

Chapitre I : Logique propositionnelle : syntaxe

Goulven GUILLOU

UBO – Département Informatique



1 / 11

La logique intervient dans plusieurs domaines informatiques :

- architecture : synthèse de circuits.
- intelligence artificielle (logique floue, systèmes experts, Prolog)
- sémantique et conception des langages de programmation (programmation fonctionnelle, logique de Hoare, lambda-calcul)
- vérification de systèmes informatiques (Coq, méthode B)
- synthèse de programmes obéissant à des spécifications données (isomorphisme de Curry-Howard, Coq).

2 / 11

En logique des propositions on manipule des expressions simples comme :

- "Il fait beau à Brest"
- "2 est plus grand que 3"
- "Pierre aime Marie"

3 / 11

On considère que de telles expressions ne peuvent prendre que deux valeurs, **Faux** (0) ou **Vrai** (1), dites **valeurs de vérité**.

Du coup, on remplace ces expressions par des lettres comme **p**, **q** ou **r** qu'on appelle **variables propositionnelles** et qui sont également des **propositions**.

4 / 11

La logique des propositions s'intéresse à la manière de combiner les propositions pour en construire de plus complexes.

Pour combiner on utilise des **connecteurs** : la négation (**non**), la conjonction (**et**), la disjonction (**ou**), l'implication (\implies), l'équivalence (\iff) ...

On peut généraliser la notion de connecteur à celle de **fonction booléenne**.

La négation peut être vue comme la fonction **non** à un seul paramètre :

$$\begin{aligned}\text{non}(0) &= 1 \\ \text{non}(1) &= 0\end{aligned}$$

Quelles sont les autres fonctions booléennes à une variable ?

On définit inductivement l'ensemble **Prop** des propositions.

Soit $\mathcal{A} = \{p, q, r, s, \dots\}$ un *alphabet* dont les lettres sont les **variables propositionnelles**, l'ensemble **Prop** est le plus petit ensemble construit par application d'un nombre fini de règles suivantes :

- Si $x \in \mathcal{A}$ alors $x \in \mathbf{Prop}$.
- Si $p \in \mathbf{Prop}$ alors $\bar{p} \in \mathbf{Prop}$.
- Si p et q sont dans **Prop** alors $(p + q)$ est dans **Prop**.
- Si p et q sont dans **Prop** alors $(p.q)$ est dans **Prop**.
- Si p et q sont dans **Prop** alors $(p \implies q)$ est dans **Prop**.
- Si p et q sont dans **Prop** alors $(p \iff q)$ est dans **Prop**.

Ainsi :

$$\begin{aligned}& p, q \\& \bar{p}, (q + r) \\& ((q + r) \implies s) \\& (\bar{p}.((q + r) \implies s))\end{aligned}$$

sont des propositions.

Alors que :

$$\begin{aligned}& \text{Pierre aime Marie} \\& (p + \implies q), (pq) \\& p + q), p + q \\& \overline{p + q}\end{aligned}$$

ne sont pas des propositions.

Pour éliminer un certain nombre de parenthèses :

- on utilise la priorité des opérateurs : \neg , $.$, $+$, \implies , \iff
 $A.\bar{B} + C \implies D.E$ se lit $((A.\bar{B}) + C) \implies (D.E)$
- on omet les parenthèses extérieures (s'il y en a).
 $(A + B)$ devient $A + B$
- quand il y a un seul connecteur, le parenthésage est par défaut à droite.
 $A \implies B \implies C$ correspond à $(A \implies (B \implies C))$

On rencontre de nombreux autres symboles pour les connecteurs. Nous présentons également ceux de **bddc** que nous utiliserons en TP :

Dénomination	Notations usuelles	Notations bddc
Variable	p	p
Négation	$\neg p, \bar{p}$	not p, -p
Disjonction	$p \vee q, p \cup q, p + q$	$p + q$, p or q
Conjonction	$p \wedge q, p \cap q, p.q, pq$	$p.q$, p and q
Implication	$p \implies q, p \longrightarrow q$	$p \Rightarrow q$
Equivalence	$p \iff q, p \longleftrightarrow q$	$p = q$

bddc est une calculette utilisable en ligne sous linux.

```
guillou> ./bddc
version 2.0e
Type "help" for informations
>>p+q;
p + q
>>p-q;
recovery on ','
error recovered
>>p+-q;
p + -q
>>(p+q).r;
p.r + q.r
>>
```