Université de Liège
Faculté des Sciences Appliquées

Représentation à états finis d'ensembles de réels

Année académique
2000 - 2001

Travail de fin d'études présenté par
Sébastien Jodogne

en vue de l'obtention du grade de
licencié en Informatique

Université de Liège
Faculté des Sciences Appliquées

Représentation à états finis d'ensembles de réels

Année académique
2000 - 2001

Travail de fin d'études présenté par
Sébastien Jodogne

en vue de l'obtention du grade de
licencié en Informatique

Remerciements

J'exprime toute ma gratitude envers Monsieur Boigelot qui m'a proposé un sujet riche et passionnant. Je le remercie aussi pour sa gentillesse, son enthousiasme contagieux et sa constante disponibilité.

Je tiens également à remercier Monsieur le Professeur Wolper pour ses idées précieuses, ses explications ainsi que ses nombreux conseils.

Mes plus vifs remerciements vont aussi à tous ceux qui, de près ou de loin, m'ont apporté leur aide, leurs conseils ou leur attention. Je suis notamment reconnaissant à Christof Löding pour ses suggestions quant aux automates faibles, à Monsieur Lejeune pour sa relecture attentive, à Fabien Boniver pour ses idées topologiques et à Philippe Niederkorn pour le temps qu'il m'a consacré. Merci aussi à Sandrine, Cyril et Pierre pour la bonne ambiance de travail qu'ensemble nous avons développée.

Enfin, ce travail n'aurait pu être possible sans le soutien de toute ma famille. Plus particulièrement, toute ma tendresse revient à ma fiancée Delphine pour ses encouragements et sa présence tout au long de ces quatre années d'études.

Table des matières

Table des figures

Première partie

Étude théorique

Chapitre 1

Introduction

Ces dernières décennies, les systèmes informatiques distribués ont connu un grand essor. Ceux-ci sont composés d'un ensemble de dispositifs coopérants qui doivent être coordonnés pour pouvoir travailler ensemble de manière cohérente. Il suffit de songer au fonctionnement des terminaux bancaires, à la gestion d'un central téléphonique ou au développement extensif du réseau Internet pour se rendre compte de l'importance de ces systèmes dans l'informatique industrielle actuelle.

Malheureusement, les problèmes distribués admettent des solutions qui sont souvent contre-intuitives et peuvent souffrir d'erreurs difficiles à déceler et à corriger. Or, à l'instar de l'acquisition et du traitement des paramètres de vol par l'informatique de bord d'un avion, certaines de ces applications doivent assurer un fonctionnement sans faille. Il est donc souhaitable de pouvoir contrôler automatiquement le bon comportement de tels systèmes.

Afin de vérifier que les exécutions d'un système (parallèle ou non) satisfont une propriété, une approche simple, l'*exploration de l'espace des états* [Wes78], consiste à explorer l'ensemble des situations auxquelles le programme peut arriver en partant de son état initial. Il suffit alors de tester pour chacune des configurations atteintes si la propriété est satisfaite.

Un état du système est caractérisé par une position dans le programme et par l'attribution d'une valeur à chacune de ses variables. Ceci est la source d'un problème fondamental inhérent à cette technique : l'espace des états accessibles peut devenir extrêmement grand, voire infini, dès lors qu'une variable a un domaine de définition qui est lui-même grand ou infini — tel est le cas pour les variables entières et réelles. L'exploration ne peut alors être effectuée avec une quantité raisonnable de ressources.

Pour pallier cette limitation, des méthodes dites *symboliques* ont été introduites. Elles consistent à regrouper un ensemble d'états pour les représenter implicitement sous la forme d'un seul et même objet, pouvant être manipulé globalement. Le procédé de représentation symbolique le plus utilisé est le diagramme de décision binaire, *BDD* [Bry92], qui présente toutefois l'inconvénient de n'être applicable qu'à des domaines finis.

C'est pourquoi les diagrammes de décision de nombres (*NDD*) ainsi que les diagrammes de décision de files d'attente (*QDD*) ont été introduits [WB95, BG96]. Ceux-ci peuvent coder respectivement des ensembles d'entiers et des ensembles de contenus de files d'attente, éventuellement infinis, sous la forme d'automates. On bénéficie ainsi de toute la puissance de la théorie bien connue des automates sur mots finis, qui se prêtent parfaitement à un traitement algorithmique et qui possèdent une expressivité élevée. Il devient alors possible par certaines techniques d'analyser en un temps fini des systèmes possédant un espace d'états infini [Boi98].

Le fait de pouvoir vérifier des systèmes qui font intervenir uniquement des variables entières ou des files d'attente est toutefois insuffisant. Par exemple, la plupart des protocoles de communication font intervenir un chronomètre pour gérer les *timeouts* [Tan96]. On voudrait pouvoir appliquer les méthodes de vérification symbolique à de tels systèmes temporisés qui évoluent de façon à la fois discrète et continue au cours du temps. Sous une forme particulière, ceux-ci constituent les *systèmes hybrides* [Hen96], caractérisés par une présence simultanée de variables entières et réelles.

Certaines techniques exactes d'analyse symbolique pour des sous-classes de ces systèmes sont connues [ACH+95], mais ces classes sont fort peu réalistes. Il existe une approche permettant de traiter des systèmes plus complexes, malheureusement elle nécessite une méthode exploitable en pratique pour la représentation d'ensembles combinant à la fois des réels et des entiers. Cette méthode faisait cruellement défaut. En théorie pourtant, une telle représentation implicite était connue. Il s'agit des *Automates sur Vecteurs de Réels*, dénomination abrégée en *RVA* pour *Real Vector Automata* [BBR97]. Les RVA peuvent être vus comme une extension des NDD leur permettant de manipuler les réels en plus des entiers.

On a longtemps craint que cette classe d'automates ne soit pas utilisable en pratique. Ils opèrent en effet sur des mots infinis et leur traitement nécessite des algorithmes nettement moins triviaux que ceux qui s'appliquent aux automates sur mots finis [Var96]. Par exemple, les opérations de complémentation et de différence sont extrêmement difficiles à mettre en œuvre, comme nous le verrons plus loin dans ce travail. De plus, on ne connaît pas d'algorithme efficace de minimisation applicable aux RVA. De ce fait, le nombre d'états des RVA peut croître jusqu'à les rendre non maniables après quelques opérations ; or de nombreux ensembles, même relativement simples, ne peuvent être obtenus que par une longue séquence de calculs. En outre, l'absence de forme minimale implique l'absence de canonicité, qui est très utile quand on veut comparer des automates.

Bien que l'étude des RVA ait été motivée par la vérification, leur champ d'application dépasse très largement ce cadre. Ils peuvent trouver une utilisation dans tous les domaines qui nécessitent la représentation d'ensembles périodiques de réels, comme par exemple certains types de bases de données. En effet, une périodicité ne peut être induite que par la présence de variables entières. Les RVA peuvent également être utilisés dans le cadre de la logique pour représenter tous les modèles d'une formule exprimée dans une certaine théorie logique simple, ce qui permet de la décider.

L'objet de ce mémoire est de prouver que les RVA peuvent être manipulés efficacement, de montrer comment réaliser en pratique ces manipulations et de proposer une implémentation informatique de l'outil théorique qui aura été développé. Tout ceci est possible moyennant une légère restriction sur l'expressivité des RVA. Cette restriction, totalement naturelle, semble être sans conséquence pour l'analyse des systèmes hybrides. Elle rend aussi possible la minimisation des RVA.

Nous verrons également que les premiers résultats indiquent que les RVA présentent un comportement fort semblable à celui des NDD qui, malgré un coût théorique élevé, offrent un très bon comportement moyen, extrêmement réduit par rapport à leur borne maximale de complexité [WB00].

Ce travail se divise en sept chapitres théoriques. Le chapitre 2 expose la manière dont il est possible d'encoder des ensembles d'entiers et de réels sous la forme d'ensembles de mots de longueur infinie (nous préférerons par la suite parler de « langages de mots infinis »).

Le chapitre 3 présente ensuite les *automates de Büchi*, qui sont précisément une extension des automates sur mots finis capables de représenter des langages de mots infinis. En combinant les résultats de ces deux premiers chapitres, nous introduirons au chapitre 4 les RVA et nous préciserons comment manipuler ceux-ci. Ce faisant, nous obtiendrons une technique de représentation implicite d'ensembles d'entiers et de réels.

Nous arriverons alors au constat selon lequel les RVA sont très difficiles à manier en pratique car ils permettent d'exprimer « trop de choses ». Le chapitre 5 montrera que si l'on pose une restriction naturelle sur les ensembles d'entiers et de réels que peuvent représenter les RVA, ces ensembles revêtent une structure très particulière dont la caractérisation va s'énoncer en termes de *topologie*.

Le chapitre 6 étudiera comment cette structure topologique se traduit dans le monde des automates. Ceci nous amènera à envisager plusieurs classes d'automates sur mots infinis, dont celle des *automates faibles* qui épousent parfaitement la structure que nous aurons mise en évidence au chapitre 5. De tels automates sont nettement plus simples à manier que les automates de Büchi. De plus, nous verrons qu'ils admettent la minimisation.

Tous ces résultats mis en commun nous donnent un moyen puissant et efficace pour représenter des ensembles périodiques de réels, comme nous le verrons au chapitre 7. Ces résultats ne sont pas seulement de nature théorique puisqu'une implémentation de la méthode a été intégrée au sein d'un outil de vérification symbolique existant [LASH]. Nous présenterons au chapitre 8 certains résultats pratiques obtenus avec cet outil.

Deux annexes complètent ce travail. La première s'attache à prouver la totalité des théorèmes qui ont été énoncés sans démonstration afin d'alléger l'exposé théorique. La seconde présente l'algorithme de minimisation applicable aux RVA dans le style de la « programmation littéraire » (*literate programming*) [Knu83].

Chapitre 2

Mots infinis et encodage de réels

Ce premier chapitre consiste en une étude des suites infinies de symboles, que nous appellerons *mots infinis*. L'enjeu est de pouvoir représenter des ensembles de réels sous la forme d'ensembles de mots, c'est-à-dire de *langages*. Ces langages pourront être à leur tour matérialisés par des *automates sur mots infinis*, qui seront étudiés au chapitre suivant.

Nous allons employer une approche progressive : on commencera par donner un système de représentation sous la forme de mots pour les naturels, puis pour les entiers, puis pour les réels et enfin pour les vecteurs de réels.

2.1 Mots finis

2.1.1 Tout commence avec des lettres

Il est nécessaire de bien saisir le concept de mot fini avant d'étudier les mots infinis. Ceci est l'objet de cette section, essentiellement adaptée de [Wol91].

Soit un ensemble fini de symboles distincts Σ, que nous nommons *alphabet*. Ces symboles sont fréquemment appelés les *caractères*. Un *mot fini* (ou simplement un *mot*) défini sur Σ est une suite *finie* d'éléments de cet alphabet. La *longueur* d'un mot est le nombre de caractères qui le composent. La longueur du mot w est dénotée par $|w|$.

Exemple. Le mot « *bonjour* » est un mot de longueur 7 sur l'alphabet $\Sigma = \{a,b,\ldots,z\}$ puisque b, o, n, j, o, u et r sont des caractères de cet alphabet. En revanche, le mot « *au revoir* » n'est pas un mot sur cet alphabet, car le caractère d'espacement n'est pas compris dans Σ. D'une façon plus inhabituelle, on peut définir des mots sur l'alphabet $\{0,1,\ldots,9\}$, comme 130, 88 ou 007. □

Nous noterons simplement $w = w_0 w_1 \ldots w_{n-1}$ pour décrire un mot w de longueur n. Ainsi, le i-ème symbole constituant le mot w sera noté w_i, l'énumération commençant à 0. Le *mot vide* est l'unique mot dont la longueur est nulle. Il sera désigné dans la suite par ε.

Comme suggéré plus haut, ce qui nous intéresse est non pas un mot pris isolément, mais un ensemble de mots. Ceci nous mène à poser la définition :

Définition 2.1. Un *langage* est un ensemble (fini ou infini) de mots définis sur le même alphabet. Le langage ne contenant aucun mot est le *langage vide*. Il est désigné par ϕ car il peut être vu comme l'ensemble vide.

Exemple. L'ensemble $\{couteau, fourchette\}$ est un langage à deux mots sur l'alphabet $\{a,b,\ldots,z\}$. L'ensemble de tous les mots de longueur arbitraire et commençant par un c est un langage infini. Un alphabet est un langage fini dont tous les mots sont de longueur 1. □

2.1.2 Combinez et itérez

Étant donnés plusieurs langages, il est possible de les combiner entre eux par un certain nombre d'opérations de base. Supposons que l'on ait défini deux langages L_1 et L_2 au préalable.

Définition 2.2. L'*union* de L_1 et L_2 est le langage contenant tous les mots qui sont soit contenus dans L_1, soit contenus dans L_2. Formellement, $L_1 \cup L_2 = \{w \mid w \in L_1 \vee w \in L_2\}$.

Exemple. L'union des langages à un élément $L_1 = \{couteau\}$ et $L_2 = \{fourchette\}$ est le langage à deux éléments $\{couteau, fourchette\}$. □

Définition 2.3. La *concaténation* (ou *produit*) de L_1 et L_2 est le langage contenant tous les mots formés d'un mot contenu dans L_1 suivi d'un mot de L_2. Formellement, $L_1 \cdot L_2 = \{w \mid (\exists x \in L_1)(\exists y \in L_2)(w = xy)\}$. Cette opération est associative et admet $\{\varepsilon\}$ comme élément neutre.

Exemple. La concaténation des langages $\{bon\}$ et $\{jour\}$ génère le langage $\{bon\} \cdot \{jour\} = \{bonjour\}$. □

On généralise la concaténation de deux mots à la répétition finie d'un mot : étant donnés un mot w et un entier k, le mot w^k est le résultat de la concaténation de k fois le mot w. Par ailleurs, nous omettrons parfois l'opérateur de concaténation de langages « $\cdot$ » quand cela n'est pas source d'ambiguïté.

Définition 2.4. La *fermeture itérative* (ou *fermeture de Kleene*) est l'ensemble de tous les mots formés par une concaténation finie de mots contenus dans L_1. Formellement, $L_1^* = \{w \mid \exists k \geq 0 : w = w^{(1)}w^{(2)}\cdots w^{(k)}$ avec $w^{(i)} \in L_1, \forall i = 1..k\}$. La fermeture itérative d'un langage quelconque admet toujours le mot vide pour élément.

Exemple. $\{a\}^*$ est le langage des mots contenant un nombre arbitraire (mais fini) de caractères a. Σ^* est l'ensemble de tous les mots finis sur l'alphabet Σ, y compris le mot vide. □

Remarque 2.5. Il existe une opération très proche de la fermeture itérative et souvent utilisée. Il s'agit de l'opération « plus » où l'on n'admet pas que $k = 0$. On prouve aisément que $L^* = L^+ \cup \{\varepsilon\}$ et que réciproquement, $L^+ = L \cdot L^*$. □

Nous prendrons dans la suite la liberté d'omettre les accolades pour désigner un langage constitué d'un seul mot. Ainsi, a^* désignera le langage $\{a\}^*$.[1]

L'ensemble des langages que l'on peut obtenir par les trois opérations « $\cup$ », « $\cdot$ » et « * » au départ d'un alphabet sont appelés les *langages réguliers*. Tous les langages ne sont pas réguliers.

Remarque 2.6. Il est également de coutume de définir le *complément* d'un langage, ensemble des mots qui ne sont pas contenus dans ce langage (i.e. $L^c = \{w \in \Sigma^* \mid w \notin L\}$[2]), ainsi que l'*intersection* de deux langages, ensemble des mots contenus à la fois dans chacun de ces deux langages (i.e. $L_1 \cap L_2 = \{w \mid w \in L_1 \wedge w \in L_2\}$). On peut prouver qu'adjoindre ces deux opérations n'augmente pas la classe des langages réguliers. □

2.2 Encodage d'entiers

2.2.1 Notation

Outre la série géométrique [Éti97], nous allons fréquemment utiliser dans ce chapitre la fonction « palier » (*floor*) qui associe à un réel le plus grand entier qui lui est inférieur ou égal :

$$\lfloor x \rfloor = n \Leftrightarrow n \leq x < n + 1 \wedge n \in \mathbb{Z}$$

Ainsi, remarquons que l'on a toujours $0 \leq x - \lfloor x \rfloor < 1$.

2.2.2 Encodage de naturels

Nous cherchons tout d'abord à représenter un nombre naturel $n \in \mathbb{N}$ sous la forme d'un mot fini. La première idée qui vient à l'esprit est d'utiliser la notation décimale classique. Nous allons faire un choix plus général, en laissant la possibilité

1. Ceci nous permet de ne pas définir les *expressions régulières* (ou *rationnelles*) [Wol91] dans le cadre de ce document, car nous aurons uniquement besoin d'appliquer les opérations précitées à des alphabets ou à des langages contenant un seul élément. Ce faisant, nous allons confondre syntaxe et sémantique, sans que cela soit cause d'ambiguïté.

2. Nous prenons pour convention d'écrire S^c pour désigner le complément de l'ensemble S. L'écriture $\overline{S}$ sera utilisée plus loin pour désigner la *fermeture topologique* d'un ensemble.

d'encoder le nombre dans une base de numération quelconque. Pour être complet, précisons que nous allons employer un système de numération *positionnel*.

Soit un entier $r > 1$ que nous appelons *base*. Définissons alors l'alphabet $\Lambda = \{0,1,\ldots,r-1\}$. Les éléments de Λ sont appelés *chiffres*. Tout mot $w \in \Lambda^*$ qui encode le naturel n doit satisfaire la relation d'encodage $n = d_{\mathbb{N}}(w)$, où l'on a défini :

$$\boxed{d_{\mathbb{N}}(w) = \sum_{i=1}^{|w|} w_{|w|-i} \cdot r^{i-1}} \tag{2.1}$$

Exemple. Le nombre 19 peut s'écrire en base deux (notation *binaire*) : $10011 \in \Lambda^*$, où il faut bien comprendre que 10011 est un mot. En effet, $19 = d_{\mathbb{N}}(10011) = 2^4+2+1$. Le nombre 183 s'écrit « 351 » en base 7 car $183 = d_{\mathbb{N}}(351) = 3\cdot 7^2+5\cdot 7+1$. Zéro peut quant à lui s'écrire ε dans toutes les bases. □

Insistons sur la non-unicité de l'encodage d'un naturel. On peut préfixer l'encodage d'autant de zéros qu'on le souhaite sans modifier la valeur du naturel encodé : $d_{\mathbb{N}}(w) = d_{\mathbb{N}}(0^k w)$ avec $k \geq 0$.

Remarque 2.7. En utilisant la progression géométrique, on prouve immédiatement que pour tout mot w de k chiffres, $0 \leq d_{\mathbb{N}}(w) < r^k$. □

Définition 2.8. Le chiffre w_{k-1} de la représentation de longueur k d'un naturel n est appelé *chiffre de poids faible*. Par extension, on appelle les chiffres de grand indice *chiffres de poids faible*. De même, le chiffre w_0 est le *chiffre de poids fort* et par extension, on appelle *chiffres de poids fort* les chiffres de faible indice.

2.2.3 Encodage d'entiers arbitraires

Une première possibilité

Nous allons maintenant tenter de lever la restriction concernant la positivité des nombres. Une première possibilité est d'étendre l'alphabet Λ avec un symbole « – » et de placer ce symbole en tête de l'encodage des nombres négatifs.

Cette façon de faire a plusieurs inconvénients. Tout d'abord, il faut ajouter certaines restrictions telles que : « le symbole "–" ne peut se trouver qu'en tête de mot ». De plus, nous augmentons la taille de l'alphabet par un symbole qui n'a de sens qu'en tête d'un mot. Et enfin, le plus gênant, cela impose de continuellement faire la distinction entre nombres négatifs et nombres positifs. Ainsi, le nombre zéro a *deux encodages minimaux distincts* pour désigner la même entité : « 0 » et « -0 ».

Complément à la base

Une idée plus fructueuse est d'exploiter le complément à la base pour encoder les entiers négatifs :

Définition 2.9. Le *complément à la base r* (en abrégé, *complément à r*) d'un naturel n qu'il est possible d'encoder sur k chiffres en base r, est défini comme le mot formé par les k chiffres de poids faible de l'encodage en base r de $r^k - n$.

Exemple. Le complément à 2 de 19 sur 5 chiffres est l'encodage binaire de $2^5 - 19 = 13$ sur 5 chiffres, soit « 01101 ». Le complément à 5 de 139 sur 6 chiffres est le mot « 443421 ». Le complément à r de 1 sur k chiffres est le mot $(r-1)^k$. □

Remarque 2.10. Si k' est la longueur du plus court encodage de n, alors le complément à r de n sur k chiffres (avec $k \geq k'$) égale le complément à r de n sur k' chiffres préfixé de $k - k'$ répétitions du chiffre $r - 1$. En effet, la remarque 2.7 nous apprend que $n < r^{k'}$ et donc que $r^k - n > r^k - r^{k'} = r^{k'} \cdot (r^{k-k'} - 1)$. Ce dernier naturel s'encode sous la forme du mot : $w = (r-1)^{k-k'}0^{k'}$. Étant donné que n peut être encodé sur k' chiffres, seuls les k' chiffres de poids faible de w sont susceptibles d'être modifiés dans le complément sur k chiffres de n, ce qui prouve l'énoncé. □

Le complément à la base r a été historiquement introduit car il permet de résumer la soustraction de deux nombres à l'addition du premier nombre avec le complément du second [Man88]. Ceci simplifie la conception d'un microprocesseur car on peut ne pas implémenter l'opération de soustraction. Si nous employons le complément à la base pour représenter un nombre négatif, nous bénéficions du même avantage : on peut confondre le traitement des entiers négatifs avec celui des entiers positifs, ce qui nous sera utile dans la suite.

L'encodage retenu

Pour représenter un entier positif, nous utilisons son encodage en base r. Les nombres négatifs peuvent quant à eux être représentés par leur complément à la base r. Rien ne permet *a priori* de distinguer ces deux encodages. C'est pourquoi nous exigeons que chaque représentation soit préfixée de 0 si elle désigne un nombre positif ou de $r - 1$ si elle désigne un nombre négatif. Ce préfixage est automatique si nous imposons que, pour un entier $z \in \mathbb{Z}$ tel que $-r^{k-1} \leq z < r^{k-1}$ où le naturel k est minimal, la représentation de z doit avoir au moins k chiffres (qu'il s'agisse de son propre encodage en base r s'il est positif ou de son complément à la base s'il est négatif).

Un encodage valide contiendra donc nécessairement un chiffre, qui sera soit 0, soit $r - 1$. Nous nommerons ce chiffre, le *chiffre de signe*. L'ensemble $V_{\mathbb{Z}}$ des encodages valides d'entiers sera dès lors le langage de mots finis $V_{\mathbb{Z}} = \{0,r-1\} \cdot \Lambda^*$. Tout mot $w \in V_{\mathbb{Z}}$ qui encode l'entier z doit satisfaire la relation d'encodage $z = d_{\mathbb{Z}}(w)$, où l'on a défini :

$$d_{\mathbb{Z}}(w) = d_{\mathbb{Z}}(\sigma w') = d_{\mathbb{N}}(w') + \begin{cases} 0 & \text{si } \sigma = 0 \\ -r^{|w'|} & \text{si } \sigma = r-1 \end{cases} \tag{2.2}$$

Exemple. L'encodage binaire de l'entier 19 peut être le mot « 010011 », ce qui correspond au plus court encodage du naturel 19 préfixé d'un zéro. Son opposé -19

peut quant à lui s'encoder sous la forme « 101101 ». En effet, le complément à 2 sur 5 chiffres de 19 est $w = 01101$ car $d_{\mathbb{N}}(w) = 2^3 + 2^2 + 1 = 13 = 2^5 - 19$. □

On sait déjà que l'on peut préfixer l'encodage d'un naturel d'un nombre arbitraire de zéros sans changer le naturel encodé. De même, la remarque 2.10 nous a appris que l'on peut préfixer un complément à la base r d'un nombre arbitraire de $r - 1$. On en déduit que l'on peut toujours répéter le chiffre de signe sans modifier l'entier encodé.

2.3 Mots infinis

2.3.1 Dénombrabilité

Avant de nous intéresser aux mots infinis qui nous permettront d'encoder des réels, il peut être important de se remémorer certaines notions concernant l'infini :

Définition 2.11. Un ensemble S est dit *dénombrable* si ses éléments peuvent être mis en bijection avec l'ensemble $\mathbb{N}$ des naturels. La cardinalité d'un ensemble dénombrable (intuitivement, son nombre d'éléments) est dénotée par $\aleph_0$.

2.3.2 Définition

Un mot fini est une suite finie de caractères. On généralise facilement cette notion en disant qu'un *mot infini* est une suite *infinie* de caractères. On donnera alors la valeur ω (infini) à sa longueur, avec la convention :

$$\omega > n \quad \text{pour tout entier } n.^3$$

Exemple. Sur l'alphabet $\{a,b\}$, le mot constitué d'une suite ininterrompue de a est un mot infini, de même que le mot constitué d'un a suivi d'un b, suivi à nouveau d'un a, et ainsi de suite. □

Les caractères d'un mot infini w peuvent donc être numérotés (i.e. indicés) par les naturels. Un mot infini est donc une *énumération* de caractères, chaque caractère étant indexé par un élément de $\mathbb{N}$, ce qui nous mène naturellement à la définition suivante :

Définition 2.12. Un mot infini w sur un alphabet Σ est une fonction $w : \mathbb{N} \mapsto \Sigma$. Le mot w sera souvent représenté sous la forme : $w = w_0 w_1 \dots w_n \dots$, où l'on a posé $w_i = w(i)$ pour tout $i \in \mathbb{N}$.

Exemple. La fonction

$$w(n) = \begin{cases} a & \text{si } n \text{ est pair} \\ b & \text{si } n \text{ est impair} \end{cases}$$

désigne le mot infini où a et b alternent, soit $ababab\dots$. □

3. Le symbole ω rappelle l'*ordinalité* de l'ensemble des entiers.

2.3.3 $\omega-$langages

Définition 2.13. Un *langage de mots infinis*, ou $\omega-$*langage*, est un ensemble de mots infinis définis sur le même alphabet.

L'opération d'union reste valable sur les $\omega-$langages, mais pas les opérations de concaténation ou de fermeture itérative car celles-ci imposeraient l'accolement de deux mots infinis, ce que l'on comprend être impossible. Nous permettrons néanmoins l'opération de concaténation quand elle s'applique à un mot fini que l'on veut faire suivre d'un mot infini.

L'opération de fermeture itérative ne permet pas de générer un $\omega-$langage. En effet, bien qu'elle puisse générer des mots de longueur arbitrairement grande, elle ne génère que des mots finis. Ceci justifie l'introduction d'une nouvelle opération qui sera la généralisation de la fermeture itérative et dont le but est d'obtenir un $\omega-$langage au départ d'un langage de mots finis :

Définition 2.14. L'*itération infinie* d'un langage sur mots finis L est le langage :

$$L^\omega = \{w \mid w = w^{(0)}w^{(1)}\cdots w^{(i)}\cdots \wedge (\forall i \in \mathbb{N})(w^{(i)} \in L \setminus \{\varepsilon\})\}$$

Exemple. Si $L = \{a\}$, L^ω désigne les mots infinis constitués uniquement de a. Si $L = \{a,b,c\}$, L^ω désigne les mots infinis constitués des lettres a, b et c en ordre quelconque. En généralisant à un alphabet quelconque Σ, on nomme Σ^ω l'ensemble de tous les mots infinis définis sur cet alphabet. □

Remarque 2.15. On a imposé que les constituants du mot infini généré soient dans $L \setminus \{\varepsilon\}$ afin de ne pas pouvoir générer le mot vide (en prenant $w_i = \varepsilon$ pour tout i, si $\varepsilon \in L$) ou un mot fini (en prenant par exemple tous les $w_i = \varepsilon$, à l'exception d'un $w_j \in L \setminus \{\varepsilon\}$). On constate ainsi qu'aucun mot fini (y compris le mot vide) n'appartient à Σ^ω. En d'autres termes, Σ^* et Σ^ω sont disjoints[4]. Remarquons enfin que $\phi^\omega = \phi$. □

2.4 Encodage de réels

Pour encoder un réel, on ajoute à sa partie entière, une partie *fractionnaire* dont la valeur appartient à l'intervalle [0,1]. La partie entière s'écrit conformément à ce que nous avons vu plus haut. La partie fractionnaire consiste quant à elle en une suite infinie de chiffres auxquels on associe un poids de plus en plus réduit, qui sera r^{-i} pour le $i-$ème chiffre. Afin d'isoler les parties entière et fractionnaire dans l'encodage d'un réel, on utilise un symbole spécial que nous écrirons $\star$: il s'agit du *séparateur décimal*.

