# DS MPSI : Transformation de Tseitin & satisfiabilité

18 juin 2025

*Solution.* □

Ce sujet est décomposé en trois parties. La partie III est largement indépendante des deux premières.

Si $E$ est un ensemble fini, on note $|E|$ son cardinal.

On se donne un ensemble dénombrable $\mathcal{V}$ de variables propositionnelles $x_0, x_1, \ldots, x_n, \ldots$, deux constantes `T` et `F` (true et false), deux symboles de parenthèses `(` et `)`, deux connecteurs binaires $\vee, \wedge$ (ou,et) et un connecteur unaire $\neg$ (non). On définit inductivement l'ensemble des propositions comme dans le cours. Il est alors naturel de considérer les propositions comme des arbres binaire non entier dont les feuilles sont les variables et les nœuds internes sont les connecteurs.

Un *contexte* $\mu$ est une application des variables vers 0 ou 1. Chaque variable possède une image par $\mu$ mais, le plus souvent, on ne s'intéresse qu'aux variables présentes dans la proposition étudiée, ignorant les autres. L'*interprétation selon* $\mu$, notée $\varepsilon_\mu$, est un prolongement de $\mu$ aux propositions comme dans le cours. Deux propositions $P, Q$ sont dites *sémantiquement équivalentes* (ou plus simplement *équivalentes*) si pour tout contexte $\mu$, $\varepsilon_\mu(P) = \varepsilon_\mu(Q)$. On note alors $P \equiv Q$.

On dit que la proposition $P$ est *satisfiable* lorsqu'il existe un contexte $\mu$ tel que $\varepsilon_\mu(P) = 1$. On dit alors que $\mu$ *satisfait* $P$ ou encore que c'est un *modèle* de $P$ et on note ce fait $\mu \vDash P$. Une proposition qui n'est pas satisfiable est dite *insatisfiable*. Une proposition est appelée une *tautologie*, si tous les contextes la rendent vraie (c.a.d. l'évaluent en 1).

On appelle *contexte partiel* une application définie seulement sur une partie des variables. Un contexte partiel peut facilement être prolongé en un contexte : chaque variable sans image dans le contexte partiel prend la valeur (par exemple) 1 dans le contexte prolongé.

Pour deux propositions $P, Q$, on dit que $Q$ *est une conséquence* de $P$, et on note $P \vDash Q$ si et seulement si tout contexte qui satisfait $P$ satisfait $Q$.

On dit que deux propositions $P, Q$ sont *équivalentes en satisfiabilité* ou encore *équisatisfiables* lorsque les deux sont satisfiables ou aucune des deux ne l'est. On note alors $P \sim Q$. Par exemple $x \wedge \neg y$ est équivalente en satisfiabilité avec $y \wedge \neg x$ puisqu'il existe un contexte qui satisfait la première propositions et un contexte qui satisfait la seconde. En revanche $x \wedge \neg x$ et $x \wedge y$ ne sont pas équisatisfiables.

On rappelle :

— qu'un *littéral* est une variable ou une négation de variable,

— qu'une *clause* est une disjonction de littéraux (avec la convention qu'une disjonction sans littéral est `F` et une disjonction d'un littéral est ce littéral)

— et une *forme normale conjonctive* (FNC) est une conjonction de clause (avec la convention qu'une conjonction sans littéral est `V` et qu'une conjonction d'un littéral est ce littéral.

On a vu en cours que toute proposition est sémantiquement équivalente à une FNC.

Si $\ell$ est un littéral, on appelle *variable de* $\ell$ et on note $v(\ell)$ l'unique variable telle que $\ell$ est égal à cette variable ou sa négation. Par exemple, si $\ell = x_0$, alors $v(\ell) = x_0$ et si $\ell = \neg x_0$ alors $v(\ell) = x_0$.

Soit $P$ une proposition. On note $N_{\text{leaf}}(P)$ le nombre de feuilles, c'est à dire de variables. Chaque variable peut être comptée plusieurs fois. Par exemple, $N_{\text{leaf}}((x \vee \neg x) \wedge x)) = 3$. Le nombre de connecteurs de $P$ est noté $N_{\text{ni}}(P)$. La *taille* de $P$, notée $|P|$ est la taille de l'arbre correspondant : $|P| = N_{\text{leaf}}(P) + N_{\text{ni}}(P)$. Avec $P = N_{\text{leaf}}((x \vee \neg x) \wedge x)) = 3$, $|P| = 6$ et $N_{\text{ni}}(P) = 3$.

