INFOF-302 — Informatique fondamentale

R. Petit

Année académique 2016 - 2017

Table des matières

1 Introduction et Logique **1**

2 Déduction naturelle **2**

3 Coupures **4**

4 DPLL **6**
4.1 Transformation de Tseitin 7
4.2 Contrainte exactement une 7

5 Complexité **8**

6 Logique du premier ordre **9**

7 Indécidabilité **10**
7.1 Problème de Correspondance de Post (PCP) 11

8 Automates finis **12**

Remarques et Notations

Toutes les occurrences du symbole $\subset$ sont à comprendre au sens non-strict de l'inclusion, i.e. $A \subset B$ est à lire $A \subseteq B$. Une inclusion non-stricte est notée $\subsetneq$ ou $\subsetneqq$.

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 $\Leftarrow$ $\square$

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 sur une conjonction, soit sur une disjonction (en ayant transformé tous les autres opérateurs en conjonctions/disjonctions au préalable).

Exemple 1.1.

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

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 ψ.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.	$\neg y \Rightarrow \neg x$	$\Rightarrow_i$ 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 prémisse, 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 :

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

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 :

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

qui se note :

$$\dfrac{\phi}{\neg\neg\phi}\,\neg\neg_{\mathrm{i}}.$$

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

$$\dfrac{}{\phi \vee \neg\phi}\,\mathrm{LEM}.$$

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* $[\![\phi]\!]_\cdot = [\![\psi]\!]_\cdot$ *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 $\phi \equiv \phi_C$. 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 P, 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 $C_i \equiv \bigwedge_{j=1}^{n_i} \ell_{ij}$:

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

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 (appelés *lettres*). On note Σ^* l'ensemble des mots sur Σ, i.e. :

$$\Sigma^* = \bigsqcup_{n\in\mathbb{N}} \Sigma^n.$$

Un mot $u = (a_1, \dots, a_n) \in \Sigma^n \subset \Sigma^*$ se représente habituellement $u = a_1 \dots a_n$. On note $|u| = n$ pour dire que u est de *longueur* n, donc composé de n lettres. Plus précisément, $|u|$ est défini comme l'entier $n \in \mathbb{N}$ tel que $u \in \Sigma^n$.
Définition 5.2. Tout alphabet permet de former un mot commun appelé le *mot vide*, noté ϵ.
Définition 5.3. Soient Σ un alphabet, $a \in \Sigma$ et $n \in \mathbb{N}$. On définit $a^n \coloneqq \underbrace{aa\dots a}_{n}$, le mot composé de n fois la lettre a.
Définition 5.4. Un *langage* sur un alphabet est un sous-ensemble $L \subseteq \Sigma^*$.
Définition 5.5. Le *complément* d'un langage L sur un alphabet Σ est donné par $\overline{L} \coloneqq \Sigma^* \setminus L$.

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 5.6. 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 5.7. 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 $m \in \Sigma^* \setminus P$), pour $n = |m|$.
Définition 5.8. 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 5.9. 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 5.10. $\mathcal{P} \subset \mathcal{NP} \subset$ ExpTime.
Définition 5.11. 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 5.12 (Théorème de Cooke)**.** *Le problème SAT est $\mathcal{NP}$-complet.*
Proposition 5.13. *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

En logique propositionnelle, les valeurs sont définies dans $\{0, 1\}$. En logique des prédicats, on généralise à des ensembles quelconques et on définit des *prédicats* (relations) sur les variables.
Définition 6.1. Un langage du premier ordre est un langage défini sur un vocabulaire (symboles de fonctions, de prédicats, de relations, etc.)

L'alphabet d'un langage du premier ordre contient les éléments communs à tous langages (connecteurs, parenthèses, etc.), et un ensemble $\mathcal{V}$ **infini** de symboles de variables et muni de symboles de symboles de relations ($p, q, r, \ldots$), de fonctions ($f, g, h, \ldots$), et de constantes ($a, b, c, \ldots$).
Définition 6.2. Soit un prédicat ou une fonction P. On appelle *arité* $n \in \mathbb{N}^*$ de P le nombre de paramètres de P. On note son arité $P|_n$.
Définition 6.3. Si $=$ fait partie du vocabulaire du langage du premier ordre, il est dit *égalitaire*.
Définition 6.4. L'ensemble $\mathcal{T}$ des termes d'un langage du premier ordre $\mathcal{L}$ est le plus petit ensemble (au sens de l'inclusion) tel que :

— tous les symboles de variables ou constantes de $\mathcal{L}$ sont dans $\mathcal{T}$;
— si f est un symbole de fonction d'arité n, alors $\forall (t_1, \ldots, t_n) \in \mathcal{T}^n : f(t_1, \ldots, t_n) \in \mathcal{T}$.

