CS 433 Automated Reasoning 2025

Lecture 2: Propositional logic - syntax and parsing

Instructor: Ashutosh Gupta

IITB India

Compile date: 2025-01-10

Commentary: In this lecture, we will cover the syntax of propositional logic carefully. We will define the involved symbols and their allowed arrangements. We will also discuss various aspects of handling formulas. Please appreciate the importance of carefully written definitions. It may appear very pedantic, but we need a solid foundation.

Topic 2.1

Propositional logic - Syntax

Syntax

We need a quick method of identifying if a group of symbols is a logical argument.

We usually define a syntax.

Example 2.1

Grammar of English

Let us define syntax for propositional logic

Propositions

The logic is over a list of propositions.

- Sky is blue
- Sun is hot
- ... *many more*

We do not care what each one says. We give each one of them a symbol.

Propositional variables

We assume that there is a countably-infinite set **Vars** of propositional variables.

- Since **Vars** is countable, we assume that variables are indexed.

$$\textbf{Vars} = \{p_1, p_2, \dots\}$$

- The variables are just names/symbols without inherent meaning
- We may also use $p, q, r, .., x, y, z$ to denote the propositional variables
- Propositional variables are also called Boolean variables

Commentary: All results presented in this course are extendable to uncountable **Vars**. For the uncountable setting, we need transfinite induction. We will ignore those extensions.

Logical connectives

The following 10 symbols are called logical connectives.

formal name	symbol	read as	
true	$\top$	top	0-ary symbols
false	$\bot$	bot	
negation	$\neg$	not	unary symbols
conjunction	$\wedge$	and	binary symbols
disjunction	$\vee$	or	
implication	$\Rightarrow$	implies	
equivalence	$\Leftrightarrow$	if and only if	
exclusive or	$\oplus$	xor	
open parenthesis	(		punctuation
close parenthesis	)		

We assume that the logical connectives are not in **Vars**.

Propositional formulas

A propositional formula is a finite string containing symbols in **Vars** and logical connectives.

Definition 2.1

The set of propositional formulas is the smallest set **P** *such that*

- $\top, \bot \in \mathbf{P}$
- *if* $p \in \mathbf{Vars}$ *then* $p \in \mathbf{P}$
- *if* $F \in \mathbf{P}$ *then* $\neg F \in \mathbf{P}$
- *if* $\circ$ *is a binary symbol and* $F, G \in \mathbf{P}$ *then* $(F \circ G) \in \mathbf{P}$

Some notation

Definition 2.2

$\top, \bot$, *and* $p \in$ **Vars** *are atomic formulas.*

Definition 2.3

For each $F \in \mathbf{P}$, *let* $\mathbf{Vars}(F)$ *be the set of variables appearing in* F.

Examples of propositional formulas

Exercise 2.1

Which of the following are in **P***?*

- $\top \Rightarrow \bot$ ✗
- $(\top \Rightarrow \bot)$ ✓
- $(p_1 \Rightarrow \neg p_2)$ ✓
- (p_1) ✗
- $\neg\neg\neg\neg\neg\neg\neg p_1$ ✓

Not all strings over **Vars** and logical connectives are in **P**.

We need a method to recognize a string belongs to **P** or not.

How can we argue that a string does or does not belong to **P**?

Commentary: Please carefully look at the generation grammar. We need to carefully understand the role of parenthesis to disambiguate formulas. It is an interesting note that $\neg$ does not need parentheses.

Topic 2.2

Encoding arguments into logic

Example : symbolic argument

Commentary: A detour to explore the use of propositional formulas. In this section, we will see how to use PL for solving fun puzzles.

Example 2.2

We have seen the following argument.

If c then if s then f. not f. Therefore, if s then not c.

where

- c = *the seed catalogue is correct*
- s = *seeds are planted in April*
- f = *the flowers bloom in July*

We can write the above argument as propositional formula as follows

$$\Big(\Big(\underbrace{(c \Rightarrow (s \Rightarrow f))}_{\text{Premise 1}} \wedge \underbrace{\neg f}_{\text{Premise 2}} \Big) \Rightarrow \underbrace{(s \Rightarrow \neg c)}_{\text{Conclusion}} \Big)$$

Example: symbolizing bad and good puzzle

Problem Context

Example 2.3

The good people always tell the truth and the not good people always tell a lie. Now let us consider the following puzzle.