4. On pourrait se demander pourquoi on n'écrit pas Σ^∞. La raison est que cette dernière notation est utilisée dans la littérature pour représenter l'ensemble des mots finis ou infinis sur Σ. Ainsi, $\Sigma^\infty = \Sigma^* \cup \Sigma^\omega$.

Exemple. En base décimale, π sera représenté par le mot infini $w = 3\star14159265\cdots$. Cette écriture signifie : $\pi = 3 + 1/10 + 4/100 + 1/1000 + 5/10000 + 9/100000 + \cdots$. En base 2, π serait écrit $011 \star 0010010000111\cdots$. □

La présence du séparateur décimal impose d'étendre l'alphabet : on pose $\Lambda' = \Lambda \cup \{\star\}$. Un encodage valide débutera nécessairement par l'encodage d'un entier, sera suivi par un et un seul séparateur décimal et s'achèvera en une suite infinie de chiffres. L'ensemble $V_{\mathbb{R}}$ des encodages valides de réels sera dès lors le langage de mots infinis $V_{\mathbb{R}} = V_{\mathbb{Z}} \cdot \star \cdot \Lambda^\omega = \{0,r-1\} \cdot \Lambda^* \cdot \star \cdot \Lambda^\omega$ qui forme un sous-ensemble du $\omega-$langage Λ'^ω.

Pour décrire de façon formelle la manière dont on encode un réel, il faut en outre considérer une nouvelle opération dont l'objet est de retrouver la position du séparateur décimal dans un mot donné. C'est le rôle de la fonction $p : \Sigma^\omega \times \Sigma \mapsto \mathbb{N}$ définie pour un alphabet quelconque Σ, qui à (w,σ) associe l'indice de la première occurrence du caractère σ dans le mot w, c'est-à-dire $\min\{n \mid w(n) = \sigma\}$.

En exploitant ces définitions, le mot $w \in V_{\mathbb{R}}$ encode le réel x si et seulement si il satisfait la relation d'encodage $x = d_{\mathbb{R}}(w)$, où l'on a défini :

$$\boxed{d_{\mathbb{R}}(w) = d_{\mathbb{Z}}(w_0 \cdots w_{k-1}) + \sum_{i=1}^{\infty} \frac{w_{k+i}}{r^i} \quad \text{avec} \quad k = p(w,\star)} \tag{2.3}$$

2.5 Encodages duals

Il est important de remarquer que deux mots distincts dont la longueur de l'encodage de la partie entière a été fixée peuvent représenter le même réel $x \in \mathbb{R}$. On parle d'*encodages duals*. Ceci s'explique par le théorème suivant :

Théorème 2.1. Tout réel $0 \leq x \leq 1$ (i.e. toute partie fractionnaire) a une écriture $w = 0 \star u_0 u_1 \ldots u_n \ldots$ en base r (décodable par la fonction $d_{\mathbb{R}}$), où :

$$\left\{ \begin{array}{lcl} u_0 & = & \lfloor rx \rfloor \\ u_i & = & \lfloor r^{i+1}x \rfloor - r\lfloor r^i x \rfloor \text{ si } i > 0 \end{array} \right. \tag{2.4}$$

Si cette écriture n'est pas unique, il y a exactement deux écritures :

$$\left\{ \begin{array}{lcccccl} w_H & = & w' & \cdot & d & \cdot & 0^\omega \quad \text{(on parle d'}\textit{encodage haut}\text{ de } x\text{)} \\ w_L & = & w' & \cdot & (d-1) & \cdot & (r-1)^\omega \quad \text{(on parle d'}\textit{encodage bas}\text{ de } x\text{)} \end{array} \right. \tag{2.5}$$

avec $d \neq 0$ et w_H qui satisfait les équations (2.4).

Preuve. La preuve est reprise à l'annexe A, section A.1. □

Exemple. $1/2$ peut s'écrire en binaire $0 \star 1(0)^\omega$ ou $0 \star 0(1)^\omega$. De même, $1/5$ peut s'écrire, en base 10, $0 \star 2(0)^\omega$ ou $0 \star 1(9)^\omega$. □

Quand on veut représenter un réel et non plus une partie fractionnaire seule, l'ambiguïté éventuelle de l'écriture de la partie fractionnaire se propage à l'écriture du réel considéré.

Exemple. On retrouve l'équivalence bien connue entre $10 \star 0000\cdots$ et $9 \star 9999\cdots$ en écriture décimale traditionnelle. De même, 13/2 peut s'y écrire $06 \star 5(0)^\omega$ ou $06 \star 4(9)^\omega$. Remarquons enfin que zéro possède deux encodages minimaux dans toute base r : $0 \star (0)^\omega$ et $(r-1) \star (r-1)^\omega$. □

En d'autres termes, la fonction de décodage $d_{\mathbb{R}}$ n'est pas bijective : à une même image (un réel) peut correspondre plusieurs mots. Plus précisément, la fonction est surjective mais pas injective. On avait déjà fait un même type de constat pour la répétition du chiffre de signe. Il y aura ainsi deux causes d'ambiguïté à la représentation d'un réel sous la forme d'un mot :

1. on peut toujours recopier un nombre arbitraire (mais fini) de fois le chiffre de signe (section 2.2.3) ;
2. il peut exister un encodage haut et un encodage bas pour représenter le même réel (théorème 2.1).

Il s'ensuit que tout réel aura une infinité dénombrable d'encodages distincts.

2.6 Encodage de vecteurs de réels

Maintenant que nous pouvons encoder un réel $x \in \mathbb{R}$ sous la forme d'un mot infini, nous désirons pouvoir encoder un *vecteur* de réels $\vec{x} \in \mathbb{R}^n$, où $n > 0$. Imaginons par exemple que l'on cherche à représenter l'encodage haut du vecteur $(13{,}5; 2{,}7)$ en base décimale. Nous allons aligner les séparateurs décimaux des deux éléments du vecteur, quitte à compléter les encodages trop courts par une répétition de leur chiffre de signe, ce qui nous donne :

$$\begin{cases} 013 \star 5(0)^\omega \\ 002 \star 7(0)^\omega \end{cases}$$

où l'on a complété l'encodage de 2,7. Ceci peut être réécrit sous la forme :

$$\left(\begin{array}{c} 0 \\ 0 \end{array}\right)\left(\begin{array}{c} 1 \\ 0 \end{array}\right)\left(\begin{array}{c} 3 \\ 2 \end{array}\right)\left(\begin{array}{c} \star \\ \star \end{array}\right)\left(\begin{array}{c} 5 \\ 7 \end{array}\right)\left(\begin{array}{c} 0 \\ 0 \end{array}\right)^\omega$$

On obtient ainsi un mot infini sur l'alphabet $(\Lambda')^n$, ce que nous cherchions à obtenir[5].

De manière plus générale, pour encoder un vecteur de réels, nous encodons chacune de ses composantes avec des mots dont la longueur de la partie entière est identique. Cette longueur k peut être choisie arbitrairement, à condition qu'elle soit suffisante pour encoder la composante dont la valeur absolue est la plus élevée.

5. On a utilisé la notation $S^n = \underbrace{S \times S \times \cdots \times S}_{n}$.

Exemple. Le vecteur $\vec{x} = (13{,}5; 2{,}7)$ déjà considéré peut s'écrire en binaire $(0;0)(1;0)(1;0)(0;1)(1;0)(\star;\star)(1;1)(0;0)(0;1)\ldots$. On peut encoder le même vecteur en répétant le $(0;0)$ initial correspondant au chiffre de signe. Toutefois, on ne peut réduire la longueur de la partie entière, sinon 13,5 ne pourra être encodé. En effet, l'encodage de longueur minimale de l'entier 13 est 01101 : la partie entière doit donc posséder au moins 5 chiffres. □

Puisque le réalignement impose que les séparateurs décimaux soient lus simultanément, on peut remplacer le symbole $(\star;\star)$ par un seul symbole $\star$. Finalement, l'encodage retenu représente des vecteurs de réels par un mot infini sur l'alphabet :

$$\Sigma = \Lambda^n \cup \{\star\} = \{0,\ldots,r-1\}^n \cup \{\star\}$$

Exemple. On écrirait $\vec{x}$ en base décimale : $(0;0)(1;0)(3;2)\star(5;7)(0;0)^\omega$. □

Nous allons formaliser la relation d'encodage pour un vecteur de réels. Pour cela, on définit une fonction de projection $\pi : \Sigma^\omega \times \{1,\ldots,n\} \mapsto (\Lambda')^\omega$ qui associe à (w,i) la $i-$ème composante du mot w, le symbole $\star$ n'étant pas affecté. Le mot $w \in \Sigma^\omega$ encode alors le vecteur $\vec{x} \in \mathbb{R}^n$ si et seulement si $\vec{x} = d(w)$ où l'on a défini :

$$\boxed{d(w) = \vec{y} \text{ si } y_i = d_{\mathbb{R}}(\pi(w,i)), \text{ pour tout } i = 1,\ldots,n} \tag{2.6}$$

2.7 Encodages valides et invalides

Tous les mots de Σ^ω ne sont pas décodables par la fonction d. Ce sont les *encodages non valides*. Les encodages non valides se classent en trois catégories : les mots ne commençant pas par des chiffres de signe valides (classe NV_s, avec « NV » utilisé pour « non valide » et « s » pour « start »), les mots ne comportant pas de séparateur décimal (classe NV_0) et les mots comportant plus d'un séparateur décimal (classe NV_+). Les mots correspondant à des encodages valides et auxquels on peut appliquer d, forment la classe valide V.

Plus formellement, on a :

1. $V = \{0,r-1\}^n \cdot (\Lambda^n)^* \cdot \star \cdot (\Lambda^n)^\omega$,
2. $NV_s = (\Sigma \setminus \{0,r-1\}^n) \cdot \Sigma^\omega$,
3. $NV_0 = \{0,r-1\}^n \cdot (\Lambda^n)^\omega$,
4. $NV_+ = \{0,r-1\}^n \cdot (\Lambda^n)^* \cdot \star \cdot (\Lambda^n)^* \cdot \star \cdot \Sigma^\omega$,

Ces classes prises deux à deux sont disjointes et leur réunion forme le langage Σ^ω : il s'agit donc d'un partitionnement de Σ^ω.

Remarque 2.16. On considère que les mots qui contiennent zéro ou plus d'un séparateur décimal et qui ne commencent pas par un chiffre de signe valide, appartiennent à la classe NV_s. □

Chapitre 3

Automates de Büchi

Un *automate* est un formalisme permettant de décrire de façon *finie* un langage. On dit que l'automate *accepte* ce langage.

Les automates sur des objets infinis ont été introduits au début des années 60. Büchi en fut le pionnier [Büc62]. Il les utilisa en vue de développer une méthode de décision pour une logique mathématique. Pour ce faire, il a prouvé que l'ensemble des modèles de toute formule de cette logique peut s'exprimer sous la forme d'un automate sur lequel il est bien plus simple de raisonner.

Plus précisément, Büchi a introduit les *automates sur mots infinis*, également appelés $\omega-$automates pour faire référence à l'ordinalité de l'ensemble infini de symboles qui composent un mot infini. De tels automates sont une description formelle d'un langage de Σ^ω. Ceci se rapproche des classiques automates finis (introduits par exemple dans [HU79, Wol91]), qui sont une description d'un langage de Σ^*.

Contrairement à ces derniers, les $\omega-$automates ne peuvent pas être vus comme une description analytique d'un langage : en effet, pour répondre si un mot appartient ou non au langage, il faudrait analyser le mot en entier, ce qui n'est faisable qu'en un temps infini. Par conséquent, un $\omega-$automate ne représente *pas* une procédure effective (un programme) pour reconnaître un mot d'un $\omega-$langage.

Leur utilité utilité en informatique existe néanmoins, car les automates sont un formalisme à la frontière entre expressivité et manipulation effective : ils permettent d'exprimer sous une forme compacte de nombreuses choses, tout en étant adaptés à un traitement algorithmique. Aujourd'hui par exemple, les automates sur objets infinis sont utilisés pour la vérification de programmes qui ne se terminent pas. Dans ce document, nous allons utiliser les $\omega-$automates pour représenter des ensembles de vecteurs de réels, encodés sous la forme d'un mot infini conformément au chapitre précédent [1].

1. Les ouvrages [Tho90, Var96, Löd97, PP01] offrent un plus large aperçu des résultats existants concernant les automates sur des objets infinis et constituent à ce titre d'excellentes références.

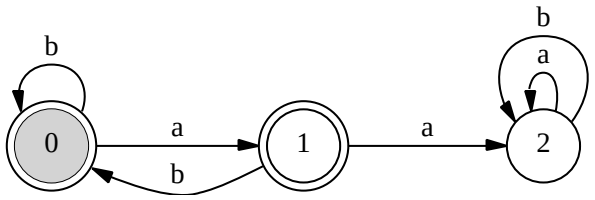


FIG. 3.1 – *Un automate.*

3.1 Automates sur mots finis

3.1.1 Définitions

Il est difficile de parler d'automates sur mots infinis sans avoir préalablement assimilé ceux sur mots finis, même si ceux-ci ne nous seront plus utiles dans le reste de cet ouvrage. Nous rappelons donc ici brièvement la définition d'un automate sur mots finis. Cette section n'a pas la prétention de se substituer à un ouvrage d'initiation aux automates. Elle n'est présente que pour fixer les idées et les notations.

Définition 3.1. Un automate fini non déterministe est défini par un quintuplet $A = (Q, \Sigma, \delta, S_0, F)$ où :

- Q est un ensemble fini d'*états*,
- Σ est l'alphabet sur lequel est défini le langage reconnu par l'automate,
- $\delta : Q \times \Sigma \mapsto 2^Q$ est la *fonction de transition*[2],
- $S_0 \subseteq Q$ est l'ensemble des *états initiaux*,
- $F \subseteq Q$ est l'ensemble des *états accepteurs* ou *terminaux*.

Remarque 3.2. Ceci est n'est pas la seule définition existante. Par exemple, on peut définir une *relation* de transition $\Delta \subseteq (Q \times \Sigma^* \times Q)$ en lieu et place de δ. □

Il est possible de représenter un automate sous la forme d'un graphe. Dans ce cas, les nœuds du graphe sont les états de l'automate, les états initiaux sont grisés et les états accepteurs sont entourés d'un double cercle. La fonction de transition est quant à elle matérialisée par un ensemble d'arcs $q_i \xrightarrow{a} q_j$ tels que $q_j \in \delta(q_i,a)$.

Exemple. La représentation sagittale d'un certain automate défini sur l'alphabet $\Sigma = \{a,b\}$ est présentée à la figure 3.1. □

3.1.2 Langage reconnu

Pour définir le langage $L(A)$ de Σ^* reconnu par l'automate, nous n'allons pas utiliser la coutumière notion de configuration, mais nous allons introduire celle de

2. 2^S représente l'ensemble des *parties* de S, à savoir l'ensemble des sous-ensembles de S.

course qui est plus adaptée à l'étude des automates sur mots infinis.

Considérons un mot fini, par exemple $w = babaab$. Définir une course sur ce mot revient à intercaler des états de l'automate entre les lettres du mot, en respectant deux règles simples :

(i) l'état à gauche du mot doit être un état initial ;

(ii) pour un état q_{i+1} intercalé après une lettre w_i elle-même précédée d'un état q_i, il doit exister un arc $q_i \xrightarrow{w_i} q_{i+1}$ dans le graphe de l'automate.

Exemple. Pour le mot *babaab* et l'automate de la figure 3.1, on obtient l'intercalation suivante :

Mot :		b		a		b		a		a		b	
Course :	$\boxed{0}$	$\longrightarrow$	0	$\longrightarrow$	1	$\longrightarrow$	0	$\longrightarrow$	1	$\longrightarrow$	2	$\longrightarrow$	$\boxed{2}$

Remarquons qu'il s'agit de la seule possibilité : c'est une manifestation de ce que nous appellerons plus loin le *déterminisme*. □

Une course peut ainsi se voir comme un *étiquetage* de caractères d'un mot par les états de l'automate de manière compatible avec les états initiaux et la fonction de transition. Plus formellement, on pose :

Définition 3.3. On appelle *course* de l'automate A sur un mot w de longueur n, une fonction $\rho : \{0, \ldots, n\} \mapsto Q$ telle que :

1. $\rho(0) \in S_0$,
2. $\forall 0 \leq i < n$, $\rho(i+1) \in \delta(\rho(i), w_i)$.

Étant donné un mot w, il peut y avoir zéro, une ou plusieurs courses. On choisit de dire que le mot est accepté quand il *existe* une course qui s'achève dans un état terminal :

Définition 3.4. Une course ρ sur w est dite *acceptrice* si $\rho(|w|) \in F$. Le mot w appartient au langage $L(A)$ si et seulement si il existe au moins une course acceptrice de A sur w.

Exemple. Dans l'automate décrit plus haut, le mot *babaab* n'est pas accepté car $2 \notin F$. Par contre, *ab*, *bbbbabab* et *abbbabbbabba* sont acceptés. Pour le voir, il suffit d'envisager tous les chemins du graphe partant d'un état initial et étiquetés par le mot étudié (dans le cas présent, il n'y en a qu'un), puis de détecter si l'état auquel on arrive est terminal. En fait, l'automate de la figure 3.1 reconnaît les mots ne contenant pas deux a consécutifs. □

3.1.3 Déterminisme

L'existence de courses multiples est permise par la présence de multiples états initiaux ainsi que par le fait que la fonction de transition définit un *ensemble* d'états

successeurs pour un même état et pour une même lettre. On dira qu'un automate fini est *déterministe* si δ est une fonction totale de la forme $Q \times \Sigma \mapsto Q$ et si $S_0 = \{s_0\}$, ce qui assure l'existence et l'unicité d'une course pour tout mot w.

Remarque 3.5. La nouvelle forme de δ impose l'absence de cul-de-sac dans le graphe : δ doit être définie pour tout état de l'automate et pour tout caractère de l'alphabet. Ceci n'est nullement restrictif. En effet, si l'on veut simuler une fonction δ partielle pour un caractère a dans un état p, il suffit d'ajouter un état « puits » (*gap*) non accepteur q à Q et de modifier δ de telle façon que $\delta(p,a) = \{q\}$ et que $\delta(q,\sigma) = \{q\}$ pour tout $\sigma \in \Sigma$. On parle de *compléter* l'automate. Par exemple, dans la figure 3.1, l'état 2 peut se concevoir comme un état puits. □

Remarque 3.6. Il est possible de montrer que tout langage reconnu par un automate fini non déterministe, l'est aussi par une automate fini déterministe [HU79, Wol91]. □

Pour un automate déterministe, nous allons prendre la liberté dans cet ouvrage d'étendre la fonction de transition δ de la manière suivante pour un mot $w \in \Sigma^*$:

$$\delta(q,w) = \begin{cases} q & \text{si } w = \varepsilon \\ \delta(q,w_0) & \text{si } |w| = 1 \\ \delta(\delta(q,a),w') & \text{si } w = a \cdot w' \text{ avec } a \in \Sigma \end{cases}$$

3.2 Automates de Büchi

Nous pouvons maintenant introduire les automates sur mots infinis. Un tel automate a la même structure qu'un automate sur mots finis, à savoir un quintuplet $A = (Q, \Sigma, \delta, S_0, F)$. On généralise très facilement la notion de course :

Définition 3.7. On appelle *course* de l'automate A sur un mot infini w, une fonction $\rho : \mathbb{N} \mapsto Q$ d'étiquetage telle que :

1. $\rho(0) \in S_0$,
2. $\forall 0 \leq i,\ \rho(i+1) \in \delta(\rho(i),w_i)$.

Malheureusement, la généralisation de la notion de course acceptrice n'est pas aussi directe. En effet, puisqu'il n'y a pas de « dernier caractère », on ne peut pas poser de condition sur le dernier élément de l'étiquetage ! L'idée de Büchi fut de considérer l'ensemble des états $\mathrm{Inf}(\rho)$ qui reviennent infiniment souvent dans la course ρ : on considère en quelque sorte que ce sont les états significatifs. Formellement, on définit :

$$\mathrm{Inf}(\rho) = \{p \in Q \mid |\{n \in \mathbb{N} : \rho(n) = p\}| = \aleph_0\}$$

où l'on a désigné par $|E|$ la cardinalité de l'ensemble E. On considère alors qu'une course est acceptrice si elle contient un nombre infini de fois un élément de F :

Définition 3.8. Une course ρ est acceptrice si $\mathrm{Inf}(\rho) \cap F \neq \phi$. Un mot w appartient au langage de Σ^ω accepté par A si il existe une course acceptrice ρ de A sur w. On écrit $w \in L_\omega(A)$.[3]

Les automates qui exploitent cette dernière condition d'acceptation sont appelés *automates de Büchi*. Nous étudierons plus loin d'autres conditions d'acceptation.

Exemple. Si l'on considère l'automate de la figure 3.1 comme un automate de Büchi, le mot b^ω est accepté puisque la course ρ correspondante est $0 \to 0 \to 0 \cdots$, que $\mathrm{Inf}(\rho) = \{0\}$ et que l'état 0 est accepteur. De même, $(a \cdot b)^\omega$ est accepté car $\mathrm{Inf}(\rho) = \{0,1\}$. Par contre, a^ω n'est pas accepté car $\mathrm{Inf}(\rho) = \{2\}$ et $2 \notin F$. □

Remarque 3.9. On préfère parler uniquement d'état accepteur et de langage accepté dans le cadre des $\omega-$automates, plutôt que d'état terminal et de langage reconnu. En effet, comme nous l'avons déjà mentionné, un $\omega-$automate n'est pas un processus analytique pour reconnaître un mot du langage car un tel processus nécessiterait, dans sa réalisation directe, un temps infini. Il n'y a dès lors pas de sens à parler de « reconnaissance » d'un $\omega-$langage. □

3.3 Manipulation d'automates de Büchi

Dans cette section, nous allons passer en revue les opérations qu'il est possible de réaliser sur les langages acceptés par des automates de Büchi. Un avantage des automates sur d'autres systèmes de représentation de $\omega-$langages, est que les algorithmes correspondant à ces opérations sont relativement directs. Nous allons néanmoins nous heurter à un écueil : une des opérations est trop complexe pour être utilisée en pratique.

3.3.1 Union

Étant donnés deux automates de Büchi *complétés* (éventuellement non déterministes) $A_1 = (Q_1, \Sigma, \delta_1, S_1, F_1)$ et $A_2 = (Q_2, \Sigma, \delta_2, S_2, F_2)$, définis sur un même alphabet Σ, il est possible de calculer un automate de Büchi $A = (Q, \Sigma, \delta, S_0, F)$ qui accepte le $\omega-$langage $L_\omega(A_1) \cup L_\omega(A_2)$. La même construction pourra d'ailleurs être appliquée aux automates sur mots finis.

L'idée est de réaliser une *construction par produit* : on considère qu'un état de A se compose d'un état de A_1 (disons p_1) et d'un état de A_2 (disons p_2). Quand

3. Cette notation n'est pas anodine : elle permet en effet de faire la distinction entre le langage sur mots finis $L(A)$ obtenu en considérant A comme automate sur mots finis et le langage sur mots infinis $L_\omega(A)$ obtenu en considérant A comme automate de Büchi. Cette distinction est nécessaire car ces deux types d'automates partagent la même structure.

- $Q = Q_1 \times Q_2$,
- $(q_1,q_2) \in \delta((p_1,p_2),a)$ si et seulement si $q_1 \in \delta_1(p_1,a)$ et $q_2 \in \delta_2(p_2,a)$,
- $S_0 = S_1 \times S_2$,
- $F = (F_1 \times Q_2) \cup (Q_1 \times F_2)$.

FIG. 3.2 – *Construction de l'union pour deux automates complétés.*

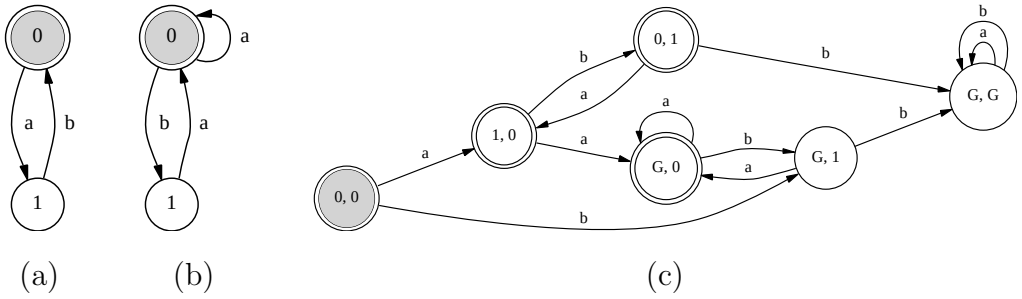


FIG. 3.3 – *Union de deux automates de Büchi : (a) A_1, (b) A_2 et (c) $A_1 \cup A_2$.*

on emprunte dans A une transition étiquetée par le caractère a, on arrive à un état de A composé d'un état q_1 de A_1 et d'un état q_2 de A_2 tels que q_1 et q_2 soient respectivement les successeurs par a de p_1 et de p_2. En d'autres termes, on effectue une exploration simultanée de A_1 et de A_2. Si en outre q_1 ou q_2 est accepteur, l'état de A correspondant sera marqué comme accepteur. L'automate A peut dès lors être construit comme proposé à la figure 3.2.

Exemple. La figure 3.3 présente un exemple d'application de cette construction. Un état «G» correspond à un état puits qui a été introduit pour compléter les automates. Remarquons que l'état «G, G» n'est jamais nécessaire : quand les deux automates sont tombés dans le puits, il n'y a plus moyen d'en sortir pour accepter un mot. □

Remarque 3.10. En pratique, on peut implémenter les constructions sur les automates par une *recherche en profondeur d'abord* : on part des états de S_0 et on progresse en envisageant toutes les transitions possibles. Ceci permet de ne créer que les états strictement nécessaires, c'est-à-dire accessibles depuis les états initiaux. □

La complexité d'une construction sur des automates peut se mesurer par le nombre d'états de l'automate résultant. Pour la construction d'union, A aura $\mathcal{O}(|Q_1| \cdot |Q_2|)$ états. Nous prouvons maintenant l'adéquation de la construction :

Proposition 3.1. Si on applique cette construction, $L_\omega(A) = L_\omega(A_1) \cup L_\omega(A_2)$.

Preuve. Soit $w \in L_\omega(A_1)$. Il existe alors une course acceptrice $\rho_1 = q_{1,0} \xrightarrow{w_0} q_{1,1} \xrightarrow{w_1} q_{1,2} \cdots$ de A_1 sur w. Il s'agit maintenant de compléter ρ_1 pour qu'elle devienne une course ρ de A sur w. On pose $\rho(0) = (\rho_1(0),\ s_2)$ avec $s_2 \in S_2$ et $\rho(k) = (\rho_1(k),\ q_{2,k})$ pour tout $k > 0$ avec $q_{2,k} \in \delta(q_{2,k-1},\ w_{k-1})$, ce qui existe toujours

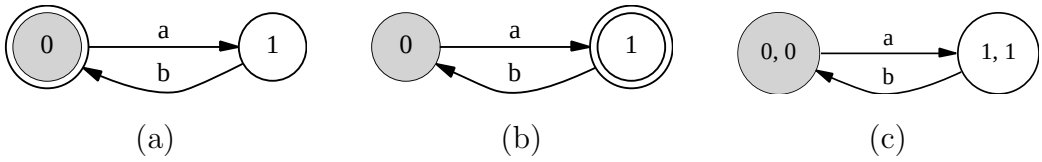


Fig. 3.4 – *Problème pour l'intersection de deux automates de Büchi.*

car on a supposé A_2 complet. Si $\rho_1(n)$ est un état accepteur, il en est de même pour $\rho(n)$ par définition de F. De ce fait, puisque ρ_1 contient par hypothèse un nombre infini d'états accepteurs, il en est de même pour ρ et ainsi $w \in L_\omega(A)$. Le même raisonnement tient si $w \in L_\omega(A_2)$; on a finalement : $L_\omega(A_1) \cup L_\omega(A_2) \subseteq L_\omega(A)$.