Remarque. Notons que les prédicats ne fournissent pas de termes.
Définition 6.5. l'ensemble des formules atomiques $\mathcal{A}$ d'un langage du premier ordre $\mathcal{L}$ est le plus petit ensemble (au sens de l'inclusion) des formules sur $\mathcal{L}$ telles que :

— pour tout prédicat p d'arité n du vocabulaire de $\mathcal{L}$:

$$\forall (t_1, \ldots, t_n) \in \mathcal{T}^n : p(t_1, \ldots, t_n) \in \mathcal{A} ;$$

— si $\mathcal{L}$ est égalitaire, alors $\forall (t_1, t_2) \in \mathcal{T}^2 : (t_1 = t_2) \in \mathcal{A}$.

Définition 6.6. Soit $\mathcal{L}$ un langage du premier ordre. On définit $\mathcal{F}(\mathcal{L})$, l'ensemble des formules ϕ bien formées obtenues sur $\mathcal{L}$, i.e. suivant la grammaire :

$$\phi ::= p(t_1, \ldots, t_n) \mid \exists x \text{ t.q. } \psi \mid \forall x : \psi \mid \phi_p,$$

pour p une relation, $t_1, \ldots, t_n$ des termes, $\psi \in \mathcal{F}(\mathcal{L})$ et ϕ_p, une formule de langage propositionnel.
Remarque. Les quantificateurs $\exists$ et $\forall$ ont la même précédence que $\neg$.
Définition 6.7. Une formule ϕ est une sous-formule de ψ lorsque ϕ apparait dans une décomposition de ψ.
Définition 6.8. Une occurrence d'une variable dans un formule est un couple constitué de la variable et d'une place effective (i.e. ne suivant pas directement un quantificateur).
Définition 6.9. Une occurrence d'une variable x dans une formule ϕ est dite *libre* si elle n'apparait dans aucune sous-formule commençant par un quantificateur. Elle est dite *liée* sinon.
Définition 6.10. Si une variable admet une occurrence libre dans une formule ϕ, elle est dite libre dans ϕ.
Définition 6.11. Une formule ϕ sans variable libre est dite *close*.
Remarque. Une formule ϕ non-close est souvent notée $\phi(x_1, \ldots, x_n)$ pour $x_1, \ldots, x_n$ ses variables libres.
Définition 6.12. Une structure $\mathcal{M}$ sur un langage $\mathcal{L}$ est un tuple contenant :

— M, un ensemble non-vide appelé *domaine* ;
— une interprétation des symboles des prédicats par des relations sur M ;
— une interprétation des symboles des fonctions par des fonctions de M^n dans M ;
— une interprétation des symboles de constantes par des éléments de M.

Pour tout prédicat $r|_n$, on note son interprétation $r^{\mathcal{M}} \subset M^n$. Pour toute fonction $f|_n$, on note $f^{\mathcal{M}} : M^n \to M$ son interprétation. Pour toute constante c, on note $c^{\mathcal{M}} \in M$ son interprétation.
Définition 6.13. Pour $\mathcal{V}$, un ensemble de variables et M, un domaine, on définit une valuation comme étant une fonction $V \in M^{\mathcal{V}}$.

Soit $\mathcal{M}$, une structure, et $\mathcal{V}$ un ensemble de variables. Pour $t \in \mathcal{T}$, on définit sa valuation par :

$$t^{\mathcal{M},V} = \begin{cases} c^{\mathcal{M}} & \text{si t est une constante c} \\ V(x) & \text{si } t = x \in \mathcal{V} \\ f^{\mathcal{M}}(t_1^{\mathcal{M},V}, \dots, t_n^{\mathcal{M},V}) & \text{si } t = f(t_1, \dots, t_n). \end{cases}$$

Définition 6.14. Une formule ϕ sur $\mathcal{L}$ est dite *satisfaite* dans $\mathcal{M}$ par $V \in M^{\mathcal{V}}$ lorsque :
— si $\mathcal{L}$ est égalitaire et $\phi \equiv t_1 = t_2$ et $t_1^{\mathcal{M},V} = t_2^{\mathcal{M},V}$;
— ou si $\phi \equiv r(t_1, \dots, t_n)$ et $(t_i^{\mathcal{M},V})_{i \in [\![1,n]\!]} \in r^{\mathcal{M}}$.
On note cela $\mathcal{M}, V \models \phi$.
Définition 6.15. On appelle *valeur* d'une formule ϕ sur $\mathcal{L}$ dans $\mathcal{M}$ l'ensemble :

$$\mathcal{W} \coloneqq \left\{ V \in M^{\mathcal{V}} \text{ t.q. } \mathcal{M}, V \models \phi \right\}.$$

