee \gamma_2)$.

Case $\psi := \neg\gamma$: Let us first assume $\beta_v \models \neg\gamma$. Therefore $\beta_v \not\models \gamma$. From induction hypothesis, therefore it follows that $\beta_v \vdash \neg\gamma$. Now, let us assume $\beta_v \not\models \neg\gamma$. This is equivalent to $\beta_v \models \gamma$ which from induction hypothesis gives us $\beta_v \vdash \gamma$. Introducing double negation will give us $\beta_v \models \neg\neg\gamma$.

Case $\psi := \gamma_1 \Rightarrow \gamma_2$: Let us consider the case $\beta_v \models \gamma_1 \Rightarrow \gamma_2$.

$$
\begin{aligned}
&\text{Let } \beta_v \models \gamma_1 \Rightarrow \gamma_2 \\
&\implies \beta_v \models \neg\gamma_1 \vee \gamma_2 && \text{(see Exercise 1.1)} \\
&\implies \beta_v \not\models \gamma_1 \text{ or } \beta_v \models \gamma_2 && \text{(semantics of or and negation)} \\
&\implies \beta_v \vdash \neg\gamma_1 \text{ or } \beta_v \vdash \gamma_2 && \text{(induction hypothesis)} \\
&\implies \beta_v \vdash \neg\gamma_1 \vee \gamma_2 && \text{(or-introduction)} \\
&\implies \beta_v \vdash \gamma_1 \Rightarrow \gamma_2 && \text{(see Exercise 4.4)}
\end{aligned}
$$

Now let us assume $\beta_v \not\models \gamma_1 \Rightarrow \gamma_2$.

$$
\begin{aligned}
&\text{Let } \beta_v \not\models \gamma_1 \Rightarrow \gamma_2 \\
&\implies \beta_v \models \neg(\neg\gamma_1 \vee \gamma_2) && \text{(see Exercise 1.1)} \\
&\implies \beta_v \models \gamma_1 \text{ and } \beta_v \not\models \gamma_2 && \text{(semantics of or and negation, demorgan)} \\
&\implies \beta_v \vdash \gamma_1 \text{ and } \beta_v \vdash \neg\gamma_2 && \text{(induction hypothesis)} \\
&\implies \beta_v \vdash \gamma_1 \wedge \neg\gamma_2 && \text{(and-introduction)} \\
&\implies \beta_v \vdash \neg(\gamma_1 \Rightarrow \gamma_2) && \text{(see Exercise 4.4)}
\end{aligned}
$$

Thus equation 4.1 is true for the implication case.

We have exhausted all the ways in which formulas can be built. Therefore, the claim in equation 4.1 holds. Let us go back to the lemma and assume its hypothesis, $\mathrm{T} \models \alpha$. That is, for all valuations v over the propositions, $\beta_v \models \alpha$. From our discussion above we have $\beta_v \vdash \alpha$. Exercise 4.8 now gives us $\mathrm{T} \vdash \alpha$. ∎

4.1.3 Completeness: Alternate proof using Hintikka sets*

In this section we give an alternate proof for completeness. We prove the simpler version, namely

Lemma 4.5. *If $T \models \alpha$, then $T \vdash \alpha$*

As seen in the previous section, the above lemma along with the exercises 1.3 and 4.3, will give us the completeness theorem. The rest of the section is for proving the above lemma.

Natural deduction gives us rules to derive proofs of statements. What is an important property these rules should satisfy? It should not help us to derive both a property and its negation. That is, we do not want natural deduction to satisfy $T \vdash \alpha$ and $T \vdash \neg\alpha$ for any formula α. In fact if it does, then using natural deduction you can prove any formula.

So what we are interested in is a property called *consistency* . We say that a formula α is *consistent* if $T \nvdash \neg\alpha$. That is, α is consistent if there is no proof for $\neg\alpha$. Note that, this does not say that there is a proof for α. The following theorem connects consistent formulas and Lemma 4.5. In fact, it also shows that natural deduction is consistent.

Claim 4.7 *The following statements are equivalent.*

1. *If $T \models \alpha$, then $T \vdash \alpha$*
2. *If $\neg\alpha$ is consistent, then $\neg\alpha$ is satisfiable.*

Proof: ($1 \implies 2$) : Let $\neg\alpha$ be consistent. That is $T \nvdash \neg\neg\alpha$. Therefore $T \nvdash \alpha$ (otherwise contradiction by double-negation introduction). From (1) we get $T \not\models \alpha$ and hence $\neg\alpha$ is satisfiable.
($2 \implies 1$) : Let $T \models \alpha$. Therefore $\neg\alpha$ is not satisfiable. From (2) we get $\neg\alpha$ is not consistent. In other words $T \vdash \neg\neg\alpha$. The claim now follows from double-negation elimination. ∎

We will now prove that "If β is consistent, then β is satisfiable". For a finite set X of formulas, we say X is consistent if the formula $\bigwedge_{\alpha\in X} \alpha$ is consistent. Given a consistent set X, we can extend the sets in a meaningful way as follows. Let us order all propositional logic formulas into a sequence

$$\alpha_0, \alpha_1, \ldots$$

We define $X_0 = X$ and for all $i \geq 0$ we define X_{i+1} as follows.

$$X_{i+1} = \begin{cases} X_i, \text{ if } X_i \cup \alpha_i \text{ is not consistent} \\ X_i \cup \alpha_i, \text{ otherwise} \end{cases}$$

We now define the maximal consistent extension of X, (denoted by $\hat{X}$) as $\bigcup_{i\geq 0} X_i$. This maximal consistent set satisfy some interesting properties.

Lemma 4.8 *Let $\hat{X}$ be as defined above. Then*

1. *For all* $i \geq 0$, $\alpha_i \in \hat{X}$ *iff* $\neg\alpha_i \notin \hat{X}$.
2. *For all* $i, j \geq 0$, $\alpha_i \wedge \alpha_j \in \hat{X}$ *iff* $\alpha_i \in \hat{X}$ *and* $\alpha_j \in \hat{X}$.
3. *For all* $i, j \geq 0$, $\alpha_i \vee \alpha_j \in \hat{X}$ *iff* $\alpha_i \in \hat{X}$ *or* $\alpha_j \in \hat{X}$.
4. *For all* $i, j \geq 0$, $\alpha_i \Longrightarrow \alpha_j \in \hat{X}$ *iff* $\alpha_i \notin \hat{X}$ *or* $\alpha_j \in \hat{X}$.

Proof: We will prove each of the above claims.

1. Let $\alpha_j = \neg\alpha_i$ and $k = max\{i, j\}$. We show that $\alpha_i \in X_k$ iff $\alpha_j \notin X_k$. Let $\beta = \bigwedge_{i=0}^{k} \alpha_i$. Let us first assume that both α_i and $\neg\alpha_i$ are not in X_k. Then, $\beta \wedge \alpha_i$ and $\beta \wedge \neg\alpha_i$ are not consistent. Therefore T $\vdash \neg(\beta \wedge \alpha_i)$ and T $\vdash \neg(\beta \wedge \neg\alpha_i)$. From Example **??**, it follows that T $\vdash \neg\beta$. This is a contradiction, since β is consistent. Now, let us assume both α_i and $\neg\alpha_i$ both are in X_k. That is, T $\nvdash \neg(\beta \wedge \alpha_i \wedge \neg\alpha_i)$. This is a contradiction from exercise 4.4. Therefore either one of α_i or α_j should be in X_k and hence in $\hat{X}$.
2. Let $\alpha_l = \alpha_i \wedge \alpha_j$ and $k = max\{i, j, l\}$. We show that $\alpha_l \in X_k$ iff $\alpha_i \in X_k$ and $\alpha_j \in X_k$. Let $\beta = \bigwedge_{i=0}^{k} \alpha_i$. First, let us assume $\alpha_l \notin X_k$. That is, T $\vdash \neg(\beta \wedge \alpha_i \wedge \alpha_j)$. From Demorgan's exercise 4.1 we know this is equivalent to T $\vdash \neg\beta \vee \neg\alpha_i \vee \neg\alpha_j$. Since β is consistent, T $\nvdash \neg\beta$. Therefore (semantics of disjunction) gives, T $\vdash \neg\alpha_i$ or T $\vdash \neg\alpha_j$. This shows that either $\alpha_i \notin X_k$ or $\alpha_j \notin X_k$. Now let us consider the other direction of the claim. Let $\alpha_i \notin X_k$ or $\alpha_j \notin X_k$. In other words, T $\vdash \neg\alpha_i$ or T $\vdash \neg\alpha_j$. Applying or-introduction and demorgan's laws we get T $\vdash \neg(\beta \wedge \alpha_i \wedge \alpha_j)$. Therefore $\alpha \in X_k$. This proves the forward direction of the claim.

We leave the rest of the claims for the reader to prove. ∎

Our next lemma says that if a formula β is in $\hat{X}$, then β is satisfiable.

