**Week 7 - Propositional Logic**

| Review Questions |
| --- |
| Q. How is a formula defined? (recursive definition)  Q. How is the value of a formula evaluated? (recursive definition)  Q. What is the satisfiability problem? |
| Q. Recursive definitions ( sub-formulas, size ) - parse tree  Q. Valid or Satisfiable?  Q. Proving by Counterexample |

| Types of Logic |
| --- |
| Propositional logic is about Interactions between propositions  proposition: statements that express concepts that are either true or false.  Predicate logic (first-order logic) deals with relations and functions of things  Introduces predicates and quantifiers to propositions. |

| Components of Logic |
| --- |
| Syntax: structure of expressions  Semantics: meaning of expressions  Deduction: syntactic mechanism for deriving new true expressions from existing statements. |

| Why care about syntax? |
| --- |
| Both humans and computer programs clearly understand the meaning of propositions.  Allows syntactical deduction |

| Standard symbols (syntax) |
| --- |
|  |

| Defining Formulas |
| --- |
| F : every atom is a formula  ¬F : negation of formula  (F ∧ G) , (F ∨ G) : conjunction, disjunction of formulas |

| Evaluating Formulas |
| --- |
| Assign truth values {0, 1} to atoms. (2^n assignments made)  Determine the value of the formula for different combinations of values atoms take. (refer to recursive procedure below)  Create a truth table [Truth Table Generator →](https://www.google.com/url?q=https://www.google.com/url?q%3Dhttps://web.stanford.edu/class/cs103/tools/truth-table-tool/%26sa%3DD%26source%3Deditors%26ust%3D1668792231190466%26usg%3DAOvVaw0iLGJ_ew_HvBP__QlOPAsy&sa=D&source=editors&ust=1668792259764837&usg=AOvVaw10qiHkwdCVw6SVT618-orB) |
| \*a = assignment function |

| Satisfiability Problem |
| --- |
| Currently, it takes 2^n assignments to determine if a formula is satisfiable (at the worst-scenario). Instead of the current (exponential & non-deterministic) approach, the satisfiability problem asks if one can create an algorithm to determine if a formula is satisfiable in a polynomial time?  If one can develop such an algorithm, the dreaded P vs NP (P = NP) problem will be ultimately solved, and it will most likely be remembered as one of the greatest discoveries in the field of (computer) science. |

| Q. Are the expressions propositional formulas? | |
| --- | --- |
| (p) | No |
| p ∧ q | Yes, outermost parentheses can be removed |
| (p ⊕ q) | No, because ⊕ is not one of the connective defined |
| (p → (q ∨ r)) | Yes, → is part of the extended definition |

| Q. Evaluate the following formulas | |
| --- | --- |
| (Smoke → Fire) → (¬Smoke → ¬Fire) | **Satisfiable**  True under Smoke = T and Fire = T  False under Smoke = F and Fire = T |
| ((Smoke ∧ Heat) → Fire) ↔   ((Smoke → Fire) ∨ (Heat → Fire)) | **Valid** |

| Q. Using satisfiability solver to determined validity | |
| --- | --- |
| F : Formula of Interest  Pass in ¬F to the solver.  If the solver returns false, conclude F is valid  If the solver returns false, the formula tested was never evaluated to be true under any assignment. This implies that the negation of the formula tested will be valid (true under every assignment) | |

| Q. Proving statements false - counterexample | |
| --- | --- |
| If F is valid and G is satisfiable, F → G is valid | Take F = (p ∨ ¬p) and G = q.  Then, F → G is false under the assignment p = 0, q = 0. |
| If F is satisfiable then ¬F is satisfiable. | Take F = ⊤ (verum)  Then ¬F = ⊥, which isn’t satisfiable |

| Q. Recursive definition - **SubF(G)** that returns set of G’s sub-formulas | |
| --- | --- |
| SubF(p) = {p} for atoms p.  SubF(¬F) = {¬F} ∪ SubF(F)  SubF(E ∨ F) = {E ∨ F} ∪ SubF(E) ∪ SubF(F)  SubF(E ∧ F) = {E ∧ F} ∪ SubF(E) ∪ SubF(F)  SubF(⊤) = {⊤} , SubF(⊥) = {⊥}  SubF(E → F) = {E → F} ∪ SubF(E) ∪ SubF(F)  SubF(E ↔ F) = {E ↔ F} ∪ SubF(E) ∪ SubF(F) | |
|  | |

| Q. Recursive definition - **|G|** that returns size of the formula G  size of a formula = number of symbols in G excluding parentheses   e.g. |(P ∨ ¬P)| = 4 | |
| --- | --- |
| |p| = 1 for atoms p  |¬F| = 1 + |F|  |F ∧ G| = 1 + |F| + |G|  |F ∨ G| = 1 + |F| + |G| | |