INFOF-302 — Informatique fondamentale

R. Petit

Année académique 2016 - 2017

Table des matières

1 Introduction et Logique

Définition 1.1. *Réduire* un problème de décision A en un problème de décision B correspond à trouver un algorithme permettant d'encoder toute entrée I_A du problème A en une entrée I_B du problème B telle que I_A a une solution pour le problème A si et seulement si I_B a une solution pour le problème B.
Définition 1.2. *Réduire* un problème général A en un problème B correspond à trouver un algorithme d'encodage de toute entrée I_A du problème A en une entrée I_B du problème B et de décodage de toute sortie O_B en une sortie O_A.
Axiome 1.3. *Pour $\prec$, une relation d'ordre sur la priorité des opérateurs logiques. On prend :*

$$\Leftrightarrow \quad \prec \quad \Rightarrow \quad \prec \quad \vee \quad \prec \quad \wedge \quad \prec \quad \neg.$$

Afin d'étudier une formule logique, on peut construire son *arbre de lecture* en séparant la formule aux opérateurs, par ordre croissant de priorité.
Définition 1.4. Pour P, un ensemble de propositions, on appelle *fonction d'interprétation* (ou *valuation*) toute fonction :

$$V : P \to \{0,1\} : p \mapsto V(p).$$

Définition 1.5. Soit ϕ une formule bien formée sur un ensemble de propositions P. On définit sa *valeur de vérité* évaluée en $V \in \{0,1\}^P$ par la fonction :

$$[\![\phi]\!]_\cdot : \{0,1\}^P \to \{0,1\} : V \mapsto [\![\phi]\!]_V,$$

dont la valeur est induite syntaxiquement.

Pour V une valuation, lorsque $[\![\phi]\!]_V = 1$, on note $V \models \phi$, que l'on lit V *satisfait* ϕ.
Définition 1.6. Soit ϕ, une formule sur P.
— lorsque $[\![\phi]\!]_\cdot^{-1}(\{1\}) \neq \emptyset$, on dit que ϕ est *satisfaisable* ;
— lorsque $[\![\phi]\!]_\cdot^{-1}(\{1\}) = \{0,1\}^P$, on dit que ϕ est *valide*.
Lemme 1.7. *Soit ϕ, une formule sur P. Si ϕ est valide, alors ϕ est satisfaisable.*
Définition 1.8. Soient $n \in \mathbb{N}^*$, $\{\phi_i\}_{i\in[\![1,n]\!]}$ et ϕ, des formules sur P. On dit que ϕ est une conséquence logique de $\{\phi_i\}_{i\in[\![1,n]\!]}$ lorsque la formule :

$$\bigwedge_{i=1}^{n} \phi_i \Rightarrow \phi$$

est valide. On note cela $\phi_1, \ldots, \phi_n \models \phi$.
Définition 1.9. Deux formules ϕ et ψ sur P sont dites *équivalentes* lorsque $\phi \Leftrightarrow \psi$ est valide. On note cela $\phi \equiv \psi$.
Théorème 1.10. *Une formule ϕ sur P est valide si et seulement si sa négation est non-satisfaisable.*

Démonstration. Soit ϕ, une formule valide sur P. On sait alors que $[\![\phi]\!]_\cdot^{-1}(\{1\}) = \{0,1\}^P$. On en déduit que :

$$[\![\phi]\!]_\cdot^{-1}(\{0\}) = \{0,1\}^P \setminus [\![\phi]\!]_\cdot^{-1}(\{1\}) = \emptyset.$$

Or $\forall V \in \{0,1\}^P : [\![\phi]\!]_V = 1 - [\![\neg\phi]\!]_V$. On en déduit que :

$$[\![\neg\phi]\!]_\cdot^{-1}(\{1\}) = [\![\phi]\!]_\cdot^{-1}(\{0\}) = \emptyset.$$

Idem pour $\Leftarrow$ □

Remarque. On en déduit qu'un algorithme qui détermine la satisfaisabilité d'une expression permet également de déterminer la validité.

