

Complexité  
*(L3 - deuxième semestre)*

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

12 mars 2012

# Table des matières

<b>1</b>	<b>Introduction aux classes de complexité</b>	<b>2</b>
1.1	Préliminaires . . . . .	2
1.2	Classes de complexité . . . . .	3
1.3	Réductions . . . . .	6
1.4	TD 1 . . . . .	8
<b>2</b>	<b>La classe NP</b>	<b>10</b>
2.1	Satisfaisabilité d'une formule propositionnelle . . . . .	10
2.2	Recherche de circuit hamiltonien . . . . .	12
2.3	Problème du sac à dos . . . . .	17
2.4	Recherche de chemins dans des graphes pondérés . . . . .	18
2.5	TD 2 . . . . .	21
<b>3</b>	<b>La classe PSPACE</b>	<b>23</b>
3.1	Le théorème de Savitch . . . . .	23
3.2	Problèmes PSPACE-complets . . . . .	24
3.2.1	Satisfaisabilité d'une formule booléenne quantifiée . . . . .	24
3.2.2	Universalité des langages réguliers . . . . .	27
3.3	TD 3 . . . . .	30
3.4	TD 4 . . . . .	32
<b>4</b>	<b>La classe PTIME</b>	<b>36</b>
4.1	Satisfaisabilité d'une conjonction de clauses de Horn . . . . .	36
4.2	Valeur d'un circuit . . . . .	38
4.3	Accessibilité dans un graphe <i>non déterministe</i> . . . . .	41
4.4	Test de la vacuité d'un langage algébrique . . . . .	42
4.5	TD 5 . . . . .	44
<b>5</b>	<b>La classe NLOGSPACE</b>	<b>46</b>
5.1	Accessibilité dans un graphe . . . . .	46
5.2	Le théorème d'Immerman-Szelepcényi . . . . .	47
<b>6</b>	<b>Inclusions strictes entre classes</b>	<b>49</b>
6.1	Machines de Turing universelles . . . . .	49
6.2	Hierarchies de complexité . . . . .	53

# 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  $f$  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 chemins enrichi de la valeur **false**. On décide l'existence du chemin et on exhibe un chemin s'il en existe.

#### 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. 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.

## 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  $f$  en une instance d'un problème  $g$  telles qu'à partir du résultat de l'instance du problème  $g$ , on calcule efficacement le résultat du problème  $f$ . 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 dans afin d'éviter les ambiguïtés, il est naturel de se tourner vers les *machines de Turing*. Nous convenons 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.

La machine de 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 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 retrancher 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 une entrée de longueur  $n$ , une sortie constituée de la représentation unaire de  $f(n)$  (ou d'un marqueur sur la cellule  $f(n)$ , e.g. le mot  $0^{n-1}1$ ) en un temps  $O(n + f(n))$  et en utilisant un espace en  $O(f(n))$ .

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

On introduit d'abord les classes de complexités relatives à des fonctions de mesure.

**Définition 2** Un langage  $L$  appartient à :

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

- $\text{TIME}(f) \subseteq \text{NTIME}(f)$
- $\text{SPACE}(f) \subseteq \text{NSPACE}(f)$
- $\text{TIME}(f) \subseteq \text{SPACE}(f)$
- $\text{NTIME}(f) \subseteq \text{NSPACE}(f)$

**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.*  $\diamond\diamond$

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

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

**Preuve**

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

*c.q.f.d.*  $\diamond\diamond$

Un théorème similaire relatif au temps d'exécution sera démontré en TD.

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

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

**Preuve**

Etant donnée une machine de Turing non déterministe  $\mathcal{M}$  opérant en temps  $f(n)$ . Cette machine 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**.

L'indice de boucle est stocké en  $O(f(n))$  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.*  $\diamond\diamond$

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

$$\text{NSPACE}(f) \subseteq \bigcup_{i \in \mathbb{N}} \text{TIME}(i^{f+\log})$$

### Preuve

Etant donnée une machine de Turing à  $k$  bandes de travail opérant en espace  $f(n)$ . Cette machine a un nombre de configurations borné par  $|Q| \times |\Sigma|^{kf(n)} \times \max(f(n), n)^k$  (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.

La machine de Turing déterministe 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.*  $\diamond\diamond$

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 :

- LOGSPACE  $\equiv$  SPACE( $\log_2(n)$ ),  
NLOGSPACE  $\equiv$  NSPACE( $\log_2(n)$ ),
- PTIME  $\equiv \bigcup_{k \in \mathbb{N}} \text{TIME}(n^k)$  (appelée aussi P)  
NPTIME  $\equiv \bigcup_{k \in \mathbb{N}} \text{NTIME}(n^k)$  (appelée aussi NP),
- PSPACE  $\equiv \bigcup_{k \in \mathbb{N}} \text{SPACE}(n^k)$ ,  
NPSPACE  $\equiv \bigcup_{k \in \mathbb{N}} \text{NSPACE}(n^k)$ ,
- EXPTIME  $\equiv \bigcup_{k \in \mathbb{N}} \text{TIME}(2^{n^k})$  (appelée aussi EXP),  
NEXPTIME  $\equiv \bigcup_{k \in \mathbb{N}} \text{NTIME}(2^{n^k})$  (appelée aussi NEXP), etc.

A l'aide des théorèmes précédents, on obtient les inclusions suivantes. Nous discuterons plus loin de cette suite d'inclusions.

$$\text{LOGSPACE} \subseteq \text{NLOGSPACE} \subseteq \text{PTIME} \subseteq \text{NPTIME} \subseteq \text{PSPACE} \subseteq \text{NPSPACE}$$

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. 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**.

## 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  $f : A \mapsto \{\mathbf{false}, \mathbf{true}\}$  à un problème  $f' : A' \mapsto \{\mathbf{false}, \mathbf{true}\}$  est une fonction calculable  $h : A \mapsto A'$  telle que  $\forall a \in A \ f'(h(a)) = f(a)$ .

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 pré-ordre 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  $f$  se réduise en temps polynomial (resp. en espace logarithmique) à  $f'$  et que  $f'$  se réduise en temps polynomial (resp. en espace logarithmique) à  $f''$  alors  $f$  se réduit en temps polynomial (resp. en espace logarithmique) à  $f''$ .*

### 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.*  $\diamond\diamond$

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  $pb$  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  $x$  de taille  $n$ . La réduction consiste donc à concevoir un algorithme

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

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 ici  $p(n)$  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  $\epsilon > 0$ ,

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

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  $\wedge, \vee, \top, \perp$ . Lorsqu'on arrive dans un état  $\top$ , la machine accepte; lorsqu'on arrive dans un état  $\perp$ , 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,
- $\Sigma$  est l'alphabet d'entrée, on y ajoute un marqueur de fin  $\$ \notin \Sigma$ ,
- $\Gamma$  est l'alphabet de travail, on y ajoute un caractère blanc  $\# \notin \Gamma$ ,
- $\delta \subseteq Q \times (\Gamma \cup \{\#\})^k \times (\Sigma \cup \{\$\}) \times Q \times \Gamma^k \times \{\text{left, right}\}^k \times \{\text{left, right}\}$  est la fonction de transition,
- $\lambda : Q \rightarrow \{\wedge, \vee, \top, \perp\}$  est la fonction d'étiquetage des états.

Une transition  $(q, (a_1, \dots, a_k), c, q', (b_1, \dots, b_k), (m_1, \dots, m_k), m) \in \delta$  indique que la machine peut passer de l'état  $q$  à l'état  $q'$  en lisant  $(a_1, \dots, a_k)$  sur les bandes de travail et  $c$  sur l'entrée, et qu'elle écrit alors  $(b_1, \dots, b_k)$  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  $\text{ATIME}(f(n))$  (respectivement  $\text{ASPACE}(f(n))$ ) 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  $\text{ATIME}(f(n)) \subseteq \text{SPACE}(f(n))$ .
2. pour  $f(n) \geq \log(n)$ , on a  $\text{ASPACE}(f(n)) \subseteq \bigcup_{c>0} \text{TIME}(c^{f(n)})$ .
3. pour  $f(n) \geq n$ , on a  $\text{NSPACE}(f(n)) \subseteq \bigcup_{c>0} \text{ATIME}(cf(n)^2)$ .
4. pour  $f(n) \geq n$  et  $c > 0$ , on a  $\text{TIME}(f(n)) \subseteq \text{ASPACE}(c \log(f(n)))$ .

### 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 propositionnelle* est bâtie à partir de propositions (atomiques) appartenant à  $\mathcal{P}$  et des connecteurs  $\neg$ ,  $\vee$  et  $\wedge$ . Par exemple  $\varphi \equiv p_1 \wedge (p_2 \vee \neg p_1)$  est une telle formule avec  $p_1, p_2 \in \mathcal{P}$ .

Une interprétation  $\nu$  est une fonction de  $\mathcal{P}$  dans  $\{\mathbf{false}, \mathbf{true}\}$ . Etant donnée une interprétation des propositions, on peut déduire par induction une interprétation pour toute formule. Par exemple, si  $\nu(p_1) = \nu(p_2) = \mathbf{false}$  alors :

$$\nu(\varphi) = \mathbf{false} \wedge (\mathbf{false} \vee \neg\mathbf{false}) = \mathbf{false} \wedge (\mathbf{false} \vee \mathbf{true}) = \mathbf{false} \wedge \mathbf{true} = \mathbf{false}$$

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

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.

**Proposition 2** *Soit  $\varphi$  une formule CNF de logique propositionnelle. Alors le problème de la satisfaisabilité de  $\varphi$  est dans NP.*

#### Preuve

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

Les deux étapes se font en temps polynomial.

*c.q.f.d.*  $\diamond\diamond$

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

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

### Preuve

Soit une machine de Turing non déterministe  $\mathcal{M} = (Q, T, A, \delta, \flat, q\theta, qf)$  qui s'exécute en temps polynomial vis à vis de son entrée  $x$ , disons  $p(n) \geq n$  où  $p$  est un polynôme et  $n$  est la taille de  $x$ .  $Q$  est l'ensemble des états,  $T$  les symboles de la bande,  $A \subseteq T \setminus \{\flat\}$  l'alphabet des entrées,  $\flat \in T$  le blanc,  $q\theta, qf$  les états initial et final. Par conséquent, la bande de la machine de Turing utilise au plus  $p(n)$  cellules. On suppose de plus qu'une fois arrivée dans l'état  $qf$ , la machine reste indéfiniment dans cet état.

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  $i$ .

- $r_{i,j}$  signifie que la tête de lecture est à l'instant  $j$  au dessus de la cellule  $i$ ;
- Pour tout  $a \in T$ ,  $a_{i,j}$  signifie qu'à l'instant  $j$  le contenu de la cellule  $i$  est  $a$ ;
- Pour tout  $q \in Q$ ,  $q_j$  signifie qu'à l'instant  $j$ , la machine est dans l'état  $q$ ;

Nous décrivons maintenant un ensemble de formules simultanément satisfaites ssi il existe un calcul de la machine (en temps  $p(n)$ ).

