Université de Montréal

Complexité raffinée du problème d'intersection d'automates

par
Michael Blondin

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

Mémoire présenté à la Faculté des arts et des sciences
en vue de l'obtention du grade de Maître ès sciences (M.Sc.)
en informatique

janvier, 2012

Université de Montréal
Faculté des arts et des sciences

Ce mémoire intitulé:

Complexité raffinée du problème d'intersection d'automates

présenté par:

Michael Blondin

a été évalué par un jury composé des personnes suivantes:

Michel Boyer,	président-rapporteur
Pierre McKenzie,	directeur de recherche
Louis Salvail,	membre du jury

Mémoire accepté le: 9 mai 2012

RÉSUMÉ

Le problème d'intersection d'automates consiste à vérifier si plusieurs automates finis déterministes acceptent un mot en commun. Celui-ci est connu PSPACE-complet (resp. NL-complet) lorsque le nombre d'automates n'est pas borné (resp. borné par une constante).

Dans ce mémoire, nous étudions la complexité du problème d'intersection d'automates pour plusieurs types de langages et d'automates tels les langages unaires, les automates à groupe (abélien), les langages commutatifs et les langages finis. Nous considérons plus particulièrement le cas où chacun des automates possède au plus un ou deux états finaux. Ces restrictions permettent d'établir des liens avec certains problèmes algébriques et d'obtenir une classification intéressante de problèmes d'intersection d'automates à l'intérieur de la classe P. Nous terminons notre étude en considérant brièvement le cas où le nombre d'automates est fixé.

Mots clés : Intersection d'automates, problèmes algébriques, complexité du calcul, espace logarithmique.

ABSTRACT

The automata non emptiness intersection problem is to determine whether several deterministic finite automata accept a word in common. It is known to be PSPACE-complete (resp. NL-complete) whenever the number of automata is not bounded (resp. bounded by a constant).

In this work, we study the complexity of the automata intersection problem for several types of languages and automata such as unary languages, (abelian) group automata, commutative languages and finite languages. We raise the issue of limiting the number of final states to at most two in the automata involved. This way, we obtain relationships with some algebraic problems and an interesting classification of automata intersection problems inside the class P. Finally, we briefly consider the bounded version of the automata intersection problem.

Keywords: Automata intersection, algebraic problems, computational complexity, logspace.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

NOTATION

$\mathbb{Z}$	ensemble des entiers : $\{\ldots, -2, -1, 0, 1, 2, \ldots\}$
$\mathbb{N}$	ensemble des entiers naturels : $\{0, 1, 2, \ldots\}$
$\mathbb{Z}_q$	ensemble des entiers modulo q : $\{0, 1, \ldots, q-1\}$
$\emptyset$	ensemble vide : $\{\}$
$\mathcal{P}(X)$	ensemble de tous les sous-ensembles de X
$[k]$	ensemble des entiers de 1 à k : $\{1, \ldots, k\}$
$\langle g_1, \ldots, g_k \rangle$	ensemble des éléments engendrés par $g_1, \ldots, g_k$
$\text{ppcm}(a_1, \ldots, a_k)$	plus petit commun multiple de $a_1, \ldots, a_k$
$\text{pgcd}(a_1, \ldots, a_k)$	plus grand commun diviseur de $a_1, \ldots, a_k$
$\log^c(x)$	puissance c du logarithme en base 2 de x
$a \bmod b$	reste de la division entière $a \div b$
$a \equiv b \pmod{q}$	$a \bmod q = b \bmod q$
n	taille de l'entrée d'un problème
s	taille de Σ
ssi	si et seulement si

« Das Wesen der Mathematik liegt in ihrer Freiheit » – Georg Cantor

« Els homes no poden ser si no son lliures » – Salvador Espriu

« Tant que l'indépendance n'est pas faite, elle reste à faire » – Gaston Miron

Je me souviens de la grève étudiante

REMERCIEMENTS

Je remercie mon directeur de recherche Pierre McKenzie. Je le remercie de m'avoir introduit à l'informatique théorique et de m'y avoir entraîné malgré lui. Je le remercie pour son enthousiasme, son encadrement exemplaire et sa rigueur. Je lui serai éternellement reconnaissant.

Je remercie ma famille, mes collègues et ami·e·s Michaël Cadilhac, Phillippe Grand'Maison, Philippe Lamontagne, Rébecca Lapointe, Mathieu Larose, François Saint-Jacques, les membres du LITQ ainsi que tous et toutes les autres qui ont pu rendre mes études plus agréables.

Je remercie Adrianne Gauvin-Sasseville pour son soutien moral, ses encouragements, sa patience et sa présence. Il serait prétentieux de lui dédier ce mémoire, car sa maîtrise de la complexité de ce monde dépasse largement la mienne.

CHAPITRE 1

INTRODUCTION

Le *problème d'intersection d'automates*, IntAuto, est défini de la façon suivante :

Données : $A_1, \ldots, A_k$ des automates, sur alphabet Σ.

Question : $\exists w \in \Sigma^*$ accepté par A_i pour tout $i \in [k]$?

Le problème IntAuto généralise plusieurs problèmes étudiés dans la littérature. Par exemple, il généralise le problème d'appartenance (à un monoïde de transformations spécifié par un ensemble de tranformations génératrices) [Koz77], le problème « pointset transporter » [LM88] et le problème « set transporter » [LM88].

Le problème d'appartenance et le problème d'intersection d'automates ont été démontrés PSPACE-complets par Kozen [Koz77]. Par la suite, le lien entre le problème d'isomorphisme de graphes et certains problèmes concernant les groupes de permutations ont conduit à une étude approfondie du problème d'appartenance. En particulier, le problème d'appartenance pour les groupes de permutations a été placé dans P [FHL80], puis dans NC^3 pour les groupes abéliens [MC87, Mul87], dans NC pour les groupes nilpotents [LM88], les groupes résolubles [LM88], les groupes avec facteurs de composition non abéliens bornés [Luk86], et finalement tous les groupes [BLS87]. Une classification similaire de la complexité du problème d'appartenance pour les monoïdes finis apériodiques est due à [Koz77, Bea88a, BMT92] qui démontrent que le problème, pour n'importe quelle variété de monoïdes apériodiques fixée, appartient ou bien à AC^0, à P, à NP, ou à PSPACE (et est complet pour cette classe à peu d'exceptions près).

D'autre part, IntAuto a reçu moins d'attention. Cela est (fort probablement) dû au fait qu'il est équivalent au problème d'appartenance lorsqu'ils sont tous deux difficiles à résoudre, mais qu'il apparaît difficile à résoudre lorsque le problème d'appartenance est résoluble efficacement. Par exemple, Beaudry [Bea88b] montre que IntAuto pour les automates à groupe abélien et IntAuto pour les automates commutatifs et idempotents sont NP-complets. Beaudry indique que ces deux cas sont des

exemples où le problème d'intersection d'automates semble strictement plus difficile à résoudre que le problème d'appartenance (dont la complexité est NC^3 pour les groupes abéliens et AC^0 pour les monoïdes commutatifs et idempotents). De plus, des résultats de [Gal76] montrent que IntAuto est NP-complet même lorsque Σ est un singleton.

Toutefois, des résultats très intéressants découlent de l'étude de IntAuto. Par exemple, le cas où k est borné par une fonction $g(n)$ a été étudié dans [LR92]. Le problème y est montré NSPACE($g(n) \log n$)-complet sous réductibilité log-espace. Cela a vraisemblablement permis d'obtenir le premier problème naturel complet pour la classe NSPACE($\log^c n$). De plus, il a été démontré par Karakostas, Lipton et Viglas que la découverte de meilleurs algorithmes pour résoudre IntAuto avec un nombre constant d'automates impliquerait NL $\neq$ P [KLV03].

Plus récemment, le problème d'intersection a aussi été étudié pour les expressions régulières sans opérateur + [Bal02], plutôt que pour les automates. Il est montré PSPACE-complet pour les expressions d'hauteur d'étoile 2 et NP-complet pour hauteur d'étoile au plus 1. Finalement, Wareham [War01] s'est penché sur la complexité paramétrée d'une variante du problème, où Σ^c est considéré plutôt que Σ^*. Différentes paramétrisations de c, k et de la taille des automates permettent de situer le problème au niveau des classes FPT, NP, W[1], W[2] et W[t] (voir [FG06] pour plus de détails sur ces classes, elles ne seront pas définies plus tard contrairement aux autres). Plus de résultats sur le problème d'intersection d'automates sont couverts dans le survol de la littérature de Holzer sur la complexité descriptionnelle et computationnelle des automates finis [HK09].

1.1 Contributions et organisation du mémoire

Dans ce mémoire, nous étudions la complexité du problème d'intersection d'automates sous plusieurs types de restrictions. En restreignant le nombre d'états finaux à 3 et moins, et en introduisant des clauses $\cup$, nous obtenons une classification intéressante de la complexité de IntAuto. Notre étude des automates possédant

un ou deux états finaux, complémente les résultats de NP-complétudes obtenus par [Bea88b] qui nécessitent tous trois états finaux. Notre classification mène ainsi à l'obtention des premières variantes (explicites) de IntAuto résolubles efficacement. Il faut toutefois noter que certains résultats déjà connus, tels que la complexité de « pointset transporter », peuvent être reformulés dans le contexte du problème d'intersection d'automates. Ainsi, l'une de nos contributions est d'avoir établi ou enrichi des connexions entre IntAuto et certains problèmes algébriques. Dans le cas des groupes abéliens, nous revisitions l'équivalence entre le problème d'appartenance et LCON (faisabilité de congruences linéaires avec petits moduli) [McK84, MC87], qui ont récemment suscité l'attention dans le contexte des classes de comptage logarithmique [VV04, Vij08, AV10].

Notre étude est divisée en cinq chapitres. Le chapitre 2 présente les notions et la notation nécessaires à la compréhension de ce mémoire. Nous y définissons également les problèmes pertinents à notre étude.

Au chapitre 3, nous étudions le problème d'intersection d'automates pour lequel le nombre d'automates n'est borné d'aucune façon. Nous donnons d'abord des relations entre IntAuto et différents problèmes algébriques. Nous classifions ensuite sa complexité pour les langages unaires, les automates à groupe (abélien), les langages commutatifs, les langages idempotents, et les langages finis.

Au chapitre 4, nous étudions brièvement $\mathsf{IntAuto}^k$, le problème d'intersection d'un nombre constant k d'automates. Nous donnons sa complexité pour les automates étudiés au chapitre précédent, puis nous abordons les résultats de Karakostas, Lipton et Viglas [KLV03] concernant l'existence de meilleurs algorithmes pour résoudre $\mathsf{IntAuto}^k$.

Nous concluons au chapitre 5 en faisant un retour sur ce travail et en évoquant certaines questions ouvertes.

CHAPITRE 2

PRÉLIMINAIRES

2.1 Théorie des automates

Nous introduisons les concepts et les définitions de la théorie des automates nécessaires à notre étude. Pour plus de détails, le lecteur peut consulter [Sip06, Pin84]. Notre notation est fortement basée sur ces deux ouvrages.

Définition 2.1.1. Soit Σ un ensemble fini, nous dénotons Σ^* l'ensemble de toutes les expressions pouvant être formées à partir de Σ sous la concaténation. Nous dénotons ε l'unique élément de Σ^* tel que $\varepsilon w = w\varepsilon = w$ pour tout $w \in \Sigma^*$.

Définition 2.1.2. Soient Σ un ensemble fini et $S \subseteq \Sigma^*$. Nous disons que S est un *langage* sur *alphabet* Σ, que les éléments de S sont des *mots* et que $\sigma \in \Sigma$ est une *lettre*. Le *mot vide* est ε. Pour $w \in \Sigma^*$, nous définissons $|w|$ comme étant le nombre de lettres constituant w et $|w|_\sigma$ comme étant le nombre d'occurrences de $\sigma \in \Sigma$ dans w.

Exemple 2.1.3. Soient $w, w' \in \{a, b, c\}^*$ tels que $w = abccb$ et $w' = bba$, alors $|w| = 5, |w|_a = 1, |w|_b = 2, |w|_c = 2, |ww'| = 8$ et $|ww'|_b = 4$.

Définition 2.1.4. Un *automate* est un automate fini déterministe. Plus formellement, un automate A est un quintuplet $(\Omega, \Sigma, \delta, \alpha, F)$ où

- Ω est un ensemble fini (*états*),
- Σ est un ensemble fini (*alphabet*),
- $\delta : \Omega \times \Sigma \to \Omega$ (*fonction de transition)*,
- $\alpha \in \Omega$ (*état initial*),
- $F \subseteq \Omega$ (*états finaux ou états acceptants*).

Il est possible (et pratique) de visualiser un automate sous forme de graphe orienté où Ω est l'ensemble des sommets et l'arc (γ, γ') étiqueté par σ est présent ssi $\delta(\gamma, \sigma) = \gamma'$. Par exemple, la Figure 2.1 illustre l'automate

$$(\{\alpha, \beta, \gamma, \omega\}, \{0, 1\}, \delta, \alpha, \{\beta\})$$

où

$$\begin{aligned} \delta(\alpha, 0) &= \beta, \qquad \delta(\alpha, 1) = \omega, \\ \delta(\beta, 0) &= \beta, \qquad \delta(\beta, 1) = \gamma, \\ \delta(\gamma, 0) &= \gamma, \qquad \delta(\gamma, 1) = \beta, \\ \delta(\omega, 0) &= \omega, \qquad \delta(\omega, 1) = \omega. \end{aligned}$$

Un automate représente également un monoïde (de transformations). Nous présentons d'abord la définition d'un monoïde.

Définition 2.1.5. Un *monoïde* est une structure algébrique composée d'un ensemble E et d'un opérateur $\cdot$ respectant les propriétés suivantes :

- Fermeture : $x, y \in E \implies x \cdot y \in E$,
- Associativité : $x, y, z \in E \implies (x \cdot y) \cdot z = x \cdot (y \cdot z)$,
- Existence d'un élément neutre : $\exists e \in E$ tel que $x \in E \implies e \cdot x = x \cdot e = x$.

Nous sommes maintenant en mesure d'associer un monoïde à chaque automate. Pour chaque automate, nous posons $T_\sigma(\gamma) = \delta(\gamma, \sigma)$ et $T_\varepsilon(\gamma) = \gamma$ pour tout état $\gamma \in \Omega$ et pour toute lettre $\sigma \in \Sigma$. Pour $w \in \Sigma^*$, nous définissons T_w par la composition des transformations associées aux lettres de w, plus formellement $T_w = T_{w_{|w|}} \circ \cdots \circ T_{w_1}$. Ainsi, $T_w(\gamma)$ est l'état atteint en lisant w à partir de l'état γ.

Définition 2.1.6. Le *monoïde de transition* $\mathcal{M}(A)$ d'un automate A est le monoïde engendré par composition de l'ensemble de transformations $\{T_\sigma : \sigma \in \Sigma\} \cup \{T_\varepsilon\}$. Autrement dit, $\mathcal{M}(A)$ est l'ensemble $\{T_w : w \in \Sigma^*\}$ muni de la composition.

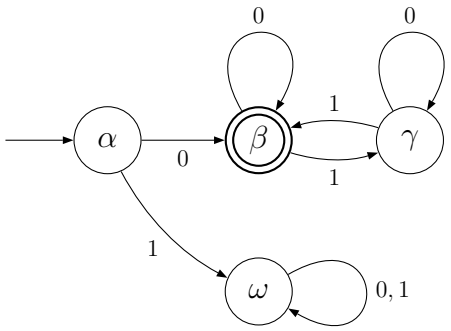
Notons que le *monoïde syntactique* d'un automate A, traité dans la littérature, est (à isomorphisme près) le monoïde de transition de l'automate minimal associé à A.

Définition 2.1.7. Le *langage* accepté par un automate $A = (\Omega, \Sigma, \delta, \alpha, F)$ est l'ensemble

$$\text{Langage}(A) = \{w \in \Sigma^* \,:\, T_w(\alpha) \in F\}.$$

Intuitivement, un mot fait partie du langage accepté par un automate ssi sa lecture à partir de l'état initial atteint un état final. Par exemple, l'automate illustré à la Figure 2.1 accepte (exactement) les mots sur alphabet $\{0, 1\}$ débutant par 0 et dont le nombre d'occurrences de 1 est pair.

Figure 2.1 – Exemple d'automate



2.2 Théorie des groupes

Parfois il peut s'avérer que le monoïde de transition d'un automate soit en fait un groupe ; une structure moins générale qu'un monoïde. Dans les chapitres suivants, nous étudierons de tels automates. Nous introduisons donc les connaissances de la théorie des groupes nécessaires à la compréhension de ce travail. Un lecteur possédant un bagage élémentaire en théorie des groupes peut se limiter aux deux dernières définitions concernant les automates. Pour plus de détails sur la théorie des groupes et l'algèbre, le lecteur peut consulter [DF04, Wie64].

Définition 2.2.1. Un *groupe* est une structure algébrique composée d'un ensemble E et d'un opérateur $\cdot$ respectant les propriétés suivantes :

- Fermeture : $x, y \in E \implies x \cdot y \in E$,
- Associativité : $x, y, z \in E \implies (x \cdot y) \cdot z = x \cdot (y \cdot z)$,
- Existence d'un élément neutre : $\exists e \in E$ tel que $x \in E \implies e \cdot x = x \cdot e = x$,
- Existence d'inverses : $x \in E \implies \exists x^{-1} \in E$ tel que $x \cdot x^{-1} = x^{-1} \cdot x = e$.

Autrement dit, un groupe est un monoïde pour lequel chaque élément g possède un (unique) inverse dénoté g^{-1}.

À partir d'ici, nous utilisons l'abus de langage $g \in G$ pour $g \in E$ où $G = (E, \cdot)$. De plus, nous dénotons l'identité de G par id_G ou tout simplement id lorsque cela ne cause aucune ambigüité.

Nous donnons maintenant quelques définitions et propositions élémentaires concernant les groupes.

Définition 2.2.2. Soit G un groupe et H un sous-ensemble de G alors H est un *sous-groupe* de G, dénoté $H \leq G$, ssi H est lui-même un groupe lorsque muni de l'opération induite par G.

Définition 2.2.3. Soit Ω un ensemble fini. Le groupe S_Ω est l'ensemble des bijections de Ω muni de la composition. Nous disons que $\gamma \in \Omega$ est un *point* et nous dénotons l'image de $g \in S_\Omega$ sur γ par γ^g. Un sous-groupe de S_Ω est nommé *groupe de permutations*.

Définition 2.2.4. Soit G un groupe de permutations sur Ω. Le *stabilisateur* de $\Gamma \subseteq \Omega$ est le sous-groupe G_Γ défini par

$$G_\Gamma = \{g \in G \,:\, \gamma^g = \gamma \text{ pour tout } \gamma \in \Gamma\}.$$

Autrement dit, le stabilisateur est l'ensemble des éléments qui fixent Γ. Lorsque $\Gamma = \{\alpha\}$, nous écrivons tout simplement G_α.

Définition 2.2.5. Soit G un groupe fini. L'ordre d'un élément $g \in G$, dénoté $\mathrm{ord}(g)$, est le plus petit entier positif i tel que $g^i = \mathsf{id}$.

Définition 2.2.6. Soit G un groupe (de permutations sur Ω) et p un nombre premier, alors G est un...

- ...groupe abélien ssi $gh = hg$ pour tout $g, h \in G$,
- ...p-groupe élémentaire abélien ssi $\mathrm{ord}(g) \in \{1, p\}$ et G est abélien,
- ...groupe transitif ssi $\forall \alpha, \beta \in \Omega\ \exists g \in G$ tel que $\alpha^g = \beta$.

Proposition 2.2.7. *Soient G un groupe, $g \in G$ et $i, q \in \mathbb{N}$. Si $g^i = \mathsf{id}$, alors i est divisible par $\mathrm{ord}(g)$. Si $\mathrm{ord}(g)$ divise q alors $g^i = g^{i \bmod q}$. De plus, $\mathrm{ord}(g^i)$ divise $\mathrm{ord}(g)$ pour tout $i \in \mathbb{N}$.*

Proposition 2.2.8 ([Dix67, McK84])**.** *Soit $G < S_\Omega$ un groupe abélien transitif. Soient $g, h \in G$ et $\alpha \in \Omega$, alors $g = h$ ssi $\alpha^g = \alpha^h$. De plus, $|G| = |\Omega|$.*

Tel que mentionné plus tôt, nous nous intéresserons au cas où le monoïde de transition d'un automate est un groupe, et ainsi un groupe de permutations sur Ω. Il est donc pratique de donner un nom à de tels automates.

