Université de Montréal

Programmes de branchement catalytiques : algorithmes et applications

par

Hugo Côté

Département d'informatique et de recherche opérationnelle
Faculté des arts et des sciences

Mémoire présenté à la Faculté des études supérieures et postdoctorales
en vue de l'obtention du grade de
Maître ès sciences (M.Sc.)
en Informatique

Orientation Informatique théorique et quantique

Août 2018

Université de Montréal

Faculté des études supérieures et postdoctorales

Ce mémoire intitulé

Programmes de branchement catalytiques : algorithmes et applications

présenté par

Hugo Côté

a été évalué par un jury composé des personnes suivantes :

Marc Feeley
(président-rapporteur)

Pierre McKenzie
(directeur de recherche)

Louis Salvail
(membre du jury)

Mémoire accepté le :
4 décembre 2018

Sommaire

Le présent mémoire étudie le modèle de calcul des programmes de branchement k-catalytiques. Un programme de branchement k-catalytique sert à effectuer le calcul de k fonctions booléennes sur une même entrée et représente la contrepartie non uniforme de la notion de machine de Turing catalytique introduite par Buhrman et al. (STOC, 2014). On montre que pour toute fonction booléenne symétrique f il existe un programme de branchement catalytique qui fait le calcul de f un nombre exponentiel de fois avec une taille $O(n \lg n)$ par copie. On montre un résultat similaire reliant la complexité de n'importe quelle fonction booléenne f sur n bits se décomposant sous la forme $f : \{0,1\}^n \xrightarrow{g} G \xrightarrow{h} \{0,1\}$, où G est un groupe, à celle de g et de h. On introduit également un ensemble de notions dont celles de gadgets, d'analyseurs et de programmes sur des groupes permettant de montrer comment le calcul catalytique non uniforme peut aller au-delà du calcul transparent. Enfin, le nouveau formalisme introduit révèle le lien jusqu'à maintenant inexploré entre le calcul catalytique et la complexité de la communication.

Mots clés : complexité de la communication ; programme de branchement catalytique ; calcul transparent ; composition de fonction ; gadget ; analyseur ; programme sur un groupe ; fonction symétrique ; produit scalaire.

Summary

The present master's thesis studies the computational model of k-catalytic branching programs. A k-catalytic branching program is used to carry out the computation of k boolean functions on the same input and represents the nonuniform counterpart to the notion of catalytic Turing machine introduced by Buhrman et al. (STOC, 2014). It is shown that for any boolean symmetric function f there exists a k-catalytic branching program which computes f an exponential number of times and which has size $O(n \lg n)$ per copy. We show a similar result linking the complexity of any boolean function f on n bits expressible as $f : \{0,1\}^n \xrightarrow{g} G \xrightarrow{h} \{0,1\}$, where G is a group, to the complexity of g and h. Furthermore, we introduce a set of notions such as gadgets, analyzers and programs over groups allowing us to show how nonuniform catalytic computation can go beyond transparent computation. Lastly, the new formalism introduced reveals the link until now unexplored between catalytic computation and communication complexity.

Keywords : dag-like communication ; communication complexity ; catalytic branching program ; tranparent computation ; function composition ; gadget ; analyzer ; program over a group ; symmetric function ; scalar product.

Table des matières

Sommaire iii

Summary iv

Table des figures viii

Glossaire ix

Acronymes x

Introduction 2

0.1. Organisation du mémoire 4

Chapitre 1. Préliminaires 5

1.1. Non uniformité 5

1.2. Quelques classes de complexité 6

1.3. Calcul catalytique 6

1.3.1. Calcul catalytique déterministe 6

1.3.2. Calcul catalytique non déterministe 8

1.3.3. Calcul transparent 9

1.4. Programme de branchement 10

1.5. Bornes connues sur le calcul de fonctions 13

1.5.1. Bornes connues sur le calcul des fonctions symétriques 14

Chapitre 2. Une idée d'algorithme catalytique 16

Chapitre 3. Gadget et analyseur 18

3.1. Gadget 18

3.2. Analyseur 20

3.2.1. Interprétation des analyseurs en terme de protocole de communication .. 21

3.3. Programme de branchement catalytique à partir d'un gadget et d'un analyseur 22

3.4. Construire un gadget à partir d'un groupe 25

3.4.1. Interprétation d'un analyseur sur un G-gadget 28

3.4.2. Fonction calculée par un PBC construit à partir d'un analyseur sur un G-gadget 29

3.5. Gadget pour le calcul transparent 30

Chapitre 4. Applications 33

4.1. Calcul du produit scalaire 33

4.2. Approximations de fonctions seuils 34

4.3. Test d'appartenance à un sous-groupe 38

4.3.1. Application possible 39

4.4. Composition 40

4.4.1. Composition de fonctions 40

4.4.2. Composition de programmes de branchement 40

4.4.3. Calcul de la composition de fonctions 41

4.4.4. Intérêt de la composition itérée de fonctions 42

4.4.5. Retour sur le calcul de la composition de fonctions 43

4.5. Algorithme général pour le calcul de fonctions booléennes 43

4.5.1. Analyseur pour fonctions élémentaires 43

4.5.2. Quelques bornes supérieures sur le calcul d'une fonction arbitraire 43
4.5.2.1. Énoncés 44
4.5.2.2. Démonstration du théorème 4.5.4 44

Chapitre 5. Une borne inférieure sur la taille des analyseurs 48

Conclusion 51

5.1. Joindre des analyseurs efficaces au calcul transparent 52

5.2. Séparer CNL et CL 53

Bibliographie 54

Annexe A. Figures ..... A-i

Annexe B. Notions de complexité de la communication ..... B-i

Table des figures

1 Exemple de programme de branchement calculant la valeur de vérité de la proposition "$x_3 \implies (x_1 \implies x_2)$". ..... A-i

2 Un gadget dont l'inverse n'est pas un gadget. ..... A-ii

3 $\mathbb{Z}_5$-PB de l'exemple 3.1.2 où les arcs étiquetés "$x_j = 0$" et "$x_j = 1$" pointent vers le bas et sont représentés respectivement en bleu et en rouge et où les noeuds de l'étage j regardent le bit x_j. ..... A-ii

4 $\mathbb{Z}_n$-gadget du théorème 4.1.3 où les arcs étiquetés "$x_j = 0$" et "$x_j = 1$" sont représentés respectivement en bleu et en rouge et où les noeuds de l'étage j regardent le bit x_j. Les arcs pointent vers le bas. ..... A-iii

5 Exemple des différentes composantes d'un programme de branchement catalytique construit à partir d'un gadget et d'un analyseur. ..... A-iv

6 Représentation graphique de substitutions. ..... A-v

7 Construction en deux étapes d'un analyseur vérifiant que $p \in \{t_s+s, ..., k-1+s\}$ en prenant $f(s) = t_s + s$ et $g(s) = k - 1 + s$. ..... A-vi

8 Analyseur D_j^σ de la définition 4.5.12 page 46. ..... A-vii

9 Analyseur du théorème 4.2.1 page 34. ..... A-vii

Glossaire

---

analyseur: . 19

bande auxiliaire: . 9

gadget: . 17

G-gadget: . 25

indolent: traduction de *oblivious* . 7

machine catalytique: . 9

machine catalytique non déterministe: . 10

machine de Turing non uniforme: . 7

M-gadget: . 25

produit scalaire modulaire avec seuil: . 13

profondeur: . 39

programme sur G: . 24

semi-gadget: . 17

stratifié: traduction de *leveled* . 7

Acronymes

TAD: TAD(f) dénote la taille du plus petit analyseur déterministe pour f. 45

TAN: TAN(f) dénote la taille du plus petit analyseur non déterministe pour f. 45

TAP: TAP(f) dénote la taille du plus petit analyseur probabiliste pour f. 45

AFD: automate fini déterministe. 15

DAG: graphe orienté acyclique, traduit de *directed acyclic graph*. 17

E-PB: E-programme de branchement. 17

F_n: ensemble des fonctions booléennes sur n bits. 5

PBC: programme de branchement catalytique. 6

NPB: programme de branchement non déterministe. 5

NPBC: programme de branchement non déterministe catalytique. 7

PB: programme de branchement déterministe. 5

PBP: programme de branchement probabiliste. 6

PBPC: programme de branchement probabiliste catalytique. 7

S_f: complexité non uniforme en espace de f. 7

w**-PB:** programme de branchement de largeur w. 24

w**-PBP:** programme de branchement à permutation de largeur w. 24

Introduction

Buhrman et al. (STOC, 2014) [**5**] introduisent la notion de calcul catalytique dont l'objet d'étude est la question suivante : comment se servir temporairement d'un espace mémoire occupé, avec la garantie d'en rétablir le contenu par la suite, de manière à contribuer à la réalisation d'une opération requérant plus d'espace mémoire libre que présentement disponible ? Par analogie avec la notion de catalyseur en chimie, la mémoire occupée ainsi utilisée et par extension l'algorithme même en faisant usage sont dits catalytiques. Le même article introduit également un type de calcul, appelé calcul transparent, permettant de faire usage d'une mémoire catalytique et donne des exemples où une mémoire catalytique peut être utilisée pour rendre possibles des opérations pour lesquelles l'espace libre disponible seul n'aurait pas suffi selon les connaissances algorithmiques actuelles.

Girard et al. (2015) [**8**] introduisent la notion de programme de branchement k-catalytique qui est la contrepartie non uniforme du modèle de [**5**]. Les programmes de branchement k-catalytiques permettent d'étendre le modèle des programmes de branchement au calcul simultané de k fonctions f_i sur la même entrée (en particulier, au calcul simultané de k copies de la même fonction f sur la même entrée). Dans ce modèle, chacune des fonctions f_i représente le calcul effectué par l'algorithme catalytique lorsque son espace mémoire se trouve initialement dans une de ses différentes configurations possibles. Dans ce modèle également, la question du calcul catalytique prend la forme suivante : pour une fonction f donnée, quelle est la taille minimum d'un programme de branchement k-catalytique calculant k copies de f sur la même entrée ?

Potechin (CCC, 2017) [**16**] montre que pour toute fonction booléenne sur n bits f_n, la borne optimale [1] de taille $O(n)$ par copie de f_n peut être atteinte par un programme de branchement k-catalytique pour $k = 2^{2^n}$. Cela répond alors par l'affirmative à la question, ouverte alors, de savoir si ce modèle de calcul permet l'amortissement du calcul de n'importe quelle fonction.

Le présent mémoire aborde un certain nombre de questions ouvertes concernant le calcul catalytique. Les prochains paragraphes en feront le survol.

Koucký (2015) [**10**] pose les deux questions suivantes :

1. Quelles autres techniques existe-t-il pour faire usage d'une mémoire catalytique au-delà du calcul transparent et quelle est la relation entre le calcul transparent et le calcul catalytique ?
2. De quelles façons peut-on relaxer les contraintes du calcul catalytique pour obtenir d'autres modèles de calcul intéressants ?

En réponse à la première question, le chapitre 2 propose une méthode distincte du calcul transparent permettant de réaliser du calcul catalytique, la section 4.3 donne une application possible de cette méthode exemplifiant sa différence avec le calcul transparent. Aussi, la section 3.5 montre comment la notion de calcul transparent peut être vue comme un cas particulier de celle de gadget introduite dans le présent ouvrage. Ainsi, on documente de quelles façons, dans sa version non uniforme, le calcul catalytique peut aller au-delà du calcul transparent.

En réponse à la deuxième question, la section 4.2 montre un exemple de séparation inconditionnelle pour les programmes de branchement entre les puissances des modèles classique et catalytique et s'en sert de motivation pour discuter de variations possibles au calcul catalytique.

[**16**] demande quelles seraient les conséquences de l'existence de programmes de branchement k_n-catalytiques permettant d'amortir à $O(n)$ le coût du calcul de n'importe quelle fonction booléenne f_n pour k_n significativement plus petit que 2^{2^n}. La section 4.5 et la sous-section 4.4.5 montrent comment de tels programmes de branchement k-catalytiques

1. Cette borne est optimale, car elle représente dans ce modèle de calcul la borne inférieure de complexité sur le calcul d'une fonction dépendant de l'entièreté de son entrée.

peuvent être utilisés dans la construction de programmes de branchement k'-catalytiques pour $k' \ll k$ et en donnent certaines applications.

Aussi, le présent mémoire met en lumière les liens existants et précédemment inexploités entre le calcul catalytique et la complexité de communication. Dans la conclusion, on mentionne certaines façons par lesquelles la deuxième discipline peut contribuer à l'avancement de la première.

Enfin, le seul résultat connu jusqu'à ce jour de l'existence d'économie d'échelle inconditionnelle sur le calcul avec un programme de branchement catalytique d'une fonction est [**16**], mais requiert en contrepartie que l'économie d'échelle soit réalisée sur un très grand nombre de copies. La question de la possibilité d'économie d'échelle inconditionnelle pour un nombre de copies inférieur à 2^{2^n} reste ouverte. Dans ce mémoire, on prouve des bornes amorties $O(n \lg n)$ sur le calcul de toute fonction symétrique (où l'amortissement est pris sur le calcul d'un nombre de copies qui dépend de la fonction calculée et peut valoir $\lg n$, n ou 4^n) ce qui bat les meilleures bornes supérieures connues, mais ne constitue pas une économie inconditionnelle car elles restent supérieures (mais de très peu) à la meilleure borne inférieure connue qui est $\Omega(n \lg n / \lg \lg n)$. On peut réalistement espérer que les méthodes développées dans le présent mémoire pourront être poussées plus loin pour montrer la possibilité d'économie échelle inconditionnelle sur le calcul de fonction répété un petit nombre de fois. L'intérêt d'une telle séparation est entre autres discuté à la section 4.4 où le lien entre les programmes de branchement catalytique et la complexité de composition de fonctions est abordé.

0.1. Organisation du mémoire

Le chapitre 2 : *Une idée d'algorithme catalytique* décrit l'essentiel de l'idée derrière la contribution algorithmique du présent mémoire au calcul catalytique et le chapitre 3 : *Gadget et analyseur* sert à définir formellement les idées qui y sont implicitement contenues. Le chapitre 4 : *Applications* donne des applications de ces idées. Le chapitre 5 : *Une borne inférieure sur la taille des analyseurs* sert à en donner certaines limites. Toutes les figures auxquelles le texte fait référence sont contenues dans l'annexe A. L'annexe B contient les résultats de théorie de la communication pertinents au présent mémoire.

Chapitre 1

Préliminaires

On prend pour acquis que le lecteur connaît les notions de machine de Turing, de circuit booléen et d'uniformité. Voir **[15]** pour une bonne introduction à ces sujets.

1.1. Non uniformité

Définition 1.1.1 ([18]) **:** *Une* ***machine de Turing non uniforme*** *est une machine de Turing équipée d'une bande oracle en lecture seule qui pour toute entrée x contient une chaîne de caractères appelée* ***conseil(***$|x|$***)*** *dont le contenu ne dépend que de la taille de l'entrée. L'espace utilisé par une machine de Turing non uniforme sur une entrée x est défini comme la somme du nombre de cases visitées sur sa bande de travail et du logarithme de la taille de conseil($|x|$).*

Définition 1.1.2 : *On note $S_f(n)$ la complexité non uniforme en espace d'une séquence $(f_n)_{n\in\mathbb{N}}$ où $f_n \in F_n$.*

Théorème 1.1.3 ([7]) **:** *Soit une séquence $(f_n)_{n\in\mathbb{N}}$ où $f_n \in F_n$. On a :*

1. $S_f(n) \in O(\lg s_1(n))$ *où* $s_1 = \max(T(f_n), n)$,
2. $T(f_n) \in 2^{O(s_2(n))}$ *où* $s_2 = \max(S_f(n), \lg n)$.

Ainsi, lorsque $S_f(n) \geq \lg n$ et $T(f_n) \geq n$ on a $S_f(n) \in \Theta(\lg T(f_n))$.

La nature de la correspondance entre la complexité non uniforme en espace d'une séquence de fonctions et la taille des programmes de branchement optimaux réalisant leur calcul motive l'étude de ces derniers.

1.2. Quelques classes de complexité

Définition 1.2.1 ([**15**]) **:** *On rappelle les définitions des classes de complexité suivantes :*

1. NC^i *est la classe des langages reconnus par une famille* L*-uniforme de circuits de taille polynomiale et de profondeur* $O(\lg^i n)$ *avec des portes* $\vee$ *et* $\wedge$ *de degré entrant 2 et des portes* NON*,*
2. TC^i *est la classe des langages reconnus par une famille* L*-uniforme de circuits de taille polynomiale et de profondeur* $O(\lg^i n)$ *avec des portes* $\vee$*,* $\wedge$ *et* MAJ *de degré entrant arbitraire et des portes* NON*,*
3. L *est l'ensemble des langages reconnus en espace déterministe logarithmique,*
4. NL *est l'ensemble des langages reconnus en espace non déterministe logarithmique,*
5. [**6**] ZPP *est la classe des problèmes de décision que peut résoudre une machine de Turing probabiliste retournant la bonne réponse à tout coup et dont le calcul a une durée espérée polynomiale.*

Soit C *une classe de complexité, on note* $\mathsf{C/poly}$ *l'ensemble des langages* L' *pour lesquels il existe* $L \in \mathsf{C}$ *et une séquence de conseils* $(conseil_n)_{n\in\mathbb{N}}$ *dont les tailles sont bornées par un polynôme en* n *tels que* $x \in L \iff (x, conseil_{|x|}) \in L'$*.*

On a les inclusions suivantes ([**15**]) :

$$\mathsf{NC}^1 \subseteq \mathsf{L} \subseteq \mathsf{NL} \subseteq \mathsf{TC}^1 \subseteq \mathsf{NC}^2 \subseteq \mathsf{ZPP}.$$

1.3. Calcul catalytique

1.3.1. Calcul catalytique déterministe

La notion de machine catalytique a été introduite dans [**5**], la définition suivante provient plutôt de [**6**] et possède quelques différences mineures par rapport à [**5**]. L'alphabet des machines de Turing considérées est $\{0, 1\}$.

Définition 1.3.1 : *Soit* M *une machine de Turing équipée d'une bande d'entrée, d'une bande de sortie, d'une bande de travail et d'une bande supplémentaire appelée* ***bande auxiliaire****.*

Pour $x, a \in \{0,1\}^*$, on note $M(x,a)$ la machine M sur l'entrée x avec a le contenu de sa bande auxiliaire.