There are two people A and B. A says, "I am not good or B is good". What are A and B?

Let us give symbols to propositions:

- p_A = *A is good.*
- p_B = *B is good.*

Therefore, we encode the puzzle as follows.

$$(\underbrace{(\neg p_A \vee p_B)}_{\text{Statement of } A} \Leftrightarrow p_A)$$

To solve the puzzle, we need a satisfying assignment to the formula.

Commentary: The puzzle is borrowed from What is The Name of This Book? by Raymond M. Smullyan

Topic 2.3

Parsing formulas

Parse tree

Commentary: Let us get back to identifying if a string of symbols is a formula a not. We present a concept of the parse tree, which gives us an algorithm for the identification. The compilers for programming languages use parse trees to compile programs. The principal is the same.

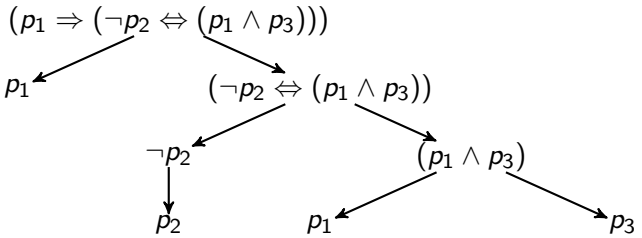
$F \in \mathbf{P}$ iff F is obtained by unfolding of the generation rules

Definition 2.4

A parse tree of a formula $F \in \mathbf{P}$ is a tree such that

- *the root is F,*
- *leaves are atomic formulas, and*
- *each internal node is formed by applying some formation rule on its children.*

Example 2.4



Parse tree and unique parsing

Theorem 2.1
$F \in \mathbf{P}$ iff there is a parse tree of F.

Proof.
The reverse direction is immediate. In the forward direction, we prove a stronger theorem, i.e., existence of unique parsing tree. □

Theorem 2.2
Each $F \in \mathbf{P}$ has a unique parsing tree.

Proof.
The proof is at the last section of the slides. □

Commentary: The proof is needed for the rigor of logic. However, we skipped the proof in the class.

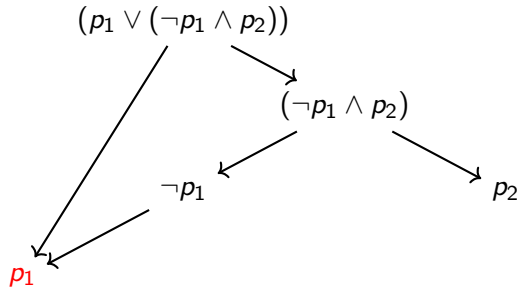
Parse tree is a directed-acyclic graph (DAG)

We have been thinking that the parsing produces parse tree.

However, the parsing produces a parse DAG.

Example 2.5

Consider formula $(p_1 \vee (\neg p_1 \wedge p_2))$. *The following is the parse tree of the above formula.*



Topic 2.4

Important notations: subformulas and substitutions

Subformula

Definition 2.5

A formula G is a subformula of formula F if G occurs within F. G is a proper subformula of F if $G \neq F$. Let $sub(F)$ denote the set of subformulas of F.

The nodes of the parse tree of F form the set of subformulas of F.

Definition 2.6

Immediate subformulas are the children of a formula in its parse tree, and leading connective is the connective that is used to join the children.

Example 2.6

$sub((\neg p_2 \Leftrightarrow (p_1 \wedge p_3))) = \{(\neg p_2 \Leftrightarrow (p_1 \wedge p_3)), \neg p_2, (p_1 \wedge p_3), p_1, p_2, p_3\}$

The leading connective of F is $\Leftrightarrow$.

Commentary: *Note that the above definition does not allow $p_2 \Leftrightarrow (p_1 \wedge p_3)$ to be a subformula of F, because $p_2 \Leftrightarrow (p_1 \wedge p_3)$ is not a formula. In later discussions, we may drop parentheses in our writings and it may cause confusion. So, when we apply the above definition we need to keep the invisible parentheses in our mind.*

Substitution

Definition 2.7

For $F \in \mathbf{P}$ and $p_1, \ldots, p_k \in \mathbf{Vars}$, let $F[G_1/p_1, \ldots, G_k/p_k]$ denote another formula obtained by simultaneously replacing all occurrence of p_i by a formula G_i for each $i \in 1..k$.