Définition 2.2.9. Un automate A, dont le monoïde de transition $\mathcal{M}(A)$ est un groupe, est dit être un automate à groupe. De plus, si $\mathcal{M}(A)$ est un groupe possédant certaines propriétés, nous nommons A de façon similaire, par exemple si $\mathcal{M}(A)$ est un groupe abélien transitif, nous disons que A est un automate à groupe abélien transitif.

Définition 2.2.10. Soit A un automate à groupe et A' l'automate obtenu en ne préservant que les états de A accessibles à partir de l'état initial. Nous définissons l'*ordre* de σ par $\mathrm{ord}(\sigma) = \mathrm{ord}(T_\sigma)$ pour $T_\sigma \in \mathcal{M}(A')$.

Ce choix de définir l'ordre d'une lettre en fonction de A' est motivé par le fait que $\mathcal{M}(A')$ est un groupe transitif et que les états inaccessibles sont inutiles dans l'étude du problème d'intersection d'automates. Ainsi, par la Proposition 2.2.8, l'ordre d'une lettre est toujours bornée par le nombre d'états.

2.3 Théorie de la complexité du calcul

Ce travail est consacré à la complexité du problème d'intersection d'automates. Nous introduisons donc les notions de la théorie de la complexité du calcul nécessaires. Les modèles de calculs utilisés sont les machines de Turing et les circuits booléens. Les classes de complexité traitées sont définies selon ces modèles en restreignant le temps, la mémoire et le « parallélisme ». Pour plus de détails, nous référons le lecteur à [Sip06, Vol99].

Définition 2.3.1. Une *machine de Turing* (non déterministe) est un septuplet $(Q, \Sigma, \Gamma, \delta, q_{\text{init}}, q_{\text{acc}}, q_{\text{rejet}})$ où

- Q est un ensemble fini (*états*),
- Σ est un ensemble fini tel que $\sqcup \notin \Sigma$ (*alphabet d'entrée*),
- Γ est un ensemble fini tel que $\sqcup \in \Gamma$ et $\Sigma \subseteq \Gamma$ (*alphabet de travail*),
- $\delta : Q \times \Gamma \to \mathcal{P}(Q \times \Gamma \times \{\leftarrow, \rightarrow\})$ (*fonction de transition)*,
- $q_{\text{init}} \in Q$ (*état initial*),
- $q_{\text{acc}} \in Q$ (*état acceptant*),
- $q_{\text{rejet}} \in Q$ tel que $q_{\text{rejet}} \neq q_{\text{acc}}$ (*état rejetant*).

Une machine de Turing fonctionne de la façon suivante. Elle débute dans l'état initial et reçoit un mot $w \in \Sigma^*$ sur les $|w|$ premières cases de son ruban de travail. Toutes les autres cases du ruban sont initialisées par le symbole $\sqcup$. Elle lit ensuite la première lettre sur son ruban et effectue l'une des transitions définies par δ. À chaque transition, la machine modifie son état, écrit une lettre et déplace sa tête de lecture (à gauche pour $\leftarrow$ et à droite pour $\rightarrow$). Puisque plusieurs transitions sont possibles dans ce modèle non déterministe, la machine explore toutes les transitions possibles. Le calcul se termine lorsque la machine entre dans l'état acceptant ou refusant. S'il existe une suite de transitions menant à l'état acceptant, alors w est accepté par la machine.

Définition 2.3.2. Une machine de Turing est *déterministe* ssi $|\delta(q, s)| = 1$ pour tout $q \in Q$ et $s \in \Gamma$.

Nous pouvons maintenant définir certaines classes de complexité. Ces classes sont des ensembles de langages définis par les ressources nécessaires (ex : temps, mémoire) à une machine de Turing les accepter.

Définition 2.3.3. La classe NTIME($f(n)$) est l'ensemble des langages acceptés par une machine de Turing non déterministe dont le nombre maximal $g(n)$ de transitions, sur n'importe quelle branche de calcul, est tel que $g(n) \in O(f(n))$. La classe TIME($f(n)$) est définie similairement pour les machines déterministes.

Définition 2.3.4. $\mathrm{P} = \bigcup_{k \geq 0} \mathrm{TIME}(n^k)$ et $\mathrm{NP} = \bigcup_{k \geq 0} \mathrm{NTIME}(n^k)$.

Nous définissons maintenant les classes de complexité bornant l'espace (mémoire) nécessaire à la résolution de problèmes. Pour y arriver, il suffit de calquer la définition précédente en bornant le nombre de cases lues sur le ruban de travail. Or, pour les fonctions sous linéaires, logarithmique par exemple, nous devons modifier légèrement cette définition. En effet, tout problème le moindrement intéressant nécessite la lecture complète de l'entrée et ainsi un espace linéaire, selon cette définition. Nous considérons donc plutôt des machines qui possèdent un ruban d'entrée en lecture seule et un ruban de travail. Seules les cases lues sur le ruban de travail ont une incidence sur l'espace utilisé.

Définition 2.3.5. La classe NSPACE($f(n)$) est l'ensemble des langages acceptés par une machine de Turing non déterministe dont le nombre maximal $g(n)$ de cases lues sur le ruban de travail, sur n'importe quelle branche de calcul, est tel que $g(n) \in O(f(n))$. La classe SPACE($f(n)$) est définie similairement pour les machines déterministes.

Définition 2.3.6. $\mathrm{L} = \mathrm{SPACE}(\log n), \mathrm{NL} = \mathrm{NSPACE}(\log n)$

Définition 2.3.7. $\mathrm{PSPACE} = \bigcup_{k \geq 0} \mathrm{SPACE}(n^k)$ et $\mathrm{NPSPACE} = \bigcup_{k \geq 0} \mathrm{NSPACE}(n^k)$.

Définition 2.3.8. La classe $\mathrm{Mod}_k\mathrm{L}$ est l'ensemble des langages acceptés par une machine de Turing non déterministe fonctionnant en espace logarithmique et dont le nombre de chemins acceptants est divisible par k si et seulement si le mot en entrée ne fait pas partie du langage. Nous dénotons $\mathrm{Mod}_2\mathrm{L}$ par $\oplus\mathrm{L}$.

Définition 2.3.9. Un *circuit booléen* est un un graphe acyclique dirigé fini. Il possède n sommets (*portes)* d'entrée d'entrance 0 et étiquetés par $x_1, \ldots, x_n$. Les sommets d'entrance 1 sont étiquetés par $\neg$ et tous les autres sommets par $\vee$ ou $\wedge$. Un circuit possède m sommets de sorties, chacun accessible à partir d'au moins un sommet d'entrée. Un circuit calcule naturellement une fonction $f : \{0,1\}^n \to \{0,1\}^m$ en assignant les n bits à $x_1, \ldots, x_n$ puis en évaluant les différents opérateurs logiques.

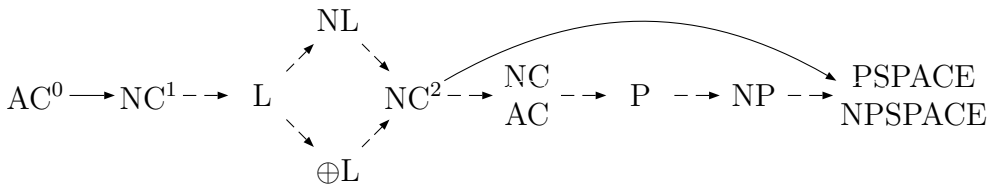
Définition 2.3.10. Une fonction $f : \{0,1\}^* \to \{0,1\}^*$ est calculée par une famille de circuits de $\{a_n\}_{n\geq 1}$ ssi a_n calcule f restreinte au domaine $\{0,1\}^n$ pour tout $n \geq 1$. Une famille de circuits est log-uniforme si une machine de Turing peut calculer le $n^{\text{ème}}$ circuit en espace logarithmique, étant donné 1^n.

Définition 2.3.11. La *taille d'un circuit booléen* est son nombre de portes. La *profondeur d'un circuit* est la longueur du plus long chemin d'une porte d'entrée à une porte de sortie.

Définition 2.3.12. La classe AC^k est l'ensemble des fonctions $f : \{0,1\}^* \to \{0,1\}^*$ calculables par une famille de circuits log-uniforme telle que a_n est de taille polynomiale et de profondeur $O(\log^k n)$. La classe NC^k est définie de façon similaire pour les circuits dont tous les sommets sont d'entrance au plus 2. Nous définissons également les classes $\mathrm{AC} = \bigcup_{k\geq 0} \mathrm{AC}^k$ et $\mathrm{NC} = \bigcup_{k\geq 0} \mathrm{NC}^k$.

La Figure 2.2 illustre les différentes inclusions connues entre les classes de complexité définies. Seules les inclusions $\mathrm{AC}^0 \subset \mathrm{NC}^1$ [FSS84] et $\mathrm{NC}^2 \subset \mathrm{PSPACE}$ sont connues être strictes.

Figure 2.2 – Chaîne d'inclusions entre classes de complexité



Définition 2.3.13. Soient deux langages A et B. Nous disons que A *se réduit à* B, dénoté $A \leq^m B$ s'il existe une fonction $f : \Sigma^* \to \Sigma^*$ calculable par une machine de Turing telle que $w \in A \Leftrightarrow f(w) \in B$ pour tout $w \in \Sigma^*$.

Définition 2.3.14. Un *oracle* pour un langage B est une instruction permettant à une machine de Turing de vérifier si un mot appartient à B en temps constant.

Définition 2.3.15. Soient deux langages A et B. Nous disons que A *se réduit à* B *sous réduction Turing*, dénoté $A \leq^T B$, s'il existe une machine de Turing, avec accès à un oracle pour B, qui accepte A.

Lorsqu'une contrainte est imposée aux ressources utilisées pour calculer une réduction, nous ajoutons un indice à la réduction. Par exemple, si A se réduit à B en espace logarithmique, alors cela est dénoté par $A \leq^m_{\log} B$.

Définition 2.3.16. Soient deux langages A et B. Nous disons que A *se réduit à* B *sous réduction Turing* NC^1 , dénoté $A \leq^T_{\mathrm{NC}^1} B$, s'il existe une réduction de A vers B calculable par une famille log-uniforme de circuits NC^1 possédant des portes qui sont des oracles pour B. Ces portes spéciales sont de taille 1 et de profondeur $\log(1 + \text{nombre d'entrées de la porte})$.

Définition 2.3.17. Nous disons que deux langages A et B sont *équivalents* si $A \leq B$ et $B \leq A$ sous une certaine réduction $\leq$. Nous utilisons une notation identique à celle des réductions.

Pour les différents problèmes algébriques spécifiés par un ensemble de générateurs d'un monoïde de transformations, nous utilisons n'importe quel encodage raisonnable permettant de composer deux transformations et d'évaluer une transformation sur un point en AC^0. Les automates sont encodés par leur monoïde de transitions. Nous disons qu'un entier m est *petit* ssi sa valeur absolue $|m|$ est plus petite que la longueur de l'entrée n. Nous disons qu'un entier est donné par la liste de ses *petits facteurs* ssi il est donné par sa factorisation et que chacun des facteurs est un petit entier. Nous encodons les entiers en base binaire, à l'exception des petits entiers qui sont encodés en base unaire.

2.4 Liste des problèmes

Nous définissons les problèmes mentionnés dans ce travail pour faciliter la lecture. L'ensemble X représente ici une famile de monoïdes finis tels que tous les monoïdes commutatifs, tous les groupes abéliens ou bien tous les groupes. Dans ce travail, X sera toujours une pseudovariété (à l'exception des problèmes concernant les langages finis), c'est-à-dire une famille de monoïdes finis fermée par produit direct fini et par division (image homomorphe d'un sous-monoïde). Le lecteur peut consulter [BMT92] pour obtenir un argument que ces familles constituent un choix riche et naturel et [Pin84] pour en connaître davantage au sujet des pseudovariétés de langages.

$\mathsf{IntAuto}_b^k(X)$ (Problème d'intersection d'automates)

Données : $A_1, \ldots, A_k$ des automates, sur alphabet Σ, tels que $\mathcal{M}(A_i) \in X$ et A_i possède au plus b états finaux pour tout $i \in [k]$.

Question : $\exists w \in \Sigma^*$ tel que pour tout $i \in [k]$, w est accepté par A_i ?

$\mathsf{PS}_b(X)$ (Problème de dispersion de points)

Données : $m > 0$, $g_1, \ldots, g_k : [m] \to [m]$ tels que $\langle g_1, \ldots, g_k \rangle \in X$, et $S_1, \ldots, S_m \subseteq [m]$, tels que $|S_i| \leq b$ ou $|S_i| = m$ pour tout $i \in [m]$.

Question : $\exists g \in \langle g_1, \ldots, g_k \rangle$ tel que pour tout $i \in [m]$, $i^g \in S_i$?

$\mathsf{IntAuto}_b^k(\cup^{k'} X)$ (Problème d'intersection d'automates généralisé)

Données : $(A[1,1], \ldots, A[1,k']), \ldots, (A[k,1], \ldots, A[k,k'])$ des automates, sur alphabet Σ, tels que $\mathcal{M}(A[i,j]) \in X$ et $A[i,j]$ possède au plus b états finaux pour tout $i \in [k], j \in [k']$.

Question : $\exists w \in \Sigma^*$ tel que $w \in \bigcap_{i=1}^{k} \bigcup_{i=1}^{k'} \text{Langage}(A[i,j])$?

Lorsque le nombre d'automates k n'est pas borné, la notation $\mathsf{IntAuto}_b$ est utilisée. De façon similaire, lorsque le nombre d'états finaux b n'est pas borné, nous utilisons la notation IntAuto et PS.

$\mathsf{Memb}(X)$ (Problème d'appartenance)

Données : $m > 0$, $g_1, \ldots, g_k : [m] \to [m]$ tels que $\langle g_1, \ldots, g_k \rangle \in X$, et $g : [m] \to [m]$.

Question : $g \in \langle g_1, \ldots, g_k \rangle$?

$\mathsf{PT}(X)$ (« Pointset transporter »)

Données : $m > 0$, $g_1, \ldots, g_k : [m] \to [m]$ tels que $\langle g_1, \ldots, g_k \rangle \in X$, et $b_1, \ldots, b_r \in [m]$ pour un certain $r \leq m$.

Question : $\exists g \in \langle g_1, \ldots, g_k \rangle$ tel que pour tout $i \in [r]$, $i^g = b_i$?

$\mathsf{ST}(X)$ (« Set transporter »)

Données : $m > 0$, $g_1, \ldots, g_k : [m] \to [m]$ tels que $\langle g_1, \ldots, g_k \rangle \in X$, $r \leq m$ et $B \subseteq [m]$.

Question : $\exists g \in \langle g_1, \ldots, g_k \rangle$ tel que $\{1^g, 2^g, \ldots, r^g\} \subseteq B$?

LCON (Faisabilité de congruences linéaires avec petits moduli)

Données : $B \in \mathbb{Z}^{k\times l}, b \in \mathbb{Z}^k$, et un entier q donné par la liste de ses petits facteurs $p_1^{e_1}, \ldots, p_r^{e_r}$.

Question : $\exists x \in \mathbb{Z}^l$ satisfaisant $Bx \equiv b \pmod q$?

LCONNULL (« Noyau » de congruences linéaires avec petits moduli)

Données : $B \in \mathbb{Z}^{k\times l}$, et un entier q donné par la liste de ses petits facteurs $p_1^{e_1}, \ldots, p_r^{e_r}$.

Problème : calculer un ensemble générateur pour le $\mathbb{Z}$-module $\{x \in \mathbb{Z}^l \,:\, Bx \equiv 0 \pmod q\}$.

Le choix d'encoder q par ses petits facteurs est justifié par la possibilité d'effectuer plusieurs opérations modulaires dans la classe NC^1 [MC87, McK84]. Lorsque q est fixé dans les deux problèmes précédents, nous utilisons la notation LCON_q et $\mathsf{LCONNULL}_q$.

2-SAT (2-satisfaisabilité)

Données : une expression logique en forme normale 2-conjonctive $(l_1 \vee l_1') \wedge \cdots \wedge (l_k \vee l_k')$ où $l_i, l_i' \in \{x_1, \neg x_1 \ldots, x_m, \neg x_m\}$.

Question : $\exists x \in \{0,1\}^m$ satisfaisant l'expression logique ?

2-⊕SAT (2-⊕-satisfaisabilité)

Données : une expression logique en forme normale 2-⊕-conjonctive $(l_1 \oplus l_1') \wedge \cdots \wedge (l_k \oplus l_k')$ où $l_i, l_i' \in \{x_1, \neg x_1 \ldots, x_m, \neg x_m\}$.

Question : $\exists x \in \{0,1\}^m$ satisfaisant l'expression logique ?

Monotone 1-in-3 3-SAT

Données : une expression logique en forme normale 3-conjonctive sans négation : $(x_{r_1} \vee x_{s_1} \vee x_{t_1}) \wedge \cdots \wedge (x_{r_k} \vee x_{s_k} \vee x_{t_k})$.

Question : $\exists x \in \{0,1\}^m$ satisfaisant l'expression logique ?

Factoring

Données : $z \in \mathbb{N}$.
Problème : calculer $x, y \in \mathbb{N}$ tels que $xy = z$.

Subset-Sum
Données : $a_1, \ldots, a_m, b \in \mathbb{Z}$.
Question : $\exists x \in \{0,1\}^m$ tel que $a_1x_1 + \ldots + a_mx_m = b$?

Change-Making
Données : $a_1, \ldots, a_m, b \in \mathbb{Z}$.
Question : $\exists x \in \mathbb{N}^m$ tel que $a_1x_1 + \ldots + a_mx_m = b$?

Fait 2.4.1. *Nous répertorions la complexité de certains des problèmes présentés :*

1. LCON $\in \mathrm{NC}^3$ *[MC87]*
2. LCONNULL $\in \mathrm{NC}^3$ *[MC87]*
3. 2-SAT *est* NL-complet *[Sip06]*
4. 2-$\oplus$SAT *est* L-complet *[JLL76, LP82]*
5. Monotone 1-in-3 3-SAT *est* NP-complet *[GJ79]*
6. Subset-Sum *est* NP-complet *[GJ79]*
7. Change-Making *est* NP-complet *[MT90, Lue75]*

Il est montré dans [JLL76, LP82] que 2-$\oplus$SAT est SL-complet sous réductibilité log-espace. Or, SL = L par [Rei05] et ainsi formulée, cette complétude est inintéressante. Toutefois, il est montré dans [CM87] que le problème d'accessibilité dans une forêt UFA est L-complet sous réductibilité NC^1. Ce problème se réduit trivialement au problème d'accessibilité dans un graphe non dirigé UGAP. De plus, la réduction de UGAP vers 2-$\oplus$SAT donnée dans [JLL76] est calculable en NC^1 bien que cela ne soit pas spécifié. Ainsi, 2-$\oplus$SAT est L-complet sous réductibilité NC^1.

CHAPITRE 3

PROBLÈME D'INTERSECTION D'AUTOMATES NON BORNÉ

Nous débutons maintenant notre étude de la complexité du problème d'intersection d'automates dans le cas où le nombre d'automates n'est borné d'aucune façon. Nous rappelons la définition du problème :

$\mathsf{IntAuto}_b(X)$ (Problème d'intersection d'automates)

Données : $A_1, \ldots, A_k$ des automates, sur alphabet Σ, tels que $\mathcal{M}(A_i) \in X$ et A_i possède au plus b états finaux pour tout $i \in [k]$.

Question : $\exists w \in \Sigma^*$ tel que pour tout $i \in [k]$, w est accepté par A_i ?

Lorsque le nombre d'automates est borné par une fonction $g(n)$, ce problème est NSPACE($g(n) \log n$)-complet [LR92]. Pour résoudre le problème, il suffit de choisir un mot de façon non déterministe et de simuler chacun des $g(n)$ automates sur ce mot. Puisque le plus petit mot accepté par n automates peut être de taille exponentielle, cet algorithme nécessite un temps exponentiel de façon générale. Ainsi, $\mathsf{IntAuto}_b(X)$ est PSPACE-complet.

Notre étude complémente ces résultats. En effet, en restreignant le nombre d'états finaux par automate à au plus 2, nous exhibons des instances du problème d'intersection d'automates à la fois intéressantes et résolubles efficacement.

Dans ce chapitre, nous montrons d'abord brièvement les relations qui existent entre $\mathsf{IntAuto}$ et certains problèmes algébriques. Nous classifions ensuite la complexité de $\mathsf{IntAuto}$ pour les langages unaires, les automates à groupe (abélien), les langages commutatifs et idempotents et les langages finis.

