Intro to logic INF432: CheatSheet

Strict Formula (SF)
$\top$, $\bot$ and x are SF.
If A is a SF then $\neg A$ is a SF.
If A and B are strict formulae, then $(B \circ C)$ is a SF.

Priority formula (PF)
$\top$, $\bot$ and x are PF.
If A is a PF then $\neg A$ is a PF.
If A is a PF then (A) is a PF.
If A and B are PF then $A \circ B$ is a PF

Priority order of the connectors
By decreasing order: $\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow$

Connective precedence
Left precedence for $\vee, \wedge, \Leftrightarrow$
meaning $A \circ B \circ C = (A \circ B) \circ C$
⚠ Right precedence for $\Rightarrow$
meaning $A \Rightarrow B \Rightarrow C = A \Rightarrow (B \Rightarrow C)$

Length of a formula
The length of A is denoted $l(A)$. It represents the number of symbols used to write A.

Size of a formula
The size of A is denoted $|A|$.
$|\top| = |\bot| = |p| = 0$
$|\neg A| = 1 + |A|$
$|(A)| = |A|$
$|A \circ B| = 1 + |A| + |B|$

Assignment
An assignment v is a function from the set of variables of the formula into $\{0,\ 1\}$.
We can write it like this: $v =< p = 0,\ q = 1 >$.
$[A]_v$ denotes the truth value of the formula A for the assignment v.

Equivalence
Two formulae A and B are equivalent (denoted $A \equiv B$) if and only if they have the same value for every assignment.

Valid formula
A formula is a valid (denoted $\models A$) if and only if it has value 1 for every assignment.
A valid formula is a tautology.

Model for a formula
An assignment v such that $[A]_v = 1$ is a model for A.
An assignment v such that $[A]_v = 0$ is a counter-model for A.

Model for a set of formulae
An assignment v is a model for a set of formulae $\{A_1, \ldots, A_n\}$ if and only if it is a model for A_k for any $k \in \{1, \ldots, n\}$

Satisfiability
A (set of) formula(e) is satisfiable if it admits at least one model.
A (set of) formula(e) is unsatisfiable if it is not satisfiable.

In short
satisfiable → at least one model.
unsatisfiable → 0 model.
valid → 0 counter-model.
invalid → at least one counter-model.

Logical consequence (entailment)
The formula A is a consequence of the set of formulae Γ if and only if every model of Γ is a model of A.
It is denoted $A \models \Gamma$.

Important property
Let $A_1, \ldots, A_n, B$ be $n+1$ formulae. The three following formulations are equivalent:
(i): $A_1, \ldots, A_n \models B$
(ii): the formula $(A_1 \wedge A_2 \wedge \ldots \wedge A_n) \Rightarrow B$ is valid.
(iii): $(A_1 \wedge A_2 \wedge \ldots \wedge A_n) \Rightarrow \neg B$ is unsatisfiable.

Propositional compactness
A set of propositional formulae has a model if an only if every finite subset of it has a model.

Properties of disjunction and conjunction
Associativity: $(A \vee B) \vee C \equiv A \vee (B \vee C)$
Commutativity: $A \vee B \equiv B \vee A$
Idempotence: $A \vee A \equiv A$
The 3 properties above also hold for $\wedge$
$p \vee \neg p \equiv 1$; $p \vee 1 \equiv 1$; $p \vee 0 \equiv p$
$p \wedge \neg p \equiv 0$; $p \wedge 1 \equiv p$; $p \wedge 0 \equiv 0$

Distributivity
$\vee$ distributes over $\wedge$:
$p \vee (q \wedge r) \equiv (p \vee q) \wedge (p \vee r)$
$\wedge$ distributes over $\vee$:
$p \wedge (q \vee r) \equiv (p \wedge q) \vee (p \wedge r)$

De Morgan laws
$\neg(p \vee q) \equiv \neg p \wedge \neg q$; $\neg(p \wedge q) \equiv \neg p \vee \neg q$

Simplification laws
$p \vee (p \wedge q) \equiv p$
$p \wedge (p \vee q) \equiv p$
$p \vee (\neg p \wedge q) \equiv p \vee q$
$p \wedge (\neg p \vee q) \equiv p \wedge q$

Substitution
A substitution σ is a function mapping variables to formulae.
We denote by $A\sigma$ the formula A where all variables x are replaced by $\sigma(x)$.