Example 2.7

1. $(p \Rightarrow (r \Rightarrow p))[(r \oplus s)/p] = ((r \oplus s) \Rightarrow (r \Rightarrow (r \oplus s)))$
2. $(p \Rightarrow (r \Rightarrow p))[(r \oplus s)/p, x/r] \neq (p \Rightarrow (r \Rightarrow p))[(r \oplus s)/p][x/r]$

Exercise 2.2

a. Definition 2.7 is informal. Give a formal inductive definition over the structure of F.

b. Write the result of substitutions in the second example.

c. Give a most general restriction on substitutions such that simultaneous and sequential substitutions (like right hand side of the second example) produce the same result.

Notation for substitution

For shorthand, we may write a formula F as

$$F(p_1, \ldots, p_k),$$

where we say that variables $p_1, \ldots, p_k$ play a special role in F.

Let $F(G_1, \ldots, G_n)$ be $F[G_1/p_1, \ldots, G_k/p_k]$.

Example 2.8

Let $F(p, q) = \neg p \oplus q$

$F((r \vee q), \top) = \neg(r \vee q) \oplus \top$

Commentary: This notation is very useful in the case when we do not know F but want to talk about the substitutions in a convenient way.

Topic 2.5

Shorthand

Too many parentheses

In the above syntax, we need to write a large number of parentheses.

Using precedence order over logical connectives, we may drop some parentheses without losing the unique parsing property.

Example 2.9

Consider $((p \wedge q) \Rightarrow (r \vee p))$

- *We may drop outermost parenthesis without any confusion*

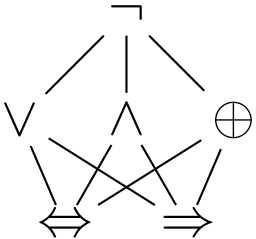
$$(p \wedge q) \Rightarrow (r \vee p)$$

- *If* $\wedge$ *and* $\vee$ *get precedence over* $\Rightarrow$ *during parsing, we do not need the rest of parentheses*

$$p \wedge q \Rightarrow r \vee p$$

Precedence order

We will use the following precedence order in writing the propositional formulas



Example: parsing using the precedence order

Example 2.10

Consider formula $p \wedge q \Rightarrow r \vee p$. *Let us try to bring back the parentheses.*

$\Rightarrow$ *has lower precedence than* $\wedge$, *therefore we can group neighbours of* $\wedge$

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

Since $\vee$ *has higher precedence over* $\Rightarrow$, *we first group* $\vee$.

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

Now we can group $\Rightarrow$ *without any confusion*

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

Example precedence order

Example 2.11

Which of the following formulas can be unambiguously parsed?

- $\neg p \vee (p \oplus q) \Leftrightarrow p \wedge q$ ✓
- $p \vee q \wedge r$ ✗
- $p \vee q \vee r$ ✗
- $p \Rightarrow q \Rightarrow r$ ✗

Associativity preference may further reduce the need of parenthesis

Associative

Problem: If a binary operator repeats, we do not know how to group.

Solution: we give preference to one side or another.

Let us make all our operators "right associative", i.e., first group the rightmost occurrence.

Example 2.12

Consider formula $p \Rightarrow q \Rightarrow r$.

We first group the right $\Rightarrow$*:* $p \Rightarrow (q \Rightarrow r)$

Then, we group the left $\Rightarrow$*:* $(p \Rightarrow (q \Rightarrow r))$

Commentary: Not all operators are affected by associativity. For example, $\wedge$ and $\vee$ operators have same meaning if we use any order of associativity. On the other hand, $\Rightarrow$ needs a convention for associativity.

Exercise 2.3

Modify the parsing procedure of the earlier slide to support the above.

Topic 2.6

Problems

Exercise: symbolizing bad and good puzzle**

Exercise 2.4

People are either good or bad. The good people always tell the truth and the bad people always tell a lie. Now let us consider the following puzzle.

There are two people A and B. A said some thing, but we could not hear. B said, "A is saying that A is bad". What are A and B?

Encode the above puzzle into a propositional logic formula.

Commentary: The puzzle is borrowed from What is The Name of This Book? by Raymond M. Smullyan. It is difficult to encode because we also need to model internal mental state of the participants.

