Université de Montréal

Complexité de variantes d'un problème P-complet

Par

Jean-Claude Desrosiers

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

Mémoire présenté en vue de l'obtention du grade de maîtrise ès sciences (M. Sc.)
en informatique, option théorique et quantique

Août 2026

Université de Montréal

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

Ce mémoire intitulé

Complexité de variantes d'un problème P-complet

présenté par

Jean-Claude Desrosiers

a été évalué par un jury composé des personnes suivantes

Louis Salvail

Président rapporteur

Pierre McKenzie

Directeur de recherche

Geňa Hahn

Membre du jury

Résumé

Un exemplaire du problème algébrique Gen est une opération binaire sur les entiers de 1 à n, encodée par un tableau $n \times n$. Étant donné que 1 est *générable*, la question est de décider si n est générable, où un élément est générable s'il est le résultat de l'opération appliquée à deux éléments générables.

Ce mémoire démontre la complétude de plusieurs variantes naturelles du problème Gen pour des classes de complexité incluses dans P. En développant la notion de témoin d'un exemplaire de Gen – un objet analogue à l'arbre de preuve (*proof tree*) pour un circuit booléen – on établit le résultat principal de ce mémoire, soit la complétude d'une variante nommée Gen-1 pour la classe des langages décidables en espace non-déterministe logarithmique. Gen-1 empêche tout élément, sauf 1, d'apparaître plus d'une fois dans le tableau. On propose aussi une variante portant sur la hauteur du témoin d'un exemplaire, qu'on fait correspondre à la profondeur d'un circuit alternant à arité d'entrée semi-illimitée (ou *semi-unbounded fan-in alternating circuit*), ainsi qu'une autre qui utilise un graphe orienté et est complète pour NP.

Ces preuves reposent en partie sur une révision du circuit multiplex, un modèle de circuit où chaque porte sélectionne un de ses arguments de données selon l'argument guide. De passage, on retravaille de manière plus explicite la preuve connue de l'équivalence entre les langages décidés par un automate à pile auxiliaire et la classe LOGDCFL

En conclusion on propose, en s'appuyant sur nos résultats, des pistes de recherche : un candidat LOGDCFL-complet, une définition d'uniformité de famille de circuits multiplex exigeant l'ordre topologique et une piste pour analyser le lien entre LOGDCFL et NL.

Mots-clés : automate à pile auxiliaire (auxpda), arbre de preuve, circuit alternant, circuit booléen, circuit multiplex, espace non-déterministe logarithmique, logdcfl, problème gen, temps polynomial, théorie de la complexité structurelle

Abstract

An instance of the algebraic problem Gen is a binary operation on integers between 1 and n, encoded by an $n \times n$ table. Given that 1 is *generable*, the question is to decide whether n is generable, where an element is generable if it is the result of the operation applied to two generable elements.

This thesis therefore establishes the completeness of several natural variants of the Gen problem for complexity classes contained in P. Using the witness of an instance of Gen – an object analogous to the proof tree of a boolean circuit – we establish a variant named Gen-1 as a new problem complete for the class of languages decidable in nondeterministic logarithmic space. Gen-1 forbids any element, except 1, from appearing more than once in the table. We also propose a variant concerning the height of the witness of an instance, which we match with the depth of a semi-unbounded fan-in alternating circuit, as well as another that uses a graph and is complete for NP.

These proofs rely in part on a revision of the multiplex circuit, a circuit model in which each gate selects one of its data inputs according to the steering bits. This also allows us to revisit, more formally, a proof of equivalence between the set of languages decided by an auxiliary pushdown automaton and the class LOGDCFL.

In conclusion, building on our results, we propose avenues for future research: a candidate LOGDCFL-complet problem, a definition of uniformity for multiplex circuit families requiring topological order, and an avenue for investigating the link between LOGDCFL and NL.

Keywords: alternating circuits, auxiliary pushdown automaton (auxpda), boolean circuit, gen problem, logdcfl, multiplex circuit, non-deterministic log-space, polynomial time, proof tree, structural complexity theory

Table des matières

Résumé 5

Abstract 7

Liste des figures 11

Liste des sigles et des abréviations 13

Remerciements 15

Chapitre 1. Introduction 17

Chapitre 2. Préliminaires 19
- 2.1. Structures Discrètes 19
- 2.2. Machines de Turing 20
- 2.3. Automates à Pile 22
- 2.4. Circuits Booléens 23

Chapitre 3. Circuits Multiplex 25
- 3.1. Classes de Complexité Multiplex 26
- 3.2. Les AuxDPDA Capturent LOGDCFL 31

Chapitre 4. Gen, un Problème Algébrique 43
- 4.1. Saut de NL à P entre Gen-1 et Gen-2 44
- 4.2. Comparer un Témoin de Gen avec les Circuits Booléens 51
- 4.3. GenPath Complet pour NP 57

Chapitre 5. Conclusion 61
- 5.1. Pistes Futures 61

Références bibliographiques 63

Liste des figures

3.1 Une porte multiplex 25

3.2 Simulation d'une porte juxtaposition avec $m - 1 \in \mathcal{O}(1)$ portes multiplex 30

3.3 Circuit partiel avec chaque configuration $k_l = (t_l\,,\, i_l\,,\, x_{i_l}\,,\, r_l\,,\, j_l\,,\, q_l\,,\, Z_l)$ représentant une étape de calcul, où selon la valeur i du guide g et h l'argument de donnée d_i et c_i est sélectionné. (g et d_i [resp. h et c_i]) sont des identifiants de portes [resp. constantes]) 35

Liste des sigles et des abréviations

AuxDPDA: AuxPDA déterministe. 9, 17, 22, 23, 31, 61

AuxNPDA: AuxPDA non-déterministe. 23, 52, 53

AuxPDA: un automate à pile avec un ruban de travail additionnel (*auxiliary pushdown automaton*). 23, 31, 41

DCFL: language hors-contexte déterministe (*deterministic context-free language*). 17

DPDA: automate à pile déterministe (*deterministic pushdown automaton*). 22, 36

mT: machine de Turing. 21, 22, 23, 29, 31, 45, 46, 51, 56

mTd: mT déterministe. 20, 21, 54, 58

mTnd: mT non-déterministe. 21, 57

PDA: automate à pile (*pushdown automaton*). 22

Remerciements

Je tiens d'abord à remercier Pierre, mon directeur de recherche, pour sa patience, ses précieux conseils et courriels, toujours appréciés, qui ont souvent permis de remettre mes réflexions sur la bonne voie. Je suis également reconnaissant pour son catalogue mental de références, apparemment inépuisable, ainsi que pour son humour, qui a rendu le parcours beaucoup plus agréable.

Je remercie aussi ma famille pour son soutien, ses encouragements et sa confiance tout au long de ce projet. Leur présence a été précieuse, particulièrement dans les moments où l'avancement du mémoire semblait plus difficile.

Enfin, je tiens à remercier ma partenaire pour sa patience et son soutien indéfectible, surtout durant les périodes où l'écriture du mémoire a fait de moi une version nettement moins agréable de moi-même. Merci de m'avoir encouragé et de m'avoir rappelé qu'il existait une vie en dehors de ce mémoire.

Chapitre 1

Introduction

La question du lien entre P et NP [Coo71b] est fondamentale en théorie de la complexité et d'envergure comparable à celle qui la précède de quelques années, toujours sans réponse à ce jour, du lien entre L, la classe des langages décidables en espace logarithmique, et P [Coo69]. En effet, résoudre cette dernière éclairerait la relation entre l'espace et le temps. Évidemment, la problématique a évolué depuis. Notamment, un candidat potentiel pour séparer L de LOGDCFL – le sous-ensemble de P formé des langages réductibles en espace logarithmique à un DCFL – a été introduit dans [Bra+09]. Ce candidat est TreeEval, qui s'énonce comme suit : calculer la valeur de la racine d'un arbre où les sommets internes sont des fonctions sur entiers, les feuilles des entiers et la valeur d'un sommet est l'évaluation de sa fonction sur la valeur de ses enfants.

La motivation initiale de ce projet de recherche ayant été d'explorer la complexité de TreeEval, le langage complet pour P, Gen [JL76] s'est imposé en tant que problème intéressant en relation avec TreeEval ainsi qu'en soi. Simplement, un exemplaire du problème algébrique Gen est une opération binaire sur les entiers de 1 à n, encodée par un tableau $n \times n$. Étant donné que 1 est générable, la question est de décider si n est générable, où un élément est *générable* s'il est le résultat de l'opération appliquée à deux éléments générables. Ce mémoire démontre donc la complétude de plusieurs variantes naturelles du problème Gen pour des classes de complexité incluses dans P.

Le deuxième chapitre est un préliminaire, qui établit la notation ainsi que les termes et définitions utilisés tout au long du document.

Dans le troisième chapitre, on traite du circuit multiplex, introduit dans [FLR96] et [Rei97], où les sommets du circuit sont des portes multiplex chacune ayant un argument guide qui indique lequel des arguments de données est utilisé comme sortie. La simplification du modèle présentée dans ce chapitre nous permet de revisiter la preuve par [MRV99] d'équivalence entre la classe des languages décidés par un AuxDPDA et LOGDCFL.

Le quatrième chapitre est le cœur de ce mémoire. On y traite de plusieurs variantes du problème Gen qui sont complètes pour des classes allant de NL – les langages décidables en espace non-déterministe logarithmique – à P. Le résultat principal concerne une variante qui restreint les dupliqués dans le tableau représentant l'opération. En effet, si l'on impose la restriction que seul l'entier 1 peut apparaître plus d'une fois dans le tableau, on obtient Gen-1, un nouveau problème algébrique complet pour NL. Tout aussi surprenant est que la variante devient complète pour P si un doublon de chaque entier est autorisé.

D'autres variantes étudiées restreignent le témoin pour un exemplaire de Gen, un objet analogue à l'arbre de preuve (*proof tree*) pour un circuit booléen. En particulier, on démontre l'équivalence entre la hauteur d'un témoin et la profondeur du circuit alternant à arité d'entrée semi-illimitée (ou *semi-unbounded fan-in alternating circuit*) – un circuit booléen où les portes conjonction sont bornées, mais les portes disjonctions ne le sont pas – qui le décide. Vers la fin du chapitre, on propose GenPath un problème d'accessibilité de graphe qui s'inspire de Gen où l'on exige qu'un arc étiquetté puisse être ajoutée au chemin seulement si le sommet associé à cette étiquette est dans le préfixe du chemin. On montre que ce langage est NP-complet.

Dans le chapitre final, on propose, en s'appuyant sur nos résultats, des pistes de recherche : un candidat LOGDCFL-complet, une définition d'uniformité de famille de circuits multiplex exigeant l'ordre topologique et une piste pour analyser le lien entre LOGDCFL et NL.

Chapitre 2

Préliminaires

Note 2.1. Sauf indication contraire, on suppose dans le texte qu'une fonction est toujours définie sur l'entièreté de son domaine (i.e. c'est une application).

Note 2.2. Sauf indication contraire, on suppose dans le texte qu'une fonction est $\mathbb{N} \to \mathbb{N}$, avec $\mathbb{N} = \{1, 2, \ldots\}$.

Définition 2.3. L'ensemble $[n]$ est $\{1, 2, \ldots, n\}$.

2.1. Structures Discrètes

On rappelle plusieurs notions de bases, qu'on peut trouver, par exemple, dans [Ros02, Chapitre 7].

Note 2.4. Soit $G = (S, A)$ un graphe orienté [resp. non-orienté], alors S est son ensemble de sommets et A est l'ensemble des arcs [resp. arêtes] tel que (s_i, s_j) indique un arc du sommet s_i au sommet s_j [resp. $\{s_i, s_j\}$ indique une arête entre les sommets s_i et s_j] avec $s_i \neq s_j$.

Définition 2.5. Soit $G = (S, A)$ un graphe orienté [resp. non-orienté], un *chemin* [resp. une *chaîne*] de s_0 à s_n dans G est une suite finie d'arcs $a_1 = (s_0, s_1), a_2 = (s_1, s_2), \ldots, a_n = (s_{n-1}, s_n)$ [resp. arêtes $a_1 = \{s_0, s_1\}, a_2 = \{s_1, s_2\}, \ldots, a_n = \{s_{n-1}, s_n\}$]. Lorsque les arcs [resp. arêtes] $a_1, a_2, \ldots, a_n$ sont distincts, le chemin [resp. la chaîne] est *simple*.

Note 2.6. Sauf indication contraire, on suppose par la suite qu'un chemin [resp. une chaîne] est simple.

Définition 2.7. Soit $G = (S, A)$ un graphe orienté [resp. non-orienté] et un chemin [resp. une chaîne] sur les sommets $s_0, s_1, \ldots, s_n$. Le chemin [resp. la chaîne] est un *circuit* [resp. *cycle*] si $s_0 = s_n$.

Définition 2.8. Soit $G = (S, A)$ un graphe orienté [resp. non-orienté] et C un chemin [resp. une chaîne] qui passe par tous les sommets (i.e. $\forall s \in S, \exists a \in A$ tel que $s \in a$), alors C est *hamiltonien*[ne].

Définition 2.9. Soit $G = (S, A)$ un graphe orienté, le *demi-degré* intérieur [resp. extérieur] d'un sommet $s \in S$ est $\deg^-(s)$ [resp. $\deg^+(s)$] le nombre d'arcs qui ont s comme destination [resp. source], c'est-à-dire $|\{ (s_i, s_j) \in A : s_j = s \}|$ [resp. $|\{ (s_i, s_j) \in A : s_i = s \}|$]. Le *degré* d'un sommet $s \in S$ est $\deg(s) = \deg^-(s) + \deg^+(s)$.

Définition 2.10. [Joh17, Chapitre 9] Une *arborescence* est un couple $T = (G, s_0)$ où $G = (S, A)$ est un arbre (graphe non-orienté connexe et acyclique) et $s_0 \in S$ est la racine. Soit une chaîne sur les sommets $s_0, s_1, \ldots, s_n$, plusieurs définitions suivent :

- i est la *profondeur* $p(s_i)$ du sommet s_i.
- La *hauteur* $h(T)$ est la profondeur maximale d'un de ses sommets.
- s_{i-1} est le *parent* de s_i.
- Si s est le parent de s', alors s' est un *enfant* de s.
- $s_0, s_1, \ldots, s_{i-1}$ sont les *ancêtres* de s_i.
- Si s est l'un des ancêtres de s', alors s' est l'un des *descendants* de s.
- Si s n'a pas d'enfants, alors c'est une *feuille*, sinon c'est un sommet *interne*.
- La *sous-arborescence* de T de racine s' est l'arborescence $T' = (G', s')$ telle que G' est le sous-graphe de G qui contient s' et tous ses descendants.

Définition 2.11. [Joh17, Chapitre 9] Une arborescence est *binaire* si chaque sommet a au plus deux enfants et si on précise, pour chaque enfant, si c'est l'enfant gauche ou droit. En plus, une arborescence binaire est *entière* si chaque sommet interne a exactement deux enfants.

2.2. Machines de Turing

On rappelle plusieurs notions de bases, qu'on peut trouver, par exemple, dans [Pér14].

Définition 2.12. $M = (\Sigma, \Gamma, B, Q, q_0, q_a, q_r, \delta)$ est une mTd à $k \geq 2$ rubans où :

- Σ est l'alphabet d'entrée, un ensemble fini non vide
- Γ est l'alphabet de travail, un ensemble fini tel que $\Sigma \subset \Gamma$
- $B \in \Gamma \setminus \Sigma$ est le symbol blanc
- Q est l'ensemble d'états, un ensemble fini
- $q_0 \in Q$ est l'état initial
- $q_a, q_r \in Q$ sont des états terminaux (respectivement d'acceptation et de refus)
- $\delta : (Q \setminus \{q_a, q_r\}) \times \Gamma^{k-1} \to Q \times \Gamma^{k-1} \times \{\mathrm{G}, \mathrm{S}, \mathrm{D}\}^k$ est la fonction de transition

$L(M)$ est le langage de M, l'ensemble des mots sur lesquels M s'arrête dans l'état d'acceptation. Pour $x \in \Sigma^*$ un mot sur lequel M s'arrête, $M(x) \in \Gamma^*$ est le mot écrit sur le ruban de sortie lorsque M s'arrête.

Définition 2.13. $N = (\Sigma, \Gamma, B, Q, q_0, q_a, q_r, \delta)$ est une mTnd à $k \geq 2$ rubans où la seule différence avec une mTd est :

$-$ $\delta : (Q \setminus \{q_a, q_r\}) \times \Gamma^{k-1} \to \mathcal{P}\left(Q \times \Gamma^{k-1} \times \{\mathrm{G}, \mathrm{S}, \mathrm{D}\}^k\right)$ est la relation de transition

$L(N)$ est le langage de N, l'ensemble des mots sur lesquels il existe un choix de transitions tels que N s'arrête dans l'état d'acceptation.

Note 2.14. Sauf indication contraire, on suppose dans le texte qu'une mT est déterministe, a $k = 3$ rubans et s'arrête sur toutes ses entrées (peu importe le choix des transitions, s'il y a lieu).

Définition 2.15. Soit f une fonction, alors $\mathsf{DTemps}(f)$ [resp. $\mathsf{DEspace}(f)$] est l'ensemble des langages décidés par une mT M telle que $\exists \alpha \in \mathbb{N}, \forall x \in \Sigma^*$, M sur x fonctionne en temps au plus $\alpha f(|x|)$ [resp. utilise au plus $\alpha f(|x|)$ espace sur chaque ruban de travail].

Si F est un ensemble de fonctions, alors $\mathsf{DTemps}(F)$ est $\bigcup_{f \in F} \mathsf{DTemps}(f)$ [resp. $\mathsf{DEspace}(F)$ est $\bigcup_{f \in F} \mathsf{DEspace}(f)$].

Définition 2.16. Soit f une fonction, alors $\mathsf{NTemps}(f)$ [resp. $\mathsf{NEspace}(f)$] est l'ensemble des langages décidés par une mTnd N telle que $\exists \alpha \in \mathbb{N}, \forall x \in \Sigma^*$, N sur x fonctionne en temps au plus $\alpha f(|x|)$ [resp. utilise au plus $\alpha f(|x|)$ espace sur chaque ruban de travail] peu importe le choix de transitions.

Si F est un ensemble de fonctions, alors $\mathsf{NTemps}(F)$ est $\bigcup_{f \in F} \mathsf{NTemps}(f)$ [resp. $\mathsf{NEspace}(F)$ est $\bigcup_{f \in F} \mathsf{NEspace}(f)$].

Définition 2.17. On note quelques classes de complexités utiles : $\mathsf{NP} = \mathsf{NTemps}\left(n^{\mathcal{O}(1)}\right)$, $\mathsf{P} = \mathsf{DTemps}\left(n^{\mathcal{O}(1)}\right)$, $\mathsf{L} = \mathsf{DEspace}(\mathcal{O}(\log(n)))$ et $\mathsf{NL} = \mathsf{NEspace}(\mathcal{O}(\log(n)))$.