Définition 1.11. Un *littéral* est soit une proposition $x \in P$, soit la négation $\neg x$ d'une proposition.
Définition 1.12. Un ensemble S de littéraux est dit *satisfaisable* lorsqu'il ne contient pas une paire de littéraux complémentaires, i.e. :

$$\forall x \in S : \neg x \notin S.$$

Pour déterminer la satisfaisabilité d'une formule, on peut créer son arbre sémantique par application des $\wedge$-règles et $\vee$-règles. Pour cela, on part de de la formule, et on applique le pas de simplification soit dur une conjonction, soit sur une disjonction (en ayant transformé tous les autres opérateurs en conjonctions/disjonctions au préalable).
Exemple 1.1.

$$\begin{aligned}\phi :=& (x \vee y) \wedge (\neg x \wedge \neg y)\\ & \big\{(x \vee y) \wedge (\neg x \wedge \neg y)\big\}\\ & \big\{(x \vee y, \neg x \wedge y\big\}\\ & \{x \vee y, \neg x, \neg y\}\\ & \{x, \neg x, \neg y\} \quad \{y, \neg x, \neg y\}.\end{aligned}$$

Tous les sous-ensembles de littéraux sont non-satisfaisables, donc la formule ϕ est non-satisfaisable (i.e. $\neg\phi$ est valide).

L'algorithme de création de tableau sémantique pour SAT est le suivant. Soit ϕ une formule sur P. On construit l'arbre T_ϕ comme suit :

1. Initialisation : l'arbre est défini par $T_\phi = \{\phi\}$.
2. Tant qu'il existe une feuille $\mathcal{L} \in T_\phi$ telle que $\exists \psi \in \mathcal{L}$, une formule simplifiable (donc contenant une conjonction ou une disjonction) :
   — si ψ est simplifiable par une $\wedge$-règle, on ajoute une feuille $\mathcal{L}'$ à $\mathcal{L}$ telle que :

   $$\mathcal{L}' = \big(\mathcal{L} \setminus \{\psi\}\big) \cup \{\psi_1, \psi_2\},$$

   pour ψ_1 et ψ_2, les deux sous-formules simplifiées de ψ ;
   — si ψ est simplifiable par une $\vee$-règle, on ajoute deux feuilles $\mathcal{L}_1$ et $\mathcal{L}_2$ à $\mathcal{L}$ telles que :

   $$\forall i \in [\![1,2]\!] : \mathcal{L}_i = \big(\mathcal{L} \setminus \{\psi\}\big) \cup \{\psi_i\},$$

   avec ψ_1 et ψ_2, les deux sous-formules simplifiées de $\psi.i$
3. S'il existe $\mathcal{L} \in T_\phi$, une feuille telle que $\mathcal{L}$ est satisfaisable, alors retourner `SATISFAISABLE`, sinon retourner `NON-SATISFAISABLE`.

2 Déduction naturelle

Définition 2.1. Soient $n \in \mathbb{N}^*$, $\{\phi_i\}_{i\in[\![1,n]\!]}$, ψ des formules sur P. Si ψ peut être dérivé des $\{\phi_i\}_{i\in[\![1,n]\!]}$, on appelle ces derniers des *prémisses* et ψ la *conclusion*. On note cela :

$$\phi_1, \ldots, \phi_n \vdash \psi.$$

On appelle cela un *séquent*.

La déduction naturelle fonctionne par élimination et introduction successives d'opérateurs.

— $\frac{\phi \quad \psi}{\phi\wedge\psi}\wedge_i$ se lit *si* ϕ *et si* ψ, *alors* ϕ *et* ψ ;

— $\frac{\phi \wedge \psi}{\psi} \wedge_{e_1}$ se lit *si* ϕ *et* ψ, *alors en particulier* ψ.

De même pour $\wedge_{e_2}$, et $\vee_i$. La double négation fonctionne également par introduction et élimination.

La règle d'élimination de l'implication s'appelle *Modus Ponens* (MP) et la règle d'élimination de l'implication par contraposée s'appelle *Modus Tollens* (MT) :

— $\frac{\phi \quad \phi \Rightarrow \psi}{\psi}$ MP se lit *si* ϕ *et si* ϕ *implique* ψ, *alors* ψ ;

— $\frac{\neg\psi \quad \phi \Rightarrow \psi}{\neg\phi}$ MT se lit *si* ϕ *implique* ψ *et non-*ψ, *alors non-*ϕ.