- Initialement la machine est dans l'état  $q\theta$  et la tête se trouve au dessus de la cellule 0. D'où les formules  $q\theta_0, r_{0,0}$ ;
- Initialement la bande contient le mot  $x$ . D'où les formules  $\$_{0,0}$ , pour tout  $1 \leq i \leq n$   $x(i)_{i,0}$  et pour tout  $n < i \leq p(n)$   $b_{i,0}$ ;
- La machine termine dans l'état  $qf$ . D'où la formule  $qf_{p(n)}$ ;
- A tout instant la formule est dans un seul état. D'où les formules, pour tout  $0 \leq j \leq p(n)$ ,  $\bigvee_{q \in Q} q_j$  et pour tout  $0 \leq j \leq p(n)$ , pour tout  $q \neq q' \in Q$   $\neg q_j \vee \neg q'_j$ .
- A tout instant la tête de lecture est au dessus d'une unique cellule. D'où les formules, pour tout  $0 \leq j \leq p(n)$ ,  $\bigvee_{0 \leq i \leq p(n)} r_{i,j}$  et pour tout  $0 \leq j \leq p(n)$ , pour tout  $i \neq i'$   $\neg r_{i,j} \vee \neg r_{i',j}$ .
- A tout instant une cellule contient une unique lettre. D'où les formules, pour tout  $0 \leq i, j \leq p(n)$ ,  $\bigvee_{a \in T} a_{i,j}$  et pour tout  $0 \leq i, j \leq p(n)$ , pour tout  $a \neq a'$   $\neg a_{i,j} \vee \neg a'_{i,j}$ .
- A tout instant, si une cellule n'est pas sous la tête de la machine elle ne change pas de contenu à l'instant suivant.  $0 \leq j < p(n)$  et tout  $0 \leq i \leq p(n)$ ,  $(\neg r_{i,j} \wedge a_{i,j}) \Rightarrow a_{i,j+1}$ .
- Soit  $\delta(q, a) \equiv \{(q^1, a^1, \pm^1), \dots, (q^s, a^s, \pm^s)\}$  avec  $\pm^k \in \{+, -\}$ . Pour tout  $0 \leq j < p(n)$  et tout  $0 \leq i \leq p(n)$ , on a la formule :  
 $(q_j \wedge r_{i,j} \wedge a_{i,j}) \Rightarrow \bigvee_{1 \leq k \leq s | 0 \leq i \pm^k 1 \leq p(n)} (q_{j+1}^k \wedge r_{i \pm^k 1, j+1} \wedge a_{i, j+1}^k)$ .

Le nombre de sous-formules est quadratique par rapport à  $p(n)$ , donc polynomial par rapport à  $n$ . Toutes les formules sauf celles associées aux changements de configurations sont des clauses. La transformation de ces formules en un ensemble de clauses entraîne une augmentation de la taille de ces formules mais cette taille reste indépendante de  $n$ . Par conséquent la transformation se fait en temps polynomial.

Un calcul se traduit immédiatement par une interprétation qui satisfait la formule et vice versa ce qui achève la preuve.

*c.q.f.d.*  $\diamond\diamond$

Afin de simplifier les futures réductions nécessaires à la démonstration que des problèmes sont NP-difficiles, nous allons encore restreindre la forme des formules propositionnelles. Une formule est sous forme 3CNF si elle est sous forme CNF et les clauses ont au plus trois littéraux.

**Proposition 4** *Soit  $\varphi$  une formule 3CNF de logique propositionnelle. Alors le problème de la satisfaisabilité de  $\varphi$  est NP-difficile (donc NP-complet).*

**Preuve**

Etant donnée une formule CNF  $\varphi$ , nous construisons en temps polynomial une formule 3CNF  $\varphi'$  telle que  $\varphi$  est satisfaisable ssi  $\varphi'$  est satisfaisable.

Pour chaque clause  $c \equiv x_1 \vee \dots \vee x_k$  de  $\varphi$  avec  $k > 3$ , on introduit  $k - 2$  nouvelles propositions  $p_{c,3}, \dots, p_{c,k}$ . Les clauses correspondantes dans  $\varphi'$  sont :

$$x_1 \vee x_2 \vee \neg p_{c,3}, p_{c,3} \vee x_3 \vee \neg p_{c,4}, \dots, p_{c,k-1} \vee x_{k-1} \vee \neg p_{c,k}, p_{c,k} \vee x_k.$$

Montrons que  $\varphi$  est satisfaisable ssi  $\varphi'$  est satisfaisable. Soit une interprétation  $\nu$  satisfaisant  $\varphi$ . Nous construisons  $\nu'$  en étendant  $\nu$  aux nouvelles propositions de la manière suivante. Soit  $x_i$  un littéral de la clause  $c$  tel que  $\nu(x_i) = \mathbf{true}$ . Si  $i \leq 2$ , on pose pour tout  $3 \leq l \leq k$  on pose  $\nu'(p_{c,l}) = \mathbf{true}$ . Sinon pour tout  $l \leq i$  on pose  $\nu'(p_{c,l}) = \mathbf{false}$  et pour tout  $l > i$  on pose  $\nu'(p_{c,l}) = \mathbf{true}$ . On laisse le soin au lecteur de vérifier que les clauses de  $\varphi'$  correspondant à  $c$  sont satisfaites.

Soit une interprétation  $\nu'$  satisfaisant  $\varphi'$ . Démontrons que  $\nu'$  satisfait  $\varphi$ . Supposons qu'une clause  $c$  qui a été transformée dans  $\varphi'$  ne soit pas satisfaite. Par conséquent, pour tout  $1 \leq i \leq k$ ,  $\nu'(x_i) = \mathbf{false}$ . La satisfaction de la clause  $x_1 \vee x_2 \vee \neg p_{c,3}$  implique  $\nu'(p_{c,3}) = \mathbf{false}$ . La satisfaction des clauses  $p_{c,i-1} \vee x_{i-1} \vee \neg p_{c,i}$  implique par récurrence que  $\forall 3 < i \leq k$   $\nu'(p_{c,k}) = \mathbf{false}$ . Ceci implique que la clause  $p_{c,k} \vee x_k$  n'est pas satisfaite contrairement à notre hypothèse.

*c.q.f.d.*  $\diamond\diamond$

On pourrait se demander si ce résultat pourrait être encore amélioré en imposant aux clauses d'avoir au plus deux littéraux (forme 2CNF). Ce n'est pas le cas : la satisfaisabilité d'une formule 2CNF se résout en temps polynomial (voir les TD).

## 2.2 Recherche de circuit hamiltonien

On se donne un graphe orienté  $G = (V, E)$  où  $V = \{v_0, \dots, v_{n-1}\}$  est l'ensemble des sommets et  $E \subseteq V \times V$  est l'ensemble des arcs. Un circuit hamiltonien est une permutation  $\alpha$  de  $\{0, \dots, n-1\}$  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

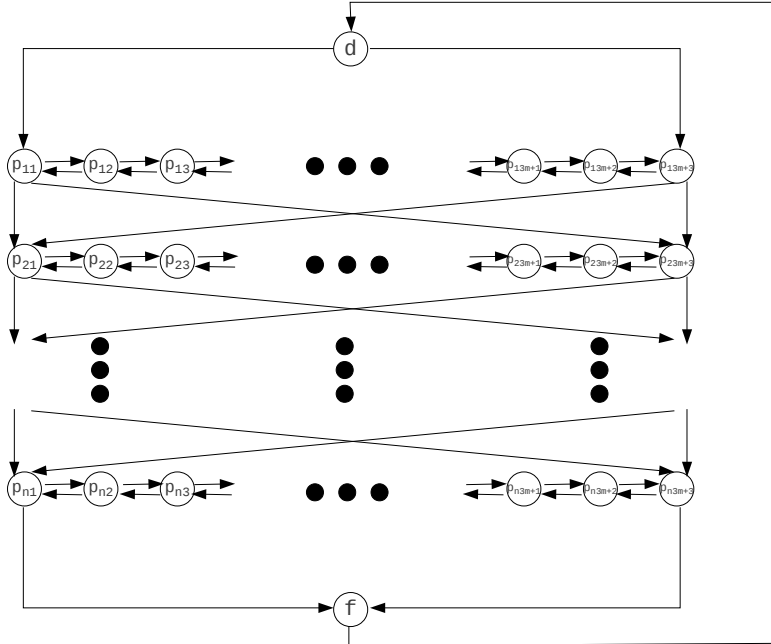


FIGURE 2.1: Le sous-graphe indépendant de la formule

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  $p_1, \dots, p_n$ . 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.

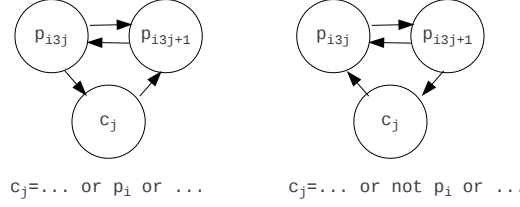


FIGURE 2.2: Ajout d'arcs par clause

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  $p_{1,2}, \dots, p_{1,3m+2}$  ou les sommets  $p_{1,3m+2}, \dots, p_{1,1}$ .
- 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.
- 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  $\varphi$  est satisfaite par l'interprétation  $\nu$ . On construit d'abord un circuit (non hamiltonien) de la manière suivante. On démarre en  $d$  et on poursuit par  $p_{1,1}$  si  $\nu(p_0) = \mathbf{true}$  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  $\nu(p_2) = \mathbf{true}$  et en  $p_{2,3m+3}$  sinon et on itère le procédé. On se termine par  $f$  et on boucle le circuit. On choisit ensuite pour chaque clause  $c_j$  un littéral  $x_j$  tel que  $\nu(x_j) = \mathbf{true}$ . 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

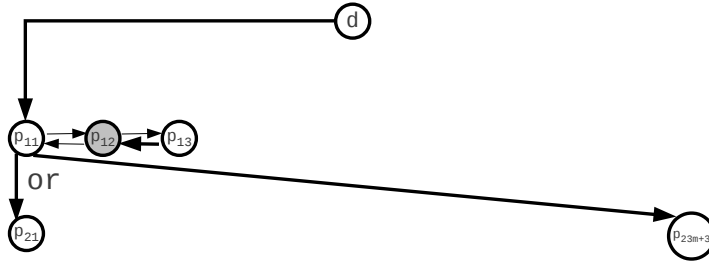


FIGURE 2.3: Un scénario impossible



FIGURE 2.4: Deux autres scénarios impossibles

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.

*c.q.f.d.*  $\diamond\diamond$

On se donne un graphe non orienté  $G = (V, E)$  où  $V = \{v_0, \dots, v_{n-1}\}$  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  $\alpha$  de  $\{0, \dots, n-1\}$  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.



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

**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 circuit 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). Nous affirmons que tous les sommets  $u_1, u_2, u_3$  apparaissent dans cet ordre et que les arêtes  $\{u_3, v_1\}$  apparaissent dans cet ordre.

