MP2I: Logique Propositionelle

Cours

- Définition des formules propositionnelles par induction avec $\wedge, \vee, \neg, \rightarrow, \leftrightarrow, \bot, \top, x \in \mathcal{V}$
- Valuation, valeur de vérité, table de vérité
- Transformer une formule en forme normale conjonctive, en forme normale disjonctive
- Le problème SAT et k-SAT
- L'algorithme de Quine
- L'opérator xor $\oplus$ et l'anneau $((\mathbb{Z}/2\mathbb{Z})^n, \oplus, \wedge)$ (HP)

Questions de Cours

- Rapeller l'algorithme de Quine
- Rapeller la table de vérité de chacun des opérateurs $\wedge, \vee, \neg, \rightarrow, \leftrightarrow$
- Pour chacune des formules suivantes, être capable de donner
  - La forme normale conjonctive,
  - La forme normale disjonctive,
  - La table de vérité,
  - Une valuation la satisfaisant si possible,
  - Une valuation la disprouvant si possible.

  Les formules sont: $A \rightarrow (A \wedge B); (B \vee A) \rightarrow B; A \leftrightarrow (\neg A \wedge B), A \rightarrow (C \rightarrow B); \neg(A \leftrightarrow \neg A)$

Questions à rajouter

- BDD & d-DNNF
- d-DNNF are DAG circuit with not only on leafs. d-DNNF are more succint than BDD (hard proof)
- Circuit language
- Weighted model counting
- Weighted model counting in d-DNNF can be done as and -> product, or -> sum
- Every BDD is a d-DNNF: a node A is a "if A then X else X'" so "A and X or not A and X'".
- Links with top down compilation and quine's algorithm: top down is branching on original program

Equivalence

1. Expliciter une bijection φ entre $[\![0; 2^n - 1]\!]$ et $\{\top, \bot\}^n$

On représente la table de vérité d'une formule F à n variables par un tableau de booléens (d'entiers) de longueur 2^n, tel que la i-ème valeur correspond à $\varphi(i)(F)$.

2. Proposer un code qui à une formule associe sa table de vérité.
3. Proposer un code qui teste si deux formules sont équivalentes. Quel est sa complexité ?

Retirer les constantes

On se donne le type suivant en OCaml d'un arbre de syntax d'une formule propositionnelle:

```
type form =
  | Var of string | Top | Bot
  | Not of form
  | And of form * form | Or of form * form | Implies of form * form
```

Ici `Top` représente $\top$, le vrai, et `Bot` représente $\bot$, le faux. Ce sont les deux "constantes".

1. Donner une tautologie et une antilogie n'utilisant pas $\top$ ou $\bot$.
2. Donner le code d'une fonction OCaml `val rem_cst: form -> form` qui renvoie une formule équivalente qui n'utilise aucune constante.

On cherche maintenant à *propager les constantes*, c'est-à-dire à effectuer cette transformation tout en diminuant la taille de la formule.

3. A quoi sont équivalent les formules $A \wedge \top$ et $A \wedge \bot$? Faire de même pour chaque opérateur.
4. En déduire une fonction OCaml `val propage_cst: form -> form` qui renvoie une formule équivalente en propageant les constantes. *On aura le droit de retourner $\top$ ou $\bot$ directement, mais ils ne devront pas apparaitre dans des sous-formules.*

Vers le 3-Fnc

On définit une formule logique en forme normale conjonctive par le type suivant:

```
type lit = Pos of int | Neg of int
type fnc = lit list list
```

Les variables propositionnelles sont indexées et représentées par des entiers.

1. Mettre sous la forme normale conjonctive la formule $\neg(X_1 \vee (\neg X_2 \wedge X_3))$
2. Donner le code OCaml d'une fonction `val new_var: fnc -> int` qui renvoie un nom de variable propositionnel non utilisé.
3. Donner le code OCaml d'une fonction `val to_3: fnc -> fnc` qui prend une formule en forme normale conjonctive et qui renvoie une formule équivalente ou chaque clause possède au plus 3 littéraux. On aura le droit d'introduire de nouvelles variables non utilisée initialement.