M est appelée une ***machine catalytique*** *avec espace de travail $s(n)$ et espace auxiliaire $w(n)$ si pour tout $n \in \mathbb{N}$ et pour tout $x \in \{0,1\}^n$ elle satisfait aux trois conditions suivantes :*

1. ***Borne sur l'espace*** *: Sur l'entrée x, M utilise un espace $O(s(n))$ sur sa bande de travail et un espace $O(w(n))$ sur sa bande auxiliaire.*
2. ***Condition catalytique*** *: Quelle que soit la configuration dans laquelle puisse se trouver sa bande auxiliaire au début d'un calcul, lorsque M s'arrête cette configuration est restaurée.*
3. ***Consistance*** *: Pour toutes configurations initiales a, a' et pour toute entrée x, $M(x,a)$ accepte (resp. refuse) ssi $M(x,a')$ accepte (resp. refuse).*

Remarque 1.3.2 ([**10**]) **:** *Le problème de décider si une machine de Turing est une machine catalytique, en particulier de décider si elle restaure toujours le contenu de sa bande auxiliaire, est indécidable.*

Définition 1.3.3 : *Soit une machine catalytique M, le langage L décidé par M est celui satisfaisant :*

1. $x \in L \implies \forall a \in \{0,1\}^*$ $M(x,a)$ *accepte,*
2. $x \notin L \implies \forall a \in \{0,1\}^*$ $M(x,a)$ *refuse.*

Par la propriété de consistance, ces deux conditions ensemble sont équivalentes à celle-ci :

$$x \in L \iff \forall a \in \{0,1\}^* \; M(x,a) \text{ accepte.}$$

Définition 1.3.4 : *Soient $s, w : \mathbb{N} \to \mathbb{N}$, on définit $\mathsf{CSPACE}(s(n), w(n))$ comme l'ensemble des langages pouvant être décidés par une machine catalytique avec espace de travail $O(s(n))$ et un espace auxiliaire $O(w(n))$. On note $\mathsf{CSPACE}(s(n)) = \mathsf{CSPACE}(s(n), 2^{O(s(n))})$ et $\mathsf{CL} = \mathsf{CSPACE}(\lg n)$.*

Lorsque $w(n) \in \Omega(2^{s(n)})$, la position de la tête de la machine de Turing sur sa bande auxiliaire encode plus d'information que l'espace de travail au complet [1]. Ainsi, il est naturel

1. Voir [**10**] pour un traitement plus long de ce cas.

de se restreindre au cas où $w(n) \in O(2^{s(n)})$. De plus, $s(n) \in \Theta(\lg(n))$ semble la plus petite taille raisonnable que peut prendre l'espace de travail. Pour ces raisons, il est intéressant de s'intéresser à CL qui correspond au cas extrême où l'espace de travail est le plus petit possible et l'espace auxiliaire associé est le plus grand possible.

1.3.2. Calcul catalytique non déterministe

On présente ici l'équivalent non déterministe de la notion de machine catalytique. A priori, il y a plusieurs façons différentes de définir ce que pourrait être une machine catalytique non déterministe, voir **[6]** pour un plaidoyer de la définition retenue :

Définition 1.3.5 : *Soit M une machine de Turing non déterministe équipée d'une bande d'entrée, d'une bande de sortie, d'une bande de travail et d'une bande auxiliaire.*

M est appelée une ***machine catalytique non déterministe*** *avec espace de travail $s(n)$ et espace auxiliaire $w(n)$ si pour tout $n \in \mathbb{N}$ et pour tout $x \in \{0,1\}^n$ elle satisfait aux trois conditions suivantes :*

1. ***Borne sur l'espace*** *: Sur l'entrée x, M utilise un espace $O(s(n))$ sur sa bande de travail et un espace $O(w(n))$ sur sa bande auxiliaire.*
2. ***Condition catalytique*** *: Pour toute configuration initiale a et pour toute séquence de choix non déterministes, $M(x,a)$ s'arrête avec a sur sa bande auxiliaire.*
3. ***Consistance*** *: Pour toutes configurations initiales a, a', pour toute entrée x et pour toute séquence de choix non déterministe, $M(x,a)$ accepte (resp. refuse) ssi $M(x,a')$ accepte (resp. refuse). Par contre, la séquence de choix non déterministes de M sur l'entrée x peut dépendre de la configuration initiale de sa bande auxiliaire.*

Définition 1.3.6 : *Soient $s, w : \mathbb{N} \to \mathbb{N}$, on définit $\mathsf{CNSPACE}(s(n), w(n))$ comme l'ensemble des langages pouvant être décidés par une machine catalytique non déterministe avec espace de travail $O(s(n))$ et un espace auxiliaire $O(w(n))$. On note $\mathsf{CNSPACE}(s(n)) = \mathsf{CNSPACE}(s(n), 2^{O(s(n))})$ et $\mathsf{CNL} = \mathsf{CNSPACE}(\lg n)$.*

Théorème 1.3.7 : *On a les inclusions suivantes :*

1. **[5]** $\mathsf{TC}^1 \subseteq \mathsf{CL}$,
2. **[6]** $\mathsf{CNL} \subseteq \mathsf{ZPP}$.

En particulier, on sait que $\mathsf{NL} \subseteq \mathsf{TC}^1$ et donc que $\mathsf{L} \subsetneq \mathsf{NL} \implies \mathsf{L} \subsetneq \mathsf{CL}$.

1.3.3. Calcul transparent

Les résultats et définitions de la présente section proviennent de [**5**]. Ils servent à donner l'idée générale de la méthode utilisée jusqu'à présent pour générer des algorithmes catalytiques, appelée calcul transparent, et comment une machine de Turing peut s'en servir.

Définition 1.3.8 : *Une machine réversible à registres M est un 4-uplet $((R,+,\times),n,m,L)$ où $(R,+,\times)$ est un anneau, $n,m \in \mathbb{N}$ et L est une liste d'instructions. M contient m registres de lecture-écriture r_j et n registres d'entrée x_i. Chaque registre contient un élément de R. Le contenu des registres d'entrées est fixé au début du calcul à partir de la valeur de l'entrée et ne change plus par après. Le contenu des registres de lecture-écriture commence dans une certaine configuration initiale puis change au cours du calcul. Chaque instruction I est de la forme $r_j \leftarrow r_j \pm u \times v$ où u,v sont des constantes prises dans R ou des registres différents de r_j. L'inverse d'une instruction I est obtenu en changeant le $+$ pour un $-$ et vice versa.*

Définition 1.3.9 : *On dit qu'une machine réversible à registre $M = ((R,+,\times),n,m,L)$ effectue le calcul transparent de $f : \{0,1\}^n \to \{0,1\}$ dans le registre r_j lorsque pour toutes configurations initiales $\tau_1,...,\tau_m$ des registres $r_1,...,r_m$ le registre r_j contient la valeur $\tau_j + f(x)$ après l'application de la suite d'instructions L.*

Le calcul transparent est une forme de calcul réversible s'en distinguant car il demande la validité du calcul sans condition sur les valeurs auxquelles sont initialisés les registres servant à effectuer le calcul.

Les résultats suivants montrent comment simuler un calcul transparent sur une machine de Turing. C'est de cette façon par exemple qu'à été généré le résultat $\mathsf{TC}^1 \subseteq \mathsf{CL}$ mentionné au théorème 1.3.7.

Définition 1.3.10 : *Soient une fonction $f : A \to B$ et $C \subseteq A$. On note $f(C)$ l'ensemble $\{f(c) \in B : c \in C\}$.*

Définition 1.3.11 : *On dit que $e : R \to \{0,1\}^*$ est un encodage compact de l'anneau R si e est bijective et $e(R)$ correspond aux $|R|$ premières chaînes de longueur $\lceil \lg |R| \rceil$ selon*

l'ordre lexicographique. On dit qu'une séquence d'anneaux $(R_n)_{n\in\mathbb{N}}$ est uniforme en espace logarithmique s'il existe une séquence d'encodages compacts $\{e_n\}_{n\in\mathbb{N}}$ de $\{R_n\}_{n\in\mathbb{N}}$ et des machines de Turing bornées en espace logarithmique $M_\star, M_2, M_3$ et M_4 tels que :

1. $M_\star(1^n, e_n(u), e_n(v))$ [2] $= e_n(u \star v)$ *où* $u, v \in R_n$ *et* $\star \in \{+, -, \times\}$,
2. M_2 *sur l'entrée* $(1^n, e_n(u))$ *transforme le contenu* $e_n(v)$ *d'une bande auxiliaire en* $e_n(u+v)$ *où* $u, v \in R_n$,
3. $M_3(1^n) = (e_n(-1), e_n(0), e_n(1))$,
4. $M_4(1^n) = |R_n|$.

La machine $M_\star$ permet de retourner l'encodage du produit de u et v dans R_n, la machine M_2 permet d'en faire la somme sur place [3]*, M_3 permet de retourner l'encodage de -1, 0 et 1 dans R_n et M_4 permet de retourner la taille de R_n.*

Lemme 1.3.12 : *S'il existe une machine de Turing M bornée en espace logarithmique qui sur entrée 1^n imprime $m_n \in \mathbb{N}$ alors $(\mathbb{Z}_{m_n})_{n\in\mathbb{N}}$ est une séquence d'anneaux uniforme en espace logarithmique.*

Par exemple, on peut prendre $m_n = 2^n$ ou $m_n = p_n$ où $\{p_n\}_{n\in\mathbb{N}}$ est une séquence de nombres premiers bornés polynomialement en n.

Lemme 1.3.13 : *Pour toute séquence d'anneaux $(R_n)_{n\in\mathbb{N}}$ uniforme en espace logarithmique, il existe une machine catalytique T qui sur entrée $x \in \{0,1\}^n$ avec conseil M_n calcule $f(x)$ où M_n est une machine réversible à registres $r_1, ..., r_m$ sur R_n effectuant le calcul transparent de $f(x)$ dans r_1. De plus, sur l'entrée x T utilise un espace auxiliaire de taille $(m\lceil \lg |R_n| \rceil)^2$ et un espace de travail $O(\lg(|P_n| + n))$.*

1.4. Programme de branchement

Définition 1.4.1 : *On note F_n l'ensemble des fonctions booléennes sur n bits.*

Définition 1.4.2 : *On note* lg *le logarithme en base 2.*

2. Pour s_1, s_2 et s_3 des chaînes binaires, $(s_1, s_2, s_3) \in \{0,1\}^*$ dénote leur encodage injectif, i.e. un encodage à partir duquel chaque s_i peut être isolé.

3. De l'anglais *in-place*.

Définition 1.4.3 : *Un programme de branchement déterministe (PB) sur n bits est un multigraphe orienté acyclique dont les arcs sont étiquetés. Ce graphe possède trois noeuds distingués, une source et deux puits, la source étant appelée "noeud initial" et les deux puits étant appelés "a" et "r", pour "acceptation" et "rejet" et vérifie la propriété suivante : tout noeud, sauf les puits, possède exactement deux arcs sortants étiquetés "$x_i = 0$" et "$x_i = 1$" pour un certain $i \in \{1, ..., n\}$.*

Pour tout $x \in \{0,1\}^n$, on définit le chemin de x dans un PB comme suit : à partir de la source et jusqu'à avoir atteint un des deux puits, successivement emprunter l'arc sortant du noeud courant dont l'étiquette concorde avec x. On dit que le PB accepte (resp. refuse) x si son chemin se termine en "a" (resp. en "r").

La fonction calculée par un programme de branchement déterministe B est $f_B : \{0,1\}^n \to \{0,1\}$ avec $f_B(x) = 1$ ssi B accepte x.

Exemple 1.4.4 : *Voir figure 1 page A-i pour un exemple de PB.*

Définition 1.4.5 : *Un chemin dans un PB qui contient, pour un certain i, un arc étiqueté "$x_i = 0$" et un arc étiqueté "$x_i = 1$" est dit invalide, sinon il est dit valide.*

Définition 1.4.6 : *Un programme de branchement non déterministe (NPB), est un PB duquel, pour un sous-ensemble des noeuds, on a retiré les étiquettes des arcs sortants. Ces noeuds sont dits non déterministes et le chemin d'une entrée peut traverser, au choix, n'importe lequel de leurs arcs sortants.*

La fonction calculée par un programme de branchement non déterministe B est $f_B : \{0,1\}^n \to \{0,1\}$ avec $f_B(x) = 1$ ssi il existe un chemin acceptant de x dans B.

Définition 1.4.7 ([**1**]) **:** *Un programme de branchement probabiliste (PBP) est un PB dont un sous-ensemble des noeuds ont les arcs sortants étiquetés "$r_i = 0$" et "$r_i = 1$" pour des valeurs de $i \in [m]$. Ces noeuds sont appelés générateurs aléatoires. L'entrée $x \in \{0,1\}^n$ du PB est adjointe à une variable aléatoire r choisie uniformément dans $\{0,1\}^m$ et la paire (x,r) traverse le PB comme dans le cas déterministe.*

La probabilité qu'un programme de branchement probabiliste B accepte une entrée $x \in \{0,1\}^n$ est définie comme ($\in_R$ représente le choix selon la distribution uniforme) :

$$\mathbb{P}[B(x)=1] = \mathop{\mathbb{P}}_{r \in_R \{0,1\}^n}[\ B \text{ accepte } (x,r)]$$

On dit alors que B (a,b)-calcule $f : \{0,1\}^n \to \{0,1\}$ si[4] *:*

1. $\mathbb{P}[B(x)=1] \leq a$ *lorsque* $f(x) = 0$,
2. $\mathbb{P}[B(x)=1] \geq b$ *lorsque* $f(x) = 1$.

La prochaine définition introduit formellement la notion de programme de branchement catalytique. Il est utile de voir un programme de branchement catalytique comme le graphe des configurations d'une machine de Turing catalytique dont les sources (resp. les puits) représentent les différentes configurations initiales (resp. finales) de la bande auxiliaire de la machine.

Définition 1.4.8 : *Un programme de branchement catalytique (PBC) est un multigraphe orienté acyclique dont les arcs sont étiquetés, possédant $3k$ noeuds distingués, k sources $s_0, ..., s_{k-1}$ et $2k$ puits $a_0, ..., a_{k-1}$, $r_0, ..., r_{k-1}$ pour $k \in \mathbb{N}$, satisfaisant la même propriété qu'un PB : tout noeud, sauf les puits, possède exactement deux arcs sortants étiquetés "$x_i = 0$" et "$x_i = 1$" pour un certain $i \in \{1, ..., n\}$. De plus, il doit vérifier la propriété suivante : pour tout $x \in \{0,1\}^n$ et pour tout $i \in [k]$, si le chemin de x commence en s_i, il doit se terminer en a_i ou r_i.*

Dans un PBC, la fonction calculée par la i^e source est $f_i : \{0,1\}^n \to \{0,1\}$ avec $f_i(x) = 1$ ssi le chemin de x, en commençant en s_i, se termine en a_i. La fonction calculée par un PBC B avec k sources est $g_B : \{0,1\}^n \to \{0,1\}^k$ où $g_B(x) = f_1(x)f_2(x)...f_k(x)$ est la concaténation des fonctions calculées par chacune des sources.

La condition que si le calcul commence en s_i, il doit se terminer en a_i ou r_i est l'analogue de la condition catalytique pour une machine de Turing.

Définition 1.4.9 : *Un programme de branchement non déterministe catalytique (NPBC) (resp. probabiliste (PBPC)) est la version non déterministe (resp. probabiliste) d'un PBC avec lequel la différence est définie de manière analogue au modèle non catalytique. Par contre, on apporte la contrainte additionnelle que peu importe les choix non déterministes (resp. aléatoires) effectués, si le chemin de l'entrée commence en s_i, il se termine en a_i ou r_i (i.e. si le chemin commence en s_i on ne peut pas se servir du non déterminisme*

4. Puisque les conditions 1. et 2. doivent être vérifiées pour toute valeur de x, il est nécessaire et suffisant qu'elles le soient en pire cas.

pour retrouver l'indice i de manière à terminer le calcul dans le bon puits et, dans le cas probabiliste, on n'autorise pas de se tromper de puits, même avec petite probabilité).

Définition 1.4.10 : *On définit la taille d'un PB comme étant son nombre de noeuds. Aussi, pour $f : \{0,1\}^n \to \{0,1\}$ et $g : \{0,1\}^n \to \{0,1\}^k$, on définit $T(f)$, la taille du plus petit PB calculant f et $T(g)$ la taille du plus petit PBC calculant g*[5]*.*

En notant $f^{||k} : \{0,1\}^n \to \{0,1\}^k$ la concaténation de f avec elle-même k fois (i.e. $f^{||k}$ calcule f sur la même entrée k fois), on abrège $T(f^{||k}) = T_k(f)$.

Lorsque toutes les k sources d'un PBC calculent la même fonction f, on dit simplement que celui-ci calcule k fois f. La propriété pour un PBC que toutes ses sources calculent la même fonction correspond à la propriété de consistance pour une machine de Turing.

Définition 1.4.11 : *Un PB est dit **stratifié** si l'ensemble de ses sommets peut être partitionné en étages avec tous les arcs sortants des noeuds d'un étage se connectant à des noeuds de l'étage suivant. Un PB stratifié dont tous les noeuds d'un même étage ont des étiquettes consultant le même bit de l'entrée est dit **indolent**.*

1.5. Bornes connues sur le calcul de fonctions

Théorème 1.5.1 ([4]) **:** *Pour $f_n \in F_n$, on a : $T(f_n) \leq (1+\varepsilon_n)2^n n^{-1}$, pour un certain ε_n vérifiant $\varepsilon_n \to 0$ lorsque $n \to \infty$.*

Théorème 1.5.2 ([8]) **:** *Pour $n \in \mathbb{N}$ et $k \leq 2^n$, on a :*

$$\mathop{\mathbb{P}}_{f_i \in_R F_n}[\ T(f_1||...||f_k) < \frac{k}{6} \cdot \frac{2^n}{n}\ ] \xrightarrow{n\to\infty} 0.$$

Autrement dit, avec haute probabilité pour $k \leq 2^n$, il n'y a aucune économie à réaliser sur le calcul catalytique d'un k-uplet de fonctions choisies uniformément et aléatoirement.

Théorème 1.5.3 ([8]) **:** *Soit une séquence de fonctions booléennes sur n bits $(f_n)_{n\in\mathbb{N}}$ et t_n telle que $T(f_n) \in \Theta(t_n)$. On a que :*

$\forall k \in \mathbb{N} \quad \exists (f_i)_{i\in[k]}$ *une séquence de k fonctions booléennes sur n bits vérifiant :*

$\forall i \in [k] \quad T(f_i) \in \Theta(t_n)$ *et* $T(f_1||...||f_k) \in O(k \lg k + t_n)$.

5. C'est le codomaine de la fonction donnée en argument à T qui détermine si son calcul requiert un PB ou un PBC.

Autrement dit, pour une de ces fonctions f_i l'utilisation de PBC permet d'amortir le coût du calcul de $\Theta(t_n)$ vers $O(\lg k + t_n/k)$.

Le théorème suivant montre que le calcul de n'importe quelle fonction peut être réalisé en taille amortie linéaire pourvu que l'amortissement soit pris sur un nombre suffisamment élevé de copies.

Théorème 1.5.4 ([**16**]) **:** *Pour toute fonction $f_n \in F_n$ et $k = 2^{2^n-1}$, on a : $T_k(f_n) \leq 64nk$.*

1.5.1. Bornes connues sur le calcul des fonctions symétriques

Définition 1.5.5 : *Pour $x \in \{0,1\}^n$, on définit :*

1. *$|x|_1$ = nombre de 1 dans x,*
2. *$|x|_0$ = nombre de 0 dans x.*

Définition 1.5.6 : *Pour $x \in \{0,1\}^n$, $a \in \mathbb{Z}_k^n$ et $k \in \mathbb{N}$, on définit $\langle a, x\rangle_k = \sum_{i=1}^{n} a_i x_i \pmod k$.*

Définition 1.5.7 : *Une fonction $f : \{0,1\}^n \to \{0,1\}$ est dite symétrique si elle ne dépend que du nombre de bits égaux à un dans l'entrée et non de leurs positions, i.e. $\forall x \in \{0,1\}^n \ f(x) = f(1^{|x|_1}0^{|x|_0})$.*

Définition 1.5.8 : *On définit les fonctions booléennes sur n bits suivantes :*

1. ***Compte exact*** : $E_n^a(x) = 1$ *ssi* $|x|_1 = a$.
2. ***Fonction seuil*** : $T_n^t(x) = 1$ *ssi* $|x|_1 \geq t$.
3. ***Majorité*** : $MAJ_n(x) = 1$ *ssi* $|x|_1 \geq |x|_0$.
4. ***Compte modulaire*** : $Mod_n^{a,k}(x) = 1$ *ssi* $|x|_1 = a \pmod k$.
5. ***Produit scalaire modulaire avec seuil*** : $PSMS_{v,n}^{t,k}(x) = 1$ *ssi* $\langle v, x\rangle_k \geq t$.