Si $\mathcal{E} = \{\ell_1, \ell_2, \ldots, \ell_n\}$ est un ensemble de $n$ littéraux, prendre $N$ *variables fraîches (par rapport à* $\mathcal{E}$*)* consiste à choisir $N$ variables distinctes qui n'apparaissent pas dans les $\ell_k$. C'est toujours possible puisque l'ensemble des variables est infini.

## Partie I Généralités

### Question 1.

Dans notre langage des propositions, il n'y a que des variables, des $\wedge$, des $\neg$ et des $\vee$.

1. Comment exprimer le vrai et le faux ?
2. Comment exprimer $P \to Q$ (implication) ?
3. Comment exprimer $P \leftrightarrow Q$ (équivalence) ?

*Solution.* $x \vee \neg x$, $x \wedge \neg x$, $\neg P \vee Q$ et enfin

$$(\neg P \vee Q) \wedge (\neg Q \vee P)$$

□

### Partie I.1 Équisatisfiabilité

**Question 2.**

Soient $P, Q, P', Q'$ trois propositions telles que $P \equiv P'$,$Q \equiv Q'$ et $P \vDash Q$. A-t-on $P' \vDash Q'$ ?

*Solution.* Oui. pour un contexte $\mu$ :

$$\begin{aligned} \varepsilon_\mu(Q') &= \varepsilon_\mu(Q) \\ &\geq \varepsilon_\mu(P) \\ &= \varepsilon_\mu(P') \end{aligned}$$

□

**Question 3.**

Soient deux propositions $P, Q$.

1. Si $P \equiv Q$, a-t-on $P \sim Q$ ?
2. Si $P \sim Q$, a-t-on $P \equiv Q$ ?

*Solution.* Si $P \equiv Q$.

Si $P$ est satisfiable, alors $Q$ l'est donc $P \sim Q$. Mais si $P$ n'est pas satisfiable (donc si $P$ est une antilogie comme $x \wedge \neg x$ avec $x$ une variable) on a bien $x \wedge \neg x \vDash \underbrace{x \vee \neg x}_{=Q}$. Et on n'a pas $P \sim Q$ puisque l'une est satisfiable et pas l'autre.

Avec $P = x, Q = y$, $x, y$ deux variables, on $P \sim Q$ et pourtant $P \not\equiv Q$ □

**Question 4.**

Soient 4 propositions $P, Q, P', Q'$ telles que $P \vDash Q$ et $P \sim P'$, $Q \sim Q'$. A-t-on $P' \vDash Q'$ ?

*Solution.* On pose $P = Q = x$, $P' = y$ et $Q' = z$ où $x, y, z$ sont 3 variables.

On a $x \vDash x$ et $x \sim y$ et $x \sim z$. Mais on n'a pas $y \vDash z$ (prendre $\mu(y) = 0 = 1 - \mu(z)$).

L'implication logique $P \vDash Q$ n'est pas préservée si l'on remplace $P$ et $Q$ par des formules seulement équivalentes en satisfiabilité.

□

### Partie I.2 FNC

**Question 5.**

Mettre la proposition suivante sous forme normale conjonctive (FNC) en détaillant les étapes de transformation.

$$\neg(p \wedge \neg q) \vee (r \wedge \neg p)$$

*Solution.*

$$\begin{aligned}
& \neg(p \wedge \neg q) \vee (r \wedge \neg p) \\
\text{(De Morgan)} \quad & = (\neg p \vee \neg\neg q) \vee (r \wedge \neg p) \\
\text{(Double négation)} \quad & = (\neg p \vee q) \vee (r \wedge \neg p) \\
\text{(Associativité de } \vee) \quad & = \neg p \vee q \vee (r \wedge \neg p) \\
\text{(Distributivité de } \vee \text{ sur } \wedge) \quad & = (\neg p \vee q \vee r) \wedge (\neg p \vee q \vee \neg p) \\
\text{(Simplification)} \quad & = (\neg p \vee q \vee r) \wedge (\neg p \vee q)
\end{aligned}$$

On peut encore simplifier en $\neg p \vee q \vee r$ puisque $(A \vee r) \wedge A \equiv A$.

□

**Question 6.**

Soit $n$ un nombre. Donner un exemple de proposition $P$ de taille linéaire en $n$ dont le passage à une FNC équivalente sémantiquement (par de Morgan, suppression des doubles négations et distributivité) est de taille exponentielle en $n$.

*Solution.* Voir cours □

**Question 7.**

Soit $P$ une FNC.