Réciproquement, soit $w \in L_\omega(A)$. Il existe alors une course acceptrice $\rho = (q_{1,0},\, q_{2,0}) \xrightarrow{w_0} (q_{1,1},\, q_{2,1}) \xrightarrow{w_1} (q_{1,2},\, q_{2,2}) \xrightarrow{w_2} (q_{1,3},\, q_{2,3}) \cdots$ de A sur w. Par projection, ρ définit une course ρ_1 de A_1 et une course ρ_2 de A_2 sur w :

$$\rho_1 = q_{1,0} \xrightarrow{w_0} q_{1,1} \xrightarrow{w_1} q_{1,2} \xrightarrow{w_2} q_{1,3} \cdots$$

$$\rho_2 = q_{2,0} \xrightarrow{w_0} q_{2,1} \xrightarrow{w_1} q_{2,2} \xrightarrow{w_2} q_{2,3} \cdots$$

Puisque ρ est acceptrice, elle passe infiniment souvent par un état de F. Il existe donc une suite infinie de (n_k) tels que soit $\rho_1(n_k) \in F_1$, soit $\rho_2(n_k) \in F_2$. Appelons (l_k) la sous-suite des (n_k) tels que $\rho_1(l_k) \in F_1$.

Supposons que (l_k) soit infinie. Cela implique que ρ_1 passe par un nombre infini d'états accepteurs pour A_1 et donc que $w \in L_\omega(A_1)$. Supposons *a contrario* que (l_k) soit finie. Puisque (n_k) est infinie, ρ doit nécessairement passer par un nombre infini d'états $(q_{1,k},\, q_{2,k})$ tels que $q_{2,k} \in F_2$. Par suite, ρ_2 passe par un nombre infini d'états accepteurs pour A_2 et ainsi $w \in L_\omega(A_2)$. Au final, on a $w \in L_\omega(A_1) \cup L_\omega(A_2)$ et donc $L_\omega(A) \subseteq L_\omega(A_1) \cup L_\omega(A_2)$. □

3.3.2 Intersection

Nous voulons maintenant construire un automate A tel que $L_\omega(A) = L_\omega(A_1) \cap L_\omega(A_2)$. Les automates A_1 et A_2 ne doivent plus nécessairement être complets. La tentation est forte de reprendre la construction de l'union en ne changeant que la définition de F que l'on pose égal à $F_1 \times F_2$. Ceci est correct dans le cadre des automates sur mots finis, mais ne l'est pas pour les automates de Büchi.

Exemple. La figure 3.4 met en évidence un problème si on prend $F = F_1 \times F_2$ pour calculer l'intersection des automates (a) et (b) : l'automate résultant (c) n'accepte aucun mot, alors qu'il devrait accepter le même $\omega-$langage que (a) et (b). □

Intuitivement, le problème vient du fait qu'il n'est pas nécessaire de visiter un état de F_1 en même temps qu'un état de F_2 : il peut exister un *déphasage* entre les deux visites car on raisonne sur le *nombre d'occurrences* d'un état accepteur dans une course et non plus sur l'occurrence d'un état accepteur *à un endroit fixé* dans une course, comme pour les mots finis. Une solution est d'ajouter à chaque état un

- $Q = Q_1 \times Q_2 \times \{f_1, \neg f_1\}$,
- $(q_1,q_2,t) \in \delta((p_1,p_2,s),a)$ si et seulement si $q_1 \in \delta_1(p_1,a)$ et $q_2 \in \delta_2(p_2,a)$, auquel cas $t = f_1$ si et seulement si $(q_1 \in F_1) \vee (s = f_1 \wedge p_2 \notin F_2)$ (sinon, $t = \neg f_1$),
- $S_0 = S_1 \times S_2 \times \{f_1\}$,
- $F = Q_1 \times F_2 \times \{f_1\}$.

Fig. 3.5 – *Construction de l'intersection.*

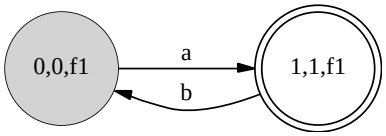


Fig. 3.6 – *Construction correcte.*

« drapeau » qui retient si le chemin courant dans le graphe de l'automate a visité un état de F_1 depuis sa dernière visite dans F_2. L'état sera accepteur si, simultanément, le drapeau est dressé et l'état courant de A_2 accepteur. Formellement, on définit A comme présenté à la figure 3.5.

Le drapeau est dressé si la troisième composante de l'état courant est f_1. Si on arrive dans un état de F_1, on dresse systématiquement le drapeau. Si on vient de visiter un état de F_2, on abaisse systématiquement le drapeau : c'est l'objet du second terme de la disjonction. Sinon, on laisse le drapeau tel qu'il est. Le nombre d'états de l'automate est $\mathcal{O}(|Q_1| \cdot |Q_2|)$, ce qui est identique à la construction de l'union.

Exemple. Si on applique cette construction aux automates de la figure 3.4, on obtient l'automate de la figure 3.6, qui accepte cette fois le bon langage. □

Proposition 3.2. Si on applique la construction de l'intersection, alors $L_\omega(A) = L_\omega(A_1) \cap L_\omega(A_2)$.

Preuve. Soit $w \in L_\omega(A_1) \cap L_\omega(A_2)$. Il existe dès lors deux courses acceptrices ρ_1 de A_1 et ρ_2 de A_2 sur w, disons :

$$\rho_1 = q_{1,0} \xrightarrow{w_0} q_{1,1} \xrightarrow{w_1} q_{1,2} \xrightarrow{w_2} q_{1,3} \cdots$$

$$\rho_2 = q_{2,0} \xrightarrow{w_0} q_{2,1} \xrightarrow{w_2} q_{2,2} \xrightarrow{w_2} q_{2,3} \cdots$$

Ces deux courses induisent une course ρ de A sur w, où la troisième composante de chaque élément de ρ est définie en accord avec la construction. Imaginons que $q_{2,n} \in F_2$. Puisque $w \in L_\omega(A_1)$, il existe un nombre *infini* d'indices n' tels que $q_{1,n'} \in F_1$. On peut dès lors choisir n' de manière telle que $n' \geq n$. La fonction de transition impose alors que la troisième composante de $\rho(n')$ soit f_1.

Le même raisonnement nous permet de sélectionner le plus petit indice $n'' > n'$ tel que $q_{2,n''} \in F_2$. La troisième composante de $\rho(n'')$ vaut toujours f_1 car on n'a préalablement rencontré aucun état de F_2. On arrive par conséquent à un état de la forme $(q_{1,n''}, q_{2,n''}, f_1)$ qui est accepteur pour A. Ainsi, tout état $q_{2,n} \in F_2$ dans la course ρ_2 sera fatalement suivi d'un état accepteur dans la course ρ. Puisque $w \in L_\omega(A_2)$, ceci arrive un nombre infini de fois et il y aura un nombre infini d'états accepteurs dans ρ. Finalement, $w \in L_\omega(A)$.

Réciproquement, soient $w \in L_\omega(A)$ et une course acceptrice $\rho = (q_{1,0}, q_{2,0}, g_0) \xrightarrow{w_0} (q_{1,1}, q_{2,1}, g_1) \xrightarrow{w_1} (q_{1,2}, q_{2,2}, g_2) \cdots$. On peut définir les courses ρ_1 et ρ_2 comme à la démonstration 3.1. ρ passe infiniment souvent par un état de F. Donc, il existe une suite infinie (n_k) d'indices tels que $q_{2,n_k} \in F_2$ et $g_{n_k} = f_1$. La première condition implique immédiatement que ρ_2 est une course acceptrice de A_2 et donc que $w \in L_\omega(A_2)$. De même, la course ρ_1 est acceptrice car la condition $g_{n_k} = f_1$ indique que, depuis la dernière visite à un état de F_2, au moins un état de F_1 a été visité. On conclut que $w \in L_\omega(A_1) \cap L_\omega(A_2)$. □

3.3.3 Produit cartésien

Le *produit cartésien* de deux $\omega-$langages L_1 et L_2 définis respectivement sur les alphabets Σ_1 et Σ_2, est défini comme suit :

$$L_1 \times L_2 = \{(w_0,w'_0)(w_1,w'_1)(w_2,w'_2)\cdots \mid w = w_0w_1w_2\cdots \in L_1 \wedge w' = w'_0w'_1w'_2\cdots \in L_2\}$$

Il s'agit d'un $\omega-$langage défini sur l'alphabet $\Sigma_1 \times \Sigma_2$.

Remarque 3.11. L'intersection peut se voir comme un cas particulier du produit cartésien, où on impose que $w_i = w'_i$ pour tout $i \geq 0$ et où on ne retient qu'une seule copie du caractère de chaque couple. □

Nous cherchons maintenant à construire un automate A qui accepte le $\omega-$langage $L_\omega(A_1) \times L_\omega(A_2)$. Le lien entre intersection et produit cartésien laisse présager que l'on ait le même problème que pour l'intersection. C'est effectivement le cas et la construction du produit cartésien est donnée à la figure 3.7 de la page 24.[4]

3.3.4 Projection

Le produit cartésien augmente la taille de l'alphabet. Il est souhaitable de pouvoir la réduire. C'est l'objet de l'opération de *projection*. Soit un $\omega-$langage L défini sur

4. Pour ne pas alourdir l'exposé, on ne donnera pas de preuve de la construction. Celle-ci se dérive très facilement de la preuve pour l'intersection. Il est également possible de la démontrer en utilisant un type plus général d'$\omega-$automates, à savoir les *automates de Büchi généralisés*, qui, étant donnés *plusieurs* ensembles d'états accepteurs, imposent qu'une course acceptrice visite un nombre infini de fois *chacun* de ces ensembles [Wol98].

- $Q = Q_1 \times Q_2 \times \{f_1, \neg f_1\}$,
- $\Sigma = \Sigma_1 \times \Sigma_2$,
- $(q_1,q_2,t) \in \delta((p_1,p_2,s),(a,a'))$ si et seulement si $q_1 \in \delta_1(p_1,a)$ et $q_2 \in \delta_2(p_2,a')$, auquel cas $t = f_1$ si et seulement si $(q_1 \in F_1) \vee (s = f_1 \wedge p_2 \notin F_2)$,
- $S_0 = S_1 \times S_2 \times \{f_1\}$,
- $F = Q_1 \times F_2 \times \{f_1\}$.

Fig. 3.7 – *Construction du produit cartésien.*

un alphabet $\Sigma = \Sigma_1 \times \Sigma_2 \times \cdots \times \Sigma_n$. La *projection* de L sur toutes ses composantes à l'exception de la $i-$ème est définie ainsi :

$$L\Big|_{\neq\sigma_i} = \{(\sigma_1,\ldots,\sigma_{i-1},\sigma_{i+1},\ldots,\sigma_n) \mid (\exists\sigma_i \in \Sigma_i)(\sigma_1,\ldots,\sigma_{i-1},\sigma_i,\sigma_{i+1},\ldots,\sigma_n) \in L\}$$

La construction pour calculer la projection du langage $L_\omega(A)$ accepté par un automate A est immédiate : il suffit de retirer la $i-$ème composante de l'étiquette de chaque transition de l'automate.

3.3.5 Complémentation et différence

Étant donné un automate de Büchi A, nous cherchons à construire un automate A' tel que $L_\omega(A') = (L_\omega(A))^c$. Remarquons d'emblée que pour calculer la différence entre les langages acceptés par deux automates de Büchi A_1 et A_2 définis sur un même alphabet, il suffit de se ramener aux constructions de complémentation et d'intersection en exploitant la relation : $L_\omega(A_1) \setminus L_\omega(A_2) = L_\omega(A_1) \cap (L_\omega(A_2))^c$.

Classiquement, pour complémenter le langage reconnu par un automate sur mots finis A, on procède en trois étapes : (i) on déterminise A, (ii) on complète A et (iii) on inverse les états accepteurs et non accepteurs (i.e. $F := Q \setminus F$). La déterminisation est une étape nécessaire, comme le montre la figure 3.8 : si l'on se contentait d'inverser les états accepteurs de l'automate complet sur mots finis (a) défini sur $\Sigma = \{a\}$, on obtiendrait l'automate (b) qui accepte exactement le même langage en plus du mot vide ε.[5]

Cette construction ne s'étend pas aux automates de Büchi : la complémentation des $\omega-$automates est un problème très difficile. À supposer même que l'on puisse déterminiser ce type d'automates, appliquer la règle $F := Q \setminus F$ ne serait pas suffisant. La figure 3.9 montre par exemple qu'appliquer cette règle à l'automate de Büchi déterministe (a) donne pour résultat l'automate (b) qui accepte toujours le même langage. Ne pas passer un nombre infini de fois dans F n'est en effet pas équivalent à passer un nombre infini de fois dans $Q \setminus F$: une course peut très bien

5. Pour que l'on puisse simplement inverser le statut accepteur des états, il faudrait pouvoir convertir la quantification existentielle présente dans la définition d'une course acceptrice par une quantification universelle. C'est l'idée intuitive qu'exploitent les *automates alternants* [MS87].

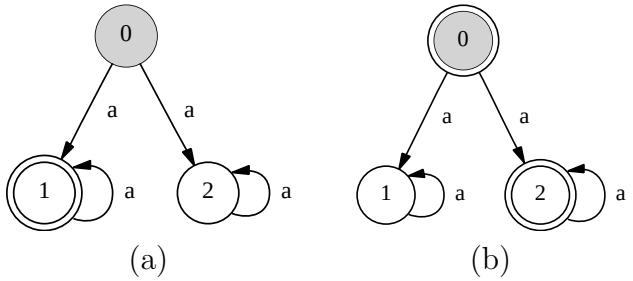


Fig. 3.8 – *Nécessité de la déterminisation pour complémenter un automate.*

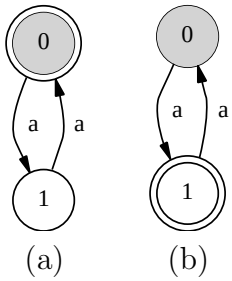


Fig. 3.9 – *Problème dans la complémentation d'automates de Büchi déterministes.*

passer infiniment souvent à la fois dans F et $Q \setminus F$. En outre, la déterminisation n'est en général pas possible dans le cadre strict des automates de Büchi [Saf89].

Des constructions de complémentation sont néanmoins connues [Büc62, SVW87, Saf89, KV97] mais elles sont d'un coût théorique élevé : il est possible de prouver que la borne optimale est de $2^{\mathcal{O}(n \log n)}$ états pour complémenter un automate de Büchi avec n états. Il faut comparer ce chiffre avec la borne optimale de la déterminisation des automates sur mots finis. Cette borne est $2^{\mathcal{O}(n)}$, ce qui n'est pas nettement meilleur. De plus, on sait qu'il est difficile d'arriver à cette dégénérescence en pratique. On pourrait donc se demander si l'on ne pourrait pas appliquer les algorithmes de complémentation évoqués.

Ce n'est pas le cas. Tout d'abord, la construction initiale de Büchi [Büc62] génère $\mathcal{O}(2^{2^n})$ états, ce qui est bien au-delà de la borne optimale. Ensuite, les techniques de [SVW87, Saf89] nécessitent la construction de structures de données annexes dont la taille est systématiquement exponentielle en fonction de la taille de l'automate. Elles ne sont donc pas directement implémentables. Le récent algorithme [KV97] a l'avantage de ne pas utiliser de structure annexe tout en étant de complexité optimale. Malheureusement, ses performances sont encore mal connues[6]. Nous partirons donc du postulat que l'on ne peut pas utiliser directement les algorithmes de complémentation pour les automates de Büchi.

6. En fait, la construction de [KV97] exploite en son sein un algorithme de déterminisation que nous utiliserons sous une forme simplifié dans la méthode que nous allons développer. Ainsi, même si cette construction donnait de bons résultats en pratique, ce que nous allons proposer serait systématiquement plus efficace.

3.3.6 Tests

Il est possible de décider si un automate de Büchi accepte ou non le langage vide. On utilise pour ce faire la proposition suivante :

Proposition 3.3. Un automate de Büchi A n'accepte pas le langage vide si et seulement si il existe un état $s \in F$ accessible depuis un état initial et accessible depuis lui-même dans le graphe de l'automate.

Preuve. ($\Rightarrow$) Supposons que $w \in L_\omega(A)$. Pour être acceptrice, la course ρ de A sur w doit partir d'un état initial et être telle qu'il existe un état accepteur $s \in \mathrm{Inf}(\rho)$. On en déduit que s est accessible depuis un état initial et que s doit être accessible depuis lui-même.

($\Leftarrow$) Considérons une course ρ qui part d'un état initial pour rejoindre s et qui, une fois s atteint, emprunte le chemin de s à s un nombre infini de fois. Il est par hypothèse possible de construire un mot w étiqueté par ρ. On a de plus par construction que $s \in \mathrm{Inf}(\rho)$. Or, $s \in F$; donc la course est acceptrice. De ce fait, $w \in L_\omega(A)$ et le langage accepté n'est pas vide. □

Cette proposition fait le lien entre langage accepté et graphe de l'automate. On peut poursuivre ce rapprochement en introduisant la notion de *composante fortement connexe* :

Définition 3.12. Soit $G = (V,E)$ un graphe dirigé dont les nœuds sont V et les arcs sont E. On peut partitionner V en classes d'équivalence Q_i telles que deux nœuds p et q sont équivalents si et seulement si il existe un chemin dans G pour aller de p à q et un chemin pour revenir de q à p. Ces classes d'équivalence forment les *composantes fortement connexes* du graphe G.

L'algorithme de Tarjan [Tar72] permet de détecter les composantes fortement connexes d'un graphe en un temps linéaire $\mathcal{O}(\max(|V|,|E|))$ [AHU74]. Les variantes de cet algorithme abondent [AHU74, Nuu95]. On en déduit :

Corollaire 3.4. Un automate de Büchi A accepte le langage vide si et seulement si il n'existe aucune composante fortement connexe Q_i dans le graphe de l'automate contenant un état accepteur et accessible depuis un état initial. Si $|Q_i| = 1$, il faut en outre que le seul état de la composante soit relié à lui-même par une transition. Cette vérification peut se faire en un temps linéaire.

Preuve. La preuve de ce théorème est immédiate si l'on utilise la proposition 3.3. Puisque l'algorithme de Tarjan est de complexité linéaire, il en est de même pour le test. □

Remarque 3.13. L'algorithme de Tarjan demande également une complexité en espace linéaire, ce qui peut être handicapant pour de gros automates. Des alternatives

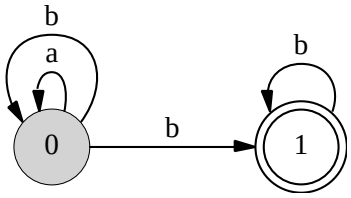


FIG. 3.10 – *Automate acceptant les mots contenant un nombre fini de a.*

à ce test ont été proposées [CVWY92, GH93]. Elles réduisent le nombre de bits à stocker par état de l'automate en ne calculant pas les composantes du graphe. Nous ne les présentons pas ici car l'algorithme de Tarjan nous sera utile plus loin pour un tout autre usage. □

Grâce au test du langage vide et à la construction de différence, il est possible de tester l'inclusion et l'égalité des $\omega-$langages acceptés par deux automates de Büchi A_1 et A_2 définis sur un même alphabet. En effet :

$$\begin{aligned} L_\omega(A_1) \subseteq L_\omega(A_2) &\Leftrightarrow L_\omega(A_1) \setminus L_\omega(A_2) = \phi \\ L_\omega(A_1) = L_\omega(A_2) &\Leftrightarrow L_\omega(A_1) \subseteq L_\omega(A_2) \wedge L_\omega(A_2) \subseteq L_\omega(A_1) \end{aligned}$$

Ces tests font tous deux usage de l'algorithme de complémentation. Il ne sont donc pas non plus exploitables tels quels dans une implémentation pratique.

3.4 $\omega-$automates et déterminisme

On définit les automates de Büchi déterministes de la même manière que les automates déterministes sur mots finis. Néanmoins, contrairement aux automates sur mots finis, les automates de Büchi non déterministes permettent d'exprimer plus de choses que leurs contreparties déterministes.

Exemple. Le langage des mots définis sur $\Sigma = \{a,b\}$ ne contenant qu'un nombre fini de a ne peut être accepté par un automate de Büchi déterministe [Saf89]. Il est cependant accepté par l'automate non déterministe de la figure 3.10. □

Nous terminons ce chapitre en étudiant les propriétés des constructions proposées plus tôt dans le cadre des automates déterministes.

1. Les constructions d'union et d'intersection appliquées à des automates déterministes génèrent un automate déterministe. On s'en rend compte en observant la forme des fonctions de transition. On dit que la classe des $\omega-$automates déterministes est *close par union et intersection*. Il en est de même pour le produit cartésien.
2. Par contre, la projection risque d'introduire systématiquement du non-déterminisme.

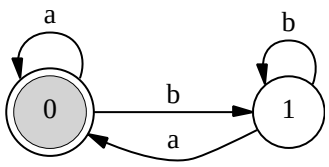


FIG. 3.11 – *Automate déterministe acceptant les mots avec un nombre infini de a.*

Exemple. Il suffit de songer à un état d'où partent deux transitions étiquetées par *aa* et *ab*. Ce branchement est déterministe mais, si on projette sur la première composante, on obtient deux étiquettes identiques *a* et *a*. □

3. L'automate *déterministe* de la figure 3.11 accepte le $\omega-$langage des mots contenant un nombre infini de *a*. Or, la discussion sur la figure 3.10 montrait que le complément de ce langage ne pouvait être accepté par un automate déterministe. Par conséquent, la complémentation (et *a fortiori* la différence) d'$\omega-$automates déterministes peut elle aussi impliquer l'émergence de non-déterminisme.

Chapitre 4

Automates sur vecteurs de réels

Le second chapitre nous a appris comment encoder un *vecteur* de $\mathbb{R}^n$ sous la forme d'un mot infini. Dans le troisième chapitre, nous avons présenté les automates de Büchi qui sont capables de représenter sous une forme compacte des *ensembles* de mots infinis, c'est-à-dire des $\omega-$langages.

L'objet du présent chapitre est de faire le lien entre les deux précédents : nous allons représenter un *ensemble de vecteurs* sous la forme d'un $\omega-$automate qui acceptera le $\omega-$langage correspondant à l'encodage de ces vecteurs. Ce système de représentation s'appelle un « RVA » (*Real Vector Automaton*).

Nous présentons ensuite des RVA de base et nous montrons comment les combiner entre eux. Malheureusement, comme nous l'avons vu au chapitre précédent, les $\omega-$automates généraux sont trop complexes pour être utilisés en pratique. C'est pourquoi nous terminons ce chapitre en proposant une solution pour faciliter la manipulation des RVA. Cette solution sera étudiée dans les chapitres suivants.

4.1 Définition

Pour rappel, il est possible de représenter un vecteur de réels à n composantes sous la forme d'un mot infini défini sur l'alphabet $\Sigma = \{0,\ldots,r-1\}^n \cup \{\star\}$, où r est la base de numération utilisée. Nous savons également par la section 2.2.3 que chaque réel admet une infinité d'encodages possibles car on peut répéter son chiffre de signe un nombre arbitraire de fois. De plus, certains réels admettent des encodages duals, comme nous l'a montré la section 2.5.

Ainsi, quand on veut représenter un ensemble de vecteurs de réels sous la forme d'un $\omega-$automate, la question se pose de savoir le(s)quel(s) de leurs encodages l'automate devra être à même d'accepter. Le choix naturel que nous allons adopter est d'accepter *tous* les encodages :

Définition 4.1. **[BBR97]** Soient $n > 0$ et $r > 1$ des entiers. Un *Automate sur Vecteurs de Réels* (*Real Vector Automaton* — RVA) A en base r pour un sous-ensemble S de $\mathbb{R}^n$ est un automate de Büchi défini sur l'alphabet $\Sigma = \{0,\ldots,r-$

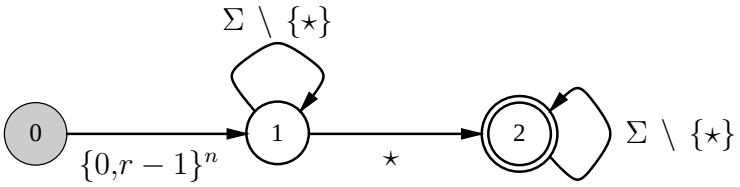


Fig. 4.1 – *RVA représentant* $\mathbb{R}^n$.

$1\}^n \cup \{\star\}$ et tel que :

1. tout mot accepté par A est un encodage *valide* en base r d'un vecteur de $\mathbb{R}^n$;
2. pour chaque vecteur $\vec{x} \in S$, A accepte *tous* les encodages de $\vec{x}$ en base r ;
3. pour chaque vecteur $\vec{x} \notin S$, A n'accepte *aucun* encodage de $\vec{x}$.

En d'autres termes, si V représente le langage des encodages valides défini à la section 2.7, le RVA A correspondant à S doit être tel que :

$$L_\omega(A) = \{w \in V \mid (\exists \vec{x} \in S)(\vec{x} = d(w))\}$$

Définition 4.2. On dit qu'un RVA *représente* l'ensemble des vecteurs encodés par les mots qui appartiennent au langage qu'il accepte. Nous noterons $S(A) \subseteq \mathbb{R}^n$ pour désigner l'ensemble représenté par le RVA A. Il ne faut surtout pas le confondre avec le langage accepté $L_\omega(A) \subseteq \Sigma^\omega$ qui est un ensemble de *mots* et non un ensemble de vecteurs de réels. Plus formellement, on pose :

$$S(A) = \{\vec{x} \in \mathbb{R}^n \mid (\exists w \in L_\omega(A))(\vec{x} = d(w))\}$$

4.2 RVA élémentaires

Le but de cette section est de fournir certains RVA de base qui représentent des sous-ensembles de $\mathbb{R}^n$ utiles.

Ensemble vide ϕ : Le premier RVA que nous évoquons est le RVA trivial, celui pour lequel $S(A) = \phi$. Un tel RVA ne doit accepter aucun mot ; ce faisant, il ne représentera aucun vecteur. Pour cet usage, l'automate vide (i.e. tel que $Q = \phi$) convient parfaitement.

Ensemble $\mathbb{R}^n$: Un RVA particulièrement important est celui qui représente exactement $\mathbb{R}^n$: on parlera du RVA *universel*. Dans ce cas, le RVA doit accepter *tous* les encodages valides de *tous* les vecteurs de $\mathbb{R}^n$. En fait, il s'agit d'accepter le $\omega-$langage V de la section 2.7. Le graphe d'un tel automate est présenté à la figure 4.1, où nous avons permis d'agglomérer un ensemble de transitions en un seul arc.

Ensemble $\mathbb{Z}$: Les RVA sont également capables de représenter l'ensemble des entiers. Le RVA correspondant doit simplement s'assurer que la partie fractionnaire de l'encodage soit une répétition du chiffre « 0 » ou « $r-1$ », après avoir

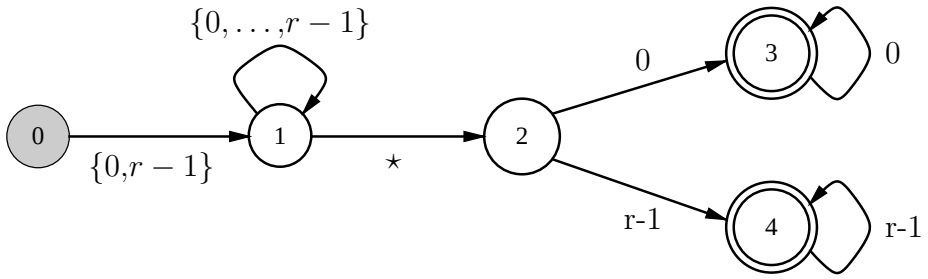


Fig. 4.2 – *RVA représentant* $\mathbb{Z}$.