Supposons que cela ne soit pas le cas et examinons la première fois que cette contrainte n'est pas satisfaite. Forcément la violation de cette contrainte correspond à une occurrence  $u_3, v_1, w_3$  (avec  $u, v, w$  tous différents). Par conséquent  $v_2$  n'est pas encore apparu. Il ne peut être ensuite atteint que par  $v_3$  mais le cycle ne peut être prolongé (voir la figure 2.6).

Puisque toutes les arêtes  $\{u_3, v_1\}$  apparaissent dans le même ordre, on leur substitue les arcs de  $G$  et on supprime les arêtes adjacentes aux sommets  $u_2$ . On obtient ainsi un circuit hamiltonien.

*c.q.f.d.*  $\diamond\diamond$

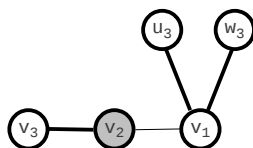


FIGURE 2.6: Un scénario impossible

## 2.3 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  $\{v_1, \dots, v_n, w\}$  et on cherche à déterminer s'il existe un sous-ensemble  $I \subseteq \{1, \dots, n\}$  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 polynomial 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 indiqué par une clause ou par une proposition.

Il y aura  $2n + 2m$  nombres que nous notons  $\{a_1, a'_1, \dots, a_n, a'_n, b_1, b'_1, \dots, b_m, b'_m\}$ .  
Le nombre  $a_i$  a pour chiffres :

- 1 pour le chiffre associé à  $p_i$  et à toute clause  $c_j$  où  $p_i$  apparaît positivement.
- 0 ailleurs.

Le nombre  $a'_i$  a pour chiffres :

- 1 pour le chiffre associé à  $p_i$  et à toute clause  $c_j$  où  $p_i$  apparaît négativement.
- 0 ailleurs.

Les nombres  $b_j = b'_j$  ont pour chiffres :

- 1 pour le chiffre associé à  $c_j$ .
- 0 ailleurs.

Le nombre à atteindre  $a$  pour chiffres :

- 3 pour les chiffres associés aux clauses.
- 1 pour les chiffres associés aux propositions.

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	$i$	$n$	$n+1$	$n+j$	$n+k$	$n+m$
$a_i$	0	1	0		1	0	
$a'_i$	0	1	0		0	1	
$b_j, b'_j$	...	0	...	0	1	0	0
	1	...	1	3	...	...	3

On observe d'abord que la somme de tous les nombres 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  $\nu$  alors on choisit pour chaque proposition  $p_i$  le nombre  $a_i$  si  $\nu(p_i) = \mathbf{true}$  et le nombre  $a'_i$  sinon. Pour chaque clause  $c_j$ , on complète par  $b_j$  et  $b'_j$  si le nombre de littéraux qui satisfont  $c_j$  est 1, et par uniquement  $b_j$  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  $a_i$  soit  $a'_i$  mais pas les deux simultanément. Définissons  $\nu(p_i) = \mathbf{true}$  si  $a_i$  est choisi et  $\nu(p_i) = \mathbf{false}$  sinon. Puisque la somme est égale au nombre recherché, pour chaque chiffre associé à une clause l'un des littéraux associé à la clause s'évalue à  $\mathbf{true}$ .

*c.q.f.d.*  $\diamond\diamond\diamond$

Nous allons maintenant montrer l'importance de la représentation binaire.

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

**Preuve**

On maintient  $T$  un tableau de booléens indicé par les paires  $(i, x)$  avec  $i \leq n$  et  $x \leq w$  tel que  $T[i, x] = \mathbf{true}$  s'il existe une solution au problème  $\{v_1, \dots, v_i, x\}$ . La formule suivante démontre la proposition :  $T[i+1, x] = T[i, x] \vee (x \geq v_{i+1} \wedge T[i, x - v_{i+1}])$ .

*c.q.f.d.*  $\diamond\diamond\diamond$

## 2.4 Recherche de 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  $p : E \mapsto \mathbb{N}$ . Etant donné un chemin  $\sigma \equiv u_0, \dots, u_n$  dans le graphe, son poids  $p(\sigma)$  est défini par :  $p(u_0, u_1) + \dots + p(u_{n-1}, u_n)$

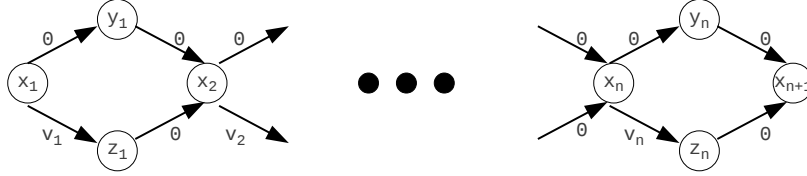


FIGURE 2.7: Le graphe pondéré de la somme de sous-ensembles

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  $\sigma$  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  $\{v_1, \dots, v_n, w\}$  un problème de somme de sous-ensembles. On construit le graphe ainsi (voir la figure 2.7) :

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

On cherche à décider s'il existe un chemin de  $x_1$  à  $x_{n+1}$  de poids  $w$ . Il existe  $2^n$  chemins de  $x_1$  à  $x_{n+1}$  dont le poids de chacun correspond à une somme de sous-ensembles. Ceci établit la correction de la réduction.

*c.q.f.d.*  $\diamond\diamond$

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 chemin 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.

**Lemme 1 (Euler)** *Soit un graphe orienté  $G = (V, E)$ ,  $u, v$  deux sommets distingués et  $\vec{s} \in \mathbb{N}^E$  un vecteur d'occurrences d'arcs qui vérifient :*

- *Pour tout sommet  $w \notin \{u, v\}$ , le nombre de transitions entrant en  $w$  est égal au nombre de transitions sortant de  $w$  dans  $\vec{s}$ . Si  $u = v$  alors la propriété est aussi vérifiée pour  $u$  et il y a au moins une transition sortant de  $u$ . Sinon le nombre de transitions sortant de  $u$  est égal au nombre de transitions entrant en  $u$  plus un et le nombre de transitions sortant de  $v$  est égal au nombre de transitions entrant de  $v$  moins un.*

- Le sous-graphe induit par les transitions du support de  $\vec{s}$  est connexe (autrement dit, on oublie l'orientation).

Alors il existe un chemin  $\sigma$  de  $u$  à  $v$  tel que son vecteur d'occurrences soit égal à  $\vec{s}$ .

### Preuve

On traite uniquement le cas  $u \neq v$  (le cas  $u = v$  est similaire).

On construit le chemin en maintenant un vecteur d'occurrences initialisé à  $\vec{s}$ . On démarre en  $u$  et on choisit un arc sortant de  $u$  (il y en a au moins un). Sinon on décrémente le vecteur de l'arc choisi et on continue à partir du sommet atteint. Examinons le sommet courant lorsque la procédure s'arrête faute d'un arc sortant.

- Le sommet ne peut pas être  $u$  car il y a plus d'arcs sortant de  $u$  que d'arcs entrant en  $u$ .
- Le sommet est donc  $v$  puisque tout autre sommet compte autant d'arcs entrants que sortants. De plus tous les arcs connexes à  $v$  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  $u$  à  $v$ . Cette procédure se termine nécessairement fournissant le chemin recherché.

*c.q.f.d.*  $\diamond\diamond$

**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  $\vec{s}$  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  $w \notin \{u, v\}$ , le nombre de transitions entrant en  $w$  est égal au nombre de transitions sortant de  $w$  dans  $\vec{s}$ . Si  $u = v$  alors la propriété est aussi vérifiée pour  $u$ . Sinon le nombre de transitions sortant de  $u$  est égal au nombre de transitions entrant en  $u$  plus un et le nombre de transitions sortant de  $v$  est égal au nombre de transitions entrant en  $v$  moins un ;
- le poids du vecteur est égal à  $a : \sum_{(x,y) \in E} \vec{s}_{(x,y)} p(x,y) = a$

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

*c.q.f.d.*  $\diamond\diamond$

## 2.5 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  $c : V \rightarrow \{0, \dots, k-1\}$  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  $G$  ?

Montrer que le problème reste NP-complet pour les graphes planaires de degré borné par 4.

**Exercice 5 (GRAPH HOMOMORPHISM)** Un homomorphisme d'un graphe  $G = (V, E)$  à un graphe  $G' = (V', E')$  est une fonction  $h : V \rightarrow V'$  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  $h : V \rightarrow V'$  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 proposée est une procédure récursive, disons **Cherche** qui prend en entrée trois paramètres, un sommet source **in**, un sommet destination **out** et une longueur **lg**. Cette procédure renvoie vrai s'il existe un chemin de longueur au plus **lg** qui joint **in** à **out**. 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  $lg \leq 1$  elle teste si  $in = out$  ou s'il existe un arc  $(in, out)$ .
- Si  $lg > 1$  elle parcourt l'ensemble des sommets avec une variable **mid**. Dans une itération, elle teste par deux appels récursifs s'il existe un chemin d'une longueur au plus  $\lceil \frac{lg}{2} \rceil$  qui joint **in** à **mid** et un chemin d'une longueur au plus  $\lfloor \frac{lg}{2} \rfloor$  qui joint **mid** à **out**. Si elle les trouve elle renvoie vrai. A la fin des itérations, elle renvoie faux.

La correction de cette procédure est évidente. Analysons sa complexité en espace. Il y a 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.*  $\diamond\diamond$

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

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

**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. 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.*  $\diamond\diamond$

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

**Corollaire 2**  $\text{PSPACE} = \text{NPSPACE}$

Notre suite d'inclusions devient :

$$\text{LOGSPACE} \subseteq \text{NLOGSPACE} \subseteq \text{PTIME} \subseteq \text{NP} \subseteq \text{PSPACE}$$

## 3.2 Problèmes PSPACE-complets

### 3.2.1 Satisfaisabilité d'une formule booléenne quantifiée

Une *formule booléenne quantifiée (QBF)* est une formule de la forme  $\varphi \equiv Q_n x_n \dots Q_1 x_1 \psi$  où  $\varphi$  est une formule propositionnelle sur l'ensemble des propositions  $\{x_1, \dots, x_m\}$  avec  $m \geq n$ . Soit une interprétation  $\bar{e} : \{x_1, \dots, x_m\} \mapsto \{\mathbf{V}, \mathbf{F}\}$ , on définit inductivement (par rapport à  $n$ ) la satisfaction de  $\varphi$  par  $\bar{e}$ , notée  $\bar{e} \models \varphi$ . Dans la suite,  $\bar{e}[v, n]$  désigne l'interprétation  $\bar{e}'$  obtenue à partir de  $\bar{e}$  en fixant  $\bar{e}'(x_n) = v$ .