Montrer que $N_{\text{leaf}}(P) = \Theta(|P|)$ (c.a.d que les deux expressions sont du même ordre de grandeur : $N_{\text{leaf}}(P) = O(|P|)$ et $|P| = O(N_{\text{leaf}}(P))$). On explicitera les constantes de domination.

*Solution.* Le nombre de négations est inférieur au nombre de variables. Compter les variables, revient à compter les feuilles d'un arbre entier ($\wedge, \vee$ sont des nœuds binaires mais pas $\neg$) : il y a une feuille de plus que de nœuds binaires.

Le nombre de négations est majoré par $N_{\text{leaf}}(P)$ (puisque les négations sont au contact des variables).

On a donc

$$N_{\text{leaf}}(P) \leq |P| = N_{\text{leaf}}(P) + N_{\text{ni}}(P) \leq \underbrace{N_{\text{leaf}}(P)}_{\text{variables}} + \underbrace{N_{\text{leaf}}(P)}_{\geq \text{nb négations}} + \underbrace{N_{\text{leaf}}(P) - 1}_{\text{nœuds binaires}} \leq 3N_{\text{leaf}}(P) - 1$$

IZP □

## Partie II Transformation de Tseitin

**Question 8.**

Soit $P$ une proposition :

1. Montrer qu'il existe $P'$, une proposition équivalente sémantiquement à $P$, n'utilisant que les variables, $\neg$ et $\vee$.
2. Montrer que $N_{\text{ni}}(P') = \Theta(N_{\text{ni}}(P))$. On explicitera les constantes de domination.

*Solution.* On remplace chaque $a \wedge b$ par $\neg(\neg a \vee \neg b)$. Au pire on multiplie le nombre de nœuds internes par 4.

$$N_{\text{ni}}(P) \leq N_{\text{ni}}(P') \leq 4N_{\text{ni}}(P)$$

□

**Question 9.**

Dans cette question seulement, on enrichit notre langage avec de nouveaux connecteurs.

1. Donner sans justification les tables de vérité de l'implication $\to$, du ou exclusif (xor) $\oplus$ et de l'équivalence $\leftrightarrow$.

*Solution.* Voici

| $x$ | $y$ | $x \to y$ | $x \leftrightarrow y$ | $x \oplus y$ |
|---|---|---|---|---|
| 1 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 0 | 1 | 1 | 1 | 0 |
| 0 | 0 | 1 | 0 | 1 |

□

2. Soient $x, y, z$ 3 variables. Mettre sous FNC :
   (a) $x \leftrightarrow \neg y$.
   (b) $x \leftrightarrow (y \vee z)$

   Justifier.

*Solution.* On passe à la table de vérité puis à la FNCC.

| $x$ | $y$ | $x \leftrightarrow \neg y$ |
|---|---|---|
| 1 | 1 | 0 |
| 1 | 0 | 1 |
| 0 | 1 | 1 |
| 0 | 0 | 0 |

Passage en FNCC :

$$(x \vee y) \wedge (\neg x \vee \neg y)$$

Raisonnement par équivalence sémantique :

$$\begin{aligned} x \leftrightarrow (y \vee z) &\equiv (x \to (y \vee z)) \wedge ((y \vee z) \to x) \\ &\equiv (\neg x \vee y \vee z) \wedge ((\neg y \wedge \neg z) \vee x) \\ &\equiv (\neg x \vee y \vee z) \wedge (x \vee \neg y) \wedge (x \vee \neg z) \end{aligned}$$

□

Le passage d'une proposition $\varphi$ quelconque à un équivalent sémantique en FNC peut prendre un temps exponentiel en la taille de $\varphi$. Plutôt que de transformer une proposition $\varphi$ en une FNC équivalente sémantiquement, on va la réduire à une FNC équivalente en satisfiabilité avec l'objectif d'une meilleure complexité : c'est le rôle de la *transformation de Tseitin*.

Pour $\varphi$ une proposition :

— On note $\mathrm{SF}_\varphi$, l'ensemble des sous-formules de $\varphi$. Par exemple si $x, y$ sont des variables et si $\varphi = \neg(x \vee \neg y)$ alors

$$\mathrm{SF}_\varphi = \{\neg(x \vee \neg y), x \vee \neg y, x, \neg y, y\}$$

La proposition $\varphi$ est une sous-formule de $\varphi$.