Lemma 4.9 *If* $\beta \in \hat{X}$, *then* β *is satisfiable.*

Proof: Let V be a set which contains either propositions or its negations. We define V as follows. If $p \in \hat{X}$, then $p \in V$. On the other hand, if $p \notin \hat{X}$, then $\neg p \in V$. It is easy to see that, there is a satisfying assignment which makes all formulas in V true. We are done, if we show that $V \vDash \beta$. We prove the following induction hypothesis by structural induction on subformulas of β.

$$\gamma \in \hat{X} \iff V \vDash \gamma$$

Case $\gamma = p$, a proposition: If $p \in \hat{X}$, by definition $V \vDash \gamma$. On the other hand, if $p \notin \hat{X}$, we have by definition $V \vDash \neg p$ and therefore $V \nvDash p$.

Case $\gamma = \neg\psi$: If $\neg\psi \in \hat{X}$, then (by properties of $\hat{X}$, Lemma 4.8) $\psi \notin \hat{X}$. By our induction hypothesis it follows $V \nvDash \psi$ and therefore (semantics of negation) $V \vDash \neg\psi$. Let us assume $\neg\psi \notin \hat{X}$. By Lemma 4.8, $\psi \in \hat{X}$, which by IH gives us $V \vDash \psi$. Therefore $V \nvDash \neg\psi$.

Case $\gamma = \psi_1 \wedge \psi_2$: If $\psi_1 \wedge \psi_2 \in \hat{X}$, then (Lemma 4.8) $\psi_1 \in \hat{X}$ and $\psi_2 \in \hat{X}$. By IH, $V \models \psi_1$ and $V \models \psi_2$ and therefore $V \models \psi_1 \wedge \psi_2$. Let us now assume $\psi_1 \wedge \psi_2 \notin \hat{X}$. Therefore (Lemma 4.8) $\psi_1 \notin \hat{X}$ or $\psi_2 \notin \hat{X}$. From IH, we get $V \not\models \psi_1$ or $V \not\models \psi_2$. Therefore $V \models \neg\psi_1 \vee \neg\psi_2$. Which by demorgan's laws proves the case.

We leave the rest of the case as exercise. ∎

We now have enough understanding to prove the completeness theorem.

Proof of Lemma 4.5. Let $\neg\alpha$ be consistent. Extend the set $X = \{\neg\alpha\}$ to $\hat{X}$, the maximal consistent set. Lemma 4.9, gives that all formulas in $\hat{X}$ are satisfiable and therefore $\neg\alpha$ is satisfiable too. ∎

Problems

4.1 If $\mathrm{T} \vdash \alpha$ and $\mathrm{T} \vdash \neg\alpha$, then $\mathrm{T} \vdash \beta$ for all β.

4.2 Prove the remaining cases in Lemma 4.8.

4.3 Prove the remaining cases in Lemma 4.9.

4.1.4 Strong Completeness*

In Section 4.1.2 we saw the completeness theorem for the case when Γ is finite. In this section we will show that the completeness theorem is true even for the infinite case. Compactness theorem will help us in this regard.

Theorem 4.10 (Strong completeness) *Let Γ be a set of formulas and ψ be a formula. Then $\Gamma \vdash \psi$ if $\Gamma \models \psi$.*

Proof: Let $\Gamma \models \psi$. Therefore $\Gamma \not\models \neg\psi$. It follows that $\Gamma \cup \neg\psi$ is not satisfiable. The compactness theorem of propositional logic (Theorem 1.10) gives us that there exists a finite subset $\Gamma' \subseteq \Gamma$ such that $\Gamma' \cup \neg\psi$ is not satisfiable. Therefore we have $\Gamma' \not\models \neg\psi$. In other words $\Gamma' \models \psi$. From the completeness theorem for propositional logic (see Theorem 4.4) we have $\Gamma' \vdash \psi$. Since $\Gamma' \subseteq \Gamma$, we have $\Gamma \vdash \psi$. ∎

4.2 Summary and Acknowledgements

This chapter is heavily influenced by the natural deduction presentation in Huth and Ryan [HR04]. It also follows Enderton [End72]. The compactness proof follows the presentation of Mukund and Suresh [MS11].

4.3 Chapter exercises

4.1 [syntactic variant of De Morgan's law] Prove the following.

1. $\neg(\alpha \wedge \beta) \dashv\vdash \neg\alpha \vee \neg\beta$
2. $\neg\alpha \wedge \neg\beta \dashv\vdash \neg(\alpha \vee \beta)$

4.2 [Hilbert's axioms] Prove the following.

1. $\mathrm{T} \vdash \big(\alpha \Rightarrow (\beta \Rightarrow \alpha)\big)$
2. $\mathrm{T} \vdash \big(\neg\beta \Rightarrow \neg\alpha\big) \Rightarrow \big(\alpha \Rightarrow \beta\big)$
3. $\mathrm{T} \vdash \big(\alpha \Rightarrow (\beta \Rightarrow \gamma)\big) \Rightarrow \big((\alpha \Rightarrow \beta) \Rightarrow (\alpha \Rightarrow \gamma)\big)$

4.3 Show that the following are equivalent

1. $\alpha_1, \alpha_2, \ldots, \alpha_n \vdash \beta$
2. $\mathrm{T} \vdash (\alpha_1 \wedge \alpha_2 \wedge \ldots \alpha_n) \Rightarrow \beta$
3. $\mathrm{T} \vdash \Big(\alpha_1 \Rightarrow \big(\alpha_2 \Rightarrow (\ldots(\alpha_n \Rightarrow \beta))\big)\Big)$

4.4 Prove the following.

1. $\neg\gamma_1 \vee \gamma_2 \dashv\vdash \gamma_1 \Rightarrow \gamma_2$
2. $\gamma_1 \wedge \neg\gamma_2 \dashv\vdash \neg(\gamma_1 \Rightarrow \gamma_2)$
3. $\neg\alpha \vdash \alpha \Rightarrow \beta$
4. $\alpha \Rightarrow \beta, \beta \Rightarrow \alpha \vdash \alpha \Rightarrow \gamma$

4.5 Show that the following are theorems

1. $\alpha \vee \neg\alpha$
2. $\alpha \Rightarrow \alpha$
3. $\alpha \Rightarrow \neg\neg\alpha$
4. $(\alpha \Rightarrow \beta) \Rightarrow (\neg\beta \Rightarrow \neg\alpha)$
5. $(\neg\alpha \Rightarrow \alpha) \Rightarrow \alpha$
6. $(\neg\alpha \Rightarrow (\alpha \Rightarrow \beta))$
7. $\neg\neg\alpha \Rightarrow \alpha$
8. $(\alpha \Rightarrow \beta) \Rightarrow ((\alpha \Rightarrow \neg\beta) \Rightarrow \neg\beta))$

4.6 Let Ψ be a formula over only the proposition p. Assume that $p \vdash \psi$ and $\neg p \vdash \psi$. Show that, $\mathrm{T} \vdash \psi$.

4.7 Let us introduce a new connective xor: $\alpha \oplus \beta$ which should abbreviate $(\neg\alpha \wedge \beta) \vee (\alpha \wedge \neg\beta)$. Design introduction and elimination rules for xor.

4.8 Let $P = \{p_1, \ldots, p_n\}$ be a set of propositions. For a valuation v over P, we define $\Gamma_v = \{p_i \mid v(p_i) = \mathrm{T}\} \cup \{\neg p_i \mid v(p_i) = \mathrm{F}\}$. Consider a formula α such that $\Gamma_v \vdash \alpha$ for all valuation v. Show that $\mathrm{T} \vdash \alpha$.

4.9 Let v be a valuation. Then the following holds for all formulas α.

$$\beta_v \not\models \alpha \text{ iff } \beta_v \models \neg\alpha$$

4.10 [Strong soundness theorem] If $\Gamma \vdash \alpha$ then $\Gamma \models \alpha$.

4.11 If Γ is satisfiable then Γ is consistent.

4.12 Consider a new natural deduction set of rules you have created. Let Γ be a set of propositional formulas (the set need not be finite). Let us also define by $\Gamma \vdash^* \alpha$ to be a proof of α from Γ using your natural deduction rules. Let us also say that Γ is consistent, if you cannot prove α or $\neg\alpha$ using your natural deduction rules for any α from Γ. Prove that the following two statements are equivalent.

1. If $\Gamma \vdash^* \alpha$ then $\Gamma \models \alpha$.
2. If Γ is satisfiable then Γ is consistent.

4.13 Design introduction and elimination rules for Nand operator. Consider the logic which uses only negation and Nand operator as logical symbols. Prove the completeness theorem using the introduction and elimination rules for Nand and negation. Prove also the soundness theorem.

4.14 Prove the deduction theorem for a set (need not be finite) Γ of propositional formulas : $\Gamma \vdash (\alpha \Longrightarrow \beta)$ iff $\Gamma \cup \{\alpha\} \vdash \beta$.

4.15 Prove or disprove the following statements.

1. If $\Gamma_1 \subseteq \Gamma_2$ and Γ_1 is consistent then Γ_2 is consistent.
2. If $\Gamma_1 \subseteq \Gamma_2$ and Γ_2 is consistent, then Γ_1 is consistent.

CHAPTER 5

RANDOMIZED ALGORITHMS*

In this section we will look at randomized algorithms for propositional formulas. In the first subsection, we give a polynomial time algorithm for checking satisfiability for 2-CNF formulas. Then we give an exponential time algorithm checking satisfiability for 3-CNF formulas. This algorithm will be better than the trivial $O(2^n)$ algorithm of going through all the assignments to the n propositions. You will observe that both the algorithms are easy to describe. The difficult part is proving that the algorithm answers correctly with "high" probability.

