Notions de logique

Judicaël Courant

2018-W38-1

1 Introduction

1.1 Manipulation de termes et formules

$$xyz + 3x - 7y + z \geq 12 \Rightarrow (x + y)(z - 2y) \geq 42$$

NB : on manipulera des formules sans quantificateurs.

1.2 Syntaxe : concrète ou abstraite ?

Quand on étudie une formule, on peut parler de

sa syntaxe concrète la chaîne de caractères qui permet de l'écrire ;

sa syntaxe abstraite une description arborescente de cette formule.

C'est le second point qui nous intéresse.

1.3 Syntaxe abstraite

La syntaxe concrète peut (ici) être décrite par une *grammaire abstraite*

⟨*formule*⟩ : := ⟨*formule atomique*⟩
| ⟨*formule*⟩ '$\wedge$' ⟨*formule*⟩
| ⟨*formule*⟩ '$\vee$' ⟨*formule*⟩
| ⟨*formule*⟩ '$\Rightarrow$' ⟨*formule*⟩
| ⟨*formule*⟩ ' $\Longleftrightarrow$ ' ⟨*formule*⟩
| '$\neg$' ⟨*formule*⟩
| '$\bot$'
| '$\top$'

⟨*formule atomique*⟩ : := ⟨*terme*⟩ '$=$' ⟨*terme*⟩
| ⟨*terme*⟩ '$<$' ⟨*terme*⟩
| ⟨*terme*⟩ '$\leq$' ⟨*terme*⟩
| ⟨*terme*⟩ '$>$' ⟨*terme*⟩
| ⟨*terme*⟩ '$\geq$' ⟨*terme*⟩

⟨*terme*⟩ ::= ⟨*variable*⟩
| ⟨*terme*⟩ '+' ⟨*terme*⟩
| ⟨*terme*⟩ '−' ⟨*terme*⟩
| ⟨*terme*⟩ '×' ⟨*terme*⟩
| ⟨*terme*⟩ '/' ⟨*terme*⟩
| ⟨*constante entière*⟩

NB : Cette grammaire ne précise rien
- ni sur le sens d'une formule ou d'un terme ;
- ni même sur la façon de lever les ambiguïtés lorsqu'on lit une formule ou un terme écrit de façon concrète (par exemple : $a + b \times c$ est-il un arbre de racine $\times$ ou $+$?).

Elle sert à représenter uniquement ce qui se passe sur les arbres.

Ces arbres se modélisent naturellement par un type OCaml :

```
type terme =
  | Variable of string
  | Constante of int
  | Plus of terme × terme
  | Moins of terme × terme
  | Mult of terme × terme
  | Div of terme × terme

type formule_atomique =
  | InfStr of terme × terme
  | Inf of terme × terme
  | SupStr of terme × terme
  | Sup of terme × terme

type formule =
  | Atome of formule_atomique
  | Conj of formule × formule
  | Disj of formule × formule
  | Impl of formule × formule
  | Equiv of formule × formule
  | Neg of formule
  | Faux
  | Vrai
```

1.4 Sémantique

Sémantique : sens, valeur.

1.4.1 Sémantique des termes

On va *interpréter* les termes comme des représentations de nombres flottants pour leur donner une *valeur*. Ici l'ensemble des valeurs pour les termes sera l'ensemble des réels.

Étant donné un terme t, on veut définir précisément sa sémantique.

Problème : quelle valeur donner aux variables ?
L'interprétation d'un terme doit dépendre d'un *environnement.*

Définition 1.4.1 (Environnement)**.** *On appelle* environnement *ou* valuation des variables *ou* interprétation des variables *toute fonction ρ définie sur une partie V de l'ensemble des variables et associant à tout $x \in V$ une* valeur $\rho(x)$.

Étant donné un environnement ρ et un terme t on notera $[\![t]\!]_\rho$ la valeur de t dans l'environnement ρ.

On définit alors la sémantique des termes par induction sur leur syntaxe abstraite de la façon suivante :

$$[\![x]\!]_\rho = \rho(x)$$
$$[\![t_1 + t_2]\!]_\rho = [\![t_1]\!]_\rho + [\![t_2]\!]_\rho \qquad [\![t_1 - t_2]\!]_\rho = [\![t_1]\!]_\rho - [\![t_2]\!]_\rho$$
$$[\![t_1 \times t_2]\!]_\rho = [\![t_1]\!]_\rho \times [\![t_2]\!]_\rho \qquad [\![t_1/t_2]\!]_\rho = \frac{[\![t_1]\!]_\rho}{[\![t_2]\!]_\rho}$$

pour toute variable x et tous termes t_1 et t_2.

Exercice 1.4.2. *Le $+$ dans $t_1 + t_2$ représente-t-il :*

A l'addition sur les flottants ($+.$ en OCaml)

B l'addition sur les termes ($Plus$ en OCaml)

L'implantation est assez directe. On s'intéressera ici uniquement à des environnements définis sur un ensemble fini de variables, et on représentera par la liste $[(x_0, v_0); \ldots; (x_{n-1}; v_{n-1})]$ la fonction qui pour tout $i \in [\![0, n[\![$ associe v_i à x_i. type $var = string$

```
type α env = (var × α) list

let rec eval_t e t = match t with
  | Variable x → List.assoc x e
  | Constante n → float_of_int n
  | Plus(t1, t2) →
    let v1 = eval_t e t1 and v2 = eval_t e t2 in
    v1 +. v2
  | Moins(t1, t2) →
    let v1 = eval_t e t1 and v2 = eval_t e t2 in
    v1 -. v2
  | Mult(t1, t2) →
    let v1 = eval_t e t1 and v2 = eval_t e t2 in
    v1 *. v2
  | Div(t1, t2) →
    let v1 = eval_t e t1 and v2 = eval_t e t2 in
    v1 /. v2
```