Support of a substitution
The support of a substitution σ is the set of variables x such that $x\sigma \neq x$.
A finite support substitution σ is denoted $< x_1 := A_1, \ldots, x_n := A_n$

Property of a substitution
Let v be a truth assignment and σ a substitution.
Let w be the assignment $x \to [\sigma(x)]_v$
Then we have $[A\sigma]_v = [A]_w$ for any formula A.

Literal, monomial & clause
A literal is a variable or its negation.
A monomial is a conjunction of literals.
A clause is a disjunction of literals.

Normal form
A formula is in normal form if it contains only the operators $\vee, \wedge$ and $\neg$, and if $\neg$ is only applied to variables.

Theorem
Every formula admits an equivalent normal form

Computing a normal form
Step 1: equivalence elimination
replace $A \Leftrightarrow B$ by $(\neg A \vee B) \wedge (\neg B \vee A)$
or by $(A \wedge B) \vee (\neg A \wedge \neg B)$
Step 2: implication elimination
replace $A \Rightarrow B$ by $\neg A \vee B$
Step 3: shift negations towards variables
replace $\neg\neg A$ by A
replace $\neg(A \vee B)$ by $\neg A \wedge \neg B$
replace $\neg(A \wedge B)$ by $\neg A \vee \neg B$

Disjunctive normal form (DNF)
A formula is in DNF if and only if it is a disjunction of monomials.
To obtain a DNF, distribute the conjunctions over the disjunctions.
The DNF highlights the models of the formula.

Conjunctive normal form (CNF)
A formula is in CNF if and only if it is a conjunction of clauses.
To obtain a DNF, distribute the disjunctions over the conjunctions.
The DNF highlights the counter-models of the formula.

Boolean Algebra
A boolean algebra is a set of:
At least two elements, 0 and 1.
Three operations, complement ($\bar{x}$), sum ($+$)
and product ($\cdot$) which respect the following:
The sum is associative, commutative, with neutral element 0
The product is associative, commutative, with neutral element 1
The product is distributive over the sum
The sum is distributive over the product
The negation laws: $x + \bar{x} = 1$ and $x \cdot \bar{x} = 0$

Properties of a boolean algebra
$\forall x, \exists! y$ such that $x + y = 1$ and $x \cdot y = 0$
$\bar{1} = 0$; $\bar{0} = 1$; $\bar{\bar{x}} = x$; $x \cdot x = x + x = x$
$x + 1 = 1$; $x \cdot 0 = 0$
$x \bar{\cdot} y = \bar{x} + \bar{y}$; $x \bar{+} y = \bar{x} \cdot \bar{y}$

Boolean function
A boolean function is a function such that its arguments and result belong to the set $\mathbb{B} = \{0,\ 1\}$

Method: making a proof by induction
It is done in 4 steps:
Define the induction hypothesis (*hypothèse de récurrence*) $\mathcal{H}(n)$
Prove that the initial case is true (for instance prove that $\mathcal{H}(0)$ is true)
Prove that if $\mathcal{H}(k)$ is true for all $k \leq n$, then $\mathcal{H}(n+1)$ is true
Conclude

Example
Prove that any formula that uses only one variable x and the connectors $\vee$ and $\wedge$ is equivalent to a formula of size 0.
We will prove this statement by induction on the size of the formula.
The first thing is to have an intuition of why this is true. We know that $x \vee x = x \wedge x = x$, and that $|x| = 0$, so we will prove that these formulas are equivalent to x, to $\top$ or to $\bot$.