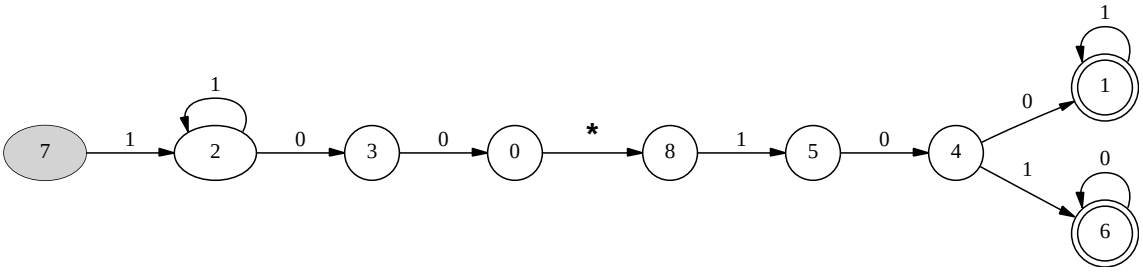


Fig. 4.3 – *RVA représentant l'ensemble* $\{x \mid x = -3{,}375\}$ *en base binaire.*

vérifié que l'encodage commence bien par un chiffre de signe. Dans ce cas en effet, la série géométrique correspondant au décodage de la partie fractionnaire vaut respectivement 0 ou 1: la contribution de la partie fractionnaire est alors entière et le réel obtenu appartient bien à $\mathbb{Z}$.

Constante rationnelle : Pour représenter un ensemble contenant pour seul élément une constante rationnelle donnée, il suffit de créer un automate qui accepte successivement tous les chiffres de celle-ci. Pour que le RVA soit bien formé, il faut rester vigilant à deux points : (i) le RVA doit permettre au chiffre de signe d'être répété et (ii) le RVA doit accepter à la fois l'encodage haut et l'encodage bas de la constante, s'ils existent.
Pour fixer les idées, nous donnons à la figure 4.3 un RVA permettant de représenter le réel -3,375 en binaire, dont l'encodage haut minimal est $100 \star 101(0)^{\omega}$.[1]

Inégalité : L'automate de la figure 4.4 de la page 32 exprime le sous-ensemble $\{(x,y) \mid x \leq y\}$ de $\mathbb{R}^2$. Nous avons fixé la base à $r = 2$ pour ne pas alourdir inutilement le diagramme. Géométriquement, cet ensemble correspond au demi-plan situé par-dessus la bissectrice du premier quadrant. Dans ce graphe de RVA ainsi que dans tous les suivants, nous avons pris pour convention de résumer un tuple de chiffres $(d_1; \ldots; d_n)$ par l'écriture $d_1 \ldots d_n$.

Le fonctionnement de cet RVA est relativement simple : tant qu'il lit $(0;0)$ ou $(1;1)$, il peut uniquement déduire que les deux réels sont égaux jusqu'à présent. Il continue dès lors sur le tronc commun formé par les états 1 et 2. Dès qu'il

1. La numérotation des états n'a aucune signification particulière. Elle a été générée automatiquement par notre implémentation présentée au chapitre 8.

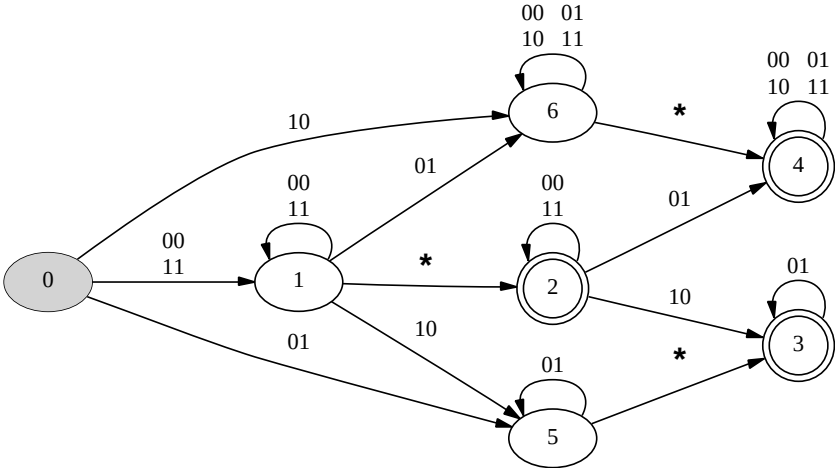


Fig. 4.4 – *RVA représentant* $\{(x,y) \in \mathbb{R}^2 \mid x \leq y\}$ *en base binaire.*

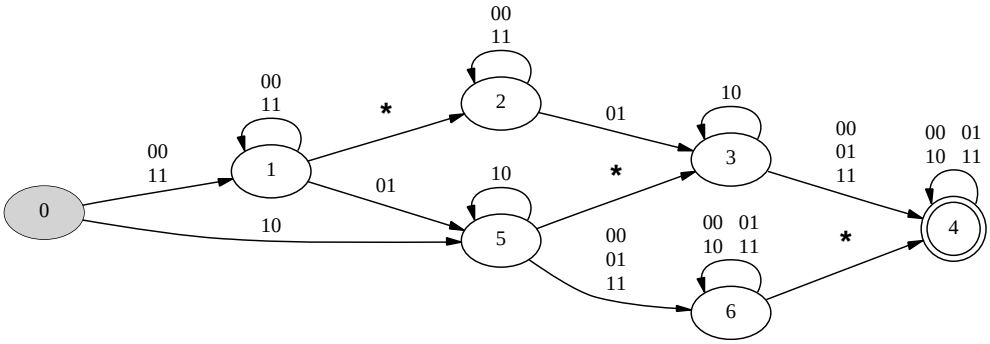


Fig. 4.5 – *RVA représentant* $\{(x,y) \in \mathbb{R}^2 \mid x < y\}$ *en base binaire.*

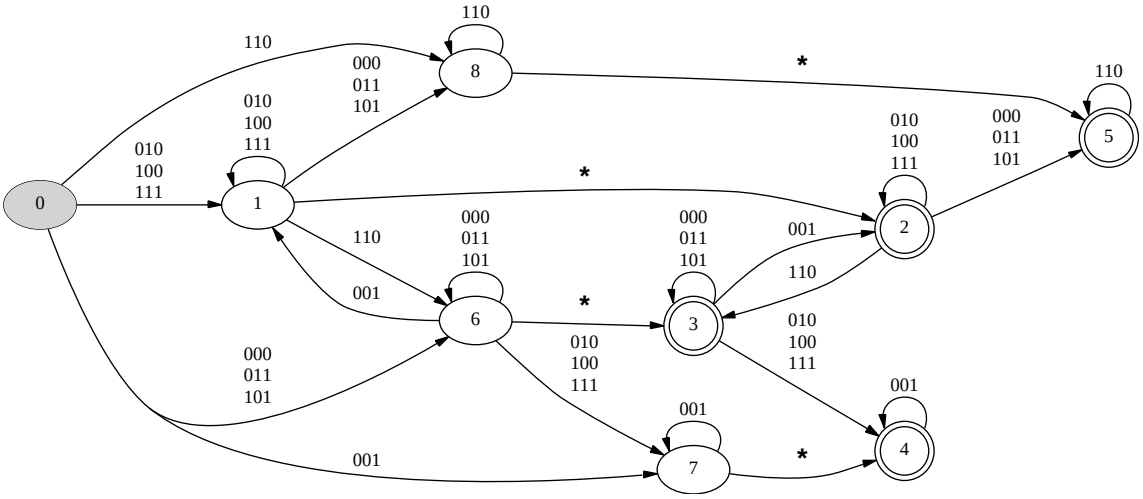


Fig. 4.6 – *RVA représentant* $\{(x,y,z) \in \mathbb{R}^3 \mid x + y = z\}$ *en base binaire.*

détecte un caractère $(0;1)$, le RVA sait que $x \leq y$ sera systématiquement vrai. C'est pourquoi il débraie automatiquement vers les états 4 et 6 à partir desquels il accepte tout encodage valide. Si l'automate lit $(1;0)$, on pourrait croire que le mot doit être rejeté. Du fait de l'existence d'encodages duals, ceci n'est pas correct : une répétition infinie de $(1;0)$ donnerait un couple tel que $x = y$. C'est pourquoi on introduit les états 3 et 5 dont le but est de détecter si x est l'encodage haut de y.

Inégalité stricte : Du RVA précédent, il est relativement facile d'en déduire un qui représente l'ensemble $\{(x,y) \in \mathbb{R}^2 \mid x < y\}$. Il suffit de s'assurer qu'aucun chemin dans l'automate de la figure 4.4 n'amène à une égalité.

On peut tout d'abord supprimer les états 3 et 5. D'autre part, il faut éviter que l'on puisse boucler dans un état accepteur en lisant les mêmes chiffres pour x et y, comme c'est le cas dans les états 2 et 4. Plus précisément, un chemin accepteur devra toujours passer par le caractère $(0;1)$ et ne devra jamais consister en une répétition infinie de ce dernier caractère pour ne pas engendrer un encodage dual. Le RVA correspondant est présenté à la figure 4.5.

Somme : Nous terminons ce tour d'horizon en proposant un sous-ensemble plus particulier de l'espace à trois dimensions qu'il est possible de représenter par un RVA, à savoir : $\{(x,y,z) \in \mathbb{R}^3 \mid x + y = z\}$. Géométriquement, cet ensemble est le plan dont les points ont pour troisième coordonnée, la somme de leur abscisse et de leur ordonnée. Ce RVA, plus complexe que les précédents, est repris à la figure 4.6.

Son principe fonctionnel est de calculer progressivement la somme de x et y et, le cas échéant, de passer dans un état particulier quand un report a été généré. Sa difficulté réside surtout dans la détection des encodages duals. Nous ne le détaillons pas plus pour l'instant car nous verrons dans une section ultérieure comment générer des RVA à partir d'(in)équations générales : le RVA que nous venons de proposer en sera un cas particulier.

4.3 Manipulation des RVA

Nous venons de voir comment construire des RVA de base. L'objet de la présente section est d'étudier les opérations que l'on peut réaliser sur ceux-ci. Plus précisément, nous allons voir comment réaliser certaines opérations nécessaires pour pouvoir appliquer les RVA à la vérification symbolique de programmes [Boi98].

L'avantage à utiliser des RVA par rapport à d'autres systèmes de représentation d'ensembles de réels est que nous bénéficions de toute la puissance de la théorie des automates : par exemple, l'algorithme de quantification existentielle sera trivial. Nous allons voir comment adapter les algorithmes de la section 3.3 en vue de manipuler les RVA. Ces algorithmes ne pourront pas toujours être appliqués de manière brutale, car il faudra toujours s'assurer que l'automate résultant est *bien formé*, en ce sens qu'il accepte *tous* les encodages des vecteurs qu'il représente.

4.3.1 Opérations booléennes

Soient des entiers $r > 1$ et n_1,$n_2 \geq 0$. Soient A_1 et A_2 deux RVA fonctionnant dans la même base r et représentant respectivement des ensembles de vecteurs de réels $S(A_1) \subseteq \mathbb{R}^{n_1}$ et $S(A_2) \subseteq \mathbb{R}^{n_2}$.

Calcul de l'union $S(A_1) \cup S(A_2)$

On suppose que $n_1 = n_2$. Dans ce cas, il suffit simplement d'appliquer la construction d'union pour les automates de Büchi (voir section 3.3.1) aux automates A_1 et A_2. L'automate A obtenu représente $S(A_1) \cup S(A_2)$.

Preuve. A_1 et A_2 sont des RVA, donc tout mot accepté par l'un ou l'autre est un encodage valide en base r d'un vecteur de $\mathbb{R}^{n_1} = \mathbb{R}^{n_2}$. Puisque la construction n'ajoute aucun mot en dehors de ceux acceptés par A_1 et A_2, tout mot accepté par A est un encodage valide en base r.

D'autre part, soit un vecteur $\vec{x}$ représenté par A_1. A_1 accepte tous les encodages valides de $\vec{x}$: chacun de ces encodages se trouvera donc représenté par A. On fait le même raisonnement pour A_2 et on conclut en réutilisant l'argument que la construction d'union n'ajoute aucun mot qui n'est accepté ni par A_1 ni par A_2. □

Calcul de l'intersection $S(A_1) \cap S(A_2)$

On suppose à nouveau que $n_1 = n_2$. On peut ici aussi appliquer simplement la construction d'intersection des automates de Büchi à A_1 et A_2 (voir section 3.3.2). La preuve en est foncièrement identique.

Calcul du produit cartésien $S(A_1) \times S(A_2) \subseteq \mathbb{R}^{n_1+n_2}$

Ici, on n'a plus nécessairement besoin que n_1 soit égal à n_2. Nous voulons construire un RVA A défini sur l'alphabet $\Sigma = \{0, \ldots, r-1\}^{n_1+n_2} \cup \{\star\} = \Lambda^{n_1+n_2} \cup \{\star\}$ qui représente $S(A_1) \times S(A_2)$. En première approximation, l'algorithme de la section 3.3.3 fonctionne parfaitement pour calculer A. Il faut toutefois prendre garde aux séparateurs décimaux. En effet, en construisant A, nous rencontrerons trois types de transitions :

1. des transitions étiquetées par $(u; v)$ où $u \in \Lambda^{n_1}$ et $v \in \Lambda^{n_2}$,
2. des transitions étiquetées par $(\star; \star)$,
3. des transitions étiquetées par $(u; \star)$ ou $(\star; v)$.

La première classe de transitions ne pose pas de problème car leur étiquette est bien définie sur l'alphabet $\Lambda^{n_1+n_2}$. La seconde classe donne un symbole qui n'appartient pas à Σ. Il est néanmoins facile de le convertir en appliquant la règle $(\star; \star) := \star$. Cette règle est sémantiquement correcte car elle exprime la lecture simultanée du séparateur décimal pour les deux vecteurs qui seront regroupés en un seul.

Par contre, les transitions $(u,\star)$ ou $(\star,v)$ vont engendrer dans l'automate A un encodage invalide puisqu'on avait supposé dans la section 2.6 que l'on devait synchroniser la lecture du séparateur décimal dans l'encodage d'un vecteur. Après construction de A, il faudra donc « épurer » l'automate de ce type de transitions. Une démarche plus simple est d'empêcher la création de telles transitions au cours de la construction de A : si une telle transition doit être générée à un moment donné, on avorte l'exploration sur celle-ci.

Calcul de la différence $S(A_1) \setminus S(A_2)$

On suppose que $n_1 = n_2$. Pour réaliser ce calcul, on procède en deux étapes : (i) complémenter le langage accepté par A_2 et (ii) construire l'intersection de ce nouveau langage avec le langage accepté par A_1.

La première étape est possible, mais très lourde algorithmiquement (voir section 3.3.5). Elle n'est donc pas implémentable directement. Dans la suite de ce travail, nous allons nous attacher à montrer comment la concrétiser.

Il faut bien comprendre que $S(A_2)^c \neq S((L_\omega(A_2))^c)$. En effet, $(L_\omega(A_2))^c$ comprend *tous* les encodages qui ne sont pas dans $L_\omega(A_2)$, y compris les encodages non valides ! Dans le cas de la différence, ce problème est caché par la seconde étape : le fait de prendre l'intersection avec un RVA bien formé retire tous les encodages invalides. Si on voulait calculer l'automate représentant $S(A_2)^c$, il faudrait non seulement complémenter A_2, mais encore ôter tous les encodages invalides : en fait, il faudrait prendre l'intersection du complément de A_2 avec le RVA représentant $\mathbb{R}^n$. On se ramènerait ainsi au cas de la différence. Pour résumer, on peut dire que la complémentation d'un RVA consiste en une différence déguisée.

Remarque 4.3. En exploitant la différence, il est possible de calculer le RVA qui représente la bissectrice du premier et du troisième quadrant, c'est-à-dire l'ensemble $V = \{(x,y) \in \mathbb{R}^2 \mid x = y\}$. En effet, $V = \{(x,y) \mid x \leq y\} \setminus \{(x,y) \mid x < y\}$, que l'on sait tous deux représenter par des RVA. □

4.3.2 Tests

La proposition suivante permet de ramener les tests d'ensemble vide, d'inclusion et d'égalité pour des ensembles représentés par des RVA, aux tests correspondants sur les automates de Büchi :

Proposition 4.1. $S(A) = \phi$ si et seulement si $L_\omega(A) = \phi$.

Preuve. Si un RVA est tel que $S(A) \neq \phi$, il doit représenter au moins un vecteur $\vec{x}$. Par conséquent, son langage comprend tous les encodages de ce vecteur. De ce fait, son langage ne peut être vide. Réciproquement, si $L_\omega(A) \neq \phi$, par définition d'un RVA, il accepte au moins un encodage valide d'un vecteur de $\mathbb{R}^n$. Soit $\vec{x}$ ce vecteur. On a $\vec{x} \in S(A)$ et donc, $S(A) \neq \phi$. □

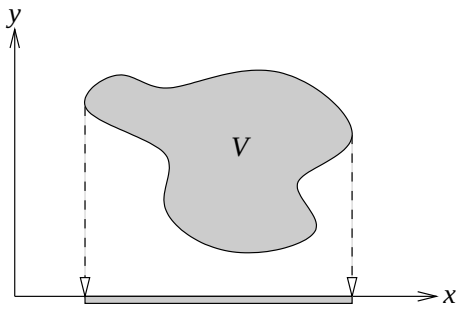


Fig. 4.7 – *Projection d'un ensemble $V \subseteq \mathbb{R}^2$ selon son ordonnée.*

4.3.3 Projection

Étant donné un ensemble $V \subseteq \mathbb{R}^n$, il est intéressant de *projeter* cet ensemble selon une coordonnée. Formellement, la projection de V sur toutes ses coordonnées à l'exception de la $i-$ème est définie ainsi :

$$V\Big|_{\neq x_i} = \left\{ \begin{array}{l} (x_1,\ldots,x_{i-1},x_{i+1},\ldots,x_n) \in \mathbb{R}^{n-1} \mid \\ \qquad (\exists x_i \in \mathbb{R})(x_1,\ldots,x_{i-1},x_i,x_{i+1},\ldots,x_n) \in V \end{array} \right\}$$

La projection est donc le pendant ensembliste de la quantification existentielle.

Exemple. La figure 4.7 présente la projection d'un sous-ensemble V du plan selon sa seconde composante. □

Il existe de manière évidente une grande similitude entre cette opération de projection et celle consistant à projeter des automates de Büchi. Pourtant, on n'a pas toujours $S(A)\big|_{\neq x_i} = S\left(A\big|_{\neq \sigma_i}\right)$.

Une première raison en est qu'on ne peut pas « projeter » le symbole $\star$. Cela n'est cependant pas véritablement gênant : on peut appliquer la règle $\star\big|_{\neq \sigma_i} := \star$. Cette règle est en accord avec la sémantique du séparateur décimal : projeter un encodage ne doit pas changer la position de son séparateur décimal.

Il y a pourtant une difficulté plus fondamentale. En effet, l'automate $A\big|_{\neq \sigma_i}$ n'est pas toujours bien formé : il peut ne pas accepter tous les encodages des vecteurs qu'il représente, contrairement aux autres opérations déjà envisagées.

Exemple. Considérons le RVA qui représente l'ensemble $V = \{(x,y) \in \mathbb{R}^2 \mid x = 0 \wedge y = 5\}$ en binaire. L'automate correspondant doit accepter tous les encodages du vecteur (0,5). Ceux-ci forment le langage :

$$\begin{array}{rcl} L & = & ((0;0)^*(0;1)(0;0)(0;1)\star(0;0)^\omega) \ \cup \ ((0;0)^*(0;1)(0;0)(0;0)\star(0;1)^\omega) \ \cup \\ & & ((1;0)^*(1;1)(1;0)(1;1)\star(1;0)^\omega) \ \cup \ ((1;0)^*(1;1)(1;0)(1;0)\star(1;1)^\omega) \end{array}$$

Si maintenant, on projette ce langage selon la seconde composante, on obtient :

$$L\big|_{\neq \sigma_2} \quad = \quad ((0)^*000\star(0)^\omega) \ \cup \ ((1)^*111\star(1)^\omega)$$

Or, $V\big|_{\neq y} = \{0\}$: le RVA correspondant devrait dès lors accepter tous les encodages du réel zéro, soit le langage $(0)^* \star (0)^\omega \cup (1)^* \star (1)^\omega \neq L\big|_{\neq \sigma_2}$. En fait, puisque 5 a un encodage minimal de longueur 4, aucun encodage de zéro dont la taille est inférieure à 4 ne sera accepté. □

D'une façon plus générale, la longueur minimale des parties entières est alignée sur la composante de plus forte valeur absolue (cf. section 2.6). Si cette composante est éliminée pendant la projection, les encodages plus courts n'appartiendront pas au nouveau langage, ce qui va à l'encontre de la définition des RVA, qui sont censés accepter *tous* les encodages valides du vecteur.

La projection d'un RVA se déroule donc en deux étapes : (i) projeter l'automate correspondant et (ii) compléter le langage accepté par l'automate pour qu'il satisfasse la définition des RVA. La seconde étape peut se réaliser ainsi [Boi98] : on considère tous les chiffres de signe possibles u pour un vecteur à n composantes (en nombre 2^n) et on explore l'automate au départ des états initiaux pour détecter quels états s il est possible d'atteindre par une répétition u^i $(i > 0)$ du chiffre de signe. Ensuite, pour chaque état atteint s, on ajoute une transition $\delta(s_0,u) = s$ où s_0 est un des états initiaux de A (peu importe lequel). Ceci complète le langage de l'automate, mais ajoute du non-déterminisme[2].

Remarque 4.4. Cette solution est satisfaisante d'un point de vue théorique, mais elle est d'une complexité exponentielle : ceci la rend inopérante pour des RVA avec un nombre même modeste de composantes. Une alternative est proposée dans [BL01]. Elle reste d'un coût théorique exponentiel dans le cas le plus défavorable, mais présente un bon comportement moyen. Bien que conçue dans le cadre des NDD (*Number Decision Diagrams*, le pendant des RVA pour les entiers uniquement), on peut l'appliquer telle quelle aux RVA. □

4.4 Arithmétique linéaire

4.4.1 Génération efficace d'équations

Les RVA peuvent représenter des ensembles correspondant à des contraintes linéaires. Nous présentons maintenant un algorithme qui permet de générer de manière efficace un RVA représentant l'ensemble S de toutes les solutions $\vec{x} \in \mathbb{R}^n$ d'une équation vectorielle $\vec{a} \,.\, \vec{x} = b$ où $n > 0$, $a \in \mathbb{Z}^n$ et $b \in \mathbb{Z}$. Cet algorithme a été proposé dans [BRW98].

Décomposition du problème

L'idée de la construction est de procéder en trois étapes : (i) on construit un automate qui accepte les parties entières des solutions, (ii) on construit un automate qui en accepte les parties fractionnaires et (iii) on relie les deux parties par un

2. De toute façon, le non-déterminisme avait déjà été induit par l'étape de projection.

séparateur décimal. Pour ce faire, on va séparer le vecteur $\vec{x}$ en ses parties entière et fractionnaire, à savoir $\vec{x}_I \in \mathbb{Z}^n$ et $\vec{x}_F \in [0,1]^n$, définies d'une manière telle que $\vec{x} = \vec{x}_I + \vec{x}_F$. Il faut maintenant chercher ce à quoi doivent satisfaire $\vec{x}_I$ et $\vec{x}_F$.

Ils doivent tout d'abord satisfaire la contrainte suivante :

$$\vec{a} \,.\, \vec{x} = b \Leftrightarrow \vec{a} \,.\, \vec{x}_I = b - \vec{a} \,.\, \vec{x}_F$$

Par ailleurs, puisque $\vec{x}_F \in [0,1]^n$, on aura toujours :

$$\alpha_- = \sum_{a_i<0} a_i \leq \vec{a} \,.\, \vec{x}_F \leq \sum_{a_i>0} a_i = \alpha_+$$

Ces deux contraintes mises ensembles donnent la relation que doit satisfaire $\vec{x}_I$:

$$b - \alpha_+ \leq \vec{a} \,.\, \vec{x}_I \leq b - \alpha_-$$

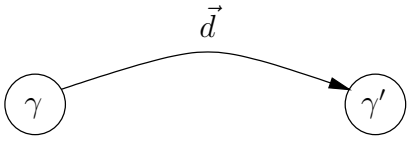
L'algorithme procède en envisageant *toutes* les valeurs β permises pour $\vec{a} \,.\, \vec{x}_I$ par l'équation ci-dessus. On a une contrainte supplémentaire pour β : puisqu'on pose $\beta = \vec{a} \,.\, \vec{x}_I$ et que $\vec{x}_I$ est un vecteur d'entiers, β doit être divisible par le plus grand commun diviseur des composantes du vecteur $\vec{a}$, que nous désignerons par $\gcd(a_1, \ldots, a_n)$. En résumé, on définit β comme une valeur entière qui satisfait :

$$\boxed{\begin{array}{c} b - \alpha_+ \leq \beta \leq b - \alpha_- \\ (\exists m \in \mathbb{Z})\beta = m \cdot \gcd(a_1, \ldots, a_n) \end{array}} \tag{4.1}$$

Il ne nous reste plus qu'à créer, pour toute valeur admissible de β, un automate sur mots finis qui correspond à la contrainte $\vec{a} \,.\, \vec{x}_I = \beta$, un automate de Büchi qui correspond à $\vec{a} \,.\, \vec{x}_F = b - \beta$ puis à les relier entre eux par une transition étiquetée par un séparateur décimal.

Reconnaître la partie entière

Le principe de la construction est de garder la trace de la valeur $\vec{a} \,.\, \vec{y}$ au fur et à mesure que les chiffres d'un vecteur d'entiers $\vec{y} \in \mathbb{Z}^n$ sont en train d'être lus. Pour ce faire, on associe à chaque état s de l'automate de la partie entière, un entier γ correspondant à la valeur de $\vec{a} \,.\, \vec{y}$ pour les chiffres de $\vec{y}$ qui ont déjà été lus. Il est facile de déterminer comment varie $\vec{a} \,.\, \vec{y}$ quand on emprunte une transition étiquetée par le chiffre $\vec{d} \in \Lambda^n$ au départ d'un état étiqueté par γ pour aboutir à un état étiqueté par γ' :



$$\boxed{\gamma' = r \cdot \gamma + \vec{a} \,.\, \vec{d}} \tag{4.2}$$

Cette manière de procéder est correcte, y compris pour les nombres négatifs, si l'on se souvient de l'équation (2.2) à la page 9, qui définit la fonction $d_{\mathbb{Z}}$ de décodage des

- $Q = \mathbb{Z} \cup \{s_0\}$;
- $\Sigma = \{0, \ldots, r-1\}^n$;
- La fonction de transition est définie en trois parties :
  1. $\delta(s_0, \vec{d_s}) = \{-\vec{a} \,.\, \delta(\vec{d_s}, r-1)\}$ si $d_s \in \{0,r-1\}^n$,
  2. $\delta(s_0, \vec{d_s}) = \phi$ si $d_s \notin \{0,r-1\}^n$,
  3. $\delta(\gamma, \vec{d}) = \{\ r \cdot \gamma + \vec{a} \,.\, \vec{d}\ \}$ où $\gamma \neq s_0$ et où $\vec{d} \in \Sigma$;
- $S_0 = \{s_0\}$;
- $F = \{\beta\}$.

Fig. 4.8 – *Automate reconnaissant les solutions entières à une équation.*

entiers. C'est ici que l'on comprend l'utilité d'avoir pris comme convention d'employer le complément à la base pour représenter des entiers négatifs : l'addition de nombres négatifs bénéficie alors d'un traitement identique à celui de l'addition de nombres positifs.

Quand on a $\vec{a} \,.\, \vec{y} = \beta$ pour un état s_F, $\vec{x}_I$ peut se confondre avec $\vec{y}$ et les chiffres que nous avons lus depuis l'état initial désignent l'encodage d'un vecteur $\vec{x}_I$ qui est solution à l'équation $\vec{a} \,.\, \vec{x}_I = \beta$: on peut ainsi marquer s_F comme accepteur.

