Complexité
(L3 - premier semestre)

Serge Haddad
Professeur de l'ENS Cachan
61, Avenue du Président Wilson
94235 Cachan cedex, France
adresse électronique : haddad@lsv.ens-cachan.fr
page personnelle : www.lsv.ens-cachan.fr/∼haddad/

15 novembre 2020

Table des matières

Chapitre 1

Introduction aux classes de complexité

1.1 Préliminaires

L'objectif de ce cours consiste à caractériser la difficulté d'un problème. Ceci soulève un certain nombre de questions que nous allons traiter essentiellement dans ce chapitre et que nous explicitons ci-dessous.

Problèmes et instances

Un *problème* est une fonction P de A dans B. Un élément de A est appelée une *instance* du problème. Lorsque B est le domaine des booléens, on parle de problème de décision.

Nous illustrons ces notions à l'aide d'un problème classique. Soit A l'ensemble des graphes dont deux sommets sont distingués (un sommet initial u et un sommet final v). Le problème de l'accessibilité consiste à savoir s'il existe un chemin de u à v. En réalité selon le choix de B, il s'agit de deux problèmes différents.

— Soit B est le domaine des booléens. On décide alors l'existence du chemin.
— Soit B est le domaine des entiers enrichi de la valeur ∞. On fournit la longueur d'un plus court chemin (∞ s'il n'en existe pas).

Nous nous consacrons ici aux problèmes de décision.

Représentation des données

Puisqu'il s'agit de problèmes destinés à être résolus par des ordinateurs, le choix de la représentation des entrées et des résultats peut être significatif.

Reprenons l'exemple d'un graphe de n sommets et de m arcs. On peut choisir de le représenter de deux façons différentes :

— On indique d'abord le nombre n de sommets puis la suite des m couples représentant les arcs. Ceci conduit à une taille d'entrée approximativement égale à $(2m+1)\log(n+1)$ bits.

— On indique d'abord le nombre n de sommets puis la matrice d'adjacence $n \times n$ de bits. Ceci conduit à une taille d'entrée approximativement égale à $n^2 + \log(n+1)$ bits.

Même en éliminant les cas pathologiques de sommets isolés qui peuvent être représentés par leur cardinalité dans les deux cas, la taille de la deuxième représentation peut ne pas être linéaire par rapport à la taille de la première. Ceci implique que la notion de complexité que nous retiendrons devra être *robuste* vis à vis des alternatives de représentations.

Il est cependant un cas qui doit être traité de manière explicite : la représentation des entiers. La représentation usuelle d'un entier (i.e. la représentation binaire) n conduit à une taille $\log(n)$. Cette représentation peut induire des complexités de résolution élevées.

Lorsqu'on veut mettre en évidence que la complexité dépend des entiers du problème, on choisit de manière artificielle une représentation unaire et on calcule la complexité dans ce cas particulier.

Notons que dans le cas d'un problème de décision si on oublie ce que représente la donnée, on a affaire à un mot et l'ensemble des mots qui conduisent à un calcul réussi constituent un *langage*. Bien que les deux points de vue soient équivalents, selon le contexte on parlera de problème ou de langage.

Mesures de complexité

Il y a *a priori* deux mesures de complexité distinctes, le temps et l'espace [1]. Nous verrons dans la suite qu'il y a cependant des relations entre ces deux mesures.

Pour l'instant, intéressons-nous aux unités de mesure. Pour ce qui est de l'espace, on pourrait compter en bits ou en mots mémoires. Ces choix sont sans importance si la mesure de complexité est insensible à un changement d'échelle. Le point essentiel est qu'il ne faut pas prendre en compte la taille de l'entrée car il est nécessaire de discriminer entre des problèmes qui occupent un espace de travail inférieur à la taille de l'entrée (comme par exemple la reconnaissance d'un mot par un automate fini).

Pour ce qui est du temps du calcul, le choix du modèle de calcul peut avoir des conséquences importantes. Ce point sera détaillé lors de la section suivante. Illustrons-le ici à l'aide d'une recherche d'un élément dans un tableau trié de n éléments. Si le modèle de calcul permet l'accès direct alors ce problème se résout en $O(\log(n))$. Si le modèle de calcul ne permet qu'un accès séquentiel alors ce problème a une complexité $\Theta(n)$.

Bornes de complexité

Il est parfois difficile de caractériser la complexité exacte d'un problème. L'existence d'un algorithme fournit une borne supérieure de complexité. L'établissement d'une borne inférieure de complexité repose sur la notion de *réduction* qui intuitivement consiste à transformer de manière efficace une instance d'un problème P en une instance d'un problème P' telles qu'à partir du résultat de

1. Ici l'*espace* désigne la mémoire utilisée lors des calculs.

l'instance du problème P', on calcule efficacement le résultat du problème P. Dans le cas de problèmes de décision, on demande à ce que le résultat des deux instances soient identiques.

1.2 Classes de complexité

Modèles de calcul

Contrairement à la calculabilité (voir la thèse de Church) le choix du modèle de calcul a une influence sur la complexité. Aussi afin d'éviter les ambiguïtés, il est naturel de se tourner vers les *machines de Turing*. Nous conviendrons de prendre des machines à plusieurs bandes unidirectionnelles avec deux bandes distinguées, *une bande d'entrée* et *une bande de sortie*. La bande d'entrée contient la description de l'instance du problème et la machine ne peut pas écrire sur cette bande. La bande de sortie contient le résultat de la machine et la fonction de transition de la machine est indépendante du contenu de cette bande. Le calcul s'arrête lorsque la machine est dans un état final.

Dans le cas de problèmes de décision, la machine ne dispose plus de bande de sortie mais de deux états finals : un état d'acceptation et un état de rejet. Le résultat du calcul correspond à l'état atteint.

La machine qui effectue le calcul sera soit *déterministe* soit *non déterministe* (généralement dans le cas de problème de décision). Si on a affaire à une machine non déterministe, le résultat sera **true** ssi l'un des calculs possibles de la machine est **true**. D'autres modèles tels que les machines alternantes seront étudiés en TD.

Le temps de calcul d'une machine déterministe est le nombre de transitions du calcul de la machine et son espace est la valeur maximale au cours du calcul d'une tête de lecture (hormis celles de la bande d'entrée et de la bande de sortie).

Le temps de calcul d'une machine non déterministe est la valeur maximale du nombre de transitions d'un des calculs de la machine (y compris les calculs infructueux) et son espace est la valeur maximale d'une tête de lecture pour l'un quelconque des calculs.

Fonctions de mesure

Afin de définir des classes de complexité significatives, il est nécessaire de se restreindre à des fonctions de mesure de complexité réalistes (et réalisables).

Définition 1 *Soit $f : \mathbb{N} \mapsto \mathbb{N}$ une fonction croissante, f est dite constructible s'il existe une machine $\mathcal{M}$ qui produit pour toute entrée de longueur n, une sortie constituée de la représentation unaire de $f(n)$ (ou d'un marqueur sur la cellule $f(n)$) en un temps $O(n + f(n))$ et en utilisant un espace $O(f(n))$.*

La plupart des fonctions usuelles sont constructibles : $\lfloor \log_2(n) \rfloor$ (pour $n > 0$), $n, n^2, 2^n$, etc.

On introduit d'abord les classes de complexités relatives à des fonctions de mesure. Nous nous intéressons à une complexité *au pire cas* et *asymptotique*. D'où la définition ci-dessous.

Définition 2 *Un langage L appartient à :*

- $\mathsf{TIME}(f)$ *(resp.* $\mathsf{NTIME}(f)$*) s'il existe un entier* n_0 *et une machine de Turing déterministe (resp. non déterministe) opérant en temps au plus* $f(n)$ *sur tout mot de longueur* $n \geq n_0$ *dont le langage est* L*.*
- $\mathsf{SPACE}(f)$ *(resp.* $\mathsf{NSPACE}(f)$*) s'il existe un entier* n_0 *et une machine de Turing déterministe (resp. non déterministe) opérant en espace au plus* $f(n)$ *sur tout mot de longueur* $n \geq n_0$ *dont le langage est* L*.*

Les théorèmes qui suivent établissent des relations d'inclusion générales entre classes de complexité.

Théorème 1 *On a pour toute fonction* f *les inclusions suivantes :*

- $\mathsf{TIME}(f) \subseteq \mathsf{NTIME}(f)$,
- $\mathsf{SPACE}(f) \subseteq \mathsf{NSPACE}(f)$,
- $\mathsf{TIME}(f) \subseteq \mathsf{SPACE}(f)$,
- $\mathsf{NTIME}(f) \subseteq \mathsf{NSPACE}(f)$.

Preuve
Une machine déterministe est un cas particulier de machine non déterministe. Une machine qui effectue au plus t transitions occupe au plus t cellules d'une bande (dans le cas de déplacements à droite ininterrompus).

c.q.f.d. $\Diamond\Diamond\Diamond$

Théorème 2 *On a pour toute fonction* f *et tout entier* $k > 0$ *les inclusions suivantes :*

- $\mathsf{SPACE}(kf) \subseteq \mathsf{SPACE}(f)$
- $\mathsf{NSPACE}(kf) \subseteq \mathsf{NSPACE}(f)$

Preuve
Étant donnée une machine $\mathcal{M}$ travaillant sur des caractères de Σ, on construit une machine $\mathcal{M}'$ qui travaille comme $\mathcal{M}$ mais sur des blocs de k caractères autrement dit sur l'alphabet $\Sigma^k \cup \{\Box\}$.

c.q.f.d. $\Diamond\Diamond\Diamond$

Le théorème précédent peut être qualifié de théorème de *compression linéaire*. Un théorème similaire relatif au temps d'exécution et donc *d'accélération linéaire* sera démontré en TD. Les deux théorèmes suivants utilisent des techniques de simulation entre machines de nature différente.

Théorème 3 *On a pour toute fonction constructible* f *telle que* $f(n) \geq n$ *l'inclusion suivante :*

$$\mathsf{NTIME}(f) \subseteq \mathsf{SPACE}(f)$$

Preuve
Soit $\mathcal{M}$ une machine de Turing non déterministe opérant en temps $f(n)$. Sur une entrée x de longueur n, $\mathcal{M}$ effectue au plus $f(n)$ choix. On définit une machine $\mathcal{M}'$ qui fonctionne ainsi :

- Elle exécute une boucle externe dont l'indice de boucle correspond aux différents $f(n)$-uplets de choix.

— Au cours d'un tour de boucle, elle simule $\mathcal{M}$ en résolvant les choix en fonction de l'indice courant. Si au cours d'un tour de boucle le calcul simulé est réussi, elle renvoie **true**.
— Si elle termine sa boucle sans sortie prématurée alors elle renvoie **false**.

La gestion de la boucle externe utilise un espace $O(f(n))$ et la simulation se fait aussi en espace $O(f(n))$. En utilisant le théorème précédent on produit une machine de Turing qui occupe un espace $f(n)$.

c.q.f.d. ◊◊◊

Théorème 4 *On a pour toute fonction f l'inclusion suivante :*

$$\mathsf{NSPACE}(f) \subseteq \bigcup_{a\in\mathbb{N}} \mathsf{TIME}\big(2^{a(f+\log)}\big) = \mathsf{TIME}\big(2^{O(f+\log)}\big)$$

Preuve
Soit $\mathcal{M}$ une machine de Turing à k bandes de travail opérant en espace $f(n)$. Sur une entrée x de longueur n, le nombre de configurations de $\mathcal{M}$ est borné par $|Q| \times |\Sigma|^{k\cdot f(n)} \times \max\big(f(n), n\big)^k$ (états possibles de l'automate, contenu des bandes et position des têtes de lecture). Le facteur additionnel n correspond au cas de la tête de lecture de la bande d'entrée qui occupe n cellules de la bande.

On associe à $\mathcal{M}$ une machine de Turing déterministe $\mathcal{M}'$ qui construit le graphe des configurations et recherche s'il y a un chemin dans ce graphe de la configuration initiale à la configuration finale (on peut toujours supposer qu'il existe une unique configuration finale sans modifier l'espace occupé). Le problème d'accessibilité est polynomial en fonction du nombre de sommets du graphe.

c.q.f.d. ◊◊◊

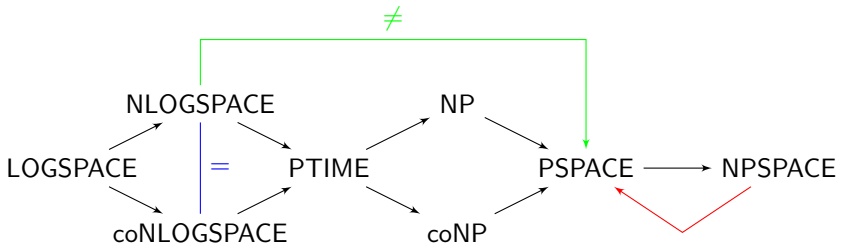
Le résultat précédent montre l'intérêt de considérer des classes plus larges que celles associées à des fonctions car ces dernières ne sont pas suffisamment robustes. De plus en pratique, seules certaines classes de complexité correspondent à des cas couramment rencontrés. Ainsi on définit :
— $\mathsf{LOGSPACE} \equiv \mathsf{SPACE}(\log_2(n))$,
$\mathsf{NLOGSPACE} \equiv \mathsf{NSPACE}(\log_2(n))$,
— $\mathsf{PTIME} \equiv \bigcup_{k\in\mathbb{N}} \mathsf{TIME}(n^k)$ (appelée aussi P)
$\mathsf{NPTIME} \equiv \bigcup_{k\in\mathbb{N}} \mathsf{NTIME}(n^k)$ (appelée aussi NP),
— $\mathsf{PSPACE} \equiv \bigcup_{k\in\mathbb{N}} \mathsf{SPACE}(n^k)$,
$\mathsf{NPSPACE} \equiv \bigcup_{k\in\mathbb{N}} \mathsf{NSPACE}(n^k)$,
— $\mathsf{EXPTIME} \equiv \bigcup_{k\in\mathbb{N}} \mathsf{TIME}(2^{n^k})$ (appelée aussi EXP),
$\mathsf{NEXPTIME} \equiv \bigcup_{k\in\mathbb{N}} \mathsf{NTIME}(2^{n^k})$ (appelée aussi NEXP), etc.

Puisque les classes de complexité sont vues comme des familles de langages, il est intéressant de se demander si ces familles sont closes par complémentation. C'est trivialement le cas pour les classes déterministes car il suffit d'inverser le résultat des calculs d'une machine de Turing. Par contre, cela n'a rien d'évident pour les classes non déterministes car en passant au complémentaire on obtient des machines non déterministes universelles qui répondent **true** si tous les calculs renvoient **true**. Aussi on préfixe ces classes de complexité par co comme coNP.

On complète les théorèmes précédents, par quelques inclusions importantes qui seront démontrées plus loin :

— NPSPACE $\subseteq$ PSPACE, une conséquence du théorème de Savitch ;
— NLOGSPACE = coNLOGSPACE, le théorème d'Immerman-Szelepscényi ;
— NLOGSPACE $\subsetneq$ PSPACE, une conséquence des hiérarchies strictes de complexité.

On obtient alors la hiérarchie suivante :



1.3 Réductions

Afin d'établir des bornes inférieures de complexité, nous aurons recours à la notion de *réduction*.

Définition 3 *Une réduction d'un problème de décision $P : A \mapsto \{\mathbf{false}, \mathbf{true}\}$ à un problème $P' : A' \mapsto \{\mathbf{false}, \mathbf{true}\}$ est une fonction calculable $r : A \mapsto A'$ telle que $\forall a \in A\ P'(r(a)) = P(a)$.*

Cette réduction est une réduction LOGSPACE *(resp.* PTIME*) si la procédure associée à h opère en espace logarithmique (resp. en temps polynomial).*

Afin que cette notion soit utile il est important qu'elle définisse un préordre entre problèmes à savoir qu'elle soit réflexive (évident avec h l'identité) et transitive (ce qui est l'objet de la proposition suivante).

Proposition 1 *Supposons que P se réduise en temps polynomial (resp. en espace logarithmique) à P' et que P' se réduise en temps polynomial (resp. en espace logarithmique) à P'' alors P se réduit en temps polynomial (resp. en espace logarithmique) à P''.*

Preuve
Nous établissons la preuve pour les réductions en temps polynomial. La preuve pour les réductions en espace logarithmique sera traitée en TD.

La procédure consiste simplement à appliquer la première transformation puis à appliquer la deuxième transformation sur le résultat de la première transformation. Si $p(n)$ (resp. $p'(n)$) est le polynôme associée à la première (resp. deuxième) alors $p'(p(n)) + p(n)$ est le polynôme associé à la transformation globale.

c.q.f.d. $\Diamond\Diamond\Diamond$

Un problème *pb* est *complet* pour une classe Cl (noté Cl-complet) lorsqu'il appartient à Cl et que tout problème *pb'* de Cl se réduit à *pb'*. Lorsque seule la deuxième condition est satisfaite, on parle de problème Cl-*difficile*. Cette définition comporte une ambiguïté : quelle type de réduction autorisons-nous ? La

réponse que nous adoptons est liée à la classe elle-même. Lorsque la classe est plus grande que PTIME, nous autorisons les réductions en temps polynomial et dans le cas contraire nous autorisons uniquement les réductions en espace logarithmique. Cette distinction est essentiellement théorique car dans les réductions que nous examinerons en cours et en TD, les réductions s'effectueront en espace logarithmique.

Comment déterminer qu'un problème est complet ?

La réponse la plus naturelle à cette question est de réduire un autre problème complet au problème étudié et c'est ainsi que nous procéderons la plupart du temps. Cependant cette approche nécessite de disposer d'au moins un problème complet. Afin d'obtenir ce premier problème, il donc indispensable d'adopter un point de vue plus générique.

Nous illustrons cette démarche avec la classe PTIME et un problème P dont on veut démontrer qu'il est PTIME-complet. Un problème est dans PTIME s'il existe une machine de Turing (disons $\mathcal{M}$) qui opère en temps polynomial (disons un polynôme p) en fonction de son entrée w de taille n. La réduction consiste donc à concevoir un algorithme

- qui dépend de $\mathcal{M}$,
- mais a pour unique entrée w,
- et produit en espace logarithmique une instance a_w du problème P telle que $P(a_w)$ soit le résultat de la machine $\mathcal{M}$ sur l'entrée w.

Le point clef à retenir est que l'algorithme de réduction s'appuie sur $\mathcal{M}$ mais aussi sur la connaissance du polynôme p (pour simuler par exemple $p(n)$ pas d'exécution de la machine et dans d'autres contextes une bande de longueur $p(n)$).

1.4 TD 1

Accélération linéaire

Question 1 *Montrer que pour tout $\epsilon > 0$,*

$$\mathsf{TIME}(f(n)) \subseteq \mathsf{TIME}(\epsilon f(n) + n + 2)\,.$$

Évaluer le nombre d'états et de transitions de la nouvelle machine de Turing. Rappeler l'intérêt de cette propriété.

Réductions LOGSPACE

Question 2 *Montrer que toute machine de Turing déterministe qui calcule une réduction en espace logarithmique, s'arrête après un nombre d'étapes polynomial.*

Question 3 *Soient $h_1 : L_1 \mapsto L_2$ et $h_2 : L_2 \mapsto L_3$ deux réductions calculables en espace logarithmique par des machines déterministes.*
Montrer que la réduction $h_2 \circ h_1 : L_1 \mapsto L_3$ peut aussi être calculée en espace logarithmique par une machine déterministe.
Rappeler l'intérêt de cette propriété.

Machines alternantes

Une *machine de Turing alternante* est une machine de Turing dont les états sont étiquetés par les symboles booléens $\wedge, \vee, \top, \bot$. Lorsqu'on arrive dans un état $\top$, la machine accepte ; lorsqu'on arrive dans un état $\bot$, la machine rejette ; lorsqu'on arrive dans un état $\wedge$, la machine explore toutes les branches et accepte si toutes les branches acceptent ; lorsqu'on arrive dans un état $\vee$, la machine explore toutes les branches et accepte si l'une des branches accepte.

Formellement, une machine de Turing alternante à k bandes est un 6-uplet $(Q, q_0, \Sigma, \Gamma, \delta, \lambda)$ où :

- Q est un ensemble fini d'états,
- $q_0 \in Q$ est l'état initial,
- Σ est l'alphabet d'entrée, on y ajoute un marqueur de fin $\$ \notin \Sigma$,
- Γ est l'alphabet de travail, on y ajoute un caractère blanc $\# \notin \Gamma$,
- $\delta \subseteq Q \times (\Gamma \cup \{\#\})^k \times (\Sigma \cup \{\$\}) \times Q \times \Gamma^k \times \{\textbf{left}, \textbf{right}\}^k \times \{\textbf{left}, \textbf{right}\}$ est la fonction de transition,
- $\lambda : Q \longrightarrow \{\wedge, \vee, \top, \bot\}$ est la fonction d'étiquetage des états.

Une transition $(q, (a_1, \ldots, a_k), c, q', (b_1, \ldots, b_k), (m_1, \ldots, m_k), m) \in \delta$ indique que la machine peut passer de l'état q à l'état q' en lisant $(a_1, \ldots a_k)$ sur les bandes de travail et c sur l'entrée, et qu'elle écrit alors $(b_1, \ldots, b_k)$ sur les bandes de travail et se déplace dans la direction m_i sur chaque bande i, et dans la direction m sur l'entrée.

Comme pour les machines de Turing non déterministes, une machine de Turing alternante accepte une entrée en temps t si elle l'accepte en n'explorant que des configurations accessibles en au plus t étapes depuis la configuration initiale.

Une machine de Turing alternante accepte une entrée en espace s si elle l'accepte en n'explorant que des configurations dans lesquelles au plus s caractères sont écrits sur les bandes.

On note $\mathsf{ATIME}(f(n))$ (respectivement $\mathsf{ASPACE}(f(n))$) l'ensemble des langages acceptés par une machine de Turing alternante en temps (respectivement en espace) $f(n)$, n étant la taille de l'entrée.

Question 4 *Montrer que*

1. *pour* $f(n) \geq n$, *on a* $\mathsf{ATIME}(f(n)) \subseteq \mathsf{SPACE}(f(n))$.
2. *pour* $f(n) \geq \log(n)$, *on a* $\mathsf{ASPACE}(f(n)) \subseteq \bigcup_{c>0} \mathsf{TIME}(c^{f(n)})$.
3. *pour* $f(n) \geq n$, *on a* $\mathsf{NSPACE}(f(n)) \subseteq \bigcup_{c>0} \mathsf{ATIME}(cf(n)^2)$.
4. *pour* $f(n) \geq n$ *et* $c > 0$, *on a* $\mathsf{TIME}(f(n)) \subseteq \mathsf{ASPACE}(c\log(f(n)))$.