5.1 CNF formulas

5.1.1 2-CNF

The algorithm is given in Algorithm 10. In the algorithm the number of times the loop needs to be iterated (i.e. m) will be fixed later depending on the confidence in the algorithm the user requires.

The following claim is an easy observation about the algorithm.

Algorithm 10 Randomized algorithm for 2CNFSAT

Input A 2-CNF formula α
Output: YES if α is satisfiable, NO otherwise.

```
1: function 2CNFSAT(α)
2:     Start with an arbitrary truth assignment to α.
3:     Let m = 2n^2 where n is the number of propositions in α
4:     for (m steps) do
5:         if (assignment makes α true) then return YES
6:         Choose a clause not satisfiable.
7:         Choose uniformly at random one of the propositions in the clause and
   change its assignment.
8:     end for
9:     return NO
10: end function
```

Lemma 5.1 *If the formula α is unsatisfiable then Algorithm 10 returns UNSAT. Contra positively, if the algorithm returns SAT, then the formula is satisfiable.*

Due to the above lemma, the important question we need to answer is, if the formula is satisfiable, how often will the algorithm return UNSAT. That is, what is the probability that the algorithm fails. So, let us assume that the formula is satisfiable and try to answer how long will the algorithm take to return SAT. This will help us in deciding what is a good value for m.

We now try to estimate the expected running time of the algorithm, assuming the formula is satisfiable and the loop runs for ever (i.e $m = \infty$). Let S be a satisfying assignment for α. We will try to find the expected running time for finding S. Note that, there may be other satisfying assignments and the algorithm might find them before it finds S. Therefore, the expected running time we find is a worst case estimate. Consider the i^{th} iteration of the loop. We define A_i and X_i as follows

$$A_i = \text{ the assignment at the beginning of the } i^{th} \text{ iteration of the loop}$$

$$X_i = \text{ the number of variables whose assignments in } A_i \text{ differ from that of } S$$

We can try to understand some properties of X_i. Note that if $X_i = n$, then all assignments to variables in A_i differ from S. The algorithm therefore will find a clause which is not satisfiable. In that clause, assignments to both the propositions are wrong and hence no matter which proposition we pick and change the assignment we get that $X_{i+1} = n - 1$.

$$\texttt{Prob}\left[X_{i+1} = n - 1 \mid X_i = n\right] = 1$$

Let us now move on to the general case when $X_i = k < n$. We are interested in identifying the probability of $X_{i+1} = k - 1$. Let us analyse our algorithm. We have k assignments differing from S and our algorithm picks a clause which is not satisfiable. Atleast one of the proposition in this clause is assigned a truth value in A_i

which is different from that in S (note that, it could happen that both the propositions are assigned differently). Our algorithm picks one of the two proposition with equal probability and changes its assignment. Therefore, we pick a proposition whose value is different with probability greater than or equal to $\frac{1}{2}$. If both the propositional values are different we pick with probability 1. Otherwise we pick with probability $\frac{1}{2}$. Therefore

$$\texttt{Prob}\left[X_{i+1} = k-1 \mid X_i = k\right] \geq \frac{1}{2}$$

A similar analysis also gives us

$$\texttt{Prob}\left[X_{i+1} = k+1 \mid X_i = k\right] \leq \frac{1}{2}$$

Our current understanding is captured by the following set of equations and in Figure 5.1.

$$\begin{aligned} \texttt{Prob}\left[X_{i+1} = n-1 \mid X_i = n\right] &= 1 \\ \texttt{Prob}\left[X_{i+1} = k-1 \mid X_i = k\right] &\geq \frac{1}{2}, \forall k, \text{ where } 0 < k < n \\ \texttt{Prob}\left[X_{i+1} = k+1 \mid X_i = k\right] &\leq \frac{1}{2}, \forall k, \text{ where } 0 < k < n \end{aligned} \tag{5.1}$$

Let us assume that when we enter the first loop, we have an assignment such that $X_1 = k$. We are interested in finding out how many steps m are required such that $X_m = 0$. Note that, at any iteration of the loop, we can go right one step in the figure (towards n) with probability greater than or equal to half and we can move left (towards 0) with probability, less than or equal to half.

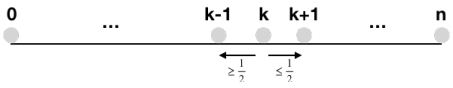


Figure 5.1 At each loop iteration, the algorithm walks one step towards the left or right with probability $\geq \frac{1}{2}$ or $\leq \frac{1}{2}$ respectively.

The walk given by Equation 5.1 is difficult to analyse and therefore we analyse a "pessimistic" version of the above probability distribution. The equations in this version are approximated by a "Markov chain" as follows (see also Figure 5.2).

$$\begin{aligned} \texttt{Prob}\left[X_{i+1} = n-1 \mid X_i = n\right] &= 1 \\ \texttt{Prob}\left[X_{i+1} = k-1 \mid X_i = k\right] &= \frac{1}{2}, \forall k, \text{ where } 0 < k < n \\ \texttt{Prob}\left[X_{i+1} = k+1 \mid X_i = k\right] &= \frac{1}{2}, \forall k, \text{ where } 0 < k < n \end{aligned} \tag{5.2}$$

Note that, in the former setting, the probability of going left at any point was greater or equal to in the latter setting. Therefore, the probability of reaching 0 in m number of steps is greater in the former than in the latter. Therefore, the expected number of steps to reach 0 from k using these set of equations is greater than what we had before. We will give an upperbound for these sets of equations.

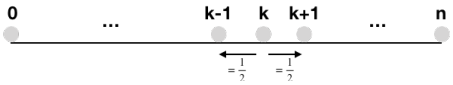


Figure 5.2 Markov Chain approximation: At each loop iteration, the algorithm walks one step towards the left or right with probability exactly $\frac{1}{2}$. This is a worst case scenario for our algorithm.

Let k be such that $0 \leq k \leq n$. We will denote by Z_k, the random variable representing the number of steps from k to 0. We are interested in the expected value of Z_n (denoted by $E[Z_n]$). The Equations 5.2 gives us the following

$$\begin{aligned} E[Z_0] &= 0 \\ E[Z_n] &= 1 + E[Z_{n-1}] \\ \forall\, 0 < k < n,\ E[Z_k] &= \frac{1}{2}(1 + E[Z_{k-1}]) + \frac{1}{2}(1 + E[Z_{k+1}])] \\ &= 1 + \frac{1}{2}(E[Z_{k-1}] + E[Z_{k+1}]) \end{aligned} \tag{5.3}$$

This contains $n + 1$ equations on $n + 1$ variables. The following claim holds for equations 5.3.

Lemma 5.2 *For all k, where* $0 \leq k < n$ *we have* $E[Z_k] = 2nk - k^2$

Proof: The proof is by induction on k. It is easy to observe that the claim holds for the base case $k = 0$. Let us assume it true for some k and show that it holds for $k + 1$. We know the following

$$E[Z_k] = 1 + \frac{1}{2}(E[Z_{k-1}] + E[Z_{k+1}])$$

Therefore, the lemma holds due to the following analysis

$$\begin{aligned} E[Z_{k+1}] &= 2\big(E[Z_k] - 1\big) - E[Z_{k-1}] \\ &= 2(2nk - k^2 - 1) - \big(2n(k-1) - (k-1)^2\big) \\ &= 2(2nk - k^2 - 1) - \big(2nk - 2n - (k^2 - 2k + 1)\big) \\ &= 2nk - k^2 - 1 + 2n - 2k \\ &= 2n(k+1) - (k+1)^2 \end{aligned}$$

■

From this, we get that

$$E[Z_n] = 1 + 2n(n-1) - (n-1)^2 = n^2$$

In other words, the expected number of steps required from any position k to 0 is less than or equal to n^2. This proves the following.

Lemma 5.3 *If a 2-CNF formula is satisfiable, then the algorithm 10 outputs SAT in an expected running time of at most* n^2.

With the above lemma, we can derive a “good" value for m. We show that, if $m = 2n^2t$, then the algorithm answers correctly with a probability greater than or equal to $(1 - \frac{1}{2^t})$.

Theorem 5.4 *Let $m = 2n^2t$. Then Algorithm 10. fails with probability less than or equal to $\frac{1}{2^t}$.*

Proof: We know that if the formula is not satisfiable, the algorithm does not fail. So, let us assume that the formula is satisfiable. Let Z be the random variable representing the number of steps taken to output SAT. Applying Markov's inequality to the above lemma gives us

$$\texttt{Prob}\,[Z \geq 2n^2] \leq \frac{1}{2}$$

Let us consider our algorithm as running t loops with each loop running $2n^2$ times. Then, an inside loop fails with probability less than or equal to half. Hence, the probability of the algorithm failing t times is given by the union bound as

$$\texttt{Prob}\,[\text{ algorithm fails }] \leq \frac{1}{2^t}$$

This ends the proof. ∎

5.1.2 3-CNF

Our algorithm will be a modification of the 2-CNF algorithm. What could go wrong if we applied the same algorithm for the 3-CNF case. The Markov chain for a 3-CNF formula is given in Figure 5.3. Exercise **??** shows that the expected running time for this algorithm is $O(2^n)$. This is not good enough, since even going through all possible solutions takes only this much time.