Il reste à régler le problème de l'état initial. Dans cet état, on ne peut lire qu'un chiffre de signe $\vec{d_s} \in \{0,r-1\}^n$ afin de définir un encodage valide. L'équation (2.2) nous affirme que la contribution générée par un chiffre de signe $\vec{d_s}$ est égale à l'entier :

$$\gamma = \sum_{i=1}^{n} (-a_i \cdot \delta(d_{s_i}, r-1))$$

où δ est le symbole de Kronecker. Il suffit d'envisager tous les chiffres de signe $\vec{d_s}$, puis de relier s_0 à l'état γ donné par l'équation ci-dessus. On voit une nouvelle fois que le complément à la base nous a permis de dégager aisément une solution.

Formellement, l'automate sur mots finis $A = (Q, \Sigma, \delta, S_0, F)$ qui reconnaît les solutions entières à l'équation $\vec{a} \,.\, \vec{x}_I = \beta$ est défini à la figure 4.8, où l'on a permis l'application du symbole de Kronecker à toutes les composantes d'un vecteur prises séparément.

Cette construction appliquée telle quelle fournit un automate avec un nombre infini d'états. Cependant, il n'y a qu'un nombre fini d'états accessibles depuis S_0. En pratique, on construit plutôt l'automate à rebours : on débute depuis l'état accepteur β, puis on remonte les transitions qui aboutissent à chacun des états déjà construits. Ce faisant, seuls les états significatifs sont ajoutés à l'automate. Dans ce cas, si on remonte depuis un état γ' jusqu'à un état γ, on modifie l'équation (4.2) ainsi :

$$\gamma = \frac{\gamma' - \vec{a} \,.\, \vec{d}}{r}$$

Si γ n'est pas un entier, il n'est pas ajouté à l'automate puisque $Q = \mathbb{Z} \cup \{s_0\}$.

- $Q = \mathbb{Z}$;
- $\Sigma = \{0, \ldots, r-1\}^n$;
- Pour tout $\vec{d} \in \Sigma$ et tout $\gamma \in Q$, la fonction de transition est définie :

$$\delta(\gamma, \vec{d}) = \begin{cases} \{\gamma'\} & \text{si } \gamma' \in [\alpha_-, \alpha_+] \\ \phi & \text{sinon} \end{cases}$$

  où l'on a posé $\gamma' = r \cdot \gamma - \vec{a} \,.\, \vec{d}$;
- $S_0 = \{b - \beta\}$;
- $F = Q$.

Fig. 4.9 – *Automate acceptant les solutions fractionnaires à une équation.*

Remarque 4.5. On peut facilement construire un automate partagé qui représente l'ensemble S des solutions à une *disjonction* d'équations $(\vec{a}_1 \,.\, \vec{x}_I = \beta_1) \vee \cdots \vee (\vec{a}_m \,.\, \vec{x}_I = \beta_m)$. Il suffit de remplacer F par l'ensemble $F' = \{\beta_1, \ldots, \beta_m\}$. □

Accepter la partie fractionnaire

Nous nous attachons maintenant à construire un automate de Büchi qui accepte le $\omega-$langage des mots représentant les parties fractionnaires $\vec{x}_F$ telles que $\vec{a} \,.\, \vec{x}_F = b - \beta$. La construction est sensiblement similaire au cas de la partie entière. On fixe maintenant de la façon suivante la sémantique d'un état s étiqueté par γ : toutes les courses issues de s définissent un mot infini w tel que $\vec{a} \,.\, d(\vec{0} \star w) = \gamma$.[3] En exploitant cette relation, il reste facile de déterminer l'étiquette γ' d'un état s' issu par une transition $\vec{d}$ d'un état s étiqueté par γ. Un mot w' accepté au départ de s' doit satisfaire $\vec{a} \,.\, d(\vec{0} \star w') = \gamma'$. Or, $w = \vec{d}w'$, donc :

$$\begin{aligned} \vec{a} \,.\, d(\vec{0} \star w) = \gamma \quad &\Leftrightarrow \quad \vec{a} \,.\, d(\vec{0} \star \vec{d}w') = \gamma \\ &\Leftrightarrow \quad \vec{a} \cdot \frac{\vec{d}}{r} + \vec{a} \cdot \frac{d(\vec{0} \star w')}{r} = \gamma \\ &\Leftrightarrow \quad \boxed{\underbrace{\vec{a} \cdot d(\vec{0} \star w')}_{\gamma'} = r \cdot \gamma - \vec{a} \,.\, \vec{d}} \end{aligned}$$

Nous devons par ailleurs écarter les états étiquetés par un nombre γ qui n'appartient pas à l'intervalle $[\alpha_-, \alpha_+]$. Dans ce cas en effet, ils ne peuvent définir un mot w tel que $\vec{a} \,.\, d(\vec{0} \star w) = \gamma$, puisque $d(\vec{0} \star w) \in [0,1]^n$.

La sémantique d'un état étiqueté par γ nous apprend également que le seul état initial s_0 doit être étiqueté par $b - \beta$: ce faisant, tous les mots w qui seront acceptés par l'automate seront tels que $\vec{a} \,.\, d(\vec{0} \star w) = b - \beta$, ce que nous cherchions à obtenir. Tous les états doivent être accepteurs afin de ne refuser aucun de ces mots qui nous conviennent tous. La construction complète est présentée à la figure 4.9.

3. Nous utilisons l'abréviation $\vec{0}$ pour désigner le symbole 0^n de Σ.

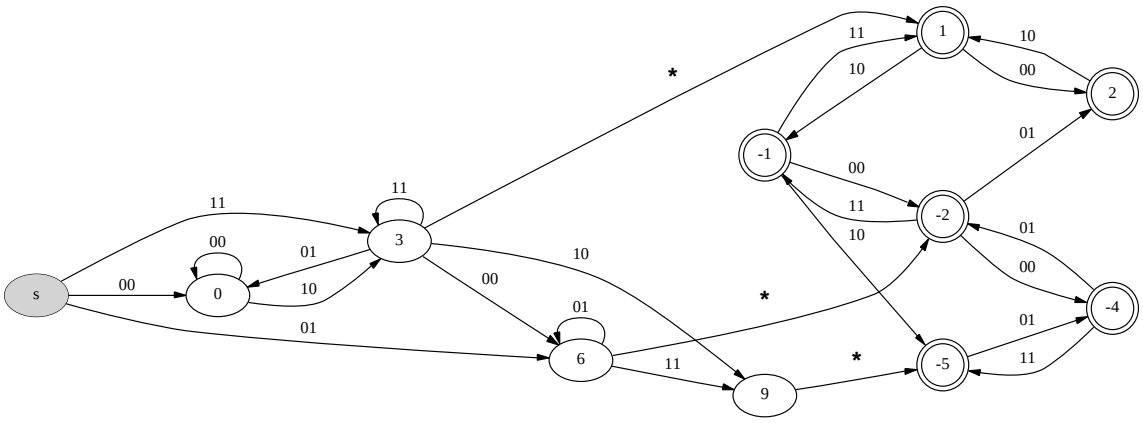


Fig. 4.10 – *RVA représentant l'ensemble de* $\mathbb{R}^2$ *décrit par l'équation* $3x - 6y = 4$.

En pratique, on effectue une recherche en profondeur d'abord au départ de S_0 pour construire l'automate afin de se limiter aux seuls états accessibles.

Remarque 4.6. On peut ici aussi construire un automate partagé pour plusieurs membres de droite $\beta'_1, \ldots, \beta'_m$ et ce, simultanément. Il suffit alors de poser $S_0 = \{\beta'_1, \ldots, \beta'_m\}$. □

Combiner les deux automates

Nous pouvons maintenant donner un schéma plus précis de l'ensemble de l'algorithme. Pour obtenir le RVA final, on procède ainsi :

1. on génère l'ensemble Δ des β qui satisfont à (4.1) ;
2. on construit un automate A_I de partie entière tel que $F_0 = \Delta$;
3. on construit un automate A_F de partie fractionnaire où $S_0 = \{b - \beta \mid \beta \in \Delta\}$;
4. on recopie côte à côte ces deux automates au sein d'un même RVA A en ajoutant une transition étiquetée par $\star$ entre chaque couple formé par un état accepteur de A_I et l'état initial de A_F qui correspond au même β. Plus précisément, on ajoute la transition $p \xrightarrow{\star} q$ où $q = \beta \in Q_I$ et $p = b - \beta \in Q_F$ pour tout $\beta \in \Delta$. Les états accepteurs de A_I ne le sont plus dans A car il faut lire le séparateur décimal avant d'obtenir un encodage valide.

Il est possible de montrer que l'algorithme se termine toujours [BRW98] et que A a un nombre d'états :

$$|Q| = \mathcal{O}\left(\log\ |b| \cdot \sum_{i=1}^{n} |a_i|\right)$$

On voit ainsi que l'ordre de grandeur des coefficients multiplicatifs dans $\vec{a}$ a un impact bien supérieur à celui de la constante additive b : le premier impact est linéaire là où le second est fonction du nombre de chiffres dans b.

Exemple. Le RVA représentant les solutions de l'équation $3x - 6y = 4$ est repris à la figure 4.10. L'automate A_I est situé à gauche des transitions étiquetées par $\star$

tandis que A_F est situé à leur droite. □

Remarque 4.7. L'automate construit est minimal et déterministe. De plus, le nombre de transitions sortantes dans un état s est toujours borné par $r^n + 1$. □

4.4.2 Particularisation aux inéquations

La méthode de construction que nous venons de voir s'adapte facilement aux inéquations. La décomposition du problème est même identique. Nous passerons seulement en revue les différences :

1. Dans l'automate de la partie entière, la sémantique d'un état s étiqueté par γ est la suivante : toute course issue de s_0 et aboutissant à s décrit un mot w tel que $\vec{a} \,.\, d(w \star \vec{0}^\omega) \leq \gamma$. Dans l'automate de la partie fractionnaire, la sémantique d'un état s étiqueté par γ est que toute course infinie issue de s décrit un mot infini w pour lequel $\vec{a} \,.\, d(\vec{0} \star w) \leq \gamma$.
2. Dans l'automate de la partie entière, on ne retire plus les états étiquetés par un γ qui n'est pas entier. Au lieu de cela, on arrondit γ au plus grand entier γ'' divisible par $\gcd(a_1, \ldots, a_n)$ qui lui est strictement inférieur. Dans ce cas en effet, $\vec{a} \,.\, d(w \star \vec{0}^\omega) \leq \gamma \Leftrightarrow \vec{a} \,.\, d(w \star \vec{0}^\omega) \leq \gamma''$ puisque le membre de gauche est nécessairement divisible par $\gcd(a_1, \ldots, a_n)$.
3. Dans l'automate de la partie fractionnaire, on n'ôte plus les états pour lesquels $\gamma' > \alpha_+$: quand cela arrive, on arrondit γ' à α_+. Cette règle est correcte, puisque dans ce cas $\vec{a} \,.\, d(\vec{0} \star w) \leq \gamma' \Leftrightarrow \vec{a} \,.\, d(\vec{0} \star w) \leq \alpha_+$, étant donné que le membre de gauche ne peut pas générer une valeur supérieure à α_+.

Remarque 4.8. La complexité de la construction est similaire au cas des équations. En revanche, le RVA résultant n'est en général ni déterministe, ni minimal. □

4.4.3 Transformations linéaires

Définition

Il est parfois utile de calculer l'image d'un ensemble $U \subseteq \mathbb{R}^n$ par une *transformation linéaire* $T \equiv P\vec{x} \leq \vec{q} \longrightarrow \vec{x} := A\vec{x} + \vec{b}$, où $P \in \mathbb{Q}^{m \times n}$, $\vec{q} \in \mathbb{Q}^m$, $A \in \mathbb{Q}^{n \times n}$ et $\vec{b} \in \mathbb{Q}^n$ ($\mathbb{Q}$ désigne l'ensemble des rationnels). $P\vec{x} \leq \vec{q}$ est appelée la « garde » de la transformation.

Intuitivement, une transformation linéaire correspond à une affectation multiple conditionnelle dans un langage de programmation. Elle considère un sous-ensemble U de l'espace d'états d'un programme dont toutes les variables sont réelles [dM95], les variables entières en étant un cas particulier, et regarde quels états de U satisfont la garde. Pour ces états, on modifie la valeur des variables en appliquant la règle : $\vec{x} := A\vec{x} + \vec{b}$. Les autres états sont éliminés. Ceci ouvre la voie à la vérification automatisée pour certaines classes de systèmes hybrides linéaires [BBR97].

Plus précisément, étant donné un ensemble $U \subseteq \mathbb{R}^n$, on veut pouvoir calculer :

$$T(U) \ =_{def} \ \{A\vec{x} + \vec{b} \mid \vec{x} \in U \wedge P\vec{x} \leq \vec{q}\}$$

Démarche

Pour calculer $T(U)$, on procède en trois étapes : (i) on isole les états qui satisfont la garde, (ii) on adjoint à chaque composante son image par T et (iii) on projette pour ne conserver que l'image. La démarche est de calculer successivement :

$$\begin{aligned} U_1 &= \{(x_1,\ldots,x_n) \in \mathbb{R}^n \mid P\vec{x} \leq \vec{q}\} \\ U_2 &= \{(x_1,\ldots,x_n,x'_1,\ldots,x'_n) \in \mathbb{R}^{2n} \mid \vec{x'} = A\vec{x} + \vec{b}\} \\ T(U) &= \overbrace{((\underbrace{(U \cap U_1)}_{(i)} \times \mathbb{R}^n) \cap U_2)}^{(ii)}\Big|_{\neq x_1,\ldots,x_n} \end{aligned}$$

Ce qu'il est possible de réécrire ainsi :

$$T(U) \ = \ ((U \times \mathbb{R}^n) \cap X)\Big|_{\neq x_1,\ldots,x_n} \qquad \text{où} \quad X \ =_{def} \ (U_1 \times \mathbb{R}^n) \cap U_2$$

Nous avons extrait $X \subseteq \mathbb{R}^{2n}$ car il s'agit d'un élément commun à tous les calculs de $T(U)$ pour un T fixé. Il est dès lors possible d'accélérer les calculs si on doit souvent appliquer T : il suffit de réaliser un pré-traitement dont le but est de calculer un RVA représentant X. On y gagne par la suite, car les algorithmes de génération de contraintes linéaires peuvent devenir relativement coûteux. En utilisant un vocable de la théorie des automates, on peut dire que X est un *transducteur*.

Composition

Une caractéristique très intéressante des transformations linéaires est qu'elles sont closes par composition. L'application successive de deux transformations linéaires $T_1 \equiv P_1\vec{x} \leq \vec{q}_1 \longrightarrow A_1\vec{x} + \vec{b}_1$ et $T_2 \equiv P_2\vec{x} \leq \vec{q}_2 \longrightarrow A_2\vec{x} + \vec{b}_2$ est toujours équivalente à la transformation :

$$T_2 \circ T_1 \equiv P\vec{x} \leq \vec{q} \longrightarrow A\vec{x} + \vec{b}$$

où l'on a posé :

$$\begin{aligned} P \in \mathbb{Q}^{(m_1+m_2)\times n} &= \begin{bmatrix} P_1 \\ P_2A_1 \end{bmatrix} \\ \vec{q} \in \mathbb{Q}^{(m_1+m_2)} &= \begin{bmatrix} \vec{q}_1 \\ \vec{q}_2 - P_2\vec{b}_1 \end{bmatrix} \\ A \in \mathbb{Q}^{n\times n} &= A_2A_1 \\ \vec{b} \in \mathbb{Q}^n &= A_2\vec{b}_1 + \vec{b}_2 \end{aligned}$$

Grâce à la composition, il devient possible de calculer l'effet d'une application répétée d'une même transformation linéaire sur un ensemble d'entiers et de réels représenté par un RVA. Ceci a des applications dans le domaine de la vérification automatisée [Boi98].

4.5 Expressivité des RVA

4.5.1 Généralités

Nous venons de voir comment combiner entre eux les RVA et comment créer des RVA satisfaisant certaines contraintes simples. Mais les RVA peuvent-ils exprimer autre chose ? En d'autres termes, la question est de savoir ce que peuvent et ne peuvent pas représenter les RVA : on parle d'*expressivité* des RVA.

Fondamentalement, un RVA représente un ensemble de vecteurs de réels. Un vecteur de n réels peut quant à lui se voir comme l'attribution d'une valeur à n variables réelles. Donc, un RVA représente un ensemble de contraintes que doivent satisfaire ces n variables pour appartenir à l'ensemble représenté. Dès lors, nous avons besoin d'un langage formel qui permette d'exprimer des propriétés vraies pour n individus pris dans l'ensemble $\mathbb{R}$, c'est-à-dire capable de décrire des *relations* :

Définition 4.9. Une *relation* $\mathcal{R}$ d'arité n sur l'ensemble D est un sous-ensemble du produit cartésien D^n.

Exemple. La relation qui nous intéresse est $\mathcal{R}(A) = \{(x_1, \ldots, x_n) \in \mathbb{R}^n \mid \vec{x} = (x_1, \ldots, x_n) \wedge \vec{x} \in S(A)\}$ pour un RVA donné A. Son domaine est l'ensemble $\mathbb{R}$ des réels. □

La logique du premier ordre est un tel langage formel [GG90]. En fait, nous allons nous limiter au calcul des prédicats sans symboles fonctionnels. Dans les sections suivantes, nous présentons un bref rappel de logique, très fortement inspiré de [BHMV94] et on énoncera sans démonstration le théorème d'expressivité. Ce rappel est indispensable pour la bonne compréhension de la suite de ce travail.

4.5.2 Notions de logique

Définition 4.10. En logique du premier ordre sans symboles fonctionnels, une *structure* est un tuple :

$$\mathcal{S} = \langle D, (c_i)_{i \in I}, (\mathcal{R}_j)_{j \in J} \rangle$$

qui consiste en un *domaine* D qui est un ensemble non vide (dans le cas présent, il s'agit de $\mathbb{R}$), en un ensemble $(c_i)_{i \in I}$ de *constantes* appartenant à D et en un ensemble de relations $(\mathcal{R}_j)_{j \in J}$ de domaine D (chacune possédant une arité). L'ensemble $\{(c_i)_{i \in I}, (\mathcal{R}_j)_{j \in J}\}$ est appelé *lexique* de $\mathcal{S}$.

Exemple. La structure $\langle \mathbb{R}, 0, \leq \rangle$ possède l'ensemble $\mathbb{R}$ des réels comme domaine et est pourvue de la relation d'inégalité. Elle admet également la constante 0. □

Outre les constantes c_i et les relations $\mathcal{R}_j$, on admet dans le lexique de toute structure la présence :

- d'un ensemble de *variables* x, y, z, … ;
- des *connecteurs* usuels $\vee$ (disjonction), $\wedge$ (conjonction), $\neg$ (négation), $\Rightarrow$ (implication) et $\equiv$ (si et seulement si) ;
- des *quantifications* $\forall$ (pour tout) et $\exists$ (il existe) ;
- de la *relation binaire d'égalité* $=$ définie sur le domaine D.

À partir du lexique, on définit :

- un *terme* est une constante c_i ou une variable x ;
- une *formule atomique* est une expression $\mathcal{R}_j(t_1, \ldots, t_n)$, où $\mathcal{R}_j$ est une relation d'arité n et $t_1, \ldots, t_n$ sont des termes.

Une *formule* est alors construite récursivement à partir des formules atomiques :

1. une formule atomique est une formule ;
2. $true$, $false$ sont des formules ;
3. si φ_1 et φ_2 sont des formules, $\neg\varphi_1$, $(\varphi_1 \wedge \varphi_2)$, $(\varphi_1 \vee \varphi_2)$ $(\varphi_1 \Rightarrow \varphi_2)$ et $(\varphi_1 \equiv \varphi_2)$ sont des formules ;
4. si φ est une formule et x une variable, alors $\forall x\varphi$ et $\exists x\varphi$ sont des formules ;
5. toute formule s'obtient ainsi (clause de fermeture).

Certaines notions syntaxiques ressortent de ces définitions :

Définition 4.11. La *portée* d'une quantification est la formule à laquelle la quantification s'applique. Ainsi, dans $\forall x\varphi$ et $\exists x\varphi$, la portée de x est φ. L'occurrence de la variable x dans la quantification $\forall x$ ou $\exists x$ est dite *quantifiée*. Toute occurrence de x dans la portée d'une quantification sur x est dite *liée*. Une variable est *libre* si elle n'est ni quantifiée, ni liée.

Remarque 4.12. On écrira souvent $\varphi(y_1, \ldots, y_m)$ pour mettre en évidence les variables libres qui apparaissent dans φ. Par ailleurs, on permettra d'ajouter ou de retirer des parenthèses si cela n'est pas source d'ambiguïté. □

Si on fixe les valeurs de $y_1, \ldots, y_m$ aux éléments $d_1, \ldots, d_m$ de D dans une formule $\varphi(y_1, \ldots, y_m)$, on dit que l'on a défini une *interprétation* pour φ, ce qui a pour effet d'attribuer une *valeur de vérité* à φ qui peut être V (vraie) ou F (fausse). Cette attribution est définie de manière usuelle [Gri00], si ce n'est que l'on a confondu syntaxe et sémantique des constantes et des relations dans la définition 4.10. [4]

Définition 4.13. Une formule φ définie sur la structure $\mathcal{S}$ est dite *close* ou *fermée* si elle ne contient aucune variable libre. En d'autres termes, toute variable apparaissant dans φ est liée ou quantifiée.

4. De ce fait, une interprétation se résume à la fonction d'interprétation des variables.

Définition 4.14. La *théorie* $Th(\mathcal{S})$ d'une structure $\mathcal{S}$ est l'ensemble des formules closes de $\mathcal{S}$ qui ont V pour valeur de vérité. Une théorie est *décidable* s'il existe un processus mécanique (un algorithme) s'arrêtant toujours et qui peut attribuer à toute formule close de cette structure sa valeur de vérité (i.e. qui peut déterminer si une formule close donnée appartient ou non à $Th(\mathcal{S})$).

4.5.3 Théorème d'expressivité

Nous allons maintenant définir la structure dont toutes les formules peuvent être représentées par des RVA et réciproquement. Pour ce faire, nous définissons les relations suivantes de domaine $\mathbb{R}$:

Z **:** la relation unaire $Z(x)$ vraie si et seulement si x est entier.

$+$ **:** la relation ternaire $+(x,y,z)$ vraie si et seulement si $x + y = z$.

$\leq$ **:** la relation binaire $\leq (x,y)$ vraie si et seulement si $x \leq y$.

X_r **:** la relation ternaire $X_r(x,u,k)$ vraie si et seulement si u est une puissance entière de la base r (i.e. il existe un entier l, éventuellement négatif, tel que $u = r^l$) et s'il existe un encodage de x tel que le chiffre à la position spécifiée par u est k, où $k \in \{0, \ldots, r-1\}$.

Théorème 4.2. [BRW98] Les ensembles de $\mathbb{R}^n$ représentables par un RVA fonctionnant en base r sont exactement ceux qui peuvent être définis par une formule de la structure $\langle \mathbb{R}, Z, +, \leq, X_r \rangle$.

Preuve. La démonstration complète est relativement longue et technique. Nous avons décidé de ne pas la reprendre en annexe, mais le lecteur intéressé peut la trouver dans [BRW98]. □

Corollaire 4.3. La théorie de la structure $\langle \mathbb{R}, Z, +, \leq, X_r \rangle$ est décidable.

Preuve. Soit une formule close φ de cette structure. On peut supposer en toute généralité que φ contient au moins une variable liée (sinon, on introduit arbitrairement une quantification sur une variable inédite). De plus, φ peut être mise sous la forme $(\exists x)\phi(x)$ ou $\neg(\exists x)\phi(x)$, quitte à propager les quantifications vers l'extérieur [Gri00]. Construisons maintenant le RVA A correspondant à ϕ, ce qu'il est possible de faire par le théorème précédent. A accepte le langage vide si et seulement si il n'existe aucune attribution de valeur de vérité à x qui rende ϕ vraie. Par conséquent, si φ possède la forme $(\exists x)\phi(x)$, il suffit de répondre si oui ou non A accepte le langage vide. Si φ possède la forme $\neg(\exists x)\phi(x)$, il suffit d'inverser la réponse. □

Remarque 4.15. Par abus de langage, nous parlerons de la *théorie* $\langle \mathbb{R}, Z, +, \leq, X_r \rangle$ pour désigner la théorie de la structure $\langle \mathbb{R}, Z, +, \leq, X_r \rangle$. □

4.5.4 Extensions syntaxiques

Nous avons introduit les formules de la structure $\langle \mathbb{R},\ Z, +\ ,\ \leq\ ,\ X_r \rangle$ dans leur forme la plus forte. Par exemple, l'égalité de plus de deux termes $x = y_1 + \cdots + y_m$ avec $m > 2$ n'est pas une formule atomique sur cette dernière structure car la relation $+$ est ternaire. Pourtant, cette relation peut être exprimée au départ des relations $+$ et $=$, si on applique la décomposition suivante un nombre suffisant de fois :

$$\exists z(+(y_1,z,x) \wedge z = y_2 + \cdots + y_m)$$

Nous en permettrons donc l'usage. Nous allons autoriser d'autres extensions syntaxiques (en fait, des abréviations) et montrer comment ramener à la forme forte une formule qui en contient. Ce faisant, nous allons retrouver certaines écritures qui nous sont familières :

Les relations d'ordre. Conventionnellement, $x \leq y$ équivaut à $\leq (x,y)$. L'inégalité stricte $x < y$ correspond à $(x \leq y) \wedge \neg(x = y)$, $x > y$ à $\neg(x \leq y)$ et $x \geq y$ à $\neg(x < y)$.

Les constantes.

- $x = 0$ peut être défini par $x + x = x$ car 0 est l'unique neutre pour l'addition dans l'ensemble des réels ;
- $x = 1$ équivaut à $Z(x) \wedge x > 0 \wedge \forall y((Z(y) \wedge y > 0) \Rightarrow x \leq y)$.

Les opérations.

- par convention, on permet l'écriture $x + y = z$ pour $+(x,y,z)$;
- la soustraction $x - y = z$ est définie par $y + z = x$;
- l'égalité entre deux sommes $x_1 + \cdots + x_l = y_1 + \cdots + y_m$ équivaut à la formule $\exists z(z = x_1 + \cdots + x_l \wedge z = y_1 + \cdots + y_m)$;
- l'inégalité entre deux sommes $x_1 + \cdots + x_l \leq y_1 + \cdots + y_m$ revient à $\exists p \exists q(p = x_1 + \cdots + x_l \wedge q = y_1 + \cdots + y_m \wedge p \leq q)$;
- la multiplication par une constante positive $y = a \cdot x$ est définie par $y = \underbrace{x + \cdots + x}_{a}$ et $0 \cdot x$ par la constante 0 ;
- les relations d'égalité et d'inégalité entre deux sommes avec des coefficients rationnels :
$$\sum_{i=1}^{l} \frac{a_i}{b_i} x_i \left\{ \begin{array}{c} = \\ \leq \end{array} \right\} \sum_{i=1}^{m} \frac{c_i}{d_i} y_i$$
sont également définissables par l'intermédiaire des formules :
$$\sum_{i=1}^{l} \left(a_i \cdot \prod_{j=1}^{m} d_j \cdot \prod_{1 \leq j \neq i \leq l} b_j \right) x_i \left\{ \begin{array}{c} = \\ \leq, \geq \end{array} \right\} \sum_{i=1}^{m} \left(c_i \cdot \prod_{j=1}^{l} b_j \cdot \prod_{1 \leq j \neq i \leq m} d_j \right) y_i$$
où il ne faut pas oublier de permuter au besoin le signe d'inégalité. Cette expression est une (in)égalité entre deux sommes ne comprenant que des variables multipliées par des constantes.

Remarque 4.16. Nous avons employé un procédé de traduction direct. Il existe des méthodes plus évoluées qui réduisent le nombre de termes présents dans la formule finale. □

4.6 Problématique

Nous venons de voir que les RVA peuvent exprimer toutes les formules de la structure $\langle \mathbb{R},\ Z, +\ ,\ \leq\ ,\ X_r \rangle$. D'un autre côté, les RVA sont des automates de Büchi et sont difficiles à manier en pratique. Notamment, la complémentation de tels automates n'est pas algorithmiquement gérable car beaucoup trop coûteuse en temps et en mémoire. Or, la complémentation est requise pour le test d'inclusion qui est lui-même nécessaire dans le cadre de la vérification symbolique [Boi98]. À cela s'ajoute le problème de l'absence d'une procédure de minimisation des automates de Büchi, comme nous l'avons déjà évoqué dans l'introduction.

