# I Sujet 0 CCP MPI

Rappelons les règles de déduction naturelle suivantes, où $A$ et $B$ sont des formules logiques et $\Gamma$ un ensemble de formules logiques quelconques :

$$\frac{}{\Gamma, A \vdash A}\ \text{AX} \qquad \frac{\Gamma \vdash \bot}{\Gamma \vdash A}\ \bot_e \qquad \frac{\Gamma, A \vdash B}{\Gamma \vdash A \to B}\ \to_i \qquad \frac{\Gamma \vdash A \qquad \Gamma \vdash A \to B}{\Gamma \vdash B}\ \to_e \qquad \frac{\Gamma, A \vdash \bot}{\Gamma \vdash \neg A}\ \neg_i \qquad \frac{\Gamma \vdash A \qquad \Gamma \vdash \neg A}{\Gamma \vdash \bot}\ \neg_e$$

1. Montrer que le séquent $\vdash \neg A \to (A \to \bot)$ est dérivable, en explicitant un arbre de preuve.
2. Montrer que le séquent $\vdash (A \to \bot) \to \neg A$ est dérivable, en explicitant un arbre de preuve.
3. Donner une règle correspondant à l'introduction du symbole $\wedge$ ainsi que deux règles correspondant à l'élimination du symbole $\wedge$. Montrer que le séquent $\vdash (\neg A \to (A \to \bot)) \wedge ((A \to \bot) \to \neg A)$ est dérivable.
4. On considère la formule $P = ((A \to B) \to A) \to A$ appelée loi de Peirce. Montrer que $\models P$, c'est-à-dire que $P$ est une tautologie.
5. Pour montrer que le séquent $\vdash P$ est dérivable, il est nécessaire d'utiliser la règle d'absurdité classique $\bot_c$ (ou une règle équivalente), ce que l'on fait ci-dessous (il n'y aura pas besoin de réutiliser cette règle). Terminer la dérivation du séquent $\vdash P$, dans laquelle on pose $\Gamma = \{(A \to B) \to A, \neg A\}$ :

$$\dfrac{\dfrac{\dfrac{\dfrac{?}{\Gamma \vdash A}\ ? \qquad \dfrac{}{\Gamma \vdash \neg A}\ \text{AX}}{\Gamma = (A \to B) \to A, \neg A \vdash \bot}\ \neg_i}{(A \to B) \to A) \vdash A}\ \bot_c}{\vdash ((A \to B) \to A) \to A}\ \to_i$$

# II Typage OCaml

On souhaite formaliser le typage OCaml. On notera $\Gamma \vdash e : \tau$ si l'expression OCaml $e$ est typée par le type $\tau$, où $\Gamma$ est un **environnement de typage**, c'est-à-dire une fonction donnant un type à chaque variable.

$$\frac{}{\Gamma \vdash \texttt{false} : \texttt{bool}} \qquad \frac{}{\Gamma \vdash \texttt{true} : \texttt{bool}} \qquad \frac{n \in \mathbb{N}}{\Gamma \vdash n : \texttt{int}}$$

$$\frac{\Gamma(x) = \tau}{\Gamma \vdash x : \tau} \qquad \frac{\Gamma, x : \sigma \vdash e : \tau}{\Gamma \vdash \texttt{fun}\ x \texttt{ -> } e : \sigma \texttt{ -> } \tau} \qquad \frac{\Gamma \vdash f : \sigma \texttt{ -> } \tau \quad \Gamma \vdash e : \sigma}{\Gamma \vdash f\ e : \tau}$$

1. Soit $\Gamma = \{$`f : a -> (b -> a)`, `g : b -> a`$\}$. Montrer que `fun x -> f (g x) x` est bien typé.
2. Quelles analogies peut-on faire entre le typage OCaml et la déduction naturelle ?
3. Montrer que `(fun f -> f 1 2)(fun x -> 3)` n'est pas typable.

On ajoute maintenant les tuples :

$$\frac{\Gamma \vdash e_1 : \tau_1 \quad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash (e_1, e_2) : \tau_1 * \tau_2}$$

On veut aussi ajouter des fonctions polymorphes.

4. En utilisant des quantificateurs, proposer des types pour `fst` et `snd`, et une règle d'élimination.
5. Montrer alors que `fst (42, true)` est bien typé.