Automates alternants

Une machine alternante qui n'a pas de bande de travail ($k = 0$) et qui ne se déplace que vers la droite sur la bande de lecture, est appelée un *automate alternant*.

Question 5 *Montrer que pour tout automate alternant à* n *états, il existe un automate non déterministe à* 2^{2^n} *états qui accepte le même langage.*

Chapitre 2

La classe NP

2.1 Satisfaisabilité d'une formule propositionnelle

Une *formule* de logique propositionnelle φ est définie inductivement par :
— φ peut être **true**, **false** ou une *proposition atomique* dans un ensemble dénombrable $Prop$;
— $\varphi = \neg\varphi_1$, $\varphi = \varphi_1 \vee \varphi_2$, $\varphi = \varphi_1 \wedge \varphi_2$ où φ_1, φ_2 sont des formules.

Une *interprétation* ν est une fonction de $Prop$ dans $\{\mathbf{true}, \mathbf{false}\}$.

Soit ν une interprétation. Alors ν s'étend aux formules comme suit.
— $\nu(\neg\varphi_1) = \overline{\neg}(\nu(\varphi_1))$;
— $\nu(\varphi_1 \vee \varphi_2) = \overline{\vee}(\nu(\varphi_1), \nu(\varphi_2))$;
— $\nu(\varphi_1 \wedge \varphi_2) = \overline{\wedge}(\nu(\varphi_1), \nu(\varphi_2))$.

Ici $\overline{\neg}$ (resp. $\overline{\vee}$, $\overline{\wedge}$) est la fonction de $\{\mathbf{true}, \mathbf{false}\}$ (resp. $\{\mathbf{true}, \mathbf{false}\}^2$) dans $\{\mathbf{true}, \mathbf{false}\}$ correspondant à l'opérateur $\neg$ (resp. $\vee$, $\wedge$).

D'autres opérateurs sont définis comme des abréviations :
— $\varphi \Rightarrow \psi \equiv (\neg\varphi) \vee \psi$;
— $\varphi \Leftrightarrow \psi \equiv (\varphi \Rightarrow \psi) \wedge (\psi \Rightarrow \varphi)$;
— etc.

Par exemple $\varphi = p_1 \wedge (p_2 \vee \neg p_1)$ est une telle formule avec $p_1, p_2 \in Prop$. Si ν est définie par $\nu(p_1) = \nu(p_2) = \mathbf{false}$ alors $\nu(\varphi) = \mathbf{false}$

On dit que φ est *satisfaite* par ν si $\nu(\varphi) = \mathbf{true}$. On le note ainsi : $\nu \models \varphi$. On dit que φ est *satisfaisable* s'il existe une interprétation ν telle que φ est *satisfaite* par ν.

Proposition 2 *Soit φ une formule de logique propositionnelle. Alors le problème de la satisfaisabilité de φ est dans* NP.

Preuve
La machine non déterministe commence par deviner une interprétation ν des propositions présentes dans la formule puis elle évalue $\nu(\varphi)$ de manière récursive en utilisant la définition des interprétations et renvoie $\nu(\varphi)$.

Les deux étapes se font en temps linéaire donc polynomial.

c.q.f.d. $\Diamond\Diamond\Diamond$

La borne supérieure est en fait optimale ainsi que le démontre la proposition suivante.

Proposition 3 *Soit φ une formule de logique propositionnelle. Alors le problème de la satisfaisabilité de φ est* NP*-difficile (donc* NP*-complet).*

Preuve
Soit une machine de Turing non déterministe $\mathcal{M}$ qui s'exécute en temps polynomial vis à vis de son entrée w, disons $p(n) \geq n$ où p est un polynôme et n est la taille de w. Q est l'ensemble des états, Σ l'alphabet de la bande, $\flat \in \Sigma$ le blanc, *qinit*, *qacc*, *qrej* les états initial, d'acceptation et de rejet. Par conséquent, la bande de la machine de Turing utilise au plus $p(n)$ cellules. On suppose de plus qu'une fois arrivée dans les états *qacc* et *qrej*, la machine reste indéfiniment dans ces états.

Nous allons simuler une exécution réussie de la machine ainsi. Soit $0 \leq i \leq p(n)$ un indice de cellule et $0 \leq j \leq p(n)$ un indice du temps d'exécution, on introduit un ensemble de propositions atomiques relatives à l'état de l'exécution à l'instant j concernant la cellule i.

Nous codons une configuration à l'instant $j \leq p(n)$ par les propositions suivantes :

— Pour tout $q \in Q$, q^j est **true** si l'état est q ;
— Pour tout $0 \leq i \leq p(n)$, i^j est **true** si la position de la tête est i ;
— Pour tout $0 \leq i \leq p(n), a \in \Sigma$, a_i^j est **true** si la i^{th} cellule contient a.

Nous notons $\mathbf{s}^j$ l'ensemble de ces propositions.

Étant donnée une configuration $\mathbf{c}$ de $\mathcal{M}$ opérant sur w, $\nu_{\mathbf{c}}^j$ est l'interprétation de $\mathbf{s}^j$ correspondant à $\mathbf{c}$.

La formule $\varphi_{\mathcal{M},w}$ est une conjonction de sous-formules parmi lesquelles, pour tout $j \leq p(n)$:

— $\bigvee_{q \in Q} q^j$ et pour tout $q \neq q' \in Q$, $\neg q^j \vee \neg (q')^j$;
— $\bigvee_{i \leq p(n)} i^j$ et pour tout $i < i' \leq p(n)$, $\neg i^j \vee \neg (i')^j$;
— pour tout $i \leq p(n)$, $\bigvee_{a \in \Sigma} a_i^j$ et pour tout $a \neq a' \in \Sigma$, $\neg a_i^j \vee \neg (a')_i^j$.

Ces sous-formules assurent qu'étant donnée une interprétation $\nu \models \varphi_{\mathcal{M},w}$, pour tout j, il y a une unique configuration $\mathbf{c}_\nu^j$ telle que $\nu(\mathbf{s}^j) = \nu_{\mathbf{c}_\nu^j}(\mathbf{s}^j)$.

Les sous-formules suivantes sont relatives aux configurations initiales et finales :

$$qinit^0,\ 1^0,\ \$^0,\ w[1]_1^0, \ldots, w[n]_n^0,\ \flat_{n+1}^0, \ldots, \flat_{p(n)}^0,\ qacc^{p(n)}.$$

Ces sous-formules assurent qu'étant donnée une interprétation $\nu \models \varphi_{\mathcal{M},w}$, $\mathbf{c}_\nu^0$ est la configuration initiale et $\mathbf{c}_\nu^{p(n)}$ est une configuration acceptante.

Soit $\delta(q,a) = \{(nq_1(q,a), na_1(q,a), dp_1(q,a)), \ldots, (nq_K(q,a), na_K(q,a), dp_K(q,a))\}$.

Les sous-formules suivantes sont relatives aux pas de $\mathcal{M}$. Pour tout $j < p(n)$, $i \leq p(n)$, $a \in \Sigma$:

— $(\neg i^j) \Rightarrow (a_i^j \Leftrightarrow a_i^{j+1})$;
— pour tout $q \in Q$, $(q^j \wedge i^j \wedge a_i^j) \Rightarrow \bigvee_{k \leq K} nq_k(q,a)^{j+1} \wedge (i + dp_k(q,a))^{j+1} \wedge na_k(q,a)_i^{j+1}$ avec une conjonction de la conclusion remplacée par **false** quand $i + dp_k(q,a) \notin [0, p(n)]$.

Ces sous-formules assurent qu'étant donnée une interprétation $\nu \models \varphi_{\mathcal{M},w}$, pour tout $j < p(n)$, $\mathbf{c}_\nu^j \rightarrow_{\mathcal{M}} \mathbf{c}_\nu^{j+1}$.

Par conséquent si $\nu \models \varphi_{\mathcal{M},w}$ alors $\mathbf{c}_\nu^0, \ldots, \mathbf{c}_\nu^{p(n)}$ est un calcul acceptant pour w.

Réciproquement supposons que $\mathbf{c}^0, \ldots, \mathbf{c}^{p(n)}$ est un calcul acceptant. Alors ν, définie pour tout j par $\nu(\mathbf{s}^j) = \nu_{\mathbf{c}^j}(\mathbf{s}^j)$, satisfait φ.

c.q.f.d. ◊◊◊

2.2 3SAT

Un *littéral* est soit une proposition atomique soit la négation d'une proposition atomique. Une *clause* (disjonctive) est une disjonction de littéraux. Une formule sous forme normale conjonctive (CNF) est une conjonction de clauses. Ainsi $\varphi \equiv p_1 \wedge (p_2 \vee \neg p_1)$ est une formule CNF composée de deux clauses. Une formule 3CNF est une formule CNF dont les clauses ont au plus trois littéraux. 3SAT est le problème correspondant à la satisfaisabilité d'une formule 3CNF.

Deux formules φ et ψ sont équivalentes si pour toute interprétation ν, on a : $\nu(\varphi) = \nu(\psi)$. Pour toute formule, on peut construire une formule CNF équivalente comme suit.

- Pousser les négations devant les propositions :
  - $\neg\neg\varphi \equiv \varphi$;
  - $\neg(\varphi_1 \vee \varphi_2) \equiv (\neg\varphi_1) \wedge (\neg\varphi_2)$;
  - $\neg(\varphi_1 \wedge \varphi_2) \equiv (\neg\varphi_1) \vee (\neg\varphi_2)$.
- Pousser les disjonctions sous les conjonctions : $(\varphi_1 \wedge \varphi_2) \vee \varphi_3 \equiv (\varphi_1 \vee \varphi_3) \wedge (\varphi_2 \vee \varphi_3)$.

Cependant la formule obtenue est taille exponentielle et cette explosion est inévitable.

Proposition 4 *3SAT est* NP*-difficile (donc* NP*-complet).*

Preuve
Soit φ une formule arbitraire, on construit en temps polynomial une formule 3CNF ψ comme suit.

- Pour toute occurrence d'un opérator, on ajoute une nouvelle proposition et on étiquette le noeud de l'arbre syntaxique de la formule par cette proposition ;
- Les clauses of ψ sont définies comme suit. La proposition étiquetant la racine est une clause et pour tout sommet interne de l'arbre :
  - Si c'est une négation étiquetée par x et y étiquette son fils, alors $\neg x \vee \neg y$ et $x \vee y$ sont des clauses ;
  - Si c'est une conjonction étiquetée par x et y, z étiquettent ses fils, alors $x \vee \neg y \vee \neg z$ et $\neg x \vee y$, $\neg x \vee z$ sont des clauses ;
  - Si c'est une disjonction étiquetée par x et y, z étiquettent ses fils, alors $\neg x \vee y \vee z$ et $x \vee \neg y$, $x \vee \neg z$ sont des clauses.

Voici un exemple de traduction.

$$\varphi = (\neg(p \wedge q)) \vee r$$
$$\psi = xv \wedge (\neg xv \vee xn \vee r) \wedge (xv \vee \neg xn) \wedge (xv \vee \neg r)$$
$$\wedge\ (\neg xn \vee \neg xw) \wedge (xn \vee xw)\ \wedge$$
$$(xw \vee \neg p \vee \neg q) \wedge (\neg xw \vee p) \wedge (\neg xw \vee q)$$

φ est satisfaisable si et seulement si ψ est satisfaisable.

• Supposons que $\nu \models \varphi$. Définissons ν' étendant ν sur les nouvelles propositions ainsi. Soit x une proposition correspondant à un noeud interne de sous-formule φ_x. On choisit $\nu'(x) = \nu(\varphi_x)$. On vérifie immédiatement que $\nu' \models \psi$.

• Supposons que $\nu \models \psi$. Par induction sur la taille des sous-formules, on établit que $\nu(x) = \nu(\varphi_x)$. En examinant la clause x où x étiquette la racine, on obtient $\nu(\varphi) = \mathbf{true}$.

c.q.f.d. ◊◊◊

On pourrait se demander si ce résultat est aussi valide pour des clauses d'au plus deux littéraux (forme 2CNF). Ce n'est pas le cas : la satisfaisabilité d'une formule 2CNF se résout en temps linéaire (voir les TD).

2.3 Recherche de circuit hamiltonien

On se donne un graphe orienté $G = (V, E)$ où $V = \{v_0, \ldots, v_{n-1}\}$ est l'ensemble des sommets et $E \subseteq V \times V$ est l'ensemble des arcs. Un circuit hamiltonien est une permutation α de $\{0, \ldots, n-1\}$ telle que $\forall 0 \leq i < n\ (v_{\alpha(i)}, v_{\alpha(i+1\%n)}) \in E$. Autrement dit, un circuit hamiltonien est un circuit qui passe une fois et une seule par tout sommet du graphe.

Proposition 5 *Soit $G = (V, E)$ un graphe orienté. Le problème de l'existence d'un circuit hamiltonien est* NP*-complet.*

Preuve
Pour tester l'existence d'un circuit hamiltonien, on « devine » une suite de n sommets tels qu'un nouveau sommet ne soit pas un sommet déjà choisi et qui soit la destination d'un arc dont le précédent sommet est la source. Si on parvient à choisir tous les sommets et qu'il y a un arc du dernier sommet au premier sommet, on a obtenu un circuit hamiltonien. Ceci démontre que ce problème appartient à NP.

Pour démontrer que ce problème est NP-difficile, on réduit (en temps polynomial) le problème de satisfaisabilité d'une formule CNF au problème de l'existence d'un circuit hamiltonien. Soit $\varphi \equiv \bigwedge_{j=1}^{m} c_j$ une formule CNF bâtie à partir des propositions $p_1, \ldots, p_n$. Sans perte de généralité (pourquoi ?), on fait l'hypothèse qu'une proposition n'apparaît pas à la fois positivement et négativement dans une clause.

Le graphe est composé des sommets d, f, $p_{i,j}$ avec $1 \leq i \leq n$ et $1 \leq j \leq 3m+3$ et des sommets c_j avec $1 \leq j \leq m$. Les arcs du graphe indépendants des clauses sont les suivants :

- On a les arcs $(d, p_{1,1})$, $(d, p_{1,3m+3})$, $(p_{n,1}, f)$, $(p_{n,3m+3}, f)$, (f, d).
- Pour tout $1 \leq i \leq n-1$ on a les arcs $(p_{i,1}, p_{i+1,1})$, $(p_{i,1}, p_{i+1,3m+3})$, $(p_{i,3m+3}, p_{i+1,1})$, $(p_{i,3m+3}, p_{i+1,3m+3})$.
- Pour tout $1 \leq i \leq n$ et tout $1 \leq j \leq 3m+2$, on a les arcs $(p_{i,j}, p_{i,j+1})$, $(p_{i,j+1}, p_{i,j})$.

Ce sous-graphe est représenté sur la figure 2.1.

Observons différentes façons de fabriquer un circuit hamiltonien pour le graphe réduit aux sommets d , f et $\{p_{i,j}\}$.

- On démarre avec le sommet d puis on continue par $p_{1,1}$ ou $p_{1,3m+3}$. Puis selon le sommet initial on parcourt (dans cet ordre) les sommets $p_{1,2}, \ldots, p_{1,3m+2}$ ou les sommets $p_{1,3m+2}, \ldots, p_{1,1}$.
- Arrivé en $p_{1,1}$ ou en $p_{1,3m+3}$, on choisit comme successeur le sommet $p_{2,1}$ ou le sommet $p_{2,3m+3}$ et on itère la construction.

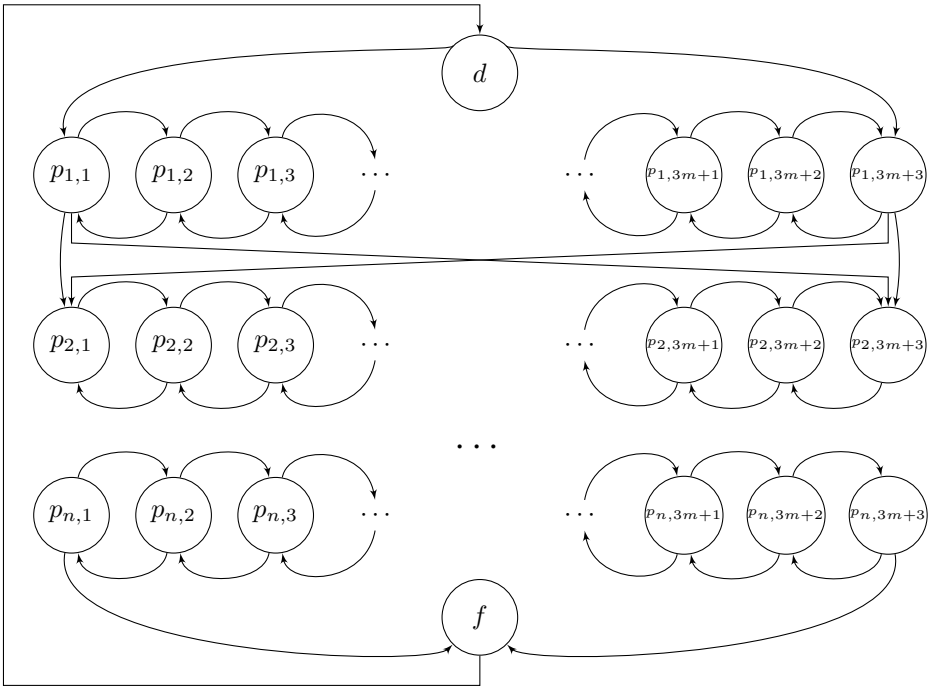


FIGURE 2.1: Le sous-graphe indépendant de la formule

— Arrivé en $p_{n,1}$ ou en $p_{n,3m+3}$, on continue par f et on boucle le circuit en d.

Pour chaque clause j où apparaît positivement la proposition p_i, on ajoute un « détour » sous forme de deux arcs $(p_{i,3j}, c_j)$, $(c_j, p_{i,3j+1})$. De même pour chaque clause j où apparaît négativement la proposition p_i, on ajoute un « détour » sous forme de deux arcs $(p_{i,3j+1}, c_j)$, $(c_j, p_{i,3j})$. Ces ajouts sont représentés sur la figure 2.2.

Supposons que φ est satisfaite par l'interprétation ν. On construit d'abord un circuit (non hamiltonien) de la manière suivante. On démarre en d et on poursuit par $p_{1,1}$ si $\nu(p_0) = \textbf{true}$ et par $p_{1,3m+3}$ sinon puis on parcourt les sommets $p_{1,j}$ ainsi qu'indiqué précédemment. On continue en $p_{2,1}$ si $\nu(p_2) = \textbf{true}$ et en $p_{2,3m+3}$ sinon et on itère le procédé. On se termine par f et on boucle le

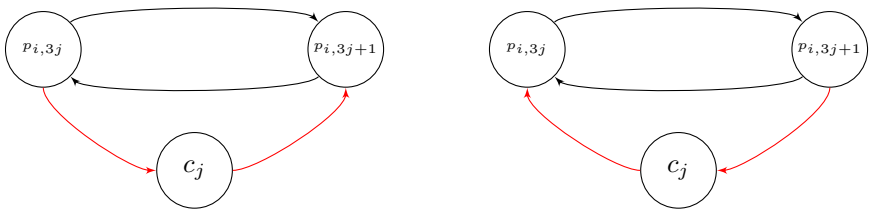


FIGURE 2.2: Ajout d'arcs par clause

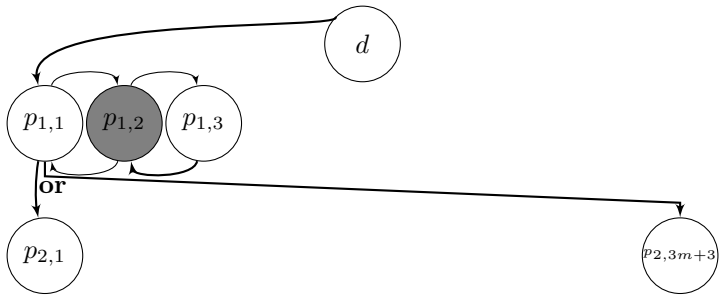


FIGURE 2.3: Un scénario impossible

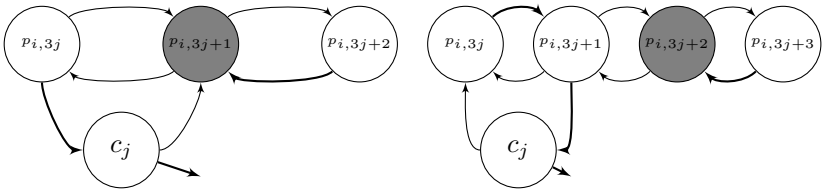


FIGURE 2.4: Deux autres scénarios impossibles

circuit. On choisit ensuite pour chaque clause c_j un littéral x_j tel que $\nu(x_j) =$ **true**. Supposons que $x_j = p_i$ alors on transforme le circuit en remplaçant l'arc $(p_{i,3j}, p_{i,3j+1})$ par le chemin $(p_{i,3j}, c_j), (c_j, p_{i,3j+1})$. Supposons que $x_j = \neg p_i$ alors on transforme le circuit en remplaçant l'arc $(p_{i,3j+1}, p_{i,3j})$ par le chemin $(p_{i,3j+1}, c_j), (c_j, p_{i,3j})$. Le nouveau circuit est bien un circuit hamiltonien.

Supposons maintenant que l'on dispose d'un circuit hamiltonien. Nous allons analyser la structure d'un tel circuit. Tout circuit contient l'arc (f, d). Nous démarrons de cet arc. Après d, on poursuit par $p_{1,1}$ ou $p_{1,3m+3}$. Supposons que le circuit visite $p_{1,1}$ (le cas $p_{1,3m+3}$ est similaire). Si le prochain sommet n'est pas $p_{1,2}$ alors $p_{1,2}$ ne peut être visité que par $p_{1,3}$ et le circuit ne peut plus être prolongé (voir la figure 2.3). Donc $p_{1,2}$ est visité après $p_{1,1}$. De $p_{1,2}$ on ne peut aller qu'en $p_{1,3}$.

A partir de $p_{1,3}$, il est possible d'avoir un arc $(p_{1,3}, c_1)$. Supposons que le prochain sommet visité soit c_1 mais qu'on ne revienne pas en $p_{1,4}$. Le sommet $p_{1,4}$ ne peut être visité qu'à partir de $p_{1,5}$ mais le circuit ne peut plus être prolongé. Supposons maintenant qu'il existe un arc $(p_{1,4}, c_1)$ (cas exclusif du précédent) et qu'on l'emprunte. Le sommet $p_{1,5}$ ne peut être ensuite visité que du sommet $p_{1,6}$ mais le circuit ne peut être ensuite prolongé. Ces deux scénarios sont représentés sur la figure 2.4.