Toutes ces fonctions, sauf $\mathrm{PSMS}_{v,n}^{t,k}$, sont symétriques.

Théorème 1.5.9 ([**2**]) **:** *$T(MAJ_n) \in \Omega(n \lg n / \lg\lg n)$*

Théorème 1.5.10 ([**2**]) **:** *Soit $S_n^\delta = \{f : \{0,1\}^n \to \{0,1\} \mid f$ est symétrique et $T(f) < \delta n \lg n / \lg\lg n\}$. On a la limite suivante[6] : $\exists \delta > 0$: $|S_n^\delta|/2^n \to 0$ pour $n \to \infty$.*

6. $|S_n^\delta|/2^n$ représente la proportion de fonctions symétriques calculables par un PB de taille inférieure à $\delta n \lg n / \lg\lg n$.

Les meilleures bornes supérieures connues pour le calcul de certaines fonctions symétriques :

Théorème 1.5.11 ([**17**]) **:** *On a les bornes suivantes*[7] *:*

1. $T(E_n^a) \in O(n \lg^2 n / \lg\lg n)$,
2. $T(T_n^t) \in O(n \lg^3 n / (\lg\lg n \cdot \lg\lg\lg n))$,
3. $T(Mod_n^{a,k}) \in O(n \lg^4 n / (\lg\lg n)^2)$.

De plus, ces bornes sont atteintes par des programmes de branchements indolents.

La meilleure borne supérieure connue pour le calcul d'une fonction symétrique quelconque est donnée au théorème suivant :

Théorème 1.5.12 ([**12**]) **:** *Pour toute séquence* $f = (f_n)_{n\in\mathbb{N}}$ *de fonctions symétriques* $f_n : \{0,1\}^n \to \{0,1\}$ *, on a que :* $T(f_n) \in O(n^2/\lg n)$.

7. L'item 2. inclut MAJ_n.

Chapitre 2

Une idée d'algorithme catalytique

Imaginons qu'une personne, disons Charlie, possède une machine M lui permettant de réaliser le calcul physiquement réversible d'une fonction booléenne $f : \{0,1\}^n \to \{0,1\}$ quelconque. Sur une entrée $x \in \{0,1\}^n$, M peut être mise en marche à l'endroit et à l'envers (on appelle s l'état initial de M et $p = p(x)$ son état final) :

1. **À l'endroit** : M effectue son calcul sur l'entrée x et passe de l'état initial s à l'état final p.
2. **À l'envers** : M effectue étape par étape l'inverse de son calcul sur l'entrée x et retourne de l'état final p à l'état initial s.

Imaginons maintenant que l'état initial s de M n'est pas connu d'avance. Plutôt, on a que $s \in S$ pour un certain ensemble S et Charlie n'apprend la valeur de s que lorsque M est mise en marche. Sous ces conditions, l'état final p de M dépend également de s et on a $p = p(s, x)$.

Supposons que Charlie possède un moniteur sur lequel s'affiche l'état de M, l'entrée x, et la capacité de faire fonctionner M à l'endroit et à l'envers. Puisque Charlie ne peut pas voir *simultanément* s'afficher sur le moniteur les états s et p de M, on exprime la situation en ces termes : on appelle Alice la personne de Charlie lorsque M est dans l'état s et on l'appelle Bob lorsque M est dans l'état p. Ainsi, l'état s est seulement connu d'Alice, l'état p est seulement connu de Bob et leur but est d'effectuer le calcul distribué suivant : $(s, p(s, x)) \mapsto f(x)$. Pour ce faire, Charlie doit alternativement jouer les rôles d'Alice et de Bob dans un protocole de communication. Chaque fois qu'Alice pose une question à Bob sur la valeur de p, M doit faire son calcul à l'endroit pour que Bob puisse y répondre

et chaque fois que Bob veut poser une question à Alice sur s, M doit faire son calcul à l'envers.

Le fait pour Charlie de possèder une telle machine M et de connaître un protocole de communication lui permettant de calculer $f(x)$ à partir de l'état initial et final de M sur une entrée x lui permet de se servir de M comme d'un *catalyseur* pour son calcul. Charlie se sert de l'espace mémoire de la machine M pour réaliser son calcul tout en restituant son contenu à la fin.

Les idées implicitement contenues dans la présente section sont définies formellement au chapitre suivant. En particulier, la forme du protocole de communication choisi par Alice et Bob pour s'échanger de l'information (un analyseur) et le paramètre à optimiser pour qu'il soit qualifié de "bon" (la taille de l'analyseur) sont discutés à la section 3.2. Une façon de s'y prendre pour concevoir une machine M ayant la propriété de pouvoir effectuer le même calcul peu importe son état initial (gadget associé à un programme sur un groupe) est discutée à la section 3.4.

Chapitre 3

Gadget et analyseur

On définit ici formellement les idées implicitement contenues dans le chapitre 2 et requises aux chapitres suivants. Un gadget constituera une unité de calcul reversible représentant un genre de graphe des configurations d'une mémoire catalytique. Un analyseur constituera une sorte de protocole de communication permettant l'échange d'informations contenues dans ces configurations. On présentera une construction qui combine un gadget et un analyseur pour obtenir un PBC. Enfin, on verra comment se servir de groupes pour la construction de gadgets.

3.1. Gadget

Définition 3.1.1 : *Soit E un ensemble fini quelconque. Un E-programme de branchement (E-PB) est comme un programme de branchement (il possède une source et des arcs étiquetés), mais possède $|E|$ puits (ou moins) étiquetés par (un sous-ensembles) des éléments de E. La fonction calculée par un E-PB va de $\{0,1\}^n$ vers E.*

Exemple 3.1.2 : *Voir le $\mathbb{Z}_5$-PB de la figure 3 page A-ii servant à compter le nombre de bits égaux à 1 d'une entrée $x \in \{0,1\}^4$. Puisqu'en tant qu'ensembles $\mathbb{Z}_5 \subseteq \mathbb{Z}_6 \subseteq \mathbb{Z}_7 \subseteq ...$, le $\mathbb{Z}_5$-PB de la figure 3 est également un $\mathbb{Z}_6$-PB, un $\mathbb{Z}_7$-PB, etc.*

Définition 3.1.3 : *Soient $B = (V, E)$ un DAG et $u \in V$, la restriction de B à u est $B_u = (V_u, E_u)$ où : $V_u = \{v \in V \mid$ il existe un chemin de u à v dans $B\}$ et $E_u = \{(x, y) \in E | x, y \in V_u\}$.*

Définition 3.1.4 : *Soient B un DAG dont les arcs sont étiquetés et P l'ensemble de ses puits. On dit que c'est un* ***semi-gadget*** *s'il satisfait la propriété suivante :*

— *Si s est une source dans B, la restriction B_s est un P-PB.*

Définition 3.1.5 : *Soient B un semi-gadget et V l'ensemble de ses sommets. On dit que B est un* ***gadget*** *s'il satisfait la propriété suivante :*

— *$\forall u \in V$, les arcs entrant dans u ont des étiquettes distinctes qui concernent la même variable de l'entrée (i.e., il y a 0,1 ou 2 arcs entrant dans u et ils sont étiquetés "$x_i = 0$" ou "$x_i = 1$" pour un certain i).*

Notation 3.1.6 : *Soit un semi-gadget B dont s est une source et p un puits, on note $s \overset{x}{\rightsquigarrow} p$ pour indiquer qu'à partir de s, à la lecture de l'entrée x, on atteint p.*

Définition 3.1.7 : *Soit B, un gadget. On définit son* ***inverse****, noté B^{-1}, comme suit :*

— *Prendre une copie de B et inverser l'orientation de ses arcs en gardant leurs étiquettes*

— *Dans le cas où un sommet u dans B n'aurait qu'un arc entrant, il faut lui ajouter, dans B^{-1}, un arc parallèle contenant l'étiquette complémentaire (i.e, si, dans B, le seul arc entrant dans u est l'arc (v, u) étiqueté "$x_i = 0$", on ajoute un arc (u, v) dans B^{-1} étiqueté "$x_i = 1$").*

Exemple 3.1.8 : *Les figures 5a et 5b page A-iv montrent un gadget et son inverse.*

Propriété 3.1.9 : *En suivant la construction en deux étapes de la définition précédente, on obtient les trois propriétés suivantes :*

1. *l'inverse d'un gadget est un semi-gadget,*
2. *un gadget B et son inverse sont reliés par la relation : $s \overset{x}{\rightsquigarrow} p$ dans B ssi $p \overset{x}{\rightsquigarrow} s$ dans B^{-1}, où s et p sont respectivement une source et un puits dans B,*
3. *en mettant bout à bout un gadget et son inverse et en considérant le chemin d'une entrée à partir d'une source quelconque, les arcs ajoutés à la 2^e étape de la construction ne seront jamais traversés.*

Démonstration.

1. B^{-1} satisfait les deux propriétés d'un semi-gadget :

   a) B est acyclique et inverser ses arcs ne crée pas de cycle, donc B^{-1} l'est aussi.

   b) Tous ses noeuds qui ne sont pas des puits ont deux arcs sortants recevant des étiquettes complémentaires concernant le même bit de l'entrée.

2. Par construction, le chemin de x dans B^{-1} est simplement son chemin dans B parcouru en sens inverse.

3. Voir la preuve de 2.

□

Remarque 3.1.10 : *L'inverse d'un gadget peut ne pas être un gadget (voir la figure 2 page A-ii).*

3.2. Analyseur

Notation 3.2.1 : *Soit $D = (V, E)$ un DAG et $u \in V$. On utilise la notation suivante pour désigner les voisins de u : $N^+(u) = \{v \in V | (u, v) \in E\}$ et $N^-(u) = \{v \in V | (v, u) \in E\}$.*

Définition 3.2.2 : *Soit B un gadget possédant sources S et puits P. Un **analyseur** pour B est un programme de branchement de degré de sortie arbitraire et dont l'ensemble des noeuds non terminaux est partitionné en deux couleurs, rouge et noir. Le noeud initial est rouge et les arcs sortants des noeuds rouges (resp. noirs) sont étiquetés par les éléments d'une partition de P (resp. S) (la partition est potentiellement différente pour chaque sommet) et vont vers des noeuds de l'autre couleur (ou vers un noeud terminal non coloré).*

Spécifiquement, soit A un analyseur pour B et soit $\sigma(u, v)$ l'étiquette de l'arc (u, v) dans A, avec $N^+(u) = \{v_1, ..., v_m\}$ il faut que $(\sigma(u, v_1), ..., \sigma(u, v_m))$ forme une partition de P (resp. S) lorsque u est rouge (resp. noir).

Aussi, un analyseur possède exactement deux puits, nommés "a" et "r" pour l'acceptation et le rejet et ils ne sont accessibles qu'à partir des noeuds noirs.

Soit A, un analyseur pour B. A prend en entrée une paire $(s, p) \in S \times P$ à laquelle la règle suivante est associée : à partir du sommet initial de A, successivement emprunter aux noeuds rouges (resp. noirs) l'arc dont l'étiquette est l'élément de la partition de P (resp. S) contenant p (resp. s). Cette règle permet d'associer à la paire (s, p) un chemin dans A, laquelle est acceptée ou rejetée selon le puits par lequel son chemin se termine.

Remarque 3.2.3 : *Dans la définition précédente, ce qui est appelé un analyseur pour B ne dépend réellement que de S et P. Ainsi, pour n'importe quel gadget B' ayant les mêmes ensembles de sources et de puits que B, un analyseur pour B est aussi un analyseur pour B'.*

Exemple 3.2.4 : *La figure 5c page A-iv est un exemple d'analyseur pour un gadget B avec $S = \{1, 2\}$ et $P = \{1, 2, 3\}$. La paire $(1, 2)$ est acceptée et la paire $(2, 1)$ est rejetée.*

3.2.1. Interprétation des analyseurs en terme de protocole de communication

Soit B un gadget avec sources S et puits P et A un analyseur pour B. On associe à A la matrice binaire $|S| \times |P|$ suivante :

$$(M_A)_{s,p} = 1 \text{ ssi } A \text{ accepte la paire } (s, p).$$

L'analyseur A peut être interprété comme suit : recevoir en entrée une paire (s, p) partagée entre deux parties, disons Alice et Bob. Alice connaît seulement s et Bob connaît seulement p. Aux noeuds rouges, Alice pose à Bob une question du type "à quel sous-ensemble de colonnes de M_A appartient p" et aux noeuds noirs Bob pose à Alice une question du type "à quel sous-ensemble de lignes de M_A appartient s" jusqu'à ce que Bob en sache assez sur s et sur p (sans avoir nécessairement identifié la valeur de s) pour pouvoir connaître la valeur de M_A en (s, p).

Remarque 3.2.5 : *Un analyseur ressemble à un protocole de communication déterministe biparti (voir l'annexe B pour un rappel des notions pertinentes de théorie de la communication) avec les différences suivantes :*

1. *un analyseur est un DAG alors qu'un protocole est un arbre binaire,*
2. *un analyseur ne limite pas Alice et Bob à poser des questions « oui/non », ce que fait un protocole,*
3. *le paramètre capturant la complexité d'un analyseur est sa taille, pour un protocole c'est sa hauteur.*

Définition 3.2.6 : *Soit un analyseur A dont le domaine est $S \times P$, on dit que A calcule le $|S|$-uplet de fonctions $\{f_s\}_{s \in S}$ défini comme $f_s(p) = (M_A)_{s,p}$ pour tout $s \in S$.*

3.3. Programme de branchement catalytique à partir d'un gadget et d'un analyseur

Soit B un gadget dont les ensembles des sources et des puits sont respectivement S et P et soit A un analyseur pour B. Pour $s \in S$ et $x \in \{0,1\}^n$, on veut se servir de B pour trouver $p \in P$ tel que $s \overset{x}{\rightsquigarrow} p$ et de B^{-1} pour retrouver s à partir de p. On veut se servir de A pour déterminer l'acceptation ou le rejet de la paire (s,p). La construction suivante permet d'associer à la paire (A,B) un PBC effectuant le calcul :

$$\{0,1\}^n \xrightarrow{\text{via } B} S \times P \xrightarrow{\text{via } A} \{0,1\}.$$

Construction 3.3.1 : *Soit B un gadget et A un analyseur pour B.* ***Le programme de branchement catalytique associé à la paire (A,B)****, noté $PBC(A,B)$, est obtenu de A en remplaçant tout noeud u rouge (resp. noir) par une copie de B (resp. B^{-1}) qui hérite du nom de u et joignant la source et le puits α des gadgets B et B^{-1} par deux arcs parallèles lorsque les sommets correspondants dans A sont adjacents et l'arc les reliant est étiqueté E avec $\alpha \in E$.*

Aussi, tout arc (u,a) (resp. (u,r)) dans A et étiqueté E pour $E \subseteq S$ est remplacé par des arcs partant des puits s de u (qui sont des sources de B car u est noir par construction) et pointant vers des noeuds "a_s" (resp. "r_s") pour $s \in E$.

Remarque 3.3.2 : *Soient B un gadget avec S et P les ensembles, respectivement, de ses sources et de ses puits et A un analyseur pour B. En notant V_G l'ensemble des sommets d'un graphe G, l'ensemble des sommets de $C = PBC(A,B)$ est $V_C = (V_A \setminus \{a,r\}) \times V_B \cup_{s \in S} \{a_s, r_s\}$.*

Soit $\gamma = v_1 v_2 ... v_k$ le chemin de $x \in \{0,1\}^n$ dans B, où $v_1 = s \in S$ et $v_k = p \in P$. Pour $u \in V_A$, on utilise les notations suivantes :

1. *On dénote γ^{-1} le chemin $v_k, v_{k-1} ... v_1$ dans B^{-1}.*
2. *On dénote (u,γ) le chemin $(u,v_1)(u,v_2)...(u,v_k)$ dans C.*

Soit $\lambda = u_1, u_2, ..., u_{l-1}, u_l, t$ le chemin de (s,p) dans A, où $t \in \{a,r\}$. Par construction, on a donc que le chemin de x dans C, à partir de la source s, est :

$$(u_1,\gamma),(u_2,\gamma^{-1})...(u_{l-1},\gamma),(u_l,\gamma^{-1}),t_s$$

Exemple 3.3.3 : *Voir la figure 5 page A-iv pour un exemple de construction d'un PBC à partir d'un gadget et d'un analyseur. La matrice associée à l'analyseur A est :*

$$M_A = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

On peut vérifier que la fonction calculée par les sources 1 et 2 du PBC C sont respectivement $f_1(x_1, x_2) = x_1 \vee x_2$ *et* $f_2(x_1, x_2) = x_1 \wedge x_2$.

Par souci d'exhaustivité, on donne maintenant une définition formelle d'un PBC construit à partir d'un analyseur et d'un gadget introduit à la construction 3.3.1.

Définition : 3.3.4 : *Soient B un gadget avec S et P les ensembles, respectivement, de ses sources et de ses puits et A un analyseur pour B. On définit PBC*(A, B) *comme le DAG suivant (appelons-le C) :*

1. $V_C = (V_A \setminus \{a, r\}) \times V_B \bigcup_{s \in S} .\{a_s, r_s\}$
2. *Si* $u \in V_A$ *est rouge et* $(v_1, v_2) \in E_B$*, alors il y a un arc entre* (u, v_1) *et* (u, v_2) *dans C héritant de l'étiquette de* (v_1, v_2) *dans B.*
3. *Si* $u \in V_A$ *est noir et* $(v_2, v_1) \in E_B$*, alors il y a un arc entre* (u, v_1) *et* (u, v_2) *dans C héritant de l'étiquette de* (v_1, v_2) *dans B.*
4. *Soit* $u_1 \neq u_2 \in V_A \setminus \{a, r\}$ *et* $v_1, v_2 \in V_B$*, il existe deux arcs parallèles de* (u_1, v_1) *vers* (u_2, v_2) *dans C recevant des étiquettes complémentaires quelconques (*$''x_i = 0''$ *et* $''x_i = 1''$ *pour* $1 \leq i \leq n$*) si un de ces énoncés est vérifié :*
   - 4.1. u_1 *est rouge dans A,* $(u_1, u_2) \in E_A$*,* v_1 *est un puits de B,* v_2 *est une source de* B^{-1} *et* $v_1 = v_2 \in \sigma(u_1, u_2)$*,*
   - 4.2. u_1 *est noir dans A,* $(u_1, u_2) \in E_A$*,* v_1 *est un puits de* B^{-1}*,* v_2 *est une source de B et* $v_1 = v_2 \in \sigma(u_1, u_2)$.
5. *Soit* $u \in V_A \setminus \{a, r\}$*,* $t \in \{a, r\}$ *et* $s \in S$*, il existe deux arcs parallèles entre* (u, s) *et* t_s *dans C recevant des étiquettes complémentaires quelconques si* $(u, t) \in E_A$ *et* $s \in \sigma(u, t)$.
6. *Les points 2, 3, 4 et 5 donnent une description complète de* E_C.

Théorème 3.3.5 : *Soient B un gadget, S l'ensemble de ses sources et A un analyseur pour B. On a :*

1. *$C = PBC(A, B)$ est bel et bien un programme de branchement catalytique.*
2. *La fonction calculée par la source $s \in S$ dans C est : $f_s(x) = 1$ ssi A accepte (s, p) où $s \overset{x}{\rightsquigarrow} p$ dans B.*
3. *La taille de C (notée $|C|$) est simplement $\Theta(|A| \cdot |B|)$.*

Démonstration.