Push down not

On se donne le type suivant en OCaml d'un arbre de syntaxe d'une formule propositionnelle:

```
type form =
  | Var of string
  | Not of form
  | And of form * form | Or of form * form | Implies of form * form
```

On dit qu'une formule est *sans non intérieur* si les seuls non sont autour de variables propositionnelle.

1. Transformer la formule $\neg(A \vee \neg B) \wedge \neg(C \wedge \neg D)$ en une version sans non intérieur
2. Montrer que toute formule est équivalente à une formule sans non intérieur.
3. Donner un programme effectuant cette transformation.

Systèmes Complet de Connecteurs[1]

On dit qu'un système de connecteurs logiques est *complet* si toute formule propositionnelle est logiquement équivalente à une formule propositionnelle écrite uniquement à l'aide de ces connecteurs et de $\top, \bot$.

1. Justifier que l'ensemble $\{\wedge, \vee, \neg\}$ forme un système complet de connecteurs
2. Les ensembles suivants forment-ils un système complet de connecteurs ? Justifier.

$\{\wedge, \neg\}$ $\{\vee, \wedge\}$ $\{\rightarrow, \neg\}$ $\{\rightarrow\}$

On introduit deux nouveaux connecteurs logiques pour écrire des formules propositionnelles:

- Le « OU exclusif » (ou XOR), noté $\oplus$, définit par $\nu(\psi_1 \oplus \psi_2) = V$ si et seulement si $\nu(\psi_1) \neq \nu(\psi_2)$.
- Le « NON-ET » (ou NAND), noté $\uparrow$, définit par $\nu(\psi_1 \uparrow \psi_2) = V$ si et seulement si $\nu(\psi_1) = \nu(\psi_2) = F$.

3. L'ensemble $\{\oplus, \neg\}$ forme t'il un système complet de connecteur ?
4. Montrer que $\{\uparrow\}$ forme un système complet de connecteurs.

[1] Exercice de Maxime Bridoux

Formule pour la bi-partition

Soit G = (V, E) un graphe fini orienté. Pour deux sommets u et v, on définit la variable propositionnelle X_{uv} qui indique s'il y a une arête entre u et v ou non. Pour toute valuation λ, on pose $G_\lambda = (V, E_\lambda)$ le graphe tel que $(u, v) \in E_\lambda$ si et seulement si $\lambda(X_{uv}) = \top$ (est vrai).

Montrer qu'il existe une formule propositionnelle ψ telle que pour tout valuation λ, $\lambda(\psi) = \top$ si et seulement si G_λ est biparti.

Formule pour le centre d'un graphe

Soit G = (V, E) un graphe fini orienté. Pour deux sommets u et v, on définit la variable propositionnelle X_{uv} qui indique s'il y a une arête dirigée de u vers v ou non. On appelle centre un sommet de G tel que tout sommets de G soit à distance au plus 2 du centre. Pour toute valuation λ, on pose $G_\lambda = (V, E_\lambda)$ le graphe tel que $(u, v) \in E_\lambda$ si et seulement si $\lambda(X_{uv}) = \top$.

Montrer qu'il existe une formule propositionnelle ψ telle que pour tout valuation λ, $\lambda(\psi) = \top$ si et seulement si G_λ admet un centre.

Formules linéaire

On dit qu'une formule F de la logique propositionnelle est *linéaire* si chaque variable propositionnelle apparaît au plus une fois.