# III Mines-Pont 2010

On appelle **variable booléenne** une variable qui ne peut prendre que les valeurs 0 (synonyme de faux) ou 1 (synonyme de vrai).
Si $x$ est une variable booléenne, on note $\overline{x}$ le complémenté (ou négation) de $x$ : $x$ vaut 1 si $x$ vaut 0 et $x$ vaut 0 si $x$ vaut 1.
On appelle **littéral** une variable booléenne ou son complémenté.
On représente la **disjonction** (« ou » logique ) par le symbole $\vee$ et la **conjonction** (« et » logique) par le symbole $\wedge$.
On appelle **clause** une disjonction de littéraux. De plus, il ne doit pas y avoir deux fois la même variable dans une clause.
On appelle **formule logique sous forme normale conjonctive** une conjonction de clauses.
On appelle **valuation** des variables d'une formule logique une application de l'ensemble de ces variables dans l'ensemble $\{0, 1\}$.
Une clause vaut 1 si au moins un de ses littéraux vaut 1 et 0 sinon. Une clause est dite **satisfaite** par une valuation des variables si elle vaut 1 pour cette valuation. Une formule logique sous forme normale conjonctive vaut 1 si toutes ses clauses valent 1 et 0 sinon. Une formule logique est dite **satisfaite** par une valuation des variables si elle vaut 1 pour cette valuation. Une formule logique est dite **satisfiable** s'il existe une valuation de ses variables qui la satisfait.

Étant donnée une formule logique $f$ sous forme normale conjonctive, on note dans ce problème **max($f$)** le nombre maximum de clauses de $f$ pouvant être satisfaites par une même valuation.
En notant $m$ le nombre de clauses de $f$, on remarque que $f$ est satisfiable si et seulement si $\max(f) = m$.

On considère la formule $f_1$ (sous forme normale conjonctive) dépendant des variables $x, y, z$ :

$$f_1 = (x \vee y \vee z) \wedge (\overline{x} \vee \overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{y}) \wedge (\overline{x} \vee \overline{z}) \wedge (x \vee \overline{y} \vee z)$$

1. Indiquer si $f_1$ est satisfiable ou non et, si elle est satisfiable, donner l'ensemble des solutions de $f_1$.

Une **instance de 3-SAT** est une formule logique sous forme normale conjonctive dont toutes les clauses contiennent 3 littéraux.

2. Déterminer une instance $f_2$ de 3-SAT non satisfiable et possédant exactement 8 clauses; indiquer $\max(f_2)$ en justifiant la réponse.

On considère une instance $f$ de 3-SAT définie sur $n$ variables. On note $V$ l'ensemble des $2^n$ valuations des variables de $f$.
Soit $val$ une valuation des $n$ variables. Si $C$ est une clause, on note $\varphi(C, val)$ la valeur de C pour la valuation $val$ et on note $\psi(f, val)$ le nombre de clauses de $f$ qui valent 1 pour la valuation $val$.
On a : $\psi(f, val) = \sum_{C \text{ clause de } f} \varphi(C, val)$ et $\max(f) = \max_{val \in V} \psi(f, val)$.

3. Soit $C$ une clause de $f$. Donner une expression simple de $\sum_{val \in V} \varphi(C, val)$, en fonction de $n$.

4. Soit $m$ le nombre de clauses dont $f$ est la conjonction.
   En considérant la somme $\sum_{C \text{ clause de } f} \sum_{val \in V} \varphi(C, val)$, donner en fonction de $m$ un minorant de $\max(f)$.

5. Donner le nombre minimum de clauses d'une instance de 3-SAT non satisfiable.

# IV Système complet

On définit les opérateurs NAND, NOR, XOR par leurs tables de vérité :

| $x$ | $y$ | $x\ NAND\ y$ | $x\ NOR\ y$ | $x\ XOR\ y$ |
|---|---|---|---|---|
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 0 |

On dit qu'un ensemble $S$ d'opérateurs logiques est **complet** si toute formule logique est équivalente à une formule qui n'utilise que des opérateurs dans $S$.

1. Exprimer NAND, NOR, XOR, à l'aide de $\vee$, $\wedge$, $\neg$.
2. Montrer que $\{\wedge, \neg\}$ est complet.
3. Montrer que $\{NAND\}$ est complet. (c'est pour cette raison que le NAND est très utilisé en électronique)
4. Montrer que $\{NOR\}$ est complet.
5. Montrer que $\{XOR\}$ n'est pas complet.