Par conséquent, en itérant le raisonnement on visite successivement les sommets $p_{1,j}$ par ordre croissant de j. Arrivé en $p_{1,3m+3}$, on ne peut continuer qu'en $p_{2,1}$ ou $p_{2,3m+2}$. On parcourt donc successivement les sommets $p_{i,j}$ par ordre i croissant et par ordre j soit croissant soit décroissant. On termine le circuit par f. Le circuit fait exactement un détour par clause.

On affecte à p_i la valeur **true** si les sommets $p_{i,j}$ sont parcourus par ordre croissant et **false** sinon. Par construction du graphe (et des détours) cette affectation satisfait la formule.

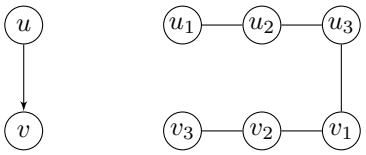


FIGURE 2.5: D'un graphe orienté à un graphe non orienté

c.q.f.d. ◊◊◊

2.4 Recherche de cycle hamiltonien

On se donne un graphe non orienté $G = (V, E)$ où $V = \{v_0, \ldots, v_{n-1}\}$ est l'ensemble des sommets et $E \subseteq 2^V$ (avec $\forall e \in E,\ |e| = 2$) est l'ensemble des arêtes. Un cycle hamiltonien est une permutation α de $\{0, \ldots, n-1\}$ telle que $\forall 0 \leq i < n\ \{v_{\alpha(i)}, v_{\alpha(i+1\%n)}\} \in E$. Autrement dit, un cycle hamiltonien est un cycle qui passe une fois et une seule par tout sommet du graphe.

Proposition 6 *Soit $G = (V, E)$ un graphe non orienté. Le problème de l'existence d'un cycle hamiltonien est* NP*-complet.*

Preuve
Pour tester l'existence d'un cycle hamiltonien, on « devine » une suite de n sommets tels qu'un nouveau sommet ne soit pas un sommet déjà choisi et qui soit adjacent à une arête dont le précédent sommet est aussi adjacent. Si on parvient à choisir tous les sommets et qu'il y a un arête liant le dernier sommet au premier sommet, on a obtenu un cycle hamiltonien. Ceci démontre que ce problème appartient à NP.

Nous allons réduire le problème du circuit hamiltonien au problème du cycle hamiltonien. Soit un graphe orienté $G = (V, E)$, on construit un graphe non orienté $G = (V', E')$ ainsi (voir la figure 2.5) :

— $V' = \{u_i \mid u \in V \wedge i \in \{1, 2, 3\}\}$.
— $E' = \{\{u_3, v_1\} \mid (u, v) \in E\} \cup \{\{u_1, u_2\}, \{u_2, u_3\} \mid u \in V\}$

Supposons qu'il existe un circuit hamiltonien dans G, on fabrique le cycle hamiltonien ainsi. Pour chaque arc (u, v) du circuit, on construit le chemin $\{u_1, u_2\}, \{u_2, u_3\}, \{u_3, v_1\}$. En concaténant ces chemins, on obtient le cycle hamiltonien recherché.

Supposons qu'il existe un cycle hamiltonien dans G' que l'on décrit par une suite de sommets débutant (et se terminant) par un sommet r_1. Soit un sommet u_2, nécessairement les trois sommets u_1, u_2, u_3 apparaissent consécutivement dans le cycle (dans l'un ou l'autre des ordres). Par conséquent soit le cycle débute par r_1, r_2, r_3 soit il se termine par r_3, r_2, r_1. Supposons qu'il débute par r_1, r_2, r_3 (l'autre cas est similaire en inversant l'énumération). Par induction, le cycle ne peut pas inverser cet ordre puisqu'il n'y a pas d'arête $\{u_3, v_3\}$.

Aussi si les sommets de G sont désignés par $\{u^1, \ldots, u^n\}$ alors le cycle peut être noté $(u_1^{\sigma(1)}, u_2^{\sigma(1)}, u_3^{\sigma(1)}, \ldots, u_1^{\sigma(n)}, u_2^{\sigma(n)}, u_3^{\sigma(n)})$ impliquant que σ, une permutation, définit un circuit dans G.

c.q.f.d. ◊◊◊

2.5 Problème du sac à dos

Nous démontrons qu'une variante du problème du sac à dos est NP-complète. Cette variante est le problème de la somme de sous-ensembles. On a en entrée $n+1$ nombres $\{v_1, \ldots, v_n, w\}$ et on cherche à déterminer s'il existe un sous-ensemble $I \subseteq \{1, \ldots, n\}$ tel que $\sum_{i \in I} v_i = w$. Notons que la taille de cette représentation est en $O(\sum_i \log(v_i + 1) + \log(w + 1))$.

Proposition 7 *Le problème de la somme de sous-ensembles est* NP*-complet.*

Preuve
On devine un sous-ensemble en temps linéaire puis on effectue la somme des éléments de ce sous-ensemble et on la compare à w. Ceci se fait en temps linéaire et démontre que le problème dans NP.

Pour démontrer que ce problème est NP-difficile, on réduit (en temps polynomial) le problème de satisfaisabilité d'une formule 3CNF au problème de la somme de sous-ensembles. Les nombres que l'algorithme produira sont écrits en base 10. Le nombre de chiffres de ces nombres sera au plus $m + n$ où m est le nombre de clauses et n le nombre de propositions atomiques. On parlera dans la suite du chiffre indicé par une clause ou par une proposition.

Il y aura $2n + 2m$ nombres que nous notons $\{v_1, \ldots, v_{2m+2}\}$ écrits avec $n + m$ chiffres. Etant donné un nombre x écrit avec $n + m$ chiffres, $x[j]$ est le j^{ieme} chiffre de x en partant de la gauche.

— Pour tout $i \leq n$, pour tout $j \leq n$, $v_{2i-1}[j] = v_{2i}[j] = 1_{i=j}$;
— Pour tout $i \leq n$, pour tout $j \leq m$,
  1. si p_i apparaît dans c_j alors $v_{2i-1}[n+j] = 1$ sinon $v_{2i-1}[n+j] = 0$;
  2. si $\neg p_i$ apparaît dans c_j alors $v_{2i}[n+j] = 1$ sinon $v_{2i}[n+j] = 0$;
— Pour tout $i \leq m$, pour tout $j \leq n+m$, $v_{2n+2i-1}[j] = v_{2n+2i}[j] = 1_{n+i=j}$.

Pour tout $j \leq n$, $w[j] = 1$ et pour tout $n < j \leq n + m$, $w[j] = 3$.

Le tableau ci-dessous décrit le problème. Ici on a supposé que p_i apparaît positivement dans c_j et négativement dans c_k.

	1	i	n	$n+1$	$n+j$	$n+k$	$n+m$
v_{2i-1}	0	1	0		1	0	
v_{2i}	0	1	0		0	1	
$v_{2n+2j-1}, v_{2n+2j}$	$\cdots$	0	$\cdots$	0	1	0	0
	1	$\cdots$	1	3	$\cdots$	$\cdots$	3

On observe d'abord que la somme de tous les v_i ne donne pas lieu à une retenue car le maximum possible pour un chiffre est égal à 5.

Supposons que la formule soit satisfaisable par l'interprétation ν alors on choisit pour $1 \leq i \leq n$ l'indice $2i - 1$ si $\nu(p_i) = \mathbf{true}$ et l'indice $2i$ sinon. Pour chaque $1 \leq j \leq m$, on choisit $2n + 2j - 1$ et $2n + 2j$ si le nombre de littéraux qui satisfont c_j est 1, et $2n + 2j - 1$ si le nombre de littéraux qui satisfont c_j est 2. Il est immédiat qu'un tel choix fournit le nombre recherché.

Supposons qu'il existe un sous-ensemble I qui est solution du problème de la somme de sous-ensembles. Par construction I contient soit $2i-1$ soit $2i-1$ mais pas les deux simultanément. Définissons $\nu(p_i) = \textbf{true}$ si $2i-1$ est choisi et $\nu(p_i) = \textbf{false}$ sinon. Puisque la somme est égale est au nombre recherché, pour chaque chiffre associé à une clause l'un des littéraux associé à la clause s'évalue à **true**.

c.q.f.d. ◊◊◊

Nous allons maintenant montrer l'importance de la représentation binaire. L'algorithme 1 est dit *pseudo-polynomial* : il est polynomial si les nombres apparaissant dans l'instance sont représentés en unaire.

Proposition 8 *Soit $\{v_1, \ldots, v_n, w\}$ une suite de $n+1$ nombres. Le problème de la somme de sous-ensembles se résout en $O(nw)$.*

Preuve
L'algorithme 1 maintient T un tableau de booléens indicé par $0 \leq j \leq w$ tel qu'au début de chaque itération l'invariant suivant soit vérifié :

$$\forall j \; T[j] = \textbf{true} \Leftrightarrow \exists I \subseteq \{1, \ldots, i-1\} \; \sum_{k \in I} v_k = j$$

La formule suivante garantit la préservation de l'invariant :

$$\exists I \subseteq \{1, \ldots, i\} \; \sum_{k \in I} v_k = j \Leftrightarrow \exists I \subseteq \{1, \ldots, i-1\} \; \sum_{k \in I} v_k = j \vee \sum_{k \in I} v_k = j - v_i$$

c.q.f.d. ◊◊◊

Algorithme 1 : Un algorithme pseudo-polynomial

`SacADos`(n, v, w) : un booléen
Input : n, w des entiers, $v_1, \ldots, v_n$ n entiers
Output : l'évaluation de $\exists I \subseteq \{1, \ldots, n\} \; \sum_{k \in I} v_k = w$
Data : i, j des indices, T un tableau de w booléens

$T[0] \leftarrow \textbf{true}$
for i **from** 1 **to** w **do** $T[i] \leftarrow \textbf{false}$

for i **from** 1 **to** n **do**
  `/*` $\forall j \; T[j] = \textbf{true} \Leftrightarrow \exists I \subseteq \{1, \ldots, i-1\} \; \sum_{k \in I} v_k = j$ `*/`
  for j **from** $w - v_i$ **downto** 0 **do if** $T[j]$ **then** $T[j + v_i] \leftarrow \textbf{true}$

end
return $T[w]$

FIGURE 2.6: Le graphe pondéré de la somme de sous-ensembles

2.6 Chemins dans des graphes pondérés

Jusqu'à présent les démonstrations que les problèmes appartenaient à NP étaient assez immédiates alors que le caractère NP-difficile nécessitait plus de raisonnement. Nous allons maintenant étudier un problème pour lequel c'est exactement l'inverse.

Un graphe orienté pondéré est un graphe orienté $G = (V, E)$ muni d'une fonction de poids sur les arcs, disons $p : E \mapsto \mathbb{N}$. Étant donné un chemin $\sigma \equiv u_0, \ldots, u_n$ dans le graphe, son poids $p(\sigma)$ est défini par : $p(u_0, u_1) + \ldots + p(u_{n-1}, u_n)$

Le problème de recherche d'un chemin dans un graphe pondéré est défini par le graphe $G = (V, E, p)$, deux sommets distingués u, v et un poids a. Il consiste à déterminer s'il existe un chemin σ de u à v tel que $p(\sigma) = a$.

Proposition 9 *Le problème de recherche d'un chemin dans un graphe pondéré est* NP*-difficile.*

Preuve
On réduit le problème de la somme de sous-ensembles au problème du chemin dans un graphe pondéré. Soit $\{v_1, \ldots, v_n, w\}$ un problème de somme de sous-ensembles. On construit le graphe ainsi (voir la figure 2.6) :

— $V = \bigcup_{1 \leq i \leq n} \{x_i, z_i\} \cup \{x_0\}$
— Pour tout $1 \leq i \leq n$, il existe des arcs (z_i, x_i) et (x_{i-1}, x_i) pondérés par 0 et un arc (x_{i-1}, z_i) pondéré par v_i.

On cherche à décider s'il existe un chemin de x_0 à x_n de poids w. Il existe 2^n chemins de x_0 à x_n dont le poids de chacun correspond à une somme de sous-ensembles. Ceci établit la correction de la réduction.

c.q.f.d. ◊◊◊

On remarque que s'il existe un chemin de poids a, alors il existe un chemin de longueur inférieure ou égale à $(a + 1)|V|$. En effet, si on associe à chaque sommet sa « distance » au sommet u alors lorsqu'un sommet est visité deux fois avec la même distance, le circuit peut être éliminé sans modifier le poids du chemin. Cependant si on essaye de deviner le chemin alors puisque la longueur du chemin est exponentielle, on n'obtient pas un algorithme dans NP. Afin de contourner ce problème, nous nous appuyons sur le lemme suivant. Nous ne traitons que le cas $u \neq v$ (le cas $u = v$ est similaire).

Nous introduisons quelques notations utiles. *L'image de Parikh* $\mathbf{v}_\rho \in \mathbb{N}^E$ d'un chemin $\rho \in E^*$ est le vecteur qui compte les occurrences d'arcs dans ρ défini inductivement par :

$$\mathbf{v}_\varepsilon[e] = 0, \text{ for } e' \neq e, \mathbf{v}_{\rho e'}[e] = \mathbf{v}_{\rho e'}[e] \text{ et } \mathbf{v}_{\rho e}[e] = \mathbf{v}_\rho[e] + 1$$

Le *support* $Supp(\mathbf{v})$ d'un vecteur $\mathbf{v} \in \mathbb{N}^E$ est défini par :

$$Supp(\mathbf{v}) = \{e \in E \mid \mathbf{v}[e] > 0\}$$

Soit $E' \subseteq E$, $G_{E'} = (V_{E'}, E')$ le graphe *induit* par E' est défini par :

$$V_{E'} = \{v, v' \mid (v, v') \in E'\}$$

Lemme 1 (Euler) *Soit $G = (V, E)$ un graphe orienté, $s \neq t \in V$ and $\mathbf{v} \in \mathbb{N}^E$. Alors $\mathbf{v}$ est l'image de Parikh d'un chemin de s à t si et seulement si :*

- *$G_{Supp(\mathbf{v})}$ est connexe ;*
- *$\sum_{(s,u)\in E} \mathbf{v}[(s,u)] = 1 + \sum_{(u,s)\in E} \mathbf{v}[(u,s)]$;*
- *$\sum_{(u,t)\in E} \mathbf{v}[(u,t)] = 1 + \sum_{(t,u)\in E} \mathbf{v}[(t,u)]$;*
- *Pour tout $w \notin \{s,t\}$, $\sum_{(u,w)\in E} \mathbf{v}[(u,w)] = \sum_{(w,u)\in E} \mathbf{v}[(w,u)]$.*

Preuve

La nécessité des conditions est immédiate.

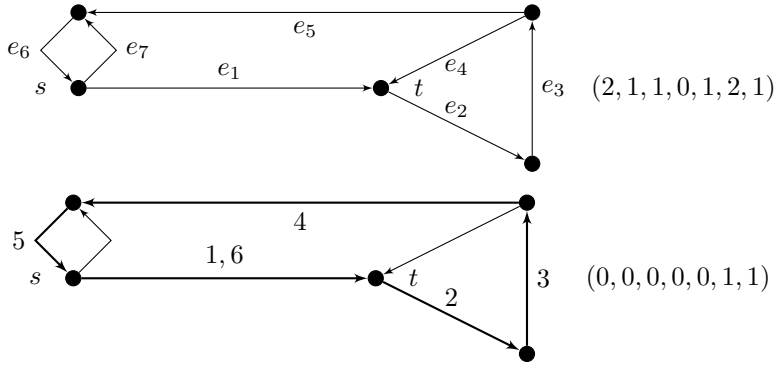
On construit le chemin en maintenant un vecteur d'occurrences initialisé à $\mathbf{v}$. On démarre en s et on choisit un arc sortant de s appartenant à $Supp(\mathbf{v})$ (il y en a au moins un) on décrémente la composante de $\mathbf{v}$ qui correspond à l'arc choisi. Le sommet courant devient l'extrémité de cet arc et on itère le procédé qui s'arrête faute d'un arc sortant dans le support.

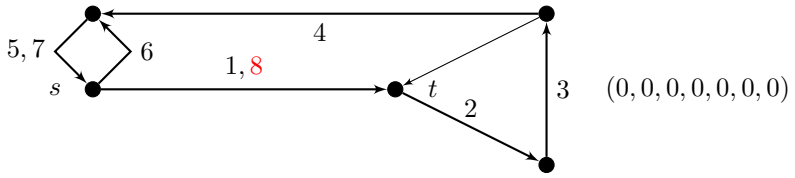
- Le sommet courant ne peut pas être s car il y plus d'arcs sortant de s que d'arcs entrant en s dans $\mathbf{v}$.
- Le sommet est donc t puisque tout autre sommet compte autant d'arcs entrants que sortants dans $\mathbf{v}$. De plus tous les arcs connexes à t ont été utilisés.

Si le vecteur courant est nul alors le chemin obtenu convient. Sinon on remarque d'abord que dans le vecteur courant chaque sommet a autant d'arcs entrants que sortants. D'autre part, puisque le sous-graphe est connexe, il existe un arc dont une extrémité (disons w) appartient au chemin. S'il s'agit d'un arc entrant, alors il existe un arc sortant de w non utilisé d'après notre remarque. On emprunte donc un arc sortant de w et on construit un nouveau chemin avec le même procédé. Lorsque le chemin ne peut plus être étendu, l'extrémité du chemin est forcément w (toujours d'après notre remarque). Il s'agit donc d'un circuit (pas nécessairement élémentaire) que l'on peut combiner avec le chemin précédent pour obtenir un chemin « plus grand » de s à t. Cette procédure se termine nécessairement fournissant le chemin recherché.

c.q.f.d. ◊◊◊

La figure ci-dessous illustre la preuve du lemme d'Euler.





Proposition 10 *Le problème de recherche d'un chemin dans un graphe pondéré est dans* NP.

Preuve
Plutôt que deviner un chemin de longueur inférieure ou égale à $(a+1)|V|$, (grâce au lemme d'Euler) on devine $\mathbf{v}$ un vecteur d'occurrences d'arcs dont les composantes sont bornées par $a+1$ (donc de taille polynomiale) puis on vérifie que :

- le sous-graphe du support du vecteur est connexe ;
- pour tout sommet $w \notin \{s,t\}$, (1) le nombre de transitions entrant en w est égal au nombre de transitions sortant de w dans $\mathbf{v}$, (2) le nombre de transitions sortant de s est égal au nombre de transitions entrant en t plus un et (3) le nombre de transitions sortant de t est égal au nombre de transitions entrant en t moins un ;
- le « poids » du vecteur est égal à a : $\sum_{(x,y)\in E} \mathbf{v}[(x,y)]p(x,y) = a$

Ces vérifications s'effectuent en temps polynomial. Ce qui permet de conclure.

c.q.f.d. ◊◊◊

2.7 TD 2

Exercice 1 (INDEPENDENT SET) *Un* ensemble indépendant *dans un graphe non orienté $G = (V, E)$ est un ensemble $C \subseteq V$ de sommets dont aucun sommet n'est relié à aucun autre par une arête de G, c'est-à-dire tel que $u, v \in C$ implique $\{u, v\} \notin E$.*
Démontrer que le langage INDEPENDENT SET défini comme suit est NP*-complet.*

ENTRÉE : *un graphe non orienté $G = (V, E)$, un entier $m \in \mathbb{N}$ écrit en unaire ou en binaire (peu importe) ;*

QUESTION : *G a-t-il un ensemble indépendant de cardinal au moins m ?*

Montrer que INDEPENDENT SET reste NP*-complet même lorsqu'il est restreint aux graphes où chaque sommet est au plus de degré 4.*

Exercice 2 (NODE COVER) *Un* recouvrement *C d'un graphe non orienté $G = (V, E)$ est un ensemble $C \subseteq V$ de sommets tel que toute arête de E est incidente à C, c'est-à-dire à au moins un élément de C. Démontrer que le langage NODE COVER défini comme suit est* NP*-complet.*

ENTRÉE : *un graphe non orienté $G = (V, E)$, un entier $m \in \mathbb{N}$ écrit en unaire ou en binaire (peu importe) ;*

QUESTION : *G a-t-il un recouvrement de cardinal au plus m ?*

Exercice 3 (CLIQUE) *Une* clique *C d'un graphe non orienté $G = (V, E)$ est un sous-ensemble $C \subseteq V$ induisant un sous-graphe complet de G, c'est-à-dire tel que pour tous $u, v \in C$ avec $u \neq v$, on a $\{u, v\} \in E$. Montrer que le problème CLIQUE défini comme suit est* NP*-complet.*

ENTRÉE : *un graphe non orienté $G = (V, E)$, un entier $m \in \mathbb{N}$ écrit en unaire ou en binaire (peu importe) ;*

QUESTION : *G a-t-il une clique de cardinal au moins m ?*

Exercice 4 (3-COLORING) *Une* k-coloration *d'un graphe non orienté $G = (V, E)$ est une fonction $c : V \to \{0, \dots k-1\}$ telle que si $\{u, v\} \in E$ alors $c(u) \neq c(v)$.*
Montrer que le problème de 3-coloration est NP*-complet.*

ENTRÉE : *un graphe non orienté G ;*

QUESTION : *Existe-t-il une 3-coloration de G ?*

Exercice 5 (GRAPH HOMOMORPHISM) *Un homomorphisme d'un graphe $G = (V, E)$ à un graphe $G' = (V', E')$ est une fonction $h : V \to V'$ telle que pour tout $\{v_1, v_2\} \in E$, on a $\{h(v_1), h(v_2)\} \in E'$.*
Montrer que le problème suivant est NP*-complet.*

ENTRÉE : *deux graphes non orientés, G_1 et G_2 ;*

QUESTION : *Existe-t-il un homomorphisme de G_1 à G_2 ?*

Exercice 6 (SUBGRAPH ISOMORPHISM) *Deux graphes $G = (V, E)$ et $G' = (V', E')$ sont isomorphes si $|V| = |V'|$ et $|E| = |E'|$ et il existe une fonction bijective $h : V \to V'$ telle que $\{v_1, v_2\} \in E$, si et seulement si $\{h(v_1), h(v_2)\} \in E'$.*
Montrer que le problème suivant est NP*-complet.*

ENTRÉE : *Deux graphes G et H.*
QUESTION : *Est-ce que G contient un sous-graphe isomorphe à H ?*

Chapitre 3

La classe PSPACE

3.1 Le théorème de Savitch

Il est bien connu que le problème de l'accessibilité dans un graphe peut être résolu à l'aide d'un algorithme opérant en temps linéaire (parcours en profondeur ou en largeur d'un graphe). Cependant ces algorithmes requièrent a minima un tableau de booléens indicé par les états pour mémoriser les sommets déjà visités. L'algorithme décrit dans la preuve du théorème suivant abaisse significativement la taille de l'espace mémoire additionnel.