Afin d'introduire l'implication, on se sert du MT :

1.	$x \Rightarrow y$	prémisse
2.	$\neg y$	hyp.
3.	$\neg x$	MT 1, 2 ; fin hyp. 2
4.	$\neg \Rightarrow \neg y$	$\Rightarrow_i$ 2, 3

On déduit donc $\neg y \Rightarrow \neg x$.

De manière plus générale, si en faisant l'hypothèse ϕ, on arrive à la conclusion ψ, pour ϕ, ψ deux formules sur P, alors :

$$\dfrac{\begin{array}{ll}\phi & \text{hyp.}\\ \vdots & \\ \psi & \text{fin hyp.}\end{array}}{\phi \Rightarrow \psi} \Rightarrow_i$$

Remarque. Les prémisses et les hypothèses sont fondamentalement différentes ! Une hypothèse peut être émise même sans hypothèse, e.g. :

1.	p	hyp.
2.	$\neg\neg p$	$\neg\neg_i$ 1 ; fin hyp. 1
3.	$p \Rightarrow \neg\neg p$	$\Rightarrow_i$ 1, 2

On peut donc déduire de cela que $\vdash p \Rightarrow \neg\neg p$.

Afin d'éliminer la disjonction, on procède de la sorte :

$$\dfrac{\phi_1 \vee \phi_2 \quad \begin{array}{ll}\phi_1 & \text{hyp.}\\ \vdots & \\ \psi & \text{fin hyp.}\end{array} \quad \begin{array}{ll}\phi_2 & \text{hyp.}\\ \vdots & \\ \psi & \text{fin hyp.}\end{array}}{\psi} \vee_e$$

Définition 2.2. Toute formule ϕ sur P telle que $\vdash \phi$ est appelée *théorème*.

Un théorème n'a donc pas besoin de prémisse. De plus, $\vdash \phi$ si et seulement si $\models \phi$, donc les théorèmes coïncident avec les formules valides.

Lemme 2.3. *Soient* $n \in \mathbb{N}^*$, $\{\phi_i\}_{i \in [\![1,n]\!]}$, ψ *des formules sur* P. *Alors :*

$$\phi_1, \ldots, \phi_n \vdash \psi \qquad \textit{si et seulement si} \qquad \vdash \phi_1 \Rightarrow \phi_2 \Rightarrow \ldots \Rightarrow \phi_n \Rightarrow \psi.$$

Afin d'introduire la négation, on suppose une formule, et on en dérive $\bot$, ce qui permet d'en déduire sa négation. Cela se formule :

$$\dfrac{\begin{array}{ll}\phi & \text{hyp.}\\ \vdots & \\ \bot & \text{fin hyp.}\end{array}}{\neg\phi} \neg_i$$

Un raisonnement par l'absurde (reductio ad absurdum) se formalise par une introduction de double négation :

$$\dfrac{\begin{array}{ll}\neg\phi & \text{hyp.}\\ \vdots & \\ \bot & \text{fin hyp.}\end{array}}{\neg\neg\phi}\,\text{RAA},$$

qui se note :

$$\frac{\phi}{\neg\neg\phi}\neg\neg_i.$$

Ce qui est également équivalent à la loi du tiers exclus, que l'on peut formuler $\phi \Rightarrow \neg(\neg\phi)$, ou encore :

$$\frac{}{\phi \vee \neg\phi}\,\text{LEM}.$$

Théorème 2.4. *Soient* $n \in \mathbb{N}^*$, $\{\phi_i\}_{i\in[\![1,n]\!]}$, ψ *des formules sur* P. *Alors :*
(Adéquation) $(\phi_1,\dots,\phi_n \vdash \psi) \Rightarrow \big(\phi_1,\dots,\phi_n \models \psi\big)$;
(Complétude) $\big(\phi_1,\dots,\phi_n \models \psi\big) \Rightarrow (\phi_1,\dots,\phi_n \vdash \psi)$.

Définition 2.5. Une formule ϕ sur P est dit en *forme normale conjonctive* (FNC ou CNF) lorsqu'elle s'exprime comme suit :

$$\phi \equiv \bigwedge_{i=1}^{n}\left(\bigvee_{j=1}^{m}\ell_{ij}\right),$$

et est dite en *forme normale disjonctive* (FND ou DNF) lorsqu'elle s'exprime comme suit :

$$\phi \equiv \bigvee_{i=1}^{n}\left(\bigwedge_{j=1}^{m}\ell_{ij}\right),$$

avec ℓ_{ij} des littéraux sur P.