Définition 2.18. Soit Σ et Γ des alphabets, $A \subseteq \Sigma^*$ et $B \subseteq \Gamma^*$ des langages et M une mTd avec alphabets d'entrée Σ et de travail Γ telle que $\forall x \in \Sigma^*, x \in A \iff M(x) \in B$, alors A ***se réduit à*** B, via la réduction M, qu'on note aussi $A \leq_M B$.

Définition 2.19. $A \leq_{\log} B$ indique que $\exists M$ une mTd telle que $A \leq_M B$ et $\exists \alpha \in \mathbb{N}, \forall x \in \Sigma^*$, M sur x utilise au plus $\alpha \log(|x|)$ espace sur chaque ruban de travail.

Définition 2.20. Soit A un langage, $\mathscr{C}$ un ensemble de langages et $\leq_M$ une réduction alors $A \in \mathscr{C}$-difficile selon $\leq_M$ si $\forall B \in \mathscr{C}$, $B \leq_M A$. Si, en plus, $A \in \mathscr{C}$ alors $A \in \mathscr{C}$-complet selon $\leq_M$.

Définition 2.21. Soit $\mathscr{C}$ un ensemble de langages, alors co-$\mathscr{C}$ est l'ensemble $\{ A : \overline{A} \in \mathscr{C} \}$.

2.3. Automates à Pile

On rappelle plusieurs notions de bases, qu'on peut trouver, par exemple, dans [Sip12].

Définition 2.22. $A = (\Sigma , \Pi , Z_0 , Q , q_0 , q_a , \delta)$ est un PDA où :

- Σ est l'alphabet d'entrée, un ensemble fini non vide
- Π est l'alphabet de pile, un ensemble fini
- $Z_0 \in \Pi \setminus \Sigma$ est le symbole initial de pile
- Q est l'ensemble d'états, un ensemble fini
- $q_0 \in Q$ est l'état initial
- $q_a \in Q$ est l'état d'acceptation
- $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \times \Pi \to \mathcal{P} (Q \times (\Pi \cup \{\varepsilon\}))$ est la relation de transition

$L (A)$ est le langage de A, l'ensemble des mots sur lesquels il existe un choix de transitions tels que A se déplace uniquement de gauche à droite et s'arrête dans l'état d'acceptation.

Définition 2.23. $A = (\Sigma , \Pi , Z_0 , Q , q_0 , q_a , \delta)$ est un DPDA où la seule différence avec un PDA est :

- $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \times \Pi \to Q \times (\Pi \cup \{\varepsilon\})$ est la fonction de transition
- $\forall q \in Q, Z \in \Pi$, si $\exists a \in \Sigma : \delta (q , a , Z) \neq \emptyset$ alors $\delta (q , \varepsilon , Z) = \emptyset$

$L (A)$ est le langage de A, l'ensemble des mots sur lesquels A se déplace uniquement de gauche à droite et s'arrête dans l'état d'acceptation.

Définition 2.24. CFL [resp. DCFL] est l'ensemble des langages décidés par un PDA [resp. DPDA].

Définition 2.25. [Sud78][Yam23] LOGCFL [resp. LOGDCFL] est l'ensemble des langages A tel que $A \leq_{\log} B \in$ CFL [resp. DCFL].

Définition 2.26. [Coo71a] $A = (\Sigma , \Gamma , B , \Pi , Z_0 , Q , q_0 , q_a , q_r , \delta)$ est un AuxDPDA où la seule différence avec une mT est :

- Π est l'alphabet de pile, un ensemble fini
- $Z_0 \in \Pi \setminus \Sigma$ est le symbole initial de pile
- $\delta : (Q \setminus \{ q_a , q_r \}) \times \Sigma \times \Gamma \times \Pi \to Q \times \Gamma \times (\Pi \cup \{\varepsilon\}) \times \{ \mathrm{G} , \mathrm{S} , \mathrm{D} \}^2$ est la fonction de transition

$L (A)$ est le langage de A, l'ensemble des mots sur lesquels A s'arrête dans l'état d'acceptation.

Définition 2.27. $N = (\Sigma, \Gamma, B, \Pi, Z_0, Q, q_0, q_a, q_r, \delta)$ est un AuxNPDA où la seule différence avec un AuxDPDA est :

- $\delta : (Q \setminus \{q_a, q_r\}) \times \Sigma \times \Gamma \times \Pi \to \mathcal{P}\left(Q \times \Gamma \times (\Pi \cup \{\varepsilon\}) \times \{\mathrm{G}, \mathrm{S}, \mathrm{D}\}^2\right)$ est la relation de transition

$L(N)$ est le langage de N, l'ensemble des mots sur lesquels il existe un choix de transitions tels que N s'arrête dans l'état d'acceptation.

Note 2.28. Sauf indication contraire, on suppose dans le texte qu'un AuxPDA est déterministe.

Définition 2.29. Soit f_1 et f_2 des fonctions, alors $\mathsf{DAuxPDA}(f_1, f_2)$ [resp. $\mathsf{NAuxPDA}(f_1, f_2)$] est l'ensemble des langages décidés par un AuxDPDA M [resp. AuxNPDA N] telle que $\exists \alpha_1, \alpha_2 \in \mathbb{N}, \forall x \in \Sigma^*$ M sur x [resp. N sur x] fonctionne en temps au plus $\alpha_1 f_1(|x|)$ et utilise au plus $\alpha_2 f_2(|x|)$ espace sur son ruban de travail [resp. peu importe le choix des transitions] (sans compter l'espace de la pile).

Soit F_1 et F_2 des ensembles de fonctions, alors $\mathsf{DAuxPDA}(F_1, F_2)$ est $\bigcup_{f_1 \in F_1, f_2 \in F_2} \mathsf{DAuxPDA}(f_1, f_2)$ [resp. $\mathsf{NAuxPDA}(F_1, F_2)$ est $\bigcup_{f_1 \in F_1, f_2 \in F_2} \mathsf{NAuxPDA}(f_1, f_2)$].

2.4. Circuits Booléens

Encore une fois, on peut trouver les notions de bases qui suivent dans [Pér14].

Définition 2.30. Soit $(C_n)_{n \in \mathbb{N}}$ une famille de circuits booléens, alors $(C_n)_{n \in \mathbb{N}}$ est *uniforme* s'il existe une mT M qui calcule $M(1^n) = \langle C_n \rangle$ – la description du circuit – telle que $\exists \alpha \in \mathbb{N}, \forall x \in \Sigma^*$, M sur x utilise au plus $\alpha \log(|x|)$ espace sur chaque ruban de travail.

Définition 2.31. Soit C_n un circuit booléen et s un sommet dans C_n, alors la *hauteur* de s est la longueur du plus long chemin de s à une entrée ou une constante

Définition 2.32. Soit C_n un circuit booléen, alors la *taille* de C_n, $T(C_n)$, est le nombre de sommets dans le circuit et sa *profondeur*, $P(C_n)$, est la hauteur de la porte résultat du circuit.

Définition 2.33. Soit $\alpha \in \mathbb{N}$, alors l'ensemble des langages décidés par une famille de circuits booléens $(C_n)_{n \in \mathbb{N}}$ uniforme telle que $\exists f_1 \in n^{\mathcal{O}(1)}, f_2 \in \mathcal{O}(\log^\alpha(n))$ avec $\forall n \in \mathbb{N}, T(C_n) \leq f_1(n)$ et $P(C_n) \leq f_2(n)$ est :

- NC^α si les portes $\wedge$ et $\vee$ sont d'arités constantes (la classe de Nick ou *Nick's class*)
- SAC^α si les portes $\wedge$ sont d'arités constantes et $\vee$ d'arités illimitées (circuit alternant à arité semi-illimitée ou *semi-unbounded fan-in alternating circuit*)
- AC^α si les portes $\wedge$ et $\vee$ sont d'arités illimitées (circuit alternant ou *alternating circuit*)

Chapitre 3

Circuits Multiplex

Définition 3.1. [FLR96], [MRV99] Soit $(C_n)_{n\in\mathbb{N}}$ une famille de circuits et $k, l \in \mathcal{O}(\log(n))$ des fonctions, alors, dans un circuit C_n, une *porte multiplex* (voir Figure 3.1) a comme arguments un paquet de $k(n)$ bits guides et $2^{k(n)}$ paquets de $l(n)$ bits de données ; sa seule sortie est un paquet de $l(n)$ bits. Le calcul de la porte décode le nombre j (où $0 \leq j \leq 2^{k(n)} - 1$) représenté en binaire dans le paquet de bits guides, puis passe le j-ème paquet de bits de données en sortie.

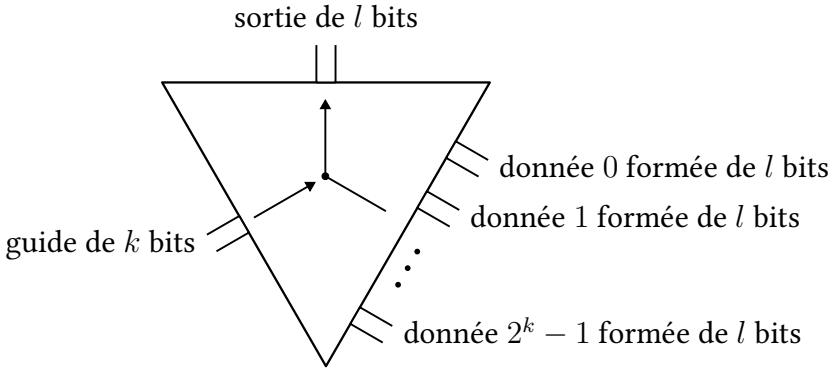

Figure 3.1 – Une porte multiplex

Définition 3.2. [MRV99] *Un circuit multiplex* C_n est un circuit booléen dont les sommets sont des entrées ou des portes multiplex. Une *entrée* n'a pas d'argument, seulement une sortie de taille 1. L'unique *porte résultat* est spéciale car ses paquets de bits de données sont de taille $l = 1$ et sa sortie n'est utilisée comme argument d'aucune porte. Le calcul de C_n sur $x = x_1x_2\ldots x_n \in \{0, 1\}^n$ s'effectue en calculant la sortie de chaque porte, considérant que les entrées du circuit $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$ ont respectivement une valeur $x_1, x_2, \ldots, x_n$. Ainsi, C_n accepte x si et seulement si la sortie de la porte résultat est 1.

Dans un circuit multiplex on ne peut combiner plusieurs paquets de bits en un, ni séparer un paquet de bits en plusieurs ; donc le paquet de bits en sortie d'un sommet doit servir, dans son entièreté, comme seul argument (guide ou donnée) d'une autre porte. La seule exception est qu'on

peut ajouter $\mathcal{O}(\log(n))$ bits constants au début (i.e. bits moins significatifs) de tout paquet de bits du circuit C_n d'une famille $(C_n)_{n\in\mathbb{N}}$.

Définition 3.3. Un *circuit multiplex fort* C_n est un circuit multiplex où l'on permet deux nouveaux types de porte :

- Une porte *juxtaposition* avec arguments $m \in \mathcal{O}(1)$ paquets de bits $b_1, b_2, \ldots, b_m$ respectivement de tailles $t_1, t_2, \ldots, t_m \in \mathcal{O}(\log(n))$ et les juxtapose en un seul paquet de bits $b_1 b_2 \ldots b_m$ de taille $\mathcal{O}(\log(n))$ en sortie.
- Une porte *extraction* avec arguments un paquet de bits b de taille $t \in \mathcal{O}(\log(n))$ et deux paquets de bits constants b_1, b_2 respectivement de tailles $t_1, t_2 \in \mathcal{O}(\log(n))$, avec $t_1 \leq t_2 \leq t$. La sortie est un paquet de bits de taille $t_2 - t_1 + 1$, constitué des bits indicés t_1 à t_2 du paquet de bits b.

3.1. Classes de Complexité Multiplex

On utilise la même notion d'uniformité que pour une famille de circuits booléens. Dans le cas d'une famille de circuits multiplex $(C_n)_{n\in\mathbb{N}}$, la description du circuit $\langle C_n \rangle$ est une liste de la description de chaque sommet dans C_n avec celle de la porte résultat en premier. Chaque description d'un sommet est dotée d'un identifiant numérique unique qui est un mot binaire. Chaque entrée $\mathbf{x}_i$ est décrite ainsi :

$$\mathbf{x}(\ s\#i\ )$$

où s est l'identifiant numérique du sommet et i est écrit en binaire. Et chaque porte est décrite comme suit :

$$\mathbf{m}(\ s\#g\#h\#d_0\#c_0\#d_1\#c_1\#\ldots\#d_{2^k-1}\#c_{2^k-1}\ )$$

où g est l'identifiant numérique du sommet dont la sortie est utilisée comme paquet de bits guides, h les bits constants du paquet de bits guides, d_j l'identifiant numérique du sommet dont la sortie est utilisée comme le j-ème paquet de bits de données et c_j les bits constants du j-ème paquet de bits de données.

On utilise la même notion d'uniformité pour une famille de circuits multiplex forts. La seule différence est la description qui doit inclure les deux nouveaux types de porte ; la porte juxtaposition :

$$\mathbf{j}\,(\ s\#b_1\#c_1\#b_2\#c_2\#\ldots\#b_m\#c_m\ )$$

où b_j est l'identifiant numérique du sommet dont la sortie est utilisée comme le j-ème paquet de bits d'entrées et c_j les bits constants du j-ème paquet de bits d'entrées, puis la porte extraction :

$$\mathbf{e}\,(\ s\#b\#c\#b_1\#b_2\ )$$

où b est l'identifiant numérique du sommet dont la sortie est utilisée comme argument b, c la partie constante de ce dernier, b_1 et b_2 les paquets de bits constants.

Avec la porte résultat comme racine, un circuit multiplex [resp. fort] n'est pas un arbre dans le cas général, car la sortie d'un sommet peut servir comme argument à plus d'une porte. Il est possible de démêler le circuit, c'est à dire, de dupliquer chaque sommet et son sous-circuit autant de fois que sa sortie est utilisée comme argument d'une autre porte. Voici un exemple de pseudo-algorithme qui, avec les sommets d'un circuit multiplex comme entrée, donne l'arbre dont la racine est la porte résultat,

1: **tant que** $\exists s$: NombreParent $(s) > 1$ **faire** ▷ Parcourt les sommets en ordre d'identifiants numériques
2: $p \leftarrow$ Parent (s) ▷ Un des parents de s
3: $s' \leftarrow$ Duplique (s) ▷ Duplique s et son sous-circuit, avec de nouveaux identifiants numériques
4: Argument s de $p \leftarrow s'$ ▷ Dans p, remplacer l'identifiant numérique de s par s'
5: **fin tant que**

Définition 3.4. [MRV99] *L'arbre de preuve* du circuit multiplex [resp. fort] C_n sur entrée $x \in \{0\,,\,1\}^n$ est le résultat de démêler le circuit, puis de supprimer les sommets qui ne contribuent pas au calcul de C_n sur x. Ainsi, on ne conserve que le sommet qui calcule le paquet de bits guides et celui qui calcule le paquet de bits de données qui est sélectionné par les bits guides, et ce, pour chaque porte conservée en commençant par la porte résultat.

La taille de l'arbre de preuve de C_n, $TAP\,(C_n)$, est la taille maximale ($\forall x \in \{0\,,\,1\}^n$) d'un arbre de preuve sur x.

Définition 3.5. La taille du circuit multiplex [resp. fort] C_n, $T\,(C_n)$, est définie identiquement aux circuits booléens, c'est le nombre de sommets dans le circuit.

Définition 3.6. [MRV99] Soit f_1 et f_2 des fonctions, alors $\mathsf{MuxTapT}\,(f_1\,,\,f_2)$ [resp. $\mathsf{MuxFTapT}\,(f_1\,,\,f_2)$] est l'ensemble des langages décidés par une famille de circuits multiplex

[resp. forts] $(C_n)_{n\in\mathbb{N}}$ uniforme telle que $\exists\alpha_1,\alpha_2 \in \mathbb{N}, \forall n \in \mathbb{N}, TAP(C_n) \leq \alpha_1 f_1(n)$ et $T(C_n) \leq \alpha_2 f_2(n)$.

Soit F_1 et F_2 des ensembles de fonctions, alors $\mathsf{MuxTapT}(F_1, F_2)$ est $\bigcup_{f_1\in F_1, f_2\in F_2} \mathsf{MuxTapT}(f_1, f_2)$ [resp. $\mathsf{MuxFTapT}(F_1, F_2)$ est $\bigcup_{f_1\in F_1, f_2\in F_2} \mathsf{MuxFTapT}(f_1, f_2)$].

Proposition 3.7. Soit f_1 et f_2 des fonctions, alors $\mathsf{MuxTapT}(f_1, f_2) \supseteq \mathsf{MuxFTapT}(f_1, f_2)$.

Démonstration. Soit $Y \in \mathsf{MuxFTapT}(f_1, f_2)$, alors $\exists (C_n)_{n\in\mathbb{N}}$ une famille de circuits multiplex forts uniforme telle que $\exists\alpha_1,\alpha_2 \in \mathbb{N}, \forall n \in \mathbb{N}, TAP(C_n) \leq \alpha_1 f_1(n)$ et $T(C_n) \leq \alpha_2 f_2(n)$. On veut montrer $\exists (C'_n)_{n\in\mathbb{N}}$ une famille de circuits multiplex uniforme qui décide aussi Y, telle que $Y \in \mathsf{MuxTapT}(f_1, f_2)$.

Il faut donc un algorithme qui calcule la description d'un circuit multiplex C'_n équivalent à un circuit multiplex fort C_n dont la description est donnée en entrée. L'algorithme remplace chaque porte de C_n par une ou plusieurs portes multiplex équivalentes.

On remplace une porte juxtaposition avec arguments $b_1, b_2, \ldots, b_m$ de tailles $t_1, t_2, \ldots, t_m$ par une série de $m-1$ portes multiplex (voir Figure 3.2). Chaque porte multiplex sélectionne le paquet de bits de données qui contient les bits constants dont la valeur est égale à celle du paquet de bits guides. Ainsi, la porte juxtapose le paquet de bits guides au paquet de bits qui est dupliqué sur chaque argument de donnée. Pour ce qui est de l'identifiant numérique de la porte juxtaposition, celui-ci est copié pour la dernière porte multiplex et les précédentes obtiennent de nouveaux identifiants. Donc, si la sortie de la porte juxtaposition était utilisée dans C_n, elle le sera de la même façon dans C'_n.

On remplace une porte extraction avec argument b de taille t et deux autres arguments de tailles t_1 et t_2 par une porte multiplex. Le paquet b sert de paquet de bits guides et chaque paquet de bits de données est un paquet de $t_2 - t_1 + 1$ bits constants. Sur le j-ème paquet de bits de données, on a j en binaire, mais seulement les bits d'indices de t_1 à t_2 (inclusivement). Pour ce qui est de l'identifiant numérique, il est simplement copié tel quel.