1. Montrer que si F est une formule propositionnelle linéaire qui n'utilise pas $\bot$, alors il existe une valuation μ telle que $\mu \vDash F$
2. Dans le cas ou F est linéaire, proposer un algorithme polynomial qui compte le nombre de valuations satisfesant F

Equivalences avec Sheffer

On se donne le type suivant en OCaml d'un arbre de syntaxe d'une formule propositionnelle:

```
type form =
  | Var of string
  | Not of form
  | And of form * form | Or of form * form | Implies of form * form
```

On pose l'opérateur de Sheffer $A \uparrow B := \neg(A \wedge B)$.

1. Montrer que $A \to B \equiv A \uparrow (B \uparrow B)$
2. Montrer que n'importe quel formule est équivalente à une formule n'utilisant que l'opérateur de Sheffer $\uparrow$

On pose le type suivant en OCaml:

```
type sheffer =
 | Sheff of sheffer * sheffer
 | SVar of string
```

3. Donner le code OCaml d'une fonction `val to_sheff: form -> sheffer` qui à une formule retourne une formule équivalente n'utilisant que l'opérateur de Sheffer.
4. Est-ce que le résultat est polynomial en la taille de l'entrée ? Si ce n'est pas le cas, proposer, en ajoutant de nouvelles variables non utilisées, une version polynomiale.

Equivalence avec ite[2]

Étant donné trois formules logiques φ, ψ, θ, on définit un opérateur ternaire f par

$$f(\varphi, \psi, \theta) := (\varphi \wedge \psi) \vee (\neg\varphi \wedge \theta)$$

1. Simplifier $f(\top, \psi, \theta)$ et $f(\bot, \psi, \theta)$ en donnant des formules équivalentes.
2. Exprimer en $\varphi \wedge \psi$ et $\varphi \vee \psi$ via une formule ne comportant qu'un seul f.
3. Exprimer $\neg\varphi$ en fonction de $f, \top, \bot$.

[2]Exercice de Maxime Bridoux

Indéterminé

On définit une sémantique non standard sur les opérateurs logique écrits avec $\wedge$, $\vee$, $\rightarrow$ et $\neg$ pour un ensemble de 3 valeurs: $\top$, $\bot$ et ?.

? représente une valeur que l'on ne connais pas encore.

Pour v une valuation et F une formule, on a:

- $v(F) = \top$ si, quel que soit F' qui est F où l'on a remplacé chaque ? par des valeurs $\top$ ou $\bot$ de manière indépendante, on a $v(F') = \top$
- $v(F) = \bot$ si, quel que soit F' qui est F où l'on a remplacé chaque ? par des valeurs $\top$ ou $\bot$ de manière indépendante, on a $v(F') = \bot$
- $v(F) =$? sinon

Ainsi, par exemple, on a :

- $\top \vee ? \equiv \top$ car $\top \vee \bot \equiv \top \vee \top \equiv \top$
- $? \rightarrow ? \equiv ?$ car $\top \rightarrow \top \equiv \top$ mais $\top \rightarrow \bot \equiv \bot$

1. Proposer des tables de vérité pour $\wedge$, $\rightarrow$ et $\neg$
2. Montrer que $p \vee \neg p$ n'est pas une tautologie pour cette sémantique. Est-ce qu'une tautologie existe?
3. Proposer un algorithme qui prend une formule indéterminée et une valuation et qui donne la valeur d'une formule indéterminée. Quelle est sa complexité?

Formules monotones

Une formule est monotone si elle ne contient pas de négation ni d'implications. On considère alors le type suivant en OCaml :

```
type prop_mono =
| Top | Bot
| Var of int
| Or of prop_mono * prop_mono
| And of prop_mono * prop_mono
```

On définit une relation $\preceq$ sur les valeures de vérité V, F par $F \prec V$ que l'ont étent aux valuations par $\mu \preceq \mu'$ si pour toute variable X on a $\mu(X) \preceq \mu'(X)$.