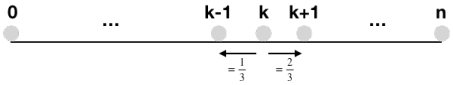


Figure 5.3 Markov Chain approximation: At each loop iteration, the algorithm walks one step towards the left or right with probability $\frac{1}{3}$, and $\frac{2}{3}$ respectively. This is a worst case scenario for our algorithm.

We modify our algorithm as follows.

As in the 2-CNF algorithm, if the formula is unsatisfiable, the algorithm will return UNSAT. Let us now calculate the expected running time of the algorithm if $m = \infty$ and assuming the formula is satisfiable. Like in the previous analysis, let S be the satisfying assignment. Let us now consider the run of the outer loop. We start from an arbitrary assignment for α. For the i^{th} iteration of the inner loop we denote by A_i the assignment at the beginning of the inner loop. We want to bound the probability that the $3n$ steps of the inner loop does not identify the satisfying assignment. Let us assume that after the initial arbitrary assignment, k many propositions are wrongly assigned. Let us denote by q_k the probability that we reach a

Algorithm 11 Randomized algorithm for 3CNFSAT

Input A 3-CNF formula α
Output: YES if α is satisfiable, NO otherwise.

```
1:  function 3CNFSAT(α)
2:      Let m = 2√n/c (4/3)^n, where n is the number of propositions in α
3:      for (m steps) do
4:          Start with an arbitrary truth assignment to α.
5:          for (3n steps) do
6:              if (assignment satisfies α) then return YES
7:              Choose a clause not satisfiable.
8:              Choose uniformly at random one of the propositions in the clause
    and change its assignment.
9:          end for
10:     end for
11:     return NO
12: end function
```

satisfying assignment within $3n$ steps of the inner loop. That is, the probability of reaching 0 from k by doing a random walk on the Markov Chain given in Figure 5.3. Note that, in the special case of $k = 0$, we have $q_0 = 1$. For a general $k > 0$, there is no bound on the number of left or right moves required to reach 0. Let us consider, a special case when the number of right moves is k and the number of left moves is $2k$. Clearly q_k is greater than the probability of this happening. Thus

$$\begin{aligned} q_k &\geq \frac{3k!}{k!} 2k! \left(\frac{1}{3}\right)^{2k} \left(\frac{2}{3}\right)^{k} \\ &\geq \frac{c_1 \sqrt{3k} (\frac{3k}{e})^{3k}}{c_2 \sqrt{k} (\frac{k}{e})^{k} c_2 \sqrt{2k} (\frac{2k}{e})^{2k}} \left(\frac{1}{3}\right)^{2k} \left(\frac{2}{3}\right)^{k} \\ &\geq c \frac{1}{\sqrt{k}} \frac{1}{2^k} \text{ , for a constant } c \end{aligned}$$

In the second step, we used Stirling's approximation. It says there are constants c_1 and c_2 such that for all $n > 0$, $c_1 \sqrt{n} \left(\frac{n}{e}\right)^n \leq n! \leq c_2 \sqrt{n} \left(\frac{n}{e}\right)^n$.

We now have a bound on q_k. Let us now calculate the probability q that we find the satisfying assignment given that we start from a random assignment. Then,

$$
\begin{aligned}
q &= \sum_{k=0}^{n} \texttt{Prob}\,[\text{assignment to exactly } k \text{ propositions are different from that of } S\,] \times q_k \\
&\geq \frac{1}{2^n} + \sum_{k=1}^{n} \binom{n}{k} \frac{1}{2^n} \frac{c}{\sqrt{k}} \frac{1}{2^k} \\
&\geq \frac{1}{2^n} \frac{c}{\sqrt{n}} + \sum_{k=1}^{n} \binom{n}{k} \frac{1}{2^n} \frac{c}{\sqrt{n}} \frac{1}{2^k} \\
&= \frac{c}{\sqrt{n}2^n} \sum_{k=0}^{n} \binom{n}{k} \frac{1}{2^k} \\
&= \frac{c}{\sqrt{n}2^n}(1 + \frac{1}{2})^n \text{ (using the expansion for } (1 + \frac{1}{2})^n) \\
&= \frac{c}{\sqrt{n}} \left(\frac{3}{4}\right)^n
\end{aligned}
$$

Thus the probability of success starting from an arbitrary assignment and walking $3n$ steps is $\geq \frac{c}{\sqrt{n}}\left(\frac{3}{4}\right)^n$. Thus the expected running time of the outer loop for success is given by

$$O(\sqrt{n}\left(\frac{4}{3}\right)^n)$$

Let a be denoted by this number. Thus the total number of steps of the algorithm is $a \times 3n$. We will now show that taking $m = 2at$ will ensure that the probability of failure of our algorithm is less than or equal to $\frac{1}{2^t}$.

Theorem 5.5 *Let $a = \frac{\sqrt{n}}{c}\left(\frac{4}{3}\right)^n)$ and $m = 2at$. Then the running time of Algorithm 11. is $O(n^{\frac{3}{2}}\left(\frac{4}{3}\right)^n)$ and the probability of failure is less than or equal to $\frac{1}{2^t}$.*

Proof: The running time of the algorithm is $3n \times \frac{\sqrt{n}}{c}\left(\frac{4}{3}\right)^n) = O(n^{\frac{3}{2}}\left(\frac{4}{3}\right)^n)$. The algorithm will fail only if the formula is satisfiable. Let Z be the random variable denoting the number of outer loops required for finding the satisfying assignments. Using Markov's inequality we can show that

$$\texttt{Prob}\,[Z \geq 2a] \leq \frac{1}{2}$$

Let us again consider that we are running the algorithm t times and each time we are running the outer loop for $2a$ times. Thus the probability of failure for all the t times is given by the union bound by

$$\texttt{Prob}\,[\text{algorithm fails }] \leq \frac{1}{2^t}$$

■

5.2 Summary and Acknowledgements

The chapter follows the presentation in Mitzenmacher and Upfal [MU05].

PART II

FIRST ORDER LOGIC

CHAPTER 6

SYNTAX AND SEMANTICS

6.1 Need for a richer logic

We had seen that propositional logic can be used for reasoning as in the following example.

EXAMPLE 6.1

(Reasoning in propositional logic:)
Consider the following two statements.

Person A: If there is fire, then there is smoke. There is no smoke on the hill. Therefore, there is no fire.

Person B: If there is fire, then there is smoke. There is smoke on the hill. Therefore, there is fire.

Clearly, Person A's statement is logically correct while that of Person B is not. We can check this by encoding these two reasoning in propositional logic. See Exercise**??**. We observe that the propositional logic formula corresponding to Person A is valid while the other one is not.

Consider the reasoning given in Table 6.1. Which one of the reasoning is correct?

Proof A	Proof B
1. All mathematicians are intelligent.	1. There is a mathematician who is intelligent.
2. Ramanujan was a mathematician.	2. Ramanujan was a mathematician.
3. Therefore, Ramanujan was intelligent.	3. Therefore, Ramanujan was intelligent.

Table 6.1 Two proofs - One is correct, one wrong.

We can see that Proof A is correct, whereas Proof B is not. Natural deduction of propositional logic will not help us to reach a correct conclusion in this case. We need a stronger, richer logic for these kind of reasoning. *First order logic* (also called *predicate* logic) helps us in this regard.

6.2 Introduction to first order logic

First order logic is a *family* of logics. First order logic might be represented in short as FO, or FOL or 1^{st} order logic. There are things which are common to all the logics and things which are different.

common among first order logic: Like in propositional logic, we need to have a set of variables. Unlike in propositional logic case, the variables in first order logic are not boolean variables. We will denote the variables by $x, y, z, x_1, x_2, \ldots, y_1, y_2, \ldots$. The next thing in common are the logical symbols. They are of three categories.

1. propositional logic symbols: This consists of $\{\wedge, \vee, \neg, \Longrightarrow\}$
2. quantifiers: This consists of $\forall$ (called *for all*) and $\exists$ (called *there exists*).
3. equality: The symbol = denotes equality. It will be used to denote an object is equal to another object. For example: The prime minister of India = Narendra Modi.

logic specific symbols: The logic specific symbols are of three types.

1. constant symbols
2. function symbols and
3. relational symbols

Before we go into a formal introduction of first order logic, let us look at a few examples from the family of first order logic.

6.2.1 Logic for number theory

The logic specific symbols are

1. constant symbols: zero and one denoted by 0 and 1
2. function symbols: addition and multiplication denoted by + and ×.
3. relation symbols: less than or equal to relation denoted by ≤.

Let us look at some example properties which can be expressed in first order logic. The following formula says that "x is a prime number".

$$prime(x) ::= \forall y \, \forall z \Big((x = y \times z) \implies \big((y = x) \vee (z = x) \big) \Big)$$

The formula says that for all numbers y and z such that their product equals x means that either y or z is equal to x. This is a property which is satisfied only by primes. In short formula $prime(x)$ becomes "true" only when x is a prime number. Note an important fact here is our assumption on what are the values y and z can take. We have assumed here that all variables will only take natural numbers. We will later see that the *universe* for this logic will be natural numbers.