Maintenant qu'on a que $(C'_n)_{n\in\mathbb{N}}$ décide aussi Y, on veut que la famille soit uniforme. L'algorithme doit donc fonctionner en espace $\mathcal{O}(\log(n))$.

On effectue le remplacement de la porte juxtaposition une porte multiplex à la fois. Pour écrire la description de la porte multiplex qui a b_i ($2 \leq i \leq m$) comme paquet de bits guides, il suffit d'écrire b_i et b_{i-1} aux bons endroits et de compter sur $t_i \in \mathcal{O}(\log(n))$ bits. Au total, le remplacement utilise donc $\mathcal{O}(m\log(n)) = \mathcal{O}(\log(n))$ espace.

Pour effectuer le remplacement d'une porte extraction, il suffit d'écrire b au bon endroit et de compter sur $t \in \mathcal{O}(\log(n))$ bits pour seulement copier les bits de t_1 à t_2 à chaque nombre, donc en espace $\mathcal{O}(\log(n))$.

Soit une mT qui, sur entrée 1^n, calcule bit à bit la description de C_n (qui s'effectue en espace $\mathcal{O}(\log(n))$, étant donné l'uniformité de $(C_n)_{n\in\mathbb{N}}$). Pour chaque bit, on réutilise l'espace pour calculer C'_n avec l'algorithme de remplacement décrit précédemment. Ainsi, $(C'_n)_{n\in\mathbb{N}}$ est uniforme.

Par la suite, on doit montrer des bornes sur $TAP(C'_n)$ et $T(C'_n)$ telles qu'on puisse affirmer $Y \in \mathsf{MuxTapT}(f_1, f_2)$.

Soit $m_{max} \in \mathcal{O}(1)$ le maximum du nombre d'arguments d'une porte juxtaposition dans $(C_n)_{n\in\mathbb{N}}$. Le remplacement d'une porte juxtaposition dans C_n par des portes multiplex dans C'_n augmente la taille d'arbre de preuve d'au plus $m_{max} - 1$. En effet, peu importe l'entrée x, l'arbre de preuve pour C'_n ne conserve qu'une seule copie de chaque porte multiplex parmi celles qui remplacent une porte juxtaposition. Pour ce qui est de la taille du circuit, celle-ci augmente aussi d'au plus $m_{max} - 1$.

Le remplacement de la porte extraction dans C_n par une porte multiplex dans C'_n ne change pas la taille du circuit car on remplace un sommet par un autre. Il en est de même pour la taille d'arbre de preuve car l'unique argument non-constant de la porte extraction est utilisé une seule fois comme argument de la porte multiplex. Ainsi, la duplication pour calculer la $TAP(C'_n)$ produit le même résultat que pour $TAP(C_n)$.

Avec $\alpha'_1 = \alpha_1(m_{max}-1)$ et $\alpha'_2 = \alpha_2(m_{max}-1)$, on a donc que $(C'_n)_{n\in\mathbb{N}}$ est une famille de circuits multiplex uniforme qui décide Y telle que $\forall n \in \mathbb{N}, TAP(C'_n) \leq \alpha'_1 f_1(n)$ et $T(C'_n) \leq \alpha'_2 f_2(n)$. □

Corollaire 3.8. Soit F_1 et F_2 des ensembles de fonctions, alors $\mathsf{MuxTapT}(F_1, F_2) = \mathsf{MuxFTapT}(F_1, F_2)$.

Démonstration. $\subseteq$: $Y \in \mathsf{MuxTapT}(F_1, F_2)$ implique que $\exists f_1 \in F_1, f_2 \in F_2$ telles que $Y \in \mathsf{MuxTapT}(f_1, f_2)$. Parce qu'une famille de circuits multiplex uniforme est aussi une famille de circuits multiplex forts uniforme avec la même taille d'arbre de preuve et taille, on a que $Y \in \mathsf{MuxFTapT}(f_1, f_2) \subseteq \mathsf{MuxFTapT}(F_1, F_2)$.

$\supseteq$: $Y \in \mathsf{MuxFTapT}(F_1, F_2)$ implique que $\exists f_1 \in F_1, f_2 \in F_2$ telles que $Y \in \mathsf{MuxFTapT}(f_1, f_2)$. Par le lemme 3.7, on a que $Y \in \mathsf{MuxTapT}(f_1, f_2) \subseteq \mathsf{MuxTapT}(F_1, F_2)$. □

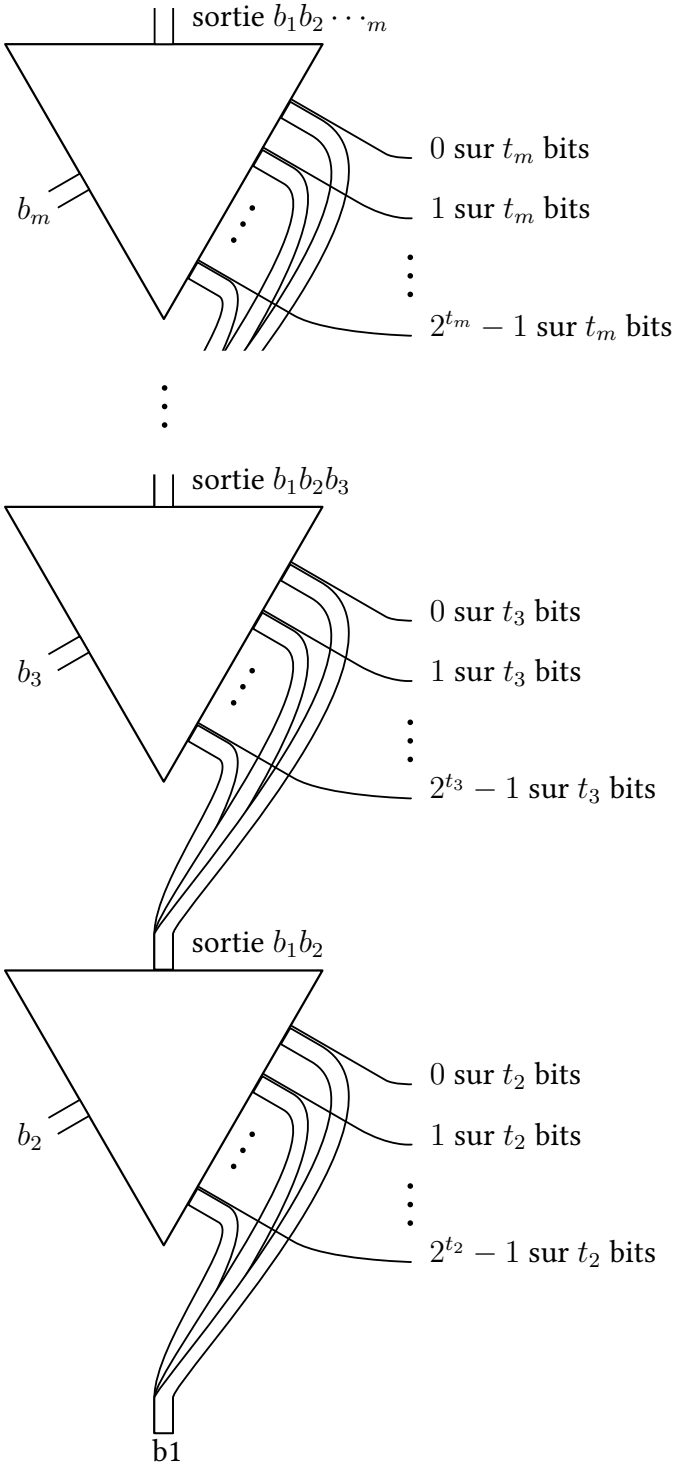

Figure 3.2 – Simulation d'une porte juxtaposition avec $m - 1 \in \mathcal{O}(1)$ portes multiplex

3.2. Les AuxDPDA Capturent LOGDCFL

Proposition 3.9. [MRV99] $\mathsf{DAuxPDA}\left(n^{\mathcal{O}(1)}\,,\,\mathcal{O}\left(\log\left(n\right)\right)\right) \subseteq \mathsf{MuxTapT}\left(n^{\mathcal{O}(1)}\,,\,n^{\mathcal{O}(1)}\right)$.

Démonstration. Soit $f_1 \in n^{\mathcal{O}(1)}$, $f_2 \in \mathcal{O}\left(\log\left(n\right)\right)$, $\alpha_1, \alpha_2 \in \mathbb{N}$ et A un AuxPDA qui – sur $x \in \Sigma^*$ quelconque – fonctionne en temps au plus $\alpha_1 f_1\left(|x|\right)$ et utilise au plus $\alpha_2 f_2\left(|x|\right)$ espace sur son ruban de travail.

Sans perte de généralité, on assume que A doit empiler ou dépiler un symbole de sa pile à chaque transition. Cette restriction fait partie du folklore [HU79]. Simplement, si $\exists A'$ un AuxPDA qui ne respecte pas cette hypothèse, il suffit de construire A qui simule A' sauf lorsque ne fait rien avec sa pile. Dans ce dernier cas A empile un symbole inutile qui dépilé à la prochaine transition. Ainsi, le temps de calcul de A est au pire le double de celui de A'.

Encore sans perte de généralité, on assume que A doit terminer avec une pile vide. En effet, si $\exists A'$ un AuxPDA qui ne respecte pas cette hypothèse, il suffit de construire A qui simule A' jusqu'à ce qu'il entre dans un état d'acceptation [resp. de refus]. Dans ce cas A dépile sa pile en entier et puis s'arrête et accepte [resp. refuse]. Similairement, le temps de calcul de A est au pire le double de celui de A'.

Afin d'expliciter la famille de circuit uniforme $\left(C_n\right)_{n\in\mathbb{N}}$ qui décide $L\left(A\right)$ avec taille d'arbre de preuve et taille polynomiales, on construit une mT M tel que $M\left(1^n\right) = \langle C_n \rangle$

Le circuit C_n simule le calcul de A sur $x \in \Sigma^n$ en travaillant sur K l'ensemble des configurations de surface (*surface configurations* dans [MRV99] et [Coo71a]), simplement configurations par la suite. Ainsi, on note $k = \left(\, t\,,\, i\,,\, x_i\,,\, r\,,\, j\,,\, q\,,\, Z\,\right)$ une configuration de A où t est le temps de calcul, i [resp. j] la position de la tête de lecture sur le ruban d'entrée [resp. de travail], x_i le symbole lu sur le ruban d'entrée, r le contenu du ruban de travail, q l'état et Z le symbole sur le dessus de la pile. Par la suite, un indice quelconque (e.g. a) sur un élément d'une configuration (e.g. t_a ou x_{i_a}) impliquera que cet élément appartient à la configuration avec le même indice (e.g. k_a). On note aussi que $\forall k \in K$, $\langle k \rangle$ est l'entier $\in \{0\,,\,1\,,\,\ldots\,,\,|K|-1\}$.

Pour chaque $k_1 \in K$, C_n contient un circuit partiel qui calcule la dernière configuration accessible à partir de k_1 sans jamais dépiler plus bas que la hauteur de pile en k_1. Voici un algortihme qui illustre ce calcul nommé DernièreConfig$\left(k_1\right)$:

1: **procédure** DernièreConfig(k_1)
2: **si** $\delta\left(k_1\right)$ empile un symbole **alors**
3: $k_2 \leftarrow$ CalculerEmpile$\left(x\,,\,k_1\right)$ ▷ k_2 est la configuration qui suit k_1 lorsque le mot en entrée est x
4: $k_3 \leftarrow$ DernièreConfig$\left(k_2\right)$

5: $k_4 \leftarrow$ CalculerDépile $(x\,,\,k_3\,,\,Z_1)$ ▷ k_4 est la configuration qui suit k_3 lorsque le mot en entrée est x et le symbole en dessous de Z_3 est Z_1

6: $k_5 \leftarrow$ DernièreConfig (k_4)

7: **renvoie** k_5

8: **sinon**

9: **renvoie** k_1

10: **fin si**

11: **fin procédure**

Avant de montrer l'exactitude de l'algorithme, on note que $\forall k_1 \in K$ tel que $\delta(k_1)$ empile un symbole, on a $t_1 < t_2 \leq t_3 < t_4 \leq t_5$ parce qu'il y a une transition qui empile entre k_1 et k_2 et une autre qui dépile entre k_3 et k_4. On a $t_2 = t_3$ [resp. $t_4 = t_5$] seulement lorsque $\delta(k_2)$ [resp. $\delta(k_4)$] est une transition qui dépile.

On pose $\mathrm{P}(h)$ qui est vérifié lorsque $\forall k_1 \in K$ tel que la pile va atteindre une hauteur supplémentaire de h symboles à partir de k_1 avant de dépiler plus bas que la hauteur de pile en k_1, DernièreConfig (k_1) est exact. Pour $h = 0$, on a DernièreConfig $(k_1) = k_1$ ce qui vérifie $\mathrm{P}(0)$. Pour $\mathrm{P}(h+1)$, on doit avoir que $\delta(k_1)$ empile un symbole, donc $k_3 =$ DernièreConfig (k_2) est calculé correctement par hypothèse d'induction. Bien que k_4 revient à la même hauteur de pile que k_1, ceci s'est produit sans jamais dépiler en dessous de cette hauteur. Ainsi, parce que $t_1 < t_4$ et que $[\alpha_1 f_1(n)]$ est fini, il suffit de répéter l'argument pour éventuellement arriver à cette dernière configuration accessible.

Pour chaque $k_1 \in K$, M écrit la description d'un circuit partiel qui calcule DernièreConfig (k_1). Si $\delta(k_1)$ dépile, alors ce circuit partiel est une porte avec identifiant numérique $3\langle k_1\rangle$ et une sortie constante de $\langle k_1\rangle$. Dans le cas contraire, il contient trois portes (voir figure 3.3) :

$$\mathbf{m}(\,3\langle k_1\rangle + 2\#3\,|K| + i_2\#\langle k_1\rangle\#d_0\#\#d_1\#\#\ldots\,) \text{ où } d_{\langle x_{i_2}\,,\,k_1\rangle} = 3\langle k_2\rangle$$

$$\mathbf{m}(\,3\langle k_1\rangle + 1\#3\langle k_1\rangle + 2\#\#d_0\#c_0\#d_1\#c_1\#\ldots\,) \text{ où } d_{\langle k_3\rangle} = 3\,|K| + i_4 \text{ et } c_{\langle k_3\rangle} = \langle k_3\,,\,Z_1\rangle$$

$$\mathbf{m}(\,3\langle k_1\rangle\#3\langle k_1\rangle + 1\#\#d_0\#\#d_1\#\#\ldots\,) \text{ où } d_{\langle x_{i_4}\,,\,k_3\,,\,Z_1\rangle} = 3\langle k_4\rangle$$

Étant donné qu'une configuration inclut le symbole lu par la tête de lecture, la configuration initiale de A – dans notre contexte – dépend du x en entrée de C_n. Ainsi, M doit ajouter une

porte, $\mathbf{m}(\,3\,|K|+n+1\#3\,|K|+1\#\#d_0\#\#d_1\#\#\dots\,)$ où $d_{\langle x_1\rangle}=3\,\langle k_{\text{init}}\,\rangle$ et k_{init} est la configuration initiale de A telle que x_1 est le symbole lu. Aussi, M ajoute $\mathbf{m}(\,3\,|K|+n+2\#3\,|K|+n+1\#\#d_0\#\#d_1\#\#\dots\,)$ où $d_{\langle k\rangle}$ est le bit 1 si k contient un état d'acceptation et le bit 0 sinon, la porte résultat qui décide si A refuse x selon l'état contenu dans DernièreConfig (k_{init}). Finalement, M ajoute la description de chaque entrée x_i, $\mathbf{x}(\,3\,|K|+i\#i\,)$.

On montre à présent que $(C_n)_{n\in\mathbb{N}}$ est uniforme. Pour ce faire, on doit calculer $\forall n\in\mathbb{N}, T(C_n)=3\,|K|+n+2$, ce qui requiert la taille de K :

$$|K|=\underbrace{\alpha_1 f_1(n)}_{\text{temps}}\cdot\underbrace{n}_{\text{position entrée}}\cdot\underbrace{|\Sigma|}_{\text{symbole entrée}}\cdot\underbrace{|\Gamma|^{\alpha_2 f_2(n)}}_{\text{contenu travail}}\cdot\underbrace{\alpha_2 f_2(n)}_{\text{position travail}}\cdot\underbrace{|Q|}_{\text{état}}\cdot\underbrace{|\Pi|}_{\text{symbole pile}}$$

Ainsi $\forall n\in\mathbb{N}, T(C_n)\in\mathcal{O}\left(f_1(n)\cdot 2^{f_2(n)}\cdot n\right)$, ce qui veut dire $\exists\alpha_3\in\mathbb{N}, f_3\in n^{\mathcal{O}(1)}$ tels que $\forall n\in\mathbb{N}, T(C_n)\leq\alpha_3 f_3(n)$. Le calcul de M consiste donc à réutiliser son espace logarithmique pour construire la description de chaque sommet de C_n. Cette tâche s'effectue bel et bien en espace logarithmique étant donné que pour écrire la description de chaque sommet, il suffit soit d'arithmétique sur les identifiants numériques des sommets soit du calcul d'une transition d'une configuration à la prochaine. Le dernier détail important pour l'uniformité est que la porte résultat – celle avec identifiant $3\,|K|+n+2$ – soit écrite en premier dans $\langle C_n\rangle$.

On montre à présent que $\forall n\in\mathbb{N}, x\in\Sigma^n$ l'arbre de preuve de C_n sur x est de taille polynomiale. On explique une notion essentielle à la définition du prédicat à venir. Soit $n\in\mathbb{N}, x\in\Sigma^n$ et $k_1\in K$, alors parcourir en profondeur l'arbre de preuve du sous-circuit à partir de k_1 sur x forme une séquence S' des portes visitées. On note donc $S(x\,,\,k_1)$ la séquence où l'on ne conserve que les paramètres t des portes de forme $\exists k\in K, 3\,\langle k\rangle$ à partir de la séquence S'.

Soit $\mathrm{P}(h)$ vérifié si $\forall n\in\mathbb{N}, x\in\Sigma^n, k_1\in K$ tel que les portes $3\,\langle k_1\rangle$ et $3\,\langle k_1\rangle+2$ ont une hauteur d'au plus h dans l'arbre de preuve de C_n sur x, $S(x\,,\,k_1)$ est strictement croissante. Pour $\mathrm{P}(0)$, la porte $3\,\langle k_1\rangle$ est constante donc $S(x\,,\,k_1)$ ne contient que t_1.

