CS 208 : Automata Theory and Logic

Spring 2024

Instructor : Prof. Supratik Chakraborty

Disclaimer

Contents

Chapter 1

Propositional Logic

In this course we look at two ways of computation: a state transition view and a logic centric view. In this chapter we begin with logic centered view with the discussion of propositional logic.

Example. Suppose there are five courses $C_1, \dots, C_5$, four slots $S_1, \dots, S_4$, and five days $D_1, \dots, D_5$. We plan to schedule these courses in three slots each, but we have also have the following requirements:

	D_1	D_2	D_3	D_4	D_5
S_1					
S_2					
S_3					
S_4					

- For every course C_i, the three slots should be on three different days.
- Every course C_i should be scheduled in at most one of $S_1, \dots, S_4$.
- For every day D_i of the week, have at least one slot free.

Propositional logic is used in many real-world problems like timetables scheduling, train scheduling, airline scheduling, and so on. One can capture a problem in a propositional logic formula. This is called as encoding. After encoding the problem, one can use various software tools to systematically reason about the formula and draw some conclusions about the problem.

1.1 Syntax

We can think of logic as a language which allows us to very precisely describe problems and then reason about them. In this language, we will write sentences in a specific way. The symbols used in propositional logic are given in Table 1.1. Apart from the symbols in the table we also use variables usually denoted by small letters $p, q, r, x, y, z, \dots$ etc. Here is a short description of propositional logic symbols:

- **Variables**: They are usually denoted by smalls ($p, q, r, x, y, z, \dots$ etc). The variables can take up only true or false values. We use them to denote propositions.
- **Constants**: The constants are represented by $\top$ and $\bot$. These represent truth values true and false.

- **Operators**: $\wedge$ is the conjunction operator (also called AND), $\vee$ is the disjunction operator (also called OR), $\neg$ is the negation operator (also called NOT), $\rightarrow$ is implication, and $\leftrightarrow$ is bi-implication (equivalence).

Name	Symbol	Read as
true	$\top$	top
false	$\bot$	bot
negation	$\neg$	not
conjunction	$\wedge$	and
disjunction	$\vee$	or
implication	$\rightarrow$	implies
equivalence	$\leftrightarrow$	if and only if
open parenthesis	(	
close parenthesis	)	

Table 1.1: Logical connectives.

For the timetable example, we can have propositional variables of the form p_{ijk} with $i \in [5]$, $j \in [5]$ and $k \in [4]$ (Note that $[n] = \{1, \ldots, n\}$) with p_{ijk} representing the proposition 'course C_i is scheduled in slot S_k of day D_j'.

Rules for formulating a formula:

- Every variable constitutes a formula.
- The constants $\top$ and $\bot$ are formulae.
- If φ is a formula, so are $\neg\varphi$ and (φ).
- If φ_1 and φ_2 are formulas, so are $\varphi_1 \wedge \varphi_2, \varphi_1 \vee \varphi_2, \varphi_1 \rightarrow \varphi_2$, and $\varphi_1 \leftrightarrow \varphi_2$.

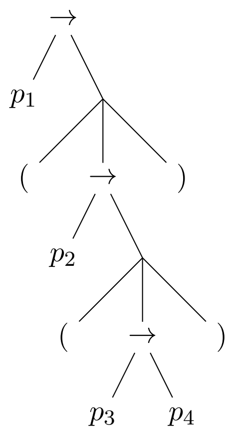
Propositional formulae as strings and trees:

Formulae can be expressed as a strings over the alphabet $\textbf{Vars} \cup \{\top, \bot, \neg, \wedge, \vee, \rightarrow, \leftrightarrow, (,)\}$. **Vars** is the set of symbols for variables. Not all words formed using the alphabet qualify as propositional formulae. A string constitutes a well-formed formula (wff) if it was constructed while following the rules. Examples: $(p_1 \vee \neg q_2) \wedge (\neg p_2 \rightarrow (q_1 \leftrightarrow \neg p_1))$ and $p_1 \rightarrow (p_2 \rightarrow (p_3 \rightarrow p_4))$.
Well-formed formulas can be represented using trees. Consider the formula $p_1 \rightarrow (p_2 \rightarrow (p_3 \rightarrow p_4))$. This can be represented using the parse tree in figure Figure 1.1a. Notice that while strings require parentheses for disambiguation, trees don't, as can be seen in Figure 1.1b and Figure 1.1c.

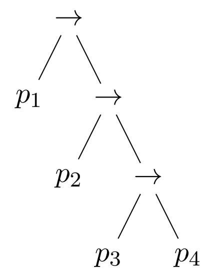
1.2 Semantics

Semantics give a meaning to a formula in propositional logic. The semantics is a function that takes in the truth values of all the variables that appear in a formula and gives the truth value of the formula. Let 0 represent "false" and 1 represent "true". The semantics of a formula φ of n variables is a function

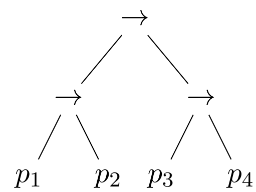
$$[\![\varphi]\!] : \{0, 1\}^n \rightarrow \{0, 1\}$$



(a)



(b) Parse tree for $p_1 \to (p_2 \to (p_3 \to p_4))$



(c) Parse tree for $(p_1 \to p_2) \to (p_3 \to p_4)$

Figure 1.1: Parse trees obviate the need for parentheses.