1.5 Sémantique des formules

On veut étendre cette interprétation aux formules.
La valeur d'une formule est alors une /valeur de vérité/ : vrai (V) ou faux (F).

$$\begin{aligned}
[\![\text{Egal}(t_1, t2)]\!]_\rho &= V && \text{si } [\![t_1]\!]_\rho = [\![t_2]\!]_\rho \\
&= F && \text{sinon} \\
[\![\text{Inf}(t_1, t2)]\!]_\rho &= V && \text{si } [\![t_1]\!]_\rho \leq [\![t_2]\!]_\rho \\
&= F && \text{sinon} \\
&\vdots
\end{aligned}$$

$$\begin{aligned}
[\![\text{Conj}(A_1, A_2)]\!]_\rho &= V && \text{si } [\![A_1]\!]_\rho = [\![A_2]\!]_\rho = V \\
&= F && \text{sinon} \\
[\![\text{Disj}(A_1, A_2)]\!]_\rho &= F && \text{si } [\![A_1]\!]_\rho = [\![A_2]\!]_\rho = F \\
&= V && \text{sinon} \\
[\![\text{Impl}(A_1, A_2)]\!]_\rho &= F && \text{si } [\![A_1]\!]_\rho = V \text{ et } [\![A_2]\!]_\rho = F \\
&= V && \text{sinon}
\end{aligned}$$

$$\begin{aligned}
[\![\text{Equiv}(A_1, A_2)]\!]_\rho &= V && \text{si } [\![A_1]\!]_\rho = [\![A_2]\!]_\rho \\
&= F && \text{sinon} \\
[\![\text{Neg}(A)]\!]_\rho &= V && \text{si } [\![A]\!]_\rho = F \\
&= F && \text{si } [\![A]\!]_\rho = V \\
[\![\bot]\!]_\rho &= F \\
[\![\top]\!]_\rho &= V
\end{aligned}$$

1.6 Implantation

1.7 Remarques

D'autres sémantiques sont possibles. Pour les termes on peut par exemple donner des sémantiques à valeurs :
— dans les entiers ;
— dans $\mathbb{Z}/n\mathbb{Z}$.
Pour les formules, on peut définir une sémantique à valeurs :

— dans un ensemble à trois valeurs (vrai/faux/indéfini) [1] ;
— dans l'ensemble des ouverts d'un espace topologique (modèles de la logique intuitionniste).

1.8 Problèmes

Définition 1.8.1. *On dit qu'un environnement ρ* satisfait *une formule A si $[\![A]\!]_\rho = V$.*

On dit qu'une formule A est satisfiable *(ou* satisfaisable*) s'il existe un environnement ρ qui satisfait A.*

On dit qu'une formule est une tautologie si dans tout environnement donnant une définition à chaque variable de A, son interprétation vaut V.

Comment tester qu'une formule est satisfiable ? est une tautologie ?
Pour répondre à cette question, on va se placer dans un cadre plus simple…

2 Logique propositionnelle

2.1 Définition

Étant donné un ensemble (au plus dénombrable) $\mathcal{V}$ de variables propositionnelles, l'ensemble $\mathcal{F}$ des formules de la logique propositionnelle est l'ensemble de formules engendré par la grammaire abstraite suivante :

⟨*formule*⟩ ::= ⟨*formule atomique*⟩
| ⟨*formule*⟩ '$\wedge$' ⟨*formule*⟩
| ⟨*formule*⟩ '$\vee$' ⟨*formule*⟩
| ⟨*formule*⟩ '$\Rightarrow$' ⟨*formule*⟩
| ⟨*formule*⟩ ' $\Longleftrightarrow$ ' ⟨*formule*⟩
| '$\neg$' ⟨*formule*⟩
| '$\bot$'
| '$\top$'

⟨*formule atomique*⟩ ::= ⟨*variable propositionnelle*⟩

Définition 2.1.1. *Les* atomes *de la logique propositionnelle sont les variables. On appelle* littéral*, toute formule qui est soit un atome a, soit la négation $\neg a$ d'un atome.*

Étant donné un littéral *l, on notera $\bar{l}$ le littéral*
— *$\neg l$ si l est un atome ;*
— *a si l est la négation $\neg a$ d'un atome a.*

Définition 2.1.2 (Logique booléenne)**.** *On se donne deux valeurs de vérités notées* **0** *et* **1** *(ou $\bot$ et $\top$, ou F et V). L'ensemble $\{ \mathbf{0}, \mathbf{1} \}$ est appelé* ensemble des booléens *et est noté $\mathcal{B}$.*

1. Séduisant pour faire de «l'intelligence artificielle» mais ne tient pas ses promesses.

On définit deux lois de composition interne sur $+$ et $.$ sur $\mathcal{B}$ par :

$$\begin{aligned} \mathbf{0}+\mathbf{0}&=\mathbf{0} & \mathbf{0}.\mathbf{0}&=\mathbf{0}\\ \mathbf{0}+\mathbf{1}&=\mathbf{1} & \mathbf{0}.\mathbf{1}&=\mathbf{0}\\ \mathbf{1}+\mathbf{0}&=\mathbf{1} & \mathbf{1}.\mathbf{0}&=\mathbf{0}\\ \mathbf{1}+\mathbf{1}&=\mathbf{1} & \mathbf{1}.\mathbf{1}&=\mathbf{1} \end{aligned}$$

ainsi qu'une application $\overline{\ }:\mathcal{B}\to\mathcal{B}$ par :

$$\begin{aligned} \overline{\mathbf{0}}&=\mathbf{1}\\ \overline{\mathbf{1}}&=\mathbf{0} \end{aligned}$$