3.1 Relations avec différents problèmes algébriques

Nous proposons le *problème de dispersion de points* PS comme étant le problème algébrique qui capture le problème d'intersection d'automates. Son introduction permet de comprendre comment IntAuto généralise certains problèmes algébriques. De plus, il est parfois plus pratique de travailler directement avec PS puisqu'un seul monoïde est impliqué. En effet, il est alors possible de s'inspirer d'un large éventail de résultats pour des problèmes algrébiques étudiés dans la littérature.

Proposition 3.1.1. $\mathsf{IntAuto}_b(X) \equiv^m_{\mathrm{AC}^0} \mathsf{PS}_b(X)$ *pour n'importe quelle variété de monoïdes finis* X.

Démonstration. $\mathsf{IntAuto}_b(X) \leq \mathsf{PS}_b(X)$: Supposons, sans perte de généralité, que Ω_i et Ω_j sont disjoints pour tout $i \neq j$. Posons $\Omega = \Omega_1 \cup \cdots \cup \Omega_k$. Pour chaque lettre $\sigma \in \Sigma$, posons T_σ la transformation induite par σ sur Ω. Pour tout $\gamma \in \Omega$, posons

$$S_\gamma = \begin{cases} F_i & \text{si } \gamma \text{ est l'état initial de } A_i, \\ \Omega & \text{si } \gamma \text{ est un autre état de } A_i. \end{cases}$$

Soit α_i l'état initial de A_i, alors il existe un mot w accepté par tous les automates ssi $T_w(\alpha_i) \in F_i$ pour tout $i \in [k]$ ssi T_w envoie chaque état initial à un état final. Pour compléter la réduction, il demeure à observer que $|S_\gamma|$ est égal à $|\Omega|$ ou borné par b. De plus, $\langle \{T_\sigma : \sigma \in \Sigma\} \rangle \in X$ puisqu'il est un sous-monoïde de $\mathcal{M}(A_1) \times \cdots \times \mathcal{M}(A_k)$.

$\mathsf{PS}_b(X) \leq \mathsf{IntAuto}_b(X)$: Pour tout $i \in [m]$ tel que $S_i \neq [m]$, posons

$$A_i = ([m], \{\sigma_1, \ldots, \sigma_k\}, \delta, i, S_i)$$

où $\delta : [m] \times \{g_1, \ldots, g_k\} \to [m]$ est défini par $\delta(j, \sigma_\ell) = j^{g_\ell}$ pour tout $j \in [m]$, $\ell \in [k]$. Lorsque $S_i = [m]$, nous ne construisons aucun automate puisque celui-ci accepterait le langage Σ^*. Soit $g = g_{\ell_1} \cdots g_{\ell_r}$, alors $i^g \in S_i$ pour tout $i \in [m]$ ssi $w = \sigma_{\ell_1} \cdots \sigma_{\ell_r}$ est accepté par tous les automates. De plus, chaque automate possède au plus b états finaux, et $\mathcal{M}(A_i) \in X$ car $\langle g_1, \ldots, g_k \rangle \in X$. ∎

La proposition suivante montre les relations entre PS (et donc IntAuto) et les problèmes Memb, PT et ST. Nous invitons le lecteur à consulter la section 2.4 du chapitre 2 pour obtenir une définition de ces problèmes.

Proposition 3.1.2. $\mathsf{Memb}(X) \leq^m_{\mathrm{AC}^0} \mathsf{PT}(X) \equiv^m_{\mathrm{AC}^0} \mathsf{PS}_1(X)$ *et* $\mathsf{ST}(X) \leq^m_{\mathrm{AC}^0} \mathsf{PS}(X)$.

Démonstration. Nous invitons le lecteur à consulter la définition des différents problèmes à la section 2.4. Nous utilisons les mêmes générateurs pour chaque réduction. Pour $\mathsf{Memb}(X) \leq \mathsf{PT}(X)$, nous posons $b_i = i^g$ pour tout $i \in [m]$ où g est la transformation test. Pour $\mathsf{PT}(X) \leq \mathsf{PS}_1(X)$, nous posons $S_i = \{b_i\}$ pour tout $i \in [r]$ et $S_i = [m]$ pour le reste. Pour $\mathsf{PS}_1(X) \leq \mathsf{PT}(X)$, si $|S_i| = 1$, nous posons b_i comme étant l'unique élément de S_i. Pour être consistant avec la définition de PT, les points sont réordonnés de façon à ce que les points transportés soient les r premiers. Finalement, pour $\mathsf{ST}(X) \leq \mathsf{PS}(X)$, nous posons $S_i = B$ pour tout $i \in [r]$, et $S_i = [m]$ pour tout i tel que $r < i \leq m$. ■

Proposition 3.1.3. *Si* $\mathsf{Memb}(X) \in$ NP *(resp.* PSPACE*) alors* $\mathsf{PS}(X) \in$ NP *(resp.* PSPACE*).*

Démonstration. Nous construisons une machine de Turing qui choisit, de façon non déterministe, une transformation g telle que $i^g \in S_i$ pour tout $i \in [m]$. Elle exécute ensuite la machine NP (PSPACE) pour $\mathsf{Memb}(X)$ afin de vérifier si $g \in \langle g_1, \ldots, g_k \rangle$. Dans le cas de PSPACE, nous utilisons l'égalité PSPACE = NPSPACE [Sav70]. ■

Corollaire 3.1.4. *Si* $\mathsf{Memb}(X)$ *est* NP-complet *(resp.* PSPACE-complet*) alors* $\mathsf{PS}(X)$ *est* NP-complet *(resp.* PSPACE-complet*).*

3.2 Langages sur alphabet unaire

Nous débutons notre étude approfondie du problème d'intersection d'automates en considérant les automates dont l'alphabet est unaire, autrement dit, lorsque $|\Sigma| = 1$. Bien que nous étudierons plus tard les langages commutatifs, qui généralisent ces langages, il est naturel et pratique d'aborder les langages unaires en premier lieu.

À première vue, les langages unaires peuvent sembler assez limités, mais c'est justement leur simplicité qui les rend intéressants. Tout d'abord, ces langages représentent naturellement des ensembles d'entiers puisqu'un mot formé d'une seule lettre est complètement caractérisé par sa longueur. De plus, la littérature regorge d'exemples où l'utilisation d'automates unaires conduit à des résultats intéressants. Par exemple, Geffert et Pighizzini [GP10] ont récemment montré que les classes L et NL pourraient être séparées en prouvant qu'il est impossible de convertir un automate unaire non déterministe à tête bidirectionelle en une version déterministe polynomialement plus grande. Cette formulation simplifie grandement les méthodes, pour séparer L et NL, de Berman et Lingas [BL77], et Sipser [Sip79] basées sur des automates arbitraires. Finalement, puisque $\mathsf{IntAuto}_b$ est PSPACE-complet pour tout $b > 0$, nul ne peut espérer obtenir une complexité à l'intérieur de P pour $\mathsf{IntAuto}_b(X)$ si $\mathsf{IntAuto}_b(\text{unaires})$ n'est pas résoluble efficacement. Toutes ces raisons indiquent que les langages unaires constituent le meilleur point de départ pour notre étude.

Nous notons d'abord qu'il suffit de s'intéresser aux automates unaires à groupe. Nous remarquons qu'un automate unaire est constitué d'une *queue* et d'un *cycle*, tel qu'illustré à la Figure 3.1. D'autre part, un automate unaire à groupe n'est formé que d'un cycle, tel qu'illustré à la Figure 3.2. La présence d'une queue ne rend pas IntAuto plus complexe. Cela est dû au fait qu'une queue ne permet de reconnaître qu'un ensemble fini de mots dont la taille est bornée par le nombre d'états. Nous le démontrons dans le lemme suivant que nous énonçons de la façon la plus générale possible :

Lemme 3.2.1. $\mathsf{IntAuto}_b^k(\cup^{k'}\ \textit{unaire}) \equiv_{\log}^m \mathsf{IntAuto}_b^k(\cup^{k'}\ \textit{groupes et unaire})$.

Démonstration. Nous montrons que $\mathsf{IntAuto}_b^k(\cup^{k'}\ \text{unaire}) \leq_{\log}^m \mathsf{IntAuto}_b^k(\cup^{k'}\ \text{groupes et unaire})$. La réciproque est directe.

Considérons un automate d'une clause $\bigcup$. Chacun des états finaux de sa queue accepte exactement un mot. Soit un tel mot, nous vérifions si celui-ci est accepté par un automate de chacune des clauses $\bigcup$. Si tel est le cas, alors l'intersection est non

vide. Autrement, l'état final associé à ce mot est retiré. Cette procédure est répétée pour tous les états finaux de la queue, puis pour tous les automates. La taille de chacun des mots à tester est bornée par n et il y donc au plus $bn \in O(n^2)$ tests. De plus, chaque test peut s'effectuer sur un seul automate à la fois, en avançant dans sa queue, cela ne nécessite donc qu'une quantité logarithmique de mémoire.

Soit $A[i,j]$ le $j^{\text{ème}}$ automate de la $i^{\text{ème}}$ clause $\bigcup$ après les retraits d'états finaux de toutes les queues de tous les automates. Soit $\lambda[i,j]$ la taille du cycle de $A[i,j]$ et $d[i,j,r]$ la position de son $r^{\text{ème}}$ état final. Chaque automate $A[i,j]$ accepte ainsi tous les mots $w \in \{a\}^*$ tels que

$$|w| \equiv d[i,j,r] \pmod{\lambda[i,j]} \text{ et } |w| \geq d[i,j,r]$$

pour un certain $1 \leq r \leq b$.

Nous construisons une deuxième instance du problème contenant les automates cycliques $A'[i,j]$ de taille $\lambda[i,j]$ dont les états finaux sont en position $d'[i,j,r] = d[i,j,r] \bmod \lambda[i,j]$. L'automate $A[i,j]$ accepte tous les mots $w \in \{a\}^*$ tels que

$$|w| \equiv d'[i,j,r] \pmod{\lambda[i,j]}$$

pour un certain $1 \leq r \leq b$. Par exemple, l'automate illustré à la Figure 3.1 devient celui de la Figure 3.2.

Nous remarquons qu'un mot accepté par l'intersection des automates $A[i,j]$ est forcément accepté par l'intersection des nouveaux automates. De façon opposée, supposons qu'il existe $w \in \bigcap_{i=1}^{k} \bigcup_{j=1}^{k'} \text{Langage}(A'[i,j])$, alors il existe $x \in [k']^k, r \in [b]^k$ et $y \in \mathbb{N}^k$ tels que

$$|w| = d'[i,x_i,r_i] + y_i\lambda[i,x_i] \quad \forall i \in [k].$$

Posons $l = \max(d[1,x_1,r_1),\ldots,d[k,x_k,r_k]) \cdot \text{ppcm}(\lambda[1,x_1],\ldots,\lambda[k,x_k])$, alors $l \geq d[i,x_i,r_i]$ pour tout $i \in [k]$ et ainsi, par définition, $wa^l \in \text{Langage}(A[i,x_i])$ pour tout $i \in [k]$. ■

Figure 3.1 – Exemple d'automate unaire

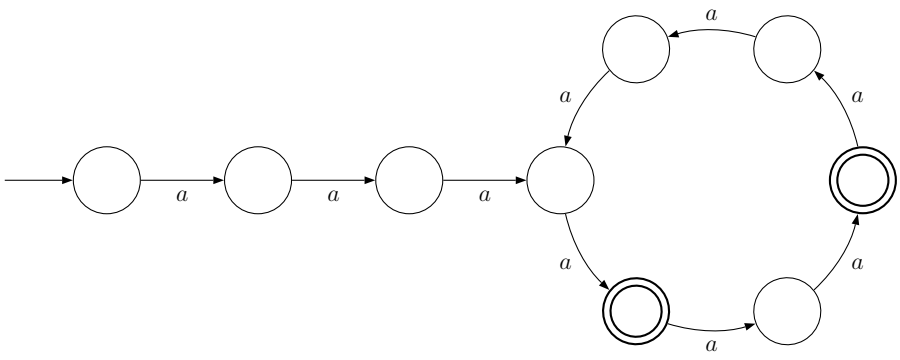
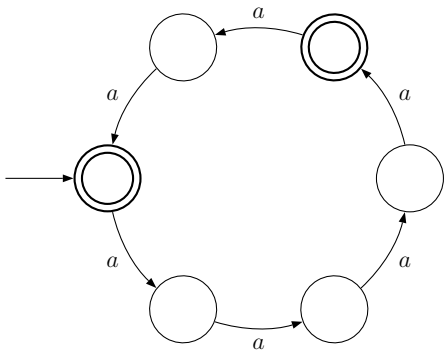


Figure 3.2 – Automate unaire à groupe obtenu après conversion



Le lemme suivant est une version généralisée du théorème des restes chinois. Celui-ci joue un rôle clef dans les résultats qui suivent.

Lemme 3.2.2 ([Knu81, p. 277 ex. 3])**.** *Soient* $a_1, \ldots, a_k \in \mathbb{N}$ *et* $q_1, \ldots, q_k \in \mathbb{N}^+$*. Il existe* $x \in \mathbb{N}$ *tel que* $x \equiv a_i \pmod{q_i}$ *pour tout* $i \in [k]$ *ssi* $a_i \equiv a_j \pmod{\text{pgcd}(q_i, q_j)}$ *pour tout* $i, j \in [k]$*.*

Nous pouvons maintenant donner le premier nouveau résultat. Nous verrons, plus tard, qu'en limitant chacun des automates à deux états finaux ou moins, il est possible de caractériser L. Nous généralisons d'abord ce résultat en remplaçant les automates à deux états finaux par l'union de deux automates ne possédant qu'un seul état final chacun. De cette façon, nous obtenons une instance NL-complète de

IntAuto. Pour ce faire, nous montrons en deux étapes que le problème est équivalent à 2-SAT.

Théorème 3.2.3. $\mathsf{IntAuto}_1(\cup^2$ *groupes et unaire*$) \leq^m_{\log}$ 2-SAT.

Démonstration. Soit $A[i,0]$ et $A[i,1]$ les automates de la $i^{\text{ème}}$ clause $\bigcup$. Nous remarquons que $A[i,x]$ accepte tous les mots $w \in \{a\}^*$ tels que

$$|w| \equiv d[i,x] \pmod{\mathrm{ord}_{i,x}(a)}$$

où $d[i,x]$ est la position de l'état final. Nous avons donc les équivalences suivantes :

$$\begin{aligned}
& \exists w \text{ tel que } w \in \bigcap_{i=1}^{k} \bigcup_{x=0}^{1} \text{Langage}(A[i,x]) \\
\Leftrightarrow \quad & \exists w \, \exists x \in \{0,1\}^k \text{ tels que } w \in \bigcap_{i=1}^{k} A[i,x_i] \\
\Leftrightarrow \quad & \exists w \, \exists x \in \{0,1\}^k \text{ tels que } \bigwedge_{i=1}^{k} |w| \equiv d[i,x_i] \pmod{\mathrm{ord}_{i,x_i}(a)} \\
\Leftrightarrow \quad & \exists x \in \{0,1\}^k \text{ tel que } \bigwedge_{i=1}^{k} \bigwedge_{j=1}^{k} C_{i,j}(x)
\end{aligned}$$

où $C_{i,j}(x) = \big(d[i,x_i] \equiv d[j,x_j] \pmod{\mathrm{pgcd}(\mathrm{ord}_{i,x_i}(a), \mathrm{ord}_{j,x_j}(a))}\big)$. Cette dernière équivalence est une conséquence du lemme 3.2.2. Nous obtenons donc finalement l'équivalence

$$\exists w \in \bigcap_{i=1}^{k} \bigcup_{x=0}^{1} \text{Langage}(A[i,x]) \quad \Leftrightarrow \quad \exists x \in \{0,1\}^k \text{ tel que } \bigwedge_{i=1}^{k} \bigwedge_{j=1}^{k} C_{i,j}(x).$$

Pour tout $i,j \in [k]$, la table de vérité de $C_{i,j}$ peut être construite en calculant les quatre congruences impliquées. Puisque $C_{i,j}$ ne dépend que de deux variables (c.-à-d. x_i et x_j), il est toujours possible de construire une expression en 2-FNC à partir de la table obtenue. Nous concluons en remarquant que les congruences peuvent être évaluées en log-espace puisque les nombres sont petits. ■

Théorème 3.2.4. 2-SAT $\leq^m_{\mathrm{NC}^1}$ $\mathsf{IntAuto}_1(\cup^2$ *groupes et unaire*).

Démonstration. Soit $C(x)$ l'expression booléenne $\bigwedge_{i=1}^{k} C_i(x)$ où

$$C_i(x) = (x_{r_i} \oplus b_i) \vee (x_{s_i} \oplus b'_i)$$

et $b_i, b'_i \in \{0, 1\}$ indiquent s'il faut prendre ou non la négation de la variable associée.

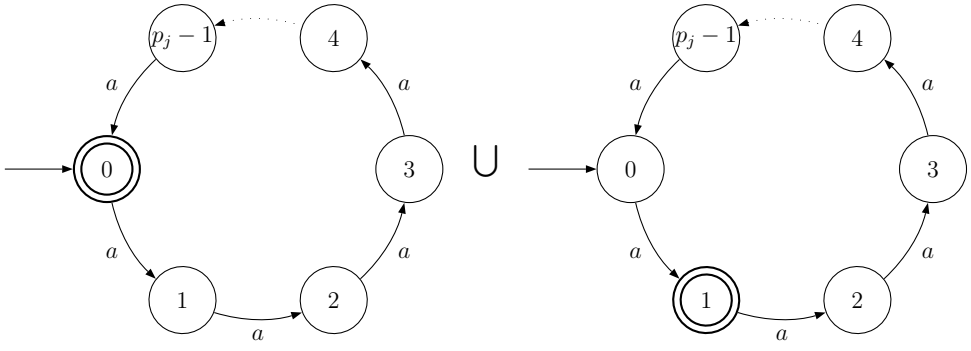
Il est possible de représenter une affectation à C par un entier en s'assurant que celui-ci est congru à 0 ou 1 modulo les m premiers nombres premiers $p_1, \ldots, p_m$. Le résidu d'un tel entier modulo p_i représente ainsi la valeur assignée à la $i^{\text{ème}}$ variable. Dans le contexte des langages sur alphabet unaire, on considère la longueur d'un mot pour obtenir une telle représentation.

Posons

$$\begin{aligned} E_j &= \{w \in \{a\}^* : |w| \equiv 0 \pmod{p_j} \vee |w| \equiv 1 \pmod{p_j}\}, \\ E &= \bigcap_{j=1}^{m} E_j. \end{aligned}$$

Le langage E représente ainsi toutes les affectations valides. Notons que E_j peut être reconnu par l'union de deux automates ne possédant qu'un seul état final, tel qu'illustré à la Figure 3.3.

Figure 3.3 – Automates acceptant le langage E_j



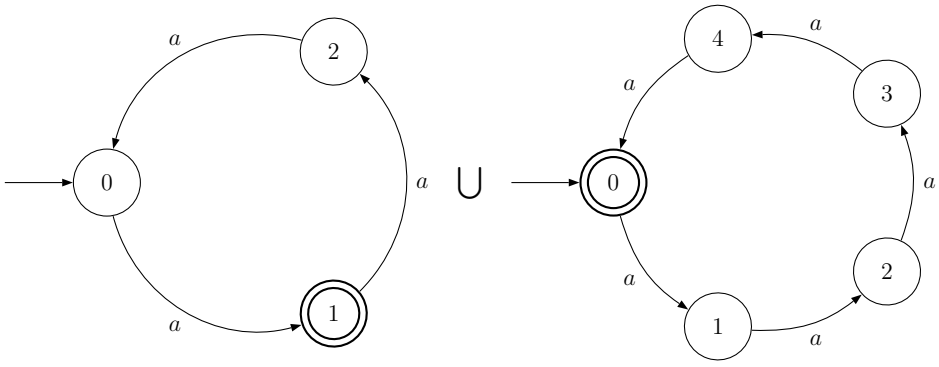
Posons

$$X_i = \{w \in \{a\}^* : |w| \equiv \neg b_i \pmod{p_{r_i}} \vee |w| \equiv \neg b'_i \pmod{p_{s_i}}\},$$

$$X = \bigcap_{i=1}^{k} X_i.$$

Intuitivement, X_i représente toutes les affectations qui satisfont C_i. Nous remarquons cependant que certaines de ces « affectations » peuvent être invalides, puisque l'un des résidus modulo p_{r_i} et p_{s_i} peut différer de 0 ou 1. En considérant $E \cap X$, nous obtenons ainsi toutes les affectations qui satisfont l'expression booléenne C. Notons que le langage X_i peut être reconnu par l'union de deux automates cycliques possédant respectivement p_{r_i} et p_{s_i} états dont un seul est final. Par exemple, si $p_{r_i} = 3, p_{s_i} = 5, b_i = 0$ et $b'_i = 1$, alors X_i est reconnu par l'union des automates illustrés à la Figure 3.4

Figure 3.4 – Automates acceptant le langage X_i pour $p_{r_i} = 3, p_{s_i} = 5, b_i = 0$ et $b'_i = 1$



Montrons formellement que C est satisfaisable ssi $E \cap X \neq \emptyset$.