1. C possède les trois propriétés suivantes :
   1.1. C est acyclique. Par l'absurde, soit γ un chemin quelconque dans B et $(u_i, \gamma^{(-1)^i})_{i\in[l]}$ un cycle dans C. Par construction, $(u_i)_{i\in[l]}$ doit être un cycle dans A, or A est un DAG.
   1.2. Par construction, tout noeud non terminal dans C possède exactement deux arcs sortants étiquetés "$x_i = 0$" et "$x_i = 1$" pour $i \in \{1, ..., n\}$.
   1.3. Par la remarque 3.3.2, tout chemin parcouru par une entrée $x \in \{0, 1\}^n$ commençant par la source s se termine par $t_s \in \{a_s, r_s\}$.
2. Par la remarque 3.3.2, le chemin de x dans C à partir de s se termine par a_s (resp. r_s) si A accepte (resp. refuse) la paire (s, p) où $s \overset{x}{\rightsquigarrow} p$.
3. L'ensemble des sommets de C est $V_C = (V_A \setminus \{a, r\}) \times V_B \cup \{a_0, ..., a_{k-1}, r_0, ..., r_{k-1}\}$. Ainsi : $|C| = (|A| - 2)|B| + 2|S| \in \Theta(|A||B|)$.

□

Définition 3.3.6 : *Soient des ensembles $A, A_1, A_2, ..., A_k$. On dit que $A_1, ..., A_k$ est un recouvrement de A si $A \subseteq \cup_{i=1}^{k} A_i$.*

Définition 3.3.7 : *Pour obtenir l'équivalent non déterministe (resp. probabiliste) d'un analyseur, noté NDA (resp. PA), on remplace la contrainte que les arcs sortants des noeuds rouges (resp. noirs) soient les éléments d'une partition de P (resp. S) par la contrainte qu'ils soient les éléments d'un recouvrement de P (resp. S).*

Remarque 3.3.8 : *En notant $NPBC(A, B)$ (resp. $PBPC(A, B)$) le programme de branchement catalytique non déterministe (resp. probabiliste) construit en combinant un NDA*

(resp. PA) A avec un gadget B (qui, lui, reste déterministe) (voir définition 1.4.9 page 12 pour la définition de NPBC, PBPC) on a que :

1. *NPBC(A, B) est bel et bien un NPBC,*
2. *PBPC(A, B) est bel et bien un PBPC.*

3.4. Construire un gadget à partir d'un groupe

Pour construire un gadget avec autant de sources que de puits et dont le comportement puisse être appréhendé facilement, on introduit la notion de programme sur un groupe qui est un cas particulier de la notion de programme de branchement à permutation présentée à la définition suivante.

Définition 3.4.1 ([3]) **:** *Pour $x \in \{0,1\}^n$, $j \in \{1,...,n\}$ et $f, g : \{1,...,w\} \to \{1,...,w\}$, une instruction est un triplet $< j, f, g >$ s'interprétant comme suit : retourner la fonction f si $x_j = 1$, retourner g sinon.*

Un programme de branchement de largeur w (w-PB) B de longueur l est une suite de l instructions (où $\forall j \in \{1,...,l\}$ on a que $f_j, g_j : \{1,...,w\} \to \{1,...,w\}$) :

$$< j_1, f_1, g_1 >, < j_2, f_2, g_2 > , \ ... \ , \ < j_l, f_l, g_l >$$

s'interprétant comme suit : sur entrée $x \in \{0,1\}^n$, $B(x)$ est la composition des fonctions retournées par chacune des instructions.

Un programme de branchement à permutation de largeur w (w-PBP) est un w-PB où les fonctions f_j, g_j sont bijectives.

Définition 3.4.2 : *Soit un groupe G, on note $\mathbb{1}$ ou $\mathbb{1}_G$ l'élément neutre de G. Soit A un ensemble quelconque, on note $\mathbb{1}$ la fonction identité sur A ($\forall a \in A : \mathbb{1}(a) = a$).*

Remarque 3.4.3 : *Soit $\mathbb{1}$, la fonction identité sur $\{1,...,n\}$. Dans la définition des w-PBP, on peut remplacer les instructions $< j, f, g >$ par des instructions des formes $< j, f >$ et $< f >$ s'interprétant comme suit :*

1. *$< j, f >$ veut dire : retourner la fonction f si $x_j = 1$ et $\mathbb{1}$ sinon.*
2. *$< f >$ veut dire : retourner la fonction f.*

Il suffit de remplacer chaque instruction $< j, f, g >$ par $< g >, < j, g^{-1}f >$. Puisque les deux formalismes sont équivalents (à une constante multiplicative près sur la longueur des w-PBP), on les utilise de manière interchangeable.

Définition 3.4.4 : *Soient $G = \{g_0, ..., g_{k-1}\}$ un groupe fini, $x \in \{0,1\}^n$ et $j \in \{1, ..., n\}$. Un* ***programme sur G****, disons M, est une suite d'instructions de la forme $< j, a_j, b_j >$, $< j, a_j >$, $< a_j >$ où $a_j, b_j \in G$ s'interprètent de la même façon que pour un k-PBP et où $M(x)$ est un élément du groupe G.*

Un programme sur G est équivalent à la restriction d'un $|G|$-PBP où les fonctions f_j, g_j choisies sont prises parmi les permutations qu'induisent sur les indices des éléments de G la multiplication par d'autres éléments de G.

Le nom de "programme de branchement" dans w-PB et w-PBP est justifié par le fait qu'il existe une correspondance simple entre une suite d'instructions $< j_1, f_1, g_1 >, < j_2, f_2, g_2 >$, ... , $< j_l, f_l, g_l >$ et un programme de branchement. Cette correspondance est semblable à celle établie à la définition suivante entre un programme sur un groupe et un gadget.

Définition 3.4.5 : *Soit G un groupe de taille k et M le programme sur G suivant :*

$$< j_1, a_1 > \ , \ ... \ , \ < j_m, a_m > \qquad \textit{calculant} \qquad z(x) = a_1^{x_{j_1}} \cdot ... \cdot a_m^{x_{j_m}}.$$

Le gadget associé à M*, appelons-le B, est obtenu comme suit :*

1. *l'ensemble des sommets de B est $V = \{1, ..., m+1\} \times G$,*
2. *pour $1 \leq i \leq m$ et $g \in G$: il y a un arc de (i, g) vers $(i+1, g)$ étiqueté "$x_{j_i} = 0$" et un arc de (i, g) vers $(i+1, g \cdot a_j)$ étiqueté "$x_{j_i} = 1$".*

La taille de B est $(m+1) \cdot |G|$ et B possède le même nombre, k, de sources et de puits. Dans B, à partir de la source $(1, s)$ à la lecture de l'entrée $x \in \{0,1\}^n$ le puits atteint est $(m+1, s \cdot a_1^{x_{i_1}} \cdot ... \cdot a_m^{x_{i_m}})$. On indique ce comportement de B sur l'entrée x via la notation :

$$s \overset{x}{\rightsquigarrow} s \cdot z(x) \qquad , \textit{ dans } B.$$

Définition 3.4.6 : *Par **G-gadget** on désigne un gadget construit à partir d'un programme sur G, un groupe. Par **M-gadget**, on désigne l'unique*[1] *gadget construit à partir de M, un programme sur un groupe.*

Définition 3.4.7 : *Soient G un groupe et B un G-gadget. La **fonction associée à B** est $z_B : \{0,1\}^n \to G$ définie comme*[2] *$z_B(x) = p$ tel que $\mathbb{1}_G \overset{x}{\rightsquigarrow} p$. Dans la définition 3.4.5, cette fonction est simplement dénotée $z(x)$.*

Remarque 3.4.8 : *Dans un G-gadget B de longueur m, les sources sont les sommets $(1, g)$ et les puits sont les sommets $(m+1, g)$ pour $g \in G$. À partir de maintenant, au lieu de dire "la source $(1, s)$" on dit simplement "la source s" et au lieu de dire "le puits $(m+1, p)$" on dit simplement "le puits p".*

Remarque 3.4.9 : *Dans un G-gadget B, en notant s la source à partir de laquelle le chemin de l'entrée commence, p le puits auquel il se termine et $z = z_B(x)$ la fonction associée à B, on a que :*

$$s \cdot z = p \qquad \textit{autrement dit :} \qquad z = s^{-1} \cdot p.$$

L'équation de gauche se lit : dans B, à partir de la source s, à la lecture de x, on atteint le puits $s \cdot z$. On conservera cette interprétation pour les lettres s, p et z tout au long du mémoire.

Remarque 3.4.10 : *Dans un G-gadget, peu importe la source à partir de laquelle le trajet de l'entrée commence, le calcul effectué est le même. L'intention est de se servir d'un G-gadget comme de la machine physiquement reversible mentionnée au chapitre 2.*

Remarque 3.4.11 : *Soient B un gadget associé à M, un programme de longueur m sur un groupe G de k éléments, A un analyseur pour B et $C = PBC(A, B)$. La taille amortie associée au calcul simultané des fonctions $f_0, ..., f_{k-1}$ calculées par les k sources de C est $\Theta(m \cdot |A|)$ (comparer avec la taille de B, qui est $\Theta(m \cdot |G|)$).*

1. En suivant la construction indiquée à la définition 3.4.5.
2. Si, à partir de la source $\mathbb{1}_G$ dans B, à la lecture de l'entrée x, le trajet de l'entrée dans le gadget se termine au puits p, alors $z_B(x) = p$.

3.4.1. Interprétation d'un analyseur sur un G-gadget

Définition 3.4.12 : *Soit un G-gadget B, on note :*

$$W_B = \{p \in G : \exists x \in \{0,1\}^n : \mathbb{1} \overset{x}{\rightsquigarrow} p\}$$

l'ensemble des puits du gadget B accessibles à partir de la source $\mathbb{1}$.

Lemme 3.4.13 : *Soit un G-gadget B, les puits accessibles à partir de la source s dans B sont les éléments de sW_B. Dit autrement, si un calcul dans B commence en s et se termine en p, on a $s^{-1}p \in W_B$.*

Démonstration.

$$\exists x \in \{0,1\}^n : \mathbb{1} \overset{x}{\rightsquigarrow} p \iff \exists x \in \{0,1\}^n : s \overset{x}{\rightsquigarrow} s \cdot p$$

□

Dans le cas où B est un G-gadget ($|G| = k$) et on souhaite se servir d'un analyseur A pour construire un PBC qui calcule k fois la *même* fonction, il suffit de prendre un analyseur dont la matrice M_A prend la forme suivante, avec $E \subseteq W_B$ quelconque et $E' = W_B - E$ (cette définition est expliquée à la section 3.4.2) :

1. $(M_A)_{s,p} = 1$ si $p \in sE$ (ssi $s^{-1}p \in E$),
2. $(M_A)_{s,p} = 0$ si $p \in sE'$ (ssi $s^{-1}p \in E'$).

Aussi, s'il existe un $E \subseteq W_B$ (et $E' = W_B - E$) tel que M_A vérifie les conditions 1 et 2, on dit que A (en tant qu'analyseur pour B) calcule $f : G \to \{0,1\}$ définie comme $f(g) = 1$ ssi $g \in E$.

L'idée derrière les contraintes 1 et 2 est que puisque (sE, sE') est une partition de sW_B et que tout calcul commençant à une source s de B doit se terminer par un puits $p \in sW_B$ (par le lemme 3.4.13), les valeurs de la matrice M_A aux cases (s,p) pour lesquelles $p \notin sE \wedge p \notin sE'$ n'ont pas d'importance. C'est pourquoi il n'y a pas de contrainte sur elles.

Puisque G est un groupe, poser une question sur l'appartenance de s à un sous-ensemble de G revient à poser une question semblable sur s^{-1}. Ainsi, l'analyseur A peut être interprété comme suit : successivement poser des questions sur l'appartenance de s^{-1} et p à

des sous-ensembles de G jusqu'à pouvoir déterminer si le produit $s^{-1}p$ appartient à E (ou E').

Remarque 3.4.14 : *Soit $f : G \times G \to \{0,1\}$ la fonction calculée par un analyseur A. S'il existe une fonction $f' : G \to \{0,1\}$ telle que f peut s'exprimer comme $f(s,p) = f'(s^{-1}p)$ alors on peut aussi simplement dire que A calcule f'.*

3.4.2. Fonction calculée par un PBC construit à partir d'un analyseur sur un G-gadget

Théorème 3.4.15 : *Soient un G-gadget B de longueur m dont la fonction associée est z_B, un analyseur A pour B calculant une fonction $f : G \to \{0,1\}$ et $C = PBC(A,B)$. On a que toutes les sources de C calculent la même fonction h où :*

$$h : \{0,1\}^n \xrightarrow{z_B} G \xrightarrow{f} \{0,1\}.$$

Démonstration. Le théorème 3.3.5 garantit que la fonction calculée par la source s dans C est : $f_s(x) = 1$ ssi A accepte $(s,p) = (s, s \cdot z_B(x))$. Or, par le lemme 3.4.13 $s^{-1}p \in W_B$ et par définition A n'accepte que les paires (s,p) de W_B vérifiant $s^{-1}p \in E$ (avec $E = W_B \cap f^{-1}(1)$). Pour ces paires (s,p) on a donc :

$$\begin{aligned}
& s^{-1}p \in E \\
\iff & z_B(x) \in E && \text{en substituant } p \text{ par sa valeur} \\
\iff & (f \circ z_B)(x) = 1 && \text{en appliquant } f \text{ des deux côtés de l'équation}
\end{aligned}$$

Ainsi $f_s(x) = 1 \iff (f \circ z_B)(x) = h(x) = 1$, ce qui conclut la preuve. □

Avec $k = |G|$ le nombre de sources dans C, la remarque 3.4.11 donne la borne suivante sur la complexité du calcul de $h^{||k}$:

$$T_k(h) \in O(m \cdot |A|).$$

Remarque 3.4.16 : *Soient un groupe G de taille k, un G-gadget B dont la fonction associée est z_B et deux matrices binaires distinctes $M, M' \in \{0,1\}^{k \times k}$. Il se peut tout à fait que pour tout analyseur A et A' dont les matrices sont respectivement M et M', on*

ait que PBC(A, B) et PBC(A', B) calculent la même fonction. De plus, il se peut que les analyseurs optimaux pour M et M' aient des tailles complètement différentes.

La leçon à retenir de la remarque précédente est la suivante : si on veut calculer une fonction h à partir d'un G-gadget B *fixe*, parmi toutes les matrices binaires M sur $|G| \times |G|$ qui calculent $f : G \to \{0, 1\}$ telle que $h = f \circ z_B$, il est avantageux de prendre celle qui se calcule par un analyseur de la plus petite taille possible.

On formalise la notion comme suit.

Définition 3.4.17 : *Soit un G-gadget B dont la fonction associée est z_B. Deux matrices M et $M' : |G| \times |G| \to \{0, 1\}$ sont dites B-équivalentes ssi elles coïncident sur l'ensemble des paires (s, p) vérifiant $s^{-1}p \in W_B$.*

Remarque 3.4.18 : *Soit un G-gadget B et A, A' deux analyseurs pour B. On a que PBC(A', B) et PBC(A, B) calculent les mêmes fonctions ssi M_A et $M_{A'}$ sont B-équivalente.*

On donne en exemple deux matrices B-équivalentes pour lesquels les analyseurs optimaux ont des tailles très différentes. La preuve utilise par contre des notions des chapitres 4 et 5 et de l'annexe B.

Exemple 3.4.19 : *Soit A et B respectivement l'analyseur et le gadget du théorème 4.2.1. On a $W_B = [n + 1]$ et pour $\Delta = n\epsilon$ la taille de A est $\Theta(1)$. Soit M' la matrice obtenue de M_A en ajoutant des 1 sur sa diagonale $(i, i + \frac{3}{2}n)$. On a que M' et M_A sont B-équivalentes car elles ne se distinguent que par des éléments qui ne satisfont pas $s^{-1}p \in W_B$. De plus, $S = \{(i, i + 3n/2) \mid 0 \leq i < n/2\}$ est un ensemble trompeur de taille $n/2$. Une application du théorème 5.0.7 permet d'obtenir une borne inférieure $\Omega(\lg n)$ sur la taille de tout analyseur calculant M' ce qui conclut l'exemple.*

3.5. Gadget pour le calcul transparent

On montre dans cette section, à titre informatif seulement car ce fait ne sera pas réutilisé dans le corps de l'ouvrage, comment construire un gadget simulant une machine réversible à compteur effectuant un calcul transparent.

Lemme 3.5.1 : *Soient $(R, +, \times)$ un anneau de taille t et $l = \lceil \lg t \rceil$. Il existe des éléments $a_j \in R$ pour $j \in \{1, ..., l^2\}$ tels que l'application $h : \{0,1\}^{l^2} \to R$ est surjective, où $h(x) = \Sigma_{j=1}^{l^2} a_j \times x_j$.*

DÉMONSTRATION. Il existe un ensemble de générateurs $\{\alpha_1, ..., \alpha_k\}$ de taille $k \leq l$ tel que, en tant que groupe additif, $< \alpha_1, ..., \alpha_k >= R$. Ainsi, en fixant $\alpha_j = 0_R$ pour $j > k$:

$$\forall r \in R \ \exists c_j \in [t] : r = \sum_{j=1}^{k} c_j \alpha_j$$

$$\implies \forall r \in R \ \exists d_{j,j'} \in \{0,1\} : r = \sum_{j=1}^{k} (\sum_{j'=1}^{l} 2^{j'} d_{j,j'}) \alpha_j$$

$$\implies \forall r \in R \ \exists d_{j,j'} \in \{0_R, 1_R\} : r = \sum_{j=1}^{k} \sum_{j'=1}^{l} (2^{j'} \alpha_j) \times d_{j,j'}$$

$$\implies \forall r \in R \ \exists d_{j,j'} \in \{0_R, 1_R\} : r = \sum_{j=1}^{l} \sum_{j'=1}^{l} (2^{j'} \alpha_j) \times d_{j,j'}$$

$$\implies \forall r \in R \ \exists x_j \in \{0_R, 1_R\} : r = \sum_{j=1}^{l^2} a_j \times x_j.$$

□

On compte se servir du lemme précédent pour définir un encodage redondant mais commode des éléments d'un anneau fini. Pour simuler une machine à compteur sur un anneau R qui inspecte le registre d'entrée $x_i \in R$, un programme de branchement inspecte ses bits d'entrée $x_{i,j}$ pour $j \in \{1, ..., l^2\}$.

Définition 3.5.2 : *Soit $M = ((R, +, \times), n, m, L)$ une machine réversible à compteurs effectuant le calcul transparent de $f(x)$ dans r_1 où L est de longueur k et où s.p.d.g l'instruction i est de la forme $I_i : r_{l_i} \leftarrow r_{l_i} \pm u_i \times v_i$ où u_i et v_i ne sont pas tous deux des registres d'entrée. Soient également les constantes $a_j \in R$ du lemme précédent. On construit le gadget B_M (sur $n \cdot l^2$ bits) associé à M comme suit (où $l = \lceil \lg |R| \rceil$) :*

1. *$V_{B_M} = \{1, ..., N_{etage} + 1\} \times R^m$, où N_{etage} est défini comme la valeur finale de e au point suivant,*
2. *pour $e = 1$, pour i de 1 à k :*

2.1. si ni v_i, ni u_i n'est un registre d'entrée à l'instruction i, mettre deux arcs parallèles de $(e, r_1, ..., r_m)$ vers $(e+1, r_1, ..., r_{l_i}+u_i \times v_i, ..., r_m)$ étiquetés $''x_1 = 0''$ et $''x_1 = 1''$ et incrémenter e,

2.2. si le registre d'entrée inspecté à l'instruction i est $u_i = x_{l'_i}$, pour $j \in \{1, ..., l^2\}$ mettre un arc de $(e, r_1, ..., r_m)$ vers $(e+1, r_1, ..., r_m)$ étiqueté $''x_{l'_i,j} = 0''$ et un arc vers $(e+1, r_1, ..., r_{l_i} + a_j \times v_i, ..., r_m)$ étiqueté $''x_{l'_i,j} = 1''$ et incrémenter e,

2.3. si le registre d'entrée inspecté à l'instruction i est $v_i = x_{l'_i}$ faire comme au point précédent en remplaçant $a_j \times v_i$ par $u_i \times a_j$.