Proposition 2.6. *Deux formules* ϕ *et* ψ *sur* P *sont équivalentes si et seulement si* $[\![\phi]\!]_. \equiv [\![\psi]\!]$ *sur* $\{0,1\}^P$.

Théorème 2.7. *Soit* ϕ *une formule sur* P. *Il existe* ϕ_C *et* ϕ_D *respectivement sous forme conjonctive et disjonctive telles que :*

$$\phi \equiv \phi_C \equiv \phi_D.$$

3 Coupures

Définition 3.1. Une formule ϕ sur P est une *clause* si $\phi = \bot$, ou si il existe $\ell_1,\dots,\ell_n$ littéraux tels que $\phi = \bigvee_{i=1}^{n}\ell_i$.

Définition 3.2. Une clause C est dite *positive* si tous les littéraux apparaissent positivement, et est dite *négative* si tous les littéraux apparaissent négativement. Elle est dite *tautologique* si elle contient deux littéraux complémentaires, i.e. si $\phi \equiv \top$.

Remarque. Une clause C est non-satisfaisable si et seulement si $C = \bot$.

Définition 3.3.

— Un ensemble de clauses S est dit *satisfaisable par une valuation* $V \in \{0,1\}^P$, noté $V \models S$ lorsque :

$$\forall C \in S : V \models C\ ;$$

— un ensemble S de clauses est dit *satisfaisable* s'il existe $V \in \{0,1\}^*$ t.q. $V \models S$;
— un ensemble S de clauses est dit *valide* si $\forall V \in \{0,1\}^P : V \models S$;

— une clause C est dite *conséquence d'un ensemble de clauses* S lorsque :

$$\forall V \in \{0,1\}^p : \big(V \models S \Rightarrow V \models C\big),$$

que l'on note $S \models C$.

Proposition 3.4. *Soient S un ensemble de clauses, et C une clause tautologique. S est satisfaisable (resp. valide) si et seulement si $S \cup \{C\}$ est satisfaisable (resp. valide).*

Proposition 3.5. *Toute formule ϕ sur P est équivalente à un nombre fini de clauses.*

Démonstration. Il existe ϕ_C telle que $\phi \equiv \phi_D$. Or ϕ_C est une conjonction de clauses. □

Définition 3.6. Soient C_1, C_2 deux clauses telles qu'il existe p, un littéral tel que :

$$C_1 = \bigvee_{i=1}^{n_1} \ell_{1i} \vee p \qquad \text{et} \qquad C_2 = \bigvee_{i=1}^{n_2} \ell_{2i} \vee (\neg p).$$

La règle de coupure dit que la clause $C_3 = \bigvee_{i=1}^{2} \bigvee_{j=1}^{n_i} \ell_{ij}$ est conséquence de C_1 et C_2, que l'on note $C_1, C_2 \vdash_p^c C_3$.

Théorème 3.7. *Soient C_1, C_2, C_3, trois clauses telles qu'il existe p tel que $C_1, C_2 \vdash_p^c C_3$. C_3 est conséquence logique de C_1 et C_2, i.e. :*

$$C_1 \wedge C_2 \models C_3.$$

Démonstration. Soit $V \in \{0,1\}^p$ telle que $V \models C_1 \wedge C_2$. Si $V(p) = 0$, alors $V \models C_2$. Or $V \models C_1 \wedge C_2$, donc il existe $i \in [\![1, n_1]\!]$ tel que $V(\ell_{1i}) = 1$. Donc $V \models C_3$.

Idem si $V(p) = 1$, on sait que $V \models C_1$, mais puisque $V \models C_1 \wedge C_2$, il doit exister $i \in [\![1, n_2]\!]$ tel que $V(\ell_{2i}) = 1$, et donc $V \models C_3$. □

Définition 3.8. Soit $C = \bigvee_{i=1}^{n} \ell_i$, une clause, pour $\{\ell_i\}_{i \in [\![1,n]\!]}$ des littéraux. S'il existe j, j' tels que $j \neq j'$ et $\ell_j \equiv \ell_{j'}$, alors la règle de retrait de redondance dit :

$$C \vdash_r \bigvee_{\substack{i \in [\![1,n]\!] \\ i \neq j}} \ell_i.$$

Définition 3.9. Soient S, un ensemble de clauses, et C une clause. Une preuve de C par coupures de S est une suite finie de clauses $\{C_i\}_{i \in [\![1,n]\!]}$ où $C_n = C$, et $\forall i \in [\![1, n]\!]$:

— soit $C_i \in S$;
— soit $\exists (k, \ell) \in [\![1, n]\!]^2$ t.q. $k < \ell < i$ et $C_k, C_\ell \vdash^c C_i$;
— soit $\exists k \in [\![1, i-1]\!]$ t.q. $C_k \vdash_r C_i$.