Exercice 2.1.3. *Vrai ou faux ?*
Dans $\mathcal{B}$:

1. *$+$ et $.$ sont deux lois associatives et commutatives ;*
2. *$+$ et $.$ possèdent un élément neutre et un élément absorbant ;*
3. *$.$ est distributive par rapport à $+$ ($a.(b+c)=a.b+a.c$) ;*
4. *$+$ est distributive par rapport à $.$ ($a+(b.c)=(a+b).(a+c)$).*
5. $b+\overline{b}=\mathbf{1}$;
6. $b.\overline{b}=\mathbf{0}$;
7. $\overline{a+b}=\overline{a}.\overline{b}$;
8. $\overline{a.b}=\overline{a}+\overline{b}$

Définition 2.1.4 (Valuation)**.** *Une* valuation (atomique) *ou* interprétation des variables *est une fonction $\rho:\mathcal{V}\to\mathcal{B}$.*

Définition 2.1.5 (Interprétation d'une formule propositionnelle)**.** *La* valeur de vérité $[\![\phi]\!]_\rho$ *d'une proposition ϕ pour l'interprétation des variables ρ est définie par induction structurelle sur les formules par :*

$$\begin{aligned} [\![a]\!]_\rho &= \rho(a) \text{ pour } a\in\mathcal{V}\\ [\![\phi\wedge\psi]\!]_\rho &= [\![\phi]\!]_\rho.[\![\psi]\!]_\rho & [\![\phi\vee\psi]\!]_\rho &= [\![\phi]\!]_\rho+[\![\psi]\!]_\rho\\ [\![\bot]\!]_\rho &= \mathbf{0} & [\![\top]\!]_\rho &= \mathbf{1}\\ [\![\phi\Rightarrow\psi]\!]_\rho &= \overline{[\![\phi]\!]_\rho}+[\![\psi]\!]_\rho & [\![\phi\iff\psi]\!]_\rho &= (\overline{[\![\phi]\!]_\rho}+[\![\psi]\!]_\rho).([\![\phi]\!]_\rho+\overline{[\![\psi]\!]_\rho})\\ [\![\neg\phi]\!]_\rho &= \overline{[\![\phi]\!]_\rho} \end{aligned}$$

On appelle *table de vérité* un tableau dans lequel la dernière colonne est étiquetée par une formule propositionnelle et les colonnes précédentes par les n variables apparaissant dans la formule.

La table comporte alors, pour chaque intepretation possible de ces n variables, exactement une ligne donnant les valeurs de vérité associées aux variables et la valeur de vérité de la formule.

Exercice 2.1.6. *Étant donné n variables propositionnelles, il y a exactement :*

A n *valuations possibles*

B $2n$ *valuations possibles*

C n^2 *valuations possibles*

D 2^n *valuations possibles*

Définition 2.1.7. *On dit qu'une formule ϕ est* satisfaite *par une valuation ρ, ou que ρ est un* modèle *de ϕ si $[\![\phi]\!]_\rho = \mathbf{1}$, et on note cela $\rho \models \phi$.*

On dit qu'une formule est satisfiable *(ou* satisfaisable*) si elle possède au moins un modèle.*

On dit que ρ falsifie *la formule ϕ si $\rho \not\models \phi$ (i.e. $[\![\phi]\!]_\rho = \mathbf{0}$).*

On dit qu'une formule ϕ est une tautologie si elle est satisfaite par toute valuation, et on note cela $\models \phi$:

$$\models \phi \text{ si et seulement si } \forall \rho \in \mathcal{B}^{\mathcal{V}}\ \rho \models \phi$$

On dit que deux formules ϕ et ψ sont équivalentes et on note $\phi \equiv \psi$, si pour toute valuation ρ, $[\![\phi]\!]_\rho = [\![\psi]\!]_\rho$.

Proposition 2.1.8. *Soit ϕ et ψ deux formules.*

L'équivalence de formules est une congruence, c'est-à-dire une relation d'équivalence qui de plus passe au contexte : notons $C[\phi]$ une formule contenant ψ comme sous-formule ϕ et $C[\psi]$ la formule obtenue en remplaçant cette sous-formule par ψ. Si $\phi \equiv \psi$ alors $C[\phi] \equiv C[\psi]$.

Les tautologies sont les formules équivalentes à $\top$, les formules non-satisfiables sont celles équivalentes à $\bot$.

ϕ est satisfiable si et seulement si $\neg\phi$ n'est pas une tautologie.

ϕ est une tautologie si et seulement si $\neg\phi$ n'est pas satisfiable.

ϕ et ψ sont équivalentes si et seulement si $(\phi \iff \psi)$ est une tautologie.

On déduit des propriétés de l'algèbre de Boole $\mathcal{B}$ les résultats suivants :

Proposition 2.1.9. *Soit ϕ, ψ et χ trois formules. Alors*

$$\phi \vee (\psi \vee \chi) \equiv (\phi \vee \psi) \vee \chi \qquad \phi \wedge (\psi \wedge \chi) \equiv (\phi \wedge \psi) \wedge \chi$$
$$\phi \vee \psi \equiv \psi \vee \phi \qquad \phi \wedge \psi \equiv \psi \wedge \phi$$
$$\phi \wedge (\psi \vee \chi) \equiv (\phi \wedge \psi) \vee (\phi \wedge \chi) \qquad \phi \vee (\psi \wedge \chi) \equiv (\phi \vee \psi) \wedge (\phi \vee \chi)$$
$$\phi \vee \phi \equiv \phi \qquad \phi \wedge \phi \equiv \phi$$
$$\phi \vee \bot \equiv \phi \qquad \phi \wedge \top \equiv \phi$$
$$\phi \wedge \bot \equiv \bot \qquad \phi \vee \top \equiv \top$$
$$\neg\neg\phi \equiv \phi$$

On en déduit également :

Proposition 2.1.10 (Lois de De Morgan)**.** *Soit ϕ et ψ deux formules. Alors*

$$\neg(\phi \vee \psi) \equiv \neg\phi \wedge \neg\psi \qquad \neg(\phi \wedge \psi) \equiv \neg\phi \vee \neg\psi$$

2.2 Le problème SAT

On appelle *taille* d'une formule le nombre de nœuds et feuilles que l'arbre qui la représente comporte.