Théorème 5 (Savitch) *Le problème de l'accessibilité dans un graphe orienté à n sommets se résout par un algorithme en taille d'espace $O(\log^2(n))$.*

Preuve
L'algorithme 2 est basé une procédure récursive, Savitch qui prend en entrée trois paramètres, un sommet source u, un sommet destination v et une longueur ℓ. Cette procédure renvoie vrai s'il existe un chemin de longueur au plus ℓ de u à v. Initialement, cette procédure sera appelée avec les deux sommets pour lesquels on veut tester l'accessibilité et une longueur égale à $n-1$. La procédure procède ainsi :

- Si $\ell \leq 1$ elle teste si $u = v$ ou s'il existe un arc (u, v).
- Si $\ell > 1$ elle parcourt l'ensemble des sommets avec une variable w. Au cours d'une itération, elle teste par deux appels récursifs s'il existe un chemin d'une longueur au plus $\lceil \frac{\ell}{2} \rceil$ de u à w et un chemin d'une longueur au plus $\lfloor \frac{\ell}{2} \rfloor$ de w à v. Si elle les trouve elle renvoie vrai. À la fin des itérations, elle renvoie faux.

La correction de cette procédure est évidente. Analysons sa complexité en espace. Il y au plus $\lceil \log_2(n) \rceil + 1$ appels emboîtés. Les sommets sont représentés par des identifiants de taille $\lceil \log_2(n) \rceil + 1$ puisqu'il y a n sommets. Enfin la longueur maximale est $n-1$ et peut aussi être codée sur $\lceil \log_2(n) \rceil + 1$ bits. Par conséquent chaque appel consomme $O(\log_2(n))$ espace.

c.q.f.d. ◊◊◊

Observez que cet algorithme ne s'exécute pas en temps polynomial ce qui s'explique intuitivement par le fait que l'algorithme effectue de nombreux appels avec les mêmes paramètres.

Algorithme 2 : Un algorithme d'accessibilité

Input : $G = (V, E)$ un graphe orienté, $s \neq t \in V$
`Savitch`(u, v, ℓ) : un booléen
Input : $u, v \in V$, ℓ un entier
Output : Existe-t-il un chemin de u à v de longueur au plus ℓ ?
Data : $w \in V$

if $\ell \leq 1$ **then return** $u = v \vee (u, v) \in E$

for $w \in E$ **do**
  if `Savitch`$(u, w, \lceil \frac{\ell}{2} \rceil) \wedge$ `Savitch`$(w, v, \lfloor \frac{\ell}{2} \rfloor)$ **then return true**
end
return false

`Savitch`$(s, t, |V| - 1)$

L'accessibilité dans un graphe est un problème très proche de l'acceptation d'un mot par une machine de Turing non déterministe qui opère en espace borné.

Corollaire 1 *Pour toute fonction f space-calculable t.q. $f(n) \geq \log(n)$,* $\mathsf{NSPACE}(f(n)) \subseteq \mathsf{SPACE}(f^2(n))$

Preuve
Soit $\mathcal{M}$ une machine de Turing non déterministe opérant en espace $f(n)$ sur un mot x de taille n. On fait l'hypothèse non restrictive qu'il existe une unique configuration acceptante. On construit $\mathcal{M}'$ une machine de Turing déterministe $\mathcal{M}'$ opérant en espace $O(f^2(n))$ ainsi. Elle calcule d'abord $f(n)$ pour déterminer la taille des configurations à considérer. Puis $\mathcal{M}'$ teste l'accessibilité de la configuration acceptante à partir de la configuration initiale en implémentant l'algorithme du théorème 5. Elle ne consulte son entrée que lorsqu'elle teste l'accessibilité en au plus un pas. Autrement dit, elle ne construit pas le graphe des configurations. La contrainte sur $\log(n)$ est nécessaire car dans une configuration de $\mathcal{M}$, la représentation de la position de la tête de lecture de la bande d'entrée occupe $O(\log(n))$ bits.

c.q.f.d. ◊◊◊

Le corollaire suivant est certainement le plus utilisé mais on a aussi $\mathsf{EXPSPACE} = \mathsf{NEXPSPACE}$, etc.

Corollaire 2 $\mathsf{PSPACE} = \mathsf{NPSPACE}$

Par contre on a uniquement :

$$\mathsf{LOGSPACE} \subseteq \mathsf{NLOGSPACE} \subseteq \mathsf{LOG^2SPACE}$$

3.2 Problèmes PSPACE-complets

3.2.1 Satisfaisabilité d'une formule booléenne quantifiée

Une formule booléenne quantifiée (QBF) φ est inductivement définie par :

— φ peut être **true**, **false** ou une proposition atomique ;
— $\varphi = \neg\varphi_1$, $\varphi = \varphi_1 \vee \varphi_2$, $\varphi = \varphi_1 \wedge \varphi_2$ où φ_1, φ_2 sont des QBF ;
— $\varphi = \exists p\ \varphi_1$, $\varphi = \forall p\ \varphi_1$, où φ_1 est une QBF et p est une proposition atomique non quantifiée dans φ_1.

Soit ν une interprétation. Alors ν s'étend aux QBF comme suit.
— $\nu(\neg\varphi_1) = \overline{\neg}(\nu(\varphi_1))$;
— $\nu(\varphi_1 \vee \varphi_2) = \overline{\vee}(\nu(\varphi_1), \nu(\varphi_2))$;
— $\nu(\varphi_1 \wedge \varphi_2) = \overline{\wedge}(\nu(\varphi_1), \nu(\varphi_2))$;
— $\nu(\exists p\ \varphi_1) = \overline{\vee}(\nu(\varphi_1[\mathbf{true}/p]), \nu(\varphi_1[\mathbf{false}/p]))$;
— $\nu(\forall p\ \varphi_1) = \overline{\wedge}(\nu(\varphi_1[\mathbf{true}/p]), \nu(\varphi_1[\mathbf{false}/p]))$.

Les quantificateurs peuvent être poussés en début de formule (*la forme prenex*) :
— $\neg\exists p\ \varphi \equiv \forall p\ \neg\varphi$;
— $(\exists p\ \varphi) \wedge \theta \equiv \exists q\ (\varphi[q/p] \wedge \theta)$ où q n'apparaît ni dans φ ni dans θ.
— etc.

Une occurrence de proposition est soit sous la portée d'un quantificateur soit *libre*. Une QBF *close* est une QBF où toutes les occurrences de propositions sont quantifiées. L'évaluation d'une QBF close est indépendante de l'interprétation et s'effectue à l'aide de l'algorithme 3. Le nombre d'appels emboîtés est au plus $n+1$ où n est le nombre de quantificateurs. Chaque appel consomme un espace linéaire pour stocker les paramètres et les variables locales. Par conséquent, cet algorithme est dans PSPACE.

Algorithme 3 : Évaluation d'une QBF close

`EvalQBF`$(Q_1 p_1 \dots Q_n p_n\ \psi)$: un booléen
Input : n, w des entiers, $v_1, \dots, v_n$ n entiers
Output : l'évaluation de $\exists I \subseteq \{1, \dots, n\}\ \sum_{k \in I} v_k = w$
Data : i, j des indices, T un tableau de w booléens
if $n = 0$ **then return** `EvalCons`(ψ)

$b_1 \leftarrow$ EvalQBF$(Q_2 p_2 \dots Q_n p_n\ \psi[\mathbf{true}/p_1])$
$b_2 \leftarrow$ EvalQBF$(Q_2 p_2 \dots Q_n p_n\ \psi[\mathbf{false}/p_1])$
if $Q_1 = \exists$ **then return** b_1 **or** b_2
else return b_1 **and** b_2

Proposition 11 *Soit φ une formule booléenne quantifiée close. Alors le problème de l'évaluation de φ est* PSPACE*-difficile et donc* PSPACE*-complet.*

Preuve
Soit $\mathcal{M}$ une machine de Turing déterministe opérant en espace polynomial $p(n)$ où n est la taille de l'entrée w.
Nous codons une configuration à l'aide des propositions suivantes indexées par x (afin de considérer plusieurs configurations) :
— Pour tout $q \in Q$, q^x est **true** si l'état est q ;
— Pour tout $0 \leq i \leq p(n)$, i^x est **true** si la position de la tête est i ;

— Pour tout $0 \leq i \leq p(n), a \in \Sigma$, a_i^x est **true** si la i^{eme} cellule contient a.

L'ensemble de ces propositions est notée $\mathbf{x}$ et aussi $\{x_i\}_{i \leq m}$ (via un renommage).

Observons que $m = O(p(n))$.

Soit $\mathbf{c}$ une configuration. L'interprétation de $\mathbf{x}$ correspondant à $\mathbf{c}$ est notée $\nu_{\mathbf{c}}^x$.

Nous allons définir une formule QBF φ_i avec $\mathbf{x}$ et $\mathbf{y}$ pour propositions libres telle que pour toutes configurations $\mathbf{c}$, $\mathbf{c}$' :

— Soit ν définie par $\nu(\mathbf{x}) = \nu_{\mathbf{c}}^x(\mathbf{x})$ et $\nu(\mathbf{y}) = \nu_{\mathbf{c}'}^y(\mathbf{y})$;

— $\nu(\varphi_i) =$ **true** ssi $\mathbf{c}$' est accessible de $\mathbf{c}$ en au plus 2^i pas ;

— Pour tout ν' tel que $\nu'(\mathbf{x}) = \nu_{\mathbf{c}}^x(\mathbf{x})$ et $\nu'(\varphi_i) =$ **true** il existe une configuration $\mathbf{c}$" telle que $\nu'(\mathbf{y}) = \nu_{\mathbf{c}''}^y(\mathbf{y})$.

La formule θ teste l'égalité de deux configurations : $\theta = \bigwedge_{i \leq m} x_i \Leftrightarrow y_i$. Afin d'examiner l'accessibilité en un pas de la machine, nous notons la fonction de transition ainsi : $\delta(q, a) = (nq(q, a), na(q, a), dp(q, a))$.

Soit ψ la conjonction des sous-formules :

— Pour tout i, a $\neg i^x \Rightarrow a_i^x \Leftrightarrow a_i^y$;

— Pour tout i, q, a $i^x \wedge q^x \wedge a_i^x \Rightarrow nq(q, a)^y \wedge na(q, a)_i^y \wedge (i + dp(q, a))^y$ avec la conclusion substituée par **false** si $i + dp(q, a) \notin [0, p(n)]$;

— Pour tout $q \neq q'$ $\neg q^y \vee \neg (q')^y$;

— Pour tout $i \neq i'$ $\neg i^y \vee \neg (i')^y$;

— Pour tout $a \neq a'$, pour tout i, $\neg a_i^y \vee \neg\ a'^y_i$.

Alors $\varphi_0 = \theta \vee \psi$ satisfait les propriétés requises. Observons que $\varphi_0 \leq Cm$ pour un certain C qui dépend de la machine.

La formule φ_{i+1} « devine » une configuration intermédiaire à l'aide de φ_i. La formule ci-dessous est correcte :

$$\varphi_{i+1} = \exists \mathbf{z}\ \varphi_i[\mathbf{z}/\mathbf{y}] \wedge \varphi_i[\mathbf{z}/\mathbf{x}] \qquad (\dagger)$$

Cependant elle présente un problème : la taille de φ_{i+1} est au moins le double de la taille de φ_i. Nous allons donc employer une astuce logique :

$$\varphi_{i+1} = \exists \mathbf{z}\ \forall \mathbf{t}\ \forall \mathbf{u}\ (\mathbf{t} = \mathbf{x} \wedge \mathbf{u} = \mathbf{z}) \vee (\mathbf{t} = \mathbf{z} \wedge \mathbf{u} = \mathbf{y}) \Rightarrow \varphi_i[\mathbf{t}/\mathbf{x}, \mathbf{u}/\mathbf{y}] \qquad (\ddagger)$$

On obtient donc l'inégalité : $|\varphi_{i+1}| \leq |\varphi_i| + 12m$. Par induction, on a donc : $|\varphi_i| \leq (12i + C)m$.

La formule correspondant à la réduction est $\varphi = \varphi_m[\nu_{\mathbf{c}_0}/\mathbf{x}, \nu_{\mathbf{c}_f}/\mathbf{y}]$ où $\mathbf{c}_0$ (resp. $\mathbf{c}_f$) est la configuration initiale (resp. acceptante) avec $|\varphi| \leq (12m + C)m$.

c.q.f.d. ◊◊◊

Il est important de remarquer que c'est l'« astuce logique » utilisée dans (‡) pour définir φ_{i+1} en fonction de φ_i qui introduit des quantificateurs universels "$\forall \mathbf{t} \forall \mathbf{u}$". En utilisant (†) on obtient une formule propositionnelle *existentielle*, i.e., une instance de SAT, mais qui est de taille exponentielle.

3.2.2 QSAT

Soit une formule QBF $\varphi = Q_1 p_1 \ldots Q_n p_n\ \psi$ où ψ est sous forme 3CNF. Le problème QSAT consiste à évaluer φ.

Proposition 12 (QSAT) *QSAT est* PSPACE*-complet.*

Preuve
Soit $\varphi = Q_1 p_1 \dots Q_n p_n\ \theta$ une formule QBF. Soit ψ la formule 3CNF de la réduction pour 3SAT en partant de θ. Notons $x_1, \dots, x_m$ les propositions additionnelles de ψ. La preuve de 3SAT établit en fait que $\theta \equiv \exists x_1 \dots \exists x_m\ \psi$. Par conséquent, $\varphi \equiv Q_1 p_1 \dots Q_n p_n\ \exists x_1 \dots \exists x_m\ \psi$.

c.q.f.d. ◊◊◊

3.2.3 Universalité des langages réguliers

Soit Σ un alphabet. Une expression rationnelle E est inductivement définie par :

— E est $\emptyset$, ε ou $a \in \Sigma$;

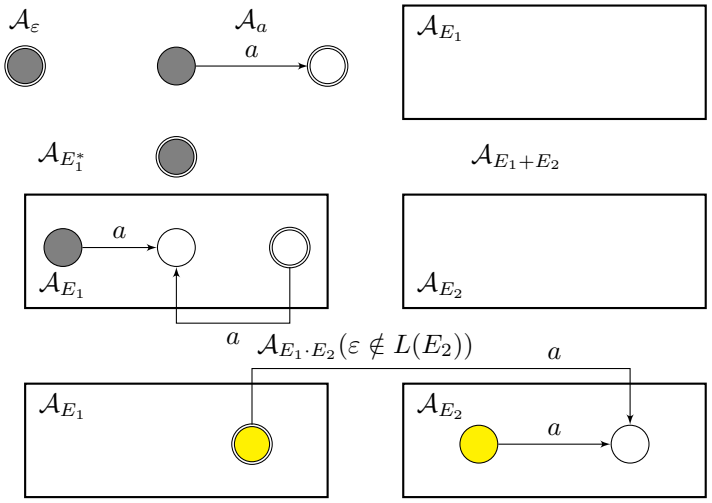
— E est $E_1 + E_2$, $E_1 \cdot E_2$, E_1^* où E_1, E_2 sont des expressions rationnelles.

Le langage régulier $L(E) \subseteq \Sigma^*$ est inductivement défini par :
— $L(\emptyset) = \emptyset$, $L(\varepsilon) = \{\varepsilon\}$, $L(a) = \{a\}$;
— $L(E_1 + E_2) = L(E_1) \cup L(E_2)$;
— $L(E_1 \cdot E_2) = \{w_1 w_2 \mid w_1 \in L(E_1) \wedge w_2 \in L(E_2)\}$;
— $L(E_1^*) = \{w_1 \dots w_n \mid \forall i \leq n\ w_i \in L(E_1)\}$.

Le *problème de l'universalité* d'une expression rationnelle E consiste à vérifier si $L(E) = \Sigma^*$.

Proposition 13 *Soit E une expression rationnelle. Alors le problème de l'universalité de $L(E)$ est dans* PSPACE.

Preuve
On construit en temps polynomial un automate non déterministe $\mathcal{A}$ qui reconnaît $L(E)$. Cette construction procède inductivement. Nous illustrons ci-dessous la plupart des constructions inductives. Les états initiaux sont grisés et les états finals sont doublement cerclés. Le fond jaune indique que les états ont perdu leur statut.



Supposons que $\mathcal{A}$ ne reconnaisse pas un mot, alors $\mathcal{A}^c$ l'automate déterministe complémentaire obtenu par la construction des sous-ensembles reconnaît ce mot. Il y a donc un chemin dans $\mathcal{A}^c$ de l'état initial vers un état final. Or l'automate complémentaire a au plus 2^n états. Donc le chemin le plus court de l'état initial vers un état final a une longueur au plus égale à $2^n - 1$.

Nous recherchons ce chemin par une procédure non déterministe sans construire $\mathcal{A}^c$. Cette procédure maintient un compteur initialisé à $2^n - 1$ et un état courant de $\mathcal{A}^c$ (i.e. un sous-ensemble d'états de $\mathcal{A}$). Elle choisit de manière non déterministe une lettre de Σ et construit le successeur de l'état courant dans $\mathcal{A}^c$ et décrémente le compteur. Ceci s'effectue uniquement à l'aide de $\mathcal{A}$. Elle itère ce processus jusqu'à ce qu'elle rencontre un état final de $\mathcal{A}^c$ et renvoie vrai ou que le compteur soit nul et elle renvoie faux.

Cette procédure occupe un espace polynomial (compteur et sous-ensemble d'états représentés en $O(n)$). Il suffit alors d'appliquer NPSPACE=PSPACE.

c.q.f.d. ◊◊◊

Algorithme 4 : Un algorithme NPSPACE

`Univers`($\mathcal{A}$) : un booléen
Input : $\mathcal{A} = (\Sigma, Q, Q_0, F, \delta)$, un automate non déterministe
Output : $L(\mathcal{A}) = \Sigma^*$?
Data : *cpt*, un compteur, Q' un sous-ensemble d'états

$cpt \leftarrow 0$; $Q' \leftarrow Q_0$
repeat
  if $Q' \cap F = \emptyset$ **then return true**

  $cpt \leftarrow cpt + 1$
  Guess $a \in \Sigma$
  $Q' \leftarrow \bigcup_{q \in Q'} \delta(q, a)$
until $cpt = 2^{|Q|}$
return false

La borne supérieure est en fait optimale ainsi que le démontre la proposition suivante.

Proposition 14 *Soit E une expression rationnelle sur un alphabet Σ et $L(E)$ son langage. Alors le problème de l'universalité de $L(E)$ est* PSPACE*-difficile.*

Preuve
Soit une machine de Turing déterministe $\mathcal{M}$ qui s'exécute en espace polynomial vis à vis de son entrée x, disons $p(n)$ où p est un polynôme et n est la taille de x. Q est l'ensemble des états, T les symboles de la bande, $\flat \in T$ le blanc, q_0, q_f les états initial et d'acceptation.

Nous associons un mot à chaque exécution de la machine. L'alphabet de ce mot est $\Sigma \equiv T \cup \{qX \mid q \in Q \wedge X \in T\} \cup \{\sharp\}$. On note $\Delta = \Sigma \setminus \{\sharp\}$ et $QT = \{qX \mid q \in Q \wedge X \in T\}$.

— Une configuration $\mathbf{c} = (t, i, q)$ où t, de taille $p(n)$, est le contenu de la bande, i la position de la tête et q est l'état est codé par un mot $v_{\mathbf{c}} = v_{\mathbf{c}}[1] \dots v_{\mathbf{c}}[p(n)]$ avec :
  1. pour tout $j \neq i$, $v_{\mathbf{c}}[j] = t[j]$;
  2. $v_{\mathbf{c}}[i] = qt[i]$.

— Une exécution $\mathbf{c}_1 \dots \mathbf{c}_k$ est codée par le mot $\sharp v_{\mathbf{c}_1} \sharp \dots \sharp v_{\mathbf{c}_k} \sharp$.

Nous allons construire une expression rationnelle $E_{\mathcal{M},w}$ qui accepte les mots qui ne sont pas des codages d'exécution ou ceux qui codent des exécutions qui ne rencontrent pas l'état d'acceptation q_f. Autrement dit, $L(E_{\mathcal{M},w}) = \Sigma^*$ ssi w n'est pas accepté par $\mathcal{M}$. Décomposons $E_{\mathcal{M},w} = A+B+C+D$ où les expressions A, B, C, D correspondent aux différents cas possibles.

1. A dénote les mots v qui ne contiennent pas de lettre $q_f X$;
2. B dénote les mots v tels que $\sharp v_{\mathbf{c_0}}$ n'est pas un préfixe de v ;
3. C dénote les mots v qui ne sont pas de la forme $v = \sharp v_1 \sharp v_2 \sharp \dots \sharp v_k \sharp$ où pour tout $i \leq k$:
   (a) $|v_i| = p(n)$;
   (b) exactement une lettre de v_i appartient à QT ;
   (c) Les autres lettres de v_i appartiennent à T.
4. D dénote les mots v avec $|v| > 2 + p(n)$ qui ne contiennent pas deux configurations successives de $\mathcal{M}$.

Introduisons les notations suivantes.

— Soit $S = \{s_1, \dots, s_k\} \subseteq \Sigma$, $\hat{S} \equiv s_1 + \dots + s_k$;

— Soit E une expression et k un entier strictement positif, $E^k \equiv E \cdot E \cdots E \cdot E$ avec k occurrences de E.

La spécification de A, B et C ne soulève pas de difficulté particulière. Soit $\Sigma_r = \Sigma \setminus \{q_f X \mid X \in T\}$. Alors $A = \hat{\Sigma}_r^*$. A est de taille indépendante de n.

Nous définissons des expressions intermédiaires afin de spécifier B.