$\Rightarrow$) Soit $x \in \{0,1\}^m$ tel que $C(x) = 1$. Puisque x satisfait C_i, nous avons

$$(x_{r_i}, x_{s_i}) \in \{(b_i, \neg b'_i), (\neg b_i, b'_i), (\neg b_i, \neg b'_i)\} \quad \forall i \in [k].$$

Par le théorème des restes chinois, il existe exactement un entier $0 \leq l < p_1 \cdots p_m$ tel que $l \bmod p_i = x_i$ pour tout $i \in [m]$. Ainsi, par définition de E et X, nous avons $a^l \in E \cap X$.

$\Leftarrow$) Soit $w \in E \cap X$. Par définition de E, il existe $x \in \{0,1\}^m$ tel que $|w| \bmod p_i = x_i$ pour tout $i \in [m]$. Par définition de X, nous avons $x_{r_i} = \neg b_i$ ou $x_{s_i} = \neg b'_i$ pour tout $i \in [k]$. Ainsi, $C_i(x) = 1$ pour tout $i \in [k]$ et du coup C est satisfaisable. ■

Corollaire 3.2.5. $\mathsf{IntAuto}_1(\cup^2$ *groupes et unaire*$)$ *est* NL-complet.

Tel que mentionné précédemment, il est possible d'obtenir une complexité plus basse que NL lorsque nous remplaçons chaque clause $\bigcup$ par un seul automate à deux états finaux ou moins. Dans la preuve du théorème 3.2.3, nous avons exploité la possibilité de vérifier localement si l'intersection entre deux automates est non vide. Nous cherchions $x \in \{0,1\}^k$ tel que $\bigcap_{i=1}^{k} A[i, x_i] \neq \emptyset$. Dans le cas des automates à deux états finaux, x_i représente plutôt le choix de l'état final si un ordre leur est attribué. Il s'agit donc de chercher $x \in \{0,1\}^k$ tel que

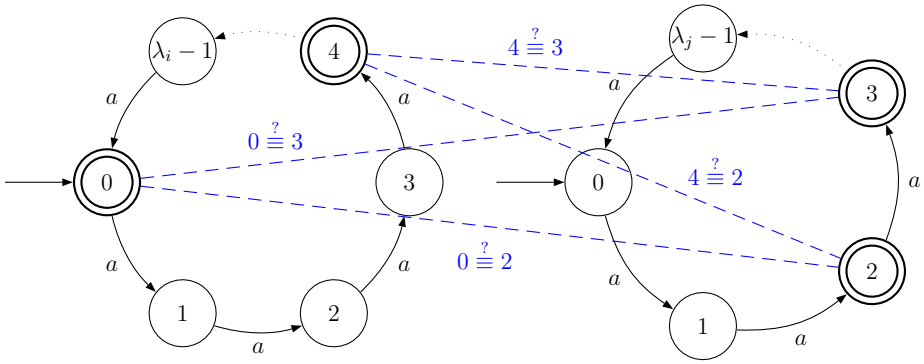
$$d[i, x_i] \equiv d[j, x_j] \pmod{\operatorname{pgcd}(\operatorname{ord}_i(a), \operatorname{ord}_j(a))}$$

où $d[i, x_i]$ est la position du $x_i{}^{\text{ème}}$ état final du $i^{\text{ème}}$ automate. Pour chaque paire d'automates, il y a donc quatre congruences à considérer tel que l'illustre l'exemple de la Figure 3.5.

Il est crucial de remarquer que si trois des congruences sont vraies, alors elles le sont toutes. Graphiquement, à la Figure 3.5, si l'on ne trace que les lignes hachurées dont la congruence associée est vérifiée, alors il ne peut y en avoir que 0, 1, 2 ou 4 au total. Cette simple observation permet de diminuer la complexité du problème de NL à L. Nous démontrons ce fait dans le lemme suivant :

Lemme 3.2.6. *Soient* $a_0, a_1, b_0, b_1 \in \mathbb{N}$ *et* $q \in \mathbb{N}^+$. *Si trois des congruences sui-*

Figure 3.5 – Exemple de congruences à tester entre deux automates



vantes sont vraies :

$$a_0 \equiv b_0 \pmod{q},$$
$$a_0 \equiv b_1 \pmod{q},$$
$$a_1 \equiv b_0 \pmod{q},$$
$$a_1 \equiv b_1 \pmod{q}.$$

alors les quatre le sont.

Démonstration. Supposons que trois congruences soient vraies. Considérons la relation d'équivalence $R(x, y) = (x \equiv y \pmod{q})$. Par hypothèse, le graphe associé à R restreinte aux sommets $\{a_0, a_1, b_0, b_1\}$ possède au moins trois arêtes. Ce graphe est donc connexe et ainsi toutes les congruences sont vraies puisque R est une relation d'équivalence. ■

Nous montrons que $\mathsf{IntAuto}_1$(Groupes et unaire) et $\mathsf{IntAuto}_2$(Groupes et unaire) sont L-complets. Pour y arriver, nous montrons que les deux problèmes sont équivalents à 2-⊕SAT. Nous rappelons que 2-⊕SAT est L-complet et qu'il est défini de façon similaire à 2-SAT, mais avec des opérateurs $\oplus$ plutôt que $\vee$.

Théorème 3.2.7. $\mathsf{IntAuto}_2$(*Groupes et unaire*) $\leq^m_{\log}$ 2-⊕SAT.

Démonstration. Nous remarquons d'abord qu'un automate à deux états finaux peut être remplacé par l'union de deux copies du même automate, possédant chacune un

seul état final. Nous pouvons donc utiliser la preuve du Théorème 3.2.3. Cependant, nous devons montrer qu'il est possible de produire une expression en forme 2-⊕FNC (plutôt que 2-FNC).

Le premier pas dans cette direction consiste à remarquer que, dans ce cas, les automates $A[i,0]$ et $A[i,1]$ possèdent un cycle de même taille. Nous utilisons donc la notation $\text{ord}_i(a)$ pour représenter $\text{ord}_{i,0}(a)$ et $\text{ord}_{i,1}(a)$. De façon similaire à la preuve du Théorème 3.2.3, nous obtenons les équivalences suivantes :

$$\begin{aligned} & \exists w \text{ tel que } w \in \bigcap_{i=1}^{k} \bigcup_{j=0}^{1} \text{Langage}(A[i,j]) \\ \Leftrightarrow \quad & \exists w \, \exists x \in \{0,1\}^k \text{ tels que } \bigwedge_{i=1}^{k} |w| \equiv d[i,x_i] \pmod{\text{ord}_i(a)} \\ \Leftrightarrow \quad & \exists x \in \{0,1\}^k \text{ tel que } \bigwedge_{i=1}^{k} \bigwedge_{j=1}^{k} C_{i,j}(x). \end{aligned}$$

où $C_{i,j}(x) = (d[i,x_i] \equiv d[j,x_j] \pmod{\text{pgcd}(\text{ord}_i(a), \text{ord}_j(a))})$. La table de vérité de $C_{i,j}$ peut être construite en calculant les quatre congruences impliquées. Par le lemme 3.2.6, si trois des congruences sont vraies, alors la quatrième l'est aussi. Nous pouvons donc écrire $C_{i,j}$ de l'une des façons identifiées au Tableau 3.I. Nous

Tableau 3.I – Expressions possibles pour $C_{i,j}$

Congruences vraies	**Expressions**
0	0
1	$(x_i \wedge x_j), (\neg x_i \wedge x_j), (x_i \wedge \neg x_j), (\neg x_i \wedge \neg x_j)$
2	$x_i, \neg x_i, x_j, \neg x_j, (x_i \oplus x_j), (\neg x_i \oplus x_j)$
4	1

remarquons donc que $C_{i,j}$ peut s'écrire en 2-⊕FNC. ■

Au chapitre suivant, nous verrons que $\mathsf{IntAuto}_1^k$(Unaire) est L-complet. Ainsi, nous pouvons conclure que $\mathsf{IntAuto}_1$(Groupes et unaire) et $\mathsf{IntAuto}_2$(Groupes et unaire) sont L-ardus et du coup L-complets par le théorème précédent. Il est ce-

pendant pertinent de montrer comment 2-⊕SAT se réduit directement au problème $\mathsf{IntAuto}_2$(Groupes et unaire) ; ce que nous faisons dans ce résultat :

Théorème 3.2.8. 2-⊕SAT $\leq^m_{\mathrm{NC}^1}$ $\mathsf{IntAuto}_2$(*Groupes et unaire*).

Démonstration. Soit $C(x)$ l'expression booléenne $\bigwedge_{i=1}^{k} C_i(x)$ où

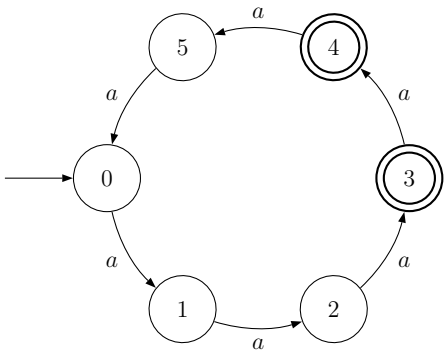
$$C_i(x) = (x_{r_i} \oplus x_{s_i} \oplus b_i)$$

et $b_i \in \{0, 1\}$. Nous associons l'ensemble X_i suivant à la clause C_i :

$$X_i = \begin{cases} \{w \in \{a\}^* \,:\, |w| \bmod (p_{r_i}, p_{s_i}) \in \{(0,1),(1,0)\}\} & \text{si } b_i = 0, \\ \{w \in \{a\}^* \,:\, |w| \bmod (p_{r_i}, p_{s_i}) \in \{(0,0),(1,1)\}\} & \text{si } b_i = 1. \end{cases}$$

Ainsi C est satisfaisable ssi $\bigcap_{i=1}^{k} X_i \neq \emptyset$.

Figure 3.6 – Automates acceptant le langage X_i pour $p_{r_i} = 2, p_{s_i} = 3$ et $b_i = 0$



Il suffit donc de montrer que X_i est reconnu par un automate unaire possédant deux états finaux. Par le théorème des restes chinois, il existe exactement quatre

entiers $0 \leq l_1, l_2, l_3, l_4 < p_{r_i} p_{s_i}$ tels que

$$\begin{aligned} l_1 \bmod (p_{r_i}, p_{s_i}) &= (0,1), \\ l_2 \bmod (p_{r_i}, p_{s_i}) &= (1,0), \\ l_3 \bmod (p_{r_i}, p_{s_i}) &= (0,0), \\ l_4 \bmod (p_{r_i}, p_{s_i}) &= (1,1). \end{aligned}$$

Nous pouvons donc construire un automate cyclique de $p_{r_i} p_{s_i}$ états avec ses deux états finaux en positions l_1 et l_2 ou l_3 et l_4 selon le cas. Par exemple, supposons que $p_{r_i} = 2, p_{s_i} = 3$ et $b_i = 0$, alors X_i est reconnu par l'automate illustré à la Figure 3.6. ■

Il découle des Théorèmes 3.2.7 et 3.2.8 :

Corollaire 3.2.9. $\mathsf{IntAuto}_2$(*Groupes et unaire*) *est* L-complet.

Tel que noté dans [HK09], le résultat suivant peut être obtenu à partir de [SM73] avec quelques modifications.

Proposition 3.2.10 ([SM73])**.** IntAuto(*Unaire*) *est* NP-complet.

Nous remarquons donc que, contrairement au cas binaire, IntAuto(Unaire) est complet pour la classe NP plutôt que PSPACE. Ce résultat peut cependant être amélioré. Nous donnons d'abord une preuve alternative de l'appartenence du problème à NP. Celle-ci inclut la notion de clauses $\bigcup$, est plus simple et est basée sur l'un des résultats précédents.

Proposition 3.2.11. IntAuto($\cup$ *groupes et unaire*) $\in$ NP.

Démonstration. Nous construisons une machine de Turing qui choisit, de façon non déterministe, un automate par clause $\bigcup$, puis un état final pour chacun de ceux-ci. La machine vérifie ensuite si l'intersection des automates sous-jacents est non vide. Par le Théorème 3.2.7, cela peut se faire en temps polynomial. ■

Corollaire 3.2.12. IntAuto($\cup$ *groupes et unaire*) *est* NP-complet.

Nous montrons maintenant que la complétude tient même lorsque les automates sont restreints à posséder trois états finaux et moins.

Théorème 3.2.13. Monotone 1-in-3 3-SAT $\leq^{m}_{\mathrm{NC}^1}$ $\mathsf{IntAuto}_3$(*Groupes et unaire*).

Démonstration. Nous donnons une réduction similaire à celle du Théorème 3.2.8 mais en utilisant l'ensemble

$$X_i = \{w \in \{a\}^* : |w| \bmod (p_{r_i}, p_{s_i}, p_{t_i}) \in \{(1,0,0),(0,1,0),(0,0,1)\}\}.$$

Par le théorème des restes chinois, il est possible de construire un automate possédant $p_{r_i}p_{s_i}p_{t_i}$ états et trois états finaux. ■

Corollaire 3.2.14. $\mathsf{IntAuto}_3$(*Groupes et unaire*) *est* NP-complet.

Le Tableau 3.II dresse un sommaire de notre classification de la complexité de $\mathsf{IntAuto}_b(\bigcup^{k'}$ unaire).

Tableau 3.II – Complexité de $\mathsf{IntAuto}_b(\bigcup^{k'}$ unaire).

		Nombre d'états finaux		
		1	2	3+
Clauses $\bigcup$	1	L-complet	L-complet	NP-complet
	2	NL-complet	NP-complet	NP-complet
	3+	NP-complet	NP-complet	NP-complet

3.3 Automates à groupes

Comme nous avons pu le remarquer dans la section précédente, la complexité du problème d'intersection d'automates diminue grandement lorsque l'alphabet des automates est restreint à une seule lettre. Un questionnement naturel est donc de se demander s'il existe des langages plus complexes pour lesquels le problème d'intersection demeure résoluble efficacement. Dans ce chapitre, nous répondons à cette question. Pour y arriver, nous considérons les langages acceptés par les automates à

groupe et, plus particulièrement, par les automates à groupe abélien. Ces langages constituent, bien sûr, une généralisation des automates unaires à groupes. Or, tous les résultats pour les langages unaires obtenus au chapitre précédent proviennent de l'étude des automates unaires à groupe. Nous enrichissons donc les résultats précédents en nous intéressant aux automates à groupe.

Nous notons d'abord que $\mathsf{PS}_1(\text{Groupes}) \in \text{NC}$ grâce à [Luk90, p. 27]. En effet, Luks réduit $\mathsf{PT}(\text{Groupes})$ au problème qui consiste à calculer un ensemble générateur pour le stabilisateur G_Γ d'un groupe de permutations G où $\Gamma \subseteq \Omega$. Ce problème est connu comme étant dans NC [BLS87] et nous avons montré que $\mathsf{PT}(X)$ est équivalent à $\mathsf{PS}_1(X)$ à la Proposition 3.1.2. De plus, puisque $\mathsf{Memb}(\text{Groupes}) \in \text{NC}$ [BLS87], alors $\mathsf{PS}(\text{Groupes})$ est dans NP par la Proposition 3.1.3, et complet pour NP par le Corollaire 3.2.14.

Proposition 3.3.1. $\mathsf{PS}_1(\textit{Groupes}) \in \text{NC}$ *et* $\mathsf{PS}(\textit{Groupes})$ *est* NP-complet.

Avant que le résultat précédent ne soit connu, il avait été démontré dans [LM88] que $\mathsf{PT}(\text{Groupes nilpotents}) \in \text{NC}$, et qu'ainsi $\mathsf{PS}_1(\text{Groupes nilpotents}) \in \text{NC}$ par la Proposition 3.1.2. Cela implique que ces problèmes appartiennent à NC pour les groupes abéliens, qui sont un cas particulier des groupes nilpotents.

Nous consacrons le reste de ce chapitre à l'étude de $\mathsf{IntAuto}$ (et donc PS) pour les groupes abéliens. Nous améliorons d'abord cette borne de NC, pour $\mathsf{PS}_1(\text{Groupes abéliens})$, à NC^3. Ainsi, $\mathsf{PS}_1(\text{Groupes abéliens})$ possède la même complexité que $\mathsf{Memb}(\text{Groupes abéliens})$.

Pour y parvenir, nous traitons directement du problème d'intersection d'automates pour les automates à groupe abélien. Nous donnons d'abord quelques définitions et lemmes nécessaires pour classifier le problème.

Définition 3.3.2. Soit $A = (\Omega, \Sigma, \delta, \alpha, F)$ un automate à groupe abélien. Nous définissons Φ_A comme étant l'ensemble suivant :

$$\Phi_A = \left\{ v \in \mathbb{Z}_q^s \,:\, T_{\sigma_1^{v_1} \cdots \sigma_s^{v_s}} \in G_\alpha \right\},$$

où $q = \text{ppcm}(\text{ord}(\sigma_1), \ldots, \text{ord}(\sigma_s))$ et $\Sigma = \{\sigma_1, \ldots, \sigma_s\}$.

Autrement dit, Φ_A est l'ensemble des vecteurs $(v_1, \ldots, v_s) \in \mathbb{Z}_q^s$ tels que la lecture du mot $\sigma_1^{v_1} \cdots \sigma_s^{v_s}$ à partir de l'état initial α revient à l'état initial α. Puisque le langage accepté par A est commutatif et que l'ordre de chaque lettre divise q, l'ensemble Φ_A permet de caractériser tous les mots acceptés par A.

Lemme 3.3.3. *Soit $A = (\Omega, \Sigma, \delta, \alpha, F)$ un automate à groupe abélien, alors Φ_A est un sous $\mathbb{Z}_q$-module de $\mathbb{Z}_q^s$ où $q = \text{ppcm}(\text{ord}(\sigma_1), \ldots, \text{ord}(\sigma_s))$.*

Démonstration. Notons tout d'abord que $(0, \ldots, 0) \in \Phi_A$ puisque $T_\epsilon(\alpha) = \alpha$. Soient $b \in \mathbb{Z}_q$ et $v, v' \in \Phi_A$. Montrons que $bv + v' \in \Phi_A$. Posons $w = \sigma_1^{v_1} \cdots \sigma_s^{v_s}$ et $w' = \sigma_1^{v'_1} \cdots \sigma_s^{v'_s}$, alors nous avons

$$\begin{aligned} T_{w^b w'}(\alpha) &= T_{w'}(T_{w^b}(\alpha)) \\ &= T_{w'}(\alpha) && (\text{car } T_w \in G_\alpha \text{ et donc } T_{w^b} \in G_\alpha) \\ &= \alpha && (\text{car } T_{w'} \in G_\alpha). \end{aligned}$$

Nous concluons donc que Φ_A est un sous-module. ∎

Définition 3.3.4. Soit $A = (\Omega, \Sigma, \delta, \alpha, F)$ un automate à groupe abélien. Posons $q = \text{ppcm}(\text{ord}(\sigma_1), \ldots, \text{ord}(\sigma_s))$. Nous définissons l'homomorphisme de monoides $\phi_A : \Sigma^* \to \mathbb{Z}_q^s$ de la façon suivante :

$$\phi_A(w) = (|w|_{\sigma_1} \bmod q, \ldots, |w|_{\sigma_s} \bmod q).$$

Cet homomorphisme est semblable à ce qui est connu dans la littérature sous le nom d'image de Parikh. La distinction ici est que chaque composante est prise modulo q pour un certain $q \in \mathbb{N}^+$.