Si ϕ est close, alors sa valeur de vérité dans $(\mathcal{M}, V)$ ne dépend pas de V. Dans le cas où une telle formule est vraie dans une structure $\mathcal{M}$, on note $\mathcal{M} \models \phi$, et on dit que $\mathcal{M}$ est un *modèle* pour ϕ.
Définition 6.16. Soit ϕ une formule ayant $x_1, \dots, x_n$ pour variables libres. On définit sa *cloture universelle* par :

$$\overline{\phi} \equiv \forall x_1 : \forall x_2 : \dots \forall x_n : \phi(x_1, \dots, , x_n).$$

Définition 6.17. Soient $\mathcal{L}$, un langage du premier ordre, $\mathcal{M}$, une structure sur $\mathcal{L}$, et ϕ, une formule sur $\mathcal{L}$. On dit que $\mathcal{M}$ satisfait ϕ lorsque $\mathcal{M} \models \overline{\phi}$.

Deux formules ϕ et ψ sont *équivalentes* si :

$$\forall \mathcal{M} : \forall V \in M^{\mathcal{V}} : \mathcal{M}, V \models \phi \Leftrightarrow \mathcal{M}, V \models \psi.$$

7 Indécidabilité

Définition 7.1. Un problème de décision $P \subseteq \Sigma^*$ est *indécidable* lorsqu'il n'existe aucun algorithme $A : \Sigma^* \to \{0, 1\}$ tel que pour tout $x \in \Sigma^*$: A retourne 1 si et seulement si $x \in P$ en un nombre fini d'étapes.
Théorème 7.2. *le problème de l'arrêt est indécidable.*

Démonstration. Par l'absurde, supposons qu'il existe un programme `HALT(C, x)` qui détermine par `C`, le code d'un programme si l'entrée `X` fera s'arrêter ledit programme. On définit le programme `PARADOX(C)` par :
— si `HALT(C, C)` retourne 1, alors ne jamais s'arrêter ;
— sinon, retourner 0.
On en déduit que si `C'` est le code du programme `PARADOX`, alors l'appel `PARADOC(C')` s'arrêtera si et seulement si il ne s'arrête pas, ce qui est impossible. Il n'existe donc pas de tel programme `HALT`. □

Proposition 7.3. *Soient P_1, P_2 deux problèmes tels que P_1 est indécidable. S'il existe une réduction de P_1 vers P_2, alors P_2 est indécidable.*

Démonstration. Soit R_{12}, une réduction du problème P_1 en P_2. R_{12} associe à toute entrée I_1 du problème P_1 une entrée I_2 du problème P_2 telle que I_1 mène à une solution pour le problème P_1 si et seulement si I_2 mène à une solution pour le problème P_2.

Supposons par l'absurde qu'il existe un algorithme A qui décide P_2. Dès lors, à l'aide de la réduction, le problème P_1 est décidable, ce qui est une contradiction. Donc il n'existe pas de tel algorithme A, et P_2 est indécidable. □

7.1 Problème de Correspondance de Post (PCP)

Soit $\Sigma = \{0, 1\}$, un alphabet, et soient $u, v \in \Sigma^*$. On définit la *concaténation* uv par :

$$uv = (u_1, \dots, u_{n_1}, v_1, \dots, v_{n_2}) \qquad \text{pour } (u, v) \in \Sigma^{n_1} \times \Sigma^{n_2}, n_1, n_2 \in \mathbb{N}.$$

Le PCP est défini par :
ENTRÉE : $n \in \mathbb{N}$ couples de mots sur Σ, $\{(u_i, v_i)\}_{i \in [\![1,n]\!]}$.
SORTIE : 1 s'il existe $k \in \mathbb{N}^*$ et $(i_1, \dots, i_k) \in [\![1, n]\!]^k$ tels que $u_{i_1} u_{i_2} \dots u_{i_k} = v_{i_1} v_{i_2} \dots v_{i_k}$.

Théorème 7.4. *Soit Σ un alphabet. Si $|\Sigma| < 2$, alors $\mathrm{PCP}(\Sigma)$ est décidable, et si $|\Sigma| > 6$, alors $\mathrm{PCP}(\Sigma)$ est indécidable.*

Démonstration. Admis. □

Remarque. La décidabilité de PCP pour $|\Sigma| \in [\![2, 6]\!]$ est un problème ouvert.