On s'intéresse à une formule de taille n contenant k variables distinctes.

Exercice 2.2.1. *Laquelle de ces affirmations est correcte :*

A $k = O(\log n)$ *(borne serrée);*

B $k = O(\sqrt{n})$ *(borne serrée);*

C $k = O(n)$ *(borne serrée);*

D Aucune des trois précédentes.

Exercice 2.2.2. *Exercice : soit une formule de taille n sans double négation et ne comportant que des variables différentes. Montrer qu'elle comporte au moins $(n+2)/4$ variables.*

Les questions qu'on aimerait résoudre :
- Comment tester qu'une formule est satisfiable ? (problème SAT)
- Comment tester qu'une formule est une tautologie ?

Remarque : dualité
- pour montrer qu'une formule est une tautologie, il suffit de montrer que sa négation n'est pas satisfiable.
- pour montrer qu'une formule est satisfiable, il suffit de montrer que sa négation n'est pas une tautologie.

On va donc s'intéresser à un seul des deux problèmes : celui de la satisfiabilité.

2.2.1 Méthode naïve

Essayer toutes les interprétations des variables.

Complexité en temps ?

A $O(n^2)$;

B $O(2^n)$;

C $O(2^n \times n)$.

D $O(2^n \times n \log n)$.

2.2.2 Culture : Test polynomial par certificat

Si on dispose d'une valuation ρ on peut tester si $\rho \models \phi$ en temps polynomial en la taille de ϕ (et même linéaire si les variables de ϕ sont numérotées par des entiers et ρ représenté par un tableau de booléens).

Donc si ϕ est satisfiable, il existe une valuation ρ, qu'on pourra appeler *certificat* (ou *témoin*, ou *preuve*) de satisfiabilité, vérifiable en temps polynomial en la taille de la formule.

On dit que la satisfiabilité est *vérifiable par certificat en temps polynomial.*[2] La classe des problèmes vérifiables par certificat en temps polynomial étant notée NP, on écrit

2. Cette définition est délibérément imprécise : la rendre rigoureuse nous entraînerait trop loin.

$$SAT \in NP$$

NB :

— La classe des problèmes dont la négation est vérifiable en temps polynomial est appelée $CoNP$.
— On ne connaît pas de notion de certificat pour la non-satisfiabilité en temps polynomial, c'est-à-dire qu'on n'a pas aujourd'hui réussi à montrer que SAT appartient à $CoNP$.
— On conjecture en fait que SAT n'appartient pas à $CoNP$. [3]
— Toute avancée sur ces questions vous vaudra probablement la célébrité.

De façon duale :

— Le caractère non-tautologique d'une proposition est vérifiable par certificat en temps polynomial, donc le caractère tautologique d'une fonction est un problème appartenant à $CoNP$.
— On ne connaît pas de notion de certificat pour le caractère tautologique en temps polynomial (et on doute fortement qu'il en existe : si ce problème était dans NP, on aurait $NP = CoNP$).

Nota Bene :

— La notion de problème NP est en général introduite légèrement différemment (à partir de machines de Turing non-déterministes) et l'existence d'une méthode de test par certificat est un critère d'appartenance à la classe NP découlant assez immédiatement de la définition.
— Pour aborder rigoureusement ces notions, il faudrait aborder sérieusement la notion de machines de Turing, qui est hors-programme.

2.2.3 Culture : La classe P

On appelle P la classe des problèmes décidables par un algorithme en temps polynomial en la taille du problème. [4]

On a $P \subseteq NP$. On conjecture que cette inclusion est stricte mais cela reste à démontrer [5]

2.2.4 Culture : SAT et la classe P

Si $SAT \notin P$, l'inclusion $P \subseteq NP$ est évidemment stricte.

On peut montrer que $SAT \in P \Rightarrow P = NP$. On pense donc que $SAT \notin P$.

Enfin, on peut montrer que $P = NP \Rightarrow NP = CoNP$, donc $NP \neq CoNP \Rightarrow NP \neq P$. [6]

2.2.5 Ce qu'il faut retenir

Vrai ou faux :

— Il existe des algorithmes efficaces pour vérifier si une formule est satisfiable.

3. $SAT \in CoNP$ impliquerait $NP = CoNP$, or on conjecture $NP \neq CoNP$.
4. Même remarque que précédemment sur l'imprécision de la définition.
5. C'est sans conteste la conjecture la plus célèbre en informatique. Démontrez-là ou infirmez-là et c'est la gloire assurée.
6. Montrer $NP \neq CoNP$ vous assure donc également la gloire.

— Il existe des algorithmes efficaces pour vérifier si une formule est une tautologie.
— Il existe un moyen de convaincre quelqu'un qu'une formule est satisfiable en lui donnant un certificat qu'il peut vérifier en temps polynomial en la taille de la formule.
— Il existe un moyen de convaincre quelqu'un qu'une formule est une tautologie en lui donnant un certificat qu'il peut vérifier en temps polynomial en la taille de la formule.

Nous allons maintenant chercher des moyens aussi efficaces que possibles pour vérifier la satisfiabilité de formules mais sans nous leurrer : nous ne trouverons pas d'algorithme général travaillant en temps polynomial dans le cas le pire.