Let us look at a few other example properties which we can write: x is an even number can be written as

$$even(x) ::= \exists y \, (x = y + y)$$

We can also write mathematical theorems in first order logic. The following formula says "there are infinite number of primes".

$$\forall x \, \Big(primes(x) \implies \big(\exists y \, (primes(y) \wedge (x \leq y) \wedge \neg (x = y)) \big) \Big)$$

The formula can be read as, for all primes x, there exists a prime number y which is greater than equal to x and not equal to x (in short greater than x).

6.2.2 Logic for Graph theory

6.2.3 Logic for set theory

In the logic for set theory, there is only one specific symbol. The relation symbol called member of (denoted by $\in$). The following formula says that "x is an empty set".

$$empty(x) ::= \forall y \, \big(\neg (y \in x) \big)$$

Similarly the following formula says that "there is only one empty set".

$$\forall x \forall y \, \Big((empty(x) \wedge empty(y)) \implies (x = y) \Big)$$

6.2.4 Logic for IIT Goa

In the logic for IIT Goa, we have the constants: Sreejith, Raj, Neha. The *unary* relations (relations with arity 1) are $student$ and $faculty$. We denote by $student(x)$ (respectively $faculty(x)$) to mean that x is a student (resp. faculty). We also have a binary relation $teaches(x, y)$ which means x teaches y. With this logic, we can state that Sreejith and Neha are teachers whereas Raj is a student. Other complicated sentences can also be formed. For example, the following statement says that there is a faculty who does not teach anyone.

$$\exists x \forall y \Big(faculty(x) \wedge \neg (teaches(x, y)) \Big)$$

6.2.5 Logic of empty symbols

In this logic, there are no constant, functions or relation symbols. Yet, we can talk about some properties. For example, we can say that the universe contains only one element.

$$\forall x \forall y \ (x = y)$$

We can also say that there are atleast 2 elements in the universe.

$$\exists x \exists y \exists z \ \Big(\neg (x = y) \wedge \neg (x = z) \wedge \neg (y = z) \Big)$$

Thus, we can talk about “atmost many elements" or “atleast many elements" in the universe.

6.3 Syntax

Like in propositional logic, we define the syntax of first order logic. The syntax gives the rules for defining “meaningful" sentences in the logic. We have mentioned earlier that first order logic is a family of logics. Therefore, to fix the syntax we need to specify the constants, functions and relation symbols. This is called the *language* of the logic. Once the language is fixed, we can talk about the syntax of that particular logic. Let us fix the language as having

1. constant symbols: We will denote $a_1, a_2, \ldots$ as the set of constant symbols.
2. function symbols: We will denote by $f_1, f_2, \ldots$ as the set of function symbols. Note that when the functions symbols are mentioned, we also need to mention the *arity* of the functions. The arity of a function is defined as the dimensionality of its domain. For example, the addition and multiplication functions have arity 2 whereas a square function has arity 1.
3. relation symbols: We will denote by $R_1, R_2, \ldots$ as the set of relation symbols. Again, we need to mention the arity of the relation symbols. For example, the arity of $\leq$ is 2, whereas the arity of $student(x)$ is one. Relations of arity 1 are called *unary relations* whereas those with arity 2 are called *binary relations* .

Once the symbols are defined, we can give the syntax of the logic.

6.3.1 Well formed formulas

6.3.1.1 Step 1 (Define the collection of words that denote objects) : Intuitively, terms represents words which denotes the objects of our universe.

Definition 6.1 (terms) *The set of* terms *is inductively defined as the smallest set X which*

1. *contains all constant symbols and variables and*
2. *is closed under all function symbols f_i. That is, if $t_1, \dots, t_n$ are terms, then $f_i(t_1, \dots, t_n)$ is a term. Here f_i is a function of arity n.*

$terms \ ::= \mathcal{T}(\ all\ words\ over\ symbols\ , Constants \bigcup Variables, \{f \mid f\ is\ a\ function\})$

In Baukus normal form, the grammar for terms can be written as follows.

$$t ::= c \mid x \mid f(t, t, \dots, t)$$

Let us look at the examples.

Case: Logic of number theory: Here are a few examples of terms in the logic of number theory: $0, 1, x, y, z$ etc. Here is another example got by applying the addition and multiplication functions: $(1+1)\times(x\times y\times y)+(1+1+1)\times(y\times y)+(x\times y)$.

As you could observe, the terms represent the polynomials. The following claim is easy to prove.

Claim 6.1 *Every term represents a polynomial. Moreover, for every polynomial, there is a term equivalent to it.*

Note that the following formulas are not terms: $x = y$, $\exists x\ (x \times x)$ or $x \leq y$.

Case: Logic of set theory: In the logic of set theory, the only terms are the variables. This is because, there are no constants or function symbols.

Case: Logic of IIT Goa: In the logic for IIT Goa, we have that there are no functions symbols. Therefore, the only terms are the variables and the constants, Sreejith, Neha and Raj.

Case: Logic of empty theory: The only terms are the variables since there are no constants or function symbols.

6.3.1.2 Step 2 (well formed formulas): To define the well formed formulas (wff) of first order logic, we need to first define *atomic formulas* . They are the smallest meaningful sentence we can write in the logic.

Definition 6.2 (atomic formulas) *In Baukus normal form, the grammar for atomic formulas can be written as follows.*

$$atoms ::= (t = t) \mid R(t, t, \dots, t)$$

In the logic for number theory, examples of atomic formulas include $0 \leq 1$, $((1+1) \times x + (y+y)) \leq (1+1)$ etc. In the logic for set theory $y \in x$ is an atomic formula. The formulas $student(x), teaches(x, y)$ are atomic formulas in the logic for IIT Goa. On the other hand the formula $(y \in x) \wedge (y \in z)$ is not an atomic formula since it uses conjunction symbol.

The wffs can now be defined easily.

Definition 6.3 (wffs) *The well formed formulas are the smallest set which*

1. *contains the atomic formulas and*
2. *is closed under the logical operations* $\{\wedge, \vee, \neg, \Longrightarrow, \forall, \exists\}$.

In our notation,

$$wff \ ::= \mathcal{T}(\ all\ words\ over\ symbols\ , atoms, \{\wedge, \vee, \neg, \Longrightarrow, \forall, \exists\})$$

In other words, all atomic formulas are wffs and if α *and* β *are wffs, then* $(\neg\alpha)$, $(\alpha vee\beta), (\alpha \wedge \beta), (\alpha \implies \beta), \exists x\ (\alpha), \forall x\ (\alpha)$ *are all wffs. In Baukus normal form, the grammar for wffs can be written as follows.*

$$\phi \ ::= atoms \mid (\neg\phi) \mid (\phi \vee \phi) \mid (\phi \wedge \phi) \mid (\phi \implies \phi) \mid \exists x(\phi) \mid \forall x(\phi)$$

Informally, the terms are like functions (which output an element from the domain), whereas wffs are like predicates (which have a true or false value).

6.3.2 Free and bound variables

An important notion in well formed formulas is *free variables* . For a formula α, we denote by $\mathrm{FV}(\alpha)$ the set of all free variables in α. The free variables of a formula are inductively defined as

$$\mathrm{FV}(\alpha) = \begin{cases} \text{all variables in } \alpha, \text{ if } \alpha \text{ is an atomic formula} \\ \mathrm{FV}(\beta_1) \cup F(\beta_2), \text{ if } \alpha \text{ is of type } (\beta_1 \vee \beta_2), (\beta_1 \wedge \beta_2), (\beta_1 \implies \beta_2) \\ \mathrm{FV}(\beta), \text{ if } \alpha \ ::= (\neg\beta) \\ \mathrm{FV}(\beta) \backslash \{x\}, \text{ if } \alpha \ ::= \forall x\ (\beta) \text{ or } \alpha \ ::= \exists x\ (\beta) \end{cases}$$

For example, in the following formula x is a free variable whereas y is not.

$$\forall y\ (x \leq y)$$

On the other hand, the *bound variables* are those variables which are quantified. The bound variables of a wff α will be denoted by $\mathrm{BV}(\alpha)$. In the above example y is bound. Consider the following example. The variable x appears free as well as bound.

$$(x + 1 = z) \wedge \exists x\ (x \leq y)$$

A wff where all variables are bound is called a *sentence* . They play a major role in first order logic.

Definition 6.4 *Substitutions*

6.4 Semantics

Semantics of first order logic over a language L consists of rules that assign true or false values to all well formed first order formulas. Before we go into the formal definition let us look at an example. Consider the following formula.

$$\begin{aligned}
&\forall x \Big(R(x) \implies \big(\forall y \, (E(x,y) \implies B(y))\big) \Big) \wedge \\
&\forall x \Big(B(x) \implies \big(\forall y \, (E(x,y) \implies R(y))\big) \Big) \wedge \\
&\forall x \, \big(R(x) \vee B(x) \big) \wedge \\
&\forall x \, \big(R(x) \iff \neg B(x) \big) \wedge \\
&\forall x \, \forall y \, \big(E(x,y) \implies E(y,x) \big)
\end{aligned}$$

We need to give a meaning to this formula. In propositional logic, a wff gets a meaning when an assignment is given. We are interested in finding out what is the "assignment" for a first order logic formula. This is called as a *structure* (in some books it is called as a*model*).