On note $S \vdash^c C$ le fait que C soit déductible par coupure de S.

Théorème 3.10. *Soient S un ensemble de clauses et C une clause. Si $S \vdash^c C$, alors $S \models C$.*

Définition 3.11. Une réfutation d'un ensemble de clauses S par coupure est une dérivation de la clause vide à partir de S.

Théorème 3.12. *S'il existe une réfutation de S, un ensemble de clauses, par coupure, alors S est non-satisfaisable.*

Afin de prouver $\psi_1, \ldots, \psi_n \models \phi$ pour $\{\psi_i\}_{i \in [\![1,n]\!]}$ et ϕ des formules sur R, on construit S l'ensemble de clauses équivalent à $\bigwedge_{i=1}^{n} \psi_i$, et S' l'ensemble de clauses équivalent à $\neg\phi$. Pour D l'ensemble des clauses dérivables depuis $S \cup S'$, si $\bot \in D$, alors $\bigwedge_{i=1}^{n} \psi_i \wedge \neg\phi$ est non-satisfaisable, et donc :

$$\psi_1, \ldots, \psi_n \models \phi.$$

Théorème 3.13. *Si un ensemble de clauses S est non-satisfaisable, alors il existe une réfutation finie de S par coupure.*

Un SAT-solver est un programme qui décide le problème SAT. Si son entrée S est satisfaisable, il retourne une valuation qui la satisfait.

Le problème SAT est $\mathcal{NP}$-complet, donc il est pensé qu'il n'existe pas d'algorithme de résolution en temps polynomial.

4 DPLL

DPLL est un algorithme de résolution du problème SAT.
Définition 4.1. Une valuation partielle est une assignation arbitraire d'un littéral x à 0 ou 1, noté $x/1$ ou $x/0$.

Pour C une clause, x un littéral de C et $b \in \{0,1\}$, une valuation partielle de la clause C, notée $C[x/b]$ est obtenue par :

— si C ne contient ni x ni $\neg x$, alors $C[x/b] = C$;
— si $C \in \{x, \neg x\}$, alors $C[x/b] = \begin{cases} \top & \text{si } (C = x \text{ et } b = 1) \text{ ou } (C = \neg x \text{ et } b = 0) \\ \bot & \text{sinon} \end{cases}$;
— si C contient x et $b = 1$ ou si C contient $\neg x$ et $b = 0$, alors $C = \top$;
— sinon on retire les occurrences de x ou $\neg x$ de C.

Donc pour $\phi = \bigwedge_{i=1}^{n} C_i$, on définit :

$$\phi[x/b] = \bigwedge_{i=1}^{n} C_i[x/b] = \bigwedge_{\substack{i \in [\![1,n]\!] \\ C_i[x/b] \neq \top}} C_i[x/b].$$

On remarque donc que s'il existe $i \in [\![1,n]\!]$ tel que $C_i[x/b] = \bot$, alors $\phi[x/b] = \bot$, et si $\forall i \in [\![1,n]\!] : C_i[x/b] = \top$, alors $\phi[x/b] = \top$.

De plus, $\phi[x/b]$ est une formule sur $P \setminus \{x\}$, donc on peut lui réappliquer une valuation partielle y/b'.

L'algorithme DPLL fonctionne par valuation partielles sur un littéral (appelé *pivot*) et teste récursivement jusqu'à déterminer une valuation qui satisfait la clause.
Définition 4.2. Une clause C est dite *unitaire* si $C = x$ ou $C = \neg x$.
Remarque. Une clause unitaire force le choix du pivot, et de sa valuation. De plus, après valuation partielle, une clause peut devenir une clause unitaire. Les SAT-solvers font donc de la *propagation de clauses unitaires*.