2.3 Formes normales conjonctives/disjonctives

Définition 2.3.1 (Disjonction/conjonction n-aire)**.** *Étant donné n formules $\phi_0, \dots, \phi_{n-1}$, avec $n \in \mathbb{N}^*$, on appelle disjonction n-aire de ces formules la formule $\phi_0 \vee (\phi_1 \vee (\dots \vee \phi_{n-1}))$ notée aussi*

$$\bigvee_{i \in [\![0,n[\![} \phi_i$$

Par convention, la disjonction de 0 formules est $\bot$.

De cette façon, si p et q sont deux entiers naturels et $\phi_0, \dots, \phi_{p+q-1}$ sont $p+q$ formules, on a

$$\bigvee_{i \in [\![0,p+q[\![} \phi_i \equiv \left(\bigvee_{i \in [\![0,p[\![} \phi_i \right) \vee \left(\bigvee_{i \in [\![p,p+q[\![} \phi_i \right)$$

De même, on définit la conjonction $\bigwedge_{i \in [\![0,n[\![} \phi_i$ de n formules, avec la convention que cette conjonction est $\top$ si $n = 0$. (De cette façon, l'équivalence précédente reste vraie même si $p = 0$ ou $q = 0$.)

Définition 2.3.2 (Forme normale conjonctive/disjonctive)**.** *On appelle* clause *toute disjonction $\bigvee_{i \in [\![0,p[\![} l_i$ de littéraux. La formule $\bot$, disjonction de 0 littéraux est appelée* clause vide.

On appelle monôme *toute conjonction $\bigwedge_{i \in [\![0,p[\![} l_i$ de littéraux. La formule $\top$, disjonction de 0 littéraux est appelée* monôme vide.

On dit qu'une formule ϕ est en forme normale conjonctive *ou* sous forme clausale *s'il s'agit d'une conjonction de clauses $\bigwedge_{i \in [\![0,q[\![} C_i$. On dit qu'elle est en* forme normale conjonctive distinguée *si, dans chaque clause C_i, on trouve une et une seule fois chaque variable de la formule ϕ.*

On dit qu'une formule est en forme normale disjonctive *(resp.* forme normale disjonctive distinguée*) s'il s'agit d'une disjonction de monômes (resp. de monômes dans lesquels on trouve une et une seule fois chaque variable de ϕ).*

Remarque :
La négation d'une FNC est équivalente à une FND :

$$\neg \bigwedge_{i=0}^{p-1} \left(\bigvee_{j=0}^{q_i-1} l_{i,j} \right) \equiv \bigvee_{i=0}^{p-1} \left(\bigwedge_{j=0}^{q_i-1} \overline{l_{i,j}} \right)$$

De même la négation d'une FND est équivalente à une FNC :

$$\neg \bigvee_{i=0}^{p-1} \left(\bigwedge_{j=0}^{q_i-1} l_{i,j} \right) \equiv \bigwedge_{i=0}^{p-1} \left(\bigvee_{j=0}^{q_i-1} \overline{l_{i,j}} \right)$$

Proposition 2.3.3. *Un monôme (resp. une clause) est insatisfiable (resp. une tautologie) si et seulement s'il contient deux littéraux l_i et l_j tels que l_i soit une variable et l_j soit sa négation.*
Une FND est satisfiable si et seulement si elle contient au moins un monôme satisfiable.
Une FNC est une tautologie si et seulement si chacune de ses clauses en est une.

On peut donc tester en temps polynomial (et même facilement en temps quadratique) :

1. si une formule en FND est satisfiable ($DNFSAT$) ;
2. si une formule en FNC est une tautologie ($CNFTAUT$).

Théorème 2.3.4. *Pour toute formule ϕ, il existe une formule ψ en forme normale conjonctive (resp. disjonctive) vérifiant $\phi \equiv \psi$.*

Deux démonstrations classiques :
— par induction sur ϕ (pratique pour une implantation mais très pénible à rédiger)
— en construisant la formule à partir des tables de vérité (beaucoup plus facile)
Soit ϕ une formule comportant un ensemble de variables $\{ v_0, \dots, v_{k-1} \}$, que nous noterons V. Contruisons une formule ψ en FND équivalente à ϕ.
Pour $\rho : V \to \mathcal{B}$, on pose

$$F_\rho = \bigwedge_{i=0}^{k} l_i \qquad \text{où } l_i = \begin{cases} v_i & \text{si } \rho(v_i) = \mathbf{1} \\ \overline{v_i} & \text{sinon.} \end{cases}$$

Alors, pour tout $\rho' : V \to \mathcal{B}$, on a

$$[\![F_\rho]\!]_{\rho'} = \mathbf{1} \iff \rho' = \rho$$

Posons alors

$$\psi = \bigvee_{\substack{\rho \in \mathcal{B}^V \\ \rho \models \phi}} F_\rho$$

Alors pour tout $\rho' : V \to \mathcal{B}$, on a

$$[\![\psi]\!]_{\rho'} = \sum_{\substack{\rho \in \mathcal{B}^V \\ \rho \models \phi}} [\![F_\rho]\!]_{\rho'}$$

Si $\rho' \models \phi$, alors $[\![F_{\rho'}]\!]_{\rho'} = \mathbf{1}$ apparaît dans cette somme, sinon, tous les termes valent $\mathbf{0}$. Dans les deux cas, $[\![\psi]\!]_{\rho'} = [\![\phi]\!]_{\rho'}$.
ψ est bien en FND et $\psi \equiv \phi$. CQFD. Pour construire une formule en FNC, on travaille de façon duale :

Pour $\rho : V \to \mathcal{B}$, on pose :

$$G_\rho = \bigvee_{i=0}^{k} l_i \qquad \text{où } l_i = \begin{cases} v_i & \text{si } \rho(v_i) = \mathbf{0} \\ \overline{v_i} & \text{sinon.} \end{cases}$$