- Si  $n = 0$ , il s'agit de satisfaction standard en logique propositionnelle.
- Si  $Q_n = \exists$ ,  $\bar{e} \models \varphi$  ssi il existe  $v \in \{\mathbf{V}, \mathbf{F}\}$  t.q.  $\bar{e}[v, n] \models Q_{n-1}x_{n-1} \dots Q_1x_1\psi$ .
- Si  $Q_n = \forall$ ,  $\bar{e} \models \varphi$  ssi pour tout  $v \in \{\mathbf{V}, \mathbf{F}\}$ ,  $\bar{e}[v, n] \models Q_{n-1}x_{n-1} \dots Q_1x_1\psi$ .

Le problème de la satisfaisabilité consiste à savoir s'il existe une interprétation des variables (booléennes) qui satisfasse la formule. Dans le cas d'une formule close, la satisfaction ne dépend pas de l'interprétation des variables. On remarque aussi que vis à vis du problème de la satisfaisabilité on peut se ramener au cas d'une formule close en quantifiant existentiellement les variables libres. Nous proposons un algorithme pour résoudre ce problème.

**Proposition 11** *Soit  $\varphi$  formule booléenne quantifiée close. Alors le problème de la satisfaisabilité de  $\varphi$  est dans PSPACE.*

**Preuve**

L'algorithme consiste en procédure récursive qui prend en entrée une formule

$QBF$ ,  $\varphi$  dont la sous-formule propositionnelle contient les variables quantifiées,  $\forall$ ,  $\exists$  et les connecteurs booléens.

Si la formule n'est pas quantifiée, alors on évalue en temps et en espace polynomial la formule qui ne contient que des constantes et on renvoie le résultat.

Si le quantificateur le plus externe est  $\forall$  (resp.  $\exists$ ) la procédure remplace dans la formule passée en paramètre la variable par  $V$ , supprime la quantification et s'appelle récursivement. Si le résultat de l'appel est  $F$  (resp.  $V$ ) la procédure renvoie  $F$  (resp.  $V$ ). Sinon la procédure remplace dans la formule passée en paramètre la variable par  $F$ , supprime la quantification, s'appelle récursivement et renvoie le résultat de son appel.

Le nombre d'appels simultanés 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  $PSPACE$ .

*c.q.f.d.*  $\diamond\diamond$

Nous voulons obtenir un résultat de complexité en restreignant la forme des formules  $QBF$  ce qui nous conduit à introduire ce lemme.

**Lemme 2** *La formule propositionnelle  $\varphi \equiv \bigwedge_{i \leq n} \varphi_i$  et la formule  $QBF$   $\psi \equiv \forall z_1 \dots \forall z_n \bigvee_{i \leq n} (\varphi_i \wedge z_i) \vee (\bigwedge_{i \leq n} \neg z_i)$  sont équivalentes.*

**Preuve**

Supposons que  $\varphi$  est vrai. Donc pour tout  $i$ ,  $\varphi_i$  est vrai. Soit  $v_1, \dots, v_n$  une interprétation de  $z_1, \dots, z_n$ . Si tous les  $v_i$  sont faux alors la dernière clause conjonctive de  $\psi$  est vrai. Sinon supposons  $v_i$  vrai, alors la clause  $\varphi_i \wedge z_i$  est vrai.

Supposons que  $\psi$  est vrai. Pour un  $i$  quelconque, choisissons la valuation  $v_1, \dots, v_n$  t.q.  $v_i$  est vrai et tous les  $v_j$ , pour  $j \neq i$ , sont faux. Alors  $\varphi_i$  doit être vrai. Donc  $\varphi$  est vrai.

*c.q.f.d.*  $\diamond\diamond$

Une formule propositionnelle est sous forme DNF si elle est une disjonction de conjonctions de littéraux. Il s'agit d'une forme duale de la forme CNF obtenue en inversant la place des  $\vee$  et des  $\wedge$ .

**Proposition 12** *Soit  $\varphi$  une formule booléenne quantifiée close dont la sous-formule propositionnelle est sous forme normale disjonctive. Alors le problème de la satisfaisabilité de  $\varphi$  est  $PSPACE$ -difficile donc  $PSPACE$ -complet.*

**Preuve**

Soit une machine de Turing déterministe  $\mathcal{M}$  qui s'exécute en espace polynomial vis à vis de son entrée.

- $newt(q, c)$  désigne le nouveau contenu de la cellule sous la tête de lecture lorsque l'état de  $\mathcal{M}$  est  $q$  et le caractère lu est  $c$ .
- $newq(q, c)$  désigne le nouvel état de  $\mathcal{M}$  lorsque l'état de  $\mathcal{M}$  est  $q$  et le caractère lu est  $c$ .
- $dep(q, c)$  désigne le déplacement de la tête de  $\mathcal{M}$  lorsque l'état de  $\mathcal{M}$  est  $q$  et le caractère lu est  $c$ .

La configuration de la machine peut être codée sous forme de bits donc de variables booléennes disons  $x_1, \dots, x_m$  où  $m$  est une fonction polynomiale de la taille  $n$  de l'entrée de la machine de Turing puisque l'espace utilisé est polynomial. Par exemple, voici un codage possible (avec renommage) :

- $state_q$  est vrai si l'état de la configuration est  $q$ .
- $head_i$  est vrai si la tête est à la position  $i$ .
- $tape_{c,i}$  est vrai si le contenu de la bande à la position  $i$  est le caractère  $c$ .

Certaines interprétations ne correspondent pas à des configurations mais cela ne nous gênera pas dans la suite.

Le nombre de configurations différentes est d'au plus  $2^m$  et une exécution acceptante exécute au plus  $2^m - 1$  pas. Nous allons écrire récursivement une formule QBF  $\varphi_i$  avec  $2m$  variables libres  $x_1, \dots, x_m, y_1, \dots, y_m$  t.q.  $\varphi_i$  est vrai si partant de la configuration représentée par  $x_1, \dots, x_m$  la machine peut atteindre en au plus  $2^i$  pas la configuration représentée par  $y_1, \dots, y_m$ . Dans la suite de la preuve  $\bar{x}$  ( $\bar{y}$ , etc.) désigne un vecteur de variables et  $Q\bar{x}$  signifie que le quantificateur  $Q$  est répété devant chacune des variables de  $\bar{x}$ .

La formule  $\varphi_0$  est définie par  $\varphi_0 \equiv \bigwedge_{i \leq m} x_i = y_i \vee \varphi'_0$  où  $\varphi'_0$  représente le fait que  $\bar{x}$  atteint  $\bar{y}$  par un pas de la machine. Nous transformons  $\bigwedge_{i \leq m} x_i = y_i$  en :  $\forall z_1 \dots \forall z_m \bigvee_{i \leq m} (x_i \wedge y_i \wedge z_i) \vee (\neg x_i \wedge \neg y_i \wedge z_i) \vee (\bigwedge_{i \leq m} \neg z_i)$  grace au lemme 2.

Cette formule a une taille linéaire par rapport à  $m$ .

La formule  $\varphi'_0$  est une conjonction de formules :

$$\begin{aligned} & \neg state_q^x \vee \neg head_i^x \vee \neg tape_{c,i}^x \vee tape_{newt(q,c),i}^y \\ & \neg state_q^x \vee \neg head_i^x \vee \neg tape_{c,i}^x \vee state_{newq(q,c)}^y \\ & \neg state_q^x \vee \neg head_i^x \vee \neg tape_{c,i}^x \vee head_{i+dep(q,c)}^y \\ & head_i^x \vee \neg tape_{c,i}^x \vee tape_{c,i}^x \\ & \text{etc.} \end{aligned}$$

Il y a un nombre linéaire de clauses (par rapport à  $m$ ) chacune d'au plus 4 littéraux. On applique le lemme 2 puis chaque sous-formule comme :

$$(\neg state_q^x \vee \neg head_i^x \vee \neg tape_{c,i}^x \vee tape_{newt(q,c),i}^y) \wedge z_j$$

est transformée en :

$$(\neg state_q^x \wedge z_j) \vee (\neg head_i^x \wedge z_j) \vee (\neg tape_{c,i}^x \wedge z_j) \vee (tape_{newt(q,c),i}^y \wedge z_j)$$

La taille de la nouvelle formule est au plus le triple de la taille originale.

Supposons que nous ayons construit  $\varphi_i$ . Alors  $\varphi_{i+1}$  est définie en introduisant une configuration intermédiaire. De plus, à l'aide d'une construction astucieuse, on évite d'écrire deux fois  $\varphi_i$  ( $\bar{z}$ ,  $\bar{t}$  et  $\bar{u}$  sont de nouveaux vecteurs de variables).

$$\varphi_{i+1} \equiv \exists \bar{z} \forall \bar{t} \forall \bar{u} ((\bar{x} = \bar{u} \wedge \bar{z} = \bar{t}) \vee (\bar{z} = \bar{u} \wedge \bar{y} = \bar{t})) \Rightarrow \varphi_i[\{\bar{x} \leftarrow \bar{u}\} \cup \{\bar{y} \leftarrow \bar{t}\}]$$

Cette formule n'est pas tout à fait sous la forme recherchée. A l'aide des transformations vues en cours de logique puisque les variables liées de  $\varphi_i$  sont absentes du reste de la formule, on peut les placer les quantificateurs de  $\varphi_i$  à la suite des quantificateurs de tête de  $\varphi_{i+1}$ . On remplace  $A \Rightarrow B$  par  $\neg A \vee B$ . Enfin :

$\neg((\bar{x} = \bar{u} \wedge \bar{z} = \bar{t}) \vee (\bar{z} = \bar{u} \wedge \bar{y} = \bar{t}))$  est transformée en une disjonction de  $16m^2$  clauses conjonctives traduisant les différentes façons de satisfaire la formule, comme par exemple :

$$\begin{aligned} & x_i \wedge \neg u_i \wedge y_j \wedge \neg t_j \\ & \neg x_i \wedge u_i \wedge y_j \wedge \neg t_j \end{aligned}$$

$z_i \wedge \neg t_i \wedge y_j \wedge \neg t_j$   
etc.

La taille de la formule croît ainsi d'un facteur additif polynomial en fonction de  $m$ . Pour terminer, la formule recherchée est la formule close  $\varphi_m[\{\bar{x} \leftarrow \overline{init}\} \cup \{\bar{y} \leftarrow \overline{fin}\}]$ .  $\overline{init}$  décrit la configuration initiale et  $\overline{fin}$  décrit la configuration finale (sans perte de généralité, on peut supposer qu'il n'en existe qu'une seule).

*c.q.f.d.*  $\diamond\diamond$

Le corollaire nous ramène à une variante d'un problème connu.