Théorème 7.5. *Le problème de validité en logique du premier ordre est indécidable.*

Démonstration. Montrons cela par réduction du PCP en validité en logique du premier ordre.

Soit $\mathcal{L} = \{p, f_0, f_1, a\}$, un langage du premier ordre avec p, une relation, f_0 et f_1 des fonctions, et a une constante. Soit $u \in \Sigma^*$ un mot, et soit $g \in \mathcal{T}$. On définit :

$$t_u(g) \coloneqq \begin{cases} g & \text{si } u = \epsilon \\ \left(\bigcirc_{i=1}^{|u|} f_{u_{|u|+1-i}} \right)(g) & \text{sinon} \end{cases}.$$

Soit $I = \{(u_i, v_i)\}_{i \in [\![1,n]\!]}$, une instance de $\mathrm{PCP}(\{0, 1\})$. On définit les formules ρ, σ, τ comme suit :

$$\begin{aligned} \rho &\equiv \bigwedge_{i=1}^{n} p(t_{u_i}(a), t_{v_i}(a)), \\ \sigma &\equiv \forall x \forall y : \left(p(x, y) \Rightarrow \bigwedge_{i=1}^{n} p(t_{u_i}(x), t_{v_i}(y)) \right), \\ \tau &\equiv \exists z \text{ t.q. } p(z, z). \end{aligned}$$

On définit alors la formule $\phi_I \coloneqq (\rho \wedge \sigma) \Rightarrow \tau$.

Supposons qu'il existe $\{i_j\}_{j \in [\![1,k]\!]}$ pour l'entrée I pour PCP, et montrons que ϕ_I est valide. Soit $\mathcal{M}$, une structure sur $\mathcal{L}$. On suppose que $\mathcal{M} \models \rho \wedge \sigma$. On sait que :

$$u \coloneqq u_{i_1} u_{i_2} \dots u_{i_k} = v_{i_1} v_{i_2} \dots v_{i_k} =: v,$$

donc, pour $n \coloneqq \sum_{j=1}^{k} i_j$:

$$\left(f_{u_n} \circ f_{u_{n-1}} \circ \dots \circ f_{u_1} \right)(a) \equiv \left(f_{v_n} \circ f_{v_{n-1}} \circ \dots \circ f_{v_1} \right)(a).$$

Notons g cette valeur (qui est dans $\mathcal{T}$). Puisque $\mathcal{M} \models \rho \wedge \sigma$, on a en particulier $\mathcal{M} \models \rho$ et $\mathcal{M} \models \sigma$. Par ρ, on a $\forall j \in [\![1, k]\!] : t_{u_{i_j}}(a) = t_{v_{i_j}}(a)$. Par induction, et par σ, on a :

$$p\left(t_{u_{i_k}}(\dots(t_{u_{i_1}}(a))), t_{v_{i_k}}(\dots(t_{v_{i_1}}(a))) \right) = 1,$$

i.e. $p(g, g)$. On en déduit $\mathcal{M} \models \tau$ sous l'hypothèse $\mathcal{M} \models \rho \wedge \sigma$, donc on déduit $\mathcal{M} \models \phi_I$.

Supposons maintenant ϕ_I valide et montrons que I admet une solution. Soit la structure $\mathcal{H} = (D,\ f^{\mathcal{H}} = f,\ p^{\mathcal{H}})$ telle que $p^{\mathcal{H}}(a, a) = 1$ et $\forall g, h \in \mathcal{T} : \forall (i, j) \in [\![1, k]\!]^2 : p^{\mathcal{H}}(t_{u_i}(g), t_{v_j}(h)) = \delta_i^j p^{\mathcal{H}}(g, h)$.

Par définition de $p^{\mathcal{H}}$, il vient aisément que $\mathcal{H} \models \rho$ et $\mathcal{H} \models \sigma$. Par validité de ϕ_I, il vient que $\mathcal{H} \models \tau$, i.e. $\mathcal{H} \models \exists z$ t.q. $p(z, z)$.

Par définition des $t_\cdot(\cdot)$, il vient que $\exists k$ et que z se décompose en :

$$z = t_{u_{i_k}}(\dots(t_{u_1}(a))) = t_{v_{i_k}}(\dots(t_{v_1}(a))).$$

Dès lors, $u_{i_k} u_{i_{k-1}} \dots u_{i_1} = v_{i_k} v_{ik-1} \dots v_{i_1}$. □

8 Automates finis

Définition 8.1. Un automate fini déterministe (ou DFA pour *deterministic finite automaton*) est défini par un 5-uple $(Q, \Sigma, \delta, q_0, F)$ où :
- Q est un ensemble fini d'*états* ;
- Σ est un alphabet ;
- $\delta : Q \times \Sigma \to Q$ est une *fonction de transition* ;
- $q_0 \in Q$ est l'*état initial* ;
- $F \subseteq Q$ est l'ensemble des *états finaux*.