On peut objecter à ce raisonnement le fait que tout automate de Büchi n'est pas un RVA : il ne peut en effet accepter que des encodages valides dans une base donnée. Malheureusement, cette propriété confère aux RVA une forme spéciale qui est trop proche de la forme générale pour autoriser une amélioration de l'efficacité des algorithmes de manipulation.

Devant ce constat, nous allons changer d'optique. Si les automates de Büchi sont trop généraux pour permettre une implémentation informatique des RVA, essayons de restreindre ces automates à une forme particulière qui en autorisera l'implémentation. Le théorème 4.2 nous apprend qu'une telle restriction ne peut se faire qu'en retirant aux RVA la possibilité d'exprimer une des relations suivantes : Z, $+$, $\leq$ et X_r.

Il est clair qu'ôter les relations $+$ et $\leq$ n'aurait aucun sens : on perdrait le fait que les RVA peuvent exprimer des formules de l'arithmétique linéaire, ce qui est nécessaire dans le cadre de la vérification de programmes intégrant des transformations linéaires. De même, supprimer Z serait contraire à un de nos objectifs, qui est de pouvoir représenter des ensembles périodiques de réels. Il semble donc naturel de choisir d'éliminer la relation X_r. Ce choix n'est pas arbitraire car X_r est une « étrange » relation, pour trois raisons :

1. elle possède une sémantique contre-intuitive ;
2. elle possède un comportement qui dépend de la base utilisée ;
3. elle est inutile dans la plupart des applications (y compris pour la représentation d'ensembles de configurations accessibles par un programme).

À partir du chapitre suivant, nous allons étudier les propriétés qui découlent de cette suppression. Cette caractérisation va se faire en termes de *topologie*, discipline mathématique qui est l'objet du prochain chapitre.

Chapitre 5

Topologie

B. Riemann a écrit que la topologie est la partie des mathématiques qui « fait abstraction de la mesure en étudiant seulement les rapports de position et d'inclusion. » Étymologiquement, la topologie est la science de la position (du grec $\tau o \pi o \sigma$, le lieu, le site et du grec $\lambda o \gamma o \sigma$, le discours).

Il peut sembler à première vue étonnant de s'intéresser à la topologie dans un cadre algorithmique. Mais quand on y réfléchit bien, les ensembles de réels définissables par une formule logique ne sont pas répartis de manière quelconque dans l'espace. Il devient dès lors intéressant de mieux comprendre cette répartition pour dégager des propriétés qui pourraient devenir utiles.

Plus précisément, l'objet de ce chapitre est de montrer que les ensembles que l'on peut définir dans la structure logique $\langle \mathbb{R},\ Z,\ +\ ,\ \leq \rangle$ possèdent une propriété topologique très particulière que nous utiliserons dans les chapitres ultérieurs afin de permettre l'utilisation pratique des RVA.

5.1 Espaces topologiques, ouverts et fermés

Dans cette section, nous allons introduire les bases théoriques de la topologie. Un exposé plus exhaustif peut être trouvé dans [Bou74].

Nous travaillons dans un espace E qui peut être quelconque (en particulier, il peut s'agir de $\mathbb{R}^n$ ou de l'ensemble des mots infinis sur un alphabet donné). Nous définissons maintenant une *topologie* sur cet ensemble E, qui devient alors un *espace topologique*. Cela se fait en précisant quels sont les *ouverts* de E (terme à définir).

Définition 5.1. $(E,\mathcal{T})$[1] est un *espace topologique* si $\mathcal{T}$ est un sous-ensemble de 2^E (dont les éléments sont les *ouverts*) possédant les propriétés suivantes :

1. $\phi \in \mathcal{T}$ et $E \in \mathcal{T}$,
2. $O_1,O_2 \in \mathcal{T} \Rightarrow O_1 \cap O_2 \in \mathcal{T}$,

1. $(E,\mathcal{T})$ peut être lu « E muni de la topologie $\mathcal{T}$ ».

3. $\mathcal{O} \subseteq \mathcal{T} \Rightarrow \bigcup_{O \in \mathcal{O}} O \in \mathcal{T}$.

Remarque 5.2. On déduit de (2) qu'une intersection finie d'ouverts est un ouvert, et de (3) qu'une union quelconque d'ouverts est un ouvert. □

Exemple. Si l'on prend $\mathcal{T} = 2^E$, on obtient la *topologie discrète* : tous les sous-ensembles de E sont considérés comme des ouverts. Il s'agit de la topologie maximale : tout ce qui peut être un ouvert est un ouvert. À l'opposé se trouve la *topologie triviale* où $\mathcal{T} = \{\phi, E\}$: seuls ϕ et E sont ouverts, ce qui est requis par la définition. Ces deux topologies existent pour tout ensemble E. □

Définition 5.3. A est un ensemble *fermé* dans l'espace topologique $(E, \mathcal{T})$ si et seulement si son complément par rapport à E (noté A^c) est un ouvert.

Théorème 5.1. Dans $(E, \mathcal{T})$, E, ϕ, les unions finies de fermés et les intersections quelconques de fermés sont des fermés.

Preuve. E est fermé car $E^c = \phi$ est un ouvert. De même, ϕ est fermé car $\phi^c = E$ est un ouvert. Pour une union finie de fermés, on utilise la règle de De Morgan : $(C_1 \cup \cdots \cup C_n)^c = C_1^c \cap \cdots \cap C_n^c$. Puisque les C_i sont des fermés, les C_i^c sont des ouverts. Une intersection finie d'ouverts reste un ouvert, donc le complément de $C_1 \cup \cdots \cup C_n$ est ouvert, donc cet ensemble est un fermé. Pour une famille quelconque $\{C_i\}_{i \in A}$ de fermés, on a :

$$\left(\bigcap_{i \in A} C_i \right)^c = E \setminus \bigcap_{i \in A} C_i = E \setminus \bigcap_{i \in A} (E \setminus C_i^c) = E \setminus \left(E \setminus \bigcup_{i \in A} C_i^c \right) = \bigcup_{i \in A} C_i^c$$

Ce dernier terme est une union quelconque d'ouverts et est donc bien un ouvert. □

Remarque 5.4. Tout sous-ensemble de E n'est pas nécessairement un ouvert ou un fermé. Ainsi, si $E = \{a, b\}$ est muni de la topologie triviale, $\{a\}$ n'est ni un ouvert ni un fermé, tout comme $\{b\}$. De plus, un ensemble peut être à la fois ouvert et fermé (c'est le cas par exemple pour E et ϕ quelle que soit la topologie définie sur E). □

La notion de *voisinage* est capitale en analyse : elle capture la notion de proximité d'un point à un autre et ce, indépendamment de toute idée de distance.

Définition 5.5. $V \in \mathcal{T}$ est appelé *voisinage* de x si $x \in V$. En d'autres termes, un voisinage de x est tout ouvert contenant x.

Théorème 5.2. Un ensemble O est ouvert si et seulement tout point $x \in O$ admet un voisinage $V(x)$ entièrement compris dans cet ensemble.

Preuve. ($\Rightarrow$) Supposons que O est ouvert. Pour tout point $x \in O$, O est un voisinage de x car c'est un ouvert et il contient x. On pose alors $V(x) = O$.

($\Leftarrow$) Posons $X = \bigcup_{x \in O} V(x)$. C'est bien un ouvert car une union arbitraire d'ouverts est un ouvert. De plus, $X \subseteq O$, car par hypothèse tous les $V(x)$ sont inclus dans O. Par ailleurs, $O \subseteq X$. En effet, soit $x \in O$: x appartient aussi à X car $x \in V(x)$ par définition d'un voisinage de x. On conclut que $X = O$. Par conséquent, O est ouvert car X est lui-même ouvert. □

5.2 Hiérarchie de Borel

Les topologistes utilisent souvent une autre notation que $\mathcal{T}$ pour désigner l'ensemble des ouverts : cette classe est conventionnellement désignée par G. La notation reste valable quel que soit l'espace topologie $(E,\mathcal{T})$ considéré : G peut par exemple désigner, selon le contexte, l'ensemble des ouverts de la topologie triviale ou bien celui de la topologie discrète. De même, on prend la liberté d'écrire F pour l'ensemble des fermés d'un espace topologique donné.

Nous avons vu qu'une union quelconque d'ouverts est un ouvert. Il n'existe pas de théorème équivalent pour l'intersection : une intersection infinie d'ouverts n'est pas nécessairement un ouvert. Cela tient même pour une intersection *dénombrable* d'ouverts. Borel a convenu de dénoter cette dernière classe d'ensembles par G_δ, δ signifiant « intersection dénombrable ». De même, les unions dénombrables de fermés ont été notées F_σ, σ signifiant « union dénombrable ».

Cette structuration des ensembles peut être poursuivie à l'infini : on parle de la *hiérarchie des classes de Borel*. Ainsi, $F_{\sigma\delta\sigma}$ signifie « les unions dénombrables d'intersections dénombrables d'unions dénombrables de fermés ». Il est facile de voir que $X_{\dots\sigma\sigma\dots} = X_{\dots\sigma\dots}$ où $X \in \{F,G\}$. En effet, on a :

$$S = \bigcup_{i \in \mathbb{N}} \bigcup_{j \in \mathbb{N}} S_{ij} = \bigcup_{(i,j) \in \mathbb{N}^2} S_{ij}$$

or, $\mathbb{N}^2$ est dénombrable, donc S est elle-même une union dénombrable des ensembles S_{ij}. Le même résultat tient pour les redoublements de δ. On en déduit qu'une nouvelle classe de Borel est créée à chaque alternance entre un symbole σ et un symbole δ. Selon que l'on part de F ou de G, on obtient ainsi deux suites infinies de classes :

$$F,\ F_\sigma,\ F_{\sigma\delta},\ F_{\sigma\delta\sigma},\ F_{\sigma\delta\sigma\delta},\ \cdots$$

$$G,\ G_\delta,\ G_{\delta\sigma},\ G_{\delta\sigma\delta},\ G_{\delta\sigma\delta\sigma},\ \cdots$$

Remarque 5.6. On pourrait croire qu'il existe une relation d'inclusion entre les différents niveaux de la hiérarchie. Ce n'est toutefois généralement pas le cas [PP01]. Par contre, une telle relation d'inclusion tient toujours dans les *espaces métriques*, introduits à la section 5.3. □

Dans la suite de cet ouvrage, nous ne nous intéresserons qu'aux classes G, F, G_δ et F_σ. Nous introduisons finalement un nouveau concept :

Définition 5.7. Une *combinaison booléenne finie* $\mathcal{B}(S_i)$ d'ensembles S_i est un en-

semble obtenu par une application en ordre quelconque mais en nombre fini d'opérations d'union, d'intersection et de complémentation au départ des ensembles S_i.

5.3 Espaces métriques

Ceci clôture notre étude de la topologie générale. Nous allons maintenant nous intéresser à une classe particulièrement importante d'espaces topologiques : les *espaces métriques*. Ce sont eux que l'on rencontre le plus fréquemment et qui nous seront utiles par la suite.

5.3.1 Généralités

Un espace E est métrique si l'on peut définir une *distance* entre toute paire d'éléments de E. Par exemple, $\mathbb{R}$ devient un espace métrique si on le munit de la traditionnelle distance définie par $d(x,y) = |x - y|$.

Définition 5.8. E désignant un ensemble non vide, (E,d) est un *espace métrique* si $d : E^2 \mapsto \mathbb{R}^+$ vérifie les propriétés :

1. $d(x,y) = 0 \Leftrightarrow x = y$,
2. $\forall x,y \in E : d(x,y) = d(y,x)$,
3. $\forall x,y,z \in E : d(x,z) \leq d(x,y) + d(y,z)$.

Exemple. La distance « à vol d'oiseau » dans le plan (distance *euclidienne*) vérifie bien entendu les deux premières conditions. Quant à la troisième, elle est l'expression mathématique du principe selon lequel le plus court chemin d'un point à un autre est la ligne droite. □

Définition 5.9. Une *boule ouverte* de centre x et de rayon $\varepsilon > 0$ est l'ensemble :

$$B(x,\varepsilon) = \{y \in E \mid d(x,y) < \varepsilon\}$$

Exemple. Dans l'ensemble des réels, la boule ouverte $B(5,1)$ correspond à l'intervalle ouvert $]4,6[$. D'une manière plus générale, tout intervalle ouvert $]x,y[$ avec x et $y \in \mathbb{R}$ est une boule ouverte. □

Remarque 5.10. Toute boule ouverte $B(x,\varepsilon)$ contient x. De plus, aucune boule ouverte ne se résume au seul élément x. □

Définition 5.11. La *topologie métrique* $\mathcal{T}$ induite par d sur E est l'ensemble des ouverts que l'on peut obtenir en prenant une réunion quelconque de boules ouvertes. On dit que les boules ouvertes forment une *base* de $(E,\mathcal{T})$.[2]

2. Nous avons défini un espace topologique $(E,\mathcal{T})$ au départ de la métrique utilisée sur E. Rien n'interdit d'employer sur E une toute autre topologie en explicitant $\mathcal{T}$ comme nous l'avons fait précédemment.

Exemple. L'intervalle $I =]1, +\infty[$ n'est pas une boule ouverte (on peut le voir en constatant que I n'a pas de « centre »). Il s'agit toutefois d'un ouvert : par exemple, $I = \bigcup_{x=1}^{\infty}]1,x[$, or on a déjà dit que $]1,x[$ est une boule ouverte. □

5.3.2 Classes de Borel dans un espace métrique

Le théorème suivant est particulièrement important, car il explique la forme que revêt la hiérarchie de Borel dans un espace métrique :

Théorème 5.3. Dans une topologie métrique, tout ensemble fermé est une intersection dénombrable d'ouverts et tout ensemble ouvert est une union dénombrable de fermés. En d'autres termes, $F \subseteq G_\delta$ et $G \subseteq F_\sigma$.

Preuve. La démonstration est reportée dans l'annexe A, section A.2.2. □

La remarque 5.6 nous a appris qu'il n'y a en général pas de structure d'inclusion entre les niveaux de la hiérarchie de Borel. Par contre, le théorème ci-dessus implique par induction que c'est le cas dans un espace métrique. Le lecteur intéressé peut en trouver une preuve dans [Sta97, PP01]. Nous ne la reportons pas ici car seuls les ensembles G_δ et F_σ nous intéressent. De ce théorème, il est également possible de tirer une conséquence sur les combinaisons booléennes finies d'ouverts et de fermés :

Théorème 5.4. Dans une topologie métrique, toute combinaison booléenne finie d'ouverts et de fermés appartient à la classe $G_\delta \cap F_\sigma$.

Preuve. Se reporter à l'annexe A, section A.2.3. □

Ce théorème peut également être généralisé aux niveaux supérieurs de la hiérarchie de Borel. En mettant tous ces résultats ensemble, on obtient le dessin de la figure 5.1 (page 54), où une flèche indique une relation d'inclusion « $\subseteq$ ».

Remarque 5.12. On a $\mathcal{B}(F) = \mathcal{B}(G)$ car par dualité, toute combinaison booléenne finie d'ouverts peut se ramener à une combinaison booléenne de fermés et réciproquement. □

5.3.3 Topologie de $\mathbb{R}^n$

Nous allons particulariser les définitions générales de la topologie à l'espace vectoriel $\mathbb{R}^n$. Usuellement, on munit $\mathbb{R}^n$ de la *métrique euclidienne canonique*, qui correspond à la notion classique de distance vectorielle :

$$d(x,y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

Il est facile de voir que cette fonction est bien une métrique [Éti97]. Les boules ouvertes $B_d(x,\varepsilon)$ engendrées par d sont des sphères « sans pelure » de rayon ε centrées sur x. La topologie métrique induite par d sur $\mathbb{R}^n$ est appelée *topologie naturelle*.

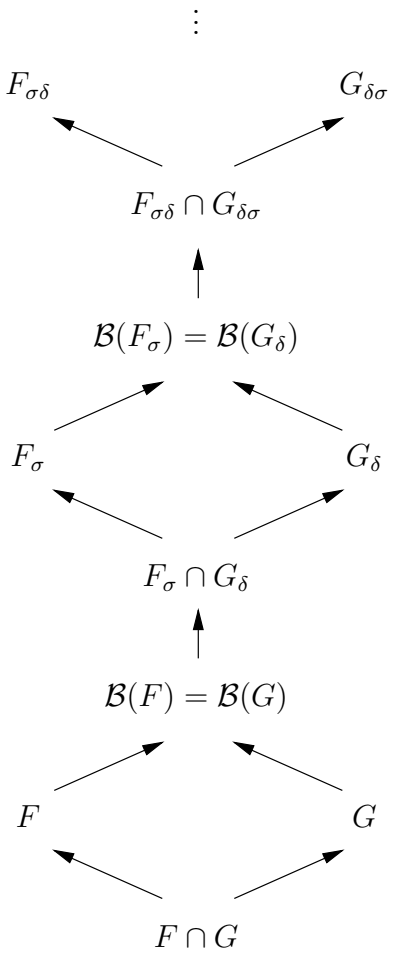


FIG. 5.1 – *Premiers niveaux de la hiérarchie de Borel dans une topologie métrique.*

5.4 Ensembles arithmétiques

5.4.1 Généralités

Arrivés au terme des rappels de topologie, nous allons maintenant mettre en œuvre les théorèmes précédents afin de prouver que la restriction sur l'expressivité des RVA évoquée à la section 4.6 permet d'obtenir uniquement des ensembles de $\mathbb{R}^n$ appartenant à la classe de Borel $G_\delta \cap F_\sigma$.

5.4.2 Caractérisation topologique

Lemme 5.5. Toute formule de la structure $\langle \mathbb{R}, Z, +, \leq \rangle$ peut être mise sous une forme équivalente qui ne contient plus la relation $Z(x)$ mais à laquelle on a ajouté les deux nouvelles quantifications $(\exists x \in \mathbb{Z})$ et $(\forall x \in \mathbb{Z})$ portant sur des entiers et non plus sur des réels. Pour éviter les ambiguïtés, les quantificateurs $(\exists x)$ et $(\forall x)$ de la structure initiale seront quant à eux respectivement écrits $(\exists x \in \mathbb{R})$ et $(\forall x \in \mathbb{R})$.

Preuve. On remplace simplement toute formule atomique $Z(x)$ par la sous-formule $(\exists z \in \mathbb{Z})(z = x)$. Réciproquement, $(\exists x \in \mathbb{Z})(\varphi(x,y_1,\ldots,y_m))$ est équivalente à $(\exists x \in \mathbb{R})(Z(x) \wedge \varphi(x,y_1,\ldots,y_m))$. De même, $(\forall x \in \mathbb{Z})(\varphi(x,y_1,\ldots,y_m))$ est équivalente à $(\forall x \in \mathbb{R})(\neg Z(x) \vee \varphi(x,y_1,\ldots,y_m))$. □

Théorème 5.6. [BJW01] Soit un ensemble $S \subseteq \mathbb{R}^n$ avec $n > 0$, défini par une formule de la structure $\langle \mathbb{R}, Z, +, \leq \rangle$. Cet ensemble appartient à la classe $G_\delta \cap F_\sigma$ de la topologie naturelle de $\mathbb{R}^n$.

Preuve. Soit une formule φ de $\langle \mathbb{R}, Z, +, \leq \rangle$. L'idée de la démonstration est de séparer le traitement des parties entière et fractionnaire de tous les nombres réels apparaissant dans la formule : un nombre réel x va être décomposé en $x_I + x_F$, où $x_I \in \mathbb{Z}$ et $x_F \in [0,1[$. Cette décomposition se rapproche de ce que nous avions réalisé à la section 4.4.1 pour générer un RVA représentant les solutions à une contrainte linéaire.

Nous allons commencer par réaliser cette séparation sur les variables libres apparaissant dans φ. Pour ce faire, on remplace chaque variable libre x par deux variables libres x_I et x_F, représentant respectivement les parties entière et fractionnaire de x. Ceci revient à remplacer dans φ chaque occurrence de x par la somme $x_I + x_F$. De plus, on ajoute à φ la conjonction $Z(x_I) \wedge 0 \leq x_F \wedge x_F < 1$.

Il est possible de faire de même pour les variables quantifiées, quant à elles séparées en appliquant les règles :

$$(\exists x \in \mathbb{R})\phi \longrightarrow (\exists x_I \in \mathbb{Z})(\exists x_F \in \mathbb{R})(0 \leq x_F \wedge x_F < 1 \wedge \phi[x/x_I + x_F]) \quad (5.1)$$

$$(\forall x \in \mathbb{R})\phi \longrightarrow (\forall x_I \in \mathbb{Z})(\forall x_F \in \mathbb{R})(x_F < 0 \vee 1 \leq x_F \vee \phi[x/x_I + x_F]) \quad (5.2)$$

$$(\exists x \in \mathbb{Z})\phi \longrightarrow (\exists x_I \in \mathbb{Z})\phi[x/x_I] \quad (5.3)$$

$$(\forall x \in \mathbb{Z})\phi \longrightarrow (\forall x_I \in \mathbb{Z})\phi[x/x_I] \quad (5.4)$$

$$
\begin{array}{rcl}
x_I = y_I & \longrightarrow & x_I = y_I \\
x_I = (y_I + y_F) & \longrightarrow & x_I = y_I \wedge y_F = 0 \\
(x_I + x_F) = (y_I + y_F) & \longrightarrow & x_I = y_I \wedge x_F = y_F \\
x_I = y_I + z_I & \longrightarrow & x_I = y_I + z_I \\
x_I = y_I + (z_I + z_F) & \longrightarrow & x_I = y_I + z_I \wedge z_F = 0 \\
x_I = (y_I + y_F) + (z_I + z_F) & \longrightarrow & (x_I = y_I + z_I \wedge y_F + z_F = 0) \\
 & & \vee (x_I = y_I + z_I + 1 \wedge y_F + z_F = 1) \\
(x_I + x_F) = y_I + z_I & \longrightarrow & x_I = y_I + z_I \wedge x_F = 0 \\
(x_I + x_F) = y_I + (z_I + z_F) & \longrightarrow & x_I = y_I + z_I \wedge x_F = z_F \\
(x_I + x_F) = (y_I + y_F) + (z_I + z_F) & \longrightarrow & (x_I = y_I + z_I \wedge x_F = y_F + z_F) \\
 & & \vee (x_I = y_I + z_I + 1 \wedge x_F = y_F + z_F - 1) \\
x_I \leq y_I & \longrightarrow & x_I \leq y_I \\
x_I \leq (y_I + y_F) & \longrightarrow & x_I \leq y_I \\
(x_I + x_F) \leq y_I & \longrightarrow & x_I < y_I \vee (x_I = y_I \wedge x_F = 0) \\
(x_I + x_F) \leq (y_I + z_I) & \longrightarrow & x_I < y_I \vee (x_I = y_I \wedge x_F \leq y_F)
\end{array}
$$

TAB. 5.1 – *Règles arithmétiques de décomposition des formules atomiques.*

où ϕ est une sous-formule de φ et où $\phi[x/f]$ désigne le remplacement dans ϕ de toutes les occurrences de x par la sous-formule f. Ces transformations n'ont aucun impact sur l'ensemble représenté par la formule (il suffit de raisonner sémantiquement), si ce n'est que les parties entières et fractionnaires de toutes les variables ont été explicitées.

Il est en outre toujours possible de remplacer φ par une formule qui est dans sa forme la plus forte (i.e. ne contenant plus que les relations Z, $+$, $\leq$) conformément aux règles de la section 4.5.4. Toutes les formules atomiques de φ sont par conséquent de la forme $p = q$, $p = q + r$ ou $p \leq q$. Les transformations sur les variables décrites ci-dessus assurent en outre que p, q et r sont soit :

- des constantes entières c (présentes dans la formule initiale) ;
- des variables entières x_I (obtenues par les règles (5.3) et (5.4)) ;
- des sommes d'une partie entière et fractionnaire $x_I + x_F$ (règles (5.1) et (5.2)).

Grâce au lemme 5.5, on peut finalement convertir toutes les formules atomiques $Z(x)$ qui subsistent dans φ en des quantifications sur des variables entières.

La seconde étape est de décomposer les formules atomiques en atomes distincts ne contenant plus que des occurrences de variables toutes entières ou fractionnaires. On utilise pour ce faire des règles arithmétiques simples présentées au tableau 5.1. Ces règles exploitent l'appartenance de x_I à $\mathbb{Z}$ et de x_F à [0,1[pour toute variable x (libre ou liée). Le tableau est exhaustif, à la commutativité des membres (pour la relation « = ») et des termes (pour la relation « + ») près. Il ne reprend pas les règles

pour les constantes, car elles sont identiques à celles pour les variables entières.

Suite à cette transformation, toute formule atomique de φ est soit une formule ϕ_I impliquant seulement des variables entières, soit une formule ϕ_F impliquant uniquement des variables fractionnaires.

Nous allons maintenant propager les quantifications vers l'intérieur de manière à ce qu'elles ne s'appliquent qu'à des sous-formules faisant exclusivement intervenir soit des variables entières, soit des variables fractionnaires (pour abréger, nous parlerons respectivement de sous-formules entières et fractionnaires). On va raisonner par induction sur la structure de la formule pour montrer que ceci est possible, sachant comme cas de base que les formules atomiques de φ sont déjà soit des sous-formules entières, soit des sous-formules fractionnaires grâce aux règles que l'on vient d'appliquer. La démarche à suivre est décrite ci-dessous :

1. On élimine tous les connecteurs autres que $\neg$, $\wedge$ et $\vee$.
2. On propage aussi loin que possible les négations vers l'intérieur et on élimine les doubles négations :

$$\begin{array}{llll} \neg(\forall x \in \mathbb{R})\phi & \longrightarrow & (\exists x \in \mathbb{R})\neg\phi & \neg(\forall x \in \mathbb{Z})\phi \longrightarrow (\exists x \in \mathbb{Z})\neg\phi \\ \neg(\exists x \in \mathbb{R})\phi & \longrightarrow & (\forall x \in \mathbb{R})\neg\phi & \neg(\exists x \in \mathbb{Z})\phi \longrightarrow (\forall x \in \mathbb{Z})\neg\phi \end{array}$$
$$\neg\neg\phi \longrightarrow \phi$$

3. On propage les quantifications vers l'intérieur, en commençant par les quantifications les plus internes (c'est ici qu'intervient l'induction structurelle) :
   - Pour une quantification existentielle $(\exists x_I \in \mathbb{Z})\phi$ ou $(\exists x_F \in \mathbb{R})\phi$, on écrit ϕ en forme normale disjonctive, où l'on considère une sous-formule quantifiée comme un (pseudo-)littéral. Pour ce faire, on peut appliquer un simple algorithme de mise en forme normale disjonctive (voir [GG90] ou la section A.2.3 pour la même méthode dans un contexte ensembliste). Il est bien connu que l'on peut distribuer une quantification existentielle sur les termes d'une disjonction. La quantification est alors démultipliée et chacune s'applique à une conjonction ϕ' de pseudo-littéraux.
     Chacun de ces pseudo-littéraux est, par hypothèse inductive, soit une sous-formule entière, soit une sous-formule fractionnaire. ϕ' peut donc être séparée en deux groupes ϕ'_I et ϕ'_F reprenant respectivement toutes les sous-formules entières et fractionnaires. Ensuite, on applique selon le contexte :

$$\begin{array}{rcl} (\exists x_I \in \mathbb{Z})(\phi'_I \wedge \phi'_F) & \longrightarrow & (\exists x_I \in \mathbb{Z})(\phi'_I) \wedge \phi'_F \\ (\exists x_F \in \mathbb{R})(\phi'_I \wedge \phi'_F) & \longrightarrow & \phi'_I \wedge (\exists x_F \in \mathbb{R})(\phi'_F) \end{array}$$

     En d'autres termes, une quantification sur une variable entière n'a aucune influence sur une formule ne contenant que des parties fractionnaires. De même, une quantification fractionnaire n'a aucun impact sur une formule entière. Ce procédé a bien l'effet recherché : la quantification existentielle ne s'applique plus qu'à une sous-formule qui est soit entière pour une quantification entière, soit fractionnaire pour une quantification réelle.