Remarque 3.5.3 : *Pour le gadget B_M de la définition précédente, on a :*

1. $N_{etage} \leq k \cdot l^2$,
2. $\forall x \in \{0,1\}^{n \cdot l^2}\ \forall s \in R\ \forall s' \in R^{m-1}\ \exists p'_x \in R^{m-1}\ : (s, s') \overset{x}{\rightsquigarrow} (s + f(x), p'_x)$ *dans B.*

Ainsi, pour identifier la valeur de $f(x)$ Alice et Bob peuvent faire comme pour un G-gadget (car $(R, +)$ est un groupe) en restreignant leurs questions sur la première composante de l'élément dans R^m que chacun possède. En effet, même si le gadget B_M de la définition 3.5.2 n'est pas à proprement parler un R^m-gadget, il peut être utilisé par un analyseur comme s'il en était un à cause de la relation unissant ses sources et ses puits : par l'item 2 de la remarque 3.5.3, on a qu'en commençant à partir de la source (s, s') dans B_M, à la lecture de x, on termine au puits (p, p') avec $p = s + f(x)$.

Chapitre 4

Applications

4.1. Calcul du produit scalaire

Notation 4.1.1 : *On note $\mathbb{Z}_k$ le groupe additif des entiers modulo k. Aussi, pour $\alpha \geq \beta$, on note $\{\alpha, ..., \beta\}$ l'ensemble $\{\alpha, ..., k-1\} \cup \{0, ..., \beta\}$.*

Exemple 4.1.2 : *Dans $\mathbb{Z}_5$, on note $\{3, ..., 1\}$ l'ensemble $\{3, 4\} \cup \{0, 1\}$.*

Théorème 4.1.3 : *Soient $G = \mathbb{Z}_k$, $\vec{t} = (t_0, ..., t_{k-1}) \in \mathbb{Z}_k^k$ et $\vec{a} \in \mathbb{Z}_k^n$. Soit M le programme sur G suivant :*

$$< 1, a_1 >, < 2, a_2 >, ..., < n, a_n > \qquad \textit{calculant} \qquad \sum_{i=1}^{n} x_i \cdot a_i = \langle a, x \rangle_k \ [1].$$

Soit B le gadget associé à M (représenté à la figure 4 page A-iii) et soit A l'analyseur de taille $\Theta(\lg k)$ pour B affiché à la figure 7 page A-vi. Le PBC $C = PBC(A, B)$ calcule les fonctions $PSMS_{\vec{a},n}^{t_j,k}$ pour $j \in [k]$ simultanément en taille amortie $\Theta(n \lg k)$ [2].

Démonstration. En effet, A accepte la paire $(s, p) \in \mathbb{Z}_k^2$ ssi $p \in \{t_s + s, ..., k - 1 + s\}$, c'est-à-dire lorsque $p - s \geq t_s$. Or, $s \overset{x}{\rightsquigarrow} p = s + \langle a, x \rangle_k$ dans B, donc $p - s = \langle a, x \rangle_k$ et on a bel et bien que C accepte l'entrée x à partir de la source s lorsque $\langle a, x \rangle_k \geq t_s$. □

L'intérêt du théorème 4.1.3 vient du fait que les fonctions $\mathrm{E}_n^a, \mathrm{T}_n^t, \mathrm{MAJ}_n, \mathrm{Mod}_n^{a,k}$ introduites à la définition 1.5.8 et dont les meilleures bornes supérieures sont énoncées au théorème 1.5.11 sont toutes réductibles à une ou deux applications de $\mathrm{PSMS}_{\vec{1},n}^{t_j,k}$. Or, la nouvelle borne $O(n \lg k)$ amortie sur k répétitions pour le calcul de $\mathrm{PSMS}_{\vec{a},n}^{t_j,k}$ fournie par

1. Voir définition 1.5.6.
2. Pour n'importe quel k-uplet de seuils t_j.

le théorème 4.1.3 donne lieu au constat suivant : soit que le théorème 4.1.3 implique une meilleure borne supérieure pour la taille des PB pour $\mathrm{E}_n^a, \mathrm{T}_n^t, \mathrm{MAJ}_n$, et $\mathrm{Mod}_n^{a,k}$, soit qu'il implique que le calcul catalytique permet des économies d'échelles pour le calcul de fonctions itéré un petit nombre de fois.

Remarque 4.1.4 (version catalytique du lemme 5 de [**17**]) **:** *Par le théorème des restes chinois, sur une entrée $x \in \{0,1\}^n$, pour vérifier que $|x|_1 = k$, il suffit de prendre une liste L de longueur $O(\lg n/\lg\lg n)$ de nombres premiers p_j de taille $O(\lg n)$ dont le produit est supérieur à n et de vérifier que $\forall p_j \in L : Mod_n^{k,p_j}(x) = 1$. Ainsi, en concaténant les PBC pour Mod_n^{k,p_j} on obtient un PBC C pour E_n^k. Par le théorème 4.1.3, le calcul de Mod_n^{k,p_j} se fait $O(\lg n)$ fois en taille amortie $O(n\lg\lg n)$, ce qui veut dire que C calcule E_n^k $O(\lg n)$ fois en taille amortie $O(n\lg\lg n\cdot\lg n/\lg\lg n) = O(n\lg n)$.*

4.2. Approximations de fonctions seuils

Disons qu'on veuille calculer Δ fois la même fonction seuil T_n^t et que le contexte rende tolérable l'approximation suivante : remplacer le seuil t par un seuil t' dont l'écart relatif avec t est petit. Est-ce que le problème de calculer Δ telles approximations (possiblement distinctes) de T_n^t est plus facile que le problème initial ? Le théorème 4.1.3 donne une borne amortie de $O(n\lg n)$ sur le calcul de MAJ_n n fois. Le théorème suivant permet de faire mieux lorsqu'on autorise le calcul approximé :

Théorème 4.2.1 : *Pour $n \in \mathbb{N}$, $t \leq n$, $\Delta \leq n - t$ et $m = \lceil \frac{n}{\Delta} \rceil$ on a :*

$$T(\{T_n^{t+j}\}_{j\in[\Delta]}) \leq 2(m+1)n^2 + 2\Delta.$$

En particulier, pour ϵ constant et $\Delta = n\epsilon$ on peut calculer les Δ fonctions seuils sur n bits dont le seuil est compris entre t et $t + \Delta - 1$ simultanément en taille amortie $\leq \frac{3}{\epsilon}n$.

Démonstration. Soit le programme M de longueur n sur $\mathbb{Z}_{2n}$ suivant :

$$< 1,1 >, < 2,1 >, ..., < n,1 >$$

et soit B le gadget associé. La fonction associée à B est $z_B(x) = |x|_1$.

Soit $(E_j)_{j\in[2m]}$ une partition de $\mathbb{Z}_{2n}$ avec :

1. $E_j = j\Delta + [\Delta] = \{j\Delta, ..., (j+1)\Delta - 1\}$ pour $0 \leq j < 2m - 1$,

2. $E_{2m-1} = \{(2m-1)\Delta, ..., 2n-1\}$.

Soit A l'analyseur sur B suivant (voir figure 9 page A-vii) où % désigne le reste de la division euclidienne :

1. Bob envoie à Alice l'indice j tel que $p \in E_j$,
2. Alice accepte ssi $((\min(E_j) - s)\%(2n) \geq t \quad \wedge \quad s \notin E_j)$.

Soit $C = \text{PBC}(A, B)$ de taille $(|A|-2)|B|+2|S| = 2(m+1)n^2+2\Delta$. On a que la fonction associée à la source s dans C est :

$$\begin{aligned}
f_s(x) = 1 &\iff p \in E_j : (\min(E_j) - s)\%(2n) \geq t \quad \wedge \quad s \notin E_j \\
&\iff (\left\lfloor \frac{p}{\Delta} \right\rfloor \Delta - s)\%(2n) \geq t \quad \wedge \quad s \notin E_j \\
&\iff (p - (p\%\Delta) - s)\%(2n) \geq t \quad \wedge \quad s \notin E_j \\
&\iff (|x|_1 - (s + |x|_1)\%\Delta)\%(2n) \geq t \quad \wedge \quad s \notin E_j \qquad \text{car } s + |x|_1 = p \\
(\star) \qquad &\iff |x|_1 - (s + |x|_1)\%\Delta \geq t.
\end{aligned}$$

Justification de $(\star)$:

1. Si $s \notin E_j$ alors $\%(2n)$ peut être négligé, en effet :

$$\begin{aligned}
s \notin E_j &\implies \exists y : 0 \leq y \leq |x|_1 : (s+y)\%\Delta = 0 \\
&\implies y - (s+y)\%\Delta \geq 0 \\
&\implies \forall y' > y : y' - (s+y')\%\Delta \geq 0 \\
&\implies |x|_1 - (s + |x|_1)\%\Delta \geq 0
\end{aligned}$$

$$\text{De plus :} \qquad |x|_1 \in [n+1] \implies |x|_1 - (s + |x|_1)\%\Delta < 2n$$

$$\text{Ainsi :} \qquad |x|_1 - (s + |x|_1)\%\Delta \in [2n].$$

2. L'implication inverse :

$$\begin{aligned}
|x|_1 - (s + |x|_1)\%\Delta \geq 0 &\implies \exists y : 0 \leq y \leq |x|_1 : (s+y)\%\Delta = 0 \\
&\implies \text{ il existe un multiple de } \Delta \text{ entre } s \text{ et } s + |x|_1 \\
&\implies s \notin E_j.
\end{aligned}$$

La ligne ($\star$) montre que f_s est symétrique et monotone, f_s est donc une fonction seuil. Le seuil en question est le plus petit entier t' plus grand que t s'écrivant sous la forme $k\Delta - s\%\Delta$ pour $k \in \mathbb{N}$. En effet :

$$\begin{aligned} & t' < t + \Delta = t + (s + t' - 1)\%\Delta + 1 \\ \Longrightarrow\ & t' - 1 < t + (s + t' - 1)\%\Delta \\ \Longrightarrow\ & f_s(x) = 0 \text{ pour } |x| = t' - 1. \end{aligned}$$

De plus,

$$\begin{aligned} & t' \geq t = t + (s + t')\%\Delta \\ \Longrightarrow\ & f_s(x) = 1 \text{ pour } |x| = t'. \end{aligned}$$

On a donc $f_s = T_n^{t'}$ et par conséquent pour tout entier j tel que $0 \leq j < 2m - 1$ on a $\{f_s\}_{s \in E_j} = \{T_n^{t+j}\}_{j \in [\Delta]}$. En prenant une de ces valeurs de j arbitrairement et en restreignant C aux sources dans E_j on conclut la preuve. □

Remarque 4.2.2 : *Soient $t_1, t_2 : \mathbb{N} \to \mathbb{N}$ tels que $t_1(n) \in \Theta(t_2(n))$, on a que $T(T_n^{t_1}) \in \Theta(T(T_n^{t_2}))$. En effet, il suffit de combler avec $\Theta(t_2(n))$ variables initialisées pour convertir un PB pour calculer $T_n^{t_1}$ en un PB pour calculer $T_n^{t_2}$ et vice versa.*

Remarque 4.2.3 : *La technique de la remarque précédente ne peut pas être utilisée avec le théorème 4.2.1 pour convertir le calcul des fonctions $\{T_n^{t+j}\}_{j \in [\Delta]}$ en le calcul de $(T_n^t)^{||\Delta}$. En effet, convertir T_n^{t+j} en T_n^t demande de combler avec des variables initialisées, mais chaque valeur de j demande un nombre différent de telles variables ce qui est incompatible avec le fait que leurs calculs doivent être effectués simultanément sur la même entrée.*

Le théorème 1.5.9 donne la borne $T(\mathrm{MAJ}_n) \in \Omega(n \lg n / \lg \lg n)$ et la remarque 4.2.2 étend cette borne aux fonctions seuil T_n^t dont le seuil t est proche de $\frac{n}{2}$. Ainsi, en considérant les fonctions T_n^t dont le seuil t est proche de $\frac{n}{2}$ comme les approximations valides de MAJ_n, le problème de calculer de manière déterministe une approximation valide de MAJ_n possède la même complexité que celui de calculer MAJ_n tout court, à savoir $\Omega(n \lg n / \lg \lg n)$. Or, le théorème 4.2.1 montre que soit il en est autrement pour le calcul catalytique, soit $T_n(\mathrm{MAJ}_n) \in O(n)$. Ainsi, le calcul catalytique permet d'obtenir des résultats qui ne sont pas possibles classiquement. Cela ouvre le problème suivant :

Définition 4.2.4 : *Pour $f : \{0,1\}^n \to \{0,1\}$ quelconque, on note H_f l'ensemble des fonctions booléennes sur n bits considérées comme des approximations acceptables de f (le contenu spécifique de H_f dépend de f et du contexte).*

On définit alors :

$$\widetilde{T}_k(f) = \min\{T(g_1, ..., g_k) \mid \forall i \in \{1, ..., k\}\ g_i \in H_f\}.$$

Problème 4.2.5 : *Pour une condition raisonnable permettant de faire la correspondance $f \mapsto H_f$ comment se comporte la différence entre $T_k(f)$ et $\widetilde{T}_k(f)$?*

On remarque que de permettre que la fonction calculée par une machine catalytique dépende (même "légèrement") du contenu initial de sa bande auxiliaire est contraire à la propriété de consistance. Il existe au moins deux façons raisonnables de relaxer cette propriété, on peut la remplacer par :

1. **Quasi-consistance** : Il existe une séquence de fonctions booléennes $(f_n)_{n\in\mathbb{N}}$ telle que pour toute configuration initiale a et pour toute entrée x de longueur n, la fonction *en* x calculée par $M(x, a)$ appartient à H_{f_n}.
2. **Non-inconsistance** : Il n'existe pas de paires de configurations initiales a, a' et d'entrée x telle que $M(x, a)$ accepte et $M(x, a')$ refuse. Par contre, on autorise que $M(x, a)$ réponde "ne sait pas".

On note que ces propriétés ont toutefois le défaut que pour certaines entrées x il existe potentiellement des configurations initiales a de la bande auxiliaire pour lesquelles, systématiquement, $M(x, a)$ se trompe (dans le cas de la quasi-consistance) ou réponde "ne sait pas" (dans le cas de la non-inconsistance).

Remarque 4.2.6 : *En adaptant la définition d'analyseur pour permettre au protocole de communication de se conclure par "ne sait pas" et modifiant l'analyseur du théorème 4.2.1 pour :*

1. Bob envoie à Alice l'indice j tel que $p \in E_j$,

2. Alice :

2.1. accepte si $\min(E_j - s) \geq t$,

2.2. refuse si $\max(E_j - s) < t$,

2.3. répond "ne sait pas" sinon.

On a que le PBC résultant satisfait la propriété de non-inconsistance adaptée, mutatis mutandis, au programme de branchement.

4.3. Test d'appartenance à un sous-groupe

Soit G un groupe et $H < G$ muni de la relation d'équivalence $\alpha \sim \beta \iff \alpha H = \beta H$. On a que :

$$\begin{aligned}
\alpha \sim \beta &\iff \alpha H = \beta H \\
&\iff H = \alpha^{-1}\beta H \\
&\iff \alpha^{-1}\beta \in H \\
&\iff \beta \in \alpha H.
\end{aligned}$$

Ainsi, les classes d'équivalences induites par $\sim$ sont les translatés à gauche de H par les éléments de G. L'ensemble de ces classes d'équivalences est noté G/H. Soit R, l'ensemble construit en choisissant arbitrairement un représentant de chacune des classes. Pour $r \in R$, le stabilisateur à droite de rH est :

$$\begin{aligned}
Stab_{droite}(rH) &= \{g \in G | rHg = rH\} \\
&= \{g \in G | Hg = H\} \\
&= \{g \in G | g \in H\} \\
&= H.
\end{aligned}$$

En se rappelant de la formule reliant la source et le puits traversés par une entrée dans un G-gadget $s \cdot z = p$, on voit que la source et le puits appartiennent à la même classe si $z \in Stab_{droite}(rH) = H$. De même, si $s, p \in rH$, on a que $s^{-1} \in Hr^{-1}$ donc $z \in (Hr^{-1})(rH) = H$.

Ainsi, pour tester si $z(x) \in H$, on partitionne l'ensemble des sources et des puits du G-gadget en translatés à gauche de H et on vérifie simplement que la source et le puits appartiennent au même ensemble. Pour ce faire, on numérote les translatés de H et on compare bit à bit leurs représentations binaires à l'aide d'un analyseur semblable à celui

de la figure 7 page A-vi. Il y a $|G : H| = |G|/|H|$ ensembles dans G/H, ainsi comparer leurs représentations binaires peut être fait par un analyseur de taille $O(\lg |G : H|)$.

4.3.1. Application possible

Définition 4.3.1 ([3]) **:** *Soient un w-PBP M* [3]*, $f : [w] \to [w]$ une bijection (différente de l'identité) et $E \subseteq \{0,1\}^n$. On dit que M f-reconnaît l'ensemble E si, pour $x \in \{0,1\}^n$:*

1. $x \in E \implies M(x) = f$,
2. $x \notin E \implies M(x) = \mathbb{1}$.

Aussi, on dit que M w-cycle reconnaît E s'il existe un w-cycle f tel que M f-reconnaît L. Une séquence $(M_n)_{n\in\mathbb{N}}$ de w_n-PBP permet de reconnaître un langage.

Théorème 4.3.2 ([3]) **:** *Tout langage dans NC^1 est 5-cycle reconnaissable par une séquence $(M_n)_{n\in\mathbb{N}}$ de 5-PBP où M_n est de longueur polynomiale en n.*

On remarque que si M est un w-PBP qui f-reconnaît un ensemble L où f est un w-cycle, on peut se servir de l'algorithme de test d'appartenance à un sous-groupe qui vient d'être exposé pour vérifier si $M(x) \stackrel{?}{=} f$. En effet, soit $G_w = \{g : [w] \to [w] \text{ bijectif }\}$ et $H_w = \{g : [w] \to [w] \text{ bijectif } \mid g(w) = w\} < G_w$, on a que $M(x) = \mathbb{1} \iff M(x) \in H_w$ ce qui peut être testé par un analyseur de taille $O(\lg |G_w : H_w|) = O(\lg (w!/(w-1)!)) = O(\lg w)$.

Ainsi, en s'inspirant du théorème 4.3.2, si on pouvait trouver des langages intéressants qui se reconnaissent significativement mieux par des w-PBP que par des 5-PBP, on pourrait s'en servir (comme gadgets et accompagnés d'analyseurs testant l'appartenance à des sous-groupes) pour construire des PBC efficaces.

3. Voir définition 3.4.1.

4.4. Composition

4.4.1. Composition de fonctions

Définition 4.4.1 : *Soient g et f des fonctions booléennes respectivement sur m et n bits. On définit :*

$$g \diamond f \colon \{0,1\}^{m \times n} \to \{0,1\}$$

$$(x_1, ..., x_{nm}) \mapsto g(f(x_1, ..., x_n), f(x_{n+1}, ..., x_{2n}), ..., f(x_{(m-1)n+1}, ..., x_{nm}))$$

4.4.2. Composition de programmes de branchement

Définition 4.4.2 : *Soit un programme de branchement (catalytique) B. Un sous-ensemble U de taille k des sommets de B satisfaisant aux conditions suivantes :*

1. $\exists i : \forall u \in U$ *les arcs sortants de u sont étiquetés par le i^e bit de l'entrée*
2. $\forall u, v \in U$*, il n'existe pas de chemin, valide ou non*[4]*, entre u et v*

est appelé une tranche de taille k de B.