— On note $\mathrm{SF}_\varphi \setminus \mathcal{V}$ l'ensemble des sous-formules de $\varphi$ qui ne sont pas des variables. Avec l'exemple précédent :

$$\mathrm{SF}_\varphi \setminus \mathcal{V} = \{\neg(x \vee \neg y), x \vee \neg y, \neg y\}$$

Pour chaque sous-formule $\psi$ de $\mathrm{SF}_\varphi$, on introduit une variable fraîche $a_\psi$ qui n'est pas dans $\varphi$ et à laquelle on donne le sens « $\psi$ est vraie ». Ainsi apparaissent $|\mathrm{SF}_\varphi|$ variables fraîches (autant que de sous-formules). Avec l'exemple précédent, on introduit les variables :

$$a_{\neg(x \vee \neg y)}, a_{x \vee \neg y}, a_{\neg y}, a_x, a_y$$

Soit $\varphi$ une proposition. La question 8 permet de ne travailler qu'avec une proposition équivalente ne comportant que les connecteurs $\vee, \neg$ et dont la taille est du même ordre de grandeur.

Dans la suite de cette partie, **la proposition $\varphi$ et ses sous-formules ne comportent que les connecteurs** $\vee, \neg$.
On définit la *transformée de Tseitin de* $\varphi$, notée $T(\varphi)$ comme la proposition :

$$T(\varphi) = a_\varphi \ \wedge \bigwedge_{\psi \in \mathrm{SF}_\varphi \setminus \mathcal{V}} t(\psi)$$

où, pour toute sous-formule $\psi$ :

— si $\psi$ est de la forme $\neg\varphi_1$ alors $t(\psi) = (a_{\varphi_1} \vee a_{\neg\varphi_1}) \wedge (\neg a_{\neg\varphi_1} \vee \neg a_{\varphi_1})$

— si $\psi$ est de la forme $\varphi_1 \vee \varphi_2$ alors

$$t(\psi) = (a_{\varphi_1 \vee \varphi_2} \vee \neg a_{\varphi_1}) \wedge (a_{\varphi_1 \vee \varphi_2} \vee \neg a_{\varphi_2}) \wedge (\neg a_{\varphi_1 \vee \varphi_2} \vee a_{\varphi_1} \vee a_{\varphi_2})$$

Ainsi, pour $\varphi = \neg(x \vee \neg y)$, la transformée$T(\varphi)$ vaut :

$$\begin{array}{ll} a_{\neg(x\vee\neg y)} & \\ \wedge & (a_{\neg(x\vee\neg y)} \vee a_{x\vee\neg y}) \wedge (\neg a_{\neg(x\vee\neg y)} \vee \neg a_{x\vee\neg y}) \\ \wedge & (a_{x\vee\neg y} \vee \neg a_x) \wedge (a_{x\vee\neg y} \vee \neg a_{\neg y}) \wedge (\neg a_{x\vee\neg y} \vee a_x \vee a_{\neg y}) \\ \wedge & (a_y \vee a_{\neg y}) \wedge (\neg a_y \vee \neg a_{\neg y}) \end{array}$$

**Question 10.**

Calculer $T(\varphi)$ pour $\varphi = (p \vee \neg q) \vee (\neg p)$ où $p, q$ sont des variables.

*Solution.* On applique cette transformation à la formule $\varphi = (p \vee \neg q) \vee (\neg p)$. On obtient

$$T(\varphi) = a_\varphi \wedge t(\neg q) \wedge t(p \vee \neg q) \wedge t(\neg p) \wedge t((p \vee \neg q) \vee (\neg p)).$$

On conclut avec

$$\begin{aligned} \varphi = a_\varphi & \\ & \wedge (a_q \vee a_{\neg q}) \wedge (\neg a_q \vee \neg a_{\neg q}) \\ & \wedge (a_p \vee a_{\neg p}) \wedge (\neg a_p \vee \neg a_{\neg p}) \\ & \wedge (a_{p\vee\neg q} \vee \neg a_p) \wedge (a_{p\vee\neg q} \vee \neg a_{\neg q}) \wedge (\neg a_{p\vee\neg q} \vee \neg a_p \vee \neg a_p) \\ & \wedge \left(a_{(p\vee\neg q)\vee(\neg p)} \vee \neg a_{p\vee\neg q}\right) \wedge \left(a_{(p\vee\neg q)\vee(\neg p)} \vee \neg a_{\neg p}\right) \wedge \left(\neg a_{(p\vee\neg q)\vee(\neg p)} \vee a_{p\vee\neg q} \vee a_{\neg p}\right). \end{aligned}$$

□

Essayons d'abord de comprendre ce que signifie $t(\psi)$ :

**Question 11.**

Soient $\psi$ une sous-formule de $\varphi$. Exprimer $t(\psi)$ à l'aide de $leftrightarrow$.