Pour $\mathrm{P}(h+1)$, le parcours en profondeur est en deux parties. La première partie commence avec $3\,\langle k_1\rangle+2$ qui a une hauteur d'au plus $h+1$ donc, par hypothèse d'induction, $S(x\,,\,k_2)$ est strictement croissante. La seconde partie commence avec $3\,\langle k_1\rangle$ qui a aussi une hauteur d'au plus $h+1$ donc, par hypothèse d'induction, $S(x\,,\,k_4)$ est strictement croissante. On note que la séquence $S(x\,,\,k_2)$ commence par t_2 et termine avec t_3, tandis que la séquence $S(x\,,\,k_4)$ commence par t_4 et termine avec t_5. En combinant $S(x\,,\,k_2)$ et $S(x\,,\,k_4)$, on obtient une séquence strictement croissante qui vérifie $\mathrm{P}(h+1)$ parce que, tel que discuté plus haut, $t_1<t_2\leq t_3<t_4\leq t_5$.

On a donc que $\forall n \in \mathbb{N}, x \in \Sigma^n$ l'arbre de preuve de C_n contient $\forall t \in [\alpha_1 f_1(n)]$, au plus une porte $3\langle k \rangle$ parce que $S(x, k_{\text{init}})$ est strictement croissante. Avec deux sommets d'entrées pour chaque circuit partiel, la borne sur la taille de l'arbre de preuve est $\forall n \in \mathbb{N}, TAP(C_n) \leq 5\alpha_1 f_1(n) + 2$.

Pour conclure, la famille de circuit multiplex uniforme $(C_n)_{n\in\mathbb{N}}$ décide $L(A)$ avec $\forall n \in \mathbb{N}, TAP(C_n) \in n^{\mathcal{O}(1)}$ et $T(C_n) \in n^{\mathcal{O}(1)}$. Donc $L(A) \in \mathsf{MuxTapT}\left(n^{\mathcal{O}(1)}, n^{\mathcal{O}(1)}\right)$. □

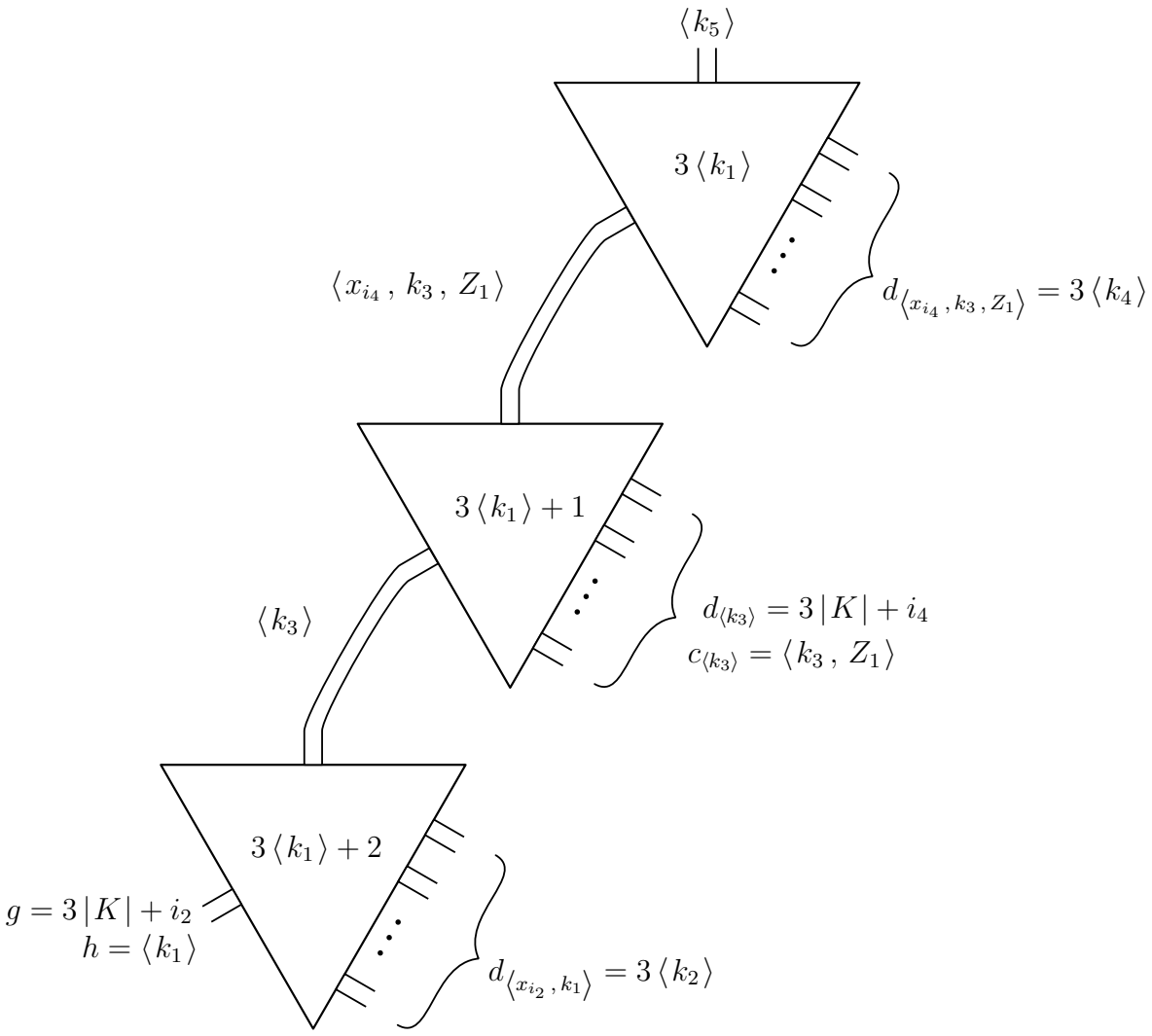

Figure 3.3 – Circuit partiel avec chaque configuration $k_l = (t_l , i_l , x_{i_l} , r_l , j_l , q_l , Z_l)$ représentant une étape de calcul, où selon la valeur i du guide g et h l'argument de donnée d_i et c_i est sélectionné. (g et d_i [resp. h et c_i]) sont des identifiants de portes [resp. constantes])

Proposition 3.10. [MRV99] $\mathsf{MuxTapT}\left(n^{\mathcal{O}(1)}, n^{\mathcal{O}(1)}\right) \subseteq \mathsf{LOGDCFL}$.

Démonstration. Soit $f_1, f_2 \in n^{\mathcal{O}(1)}$, $\alpha_1, \alpha_2 \in \mathbb{N}$ et $(C_n)_{n\in\mathbb{N}}$ une famille de circuits multiplex uniforme qui décide Y telle que $\forall n \in \mathbb{N}, TAP(C_n) \leq \alpha_1 f_1(n)$ et $T(C_n) \leq \alpha_2 f_2(n)$. Il faut donc exhiber un $Y' \in \mathsf{DCFL}$ et un transducteur f qui fonctionne en espace logarithmique tel que $x \in Y \iff f(x) \in Y'$. On montre donc que Y' est reconnu par un DPDA A.

Description de Y' : La forme générale d'un $f(x)$ produit par le transducteur est une liste des descriptions de chacun des sommets de $C_{|x|}$ (similaire à la description naturelle d'un circuit multiplex) qui permettra à A d'évaluer $C_{|x|}$ à la lecture de cette liste.

Pour simplifier la construction de A, tous les sommets de $f(x)$ sont des portes. f convertit donc chaque entrée $\mathbf{x}_i$ en une porte où chaque argument est un paquet d'un bit constant de valeur x_i. Ainsi, pour évaluer le circuit décrit par $f(x)$, il est inutile de lire x, car ce dernier est encodé dans $f(x)$.

Le calcul de A sur $f(x)$ commence par évaluer la porte résultat. Pour évaluer une porte P avec identifiant numérique s dans $C_{|x|}$, A empile s, se déplace jusqu'à la description de P, puis évalue la porte qui fournit à P son paquet de bits guides. Si, par contre, le paquet de bits guides de P est entièrement constant, A empile ces bits constants. Ensuite, A se déplace jusqu'à la description de P, dans la section qui décrit le paquet de bits de données sélectionné par la valeur tout juste empilée, dépile cette valeur et s, puis empile l'identifiant de la porte qui fournit à P le paquet de bits de données (ou bien les bits constants sélectionnés). A continue ainsi de suite, jusqu'à ce que la pile ne contienne qu'un bit constant, la sortie de la porte résultat.

Un premier problème avec cette stratégie est que la liste des descriptions de portes n'est pas nécessairement dans le bon ordre pour évaluer $f(x)$ en la lisant d'une seule direction. Une solution est de répéter la liste plusieurs fois afin de permettre à A de trouver la porte voulue même si l'automate la dépasse, elle sera dans la prochaine répétion de la liste. Le calcul de A effectue un parcours en profondeur de l'arbre de preuve de $C_{|x|}$ sur x, en visitant chaque porte deux fois (une fois pour évaluer son paquet de bits guides et l'autre pour le paquet de bits de données). Répéter la liste $2\alpha_1 f_1(|x|)$ fois est donc suffisant.

Un second problème avec le calcul de A est que, pour se déplacer jusqu'à la description de la porte P avec identifiant numérique s recherchée, l'automate doit comparer s sur sa pile avec chaque élément de la liste. Or, cette comparaison demande de dépiler les bits de s pour y accéder. Si on ne trouve pas la description de P immédiatement, le s dépilé est perdu sans qu'on puisse se déplacer jusqu'à P. L'encodage de chaque identifiant numérique s doit donc être suivi de son inverse s^R. Supposons que A calcule sur entrée $f(x) = \ldots \left(ubv\#v^Rbu^R \right) \ldots$ et le dessus de la

pile est $ub' \dots$ (ici, b' est sous u et le premier symbole de u est au dessus de la pile), avec u, v deux mots non-vides, b un bit et b' son complément. En lisant u, A le dépile, puis l'automate détecte que b diffère de b' et continue en empilant v (sa pile est donc maintenant $v^R b' \dots$). Par la suite, l'automate continue à lire l'entrée (en dépilant v^R) jusqu'à détecter à nouveau une différence puis empile u^R pour se retrouver à nouveau avec $ub' \dots$ au dessus de sa pile.

Avec ces problèmes résolus, il est maintenant possible de formaliser le langage Y'. On pose donc $f(x) := w^{2\alpha_1 f_1(|x|)}$ où w est la concaténation $w_1 w_2 \dots w_{TAP(C_{|x|})}$ des descriptions de chaque sommet du circuit en tant que porte. Comme pour la description naturelle d'un circuit multiplex, les identifiants numériques et les bits constants sont tous écrits en binaire. Pour une porte avec identifiant numérique s avec un paquet de bits guides de taille k, sa description est comme suit :

$$w_s := \left(s\#s^R\#h^R\#g^R \right) w_{s,0} w_{s,1} \dots w_{s,2^k-1}$$

où g est l'identifiant numérique de la porte dont la sortie est utilisée comme paquet de bits guides et h est la partie de bits constants du paquet de bits guides. Il est possible que g ou h soit vide, mais pas les deux. Aussi, $w_{s,j}$ est la description du j-ème paquet de bits de données d'une porte multiplex avec identifiant numérique s, comme suit :

$$w_{s,j} := \left(j\$s\#s^R\$j^R\#c_j^R\#d_j^R \right)$$

Dans cette description j est écrit en binaire, d_j est l'identifiant numérique de la porte dont la sortie est utilisée comme j-ème paquet de bits de données et c_j est la partie de bits constants. Encore une fois, au plus un de d_j et c_j peut être vide.

Description de A : Voici l'algorithme qui décrit le comportement de A sur $f(x)$,

1: **boucler**
2: **si** VoirPile() $= \#$ **alors** ▷ la pile ressemble à $\#j\$s\# \dots$
3: Dépiler()
4: TesterConditionFin() ▷ la condition de fin du calcul se trouve dans cette procédure
5: TrouverIdentifiant() ▷ ceci trouve le $w_{s,j}$ qui correspond au $j\$s$ sur la pile
6: **tant que** VoirPile() $\neq \#$ **faire** ▷ on dépile $j\$s$ pour empiler la sortie de la porte s
7: Dépiler()
8: **fin tant que**
9: Dépiler()

```
        EmpilerJusqua(#)                    ▷ ceci empile c_j du w_{s,j} trouvé
        Empiler(#)
        EmpilerJusqua())                    ▷ ceci empile d_j du w_{s,j} trouvé
    sinon                                   ▷ la pile ressemble à s#...
        TrouverIdentifiant()                ▷ ceci trouve le w_s qui correspond au s sur la pile
        Empiler($)
        EmpilerJusqua(#)                    ▷ ceci empile h du w_s trouvé
        Empiler(#)
        EmpilerJusqua())                    ▷ ceci empile g du w_s trouvé
    fin si
fin boucler
```

LireRuban() retourne le symbole lu sur le ruban, AvancerTête() déplace la tête de lecture d'une case à droite sur le ruban, VoirPile() retourne le symbole au dessus de la pile, Dépiler() dépile le symbole au dessus de la pile et le retourne, Empiler(Z) empile le symbole Z sur la pile. Les autres procédures utilisées sont décrites ici :

```
procédure AvancerTêteJusqua(a)
    tant que LireRuban() ≠ a faire
        AvancerTête()
    fin tant que
    AvancerTête()
fin procédure
```

```
procédure EmpilerJusqua((1, k) a)
    tant que LireRuban() ≠ a faire
        Empiler(LireRuban())
        AvancerTête()
    fin tant que
    AvancerTête()
fin procédure
```

La procédure TrouverIdentifiant fonctionne selon le principe expliqué en solution au second problème de la stratégie de calcul initiale de A.

```
procédure TrouverIdentifiant ▷ trouve w_s ou w_{s,j} qui correspond au s ou j$s sur la pile
    trouvé ← non
    tant que trouvé = non faire
```

```
        AvancerTêteJusqua ( ()
        différence ← inconnu
        tant que différence = inconnu faire
            si LireRuban( ) = VoirPile( ) = # alors ▷ on atteint la fin de l'identifiant, sans différence
                différence ← non
                trouvé ← oui
            sinon si LireRuban( ) ≠ VoirPile( ) alors
                différence ← oui
            sinon
                AvancerTête( )
                Dépiler( )
            fin si
        fin tant que
        EmpilerJusqua ( # )          ▷ on empile ce qui suit la différence trouvée (sinon, rien)
        tant que LireRuban( ) = VoirPile( ) faire
            AvancerTête( )
            Dépiler( )
        fin tant que
        EmpilerJusqua ( # )  ▷ ceci rétablit la pile comme elle était au début de la procédure
    fin tant que
fin procédure
```

La procédure TesterConditionFin arrête le calcul de A si la condition de fin est atteinte. Pour arrêter, A doit avoir exactement bZ_0 sur sa pile, où Z_0 est le symbole initial de pile et b un bit qui indique l'acceptation ou non. La procédure dépile donc le symbole du dessus de pile pour vérifier que celui en dessous est Z_0, sinon elle rétablit la pile.

```
procédure TesterConditionFin
    Z ← Dépiler( )                ▷ À la fin de la simulation, ceci contient la sortie du circuit
    si VoirPile( ) = Z_0 et Z = 1 alors
        A s'arrête et accepte
    sinon si VoirPile( ) = Z_0 et Z = 0 alors
        A s'arrête et refuse
    sinon
```

8: EMPILER (Z) ▷ Il faut empiler à nouveau, car ce n'est pas la fin de la simulation

9: **fin si**

10: **fin procédure**

Il faut maintenant prouver que l'algorithme qui décrit le comportement de A sur $f(x)$ est correct. On montrera par induction que $\mathrm{P}(n)$ est vérifié $\forall n \geq 0$, où $\mathrm{P}(n)$ est vérifié si $\forall s$ identifiant numérique d'une porte de hauteur n, A commence avec $s\#u$ sur le dessus de sa pile (u est le mot quelconque qu'est le reste de la pile), à la ligne 1 et (sans changer u) obtient $\#cu$ sur le dessus de sa pile, à la même ligne 1, où $c \in \{0, 1\}^*$ est la sortie de s.

Pour $\mathrm{P}(0)$, A a $s\#u$ sur le dessus de sa pile, à la ligne 1. A entre dans la condition à la ligne 13 et parcourt son ruban jusqu'au prochain w_s, puis empile le h correspondant, ce qui donne un dessus de pile $\#h\$s\#u$ (la hauteur de s est 0, ce qui implique que le g correspondant est vide, ce qui implique que h est non-vide). Ensuite, A revient à la ligne 1 puis entre dans la condition à la ligne 2. A ne termine pas en ligne 4, mais parcourt son ruban jusqu'au prochain $w_{s,h}$, puis dépile pour avoir u sur sa pile et puis empile c_h, la sortie de s, ce qui donne $\#c_h u$ (encore une fois, hauteur de s est 0, ce qui implique que d_h est vide, ce qui implique que c_h est non-vide).

Pour $[\,\forall m \leq n, \mathrm{P}(m)\,] \implies \mathrm{P}(n+1)$, A a $s\#u$ sur le dessus de sa pile, à la ligne 1, avec s l'identifiant d'une porte de hauteur $n+1$. A entre dans la condition à la ligne 13 et parcourt son ruban jusqu'au prochain w_s, puis empile les h et/ou g correspondant(s), ce qui donne trois dessus de piles possibles (mais tous reviennent à la ligne 1) :

(1) $g\#h\$s\#u$ (si h et g sont non-vides)

(2) $g\#\$s\#u$ (si h est vide)

(3) $\#h\$s\#u$ (si g est vide)

Pour les cas un et deux, on sait que la hauteur de la porte avec identifiant g est bornée par n, on utilise donc l'hypothèse d'induction et ces cas deviennent respectivement $\#ch\$s\#u$ et $\#c\$s\#u$ avec c la sortie de la porte g. Pour la suite, on considère que h' est une constante qui représente ch, c ou h selon l'un des trois cas. Donc, A a $\#h'\$s\#u$ comme dessus de pile et est à la ligne 1. Ensuite, A entre dans la condition à la ligne 2 et parcourt son ruban jusqu'au prochain $w_{s,h'}$, puis remplace le dessus de sa pile par $c_{h'}$ et/ou $d_{h'}$, ce qui donne trois dessus de piles possibles (mais tous reviennent à la ligne 1) :

(1) $d_{h'}\#c_{h'}u$ (si $c_{h'}$ et $d_{h'}$ sont non-vides)

(2) $d_{h'}\#u$ (si $c_{h'}$ est vide)

(3) $\#c_{h'}u$ (si $d_{h'}$ est vide)

Pour les cas un et deux, on sait que la hauteur de la porte avec identifiant $d_{h'}$ est bornée par n, on utilise donc l'hypothèse d'induction et ces cas deviennent respectivement $\#c'c_{h'}u$ et $\#c'u$ avec c' la sortie de la porte $d_{h'}$. Ainsi, A obtient la sortie de s (sous forme de $c'c_{h'}$, c' ou $c_{h'}$) sur sa pile et en revenant à la même ligne 1.

