L02 Propositional Logic

* latex typesetting sample and demo on algorithms etc. link

Motivating examples of propositional logic

- Motivating question: what laws do our thoughts follow?
- Example of Parent:
    1. **If** Peter is the father of John **or** Peter is the mother of John, **then** Peter is a parent of John.
    2. Peter is the father of John
    3. By 1 and 2, Peter is a parent of John
- Map coloring
    - Map of texas and new mexico, color it with red and blue to make sure their color is different
        - c(t, r): the color of Texas is red (value: true or false)
        - c(t, b): …
        - c(nm, r):
        - c(nm, b): …
        - If c(t, r) is true then c(nm, r) is not true
        - If c(t, b) is true then c(nm, b) is not true
        - Only one of c(t, r) and c(t, b) is true
        - Only one of c(nm, r) and c(nm, b) is true.
- We may realize that we are “following some logic” or “doing some reasoning”. To formalize this “logic” or “reasoning,” we need to figure out key elements in the examples above.
    - Example of parent
        - (Syntax) Variables (Propositional Variable whose value is true or false)
            - F: Peter is the father of J
            - M: Peter is the mother of John
            - P: Peter is a parent of John
        - (Synatx) Sentences using variables and logical connectives (Propositions)
            - p1: (F or M) → P
            - p2: F
            - p3: P
        - (Semantics/reasoning) Note that the parent information now is represented as a set of sentences. They are informally read as F is true, P is true, and If (F or M) is true then P is true. Now we have a pretty explicit reasoning to obtain that P is true.
    - Observations
        - We seem to need to invent a **language** (with syntax and semantics) which can allow us to address our intuitive understanding of “logic” or

"reasoning". The first language we introduce is called **propositional language.**

Propositional Logic

- Propositional Logic Language (we abuse proposal logic and propositional logic language)
    - Syntax (of the language by which we can *represent* information (in our mind or in English), called *propositions*)

      E.g., in parent example
      p1: (F or M) → P
      p2: F
      p3: P

    - Semantics (of the language by which we can *reason* with the propositions)

      E.g., given the propositions
      p1: (F or M) → P
      p2: F
      p3: P
      We can say that p3 is a **consequence** of p1 and p2.

Syntax of propositional logic language

- Recall parent example
  p1: (F or M) → P
  p2: F
  p3: P
- Observations on sentences in this language (syntax)
    - "Basic propositions" without logical connectives: F, M, P, …
    - we have *logical connectives*: **or**, → , **and**, **if and only if**, …
    - We can combine them in any "Legal way":
        - F -> M
        - (F and (P or F)) → (M or F), …
    - Now can you write a math definition of a good sentence in this language? Note focus on defining "legal ways" to combine sentences.
        - Reminder: how they are related to arithmetic expressions?
            - Arithmetic expressions
                - "Basic expressions": 1, 2, x, y, …
                - "Expression connectives" (functions): +, -,
                - Expression (syntax) examples
                    - 1
                    - x
                    - x+1

- (x * (y-1))/(y-x) …
    - A number is an **arithmetic expression.**
    - A variable is an **arithmetic expression.**
    - If t1 is an arithmetic expression and t2 is an arithmetic expression, (*t1 op t2), op* is an operator (+, -, *, /), is an **arithmetic expression.**

      Prove ((1+2)+x) is an arithmetic expression.

- Write a definition for arithmetic expression.
    - Some of us did offer a few attempts, but they are not at the mathematically level we need.
    - Here is one we tried. Note we define some other concepts before defining arithmetic expression
      Def (arithmetic expression)
      1. An ***arithmetic constant*** is one of 1, 2, 3, …
      2. A ***variable*** is *x, y, z, …*
      3. An ***operator*** is +, -, /, *
      3. If t1 is an arithmetic expression and t2 is an arithmetic expression, *t1 op t2, op* is an operator, is an ***arithmetic expression*.**
        - How about 1+1 = 2? Is it defined by the above? How about x+y being 5+6 when x is substituted for by 5 and y by 6? Now we are talking about **semantics** of arithmetic expressions, i.e., we associate a value to a answer expression (whose variables will first be given some values). We will talk about this a bit later.

Learning skills: reading/writing definitions (informal)

A *Formal Definition* contains the following components

- ***Meta Variables*** - It can be defined as a symbol or string of symbols used to denote or refer to another object. (Note: *variables x* and *y* are not meta variables. But t1 and t2 are meta variables.)
- *Concepts to be defined* - the goal of the definition.
    - *Name* of the concept*,* for example, *arithmetic expression*.
    - *Parameters of the concept* - a concept usually has inputs (or parameters or given information). E.g., *arithmetic expression* has *t* (a “string”), that means *t* is an arithmetic expression. This is not (always) crisp. People may have different opinions.
- *Concepts defined before the definition* - a formal definition uses only previously defined concepts and *logical connectives* (e.g., if, and, …)
- Note also the typesetting of the definition: starts with **def** followed by the concept name to be defined, the concept name is in bold and italicized font where it is defined.