Step 1: define the induction hypothesis
With $n \geq 0$, let $\mathcal{H}(n)$ be: "Any formula of size n that uses only one variable x and the connectors $\vee$ and $\wedge$ is equivalent to a formula of size 0"

Step 2: Prove the initial case
Let A be a formula such that $|A| = 0$ and A uses only one variable x and the connectors $\vee$ and $\wedge$.
Then we have $A = A$, with $|A| = 0$, so $\mathcal{H}(0)$ is true.

Step 3: Prove that if $\mathcal{H}(k)$ is true for all $k \leq n$, then $\mathcal{H}(n+1)$ is true
Let $n \in \mathbb{N}^*$. We suppose that for all $k \leq n$, $\mathcal{H}(k)$ is true.
Let A be a formula such that $|A| = n + 1$ and A uses only one variable x and the connectors $\vee$ and $\wedge$.
Since $|A| > 0$, we have $A = B \vee C$ or $B \wedge C$.
If $A = B \vee C$, then $|A| = 1|B| + |C|$, so $|B| \leq n$ and $C \leq n$
By induction, we know that B and C are both equivalent to a formula of size 0, meaning $B \equiv \top$ or $B \equiv \bot$ or $B \equiv x$, and same for C).
Since we know that $\bot \vee x = x$, $\bot \wedge x = \bot$, $\top \vee x = \top$, $\top \wedge x = x$, $x \vee x = x \wedge x = x$, we can see that in all cases, we will have $A \equiv \top$ or $A \equiv \bot$ or $A \equiv x$.
So $\mathcal{H}(k)$ true for all $k \leq n \Rightarrow \mathcal{H}(n+1)$.

Step 4: Conclude
We have proved by induction that for all $n \in \mathbb{N}, \mathcal{H}(n)$ is true. So any formula that uses only one variable x and the connectors $\vee$ and $\wedge$ is equivalent to a formula of size 0

Vocabulary and Notations

A literal is a variable or its negation.
Given a clause A, we denote by $s(A)$ the set of its literals.
For instance, $s(\bar{q}+p+r+\bar{p}+r) = \{\bar{q}, p, r, \bar{p}\}$
Since we identify a clause by the set of its literals, we can say that:

- A literal is a *member* of a clause.
- A clause A is *included* in a clause B (if $s(A) \subseteq s(B)$)
- Two clauses A and B are equal (if $s(A) = s(B)$)

Given a literal L, we denote by L^c the negation of L
For instance, $x^c = \bar{x}$ and $\bar{x}^c = x$

Resolvent

Let A and B be two clauses.
The clause C is a *resolvent* of A and B if and only if there exists a literal L such that:

$$L \in A,\ L^c \in B, \quad \text{and } s(C) = (A \setminus \{L\}) \cup (B \setminus \{L^c\})$$

We denote C is a resolvent of A and B by $\frac{A \quad B}{C}$ We then say that C is *generated* by A and B, and that A and B are the parents of clause C.

Resolution Proof

Let Γ be a set of clauses and C a clause. A proof of C starting from Γ is a list of clauses such that:

- Every clause of the proof is either a member of Γ or a resolvent of two clauses already obtained
- Ending with C

We then say that the clause C is *deduced* from Γ (or alternatively, that Γ *yields* C, or that Γ *proves* C), denoted $\Gamma \models C$.
The size of the proof is the number of lines in it.

Resolution algorithms

We study two resolution algorithms in this course: the *complete strategy algorithm* and the *DPLL algorithm*.
In this cheatsheet, only the DPLL algorithm will be detailed, since it is basically an improvement on the complete strategy, and is better in every regards.

Reduction of a set of clauses

To *reduce* a set of clauses, we remove all the *valid clauses* and the clauses that contain another clause.
A valid clause is a clause containing a literal and its negation
A clause contains another clause if the set of it literals contains the set of literals of the other
The reduction phase preserves the satisfiability of the set of clauses.

DPLL algorithm

It is named after Davis, Putnam, Logemann and Loveland, who designed and refined it in the 60'.
It indicates whether a set of clauses is satisfiable or not, and exhibits a model in the former case.