Remarque. La fonction δ n'est pas nécessairement complète, i.e. il peut exister un couple $(q, a) \in Q \times \Sigma$ tel que $\delta(q, a)$ n'est pas défini. Dans ce cas, on complète la fonction δ en ajoutant q_{-1} à $Q \setminus F$, un état dit *mort* tel que :
- $\forall a \in \Sigma : \delta(q_{-1}, a) = q_{-1}$;
- $\forall (q, a) \in Q \times \Sigma$ t.q. $\delta(q, a)$ n'est pas défini, on pose $\delta(q, a) \coloneqq q_{-1}$.

L'état q_{-1} est donc un état non-final dont il est impossible de *sortir*.

Définition 8.2. Un automate $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ est dit *complet* si sa fonction de transition δ est complète.

On étend la définition de la fonction δ en $\hat{\delta} : Q \times \Sigma^* \to Q$ tel que :
- $\forall q \in Q : \hat{\delta}(q, \epsilon) = q$;
- $\forall (q, u, a) \in Q \times \Sigma^* \times \Sigma : \hat{\delta}(q, ua) = \hat{\delta}(\delta(q, u), a)$.

Définition 8.3. Un mot $u \in \Sigma^*$ est dit *admissible* (ou encore *accepté* ou *reconnu*) par un automate $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ lorsque :

$$\hat{\delta}(q_0, u) \in F,$$

i.e. lorsque l'exécution de $\mathcal{A}$ sur le mot u finit sur un état final.

Définition 8.4. Soit $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ un DFA. On définit le *langage accepté* (ou *langage reconnu*) par $\mathcal{A}$ par :

$$L(\mathcal{A}) \coloneqq \left\{ u \in \Sigma^* \text{ t.q. } \hat{\delta}(q_0, u) \in F \right\}.$$

Définition 8.5. Soient Σ un alphabet, et $L \subseteq \Sigma^*$, un langage sur Σ. L est dit *reconnaissable par un automate* lorsqu'il existe $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ tel que $L(\mathcal{A}) = L$.

Remarque. Les DFAs forment la plus petite classe de *machines* avec lesquelles il est possible d'étudier la théorie de la calculabilité (décidabilité, déterminisme, etc.)

De par leur simplicité, certains problèmes indécidables pour les machines de Turing deviennent décidables pour les DFAs, e.g. minimalité du nombre d'états d'un DFA pour un langage reconnu donné, égalité des langages reconnus entre deux automates, etc.

Lemme 8.6. *Soit un automate $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ non-complet. La prolongation $\bar{\delta}$ de δ ci-dessus donne un automate $\mathcal{B} = (Q \cup \{q_{-1}\}, \Sigma, \bar{\delta}, q_0, F)$ tel que $\mathcal{B}$ est complet, et $L(\mathcal{A}) = L(\mathcal{B})$.*

Définition 8.7. Soit $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ un automate. Un état $q \in Q$ est dit *atteignable* lorsqu'il existe $u \in \Sigma^*$ tel que $\hat{\delta}(q_0, u) = q$.

Définition 8.8. On appelle *transition* dans un automate $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ un couple $(a, q) \in \Sigma \times Q$ tel que $\delta(q, a)$ est défini.
Théorème 8.9. *Soit $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$. On peut tester si $L(\mathcal{A}) = \emptyset$ en $O(n + m)$, avec $n = |Q|$ et m le nombre de transitions de $\mathcal{A}$.*
Définition 8.10. On appelle *pré-automate* un 4-uple (Q, Σ, δ, q_0), où Q est un ensemble fini d'états, Σ un alphabet, δ une fonction de transition, et $q_0 \in Q$, un état initial.
Remarque. Un automate est donc un pré-automate avec des états finaux.
Définition 8.11. Soient $\mathcal{A} = (Q_A, \Sigma, \delta_A, q_{0,A}, F_A)$ et $\mathcal{B} = (Q_B, \Sigma, \delta_B, q_{0,B}, F_B)$ deux automates sur un même alphabet. On définit leur produit par le pré-automate :

$$\mathcal{A} \otimes \mathcal{B} = (Q_A \times Q_B, \Sigma, \delta_{AB}, (q_{0,A}, q_{0,B})),$$

avec $\delta_{AB} : (Q_A \times Q_B) \times \Sigma \to Q_A \times Q_B$ telle que $\forall (q_A, q_B, a) \in Q_A \times Q_B \times \Sigma$ tels que $\delta_A(q_A, a)$ et $\delta_B(q_B, a)$ sont définis, δ_{AB} est définie par $\delta_{AB}((q_A, q_B), a) = (\delta_A(q_A, a), \delta_B(q_B, a))$.