Avant d'exécuter l'algorithme principal, A initialise son calcul en empilant l'identifiant numérique s_0 de la première porte dans $f(x)$, qui est la porte résultat tel que promit par l'uniformité de $(C_n)_{n\in\mathbb{N}}$. A commence donc son calcul avec $s_0\#Z_0$ sur le dessus de sa pile à la ligne 1 et, parce que $\mathrm{P}(|x|)$ est vérifié, va revenir sur cette même ligne avec sa pile $\#bZ_0$, où b est la sortie 0 ou 1 de la porte résultat. Finalement, A entre dans la condition à la ligne 2 et, avec TesterConditionFin() va accepter ou refuser selon b.

Description de f : la réduction parcourt la description naturelle de $C_{|x|}$ puis écrit sur son ruban de sortie pour chaque sommet rencontré dans le même ordre. On répète ensuite cette procédure $2\alpha_1 f_1(|x|)$ fois.

Pour un sommet d'entrée $\mathbf{x}_i$ avec identifiant numérique s, f écrit :

$$w_s := \left(s\#s^R\#x_i\# \right) \left(0\$s\#s^R\$0\#0\# \right) \left(1\$s\#s^R\$1\#1\# \right)$$

Pour un sommet porte multiplex avec identifiant numérique s, f écrit w_s selon les s, g, h dans la description naturelle (inversés au besoin). Pour écrire les $w_{s,j}$, f compte $0 \leq j \leq 2^k - 1$ puis utilise c_j et d_j inversés.

f produit donc correctement $f(x)$ et ce, en espace logarithmique. En effet, la réduction peut accéder à un bit de la description naturelle de $C_{|x|}$ (grâce à l'uniformité de $(C_n)_{n\in\mathbb{N}}$). Pour ce qui est du comptage nécessaire jusqu'à $\alpha_2 f_2(|x|)$, $2\alpha_1 f_1(|x|)$ ou $2^k - 1$, cela se fait sur $\mathcal{O}(\log(|x|))$ bits car $f_1(|x|) \in |x|^{\mathcal{O}(1)}$ et $k \in \mathcal{O}(\log(|x|))$. □

Corollaire 3.11. [MRV99] LOGDCFL = DAuxPDA $\left(n^{\mathcal{O}(1)}, \mathcal{O}(\log(n))\right)$.

Démonstration. $\subseteq$: La AuxPDA réutilise son espace $\mathcal{O}(\log(n))$ pour calculer chaque bit du résultat de la réduction promise par LOGDCFL, puis utilise sa pile pour effectuer une transition de l'automate à pile propre au $DCFL$ selon le bit obtenu. Étant donné que la réduction promise par LOGDCFL utilise un espace $\mathcal{O}(\log(n))$, le calcul de chaque bit nécessite un temps $n^{\mathcal{O}(1)}$ et il y a $n^{\mathcal{O}(1)}$ bits, le produit est donc un temps $n^{\mathcal{O}(1)}$.

$\supseteq$: Est une conséquence directe des lemmes 3.9 et 3.10. □

Chapitre 4

Gen, un Problème Algébrique

Définition 4.1. Soit X un ensemble, $\bullet$ une opération binaire sur X, et $R \subseteq X$, alors $\langle R \rangle_\bullet$ est la *clôture* de R pour l'opération $\bullet$, qu'on définit par sa construction :

1: $R', \langle R \rangle_\bullet \leftarrow S$
2: **tant que** $R' \neq \emptyset$ **faire**
3: $\quad R' \leftarrow \{\, k \notin \langle R \rangle_\bullet \,:\, \exists i, j \in \langle R \rangle_\bullet \text{ et } i \bullet j = k \,\}$
4: $\quad \langle R \rangle_\bullet \leftarrow \langle R \rangle_\bullet \cup R'$
5: **fin tant que**

Définition 4.2. [JL76] GenR
Donnée : Un ensemble X, une opération binaire $\bullet$ sur X (donné sous forme d'un tableau de taille $|X|^2$), un sous-ensemble $R \subseteq X$ et un élément $x \in X$.
Question : Est-ce que $x \in \langle R \rangle_\bullet$?

Proposition 4.3. [JL76] GenR est P-complet selon $\leq_{\log}$.

Définition 4.4. [BM91] Gen
Donnée : Un tableau $n \times n$ qui décrit une opération binaire $\bullet$ sur $[n]$.
Question : Est-ce que $n \in \langle \{1\} \rangle_\bullet$?

Proposition 4.5. [BM91][JL76] Gen est P-complet selon $\leq_{\log}$.

Définition 4.6. Soit $\bullet$ une opération binaire sur $[n]$, R un sous-ensemble non-vide de $[n]$ et $x \in \langle R \rangle_\bullet$, alors un *témoin* de $x \in \langle R \rangle_\bullet$ est une arborescence binaire entière (S, A, s_0) munie d'un étiquetage $\lambda : S \to [n]$ tel que

- $\lambda(s_0) = x$
- Si s est interne, alors $\lambda(\text{enfant gauche de } s) \bullet \lambda(\text{enfant droit de } s) = \lambda(s) \notin R$
- Si s est feuille, alors $\lambda(s) \in R$.

Deux témoins T_1 et T_2 pour un même $\bullet$, R et x sont *distincts* (i.e. $T_1 \neq T_2$) si leurs graphes sous-jacents ne sont pas isomorphes, ou bien si l'isomorphisme existant lie deux sommets qui n'ont pas la même étiquette.

Il est évident que $x \in \langle R \rangle_\bullet \iff \exists$ un témoin de $x \in \langle R \rangle_\bullet$.

4.1. Saut de NL à P entre Gen-1 et Gen-2

Définition 4.7. GEN-t

Même problème que GEN avec une restriction sur la donnée : $\forall k \in [n] \setminus \{1\}$, k est présent au plus t fois dans le tableau (i.e. il existe au plus t couple(s) $(i\,,\,j)$ tel(s) que $i \bullet j = k$).

Lemme 4.8. Soit $\bullet$ une opération binaire sur $[n]$, R un sous-ensemble non-vide de $[n]$ et $x \in \langle R \rangle_\bullet$, alors $\exists T$ témoin de $x \in \langle R \rangle_\bullet$ d'une hauteur $h(T) \leq n$.

Démonstration. Soit T le témoin de $x \in \langle R \rangle_\bullet$, promis par $x \in \langle R \rangle_\bullet$, qu'on suppose tel que $h(T) > n$. Ainsi, il existe au moins une chaîne entre la racine s_0 et une feuille telle que $s \neq s'$ dans la chaîne ont une étiquette $\lambda(s) = \lambda(s')$. On suppose sans perte de généralité que s est un ancêtre de s', puis on remplace la sous-arborescence de racine s par celle de racine s' afin d'obtenir un nouveau témoin de $x \in \langle R \rangle_\bullet$ qui a une répétition d'étiquette en moins. Ce processus est répété jusqu'à ce qu'aucune telle chaîne ait une répétition d'étiquette, ce qui résulte en un témoin d'hauteur au plus n. □

Lemme 4.9. Soit $\bullet$ sur $[n]$ tel que $\forall k \in [n] \setminus \{1\}$ il existe au plus un couple $(i\,,\,j) : i \bullet j = k$ et R un sous-ensemble non-vide de $[n]$. Alors, $\forall x \in \langle R \rangle_\bullet$, il existe un témoin unique de $x \in \langle R \rangle_\bullet$.

Démonstration. On montre par induction que $\mathrm{P}(m)$ est vrai $\forall m \geq 0$, où $\mathrm{P}(m)$ est vérifié si $\forall T_1, T_2$ deux témoins de $x \in \langle R \rangle_\bullet$ tels que $m = \max\{h(T_1)\,,\,h(T_2)\}$, on a $T_1 = T_2$.

Soient T_1 et T_2 deux témoins de $x \in \langle R \rangle_\bullet$ tels que $\max\{h(T_1)\,,\,h(T_2)\} = 0$. On a que $h(T_1) = h(T_2) = 0$ et donc, par 4.6, T_1 et T_2 doivent chacun être une arborescence composée d'une seule feuille racine avec étiquette $\lambda(s_0) = x$. Ainsi, $T_1 = T_2$, ce qui prouve $\mathrm{P}(0)$.

Avec l'hypothèse d'induction $\forall l \leq m, \mathrm{P}(l)$, on suppose par contradiction $\exists T_1 \neq T_2$ deux témoins de $x \in \langle R \rangle_\bullet$ de hauteur au plus $m+1$. Étant donné que $i \bullet j$ est l'unique produit possible qui donne x, on a que l'étiquette de l'enfant gauche des racines de T_1 et T_2 est i. On suppose sans perte de généralité que la sous-arborescence G_1 de T_1 avec racine étiquetée i diffère de la sous-arborescence G_2 de T_2 avec racine étiquetée i. Pourtant, avec l'hypothèse d'induction on doit avoir $G_1 = G_2$, parce que $h(G_1)\,,\,h(G_2) \leq m$. Cette contradiction indique que $\mathrm{P}(m+1)$ doit être vrai et donc $\mathrm{P}(m)$ est vrai $\forall m \geq 0$. □

Définition 4.10. [Jon75] STConn
Donnée : Un graphe orienté $G = (S\,,\,A)$ donné par sa matrice d'adjacence, un sommet source s_{src} et un sommet destination s_{dest}
Question : Est-ce qu'il existe un chemin de s_{src} à s_{dest} dans G ?

Définition 4.11. STConn$_{\leq 2}$
Même problème que STConn avec quelques restrictions sur la donnée :

1. G ne contient aucun circuit
2. $\deg^+(s_{\text{dest}}) = 0$
3. $\forall s \in S, \deg^+(s) \leq 2$
4. $\forall i \neq j \in S, \exists$ au plus un sommet $s \in S$ tel que $(s\,,\,i)\,,(s\,,\,j) \in A$

Lemme 4.12. STConn$_{\leq 2} \in$ NL-difficile.

Démonstration. Par [Jon75] on a que STConn $\in$ NL-difficile selon $\leq_{\log}$. Par transitivité de $\leq_{\log}$, il suffit de montrer que STConn $\leq_{\log}$ STConn$_{\leq 2}$ afin d'avoir STConn$_{\leq 2} \in$ NL-difficile.

Plutôt que construire une seule mT, on en construit trois qui, une à la suite de l'autre, réduisent STConn à STConn$_{\leq 2}$. Si, en plus, chacune des mT utilise un espace logarithmique, on conclut que STConn $\leq_{\log}$ STConn$_{\leq 2}$ par la transitivité de $\leq_{\log}$.

La première mT M_1 va simplement, pour chaque sommet $s \in S : \deg^+(s) > 2$, ajouter de nouveaux sommets afin de créer une arborescence binaire dont la racine est s et les feuilles sont les sommets $t \in S : (s\,,\,t) \in A$. On a aussi que la taille de chaque arborescence construite est polynomiale, ainsi, M_1 réutilise son espace logarithmique pour compter les sommets ajoutés, pour chaque $s \in S : \deg^+(s) > 2$.

La seconde mT M_2 construit un graphe $G' = (S'\,,\,A')$ qui satisfait la condition **1** à partir de G en entrée. On applique la technique de *timestamp* utilisée dans [Jon75, Corollaire 25], donc M_1 copie les sommets du graphe G initial $|S|$ fois, puis ajoute un arc d'un sommet d'une copie au sommet de la suivante s'il y avait un tel arc dans G. Formellement, $S' = \{\,(i\,,\,s) \,:\, 1 \leq i \leq |S| \text{ et } s \in S\,\}$ et $A' = \{\,((i\,,\,s)\,,(i+1\,,\,t)) \,:\, (s\,,\,t) \in A\,\}$. Afin d'avoir $s'_{\text{src}} = (1\,,\,s_{\text{src}})$ et $s'_{\text{dest}} = (n\,,\,s_{\text{dest}})$, on doit aussi ajouter les arcs $\{\,((i\,,\,s_{\text{dest}})\,,(i+1\,,\,s_{\text{dest}})) \,:\, 1 \leq i < |S|\,\}$.

Clairement, aucun circuit n'est possible dans G' car un arc existe seulement d'une copie à la suivante, ce qui satisfait la condition **1**. Le graphe G' construit satisfait aussi la conditon **2**, car $\deg^+(s'_{\text{dest}}) = 0$. On a aussi que M_1 fonctionne en espace logarithmique tel que promis par [Jon75, Corollaire 25].

Le calcul de la dernière mT M_3 concerne chaque sommet $s \in S' : \deg^+(s) = 2$. On pose donc $i, j \in S' : (s\,,\,i)\,,(s\,,\,j) \in A'$. M_3 va créer de nouveaux sommets i' et j', puis remplacer

les arcs $(s\,,\,i)$ et $(s\,,\,j)$ par de nouveaux arcs $(s\,,\,i')$, $(s\,,\,j')$, $(i'\,,\,i)$ et $(j'\,,\,j)$ dans A'. Ainsi, s n'est plus candidat avec un arc vers i et j à la fois et il n'y a que s avec un arc vers i' et j'. On a que M_3 utilise évidemment un espace logarithmique.

Chaque mT préserve l'existence d'un chemin de s_{src} à s_{dest} dans G. En effet, M_1 et M_3 remplacent possiblement une arcs par plusieurs et pour M_2, si un chemin existe de longueur l existe de s à t dans G alors un chemin existe de $(1\,,\,s)$ à $(l\,,\,t)$ dans G'. □

Définition 4.13. STCONN$_2$
Même problème que STCONN avec quelques restrictions sur la donnée :

1. $G = ([n]\,,\,A)$ ne contient aucun circuit
2. $\forall s \in [m+1]\,, \deg^+(s) = 0$
3. $\forall s \in [n] \setminus [m+1]\,, \deg^+(s) = 2$
4. $\forall i \neq j \in [n]\,, \exists$ au plus un sommet $s \in [n]$ tel que $(s\,,\,i)\,,(s\,,\,j) \in A$

Avec $1 \leq m < n-1$

Lemme 4.14. STCONN$_2 \in$ NL-difficile.

Démonstration. Comme pour la preuve précédente, il suffit de montrer que STCONN$_{\leq 2}$ $\leq_{\log}$ STCONN$_2$ afin d'avoir STCONN$_2 \in$ NL-difficile.

On construit deux mT qui réduisent STCONN$_{\leq 2}$ à STCONN$_2$. Si, en plus, chacune des mT utilise un espace logarithmique, on conclut que STCONN$_{\leq 2} \leq_{\log}$ STCONN$_2$ par la transitivité de $\leq_{\log}$.

La première mT M_1 s'assure simplement que $\forall s \in S, \deg^+(s) = 0$ ou 2. Pour chaque sommet $s \in S : \deg^+(s) = 1$, M_1 ajoute un nouveau sommet t et un arc $(s\,,\,t)$. Un cas spécial est le sommet s_{src}, pour garantir $\deg^+(s_{\text{src}}) = 2$, on lui ajoute jusqu'à deux arcs vers de nouveaux sommets. Chaque ajout se fait en espace $\mathcal{O}(\log(n))$, en bouclant sur les $n^{\mathcal{O}(1)}$ sommets M_1 réutilise donc son espace logarithmique.

La seconde mT M_2 construit un graphe $G' = (S'\,,\,A')$ à partir de G en entrée. On pose $n = |S|, S' = [n]$ et $m+1 = \left|\{\ s \in S : \deg^+(s) = 0\ \}\right|$. Le renommage des sommets s'effectue à l'aide d'une bijection $f : S \to [n]$ où $\forall s \in S$ tel que $\deg^+(s) = 0$ [resp. $\deg^+(s) = 0$], $f(s) = i$ tel que s est le i-ème sommet avec $\deg^+(s) = 0$ [resp. $\deg^+(s) = 0$]. Compter les sommets et calculer le demi-degré extérieur se fait en espace $\mathcal{O}(\log(n))$ pour chaque sommet, en bouclant sur les $n^{\mathcal{O}(1)}$ sommets et arcs M_2 réutilise donc son espace logarithmique. On note que $\deg^+(s_{\text{src}}) = 2$ et la condition **1** de STCONN$_{\leq 2}$ nous assure que $1 \leq m < n-1$.

Chaque mT préserve l'existence d'un chemin de s_{src} à s_{dest} dans G, ainsi que les conditions **1** et **4** de STCONN$_{\leq 2}$. □

Théorème 4.15. GEN-1 $\in$ NL-complet selon $\leq_{\log}$.

Démonstration. Par le théorême d'Immerman-Szelepcsényi[Imm88][Sze88], NL = co-NL. Nous obtiendrons GEN-1 $\in$ NL en démontrant en première partie de preuve du théorème que le complément $\overline{\text{GEN-1}}$ de GEN-1 est dans co-NL.

On a $w \in$ GEN-1 si w décrit (sous forme de tableau) l'opération $\bullet$ sur $[n]$ telle que $n \in \langle \{1\} \rangle_\bullet$ et $\forall k \in [n] \backslash \{1\}$, au plus un couple $(i\,,\,j)$ donne $i \bullet j = k$. La machine de Turing non-déterministe N que l'on construit pour décider $\overline{\text{GEN-1}}$ doit donc accepter dès qu'une des deux conditions n'est pas respectée.

Pour déterminer si la condition d'unicité n'est pas respectée, N boucle sur $k \in [n] \backslash \{1\}$. Pour chaque k, N parcourt le tableau x et accepte si k est présent plus d'une fois, sinon N continue avec l'exécution de PARCOURSTÉMOIN $(n\,,\,n)$ selon l'algorithme non-déterministe suivant :

1: **procédure** PARCOURSTÉMOIN(h, k)
2: **si** $k = 1$ **alors**
3: N refuse
4: **sinon si** $h = 0$ ou $\nexists\,(i\,,\,j)$ tel que $i \bullet j = k$ **alors**
5: N accepte
6: **fin si**
7: $(i\,,\,j) \leftarrow$ l'unique couple $(i\,,\,j)$ tel que $i \bullet j = k$
8: Choix non-déterministe entre PARCOURSTÉMOIN $(h - 1\,,\,i)$ et PARCOURSTÉMOIN $(h - 1\,,\,j)$
9: **fin procédure**

La recherche d'un couple $(i\,,\,j)$ dans le tableau GEN-1, ainsi que mémoriser les variables $h, k \in [n]$, se fait en espace logarithmique, qui est réutilisé pour chaque exécution de la procédure.

$\mathrm{P}\,(h)$ est vérifié si $[\,\forall k \in [n]\,]\,,\exists$ témoin de $k \in \langle \{1\} \rangle_\bullet$ avec hauteur $\leq h \iff$ tous les chemins de PARCOURSTÉMOIN $(h\,,\,k)$ mènent au rejet. Pour $\mathrm{P}\,(0)$, on observe que $[\,\forall k \in [n]\,]\,,$ PARCOURSTÉMOIN $(h\,,\,k)$ refuse si et seulement si $k = 1$, et que le seul témoin avec hauteur 0 est celui de $1 \in \langle \{1\} \rangle_\bullet$.