— $E_{2,1} = \hat{\Delta} \cdot \hat{\Sigma}^*$;

— Soit $\Sigma_1 = \Sigma \setminus \{q_0 w[1]\}$. Alors $E_{2,2} = \hat{\Sigma} \cdot \hat{\Sigma}_1 \cdot \hat{\Sigma}^*$;

— Pour tout $2 \leq i \leq n$, soit $\Sigma_i = \Sigma \setminus \{w[i]\}$. Alors $E_{2,i} = \hat{\Sigma}^i \cdot \hat{\Sigma}_i \cdot \hat{\Sigma}^*$;

— Soit $\Sigma_\flat = \Sigma \setminus \{\flat\}$. Alors pour tout $n+1 \leq i \leq p(n)$, $E_{2,i} = \hat{\Sigma}^i \cdot \hat{\Sigma}_\flat \cdot \hat{\Sigma}^*$.

On obtient B ainsi : $B = E_{2,1} + \dots + E_{2,p(n)} + \varepsilon + \hat{\Sigma} + \dots + \hat{\Sigma}^{p(n)}$. La taille de B appartient à $O(p^2(n))$.

Nous définissons des expressions intermédiaires afin de spécifier C.

— Pour tout $0 \leq i \leq p(n) - 1$, $E_{3,i} = \hat{\Sigma}^* \cdot \sharp \cdot \hat{\Delta}^i \cdot \sharp \cdot \hat{\Sigma}^*$ *(les configurations sont trop courtes)*

— $E_{3,p(n)+1} = \hat{\Sigma}^* \cdot \sharp \cdot \hat{\Delta}^{p(n)+1} \cdot \hat{\Sigma}^*$ *(les configurations sont trop longues)*

— $F_3 = \hat{\Delta}^* + \hat{\Delta}^* \cdot \sharp \cdot \hat{\Delta}^* + \hat{\Delta} \cdot \hat{\Sigma}^* + \hat{\Sigma}^* \cdot \hat{\Delta}$ *(les occurrences de $\sharp$ ne respectent pas la spécification)*

— $G_3 = \hat{\Sigma}^* \cdot \sharp \cdot T^* \cdot \sharp \cdot \hat{\Sigma}^* + \hat{\Sigma}^* \cdot \widehat{QT} \cdot \hat{T}^* \cdot \widehat{QT} \cdot \hat{\Sigma}^*$ *(les occurrences des lettres de QT ne respectent pas la spécification)*

On obtient C ainsi : $C = E_{3,1} + \dots + E_{3,p(n)-1} + E_{3,p(n)+1} + F_3 + G_3$. La taille de C appartient à $O(p^2(n))$.

Supposons qu'un mot v tel que $|v| > 2+p(n)$ code une exécution. Alors pour tout $i > 1$, $v[i+p(n)+1] = f_{\mathcal{M}}(v[i-1]v[i]v[i+1])$ pour une fonction $f_{\mathcal{M}}$ qui ne dépend que de $\mathcal{M}$ (arbitrairement définie pour les triplets qui ne peuvent apparaître dans une exécution). Soit $g_{\mathcal{M}}$ de Σ^3 vers 2^{Σ} définie par $g_{\mathcal{M}}(abc) = \Sigma \backslash \{f_{\mathcal{M}}(abc)\}$.

Pour tout triplet abc, notons $D_{abc} = \hat{\Sigma}^* \cdot a \cdot b \cdot c \cdot \hat{\Sigma}^{p(n)} \cdot \widehat{g_{\mathcal{M}}(abc)} \cdot \hat{\Sigma}^*$ Alors $D = \sum_{abc \in \Sigma^3} D_{abc}$. La taille de C appartient à $O(p(n))$.

Nous laissons le soin au lecteur de vérifier que la construction de E se fait en temps polynomial vis à vis de n.

c.q.f.d. $\Diamond\Diamond\Diamond$

3.3 TD 3

Exercice 1

On note $I = \{1, \ldots, n\}$. Un problème de planification est donné par :

- n variables booléennes $\{x_i\}_{i \in I}$;
- m opérations où chaque opération est définie par une condition de la forme $\bigwedge_{i \in I'} x_i = \alpha_i$ avec $I' \subseteq I$ et une mise à jour de la forme $\{x_i \leftarrow \beta_i\}_{i \in I''}$ avec $I'' \subseteq I$.
  Voici un exemple d'opération : Si $x_1 = \texttt{V} \wedge x_3 = \texttt{F}$ Alors $x_1 \leftarrow \texttt{F}; x_2 \leftarrow \texttt{V}$
- une configuration initiale s_{init} et une configuration finale s_{fin} où une configuration est un assignement de valeurs (i.e. une valuation) aux variables.
- Une opération est applicable à une configuration si la condition de l'opération s'évalue à `V`. Son application consiste à effectuer ses mises à jour pour obtenir la nouvelle configuration. Par exemple l'opération précédente est applicable à la configuration $(\texttt{V}, \texttt{F}, \texttt{F})$ et conduit à la configuration $(\texttt{F}, \texttt{V}, \texttt{F})$.

Le problème consiste à déterminer s'il existe une suite d'applications des opérations (avec éventuellement plusieurs applications d'une même opération) qui conduise de la configuration initiale à la configuration finale.

Question 1. Montrez que le problème de planification est dans PSPACE.

Question 2. Montrez que le problème de planification est PSPACE-difficile. *Indication : on établira une réduction du problème d'acceptation d'un mot w de longueur n par une machine de Turing $\mathcal{M}$ opérant en espace n^k.*

Exercice 2

Un automate déterministe concurrent $\mathcal{A}$ est donné par un ensemble d'automates déterministes $\{\mathcal{A}_i\}_{i \leq n}$ appelés composants. Un état de l'automate déterministe concurrent est un tuple $(s_1, \ldots, s_n)$ composé d'un état par composant. Lorsqu'une lettre a est lue, les automates $\mathcal{A}_i$ qui ont une transition issue de s_i étiquetée par la lettre effectuent simultanément leur transition tandis que les autres conservent leur état. Pour qu'une lettre puisse être lue, au moins un automate doit effectuer une transition. Un mot est reconnu s'il conduit à un tuple d'états terminaux. Le problème de la vacuité consiste à savoir s'il existe au moins un mot accepté par l'automate.

Question 3. Montrez que le problème de la vacuité des automates déterministes concurrents est dans PSPACE.

Question 4. Montrez que le problème de la vacuité est PSPACE-difficile.

Exercice 3

Un problème de géographie est donné par :

- Un graphe orienté $G = (V, E)$ avec un sommet initial v_0 ;

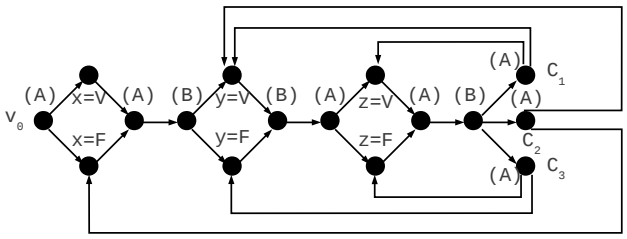


FIGURE 3.1: Réduction de $QSAT$ vers Géographie

— Deux joueurs Alice et Bob qui jouent à tour de rôle en commençant par Alice ;
— Une configuration est donnée par l'ensemble des sommets déjà visités V', le sommet courant $v \in V'$, le joueur qui doit jouer P ;
— Lorsque P joue, il choisit un sommet non encore visité (i.e. $v' \notin V'$) et accessible depuis v (i.e. $(v, v') \in E$). La nouvelle configuration est donnée par le sous-ensemble $V \cup \{v'\}$, le sommet courant v' et le joueur $P' \neq P$. S'il n'existe pas de sommet v' vérifiant ces conditions alors P' gagne.

Le problème consiste à déterminer si Alice a une stratégie gagnante dans la configuration initiale $(\{v_0\}, v_0, Alice)$. On rappelle que pour ce type de jeu, dans une configuration quelconque l'un des deux joueurs a forcément une configuration gagnante.

Question 5. Montrez que le problème de géographie est dans PSPACE en proposant un algorithme récursif qui prend en entrée une configuration quelconque et qui renvoie `V` si le joueur courant a une stratégie gagnante. Vous devrez détailler votre analyse de complexité spatiale.

Question 6. Montrez que le problème de géographie est PSPACE-difficile en réduisant le problème de $QSAT$ au problème de géographie t.q. la formule est vraie ssi Alice gagne. Un exemple de réduction est représenté sur la figure 3.1 pour la formule :

$$\exists x \ \forall y \ \exists z \ C_1 \wedge C_2 \wedge C_3$$

avec $C_1 = \neg y \vee \neg z$, $C_2 = x \vee \neg y$, $C_3 = y \vee z$ Nous avons indiqué entre parenthèses lorsque cela était possible le joueur associé à un sommet quelque soit le déroulement du jeu. Notez aussi qu'un sommet étiqueté par une clause a pour successeurs les sommets étiquetés par les négations de ses littéraux. Il s'agit pour vous de généraliser et de justifier la réduction.

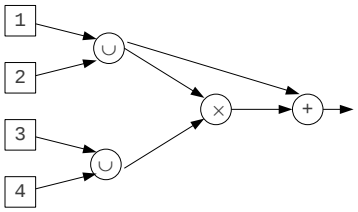
3.4 TD 4

Exercice 1

Un *circuit entier* $\mathcal{C}$ est défini par un graphe orienté sans circuit dont :
- les sommets sans prédécesseur sont appelés les *entrées* et sont étiquetés par un élément de $\mathbb{N}$ (confondu avec le singleton contenant cet entier).
- Les autres sommets sont appelés des *portes* et sont étiquetés par l'une des trois opérations $+, \times, \cup$. Toutes les portes admettent deux prédécesseurs. L'une des portes est appelée *sortie*.

La sémantique d'un circuit consiste à associer à toute porte un ensemble d'entiers défini comme suit. Si g est une porte et E_1, E_2 sont les ensembles associés à ses deux prédécesseurs alors $E(g)$ est défini par :
- g est étiqueté par $+$: $E(g) = \{x + y \mid x \in E_1 \wedge y \in E_2\}$.
- g est étiqueté par $\times$: $E(g) = \{x \times y \mid x \in E_1 \wedge y \in E_2\}$.
- g est étiqueté par $\cup$: $E(g) = E_1 \cup E_2$.

Question 1. Calculez l'ensemble d'entiers associé à la porte de sortie du circuit représenté ci-dessous.



Question 2. Soient $j, k > 0$ deux entiers. Construisez un circuit de sortie *out* ayant pour seule entrée j, tel que $E(out) = \{j, 2j, 3j, \ldots, 2^k j\}$ et comprenant $O(k)$ portes.

Problème du circuit entier. Le problème du circuit entier, noté ICE, est défini par un couple $(\mathcal{C}, x)$ où $\mathcal{C}$ est un circuit entier et x est un entier. Il consiste à déterminer si x appartient à $E(out)$ où *out* désigne la porte de sortie de $\mathcal{C}$.

Question 3. Proposez un algorithme récursif qui prend en entrée un circuit $\mathcal{C}$, un sommet g du circuit $\mathcal{C}$ et un entier x, et détermine si x appartient à $E(g)$.

Question 4. Analysez la complexité spatiale de votre algorithme et en déduire que le problème du circuit entier est dans PSPACE.

Exercice 2

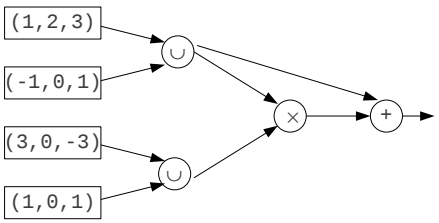
Un *circuit vectoriel* $\mathcal{C}$ de dimension n est défini par un graphe orienté sans circuit dont :

— les sommets sans prédécesseur sont appelés les entrées et sont étiquetés par un vecteur de $\mathbb{Z}^n$ (confondu avec le singleton contenant ce vecteur).
— Les autres sommets sont appelés des portes et sont étiquetés par l'une des trois opérations $+, \times, \cup$. Toutes les portes admettent deux prédécesseurs. L'une des portes est appelée sortie.

La sémantique d'un circuit consiste à associer à toute porte un ensemble de vecteurs défini comme suit. Si g est une porte et E_1, E_2 sont les ensembles associés à ses deux prédécesseurs alors $E(g)$ est défini par :

— g est étiqueté par $+$: $E(g) = \{x + y \mid x \in E_1 \wedge y \in E_2\}$.
— g est étiqueté par $\times$: $E(g) = \{x \times y \mid x \in E_1 \wedge y \in E_2\}$ avec $(x \times y)_i = x_i y_i$, i.e. le produit terme à terme.
— g est étiqueté par $\cup$: $E(g) = E_1 \cup E_2$.

Question 5. Calculez l'ensemble de vecteurs associé à la porte de sortie du circuit vectoriel représenté ci-dessous.



Problème du circuit vectoriel. Le problème du circuit vectoriel, noté VCE, est défini par un couple $(\mathcal{C}, x)$ où $\mathcal{C}$ est un circuit vectoriel et x est un vecteur de même dimension. Il consiste à déterminer si x appartient à $E(out)$ où out désigne la porte de sortie de $\mathcal{C}$.

Dans la suite de cette partie, on établit une réduction en temps polynomial du problème QBF-DNF de l'évaluation d'une formule quantifiée booléenne $\varphi \equiv Q_n x_n \dots Q_1 x_1 \psi$ où ψ est donnée sous forme DNF (i.e. ψ est une disjonction de clauses conjonctives) vers le problème VCE.

Préliminaires.

— Les variables (libres ou liées) des formules de cette partie sont incluses dans $\{x_1, \dots, x_n\}$ et les circuits sont de dimension n.
— Un vecteur $\vec{v} = (v_1, \dots, v_n)$ tel que pour tout i, $v_i \in \{-1, 1\}$ est appelé un *vecteur d'affectation* ; il correspond à l'affectation des variables définie par $x_i = \mathbf{true}$ (resp. $x_i = \mathbf{false}$) si $v_i = 1$ (resp. $v_i = -1$).
— Soit φ une formule, la *table de vérité* de φ, notée $\mathbf{T}(\varphi)$ est l'ensemble des vecteurs d'affectation correspondant aux affectations satisfaisant φ. Attention, les affectations n'affectent que les variables libres. Par conséquent, la table de vérité d'une formule close est soit l'ensemble de tous les vecteurs d'affectation, soit l'ensemble vide.
— $\vec{1}[k]$ désigne le vecteur défini par $\vec{1}[k]_k = 1$ et pour tout $i \neq k$, $\vec{1}[k]_i = 0$. On note aussi $\vec{1} = \sum_{k=1}^{n} \vec{1}[k]$ et pour $p \in \mathbb{Z}$, $\vec{p} = p\vec{1}$.

— $I\vec{n}v[k]$ désigne le vecteur défini par $I\vec{n}v[k]_k = -1$ et pour tout $i \neq k$, $I\vec{n}v[k]_i = 1$.

Question 6. Soit $n = 4$ et la clause $\varphi = x_1 \wedge \neg x_2$. Construisez un circuit $\mathcal{C}$ de sortie *out* dont les entrées sont $(1, -1, 0, 0)$, $\vec{1}[3]$, $-\vec{1}[3]$, $\vec{1}[4]$ et $-\vec{1}[4]$ tel que $E(out) = \mathbf{T}(\varphi)$.

Question 7. Généralisez la construction de la question précédente pour toute clause conjonctive φ et montrez que la taille du circuit construit est polynomiale par rapport à n.

Question 8. Soit φ une formule DNF composée de m clauses conjonctives. Utilisez la construction précédente pour construire un circuit $\mathcal{C}$ de sortie *out* tel que $E(out) = \mathbf{T}(\varphi)$ et dont la taille est polynomiale par rapport à $\max(m, n)$.

Soit la formule $\varphi \equiv Q_n x_n \dots Q_1 x_1 \psi$. On note $\varphi^i \equiv Q_i x_i \dots Q_1 x_1 \psi$. Ainsi $\varphi^0 \equiv \psi$ et $\varphi^n \equiv \varphi$. Un *k-vecteur d'affectation* est un vecteur $\vec{v} = 2^k \vec{w}$ avec $\vec{w}$ vecteur d'affectation ; ainsi un vecteur d'affectation est un 0-vecteur d'affectation. On note $\mathbf{Af}^k$ l'ensemble des k-vecteurs d'affectation et de manière similaire pour une formule φ, on définit $\mathbf{T}^k(\varphi) = \{\vec{v} \mid \exists \vec{w} \in \mathbf{T}(\varphi)\ \vec{v} = 2^k \vec{w}\}$. Un circuit $\mathcal{C}$ de sortie *out*, *k-représente* une formule θ si :

1. $\forall \vec{v} \in E(out)\ \forall i \leq n\ |v_i| \leq 2^k$
2. $\mathbf{T}^k(\theta) = E(out) \cap \mathbf{Af}^k$

Question 9. On suppose que $\varphi^{k+1} \equiv \exists x_{k+1}\ \varphi^k$ et qu'un circuit $\mathcal{C}'$ k-représente φ^k. Construisez à partir de $\mathcal{C}'$, un circuit $\mathcal{C}$ qui $(k+1)$-représente φ^{k+1}.

Question 10. On suppose que $\varphi^{k+1} \equiv \forall x_{k+1}\ \varphi^k$ et qu'un circuit $\mathcal{C}'$ k-représente φ^k. Construisez à partir de $\mathcal{C}'$, un circuit $\mathcal{C}$ qui $(k+1)$-représente φ^{k+1}. On vous demande de justifier la construction.

Question 11. Déduire des questions précédentes qu'il existe une réduction en temps polynomial de QBF-DNF vers VCE (et donc que VCE est PSPACE-difficile). On précisera le vecteur testé dans le problème VCE.

Exercice 3

Dans cette partie on transforme la réduction précédente en une réduction vers le problème du circuit entier noté ICE ce qui démontrera que le problème ICE est PSPACE-difficile et donc PSPACE-complet.

Question 12. Soit $\mathcal{C}$ le circuit vectoriel de la question 11. Supposons que chaque entrée vectorielle soit remplacée par un entier borné par $M \geq 2$ afin de fabriquer un circuit entier $\mathcal{C}'$ de sortie out'. Montrez que tout entier de $E(out')$ est borné par M^{2n+1}.

On admet les deux résultats suivants.

1. Soit p_k le $k$ième nombre premier alors pour k assez grand, $p_k < k^2$.
2. Soient $m_1, \ldots, m_n \in \mathbb{N}$ impairs et premiers entre eux et $M = \prod_{i=1}^{n} m_i$. On note $V = [(-m_1+1)/2, (m_1-1)/2] \times \ldots \times [(-m_n+1)/2, (m_n-1)/2]$. Alors il existe $z_1, \ldots, z_n \in \mathbb{Z}$ calculables en temps polynomial tels que :
   — $h : V \mapsto [0, \ldots, M-1]$ définie par :
   $h(x_1, \ldots, x_n) = \sum_{i=1}^{n} z_i x_i \mod M$ est bijective.
   — Pour tout $\vec{v}, \vec{w} \in V$, si $\vec{v} + \vec{w} \in V$ alors $h(\vec{v} + \vec{w}) = h(\vec{v}) + h(\vec{w}) \mod M$
   — Pour tout $\vec{v}, \vec{w} \in V$, si $\vec{v} \times \vec{w} \in V$ alors $h(\vec{v} \times \vec{w}) = h(\vec{v})h(\vec{w}) \mod M$
   On choisit pour définir h, $m_i = (p_{i+1})^{n+1}$.

Question 13. Montrez que la représentation binaire de M est de taille polynomiale par rapport à n.

Question 14. Soit $\mathcal{C}$ le circuit vectoriel de sortie out de la question 11. Supposons que chaque entrée vectorielle $\vec{v}$ soit remplacée par $h(\vec{v})$ ce qui nous donne un circuit entier $\mathcal{C}'$ de sortie out'. Soit $\vec{v}$ le vecteur associé au problème de la question 11. Montrez que $\vec{v} \in E(out)$ ssi $\exists x \in E(out')\ x \mod M = h(\vec{v})$.

Question 15. En ajoutant une entrée et une porte complétez le circuit $\mathcal{C}'$ en un circuit $\mathcal{C}''$ tel que $\vec{v} \in E(out)$ ssi $\exists x \in E(out'')\ x \mod M = 0$.

Question 16. En vous servant des questions 2 et 12, complétez le circuit $\mathcal{C}''$ en un circuit $\mathcal{C}^*$ de taille polynomiale vis à vis de la taille de $\mathcal{C}$ et de n tel que $\vec{v} \in E(out)$ ssi $M^{2n+2} \in E(out^*)$. Conclure.

Chapitre 4

La classe PTIME

4.1 Satisfaisabilité d'une formule HNF

Soit φ une formula CNF. Alors φ est sous forme normale de Horn (HNF) si pour toute clause ψ de φ :

— soit $\psi = \neg p_1 \vee \cdots \vee \neg p_k \vee q$ (un littéral positif).
Ce qui peut s'écrire $\psi \equiv (p_1 \wedge \cdots \wedge p_k) \Rightarrow q$;

— soit $\psi = \neg p_1 \vee \cdots \vee \neg p_k$ (pas de littéral positif).
Ce qui peut s'écrire $\psi \equiv (p_1 \wedge \cdots \wedge p_k) \Rightarrow \mathbf{false}$.

L'hypothèse de la clause ψ est définie par $H_\psi = p_1 \wedge \cdots \wedge p_k$ avec lorsque $k = 0$, $H_\psi = \mathbf{true}$. La *conclusion* de la clause ψ est définie par $C_\psi = q$ dans le premier cas et $C_\psi = \mathbf{false}$ dans le deuxième cas.

Exemple. Soit la formule φ définie par

$$\varphi = (p) \wedge (p \Rightarrow q) \wedge (p \wedge q \Rightarrow r) \wedge (r \wedge s \Rightarrow \mathbf{false})$$

φ est satisfaisable par une seule interprétation : $\nu(p) = \nu(q) = \nu(r) = \mathbf{true}$ and $\nu(r) = \mathbf{false}$. Cette interprétation a été obtenue car toute interprétation doit satisfaire p, puis par conséquent q et enfin r. La dernière clause a forcé l'interprétation de s.

Le *problème* HORNSAT est le problème de satisfaisabilité d'une formule HNF.

Proposition 15 *HORNSAT appartient à* PTIME.

Preuve
Soit φ, une formule HNF. L'algorithme 5 maintient une interprétation partielle ν (initialement l'interprétation vide) telle que pour toute proposition p, $\nu(p)$ est soit **true** soit non définie, ainsi qu'un ensemble de clauses actives *Clauses* (initialement l'ensemble des clauses de φ). À l'intérieur de la boucle principale, toute interprétation ν' qui satisfait φ étend ν.