Alors, pour tout $\rho' : V \to \mathcal{B}$, on a

$$[\![G_\rho]\!]_{\rho'} = \mathbf{1} \iff \rho' \neq \rho$$

Posons alors

$$\psi' = \bigwedge_{\substack{\rho \in \mathcal{B}^V \\ \rho \not\models \phi}} G_\rho$$

Alors pour tout $\rho' : V \to \mathcal{B}$, on a

$$[\![\psi']\!]_{\rho'} = \prod_{\substack{\rho \in \mathcal{B}^V \\ \rho \not\models \phi}} [\![G_\rho]\!]_{\rho'}$$

Si $\rho' \not\models \phi$, alors $[\![F_{\rho'}]\!]_{\rho'} = \mathbf{0}$ apparaît dans ce produit, sinon, tous les termes valent $\mathbf{1}$.
Dans les deux cas, $[\![\psi']\!]_{\rho'} = [\![\phi]\!]_{\rho'}$.
ψ' est bien en FNC, et $\psi' \equiv \phi$. CQFD.

2.4 Résoudre SAT par transformation en FND

On a un algorithme pour transformer toute formule ϕ en une formule ψ équivalente. En particulier ϕ est satisfiable si et seulement si ψ l'est.
On dit qu'on peut *réduire* le problème SAT au problème $DNFSAT$.

2.5 Un problème ?

Vrai ou faux :

1. On a un algorithme efficace (quadratique) pour résoudre $DNFSAT$.
2. On peut réduire le problème SAT au problème $DNFSAT$.
3. Donc on peut résoudre SAT de façon polynomiale.
4. Or $SAT \in P \Rightarrow P = NP$.
5. Donc $P = NP$.

Exercice 2.5.1. *Pour $n \in \mathbb{N}^*$, notons $\phi_n = (X_1 \vee Y_1) \wedge \ldots \wedge (X_{\lfloor n/4 \rfloor} \vee Y_{\lfloor n/4 \rfloor})$ où les X_i et les Y_i sont des variables. ϕ_n est de taille $4\lfloor n/4 \rfloor - 1 = \Theta(n)$.*

La FND ψ_n de ϕ_n obtenue par l'algorithme précédent est de taille $\Theta(n \cdot 2^{n/4})$, donc le test de satisfiabilité de cette FND demandera un temps quadratique en $n \cdot 2^{n/4}$ qui n'est pas polynomial en n.

Si on veut réduire un problème à un autre et garder une complexité acceptable (polynomiale), il faut et il suffit que la réduction :

1. *donne un nouveau problème de taille polynomiale en la taille du problème initial ;*
2. *et fonctionne en temps polynomial.*

2.6 Unicité des FNC/FND ?

Il n'y a pas unicité, pour deux types de raisons :

1. Des raisons non-essentielles de syntaxe : si a et b sont deux atomes, on a $a \vee b \equiv b \vee a$ et ces deux clauses sont en FNC mais sont distinctes. Même problème avec $a \vee (b \vee c)$ et $a \vee (b \vee c)$
2. Des variables en doublon dans certaines clauses/conjonction de littéraux : $a \equiv a \wedge (b \vee \neg b)$.
3. Des problèmes liés à l'ajout/l'omission de certaines variables :

$$(a \vee c) \wedge (a \vee \neg c) \equiv a.$$

Proposition 2.6.1. *Pour tout monôme distingué M sur les variables $a_0, \ldots, a_{n-1}$, il existe une unique valuation ρ vérifiant $\rho \models M$, c'est la valuation $\rho : \{ a_0, \ldots, a_{n-1} \}$ définie par $\rho(a_i) = \mathbf{0}$ si $\neg a_i$ apparaît dans M, et $\rho(a_i) = \mathbf{1}$ sinon. Réciproquement, pour toute valuation, il existe un unique[7] monôme distingué M vérifiant $\rho \models M$: c'est le monôme F_ρ précédemment construit.*

Pour toute clause distinguée C sur les variables $a_0, \ldots, a_{n-1}$, il existe une unique valuation ρ vérifiant $\rho \not\models C$, c'est la valuation $\rho : \{ a_0, \ldots, a_{n-1} \}$ définie par $\rho(a_i) = \mathbf{1}$ si $\neg a_i$ apparaît dans M, et $\rho(a_i) = \mathbf{0}$ sinon. Réciproquement, pour toute valuation, il existe une unique[8] clause distinguée C vérifiant $\rho \not\models C$, c'est la clause G_ρ précédemment construite.

Théorème 2.6.2. *Pour toute ϕ dont les variables sont parmi $a_0, \ldots, a_{n-1}$, il existe une formule ψ en forme normale disjonctive (resp. conjonctive) distinguée sur $a_0, \ldots, a_{n-1}$ équivalente à ϕ.*

Cette formule est unique, au sens où l'ensemble de ses monômes (resp. clauses) est exactement l'ensemble des monômes (resp. clauses) associés aux valuations $\rho : \{ a_0, \ldots, a_n \} \to \mathcal{B}$ tels que $\rho \models \phi$ (resp. $\rho \not\models \phi$).

Démonstration : on a déjà vu l'existence (on avait bien des formes distinguées), il reste à voir l'unicité.

Proposition 2.6.3. *L'unique FND distinguée équivalente à une formule insatisfiable est la disjonction vide de 0 monômes.*

L'unique FNC distinguée équivalente à une tautologie est la conjonction vide.

Démonstration : application directe du théorème.
Rem :
— Tester si une FND distinguée (resp. FNC distinguée) est satisfiable (resp. tautologie) se fait donc en temps constant !
— Cela n'a aucune utilité pratique, les FND distinguées étant longues à calculer.