Definition 6.5 (structure) *A structure over the language L of function symbols, relation symbols and constant symbols consists of the following.*

1. *A set (usually called* universe *) U.*
2. *An interpretation $\mathcal{I}$ which*
   (a) *assigns a function $f^{\mathcal{I}} : U^k \to U$ for every function symbol f of arity k in L.*
   (b) *and assigns a relation $f^{\mathcal{I}} \subseteq U^k$ for every relation symbol f of arity k in L.*
   (c) *and assigns a constant $c^{\mathcal{I}} \in U$ for every constant c in L.*

For example, for an $L = \{R, F, G, c, d\}$ where R is a relation of arity 2, F and G are relations of arity 2 and c, d are constant, a structure can $(\mathbb{N}, \leq, +, \times, 0, 1)$. Consider another example.

EXAMPLE 6.2

Let $L = \{E\}$ where E is a relation E of arity 2.

$$\alpha ::= \forall x \, \exists y \, \exists z \Big(\big(E(x,y) \wedge E(x,z) \wedge y \neq z \big) \wedge \forall t \big(E(x,t) \implies ((t = y) \vee (t = z)) \big) \Big)$$

One way to build a structure for L is to think of E as an edge relation in graph. Then, the formula above is talking about the property that "every vertex has out degree exactly 2". Therefore an example structure which does not satisfies this formula is: $(U = \{1,2,3\}, E^{\mathcal{I}} = \{(1,2),(2,3),(3,1)\})$. A structure which does satisfy this property is $(U = \{1,2\}, E^{\mathcal{I}} = \{(1,2),(1,1),(2,1),(2,2)\})$.

Even though we have not formally defined the semantics, you already have an idea of how to check if a formula satisfies a structure. We are now in a position to give the semantics of FO. To define semantics over wffs which are not sentences (that is those wffs which have free variables) we also need to assign every free variable to some element in the universe. An *assignment* , s is therefore a function which maps every free variable to an element in the universe. That is,

$$s : \{\text{Free variable}\} \longrightarrow U$$

The question we now try to answer is, when can we say that (M, s) satisfies α where α is a wff over language L and M is a structure over L and s assigns values to the free variables in α. It is first easier to write this in terms of a "recursive" algorithm (See Algorithm 13 which uses 12 for evaluating terms in the formula).

Algorithm 12 Evaluate a term of first order logic.

Input A term t over language L, a structure M over L and an assignment s which maps the free variables to the universe.

Output: an element $a \in U$ such that the term on assignment s evaluates to a.

```
1: function EVALUATE(t, M, s)
2:     if (t ::= x) then return s(x)
3:     if (t ::= c) then return c^I
4:     if (t ::= f(t_1, ..., t_n)) then return f^I(EVALUATE(t_1), ..., EVALUATE(t_n))
5: end function
```

This algorithmic way to define semantics has a problem because the universe in M can be infinite. We therefore need to give a "non-deterministic" definition.

Definition 6.6 (semantics for first order logic) *Let α be a wff over a language L and M, s be a structure over L and s maps all variables to assignments. We define (M, s) satisfies α inductively.*

1. *(M, s) satisfies $(t_1(x_1, \ldots, x_n) = t_2(x_1, \ldots, x_n))$ if evaluating t_1 and t_2 using s satisfies α.*
2. *(M, s) satisfies $R(x_1, \ldots, x_n)$, if $R^{\mathcal{I}}(s(x_1), \ldots, s(x_n)) = T$.*
3. *(M, s) satisfies $\psi_1 \vee \psi_2$, if (M, s) satisfies ψ_1 or (M, s) satisfies ψ_2.*
4. *(M, s) satisfies $\neg\psi$, if (M, s) does not satisfy ψ.*
5. *(M, s) satisfies $\exists x\ \psi$, if there exists an $a \in U$ such that $(M, s \cup \{x = a\})$ satisfies ψ.*
6. *(M, s) satisfies $\forall x\ \psi$, if for all $a \in U$, $(M, s \cup \{x = a\})$ satisfies ψ.*

Algorithm 13 Semantics of First order logic.

Input A FO wff α over language L, a structure M over L and an assignment s which maps the free variables to the universe.

Output: T if α is satisfiable, F otherwise.

```
1:  function checkFO-SAT(α, M, s)
2:      if (α ::= (t1(x1, ..., xn) = t2(x1, ..., xn))) then
3:          if (Evaluate(t1, M, s) = Evaluate(t2, M, s)) then return T
4:      end if
5:      if (α ::= R(x1, ..., xn)) then
6:          if (R^I(s(x1), ..., s(xn)) = T) then return T
7:      end if
8:      if (α ::= (ψ1 ∨ ψ2)) then
9:          return (checkFO-SAT(ψ1, M, s) ∨ checkFO-SAT(ψ2, M, s))
10:     end if
11:     if (α ::= ¬ψ1) then return ¬ checkFO-SAT(ψ1, M, s)
12:     if (α ::= ∃x ψ) then
13:         for (a ∈ U) do
14:             if (checkFO-SAT(ψ, M, s ∪ {x = a}) = T) then return T
15:         end for
16:         return F
17:     end if
18:     if (α ::= ∀x ψ) then
19:         for (a ∈ U) do
20:             if (checkFO-SAT(ψ, M, s ∪ {x = a}) = F) then return F
21:         end for
22:         return T
23:     end if
24: end function
```

6.5 Satisfiable and Valid formulas

We say that a first order wff α over language L is satisfiable, if there exists a structure M over L and an assignment $s : \{ \text{ Free variables } \} \longrightarrow U$ such that (M, s) satisfies α. Similarly, we say that a first order wff α over language L is valid, if for all structures M over L and for all assignments $s : \{ \text{ Free variables } \} \longrightarrow U$ we have that (M, s) satisfies α.

Let us look at some example formulas.

Claim 6.2 *The following formulas (called as Hilbert's axioms) are valid.*

1. $(\forall x\ Q(x)) \Longrightarrow Q(y)$
2. $\Big(\forall x\ (P(x) \Longrightarrow Q(x))\Big) \Longrightarrow \forall x(P(x) \Longrightarrow Q(x))$
3. $\phi \Longrightarrow \forall x\ \phi$ *if* x *is not a free variable in* ϕ.

Proof: We give the proof for the first formula. Let $\alpha ::= \big(\forall x\ Q(x)) \implies Q(y)\big)$. Consider an arbitrary structure M and assignment s which satisfies α. Consider the following two cases

1. (M, s) does not satisfy $(\forall x\ Q(x))$: Then α is satisfied by (M, s) due to the semantics of implication (if the left hand side of the implication is false, then formula is true).
2. (M, s) satisfies $(\forall x\ Q(x))$: Then for all $a \in U$, where U is the universe in our structure M, we have that $Q^{\mathcal{I}}(a)$ is true. In particular $Q^{\mathcal{I}}(s(y))$ is true. Again, due to the semantics of implication, we have that (M, s) satisfies α.

The formula α is valid since the formula is satisfied by the structure in all cases.

A similar argument will show that the other two formulas are also valid. ∎

We will now define equivalent formulas.

Definition 6.7 (semantic equivalence) *Let ϕ_1 and ϕ_2 be wffs over a language L. We say that ϕ_1 is* semantically equivalent *(equivalent for short) if for all structures M and assignment s, we have that*

$$(M, s) \textit{ satisfies } \phi_1 \textit{ iff } (M, s) \textit{ satisfies } \phi_2$$

We denote by $\phi_1 \equiv \phi_2$ to mean that ϕ_1 and ϕ_2 are equivalent.

The following claim shows an equivalence.

Claim 6.3

$$\neg(\forall x\ \phi) \equiv \exists x\ \neg\phi$$

Proof: We will first show the forward direction of the proof. Consider an arbitrary (M, s) which satisfy the formula $\neg(\forall x\ \phi)$. Then

$$\begin{aligned}
&(M, s) \text{ satisfy } \neg(\forall x\ \phi)\\
&\iff (M, s) \text{ does not satisfy } \forall x\ \phi\\
&\iff (M, s \cup \{x = a\}) \text{ does not satisfy } \phi \text{ for some } a\\
&\iff (M, s \cup \{x = a\}) \text{ satisfy } \neg\phi)\\
&\iff (M, s) \text{ satisfy } \exists x\ \phi
\end{aligned}$$

We now show the other direction. Again, consider an arbitrary (M, s) which satisfy the formula $\exists x\ \neg\phi$. Then

$$\begin{aligned}
&(M, s) \text{ satisfy } \exists x\ \neg\phi\\
&\iff (M, s \cup \{x = a\}) \text{ satisfy } \neg\phi\\
&\iff (M, s \cup \{x = a\}) \text{ does not satisfy } \phi\\
&\iff (M, s) \text{ does not satisfy } \forall x\ \neg\phi\\
&\iff (M, s) \text{ satisfy } \neg\forall x\ \neg\phi
\end{aligned}$$

∎

Problems

6.1 Show that the following formulas are valid

1. $\exists x \left(P(x) \Longrightarrow \forall x\ P(x)\right)$
2. $\Big(\forall x \left(P(x) \Longrightarrow Q(x)\right)\Big) \Longrightarrow \Big(\forall x\big(P(x) \Longrightarrow Q(x)\big)\Big)$
3. $(\phi \Longrightarrow \forall x\ \phi)$, if x is not a free variable in ϕ.
4. $\big(\exists y\ \forall x\ R(x, y)\big) \Longrightarrow (\forall x \exists y\ R(x, y))$
5. $\big(\forall x\ P(x)\big) \Longrightarrow \big(\exists x\ P(x)\big)$