Une itération de l'algorithme consiste à parcourir les clauses actives et à traiter chaque clause active ψ telle que $\nu(H_\psi) = \mathbf{true}$ (avec la convention $\mathbf{true} \wedge \bot = \bot \wedge \bot = \bot$).

— si $C_\psi = \mathbf{false}$ alors φ n'est pas satisfaisable ;

— si $C_\psi = q$ alors on définit $\nu(q) = \textbf{true}$ et on désactive ψ. Il est immédiat que les invariants de boucle restent vrais.

Si la boucle interne a modifié ν —comme enregistré par *done*— on recommence.

S'il n'y a plus de clauses actives ψ avec $\nu(H_\psi) = \textbf{true}$, il suffit de compléter l'interprétation ν en interprétant toutes les propositions non encore définies à **false** pour obtenir une interprétation satisfaisant φ.

L'algorithme s'effectue en un temps polynomial mais il peut être optimisé à l'aide d'une implémentation astucieuse pour s'exécuter en temps linéaire.

c.q.f.d. ◊◊◊

Algorithme 5 : Un algorithme de résolution de HORNSAT

$\texttt{Sat}(\varphi)$: un booléen
Input : φ, une formule HNF
Output : φ est-elle satisfaisable ?
Data : *Clauses*, un sous-ensemble de clauses, ψ une clause
Data : p une proposition, ν une interprétation, *done* un booléen

$Clauses \leftarrow Clauses_\varphi$; **for** $p \in Prop_\varphi$ **do** $\nu(p) = \bot$

repeat
  /* Invariant : Tout $\nu' \models \varphi$ étend ν. */
  /* Invariant : $\nu(\psi) = \textbf{true}$ pour $\psi \in Clauses_\varphi \setminus Clauses$. */
  /* Invariant : Soit $p \in Prop_\varphi$, $\nu(p) = \textbf{true}$ ou $\nu(p) = \bot$. */
  $done \leftarrow \textbf{true}$
  for $\psi \in Clauses$ **do**
    if $\nu(H_\psi) = \textbf{true}$ **then**
      if $C_\psi = \textbf{false}$ **then return false**
      $\nu(C_\psi) \leftarrow \textbf{true}$; $Clauses \leftarrow Clauses \setminus \{\psi\}$; $done \leftarrow \textbf{false}$
    end
  end
until *done*
/* Assertion : soit $\psi \in Clauses$, $\exists p \in H_\psi$ avec $\nu(\psi) = \bot$ */
for $p \in Prop_\varphi$ **do if** $\nu(p) = \bot$ **then** $\nu(p) = \textbf{false}$

return ν

La borne supérieure est en fait optimale ainsi que le démontre la proposition suivante.

Proposition 16 *HORNSAT est* PTIME*-difficile (donc* PTIME*-complet).*

Preuve
Soit $\mathcal{M}$ une machine de Turing déterministe opérant en temps polynomial $p(n)$ où n est la taille de l'entrée w.

Nous codons une configuration au temps $j \leq p(n)$ par les propositions suivantes :

— pour tous $q \in Q$, q^j est **true** si l'état est q ;
— pour tous $0 \leq i \leq p(n)$, i^j est **true** si la tête de lecture est en position i ;
— pour tous $0 \leq i \leq p(n)$ et $a \in \Sigma$, a_i^j est **true** si la $i^{\text{ème}}$ cellule contient a.

Nous notons $\mathbf{s}^j$ l'ensemble de ces propositions. Soit $\mathbf{c}$ une configuration de $\mathcal{M}$ avec l'input w, $\nu_{\mathbf{c}}^j$ est l'interprétation de $\mathbf{s}^j$ correspondant à $\mathbf{c}$.

La formule $\varphi_{\mathcal{M},w}$ est une conjonction de clauses de Horn parmi lesquelles pour tout $j \leq p(n)$:

— pour tous $q \neq q' \in Q$: $q^j \wedge (q')^j \implies$ **false** ;
— pour tous $i < i' \leq p(n)$: $i^j \wedge (i')^j \implies$ **false** ;
— pour tous $i \leq p(n)$ et tous $a \neq a' \in \Sigma$: $a_i^j \wedge (a')_i^j \implies$ **false**.

Ces clauses garantissent que pour une interprétation $\nu \models \varphi_{\mathcal{M},w}$, pour tout j, il y a *au plus* une configuration $\mathbf{c}_\nu^j$ telle que $\nu(\mathbf{s}^j) = \nu_{\mathbf{c}_\nu^j}(\mathbf{s}^j)$. [1]

Les clauses suivantes sont relatives aux configurations initiale et finale :

$$qinit^0,\ 1^0,\ \$^0,\ w[1]_1^0, \ldots, w[n]_n^0,\ \flat_{n+1}^0, \ldots, \flat_{p(n)}^0,\ qacc^{p(n)}.$$

Ces clauses garantissent que pour une interprétation $\nu \models \varphi_{\mathcal{M},w}$, $\mathbf{c}_\nu^0$ est définie et est la configuration initiale et, *si elle est définie*, $\mathbf{c}_\nu^{p(n)}$ est une configuration acceptante.

Puisque la machine $\mathcal{M}$ est déterministe, on peut donner ses règles de transitions sous la forme d'une fonction $\delta(q,a) = (nq(q,a), na(q,a), dp(q,a))$. Les clauses suivantes sont relatives aux pas de $\mathcal{M}$. Pour tout $j < p(n)$, $0 \leq i \neq i' \leq p(n)$, $a \in \Sigma$, $q \in Q$ on a :

$$i^j \wedge q^j \wedge a_i^j \implies nq(q,a)^{j+1}, \qquad i^j \wedge q^j \wedge a_i^j \implies na(q,a)_i^{j+1},$$
$$i'^j \wedge a_i^j \implies a_i^{j+1}, \qquad i^j \wedge q^j \wedge a_i^j \implies (i + dp(q,a))^{j+1},$$

où on remplace $(i + dp(q,a))^{j+1}$ par **false** si $i + dp(q,a) \notin [0, p(n)]$.

Ces sous-formules garantissent que pour une interprétation $\nu \models \varphi_{\mathcal{M},w}$, pour tout $j < p(n)$, si $\mathbf{c}_\nu^j$ est définie alors $\mathbf{c}_\nu^{j+1}$ est définie et $\mathbf{c}_\nu^j \to_{\mathcal{M}} \mathbf{c}_\nu^{j+1}$.

• Par conséquent si $\nu \models \varphi_{\mathcal{M},w}$ alors $\mathbf{c}_\nu^0, \ldots, \mathbf{c}_\nu^{p(n)}$ sont toutes définies et correspondent à une exécution acceptant w.

• Réciproquement supposons que $\mathbf{c}^0, \ldots, \mathbf{c}^{p(n)}$ est une exécution acceptant w. Alors ν, définie pour tout j par : $\nu(\mathbf{s}^j) = \nu_{\mathbf{c}^j}(\mathbf{s}^j)$, satisfait φ.

La construction de $\varphi_{\mathcal{M},w}$ s'effectue en LOGSPACE. Par exemple pour construire les sous-formules,

$$(i^j \wedge q^j \wedge a_i^j) \implies nq(q,a)^{j+1}$$

pour tous $j < p(n)$, $i \leq p(n)$, $q \in Q$, $a \in \Sigma$, on utilise quatre boucles emboîtées où :

— les tailles de i et j appartiennent à $O(\log(p(n))) = O(\log(n))$;
— les tailles de q et a appartiennent à $O(1)$.

c.q.f.d. ◊◊◊

1. Notons que des clauses garantissant que $\mathbf{s}^j$ code une configuration, p.ex., "$\bigvee_{q \in Q} q^j$" qui exige qu'au moins une des propositions correspondant aux états de contrôle est valide à l'étape j, ne sont pas des clauses de Horn. C'est pour cette raison que nous ne les utilisons pas ici.

4.2 3HORNSAT

Soit φ une formule HNF dont les clauses ont au plus trois littéraux. Le *problème* 3HORNSAT est le problème de satisfaisabilité de φ.

Proposition 17 *3HORNSAT est* PTIME-*difficile (donc* PTIME-*complet).*

Preuve
Nous procédons par réduction du problème HORNSAT. Soit φ une formule HNF. La réduction construit φ' comme suit. Les clauses de φ d'au plus trois littéraux sont conservées dans φ'. Pour toute clause ψ de φ de plus de trois littéraux on construit les clauses suivantes :

— Si $\psi = (p_1 \wedge \cdots \wedge p_k) \Rightarrow q$ alors
φ' inclut les clauses $(p_1 \wedge p_2) \Rightarrow c_2$, $(c_2 \wedge p_3) \Rightarrow c_3$, ... , $(c_{k-1} \wedge p_k) \Rightarrow q$;

— Si $\psi = (p_1 \wedge \cdots \wedge p_k) \Rightarrow$ **false** alors
φ' inclut les clauses $(p_1 \wedge p_2) \Rightarrow c_2$, $(c_2 \wedge p_3) \Rightarrow c_3$, ... , $(c_{k-1} \wedge p_k) \Rightarrow$ **false**.

où $c_2, \ldots, c_{k-1}$ sont de nouvelles propositions (différentes pour chaque clause transformée).

Montrons la correction de cette construction. Soit $\psi = (p_1 \wedge \cdots \wedge p_k) \Rightarrow q$.

• Supposons que $\nu(\varphi) =$ **true**. Nous étendons ν en ν' comme suit.
Si $\nu(H_\psi) =$ **true** alors $\nu'(c_2) = \cdots = \nu'(c_k) =$ **true**.
Sinon soit i le premier i tel que $\nu(p_i) =$ **false**
Si $i \leq 2$ alors $\nu'(c_2) = \cdots = \nu'(c_k) =$ **false**
Sinon pour $j < i$ alors $\nu(c_j) =$ **true** et pour $j \geq i$, $\nu(c_j) =$ **false**

• Supposons que $\nu'(\varphi') =$ **true**.
Si pour un certain i, $\nu'(p_i) =$ **false** alors $\nu'(\psi) =$ **true**.
Sinon par induction, pour tout i, $\nu'(c_i) =$ **true** impliquant $\nu'(q) =$ **true**.

Le cas où la conclusion est **false** se traite de manière similaire.

c.q.f.d. ◊◊◊

4.3 Clôture d'une loi binaire

Nous introduisons un problème auxiliaire bien utile pour établir les bornes inférieures de complexité. Soit G un ensemble muni d'une loi binaire $\bullet$, soit H un sous-ensemble de G, la clôture de H notée $Cl(H)$ est défini comme le plus petit sous-ensemble contenant H et clos pour la loi binaire. Le *problème de la clôture* est défini par un élément $g \in G$ et un sous-ensemble $H \subseteq G$ et consiste à déterminer si $g \in Cl(H)$.

Proposition 18 *Le problème de la clôture appartient à* PTIME.

Preuve
L'algorithme 6 consiste à calculer par saturation $Cl(H)$ avec arrêt anticipé si $g \in Cl(H)$. Après initialisation de CH à H, on itère le procédé suivant : on parcourt la table de la loi $\bullet$ et pour chaque couple (g', g'') tel que $g' \bullet g'' \notin CH$ et $g', g'' \in CH$, on ajoute $g' \bullet g''$ à CH. Il y a au plus $|G|$ itérations. Chaque itération est en $O(|G|^2)$. L'algorithme opère donc en temps polynomial.

c.q.f.d. ◊◊◊

Algorithme 6 : Un algorithme pour la clôture

`Clôture`(G, H, g) : un booléen
Input : G, un ensemble muni d'une loi binaire $\bullet$,$H \subseteq G$, $g \in G$
Output : $g \in Cl(H)$?
Data : $CH \subseteq H$, $g', g'' \in G$, *done* un booléen

$CH \leftarrow H$
repeat
  `/* Invariant :` $H \subseteq CH \subseteq Cl(H)$ `*/`
  if $g \in CH$ **then return true**

  done $\leftarrow$ **true**
  for $g', g'' \in CH$ **do**
    if $g' \bullet g'' \notin CH$ **then** $CH \leftarrow CH \cup \{g' \bullet g''\}$; *done* $\leftarrow$ **false**

  end
  `/* Invariant :` $H \subseteq CH \subseteq Cl(H)$ `et` $g \notin G$ `*/`
  `/* Invariant :` *done* $\Leftrightarrow CH$ `est clos */`
until *done*
`/* Assertion :` $CH = Cl(H)$ `et` $g \notin G$ `*/`
return false

Proposition 19 *Le problème de la clôture même dans le cas d'une loi commutative est* PTIME*-difficile.*

Preuve
Nous exhibons une réduction du problème 3HORNSAT. Soit φ une formule sous forme 3HNF.

Soit $\psi = \bigvee_{i \in I} \ell_i$ une clause de φ. Les *composants* de ψ sont les ensembles de littéraux $\{\ell_i\}_{i \in J}$ tels que $J \subseteq I$. Les éléments de G sont les composants des clauses de φ et un élément spécial \$. Observons qu'il y a au plus 8 composants par clause.

La loi (commutative) $\bullet$ est définie ainsi :

— Si $u = \{p\}$ et $v = \{\neg p\} \cup C$ alors $u \bullet v = C$;
(en particulier si $v = \{\neg p\}$ *alors* $C = \emptyset$*)*

— Dans les autres cas, $u \bullet v =$ \$.

$g = \emptyset$ et $H = \{\{\ell_i\}_{i \in I} \mid \bigvee_{i \in I} \ell_i$ est une clause de $\varphi\}$.

• Supposons qu'il existe ν telle que $\nu(\varphi) =$ **true**.
Par induction, on établit que pour tout $\{\ell_i\}_{i \in J} \in Cl(H)$, $\nu(\bigvee_{i \in J} \ell_i) =$ **true**. Par conséquent, l'ensemble vide ne peut appartenir à $Cl(H)$.

• Supposons que l'ensemble vide n'appartient pas à $Cl(H)$. Définissons $\nu(p) =$ **true** ssi $\{p\} \in Cl(H)$.
◦ Soit $\psi = \neg p_1 \vee \neg p_2 \vee \neg p_3$ une clause de φ avec $\nu(\psi) =$ **false**. Alors $\{p_1\}$, $\{p_2\}$ et $\{p_3\}$ appartiennent à $Cl(H)$. Puisque $\{p_1\} \bullet (\{p_2\} \bullet (\{p_3\} \bullet \{\neg p_1, \neg p_2, \neg p_3\}))$ est l'ensemble vide, ceci conduit à une contradiction.
◦ Soit $\psi = \neg p_1 \vee \neg p_2 \vee q$ une clause de φ avec $\nu(\psi) =$ **false**. Alors $\{p_1\}$ et $\{p_2\}$ appartiennent à $Cl(H)$ et $\{q\}$ n'appartient pas à $Cl(H)$. Puisque $\{p_1\} \bullet$

$(\{p_2\} \bullet \{\neg p_1, \neg p_2, q\}) = \{q\}$. Par conséquent $\{q\} \in Cl(H)$ conduisant à une contradiction.

La réduction se fait en espace logarithmique car il suffit de maintenir l'identité de deux éléments de G pour calculer la loi.

c.q.f.d. ◊◊◊

Nous laissons au lecteur le soin de comprendre pourquoi cette réduction n'est pas valide pour une formule CNF.

4.4 Accessibilité dans un graphe *non déterministe*

Un graphe non déterministe $G = (V, A)$ est défini par un ensemble de sommets V et de triplets (arcs non déterministes) $A \subseteq V^3$ avec l'interprétation que si $(u, v, w) \in A$ alors partant de u si on choisit cet arc alors on peut être conduit soit en v soit en w. La définition suivante formalise cette interprétation.

Définition 4 *Soit $G = (V, A)$ un graphe non déterministe et W un sous-ensemble de sommets. Alors $Acc(W)$ est défini comme le plus petit sous-ensemble $Z \supseteq W$ tel que pour tout $(u, v, w) \in A$ si $\{v, w\} \in Z$ alors $u \in Z$. On dit que W est* accessible *depuis $u \in V$ si $u \in Acc(W)$.*

Le problème de *l'accessibilité dans un graphe non déterministe* est donné par une graphe non déterministe $G = (V, A)$, un sous-ensemble de sommets W, un sommet s et consiste à déterminer si $s \in Acc(W)$.

Proposition 20 *Le problème de l'accessibilité dans un graphe non déterministe appartient à* PTIME.

Preuve
L'algorithme 7 consiste à calculer par saturation $Acc(W)$. Après initialisation de $Acc(W)$ à W, on itère le procédé suivant : on parcourt les arcs de A et pour chaque arc (u, v, w) tel que $u \notin Acc(W)$ et $v, w \in Acc(W)$, on ajoute u à $Acc(W)$. Il y a au plus $|V|$ itérations. Chaque itération est en $O(|A|)$. L'algorithme opère donc en temps polynomial.

c.q.f.d. ◊◊◊

Proposition 21 *Le problème de l'accessibilité dans un graphe non déterministe est* PTIME-*difficile.*

Preuve
Le problème de clôture est un cas particulier du problème de l'accessibilité en posant $(u, v, w) \in A$ ssi $u = v \bullet w$.

c.q.f.d. ◊◊◊

Algorithme 7 : Un algorithme pour l'accessibilité

`AccNonDet`(G, W, s) : un booléen;
Input : $G = (V, A)$, $W \subseteq G$, $s \in V$
Output : s accède-t-il à W ?
Data : $Z \subseteq V$, $u, v, w \in V$, *done* un booléen

$Z \leftarrow W$;
repeat
  if $s \in Z$ **then return true**;
  ;
  done $\leftarrow$ **true**;
  for $(u, v, w) \in A$ **do**
    if $u \notin Z \wedge \{v, w\} \subseteq Z$ **then** $Z \leftarrow Z \cup \{u\}$; *done* $\leftarrow$ **false**;
    ;
  end
until *done*;
return false;

4.5 Test de la vacuité d'un langage algébrique

Une grammaire algébrique G est définie par un alphabet terminal Σ, un alphabet de symboles (non terminaux) Γ, un axiome (l'un des symboles non terminaux) S et des règles de productions R de la forme $T \to w$ avec $T \in \Gamma$ et $w \in (\Sigma \cup \Gamma)^*$. Les langages associés $\{L(G,T)\}_{T\in\Gamma}$ sont simultanément définis inductivement par :

— Si $T \to w$ est une règle avec $w \in \Sigma^*$ alors $w \in L(G,T)$;
— Si $T \to w_0 T_1 w_1 \dots T_n w_n$ est une règle avec pour tout $w_i \in \Sigma^*$ et si $u_i \in L(G,T_i)$ alors $w_0 u_1 \dots u_n w_n \in L(G,T)$.

Le langage de la grammaire $L(G)$ est alors $L(G,S)$.

Exemple. Nous illustrons cette notion par un grammaire pour décrire des expressions arithmétiques. Les alphabets sont $\Sigma = \{(,),+,\cdot,0,\dots,9\}$ et $\Gamma = \{E,T,F\}$ (E pour expression, T pour terme et F pour facteur) avec E pour axiome.
Une expression est un somme de termes : $E \to T \mid E+T$ (avec $\mid$ une abréviation qui évite de répéter la symbole gauche d'un règle).
Un terme est un produit de facteurs : $T \to F \mid T \cdot F$.
Un facteur est un chiffre ou un expression en parenthèses :
$F \to 0 \mid 1 \mid 2 \mid \dots \mid 9 \mid (E)$
Ainsi pour obtenir $(3+5)\cdot 2$, on applique la suite de règles suivantes où nous avons souligné le symbole substitué par la règle :
$\underline{E} \to \underline{T} \to \underline{T}\cdot F \to F \cdot \underline{F} \to \underline{F}\cdot 2 \to (\underline{E})\cdot 2 \to (T+\underline{E})\cdot 2 \to (T+\underline{T})\cdot 2$
$\to (T+\underline{F})\cdot 2 \to (\underline{T}+5)\cdot 2 \to (\underline{F}+5)\cdot 2 \to (3+5)\cdot 2$

Proposition 22 *Le problème de la vacuité d'un langage algébrique est en* PTIME.

Preuve
L'algorithme 8 consiste à calculer par saturation l'ensemble $Prod(G)$ des symboles non terminaux tels que $L(G,T) \neq \emptyset$ et à vérifier au fur et à mesure que $S \in Prod(G)$. On initialise $Prod(G)$ à l'ensemble vide. Puis on itère le procédé

suivant : on parcourt les règles de G et pour chaque règle $T \to w_0 T_1 w_1 \dots T_n w_n$ avec pour tout $i, T_i \in Prod(G)$, $w_i \in \Sigma^*$ et $T \notin Prod(G)$ on ajoute T à $Prod(G)$. Il y a au plus $|\Gamma|$ itérations. Chaque itération est en $O(|G|)$. L'algorithme opère donc en temps polynomial.

c.q.f.d. ◊◊◊

Algorithme 8 : Un algorithme pour la vacuité des langages algébriques

`VacAlg`(G) : un booléen;
Input : $G = (\Sigma, \Gamma, R, S)$ une grammaire algébrique
Output : $L(G)$ est-il vide ?
Data : $Prod \subseteq \Gamma$, *done* un booléen

$Prod \leftarrow \emptyset$;
repeat
  if $S \in Prod$ **then return true**;
  ;
  done $\leftarrow$ **true**;
  for $T \to w_0 T_1 w_1 \dots T_n w_n \in R$ **do**
    if $T \notin Prod \wedge \{T_1, \dots, T_n\} \subseteq Prod$ **then** $Prod \leftarrow Prod \cup \{T\}$;
      done $\leftarrow$ **false**;
    ;
  end
until *done*;
return false;

Proposition 23 *Le problème de la vacuité d'un langage algébrique est* PTIME-*difficile.*

Preuve
On réduit le problème de clôture au problème de la vacuité d'un langage algébrique. Étant donné $((G, \bullet), g, H)$ un problème de clôture, la grammaire algébrique est définie ainsi :

— L'alphabet non terminal est l'ensemble G. L'axiome est g. L'alphabet terminal est sans importance et peut être vide.
— Les règles de production sont définies par :
$\{u \to vw \mid u = v \bullet w\} \cup \{u \to \varepsilon \mid u \in H\}$.