1. Montrer que $\preceq$ est une relation d'ordre.
2. Montrer que si une formule monotone est satisfaite par une valuation v, alors elle est satisfaite par toute valuation w telle que $v \preceq w$.
3. Est-ce que cela marche toujours si on rajoute les implications?
4. En déduire un algorithme en ocaml `val is_sat : prop_mono -> bool` pour tester si une formule monotone est satisfiable.
5. En déduire un algorithme en ocaml `val is_taut : prop_mono -> bool` pour tester si une formule monotone est une tautologie.

Transformation de Tseitin

On souhaite transformer une formule quelconque en une FNC satisfiable si et seulement si la formule de départ est satisfiable, de taille polynomiale.

```
type lit = Pos of int | Neg of int
type cnf = lit list list

type prop =
| Top | Bot
| Var of int
| Not of prop
| Or of prop * prop
| And of prop * prop
| Imp of prop * prop
```

Une clause est représentée par une liste de littéraux, et une FNC par une liste de clauses. Pour chaque sous-formule φ de F, on introduit une nouvelle variable X_φ. On cherche à imposer localement que

$$X_\varphi \leftrightarrow \varphi$$

1. Écrire une fonction `val max_var : prop -> int` qui renvoie le plus grand indice de variable apparaissant dans une formule. S'il n'y a aucune variable, on pourra renvoyer -1.
2. Pour trois variables propositionnelles z, a, b, transformer les formules suivantes en une FNC avec des clauses de taille au plus 3 :

$$z \leftrightarrow a \wedge b \qquad z \leftrightarrow a \vee b \qquad z \leftrightarrow \neg a \qquad z \leftrightarrow (a \to b)$$

3. Proposer un algorithme `val to_cnf : prop -> cnf` qui construit une FNC équisatisfiable à la formule donnée.
4. Montrer que la formule de sortie est de taille polynomiale en la taille de la formule d'entrée.

Formules duales

Étant donné une formule de la logique propositionnelle F, utilisant les symboles $\top, \bot, \neg, \wedge, \vee$ et sur l'ensemble de variables X. On définit F^* la formule *duale* obtenue en remplaçant les $\wedge$ par des $\vee$, les $\vee$ par des $\wedge$ et en inversant les $\top$ et $\bot$.

1. Montrer que si deux formules sont équivalentes, alors leurs duales le sont aussi.
2. En déduire que si une formule est une tautologie, sa formule duale est contradictoire.

XOR SAT

On ajoute à la syntaxe de la logique propositionnelle l'opérateur binaire XOR, noté $\oplus$, ayant pour sémantique $v \vDash \psi_1 \oplus \psi_2$ si et seulement si $v(\psi_1) \neq v(\psi_2)$.

1. Exprimer l'opérateur $\oplus$ en fonction des opérateurs classique
2. Donner un algorithme polynomial qui, étant donnée une formule de la forme $\psi = \bigwedge_{i=1}^{m} \bigoplus_{j=1}^{n} l_i^{(j)}$ (avec $l_i^{(j)}$ des littéraux de la forme $X, \neg X$ pour X une variable) décide si ψ est satifiable

Interpolation de Craig[3]

On note $\mathrm{Var}(\varphi)$ l'ensemble des variables propositionnelles apparaissant dans la formule φ. Étant donné deux formules propositionnelles φ, ψ telles que $\varphi \vDash \psi$, un *interpolant* est une formule propositionnelle θ telle que:

- $\varphi \vDash \theta$
- $\theta \vDash \psi$
- $\mathrm{Var}(\theta) \subseteq \mathrm{Var}(\varphi) \cap \mathrm{Var}(\psi)$

1. Donner un interpolant de $\varphi = (x \vee y) \wedge z$ et $\psi = (t \to x) \vee (t \to y)$
2. Soient φ, ψ deux formules propositionnelles et $X \in \mathrm{Var}(\varphi)$, on note $\varphi[X \leftarrow \psi]$ la formule où l'on a remplacé toutes les variables propositionnelles X par ψ. On pose $\varphi_1 := \varphi[X \leftarrow \psi]$ et $F_2 := \varphi[X \leftarrow \neg\psi]$. Montrer que $\varphi \to \varphi_1 \vee \varphi_2$ est une tautologie.