Également, si un littéral apparait toujours positivement (ou négativement) dans les clauses, alors il est possible de les éliminer du problème par valuation partielle à 0 si négatif, et 1 si positif. Le procédé DPLL peut donc être exprimé comme suit :

1. Si $\phi = \top$, retourner 1 et si $\phi = \bot$, retourner 0.
2. Si ϕ contient une clause unitaire C_1, retourner $\begin{cases} \text{DPLL}(\phi[x/1]) & \text{si } C_1 = x \\ \text{DPLL}(\phi[x/0]) & \text{si } C_1 = \neg x \end{cases}$.
3. Si ϕ contient un littéral x de polarité constante π, alors retourner $\begin{cases} \text{DPLL}(\phi[x/1]) & \text{si } \pi = + \\ \text{DPLL}(\phi[x/0]) & \text{si } \pi = - \end{cases}$.
4. Sinon, choisir un littéral x au hasard, et renvoyer $\text{DPLL}(\phi[x/0]) \vee \text{DPLL}(\phi[x/1])$.

4.1 Transformation de Tseitin

Lorsqu'une formule n'est pas simple à mettre sous FNC (e.g. si elle est sous FND), on peut lui appliquer la transformation de Tseitin. Soit $\phi \equiv \bigwedge_{i=1}^{n} C_i$, une formule sous FND, avec les C_i, des conjonctions de littéraux. Sa transformation de Tseitin donne :

$$\psi \coloneqq \mathscr{T}(\phi) \equiv \left(\bigvee_{i=1}^{n} x_i\right) \wedge \left(\bigwedge_{i=1}^{n} (x_i \Leftrightarrow C_i)\right).$$

le but est donc d'ajouter de nouvelles variables x_i, en les forçant à être équivalentes aux clauses C_i.

Une fois la transformation appliquée, il est facile de la mettre sous FNC. Pour $C_i \equiv \bigwedge_{j=1}^{n_i} \ell_{ij}$:

$$\begin{aligned} x_i \Leftrightarrow C_i &\equiv (x_i \Rightarrow C_i) \wedge (C_i \Rightarrow x_i) \\ &\equiv \left(\neg x_i \vee \bigwedge_{j=1}^{n_i} \ell_{ij}\right) \wedge \left(\bigvee_{j=1}^{n_i} \neg \ell_{ij} \vee x_i\right) \\ &\equiv \bigwedge_{j=1}^{n_i} (\neg x_i \vee \ell_{ij}) \wedge \left(\bigvee_{j=1}^{n_i} \ell_{ij} \vee x_i\right). \end{aligned}$$

4.2 Contrainte exactement une

Afin d'encoder la contrainte *exactement une parmi* n, la solution naïve est :

$$\left(\bigvee_{i=1}^{n} x_i\right) \wedge \bigwedge_{i=1}^{n} \bigwedge_{j=i+1}^{n} (\neg x_i \vee \neg x_j).$$

Cela fait $1 + n(n-1)/2$ contraintes. Si $n = 2^k$, une solution plus intéressante est d'introduire les variables $\{b_i\}_{i \in [\![1,k]\!]}$, et d'introduire la notation $\overline{\alpha_1 \dots \alpha_k} = \overline{i-1}$ pour la représentation binaire de $i-1$. On peut alors exprimer :

$$\left(\bigvee_{i=1}^{n} x_i\right) \wedge \bigwedge_{i=1}^{n} (x_i \Rightarrow B_i),$$

avec B_i défini pour $i \in [\![1, n]\!]$ comme étant :

$$B_i = \bigwedge_{j=1}^{k} \ell_{ij},$$

pour $\ell_{ij} = \begin{cases} b_j & \text{si } \alpha_j = 1 \\ \neg b_j & \text{sinon} \end{cases}$. Puisque :

$$\bigwedge_{i=1}^{n} (x_i \Rightarrow B_i) \equiv \bigwedge_{i=1}^{n} \bigwedge_{j=1}^{k} (\neg x_i \vee \ell_{ij}),$$

cela fait descendre le nombre de contraintes à $1 + nk = 1 + n \log_2(n)$. L'idée est d'utiliser l'unicité de la représentation binaire d'un nombre pour garantir qu'une seule des variables est assignée positivement.