Cette réduction se fait en espace logarithmique. La correction de la réduction est immédiate.

c.q.f.d. ◊◊◊

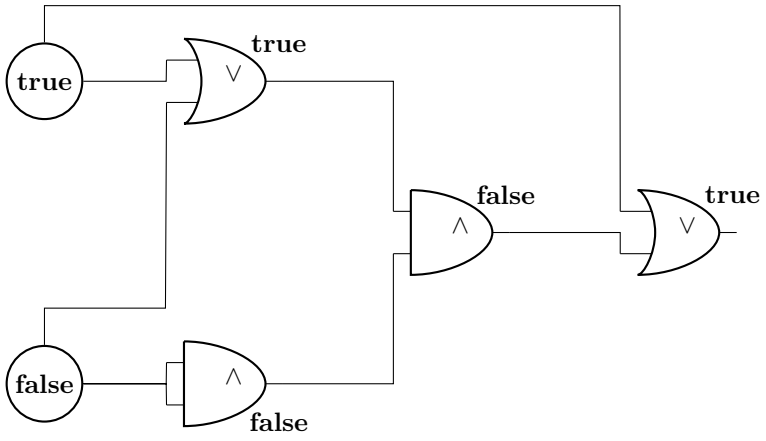
4.6 Valeur d'un circuit

Un circuit $\mathcal{C}$ est un graphe acyclique dont les sommets sont appelés des *portes*. Chaque porte a un *identifiant*, un *type* parmi **false**, **true**, $\wedge$, $\vee$. Les prédécesseurs immédiats d'une porte sont appelés ses *entrées*. La porte 0 (resp. 1) est l'unique porte de type **false** (resp. **true**). Ces deux portes n'ont pas d'entrées. Les portes

de type $\wedge$ et $\vee$ ont au moins une entrée. Les identifiants sont définis en accord avec l'un des tris topologiques du graphe.

Chaque porte a une *valeur* définie inductivement comme suit :
— $Val(0) = \mathbf{false}$, $Val(1) = \mathbf{true}$; ;
— Soit g une porte $\vee$ dont les entrées sont $\{g_i\}_{i \leq k}$.
Si tous les $Val(g_i)$ sont définies alors $Val(g) = \bigvee_{i \leq k} Val(g_i)$;
— Soit g une porte $\wedge$ dont les entrées sont $\{g_i\}_{i \leq k}$.
Si tous les $Val(g_i)$ sont définies alors $Val(g) = \bigwedge_{i \leq k} Val(g_i)$.

Nous avons illustré la syntaxe et la sémantique d'un circuit par l'exemple ci-dessous.



Etant donné un circuit $\mathcal{C}$ et une porte distinguée **out**, le problème de la *valeur d'un circuit* consiste à déterminer la valeur de **out**. Nous supposons (sans perte de généralité) que les spécifications des portes suivent l'ordre des identifiants.

Proposition 24 *Le problème de la valeur d'un circuit appartient à* PTIME.

Preuve
L'algorithme 9 met en oeuvre la définition des valeurs des portes d'un circuit.

c.q.f.d. ◊◊◊

Observons qu'on pourrait étendre le type des portes par : $\neg$, $\Rightarrow$, etc. sans changer la complexité de l'algorithme.

Proposition 25 *Le problème de la valeur d'un circuit est* PTIME*-difficile.*

Preuve
Nous procédons par réduction du problème de la clôture. Soit $(G, \bullet), H, g$ un tel problème. Nous définissons inductivement une suite croissante de sous-ensembles $Cl_i(H)$ pour i de 0 à n ainsi :
— $Cl_0(H) = H$;
— $Cl_{i+1}(H) = \{x \mid \exists y, z \in Cl_i(H)\ x = y \bullet z\} \cup Cl_i(H)$.
Si $Cl_{i+1}(H) = Cl_i(H)$ alors pour tout $j > i$, $Cl_j(H) = Cl_i(H)$. On conclut donc que $Cl_n(H) = Cl(H)$ où $n = |G|$.

La réduction consiste à construire à un circuit dont les portes sont les suivantes :

Algorithme 9 : Calcul de la valeur d'un circuit

`ValCirc`$(\mathcal{C}, \textbf{out})$: un booléen;
Input : $\mathcal{C}$ un circuit, **out** une porte de $\mathcal{C}$
Output : la valeur de **out**
Data : g une porte, Val un tableau de booléens indicé par les portes de $\mathcal{C}$

```
for g ∈ C do
    switch tp(g) do
        case false : do Val[g] ← false;
        ;
        case true : do Val[g] ← true;
        ;
        case ∨ : do Val[g] ← ⋁_{h∈In(g)} Val[h];
        ;
        case ∧ : do Val[g] ← ⋀_{h∈In(g)} Val[h];
        ;
    end
end
return Val[out]
```

— Pour tout $0 \leq i \leq n$ et tout $x \in G$, des portes (i, x) qui ont la valeur **true** ssi $x \in Cl_i(H)$;
— Pour tout $1 \leq i \leq n$ et tout $y, z \in G$ des portes (i, y, z) qui ont la valeur **true** ssi $\{y, z\} \subseteq Cl_{i-1}(H)$.

La porte **out** sera donc (n, g).

En utilisant la définition inductive, on déduit que :

— Les portes (i, x) sont des portes $\vee$;
— Pour tout x, $(0, x)$ a une entrée : la porte **true** (resp. **false**) si $x \in H$ (resp. $x \notin H$) ;
— Pour tout x et $i \geq 1$, la porte (i, x) a pour entrées : la porte $(i-1, x)$ et les portes (i, y, z) telles que $x = y \bullet z$;
— Pour tout y, z les portes (i, y, z) sont des portes $\wedge$ et ont deux entrées : les portes $(i-1, y)$ et $(i-1, z)$.

L'algorithme 10 qui opère en espace logarithmique effectue cette réduction.

c.q.f.d. ◊◊◊

Algorithme 10 : De la clôture à la valeur d'un circuit

```
RedCirc((G, •), H, g) : un problème de valeur de circuit;
Input : (G, •), un semble muni d'une loi binaire, H ⊆ G, h ∈ G
Output : un circuit avec une porte distinguée
Data : x, y, z ∈ G, i ∈ {1, ..., n}
Write(id : 0) ; Write(tp : false) ; Write(id : 1) ; Write(tp : true);
for x ∈ G do
    Write(id : (0, x)) ; Write(tp : ∨);
    if x ∈ H then Write(in : 1);
    else Write(in : 0);
    ;
    for i from 1 to n do
        for y, z ∈ G do
            Write(id : (i, y, z)) ; Write(tp : ∧);
            Write(in : (i − 1, y)) ; Write(in : (i − 1, z));
        end
        for x ∈ G do
            Write(id : (i, x)) ; Write(tp : ∨) ; Write(in : (i − 1, x));
            for y, z ∈ G do
                if x = y • z then Write(in : (i, y, z));
            end
        end
    end
end
Write(out : (n, g))
```

4.7 TD 5

La logique CTL (Computational Tree Logic) permet d'exprimer des propriétés portant sur les évolutions possibles d'un système de transitions à partir d'un état. Un ensemble AP de propositions atomiques est donné et permet de distinguer les propriétés des différents états.

Une structure de Kripke sur AP est un système de transitions dont les états sont étiquetés par les propositions atomiques qu'ils vérifient. Une structure de Kripke est définie formellement par un triplet $M = (S, \to, L)$, où :

— S est un ensemble fini d'états,
— $\to \subseteq S \times S$ est une fonction de transition telle que $\forall s \in S \;\; \exists s' \in S \;\; s \to s'$,
— $L : S \longrightarrow 2^{AP}$ définit quelles propriétés atomiques sont vraies dans chaque état.

Une *exécution* de M est une suite infinie d'états $(s_0, s_1, \dots)$ telle que pour tout $i \geq 0$, $s_i \to s_{i+1}$.

Les formules de la logique CTL sont définies par la grammaire suivante :

$$\begin{aligned} \phi ::= \;& \top \mid \bot \mid p \qquad (\text{avec } p \in AP) \\ & \mid \neg\phi \mid \phi \wedge \phi \mid \phi \vee \phi \\ & \mid AX\phi \mid EX\phi \mid AF\phi \mid EF\phi \mid AG\phi \mid EG\phi \mid A[\phi U \phi] \mid E[\phi U \phi] \end{aligned}$$

Une formule CTL s'interprète sur un état s d'une structure de Kripke M selon la définition suivante (par induction sur la taille de la formule) :

— $M, s \models \top$
— $M, s \not\models \bot$
— $M, s \models p$ with $p \in AP$ ssi $p \in L(s)$
— $M, s \models \neg\phi$ ssi $M, s \not\models \phi$
— $M, s \models \phi \wedge \psi$ ssi $M, s \models \phi$ et $M, s \models \psi$
— $M, s \models \phi \vee \psi$ ssi $M, s \models \phi$ ou $M, s \models \psi$
— $M, s \models AX\phi$ ssi $\forall s' \in S \quad s \to s' \implies M, s' \models \phi$
— $M, s \models EX\phi$ ssi $\exists s' \in S \quad s \to s' \wedge M, s' \models \phi$
— $M, s \models AF\phi$ ssi toute exécution commençant à l'état s passe par au moins un état s' tel que $M, s' \models \phi$
— $M, s \models EF\phi$ ssi il existe une exécution commençant à l'état s qui passe par au moins un état s' tel que $M, s' \models \phi$
— $M, s \models AG\phi$ ssi toute exécution commençant à l'état s ne passe que par des états s' tel que $M, s' \models \phi$
— $M, s \models EG\phi$ ssi il existe une exécution commençant à l'état s qui ne passe que par des états s' tel que $M, s' \models \phi$
— $M, s \models A[\phi U \psi]$ ssi pour toute exécution $s_0 \to s_1 \to \dots$ commençant à l'état $s_0 = s$, il existe un indice n tel que $M, s_n \models \psi$ et pour tout $i < n$, $M, s_i \models \phi$
— $M, s \models E[\phi U \psi]$ ssi il existe une exécution $s_0 \to s_1 \to \dots$ commençant à l'état $s_0 = s$ et un indice n tel que $M, s_n \models \psi$ et pour tout $i < n$, $M, s_i \models \phi$

Dans le problème de model checking CTL, un modèle M et un état initial s sont donnés, ainsi qu'une formule CTL ϕ. La question est de savoir si $M, s \models \phi$.

Question 1. En partant du problème de l'évaluation d'un circuit booléen, montrer que le problème de model checking CTL est PTIME-difficile.

Question 2. Montrer que l'on peut se restreindre aux opérateurs $\neg$, $\wedge$, EX, EG et $E[U]$ sans restreindre l'expressivité de la logique CTL.

Question 3. Proposer un algorithme récursif qui décide si une formule est satisfaite par un état d'un modèle. Améliorer éventuellement votre algorithme pour obtenir une complexité en $O(|M| \times |\phi|)$.

Model checking sous condition d'équité

Selon le système qu'elle modélise, il arrive qu'une structure de Kripke génère des exécutions qui ne sont pas pertinentes pour le problème étudié car elles sont trop irréalistes, peu probables...

On propose de sélectionner les exécutions pertinentes en donnant un ensemble $\{F_1, \ldots, F_n\}$ d'ensembles d'états et en définissant les exécutions équitables comme celles qui passent infiniment souvent par un état de chaque ensemble F_i.

On peut alors introduire des versions équitables E_f et A_f des quantificateurs E et A, qui s'interprètent en ne quantifiant que sur les exécutions équitables.

Question 4. Proposer un algorithme qui calcule l'ensemble des états de M à partir desquels il existe une exécution équitable. On appelle ces états des états équitables.

Question 5. En utilisant une proposition atomique fair satisfaite par tous les états équitables, montrer comment ramener le model checking sous condition d'équité au model checking sans équité.

Chapitre 5

La classe NLOGSPACE

5.1 Accessibilité dans un graphe

On se donne un graphe orienté $G = (V, E)$ et deux sommets s et t, le problème de *l'accessibilité* consiste à décider s'il existe un chemin de s à t.

Proposition 26 *Le problème de l'accessibilité appartient à* NLOGSPACE.

Preuve
S'il existe un chemin de s à t alors il en existe un qui comporte au plus n sommets avec $n = |V|$ (en éliminant les circuits).

L'algorithme 11 (non déterministe) maintient deux variables : un compteur initialisé à n et un sommet courant initialisé à 0. À chaque étape, il choisit de manière non déterministe un successeur au sommet courant et incrémente le compteur. Si un sommet courant est t alors il renvoie **true**. Si le sommet courant n'a pas de successeur ou si le compteur atteint n il renvoie **false**.

Toutes les variables sont représentables en $O(log(n))$ bits ce qui achève la démonstration.

c.q.f.d. ◊◊◊

Proposition 27 *Le problème de l'accessibilité est* NLOGSPACE*-difficile.*

Preuve
Soit $\mathcal{M}$ une machine de Turing non déterministe opérant en espace $\log(n)$ sur un mot w de longueur n. Sans perte de généralité, on fait l'hypothèse que lorsque la machine s'arrête et décide, les têtes (des bandes d'entrée et de travail) sont au-dessus du premier caractère et la bande de travail ne contient que des blancs.

La réduction consiste à construire le graphe des configurations :

- Une configuration est donnée par la position des têtes de lecture, l'état de la machine et le contenu de la bande. Il suffit donc de $O(\log(n))$ bits pour coder une configuration (où la constante ne dépend que de $\mathcal{M}$).
- Il y a un arc d'une configuration $\mathbf{c}$ à une autre configuration $\mathbf{c}'$, si en partant de $\mathbf{c}$ il existe un pas de $\mathcal{M}$ qui conduit à $\mathbf{c}'$.

Algorithme 11 : Un algorithme non déterministe pour l'accessibilité

`Reach`(G, s, t) : un booléen;
Input : $G = (V, E)$ un graphe orienté, $s, t \in V$
Output : Existe-t-il un chemin de s à t ?
Data : $cpt \in \{0, \ldots, n\}$, $u, v \in V$

$cpt \leftarrow 0$; $u \leftarrow s$;
repeat
  if $u = t$ **then return true**;
  ;
  if $\nexists (u, v) \in E$ **then return false**;
  ;
  Guess $(u, v) \in E$; $u \leftarrow v$; $cpt \leftarrow cpt + 1$;
until $cpt = n$;
return false

Le sommet source est la configuration initiale et le sommet destination est la configuration finale (il y en a une seule d'après nos hypothèses).

La réduction construit les arcs du graphe en examinant pour chaque configuration ses successeurs par la machine. Elle maintient une configuration courante et calcule successivement chacun de ses successeurs. Elle opère donc en espace logarithmique.

c.q.f.d. ◊◊◊

Le résultat précédent peut aussi se reformuler dans le contexte de la théorie des langages : le langage d'un automate non déterministe est-il vide ? La table ci-dessous résume les résultats de complexité que nous avons obtenus en théorie des langages. Le résultat d'indécidabilité sera établi en cours de langages formels.

	Vacuité	Universalité
Automate non déterministe	NLOGSPACE-complet	PSPACE-complet
Grammaire algébrique	PTIME-complet	Indécidable

5.2 Le théorème d'Immerman-Szelepscényi

Dans le lemme suivant, la sémantique d'une machine de *calcul non déterministe* est la suivante : lorsqu'elle accepte, le résultat est sur sa bande de sortie et le résultat doit être le même quelque soit la configuration acceptante. De plus il existe au moins une configuration acceptante.

Exemple de calcul non déterministe. Supposons que l'entrée soit un mot w tel qu'une lettre inconnue a plus de $\frac{|w|}{2}$ occurrences. L'algorithme 12 non déterministe calcule les occurrences de cette lettre.

Algorithme 12 : Un calcul non déterministe

`Reach`(w) : un entier;
Input : w un mot de longueur n avec une lettre majoritaire
Output : le nombre d'occurrences de la lettre majoritaire
Data : $i, cpt \in \{0, \ldots, n\}$, $a \in \Sigma$

Guess $a \in \Sigma$; $cpt \leftarrow 0$;
for i **from** 1 **to** n **do**
  if $w[i] = a$ **then** $cpt \leftarrow cpt + 1$;
  ;
end
if $cpt > \frac{|w|}{2}$ **then return** cpt **else return reject**;
;
;

Lemme 2 *Soit $\mathcal{M}$ une machine de Turing non déterministe s'exécutant en un espace de taille $s(n) \geq \log(n)$ constructible*[1] *où n est la taille de l'entrée. Alors il existe une machine de Turing non déterministe $\mathcal{M}'$ s'exécutant en un espace de taille $O(s(n))$, qui calcule N, le nombre de configurations atteignables depuis la configuration initiale.*

Preuve
Tout d'abord, la machine $\mathcal{M}'$ décrite par l'algorithme 13 calcule la taille d'une configuration ($s(n)$).

Notons N_d, le nombre de configurations différentes atteintes par la machine $\mathcal{M}$ depuis la configuration initiale en au plus d pas. $N_0 = 1$. La machine $\mathcal{M}'$ calcule itérativement N_d et le stocke dans la variable N en fin d'itération puis dans $oldN$ en début d'itération. Décrivons plus précisément une itération.

- Elle mémorise N dans $OldN$. Puis elle initialise N à 0. Elle énumère ensuite les configurations occupant une place $s(n)$.
- Pour chaque configuration, stockée dans la variable *current*, elle initialise le compteur *cpt* et énumère de nouveau les configurations occupant une place $s(n)$. Dans cette boucle interne, elle teste de manière non déterministe si la configuration courante de cette boucle stockée dans *local* est accessible en au plus d pas à partir de la configuration initiale *init* en devinant un chemin de longueur au plus d. Ce test non déterministe nécessite seulement une configuration et un compteur et se fait à l'aide d'une variante de l'algorithme 11 qui *ne construit pas le graphe de configurations mais teste l'existence d'un arc à l'aide de la description de $\mathcal{M}$*. Si c'est le cas, elle incrémente *cpt* puis elle teste si *current* est accessible en au plus un pas à partir de *local* et dans ce cas incrémente N et sort de la boucle interne. Si elle termine sa boucle interne (sans avoir incrémenté N). elle contrôle si le compteur *cpt* est égal à $OldN$. Si ce n'est pas le cas elle s'arrête et rejette car l'un des verdicts d'accessibilité de *local* était erroné.
- À la fin de l'énumération la plus externe N est égal à N_{d+1} puisque les

1. Cette hypothèse n'est nécessaire ni ici ni dans la suite de cette section mais elle simplifie la preuve.

seules erreurs possibles ont été détectées par le contrôle effectué *via cpt*. d est alors incrémenté.

La machine s'arrête lorsque $N = OldN$ (i.e. $N_{d+1} = N_d$). La machine occupe un espace en $O(s(n))$ puisqu'une configuration est représentée en $O(s(n))$ bits et par conséquent les valeurs des compteurs sont bornées par $2^{O(s(n))}$. L'hypothèse $s(n) \geq \log(n)$ est requise car la tête de lecture de la bande d'entrée occupe déjà $\lceil \log(n) \rceil$ bits.

c.q.f.d. ◊◊◊

Algorithme 13 : L'algorithme d'Immerman-Szelepscényi

```
Calcul du nombre de configurations de M sur w
d ← 0 ; N ← 1
repeat
    OldN ← N ; N ← 0
    /* oldN est le ♯ de configurations accessibles en au plus d pas.   */
    for current ∈ Conf_{M,w} do
        /* teste si current est accessible en au plus d+1 pas.         */
        acc ← false ; cpt ← 0
        /* cpt contrôle la validité des choix non déterministes.       */
        for local ∈ Conf_{M,w} do
            Deviner un chemin σ de init à local en au plus d pas
            /* Ceci se fait avec une variante de l'algorithme 11       */
            /* qui ne construit pas le graphe des configurations.      */
            if σ existe then
                cpt ← cpt + 1
                if local = current or local →_M current then
                    /* if current is accepting then reject
                                                                        */
                    N ← N + 1 ; acc ← true ; break
                    /* current est accessible en au plus d+1 pas.      */
                end
            end
        end
        if not acc and cpt < OldN then reject

        /* Un local a été déclaré à tort inaccessible en au plus d pas. */
    end
    d ← d + 1
until N = OldN
return N/* accept                                                       */
```

Théorème 6 (Immerman-Szelepscényi) *Soit $\mathcal{M}$ une machine de Turing non déterministe s'exécutant en un espace de taille $s(n) \geq \log(n)$ constructible où n est la taille de l'entrée. Alors il existe une machine de Turing non déterministe $\mathcal{M}'$ s'exécutant en un espace de taille $O(s(n))$, qui accepte le langage complémentaire de celui accepté par $\mathcal{M}$.*

Preuve
La machine $\mathcal{M}'$ est presque identique celle du lemme 2. La seule différence réside dans le fait que lorsqu'elle trouve une configuration acceptante de $\mathcal{M}$, elle s'arrête et rejette. Les changements sont indiqués dans des commentaires rouges dans l'algorithme 13. Par conséquent, si elle se termine en acceptant cela signifie qu'à son dernier calcul de N_d, elle a rencontré toutes les configurations accessibles de $\mathcal{M}$ et qu'aucune n'est acceptante.

c.q.f.d. ◊◊◊

Le corollaire le plus important est relatif à NLOGSPACE.

Corollaire 3 coNLOGSPACE = NLOGSPACE

Chapitre 6

Inclusions strictes entre classes

6.1 Machines de Turing universelles

Théorème 7 *Il existe une machine de Turing déterministe $\mathcal{U}$ qui prend en entrée la représentation d'une machine de Turing déterministe $\mathcal{M}$ et un mot x de l'alphabet de $\mathcal{M}$ t.q. si T est le temps de calcul de $\mathcal{M}$ sur x alors $\mathcal{U}$ produit le même résultat en temps $O(T \log(T))$ (avec une constante qui dépend de $\mathcal{M}$ mais pas de x).*

Preuve
Dans un premier temps, nous construisons une machine $\mathcal{U}$ à bandes bidirectionnelles. $\mathcal{U}$ a une bande d'entrée, une bande de sortie et deux bandes de travail, appelées bande de simulation et bande d'état. La bande d'état contient une représentation de l'état courant de $\mathcal{M}$ et la position de la tête de lecture de $\mathcal{M}$ sur sa bande d'entrée et sert aussi de bande de stockage temporaire pour les déplacements de blocs de caractère sur la bande de simulation (voir plus bas). La bande de simulation, la seule à être utilisée de façon bidirectionnelle contient le contenu « juxtaposé » des bandes de travail. Le codage traditionnel de plusieurs bandes par une unique bande à alphabet fixé se fait en deux étapes. Tout

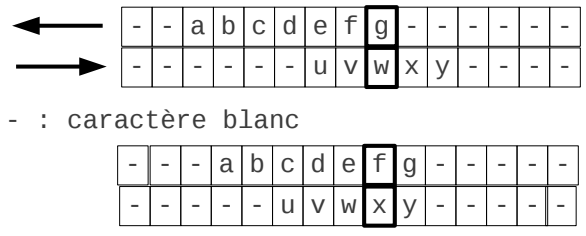


FIGURE 6.1: Représentation compacte et simulation de plusieurs bandes

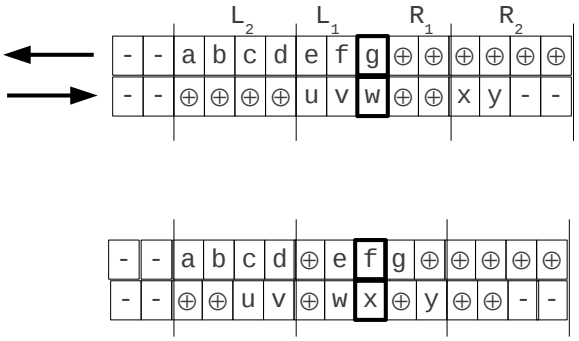


FIGURE 6.2: Représentation expansée et simulation de plusieurs bandes

d'abord si l'alphabet de $\mathcal{M}$ est Σ (avec $|\Sigma| \geq 2$) et si k est le nombre de bandes de travail de $\mathcal{M}$, on considère l'alphabet Σ^k. Chaque lettre de cet alphabet est alors associé à une représentation binaire de longueur $m = \lfloor \log_2(|\Sigma^k| - 1) \rfloor + 1$ et chaque groupe de k cellules juxtaposées est codé par un bloc de m cellules.

