COMP1002 midterm exam study sheet

Propositional logic:

- *Propositional statement*: expression that has a truth value (true/false). It is a *tautology* if it is always true, *contradiction* if always false.

- *Logic connectives*: negation ("not") $\neg p$, conjunction ("and") $p \wedge q$, disjunction ("or") $p \vee q$, implication $p \rightarrow q$ (equivalent to $\neg p \vee q$), biconditional $p \leftrightarrow q$ (equivalent to $(p \rightarrow q) \wedge (q \rightarrow p)$). The order of precedence: $\neg$ strongest, $\wedge$ next, $\vee$ next, $\rightarrow$ and $\leftrightarrow$ the same, weakest.

- If $p \rightarrow q$ is an implication, then $\neg q \rightarrow \neg p$ is its *contrapositive*, $q \rightarrow p$ a *converse* and $\neg p \rightarrow \neg q$ an *inverse*. An implication is equivalent to its contrapositive, but not to converse/inverse or their negations. A negation of an implication $p \rightarrow q$ is $p \wedge \neg q$ (it is not an implication itself!)

- A *syntax tree* of a formula visually encodes its structure and the order of operations. Each occurrence of a variable or a logical connective is represented by a circle (a node in a tree). All variables are on the bottom of the tree. If an operation applied to subformula(s), the node for this operation is drawn above and connected to top nodes of subformulas to which this operation is applied.

- A *truth assignment* is a string of values of variables to the formula, usually a row with values of first several columns in the truth table (number of columns = number of variables). A truth assignment is *satisfying* the formula if the value of the formula on these variables is T, otherwise the truth assignment is *falsifying*. A formula is *satisfiable* if it has a satisfying assignment, otherwise it is *unsatisfiable* (a contradiction). A truth assignment can be encoded by a formula that is a $\wedge$ of variables and their negations, with negated variables in places that have F (false) in the assignment, and non-negated that have T (true). For example, $x = T, y = F, z = F$ is encoded as $(x \wedge \neg y \wedge \neg z)$.It is an encoding in a sense that this formula is true only on this truth assignment and nowhere else.

- A *truth table* has a line for each possible values of propositional variables (2^k lines if there are k variables), and a column for each variable and subformula, up to the whole statement.Its cells contain T and F depending whether the (sub)formula is true for the corresponding scenarios.

- Finding a method for checking if a formula has a satisfying assignment that is always significantly faster than using truth tables (that is, better than brute-force search) is a one of Clay Mathematics Institute $1,000,000 prize problems, known as "P vs. NP".

- Two formulas are *logically equivalent*, written $A \equiv B$, if they have the same truth value in all scenarios (truth assignments). $A \equiv B$ if and only if $A \leftrightarrow B$ is a tautology.

- There are several other important pairs of logically equivalent formulas, called *logical identities* or *logic laws*. We will talk more about them when we talk about Boolean algebras. The most famous example of logically equivalent formulas is $\neg(p \vee q) \equiv (\neg p \wedge \neg q)$ (with a dual version $\neg(p \wedge q) \equiv (\neg p \vee \neg q)$) where p and q can be arbitrary (propositional, here) formulas. These pairs of logically equivalent formulas are called *DeMorgan's law*. Here, remember that $FALSE \wedge p \equiv p \wedge \neg p \equiv FALSE$, $FALSE \vee p \equiv TRUE \wedge p \equiv p$ and $TRUE \vee p \equiv p \vee \neg p \equiv TRUE$.

- A set of logic connectives is called *functionally complete* if it is possible to make a formula with any truth table out of these connectives. For example, $\neg, \wedge$ is a complete set of connectives, and so is the

Sheffer's stroke | (where $p|q \equiv \neg(p \wedge q)$), also called NAND for "not-and". But $\vee, \wedge$ is not a complete set of connectives since then it is impossible to express a truth table line with a 0 when all variables are 1.

- An *argument* consists of several formulas called *premises* and a final formula called a *conclusion*. If we call premises $A_1 \dots A_n$ and conclusion B, then an argument is *valid* iff premises imply the conclusion for all assignments to their free variables, that is, $A_1 \wedge \dots \wedge A_n \rightarrow B$. We usually write them in the following format:

  Today is either Thursday or Friday
  On Thursdays I have to go to a lecture
  Today is not Friday (alternatively, On Friday I have to go to the lecture)
  ———————————————————————
  $\therefore$ I have to go to a lecture today

- A valid form of argument is called *rule of inference*. The most prominent such rule is called *modus ponens*.