- Le procédé est sensiblement identique pour les quantifications universelles $(\forall x_I \in \mathbb{Z})\phi$ ou $(\forall x_F \in \mathbb{R})\phi$. Synthétiquement, on applique l'algorithme suivant :
  (a) on met ϕ en forme normale conjonctive ;
  (b) on distribue la quantification universelle sur la conjonction : la quantification est démultipliée, chacune s'appliquant à une disjonction ;
  (c) chaque terme d'une disjonction est par induction une formule entière ou une formule fractionnaire : on divise donc chaque disjonction en deux groupes ϕ'_I et ϕ'_F ;
  (d) on applique les règles :

$$\begin{aligned}(\forall x_I \in \mathbb{Z})(\phi'_I \ \vee \ \phi'_F) \ &\longrightarrow \ (\forall x_I \in \mathbb{Z})(\phi'_I) \ \vee \ \phi'_F \\ (\forall x_F \in \mathbb{R})(\phi'_I \ \vee \ \phi'_F) \ &\longrightarrow \ \phi'_I \ \vee \ (\forall x_F \in \mathbb{R})(\phi'_F)\end{aligned}$$

4. En répétant cette opération sur toutes les quantifications, on arrive à une formule φ' qui est une combinaison booléenne finie de sous-formules quantifiées, de formules atomiques et de négations de formules atomiques. De plus, chacune de ces sous-formules et formules atomiques ne fait intervenir que des variables entières ou fractionnaires. En résumé, nous avons obtenu une formule φ' équivalente à φ, mais sous la forme d'une combinaison booléenne finie :

$$\varphi' \equiv \mathcal{B}(\phi_I^{(1)},\phi_I^{(2)},\ldots,\phi_I^{(m)},\phi_F^{(1)},\phi_F^{(2)},\ldots,\phi_F^{(m')}) \tag{5.5}$$

où chaque sous-formule $\phi_I^{(i)}$ et $\phi_F^{(j)}$ implique respectivement seulement des variables entières ou fractionnaires.

Revenons maintenant aux variables libres, qui sont en nombre $k \leq n$, et considérons les variables entières. Celles-ci sont $x_I^{(1)},\ldots,x_I^{(k)}$ pour reprendre les notations introduites plus haut. Dans les sous-formules $\phi_I^{(i)}$, n'interviennent que ces variables libres entières et des variables entières quantifiées. Par conséquent, le fait d'attribuer une valeur aux $x_I^{(j)}$ suffit à attribuer une valeur de vérité à chaque sous-formule $\phi_I^{(i)}$.

Nous ne connaissons toutefois rien de cette valeur : tout élément de $\mathbb{Z}$ pourrait être introduite pour une variable $x_I^{(j)}$ lors de l'évaluation de φ'. Nous allons donc réaliser une grande disjonction pour laisser tous les choix possibles tout en attribuant une valeur de vérité aux $\phi_I^{(i)}$. Plus précisément, nous avons l'équivalence :

$$\varphi'(x_I^{(1)},x_F^{(1)},\ldots,x_I^{(k)},x_F^{(k)}) \equiv \bigvee_{(a_1,\ldots,a_k)\in\mathbb{Z}^k} \left(\begin{array}{c} (x_I^{(1)},\ldots,x_I^{(k)}) = (a_1,\ldots,a_k) \ \wedge \\ \varphi'(a_1,x_F^{(1)},a_2,x_F^{(2)},\ldots,a_k,x_F^{(k)}) \end{array} \right) \tag{5.6}$$

Nous avons tout simplement énuméré tous les choix possibles pour les variables libres entières. Ce faisant, nous savons maintenant que pour chacun de ces choix, toute sous-formule $\phi_I^{(i)}$ de (5.5) est identiquement vraie ou fausse.

Par conséquent, dans chaque terme de la grande disjonction, tout terme de la combinaison booléenne correspondant à φ' est soit $true$, soit $false$, soit une sous-formule ϕ_F portant exclusivement sur des variables fractionnaires. De ce fait, chaque

terme de la grande disjonction peut se voir comme une formule ne faisant intervenir que des variables réelles. Il s'agit donc d'un terme que l'on peut exprimer dans la structure $\langle \mathbb{R},\ Z, +\ , \leq \rangle$ dont on a ôté les variables entières, soit une formule de la structure $\langle \mathbb{R},\ 1, +\ , \leq \rangle$.[3]

Or, il est possible de montrer que tout ensemble décrit par $\langle \mathbb{R},\ 1, +, \leq \rangle$ appartient à la classe F_σ (théorème A.9 de l'annexe A, section A.2.4). Donc, chaque terme de (5.6) décrit un ensemble qui est une union dénombrable de fermés dans la topologie naturelle de $\mathbb{R}^n$.

Il faut maintenant remarquer que le nombre de termes dans la grande disjonction est infini, mais il reste dénombrable car $\mathbb{Z}^k$ est un ensemble dénombrable. Par conséquent, la formule (5.6) est une union dénombrable d'unions dénombrables de fermés, soit une union dénombrable de fermés (cf. section 5.2). Donc, φ décrit un ensemble F_σ, ce qui prouve la première partie de l'énoncé.

Soit une formule φ de $\langle \mathbb{R},\ Z, +\ , \leq \rangle$. $\neg\varphi$ reste une formule de cette structure. Or, nous venons de montrer que dans ce cas, l'ensemble décrit par $\neg\varphi$ appartient à F_σ. Par dualité, φ décrit un ensemble G_δ, ce qui achève la démonstration. □

5.4.3 Commentaires

Nous appellerons la théorie de la structure $\langle \mathbb{R},\ Z, +\ , \leq \rangle$, théorie additive des entiers et des réels. Il s'agit de l'extension de l'arithmétique de Presburger[4] qui contient à la fois des variables entières et réelles. Nous avons vu que les RVA permettent de décider la théorie $\langle \mathbb{R},\ Z, +\ , \leq\ , X_r \rangle$. On en déduit que la théorie additive des entiers et des réels (et *a fortiori* l'arithmétique de Presburger) est décidable car elle forme un sous-ensemble de cette structure.

Le résultat topologique que nous venons de présenter va nous permettre d'améliorer la procédure de décision que l'on peut appliquer à cette structure. Pour exploiter ce résultat, il est maintenant nécessaire de comprendre comment la classe $G_\delta \cap F_\sigma$ se traduit dans le monde des automates sur mots infinis.

3. Il faut inclure la constante 1 dans la structure $\langle \mathbb{R}, +\ , \leq \rangle$ car pour définir 1 dans $\mathbb{R}$, il faut nécessairement disposer de la relation $Z(x)$ (x est entier).

4. L'arithmétique de Presburger se définit fréquemment comme la théorie logique de la structure $\langle \mathbb{N}, +\ , \leq \rangle$. Elle peut se voir comme l'arithmétique usuelle sans l'opération de multiplication. La relation $\leq$ (x,y) n'est pas vraiment nécessaire, car elle peut se déduire de $+(x,y,z)$. Notons enfin que l'extension de l'arithmétique de Presburger au domaine $\mathbb{Z}$ des entiers est triviale.

Chapitre 6

Topologie et automates

Dans le chapitre précédent, nous avons mis en évidence le fait que les formules de la structure logique $\langle \mathbb{R}, Z, +, \leq \rangle$ définissent des ensembles de points appartenant à la classe de Borel $G_\delta \cap F_\sigma$ dans l'espace euclidien $\mathbb{R}^n$. Le but du présent chapitre est de comprendre ce que deviennent les ensembles $G_\delta \cap F_\sigma$ quand ils sont encodés sous la forme de mots infinis.

Le résultat majeur, qui peut sembler évident, est que les ensembles de Σ^ω qui représentent des ensembles $G_\delta \cap F_\sigma$ de $\mathbb{R}^n$ sont eux-mêmes des ensembles $G_\delta \cap F_\sigma$. Si l'énoncé est intuitivement évident, la preuve l'est moins car elle nécessite une transposition de l'espace euclidien dans l'espace des mots infinis.

Afin de pouvoir exploiter cette caractérisation, nous allons ensuite étudier la topologie des langages acceptés par des $\omega-$automates. Cette topologie est bien comprise, comme on peut s'en rendre compte en parcourant [Lan69, WS75, MS97]. À la fin de ce chapitre, nous construirons une famille très particulière d'automates sur mots infinis qui acceptent exactement les langages appartenant à la classe $G_\delta \cap F_\sigma$.

6.1 Correspondance entre $\mathbb{R}^n$ et Σ^ω

6.1.1 Topologie de Σ^ω

Avant toute chose, il faut pouvoir munir d'une topologie l'espace Σ^ω des mots infinis sur un alphabet quelconque Σ. Nous savons qu'il suffit pour ce faire de définir une distance sur cet espace (cf. section 5.3).

Intuitivement, quand nous comparons deux mots, nous les parcourons simultanément en partant de leur début. Plus la première différence entre les deux mots est loin, plus les mots nous paraissent semblables. En d'autres termes, deux mots infinis sont d'autant plus « proches » l'un de l'autre que leur préfixe commun est long. En accord avec cette intuition, on munit Σ^ω d'une métrique qui est une fonction décroissante de la longueur de leur préfixe commun :

$$d(u,v) = 2^{-r} \in \mathbb{R}^+, \text{ avec } r = \min\{n \mid u_n \neq v_n\}$$

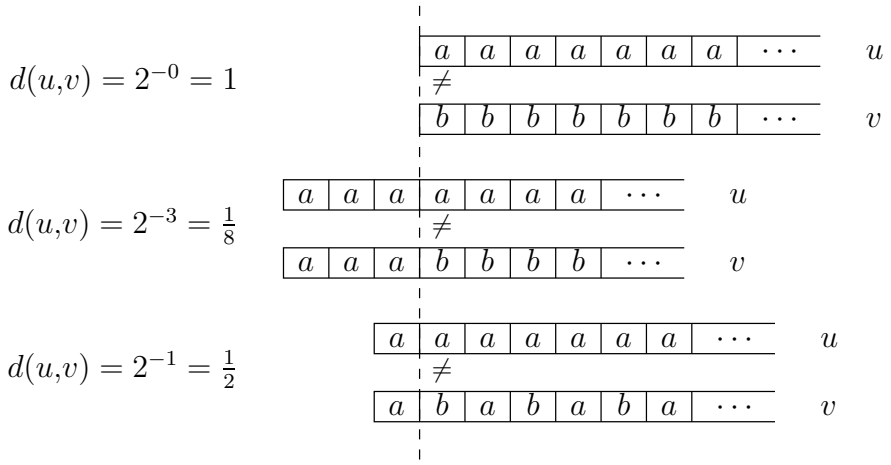


Fig. 6.1 – *Distance sur les mots infinis.*

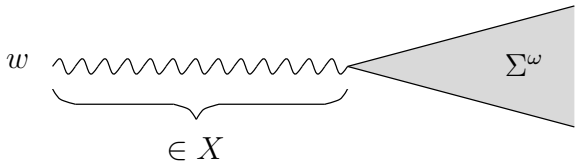


Fig. 6.2 – *Ouverts dans la topologie des mots infinis.*

où l'on pose par convention que $\min \phi = \infty$ et que $2^{-\infty} = 0$. Ainsi, quand deux mots sont identiques, leur distance est nulle[1].

Exemple. Sur l'alphabet $\Sigma = \{a,b\}$, on a :

$$\begin{aligned} d(a^\omega, b^\omega) &= d(a^\omega, b \cdot a^\omega) = d((a \cdot b)^\omega, (b \cdot a)^\omega) = 1, \\ d(a^\omega, a^k \cdot b^\omega) &= 2^{-k} \quad \text{pour tout } k \in \mathbb{N}, \\ d(a^\omega, (a \cdot b)^\omega) &= \frac{1}{2} \end{aligned}$$

La figure 6.1 présente ces résultats de manière graphique. □

Il est facile de vérifier que d est bien une métrique[2]. Le théorème suivant va nous permettre de caractériser plus facilement un ensemble ouvert de la topologie naturelle de Σ^ω.

Théorème 6.1. Les ouverts de (Σ^ω,d) sont exactement les ensembles de la forme $X \cdot \Sigma^\omega$ avec $X \subseteq \Sigma^+$ (i.e. tels que X un langage de mots finis).

Preuve. Se reporter à l'annexe A, section A.2.5. □

Un élément d'un ouvert est représenté schématiquement à la figure 6.2. Il possède un préfixe qui appartient au langage de mots finis X. Le préfixe est ensuite suffixé d'un mot infini pouvant être quelconque.

1. Nous avons défini la distance *exponentielle*, qui est la plus fréquemment utilisée. On utilise parfois aussi la distance *linéaire* pour laquelle $d(u,v) = 1/(r+1)$. Ces distances induisent la même topologie, mais se distinguent par certaines propriétés concernant les fonctions continues [Pri00].

2. En fait, d est même une *ultramétrique*.

6.1.2 Ensembles de base

À partir de maintenant, nous allons nous focaliser sur l'alphabet Σ introduit au chapitre 2 dans le but d'encoder des vecteurs de réels. Nous avons vu à la section 2.7 que l'on pouvait scinder le langage Σ^ω en quatre parties : V, NV_s, NV_0 et NV_+. Le but de cette section est de donner plusieurs lemmes qui caractérisent la topologie de chacun de ces ensembles.

Lemme 6.2. Il est possible d'établir le tableau suivant :

Ensemble	Nature
NV_s	Ouvert
NV_+	Ouvert
$NV_+ \cup V$	Ouvert
$NV_0 \cup V$	Fermé

Preuve. Pour NV_s et NV_+, il suffit de se reporter à la section 2.7, de constater que leurs définitions se terminent toutes deux par Σ^ω et d'appliquer le théorème 6.1. D'autre part, on a la relation : $NV_+ \cup V = \{0,r-1\}^n \cdot (\Lambda^n)^* \cdot \star \cdot \Sigma^\omega$. Par conséquent, le théorème 6.1 affirme directement que $NV_+ \cup V$ est un ouvert. Finalement, $NV_0 \cup V = (NV_+ \cup NV_s)^c = NV_+^c \cap NV_s^c$. Nous venons de voir que NV_+ et NV_s sont des ouverts. Ainsi, $NV_0 \cup V$ est l'intersection de deux fermés et est donc lui-même fermé. □

Définition 6.1. Étant donné $S \subseteq \mathbb{R}^n$, on appelle $L_r(S) \subseteq V$ le $\omega-$langage que doit accepter tout RVA qui représente l'ensemble S en base r (cf. la définition 4.1).

Lemme 6.3. Pour tout ensemble ouvert $O \subseteq \mathbb{R}^n$, $L' = L_r(O) \cup NV_+$ est un ouvert.

Preuve. Considérons un vecteur $\vec{x} \in O$. Puisque O est un ouvert, on peut appliquer le théorème 5.2 et $\vec{x}$ admet une boule ouverte $B(\vec{x},\varepsilon')$ entièrement incluse dans O. Cette boule n'est pas vide et admet donc en son sein un cube à n dimensions « sans pelure » centré sur x dont les côtés sont de longueur $2 \cdot \varepsilon$, où $\varepsilon = \varepsilon'/\sqrt{n}$. Choisissons maintenant $k \in \mathbb{N}$ d'une manière telle que $r^{-k+1} \leq \varepsilon$ où r est la base de numération utilisée (on pose par exemple $k = \lceil -\log_r \varepsilon \rceil + 1$).

Soit $w \in L_r(\{\vec{x}\})$ un des encodages valides de $\vec{x}$ en base r. On définit alors :

$$U = \{u \mid (\exists z \in \Sigma^\omega)(u = \underbrace{w_0 \dots w_{q-1} \star w_{q+1} \dots w_{q+k}}_{v} \cdot z), \text{ si } q = p(w,\star)\}$$

U est un ouvert de Σ^ω, car il est de la forme $\{v\} \cdot \Sigma^\omega$. Il contient de plus w : il suffit de prendre $z = w_{q+k+1} w_{q+k+2} \dots$. On en déduit que U est un voisinage de w. Il faut remarquer que le suffixe z peut très bien contenir des séparateurs décimaux excédentaires et donc que $U \cap NV_+ \neq \phi$.

L'ensemble $A = U \setminus NV_+$ ne contient que des encodages valides. On peut donc lui appliquer la fonction de décodage d du chapitre 2. Si nous arrivons à prouver

que $d(w)$ est bien inclus dans le cube ouvert de côté $2 \cdot \varepsilon$ centré sur x pour tout mot $w \in A$, ceci impliquera que $d(A) \subseteq O$, donc que $A \subseteq L_r(O)$ et que finalement $U = A \cup NV_+ \subseteq L_r(O) \cup NV_+$. Ainsi, U sera un voisinage de w entièrement inclus dans $L_r(O) \cup NV_+$. Puisque cela reste valable pour tout $\vec{x} \in O$ et pour tout encodage w de $\vec{x}$, le théorème 5.2 permettra de conclure directement.

À cet effet, nous allons procéder composante par composante. Plus précisément, afin que A appartienne au cube ouvert, chaque composante i de $d(A)$ doit appartenir à l'intervalle $]\vec{x}_i - \varepsilon\ ; \vec{x}_i + \varepsilon[$. Le plus petit réel que peut engendrer la composante i de $d(A)$ est $d_{\mathbb{R}}(\pi(v,i) \cdot 0^\omega)$, qu'il faut prouver être strictement supérieure à $\vec{x}_i - \varepsilon = d_{\mathbb{R}}(\pi(w,i)) - \varepsilon$. Les parties entières de ces deux mots sont par construction égales et s'annihilent : on peut donc se concentrer sur les parties fractionnaires. On a successivement :

$$\begin{aligned} d_{\mathbb{R}}(\pi(w,i)) - d_{\mathbb{R}}(\pi(v,i) \cdot 0^\omega) &= \sum_{j=0}^{\infty} \frac{\pi(w,i)_{q+j+1}}{r^{j+1}} - \sum_{j=0}^{k-1} \frac{\pi(v,i)_{q+j+1}}{r^{j+1}} \\ &= \sum_{j=k}^{\infty} \frac{\pi(w,i)_{q+j+1}}{r^{j+1}} \leq (r-1) \cdot \sum_{j=k}^{\infty} \frac{1}{r^{j+1}} \\ &= \frac{r-1}{r^k \cdot (r-1)} = \frac{r^{-k+1}}{r} \leq \frac{\varepsilon}{r} < \varepsilon \end{aligned}$$

La dernière inégalité démontre ce que nous voulions. Il reste à faire une preuve similaire pour la borne supérieure sur la $i-$ème composante de $d(A)$, qui correspond cette fois à $d_{\mathbb{R}}(\pi(v,i) \cdot (r-1)^\omega)$ et qui doit être inférieure à $d_{\mathbb{R}}(\pi(w,i)) + \varepsilon$. On raisonne de manière similaire :

$$\begin{aligned} & d_{\mathbb{R}}(\pi(v,i) \cdot (r-1)^\omega) - d_{\mathbb{R}}(\pi(w,i)) \\ = & \left(\sum_{j=0}^{k-1} \frac{\pi(v,i)_{q+j+1}}{r^{j+1}} + \sum_{j=k}^{\infty} \frac{r-1}{r^{j+1}} \right) - \sum_{j=0}^{\infty} \frac{\pi(w,i)_{q+j+1}}{r^{j+1}} \\ = & \sum_{j=k}^{\infty} \frac{(r-1) - \pi(w,i)_{q+j+1}}{r^{j+1}} \leq (r-1) \cdot \sum_{j=k}^{\infty} \frac{1}{r^{j+1}} \end{aligned}$$

On constate que l'on retombe sur une expression déjà obtenue plus haut et que l'on a montré être strictement inférieure à ε, ce que nous voulions obtenir. Ces inégalités restant valables quelle que soit la composante $i = 1, \ldots, n$, on a que $d(A)$ est inclus dans le cube ouvert, ce que nous voulions démontrer. □

6.1.3 Ensembles $G_\delta \cap F_\sigma$

Lemme 6.4. Soit O un ouvert de $\mathbb{R}^n$. $L_r(O)$ est alors un ensemble G_δ de Σ^ω.

Preuve. Posons $L' = L_r(O) \cup NV_+$. Cette définition implique que :

$$L_r(O) = L' \setminus NV_+ = L' \cap NV_+^c = L' \cap (NV_s \cup (NV_0 \cup V))$$

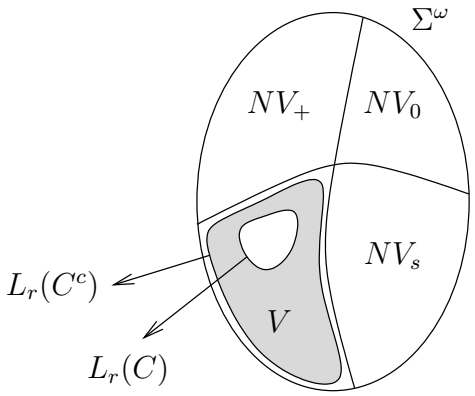


Fig. 6.3 – *Agencement des éléments du lemme 6.5.*

Par le lemme 6.3, L' est un ouvert car O est lui-même ouvert. D'autre part, le lemme 6.2 affirme que NV_s est un ouvert et que $NV_0 \cup V$ est un fermé. Finalement, $L_r(O)$ est une combinaison booléenne de trois ouverts et d'un fermé (à savoir $NV_0 \cup V$). Le théorème 5.4 permet de conclure. □

Lemme 6.5. Soit C un fermé de $\mathbb{R}^n$. $L_r(C)$ est alors un ensemble F_σ de Σ^ω.

Preuve. Pour cette démonstration, le lecteur pourra s'aider de la figure 6.3. Posons $L' = L_r(C) \cup NV_0$. La figure montre immédiatement que $L'^c = L_r(C^c) \cup NV_+ \cup NV_s$. Puisque C est un fermé, on sait par le lemme 6.3 que $L_r(C^c) \cup NV_+$ est ouvert. De plus, NV_s est un ouvert (lemme 6.2). L'^c est donc une union d'ouverts, ce qui implique que L' est un fermé. Maintenant, la définition de L' fait que l'on a :

$$L_r(C) = L' \setminus NV_0 = L' \cap (NV_s \cup NV_+ \cup V)$$

On sait en outre que NV_s et $NV_+ \cup V$ sont des ouverts. Ainsi, on voit que $L_r(C)$ est une combinaison booléenne de deux ouverts et d'un fermé (à savoir L'). On conclut à nouveau en utilisant le théorème 5.4. □

Théorème 6.6. (Correspondance entre $\mathbb{R}^n$ et Σ^ω) Pour tout ensemble $S \subseteq \mathbb{R}^n$ appartenant à la classe $G_\delta \cap F_\sigma$, $L_r(S)$ appartient aussi à $G_\delta \cap F_\sigma$ de Σ^ω.[3]

Preuve. D'un côté, puisque $S \in F_\sigma$, il existe des C_i fermés tels que $S = \bigcup_{i=1}^{\infty} C_i$. Le langage représentant C_i étant par définition $L_r(C_i)$, on a : $L_r(S) = \bigcup_{i=1}^{\infty} L_r(C_i)$. Par le lemme 6.5, chaque $L_r(C_i)$ est une union dénombrable de fermés, d'où $L_r(S)$ est une union dénombrable d'unions dénombrables de fermés, c'est-à-dire une union dénombrable de fermés, ce qui prouve la première partie de l'énoncé : $L_r(S) \in F_\sigma$.

D'un autre côté, puisque $S \in G_\delta$, on a un ensemble d'ouverts O_i tels que $S = \bigcap_{i=1}^{\infty} O_i$. Ainsi, $L_r(S) = \bigcap_{i=1}^{\infty} L_r(O_i)$. Par le lemme 6.4, chaque $L_r(O_i)$ est un ensemble G_δ. De ce fait, $L_r(S) \in G_{\delta\delta}$, donc $L_r(S) \in G_\delta$, ce qui achève la démonstration. □

3. Il est possible de démontrer le même théorème en utilisant la notion topologique d'*application continue* [Jod01].

6.2 Topologie et $\omega-$automates déterministes

Nous venons de prouver que tout ensemble $G_\delta \cap F_\sigma$ de $\mathbb{R}^n$ se transpose par le biais de la relation d'encodage en un ensemble $G_\delta \cap F_\sigma$ de Σ^ω. En rapprochant ce résultat avec le théorème 5.6, on obtient directement pour conséquence :

Corollaire 6.7. Soit un ensemble $S \subseteq \mathbb{R}^n$ avec $n > 0$ défini par une formule de la structure $\langle \mathbb{R}, Z, +, \leq \rangle$. Alors $L_r(S)$ appartient à la classe $G_\delta \cap F_\sigma$.

Il est donc souhaitable de disposer d'une classe d'$\omega-$automates qui accepteraient les $\omega-$langages de la classe de Borel $G_\delta \cap F_\sigma$. En première approximation, nous allons caractériser la topologie des automates de Büchi déterministes. Il est possible de démontrer (voir annexe A, section A.2.7) le théorème suivant :

Théorème 6.8. [Lan69] Les automates de Büchi déterministes épousent exactement la classe des $\omega-$langages G_δ qui peuvent être acceptés par un automate.

Une conséquence de ce théorème est que l'on peut représenter toute formule de la structure $\langle \mathbb{R}, Z, +, \leq \rangle$ sous la forme d'un automate de Büchi déterministe puisque trivialement la classe $G_\delta \cap F_\sigma$ est incluse dans la classe G_δ. En revanche, on n'a aucune garantie que la manipulation *a posteriori* d'un tel automate resterait déterministe : la projection ou la complémentation pourrait faire que l'on quitte cette forme particulière (cf. section 3.4). Et bien que l'on sache que l'automate auquel on arrive pourrait lui aussi être exprimé sous la forme d'un automate de Büchi déterministe (après tout, si φ est une formule de notre structure, $(\exists x)\varphi$ et $\neg\varphi$ restent des formules de la même structure), on ne saurait pas nécessairement comment s'y ramener.

Nous allons donc tenter de mieux cerner la classe $G_\delta \cap F_\sigma$. Pour ce faire, nous allons rechercher une classe d'automates qui acceptent exactement les ensembles F_σ. Le principe de dualité affirme que le complément de tout langage de F_σ est un langage G_δ. Il semblerait donc qu'il nous faut considérer une famille d'$\omega-$automates qui accepteraient exactement les langages compléments de ceux acceptés par les automates de Büchi déterministes : cette classe s'appelle tout logiquement les *automates co-Büchi déterministes*.

6.3 Automates co-Büchi

6.3.1 Définition

Les automates co-Büchi ressemblent en tous points aux automates de Büchi introduits au chapitre 3, si ce n'est qu'ils possèdent une condition d'acceptation duale :

Définition 6.2. Une course ρ sur w d'un automate co-Büchi A' est dite *acceptrice* si $\mathrm{Inf}(\rho) \cap F = \phi$.

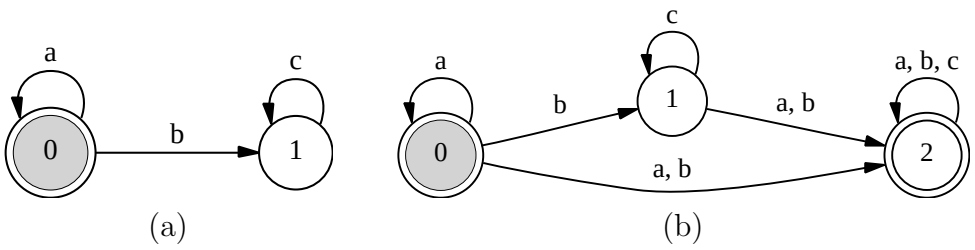


FIG. 6.4 – *(a) Un automate co-Büchi, (b) le même complété.*

Toutes les autres définitions restent identiques. Placé dans un contexte de non-déterminisme, le mot w sera accepté par A' (ce que nous écrirons à nouveau[4] $w \in L_\omega(A')$) si et seulement si il existe une course de A' sur w qui passe seulement *un nombre fini de fois* par un état accepteur.