It is often presented in the form of a truth table. Truth tables of operators can be found in table Table 1.2.

φ	$\neg\varphi$
0	1
1	0

(a) Truth table for $\neg\varphi$.

φ_1	φ_2	$\varphi_1 \wedge \varphi_2$
0	0	0
0	1	0
1	0	0
1	1	1

(b) Truth table for $\varphi_1 \wedge \varphi_2$.

φ_1	φ_2	$\varphi_1 \to \varphi_2$
0	0	0
0	1	1
1	0	1
1	1	1

(c) Truth table for $\varphi_1 \vee \varphi_2$.

φ_1	φ_2	$\varphi_1 \to \varphi_2$
0	0	1
0	1	1
1	0	0
1	1	1

(d) Truth table for $\varphi_1 \to \varphi_2$.

φ_1	φ_2	$\varphi_1 \to \varphi_2$
0	0	1
0	1	0
1	0	0
1	1	1

(e) Truth table for $\varphi_1 \leftrightarrow \varphi_2$.

Table 1.2: Truth tables of operators.

Remark. Do not confuse 0 and 1 with $\top$ and $\bot$: 0 (false) and 1 (true) are meanings, while $\top$ and $\bot$ are symbols.

Rules of semantics:

- $[\![\neg\varphi]\!] = 1$ iff $[\![\varphi]\!] = 0$.
- $[\![\varphi_1 \wedge \varphi_2]\!] = 1$ iff $[\![\varphi_1]\!] = [\![\varphi_2]\!] = 1$.
- $[\![\varphi_1 \vee \varphi_2]\!] = 1$ iff at least one of $[\![\varphi_1]\!]$ or $[\![\varphi_2]\!]$ evaluates to 1.

- $[\![\varphi_1 \rightarrow \varphi_2]\!] = 1$ iff at least one of $[\![\varphi_1]\!] = 0$ or $[\![\varphi_2]\!] = 1$.
- $[\![\varphi_1 \leftrightarrow \varphi_2]\!] = 1$ iff at both $[\![\varphi_1 \rightarrow \varphi_2]\!] = 1$ and $[\![\varphi_2 \rightarrow \varphi_1]\!] = 1$.

Truth Table: A truth table in propositional logic enumerates all possible truth values of logical expressions. It lists combinations of truths for individual propositions and the compound statement's truth.

Example. Let us construct a truth table for $[\![(p \vee s) \rightarrow (\neg q \leftrightarrow r)]\!]$ (see Table 1.3).

p	q	r	s	$p \vee s$	$\neg q$	$\neg q \leftrightarrow r$	$(p \vee s) \rightarrow (\neg q \leftrightarrow r)$
0	0	0	0	0	1	0	1
0	0	0	1	1	1	0	0
0	0	1	0	0	1	1	1
0	0	1	1	1	1	1	1
0	1	0	0	0	0	1	1
0	1	0	1	1	0	1	1
0	1	1	0	0	0	0	1
0	1	1	1	1	0	0	0
1	0	0	0	1	1	0	0
1	0	0	1	1	1	0	0
1	0	1	0	1	1	1	1
1	0	1	1	1	1	1	1
1	1	0	0	1	0	1	1
1	1	0	1	1	0	1	1
1	1	1	0	1	0	0	0
1	1	1	1	1	0	0	0

Table 1.3: Truth table of $(p \vee s) \rightarrow (\neg q \leftrightarrow r)$.

1.2.1 Important Terminology

A formula φ is said to (be)

- **satisfiable** or **consistent** or SAT iff $[\![\varphi]\!] = 1$ for some assignment of variables. That is, there is at least one way to assign truth values to the variables that makes the entire formula true. Both a formula and its negation may be SAT at the same time (φ and $\neg\varphi$ may both be SAT).
- **unsatisfiable** or **contradiction** or UNSAT iff $[\![\varphi]\!] = 0$ for all assignments of variables. That is, there is no way to assign truth values to the variables that makes the formula true. If a formula φ is UNSAT then $\neg\varphi$ must be SAT (it is in fact valid).
- **valid** or **tautology**: $[\![\varphi]\!] = 1$ for all assignments of variables. That is, the formula is always true, no matter how the variables are assigned. If a formula φ is valid then $\neg\varphi$ is UNSAT.
- **semantically entail** φ_1 iff $[\![\varphi]\!] \preceq [\![\varphi_1]\!]$ for all assignments of variables, where 0 (false) $\preceq$ 0 (true). This is denoted by $\varphi \models \varphi_1$. If $\varphi \models \varphi_1$, then for every assignment, if φ evaluates to 1 then φ_1 will evaluate to 1. Equivalently $\varphi \rightarrow \varphi_1$ is valid.