*Solution.* D'après la question 9, :

— si $\psi$ est de la forme $\neg\varphi_1$ alors

$$t(\psi) = (a_{\varphi_1} \vee a_{\neg\varphi_1}) \wedge (\neg a_{\neg\varphi_1} \vee \neg a_{\varphi_1}) \equiv a_{\neg\varphi_1} \leftrightarrow \neg a_{\varphi_1}$$

— si $\psi$ est de la forme $\varphi_1 \vee \varphi_2$ alors

$$t(\psi) = (a_{\varphi_1 \vee \varphi_2} \vee \neg a_{\varphi_1}) \wedge (a_{\varphi_1 \vee \varphi_2} \vee \neg a_{\varphi_2}) \wedge (\neg a_{\varphi_1 \vee \varphi_2} \vee a_{\varphi_1} \vee a_{\varphi_2}) \equiv a_{\varphi_1 \vee \varphi_2} \leftrightarrow (a_{\varphi_1} \vee a_{\varphi_2})$$

□

**Question 12.**

Étude de la taille de $T(\varphi)$.

1. Montrer par induction que $|\mathrm{SF}_\varphi| \leq 2N_{\mathrm{ni}}(\varphi) + 1$.
2. Montrer qu'il existe $\alpha, \beta$ positifs tels que $\alpha \neq 0$ et vérifiant $N_{\mathrm{ni}}(T(\varphi)) \leq \alpha N_{\mathrm{ni}}(\varphi) + \beta$.

*Solution.* **Point 1** En effet $|\mathrm{SF}_\varphi|$ est plus petit que le nombre de nœuds de $\varphi$ (dans $\mathrm{SF}_\varphi$ on ne compte pas deux fois la même formule alors que $\varphi$ peut être $\varphi = \varphi_1 \vee \varphi_1$).
Par induction : si $\varphi$ est une variable $\mathrm{SF}_\varphi = \{\varphi\}$ et on a bien $1 \leq 2 \times 0 + 1$.
Si $\varphi = \neg\psi$ et si $\left|\mathrm{SF}_\psi\right| \leq 2N_{\mathrm{ni}}(\psi) + 1$.
Alors comme $\mathrm{SF}_\psi \cup \{\neg\psi\} = \mathrm{SF}_\varphi$, et $N_{\mathrm{ni}}(\psi) + 1 = N_{\mathrm{ni}}(\varphi)$, on a

$$|\mathrm{SF}_\varphi| = 1 + \left|\mathrm{SF}_\psi\right| \leq (2N_{\mathrm{ni}}(\psi) + 1) + 1 \leq 2(N_{\mathrm{ni}}(\psi) + 1) + 1 \leq 2N_{\mathrm{ni}}(\varphi) + 1$$

Si $\varphi = \psi_1 \vee \psi_2$ avec $\psi_1, \psi_2$ vérifiant HI. Alors $\mathrm{SF}_{\psi_1} \cup \mathrm{SF}_{\psi_2} \cup \{\varphi\} = \mathrm{SF}_\varphi$ (union pas forcément disjointe), et $N_{\mathrm{ni}}(\psi_1) + 1 + N_{\mathrm{ni}}(\psi_2) = N_{\mathrm{ni}}(\varphi)$.

$$|\mathrm{SF}_\varphi| \leq \left|\mathrm{SF}_{\psi_1}\right| + \left|\mathrm{SF}_{\psi_2}\right| + 1 \leq (2N_{\mathrm{ni}}(\psi_1) + 1) + (2N_{\mathrm{ni}}(\psi_2) + 1) + 1 \leq 2(N_{\mathrm{ni}}(\psi_1) + 1 + N_{\mathrm{ni}}(\psi_2)) + 1 \leq 2N_{\mathrm{ni}}(\varphi) + 1$$

**Point 2** Si $\psi$, une sous-formule, est une négation, alors $N_{\mathrm{ni}}(t(\psi)) = 5$ et sinon (disjonction) $N_{\mathrm{ni}}(t(\psi)) = 9$.
Le nombre de connecteurs de $T(\varphi)$ est 1 (premier $\wedge$) + le nombre de connecteurs dans la conjonction sur les sous-formules. Donc

$$N_{\mathrm{ni}}(T(\varphi)) \leq 1 + 9\,|\mathrm{SF}_\varphi \setminus \mathcal{V}| + \underbrace{|\mathrm{SF}_\varphi \setminus \mathcal{V}|}_{\text{les } \wedge \text{ entre les } t(\psi)} \leq 1 + 10\,|\mathrm{SF}_\varphi| \leq 1 + 10(2N_{\mathrm{ni}}(\varphi) + 1) = 11 + 20N_{\mathrm{ni}}(\varphi)$$

Ainsi $\alpha = 20, \beta = 11$. Il vient que $T(\varphi)$ et $\varphi$ sont bien du même ordre de grandeur.