Lemme 3.3.5. *Soient $A = (\Omega, \Sigma, \delta, \alpha, F)$ un automate à groupe abélien, $\beta \in \Omega$, $0 \leq i \leq s$ et $b_1, \ldots, b_i \in \mathbb{N}$. Il est possible de vérifier s'il existe un mot $w \in \Sigma^*$, tel que $T_w(\alpha) = \beta$ et $|w_{\sigma_j}| = b_j$ pour tout $1 \leq j \leq i$, en espace logarithmique. De plus, s'il existe un tel mot, il est possible d'en calculer un en espace logarithmique.*

Démonstration. Nous remarquons d'abord que A peut être considéré comme un graphe non dirigé. En effet, puisque $\mathcal{M}(A)$ est un groupe, l'emprunt d'un arc, étiqueté par σ, en sens inverse équivaut à appliquer T_σ^{-1}. Pour un arc (une transition) de γ vers γ' étiqueté par σ, nous ajoutons donc l'arc (γ', γ) étiqueté σ^{-1}. Puisque $\mathcal{M}(A)$ est abélien, nous pouvons supposer, sans perte de généralité, que les lettres $\sigma_1, \ldots, \sigma_i$ sont lues au tout début. Posons donc $\alpha' = T_{w'}(\alpha)$ où $w' = \sigma_1^{b_1} \cdots \sigma_i^{b_i}$ et retirons toutes les transitions associées à $\sigma_1, \ldots, \sigma_i$. Il suffit maintenant de chercher un chemin de α' vers β dans le graphe obtenu pour construire w tel que $T_w(\alpha) = \beta$. Par [Rei05], il est possible de trouver un tel chemin en espace logarithmique. ■

Lemme 3.3.6. *Soit $A = (\Omega, \Sigma, \delta, \alpha, F)$ un automate à groupe abélien. Il est possible de calculer un ensemble générateur U de Φ_A tel que $|U| \leq \text{ord}(\sigma_1)+\ldots+\text{ord}(\sigma_s)+|\Sigma|$ en espace logarithmique.*

Démonstration. Considérons l'algorithme suivant :

pour $i = 1$ a $|\Sigma|$ faire
  pour $j = 0$ a $\text{ord}(\sigma_i) - 1$ faire
    calculer w tel que $T_w(\alpha) = \alpha$,
      $|w_{\sigma_r}| = 0$ pour tout $1 \leq r < i$, et $|w_{\sigma_i}| = j$
    ecrire $\phi_A(w)$
  ecrire v tel que $v_i = \text{ord}(\sigma_i)$ et $v_r = 0$ pour tout $r \neq i$

Nous remarquons d'abord que l'algorithme produit au plus $\text{ord}(\sigma_1)+\ldots+\text{ord}(\sigma_s)+|\Sigma|$ éléments. De plus, le mot w obtenu à la ligne 4 est calculable en espace logarithmique par le Lemme 3.3.5.

Montrons maintenant que $\langle U \rangle = \Phi_A$. Soit $v \in \Phi_A$. Montrons, par induction sur s, qu'il existe $u_1, \ldots, u_s \in \langle U \rangle$ tels que $u_{i,j} = 0$ pour $1 \leq j < i$, et $u_{i,i} = v_i - \sum_{j=1}^{i-1} u_{j,i}$. Avant de procéder, notons que si cette affirmation est vraie, alors $v = u_1 + \ldots + u_s$ et ainsi $v \in \langle U \rangle$.

Nous remarquons d'abord qu'il existe $x < q$ tel que $v_1 = (v_1 \bmod \mathrm{ord}(\sigma_1)) + x \cdot \mathrm{ord}(\sigma_1)$. Soit $u'_1 \in U$ tel que $u'_{1,1} = v_1 \bmod \mathrm{ord}(\sigma_1)$, alors posons $u_1 = u'_1 + (x \cdot \mathrm{ord}(\sigma_1), 0, \ldots, 0)$. Nous avons donc $u_1 \in \langle U \rangle$ et $u_{1,1} = v_1$. Nous remarquons qu'il existe $v' \in \Phi_A$ tel que $v'_1 = v_1 \bmod \mathrm{ord}(\sigma_1)$, puisque le vecteur obtenu en changeant la première composante de v par $v_1 \bmod \mathrm{ord}(\sigma_1)$ est dans Φ_A. La ligne 4 de l'algorithme va donc nécessairement générer un tel vecteur u'_1.

Supposons que l'hypothèse d'induction est vraie pour $u_1, \ldots, u_{i-1}$. Posons $v' = v - (u_1 + \ldots + u_{i-1})$, alors nous avons $v' \in \Phi_A$. De plus $v'_j = 0$ pour $1 \leq j < i$ et $v'_i = v_i - \sum_{j=1}^{i-1} u_j[i]$. Soit $u'_i \in U$ tel que $u'_{i,j} = 0$ pour $j < i$ et $u'_{i,i} = v'_i \bmod \mathrm{ord}(\sigma_i)$, alors posons $u_i = u'_1 + (0, \ldots, y \cdot \mathrm{ord}(\sigma_i), \ldots, 0)$. Nous avons donc $u_i \in \langle U \rangle$ et $u_{i,i} = v'_i$ pour un certain $y < q$. Pour les mêmes raisons évoquées plus tôt, la ligne 4 de l'algorithme va générer un tel vecteur u'_i. ■

Définition 3.3.7. Soit V un sous-module de $\mathbb{Z}_q^s$, alors

$$V^\perp = \{u \in \mathbb{Z}_q^s \,:\, \forall v \in V \quad v \cdot u = 0\},$$

où $\cdot$ désigne le produit scalaire standard.

Lemme 3.3.8. *Soit V un sous-module de $\mathbb{Z}_q^s$, alors $(V^\perp)^\perp = V$.*

Démonstration. Soient $v \in V$ et $v' \in V^\perp$, alors $v' \cdot v = v \cdot v' = 0$. Ainsi $v \in (V^\perp)^\perp$ et du coup $V \subseteq (V^\perp)^\perp$.

Il suffit maintenant de montrer que $|(V^\perp)^\perp| = |V|$ pour conclure la preuve, puisque V est fini. Pour y arriver, nous faisons appel à la théorie de la représentation des groupes abéliens finis. Soit G un groupe abélien fini et $\widehat{G}$ l'ensemble de tous les caractères de G. Autrement dit, $\widehat{G}$ est le groupe constitué de tous les homomorphismes de G vers le groupe multiplicatif $\mathbb{C}^\times$. Soit $H \leq G$, posons $H^\# = \{\chi \in \widehat{G} \,:\, \chi(H) = 1\}$. Il est connu que $G \cong \widehat{G}$ et $(H^\#)^\# \cong \widehat{H}$ (par exemple, voir [Luo09]).

Posons $\omega_q = e^{(2\pi i)/q}$ et $\chi_u(v) = \omega_q^{u \cdot v}$ pour tout $u, v \in \mathbb{Z}_q^s$. Nous avons $\widehat{\mathbb{Z}_q^s} = \left\{\chi_u \,:\, u \in \mathbb{Z}_q^s\right\}$ et l'isomorphisme entre $\mathbb{Z}_q^s$ et $\widehat{\mathbb{Z}_q^s}$ est $u \mapsto \chi_u$. Soit $v \in \mathbb{Z}_q^s$ et $\chi \in \widehat{\mathbb{Z}_q^s}$,

alors $\chi_u(v) = 1$ ssi $u \cdot v = 1$. Ainsi, nous avons $V^\perp \cong V^\#$ et du coup $(V^\perp)^\perp \cong (V^\#)^\# \cong \widehat{V} \cong V$. Nous obtenons donc $|(V^\perp)^\perp| = |V|$. ■

Lemme 3.3.9. *Soient $x, x' \in \mathbb{N}^s$ et $U = \{u_1, \ldots, u_{|U|}\}$ un ensemble générateur de $\Phi_A^\perp$. Posons $q = \text{ppcm}(\text{ord}(\sigma_1), \ldots, \text{ord}(\sigma_s))$ et B la matrice dont la $i^{ème}$ ligne est u_i, alors*

$$Bx \equiv Bx' \pmod{q} \Leftrightarrow T_w(\alpha) = T_{w'}(\alpha)$$

où $w = \sigma_1^{x_1} \cdots \sigma_s^{x_s}$ et $w' = \sigma_1^{x'_1} \cdots \sigma_s^{x'_s}$.

Démonstration. $\Rightarrow$) Posons $v = \phi_A(w), v' = \phi_A(w')$, alors $B(v - v') \equiv Bv - Bv' \equiv 0 \pmod{q}$ et ainsi $v - v' \in \Phi_A^{\perp\perp}$. Par le Lemme 3.3.8, nous avons donc $v - v' \in \Phi_A$ et du coup $v + \Phi_A = v' + \Phi_A$. Il existe donc $v'' \in \Phi_A$ tel que $v = v' + v''$ et ainsi nous avons

$$\begin{aligned}
T_w(\alpha) &= T_{\sigma_1^{x_1} \cdots \sigma_s^{x_s}}(\alpha) && \text{(Par définition de } w\text{)} \\
&\equiv T_{\sigma_1^{|w|_{\sigma_1} \bmod q} \cdots \sigma_s^{|w|_{\sigma_s} \bmod q}}(\alpha) && (\text{ord}(\sigma_i) \mid q) \\
&= T_{\sigma_1^{v_1} \cdots \sigma_s^{v_s}}(\alpha) && \text{(Par définition de } v\text{)} \\
&= T_{\sigma_1^{v'_1 + v''_1 \bmod q} \cdots \sigma_s^{v'_s + v''_s \bmod q}}(\alpha) && (v = v' + v'') \\
&\equiv T_{\sigma_1^{v'_1 + v''_1} \cdots \sigma_s^{v'_s + v''_s}}(\alpha) && (\text{ord}(\sigma_i) \mid q) \\
&\equiv T_{(\sigma_1^{v'_1} \cdots \sigma_s^{v'_s}) \cdot (\sigma_1^{v''_1} \cdots \sigma_s^{v''_s})}(\alpha) && (\mathcal{M}(A) \text{ est abélien}) \\
&\equiv T_{\sigma_1^{v'_1} \cdots \sigma_s^{v'_s}}(\alpha) && (v'' \in \Phi_A) \\
&\equiv T_{w'}(\alpha) && \text{(Par symétrie aux lignes 1–3).}
\end{aligned}$$

Nous concluons donc que $T_w(\alpha) = T_{w'}(\alpha)$.

$\Leftarrow$) Puisque $T_w(\alpha) = T_{w'}(\alpha)$, alors $T_w {T_{w'}}^{-1} \in G_\alpha$. Soit $u \in \Sigma^*$ tel que $T_u = {T_{w'}}^{-1}$, alors $\phi_A(wu) \in \Phi_A$. Puisque ϕ_A est un homomorphisme, nous avons $\phi_A(w) + \phi_A(u) \in \Phi_A$. Par le Lemme 3.3.8, $\Phi_A = (\Phi_A)^{\perp\perp}$ et ainsi

$$B\phi_A(w) + B\phi_A(u) \equiv B(\phi_A(w) + \phi_A(u)) \equiv 0 \pmod{q}.$$

Nous avons donc

$$B\phi_A(w) \equiv B(-\phi_A(u)) \pmod{q}.$$

Ainsi, nous concluons que $Bx \equiv Bx' \pmod{q}$ car $x \equiv \phi_A(w) \pmod{q}$ et $x' \equiv \phi_A(w') \equiv -\phi_A(u) \pmod{q}$. ■

Nous possédons maintenant tous les outils nécessaires pour classifier le problème d'intersection pour les automates à groupe abélien. Nous débutons par le résultat le plus général, autrement dit, nous classifions IntAuto(Groupes abéliens). Nous montrons, à l'aide de réductions, que la complexité de $\mathsf{IntAuto}_1$(Groupes abéliens) se situe entre celles de LCON et de LCONNULL. Les deux réductions utilisées sont nécessaires à l'obtention des résultats suivants, le lecteur devrait donc leur porter une attention particulière.

Théorème 3.3.10. $\mathsf{IntAuto}_1$(*Groupes abéliens*) $\leq^T_{\mathrm{NC}^1}$ LCONNULL.

Démonstration. Soient $A_1, \ldots, A_k$ les automates en entrée et α_i, β_i respectivement leurs états initiaux et finaux. Nous construisons un système de congruences linéaires pour chaque automate. Nous calculons d'abord un ensemble générateur pour Φ_{A_i}. Par le Lemme 3.3.6, cela est possible en espace logarithmique. Étant donné cet ensemble, nous pouvons obtenir un ensemble générateur U_i de $\Phi^{\perp}_{A_i}$ via l'oracle pour LCONNULL. Soit $w_i \in \Sigma^*$ un mot tel que $T_{w_i}(\alpha_i) = \beta_i$. Par le Lemme 3.3.5, un tel mot se calcule en espace logarithmique. Posons B_i la matrice dont chacune des lignes est un vecteur de U_i, et $b_i = B_i\phi_{A_i}(w_i)$. Par le Lemme 3.3.9, $B_ix \equiv b_i \pmod{q_i}$ ssi $w = \sigma_1^{x_1} \cdots \sigma_s^{x_s}$ est accepté par l'automate A_i où $q_i = \mathrm{ppcm}(\mathrm{ord}(\sigma_1), \ldots, \mathrm{ord}(\sigma_s))$. Ainsi, il existe une solution $x \in \mathbb{Z}^s$, pour tout $i \in [k]$, à

$$B_ix \equiv b_i \pmod{q_i} \qquad (*)$$

si et seulement si un mot w est accepté par tous les automates. Nous réduisons

donc l'instance du problème d'intersection à cette instance de LCON :

$$\begin{pmatrix} B_1 & q_1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ B_k & 0 & \cdots & q_k \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_s \\ y_1 \\ \vdots \\ y_k \end{pmatrix} \equiv \begin{pmatrix} b_1 \\ \vdots \\ b_k \end{pmatrix} \pmod{\text{ppcm}(q_1, \ldots, q_k)}$$

qui est équivalente au système $(*)$. Notons que $\text{ppcm}(q_1, \ldots, q_k)$ peut être un grand nombre, mais que ses facteurs sont petits puisque $q_1, \ldots, q_k$ sont petits. De plus, il est important de noter que LCON se réduit à LCONNULL et que LCONNULL est ardu pour NL [MC87], et ainsi pour L, sous réductions $\leq^T_{\text{NC}^1}$. Il est donc possible de calculer cette réduction avec un circuit NC^1. En effet, les différents calculs effectués en espace logarithmique peuvent être effectués par des oracles pour LCONNULL et la sortie peut être réduite à une instance équivalente de LCONNULL. ■

Grâce au théorème précédent, il est maintenant possible de placer le problème IntAuto_1(Groupes abéliens) dans la classe NC^3. En effet, il a été démontré que $\text{LCONNULL} \in \text{NC}^3$ dans [MC87].

Corollaire 3.3.11. IntAuto_1(*Groupes abéliens*) $\in \text{NC}^3$.

Cela nous permet aussi d'obtenir un second corollaire puisque le problème LCONNULL a récemment été classifié dans la classe $\text{FL}^{\text{ModL}}/\text{poly}$ [AV10].

Corollaire 3.3.12. IntAuto_1(*Groupes abéliens*) $\in \text{FL}^{\text{ModL}}/poly$.

Montrons maintenant que le problème LCON se réduit à IntAuto_1(Groupes abéliens).

Théorème 3.3.13. $\text{LCON} \leq^m_{\text{NC}^1} \text{IntAuto}_1$(*Groupes abéliens*).

Démonstration. Considérons l'instance suivante de LCON : $B \in \mathbb{Z}^{k \times l}$, $b \in \mathbb{Z}^k$ et $q \in \mathbb{N}$ un entier décrit par ses petits facteurs $p_1^{e_1}, \ldots, p_r^{e_r}$. Nous construisons un automate

$A_{i,j}$ pour chacune des lignes de B et chacun des facteurs. Tous les automates ont le même état initial et alphabet :

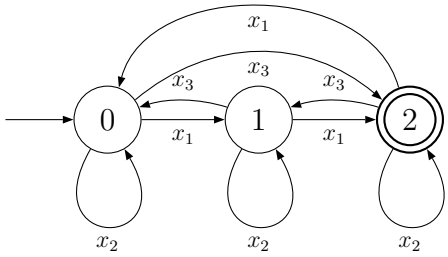
$$\begin{aligned} \alpha &= 0 \\ \Sigma &= \{x_1, \ldots, x_n\}. \end{aligned}$$

Les états, les états finaux et les fonctions de transitions sont définis ainsi :

$$\begin{aligned} Q &= \{0, \ldots, p_j^{e_j} - 1\} \\ F_{i,j} &= \{b_i \bmod p_j^{e_j}\} \\ \delta_{i,j}(\alpha, x_t) &= (\alpha + B_{i,t}) \bmod p_j^{e_j}. \end{aligned}$$

Par exemple, supposons qu'une ligne de la matrice définit l'équation $x_1 + 2x_3 \equiv 2 \pmod{3^1}$, alors nous obtenons l'automate illustré à la Figure 3.7.

Figure 3.7 – Exemple d'automate pour $x_1 + 2x_3 \equiv 2 \pmod{3^1}$



Soit $w \in \Sigma^*$, montrons que

$$\delta_{i,j}(\gamma, w) = (\gamma + B_{i,1}|w|_{x_1} + \ldots + B_{i,s}|w|_{x_s}) \bmod q.$$

Nous avons bien $\delta_{i,j}(\gamma, \epsilon) = \gamma$. Supposons maintenant que l'hypothèse est vraie

pour tout mot de taille l. Soit $w = w'x_t$ tel que $|w'| = l$. Nous avons donc

$$\begin{aligned}
\delta_{i,j}(\gamma, w) &= \delta_{i,j}(\gamma, w'x_t) \\
&= \delta_{i,j}(\delta_{i,j}(\gamma, w'), x_t) \\
&= \delta_{i,j}((\gamma + B_{i,1}|w'|_{x_1} + \ldots + B_{i,l}|w'|_{x_l}) \bmod p_j^{e_j}, x_t) \\
&= ((\gamma + B_{i,1}|w'|_{x_1} + \ldots + B_{i,l}|w'|_{x_l}) \bmod p_j^{e_j} + B_{i,t}) \bmod p_j^{e_j} \\
&= (\gamma + B_{i,1}|w'|_{x_1} + \ldots + B_{i,s}|w'|_{x_l}) + B_{i,t}) \bmod p_j^{e_j} \\
&= (\gamma + B_{i,1}|w|_{x_1} + \ldots + B_{i,s}|w|_{x_l}) \bmod p_j^{e_j}.
\end{aligned}$$

Nous avons donc $\delta_{i,j}(0, w) \equiv B_{i,1}|w|_{x_1} + \ldots + B_{i,l}|w|_{x_l} \pmod{p_j^{e_j}}$ et ainsi $w \in$ Langage$(A_{i,j})$ ssi

$$B_{i,1}|w|_{x_1} + \ldots + B_{i,l}|w|_{x_l} \equiv b_i \pmod{p_j^{e_j}}.$$

Un mot w est donc accepté par $A_{i,j}$ pour tout $1 \leq i \leq k$ ssi $(|w|_{x_1}, \ldots, |w|_{x_l})$ est une solution à $B_i x \equiv b \pmod{p_j^{e_j}}$. Ainsi, un mot est accepté par tous les automates ssi $(|w|_{x_1}, \ldots, |w|_{x_l})$ est une solution de $Bx \equiv b \pmod{q}$, puisque

$$Bx \equiv b \pmod{q} \text{ est résoluble } \Leftrightarrow Bx \equiv b \pmod{p_j^{e_j}} \text{ est résoluble } \forall j \in [r].$$

Le lecteur peut consulter [AV10, Lemme 3.4] pour une preuve formelle de cette dernière équivalence. Pour conclure la preuve, il suffit de remarquer que $\mathcal{M}(A_{i,j})$ est un groupe abélien. ■

Puisque LCON $\in$ NC^3 et que LCON n'est pas connu comme étant dans une plus petite classe, nous obtenons une localisation assez précise de la complexité du problème d'intersection d'automates à groupe abélien.

Nous considérons maintenant IntAuto pour les automates à p-groupe élémentaire abélien. Nous montrons que le problème est Mod$_p$L-complet (dénoté $\oplus$L-complet lorsque $p = 2$). Cela mène donc à une caractérisation de Mod$_p$L en terme d'auto-

mates.

Théorème 3.3.14. $\mathsf{IntAuto}_1$(*p-groupes élémentaires abéliens*) *est* $\mathrm{Mod}_p\mathrm{L}$-complet.