Pour $\mathrm{P}\,(h+1)$, on suppose $\exists$ témoin de $k \in \langle \{1\} \rangle_\bullet$ avec hauteur $\leq h+1$. Ainsi, on doit avoir l'existence de témoins de $i \in \langle \{1\} \rangle_\bullet$ et $i \in \langle \{1\} \rangle_\bullet$ (tels que $i \bullet j = k$) avec hauteurs $\leq h$. On sait donc que PARCOURSTÉMOIN $(h+1\,,\,k)$ mène au rejet si $k = 1$ ou bien, par hypothèse d'induction, à la dernière ligne en choisissant PARCOURSTÉMOIN $(h\,,\,i)$ ou PARCOURSTÉMOIN $(h\,,\,j)$.

Inversement, on suppose $\nexists$ témoin de $k \in \langle \{1\} \rangle_\bullet$ avec hauteur $\leq h+1$. On ne peut avoir $k = 1$, mais s'il n'y a pas de i, j tels que $i \bullet j = k$, alors N accepte. Pour la suite, on suppose sans perte de généralité que $\nexists$ témoin de $i \in \langle \{1\} \rangle_\bullet$ avec hauteur $\leq h$. Donc, par hypothèse d'induction, ParcoursTémoin $(h\,,\,i)$ ne refuse pas sur tous les chemins, de même pour ParcoursTémoin $(h+1\,,\,k)$.

Par P (n), on a donc que $\exists$ témoin de $n \in \langle \{1\} \rangle_\bullet$ si et seulement si tous les chemins de ParcoursTémoin $(n\,,\,n)$ mènent au rejet. Ainsi, N décide $\overline{\text{Gen-1}}$ en espace logarithmique. Ceci conclut la preuve que Gen-1 $\in$ NL

Il nous reste à démontrer que Gen-1 est NL-difficile selon $\leq_{\log}$. Parce que NL $=$ co-NL, un langage est NL-difficile selon $\leq_{\log}$ si et seulement si son complément l'est aussi. Il suffit donc de montrer que STConn$_2 \leq_{\log} \overline{\text{Gen-1}}$.

Soit un exemplaire de STConn$_2$ donné par $G = ([n]\,,\,A)$ et m. La réduction consiste à définir sur $[n]$ le produit

$$i \bullet j = \begin{cases} k & \text{si } i < j \text{ et } (k\,,\,i)\,,(k\,,\,j) \in A \\ j+1 & \text{si } i = j < m \\ 1 & \text{sinon} \end{cases}$$

Le produit est bien défini sur tout $[n]$ et chaque paire i, j donne un unique résultat. En effet, les deux premières lignes ont des conditions mutuellement exclusives et chacune offre un seul résultat possible ; la première par la condition **4** de STConn$_2$ qui assure l'unicité d'un tel k, la seconde par évidence. Aussi, ce produit respecte la condition d'unicité de Gen-1. C'est-à-dire qu'il n'y a pas de collision parmi les k et $j+1$ ou entre les deux. En effet, par la condition **3** de STConn$_2$, k est le résultat d'un seul produit $i \bullet j$ et seulement lorsque $i < j$ (donc, $j \bullet i \neq k$) et on ne peut avoir $k = j+1$, car $\deg^+(k) = 2$ donc $k > m+1$ par la condition **2** de STConn$_2$, mais $j+1 < m+1$.

Le calcul du tableau qui décrit le produit $\bullet$ s'effectue en espace logarithmique, car, pour chaque paire i, j, on réutilise l'espace nécessaire pour chercher dans A, ce qui se fait en espace logarithmique.

On note les propriétés suivantes du produit construit, que l'on utilisera par la suite

a. $\langle [m] \rangle_\bullet = \langle \{1\} \rangle_\bullet$

b. $\forall k \in [n]\,, k \in \langle [m+1] \rangle_\bullet$

c. $\forall p \in [m+1]$, un puits du graphe G, et $\forall k \in [n]$, un chemin de k à p existe dans G si et seulement si l'unique témoin de $p \in \langle [m+1] \rangle_\bullet$ possède une feuille avec étiquette p

Parce que $\forall 1 \leq j < m, j \bullet j = j + 1$, on a que $[m] \subseteq \langle \{1\} \rangle_\bullet$. Ainsi, avec $\langle \{1\} \rangle_\bullet \subseteq \langle [m] \rangle_\bullet$ qui est évident, on a que la propriété **a** est vérifiée.

Avant de prouver **b**, on introduit $L(k)$ la longueur d'un des plus long(s) chemin(s) de $k \in [n]$ à un des puits $p \in [m+1]$ dans G. On note que $L(k)$ est bien défini parce que G ne contient aucun circuit, ainsi la longueur des plus long(s) chemin(s) entre chaque pair de sommets est finie. Aussi, un chemin existe toujours de k à un des puits $p \in [m+1]$ dans G car les puits sont les seuls sommets avec demi-degré extérieur nul.

Pour prouver **b**, on procède par induction sur l avec le prédicat $\mathrm{P}(l)$ qui est vérifié si $\forall k \in [n] : L(k) = l, k \in \langle [m+1] \rangle_\bullet$. On a $\mathrm{P}(0)$, car $k \in [m+1]$. Ensuite, avec $k \in [n] : L(k) = l+1$, on pose $(k, i), (k, j) \in A$. Évidemment, on a $L(i), L(j) < L(k) = l+1$ et donc, avec l'hypothèse d'induction, $i, j \in \langle [m+1] \rangle_\bullet$. Ainsi, $i \bullet j = k \in \langle [m+1] \rangle_\bullet$ et la propriété **b** est vérifiée.

On démontre la propriété **c**. Soient d'abord un puits $p \in [m+1]$ et un sommet $k \in [n]$ tels qu'un chemin de k à p existe dans G. Si un tel chemin est de longueur 0, alors $k = p$ et le témoin de $k \in \langle [m+1] \rangle_\bullet$ est une feuille étiquetée p. Sinon la distance de k à p dans G est réalisée par un chemin de longueur $l > 0$ sur les sommets $k, i_1, \ldots, i_{l-1}, p$. L'unique témoin de $k \in \langle [m+1] \rangle_\bullet$ a donc une racine étiquetée k et un enfant dont l'étiquette est un facteur de l'unique produit qui donne k, c'est-à-dire i_1. Puisque la distance de i_1 à p dans G est $l-1$, par l'hypothèse d'induction, l'unique témoin de $i_1 \in \langle [m+1] \rangle_\bullet$ possède une feuille avec étiquette p. Il en va de même pour le témoin de $k \in \langle [m+1] \rangle_\bullet$.

Inversement, supposons que l'unique témoin T de $k \in \langle [m+1] \rangle_\bullet$ possède une feuille étiquetée $p \in [m+1]$. Si la hauteur de T est 0, alors $k = p$ et donc un chemin de longueur nul existe de k à p dans G. Sinon, la racine étiquetée k de T possède deux enfants étiquetés i et j tels que $i \bullet j = k$ et donc $(k, i), (k, j) \in A$. Supposons, sans perte de généralité, qu'une feuille promise avec étiquette p du témoin T provient de la sous-arborescence T_i de T enraciné au sommet étiqueté i. T_i est donc l'unique témoin de $i \in \langle [m+1] \rangle_\bullet$ et possède une feuille étiquetée p. Puisque la hauteur de T_i est strictement moins que celle de T, par induction, il existe un chemin de i à p dans G. Puisque l'arc $(k, i) \in A$, alors un chemin de k à p existe aussi dans G.

Maintenant que les propriétés **a**, **b** et **c** sont vérifiées, nous pouvons confirmer que la construction réduit correctement STConn_2 à $\overline{\text{Gen-1}}$.

Supposons qu'un chemin de n à $m+1$ existe dans G. Il ne peut exister un témoin $n \in \langle [m] \rangle_\bullet$ car ce dernier serait aussi témoin de $n \in \langle [m+1] \rangle_\bullet$, sans feuille étiquetée $m+1$, mais la propriété **c** nous promet que l'unique témoin a une feuille avec cette étiquette. Ainsi, $n \notin \langle [m] \rangle_\bullet = \langle \{1\} \rangle_\bullet$.

Inversement, supposons qu'aucun chemin de n à $m+1$ n'existe dans G. L'unique témoin de $n \in \langle [m+1] \rangle_\bullet$ ne possède pas de feuille étiquetée $m+1$, par la propriété **c**. Ce témoin en est donc un de $n \in \langle [m] \rangle_\bullet = \langle \{1\} \rangle_\bullet$. □

Proposition 4.16. Gen-2 est P-complet selon $\leq_{\log}$.

Démonstration. Évidemment Gen-2 $\in$ P, car une instance de Gen-2 est aussi une instance de Gen $\in$ P.

Étant donné que Gen est P-difficile, il suffit d'utiliser la transitivité de $\leq_{\log}$ et de montrer que Gen $\leq_{\log}$ Gen-2 pour avoir Gen-2 est P-difficile.

On pose Σ et Σ' les alphabets de Gen et Gen-2 respectivement, et $f : \Sigma^* \to \Sigma'^*$ la réduction qui prend la description d'une opération $\bullet$ sur $[n]$ en entrée et écrit la description d'une opération $\circ$ 'équivalente ?'.

L'idée générale de la réduction est de renommer les entrées dupliquées. Par exemple, si $i_1 \bullet j_1 = i_2 \bullet j_2 = i_3 \bullet j_3 = 2$, alors on introduit plusieurs "2" tels que $i_1 \circ j_1 = 2_1$, $i_2 \circ j_2 = 2_2$, $i_3 \circ j_3 = 2_3$. Par contre, il n'est pas possible pour f de déterminer quel produit est nécessaire au calcul de la fermeture de $\{1\}$ sous $\bullet$. Donc, en continuant l'exemple, on ajoute les produits qui cascadent vers le "vrai" 2, $1 \circ 2_1 = 2_2$, $1 \circ 2_2 = 2_3$ et $1 \circ 2_3 = 2'$. Ainsi, peu importe lequel des "2" est généré, celui-ci va générer le "vrai" 2.

Le tableau qui décrit l'opération $\circ$ a deux sections importantes : la première est la rangée 1 avec n régions de chacune n^2 cases, la seconde est un carré n par n. Chaque région de la première rangée représente un élément de $[n]$

On commence par décrire chaque région de la rangée 1. À l'exception de la dernière case de la région, la j-ème case de d'une région contient l'élément $1 \circ j = j+1$. Pour ce qui est de la dernière case, celle-ci dépend de la région. Dans la k-ème région, la dernière case contient l'élément $1 \circ j = n^3 + k$. Ainsi, dès qu'un élément de la région est généré, la dernière case l'est aussi. Dans l'exemple plus haut, le "vrai" 2 est cette dernière case.

La section du carré commence aux coordonnées $(n^3+1\,,\, n^3+1)$ et est définie en termes du tableau original qui décrit $\bullet$. L'élément aux coordonnées $(i\,,\, j)$ du carré (et donc $(i+n^3+1\,,\, j+n^3+1)$ du tableau qui décrit l'opération $\circ$) contient un élément qui génère la $(\, i \bullet j \,)$-ème région de la rangée 1. Cet élément doit aussi être unique dans le cas où $i_1 \bullet j_1 = i_2 \bullet j_2$. Pour qu'il soit unique, on décale l'élément de $(\, i-1 \,)\, n + j$. Finalement, on a que l'élément aux coordonnées $(i\,,\, j)$ du carré est $(\, (\, i \bullet j \,) - 1 \,)\, n^2 + (\, i-1 \,)\, n + j$.

Le tableau construit décrit donc l'opération $i \circ j$ suivante :

$$\begin{cases} n^3 + k & \text{si } i = 1 \text{ et } \exists k \in [n] : j = kn^2 \\ j + 1 & \text{si } i = 1 \text{ et } j < n^3 \\ (\,(\, i - n^3 \bullet j - n^3 \,) - 1 \,)\, n^2 + (\, i - n^3 - 1 \,)\, n + (\, j - n^3 \,) & \text{si } i, j > n^3 \\ 1 & \text{sinon} \end{cases}$$

On montre à présent que chaque élément $2 \leq \alpha \leq n^3 + n$ apparaît au plus deux fois dans le tableau qui décrit $\circ$. $\forall \alpha : n^3 + 1 \leq n^3 + n$, le couple $(\, i \,,\, j \,)$ tel que $i \circ j = \alpha$ est unique. En effet, il n'y a que $n^3 + k$ dans la définition de $\circ$ qui soit supérieur à n^3 et $\forall k \in [n]$, un seul j existe tel que $j = kn^2$. $\forall \alpha : 2 \leq \alpha \leq n^3$, outre $i \circ j = 1$, les deux conditions restantes sont mutuellement exclusive, α peut donc être le résultat d'au plus deux produits distincts.

On doit aussi montrer que la réduction préserve l'inclusion dans le langage. C'est-à-dire qu'on veut $x \in$ Gen $\iff$ $f\,(x) \in$ Gen-2. Il suffit de simplifier un $f\,(x)$ sans changer le fait que le dernier élément $n^3 + n$ est dans la fermeture de $\{\,1\,\}$ selon $\circ$. Première simplification, on retire l'unicité des éléments dans la section du carré, $(\,(\, i - n^3 \bullet j - n^3 \,) - 1 \,)\, n^2 + (\, i - n^3 - 1 \,)\, n + (\, j - n^3 \,)$ devient donc $(\, i - n^3 \bullet j - n^3 \,)\, n^2$. On retire donc la "cascade" des $1 \circ j = j + 1$ de la fermeture de $\{\,1\,\}$ selon $\circ$, si et seulement si cette dernière incluait un élément α c'est toujours le cas parce qu'à présent on le génère directement. Seconde simplification, on retire tout simplement les "cascades" qui ne sont plus utilisées, ceci ne change pas la fermeture de $\{\,1\,\}$ selon $\circ$.

On se retrouve donc avec une opération $i \circ j$ modifiée :

$$\begin{cases} n^3 + k & \text{si } i = 1 \text{ et } \exists k \in [n] : j = kn^2 \\ (\, i - n^3 \bullet j - n^3 \,)\, n^2 & \text{si } i, j > n^3 \\ 1 & \text{sinon} \end{cases}$$

Cette dernière opération est semblable à $\bullet$ décrite par x. En effet, $i \bullet j = \alpha \iff i + n^3 \bullet j + n^3 = \alpha n^2$ et $1 \circ \alpha n^2 = n^3 + \alpha$. Donc, n est dans la fermeture de $\{\,1\,\}$ selon $\bullet$ si et seulement si $n^3 + n$ est dans la fermeture de $\{\,1\,\}$ selon notre $\circ$ modifié. □

4.2. Comparer un Témoin de Gen avec les Circuits Booléens

Définition 4.17. Soit f une fonction, alors f est *log-constructible* si $\exists M$ un mT qui, sur entrée 1^n, calcule $1^{f(n)}$ sur son ruban de sortie, en utilisant au plus $\mathcal{O}\,(\log\,(n))$ espace.

Définition 4.18. Soit f une fonction, GenR-TailleTem (f) [resp. GenR-HautTem (f)]
Donnée : Un tableau $n \times n$ qui décrit une opération binaire $\bullet$ sur $[n]$ et un sous-ensemble $R \subseteq [n]$.
Question : Est-ce que $\exists$ un témoin de $n \in \langle R \rangle_\bullet$ de taille [resp. hauteur] au plus $f(n)$?

Proposition 4.19. Soit $f \in \Omega(\log(n))$ et log-constructible, alors GenR-HautTem $(f) \in$ NAuxPDA $\left(2^{\mathcal{O}(f(n))}, \mathcal{O}(\log(f(n)) + \log(n))\right)$.

Démonstration. Soit $\bullet$ une opération binaire sur $[n]$, R un sous-ensemble non-vide de $[n]$. Avant de construire une AuxNPDA A qui décide si $\exists$ un témoin de $n \in \langle R \rangle_\bullet$ de hauteur au plus $f(n)$, il faut définir ParcoursTémoin (h, k).

Pour décrire un témoin de hauteur au plus $f(n)$, il suffit d'une séquence C d'au plus $2^{f(n)} - 1$ paires de $(i, j) \in [n]^2$, chacune représentant les enfants d'un sommet interne du témoin. Afin de déterminer si un C donné – qu'on peut imaginer sur un ruban en lecture seule avec sa propre tête – décrit un témoin de $n \in \langle R \rangle_\bullet$ de hauteur au plus $f(n)$, un algorithme accepte si ParcoursTémoin $(f(n), n)$ ne refuse pas, où ParcoursTémoin (h, k) est :

1: **procédure** ParcoursTémoins(h, k)
2: **si** $k \notin R$ **alors**
3: $i, j \leftarrow$ Prochaine paire (i, j) dans C
4: **si** $h = 0$ ou $i \bullet j \neq k$ **alors**
5: arrêt de l'algorithme et refus
6: **sinon**
7: ParcoursTémoin $(h - 1, i)$
8: ParcoursTémoin $(h - 1, j)$
9: **fin si**
10: **fin si**
11: **fin procédure**

On pose $\mathrm{P}_0(h)$ vérifié si $\forall k \in [n]$ tel que $\nexists$ un témoin de $k \in \langle R \rangle_\bullet$ de hauteur au plus h, ParcoursTémoin (h, k) refuse $\forall$ séquence C donnée. Dans le cas de base et le pas inductif, on a $k \notin R$ sinon il existerait un témoin d'hauteur 0. Pour $\mathrm{P}_0(0)$, on a donc un refus peu importe C parce que $h = 0$. Pour $\mathrm{P}_0(h+1)$, on suppose que la paire (i, j) est telle que $i \bullet j = k$, sinon le prédicat est immédiatement vérifié. Étant donné que $\nexists$ un témoin de $k \in \langle R \rangle_\bullet$ de hauteur au plus $h + 1$, on doit avoir $\nexists$ un témoin de i ou $j \in \langle R \rangle_\bullet$ de hauteur au plus h (on suppose sans perte de généralité que c'est i). Par hypothèse d'induction, on a que ParcoursTémoin (h, i) refuse $\forall$ séquence C donnée.

On pose $\mathrm{P}_1(h)$ vérifié si $\forall k \in [n]$ tel que $\exists$ un témoin de $k \in \langle R \rangle_\bullet$ de hauteur au plus h, $\exists$ une séquence C telle que ParcoursTémoin(h, k) ne refuse pas. Parce que $h = 0$, on a $\mathrm{P}_1(h)$ étant donné que $k \in R$. Pour $\mathrm{P}_1(h+1)$, on pose i et j tels que $i \bullet j = k$ et $\exists$ témoins de $i, j \in \langle R \rangle_\bullet$ de hauteurs au plus h, ce qui est possible sinon on ne pourrait avoir $\exists$ un témoin de $k \in \langle R \rangle_\bullet$ de hauteur au plus $h+1$. Ainsi, on peut supposer que la prochaine paire dans C est (i, j) et, par hypothèse d'induction, on a que ParcoursTémoin(h, i) et ParcoursTémoin(h, j) ne refusent pas.