□

**Question 13.**

1. Montrer que si $\varphi$ est satisfiable alors $T(\varphi)$ aussi.

   *Indication.* Si $\mu$ satisfait $\varphi$, définir le contexte $\mu'$ par $\mu'(a_\psi) = \varepsilon_\mu(\psi)$ pour tout $\psi \in \mathrm{SF}_\varphi \setminus \mathcal{V}$.

2. Montrer que si $T(\varphi)$ est satisfiable alors $\varphi$ aussi.

   *Indication.* Si $\mu$ satisfait $T(\varphi)$, définir le contexte $\mu'$ par $\mu'(x) = \mu(a_x)$ pour toute variable $x$ de $\varphi$.

*Solution.* **Supposons $\varphi$ satisfiable** . Soit $\mu$ un contexte défini sur les variables de $\varphi$ et rendant $\varphi$ vrai.
On définit le contexte $\mu'$ par $\mu'(a_\psi) = \varepsilon_\mu(\psi)$ pour tout $\psi \in \mathrm{SF}_\varphi \setminus \mathcal{V}$.
On montre que $\varepsilon_{\mu'}(t(\varphi')) = 1$ pour toute sous-formule non triviale (*i.e.* non une variable) $\varphi'$ de $\varphi$.

— Si $\varphi' = \neg\psi$. Alors, On a

$$t(\varphi') = t(\neg\psi) \equiv a_{\neg\psi} \leftrightarrow \neg a_\psi$$

Or,

$$\varepsilon_{\mu'}(\neg a_\psi) = 1 - \varepsilon_{\mu'}(a_\psi) = 1 - \varepsilon_\mu(\psi) = \varepsilon_\mu(\neg\psi) = \varepsilon_\mu(\varphi') = \varepsilon_{\mu'}(a_{\varphi'}) = \varepsilon_{\mu'}(a_{\neg\psi}).$$

Donc $\varepsilon_{\mu'}(t(\varphi')) = 1$ puisque les interprétations des deux membres de l'équivalence sont les mêmes.

— Si $\varphi' = \psi_1 \vee \psi_2$ alors

$$\varepsilon_{\mu'}(a_{\psi_1 \vee \psi_2}) = \varepsilon_{\mu'}(a_{\varphi'}) = \varepsilon_\mu(\varphi') = \varepsilon_\mu(\psi_1 \vee \psi_2) = \max(\varepsilon_\mu(\psi_1), \varepsilon_\mu(\psi_2)) = \max(\varepsilon_{\mu'}(a_{\psi_1}), \varepsilon_{\mu'}(a_{\psi_2})) = \varepsilon_{\mu'}(a_{\psi_1} \vee a_{\psi_2})$$

Comme on a

$$t(\varphi') = t(\psi_1 \vee \psi_2) \equiv a_{\psi_1 \vee \psi_2} \leftrightarrow a_{\psi_1} \vee a_{\psi_2}$$

On obtient :

$$\varepsilon_{\mu'}(t(\varphi)) = \varepsilon_{\mu'}(a_\varphi \leftrightarrow a_{\psi_1} \vee a_{\psi_2}) = 1$$

puisque les interprétations des deux membres de l'équivalence sont les mêmes

On a enfin :

$$T(\varphi) = a_\varphi \wedge \bigwedge_{\psi \in \mathrm{SF}_\varphi \setminus \mathcal{V}} t(\psi)$$

Avec

$$\varepsilon_{\mu'}(a_\varphi) = \varepsilon_\mu(\varphi) = 1$$

Pour terminer

$$\varepsilon_{\mu'}(T(\varphi)) = \varepsilon_{\mu'}(a_\varphi) \times \varepsilon_{\mu'}\left(\bigwedge_{\psi \in \mathrm{SF}_\varphi \setminus \mathcal{V}} t(\psi)\right) = \varepsilon_{\mu'}(a_\varphi) \times \prod_{\psi \in \mathrm{SF}_\varphi \setminus \mathcal{V}} \underbrace{\varepsilon_{\mu'}(t(\psi))}_{=1} = 1 \times 1 = 1$$

Et donc $T(\varphi)$ est satisfiable.

**Supposons $T(\varphi)$ satisfiable** Soit $\mu$ un contexte qui satisfait $T(\varphi)$.

On définit $\mu'$ pour toute variable $x$ de $\varphi$ par $\mu'(x) = \mu(a_x)$.