Soient φ, ψ deux formules propositionnelles telles que $\varphi \vDash \psi$.

3. Montrer que si $\mathrm{Var}(\varphi) \cap \mathrm{Var}(\psi) = \emptyset$, alors soit $\neg\varphi$ soit ψ est une tautologie.
4. Montrer que toutes formules propositionnelles φ, ψ vérifiants $\varphi \vDash \psi$ admettent un interpolant.

---

[3] Un mix d'un exercice de Maxime Bridoux et d'un exercice du cours de logique de l'ENS de Lyon

FNC-SAT vers CLIQUE

On cherche à montrer que 3-SAT est résoluble en temps polynomial si et seulement si le problème CLIQUE l'est.

Soit $F = \bigwedge_{1 \leq i \leq n} C_i$ une formule en FNC possédant n clauses, que l'on écrit $C_i = l_i^{(1)} \vee ... \vee l_i^{(k_i)}$. On construit un graphe G_F de la manière suivante :

- pour chaque clause C_i et chaque occurrence d'un littéral $l_i^{(j)}$, on crée un sommet (i, j)
- on met une arête entre (i, j) et (i', j') si $i \neq i'$ et si les deux littéraux correspondants ne sont pas contradictoires, c'est-à-dire si $l_i^{(j)}$ n'est pas la négation de $l_{i'}^{(j')}$.

1. Dessiner le graphe correspondant à la formule

$$(x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_4 \vee x_3) \wedge (\neg x_3 \vee x_1 \vee x_2).$$

Une *clique* $S' \subseteq S$ d'un graphe $G = (S, A)$ est un ensemble de sommets deux à deux reliés par une arête.

2. Montrer que si F est satisfiable par une valuation μ, alors on peut choisir un littéral vrai dans chaque clause de F de sorte que les sommets correspondants forment une clique de taille n dans G_F
3. Montrer que G_F possède une clique de taille n si et seulement si F est satisfiable.
4. En déduire que s'il existe un algorithme polynomial pour le problème CLIQUE, alors FNC-SAT est résoluble en temps polynomial.

On cherche maintenant à montrer la réciproque.

Soit $G = (S, A)$ un graphe non orienté et soit $k \in \mathbb{N}$. On veut décider si G possède une clique S de taille k. Pour cela, pour tout $p \in \{1, ..., k\}$ et tout sommet $s \in S$, on crée une variable propositionnelle $X_{p,s}$, dont l'interprétation est :

$$X_{p,s} = \text{le p-ième sommet choisi est s.}$$

5. Pour chacune des contraintes suivantes, proposer une formule FNC qui dépend seulement du graphe :
   - pour chaque position p, au moins un sommet est choisi
   - pour chaque position p, au plus un sommet est choisi
   - deux positions différentes ne choisissent pas le même sommet
   - deux sommets choisis à deux positions différentes sont toujours reliés par une arête.
6. Montrer que F_G est satisfiable si et seulement si G possède une clique de taille k.
7. Expliquer pourquoi F_G est de taille polynomiale en $|S|$ et k.
9. Conclure que FNC-SAT est résoluble en temps polynomial si et seulement si CLIQUE l'est.

Binary Decision Diagram

Soit $n \in \mathbb{N}^*$. On considère des variables booléennes $X_1, ..., X_n$.

Un *BDD ordonné* est un graphe orienté acyclique dont les feuilles sont étiquetées par `true` ou `false`, et dont les noeuds internes sont étiquetés par une variable. Depuis un noeud étiqueté par X_i, l'arête gauche correspond à $X_i = F$ et l'arête droite à $X_i = V$. On impose que, sur tout chemin de la racine vers une feuille, les variables apparaissent dans le meme ordre $X_1, ..., X_n$. La racine sera toujours 1. On signale que ce n'est pas toujours un arbre : il est possible que deux noeud différents pointent vers le meme fils.