Avec $\mathrm{P}_0(h+1)$ et $\mathrm{P}_1(h+1)$ vérifiés, on sait que l'algorithme qui l'utilise accepte si et seulement si $\exists$ un témoin de $k \in \langle R \rangle_\bullet$ de hauteur au plus $f(n)$ et que C est la – ou l'une des – séquence qui mène à l'acceptation.

Maintenant, on construit A qui simule l'algorithme. L'AuxNPDA utilise sa pile en tant que pile d'exécution (i.e. *call stack*) afin d'effectuer les appels récursifs de ParcoursTémoin(h, k). Pour ce qui est de la séquence C, celle-ci n'est pas donnée en entrée, mais plutôt construite pendant l'exécution à partir de choix non-déterministe. En effet, on remplace la ligne 3 par un choix non-déterministe de (i, j) parmi $[n]^2$.

Avant de calculer les bornes d'espace et de temps de A, on note $\exists \alpha_1, \alpha_2 \in \mathbb{N}$ tels que $\forall$ mot $w \in \Sigma^*$ qui encode $\bullet$ et R, $|w| \leq \alpha_1 n^{\alpha_2}$.

Le temps de calcul de A est principalement borné par la taille du témoin $\in 2^{\mathcal{O}(f(n))}$. Pour chaque sommet, A doit – outre l'arithmétique et la lecture du ruban – empiler et dépiler le cadre d'exécution (i.e. *stack frame*) de ParcoursTémoin(h, k) qui est de taille $\in \mathcal{O}(\log(f(n)) + \log(n))$ afin de mémoriser h, i et j. Un petit détail est que A doit calculer $f(n)$ en temps $2^{\mathcal{O}(\log(n))}$, ce qui ne change pas la borne car $f \in \Omega(\log(n))$. La borne supérieure du temps de calcul est donc $2^{\mathcal{O}(f(|w|))}$.

Étant donné que A réutilise son espace pour simuler chaque appel de ParcoursTémoin(h, k) et qu'on sait que la taille de ceci est $\in \mathcal{O}(\log(f(n)) + \log(n))$. L'espace utilisé pour calculer $f(n)$ ne change pas la borne d'espace non plus. La borne supérieure d'espace est donc $\mathcal{O}(\log(f(|w|)) + \log(|w|))$.

Pour finir, on a que A accepte si et seulement si $\exists$ une séquence C qui décrit un témoin de $n \in \langle R \rangle_\bullet$ de hauteur au plus $f(n)$. Avec les bornes de temps et d'espace calculées plus haut, on a GenR-HautTem$(f) \in \mathsf{NAuxPDA}\left(2^{\mathcal{O}(f(n))}, \mathcal{O}(\log(f(n)) + \log(n))\right)$.

□

Corollaire 4.20. Soit f une fonction log-constructible et constructible en temps, alors GenR-TailleTem$(f) \in \mathsf{NAuxPDA}(\mathcal{O}(f(n)), \mathcal{O}(\log(f(n)) + \log(n)))$.

Démonstration. Afin d'appliquer la démonstration de la proposition 4.19 à GenR-TailleTem (f), il faut effectuer deux changements.

On réutilise le résultat que ParcoursTémoin (h, k) ne refuse pas si $\exists$ un témoin de $k \in \langle R \rangle_\bullet$ de hauteur au plus h et que la bonne séquence C est donnée et refuse s'il n'existe pas de tel témoin, peu importe la séquence C donnée. Étant donné que $f(n)$ représente ici la taille et non la hauteur, la simulation directe de l'algorithme est incorrecte car elle accepte les mots où $\exists$ un témoin de $n \in \langle R \rangle_\bullet$ de taille entre $f(n)$ et $2^{f(n)}$. La solution est que A doit compter le nombre d'appels de ParcoursTémoin (h, k) – le nombre de sommets du témoin parcourus – et refuser dès que ce compte excède $f(n)$.

L'autre changement est le temps de calcul. Étant donné que la taille du témoin est maintenant $\mathcal{O}(f(n))$ ceci devient la nouvelle borne supérieure.

La borne sur l'espace est inchangée puisque $\mathcal{O}(\log(f(n)) + \log(n))$ inclut déjà l'espace requit pour le compteur du nombre de sommets parcourus $\log(f(n))$.

□

Proposition 4.21. Soit $\alpha \in \mathbb{N}, f \in \Omega(\log^\alpha(n))$, alors GenR-HautTem (f) est SAC^α-difficile selon $\leq_{\log}$.

Démonstration. Soit $(C'_n)_{n\in\mathbb{N}}$ la famille de circuit uniforme qui décide un langage $Y \in \mathsf{SAC}^\alpha$ quelconque. Avant de réellement commencer la preuve, on doit transformer $\forall n \in \mathbb{N}, C'_n$ en un circuit C_n fonctionnellement équivalent.

Afin de pouvoir supposer que chaque conjonction est la seule avec sa paire d'arguments, on remplace chaque arête entre un parent k et un de ses enfants i, par $k = i \vee 1$. Parce que cette transformation se calcule en espace $\mathcal{O}(fofOne\log|\langle C'_n \rangle|)$, on sait que $(C_n)_{n\in\mathbb{N}}$ est aussi uniforme. On a donc $\exists c \in \mathbb{N}$ tel que $\forall n \in \mathbb{N}$ la profondeur de C_n est d'au plus $c\log^\alpha(n)$. La mTd M_c calcule donc sur $w \in \{0, 1\}^n$ la réduction qui permet d'obtenir $Y \leq_{\log}$ GenR-HautTem (f).

On considère que chaque entrée w_i [resp. $\neg w_i$] dans C_n est plutôt une constante à valeur w_i [resp. $\neg w_i$]. On suppose sans perte de généralité que le sommet $N = |C_n|$ (celui avec l'identifiant maximum) est le sommet résultat.

M_c calcule la description du produit sur $\mathbf{N} = \{(0, k), (1, k) : k \in [N^c]\}$ tel qu'une conjonction $k = i \wedge j$ devient $(1, i) \bullet (1, j) = (1, k)$, une disjonction $k = k = \bigvee_{j=1}^m i_j$ devient $(0, k) \bullet (1, i_j) = (1, k)$, et $(0, 1) \bullet (1, N) = (1, N^c)$. Pour tous les autres produits, le résultat est $(0, 1)$.

Ce produit sur $\mathbf{N}$ a une définition valide et non-ambiguë. Pour les conjonctions, étant donné que la porte k est l'unique ayant i et j comme arguments, $(1, i) \bullet (1, j)$ doit être $(1, k)$. Pour

les disjonctions, même si i est répété comme argument d'une même porte k, tous les produits $(0\,,\,k)\bullet(1\,,\,i)$ donnent le même résultat. De plus, $(1\,,\,N)$ ne peut être l'argument d'une porte car N est l'identifiant du sommet résultat, donc $(0\,,\,1)\bullet(1\,,\,N)$ doit être $(1\,,\,N^c)$.

On pose $R = \{\,(0\,,\,k)\,:\,k\in[N]\,\}\cup\{\,(1\,,\,k)\,:\,k$ identifie un sommet constant $1\,\}$.

Parce que $(C_n)_{n\in\mathbb{N}}$ est uniforme, M_c sur entrée $w\in\{0\,,\,1\}^n$ peut calculer la description de C_n en espace logarithmique. Aussi, $|C_n|\in n^{\mathcal{O}(1)}$ parce que $Y\in\mathsf{SAC}^\alpha$, donc $|\mathbf{N}| = 2\,|C_n|^c \in n^{\mathcal{O}(1)}$. Étant donné la taille polynomiale de la description du produit, M_c sur w utilise un espace $\mathcal{O}(\log(|w|))$.

On remarque que la définition du produit est sur l'ensemble $\mathbf{N}$, ce n'est que pour faciliter l'explication. En réalité, M_c opère sur $[n']$ où $n' = 2N^c$. Par une simple bijection, $(i\,,\,k)$ devient le nombre $in+k$.

Maintenant que le produit $\bullet$ sur $\mathbf{N}$ et l'ensemble R sont bien définis, il faut démontrer la validité de M_c en tant que réduction de Y à GenR-HautTem(f). Pour ce faire, on utilise les prédicats $\mathrm{P}_0(h)$ et $\mathrm{P}_1(h)$.

Soit $\mathrm{P}_0(h)$ vérifié si $\forall k\in[N]$, (la valeur du sommet k de hauteur au plus h est 0) $\implies$ ($\nexists$ témoin de $(1\,,\,k)\in\langle R\rangle_\bullet$). On a $\mathrm{P}_0(0)$ parce qu'un sommet k de hauteur et valeur 0 est une constante 0, donc $(1\,,\,k)$ n'est ni le résultat d'un produit, ni dans R.

On traite deux cas pour $\mathrm{P}_0(h+1)$. Si $k = i\wedge j$, alors (sans perte de généralité) le sommet i est de hauteur au plus h et de valeur 0. Par hypothèse d'induction, $\nexists$ témoin de $(1\,,\,i)\in\langle R\rangle_\bullet$. Parce que $(1\,,\,k)$ n'est présent qu'en tant que résultat de $(1\,,\,i)\bullet(1\,,\,j)$, $\nexists$ témoin de $(1\,,\,k)\in\langle R\rangle_\bullet$. Si $k = \bigvee_{j=1}^m i_j$, alors tous les sommets i_j sont de hauteur au plus h et de valeur 0. Par hypothèse d'induction, $\forall i_j$, $\nexists$ témoin de $(1\,,\,i_j)\in\langle R\rangle_\bullet$. Parce que $(1\,,\,k)$ ne sont présents qu'en tant que résultats de $(0\,,\,k)\bullet(1\,,\,i_j)$, $\nexists$ témoin de $(1\,,\,k)\in\langle R\rangle_\bullet$.

Notre second prédicat, $\mathrm{P}_1(h)$, est vérifié si $\forall k\in[N]$, (la valeur du sommet k de hauteur au plus h est 1) $\implies$ ($\exists$ témoin de $(1\,,\,k)\in\langle R\rangle_\bullet$ de hauteur au plus h). On a $\mathrm{P}_1(0)$ parce qu'un sommet k de hauteur 0 et de valeur 1 est une constante 1, donc $\exists$ témoin de $(1\,,\,k)\in\langle R\rangle_\bullet$ de hauteur 0 car $(1\,,\,k)\in R$.

Similairement à $\mathrm{P}_0(h+1)$, on traite deux cas pour $\mathrm{P}_1(h+1)$. Si $k = i\wedge j$, alors $i,\,j$ sont tous deux de hauteur au plus h et de valeur 1. Par hypothèse d'induction, $\exists$ témoins de $(1\,,\,i)\in\langle R\rangle_\bullet$ et de $(1\,,\,j)\in\langle R\rangle_\bullet$ chacun de hauteur au plus h. Parce que $(1\,,\,i)\bullet(1\,,\,j) = (1\,,\,k)$, on a $\exists$ témoin de $(1\,,\,k)\in\langle R\rangle_\bullet$ de hauteur au plus $h+1$. Si $k = \bigvee_{j=1}^m i_j$, alors $\exists i_j$ un sommet de hauteur au plus h et de valeur 1. Par hypothèse d'induction, $\exists$ témoin de $(1\,,\,i_j)\in\langle R\rangle_\bullet$ de hauteur au plus h. Parce que $(0\,,\,k)\bullet(1\,,\,i_j) = (1\,,\,k)$, on a $\exists$ témoin de $(1\,,\,k)\in\langle R\rangle_\bullet$ de hauteur au plus $h+1$.

Par $\mathrm{P}_1(h)$, on a donc que la valeur du sommet N est 1 implique $\exists$ témoin de $(1, N) \in \langle R \rangle_\bullet$ de hauteur au plus $c\log^\alpha(n)$ et, parce que $(0, 1) \bullet (1, N) = (1, N^c)$, implique $\exists$ témoin de $n' \in \langle R \rangle_\bullet$ de hauteur au plus $c\log^\alpha(n)+1$. Ce dernier témoin est donc aussi de hauteur au plus $\log^\alpha(n' = 2N^c)$, parce que $N = |C_n| \geq n$. Puis, $\mathrm{P}_0(h)$ permet de conclure que la valeur du sommet N est 1 si et seulement si $\exists$ témoin de $n' \in \langle R \rangle_\bullet$ de hauteur au plus $\log^\alpha(n')$.

Pour conclure, $\forall Y \in \mathsf{SAC}^\alpha$, $\exists M_c$ tel que $\forall w$, $w \in Y$ si et seulement si $M_c(w) \in$ GenR-HautTem(f) et donc, parce que M_c fonctionne en espace logarithmique en $|w|$, $Y \leq_{\log}$ GenR-HautTem(f). □

Corollaire 4.22. Soit $\alpha \in \mathbb{N}$, $f \in 2^{\Omega(\log^\alpha(n))}$, alors GenR-TailleTem$(f)$ est SAC^α-difficile selon $\leq_{\log}$.

Démonstration. Le seul changement qu'il faut effectuer par rapport à la proposition 4.21, est dans le paragraphe précédant la conclusion.

On sait que par $\mathrm{P}_1(h)$, la valeur du sommet N est 1 implique $\exists$ témoin de $n' \in \langle R \rangle_\bullet$ de hauteur au plus $\log^\alpha(n')$. Ce dernier témoin est donc de taille au plus $2^{\log^\alpha(n')}$. Puis, $\mathrm{P}_0(h)$ permet de conclure que la valeur du sommet N est 1 si et seulement si $\exists$ témoin de $n' \in \langle R \rangle_\bullet$ de taille au plus $2^{\log^\alpha(n')}$. □

Corollaire 4.23. $\forall \alpha \in \mathbb{N}$, GenR-HautTem$(\log^\alpha(n))$ est SAC^α-complet selon $\leq_{\log}$.

Démonstration. Soit $f(n) = \log^\alpha(n)$, on a que f est log-constructible car une mT – en espace logarithmique – peut effectuer une multiplication avec des nombres bornés par $\log^\alpha(n)$. Ainsi, la proposition 4.19 nous donne GenR-HautTem$(\log^\alpha(n)) \in \mathsf{NAuxPDA}\left(2^{\mathcal{O}(\log^\alpha(n))}, \mathcal{O}(\log(n))\right)$. Combinée avec [NR95][Ven87] qui montre $\mathsf{SAC}^\alpha = \mathsf{NAuxPDA}\left(2^{\mathcal{O}(\log^\alpha(n))}, \log(n)\right)$, on obtient la borne supérieure. Pour la borne inférieure, il suffit d'appliquer directement la proposition 4.21. □

Corollaire 4.24. $\forall k \in \mathbb{N}$, GenR-TailleTem$\left(n^k\right)$ est SAC^1-complet selon $\leq_{\log}$.

Démonstration. Soit $f(n) = n^k$, on a que f est log-constructible et constructible en temps car une mT – en espace logarithmique – peut effectuer une multiplication avec des nombres sur $k\log(n)$ bits. Ainsi, le corollaire 4.20 nous donne GenR-TailleTem$\left(n^k\right) \in \mathsf{NAuxPDA}\left(\mathcal{O}\left(n^k\right), \mathcal{O}(\log(n))\right)$. Combiné avec [NR95][Ven87] qui montre $\mathsf{SAC}^1 = \mathsf{NAuxPDA}\left(2^{\mathcal{O}(\log(n))}, \log(n)\right)$, on obtient la borne supérieure parce que $\mathcal{O}\left(n^k\right) \subseteq 2^{\mathcal{O}(\log(n))}$. Pour la borne inférieure, il suffit d'appliquer directement le corollaire 4.23. □

4.3. GenPath Complet pour NP

Définition 4.25. Soit $G = (S, A)$ un graphe orienté et f une fonction $A \to S$, alors $G = (S, A, f)$ est un *graphe orienté avec étiquettes*.

Note 4.26. On encode un graphe $G = (S, A)$ avec sa matrice binaire d'adjacence. Le mot $w = \langle S, A \rangle$ est donc simplement un tableau de $|S|^2$ bits. Lorsque qu'on doit aussi encoder f la fonction $A \to S$ qui étiquette G, on ajoute un tableau de $|S|^2$ nombres chacun sur $\lceil \log(|S|) \rceil$ bits.

Définition 4.27. Soit $G = (S, A, f)$ un graphe orienté avec étiquettes. Alors le chemin (possiblement non-simple) sur les sommets $s_0, s_1, \ldots, s_l$ est *générable* si $\forall i \in [l]\, f(s_{i-1}, s_i) \in \{s_0, s_1, \ldots, s_{i-1}\}$

Définition 4.28. GenPath
Donnée : $G = ([n], A, f)$ un graphe orienté avec étiquettes.
Question : $\exists$ un chemin (possiblement non-simple) *générable* du sommet 1 au sommet n, dans G ?

Proposition 4.29. GenPath $\in$ NP.

Démonstration. Voici le calcul de N, une mTnd, sur $w = \langle [n], A, f \rangle$:

(1) Choisir de manière non-déterministe l parmi $\{0, 1, \ldots, n^2\}$ et l'écrire sur le ruban.

(2) Choisir de manière non-déterministe un chemin (possiblement non-simple) C sur les sommets $s_0, s_1, \ldots, s_l$ de longueur l de 1 à n et l'écrire sur le ruban, sinon refuser.

(3) Accepter si $\forall i \in [l]\, f(s_{i-1}, s_i) \in \{s_0, s_1, \ldots, s_{i-1}\}$, sinon refuser.

Quels que soient l et C, les deux premiers points s'effectuent en $n^{\mathcal{O}(1)}$ étapes non-déterministes. Pour un seul i, la vérification de $f(s_{i-1}, s_i) \in \{s_0, s_1, \ldots, s_{i-1}\}$ s'effectue en $n^{\mathcal{O}(1)}$ étapes déterministes. Ainsi, le dernier point s'effectue aussi en $n^{\mathcal{O}(1)}$ étapes déterministes, car il suffit de boucler sur $i \in [l]$. On a donc $\exists c \in \mathbb{N}$ tel que N sur toutes entrées w, peu importe les choix non-déterministes, fonctionne en temps $|w|^c$, parce que $|w| \in n^{\mathcal{O}(1)}$.

Si N accepte $w = \langle [n], A, f \rangle$ alors la suite de choix non-déterministes ayant mené à l'acceptation identifie un chemin de 1 à n (par le point deux) et que ce dernier est générable (par le point trois).

Inversement, supposons $w \in$ GenPath. Pour garantir l'acceptation par N nous devons démontrer que si $\exists$ un chemin non-simple générable de 1 à n de longueur $l > n^2$, alors $\exists$ un chemin (possiblement non-simple) générable de 1 à n de longueur $l' \leq n^2$.