Par induction, on montre que $\varepsilon_{\mu'}(\psi) = \varepsilon_\mu(a_\psi)$ pour toute souss-formule de $\varphi$.

**Cas de base** Si $\psi$ est une variable : OK par définition.

**Hérédité** — Si $\psi = \neg\gamma$ alors $\varepsilon_\mu(T(\varphi)) = 1$ entraîne que $1 = \varepsilon_\mu(t(\neg\gamma)) = \varepsilon_\mu(a_{\neg\gamma} \leftrightarrow \neg a_\gamma)$.

Ainsi $\varepsilon_\mu(a_{\neg\gamma}) = 1 - \varepsilon_\mu(a_\gamma)$. Or, par HI $\varepsilon_\mu(a_\gamma) = \varepsilon_{\mu'}(\gamma)$.

Il vient :

$$\varepsilon_\mu(a_\psi) = 1 - \varepsilon_{\mu'}(\gamma) = \varepsilon_{\mu'}(\neg\gamma) = \varepsilon_{\mu'}(\psi)$$

Hérédité OK.

— Si $\psi = \psi_1 \vee \psi_2$ alors $\varepsilon_\mu(T(\varphi)) = 1$ entraîne que $1 = \varepsilon_\mu(t(\psi_1 \vee \psi_2)) = \varepsilon_\mu(a_{\psi_1 \vee \psi_2} \leftrightarrow (a_{\psi_1} \vee a_{\psi_2}))$.

Ainsi

$$\varepsilon_\mu(a_\psi) = \varepsilon_\mu(a_{a_{\psi_1 \vee \psi_2}}) = \varepsilon_\mu(a_{\psi_1} \vee a_{\psi_2}) = \max(\varepsilon_\mu(a_{\psi_1}), \varepsilon_\mu(a_{\psi_2})) \underbrace{=}_{\text{HI}} \max(\varepsilon_{\mu'}(\psi_1), \varepsilon_{\mu'}(\psi_2)) = \varepsilon_{\mu'}(\psi_1 \vee \psi_2) = \varepsilon_{\mu'}(\psi)$$

Hérédité OK.

Au final on obtient $\varepsilon_\mu(a_\varphi)) = \varepsilon_{\mu'}(\varphi)$.

Comme $\varepsilon_\mu(T(\varphi)) = 1$, on en déduit que $\varepsilon_\mu(a_\varphi) = 1$ et donc que $\varepsilon_{\mu'}(\varphi) = 1$.

La proposition $\varphi$ est donc satisfiable.

□

**Question 14.**

A-t-on $T(\varphi) \equiv \varphi$

*Solution.* Non, il suffit de prendre $\varphi = x$ où $x$ est une variable.

Alors $T(\varphi) = a_\varphi \neq x$.

Et deux variables différentes ne sont pas en équivalence sémantique. □

On dit qu'une proposition est une 3-FNC si c'est une FNC dont les clauses contiennent au plus 3 littéraux.

On appelle problème **SAT** le problème de décision qui consiste à déterminer si une proposition est satisfiable ou non. Ce problème est connu pour être NP-complet.

Le problème **3-FNC-SAT** est le problème de décision qui consiste à déterminer si une 3-FNC est satisfiable ou non.

Pour toute proposition $\varphi$, on a montré qu'on peut la tranformer en une 3-FNC $T(\varphi)$ dont la taille est du même ordre de grandeur que celle de $\varphi$. Cette transformation s'effectue en temps linéaire en celle de $\varphi$. De plus, vérifier si un contexte satisfait $T(\varphi)$ se déroule en temps linéaire en la taille de $T(\varphi)$ donc de $\varphi$.

Ces deux faits permettent d'établir que **3-FNC-SAT** est NP-Complet.

## Partie III Algorithme de Quine

On vient de voir une méthode pour transformer en temps polynômial une proposition $\varphi$ quelconque en $T(\varphi)$, une FNC (dont les clauses sont de longueurs au plus 3) et en équivalence de satisfiabilité avec $\varphi$.

Dans cette partie, indépendante de la première, on suppose la transformation efectuée. Les propositions sont toutes des FNC.

On donne les types suivants :

```
type var = int;;
type literal = V of var | NV of var;;(*les littéraux positifs et négatifs*)
type clause = literal list;;(*les clauses*)
type cnf = clause list;;(*les FNC*)
```

Par exemple $(x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee x_3)$ s'implémente en

```
  [[V 1; V2; V 3];[NV 1; V3]]
```

On rappelle qu'une clause vide représente le faux et une FNC vide, le vrai.