**Corollaire 3 (QSAT)** *Soit  $\varphi$  une formule booléenne quantifiée close dont la sous-formule propositionnelle est sous forme normale conjonctive. Alors le problème de la satisfaisabilité de  $\varphi$  est PSPACE-complet.*

**Preuve**

Soit un problème  $Pb$  dans PSPACE, le problème complémentaire  $Pbc$  est dans PSPACE et se réduit à l'évaluation d'une formule QBF sous forme normale disjonctive  $\varphi$ . Par conséquent tout problème dans PSPACE se réduit à l'évaluation de  $\neg\varphi$  où  $\varphi$  est une formule QBF sous forme normale disjonctive obtenue par les transformations standard :

- $\neg\forall\theta \equiv \exists\neg\theta$ ,  $\neg\exists\theta \equiv \forall\neg\theta$
- $\neg(\Upsilon \vee \theta) \equiv \neg\Upsilon \wedge \neg\theta$ ,  $\neg(\Upsilon \wedge \theta) \equiv \neg\Upsilon \vee \neg\theta$
- $\neg\neg\theta \equiv \theta$

La formule  $\neg\varphi$  est équivalente à une formule QBF sous forme normale conjonctive d'une taille égale au plus au double de la taille de  $\varphi$ .

*c.q.f.d.*  $\diamond\diamond$

### 3.2.2 Universalité des langages réguliers

Tester la vacuité d'un langage régulier  $L = \emptyset$  ? (où  $L$  est donné par un automate ou une expression régulière) se fait en temps polynomial car ce problème se réduit à un problème d'accessibilité dans un graphe (e.g. existe-t-il un chemin de l'état initial vers un état final ?). Tester l'universalité  $L = \Sigma^*$  ? n'est pas *a priori* un problème équivalent car le passage à l'automate complémentaire entraîne un facteur d'accroissement exponentiel. Nous allons donc étudier ce problème. Dans notre étude, nous utiliserons indifféremment la spécification du langage régulier par un automate non déterministe sans  $\varepsilon$ -transition ou par une expression rationnelle car il existe des traductions en temps polynomial d'une représentation vers l'autre.

**Proposition 13** *Soit  $\mathcal{A}$  un automate non déterministe sur un alphabet  $\Sigma$  sans  $\varepsilon$ -transition et  $L(\mathcal{A})$  son langage. Alors le problème de l'universalité de  $L(\mathcal{A})$  est dans PSPACE.*

**Preuve**

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  $\Sigma$  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 état représentés en  $O(n)$ ). Il suffit alors de la déterminer par la procédure de Savitch.

*c.q.f.d.*  $\diamond\diamond\diamond$

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

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

**Preuve**

Soit une machine de Turing déterministe  $\mathcal{M} = (Q, T, A, \delta, \flat, q_0, q_f)$  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,  $A \subseteq T \setminus \{\flat\}$  l'alphabet des entrées,  $\flat \in T$  le blanc,  $q_0, q_f$  les états initial et final.

Nous associons un mot à chaque exécution de la machine. L'alphabet de ce mot est  $\Sigma \equiv T \cup \{qX \mid q \in Q \wedge X \in T\} \cup \{\#\}$ . On note  $\Delta = \Sigma \setminus \{\#\}$  et  $QT = \{qX \mid q \in Q \wedge X \in T\}$ . Le mot s'écrit  $w = \#w_1\#w_2\#\dots\#w_n\#$  pour une exécution de longueur  $n$  avec  $w_i$  la représentation de la  $i$ ème configuration. Chaque mot  $w_i$  a une longueur  $p(n)$  et représente le contenu de la bande avec un unique symbole  $qX$  signalant à la fois l'état de la machine et la position de la machine.

Nous allons construire une expression régulière  $E$  qui accepte les mots  $w$  qui ne sont pas des codages d'exécution ou ceux qui codent des exécutions qui ne rencontrent pas l'état  $q_f$ . Autrement dit,  $E = \Sigma^*$  ssi  $x$  n'est pas accepté par  $\mathcal{M}$ . Enumérons les différents cas possibles.

1.  $w$  ne contient pas un symbole  $q_f X$ . Appelons  $A$  ce langage.
2.  $w$  ne contient pas la configuration initiale.  
Autrement dit,  $\#(q_0 x_1) x_2 \dots x_n \flat \dots \flat$  où  $x = x_1 \dots x_n$  et  $\flat$  apparaît  $p(n) - n$  fois, n'est pas un préfixe de  $w$ . Appelons  $B$  ce langage.
3.  $w$  n'est pas de la forme  $w = \#w_1\#w_2\#\dots\#w_n\#$  avec pour tout  $i$ ,  $|w_i| = p(n)$ , une lettre de  $w_i$  est de la forme  $qX$  et toutes les autres lettres de  $w_i$  appartiennent à  $T$ . Appelons  $C$  ce langage.
4.  $w$  contient deux configurations successives qui ne correspondent pas à un pas de la machine. Appelons  $D$  ce langage.

Dans la suite, si  $S = \{s_1, \dots, s_k\}$  est un sous-ensemble de lettres, on utilise  $S$  comme abréviation de l'expression rationnelle  $s_1 + \dots + s_k$  et  $E^i$  comme abréviation de  $E.E.\dots.E$  où  $E$  apparaît  $i$  fois,  $E^0$  étant  $\varepsilon$ . Soit l'expression  $E_1 \equiv (\Sigma \setminus \{q_f X \mid X \in T\})^*$ , alors  $L(E_1) = A$ .

Soit :

- $E_{2,1} = \Delta.\Sigma^*$
- $E_{2,2} = \Sigma.(\Sigma \setminus \{q_0x_1\}).\Sigma^*$
- $\forall 2 \leq i \leq n \ E_{2,i} = \Sigma^{i+1}.(\Sigma \setminus \{x_i\}).\Sigma^*$
- $\forall n+2 \leq i \leq p(n)+1 \ E_{2,i} = \Sigma^i.(\Sigma \setminus \{b\}).\Sigma^*$

Posons  $E_2 = E_{2,1} + \dots + E_{2,p(n)+1} + \varepsilon + \Sigma + \dots + \Sigma^{p(n)}$ . Alors  $L(E_2) = B$ .

Soit :

- $\forall 0 \leq i \leq p(n)-1 \ E_{3,i} = \Sigma^*.\#.\Delta^i.\#.\Sigma^*$   
(mots avec des configurations trop courtes)
- $E_{3,p(n)+1} = \Sigma^*.\#\Delta^{p(n)+1}.\Delta^*.\#\Sigma^*$   
(mots avec des configurations trop longues)
- $F_3 = \Delta^* + \Delta^*.\#\Delta^* + \Delta.\Sigma^* + \Sigma^*.\Delta$   
(mots sans ou avec un seul  $\#$ , ne commençant ou ne finissant pas par  $\#$ )
- $G_3 = \Sigma^*.\#T^*.\#\Sigma^* + \Sigma^*.\#T^*.QT.T^*.QT.T^*.\#\Sigma^*$   
(mots avec des configurations ne contenant pas d'état ou contenant deux états)

Posons  $E_3 = E_{3,1} + \dots + E_{3,p(n)-1} + E_{3,p(n)+1} + F_3 + G_3$ . Alors  $L(E_3) = C$ .

Remarquons que si un mot code une exécution alors trois lettres consécutives  $a_{i-1}a_i a_{i+1}$  de ce mot appartenant à  $\Delta$ , déterminent de façon unique la lettre  $a_{p(n)+i+1}$  si elle existe. Appelons  $f : \Delta^3 \mapsto 2^\Sigma$ , la fonction qui associe à trois lettres de  $\Delta$ , le sous-ensemble de  $\Sigma$  privé de la lettre ainsi déterminée (où  $\Sigma$  lui-même dans le cas d'une suite de trois lettres contenant au moins deux lettres de  $QT$ ). Posons  $D_{c_1,c_2,c_3} = \Sigma^*.c_1.c_2.c_3.\Sigma^{p(n)}.f(c_1,c_2,c_3).\Sigma^*$  et  $E_4$  la somme des  $D_{c_1,c_2,c_3}$ . Alors  $L(E_4) = D$ .

Nous laissons le soin au lecteur de vérifier que  $E = E_1 + E_2 + E_3 + E_4$  est de taille polynomiale vis à vis de  $n$ .

c.q.f.d.  $\diamond\diamond\diamond$

### 3.3 TD 3

#### Exercice 1

On note  $I = \{1, \dots, n\}$ . 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  $x_1 = V \wedge x_3 = F$  Alors  $x_1 \leftarrow F; x_2 \leftarrow V$

- 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  $(V, F, F)$  et conduit à la configuration  $(F, V, F)$ .

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  $(s_1, \dots, s_n)$  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  $v_0$  ;
- Deux joueurs Alice et Bob qui jouent à tour de rôle en commençant par Alice ;

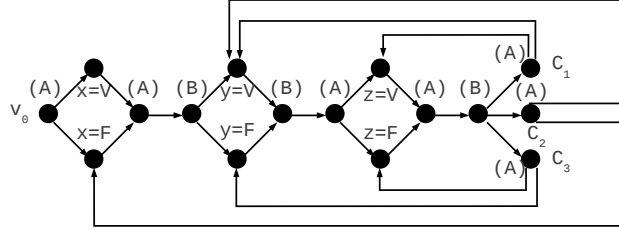


FIGURE 3.1: Réduction de  $QSAT$  vers Géographie

- 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 :

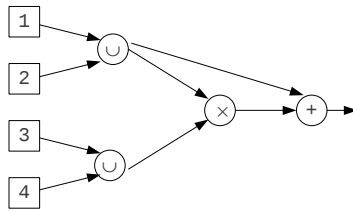
- les sommets sans prédécesseur sont appelés les *entrées* et sont étiquetés par un élément de  $\mathbb{N}$  (confondu avec le singleton contenant cet entier).
- Les autres sommets sont appelés des *portes* et sont étiquetés par l'une des trois opérations  $+$ ,  $\times$ ,  $\cup$ . Toutes les portes 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  $E(out) = \{j, 2j, 3j, \dots, 2^k j\}$  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

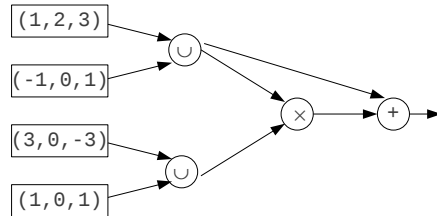
Un *circuit vectoriel*  $\mathcal{C}$  de dimension  $n$  est défini par un graphe orienté sans circuit dont :

- 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  $\psi$  est donnée sous forme DNF (i.e.  $\psi$  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  $\varphi$  une formule, la *table de vérité* de  $\varphi$ , notée  $\mathbf{T}(\varphi)$  est l'ensemble des vecteurs d'affectation correspondant aux affectations satisfaisant  $\varphi$ . 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  $p \in \mathbb{Z}$ ,  $p\vec{1} = p\vec{1}$ .

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

**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  $\varphi$  et montrez que la taille du circuit construit est polynomiale par rapport à  $n$ .

**Question 8.** Soit  $\varphi$  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  $\varphi$ , 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  $\theta$  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  $\varphi^k$ . Construisez à partir de  $\mathcal{C}'$ , un circuit  $\mathcal{C}$  qui  $(k+1)$ -représente  $\varphi^{k+1}$ .