Démonstration. Dans un p-groupe élémentaire abélien, tout élément est d'ordre p (à l'exception de l'identité qui est d'ordre 1), ainsi $\mathrm{ppcm}(\mathrm{ord}_i(\sigma_1), \ldots, \mathrm{ord}_i(\sigma_s)) = p$ (ou 1). Donc, la réduction vers $\mathsf{LCONNULL}$ du Théorème 3.3.10 peut être utilisée pour obtenir une réduction vers $\mathsf{LCONNULL}_p$. De plus, cette réduction peut être convertie en une réduction log-espace, sans modification significative. En effet, chaque calcul effectué dans la preuve peut être accompli en espace logarithmique, et la réduction NC^1 de LCON vers $\mathsf{LCONNULL}$ de [MC87] peut être convertie en une réduction log-espace tel que noté dans [AV10]. Ainsi,

$$\mathsf{IntAuto}_1(p\text{-groupes élémentaires abéliens}) \leq^T_{\log} \mathsf{LCONNULL}_p.$$

Puisque $\mathsf{LCONNULL}_p \in \mathrm{Mod}_p\mathrm{L}$ [BDHM92], et $\mathrm{Mod}_p\mathrm{L} = \mathrm{Mod}_p\mathrm{L}^{\mathrm{Mod}_p\mathrm{L}}$ ($\mathrm{FMod}_p\mathrm{L} = \mathrm{FL}^{\mathrm{Mod}_p\mathrm{L}}$) [HRV00], nous obtenons

$$\mathsf{IntAuto}_1(p\text{-groupes élémentaires abéliens}) \in \mathrm{Mod}_p\mathrm{L}.$$

De façon similaire, la réduction à partir de LCON dans la preuve du Théorème 3.3.13 peut être utilisée pour obtenir une réduction log-espace à partir de LCON_p. Puisque LCON_p est complet pour $\mathrm{Mod}_p\mathrm{L}$ [BDHM92], cela complète la preuve. ■

Nous nous penchons maintenant sur le problème d'intersection d'automates à groupe abélien possédant au plus deux états finaux. Lorsque nous nous restreignons aux $2-$groupes élémentaires abéliens, nous sommes en mesure d'obtenir une réduction vers LCON_2. Ainsi, dans ce cas, le problème $\mathsf{IntAuto}_2$ est équivalent à $\mathsf{IntAuto}_1$.

Théorème 3.3.15. $\mathsf{IntAuto}_2$(2-*groupes élémentaires abéliens*) $\leq^T_{\mathrm{NC}^1} \mathsf{LCON}_2$.

Démonstration. Nous nous inspirons de la preuve du Théorème 3.3.10. Soient α_i l'état initial et β_i, β_i' les deux états finaux de l'automate A_i. Nous utilisons la même notation que dans la preuve du Théorème 3.3.10, U_i est un ensemble de générateurs de $\Phi_{A_i}^{\perp}$; $w_i, w_i' \in \Sigma^*$ sont des mots tels que $\alpha_i^{w_i} = \beta_i$ et $\alpha_i^{w_i'} = \beta_i'$; B_i est la matrice dont chacune des lignes est un vecteur de U_i ; $b_i = B_i\phi_{A_i}(w_i)$, et $b_i' = B_i\phi_{A_i}(w_i')$.

Par le Lemme 3.3.9, il existe donc une solution $x \in \mathbb{Z}^s$, pour tout $i \in [k]$, à

$$(B_i x \equiv b_i \pmod 2) \vee (B_i x \equiv b_i' \pmod 2)$$

si et seulement si un mot est accepté par tous les automates.

Nous construisons ce système dans lequel nous nous débarrassons des clauses $\vee$ en introduisant des variables z_i, z_i' :

$$\begin{pmatrix} 0 & 1 & 1 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & 1 & 1 \\ B_1 & b_1 & b_1' & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ B_k & 0 & 0 & \cdots & b_k & b_k' \end{pmatrix} \begin{pmatrix} x \\ z_1 \\ z_1' \\ \vdots \\ z_k \\ z_k' \end{pmatrix} \equiv \begin{pmatrix} 1 \\ \vdots \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \pmod 2 .$$

Nous remarquons que ce système est équivalent à

$$B_i x + z_i b_i + z_i' b_i' \equiv 0 \pmod 2 \quad \forall i \in [k],$$

avec les contraintes $z_i + z_i' \equiv 1 \pmod 2$ pour tout $i \in [k]$. Puisque $-z_i b_i \equiv z_i b_i \pmod 2$ et $-z_i' b_i' \equiv z_i' b_i' \pmod 2$, ce système est équivalent à

$$B_i x \equiv z_i b_i + z_i' b_i' \pmod 2 \quad \forall i \in [k].$$

Les contraintes $z_i + z_i' \equiv 1 \pmod 2$ ont comme conséquence de forcer la sélection d'ou bien le vecteur b_i d'ou bien le vecteur b_i'. Ce système de congruences est donc

une instance de LCON_2 qui possède une solution ssi un mot est accepté par tous les automates. ■

Nous considérons maintenant une généralisation, plus restreinte, des automates unaires à groupe. Nous nous intéressons au cas où l'ensemble

$$\left\{v \in \mathbb{Z}_{\mathrm{ord}(\sigma_1)} \times \cdots \times \mathbb{Z}_{\mathrm{ord}(\sigma_s)} \,:\, \sigma_1^{v_1} \cdots \sigma_s^{v_s} \in \mathrm{Langage}(A_i)\right\}$$

ne contient qu'un seul élément pour chacun des automates A_i en entrée. Nous nommons un tel automate un *automate à groupe abélien serré*. Nous notons qu'un automate unaire à groupe possède toujours cette propriété. En effet, dans ce cas, l'automate est un cycle dirigé de taille $\mathrm{ord}(a)$ et ainsi il n'existe qu'un seul mot accepté qui soit de taille plus petite que $\mathrm{ord}(a)$.

Une autre façon d'obtenir un automate répondant à ce critère est de construire le produit cartésien de différents automates unaires à groupe, travaillant sur des lettres distinctes. Nous obtenons ainsi des automates ayant la forme « d'hypercubes » constitués de cycles dirigés où les « axes » sont restreints à des lettres distinctes. Nous donnons un exemple d'un tel automate, à la Figure 3.8, qui accepte le langage

$$\left\{w \in \{\sigma_1, \sigma_2\}^* \,:\, |w|_{\sigma_1} \equiv 2 \pmod 3 \wedge |w|_{\sigma_2} \equiv 1 \pmod 2\right\}.$$

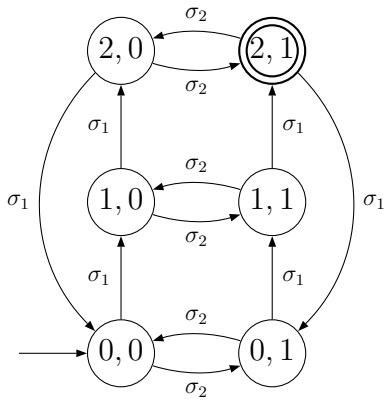
Tous les automates à groupe abélien ne sont pas serrés toutefois, comme l'illustre la Figure 3.7.

Nous remarquons que, dans un automate à groupe abélien serré, les lettres sont indépendantes. Ainsi, pour tester l'appartenance d'un mot à un langage, il suffit de vérifier localement des congruences associées à chacune des lettres. Grâce à cette remarque, nous pouvons maintenant généraliser le Théorème 3.2.3.

Théorème 3.3.16. $\mathsf{IntAuto}_1(\cup^2$ *Automates à groupe abélien serré*$) \leq^m_{\log} \mathsf{2\text{-}SAT}$.

Démonstration. Nous nous inspirons de la preuve du Théorème 3.2.3. Soient $A[i, 0]$

Figure 3.8 – Exemple d'automate à groupe abélien serré



et $A[i,1]$ les deux automates de la $i^{\text{ème}}$ clause $\bigcup$. Soit $v[i,x]$ l'unique élément de

$$V[i,x] = \left\{ v \in \mathbb{Z}_{\mathrm{ord}_{i,x}(\sigma_1)} \times \cdots \times \mathbb{Z}_{\mathrm{ord}_{i,x}(\sigma_s)} \,:\, \sigma_1^{v_1} \cdots \sigma_s^{v_s} \in \text{Langage}(A[i,x]) \right\},$$

calculable en log-espace par le Lemme 3.3.5. Nous notons d'abord que $A[i,x]$ accepte (exactement) les mots $w \in \Sigma^*$ tels que

$$|w|_{\sigma_j} \equiv v[i,x]_j \pmod{\mathrm{ord}_{i,x}(\sigma_j)} \text{ pour tout } 1 \leq j \leq s,$$

par définition de $V[i,x]$. Nous avons donc les équivalences suivantes :

$$\begin{aligned}
& \exists w \text{ tel que } w \in \bigcap_{i=1}^{k} \bigcup_{x=0}^{1} \text{Langage}(A[i,x]) \\
\Leftrightarrow\ & \exists w\, \exists x \in \{0,1\}^k \text{ tels que } w \in \bigcap_{i=1}^{k} A[i,x_i] \\
\Leftrightarrow\ & \exists w\, \exists x \in \{0,1\}^k \text{ tels que } \bigwedge_{i=1}^{k} \bigwedge_{j=1}^{s} |w|_{\sigma_j} \equiv v[i,x_i]_j \pmod{\text{ord}_{i,x_i}(\sigma_j)} \\
\Leftrightarrow\ & \exists w\, \exists x \in \{0,1\}^k \text{ tels que } \bigwedge_{j=1}^{s} \left(\bigwedge_{i=1}^{k} |w|_{\sigma_j} \equiv v[i,x_i]_j \pmod{\text{ord}_{i,x_i}(\sigma_j)} \right) \\
\Leftrightarrow\ & \exists x \in \{0,1\}^k \text{ tel que } \bigwedge_{j=1}^{s} \left(\bigwedge_{i=1}^{k} \bigwedge_{i'=1}^{k} C_{i,i',j}(x) \right),
\end{aligned}$$

où

$$C_{i,i',j}(x) = \left(v[i,x_i]_j \equiv v[i',x_{i'}]_j \pmod{\text{pgcd}(\text{ord}_{i,x_i}(\sigma_j), \text{ord}_{i',x_{i'}}(\sigma_j))} \right).$$

La dernière équivalence est une conséquence du Lemme 3.2.2. Nous obtenons donc

$$\exists w \in \bigcap_{i=1}^{k} \bigcup_{x=0}^{1} \text{Langage}(A[i,x]) \quad \Leftrightarrow \quad \exists x \in \{0,1\}^k \text{ t.q. } \bigwedge_{j=1}^{s} \bigwedge_{i=1}^{k} \bigwedge_{i'=1}^{k} C_{i,i',j}(x).$$

Pour tout $i, i' \in [k]$ et $j \in [s]$, la table de vérité de $C_{i,i',j}$ peut être calculée en évaluant les quatre congruences. Pour les raisons évoquées dans la preuve du Théorème 3.2.3, nous obtenons une expression en forme 2-FNC calculable en espace logarithmique. ■

Corollaire 3.3.17. $\mathsf{IntAuto}_1(\cup^2$ *Automates à groupe abélien serré*$)$ *est* NL-complet.

Nous pouvons également généraliser le Théorème 3.2.7 de façon similaire. Nous n'en donnons pas une preuve formelle puisqu'il suffit de calquer la preuve du Théorème 3.2.7 en appliquant les idées du théorème précédent.

Théorème 3.3.18. $\mathsf{IntAuto}_2$(*Automates à groupe abélien serré*) *est* L-complet.

Nous résumons la complexité du problème d'intersection pour les automates à groupes au Tableau 3.III.

Tableau 3.III – Complexité de IntAuto(Groupes)

	Nombre d'états finaux		
	1	2	3+
$\lvert\Sigma\rvert = 1$	L-complet	L-complet	NP-complet
Abélien serré	L-complet	L-complet	NP-complet
2-élémentaire abélien	$\oplus$L-complet	$\oplus$L-complet	NP-complet [Bea88b]
p-élémentaire abélien	Mod_pL-complet	? ($\in$ NP)	NP-complet [Bea88b]
Abélien	NC^3, $\mathrm{FL}^{\mathrm{Mod}_p\mathrm{L}}$	? ($\in$ NP)	NP-complet [Bea88b]
Cas général	NC [Luk90]	? ($\in$ NP)	NP-complet [Bea88b]

3.4 Langages commutatifs et idempotents

Nous étudions maintenant le problème d'intersection d'automates pour les langages commutatifs et les langages idempotents. Nous rappelons qu'un monoïde est idempotent ssi $x^2 = x$ pour tous ses éléments x. Le cas des langages commutatifs généralise les cas des langages unaires et des langages acceptés par les automates à groupe abélien. Toutefois, contrairement à ces deux instances, le problème est NP-complet, même s'il n'y a qu'un seul état final. En effet, nous remarquons d'abord que PS(Commutatif) et PS(Idempotent) sont NP-complets. Cela découle du Corollaire 3.1.4, car leur problème d'appartenance Memb est NP-complet [Bea88a, Bea88b, BMT92].

Proposition 3.4.1 ([Bea88a, Bea88b, BMT92])**.** PS(*Commutatif*) *et* PS(*Idempotent*) *sont* NP-complets*, même pour un état final.*

Cela étant dit, la complexité de PS_1 diminue fortement lorsque les monoïdes proviennent de la variété $J1$ des monoïdes commutatifs et idempotents. En effet,

la complexité chute drastiquement de NP à AC^0 comme le montre le théorème suivant :

Proposition 3.4.2. $\mathsf{PT}(J1) \in \mathrm{AC}^0$.

Démonstration. Nous nous inspirons de la technique de [BMT92] utilisée pour obtenir $\mathsf{Memb}(J1) \in \mathrm{AC}^0$. Soient $A = \{g \in \{g_1, \ldots, g_k\} : {b_i}^g = b_i \quad \forall i \in [m]\}$ et $a = \prod_{g \in A} g$. Montrons qu'il existe $f \in \langle g_1, \ldots, g_k \rangle$ tels que $i^f = b_i$ pour tout $i \in [m]$ ssi $i^a = b_i$ pour tout $i \in [m]$. Puisque ce dernier test peut être facilement vérifié en AC^0, cela complète la preuve.

$\Leftarrow$) Direct puisque $a \in \langle g_1, \ldots, g_k \rangle$.

$\Rightarrow$) Supposons qu'il existe $f \in \langle g_1, \ldots, g_k \rangle$ tel que $i^f = b_i$ pour tout $i \in [m]$. Nous montrons d'abord que $i^{af} = i^f$ pour tout $i \in [m]$. Sans perte de généralité, posons $A = \{a_1, \ldots, a_l\}$. Soit $i \in [m]$, alors

$$
\begin{aligned}
i^{af} &= i^{fa} && (\langle g_1, \ldots, g_k \rangle \text{ est commutatif}) \\
&= (i^f)^a \\
&= {b_i}^a && (\text{par définition de } f) \\
&= {b_i}^{a_1 \cdots a_l} && (\text{par définition de } a) \\
&= ({b_i}^{a_1})^{a_2 \cdots a_l} \\
&= {b_i}^{a_2 \cdots a_l} && (\text{par définition de } A) \\
&= b_i && (\text{de façon similaire, par induction sur } l) \\
&= i^f && (\text{par définition de } f).
\end{aligned}
$$

Nous montrons maintenant que $i^f = i^g$ pour tout $i \in [m]$. Puisque $\langle g_1, \ldots, g_k \rangle$ est commutatif et idempotent, nous pouvons supposer, sans perte de généralité, que

$f = f_1 \cdots f_{k'}$ pour des $f_j \in \{g_1, \ldots, g_k\}$ distincts. Soit $i \in [m]$, alors

$$\begin{aligned}
{b_i}^{f_j} &= (i^f)^{f_j} && \text{(par définition de } f\text{)} \\
&= i^{ff_j} \\
&= i^{f_1 \cdots f_{k'} f_j} && (f = f_1 \cdots f_{k'}) \\
&= i^{f_1 \cdots f_{k'}} && \text{(par commutativité et idempotence)} \\
&= i^f && (f = f_1 \cdots f_{k'}) \\
&= b_i && \text{(par définition } f\text{)}.
\end{aligned}$$

Ainsi, $f_j \in A$ et $\{f_1, \ldots, f_{k'}\} \subseteq A$. Nous concluons donc que $i^{af} = i^a$, et ainsi que $i^a = i^{af} = i^f = b_i$ pour tout $i \in [m]$. ■

Corollaire 3.4.3. $\mathsf{IntAuto}_1(J1) \in \mathrm{AC}^0$.

À défaut de pouvoir bien localiser la complexité de $\mathsf{PS}_2(J1)$, nous montrons qu'elle est strictement plus grande que celle de $\mathsf{PS}_1(J1)$. En effet, $\mathsf{PS}_2(J1)$ est ardu pour L.

Proposition 3.4.4. $\mathsf{IntAuto}_2(J1)$ *est ardu pour* L.

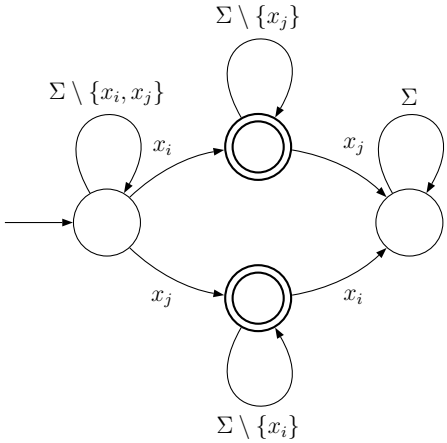
Démonstration. Nous donnons une réduction à partir de 2-$\oplus$SAT. Pour une clause $(x_i \oplus x_j)$, nous construisons un automate possédant quatre états (dont deux états finaux) qui accepte le langage $(\Sigma \backslash \{x_i, x_j\})^*(x_i x_i^* + x_j x_j^*)(\Sigma \backslash \{x_i, x_j\})^*$. Cet automate est illustré à la Figure 3.9.

Pour une clause $(x_i \oplus x_j \oplus 1)$, nous construisons un automate acceptant le complément du langage précédent. ■

Malheureusement, une classification exacte de $\mathsf{IntAuto}_2(J1)$ n'est pas obtenue dans ce travail. Par contre, les automates utilisés dans la preuve précédente ont une structure bien précise. En effet, ils ont exactement quatre états et au plus une transition entre deux états distincts. En se restreignant à de tels automates, nous pouvons montrer que le problème est L-complet.

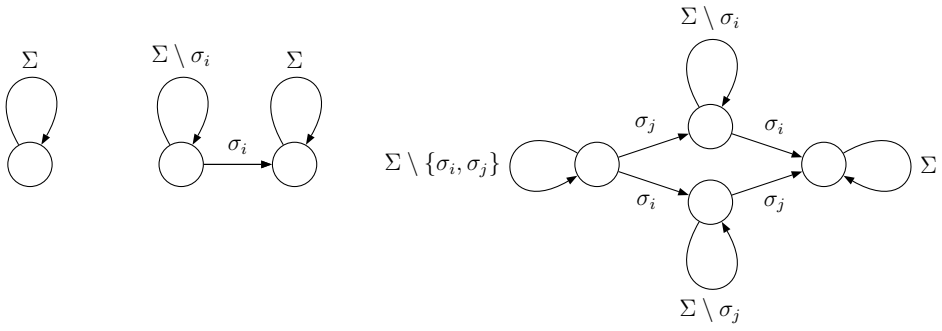
Proposition 3.4.5. $\mathsf{IntAuto}_2(J1)$ *est* L-complet *lorsque les automates ont au plus quatre états et possèdent au plus une transition entre chaque paire d'états distincts.*

Figure 3.9 – Automate acceptant $(\Sigma \setminus \{x_i, x_j\})^*(x_i x_i^* + x_j x_j^*)(\Sigma \setminus \{x_i, x_j\})^*$



Démonstration. Nous montrons que le problème est dans L en le réduisant à 2-⊕SAT. Nous remarquons d'abord qu'il n'y a que trois façons d'obtenir de tels automates. Nous illustrons ces automates à la Figure 3.10.