- **semantically equivalent** to φ_1 iff $\varphi \models \varphi_1$ and $\varphi_1 \models \varphi$. Basically φ and φ_1 have identical truth tables. Equivalently, $\varphi \leftrightarrow \varphi_1$ is valid.
- **equisatisfiable** to φ_1 iff either both are SAT or both are UNSAT. Also note that, semantic equivalence implies equisatisfiability but **not** vice-versa.

Term	Example
SAT	$p \vee q$
UNSAT	$p \wedge \neg p$
valid	$p \vee \neg p$
semantically entails	$\neg p \models p \rightarrow q$
semantically equivalent	$p \rightarrow q$, $\neg p \vee q$
equisatisfiable	$p \wedge q$, $r \vee s$

Table 1.4: Some examples for the definitions.

Example. Consider the formulas $\varphi_1 : p \rightarrow (q \rightarrow r)$, $\varphi_2 : (p \wedge q) \rightarrow r$ and $\varphi_3 : (q \wedge \neg r) \rightarrow \neg p$. The three formulas φ_1, φ_2 and φ_3 are semantically equivalent. One way to check this is to construct the truth table.

On drawing the truth table for the above example, one would realise that it is laborious. Indeed, for a formula with n variables, the truth table has 2^n entries! So truth tables don't work for large formulas. We need a more systematic way to reason about the formulae. That leads us to proof rules...
But before that let us get a closure on the example at the beginning of the chapter. Let p_{ijk} represent the proposition 'course C_i is scheduled in slot S_k of day D_j'. We can encode the constraints using the encoding strategy used in tutorial 1 - problem 3. That is, by introducing extra variables that bound the sum for first few variables (sum of i is atmost j). Using this we can encode the constraints as : $\sum_{k=1}^{4} p_{ijk} \leq 1$, $\sum_{j=1}^{5} p_{ijk} \leq 1$, $\sum_{i=1}^{5} p_{ijk} \leq 1$, $\sum_{k=1}^{4} \sum_{i=1}^{5} p_{ijk} \leq 3$, $\sum_{k=1}^{4} \sum_{j=1}^{5} p_{ijk} \leq 3$ and $\neg\big(\sum_{k=1}^{4} \sum_{j=1}^{5} p_{ijk} \leq 2\big)$.

1.3 Proof Rules

After encoding a problem into propositional formula we would like to reason about the formula. Some of the properties of a formula that we are usually interested in are whether it is SAT, UNSAT or valid. We have already seen that truth tables do not scale well for large formulae. It is also not humanly possible to reason about large formulae modelling real-world systems. We need to delegate the task to computers. Hence, we need to make systematic rules that a computer can use to reason about the formulae. These are called as proof rules.
The overall idea is to convert a formula to a normal form (basically a standard form that will make reasoning easier - more about this later in the chapter) and use proof rules to check SAT etc.

Rules are represented as

$$\frac{\text{Premises}}{\text{Inferences}}\ \text{Connector}_{i/e}$$

- **Premise**: A premise is a formula that is assumed or is known to be true.

- **Inference**: The conclusion that is drawn from the premise(s).
- **Connector**: It is the logical operator over which the rule works. We use the subscript i (for introduction) if the connector and the premises are combined to get the inference. The subscript e (for elimination) is used when we eliminate the connector present in the premises to draw inference.

Example. Look at the following rule

$$\frac{\varphi_1 \wedge \varphi_2}{\varphi_1} \wedge_{e_1}$$

In the rule above $\varphi_1 \wedge \varphi_2$ is assumed (is premise). Informally, looking at $\wedge$'s truth table, we can infer that both φ_1 and φ_2 are true if $\varphi_1 \wedge \varphi_2$ is true, so φ_1 is an inference. Also, in this process we eliminate (remove) $\wedge$ so we call this AND-ELIMINATION or $\wedge_e$. For better clarity we call this rule $\wedge_{e_1}$ as φ_1 is kept in the inference even when both φ_1 and φ_2 could be kept in inference. If we use φ_2 in inference then the rule becomes $\wedge_{e_2}$.

Table 1.5 summarises the basic proof rules that we would like to include in our proof system.

Connector	**Introduction**	**Elimination**
$\wedge$	$\frac{\varphi_1 \quad \varphi_2}{\varphi_1 \wedge \varphi_2} \wedge_i$	$\frac{\varphi_1 \wedge \varphi_2}{\varphi_1} \wedge_{e_1}$ $\quad$ $\frac{\varphi_1 \wedge \varphi_2}{\varphi_2} \wedge_{e_2}$
$\vee$	$\frac{\varphi_1}{\varphi_1 \vee \varphi_2} \vee_{i_1}$ $\quad$ $\frac{\varphi_2}{\varphi_1 \vee \varphi_2} \vee_{i_2}$	$\frac{\varphi_1 \vee \varphi_2 \quad \varphi_1 \rightarrow \varphi_3 \quad \varphi_2 \rightarrow \varphi_3}{\varphi_3} \vee_e$

Table 1.5: Proof rules.