Une signification que l'on peut donner à un tel automate est qu'il exécute simultanément l'automate $\mathcal{A}$ et l'automate $\mathcal{B}$ sur leur automate produit. Remarquons que la notion d'exécution ne fait pas intervenir la notion d'état final, et qu'elle est donc définie sur les pré-automates également.
Théorème 8.12. *Soient $\mathcal{A}_1, \mathcal{A}_2$ deux DFAs. Il existe $\overline{\mathcal{A}_1}, \mathcal{A}_U, \mathcal{A}_I$, trois DFAs tels que :*

$$\begin{aligned} L(\overline{\mathcal{A}_1}) &= \overline{L(\mathcal{A}_1)}, \\ L(\mathcal{A}_U) &= L(\mathcal{A}_1) \cup L(\mathcal{A}_2), \\ L(\mathcal{A}_I) &= L(\mathcal{A}_1) \cap L(\mathcal{A}_2). \end{aligned}$$

Démonstration. Pour le complément, on prend $\mathcal{A}_1 = (Q, \Sigma, \delta, q_0, F)$, que l'on suppose complet sans perte de généralité (Lemme 8.6), et on définit $\overline{\mathcal{A}_1} \coloneqq (Q, \Sigma, \delta, q_0, Q \setminus F)$. On en déduit qu'un mot $u \in \Sigma^*$ est admissible par $\mathcal{A}_1$ si et seulement il n'est pas admissible par $\overline{\mathcal{A}_1}$.

À nouveau, on suppose $\mathcal{A}_1$ et $\mathcal{A}_2$ complets. On pose ensuite le pré-automate $\mathcal{A}_P = \mathcal{A}_1 \otimes \mathcal{A}_2$. Pour l'automate d'union, on définit que l'automate $\mathcal{A}_U$ correspond à $\mathcal{A}_P$ muni de l'ensemble d'états finaux $F_U = (F_1 \times Q_2) \cup (Q_1 \times F_2) \subseteq Q_A \times Q_B$, et pour l'automate d'intersection, on définit que l'automate $\mathcal{A}_I$ correspond à $\mathcal{A}_P$ muni de l'ensemble d'états finaux $F_I = F_1 \times F_2 \subseteq Q_A \times Q_B$. □

Définition 8.13. Deux automates $\mathcal{A} = (Q_A, \Sigma, \delta_A, q_0^A, F_A)$ et $\mathcal{B} = (Q_B, \Sigma, \delta_B, q_0^B, F_B)$ sont dits *isomorphes* s'il existe $\alpha : Q_A \to Q_B$ bijective telle que :

$$\forall (q_A, a) \in Q_A \times \Sigma : \alpha(\delta_A(q_A, a)) = \delta_B(\alpha(q_A), a),$$

i.e. lorsqu'il existe une bijection sur l'ensemble des états qui conserve les transitions.
Définition 8.14. Deux automates $\mathcal{A}$ et $\mathcal{B}$ sont dits *équivalents* lorsque $L(\mathcal{A}) = L(\mathcal{B})$.
Théorème 8.15. *Pour deux automates $\mathcal{A}$ et $\mathcal{B}$, il est décidable en temps polynomial si $L(\mathcal{A}) \subseteq L(\mathcal{B})$.*

Démonstration. Remarquons que pour deux ensembles E et F, on a $E \subseteq F \iff E \cap \overline{F} = \emptyset$. Dans le cas des automates, on veut donc savoir si $L(\mathcal{A}) \cap L(\overline{\mathcal{B}}) = \emptyset$. Or on sait par le Théorème 8.12 qu'il est possible, en temps polynomial, de déterminer un automate étant le complémentaire d'un autre, et de déterminer un automate dont le langage est l'intersection de deux autres.

Par le Théorème 8.9, on peut déterminer en temps linéaire si un automate est de langage vide.