To use a formally defined concept/word, We need to **trust** and **literally/relentlessly follow** the definitions. (Typically when facing conflicting info, we start to doubt the defs and start to create our own versions - useful for INFORMAL defs but not formal ones.)

Syntax definitions (propositional Logic)

- Examples: propositions can be of the following form:
  P
  Q
  P -> Q
  (P -> Q) -> R
  ((P /\ (P -> Q)) -> Q)
  (P and Q) -> (P or R)
  …
  Can you try to write a formal definition of **proposition**?

- **Def (The language of propositional logic)** ***The language of propositional logic*** consists of the following symbols
  Connectives : $\wedge$, $\vee$, $\neg$, $\rightarrow$,$\longleftrightarrow$
  Parentheses :),(
  Propositional Variables (letters) : A, A1, A2, . . . , B, B1, B2, . . . , . . .
- ***Propositions*** **(**note the technique used: **inductive definition)**
  [go to book def 2.1]

  Definition 2.1 (Propositions):

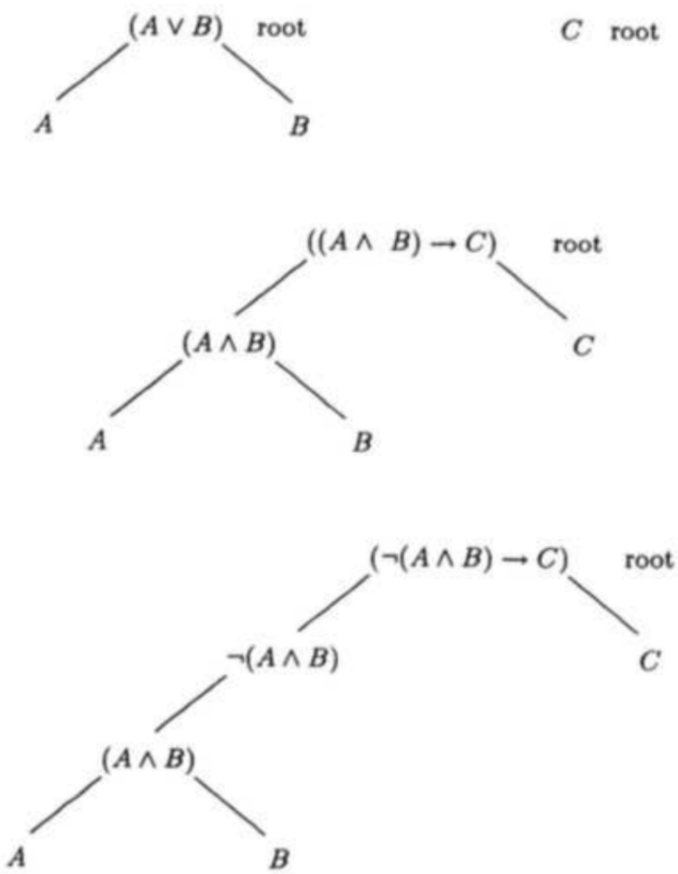
  (i) Propositional letters are propositions.

  (ii) If α and β are propositions, then $(\alpha \wedge \beta)$, $(\alpha \vee \beta)$, $(\neg\alpha)$, $(\alpha \rightarrow \beta)$ and $(\alpha \leftrightarrow \beta)$ are propositions.

  (iii) A string of symbols is a proposition if and only if it can be obtained by starting with propositional letters (i) and repeatedly applying (ii).

  Example: from defs above
  - figure out the *concepts* in the definitions above and t*heir parameters*.
    - “***The language of propositional logic***” no parameter!
    - “**proposition**” parameter?
  - *Meta variables***?**

- **Formation trees** (we just use intuitive understanding of it. You can study the formal definitions which needs definitions of trees in earlier part of the book.) A proposition has a structure (just like English sentence has a structure), and we use tree to visualize. The structure shows a “decomposition” of the proposition. **Problem Decomposition** is a problem solving method (bread and butter for computer science). For understanding properties of propositions, we need to understand its decomposition (or know its components) and thus apply the problem decomposition methods.



Formal proofs. Once we have definitions, we can provide a formal proof.
Check out [L03.pdf]

Semantics of propositional logic will be detailed in future lectures.