  $p \rightarrow q$
  p ————————
  $\therefore q$

- We studied three methods for proving that a formula is a tautology: *truth tables*, *natural deduction* and *resolution* (where resolution proves that a formula is a tautology by proving that its negation is a contradiction).

- A *natural deduction proof* consists of a sequence of applications of modus ponens (and other rules of inference) until a desired conclusion is reached, or there is nothing new left to derive. Example: treasure hunt, where "desired conclusion" is a statement that the treasure is in a specific location.

- There are two main normal forms for the propositional formulas. One is called *Conjunctive normal form* (CNF, also known as Product-of-Sums) and is an $\wedge$ of $\vee$ of either variables or their negations (here, by $\wedge$ and $\vee$ we mean several formulas with $\wedge$ between each pair, as in $(\neg x \vee y \vee z) \wedge (\neg u \vee y) \wedge x$. A *literal* is a variable or its negation (x or $\neg x$, for example). A $\vee$ of (possibly more than 2) literals is called a *clause*, for example $(\neg u \vee z \vee x)$, so a CNF is true for some truth assignment whenever this assignment makes each of the clauses is true, that is, each clause has a literal that evaluates to true under this assignment. A *Disjunctive normal form* (DNF, Sum-of-Products) is like CNF except the roles of $\wedge$ and $\vee$ are reversed. A $\wedge$ of literals in a DNF is called a *term*.

- A CNF (DNF) is called *canonical* if it has a clause (respectively, term) for every falsifying (resp. satisfying) assignment. To construct canonical DNF and a CNF, start from a truth table and then for every satisfying truth assignment $\vee$ its encoding to a DNF, and for every falsifying truth assignment $\wedge$ the negation of its encoding to the CNF, and apply DeMorgan's law. This may result in a very large CNFs and DNFs, comparable to the size of the truth table itself ($2^{\text{number of variables}}$).

- A *resolution proof system* is used to find a contradiction in a formula (and, similarly, to prove that a formula is a tautology by finding a contradiction in its negation). Resolution starts with a formula

in a CNF form, and applies the rule "from clause $(C \vee x)$ and clause $(D \vee \neg x)$ derive clause $(C \vee D)$ until a falsity F (equivalently, empty clause ()) is reached (so in the last step one of the clauses being *resolved* contains just one variable and another clause being resolved contains just that variable's negation.) Note that if a clause has opposing literals (e.g., from resolving $(x \vee y)$ with $(\neg x \vee \neg y)$ then it evaluates to true, and so is useless for deriving a contradiction. Resolution can be used to check the validity of an argument by running it on the $\wedge$ of all premises (converted, each, to a CNF) $\wedge$ together with the negation of the conclusion.

- *Pigeonhole principle* If n pigeons sit in $n-1$ holes, so that each pigeon sits in some hole, then some hole has at least two pigeons. There is no small resolution proof of the pigeonhole principle.

- *Boolean functions* are functions which take as argument boolean (ie, propositional) variables and return 1 or 0 (or, the convention here is 1 instead of T, and 0 instead of F). Each Boolean function on n variables can be fully described by its truth table. A size of a truth table of a function on n variables is 2^n. Even though we often can have a smaller description of a function, vast majority of Boolean functions cannot be described by anything much smaller. Every Boolean function can be described by a CNF or DNF, using the above construction.

Predicate logic:

- A *predicate* is like a propositional variable, but with *free variables*, and can be true or false depending on the values of these free variables. A *domain* (universe) of a predicate is a set from which the free variables can take their values (e.g., the domain of $Even(n)$ can be integers). Some common domains are natural numbers $\mathbb{N}$ (here, $0 \in \mathbb{N}$), integers $\mathbb{Z}$, rationals $\mathbb{Q}$, reals $\mathbb{R}$, empty domain $\emptyset$, binary strings $\{0,1\}^*$.