Dès lors, si $V : \{0,1\}^{\{x_i\}_{i \in [\![1,n]\!]} \cup \{b_i\}_{i \in [\![1,k]\!]}} \to \{0,1\}$ est une solution, alors $\left|V^{-1}(\{1\})\right| = 1$.

Un principe souvent valable pour les SAT-solvers est qu'il est préférable d'ajouter des variables afin de minimiser le nombre de clauses.

5 Complexité

Définition 5.1. Un alphabet Σ est un ensemble fini de symboles. On note Σ^* l'ensemble des mots sur Σ, i.e. :

$$\Sigma^* = \lim_{n \to +\infty} \bigsqcup_{\ell=1}^{n} \Sigma^\ell.$$

Définition 5.2. Tout alphabet permet de former un mot commun appelé le *mot vide*, noté ϵ.
Définition 5.3. Un langage sur un alphabet est un sous-ensemble $L \subseteq \Sigma^*$.

Un problème de décision est un langage P sur un alphabet Σ. On peut alors définir :

$$\chi_P : \Sigma^* \to \{0,1\} : m \mapsto \begin{cases} 1 & \text{si } m \in P \\ 0 & \text{sinon.} \end{cases}$$

La représentation abstraite du codage sert en théorie de la complexité et de la calculabilité, mais bien souvent, il n'est pas nécessaire de s'y restreindre. Notons que l'encodage peut influencer la complexité.
Définition 5.4. Un problème de décision $P \subset \Sigma^*$ est dit *décidé par un algorithme* A si pour tout $m \in \Sigma^*$, $A(m)$ termine et retourne 1 si $m \in P$ et 0 si $m \in \Sigma^* \setminus P$. Si un tel algorithme existe, A est dit décidable.
Définition 5.5. La classe $\mathcal{P}$ définit les problèmes décidables en temps polynomial, i.e. $P \subset \Sigma^*$ est dans $\mathcal{P}$ s'il existe $k \in \mathbb{N}^*$ et A, un algorithme tels que $\forall m \in \Sigma^n \subset \Sigma^* : A$ retourne 1 (resp. 0) en temps $O(n^k)$ si $m \in P$ (resp. si $m \in \Sigma^* \setminus P$).
Définition 5.6. Un algorithme de vérification pour un problème $P \subseteq \Sigma^*$ est un algorithme :

$$A : \Sigma^* \times \Sigma^* \to \{0,1\} \text{ t.q. } P = \left\{ u \in \Sigma^* \text{ t.q. } \exists v \in \Sigma^* \text{ t.q. } A(u,v) = 1 \right\}.$$

Pour $u \in \Sigma^*$, tout $v \in \Sigma^*$ tel que $A(u,v) = 1$ est appelé *certificat pour* u.
Définition 5.7. La classe $\mathcal{NP}$ définit les problèmes vérifiables en temps polynomial et la classe ExpTime définit les programmes qui sont décidables en temps exponentiel.
Théorème 5.8. $\mathcal{P} \subset \mathcal{NP} \subset$ ExpTime.
Définition 5.9. un problème $P \in \mathcal{NP}$ est dit $\mathcal{NP}$*-complet* si tout problème $Q \in \mathcal{NP}$ peut être réduit à P en temps polynomial.

Un problème P est dit $\mathcal{NP}$-dur si tout problème $\mathcal{NP}$-complet Q peut être réduit à P en temps polynomial.
Remarque. Les problèmes $\mathcal{NP}$-complets sont donc $\mathcal{NP}$-durs, mais tout les problèmes $\mathcal{NP}$-durs ne sont pas forcément $\mathcal{NP}$.
Théorème 5.10 (Théorème de Cooke)**.** *Le problème SAT est $\mathcal{NP}$-complet.*
Proposition 5.11. *Soient $P, Q \in \mathcal{NP}$. Si P est $\mathcal{NP}$-complet et P se réduit à Q en temps polynomial, alors Q est $\mathcal{NP}$-complet.*

Démonstration. La composition de deux algorithmes polynomiaux en temps est polynomiale en temps. Un problème $P' \in \mathcal{NP}$ peut être réduit en P par un algorithme $R_{P'P}$ polynomial en temps par hypothèse, et il existe un algorithme R_{PQ} polynomial en temps qui réduit P en Q. La composition $R_{PQ} \circ R_{P'P}$ réduit donc P' en Q en temps polynomial. □

6 Logique du premier ordre