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

**Question 11.** Dédurre 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  $m_1, \dots, m_n \in \mathbb{N}$  impairs et premiers entre eux et  $M = \prod_{i=1}^n m_i$ .  
On note  $V = [(-m_1 + 1)/2, (m_1 - 1)/2] \times \dots \times [(-m_n + 1)/2, (m_n - 1)/2]$ .  
Alors il existe  $z_1, \dots, z_n \in \mathbb{Z}$  calculables en temps polynomial tels que :
  - $h : V \mapsto [0, \dots, M - 1]$  définie par :  
$$h(x_1, \dots, x_n) = \sum_{i=1}^n z_i x_i \pmod{M}$$
 est bijective.
  - Pour tout  $\vec{v}, \vec{w} \in V$ , si  $\vec{v} + \vec{w} \in V$  alors  $h(\vec{v} + \vec{w}) = h(\vec{v}) + h(\vec{w}) \pmod{M}$
  - Pour tout  $\vec{v}, \vec{w} \in V$ , si  $\vec{v} \times \vec{w} \in V$  alors  $h(\vec{v} \times \vec{w}) = h(\vec{v})h(\vec{w}) \pmod{M}$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  $\exists x \in E(out') \ x \pmod{M} = h(\vec{v})$ .

**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  $\exists x \in E(out'') \ x \pmod{M} = 0$ .

**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 conjonction de clauses de Horn

Une clause de Horn est une clause dans laquelle il y a au plus un littéral positif. Une formule de Horn est une conjonction de clauses de Horn. Les clauses de Horn s'expriment de manière alternative à l'aide du connecteur  $\Rightarrow$  ; une clause de Horn est :

- soit  $(\bigwedge_{i \in I} p_i) \Rightarrow q$  où les  $p_i$  sont les hypothèses,  $q$  est la conclusion ( $I$  peut être vide) ;
- soit  $(\bigwedge_{i \in I} p_i) \Rightarrow \mathbf{false}$ .

**Proposition 15** *Soit  $\varphi$  une formule de Horn. Alors le problème de la satisfaisabilité de  $\varphi$  est dans PTIME.*

#### Preuve

L'algorithme maintient une interprétation partielle  $\nu$  (initialement l'interprétation vide) pour laquelle toutes les interprétations des propositions ont pour valeur **true** et un ensemble de clauses actives (initialement toutes les clauses). De plus ces interprétations sont nécessaires afin de satisfaire  $\varphi$ .

L'algorithme associe à chaque clause active  $cl$  le nombre d'hypothèses non incluses dans l'interprétation courante  $nb_{cl}$ . Une itération de l'algorithme consiste à parcourir les clauses actives et pour chaque clause active  $cl$  dont  $nb_{cl} = 0$  à désactiver la clause puis

- si  $cl$  a une conclusion  $q$  telle que  $\nu$  n'est pas définie pour  $q$ , à enrichir l'interprétation  $\nu$  avec  $q$  et à décrémenter  $nb_{cl'}$  pour les clauses  $cl'$  dont  $q$  est une hypothèse.
- sinon à s'arrêter en détectant que  $\varphi$  n'est pas satisfaisable.

S'il n'y a plus de clauses actives  $cl$  avec  $nb_{cl} = 0$ , il suffit de compléter l'interprétation  $\nu$  en interprétant toutes les propositions non encore définies à **false** pour obtenir une interprétation satisfaisant  $\varphi$ .

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

*c.q.f.d.*  $\diamond\diamond$

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

**Proposition 16** *Soit  $\varphi$  une formule de Horn dont chaque clause a au plus 3 littéraux. Alors le problème de la satisfaisabilité de  $\varphi$  est PTIME-difficile (donc PTIME-complet).*

**Preuve**

Soit une machine de Turing déterministe  $\mathcal{M} = (Q, T, A, \delta, b, q\theta, qf)$  qui s'exécute en temps polynomial vis à vis de son entrée  $x$ , disons  $p(n) \geq n$  où  $p$  est un polynôme et  $n$  est la taille de  $x$ .  $Q$  est l'ensemble des états,  $T$  les symboles de la bande,  $A \subseteq T \setminus \{b\}$  l'alphabet des entrées,  $b \in T$  le blanc,  $q\theta, qf$  les états initial et final. Par conséquent, la bande de la machine de Turin utilise au plus  $p(n)$  cellules. On suppose de plus qu'une fois arrivée dans l'état  $qf$ , la machine reste indéfiniment dans cet état.

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  $i$ .

- $q_{i,j}$  signifie que la tête de lecture est à l'instant  $j$  au dessus de la cellule  $i$  et que la machine est dans l'état  $q$ ;
- Pour tout  $a \in T$ ,  $a_{i,j}$  signifie qu'à l'instant  $j$  le contenu de la cellule  $i$  est  $a$ ;

Nous décrivons maintenant un ensemble de formules simultanément satisfaites ssi il existe un calcul de la machine (en temps  $p(n)$ ).

- Initialement la machine est dans l'état  $q\theta$  et la tête se trouve au dessus de la cellule 0. D'où les formules  $q\theta_{0,0}$ ;
- Initialement la bande contient le mot  $x$ . D'où les formules  $\$_{0,0}$ , pour tout  $1 \leq i \leq n$   $x(i)_{i,0}$  et pour tout  $n < i \leq p(n)$   $b_{i,0}$ ;
- La machine termine dans l'état  $qf$ . On fait sans perte de généralité, l'hypothèse qu'alors la tête se trouve au-dessus de la première cellule. D'où la formule  $qf_{0,p(n)}$ ;
- A tout instant la formule est dans un seul état et la tête est au dessus d'une unique cellule. D'où les formules, pour tout  $0 \leq j \leq p(n)$ , pour tout  $0 \leq i, i' \leq p(n)$ , pour tout  $q, q' \in Q$  avec  $(q, i) \neq (q', i')$   $\neg q_{i,j} \vee \neg q'_{i',j}$ .
- A tout instant une cellule contient une unique lettre. D'où les formules, pour tout  $0 \leq i, j \leq p(n)$ , pour tout  $a \neq a'$   $\neg a_{i,j} \vee \neg a'_{i,j}$ .
- A tout instant, si une cellule n'est pas sous la tête de la machine elle ne change pas de contenu à l'instant suivant.  $0 \leq j < p(n)$ , pour tout  $q \in Q$  et tout  $0 \leq i \neq i' \leq p(n)$ ,  $(q_{i',j} \wedge a_{i,j}) \Rightarrow a_{i,j+1}$ .
- Soit  $\delta(q, a) = (q', a', \pm)$  avec  $\pm \in \{+, -\}$ . Pour tout  $0 \leq j < p(n)$  et tout  $0 \leq i \leq p(n)$ , on a les formules :  
 $(q_{i,j} \wedge a_{i,j}) \Rightarrow q'_{i\pm 1, j+1}$  et  $(q_{i,j} \wedge a_{i,j}) \Rightarrow a'_{i, j+1}$ .

Le nombre de sous-formules est quadratique par rapport à  $p(n)$ , donc polynomial par rapport à  $n$ . Toutes les formules sont des clauses de Horn d'au plus trois littéraux.

Un calcul se traduit immédiatement par une interprétation qui satisfait la formule et vice versa ce qui achève la preuve.

*c.q.f.d.*  $\diamond\diamond$

## 4.2 Valeur d'un circuit

Un circuit  $\mathcal{C}$  est composé de portes de cinq types **false**, **true**,  $\wedge$ ,  $\vee$ ,  $\neg$ . Chaque porte a une sortie. Les portes **false** et **true** (qui sont uniques) n'ont pas d'entrées. Les portes  $\neg$  ont une seule entrée et les portes  $\wedge$  et  $\vee$  ont au moins deux entrées. Toute entrée d'une porte est connectée à une sortie d'une autre porte. Si on définit une relation suiv entre portes par  $a$  suiv  $b$  ssi une entrée de  $a$  est connectée à une sortie de  $b$  alors la fermeture transitive de suiv est irréflexive. Autrement dit, suiv définit un ordre partiel entre portes. L'une des portes du circuit notée **out** est distinguée.

La valeur des entrées et des sorties d'un circuit est définie inductivement suivant la relation suiv. La sortie de la porte **false** (resp. **true**) est **false** (resp. **true**). L'entrée d'une porte a pour valeur la valeur de la sortie à laquelle elle est connectée. La sortie d'une porte  $\wedge, \vee, \neg$  s'obtient par application de la table de vérité du connecteur associé.

Le problème de la valeur d'un circuit consiste à déterminer la valeur de la sortie de la porte **out**.

**Proposition 17** *Soit  $\mathcal{C}$  un circuit. Alors le problème de la valeur de  $\mathcal{C}$  est PTIME-complet.*

### Preuve

Montrons d'abord que ce problème est dans PTIME. On suppose que le circuit est donné par la suite des portes (en respectant l'ordre suiv). Chaque porte comporte son type, son nombre d'entrées et pour chaque entrée l'indice de la porte dont la sortie lui est associée. La dernière porte est la porte **out**.

L'évaluation consiste à maintenir un tableau des valeurs des portes déjà calculées. L'algorithme parcourt les portes, et durant une itération calcule la valeur de la sortie en fonction de la valeur des entrées. Le tout se fait en temps linéaire par rapport à la taille du circuit.

Soit une machine de Turing déterministe  $\mathcal{M} = (Q, T, A, \delta, b, q\theta, qf)$  qui s'exécute en temps polynomial vis à vis de son entrée  $x$ , disons  $p(n) \geq n$  où  $p$  est un polynôme et  $n$  est la taille de  $x$ .  $Q$  est l'ensemble des états,  $T$  les symboles de la bande,  $A \subseteq T \setminus \{b\}$  l'alphabet des entrées,  $b \in T$  le blanc,  $q\theta, qf$  les états initial et final. Par conséquent, la bande de la machine de Turing utilise au plus  $p(n)$  cellules. On suppose de plus qu'une fois arrivée dans l'état  $qf$ , la machine reste indéfiniment dans cet état.

Nous allons décrire l'exécution de la machine ainsi. On construit d'abord un circuit  $\mathcal{C}'$  qui ne dépend que de  $\mathcal{M}$  et pas du mot  $w$ . Par souci de lisibilité, on nommera les entrées et les sorties. Il y a trois fois plus d'entrées que de sorties car les entrées désignent l'état de trois cellules contiguës disons  $g, s, d$  en un instant du calcul et les sorties désignent l'état de la cellule  $s$  à l'instant suivant. Décrivons les entrées ( $i \in \{g, s, d\}$ ).

- $q_i^-$  signifie que la tête de lecture est à l'instant courant au dessus de la cellule  $i$  et que l'état courant est  $q$ ;
- Pour tout  $a \in T$ ,  $a_i^-$  signifie qu'à l'instant courant le contenu de la cellule  $i$  est  $a$ ;

Les sorties sont définies de manière similaire.