Figure 3.10 – Automates possibles



Le langage accepté par ces automates est complètement caractérisé par la présence ou l'absence d'occurrences des lettres σ_i et σ_j dans un mot. Par exemple, dans le troisième automate, si l'état de gauche est initial et que l'état du haut est final, les mots acceptés sont exactement ceux qui possèdent au moins un σ_j et aucun σ_i. Nous exploitons cette remarque pour construire des formules logiques décrivant les mots acceptés par les automates. Dans l'exemple précédent, cette formule serait $\neg x_i \wedge x_j$. Puisque nous permettons deux états finaux, il est possible que nous ayons

à prendre la disjonction de deux formules. Or, il est toujours possible d'obtenir une formule en 2-⊕FNC. Cela peut être montré exhaustivement. Par exemple, dans l'exemple précédent, si nous considérons aussi l'état du bas comme étant final, nous obtenons $(x_i \wedge \neg x_j) \vee (x_i \wedge \neg x_j) = x_i \oplus x_j$. ■

À des fins de complétude, nous notons que le problème est NP-complet pour trois états finaux tel que démontré dans [Bea88b].

Proposition 3.4.6 ([Bea88b])**.** $\mathsf{IntAuto}_3(J1)$ *est* NP-complet.

Nous résumons la complexité du problème d'intersection pour les langages commutatifs et les langages idempotents au Tableau 3.IV.

Tableau 3.IV – Complexité de IntAuto(Commutatifs) et IntAuto(Idempotents)

	Nombre d'états finaux		
	1	2	3+
$J1$	AC^0	L-ardu	NP-complet
Idempotent	NP-complet	NP-complet	NP-complet
Commutatif	NP-complet	NP-complet	NP-complet

3.5 Langages finis

Nous terminons notre analyse de $\mathsf{IntAuto}(X)$ en considérant brièvement les langages finis.

Nous introduisons d'abord un nouveau type d'automate. Rappelons qu'une arborescence est un graphe acyclique dirigé possédant un sommet u (*racine*) tel que pour tout sommet v il existe exactement un chemin de u vers v. Rappelons également qu'un *état puits* est un état γ tel que $T_\sigma(\gamma) = \gamma$ pour tout $\sigma \in \Sigma$.

Définition 3.5.1. Un automate est *sous forme d'arborescence* si son graphe est une union disjointe d'arborescences, pour lesquelles chaque feuille est transformée en état puits. Notons que la présence de plusieurs transitions d'un état α vers un état β est considérée comme un seul arc.

Les automates sous forme d'arborescence s'avèrent pertinents puisqu'ils acceptent des langages finis lorsqu'aucun état puits n'est final. Un exemple d'un tel automate est illustré à la Figure 3.11. Nous obtenons une classification partielle de la complexité de $\mathsf{IntAuto}_b$ pour ces automates.

Théorème 3.5.2. $\mathsf{IntAuto}_1$(*Langage fini et arborescence*) $\in$ L.

Démonstration. Pour chaque automate A_i, il existe un unique chemin de longueur m_i de l'état initial vers l'état final. Ce chemin définit un langage de la forme $S_{i,1} \cdots S_{i,m_i}$ où $S_{i,j} \subseteq \Sigma$. Pour qu'un mot soit accepté par tous les automates, il est nécessaire que $m_1 = \cdots = m_k$ puisque A_i n'accepte que des mots de longueur m_i. Il suffit donc de vérifier que $m_1 = \cdots = m_k$ puis que $S_{1,j} \cap \cdots \cap S_{k,j} \neq \emptyset$ pour tout $j \in [m_1]$. Nous remarquons qu'il est possible de vérifier si $\sigma \in S_{i,j}$ en espace logarithmique. En effet, il suffit de remonter l'arbre défini par A_i, à partir de son état final, en suivant toujours l'unique prédécesseur jusqu'au $j^{\text{ème}}$ arc. Il suffit ensuite de vérifier si cet arc est étiqueté par σ. L'algorithme global est donc le suivant :

```
si ¬(m_1 = ··· = m_k) alors rejeter
pour j = 1 a m_1 faire
    x ← faux
    pour σ ∈ Σ faire
        y ← vrai
        pour i = 1 a k faire
            y ← y ∧ (σ ∈ S_{i,j})
        x ← x ∨ y
    si ¬x alors rejeter
accepter
```

■

Au chapitre suivant, nous verrons que $\mathsf{IntAuto}_1^k$(Langages finis et arborescence) est L-complet. Nous obtenons donc le corollaire suivant :

Corollaire 3.5.3. $\mathsf{IntAuto}_1$(*Langages finis et arborescence*) *est* L-complet.

Proposition 3.5.4. $\mathsf{IntAuto}_2$(*Langages finis, arborescence et* $|\Sigma| = 2$) $\leq_{\log}^m$ 2-SAT.

Démonstration. Soient m_i et m'_i, respectivement les longueurs des chemins menant de l'état initial de A_i vers son premier état final et vers son second état final. L'automate A_i accepte un langage de la forme $S_{i,j} \cdots S_{i,m_i} \cup S_{i,1} \cdots S_{i,m'_i}$. Il existe donc un mot accepté par tous les automates ssi

$$\bigcap_{i=1}^{k} \left(S_{i,1} \cdots S_{i,m} \cup S'_{i,1} \cdots S'_{i,m}\right) \neq \emptyset$$

où $S_{i,j} = \emptyset$ (resp. $S'_{i,j} = \emptyset$) lorsque $j > m_i$ (resp. $j > m'_i$).

Posons

$$l_{i,j} = \begin{cases} 0 & \text{si } S_{i,j} = \emptyset \\ x_j & \text{si } S_{i,j} = \{1\} \\ \neg x_j & \text{si } S_{i,j} = \{0\} \\ 1 & \text{si } S_{i,j} = \{0,1\} \end{cases}$$

et $l'_{i,j}$ de façon similaire à partir de $S'_{i,j}$ où nous supposons $\Sigma = \{0,1\}$. Nous construisons l'expression booléenne suivante, satisfaite par exactement tous les mots acceptés par A_i :

$$\begin{aligned} C_i(x) &= \bigwedge_{j=1}^{m} l_{i,j} \vee \bigwedge_{j=1}^{m} l'_{i,j} \\ &= \bigwedge_{j=1}^{m} \bigwedge_{j'=1}^{m} \left(l_{i,j} \vee l'_{i,j'}\right) \quad \text{(par distributivité).} \end{aligned}$$

Nous avons ainsi $\bigcap_{i=1}^{k} A_i \neq \emptyset$ si et seulement si $\bigwedge_{i=1}^{k} C_i(x)$ est satisfaisable. De plus, l'expression utilisée est en 2-FNC. ■

Théorème 3.5.5. 2-SAT $\leq^m_{\log}$ IntAuto$_2$(*Langages finis, arborescence et* $|\Sigma| = 2$).

Démonstration. Soit $C(x)$ l'expression booléenne $\bigwedge_{i=1}^{k} C_i(x)$ où

$$C_i(x) = (x_{r_i} \oplus b_i) \vee (x_{s_i} \oplus b'_i)$$

et $b_i, b'_i \in \{0,1\}$ indiquent s'il faut prendre ou non la négation de la variable associée. Pour tout $i \in [k]$, nous définissons le langage suivant :

$$X_i = \{w \in \{0,1\}^m \,:\, w_{r_i} = \neg b_i \vee w_{s_i} = \neg b'_i\}\,.$$

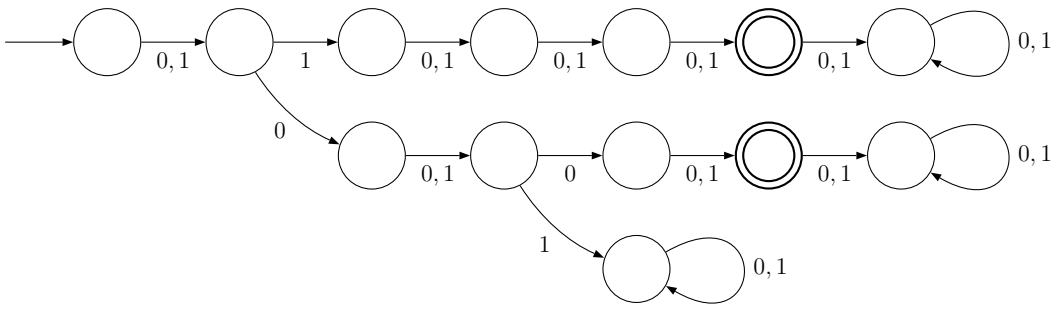
Celui-ci est l'ensemble des affectations satisfaisant C_i. Ainsi $\bigcap_{i=1}^{k} X_i \neq \emptyset$ si et seulement si $C(x)$ est satisfaisable. De plus, il est possible de construire un automate acceptant X_i qui soit sous forme d'arborescence et qui possède deux états finaux. Par exemple, supposons que $C_i(x) = x_2 \vee \neg x_4$ et que l'ensemble des variables soit $\{x_1, \ldots, x_5\}$, alors l'automate obtenu est illustré à la Figure 3.11. ■

Corollaire 3.5.6. IntAuto$_2$(*Langages finis, arboresence et* $|\Sigma| = 2$) *est* NL-complet.

Théorème 3.5.7. IntAuto(*Langages finis*) $\in$ NP.

Démonstration. Il suffit de construire une machine de Turing qui choisit, de façon non déterministe, un mot accepté par tous les automates. Puisque les langages

Figure 3.11 – Exemple d'automate acceptant X_i pour $C_i(x) = x_2 \vee \neg x_4$



sont finis, la taille des mots acceptés pas les automates sont bornés par le nombre d'états. Ainsi, la taille d'un mot accepté par tous les automates est forcément borné par la longueur de l'entrée. ■

Théorème 3.5.8. Monotone 1-in-3 3-SAT $\leq^m_{\log}$ $\mathsf{IntAuto}_3$(*Langages finis et arborescence*).

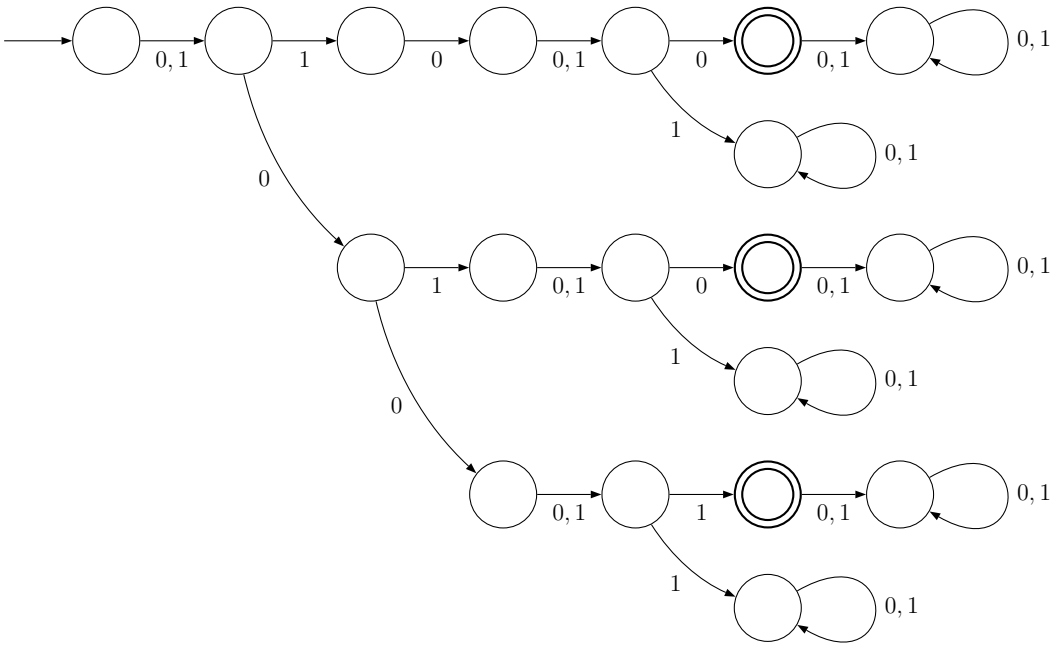
Démonstration. Soit $C(x)$ l'expression booléenne $\bigwedge_{i=1}^{k} C_i(x)$ où

$$C_i(x) = x_{r_i} \vee x_{s_i} \vee x_{t_i}.$$

Pour tout $i \in [k]$, nous définissons le langage suivant :

$$X_i = \{w \in \{0,1\}^m \ : \ w_{r_i} w_{s_i} w_{t_i} \in \{100, 010, 001\}\}.$$

Figure 3.12 – Exemple d'automate acceptant X_i pour $C_i(x) = x_2 \vee x_3 \vee x_5$



Nous avons $\bigcap_{i=1}^{k} X_i \neq \emptyset$ ssi $C(x)$ est satisfaisable. De plus, il est possible de

construire un automate acceptant X_i qui soit sous forme d'arborescence et qui possède trois états finaux. Par exemple, supposons que $C_i(x) = x_2 \vee x_3 \vee x_5$, alors l'automate obtenu est illustré à la Figure 3.12. ■

Corollaire 3.5.9. $\mathsf{IntAuto}_3$(*Langages finis et arboresence*) *est* NP-complet.

En fusionnant les trois états finaux de chacun des automates dans la preuve du théorème précédent, nous obtenons des automates qui possèdent un seul état final et qui acceptent toujours un langage fini. Cependant, ceux-ci ne sont plus sous forme d'arborescence. Nous obtenons donc le résultat suivant :

Corollaire 3.5.10. $\mathsf{IntAuto}_1$(*Langages finis*) *est* NP-complet.

Nous résumons la complexité du problème d'intersection pour les langages finis au Tableau 3.V.

Tableau 3.V – Complexité de IntAuto(Langages finis)

	Nombre d'états finaux		
	1	2	3+
Arborescences	L-complet	NL-ardu	NP-complet
Cas général	NP-complet	NP-complet	NP-complet

CHAPITRE 4

PROBLÈME D'INTERSECTION D'AUTOMATES BORNÉ

Nous complétons notre étude en nous penchant sur le problème d'intersection pour lequel le nombre d'automates est borné. Tel que mentionné plus tôt, ce problème est NSPACE($g(n) \log n$)-complet [LR92] lorsque le nombre d'automates est borné par $1 \leq g(n) \leq n$. Nous nous concentrerons sur le cas où $g(n) = k$ pour une certaine constante $k \in \mathbb{N}$. De façon générale, ce problème est NL-complet. Nous répertorions certaines instances de $\mathsf{IntAuto}^k(X)$ plus simples à résoudre pour les langages étudiés au chapitre précédent.

Le problème $\mathsf{IntAuto}^k$ se résoud également en temps $O(n^k)$. En effet, il suffit d'utiliser la construction classique du produit cartésien de k automates puis de vérifier s'il existe un chemin de l'état initial vers un état final. Karakostas, Lipton et Viglas ont démontré que l'existence de meilleurs algorithmes que celui-ci pour résoudre $\mathsf{IntAuto}^k$ permettrait de séparer NL et P [KLV03]. Nous nous penchons donc brièvement sur la complexité en temps du problème $\mathsf{IntAuto}^k$.

4.1 Complexité

Tel que mentionné dans [LR92], $\mathsf{IntAuto}^k$(Unaire) est L-complet.

Proposition 4.1.1. $\mathsf{IntAuto}^k$(*Unaire*) *et* $\mathsf{IntAuto}_1^k$(*Unaire*) *sont* L-complets.

Démonstration. Pour l'appartenance à L, il suffit de noter qu'il existe un mot accepté par tous les automates ssi un mot de taille inférieure à n^k est accepté par tous les automates. Puisque l'alphabet est unaire, il n'y a que n^k mots à tester.

La complétude pour $\mathsf{IntAuto}_1^k$(Unaire) peut être obtenue à partir du problème L-complet [CM87] DFA qui consiste à déterminer s'il existe un chemin entre deux sommets s et t dans une forêt dirigée. Il suffit de considérer le graphe comme un automate unaire et de fixer s comme état initial et t comme état final. Il existe ainsi un mot accepté par l'automate ssi t est accessible à partir de s. ■

Proposition 4.1.2. $\mathsf{IntAuto}^k$(*Groupes*), $\mathsf{IntAuto}^k$(*Groupes abéliens*) *et* $\mathsf{IntAuto}^k$(*Unaire et groupes abéliens*) *sont* L-complets*, même pour un état final par automate.*

Démonstration. Pour l'appartenance à L, nous utilisons tout simplement l'algorithme classique. En effet, il suffit de construire le produit cartésien des k automates puis de vérifier s'il existe un chemin de l'état initial vers un état final. Notons d'abord que le produit cartésien d'automates à groupe est aussi un automate à groupe. Un tel automate peut être considéré comme un graphe non dirigé. L'emprunt d'un arc, étiqueté par σ, en sens inverse équivaut à appliquer T_σ^{-1}. Pour un arc (une transition) de γ vers γ' étiqueté par σ, nous ajoutons donc l'arc (γ', γ) étiqueté σ^{-1}. Nous obtenons ainsi un graphe non orienté. Par [Rei05], il est possible de vérifier s'il existe un chemin dans un graphe non orienté en espace logarithmique.

La complétude peut être obtenue à partir du problème L-complet [CM87] DCA. Celui-ci consiste à déterminer si deux points a et b appartiennent au même cycle d'une permutation. Il suffit de représenter la permutation par un automate (unaire) et de fixer a comme état initial et b comme état final. Il existe ainsi un mot accepté par l'automate ssi a et b font parties du même cycle. ■

Proposition 4.1.3. $\mathsf{IntAuto}^k(J1) \in \mathrm{AC}^0$.

Démonstration. Considérons un seul état final par automate, par la Proposition 3.4.2, il est possible de vérifier si un mot est accepté par tous les automates en AC^0. Le nombre de tuplets d'états finaux est borné par n^k, il est donc possible de tous les tester en parallèle et de combiner leur résultat par une porte $\vee$. ■

Proposition 4.1.4. $\mathsf{IntAuto}^k$(*Langages finis et arboresence*) *et* $\mathsf{IntAuto}_1^k$(*Langages finis et arboresence*) *sont* L-complets*.*

Démonstration. Considérons un seul état final par automate, par le Théorème 3.5.2, il est possible de vérifier si un mot est accepté par tous les automates en espace logarithmique. Le nombre de tuplets d'états finaux est borné par n^k, il est donc possible de tous les tester. La complétude découle de la réduction à partir de DFA donnée à la Proposition 4.1.1. ■

Il est possible de généraliser les résultats précédents et de préserver la même complexité en ajoutant des clauses $\cup$.

Proposition 4.1.5. *Si* $\mathsf{IntAuto}^k(X)$ *est* L-complet *(resp.* $\mathsf{IntAuto}^k(X) \in \mathrm{AC}^0$*) alors* $\mathsf{IntAuto}^k(\cup X)$ *est* L-complet *(resp.* $\mathsf{IntAuto}^k(\cup X) \in \mathrm{AC}^0$*).*

Démonstration. Le nombre d'automates dans chaque clause $\bigcup$ est borné par la longueur de l'entrée. Il y a donc, au plus, n^k façon de sélectionner un automate par clause. Nous vérifions s'il existe un mot accepté par tous les automates pour chacun des tuplets possibles. Lorsque $\mathsf{IntAuto}^k(X)$ est L-complet, nous procédons séquentiellement en espace logarithmique. Lorsque $\mathsf{IntAuto}^k(X) \in \mathrm{AC}^0$, nous procédons en parallèle en combinant le résultat par une porte $\vee$. ■

4.2 Meilleurs algorithmes

Nous survolons des résultats récents de Karakostas, Lipton et Viglas. Le travail développé dans [KLV03] permet, sous certaines hypothèses, d'obtenir de meilleurs algorithmes pour certains problèmes et de séparer les classes NL et P. Les hypothèses considérées par les auteurs sont les suivantes :

Hypothèse 4.2.1 ([KLV03])**.** Étant donné k automates de taille t, alors il existe un algorithme résolvant $\mathsf{IntAuto}^k$ en temps $t^{(k/f(k))+d}$ où $d > 0$ est une constante et f est une fonction non bornée qui dépend seulement de k.

Hypothèse 4.2.2 ([KLV03])**.** Étant donné k automates de taille t et un automate de taille t', alors il existe un algorithme résolvant $\mathsf{IntAuto}^{k+1}$ en temps $t^{(k/f(k))+d}t'$ où $d > 0$ est une constante et f est une fonction non bornée qui dépend seulement de k.

La véracité de la première hypothèse impliquerait l'existence de meilleurs algorithmes pour les problèmes NP-complets Subset-Sum et Factoring, tel qu'énoncé dans les deux théorèmes suivants.

Théorème 4.2.3 ([KLV03])**.** *Si l'Hypothèse 4.2.1 est vraie, alors il existe une famille d'algorithmes résolvant* Subset-Sum *en temps* $O(2^{\epsilon n})$ *pour tout* $\epsilon > 0$.