Exercise: more puzzles (midterm 2021)

Exercise 2.5

The following facts are known about three suspects X, Y, and Z of a crime.

1. *If X is guilty and Y is innocent, then Z is guilty.*
2. *If Z is guilty, then one of the other two worked with Z.*
3. *X never works with Z.*
4. *At least one of them is guilty.*

Is X necessarily guilty?

Encode the puzzle into a formula, whose satisfiability gives the answer of the puzzle? Please note that the purpose of the question is the encoding *not solving the puzzle.*

Commentary: The puzzle is borrowed from What is The Name of This Book? by Raymond M. Smullyan

Exercise: more puzzles (quiz 2020)

Exercise 2.6

People are either good or bad. The good people always tell the truth and the bad people always tell a lie. Now let us consider the following puzzle.

There are three people A, B, and C. A said, "All of us are bad.". B said, "Exactly one of us is good.". What are A, B, and C?

Encode the above puzzle into a propositional logic formula.

Commentary: The puzzle is borrowed from What is The Name of This Book? by Raymond M. Smullyan

Let expression

We may extend the grammar of proportional logic with let expressions.

$$(\text{let } p = F \text{ in } F)$$

Let-expression is a syntactic device to represent large formulas succinctly.

$$(\text{let } p = F \text{ in } G) \text{ represents } G[F/p]$$

Example 2.13

$$(\textit{let } p = (q \wedge r) \textit{ in } ((p \wedge s) \vee (q \Rightarrow \neg p))) \qquad \textit{represents} \qquad ((q \wedge r) \wedge s) \vee (q \Rightarrow \neg(q \wedge r))$$

Exercise 2.7

Give a formula, which can be represented by let expressions in exponentially less space.

Exercise: geopolitics (quiz 2022)

Exercise 2.8

If Ukraine applies for NATO membership and NATO has an open-door policy, Russia invades Ukraine. If NATO does not have an open-door policy, Ukraine does not apply for NATO membership and Russia does not invade Ukraine. NATO has an open-door policy. Will Russia necessarily invade?

Encode the above query into a propositional formula, whose satisfiability gives the answer of the query. Please note that the purpose of the question is the encoding *not solving the query.*

Topic 2.7

Extra lecture slides: unique parsing

Matching parentheses

Theorem 2.3

Every $F \in \mathbf{P}$ has matching parentheses, i.e., equal number of '(' and ')'.

Proof.

base case:
atomic formulas have no parenthesis. Therefore, matching parenthesis

induction steps:
We assume $F, G \in \mathbf{P}$ has matching parentheses.
Let n_F and n_G be the number of '(' in F and G respectively.
Trivially, $\neg F$ has matching parentheses.
For some binary symbol $\circ$, the number of both '(' and ')' in $(F \circ G)$ is $n_F + n_G + 1$.

Due to the structural induction, the property holds. □

Prefix of a formula

Theorem 2.4

A proper prefix of a formula is not a formula.

Proof.

We show a proper prefix of a formula is in one of the following forms.

1. strictly more '(' than ')',
2. a (possibly empty) sequence of $\neg$.

Clearly, both the cases are not in **P**.

Base case:

A proper prefix of atomic formulas is empty string, which is the second case ...

Exercise 2.9

Give examples of the above two cases

Prefix of a formula II

Proof(contd.)

Induction step:
Let $F, G \in \mathbf{P}$.

Consider proper prefix F' of $\neg F$. There are two cases.

- $F' = \epsilon$, case 2
- $F' = \neg F''$, where F'' is a proper prefix of F. Now we again have two subcases for F''.
  - If F'' is in case 1, F' belongs to case 1
  - If $F'' = \neg..\neg$, F' belongs to case 2

...

Prefix of a formula III

Proof(contd.)

By induction F and G have balanced parenthesis.

Consider proper prefix H of $(F \circ G)$, F' be prefix of F, and G' be prefix of G.

- If $H = (F \circ G$, H belongs to case 1 because H has one extra '('
- If $H = (F \circ G'$, H belongs to case 1(Why?)

Similarly the following cases are handled

- $H = (F \circ$
- $H = (F$
- $H = (F'$
- $H = ($

□

Exercise 2.10

Complete the (Why?).

Unique parsing

Theorem 2.5

Each $F \in \mathbf{P}$ has a unique parsing tree.

Proof.