On propose la représentation OCaml suivante :

```
type node =
  | Bot (* feuille étiqueté par false *)
  | Top (* feuille étiqueté par true *)
  | If of int * int * int
  (* If(i,j,k) veux dire "si Xi = V alors aller au noeud j sinon aller au noeud
k" *)

type bdd = {
  n : int;
  root : int;
  nodes : node array;
}
```

1. Desiner le BDD associé à la formule $(x \wedge y) \vee (\neg y \vee z)$ pour l'ordre y, x, z.
2. Écrire une fonction `val eval : bool array -> bdd -> bool` qui teste si une valuation `val mu : bool array` satisfait le BDD.
3. Deux BDD B_1 et B_2 ayant le même ordre de variables étant donnés, expliquer comment construire un BDD représentant $B_1 \vee B_2$. On ne donnera pas l'algorithme en OCaml.
4. Montrer qu'il existe des BDD de taille polynomiale reconnaissant des formules de taille exponentielle.
5. Pour $n \geq 1$, on considère la formule suivante :

$$E_n := \bigwedge_{i=1}^{n} (X_i \leftrightarrow X_{i+n})$$

Montrer que toute représentation par BDD ordonné de E_n pour l'ordre $X_1 < ... < X_{2n}$ possède une taille exponentielle en n.

6. Montrer que l'on peut calculer en temps polynomiale en la taille du BDD le nombre de valuations satisfaisantes.

On peut utilliser les BDD pour calculer la probabilité qu'une formule soit vrai si on prend une valuation aléatoire uniformément.

Horn-SAT

On fixe $\mathcal{V} = \{x_1, ..., x_n\}$ un ensemble de variables propositionnelles. Une *clause de Horn* est une clause contenant au plus un littéral positif (qui est directement une variable sans négation).

1. Soit C une clause de Horn, montrer que C est équivalente à une formule de la forme suivante, avec $y_1, ..., y_k \in \mathcal{V}$ et $z \in \{\bot\} \cup \mathcal{V}$:

$$y_1 \wedge ... \wedge y_k \rightarrow z$$

On considère l'algorithme suivant :

- Convertir chaque clause comme vu à la question précédente
- Initialement, toutes les variables sont fausses
- Tant qu'il existe une implication $x_1 \wedge ... \wedge x_k \rightarrow y$ dont toutes les hypothèses sont vraies mais dont la conclusion y est fausse, on rend y vraie
- Si à un moment une implication $x_1 \wedge ... \wedge x_k \rightarrow \bot$ a toutes ses hypothèses vraies, on rejette
- Sinon, quand plus rien ne change, on accepte.

2. Exécuter l'algorithme sur la formule $F = x \wedge (\neg x \vee y) \wedge (\neg y \vee \neg z) \wedge (\neg x \vee \neg y \vee z)$.
3. Montrer que l'algorithme termine. Quelle est sa complexité?
4. Montrer la correction de l'algorithme.
5. En déduire que tester si une formule de Horn est satisfiable est décidable en temps polynomial.
6. Montrer que l'ensemble des valuations satisfaisant une formule de Horn est stable par intersection.

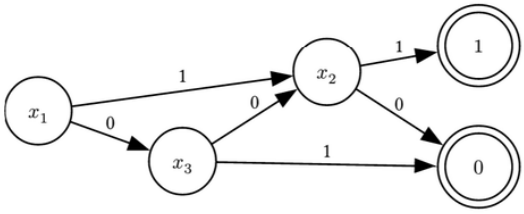
Formules Evasives[4]

On fixe un ensemble $X = \{x_1, ..., x_n\}$ de variables propositionnelle. Un *diagramme de décision* D sur X est la donnée d'un graphe orienté (V, E), supposé sans cycle, d'un nœud initial v_{init} et d'un ensemble d'états finaux $F = F_1 \sqcup F_0 \subseteq V$ non vide ne possédant aucune arête sortante.