- *Quantifiers* For a predicate $P(x)$, a quantified statement "for all" ("every", "all") $\forall x P(x)$ is true iff $P(x)$ is true for every value of x from the domain (also called universe); here, $\forall$ is called a *universal quantifier*. A statement "exists" ("some", "a") $\exists x P(x)$ is true whenever $P(x)$ is true for at least one element x in the universe; $\exists$ is an existential quantifier. The word "any" means sometimes $\exists$ and sometimes $\forall$. A domain (universe) of a quantifier, sometimes written as $\exists x \in D$ and $\forall x \in D$ is the set of values from which the possible choices for x are made. If the domain of a quantifier is empty, then if the quantifier is universal then the formula is true, and if quantifier is existential, false. A *scope* of a quantifier is a part of the formula (akin to a piece of code) on which the variable under that quantifier can be used (after the quantifier symbol/inside the parentheses/until there is another quantifier over a variable with the same name). A variable is *bound* if it is under a some quantifier symbol, otherwise it is free.

- *First-order formula* A predicate is a first-order formula (possibly with free variables). If A and B are first-order formulas, then so are $\neg A$, $A \wedge B$, $A \vee B$. If a formula $A(x)$ has a free variable (that is, a variable x that occurs in some predicates but does not occur under quantifiers such as $\forall x$ or $\exists x$), then $\forall x\ A(x)$ and $\exists x\ A(x)$ are also first-order formulas. A first-order formula is in *prenex form* when all variables have different names and all quantifiers are in front of the formula.

- More precisely, a *signature* is a list of names of predicates with their arities (as well as names and arities of functions on elements, if we have them); once we specified a signature, we can write first-order formulas in this signature. A *structure* of a given signature consists of a domain, and *interpretations* of all predicate and function symbols (an *interpretation* tells us the values predicates

and functions on elements of the domain). A *model* of a formula is an interpretation that makes this formula true.

Example: in our Tarski world, the signature consists of 5 unary predicates Circle(), Square(), Triangle(), Big(), Little(), and two binary predicates NextTo() and Aligned(). Each Tarski board is a structure of this signature, with the domain consisting of all pieces on the board, and interpretations of predicates reflecting what these pieces are and how they are positioned with respect to each other (for example, if the first piece, call it "a", is a triangle, then Triangle(a) would be true, and Circle(a), Square(a) would be false). A board which satisfies a given formula is a model of that formula.

- *Type checking*: there are different types of items and operations in a first-order formula: elements of the domains which occur only in inputs to predicates (and as variable names in quantifiers), sets (domains), and Booleans (returning true/false). All logical connectives take Booleans as inputs and return Booleans (so the whole formula evaluates to a Boolean). Predicates take elements as inputs and return a Boolean. Functions such as addition take elements as inputs and return elements. Finally, quantifiers take a name of a variable denoting an element, the name of the domain (a set), and a Boolean (ie a formula or a predicate) and returns a Boolean.

- The order of quantifiers in a formula matters (as well as the order of variables in a predicate). $\forall x \exists y P(x, y)$ is not the same as $\exists y \forall x P(x, y)$, since in the first formula for different values of x different values of y can be chosen, whereas the second formula is true if there is a single value of y which should work for all x. If you have done programming, it might help to think of nested quantifiers as nested "for" loops: in the first case, the inner loop is on the y's, and in the second, the inner loop is on the x's.

- To evaluate a formula with nested quantifiers, think of a game between two players: a universal player is trying to make the formula false, and existential player trying to make it true. They go left to right through the formula, with universal player suggesting counterexamples, and existential player suggesting witnesses; if after all quantifiers have been instantiated the resulting formula is true, the existential player wins, if it is false, universal player wins. Now, a formula is true iff there is a way for an existential player to win no matter what universal player's choices are.

- *Negating quantifiers.* Remember that $\neg \forall x P(x) \equiv \exists x \neg P(x)$ and $\neg \exists x P(x) \equiv \forall x \neg P(x)$. This is because $\forall$ is like a big $\wedge$ over all scenarios, and $\exists$ is an $\vee$.

- *Prenex normal form* In a first-order formula, it is possible to rename variables under quantifiers so that they all have different names. Then, after pushing negations into the formulas under the quantifiers, the quantifier symbols can be moved to the front of a formula (making their scope the whole formula).

- *Formulas with finite domains* If the domain of a formula is finite, a formula can be converted into a propositional formula by changing each $\forall x$ quantifier with a $\wedge$ of the formula on all possible values of x; an $\exists$ quantifier becomes a $\vee$. Then terms of the form $P(value)$ (e.g., $Even(5)$) are treated as propositional variables.