2.7 SAT : classifions

SAT : une formule est-elle satisfiable ?
On peut s'intéresser à la satisfiabilité d'ensembles de formules plus restreints :

7. À permutation près des littéraux.
8. À permutation près des littéraux.

DNFSAT pour les FND [polynomial].

DDNFSAT pour les FND distinguées [polynomial, trivial].

CNFSAT pour les FNC.

3SAT/3CNFSAT pour les FNC où chaque clause contient au plus 3 littéraux.

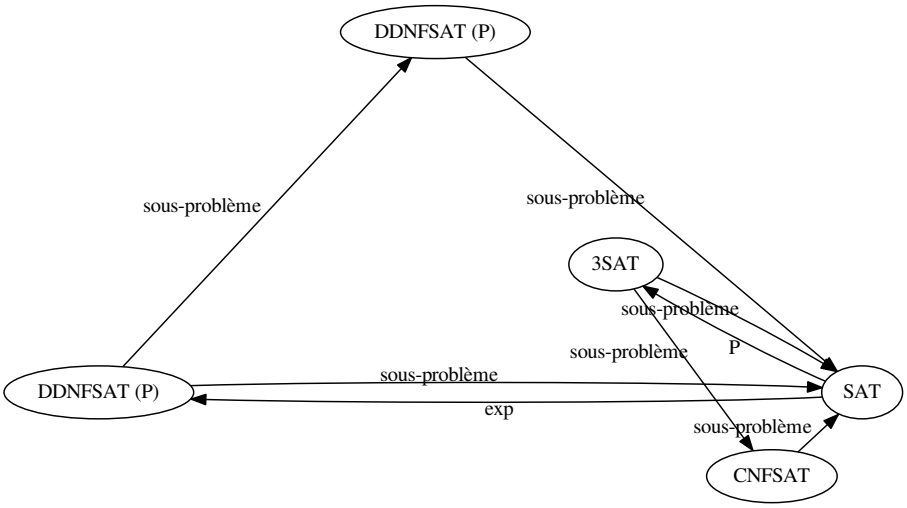
— Réduction triviale de chacun de ces problèmes à SAT (ce sont des sous-problèmes).
— Réduction de SAT à ces sous-problèmes ?

Réduction déjà vue de SAT à DNFSAT et au problème des FND distinguées : méthode trouvée pas polynomiale ($\Omega(2^n)$).

Réduction évidente de $3SAT$ à $CNFSAT$. On peut clairement réduire $3SAT$ à $CNFSAT$, puisque c'est un cas particulier, et $CNFSAT$ à SAT.

Peut-on réduire polynomialement SAT à $CNFSAT$?

Oui ! On peut même le réduire polynomialement à $3SAT$!

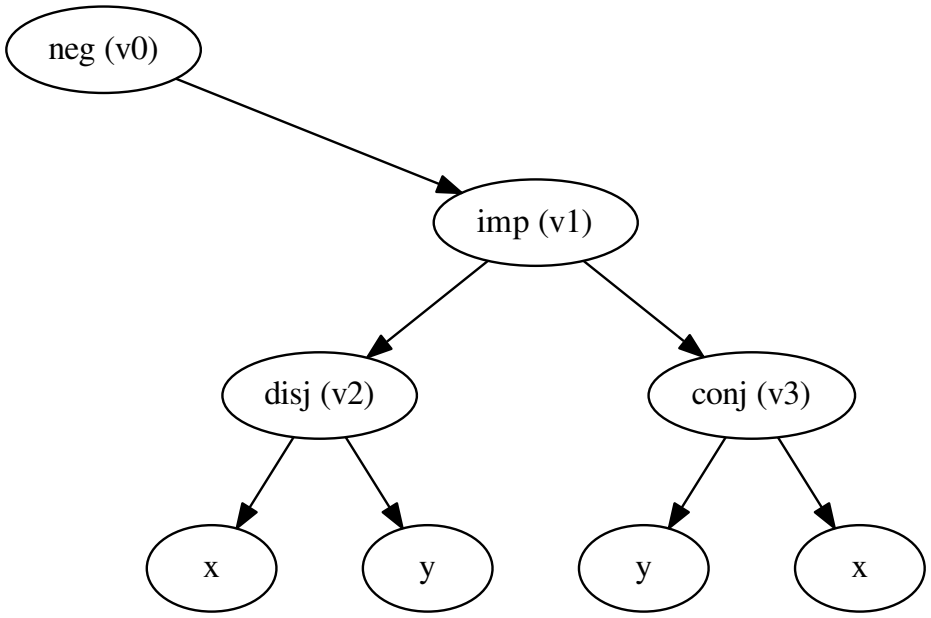


Comment réduire SAT à $3SAT$ polynomialement ?

2.7.1 Transformation de Tseitin

Étant donné une formule ϕ, on construit une formule ψ en $3CNF$ *équisatisfiable* à ϕ, de taille linéaire en la taille de ϕ, en temps linéaire en la taille de ϕ.