Théorème 4.2.4 ([KLV03]). *Si l'Hypothèse 4.2.1 est vraie, alors il existe une famille d'algorithmes résolvant* Factoring *en temps* $O(2^{\epsilon n})$ *pour tout* $\epsilon > 0$.

De plus, il devient possible de simuler plus efficacement une machine de Turing non déterministe par une machine déterministe.

Théorème 4.2.5 ([KLV03]). *Si l'Hypothèse 4.2.1 est vraie, alors*

$$\mathrm{NTIME}(f(n)) \subseteq \mathrm{TIME}(2^{\epsilon f(n)})$$

pour tout $\epsilon > 0$.

La véracité de la seconde hypothèse permet de séparer NL (et donc L) de P.

Théorème 4.2.6 ([KLV03]). *Si l'Hypothèse 4.2.2 est vraie, alors* $\mathrm{NL} \neq \mathrm{P}$.

Ce dernier résultat est obtenu en construisant k automates vérifiant si une suite de calculs d'une machine de Turing NL est valide sur une portion du ruban de travail. En exploitant l'hypothèse, cela permet d'obtenir une simulation d'une machine NL en temps quadratique. Or, il est bien connu que P est strictement plus grand que $\mathrm{TIME}(n^2)$, ce qui mène à la séparation.

Une analyse attentive des résultats de [KLV03] permet de remarquer que tous les automates utilisés acceptent des langages finis et ne possèdent qu'un seul état final. Les hypothèses peuvent donc être relaxées au problème $\mathsf{IntAuto}_1^k$(Langages finis) qui, à priori, semble plus facilement attaquable. En fait, lors de la publication de l'article, la taille de l'automate minimal acceptant l'intersection de deux langages finis n'était pas connue. Nous savons maintenant, par [HS07], que $O(mn)$ états finaux sont nécessaires dans le pire cas, de la même façon que pour les langages réguliers en général. L'algorithme classique pour résoudre $\mathsf{IntAuto}^k$, qui repose sur la construction de l'automate acceptant l'intersection des k automates, est donc aussi limité à un temps de calcul de l'ordre de $O(n^k)$ pour les langages finis.

Toutefois, cela soulève certains questionnements. Pour quels types d'automates les Hypothèses 4.2.1 et 4.2.2 sont-elles vraies ? Où réside la difficulté de battre la

barrière du $O(n^k)$? Nous ne répondons que très partiellement à ces questions, à la lumière des résultats du chapitre précédent.

Nous montrons d'abord qu'il est possible de faire mieux que $O(n^k)$ pour les problèmes $\mathsf{IntAuto}_b^k$(Unaire) et $\mathsf{IntAuto}_b^k$(Langages finis et arboresence) pour certains b. Le second cas contraste avec $\mathsf{IntAuto}_1^k$(Langages finis) qui est à la base de l'hypothèse originale.

Proposition 4.2.7. $\mathsf{IntAuto}_b^k(\textit{Unaire}) \in \mathrm{TIME}(b^k n^c)$ *pour un certaine constante* $c \in \mathbb{N}$ *indépendante de* k.

Démonstration. Considérons un état final par automate. Tel qu'observé au chapitre précédent, pour vérifier s'il existe un mot accepté par tous les automates, il suffit de vérifier si $d_i \equiv d_j \pmod{\mathrm{pgcd}(\mathrm{ord}_i(a), \mathrm{ord}_j(a))}$ pour tout $i, j \in [k]$ où d_i est la position de l'état final dans l'automate A_i. Ce test peut être effectué en temps polynomial. Puisque le nombre d'états finaux de chaque automate est borné par b, nous n'avons qu'à répéter cette procédure b^k fois. ■

Corollaire 4.2.8. *Soient* $d \in \mathbb{N}$, $g : \mathbb{N} \to \mathbb{R}^+$, c *la constante obtenue à la proposition précédente et* $b = n^{\frac{1}{g(k)} + \frac{d-c}{k}}$, *alors*

$$\mathsf{IntAuto}_b^k(\textit{Unaire}) \in \mathrm{TIME}\left(n^{\frac{k}{g(k)}+d}\right).$$

Proposition 4.2.9. $\mathsf{IntAuto}_b^k(\textit{Langages finis et arborescence}) \in \mathrm{TIME}(b^k n^{c'})$ *pour un certaine constante* $c' \in \mathbb{N}$ *indépendante de* k.

Démonstration. Considérons un état final par automate. L'algorithme utilisé dans la preuve du Théorème 3.5.2 permet de vérifier si les k automates acceptent un mot en commun en temps polynomial. Puisque le nombre d'états finaux de chaque automate est borné par b, nous pouvons n'avons qu'à répéter cette procédure b^k fois. ■

Corollaire 4.2.10. *Soient* $d \in \mathbb{N}$, $g : \mathbb{N} \to \mathbb{R}^+$, c' *la constante obtenue à la*

proposition précédente et $b = n^{\frac{1}{g(k)}+\frac{d-c'}{k}}$*, alors*

$$\mathsf{IntAuto}_b^k(\textit{Langages finis et arborescence}) \in \text{TIME}\left(n^{\frac{k}{g(k)}+d}\right).$$

Nous montrons maintenant que des conséquences similaires aux Théorèmes 4.2.3 et 4.2.4 valent si l'Hypothèse 4.2.1, restreinte aux automates à groupe abélien ne possédant qu'un seul état final, s'avère vraie. Cela est intéressant puisque le problème d'intersection pour les automates à groupe abélien est plus simple à résoudre, en ce sens qu'il appartient à NC^3 lorsque k n'est pas borné.

Proposition 4.2.11. *Si l'Hypothèse 4.2.1 restreinte à* $\mathsf{IntAuto}_1^k(\textit{Groupes abéliens})$ *est vraie, alors il existe une famille d'algorithmes résolvant* Change-Making *en temps* $O(2^{\epsilon n})$ *pour tout* $\epsilon > 0$*.*

Démonstration. Nous choisissons k nombres premiers $p_1, \ldots, p_k$ de taille plus grande que n/k. Nous construisons k automates $A_1, \ldots, A_k$ tels que

$$\text{Langage}(A_i) = \{w \in \{\sigma_1, \ldots, \sigma_m\}^* : a_1|w|_{\sigma_1} + \ldots + a_m|w|_{\sigma_m} \equiv b \pmod{p_i}\}.$$

L'automate A_i possède $p_i \in O(2^{n/k})$ états et un seul état final. S'il existe une solution à l'instance du problème Change-Making, alors il existe forcément un mot accepté par tous les automates. À l'opposé, s'il existe un mot w accepté par tous les automates, alors $a_1|w|_{\sigma_1} + \ldots + a_m|w|_{\sigma_m} \equiv b \pmod{p_i}$ pour tout $i \in [k]$. Ainsi, $a_1|w|_{\sigma_1} + \ldots + a_m|w|_{\sigma_m} \equiv b \pmod{p_1 \cdots p_k}$. Or, puisque $p_1 \cdots p_k > 2^n \geq b$, nous avons $a_1|w|_{\sigma_1} + \ldots + a_m|w|_{\sigma_m} = b$. Le temps nécessaire pour construire les automates est $O(2^{n/k})$. Le temps total est donc de

$$O(2^{n/k} + 2^{n/f(k)+dn/k}).$$

En supposant que $f(k)$ croît moins rapidement que k, nous obtenons un temps de $O(2^{n/f(k)})$. Puisque $f(k)$ n'est pas bornée, $1/f(k)$ peut devenir aussi petit que désiré en choisissant un k approprié. ∎

CHAPITRE 5

CONCLUSION

5.1 Bilan

Nous avons d'abord établi des relations entre le problème d'intersection d'automates et différents problèmes algébriques. Pour une pseudovariété de monoïdes X, nous avons :

$$\mathsf{Memb}(X) \leq^m_{\mathrm{AC}^0} \mathsf{PT}(X) \equiv^m_{\mathrm{AC}^0} \mathsf{PS}_1(X) \leq^m_{\mathrm{AC}^0} \mathsf{PS}_b(X) \equiv^m_{\mathrm{AC}^0} \mathsf{IntAuto}_b(X)$$

ainsi que $\mathsf{ST}(X) \leq^m_{\mathrm{AC}^0} \mathsf{PS}(X)$.

Par la suite, nous avons étudié $\mathsf{IntAuto}_b(X)$ pour les langages unaires, les automates à groupe, les langages finis, les langages commutatifs, et les langages idempotents. Cette étude nous a permis de placer $\mathsf{IntAuto}_b(X)$ dans les classes AC^0, L, NL, $\oplus$L, Mod_pL, NC^3, NC et NP. Dans la plupart des cas, nous obtenons ainsi une caractérisation de ces classes en fonction du problème d'intersection d'automates. Cela est particulièrement intéressant pour la classe Mod_pL puisque cela permet de traiter de classes de comptage logarithmique en terme d'automates.

Nous avons également montré que $\mathsf{IntAuto}_b(X)$ ne semble pas efficacement résoluble pour $b \geq 3$, mais que sa complexité est (ou apparaît être) à l'intérieur de P pour $b = 1, 2$. Le cas où $b = 2$ demeure toutefois incomplet pour les restrictions étudiées. De plus, la généralisation $\mathsf{IntAuto}_b(\cup^{k'} X)$ du problème d'intersection a été introduite et nous avons démontré que $\mathsf{IntAuto}_1(\cup^2$ unaire) est NL-complet.

Nous avons terminé notre étude en considérant brièvement le problème d'intersection d'automates borné $\mathsf{IntAuto}^k(X)$. Nous avons montré que pour les différentes familles de langages considérées dans ce travail, le problème $\mathsf{IntAuto}^k(X)$ se situe dans AC^0 ou est L-complet. De plus, nous avons énoncé des résultats de Karakostas, Lipton et Viglas concernant des hypothèses sur la découverte de meilleurs algorithmes pour résoudre $\mathsf{IntAuto}^k(X)$, et nous les avons étoffés à la lumière de

notre étude de $\mathsf{IntAuto}_b(X)$.

5.2 Questions ouvertes

Les points d'interrogations au Tableau 3.III et la faiblesse des Propositions 3.4.5 et 3.5.4 indiquent que des travaux subséquents sont nécessaires pour localiser adéquatement la complexité du problème d'intersection d'automates à deux états finaux.

Les résultats obtenus pour les langages unaires et les 2-groupes élémentaires abéliens suggèrent que $\mathsf{IntAuto}_2(X)$ demeure résoluble efficacement pour les groupes abéliens et les monoïdes commutatifs et idempotents. Toutefois, notre preuve pour $\mathsf{IntAuto}_2$(2-groupes élémentaires abéliens) $\in \oplus$L ne peut être généralisée. Est-il possible que le problème $\mathsf{IntAuto}_2$(p-groupes élémentaires abéliens) soit plus difficile à résoudre pour $p > 2$ que pour $p = 2$? Est-ce que $\mathsf{IntAuto}_2$(Groupes abéliens) $\in$ NC ?

Il serait intéressant de répondre à de telles questions puisque $\mathsf{IntAuto}_2$(Groupes abéliens) est équivalent au problème de faisabilité de congruences linéaires de la forme $(B_1x \equiv b_1 \pmod{q_1} \vee B_1x \equiv b'_1 \pmod{q_1}) \wedge \cdots \wedge (B_kx \equiv b_k \pmod{q_k} \vee B_kx \equiv b'_k \pmod{q_k})$, ce qui constitue une généralisation de LCON. Le problème $\mathsf{IntAuto}_1(\cup^2$ groupes abéliens) conduit à des congruences similaires, mais dans lesquelles B_i et q_i diffèrent dans chaque clause $\vee$. Ainsi, l'étude de $\mathsf{IntAuto}_2(X)$ et $\mathsf{IntAuto}_2(\cup^2)$ semble être une piste de recherche fructueuse pour obtenir des variantes du problème d'intersection d'automates à la fois intéressantes et efficacement résolubles.

BIBLIOGRAPHIE

[AV10] Arvind, V. et T. C. Vijayaraghavan: *Classifying problems on linear congruences and abelian permutation groups using logspace counting classes*. Comput. Complex., 19:57–98, March 2010, ISSN 1016-3328.

[Bal02] Bala, S.: *Intersection of regular languages and star hierarchy*. Dans *Proceedings of the 29th International Colloquium on Automata, Languages and Programming*, ICALP '02, pages 159–169. Springer-Verlag, 2002, ISBN 3-540-43864-5.

[BDHM92] Buntrock, G., C. Damm, U. Hertrampf et C. Meinel: *Structure and importance of logspace-mod class*. Theory of Computing Systems, 25:223–237, 1992, ISSN 1432-4350.

[Bea88a] Beaudry, M.: *Membership testing in commutative transformation semigroups*. Information and Computation, 79(1):84–93, 1988, ISSN 0890-5401.

[Bea88b] Beaudry, M.: *Membership testing in transformation monoids*. Thèse de doctorat, McGill University, 1988.

[BL77] Berman, J. et A. Lingas: *On the complexity of regular languages in terms of finite automata*. Rapport technique 304, Polish Academy of Sciences, 1977.

[BLS87] Babai, L., E. M. Luks et A. Seress: *Permutation groups in* NC. Dans *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, STOC '87, pages 409–420. ACM, 1987, ISBN 0-89791-221-7.

[BMT92] Beaudry, M., P. McKenzie et D. Thérien: *The membership problem in aperiodic transformation monoids*. J. ACM, 39(3):599–616, 1992.

[CM87] Cook, S. A. et P. McKenzie: *Problems complete for deterministic logarithmic space*. J. Algorithms, 8(3):385–394, 1987.

[DF04] Dummit, D. S. et R. M. Foote: *Abstract algebra*. Wiley, 2004, ISBN 9780471452348.

[Dix67] Dixon, J.D.: *Problems in group theory*. Blaisdell Publishing Company, 1967.

[FG06] Flum, J. et M. Grohe: *Parameterized complexity theory*. Texts in theoretical computer science. Springer, 2006, ISBN 9783540299523.

[FHL80] Furst, M. L., J. E. Hopcroft et E. M. Luks: *Polynomial-time algorithms for permutation groups*. Dans *FOCS*, pages 36–41, Octobre 1980.

[FSS84] Furst, M., J. B. Saxe et M. Sipser: *Parity, circuits, and the polynomial-time hierarchy*. Theory of Computing Systems, 17:13–27, 1984, ISSN 1432-4350.

[Gal76] Galil, Z.: *Hierarchies of complete problems*. Acta Informatica, 6:77–88, 1976, ISSN 0001-5903.

[GJ79] Garey, M.R. et D.S. Johnson: *Computers and intractability: a guide to the theory of NP-completeness*. Series of books in the mathematical sciences. W. H. Freeman, 1979.

[GP10] Geffert, V. et G. Pighizzini: *Two-way unary automata versus logarithmic space*. Dans *Developments in Language Theory*, tome 6224 de *Lecture Notes in Computer Science*, pages 197–208. Springer, 2010, ISBN 978-3-642-14454-7.

[HK09] Holzer, M. et M. Kutrib: *Descriptional and computational complexity of finite automata*. Dans *Language and Automata Theory and Applications*, tome 5457 de *Lecture Notes in Computer Science*, pages 23–42. Springer Berlin / Heidelberg, 2009.

[HRV00] Hertrampf, U., S. Reith et H. Vollmer: *A note on closure properties of logspace mod classes*. Inf. Process. Lett., 75:91–93, Août 2000, ISSN 0020-0190.

[HS07] Han, Y. S. et K. Salomaa: *State complexity of union and intersection of finite languages*. Dans *Developments in Language Theory*, tome 4588 de *Lecture Notes in Computer Science*, pages 217–228. Springer Berlin / Heidelberg, 2007.

[JLL76] Jones, N. D., Y. E. Lien et W. T. Laaser: *New problems complete for nondeterministic log space*. Theory of Computing Systems, 10:1–17, 1976, ISSN 1432-4350.

[KLV03] Karakostas, G., R. J. Lipton et A. Viglas: *On the complexity of intersecting finite state automata and* NL *versus* NP . Theoretical Computer Science, 302(1-3):257–274, 2003, ISSN 0304-3975.

[Knu81] Knuth, D.: *The Art Of Computer Programming*, tome 2: Seminumerical Algorithms. Addison-Wesley, 2^{e} édition, 1981, ISBN 9788177583359.

[Koz77] Kozen, D.: *Lower bounds for natural proof systems*. Dans *18th Annual Symposium on Foundations of Computer Science*, pages 254–266, 1977.

[LM88] Luks, E. M. et P. McKenzie: *Parallel algorithms for solvable permutation groups*. J. Comput. Syst. Sci., 37(1):39–62, 1988.

[LP82] Lewis, H. R. et C. H. Papadimitriou: *Symmetric space-bounded computation*. Theoretical Computer Science, 19(2):161–187, 1982, ISSN 0304-3975.

[LR92] Lange, K. J. et P. Rossmanith: *The emptiness problem for intersections of regular languages*. Dans *Mathematical Foundations of Computer Science 1992*, tome 629 de *Lecture Notes in Computer Science*, pages 346–354. Springer Berlin / Heidelberg, 1992.

[Lue75] Lueker, G. S.: *Two* NP*-completes problems in nonnegative integer programming.* Rapport technique 178, Computer Science Laboratory, Princeton University, 1975.

[Luk86] Luks, E. M.: *Parallel algorithms for permutation groups and graph isomorphism.* Dans *FOCS*, pages 292–302. IEEE Computer Society, 1986.

[Luk90] Luks, E. M.: *Lectures on polynomial-time computation in groups.* Technical report. College of Computer Science, Northeastern University, 1990.

[Luo09] Luong, B.: *Fourier Analysis on Finite Abelian Groups.* Birkhäuser, 2009, ISBN 9780817649159.

[MC87] McKenzie, P. et S. A. Cook: *The parallel complexity of abelian permutation group problems.* SIAM J. Comput., 16:880–909, Octobre 1987, ISSN 0097-5397.

[McK84] McKenzie, Pierre: *Parallel complexity and permutation groups.* Thèse de doctorat, University of Toronto, 1984.

[MT90] Martello, S. et P. Toth: *Knapsack problems: algorithms and computer implementations.* J. Wiley & Sons, 1990, ISBN 9780471924203.

[Mul87] Mulmuley, K.: *A fast parallel algorithm to compute the rank of a matrix over an arbitrary field.* Combinatorica, 7:101–104, 1987, ISSN 0209-9683.

[Pin84] Pin, J.E.: *Variétés de langages formels.* Études et recherches en informatique. Masson, 1984, ISBN 9782225802669.

[Rei05] Reingold, O.: *Undirected st-connectivity in log-space.* Dans *Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*, STOC '05, pages 376–385. ACM, 2005, ISBN 1-58113-960-8.

[Sav70] Savitch, W. J.: *Relationships between nondeterministic and deterministic tape complexities*. Journal of Computer and System Sciences, 4(2):177–192, 1970, ISSN 0022-0000.

[Sip79] Sipser, M.: *Lower bounds on the size of sweeping automata*. Dans *Proceedings of the eleventh annual ACM symposium on Theory of computing*, STOC '79, pages 360–364. ACM, 1979.

[Sip06] Sipser, M.: *Introduction to the theory of computation*. Computer Science Series. Thomson Course Technology, 2006, ISBN 9780534950972.

[SM73] Stockmeyer, L. J. et A. R. Meyer: *Word problems requiring exponential time (preliminary report)*. Dans *Proceedings of the fifth annual ACM symposium on Theory of computing*, STOC '73, pages 1–9. ACM, 1973.

[Vij08] Vijayaraghavan, T.C.: *Classifying certain algebraic problems using Logspace counting classes*. Thèse de doctorat, Homi Bhabha National Institute, 2008.

[Vol99] Vollmer, H.: *Introduction to circuit complexity: a uniform approach*. Texts in theoretical computer science. Springer, 1999, ISBN 9783540643104.

[VV04] Vikraman, A. et T. C. Vijayaraghavan: *Abelian permutation group problems and logspace counting classes*. Dans *IEEE Conference on Computational Complexity*, pages 204–214. IEEE Computer Society, 2004, ISBN 0-7695-2120-7.

[War01] Wareham, H. T.: *The parameterized complexity of intersection and composition operations on sets of finite-state automata*. Dans *Implementation and Application of Automata*, tome 2088 de *Lecture Notes in Computer Science*, pages 302–310. Springer Berlin / Heidelberg, 2001, ISBN 978-3-540-42491-8.

[Wie64] Wielandt, H.: *Finite permutation groups*. Academic Press, 1964.