$\nu(F) \triangleq$ number of logical connectives in F. We apply induction over $\nu(F)$.

base case: $\nu(F) = 0$

F is an atomic formula, therefore has a single node parsing tree.

inductive steps: $\nu(F) = n$

We assume that each F' with $\nu(F') < n$ has a unique parsing tree.

case $F = \neg G$: Since G has a unique parsing tree, F has a unique parsing tree.

case $F = (G \circ H)$:

Suppose there is another formation rule such that $F = (G' \circ' H')$.

Since $F = (G \circ H) = (G' \circ' H')$, $G \circ H) = G' \circ' H')$.

Without loss of generality, G is prefix of G'.

Since $G, G' \in \mathbf{P}$, G can not be proper prefix of G'. Therefore, $G = G'$.

Therefore, $\circ = \circ'$. Therefore, $H = H'$. Therefore, only one way to unfold F.

F has a unique parsing tree. □

Parsing algorithm

Algorithm 2.1: PARSER

Input: F : a string over **Vars** and logical connectives
Output: parse tree if $F \in \mathbf{P}$, exception FAIL otherwise
if $F = p$ *or* $F = \top$ *or* $F = \bot$ **then return** $(\{F\}, \emptyset)$;
if $F = \neg G$ **then**
    $(V, E) := \text{PARSER}(G)$;
    return $(V \cup \{F\}, E \cup \{(F, G)\})$;
if F *has matching parentheses and* $F = (F')$ **then**
    $G :=$ smallest prefix of F' where non-zero parentheses match or atom after a sequence of '$\neg$'s;
    $oH := tail(F', len(G))$;
    if *the above two match succeed* **then**
        $(V_1, E_1) := \text{PARSER}(G)$;
        $(V_2, E_2) := \text{PARSER}(H)$;
        return $(V_1 \cup V_2 \cup \{F\}, E_1 \cup E_2 \cup \{(F, G), (F, H)\})$;
Throw FAIL

Commentary: The previous proofs suggest a parsing algorithm to generate parsing tree.

Parse Algorithm

Exercise 2.11

Show the run of Algorithm 2.1 on the following formulas.

1. $\neg q \Rightarrow (p \oplus r \Leftrightarrow s)$
2. $(\neg(p \Rightarrow q) \wedge (r \Rightarrow (p \Rightarrow q)))$

Using precedence order

Consider the following formula for $n > 1$

$$F_0 \circ_1 F_1 \circ_2 F_2 \circ_3 \cdots \circ_n F_n,$$

where F_0,..,F_n are either atomic or enclosed by parentheses, or their negation.

We transform the formula as follows

- Find an $\circ_i$ such that $\circ_{i-1}$ and $\circ_{i+1}$ have lower precedence if they exist.
- Introduce parentheses around $F_{i-1} \circ_i F_i$ and call it $F'_i \triangleq (F_{i-1} \circ_i F_i)$.

$$F_0 \circ_1 \cdots \circ_{..} F_{i-2} \circ_{i-1} F'_i \circ_{i+1} F_{i+1} \circ_{..} \cdots \circ_n F_n$$

We apply the above until $n = 1$ and then apply the normal parsing.

Inside of F_is may also have ambiguities, which are recursively resolved using the above procedure.

Commentary: We have not presented the above as a formal algorithm. However, we can present the procedure in the above style. You will find many computer science texts do not write their algorithms in a formal presentation to avoid cumbersome notation. Please learn to handle.

Precedence order

Exercise 2.12

Add minimum parentheses in the following strings such that they have unique parsing under our precedence order

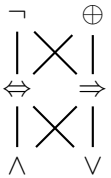
1. $p \wedge q \vee r \wedge s \wedge t \vee u \vee v \wedge w$
2. $p \Rightarrow \neg q \oplus p \vee p \wedge \neg r \Leftrightarrow s \wedge t$

Commentary: Please work the above problems with and without associative preference rules. In the exams, we will make it clear.

Custom precedence order

Exercise 2.13

Consider the following precedence order



Add minimal parentheses in the following strings such that they have unique parsing tree

1. $\neg p \Rightarrow q \wedge r \Rightarrow p \Rightarrow q$
2. $p \Rightarrow \neg q \oplus p \vee p \wedge \neg r \Leftrightarrow s \wedge t$

Commentary: *Please work the above problems with and without associative preference rules. In the exams, we will make it clear.*

End of Lecture 2