Exemple : considérons $\phi = \neg((x \vee y) \Rightarrow (y \wedge x))$. On représente cette formule sous forme d'un arbre en étiquetant chaque nœud interne avec une nouvelle variable :



On remarque alors que ϕ est satisfiable si et seulement si la conjonction des formules suivantes l'est :

(1) v_0 ;
(2) $v_0 \iff \neg v_1$;
(3) $v_1 \iff (v_3 \Rightarrow v_4)$;
(4) $v_2 \iff (x \vee y)$;
(5) $v_3 \iff (y \wedge x)$;

Ces formules sont respectivement équivalentes aux formules suivantes qui sont en $3CNF$:

(1) v_0 ;
(2) $(v_0 \vee v_1) \wedge (\neg v_0 \vee \neg v_1)$;
(3) $(\neg v_1 \vee \neg v_3 \vee v_4) \wedge (v_1 \vee \neg v_3 \vee \neg v_4) \wedge (v_1 \vee v_3)$;
(4) $(\neg v_2 \vee x \vee y) \wedge (v_2 \vee \neg x) \wedge (v_2 \vee \neg y)$;
(5) $(\neg v_3 \vee x) \wedge (\neg v_3 \vee y) \wedge (v_3 \vee \neg x \vee \neg y)$;

La formule initiale est donc équisatisfiable à

$$
\begin{array}{ll}
 & v_0 \\
\wedge & (v_0 \vee v_1) \wedge (\neg v_0 \vee \neg v_1) \\
\wedge & (\neg v_1 \vee \neg v_3 \vee v_4) \wedge (v_1 \vee \neg v_3 \vee \neg v_4) \wedge (v_1 \vee v_3) \\
\wedge & (\neg v_2 \vee x \vee y) \wedge (v_2 \vee \neg x) \wedge (v_2 \vee \neg y) \\
\wedge & (\neg v_3 \vee x) \wedge (\neg v_3 \vee y) \wedge (v_3 \vee \neg x \vee \neg y)
\end{array}
$$

Et cette formule est une formule en $3CNF$.

La transformation de Tseitin consiste simplement à appliquer cette idée de façon systématique. Formellement, on considère une formule ϕ contenant les variables v_0, …, v_{n-1}. Pour tout $x \in [\![0, n[\![$, la variable v_i est une sous-formule ϕ_i de la formule ϕ (pouvant éventuellement apparaître plusieurs fois dans ϕ).

On note alors ϕ_n, …, ϕ_{N-1} les autres sous-formules de ϕ, ϕ_{N-1} désignant la formule ϕ elle-même. Pour chacune de ces sous-formules ϕ_j, pour $j \in [\![n, N[\![$, on introduit une variable supplémentaire v_j.

On construit alors, pour tout $j \in [\![n, N[\![$, une formule F_j en fonction de la forme de la formule ϕ_j comme ceci :

ϕ_j	F_j
$\neg\phi_k$	$v_j \iff \neg v_k$
$\phi_k \wedge \phi_l$	$v_j \iff (v_k \wedge v_l)$
$\phi_k \vee \phi_l$	$v_j \iff (v_k \vee v_l)$
$\phi_k \Rightarrow \phi_l$	$v_j \iff (v_k \Rightarrow v_l)$
$\phi_k \iff \phi_l$	$v_i \iff (v_k \iff v_l)$
$\bot$	$v_j \iff \bot$
$\top$	$v_j \iff \top$

Alors, si ϕ est satisfaite par une valuation ρ sur $v_0, \dots v_{n-1}$, on peut étendre ρ en une valuation ρ' sur $v_0, \dots, v_{n-1}, v_n, v_{n+11}, \dots, v_{N-1}$ en posant, pour tout $j \in [\![n, N[\![$, $\rho'(v_j) = [\![\phi_j]\!]_\rho$ et ρ' satisfait alors simultanément toutes les formules F_j pour $j \in [\![n, N[\![$ ainsi que v_{N-1}.

Réciproquement, si on peut trouver une valuation ρ' sur a_0, …, a_{n-1}, v_0, v_1,… satisfaisant simultanément toutes les formules F_i pour $i \in [\![0, N[\![$ ainsi que v_{N-1} alors en notant ρ la restriction de ρ' à $\{ a_0, \dots, a_{n-1} \}$, on a pour tout $i \in [\![0, N[\![$, $[\![\phi_i]\!]_\rho = \rho'(v_i)$ et en particulier $[\![\phi]\!]_\rho = \rho'(v_{N-1}) = \mathbf{1}$.

De plus, les formules F_j sont de taille bornée et comportent chacune au plus 3 littéraux. De plus chacune des 7 formes de formules possibles pour F_j peut être mise sous forme normale conjonctive dont toute clause comporte au plus 3 littéraux et il y a un nombre fini de formes de formules F_j différentes possibles. On peut donc transformer chaque F_j en une FND F'_i en temps constant (donc au total en $O(N) = O(n)$).

2.8 Et alors ?

Les problèmes SAT et $3 - CNFSAT$ sont équivalents au sens où on sait réduire chacun des deux en l'autre, en temps polynomial.

Malheureusement, on ne connaît pas d'algorithme polynomial pour résoudre $3-CNFSAT$. On conjecture même qu'il n'y en a pas.

2.9 P et NP

On dit qu'un problème est NP-dur, si tout problème NP peut se réduire polynomialement à ce problème.

On dit qu'un problème est NP-complet s'il est NP-dur et NP.

Théorème 2.9.1 (Cook-Levin). *Le problème SAT est NP-complet.*

La démonstration (et même les définitions précises de NP et NP-dur) sont hors de portée du programme.

Proposition 2.9.2. *Les problèmes $CNFSAT$ et $3-CNFSAT$ sont NP-complets.*

Démonstration facile en raison des réductions polynomiales de SAT à $3-CNFSAT$ et à $CNFSAT$ … à partir du moment où on a les définitions !

Proposition 2.9.3. *S'il existe un algorithme polynomial pour résoudre SAT (ou $3-SAT$), alors $NP=P$.*

Démonstration : là encore hors programme. L'idée est que $P \subseteq NP$ est évident et que d'autre part, SAT étant NP-complet, tout problème NP peut polynomialement se réduire à SAT, donc si SAT était résoluble en temps polynomial, tout problème NP le serait aussi, donc on aurait $NP \subseteq P$, donc $P=NP$.

On conjecture plutôt $P \neq NP$ mais c'est un problème ouvert depuis qu'il a été posé en 1970.