A l'aide d'une bande bidirectionnelle, on peut s'arranger pour déplacer le contenu des bandes simulées de telle sorte qu'au début d'un pas de simulation, la tête de lecture de la bande de simulation soit toujours à la position 0 (voir la figure 6.1). Cependant ce déplacement même limité aux caractères différents du blanc conduit à une simulation d'un pas en $O(T)$ (puisqu'il peut y avoir T caractères utiles) et par conséquent à une simulation en $O(T^2)$.

Afin de limiter les déplacements des bandes, on a recours à une représentation expansée des bandes à l'aide d'un caractère supplémentaire $\oplus$ différent du blanc. Dans cette représentation la bande est implicitement partitionnée en segments $\ldots, L_i, \ldots, L_1, [0,0], R_1, \ldots, R_i, \ldots$ t.q. $L_i = [-2^{i+1} + 2, -2^i + 1]$ et $R_i = [2^i - 1, 2^{i+1} - 2]$. Au début de tout pas de simulation, chaque (représentation de) bande vérifie les propriétés suivantes :

- Chaque segment (L_i, R_i) est soit *vide*, soit *plein* soit *à moitié-plein*. Il est vide s'il ne contient que des $\oplus$, plein s'il ne contient pas de $\oplus$, et à moitié plein si la moitié de ces caractères sont des $\oplus$.
- L_i est vide (resp. plein, à moitié plein) ssi R_i est plein (resp. vide, à moitié plein)
- Initialement, tous les segments sont à moitié-plein (il suffit de remplacer la moitié des blancs de L_i et de R_i par des $\oplus$ la première fois qu'on rencontre L_i ou R_i).
- La position 0 ne contient pas de $\oplus$.

Nous décrivons maintenant sur chaque représentation de bande l'effet d'un déplacement. Nous nous limitons à un déplacement à droite car l'autre cas est symétrique.

- $\mathcal{U}$ cherche le plus petit R_i non vide (et donc L_i n'est pas plein).
- $\mathcal{U}$ recopie le premier caractère différent de $\oplus$ de R_i à la position 0 et les $2^{i-1} - 1$ autres caractères suivants différents de $\oplus$ de R_i dans $R_1, R_2, \ldots, R_{i-1}$ en remplissant à moitié ces segments.

— À gauche de la tête de lecture, il y a l'ancien caractère de la position 0 et $2(2^i-1)$ caractères différents de 0 dans $L_{i-1}, \ldots, L_1$. $\mathcal{U}$ recopie ensuite les 2^i caractères les plus à gauche dans L_i et dispose les $2^{i-1}-1$ caractères restants de telle sorte que $L_{i-1}, \ldots, L_1$ soient à moitié pleins.

La représentation expansée et la simulation du déplacement sont illustrées par la figure 6.2 sur le même exemple que pour la représentation compacte.

Le point clef de la simulation est que lorsqu'un caractère de la bande L_i ou R_i est mis en position 0, alors ces deux segments ne peuvent être accédés qu'après au mois 2^{i-1} déplacements de la tête dans une direction car les segments $R_1, R_2, \ldots, R_{i-1}$ et $L_{i-1}, \ldots, L_1$ sont à moitié pleins. Nous avons illustré ce phénomène à la figure 6.3 pour $i = 3$.

Par conséquent si la machine $\mathcal{M}$ comporte b bandes de travail, alors les segments L_i et R_i sont déplacés au plus $\frac{bT}{2^{i-1}}$ fois. En utilisant un ruban auxiliaire pour mémoriser le contenu d'un segment, un déplacement de L_i ou de R_i se fait en temps $O(2^i)$. Par conséquent, le temps global de la simulation est :

$$O\left(\sum_{i=1}^{\log_2(T)+1} \frac{bT2^i}{2^{i-1}}\right) = O(T\log_2(T)) .$$

Notons qu'il faut à chaque étape parcourir la table de transition de la machine $\mathcal{M}$ pour trouver quelle règle simuler : ceci se fait en temps $O(1)$ avec une constante qui dépend seulement de $\mathcal{M}$.

La simulation d'une machine à bande bidirectionnelle par une machine à bande unidirectionnelle se fait en repliant la bande sur elle-même et le temps d'un pas de cette simulation est en $O(1)$.

Puisque seul le choix du pas de $\mathcal{M}$ est non déterministe, $\mathcal{U}$ est déterministe si elle se borne à simuler des machines déterministes.

c.q.f.d. ◊◊◊

Grâce au pouvoir de calcul du non déterminisme, on peut produire une machine universelle plus efficace.

Théorème 8 *Il existe une machine de Turing non déterministe $\mathcal{U}$ qui prend en entrée la représentation d'une machine de Turing non déterministe $\mathcal{M}$ et un mot x de l'alphabet de $\mathcal{M}$ telle que :*

— *Si $\mathcal{M}$ accepte x par un calcul en temps T alors $\mathcal{U}$ accepte $\mathcal{M}, x$ par un calcul en temps $O(T)$ (avec une constante qui dépend de $\mathcal{M}$ mais pas de x).*

— *Si $\mathcal{M}$ rejette x et tous les calculs se font en un temps au plus T' alors $\mathcal{U}$ n'accepte pas $\mathcal{M}, x$ et tous les calculs se font en temps $O(T')$ (avec une constante qui dépend de $\mathcal{M}$ mais pas de x).*

Preuve

Sans perte de généralité, on suppose que $\mathcal{M}$ dispose d'une transition qui ne fait rien et qui est possible uniquement dans l'état d'acceptation ou de rejet.

La machine $\mathcal{U}$ a quelques bandes de travail : une bande de prédiction, une bande auxiliaire, une bande de compteur et une bande de sortie simulée correspondant à la bande sortie de $\mathcal{M}$. $\mathcal{U}$ itère le processus suivant à l'aide d'un compteur

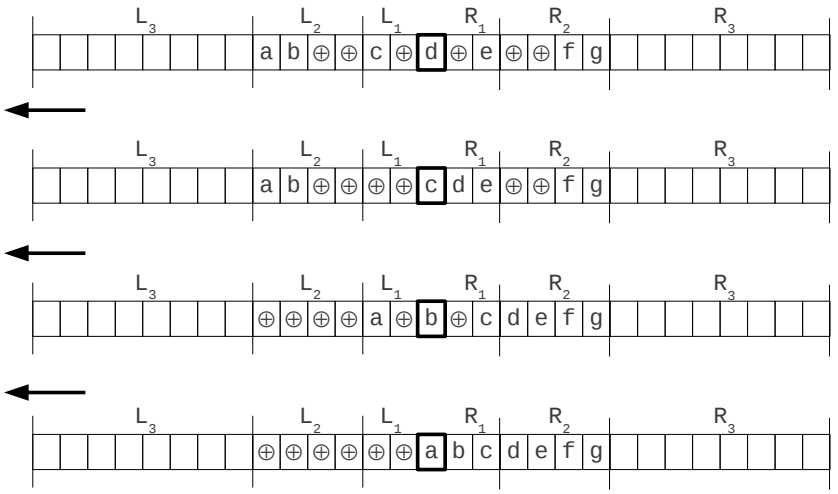


FIGURE 6.3: Déplacement simulé d'une bande « équilibrée »

initialisé à 1. Sur sa bande de prédiction, il construit (ou complète la suite déjà construite) de manière non déterministe une suite de transitions de $\mathcal{M}$ de longueur égale au compteur. Deux transitions successives de cette suite sont telles que l'état d'arrivée de la première transition est aussi l'état de départ de la deuxième transition. Dans le cas contraire, il vérifie une bande après l'autre que l'exécution prédite est réalisable sur chaque bande (à l'aide de sa bande auxiliaire). Si l'exécution n'est pas réalisable il rejette. Sinon il y a trois cas possibles :

- Le dernier état est l'état d'acceptation : $\mathcal{U}$ accepte.
- Le dernier état est l'état de rejet : $\mathcal{U}$ rejette.
- Le dernier état est un autre état : $\mathcal{U}$ double la valeur du compteur et passe à l'itération suivante.

La correction de la simulation ne présente pas de difficulté. Supposons que $\mathcal{M}$ accepte x en temps T. Il existe alors une simulation acceptante qui effectuera au plus $O(\log(T))$ tours. Le temps d'exécution de chaque tour est proportionnel à la valeur courante du compteur $1, 2, 4, \ldots$ qui ne peut excéder $2T$. D'où un temps cumulé en $O(\sum_k O(T/2^k)) = O(T)$. Le même raisonnement s'applique au cas du rejet.

Le point clef de cette simulation est la possibilité d'effectuer les simulations sur chaque bande de manière indépendante et donc d'éviter la recherche des têtes de lecture.

c.q.f.d. ◊◊◊

Théorème 9 *Il existe une machine de Turing (déterministe) $\mathcal{U}$ qui prend en entrée la représentation d'une machine de Turing (déterministe) $\mathcal{M}$ et un mot x de l'alphabet de $\mathcal{M}$ t.q. si E est l'espace nécessaire au calcul de $\mathcal{M}$ sur x alors*

$\mathcal{U}$ produit le même résultat en espace $O(\max(E, \log(|x|))$ (avec une constante qui dépend de $\mathcal{M}$ mais pas de x).

Preuve
Il suffit d'adopter la représentation compacte du théorème 7. Le facteur $\log_2(|x|)$ provient du stockage de la tête de lecture de la bande d'entrée de $\mathcal{M}$.

c.q.f.d. ◊◊◊

6.2 Hiérarchies de complexité

Le deuxième ingrédient dont nous avons besoin est une représentation des machines de Turing telle qu'une machine admette une infinité de représentations et plus précisément telle que pour toute machine $\mathcal{M}$, il existe n_0 vérifiant $\forall n \geq n_0$ il existe une représentation de $\mathcal{M}$ de taille n. Cette représentation est très facile à construire. Donnons-nous une représentation quelconque des machines de Turing disons $x_{\mathcal{M}}$, la représentation recherchée est de la forme $1^n 0 x_{\mathcal{M}}$ pour n quelconque.

Nous présentons les théorèmes de hiérarchie par ordre de difficulté croissante.

Théorème 10 *Soient $f(n) \geq \log(n)$ et $g(n) \geq \log(n)$ deux fonctions constructibles vérifiant :*

$$\liminf_{n\to\infty} \frac{f(n)}{g(n)} = 0$$

Alors il existe un langage L accepté par une machine de Turing opérant sur une entrée x en espace $g(|x|)$ mais par aucune machine de Turing opérant sur une entrée x en espace $f(|x|)$.

Preuve
Nous construisons une machine $\mathcal{U}'$ qui est une variante de la machine universelle du théorème 9. $\mathcal{U}'$ a une bande supplémentaire dite bande de compteur pour stocker un compteur. Soit x une entrée de taille n, $\mathcal{U}'$ commence par marquer ses bandes de travail avec un marqueur en position $g(n)$. Par la suite, si sa simulation (y compris dans la phase initiale) le conduit à dépasser son marqueur il s'arrête et rejette.

Si x n'est pas la représentation d'une machine de Turing (disons $\mathcal{M}$) alors $\mathcal{U}'$ rejette. Dans le cas contraire, il initialise son compteur à $n_q f(n)^{n_t} n_a^{n_t f(n)}$ avec n_q le nombre d'états de $\mathcal{M}$, n_t le nombre de bandes de $\mathcal{M}$ et n_a le nombre de lettres de $\mathcal{M}$. Observons que d'une part ce compteur représente un majorant strict du plus grand nombre de pas que peut faire une machine qui se termine en opérant en espace $f(n)$ et d'autre part que ce compteur occupe une place en $O(f(n))$ si on fait croître n en laissant la machine $\mathcal{M}$ fixe.

Puis $\mathcal{U}'$ entreprend la simulation de $\mathcal{M}$ sur x en décrémentant son compteur et en avortant sa simulation si son compteur s'annule et en rejetant. Lorsque la simulation se termine, alors $\mathcal{U}'$ rejette ssi $\mathcal{M}$ accepte.

Soit L le langage accepté par $\mathcal{U}'$. Par construction $\mathcal{U}'$ opère en espace $g(n)$. Supposons que L soit reconnu par une machine $\mathcal{M}$ qui opère en espace $f(n)$. D'après le théorème 9 la simulation de $\mathcal{M}$ requiert un espace $O(f(n))$. Sachant que $\liminf_{n\to\infty} \frac{f(n)}{g(n)} = 0$ et qu'on peut choisir n quelconque suffisamment grand

pour la taille d'une représentation x de $\mathcal{M}$, la simulation de $\mathcal{M}$ pour un tel x se poursuit jusqu'à son terme conduisant à un résultat différent pour $\mathcal{M}$ et $\mathcal{U}'$ d'où la contradiction.

c.q.f.d. ◊◊◊

On a donc NLOGSPACE $\subsetneq$ PSPACE car NLOGSPACE $\subseteq$ SPACE$(\log^2(n))$.

Théorème 11 *Soient $f(n) \geq n$ et $g(n) \geq n$ deux fonctions constructibles vérifiant :*

$$\liminf_{n\to\infty} \frac{f(n)\log(f(n))}{g(n)} = 0 \text{ et } \lim_{n\to\infty} \frac{g(n)}{n} = \infty$$

Alors il existe un langage L accepté par une machine de Turing déterministe opérant sur une entrée x en temps $g(|x|)$ mais par aucune machine de Turing déterministe opérant sur une entrée x en temps $f(|x|)$.

Preuve
Nous construisons une machine $\mathcal{U}'$ qui est une variante de la machine universelle du théorème 7. $\mathcal{U}'$ a une bande supplémentaire dite bande de compteur pour stocker un compteur. Soit x une entrée de taille n, $\mathcal{U}'$ commence par calculer $g(n)$ ce qui revient à marquer sa bande de compteur en position $g(n)$ (le tout en temps $O(g(n))$). Par la suite, chaque pas d'exécution de $\mathcal{U}'$ déplace le marqueur à gauche avec arrêt et rejet si le marqueur se « déplace » à gauche de la bande.

Si x n'est pas la représentation d'une machine de Turing (disons $\mathcal{M}$) alors $\mathcal{U}'$ rejette.

Puis $\mathcal{U}'$ entreprend la simulation de $\mathcal{M}$ sur x. Si la simulation se termine sans rejet dû au compteur, alors $\mathcal{U}'$ rejette ssi $\mathcal{M}$ accepte.

Soit L le langage accepté par $\mathcal{U}'$. Par construction $\mathcal{U}'$ opère en temps $O(g(n))$. Supposons que L soit reconnu par une machine $\mathcal{M}$ qui opère en temps $f(n)$. D'après le théorème 7 la simulation de $\mathcal{M}$ requiert un temps $O(f(n)\log(f(n)))$. Sachant que $\liminf_{n\to\infty} \frac{f(n)\log(f(n))}{g(n)} = 0$ et qu'on peut choisir n quelconque suffisamment grand pour la taille d'une représentation x de $\mathcal{M}$, la simulation de $\mathcal{M}$ pour un tel x se poursuit jusqu'à son terme conduisant à un résultat différent pour $\mathcal{M}$ et $\mathcal{U}'$. Donc L n'est pas reconnu par une machine qui opère en espace $f(n)$.

Puisque $\lim_{n\to\infty} \frac{g(n)}{n} = \infty$, il est possible de construire une machine $\mathcal{U}''$ qui reconnaît L et qui opère en temps $g(n)$ (ceci se fait en groupant les cellules par bloc sur les bandes de travail et en utilisant une bande de travail supplémentaire pour recopier l'entrée sous forme bloc).

c.q.f.d. ◊◊◊

Le théorème de hiérarchie pour les machines non déterministes est plus difficile à établir car l'argument de diagonalisation requiert pour la machine « universelle » le calcul de tous les chemins d'exécution de la machine à simuler afin d'accepter si tous ces chemins ne sont pas acceptants.

Théorème 12 *Soient $f(n) \geq n$ et $g(n) \geq n$ deux fonctions constructibles vérifiant :*

$$\lim_{n\to\infty} \frac{f(n+1)}{g(n)} = 0 \text{ et } \lim_{n\to\infty} \frac{g(n)}{n} = \infty$$

Alors il existe un langage L accepté par une machine de Turing non déterministe opérant sur une entrée x en temps $g(|x|)$ mais par aucune machine de Turing non déterministe opérant sur une entrée x en temps $f(|x|)$.

Preuve

On définit d'abord une suite d'intervalles de $\mathbb{N}$ définis par $]u_k, u_{k+1}]$ (et l'intervalle $[0,0]$) avec $u_0 = 0$ et $u_{k+1} = f(u_k + 1)2^{f(u_k+1)}$.

Nous construisons une machine $\mathcal{U}'$ qui est une variante de la machine universelle non déterministe du théorème 8. $\mathcal{U}'$ a une bande supplémentaire dite bande de compteur pour stocker un compteur. Soit x une entrée de taille n, $\mathcal{U}'$ commence par calculer $g(n)$ ce qui revient à marquer sa bande de compteur avec un marqueur en position $g(n)$ (le tout en temps $O(g(n))$). Par la suite, chaque pas d'exécution de $\mathcal{U}'$ déplace le marqueur à gauche avec arrêt et rejet si le marqueur se « déplace » à gauche de la bande.

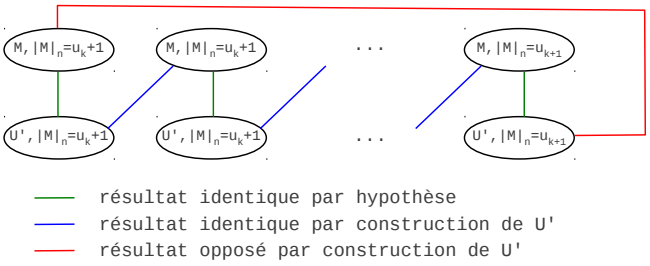
Si x n'est pas la représentation d'une machine de Turing (disons $\mathcal{M}$) alors $\mathcal{U}'$ rejette. Dans le cas contraire, $\mathcal{U}'$ détermine à quel intervalle $]u_k, u_{k+1}]$, $|x|$ appartient. Ceci se fait en un temps $O(|x|)$ en raison de la croissance exponentielle de la suite $\{u_k\}$.

Si $|x| < u_{k+1}$, $\mathcal{U}'$ entreprend la simulation de $\mathcal{M}$ sur $1x$. Si la simulation se termine sans rejet dû au compteur, alors $\mathcal{U}'$ accepte ssi $\mathcal{M}$ accepte.

Si $|x| = u_{k+1}$, $\mathcal{U}'$ entreprend une simulation *déterministe* de $\mathcal{M}$ sur le suffixe de x de taille $u_k + 1$. Si la simulation se termine sans rejet dû au compteur, alors $\mathcal{U}'$ rejette ssi $\mathcal{M}$ accepte. Observons que cette simulation se fait en $O(T2^T)$ où T est le temps d'exécution de $\mathcal{M}$ sur ce suffixe.

Soit L le langage accepté par $\mathcal{U}'$. Par construction $\mathcal{U}'$ opère en temps $O(g(n))$. Supposons que L soit reconnu par une machine $\mathcal{M}$ qui opère en espace $f(n)$. D'après le théorème 8 la simulation de $\mathcal{M}$ sur une entrée de taille différente d'un u_k requiert un temps $O(f(n+1))$. Sachant que $\lim_{n\to\infty} \frac{f(n+1)}{g(n)} = 0$ et qu'on peut choisir n quelconque suffisamment grand pour un intervalle $]u_k, u_{k+1}]$ de taille de représentation de $\mathcal{M}$, la simulation de $\mathcal{M}$ pour les x se poursuit jusqu'à son terme. Seul le cas $|x| = u_{k+1}$ nécessite une explication. Puisque la simulation se fait sur un x' de taille $u_k + 1$ la simulation déterministe prend un temps $O(f(u_k+1)2^{f(u_k+1)}) = O(u_{k+1}) = O(f(u_{k+1}+1))$ car la simulation déterministe doit examiner tous les calculs de la machine non déterministe et il y en a au plus $2^{f(u_k+1)}$.

Examinons maintenant les différents résultats. Dans la suite, x est la représentation de $\mathcal{M}$ de taille $|x|$. Pour tout $u_k + 1 \leq |x| < u_{k+1}$, le résultat de $\mathcal{M}$ et de $\mathcal{U}'$ coïncident mais le résultat de $\mathcal{U}'$ sur x est par définition le résultat de $\mathcal{M}$ sur $1x$. Donc pour tout $u_k + 1 \leq |x| \leq u_{k+1}$, le résultat de $\mathcal{M}$ est identique. Or $\mathcal{U}'$ sur l'entrée x de taille u_{k+1} a le résultat opposé à celui de $\mathcal{M}$ sur l'entrée x de taille $u_k + 1$ donc à celui sur l'entrée x de taille u_{k+1}. D'où la contradiction. Ce raisonnement est illustré ci-dessous.



Donc L n'est pas reconnu par une machine qui opère en espace $f(n)$.

Puisque $\lim_{n\to\infty} \frac{g(n)}{n} = \infty$, il est possible de construire une machine $\mathcal{U}''$ qui reconnaît L et qui opère en temps $g(n)$ (ceci se fait en groupant les cellules par bloc sur les bandes de travail et en utilisant une bande de travail supplémentaire pour recopier l'entrée sous forme bloc).

c.q.f.d. ◊◊◊