Définition 4.4.3 : *Soit un programme de branchement (catalytique) B, une tranche U de taille k de B et un PBC B' avec k noeuds initiaux. La substitution de U par B', illustrée à la figure 6 page A-v, consiste à substituer dans B les sommets de la tranche U par une copie de B' dont les $2k$ noeuds finaux sont substitués par les successeurs*[5] *des sommets de la tranche U.*

Dans la définition précédente, la tranche U peut ne contenir qu'un sommet auquel cas B' est simplement un PB.

Définition 4.4.4 : *Soient B et B' deux PB respectivement sur m et n bits et B'' un m-uplet de PB sur n bits. Soient B'_j une copie de B' dont chaque étiquette d'arc "x_i" est changée pour "$x_{(n-1)j+i}$" et B''_j le j^e PB de B'' pour $j \in \{1, ..., m\}$.*

1. *On appelle $B \diamond B'$ le PB obtenu en substituant chaque noeud inspectant le j^e bit de l'entrée dans B par une copie de B'_j.*

4. voir définition 1.4.5.
5. v est un successeur de u dans B s'il y a un arc de u vers v.

2. *On appelle $B \circ B''$ le PB obtenu en substituant chaque noeud inspectant le j^e bit de l'entrée dans B par une copie de B''_j.*

Lemme 4.4.5 : *Soient B, B' et B'' de la définition précédente calculant respectivement les fonctions $g : \{0,1\}^m \to \{0,1\}$, $f : \{0,1\}^n \to \{0,1\}$ et $h : \{0,1\}^n \to \{0,1\}^m$. On a que :*

1. *$B \diamond B'$ calcule $g \diamond f$ et est de taille $|B| \cdot |B'|$,*
2. *$B \circ B''$ calcule $g \circ h$ et est de taille $\sum_{i=1}^{m} (\#$ de $"x_i"$ dans $B) \cdot |B''_j| \leq |B| \cdot \max |B''_j|$.*

Soient $g : \{0,1\}^m \to \{0,1\}$, $f : \{0,1\}^n \to \{0,1\}$ et $h : \{0,1\}^m \to \{0,1\}^m$ fixes pour lesquels on connaît B, B' et B'' de tailles optimales permettant leur calcul. On sait que $T(g \diamond f) \leq |\mathrm{B} \diamond \mathrm{B}'|$ et $T(g \circ h) \leq |\mathrm{B} \diamond \mathrm{B}''|$. Peut-on faire mieux ?

4.4.3. Calcul de la composition de fonctions

Voici des exemples de situation où on peut convertir une borne catalytique en borne sur la composition de fonctions de manière à faire mieux que le résultat précédent.

Soit $g : \{0,1\}^m \to \{0,1\}$ se calculant par un PB de taille t_g et par un PB indolent de taille t'_g dont chaque niveau est de largeur au moins l.

Soit $f : \{0,1\}^n \to \{0,1\}$, tel que $f^{||l}$ se calcule par un PBC indolent de taille amortie t'_f et $T(f) = t_f$.

Soit $h : \{0,1\}^n \to \{0,1\}^m$, $h(x) = (h_1(x), ..., h_m(x))$ où, pour tout $i \in \{1, ..., m\}$, on a que $h_i^{||l}$ se calcule par un PBC de taille amortie $\leq t'_h$ alors que $T(h_i) \in O(t_h)$.

La composition "$\circ$" du PB pour g avec les PB pour les h_i donne une borne immédiate de $O(t_g \cdot t_h)$ sur $\mathrm{T}(g \circ h)$. En utilisant la borne catalytique, la composition "$\circ$" du PB indolent pour g et des PBC indolents pour les $h_i^{||l}$ donne plutôt la borne $O(t'_g \cdot t'_h)$.

De même, la composition "$\diamond$" du PB pour g avec le PB pour f donne une borne immédiate de $O(t_g \cdot t_f)$ sur $\mathrm{T}(g \diamond f)$. En utilisant la borne catalytique, la composition "$\diamond$" du PB indolent pour g et des PBC indolents pour $f^{||l}$ donne plutôt une borne de $O(t'_g \cdot t'_f)$.

On note que le théorème 1.5.11 offre des exemples de fonctions g pour lesquelles le meilleur PB connu est indolent et que le théorème 4.1.3 offre des exemples de fonctions f pour lesquels $t'_f \leq t_f$ avec l petit.

L'intérêt d'utiliser des PBC pour améliorer les bornes connues sur la composition de fonctions est apparent lorsqu'on considère la composition itérée de fonctions. Soient f et l pour lesquels on a $\Delta = t_f/t'_f \gg 1$. On note $f^{(1)} = f$ et $f^{(k+1)} = f \diamond f^{(k)}$ pour $k \geq 1$. Utiliser la construction précédente permet de remplacer pour le calcul de $g \diamond f^{(k)}$ la borne immédiate $O(t_g \cdot t_f^k)$ par $O(t'_g \cdot t_f^k/\Delta^k)$.

4.4.4. Intérêt de la composition itérée de fonctions

Définition 4.4.6 : *La **profondeur** d'une fonction booléenne sur n bits, notée $PROF(f)$ est la plus petite profondeur d'un circuit booléen calculant f construit à partir de portes* ET *et* OU *de degré entrant 2 et de portes* NON.

Théorème 4.4.7 ([**9**]) **:** *Soit f une fonction booléenne aléatoire sur n bits, la proposition suivante :*

$$\exists \epsilon > 0 \ \forall m \geq n \ \forall g : \{0,1\}^m \to \{0,1\} : PROF(f \diamond g) \geq \epsilon \, PROF(f) + PROF(g)$$

implique la séparation suivante : $\mathsf{NC}^1/\mathsf{poly} \neq \mathsf{NC}^2/\mathsf{poly}$ [6].

En particulier, la conjecture suivante (ou n'importe quelle variation qui n'en atténue pas significativement la portée) suffit pour obtenir la séparation du théorème précédent.

Conjecture 4.4.8 ([**9**]) **:** *Soient f et g des fonctions booléennes non constantes respectivement sur n et m bits. On a que :*

$$PROF(g \diamond f) \simeq PROF(g) + PROF(f).$$

La preuve du théorème 4.4.7 tourne autour de l'idée suivante : pour $k = \lg n / \lg \lg n$ prendre un k-uplet $(f_i)_{i \in \{1,\ldots,k\}}$ de fonctions booléennes aléatoires sur $\lg n$ bits (et donc pour lesquels $\mathrm{PROF}(f_i) \in \Omega(\lg n)$) et montrer comment les limitations sur l'efficacité de la composition de fonctions permettent d'obtenir une borne $\omega(\lg n)$ sur la profondeur de $g = f_1 \diamond \ldots \diamond f_k$ (par construction g est une fonction booléenne sur n bits satisfaisant $\mathrm{PROF}(g) \in O(\lg^2 n)$).

6. Et donc en particulier $\mathsf{NC}^1/\mathsf{poly} \neq \mathsf{P}/\mathsf{poly}$.

4.4.5. Retour sur le calcul de la composition de fonctions

La sous-section précédente montre l'intérêt théorique de la complexité de composition de fonctions. À défaut de pouvoir directement adresser la conjecture 4.4.8, la sous-section 4.4.3 montre comment il est possible de se servir du calcul catalytique pour améliorer au moins *en taille* le calcul de la composition de fonctions.

Remarque 4.4.9 : *Par le théorème 1.5.4, il existe une valeur constante de c telle que pour toute fonction f_i sur $\lg n$ bits il existe un PBC B_i calculant f_i 2^n fois en taille amortie $\leq c \lg n$. Pour $k = \lg n / \lg\lg n$ la composition $B_1 \diamond ... \diamond B_k$ donne alors un PBC B pour le calcul de $g = f_1 \diamond ... \diamond f_k$ 2^n fois en taille amortie $\prod |B_i| = (c \lg n)^k = n^{1+\lg(c)/\lg\lg n} \in O(n^{1+\varepsilon})$ pour $\varepsilon > 0$ constant.*

Ainsi, même si g peut ne pas être calculable dans NC^1, il se calcule catalytiquement très efficacement (et pour un nombre de copies significativement inférieur à 2^{2^n}).

4.5. Algorithme général pour le calcul de fonctions booléennes

Définitions 4.5.1 : *Soient un ensemble S et $f : S \to \{0,1\}$, on définit le* ***support de f*** *comme étant $supp(f) = \{s \in S | f(s) = 1\}$. Aussi, on appelle* ***fonction élémentaire*** *toute fonction f pour laquelle $|supp(f)| = 1$.*

4.5.1. Analyseur pour fonctions élémentaires

Théorème 4.5.2 : *Soit G, un groupe de taille k. Tout k-uplet de fonctions élémentaires sur n bits peut être calculé*[7] *par un analyseur de taille $\Theta(\lg k)$*

Démonstration. Pour $i \in \{1, ..., \lceil \lg k \rceil\}$, Alice demande à Bob le i^e bit de p et vérifie que la réponse coïncide avec le i^e bit de l'unique p' tel que $f_s(p') = 1$. □

4.5.2. Quelques bornes supérieures sur le calcul d'une fonction arbitraire

Les résultats sont d'abord énoncés, la preuve du théorème 4.5.4 est reportée à la section 4.5.2.2.

7. Voir définition 3.2.6.

4.5.2.1. *Énoncés*

Théorème 4.5.3 : *Soit G, un groupe de taille k. Toute fonction $f : G \to \{0,1\}$ peut être calculée par un analyseur de taille $\Theta(m \lg k)$ où $m = |supp(f)|$.*

DÉMONSTRATION. Soit $supp(f) = \{p_i | i \in [m]\}$ et $f_i : G \to \{0,1\}$ où $f_i(p) = 1 \iff p = p_i$ se calculent par un analyseur de taille $\Theta(\lg k)$ par le théorème 4.5.2. f s'écrit alors simplement comme $f(s) = \bigvee f_i(s)$ ce qui donne la borne supérieure désirée. □

Le théorème suivant donne une borne amortie pour le calcul de n'importe quelle fonction pouvant être décomposée sous la forme $f : \{0,1\}^n \xrightarrow{z} G \xrightarrow{h} \{0,1\}$ où G est un groupe fini. La borne ne dépend que de la complexité de z et de la taille du groupe G.

Théorème 4.5.4 : *Soient G un groupe de taille k et B un G-gadget de longueur m. Toute fonction $f \in F_n$ qui peut être décomposée sous la forme $f = h \circ z_B$ où $h : G \to \{0,1\}$ et $z_B : \{0,1\}^n \to G$ est la fonction associée à B (voir définition 3.4.7) peut être calculée $k \cdot N$ fois par un PBC de taille $\Theta(m \cdot k \cdot N \cdot \lg k)$ (donc en taille amortie $\Theta(m \lg k)$) pour $N = 2^{2^{\lceil \lg k \rceil}} \leq 4^k$.*

Corollaire 4.5.5 : *Toute fonction symétrique sur $\{0,1\}^n$ peut être calculée nN fois par un PBC en taille amortie $\Theta(n \lg n)$ pour $N = 2^{2^{\lceil \lg n \rceil}} \leq 4^n$.*

DÉMONSTRATION. Soit le programme M de longueur n sur $\mathbb{Z}_n$ suivant :

$$< 1,1 >, < 2,1 >, ..., < n,1 >$$

et soit B le gadget associé. La fonction associée à B est $z_B(x) = |x|$. Or, toute fonction symétrique se décompose sous la forme $h \circ z_B = h(|\cdot|)$ avec $h : \mathbb{Z}_n \to \{0,1\}$. Une application du théorème 4.5.4 conclut la preuve. □

4.5.2.2. *Démonstration du théorème 4.5.4*

Définition 4.5.6 : *Soient un groupe G de taille k et $l = \lceil \lg k \rceil$. On appelle numérotation des éléments de G n'importe quelle fonction injective de la forme $\sigma : G \to \{0,1\}^l$.*

Définition 4.5.7 : *Soient $k \in \mathbb{N}$ et une fonction $f : A \to B^k$. Pour $i \in \{1, ..., k\}$ et $a \in A$, on note $f_i(a)$ la i^e composante de $f(a)$.*

Définition 4.5.8 : *Soient un groupe G de taille k et une numérotation σ de ses éléments. On note l'ensemble des éléments de G dont le i^e bit de la numérotation par σ est $b \in \{0,1\}$ comme suit :*

$$E^{\sigma}_{i,b} = \{g \in G \ | \sigma_i(g) = b \ \} = \sigma_i^{-1}(b) \ , \ pour \ 1 \leq i \leq \lceil \lg k \rceil.$$

Lemme 4.5.9 : *Soit un groupe G de taille k, $l = \lceil \lg k \rceil$ et une numérotation σ de ses éléments. On a que :*

$$\forall \alpha \in G \ \forall g \in G \ \exists ! x \in \{0,1\}^l : \{g\} = \bigcap_{i=1}^{l} \alpha E^{\sigma}_{i,x_i}.$$

Démonstration.

$$\forall i \in \{1,...,l\} \ \exists ! x_i \in \{0,1\} : \alpha^{-1} g \in E^{\sigma}_{i,x_i}$$

$$\implies \{\alpha^{-1} g\} \subseteq \bigcap_{i=1}^{l} E^{\sigma}_{i,x_i} = \sigma^{-1}(x_1,...,x_l)$$

or, par injectivité de σ, on a : $|\sigma^{-1}(x_1,...,x_l)| \leq 1$, ainsi :

$$\{\alpha^{-1} g\} = \bigcap_{i=1}^{l} E^{\sigma}_{i,x_i}$$

$$\implies \{g\} = \bigcap_{i=1}^{l} \alpha E^{\sigma}_{i,x_i}.$$

□

Définition 4.5.10 : *Soit un groupe G de taille k, $l = \lceil \lg k \rceil$, et une numérotation σ de ses éléments. On appelle l'$\alpha-$numérotation de g par σ, notée $\sigma_\alpha(g)$, la chaîne $x \in \{0,1\}^l$ telle que $\{g\} = \cap_{i=1}^{l} \alpha E^{\sigma}_{i,x_i}$. La preuve du lemme précédent donne $\sigma_\alpha(g) = \sigma(\alpha^{-1} g)$.*

On se sert de la notion d'$\alpha-$numérotation comme suit : à toute fonction $h : G \to \{0,1\}$, on associe une fonction booléenne h'_α sur $\{0,1\}^l$ qui interprète son entrée x comme l'$\alpha-$numérotation d'un élément g et calcule $h(g)$.

Définition 4.5.11 : *Soit un groupe G de taille k et $l = \lceil \lg k \rceil$. Pour tout $\alpha \in G$ et pour toute fonction $h : G \to \{0,1\}$, on définit la fonction suivante :*

$$h'_\alpha \colon \{0,1\}^l \to \{0,1\}$$
$$x \mapsto \begin{cases} 0 & , \text{ si } \sigma_\alpha^{-1}(x) = \emptyset \\ h(\sigma_\alpha^{-1}(x)) & , \text{ sinon} \end{cases}$$

Autrement dit, h'_α est construite de telle sorte que $h = (h'_\alpha \circ \sigma_\alpha)$.

Définition 4.5.12 : *Soit un groupe G de taille k, une numérotation σ de ses éléments et un G-gadget B de longueur m.*

On appelle D_j^σ l'analyseur suivant (voir figure 8 page A-vii) :

1. *Bob vérifie si $p \in E_{j,1}^\sigma$ et envoie la réponse à Alice,*
2. *Alice accepte ssi la réponse de Bob est oui.*

On note $C_{j,B}^\sigma = PBC(D_j^\sigma, B)$ (dont la taille est $\leq 5mk$). La fonction calculée par la source s de $C_{j,B}^\sigma$ est le j^e bit de $\sigma_s(z_B(x))$.

Les PBC $C_{j,B}^\sigma$ de la définition précédente servent à extraire les bits de la numérotation des éléments d'un groupe avec le détail suivant : chaque source du PBC utilise une numérotation distincte.

Démonstration du théorème 4.5.4. Par le théorème 1.5.4, toute fonction $h'_s : \{0,1\}^l \to \{0,1\}$ se calcule $N = 2^{2^l} \leq 2^{2k} = 4^k$ fois en taille $\leq 64Nl$. Appelons C_s le PBC effectuant ce calcul et C le PBC obtenu en mettant côte à côte les PBC C_s pour $s \in G$. C et C_s sont indolents.

On partitionne les noeuds de C en tranches comme ceci (où t est la hauteur de C) :

1. Pour tout $j \in [t]$, à l'étage j de C, disons que le bit qui y est inspecté est p_j, on prend un sommet de chaque C_s et on appelle la tranche ainsi constituée U_{p_j},
2. on répète jusqu'à ce que tout sommet appartienne à une tranche.

Cela fait, on substitue chaque tranche U_{p_j} par un PBC $C_{p_j,B}^\sigma$ et on appelle C′ et C′$_s$, respectivement, C et C_s après cette substitution. Cela a pour effet à chaque sommet de chaque C_s de remplacer l'inspection du p_j^e bit de l'entrée par le calcul du p_j^e bit de $\sigma_s(z_B(x))$.

Avant la substitution, C_s calculait f'_s. Après, chaque C'_s calcule plutôt la même fonction :

$$(h'_s \circ \sigma_s \circ z_B)(x) = (h \circ z_B)(x).$$

Chacune effectue ce calcul N fois, donc C' effectue ce calcul kN fois.

La substitution a pour effet de remplacer les tranches de taille k par des PBC de taille $\leq 5mk$, ainsi la taille de C' devient $\leq k \cdot (64Nl) \cdot \frac{5mk}{k} = 320mkNl$. Ce qui conclut la preuve.

Chapitre 5

Une borne inférieure sur la taille des analyseurs

On aborde dans ce chapitre la question des bornes inférieures sur la taille des analyseurs requis pour résoudre des problèmes de communication. On montre comment importer les notions de rectangles monochromatiques et d'ensembles trompeurs pour établir de telles bornes. Voir l'annexe B pour un rappel des définitions et des résultats de complexité de communication.

Définition 5.0.1 : *Soit G un groupe et $f : G \times G \to \{0,1\}$ quelconque. On note $TAD(f)$ la taille du plus petit analyseur pour f. On note également $TAN(f)$ (resp. $TAP(f)$) la taille du plus petit analyseur non déterministe (resp. probabiliste) pour f.*

Remarque 5.0.2 : *Soit G un groupe de taille k et B un G-gadget de longueur m dont la fonction associée est z_B. Toute fonction $f : G \to \{0,1\}$ peut être calculée par un analyseur de taille $\Theta(k)$. Alice connaît $s \in G$, Bob connaît $p \in G$ et Alice demande à Bob directement quelle est la valeur de p, avec k réponses possibles, puis calcule par elle-même $f(s^{-1}p)$. Par contre, cette construction évidente ne donne aucun gain par rapport au calcul non catalytique.*

Démonstration. Soit A l'analyseur construit de cette manière. On a que :

$$\frac{|\mathrm{PBC}(A,B)|}{k} \in \Theta(m \cdot k).$$

Or, soit $B_{\mathbb{1}}$ la restriction de B à la source $\mathbb{1}$. Pour $p \in G$, en substituant le puits p de $B_{\mathbb{1}}$ par a si $f(p) = 1$ et par r sinon, on obtient un PB valide, appelons-le D, calculant la même fonction et dont la taille est $O(m \cdot k)$.