### Question 15.

Écrire la fonction `litt_of_name_and_bool (x:var) (b:bool) : literal` qui prend en paramètre une variable et un bouléen et construit le littéral correspondant positif si le bouléen est vrai et négatif sinon.

```
# litt_of_name_and_bool 2 true;;
- : litteral = V 2
# litt_of_name_and_bool 2 false;;
- : litteral = NV 2
```

Soit $\varphi$ une FNC.

Considérons la clause

$$C = \ell_1 \vee \cdots \vee \ell_{k-1} \vee x \vee \ell_{k+1} \vee \cdots \vee \ell_n$$

— Dans un contexte où la variable $x$ est évaluée à vraie, $C$ devient vraie. On en déduit qu'on peut supprimer de $\varphi$ toutes les clauses qui contiennent $x$ sans changer la sémantique.

— Dans un contexte où la variable $x$ est évaluée à faux, $C$ est équivalente à $\ell_1 \vee \cdots \vee \ell_{k-1} \vee \ell_{k+1} \vee \cdots \vee \ell_n$. On en déduit qu'on peut supprimer le littéral $x$ de toutes les clauses de $\varphi$ sans changer la sémantique.

Symétriquement :

— Dans un contexte où la variable $x$ est évaluée à vraie, on peut supprimer le littéral $\neg x$ de toutes les clauses de $\varphi$ sans changer la sémantique.

— Dans un contexte où la variable $x$ est évaluée à faux, on peut supprimer toutes les clauses de $\varphi$ qui contiennent $\neg x$ sans changer la sémantique.

Ces deux remarques sont à la base de l'*algorithme de Quine*.

### Question 16.

Écrire la fonction `delete_lit (c:clause) (x:var) (b:bool)` qui supprime `NV x` de la clause si `b = true` et `V x` si `b = false`.

```
# delete_lit [V 1;NV 2; V 3] 1 true;;
- : litteral list = [V 1; NV 2; V 3]
# delete_lit [V 1;NV 2; V 3] 1 false;;
- : litteral list = [NV 2; V 3]
```

**Question 17.**

Écrire la fonction `subst (f:cnf) (x:var) (b:bool) : cnf option` qui remplace la variable `x` par `b`. On récupère :

— soit une option vide `None` (signifiant que mettre `x` à `b` amène une contradiction) ,

— soit une option contenant une FNC dans laquelle il ne reste plus de littéraux de variable $x$ et dont certaines clauses ont été supprimées.

```
# subst [[V 1;NV 2; V 3];[V 1; V 2];[V 2; V 3];[V 3]] 2 true;;
- : cnf option = Some [[V 1; V 3]; [V 3]]
# subst [[V 1;NV 2; V 3];[V 1; V 2];[V 2; V 3];[V 3]] 3 false;;
- : cnf option = None
# subst [[NV 3];[NV 3; V 1]] 3 false;;
- : cnf option = Some []
```

**Question 18.**

Écrire la fonction `get_var (f:cnf) : var` qui retourne la première variable encore présente dans la FNC.

```
# get_var [[V 4;NV 2; V 3];[V 1; V 2];[V 2; V 3];[V 3]];;
- : var = 4
# let f = [] in get_var f;;
Exception: Failure "pas de variable".
```

L'*algorithme de Quine* prend en paramètre une FNC et renvoie un bouléen qui indique si elle est satisfiable. Il s'agit d'un *backtracking* avec *élagage* qui exploite la méthode décrite plus haut :

— C'est un backtraking puisqu'il explore un arbre de décision dont les feuilles représentent tous les contextes possibles ;

— Il y a de l'élagage puisque certaines branches de l'arbre de décision ne sont pas parcourues jusqu'aux feuilles (typiquement, lorsque une substitution vide une clause, l'exploration abandonne la branche courante).

Le principe de l'algorithme de Quine est le suivant :

— Si la FNC est vide, alors elle est satisfiable et on renvoie vrai.

— Sinon, on attribue la valeur `true` à `x`, la première variable disponible, et on effectue la substitution. Soit on récupère une FNC (forcément plus courte avec moins de variables et de clauses) et on applique l'algorithme récursivement, soit on obtient une indication de contradiction.

En cas de contradiction, on tente d'attribuer la valeur `false` à `x` et on agit de même que précédemment. On effectue la substitution : si elle fournit une FNC plus courte on fait un appel récursif sinon on renvoie faux.

**Question 19.**

Écrire la fonction `quine (f:cnf) : bool` qui réalise l'algorithme de Quine.