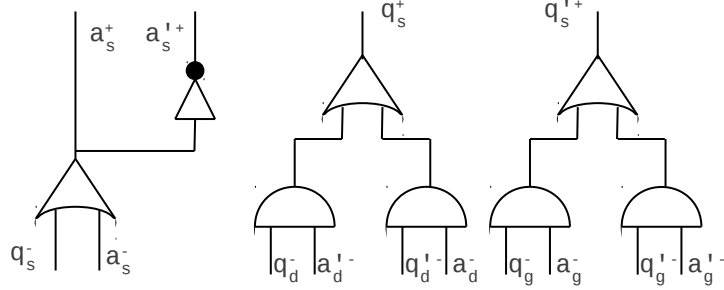


FIGURE 4.1: Le circuit associé à la fonction de transition

- $q_s^+$  signifie que la tête de lecture est à l'instant suivant au dessus de la cellule  $s$  et que l'état courant est  $q$  ;
- Pour tout  $a \in T$ ,  $a_s^+$  signifie qu'à l'instant suivant le contenu de la cellule  $s$  est  $a$  ;

La figure 4.1 représente le circuit  $\mathcal{C}'$  dans le cas d'une machine de Turing dont l'alphabet est  $\{a, a'\}$ , l'ensemble des états est  $\{q, q'\}$  et la fonction de transition est donnée par :

- $\delta(q, a) = (q', a, +)$ ,  $\delta(q, a') = (q, a, -)$
- $\delta(q', a) = (q, a, -)$ ,  $\delta(q', a') = (q', a', +)$

Pour les cellules 0 et  $p(n)$ , on construit des circuits similaires  $\mathcal{C}'_g$  et  $\mathcal{C}'_d$  qui ne dépendent que de deux « jeux » d'entrées. La construction des trois circuits se fait en temps constant.

On construit le circuit  $\mathcal{C}$  ainsi. Après les deux portes **false** et **true**, on construit une première « tranche » composée d'une copie du circuit  $\mathcal{C}'_g$ , de  $p(n) - 2$  copies du circuit  $\mathcal{C}'$  et d'une copie du circuit  $\mathcal{C}'_d$ . Cette tranche correspond à l'évolution entre l'instant 0 et l'instant 1. Les entrées de cette tranche sont connectées aux portes **false** et **true** selon le mot d'entrée  $w$ .

On construit ensuite  $p(n) - 1$  « tranches » consécutives en connectant les entrées de la tranche courante aux sorties de la tranche précédente. On termine par la porte **out** qui est une porte  $\vee$  et qui a pour entrées les sorties  $qf_s^+$  de la dernière tranche.

La correction de la réduction provient de la spécification des portes  $\mathcal{C}'$ ,  $\mathcal{C}'_g$  et  $\mathcal{C}'_d$ . Il reste à prouver que cette réduction se fait en  $O(\log(n))$  mais pour spécifier les sorties de la tranche précédente, il suffit de calculer l'identité de leur porte à partir des indices de la tranche courante et du sous-circuit courant soit des compteurs en  $O(\log(p(n))) = O(\log(n))$ .

*c.q.f.d.*  $\diamond\diamond$

On peut renforcer ce résultat. Un circuit est dit *monotone*, s'il ne comporte pas de portes  $\neg$ .

**Proposition 18** *Soit  $\mathcal{C}$  un circuit monotone. Alors le problème de la valeur de  $\mathcal{C}$  est PTIME-difficile (donc PTIME-complet).*

**Preuve**

Une première preuve possible consisterait à montrer qu'on peut se passer de porte  $\neg$  dans la définition des circuits  $\mathcal{C}'$ ,  $\mathcal{C}'_g$  et  $\mathcal{C}'_d$  de la preuve précédente. Nous allons établir une preuve alternative qui permet de démontrer qu'un circuit non monotone peut être réduit en espace logarithmique en un circuit monotone en doublant au plus sa taille.

Soit  $\mathcal{C}$  un circuit quelconque. On construit en espace logarithmique  $\mathcal{C}'$  un circuit qui a même valeur que  $\mathcal{C}$ . Pour cela on suppose que dans la représentation de  $\mathcal{C}$ , les portes d'un circuit ont un identifiant en  $O(\log(n))$  où  $n$  désigne le nombre de portes de  $\mathcal{C}$ . De plus la numérotation préserve la relation suiv. Ceci ne modifie pas l'ordre de grandeur de la représentation et facilite la description de la transformation.

Le principe de la transformation est relativement simple :

- Pour chaque porte de type  $\vee$  (resp.  $\wedge$ ) dans  $\mathcal{C}$  d'identité  $i$ , on construit une porte  $\vee$  (resp.  $\wedge$ ) d'identité  $2i$  et une porte  $\wedge$  (resp.  $\vee$ ) d'identité  $2i + 1$ . La sortie de la porte  $\vee$  (resp.  $\wedge$ ) aura même valeur que la porte originale tandis que la sortie de la porte  $\wedge$  (resp.  $\vee$ ) aura la valeur complémentaire.
- La construction globale repose sur une procédure **search(e, id, inverse)** qui étant donnée une entrée **e** d'une porte renvoie **id** l'identité d'une porte (située en amont) et un booléen **inverse** telle que (1) la porte **id** n'est pas une porte  $\neg$  et (2) si **inverse** est faux alors **e** a pour valeur la sortie de **id** sinon **e** a la valeur complémentaire. Pour ce faire, elle initialise **id** à la porte dont la sortie est connectée à **e** et **inverse** à faux. Puis elle itère le procédé suivant : tant que **id** est une porte  $\neg$  elle substitue à **id** la porte dont la sortie est connectée à l'entrée de **id** et complémente la valeur de **inverse**.

Nous décrivons maintenant la procédure dans le cas où la porte  $i$  est une porte  $\vee$ , l'autre cas étant analogue. La construction des portes d'identité  $2i$  et  $2i + 1$  se fait ainsi (les portes d'identité inférieure ont déjà été construites). Pour chaque entrée, **e** de la porte  $i$ , on appelle **search(e, id, inverse)**.

- Si **id** est l'identité de la porte **true** et **inverse** est faux (resp. vrai) alors l'entrée **e** de la porte  $2i$  est connectée à la sortie de la porte **true** (resp. **false**) et l'entrée **e** de la porte  $2i + 1$  est connectée à la sortie de la porte **false** (resp. **true**).
- Si **id** est l'identité de la porte **false** et **inverse** est faux (resp. vrai) alors l'entrée **e** de la porte  $2i$  est connectée à la sortie de la porte **false** (resp. **true**) et l'entrée **e** de la porte  $2i + 1$  est connectée à la sortie de la porte **true** (resp. **false**).
- Dans les autres cas, si **inverse** est faux (resp. vrai) alors l'entrée **e** de la porte  $2i$  est connectée à la sortie de la porte **2id** (resp. **2id + 1**) et l'entrée **e** de la porte  $2i + 1$  est connectée à la sortie de la porte **2id + 1** (resp. **2id**).

La correction de cette construction repose sur les équivalences  $\neg \bigwedge_i \varphi_i \equiv \bigvee_i \neg \varphi_i$  et  $\neg \bigvee_i \varphi_i \equiv \bigwedge_i \neg \varphi_i$ .

Il reste à déterminer la porte de sortie du circuit  $\mathcal{C}'$ . Si la porte *out* n'est pas une porte  $\neg$  alors la sortie est la porte  $2out$ . Sinon on appelle **search(e, id, inverse)** avec **e** l'entrée de *out*.

- Si **id** est l'identité de la porte **true** et **inverse** est vrai (resp. faux) alors  $out'$  est la porte **true** (resp. **false**).
- Si **id** est l'identité de la porte **false** et **inverse** est vrai (resp. faux) alors  $out'$  est la porte **false** (resp. **true**).
- Dans les autres cas, si **inverse** est vrai (resp. faux) alors  $out'$  est la porte **2id** (resp. **2id + 1**).

*c.q.f.d.*  $\diamond\diamond$

### 4.3 Accessibilité dans un graphe *non déterministe*

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 19** *Le problème de la clôture est en PTIME.*

**Preuve**

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

*c.q.f.d.*  $\diamond\diamond$

**Proposition 20** *Le problème de la clôture même dans le cas d'une loi commutative est PTIME-difficile.*

**Preuve**

On réduit le problème de satisfaisabilité de  $\varphi$ , une formule de Horn avec au plus 3 littéraux par clause au problème de la clôture. Les éléments de l'ensemble sont les clauses incluses dans une clause de  $\varphi$ , soient au plus 8 clauses par clause originelle et un élément spécial  $\$$ . La loi commutative est donnée par les règles suivantes :

- Si  $u = p$  et  $v = \neg p \vee \psi$  alors  $u \bullet v = \psi$  (avec la convention que si  $v = \neg p$  alors  $\psi$  est la clause vide).
- Dans les autres cas,  $u \bullet v = \$$

$g$  est la clause vide et  $H$  l'ensemble des clauses de  $\varphi$ . La clause vide appartient à  $H$  ssi  $\varphi$  n'est pas satisfaisable. Pour s'en convaincre il suffit d'examiner la preuve que le problème de satisfaisabilité d'une formule de Horn est en PTIME.

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.*  $\diamond\diamond$

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  $\{v, w\} \subseteq Z$  alors  $u \in Z$ . On dit que  $W$  est accessible depuis  $u \in V$  si  $u \in Acc(W)$ .

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

**Preuve**

L'algorithme 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.*  $\diamond\diamond$

**Proposition 22** Le problème de l'accessibilité dans un graphe non déterministe est PTIME-difficile.

**Preuve**

Le problème de cloture 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.*  $\diamond\diamond$

## 4.4 Test de la vacuité d'un langage algébrique

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

- Si  $T \rightarrow w$  est une règle avec  $w \in \Sigma^*$  alors  $w \in L(G, T)$ ;
- Si  $T \rightarrow w_0 T_1 w_1 \dots T_n w_n$  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 est alors  $L(G, S)$ .

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

**Preuve**

L'algorithme consiste à calculer par saturation l'ensemble  $Prod(G)$  des symboles non terminaux tels que  $L(G, T) \neq \emptyset$  puis à vérifier que  $S \in Prod(G)$ . On initialise  $Prod(G)$  par les symboles  $T$  tels qu'il existe une règle  $T \rightarrow w$  avec  $w \in \Sigma^*$ . Puis on itère le procédé suivant : on parcourt les règles de  $G$  et pour chaque règle  $T \rightarrow w_0 T_1 w_1 \dots T_n w_n$  avec pour tout  $i$ ,  $T_i \in Prod(G)$  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.*  $\diamond\diamond$

**Proposition 24** *Le problème de la vacuité d'un langage algébrique. est PTIME-difficile.*

**Preuve**

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

- L'alphabet non terminal est l'ensemble  $G$ . L'axiome est  $g$ . L'alphabet terminal est un singleton.
- Les règles de production sont définies par  $\{u \rightarrow vw \mid u = v \bullet w\} \cup \{u \rightarrow \varepsilon \mid u \in H\}$ .