De plus, il n'y a aucune raison de croire que D corresponde à une construction optimale. □

Remarque 5.0.3 : *Soient G un groupe et $f : G \times G \to \{0,1\}$. Tout protocole pour f peut être converti en analyseur pour f en doublant sa taille tout au plus.*

DÉMONSTRATION. En colorant tout noeud v étiqueté par une fonction a_v (resp. b_v) en noir (resp. rouge) et en interposant entre deux noeuds d'une même couleur un noeud de l'autre couleur, un protocole est converti en analyseur.

Il est à noter que la taille de l'analyseur ainsi obtenu peut être inférieure à celle de la construction évidente décrite à la remarque 5.0.2. □

Définition 5.0.4 : *Un chemin dans un DAG D est dit **complet** s'il commence par une source et se termine par un puits. On note Γ_D l'ensemble des chemins complets de D.*

Lemme 5.0.5 : *Tout DAG D de taille t vérifie $|\Gamma_D| \in O(2^t)$. Cette borne est optimale.*

DÉMONSTRATION. Tout chemin dans D correspond à un sous-ensemble des sommets de D. Ainsi, $\Gamma_D \leq |\mathcal{P}(V_D)| = 2^t$.

Soit le DAG suivant : $V_D = [t]$, $E_D = \{(i,j)|0 \leq i < j < t\}$. Dans D, tout sous-ensemble des sommets contenant les sommets 0 et $t-1$ correspond à un chemin complet, ainsi : $\Gamma_D = 2^{t-2}$. □

Lemme 5.0.6 : *Soit un analyseur A calculant une fonction $f : S \times P \to \{0,1\}$ quelconque. Γ_A induit une partition de $S \times P$ en rectangles monochromatiques.*

DÉMONSTRATION. En effet, si (s,p') et (s',p) empruntent un même chemin $\gamma \in \Gamma_A$ c'est que, pour tout noeud rouge u dans γ, p et p' appartiennent au même élément de la partition de P induite par les arcs sortants de u et *idem* pour les noeuds noirs. Ainsi, (s',p') emprunte aussi le chemin γ.

Soit $E_\gamma = \{(s,p) \in S \times P|\ (s,p)$ emprunte le chemin γ dans $A\ \}$. L'argument précédent montre que E_γ est un rectangle. De plus, E_γ est monochromatique car chaque paire $(s,p) \in E_\gamma$ termine son chemin par le même puits de A.

Enfin, $\{E_\gamma|\gamma \in \Gamma_A\}$ est une partition de $S \times P$, ce qui conclut la preuve. □

Théorème 5.0.7 : *Soient G un groupe et $f : G \times G \to \{0,1\}$. Si f possède un ensemble trompeur S de taille t alors $TAD(f) \geq \lg t$.*

DÉMONSTRATION. Soit A le plus petit analyseur pour f :

1. Par B.0.13, $\Gamma_A \geq |S| = t$.
2. Par le lemme 5.0.6, Γ_A induit une partition de $G \times G$ en rectangles monochromatiques.
3. Par le lemme 5.0.5, le plus petit DAG possédant t chemins complets a une taille $\Theta(\lg t)$.

Ainsi, la taille de A est bornée inférieurement par $\lg t$. □

Définition 5.0.8 : *On définit la fonction suivante :*

$$EQ_n : \mathbb{Z}_n \times \mathbb{Z}_n \to \{0,1\} \text{ où } EQ(x,y) = 1 \text{ ssi } x = y.$$

Lemme 5.0.9 : *([11]) EQ_n possède un ensemble trompeur de taille n.*

Corollaire 5.0.10 : *L'analyseur de la figure 7b page A-vi est optimal.*

DÉMONSTRATION. L'analyseur de la figure 7b (de taille $O(\lg k)$), appelons-le A, prend en entrée une paire $(s,p) \in \mathbb{Z}_k \times \mathbb{Z}_k$ et n'accepte que si $p \in \{f(s), ..., g(s)\}$ pour f, g au choix. Ainsi, le calcul de EQ_k se réduit à deux applications de A. Or, en appliquant le théorème 5.0.7 au lemme 5.0.9, on obtient une borne de $\Omega(\lg k)$ sur la taille de tout analyseur calculant EQ_k ce qui conclut la preuve. □

Conclusion

On rappelle compendieusement le contenu du présent mémoire. Au chapitre 3, on a d'abord introduit les notions de gadgets et d'analyseurs, en expliquant pour chacun la composante du calcul catalytique qu'il représente, et on a montré comment les combiner pour obtenir un PBC. On a introduit la notion de programme sur un groupe, comment s'en servir pour calculer et comment l'employer dans la construction d'un gadget. On a également montré comment exprimer un calcul transparent en terme de gadget. Au chapitre 4, on a d'abord montré une borne amortie $O(n \lg k)$ pour le calcul simultané de k fonctions $\mathrm{PSMS}_{v,n}^{t,k}$ en se servant des notions du chapitre 3. On a montré comment se servir d'un PBC pour faire le calcul approximé de fonctions seuils en taille linéaire par copie et pour un nombre linéaire de copies et on s'en est servi pour motiver l'introduction des notions de quasi-consistance et de non-inconsistance. On a montré l'existence d'un analyseur de taille $O(\lg |G : H|)$ pour résoudre le problème de communication $f : G \times G \to \{0,1\}$ où $f(s,p) = 1$ ssi $s^{-1}p \in H$ sur un groupe G avec sous-groupe H et discuté de l'utilisation potentielle d'un tel analyseur pour généraliser le résultat de Barrington sur la reconnaissance de langage par w-PBP. On a parlé de l'utilité des PBC pour le calcul de la composition de fonctions. On a montré une borne amortie $O(m \lg |G|)$ sur le calcul répété $\lg(|G|) \cdot 2^{2^{\lceil \lg |G| \rceil}}$ fois de toute fonction $f : \{0,1\}^n \to G \xrightarrow{h} \{0,1\}$ où h est calculable par un G-gadget de longueur m. En particulier, on a tiré comme corollaire de ce résultat une borne amortie $O(n \lg n)$ sur le calcul répété un nombre exponentiel de fois de toute fonction booléenne symétrique sur n bits. Au chapitre 5, on a montré que $\mathrm{TAD}(f) \geq \lg t$ pour $f : G \times G \to \{0,1\}$ ayant un ensemble trompeur de taille t et ainsi exemplifié la façon de récupérer des notions connues de théorie de la communication pour les appliquer aux analyseurs.

Finalement, on indique les futures pistes à explorer utilisant les notions introduites dans le présent mémoire.

5.1. Joindre des analyseurs efficaces au calcul transparent

Certains protocoles de communication peuvent être directement convertis en analyseurs sans accroitre leur complexité. Par exemple, soit X un ensemble fini quelconque avec $n = \lg |X|$ et EQ'_X le problème de communication suivant : Alice reçoit $a \in X$, Bob reçoit $b \in X$ et leur but est de vérifier si $a = b$. [**11**] donne une borne inférieure déterministe et non déterministe $\geq n$ sur la complexité de communication de EQ'_X. Par contre, [**14**] donne un protocole de communication probabiliste public à coût $O(1)$ avec probabilité d'erreur unilatérale bornée par une constante pour EQ'_X. L'idée étant de représenter a et b par des vecteurs dans $\mathbb{F}_2^n$, générer des vecteurs aléatoires $v_i \in \mathbb{F}_2^n$ pour $i \in \{1, ..., c\}$ et c constant, Alice et Bob calculent alors chacun de leur côté le produit scalaire (dans $\mathbb{F}_2$) de leur vecteur avec les v_i et concluent que $a = b$ si ces produits scalaires sont égaux.

Ce protocole de communication probabiliste public peut être converti en analyseur probabiliste de *taille* constante. Ainsi, si on connaît une machine réversible à compteur sur un anneau R effectuant le calcul transparent de $f(x)$ dans un des ses registres, on peut alors se servir de l'analyseur probabiliste pour EQ'_R pour vérifier que $f(x) = 0$. Par exemple, si f est un polynôme et que la condition d'acceptation sur x est d'être une racine de f, la capacité de générer des bits aléatoires peut être utilisée pour accélérer le calcul.

[**10**] demande ce que l'on peut dire au sujet du calcul catalytique probabiliste. On pense que l'intérêt de ce modèle est justifié par la différence de performance existant entre les protocoles de communication déterministes et probabilistes et par l'usage qu'il peut en faire. De plus, on note l'importance particulière du théorème B.0.20 dans le contexte du calcul catalytique. Il serait raisonnable à priori de croire qu'une machine catalytique en espace logarithmique ne peut pas simuler un protocole de communication probabiliste public car il lui faudrait garder en mémoire les bits aléatoires générés durant le protocole pour qu'ils puissent être utilisés par les deux parties et qu'elle n'a pas la mémoire suffisante pour le faire. Le théorème B.0.20 assure au contraire qu'un petit nombre de bits aléatoires suffit. Par exemple, pour un protocole de communication public $f : R^m \times R^m \to \{0, 1\}$ où

$(R, +, \times)$ est un anneau et m est borné polynomialement en $|R|$, $O(\lg \lg |R|)$ bits aléatoires suffisent.

5.2. Séparer CNL et CL

[6] remarque un lien entre la puissance des modèles déterministes et non déterministes de calcul catalytique et demande s'il serait possible d'établir un équivalent catalytique du théorème de Savitch. Il laisse aussi ouverte la question de savoir si $\mathsf{CNL} \neq \mathsf{CL}$. On remarque que si on pouvait identifier une séquence d'anneaux R_n et des problèmes de communication associés $f_n : R_n \times R_n \to \{0, 1\}$ pour lesquels on aurait $\mathrm{TAD}(f_n) \in \omega(n)$ et $\mathrm{TAN}(f_n) \in O(n)$, on pourrait se servir de la notion d'analyseur non déterministe introduite dans le présent mémoire dans la construction d'un langage qu'on puisse placer dans CNL, mais pas dans CL. Aussi, il serait envisageable d'exploiter la différence entre la complexité déterministe et non déterministe de certains problèmes de communications pour étayer la thèse qu'il ne peut pas exister d'équivalent catalytique du théorème de Savitch.

Bibliographie

[1] Farid Ablayev and Marek Karpinski. On the power of randomized ordered branching programs. *Electronic Colloquium on Computational Complexity - ECCC*, 5, 01 1998.

[2] L. Babai, P. Pudlák, V. Rödl, and E. Szemeredi. Lower bounds to the complexity of symmetric boolean functions. *Theoretical Computer Science*, 74(3) :313 – 323, 1990.

[3] David A. Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in nc1. *Journal of Computer and System Sciences*, 38(1) :150 – 164, 1987.

[4] Y. Breitbart, H. Hunt, and D. Rosenkrantz. On the size of binary decision diagrams representing boolean functions. *Theoretical Computer Science*, 145(1) :45 – 69, 1995.

[5] Harry Buhrman, Richard Cleve, Michal Koucký, Bruno Loff, and Florian Speelman. Computing with a full memory : Catalytic space. In *Proceedings of the Forty-sixth Annual ACM Symposium on Theory of Computing*, STOC '14, pages 857–866, New York, NY, USA, 2014. ACM.

[6] Harry Buhrman, Michal Koucký, Bruno Loff, and Florian Speelman. Catalytic space : Non-determinism and hierarchy. *Theory Comput. Syst.*, 62(1) :116–135, 2018.

[7] Alan Cobham. The recognition problem for the set of perfect squares. In *Proceedings of the 7th Annual Symposium on Switching and Automata Theory (Swat 1966)*, SWAT '66, pages 78–87, Washington, DC, USA, 1966. IEEE Computer Society.

[8] Vincent Girard, Michal Koucký, and Pierre McKenzie. Nonuniform catalytic space and the direct sum for space. 2015. https ://eccc.weizmann.ac.il/report/2015/138/.

[9] Mauricio Karchmer, Ran Raz, and Avi Wigderson. Super-logarithmic depth lower bounds via the direct sum in communication complexity. *Computational complexity*, 5 :191–204, 1995.

[10] Michal Koucký. Catalytic computation. *Bulletin of the EATCS*, 118, 2016.

[11] Eyal Kushilevitz and Noam Nisan. *Communication Complexity*. Cambridge University Press, New York, NY, USA, 1997.

[12] O.B. Lupanov. On the problem of realization of symmetric boolean functions by contact schemes. *Problems of Cybernetics*, 15 :85–99, 1965. (en russe).

[13] Ilan Newman. Private vs. common random bits in communication complexity. *Information Processing Letters*, 39(2) :67 – 71, 1991.

[14] Noam Nisan. The communication complexity of threshold gates. In *In Proceedings of "Combinatorics, Paul Erdos is Eighty*, pages 301–315, 1994.

[15] Sylvain Perifel. *Complexité Algorithmique*. Ellipses, 2014.

[16] Aaron Potechin. A note on amortized branching program complexity. In *Proceedings of the 32nd Computational Complexity Conference*, CCC '17, pages 4 :1–4 :12, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[17] Rakesh Kumar Sinha and Jayram S Thathachar. Efficient oblivious branching programs for threshold and mod functions. *Journal of Computer and System Sciences*, 55(3) :373 – 384, 1997.

[18] Ingo Wegener. *Branching Programs and Binary Decision Diagrams*, pages 19–44. 1987.

[19] Andrew Chi-Chih Yao. Some complexity questions related to distributive computing (preliminary report). In *Proceedings of the Eleventh Annual ACM Symposium on Theory of Computing*, STOC '79, pages 209–213, New York, NY, USA, 1979. ACM.

Annexe A

Figures

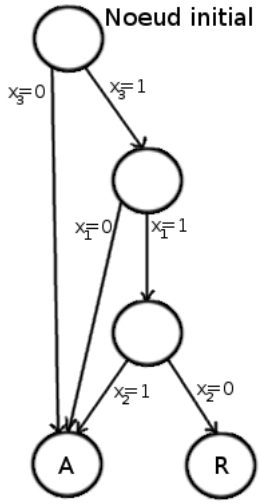
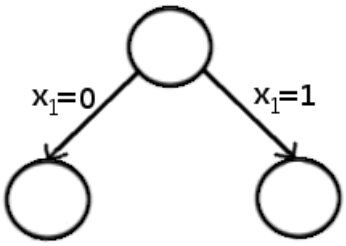
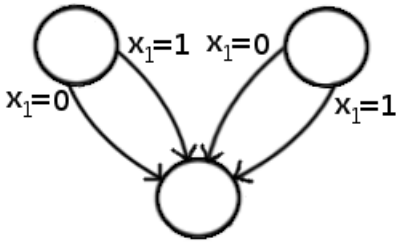


FIGURE 1. Exemple de programme de branchement calculant la valeur de vérité de la proposition "$x_3 \implies (x_1 \implies x_2)$".



(A) Exemple de gadget avec une source et deux puits.



(B) Inverse du gadget de gauche.

FIGURE 2. Un gadget dont l'inverse n'est pas un gadget.

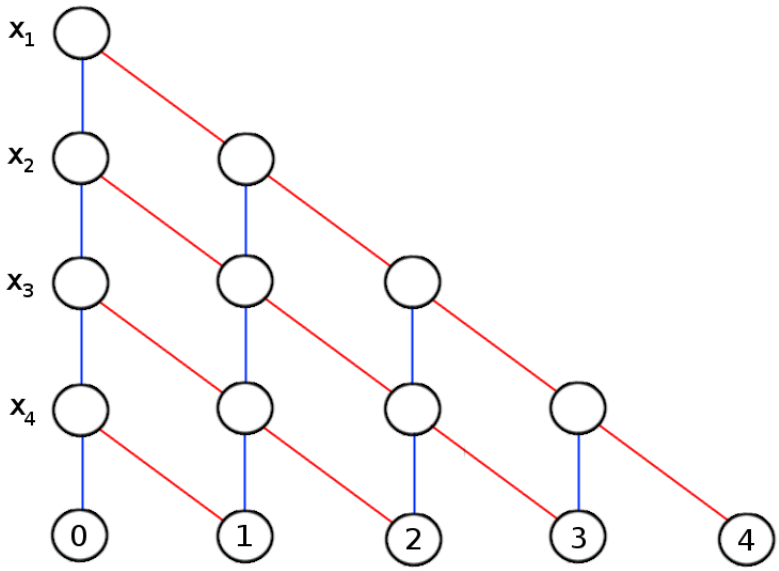


FIGURE 3. $\mathbb{Z}_5$-PB de l'exemple 3.1.2 où les arcs étiquetés "$x_j = 0$" et "$x_j = 1$" pointent vers le bas et sont représentés respectivement en bleu et en rouge et où les noeuds de l'étage j regardent le bit x_j.

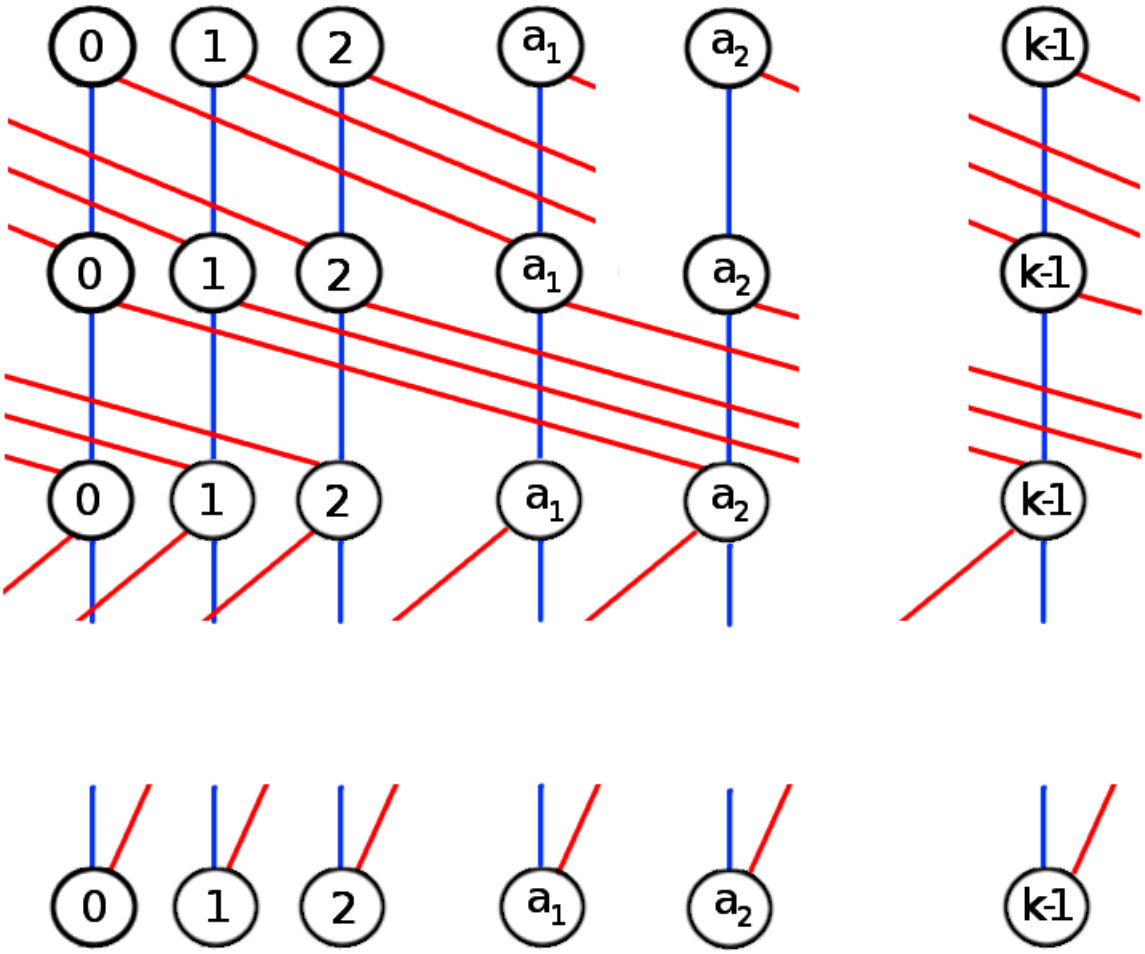
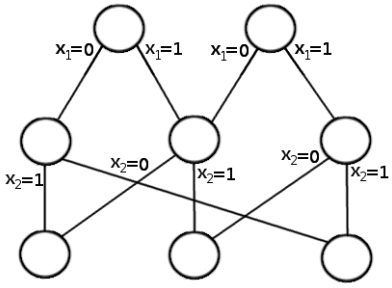
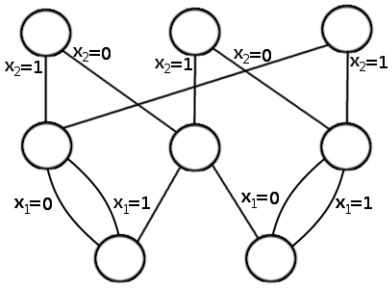


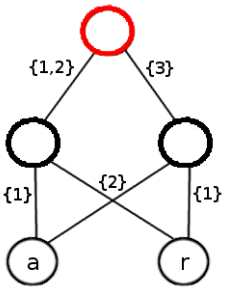
FIGURE 4. $\mathbb{Z}_n$-gadget du théorème 4.1.3 où les arcs étiquetés "$x_j = 0$" et "$x_j = 1$" sont représentés respectivement en bleu et en rouge et où les noeuds de l'étage j regardent le bit x_j. Les arcs pointent vers le bas.