On peut donc décider en temps polynomial si deux automates ont le même langage reconnu. □

Corollaire 8.16. *Il est donc décidable en temps polynomial si deux automates sont équivalents.*

Démonstration. Deux automates $\mathcal{A}$ et $\mathcal{B}$ sont équivalents si $L(\mathcal{A}) = L(\mathcal{B})$. Ils le sont donc si et seulement si $L(\mathcal{A}) \subseteq L(\mathcal{B})$ et $L(\mathcal{B}) \subseteq L(\mathcal{A})$. □

Définition 8.17. Un automate $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ est dit *minimal* s'il n'existe pas d'automate $\mathcal{B} = (Q', \Sigma, \delta', q_0', F')$ tel que $L(\mathcal{A}) = L(\mathcal{B})$ et $|Q'| \lneq |Q|$.

Définition 8.18. Soient Σ un alphabet, $L \subseteq \Sigma^*$ un langage, et $u, v \in \Sigma^*$. u et v sont dits *équivalents pour* L lorsque :

$$\forall w \in \Sigma^* : uw \in L \iff vw \in L.$$

On note cela $u \equiv_L v$.

Proposition 8.19. *Soient Σ un alphabet, $L \subseteq \Sigma^*$ un langage, et $u, v \in \Sigma^*$ deux mots équivalents sur L. Alors pour tout automate minimal $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ tel que $L(\mathcal{A}) = L$: $\hat{\delta}(q_0, u) = \hat{\delta}(q_0, v)$.*

Proposition 8.20. *Pour $L \subseteq \Sigma^*$, un langage sur un alphabet Σ, la relation $\equiv_L$ est une relation d'équivalence.*

Théorème 8.21 (Myhill-Nerode)**.** *Soient Σ un alphabet et $L \subseteq \Sigma^*$ un langage sur Σ. L est reconnaissable par un automate si et seuelement si $\left|\Sigma^*/\equiv_L\right| \in \mathbb{N}$.*

Lemme 8.22. *Soient Σ un alphabet, $L \subseteq \Sigma^*$ un langage sur Σ. Alors :*

$$\forall (u, v, a) \in \Sigma^* \times \Sigma^* \times \Sigma : u \equiv_L v \Rightarrow ua \equiv_L va.$$

Démonstration. Par associativité de la concaténation, on a :

$$\forall (u, a, w) \in \Sigma^* \times \Sigma \times \Sigma^* : uaw = (ua)w = u(aw).$$

Dès lors, soient u, v équivalents pour L, et soient $(a, w) \in \Sigma \times \Sigma^*$. On a, pour $x \coloneqq aw$:

$$uaw = ux = \in L \iff L \ni vx = vaw.$$

Donc $ua \equiv_L va$. □

Définition 8.23. Soient Σ un alphabet, et $L \subseteq \Sigma^*$ un langage reconnaissable par un automate. On définit l'automate quotient $\mathcal{A}_L = (Q_L, \Sigma, \delta_L, q_0, F_L)$, où :

- $Q_L = \Sigma^*/\equiv_L$;
- $\delta : Q_L \times \Sigma$ est définie par $\forall ([u], a) \in Q_L \times \Sigma : \delta([u], a) = [ua]$;
- $q_0 = [\epsilon] \in Q_L$;
- $F_L = \left\{[u] \text{ t.q. } u \in L\right\}$.

Remarque. l'automate est bien défini car Q_L est bien fini par le théorème de Myhill-Nerode, et δ_F est bien définie par le Lemme 8.22.

Théorème 8.24. *Soient Σ un alphabet et $L \subseteq \Sigma^*$ un langage sur Σ. L'automate quotient $\mathcal{A}_L$ est minimal et est unique, à isomorphisme près.*

Définition 8.25. Soit $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$, un DFA complet. Pour $q \in Q$, on note :

$$L_q(\mathcal{A}) \coloneqq \{u \in \Sigma^* \text{ t.q. } \delta(q, u) \in F\}.$$

pour toute paire $(p, q) \in Q^2$, on note $p \equiv_{\mathcal{A}} q$ lorsque $L_p(\mathcal{A}) = L_q(\mathcal{A})$.

Remarque. La relation $\equiv_{\mathcal{A}}$ étant une relation d'égalité est une relation d'équivalence.

Proposition 8.26. *Soient $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$, un DFA, et $p, q \in Q$. Alors :*

$$\forall a \in \Sigma : p \equiv_{\mathcal{A}} q \Rightarrow \delta(p, a) \equiv_{\mathcal{A}} \delta(q, a).$$

Démonstration. Si $p \equiv_{\mathcal{A}} q$, alors $L_p(\mathcal{A}) = L_q(\mathcal{A})$. On pose $p' \coloneqq \delta(p, a)$ et $q' \coloneqq \delta(q, a)$. Soit $w \in L_{p'}(\mathcal{A})$. On en déduit que $aw \in L_p(\mathcal{A})$, et donc $aw \in L_q(\mathcal{A})$ et $w \in L_{q'}(\mathcal{A})$. La réciproque se fait de la même manière, et on a donc bien $p' \equiv_{\mathcal{A}} q'$. □