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

*c.q.f.d.*  $\diamond\diamond$

## 4.5 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  $M = (S, \rightarrow, L)$ , où :

- $S$  est un ensemble fini d'états,
- $\rightarrow \subseteq S \times S$  est une fonction de transition telle que  $\forall s \in S \exists s' \in S \ s \rightarrow s'$ ,
- $L : S \rightarrow 2^{AP}$  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  $i \geq 0$ ,  $s_i \rightarrow s_{i+1}$ .

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

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

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) :

- $M, s \models \top$
- $M, s \not\models \perp$
- $M, s \models p$  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  $\forall s' \in S \ s \rightarrow s' \implies M, s' \models \phi$
- $M, s \models EX\phi$  ssi  $\exists s' \in S \ s \rightarrow s' \wedge M, s' \models \phi$
- $M, s \models AF\phi$  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  $s_0 \rightarrow s_1 \rightarrow \dots$  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  $s_0 \rightarrow s_1 \rightarrow \dots$  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  $\phi$ . 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  $\{F_1, \dots, F_n\}$  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  $G = (V, A)$  et deux sommets  $s$  et  $t$ , le problème de l'accessibilité consiste à décider s'il existe un chemin de  $s$  à  $t$ .

**Proposition 25** *Le problème de l'accessibilité est en 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).

Le procédé non déterministe maintient deux variables : un compteur initialisé à  $n$  et un sommet courant initialisé à  $s$ . A chaque étape, elle choisit de manière non déterministe un successeur au sommet courant et décrémente le compteur. Si un sommet courant est  $t$  alors elle renvoie **true**. Si le sommet courant n'a pas de successeur ou si le compteur atteint 0 elle renvoie **false**.

Les deux variables sont représentables en  $O(\log(n))$  bits ce qui achève la démonstration.

*c.q.f.d.*  $\diamond\diamond$

**Proposition 26** *Le problème de l'accessibilité est NLOGSPACE-difficile.*

#### Preuve

Soit  $\mathcal{M}$  une machine de Turing non déterministe opérant en espace  $c\lceil\log(n)\rceil$  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  $c'\lceil\log(n)\rceil$  bits pour coder une configuration ( $c' \geq c$  ne dépend que de  $\mathcal{M}$ ).
- Il y a un arc d'une configuration  $cf$  à une autre configuration  $cf'$ , si en partant de  $cf$  il existe un pas de  $\mathcal{M}$  qui conduit à  $cf'$ .

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.*  $\diamond\diamond$

## 5.2 Le théorème d'Immerman-Szelepcényi

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 calcul (ou sur une bande spécifique) et le résultat doit être le même quelque soit la configuration acceptante. De plus il existe au moins une configuration acceptante.

**Lemme 3** *Soit  $\mathcal{M}$  une machine de Turing non déterministe s'exécutant en un espace de taille  $s(n) \geq \log(n)$  constructible<sup>1</sup> 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

Avant tout, la machine  $\mathcal{M}'$  calcule  $s(n)$ .

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  $N_d$  (voir l'algorithme 1). Supposons que  $\mathcal{M}'$  ait une exécution qui a calculé  $N_d$ , elle continue son calcul de la façon suivante :

- Elle mémorise  $N$  dans *OldN*. Puis elle initialise  $N$  à 0. Elle énumère ensuite les configurations occupant une place  $s(n)$ .
- Pour chaque configuration, disons *current*, elle initialise un compteur, disons *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, disons *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. 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.
- À la fin de l'énumération la plus externe  $N$  est égal à  $N_{d+1}$  puisque les 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

---

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

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.*  $\diamond\diamond$

---

**Algorithme 1:** L'algorithme d'Immerman-Szelepcényi

---

```

 $m \leftarrow s(n)$ 
 $d \leftarrow 0$ 
 $OldN \leftarrow 0; N \leftarrow 1$ 
while  $N > OldN$  do
     $OldN \leftarrow N; N \leftarrow 0$ 
    for  $current \in Conf(m)$  do
         $acc \leftarrow \text{false}; cpt \leftarrow 0$ 
        for  $local \in Conf(m)$  do
            Deviner un chemin  $\sigma$  de init à local en au plus  $d$  pas
            if  $\sigma$  existe then
                 $cpt \leftarrow cpt + 1$ 
                if  $local = current$  or  $local \rightarrow_{\mathcal{M}} current$  then
                     $N \leftarrow N + 1; acc \leftarrow \text{true}; \text{break}$ 
                end
            end
        end
        end
        if not  $acc$  and  $cpt < OldN$  then reject
    end
     $d \leftarrow d + 1$ 
end
return  $N$ 

```

---

**Théorème 6 (Immerman-Szelepcényi)** *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 3. La seule différence réside dans le fait que lorsqu'elle trouve une configuration acceptante de  $\mathcal{M}$ , elle s'arrête et rejette. 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.*  $\diamond\diamond$

Le corollaire le plus important est relatif à NLOGSPACE.

**Corollaire 4**  $\text{coNLOGSPACE} = \text{NLOGSPACE}$

# 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

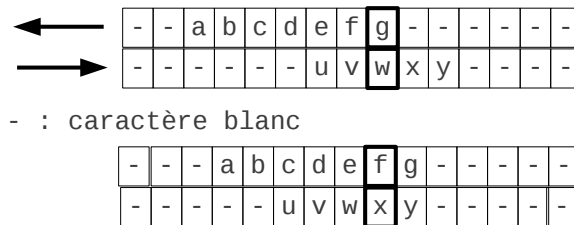


FIGURE 6.1: Représentation compacte et simulation de plusieurs bandes

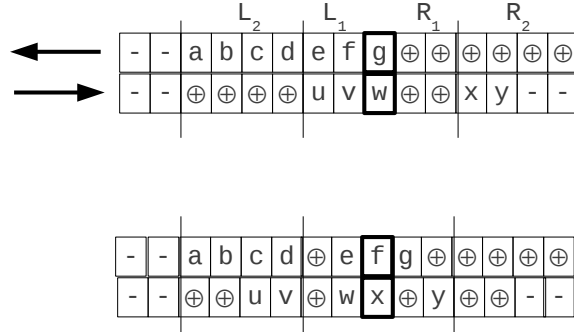


FIGURE 6.2: Représentation élargie et simulation de plusieurs bandes

d'abord si l'alphabet de  $\mathcal{M}$  est  $\Sigma$  (avec  $|\Sigma| \geq 2$ ) et si  $k$  est le nombre de bandes de travail de  $\mathcal{M}$ , on considère l'alphabet  $\Sigma^k$ . Chaque lettre de cet alphabet est alors associée 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  $O(T)$  (puisque'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 élargie des bandes à l'aide d'un caractère supplémentaire  $\oplus$  différent du blanc. Dans cette représentation la bande est implicitement partitionnée en segments  $\dots, L_i, \dots, L_1, [0, 0], R_1, \dots, R_i, \dots$  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.

- $U$  cherche le plus petit  $R_i$  non vide (et donc  $L_i$  n'est pas plein).
- $U$  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  $R_1, R_2, \dots, R_{i-1}$  en remplissant à moitié ces segments.

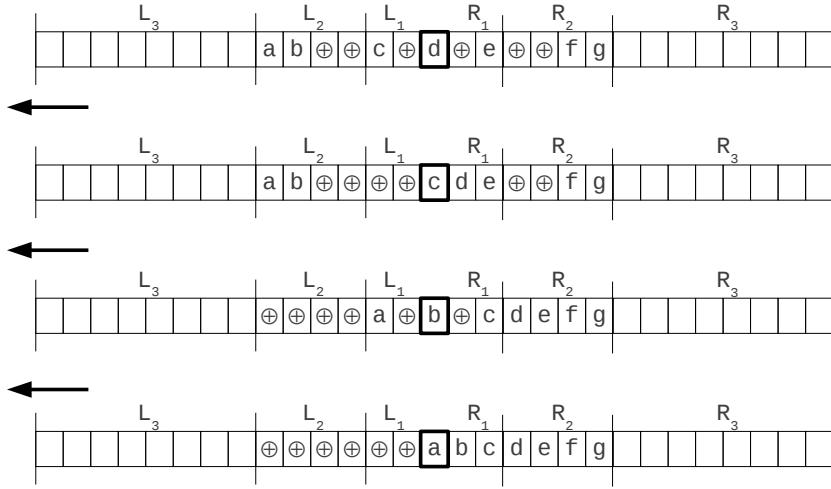


FIGURE 6.3: Déplacement simulé d'une bande « équilibrée »

- A 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  $L_{i-1}, \dots, L_1$ .  $\mathcal{U}$  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  $L_{i-1}, \dots, L_1$  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 moins  $2^{i-1}$  déplacements de la tête dans une direction car les segments  $R_1, R_2, \dots, R_{i-1}$  et  $L_{i-1}, \dots, L_1$  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. Un déplacement de  $L_i$  ou de  $R_i$  se fait en  $O(2^i)$ . Par conséquent, le temps global de la simulation est :

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

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.*  $\diamond\diamond$

Grace 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 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  $1, 2, 4, \dots$  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.*  $\diamond\diamond$

**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.*  $\diamond\diamond$

## 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 :

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

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  $\liminf_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0$  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.*  $\diamond\diamond$

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

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

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

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 espace  $f(n)$ . D'après le théorème 7 la simulation de  $\mathcal{M}$  requiert un temps  $O(f(n) \log(f(n)))$ . Sachant que  $\liminf_{n \rightarrow \infty} \frac{f(n) \log(f(n))}{g(n)} = 0$  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  $\lim_{n \rightarrow \infty} \frac{g(n)}{n} = \infty$ , 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.*  $\diamond\diamond\diamond$

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 :

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

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  $u_{k+1} = f(u_k + 1)2^{f(u_k+1)}$ .

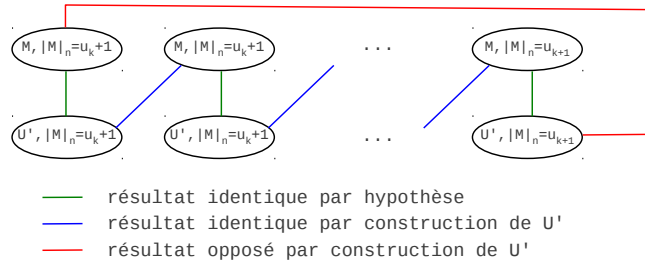
Nous construisons une machine  $\mathcal{U}'$  qui est une variante de la machine universelle non déterministe du théorème 8.  $\mathcal{U}'$  a une bande supplémentaire dite bande de compteur pour stocker un compteur. Soit  $x$  une entrée de taille  $n$ ,  $\mathcal{U}'$  commence par calculer  $g(n)$  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  $|x| < u_{k+1}$ ,  $\mathcal{U}'$  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  $\lim_{n \rightarrow \infty} \frac{f(n+1)}{g(n)} = 0$  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  $O(f(u_k + 1)2^{f(u_k+1)}) = O(u_{k+1}) = O(f(u_{k+1} + 1))$ .

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  $\lim_{n \rightarrow \infty} \frac{g(n)}{n} = \infty$ , 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.  $\diamond\diamond$