6.2 Show that the following formulas are not valid.

1. $\Big(\forall x\big(P(x) \Longrightarrow Q(x)\big)\Big) \Longrightarrow \Big(\forall x \left(P(x) \Longrightarrow Q(x)\right)\Big)$
2. $\big(\forall x\ \exists y\ R(x, y)\big) \Longrightarrow \big(\exists y \forall x\ R(x, y)\big)$
3. $\big(\exists x\ P(x)\big) \Longrightarrow \big(\forall x\ P(x)\big)$

6.3 Show the following equivalences

1. $\neg\exists x\ \phi \equiv \forall x\ \neg\phi$
2. $\forall x \forall y\ \phi \equiv \forall x \forall y\ \phi$
3. $\exists x \exists y\ \phi \equiv \exists y \exists x\ \phi$
4. $\forall x \phi \wedge \forall x \psi \equiv \forall x\ (\phi \wedge \psi)$
5. $\exists x \phi \vee \exists x \psi \equiv \exists x\ (\phi \vee \psi)$

6.4 Show that the following are not equivalent

1. $\forall x \phi \vee \forall x \psi \not\equiv \forall x\ (\phi \vee \psi)$
2. $\exists x \phi \wedge \exists x \psi \not\equiv \exists x\ (\phi \wedge \psi)$
3. $\forall x \exists y \phi \not\equiv \exists y \forall x \phi$

6.5 Let ϕ and ψ be wffs such that $y \notin \mathrm{FV}(\phi)$ and $x \notin \mathrm{FV}(\psi)$. Show the following

1. $\exists x \phi \wedge \exists y \psi \equiv \exists x\ \exists y\ (\phi \wedge \psi)$
2. $\exists x \phi \vee \exists y \psi \equiv \exists x\ \exists y\ (\phi \vee \psi)$
3. $\exists x \phi \wedge \forall y \psi \equiv \exists x\ \forall y\ (\phi \wedge \psi)$

4. $\exists x \phi \vee \forall y \psi \equiv \exists x \, \forall y \, (\phi \vee \psi)$
5. $\forall x \, \exists y \, (\phi \wedge \psi) \equiv \exists x \, \forall y \, (\phi \wedge \psi)$

6.6 Normal forms

6.7 Substitution

6.8 Semantic Entailment

First we define what it means to say a structure and assignment satisfies a set of wffs. Let Γ be a set of wffs and (M, s) be a structure and assignment pair. Then we say that (M, s) satisfy Γ if (M, s) satisfy ϕ for all ϕ in Γ. That is

$$(M, s) \text{ satisfy } \Gamma \quad \text{if} \quad \forall \phi \in \Gamma, \ (M, s) \text{ satisfy } \phi$$

For a set of wffs Γ and a wff ϕ, we denote by $\Gamma \models \phi$ to mean that for all structure M and assignment s such that (M, s) satisfy Γ, we have that (M, s) satisfy ϕ. That is,

$$\Gamma \models \phi \quad \text{if} \quad \text{for all } (M, s) \text{ pair, } \big((M, s) \text{ satisfy } \Gamma\big) \implies \big((M, s) \text{ satisfy } \phi\big)$$

6.9 Summary and Acknowledgements

The chapter is heavily influenced by the video lecture series of Dr. Shai Ben David [BD]. The lectures itself follow the book by Enderton [End72]. The chapter also follows the book by Huth and Ryan [HR04].

6.10 Chapter exercises

6.1 Let h stand for Holmes (Sherlock Holmes) and m stand for Moriarty. Give first order logic formulas to express the following:

1. Holmes can catch anyone whom Moriarty can catch.
2. If anyone can catch Moriarty, then Holmes can.
3. If everyone can catch Moriarty, then Holmes can.
4. No one can catch Holmes unless he can catch Moriarty.

5. Everyone can catch someone who cannot catch Moriarty.

6.2 Show that the following sentences are valid.

1. $\neg\alpha \Longrightarrow (\alpha \Longrightarrow (\alpha \Longrightarrow \beta))$
2. $(\alpha \vee \neg\alpha)$
3. $(\neg p \Longrightarrow \neg q) \Longrightarrow (q \Longrightarrow p)$
4. $\forall x(p(x) \Longrightarrow q(x)) \Longrightarrow (\exists x\ p(x) \Longrightarrow \exists x\ q(x))$
5. $\big(\forall x\ (P(x) \vee Q(x)) \wedge \exists x \neg Q(x) \wedge \forall x\ (R(x) \Longrightarrow \neg P(x))\big) \Longrightarrow \exists x \neg R(x)$

6.3 Prove or disprove the validity of the following formulas.

1. $\exists x\ (D(x) \Longrightarrow \forall y D(y))$
2. $(\forall x \exists y\ (P(x) \Longrightarrow Q(y))) \Longrightarrow (\exists y \forall x (P(x) \Longrightarrow Q(y)))$
3. $(\exists x p(x) \Longrightarrow \exists x q(x)) \Longrightarrow \forall x(p(x) \Longrightarrow\ q(x))$

6.4 Consider the logic of numbers where the language of symbols is $\mathcal{L} = \{+, \times, 0, 1\}$ where + and × are the addition and multiplication functions and 0 and 1 are the constant symbols for numbers zero and one. Answer the following.

1. Write a formula **even**(x) which expresses the property that x is an even number. In other words, the formula is true whenever x is substituted with an even number and false when you substitute it with a number not even.
2. Write a formula **lessthan**(x, y) which says that x is less than y.
3. Write a formula **div**(x, y) which says that x divides y.
4. Write a formula **prime**(x) which says that x is a prime number.
5. Write a sentence which says that there are infinitely many primes.
6. Write a formula **lcm**(x, y, z) which says that the lcm of x and y is z.
7. Goldbach's conjecture states that every even integer greater than 2 is the sum of two primes. Whether or not this is true is an open question of number theory. Write a first order sentence which expresses Goldbach's conjecture.

6.5 For each of the following wffs, write an equivalent formulas which uses only the symbols $\{\exists, \neg, \vee, \wedge\}$.

1. $\big(\forall x(P(x) \Longrightarrow Q(x))\big) \Longrightarrow \big(\forall x(P(x) \Longrightarrow Q(x))\big)$
2. $\big(\exists x \forall y\ R(x, y)\big) \Longrightarrow \big(\forall x \exists y\ R(x, y)\big)$

6.6 Formulate a satisfiable formula α in which a binary function symbol f occurs such that for every structure A, if $A \vdash \alpha$ then (A, f_A) is a group.

6.7 Prove that every formula α can be transformed to an equivalent formula β such that β is in rectified form.

6.8 Prove that every formula α can be transformed to an equi-satisfiable formula β such that β is in Skolem form.

CHAPTER 7

THE COMPUTATIONAL COMPLEXITY OF SATISFIABILITY

7.1 Undecidability of first order logic

Consider the following problem.

FO-SAT	
Input:	A first order wff α.
Output:	YES if α is satisfiable, otherwise NO.

The above problem is undecidable. This follows from the fact that the validity problem is undecidable.

FO-VALIDITY	
Input:	A first order wff α.
Output:	YES if α is valid, otherwise NO.

Theorem 7.1 *The problem of checking whether a first order formula is valid is undecidable.*

Proof : The proof is given by a reduction from halting problem of first order logic. ■

As a corollary we get this.

Theorem 7.2 *The problem of checking whether a first order wff is satisfiable is undecidable.*

Proof : Assume the problem is decidable. We show that we can then decide the validity problem. Let ϕ be a formula given as input to the validity problem. Check whether $\neg\phi$ is satisfiable or not. If it is satisfiable, then ϕ is not valid. Otherwise ϕ is valid. This is a contradiction, since we have shown above that the validity problem is undecidable. Therefore, the satisfiability problem is undecidable. ■

7.2 Model checking problem

The model checking problem is decidable and it is PSpace-complete.

MODEL CHECKING	
Input:	A FO structure M, assignment s and wff α.
Output:	YES if (M, s) satisfies α, otherwise NO.

We can show that this problem runs in time $(|M| + |s| + |\phi|)^{|\phi|}$.

7.3 Resolution and semi-decidability of Validity

7.4 Chapter exercises

7.1 Consider the model-checking problem: The input is a well formed first order logic sentence α, and a finite structure M. The output is Yes if M satisfies α. Otherwise the output is No. Answer the following questions.

1. Give an algorithm which solves the model-checking problem.
2. Find out the running time of the algorithm.
3. How much space does the algorithm take.

CHAPTER 8

PROOF SYSTEM

8.1 Natural Deduction

The natural deduction rules are as follows

Recollect that for a wff $\phi(x)$ and term t, we denote by $\phi[t \mapsto x]$ a substitution of x by t in ϕ. We will be interested in good substitutions.

Let us now look at natural deduction rules for first order logic. All the rules for propositional logic are going to hold for first order logic too. We will now talk about the extra rules.