En effet, soit un chemin non-simple générable C de 1 à n de longueur $l > n^2$, on note $D = (s_0, s_1, \ldots, s_l)$ la séquence des sommets traversés par C. Alors $\exists t \in S$ tel que D contient t

exactement $m > n$ fois. Pour $j \in [m]$, on pose A_j l'ensemble des sommets qui précèdent le j-ème t dans D. Parce que $A_1 \subseteq A_2 \subseteq \ldots \subseteq A_m$, on doit avoir $\exists k \in [m-1] : A_k = A_{k+1}$. On construit D' en retirant les sommets après le k-ème t jusqu'au t suivant, inclusivement. Il existe un nouveau chemin (possiblement non-simple) sur D' celui-ci est de 1 à n (car on retire un circuit d'un chemin valide) et reste générable (car $A_k = A_{k+1}$). Ce nouveau chemin est de longueur $l' < l$, si on n'a pas $l' \leq n^2$, il suffit de répéter l'argument jusqu'à ce que ce soit le cas.

Ainsi, $\exists$ une suite de choix non-déterministes telle que N accepte sur $w = \langle [n] \, , \, A \, , \, f \rangle$ si et seulement si $\exists$ un chemin (possiblement non-simple) générable du sommet 1 au sommet n, dans $G = ([n] \, , \, A \, , \, f)$, et refuse sinon. □

Définition 4.30. HamPath
Donnée : $G = ([n] \, , \, A)$ un graphe orienté.
Question : $\exists$ un chemin hamiltonien du sommet 1 au sommet n, dans G ?

Proposition 4.31. HamPath $\leq_{\log}$ GenPath.

Démonstration. Voici comment notre mTd M construit $w' = \langle G' = (S' \, , \, A' \, , \, f) \rangle$, à partir de $w = \langle [n] \, , \, A \rangle$. On explique la construction en deux parties.

Le premier sous-graphe de G' est une application de la technique de *timestamp* utilisée dans [Jon75, Corollaire 25], donc M copie les sommets $[n]$ initiaux n fois, puis ajoute un arc d'un sommet d'une copie au sommet de la suivante s'il y a un tel arc dans A. Formellement :

$$S'_1 = \{ \, (i \, , \, s)_1 \, : \, i, s \in [n] \, \}$$

$$A'_1 = \{ \, ((i \, , \, s)_1 \, , \, (i+1 \, , \, t)_1) \, : \, i \in [n-1] \text{ et } (s \, , \, t) \in A \, \}$$

Le second sous-graphe de G' est une suite de gagdets qui permettent d'emprunter n chemin différents pour passer d'un sommet $(0 \, , \, s)_2$ au suivant $(0 \, , \, s+1)_2$:

$$S'_2 = \{ \, (i \, , \, s)_2 \, : \, 0 \leq i, s \leq n \, \}$$

$$A'_2 = \{ \, ((0 \, , \, s-1)_2 \, , \, (i \, , \, s-1)_2) \text{ et } ((i \, , \, s-1)_2 \, , \, (0 \, , \, s)_2) \, : \, i, s \in [n] \, \}$$

Pour compléter la construction de $G' = (S' = S'_1 \bigcup S'_2 \, , \, A' = A'_1 \bigcup A'_2 \, , \, f)$, il faut joindre les deux sous graphes. On ajoute donc à A' l'arc $((n \, , \, n)_1 \, , \, (0 \, , \, 0)_2)$. On considère le sommet $(1 \, , \, 1)_1$ la source et $(0 \, , \, n)_2$ la destination.

Le but de chaque gadget est de vérifier que le chemin qui y mène à partir de la source $(1 \, , \, 1)_1$ contient une des copies d'un sommet s, dans le premier sous-graphe de G'. Pour ce faire, la sortie de f est $(1 \, , \, 1)_1$ sur toutes entrées sauf :

$$\forall i, s \in [n] \, , f \, ((i \, , \, s-1)_2 \, , \, (0 \, , \, s)_2) = (i \, , \, s)_1$$

On note que M ne manipule G' en termes décrits plus haut que sur son ruban de travail. Lorsque M doit écrire un tuple sur son ruban de sortie, elle écrit plutôt un nombre. Donc, $(i\,,\,s)_1$ devient le nombre $(\,i-1\,)\,n+s$ et $(i\,,\,s)_2$ devient le nombre $n^2+(\,n-i\,)\,n+s+1$. Cette bijection, calculable en espace $\mathcal{O}\left(\log\left(|w|\right)\right)$, est telle que $(1\,,\,1)_1$ devient 1 et $(0\,,\,n)_2$ devient $n'=2n^2+n+1$, ce qui permet de respecter le format attendu de GenPath.

Avant de procéder à la preuve de correctitude, on note qu'un chemin hamiltonien correspond aussi à une permutation des sommets du graphe. En effet, un tel chemin doit être de longueur $n-1$. Similairement, un chemin de longueur $n-1$ est simple si et seulement si il contient tous les sommets du graphe.

Démontrons $w \in$ HamPath $\implies$ $w' \in$ GenPath. On suppose $\exists C$ un chemin hamiltonien sur les sommets $s_1, s_2, \ldots, s_n$ dans G de $s_1=1$ à $s_n=n$. Parce que C est simple et de longueur $n-1$, on sait $\exists C_0$ un chemin sur les sommets $(1\,,\,s_1)_1, (2\,,\,s_2)_1, \ldots, (n\,,\,s_n)_1, (0\,,\,0)_2$ dans G' et que celui-ci est générable car l'étiquette de chaque arc impliquée est $(1\,,\,1)_1$. Parce que C passe par chaque $s \in [n]$, on sait que $\forall s \in [n]\,, \exists i \in [n]$ tel que $f\left((i\,,\,s-1)_2\,,\,(0\,,\,s)_2\right)=(i\,,\,s)_1 \in C$. Donc $\forall s \in [n]\,, \exists i \in [n]$ tel que la prolongation de C_{s-1} par $\left((i\,,\,s-1)_2\,,\,(0\,,\,s)_2\right)$ demeure un chemin C_s générable. Ainsi C_n est un chemin générable de $(1\,,\,1)_1$ à $(0\,,\,n)_2$ dans G'.

Démontrons $w' \in$ GenPath $\implies$ $w \in$ HamPath. On suppose $\exists C'$ un chemin générable de $(1\,,\,1)_1$ à $(0\,,\,n)_2$ dans G'. Un tel chemin doit passer par $(n\,,\,n)_1$ afin d'accéder au second sous-graphe de G'. Donc, avec $s_1=1$ et $s_n=n$, C_0 un chemin sur les sommets $\left((1\,,\,s_1)_1\,,\,(2\,,\,s_2)_1\,,\,\ldots\,,\,(n\,,\,s_n)_1\,,\,(0\,,\,0)_2\right)$ est le préfixe de C'. On a bien $|C_0|=n$ promis par la technique de *timestamp*. Afin de se rendre à $(0\,,\,n)_2$, C' doit inclure un arc avec destination $(0\,,\,s)_2$ et ce $\forall s \in [n]$. Les seuls arcs avec $(0\,,\,s)_2$ comme destination, sont $\forall i \in [n]\,, \left((i\,,\,s-1)_2\,,\,(0\,,\,s)_2\right)$ et donc C' doit contenir un de ceux-ci. C' est aussi générable, il est donc nécessaire que $\forall s \in [n]\,, \exists i \in [n]$ tel que C_0 passe par $f\left((i\,,\,s-1)_2\,,\,(0\,,\,s)_2\right)=(i\,,\,s)_1$. On a donc $\exists C$ un chemin de longueur $n-1$ tel que $\forall s \in [n]$, C passe par s, donc hamiltonien, de 1 à n dans G.

Les conditions dans les définitions de S', A' et f se décident en espace $\mathcal{O}\left(\log\left(|w|\right)\right)$. Puisque le calcul de chaque arc (pour l'écrire en sortie dans A' et écrire son étiquette dans f) ne dépend que du sommet source et destination, on peut réutiliser cet espace logarithmique. Finalement, il suffit de répéter ce calcul un nombre proportionnel à $|A'|$, ce qui reste polynomial en $|w|$.

Ainsi, M calcul w' en espace $\mathcal{O}\left(\log\left(|w|\right)\right)$ tel que $w \in$ HamPath $\iff$ $w' \in$ GenPath.

□

Corollaire 4.32. GenPath est NP-complet selon $\leq_{\log}$

Démonstration. On sait transformer la réduction $\leq_{\mathsf{P}}$ dans [Coo71b] et [Sip12, Chapitre 7.5] afin d'obtenir 3Sat est NP-complet selon $\leq_{\log}$. Similairement, on peut obtenir 3Sat $\leq_{\log}$ HamPath à partir de [Kar72] et [Sip12, Chapitre 7.5]. On a donc, par la proposition 4.31 et la transitivité de $\leq_{\log}$, que HamPath est NP-difficile selon $\leq_{\log}$. Avec la proposition 4.29, on obtient la complétude. □

Chapitre 5

Conclusion

Bien que la preuve d'équivalence détaillée dans ce mémoire entre la classe des languages décidés par un AuxDPDA et LOGDCFL ne soit pas nouvelle, le niveau de formalisme avec lequel elle fut traitée dans ce texte l'est. L'observation qu'interdire de combiner ou de séparer un paquet de bits dans un circuit multiplex n'est pas nécessaire afin de conserver la puissance de calcul de la classe $\mathsf{MuxTapT}\left(F_1\,,\,F_2\right)$ est également nouvelle.

Le problème Gen-1 discuté n'est pas la première variante de Gen en lien avec NL. En effet, Jones et Laaser [JLL76] montrent que STConn se réduit à AGen – Gen où l'opération donnée est associative – et que AGen est dans NL. L'importante différence est que la preuve de complétude pour Gen-1 repose sur la notion du témoin. Pour ce qui est de montrer que Gen-2 est P-complet, cela nous donne un problème plus simple à manipuler que Gen pour démontrer une borne inférieure d'un autre problème.

Le corollaire 4.23 est similaire à la proposition 3.2 dans [BM91] qui montre qu'un exemplaire de Gen avec *bracketing depth* α – équivalent à GenR-HautTem $\left(\log^{\alpha}\left(n\right)\right)$ – est NC^{α}-difficile et inclu dans $\mathsf{NC}^{\alpha+1}$. On a donc un rafinement des bornes, étant donné que $\forall\alpha\in\mathbb{N},\mathsf{NC}^{\alpha}\subseteq\mathsf{SAC}^{\alpha}\subseteq\mathsf{AC}^{\alpha}\subseteq\mathsf{NC}^{\alpha+1}$.

5.1. Pistes Futures

Une idée intéressante est d'exiger que la description de chaque élément d'une famille de circuits multiplex uniforme respecte un ordre topologique. Un simple ajustement au lemme 3.9 le rendrait conforme à cette nouvelle définition. Ainsi, cette exigence supplémentaire préserve la classe $\mathsf{MuxTapT}\left(n^{\mathcal{O}(1)}\,,\,n^{\mathcal{O}(1)}\right)$.

Une piste de travaux futurs serait de généraliser le lemme 3.9 avec des fonctions quelconques. Pour ce faire, il faudrait paramétrer la taille des paquets de bits d'une famille de circuits multiplex. En effet, la borne d'espace du AuxDPDA simulé semble en lien direct avec la taille des paquets de bits, on ne peut donc pas introduire un paramètre dans l'un sans faire de même pour l'autre.

La question du lien entre LOGDCFL et NL n'est toujours pas résolue (le rapport de Mahajan [MT07] est malheureusement encore d'actualité à cet égard). C'est pourquoi il serait intéressant d'étudier un problème GenR-1-TailleTem $\left(n^{\mathcal{O}(1)}\right)$ combinant les restrictions de GenR-TailleTem $\left(n^{\mathcal{O}(1)}\right)$ et Gen-1. En effet, ce nouveau problème semble un bon candidat à la LOGDCFL-complétude car l'algorithme qui décide GenR-TailleTem $\left(n^{\mathcal{O}(1)}\right)$ – un problème démontré LOGCFL-complet dans le présent mémoire – pourrait être exécuté de manière déterministe avec la promesse de Gen-1. On aurait donc deux problèmes similaires qui capturent respectivement LOGDCFL et NL.

Bien qu'on ait trouvé un candidat potentiel pour LOGDCFL-complet, la situation est différente pour TreeEval. En effet, si l'on pouvait conjecturer la complétude de ce dernier pour LOGDCFL en 2023, une telle conjecture n'est plus viable car Cook et Mertz [CM24] ont prouvé que TreeEval est dans DEspace $(\mathcal{O}(\log(n)\log(\log(n))))$.

Une piste possible pour montrer la LOGDCFL-difficulté de GenR-1-TailleTem $\left(n^{\mathcal{O}(1)}\right)$ serait certainement d'utiliser les circuits multiplex comparablement à l'utilisation de circuits semi-bornés pour montrer la SAC^1-difficulté de GenR-TailleTem $\left(n^{\mathcal{O}(1)}\right)$.

Références bibliographiques

[BM91] David A Mix Barrington et Pierre McKenzie. « Oracle Branching Programs and Logspace versus P ». In : *Information and Computation* 95 (1991), p. 96-115.

[Bra+09] Mark Braverman et al. « Branching Programs for Tree Evaluation ». In : *Mathematical Foundations of Computer Science 2009*. Berlin, Heidelberg : Springer Berlin Heidelberg, 2009, p. 175-186. isbn : 978-3-642-03816-7. doi : 10.1007/978-3-642-03816-7_16.

[CM24] James Cook et Ian Mertz. « Tree Evaluation Is in Space O(log n · log log n) ». In : *Proceedings of the 56th Annual ACM Symposium on Theory of Computing*. New York, NY, USA : ACM, 2024, p. 1268-1278. doi : 10.1145/3618260.3649664.

[Coo69] Stephen A. Cook. « Variations on Pushdown Machines (Detailed Abstract) ». In : *Proceedings of the First Annual ACM Symposium on Theory of Computing*. Marina del Rey, CA, USA : ACM, mai 1969, p. 229-231. doi : 10.1145/800169.805437.

[Coo71a] Stephen A. Cook. « Characterizations of pushdown machines in terms of time-bounded computers ». In : *Journal of the ACM (JACM)* 18.1 (1971), p. 4-18.

[Coo71b] Stephen A. Cook. « The Complexity of Theorem-Proving Procedures ». In : *Proceedings of the Third Annual ACM Symposium on Theory of Computing*. ACM, 1971, p. 151-158. doi : 10.1145/800157.805047.

[FLR96] Henning Fernau, Klaus-Jörn Lange et Klaus Reinhardt. « Advocating Ownership ». In : *Foundations of Software Technology and Theoretical Computer Science*. Sous la dir. de V. Chandru et V. Vinay. T. 16. Hyderabad, India : Springer Berlin, Heidelberg, déc. 1996, p. 286-297. doi : 10.1007/3-540-62034-6_57.

[HU79] John E. Hopcroft et Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. 1re éd. Addison-Wesley, 1979.

[Imm88] Neil Immerman. « Nondeterministic Space is Closed under Complementation ». In : *SIAM Journal on Computing* 17.5 (1988), p. 935-938. doi : 10.1137/0217058.

[JL76] Neil D. Jones et William T. Laaser. « Complete Problems for Deterministic Polynomial Time ». In : *Theoretical Computer Science* 3.1 (1976), p. 105-117. doi : 10.1016/0304-3975(76)90068-2.

[JLL76] Neil D. Jones, Y. Edmund Lien et William T. Laaser. « New Problems Complete for Nondeterministic Log Space ». In : *Mathematical Systems Theory* 10.1 (1976), p. 1-17. doi : 10.1007/BF01683259.

[Joh17] Richard Johnsonbaugh. *Discrete mathematics (8th ed.)* en. Addison-Wesley Longman, 2017.

[Jon75] Neil D. Jones. « Space-bounded Reducibility Among Combinatorial Problems ». In : *Journal of Computer and System Sciences* 11.1 (1975), p. 68-85. doi : 10.1016/S0022-0000(75)80050-X.

[Kar72] Richard M. Karp. « Reducibility Among Combinatorial Problems ». In : *Complexity of Computer Computations*. Plenum Press, 1972, p. 85-103.

[MRV99] Pierre McKenzie, Klaus Reinhardt et V. Vinay. « Circuits and Context-Free Languages ». In : *Computing and Combinatorics*. Sous la dir. de Takano Asano et al. T. 5. Tokyo, Japan : Springer Berlin, Heidelberg, juill. 1999, p. 194-203. doi : 10.1007/3-540-48686-0_19.

[MT07] Meena Mahajan et Jacobo Toran. « Polynomial Size Log Depth Circuits : Between NC1 and AC1 ». In : *Bulletin of the European Association for Theoretical Computer Science* 91 (2007), p. 42-56. issn : 0252-9742. url : https://www.eatcs.org/images/bulletin/beatcs91.pdf.

[NR95] Rolf Niedermeier et Peter Rossmanith. « Unambiguous Auxiliary Pushdown Automata and Semi-unbounded Fan-in Circuits ». In : *Information and Computation* 118.2 (1995), p. 227-245. doi : 10.1006/inco.1995.1064.

[Pér14] Sylvain Périfel. *Complexité Algorithmique*. fr. Paris, France : Ellipses-Marketing, avr. 2014.

[Rei97] Klaus Reinhardt. « Strict Sequential P-Completeness ». In : *STACS 97*. Sous la dir. de Rüdiger Reischuk et Michel Morvan. Berlin, Heidelberg : Springer Berlin Heidelberg, 1997, p. 329-338. isbn : 978-3-540-68342-1. doi : 10.1007/BFb0023470.

[Ros02] Kenneth H. Rosen. *Mathématiques Discrètes*. Chenelière McGraw-Hill, 2002.

[Sip12] Michael Sipser. *Introduction to the Theory of Computation*. 3e éd. MIT Press, 2012.

[Sud78] Ivan Hal Sudborough. « On the Tape Complexity of Deterministic Context-Free Languages ». In : *Journal of the ACM (JACM)* 25.3 (1978), p. 405-414.

[Sze88] Róbert Szelepcsényi. « The Method of Forced Enumeration for Nondeterministic Automata ». In : *Acta Informatica* 26.3 (nov. 1988), p. 279-284. doi : 10.1007/BF00299636.

[Ven87] H. Venkateswaran. « Properties that Characterize LOGCFL ». In : *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*. T. 999. STOC '87. New York, NY, USA : Association for Computing Machinery, 1987, p. 141-150. isbn : 0897912217. doi : 10.1145/28395.28411.

[Yam23] Tomoyuki Yamakami. « Between SC and LOGDCFL : families of languages accepted by logarithmic-space deterministic auxiliary depth-k storage automata ». In : *International Journal of Computer Mathematics : Computer Systems Theory* 8.1 (2023), p. 1-31. doi : 10.1080/23799927.2023.2166872.