Chaque arrête est étiquetée par un booléen et chaque nœud par une variable de X. Chaque nœud non final possède exactement deux arêtes sortantes, une étiquetée par 1 et une par 0.

Pour toute *valuation* $\mu : X \longrightarrow \{1, 0\}$, un diagramme de décision D associe une valeur $D(\mu) \in \{1, 0\}$ obtenus en parcourant le graphe en commençant à v_{init} et à chaque sommet v, prendre l'arête étiqueté par $\mu(v)$, jusqu'à arriver dans F_1 (dans ce cas $D(\mu) = 1$) ou arriver dans F_0 (dans ce cas, $D(\mu) = 0$)

1. On considère $X = \{x_1, ..., x_n\}$ et le graphe de décision D_1 suivant ou l'état initial est x_1, les états finaux ont été entouré (F_1 correspond aux état avec un 1 et respectivement F_0 correspond au états avec un 0):



À quelle valeur est associée la valuation qui envoie (x_1, x_2, x_3, x_4) sur $(1, 1, 0, 1)$? sur $(0, 1, 0, 0)$?

2. Donner une formule de la logique propositionnelle équivalente à la fonction booléenne représentée par D_1.

Si on se donne une formule logique φ, on dit que D *représente* φ si φ est équivalente à la fonction booléenne représentée par D.

3. Donner un diagramme de décision D_2 représentant la fonction $\neg(x_1 \wedge x_2) \vee x_3$

La profondeur d'un diagramme de décision est la plus grande longueur possible d'un chemin orienté à partir du nœud initial v_{init}, en comptant le nombre de nœuds de traversés. On appelle profondeur minimale d'une fonction booléenne φ la plus petite profondeur possible pour un diagramme de décision représentant φ.

4. Décrire la profondeur du diagramme D_2 et celle du diagramme D_1.
5. Quelle est la profondeur minimale de la fonction identifiée en question 1?

Une fonction booléenne φ sur n variables est dite évasive si sa profondeur minimale est de n.

6. Donner un exemple d'une famille infinie de fonctions évasives, et justifier.
7. Pour tout $n \in \mathbb{N}$, on pose $\varphi_n := (x_0 \wedge x_1) \vee (x_1 \wedge x_2) \vee ... \vee (x_{n-1} \wedge x_n)$. Ces fonctions sont-elles évasives ? Justifier.

[4]Sujet d'ENS

Compacité

Pour A un ensemble de formules de la logique propositionnelle, on dit qu'une valuation μ *satifiait* A si $\forall F \in A, \mu(F) = \top$

On dit que A est *finement satifiable* si pour tout $E \subseteq A$ fini, E est satifiable.

On pose $(X_n)_n$ une suite de variables propositionnelle.

1. Les ensembles suivants sont-ils satisfiables ?

- $A_1 = \{X_{2n} : n \in \mathbb{N}\} \cup \{\neg X_{2n+1} : n \in \mathbb{N}\}$
- $A_2 = \{X_i \vee \neg X_{i+1} : i \in \mathbb{N}\}$
- $A_3 = \{X_i \wedge \neg X_{i+1} : i \in \mathbb{N}\}$

On cherche à montrer le *théorème de compacité*: A est satisfiable ssi A est finement satifiable.

2. Montrer que si A est satifiable alors il est finement satifiable.
3. ($*$) Dans le cas ou l'ensemble des variables propositonnels de A est dénombrable, démontrer en construisant par récurrence une valuation que si A est finement satifiable alors A est satifiable.
4. Utilliser le théorème de compacité pour montrer qu'un graphe infini dénombrable est N coloriable ssi tout ses sous-graphes fini sont N coloriable.