(equality introduction) For all terms t and for all premises Γ, $\Gamma \vdash (t = t)$. This rule says that for any term t, the formula $(t = t)$ holds.

$$\frac{}{(t = t)}\ {=}i$$

Figure 8.1 equal to Introduction ($= i$)

$$\frac{(t_1 = t_2) \qquad \phi[t_1 \mapsto x]}{\phi[t_2 \mapsto x]}\ {=}e$$

Figure 8.2 equal to elimination ($= e$)

(equality elimination) If $\Gamma \vdash (t_1 = t_2)$ *and* $\Gamma \vdash \phi[t_1 \mapsto x]$*, then* $\Gamma \vdash \phi[t_2 \mapsto x]$. The rule says that, let us assume we can prove that $(t_1 = t_2)$. Then in any formula where we can substitute t_1, we should also be able to substitute t_2. One small catch is that, we need to assume that $\phi[t_1 \mapsto x]$ and $\phi[t_1 \mapsto x]$ are good substitutions. Consider the following example. Let $x \times (y + 1) = x \times y + x$ and $z = x \times (y + 1)$. Then $z = x \times y + x$. This can be deduced using equality elimination as shown below.

1.	$x \times (y + 1) = x \times y + x$	premise
2.	$z = w[x \times (y + 1) \mapsto w]$	premise
3.	$z = w[x \times y + x \mapsto w]$	$= e$ 1,2

Figure 8.3 $\{x \times (y + 1) = x \times y + x, z = x \times (y + 1)\} \vdash z = x \times y + x$

In the above proof, we identified a different way to write the second premise. It is written as a substitution on a variable. This helps us to use the equal to elimination rule.

(forall elimination) If $\Gamma \vdash \forall x\ \phi$*, then* $\Gamma \vdash \phi[t \mapsto x]$ *for any term* t. This rule is fairly obvious. If we can prove $\forall x\ \phi$, it means that $\phi[t \mapsto x]$ is true for any term t.

$$\frac{\forall x\ \phi}{\phi[t \mapsto x]}\ \forall e$$

Figure 8.4 forall elimination ($\forall e$)

(forall introduction) If $\Gamma \vdash \phi[a \mapsto x]$ *for a fresh variable* a*, then* $\Gamma \vdash \forall x \phi$. This rule is tricky. It is better to understand this rule through an example. Let us see how we prove the following $A \subseteq C$ from the premises $\{A \subseteq B, B \subseteq C\}$.

1.	$\forall x \left((x \in A) \implies (x \in B)\right)$	premise $A \subseteq B$
2.	$\forall x \left((x \in B) \implies (x \in C)\right)$	premise $B \subseteq C$
3.	Let a be an arbitrary element.	
4.	Assume $a \in A$	
5.	Then $a \in B$	(from $A \subseteq B$, $\forall e$ and $\implies e$)
6.	$\therefore a \in C$	(from $B \subseteq C$, $\forall e$ and $\implies e$)
7.	$(a \in A) \implies (a \in C)$	$\implies i$ 4-6
8.	Therefore, $\forall x \left((x \in A) \implies (x \in C)\right)$	(another way to say $A \subseteq C$)

Figure 8.5 A proof which picks an arbitrary element

The proof above is not a natural deduction proof (since we used other rules). The proof was mentioned to show how we would have proved the transitivity property of sets. The proof involves introducing a for all quantifier in the conclusion. In the above proof, we pick an arbitrary element and proved some property. Since, we started of with an arbitrary element, the proof holds for any element. This is the logic behind the proof. The below pictorial representation gives the rule.

$$\begin{array}{c} \text{Take arbitrary } a \\ \vdots \\ \phi[a \mapsto x] \end{array} \over \forall x\ \phi(x)\ \forall i$$

Figure 8.6 forall introduction ($\forall i$)

Let us look at another example: From $\{\forall x \left(Lion(x) \implies Strong(x)\right), \forall x\ Lion(x)\}$ we prove that $\forall x\ Strong(x)$.

1.	$\forall x \left(Lion(x) \Longrightarrow Strong(x)\right)$	premise
2.	$\forall x\ Lion(x)$	premise
3.	Take a	arbitrary
4.	$\left(Lion(a) \Longrightarrow Strong(a)\right)$	$\forall e$ 1
5.	$Lion(a)$	$\forall e$ 2
6.	$Strong(a)$	$\Longrightarrow e$ 5,4
7.	$\forall x\ Strong(x)$	$\forall i$ 3-6

Figure 8.7 Proof of $\{\forall x\ (P(x) \Longrightarrow Q(x)), \forall x\ P(x)\} \vdash \forall x\ Q(x)$

(existential introduction) If $\Gamma \vdash \phi[a \mapsto x]$ *for some term* t*, then* $\Gamma \vdash \exists x \phi$. This rule is easy to understand. It says that, if there is a proof of ϕ where x is substituted with a term t, then it shows that there exists some element x which satisfies ϕ. Note that this rule is applicable only if the substitution is good.

$$\frac{\phi[t \mapsto x]}{\exists x\ \phi(x)}\ \exists i$$

Figure 8.8 exists introduction ($\exists i$)

(existential elimination) If $\Gamma \vdash \exists \phi$ *and* $\Gamma \cup \{\phi[a \mapsto x]\} \vdash \alpha$ *for a fresh variable* a*, then* $\Gamma \vdash \alpha$. Since the rule is difficult to see, let us explain by an example. Consider the following two wffs: $\{\forall x \left(Lion(x) \Longrightarrow \forall y\ Afraid(y)\right), \exists x\ Lion(x)\}$. From these premises we should be able to derive that $\forall y\ Afraid(y)$. Here is an argument. Since there exists a lion, consider a lion a. Therefore, we have that for all $y\ Afraid(y)$.

1.	$\forall x \left(Lion(x) \Longrightarrow \forall y\ Afraid(y)\right)$	premise
2.	$\exists x\ Lion(x)$	premise
3.	Take an arbitrary a such that $Lion(a)$	
4.	Therefore, $Lion(a) \Longrightarrow \forall y\ Afraid(y)$	$\forall e$, 1
5.	$\forall y\ Afraid(y)$	$\Longrightarrow e$ 4,5
6.	$\forall y\ Afraid(y)$	from the above derivation

Figure 8.9 An example for existential elimination

We capture this rule in the following figure.

$$\dfrac{\exists x\ \phi \qquad \begin{array}{c}\text{Take arbitrary } a \text{ such that } \phi[a \mapsto x] \\ \vdots \\ \gamma\end{array}}{\gamma}\ \forall i$$

Figure 8.10 exists elimination ($\exists e$)

With this we finish the rules of natural deduction. In the next section we will see the soundness and completeness of natural deduction of first order logic.

8.2 Soundness and Completeness of Natural deduction

Soundness follows by a case analysis for each of the rules. It says that, the rules of natural deduction "make sense".

Theorem 8.1 (soundness) *Let Γ be a set of wffs and ϕ a wff such that $\Gamma \vdash \phi$. Then, we have that $\Gamma \models \phi$.*

The question to ask is, what about the other direction. Can we show that, all formulas which make sense, can be proved from the premises. Many mathematicians were pondering over this quesion, when an young Gödel showed this in his PhD. He argued that, if a set of wffs Γ satisfies a wff ϕ, then we can prove ϕ from Γ using natural deduction.

Theorem 8.2 (Gödel's completeness) *Let Γ be a set of wffs and ϕ a wff such that $\Gamma \models \phi$. Then $\Gamma \vdash \phi$.*

We skip the proof of completeness.

8.3 Chapter exercises

Use natural deduction to prove the following

1. $T \vdash \neg\alpha \implies (\alpha \implies (\alpha \implies \beta))$
2. $(\neg p \implies \neg q) \vdash (q \implies p)$
3. $T \vdash \forall x(p(x) \implies q(x)) \implies (\exists x\ p(x) \implies \exists x\ q(x))$
4. $\{\forall x\ (P(x) \vee Q(x)), \exists x \neg Q(x), \forall x\ (R(x) \implies \neg P(x))\} \vdash \exists x \neg R(x)$
5. $T \vdash \exists x\ (D(x) \implies \forall y D(y))$

Index

Bibliography

[BA12] Mordechai Ben-Ari. *Mathematical Logic for Computer Science, 3rd Edition.* Springer, 2012.

[BD] Shai Ben-David. Logic for CS, https://www.youtube.com/watch?v=kceafjjtjyu.

[CLRS01] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms.* MIT Press, Cambridge, MA, 2001.

[End72] H. B. Enderton. *A Mathematical Introduction to Logic.* Academic Press, 1972.

[HR04] Michael Huth and Mark Ryan. *Logic in Computer Science.* Cambridge, second edition, 2004.

[MS11] Madhavan Mukund and S. P. Suresh. *An Introduction to Logic.* https://www.cmi.ac.in/ madhavan/papers/pdf/logic-aug2011.pdf, 2011.

[MU05] Michael Mitzenmacher and Eli Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis.* Cambridge University Press, 2005.

[RN95] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach.* Prentice Hall, 1995.

[Sch89] Uwe Schöning. *Logic for Computer Scientists.* Birkhauser, Boston, 1989.

[Smu82] Raymond Smullyan. *The Lady or the Tiger? and other Logic Puzzles.* Oxford, 1982.

[Smu85] Raymond Smullyan. *To Mock a Mockingbird.* Knopf, 1985.