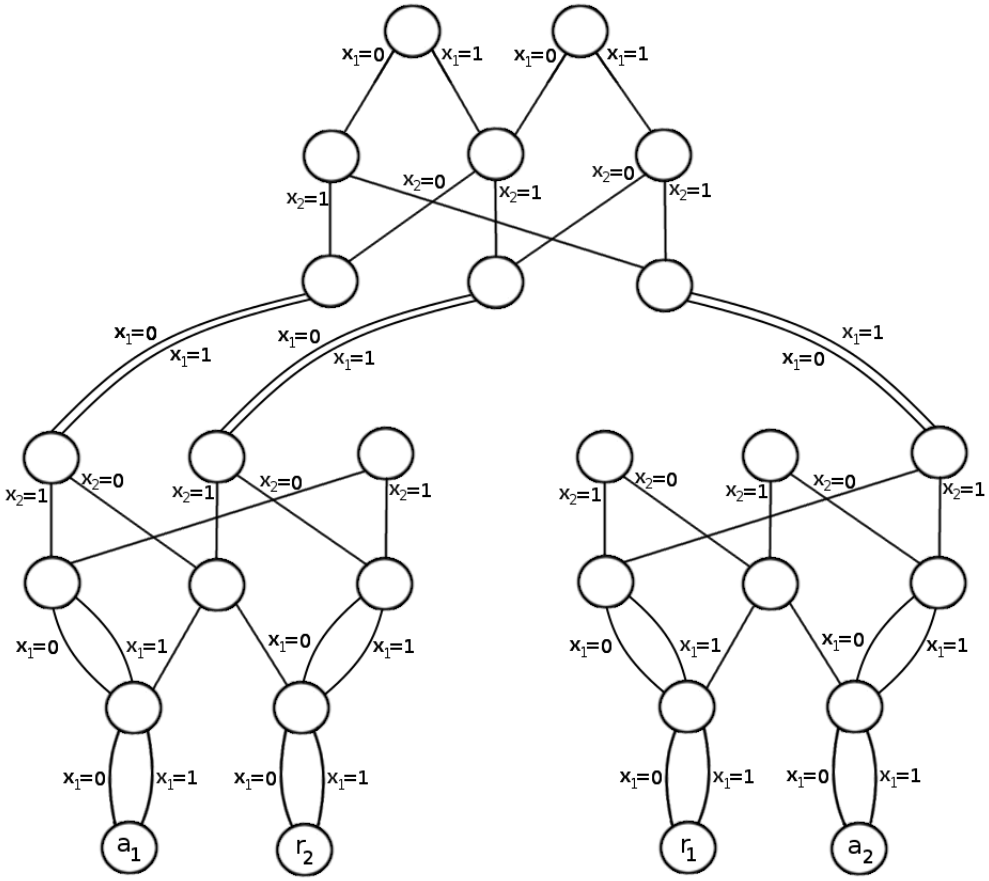
(A) Gadget B.



(B) Semi-gadget B^{-1}.

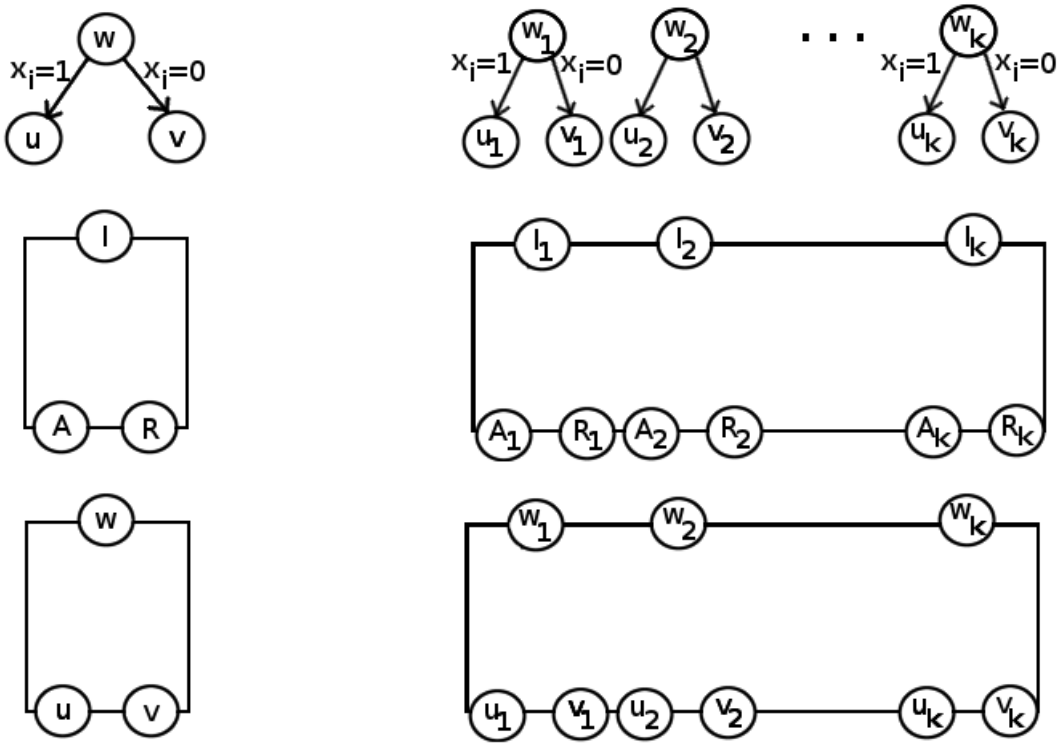


(C) Un analyseur A pour le gadget B.



(D) PBC(A,B)

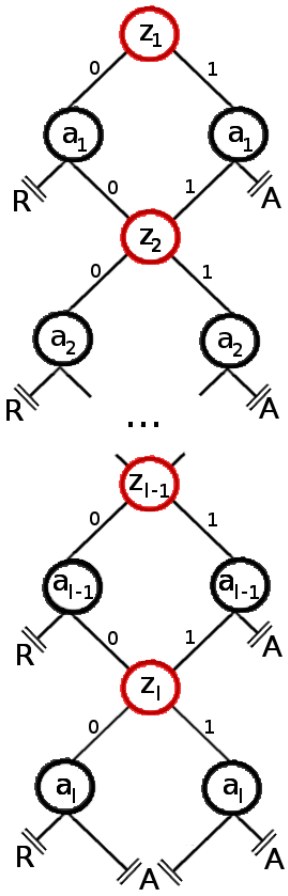
FIGURE 5. Exemple des différentes composantes d'un programme de branchement catalytique construit à partir d'un gadget et d'un analyseur.



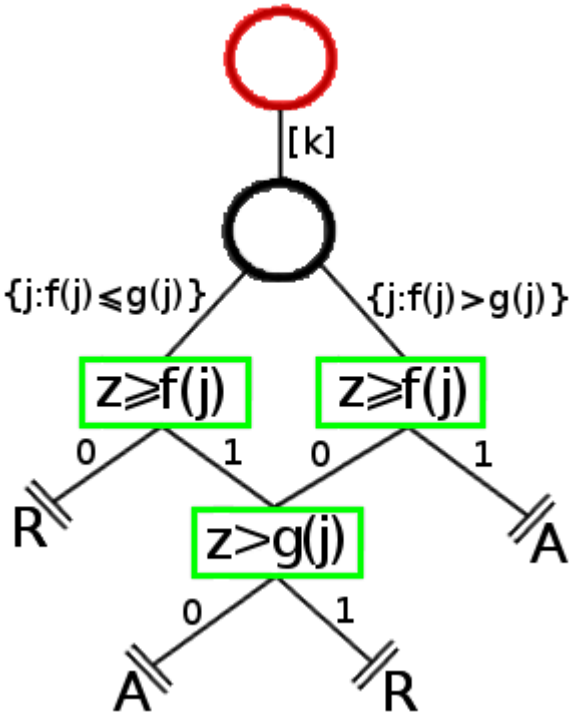
(A) De haut en bas : un noeud d'un PB, un PB avec noeud initial I et noeuds finaux A et R et le résultat de la substitution du premier par le deuxième.

(B) De haut en bas : une tranche de taille k d'un PB, un PBC à k noeuds initiaux et le résultat de la substitution du premier par le deuxième.

FIGURE 6. Représentation graphique de substitutions.



(A) Analyseur pour le calcul du test $z \geq f(s)$ où $a_i = c$ représente l'ensemble $\{s \in \mathbb{Z}_k | \text{le i}^{me} \text{ bit de } f(s) \text{ est c}\}$, où $z_i = c$ représente l'ensemble $\{z \in \mathbb{Z}_k | \text{Le i}^{me} \text{ bit de z est c}\}$ où $f : \mathbb{Z}_k \to \mathbb{Z}_k$ est une fonction quelconque et où l est la longueur de k en binaire.



(B) Analyseur pour le calcul du test $p \in \{f(s), ..., g(s)\}$ où les noeuds carrés "$p \geq f(s)$" et "$p < g(s)$" sont des instances de l'analyseur de la figure de gauche.

FIGURE 7. Construction en deux étapes d'un analyseur vérifiant que $p \in \{t_s + s, ..., k - 1 + s\}$ en prenant $f(s) = t_s + s$ et $g(s) = k - 1 + s$.

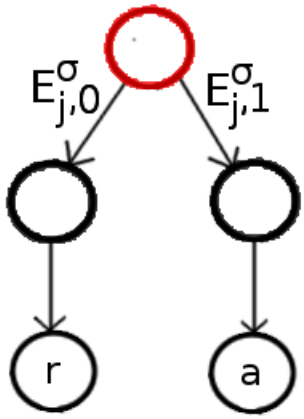


FIGURE 8. Analyseur D^{σ}_{j} de la définition 4.5.12 page 46.

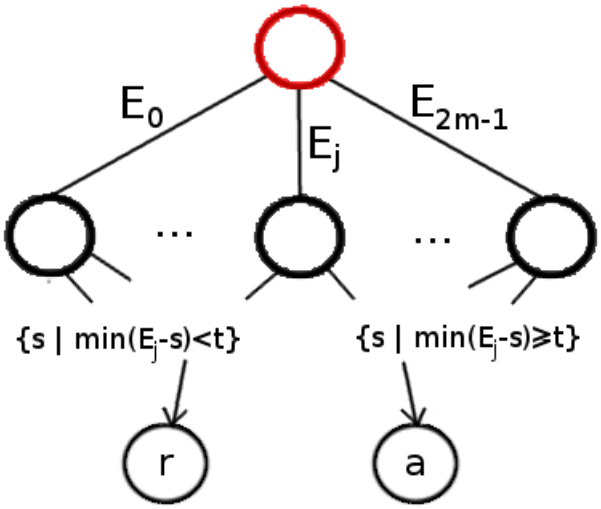


FIGURE 9. Analyseur du théorème 4.2.1 page 34.

Annexe B

Notions de complexité de la communication

Les définitions et les résultats présentés ici sont tirés directement de [**11**], les preuves sont omises. Le contexte est le suivant : Alice et Bob veulent calculer $f(x, y)$ où $f : X \times Y \to Z$, Alice possède $x \in X$ et Bob $y \in Y$. Le défi est de réaliser le calcul en échangeant le moins d'informations possible.

Définition B.0.1 ([**19**]) **:** *Un* ***protocole*** P *de* $X \times Y$ *vers* Z *est un arbre binaire où chaque noeud interne* v *est étiqueté avec une fonction* $a_v : X \to \{0, 1\}$ *ou* $b_v : X \to \{0, 1\}$ *et chaque feuille, par un élément de* Z*. La valeur du protocole* P *sur l'entrée* (x, y) *est l'étiqueté de la feuille atteinte en traversant l'arbre comme suit, en commençant par la racine de l'arbre : chaque fois qu'on atteint un sommet* v*, s'il est étiqueté* a_v *(resp.* b_v*) on calcule* $a_v(x)$ *(resp.* $b_v(y)$*), si la réponse est 0, on suit l'arc de gauche, sinon on suit l'arc de droite.*

Le coût du protocole sur l'entrée (x, y) *est la longueur du chemin traversé par cette entrée. Le coût du protocole* P *est la hauteur de l'arbre.*

Définition B.0.2 : *La* ***complexité déterministe de communication*** *d'une fonction* $f : X \times Y \to Z$*, noté* $D(f)$ *est le plus petit coût d'un protocole calculant* f*.*

Définition B.0.3 : *Soit* P *un protocole sur* $X \times Y$ *et* v *un noeud de* P*. On définit* R_v *comme l'ensemble des entrées* (x, y) *passant par* v*.*

Proposition B.0.4 : *Soient* P *un protocole sur* $X \times Y$ *et* L *l'ensemble de ses feuilles.* $\{R_l\}_{l \in L}$ *est une partition de* $X \times Y$*.*

Définition B.0.5 : *Un* ***rectangle*** *dans $X \times Y$ est un sous-ensemble $R \subseteq X \times Y$ tel que $R = A \times B$ pour $A \subseteq X$ et $B \subseteq Y$.*

Proposition B.0.6 : *$R \subseteq X \times Y$ est un rectangle ssi :*

$$(x, y') \in R \wedge (x', y) \in R \implies (x', y') \in R.$$

Proposition B.0.7 : *Soit P un protocole, pour tout sommet (noeud interne ou feuille) v de P R_v est un rectangle.*

Définition B.0.8 : *Soit $f : X \times Y \to Z$ fixe, un sous-ensemble $E \subseteq X \times Y$ est dit* ***monochromatique*** *si f est constante sur E.*

Lemme B.0.9 : *Soit $f : X \times Y \to Z$ fixe, tout protocole P induit une partition de $X \times Y$ en rectangles monochromatiques. Le nombre de rectangles étant le nombre de feuilles de P.*

Remarque B.0.10 : *La réciproque du lemme précédent n'est pas vraie, i.e. il existe des ensembles X et Y, des fonctions $: X \times Y \to Z$ et des partitions de $X \times Y$ en rectangles monochromatiques auxquels aucun protocole ne correspond.*

Corollaire B.0.11 : *Soit $f : X \times Y \to Z$ fixe, si toute partition de $X \times Y$ requiert au moins t rectangles monochromatiques, alors $D(f) \geq \lg t$.*

Définition B.0.12 : *Soit $f : X \times Y \to Z$ fixe, un sous-ensemble $E \subset X \times Y$ est appelé un* ***ensemble trompeur*** *s'il existe $z \in Z$ tel que :*

1. *$\forall (x, y) \in E$, $f(x, y) = z$,*
2. *$\forall (x, y'), (x', y) \in E$, on a que $f(x, y) \neq z$ ou $f(x', y') \neq z$.*

Lemme B.0.13 : *Si $f : X \times Y \to Z$ possède un ensemble trompeur de taille t, alors toute partition de $X \times Y$ requiert au moins ce nombre de rectangles monochromatiques.*

Définition B.0.14 : *Pour $b \in \{0, 1\}$ et $f : X \times Y \to \{0, 1\}$, $C^b(f)$ est le nombre de rectangles monochromatiques requis pour couvrir les entrées de f égales à b.*

Définition B.0.15 : *La complexité de communication non déterministe d'une fonction $f : X \times Y \to \{0, 1\}$ est $N^1(f) = \lg C^1(f)$.*

Définition B.0.16 : *Dans un protocole probabiliste (privé) P, en plus de recevoir x et y Alice et Bob reçoivent des chaînes aléatoires indépendantes r_a et r_b. Dans l'arbre binaire définissant P, les étiquettes des noeuds sont des fonctions binaires de (x, r_a) et (y, r_b). Comme dans le cas déterministe, à chaque 4-uplet (x, y, r_a, r_b) est associé une feuille de l'arbre. Le protocole peut prendre des valeurs différentes selon la valeur de r_a et r_b et peut se tromper.*

Un protocole probabiliste[1] *est dit public si on fixe $r_a = r_b$.*

Définition B.0.17 : *Un protocole probabiliste (public ou privé) P calcule une fonction f avec erreur (bilatérale) ε si $\Pr[P(x, y) = f(x, y)] \geq 1 - \varepsilon$. P calcule une fonction f avec erreur unilatérale ε si :*

1. *$\forall (x, y) : f(x, y) = 0$ on a que $\Pr[P(x, y) = 0] = 1$,*
2. *$\forall (x, y) : f(x, y) = 1$ on a que $\Pr[P(x, y) = 1] \geq 1 - \varepsilon$.*

Définition B.0.18 : *Soit un protocole probabiliste (public ou privé) P. Le coût en pire cas de P sur (x, y) est le nombre maximum de bits communiqués par Alice et Bob sur l'entrée (x, y, r_a, r_b). Le coût en pire cas de P est le plus grand coût en pire cas de P sur une entrée (x, y).*

Définition B.0.19 : *Pour $f : X \times Y \to \{0, 1\}$ et $\varepsilon \in (0, 1)$, on définit :*

1. *$R_\varepsilon(f)$ est le coût en pire cas minimum d'un protocole privé calculant f avec erreur (bilatérale) ε,*
2. *$R^1_\varepsilon(f)$ est le coût en pire cas minimum d'un protocole privé calculant f avec erreur unilatérale ε.*

On définit de la même façon $R^{pub}_\varepsilon(f)$ et $R^{1,pub}_\varepsilon(f)$.

Théorème B.0.20 : *$\forall \varepsilon > 0$ et $\forall \delta > 0$ tout protocole probabiliste public P calculant $f : \{0, 1\}^n \times \{0, 1\}^n \to \{0, 1\}$ avec erreur bilatérale ε (utilisant n'importe quel nombre de bits aléatoires) peut être converti en protocole probabiliste public P' calculant f avec erreur bilatérale $\varepsilon + \delta$ en concervant le même coût en pire cas, mais en n'utilisant que $O(\lg n + \lg(\delta^{-1}))$ bits aléatoires.*

1. Cette définition de protocole probabiliste public est tirée de [**13**] plutôt que de [**11**].