Proposition 8.27. *Soit $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ un automate complet. Alors :*

$$\forall u, v \in \Sigma^* : u \equiv_{L(\mathcal{A})} v \iff \delta(q_0, u) \equiv_{\mathcal{A}} \delta(q_0, v).$$

Démonstration. Supposons que $u \equiv_{L(\mathcal{A})} v$. Soient $p \coloneqq \delta(q_0, u)$ et $q \coloneqq \delta(q_0, v)$, et montrons que $L_p(\mathcal{A}) = L_q(\mathcal{A})$.

Soit $w \in L_p(\mathcal{A})$. On a alors $F \ni \delta(p, w) = \delta(\delta(q_0, u), w) = \delta(q_0, uw)$, et donc $uw \in L(\mathcal{A})$. Par hypothèse, on déduit que $vw \in L(\mathcal{A})$. Donc $F \ni \delta(q_0, vw) = \delta(q, w)$, d'où $w \in L_q(\mathcal{A})$. On en déduit que $L_p(\mathcal{A}) \subseteq L_q(\mathcal{A})$. Le raisonnement pour montrer que $L_q(\mathcal{A}) \subseteq L_p(\mathcal{A})$ est le même, et on en conclut que $L_p(\mathcal{A}) = L_q(\mathcal{A})$.

Supposons ensuite que $p = \delta(q_0, u) \equiv_{\mathcal{A}} \delta(q_0, v) = q$. Par définition, on déduit que $L_p(\mathcal{A}) = L_q(\mathcal{A})$. Montrons alors que $u \equiv_{L(\mathcal{A})} v$. Soit $w \in \Sigma^*$ tel que $uw \in L(\mathcal{A})$. On déduit de la même manière que précédemment que $w \in L_p(\mathcal{A}) = L_q(\mathcal{A})$, donc $w \in L_q(\mathcal{A})$, et $vw \in L(\mathcal{A})$. Finalement, on en déduit que :

$$\forall w \in \Sigma^* : uw \in L(\mathcal{A}) \Rightarrow vw \in L(\mathcal{A}).$$

la réciproque se démontre de la même manière, et l'on déduit que $u \equiv_{L(\mathcal{A})} v$. □

Corollaire 8.28. *Il existe une bijection* $\phi : \Sigma^*/\equiv_{L(\mathcal{A})} \to Q/\equiv_{\mathcal{A}} : [u] \mapsto [\delta(q_0, u)]$.

L'automate minimal équivalent à $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ est alors donné par :

$$\mathcal{B} = (Q/\equiv_{\mathcal{A}}, \Sigma, \delta', [q_0], \{[q] \text{ t.q. } q \in F\}),$$

pour $\delta' : Q/\equiv_{\mathcal{A}} \times \Sigma \to Q/\equiv_{\mathcal{A}} : ([q], a) \mapsto [\delta(q, a)]$.

On remarque que la fonction δ' est bien définie car elle ne dépend pas du représentant de la classe d'équivalence.

Définition 8.29. Soit Σ un alphabet. Une *expression rationnelle* sur Σ est une expression qui suit la grammaire :

$$E ::= \epsilon \mid a \mid \emptyset \mid (E + E) \mid (E.E) \mid E*,$$

pour $a \in \Sigma$.

Définition 8.30. Soient Σ un alphabet et L_1, L_2 deux langages sur Σ. On définit :

$$L_1 \cdot L_2 \coloneqq \left\{u_1 u_2 \text{ t.q. } (u_1, u_2) \in L_1 \times L_2\right\}$$
$$L_1^* \coloneqq \left\{u_1 \dots u_k \text{ t.q. } k \in \mathbb{N} \text{ et } \forall i \in [\![1, k]\!] : u_i \in L\right\}.$$

Définition 8.31. La sémantique d'une expression rationnelle sur un alphabet Σ est donnée par un langage $L(E)$ défini inductivement par :

- $L(\epsilon) = \{\epsilon\}$;
- $\forall a \in \Sigma : L(a) = \{a\}$;
- $L(\emptyset) = \emptyset$;
- $L(E_1 + E_2) = L(E_1) \cup L(E_2)$;
- $L(E_1 \cdot E_2) = L(E_1) \cdot L(E_2)$;
- $L(E^*) = L(E)^*$.

Théorème 8.32. *Tout langage* L *sur un alphabet* Σ *est reconnaissable par un automate si et seulement si il est définissable par une expression rationnelle.*