Exemple. Les automates de la figure 6.4 acceptent exactement le $\omega-$langage $a^* \cdot b \cdot c^\omega$. Le mot a^ω n'est pas accepté car la course correspondante boucle dans l'état 0 qui est accepteur. De même, le mot c^ω n'est pas accepté car il n'existe aucune course sur ce mot et donc *a fortiori*, de course acceptrice. □

6.3.2 Topologie

Les automates co-Büchi ont la propriété attendue suivante :

Proposition 6.9. Soit un automate co-Büchi déterministe $A = (Q, \Sigma, \delta, \{s_0\}, F)$. Il est possible de le convertir en un automate de Büchi A'' tel que $L_\omega(A'') = (L_\omega(A))^c$.

Preuve. Soit un mot w. On complète l'automate A pour obtenir l'automate co-Büchi A'. Pour ce faire, on crée comme à la figure 6.4 un état puits qui est, à l'inverse des automates de Büchi, accepteur. Ensuite, on regarde A' comme un automate de Büchi A''. On a alors successivement :

$$\begin{aligned}
w \in L_\omega(A) &\Leftrightarrow w \in L_\omega(A') \\
&\Leftrightarrow (\exists \rho(A',w))(\mathrm{Inf}(\rho) \cap F = \phi) \\
&\Leftrightarrow (\exists \rho(A',w))\neg(\mathrm{Inf}(\rho) \cap F \neq \phi) \\
&\Leftrightarrow \neg(\forall \rho(A',w))(\mathrm{Inf}(\rho) \cap F \neq \phi) \\
&\Leftrightarrow \neg(\exists \rho(A',w))(\mathrm{Inf}(\rho) \cap F \neq \phi) \qquad (*) \\
&\Leftrightarrow w \notin L_\omega(A'')
\end{aligned}$$

Où l'équivalence $(*)$ est valide car A' est supposé déterministe et complet : dans ce cas, il y a une et une seule course pour tout mot w et la quantification universelle se ramène à une quantification existentielle. Remarquons que si A' n'était pas supposé complet, cette équivalence serait erronée car elle se réduirait à $\neg true \Leftrightarrow \neg false$

4. Il faudra donc prendre garde au contexte.

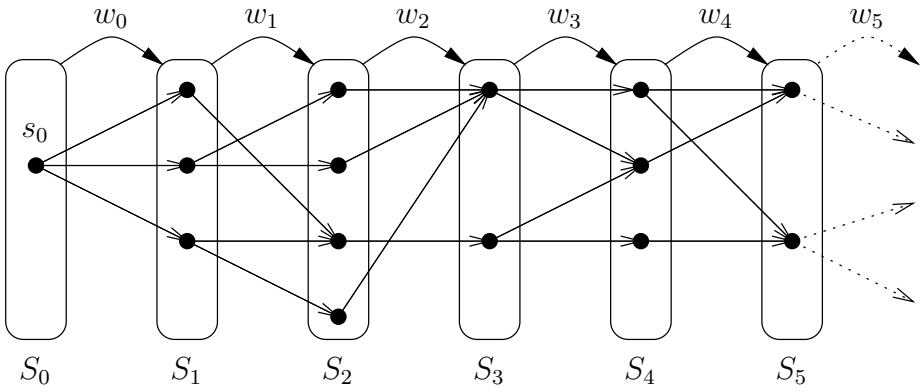


Fig. 6.5 – *Courses d'un automate non déterministe A sur un mot w.*

pour un mot sur lequel il n'existe aucune course de A (une quantification universelle appliquée à un ensemble vide est toujours valide, à l'inverse d'une existentielle). On en déduit que l'on peut convertir tout automate co-Büchi déterministe A en un automate de Büchi déterministe A'' qui accepte le complément de son langage. □

Cette proposition admet un important corollaire :

Corollaire 6.10. Les automates co-Büchi déterministes acceptent exactement la classe des langages F_σ qui peuvent être acceptés par un $\omega-$automate.

Preuve. Soit un automate co-Büchi A. On applique la proposition précédente pour construire un automate de Büchi A'' qui accepte le complément de son langage. Or, $L_\omega(A'') \in G_\delta$ (théorème 6.8). Par dualité, $L_\omega(A) = (L_\omega(A''))^c \in F_\sigma$. □

Ce corollaire affirme que les automates co-Büchi sont eux aussi à même de pouvoir représenter toute formule de la structure $\langle \mathbb{R}, Z, +, \leq \rangle$. La section suivante prouve même que les automates co-Büchi non déterministes ne peuvent pas exprimer plus de choses que leurs contreparties déterministes, à l'inverse des automates de Büchi (voir section 3.4).

6.3.3 Déterminisation

La déterminisation d'$\omega-$automates devient possible dans le cadre des automates co-Büchi. La construction correspondante est dite à « points de rupture » (*breakpoint construction*), d'abord suggérée par Miyano et Hayashi dans [MH84], puis simplifiée dans [Var96, KV97].[5] Fondamentalement, il s'agit une variante de la méthode des sous-ensembles utilisée pour déterminiser un automate sur mots finis [HU79, Wol91] mais elle s'en distingue par un marquage supplémentaire.

Imaginons que nous voulions simuler le comportement d'un automate co-Büchi non déterministe A sur un mot w. Puisque A n'est pas déterministe, il peut exister

5. Cette technique se place dans un contexte plus général, celui des automates de Büchi *alternants*. Dans ce document, nous l'appliquons directement aux automates co-Büchi.

plusieurs courses de A sur w qui peuvent se rejoindre ou se séparer au cours de la lecture de w, comme schématisé à la figure 6.5. Appelons S_n l'ensemble des états dans lesquels pourrait se trouver A après avoir lu les n premiers caractères de w. Si l'on considère ces ensembles S_n comme des états, la suite des (S_n) décrit une course agglomérée ρ qui devient déterministe : étant donnés l'ensemble S_k et le symbole de l'alphabet w_k, il ne peut y avoir qu'un seul successeur de S_k par w_k.

Une telle course agglomérée ρ est acceptrice si et seulement si elle contient une sous-course qui ne rencontre qu'un nombre fini de fois les états de F. Il nous faut donc mémoriser une information supplémentaire concernant les occurrences des états accepteurs dans ρ. À cette fin, en plus de la suite (S_n), nous allons retenir une suite d'ensembles (R_n) qui contiennent l'état courant de chacune des courses n'ayant pas encore visité un état de F. Si jamais $R_k = \phi$, nous disons que nous avons atteint un point de rupture : cela signifie qu'aucune sous-course n'a réussi à louvoyer entre les états accepteurs. Dans ce cas, on réinitialise R_{k+1} à S_{k+1} duquel on a retiré les états accepteurs et on recommence la recherche d'une sous-course qui ne passe pas par un état accepteur.

Il y a alors deux cas dans lesquels la suite des (S_n,R_n) décrit une course agglomérée ρ qui est acceptrice :

- Si ρ ne rencontre aucun point de rupture, cela signifie qu'il existe toujours une sous-course de ρ qui évite les états de F. Cela implique aussi qu'il existe une course acceptrice de A sur w et donc que $w \in L_\omega(A)$.
- Si ρ rencontre seulement un nombre fini de fois un point de rupture, cela signifie qu'à partir d'un certain n', il existe une sous-course qui parvient à échapper aux états de F. On se ramène alors au premier cas.

Ainsi, de deux choses l'une : la course ρ n'est pas acceptrice si et seulement si elle passe un nombre fini de fois par un point de rupture. Or, ceci correspond parfaitement à la condition d'acceptation d'un automate co-Büchi : il suffit de marquer comme accepteurs les points de rupture.

Remarque 6.3. Intuitivement, cette technique ne fonctionne pas pour les automates de Büchi car la section 3.4 nous avait appris qu'il n'est pas possible d'exprimer sous la forme d'un automate de Büchi déterministe une condition du type « une course doit passer un nombre fini de fois par un certain état ». □

Plus formellement, la construction qui crée un automate co-Büchi déterministe $A' = (Q', \Sigma, \delta', S'_0, F')$ équivalent au départ d'un automate co-Büchi non déterministe $A = (Q, \Sigma, \delta, S_0, F)$ est donnée à la figure 6.6 de la page 69.

Il est clair que A' est déterministe : la définition de δ' fait qu'il n'y a qu'un seul successeur pour chaque état et chaque lettre. Le nombre d'états de A' est $\mathcal{O}(2^{2n}) = 2^{\mathcal{O}(n)}$ si A possède n états. Une preuve formelle de l'adéquation de cette construction peut être trouvée à la section A.3 de l'annexe A.

Remarque 6.4. Outre un gain de *simplicité* par rapport à la déterminisation des automates de Büchi (cf. section 3.3.5), on observe un gain dans la *complexité*

- $Q' = 2^Q \times 2^Q$, où la première composante représente notre ensemble S_n et la seconde, notre ensemble R_n,
- Étant donnés un état $(S,R) \in Q'$ et une lettre $\sigma \in \Sigma$, la fonction de transition est définie comme suit :
  - Si $R \neq \phi$, alors $\delta'((S,R), \sigma) = \{(T(S,\sigma), T(R,\sigma) \setminus F)\}$ où $T : 2^Q \times \Sigma \mapsto 2^Q : (E,\sigma) \mapsto \{q \mid (\exists p \in E)(q \in \delta(p,\sigma))\}$ est une fonction qui associe à un ensemble d'états E l'ensemble des états qui sont reliés à un état de E par une transition étiquetée par σ.
  - Si $R = \phi$, un point de rupture est atteint : $\delta'((S,\phi), \sigma) = \{(T(S), T(S) \setminus F)\}$.

  Dans les deux cas, S' est obtenu depuis S de la même façon que pour les automates sur mots finis. Si $R = \phi$, on retire de S' les états accepteurs pour obtenir R'. Si $R \neq \phi$, on applique une seconde fois la construction classique au départ de R pour obtenir R'.
- $S'_0 = (S_0, \phi)$ (on commence toujours par un point de rupture),
- $F' = 2^Q \times \{\phi\}$ (ce qui correspond à l'ensemble des points de rupture).

Fig. 6.6 – *Construction de déterminisation d'un automate co-Büchi.*

de l'opération. En effet, déterminiser un automate de Büchi générait un automate possédant $2^{\mathcal{O}(n \log n)}$ états, là où la déterminisation d'un automate co-Büchi fournit $2^{\mathcal{O}(n)}$ états, ce qui est du même ordre de complexité que la déterminisation d'un automate sur mots finis. □

Remarque 6.5. Par définition de la fonction de transition, il est facile de constater que l'on a toujours $R \subseteq S$. Dans l'implémentation qui a été réalisée dans le cadre de ce travail, cela nous a permis d'utiliser un simple marquage des états de S pour représenter les états de R : on utilise un bit supplémentaire pour marquer les états de S qui appartiennent à R. □

6.3.4 Conclusion

Si on prend pour convention de représenter une formule de $\langle \mathbb{R}, Z, +, \leq \rangle$ par des automates co-Büchi, la déterminisation devient possible. On peut dès lors compléménter un automate co-Büchi : il suffit de considérer l'automate déterminisé et complété comme un automate de Büchi. Malheureusement, après complémentation, il devient impossible de combiner l'automate obtenu avec d'autres automates co-Büchi puisqu'il s'agit alors d'un automate de Büchi.

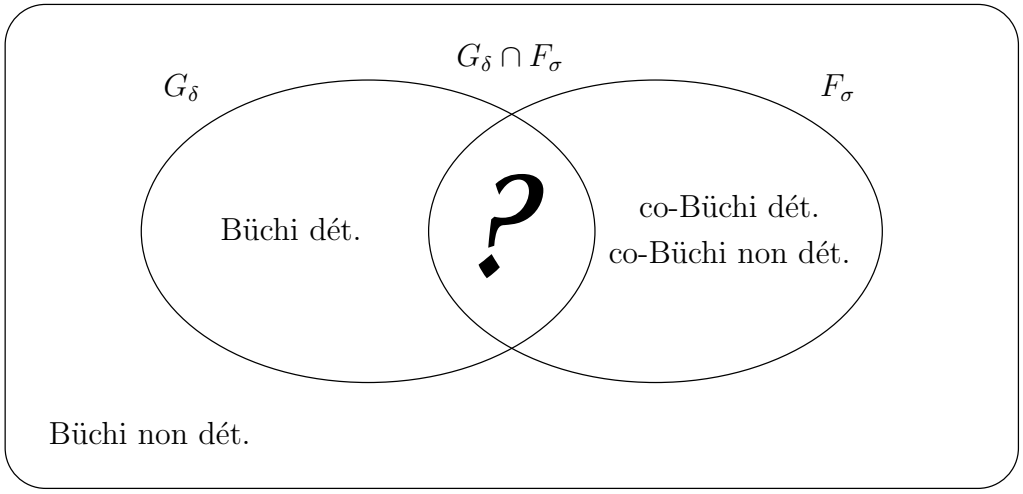


Fig. 6.7 – *Hiérarchie dans l'expressivité des différentes conditions d'acceptation.*

6.4 Synthèse

La synthèse de ce que nous avons appris jusqu'à présent est reprise à la figure 6.7. Rappelons que nous cherchons une catégorie d'$\omega-$automates qui acceptent exactement les langages $G_\delta \cap F_\sigma$. Tant qu'à présent, nous avons deux classes en lice, chacune possédant ses avantages et ses défauts :

- les automates de Büchi déterministes, qui sont relativement simples à manipuler mais sur lesquels on ne peut pas toujours mener à bien les opérations de complémentation et de projection (cf. la section 3.4), nécessaires pour les tests ou pour les quantifications ;
- les automates co-Büchi, qui permettent la déterminisation et donc la complémentation, mais pour lesquels il est impossible d'exploiter le résultat de cette dernière opération.

Nous allons tenter de nous restreindre à l'intersection de ces deux classes afin de bénéficier simultanément de leurs avantages, tout en acceptant exactement les langages $G_\delta \cap F_\sigma$. Le sujet des sections suivantes est l'étude d'une telle classe très particulière d'$\omega-$automates, les « *automates faibles* », qui peuvent être vus comme des automates qui sont à la fois de Büchi et co-Büchi.

6.5 Automates faibles

6.5.1 Définition

Définition 6.6. [**MSS86**] Soit un automate de Büchi A. A est dit *faible* si dans chacune de ses composantes fortement connexes, soit *tous* les états sont accepteurs, soit *aucun* n'est accepteur. Nous parlerons dans la suite de composante fortement

connexe « acceptrice » ou « non acceptrice »[6].

Exemple. Les automates des figures 3.1 (page 16) et 3.10 (page 27) sont faibles, ainsi que tous les exemples du chapitre 4. Par contre, les automates des figures 3.3, 3.4 et 3.11 (respectivement aux pages 20, 21 et 28) ne sont pas faibles. □

Remarque 6.7. D'un point de vue historique, nous nous sommes intéressés aux automates faibles car tous les RVA de base possèdent cette forme particulière, comme on peut le constater en parcourant le chapitre 4. □

6.5.2 Caractérisation topologique

Par définition, un automate faible est un automate de Büchi particulier. On peut également démontrer qu'il peut se voir comme un automate co-Büchi :

Proposition 6.11. Soit l'automate faible (éventuellement non déterministe) $A = (Q, \Sigma, \delta, S_0, F)$. L'automate co-Büchi $A' = (Q, \Sigma, \delta, S_0, Q \setminus F)$ accepte le même langage que A.

Preuve. Soient un mot $w \in L_\omega(A)$ et une course acceptrice ρ de A sur w. Par un raisonnement similaire à celui développé dans la preuve de la proposition 3.3 (à la page 26), on déduit que ρ doit finir par boucler dans un cycle qui contient un état accepteur, auquel cas on a bien $\mathrm{Inf}(\rho) \cap F \neq \phi$. Puisque l'automate est faible, ce cycle est inclus dans une composante fortement connexe Q_i acceptrice et donc, *tous* les états du cycle sont accepteurs. Par conséquent, on a même $\mathrm{Inf}(\rho) \subseteq F \Leftrightarrow \mathrm{Inf}(\rho) \cap (Q \setminus F) = \phi$. Cette dernière condition correspond à la définition d'une course acceptrice de l'automate co-Büchi A'. De ce fait, $w \in L_\omega(A')$.

Réciproquement, soient un mot $w \in L_\omega(A')$ et une course acceptrice ρ' de A' sur w. ρ' doit finir par boucler dans un cycle qui ne contient aucun état accepteur de A' : $\mathrm{Inf}(\rho') \cap (Q \setminus F) = \phi \Leftrightarrow \mathrm{Inf}(\rho') \subseteq F$. La course ρ' est donc acceptrice pour l'automate faible A et ainsi $w \in L_\omega(A)$, ce qui achève la démonstration. □

Corollaire 6.12. Tout automate faible peut se voir soit comme un automate de Büchi, soit comme un automate co-Büchi.

La figure 6.7 a maintenant pour conséquence immédiate :

Corollaire 6.13. Pour tout automate faible déterministe A, $L_\omega(A) \in G_\delta \cap F_\sigma$. Pour tout automate faible non déterministe A, $L_\omega(A) \in F_\sigma$.

6. Une autre définition équivalente est parfois rencontrée dans la littérature. On dit que A est faible si l'on peut partitionner l'ensemble Q de ses états en sous-ensembles disjoints $Q_1, \ldots, Q_m$ tels que : (i) pour tout Q_i, soit $Q_i \subseteq F$, soit $Q_i \cap F = \phi$, et (ii) il existe un ordre partiel $\preceq$ entre les partitions tel que pour tout $q \in Q_i$, $q' \in Q_j$ et $\sigma \in \Sigma$, on a : $q' \in \delta(q,\sigma) \Rightarrow Q_j \preceq Q_i$.

Opération	Algorithme à utiliser	Résultat
Union [non dét.]	Büchi	Faible [non dét.]
Intersection [non dét.]	Büchi	Faible [non dét.]
Produit cartésien [non dét.]	Büchi	Faible [non dét.]
Projection	Büchi	Faible non dét.
Test langage vide	Büchi	/
Complémentation dét.	Faible	Faible dét.
Complémentation non dét.	co-Büchi	Büchi dét.
Déterminisation	co-Büchi	co-Büchi dét.

Fig. 6.8 – *Opérations réalisables sur les automates faibles.*

6.5.3 Manipulation d'automates faibles

Le corollaire 6.12 a d'autres implications. Il permet en effet de choisir le type d'automate qui convient le mieux pour mener à bien une opération sur un ou plusieurs automate(s) faible(s), éventuellement non déterministe(s). Le tableau 6.8 reprend les différentes opérations et donne pour chacune d'elles, l'algorithme à utiliser ainsi que le type d'automate qui en résulte.

Exemple. Si l'on veut calculer le langage intersection des automates faibles A_1 et A_2, le tableau nous dit de considérer A_1 et A_2 comme des automates de Büchi et d'appliquer la construction d'intersection pour les automates de Büchi. L'automate résultant possède toujours la forme faible. Il peut être non déterministe si l'un des deux automates de départ (voire les deux) était non déterministe. □

Les trois dernières lignes sont essentielles. On y voit que complémenter un automate faible déterministe donne toujours un automate *faible déterministe* si l'on s'y prend bien. Pour ce faire, on utilise la proposition :

Proposition 6.14. Le langage complémenté d'un automate faible déterministe complété $A = (Q, \Sigma, \delta, \{s_0\}, F)$ est accepté par l'automate faible déterministe $A' = (Q, \Sigma, \delta, \{s_0\}, Q \setminus F)$.

Preuve. A' vu comme un automate co-Büchi admet le même langage que A (proposition 6.11). A' est complet : le raisonnement de la proposition 6.9 nous affirme alors directement que A' vu comme un automate de Büchi admet le langage complément de A. De plus, puisqu'on n'a fait qu'inverser le statut accepteur et non accepteur des états, A' possède toujours la forme faible. □

Pour complémenter un automate faible déterministe, il suffit donc de le compléter et d'inverser ses états accepteurs et non accepteurs. Ce procédé ne fonctionnait pas pour les automates de Büchi, même déterministes (cf. section 3.3.5).

Exemple. La figure 6.9 reprend un exemple d'application de cet algorithme. □

Par contre, pour complémenter un automate faible non déterministe A, on n'a pas d'autre choix que de le déterminiser en le considérant comme un automate co-Büchi par la proposition 6.11 (ce faisant, on obtient un automate co-Büchi $\mathcal{C}$), puis de

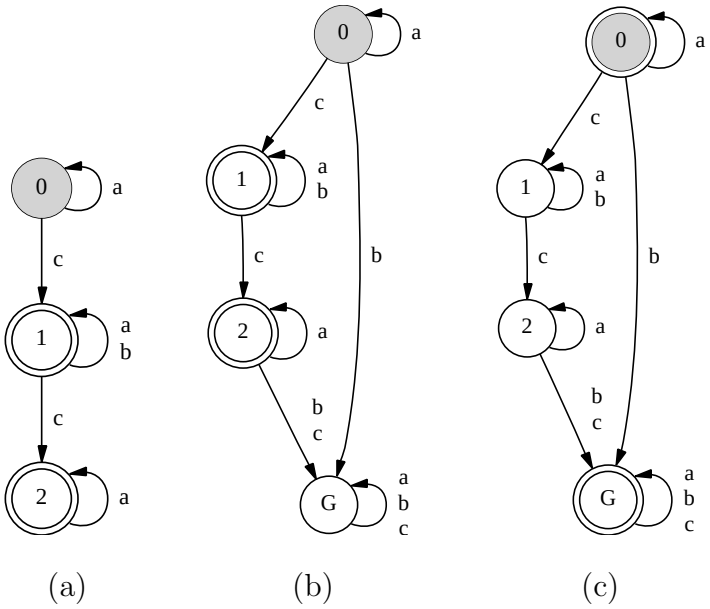


Fig. 6.9 – *Complémentation d'un automate faible déterministe : (i) A, (ii) A complété et (iii) A complémenté.*

regarder $\mathcal{C}$ comme un automate de Büchi $\mathcal{B}$. La proposition 6.9 prouve l'adéquation de cette construction. Si on voulait simplement le déterminiser, il suffit de s'arrêter à l'étape de déterminisation : on obtient alors bien un automate co-Büchi $\mathcal{C}$ équivalent à l'automate faible.

6.5.4 Commentaires

La section précédente nous montre qu'il est possible de mener à bien toutes les opérations de base sur les automates faibles. Il est intéressant de constater que la classe des automates faibles déterministes est close par complémentation, à l'inverse de celle des automates de Büchi déterministes. Malheureusement, dès que du non-déterminisme est introduit et que l'on cherche à réaliser une complémentation ou une déterminisation, on quitte cette forme particulière. Remarquons que seule l'opération de projection (qui correspond à une quantification logique) peut introduire du non-déterminisme initialement absent : cela confirme l'intuition selon laquelle les quantifications sont souvent plus difficiles à gérer que les opérations booléennes.

Le chapitre suivant va nous apprendre que l'on peut se limiter exclusivement aux automates faibles pour matérialiser les RVA représentant des formules de la théorie additive des entiers et des réels. Ceci nous permettra d'utiliser des algorithmes considérablement plus simples que ceux nécessaires à la manipulation des RVA généraux.

Chapitre 7

Manipulation pratique de RVA

Nous avons proposé dans le chapitre précédent une classe très particulière de $\omega-$automates, les *automates faibles*, qui acceptent des langages de la classe $G_\delta \cap F_\sigma$ dans l'espace topologique des mots infinis. Le but de ce chapitre est de montrer que tout RVA représentant une formule de la théorie additive des entiers et des réels peut toujours être converti sous la forme d'un automate faible déterministe. De ce résultat, nous pourrons conclure que l'on peut se limiter exclusivement aux automates faibles pour traiter les RVA restreints.

La manipulation des automates faibles est nettement plus aisée que celle des autres classes d'$\omega-$automates. Ils peuvent même être minimisés. Les résultats des deux chapitres précédents ne sont donc pas seulement d'intérêt théorique : la légère et naturelle restriction d'expressivité que nous avons imposée sur les RVA autorise leur manipulation en pratique. Les algorithmes à mettre en œuvre sont en effet de complexité très comparable à celle des algorithmes applicables aux automates sur mots finis, ce qui rend envisageable une implémentation informatique.

7.1 Automates faibles et RVA

Dans le chapitre 6, nous nous sommes attachés à trouver une famille d'automates qui acceptent les $\omega-$langages appartenant à la classe de Borel $G_\delta \cap F_\sigma$. Nous venons de voir que cette famille correspond exactement aux automates faibles déterministes[1] : le corollaire 6.7 nous affirme donc que tout RVA représentant une formule de la structure $\langle \mathbb{R}, Z, +, \leq \rangle$ peut s'exprimer sous la forme d'un automate faible déterministe. Il est tentant d'en conclure que toute manipulation de RVA donnera systématiquement un automate faible déterministe.

Malheureusement, ce n'est pas le cas. Effectivement, le tableau de la figure 6.8 nous a également appris que l'on quitte cette forme faible déterministe dès que l'on cherche à réaliser une projection. Or, la projection est nécessaire pour réaliser les quantifications. Pire, une complémentation ou une déterminisation consécutive

1. À dire vrai, nous n'avons pas prouvé que tout langage $G_\delta \cap F_\sigma$ peut être accepté par un automate faible déterministe. Cela est néanmoins le cas, comme le montre la section A.4 de l'annexe A.

à une introduction de non-déterminisme ne donne même plus un automate faible non déterministe : par exemple, dès que l'on cherche à réaliser une quantification universelle, on se retrouve avec un automate de Büchi déterministe. Et même si nous savons qu'il existe quelque part un automate faible déterministe équivalent, rien ne nous dit comment le retrouver : il pourrait très bien être à mille lieues du résultat de l'opération. Nous avons déjà fait un constat similaire à la section 6.2.

Heureusement, nous allons prouver que tous les automates issus d'une manipulation de RVA possèdent une structure très particulière. Cette structure fait qu'il devient aisé de retrouver l'automate faible déterministe équivalent.

7.2 Non-déterminisme et RVA

Afin de mieux cerner le problème, commençons par nous demander d'où peut venir le non-déterminisme consécutif à la manipulation de RVA restreints, celui-ci étant la source de nos problèmes. La projection d'un RVA peut induire du non-déterminisme, mais ce n'est pas la seule cause.

Il est vrai que tous les RVA de base que nous avons étudiés au chapitre 4 (section 4.2) sont des automates faibles déterministes : il suffit de regarder leur représentation sagittale. Ils ne posent donc pas problème.

De même, les RVA qui résultent de la génération d'une équation par une application des algorithmes de la section 4.4.1 sont des automates faibles déterministes. En effet, les transitions étiquetées par $\star$ scindent l'automate en deux parties A_I et A_F. Il est par construction impossible de revenir à A_I une fois que l'on a franchi une transition $\star$: de ce fait, les deux parties ne peuvent partager aucune composante fortement connexe. De plus, A_I contient exclusivement des états non accepteurs alors que A_F contient exclusivement des états accepteurs. Ainsi, l'automate construit est toujours faible et nous savions déjà qu'il était déterministe.

Les RVA qui représentent les solutions à une inéquation sont eux aussi faibles, par le même argument que celui développé pour les équations. Par contre, ils ne sont en général pas déterministes. Nous avons donc deux sources de non-déterminisme quand nous manipulons des automates faibles qui représentent des formules de la théorie additive des entiers et des réels :

- la projection, qui correspond à la quantification logique $(\exists x \in \mathbb{R})$;
- la génération d'un RVA correspondant à une inéquation linéaire.

7.3 Exploitation du théorème de correspondance

Nous en savons néanmoins légèrement plus. En effet, si φ est une formule de $\langle \mathbb{R}, Z, +, \leq \rangle$, il en est de même pour $(\exists x \in \mathbb{R})\varphi$. Nous avons également appris à la section 4.5 que toute inéquation peut se ramener à une formule de notre structure.

Par conséquent, les RVA issus d'une projection ou d'une inéquation représentent encore des formules de la théorie additive des entiers et des réels. Nous savons donc qu'ils définissent toujours un langage de la classe $G_\delta \cap F_\sigma$, malgré le fait qu'ils soient non déterministes. Si nous manipulions des automates faibles généraux, nous n'aurions pas pu faire cette conclusion : les automates faibles non déterministes acceptent seulement la classe F_σ, comme l'a montré le corollaire 6.13. Il nous faut maintenant pouvoir exploiter cette particularité.

À cet effet, nous allons définir une forme particulière d'automates de Büchi déterministes qui capturent exactement les langages $G_\delta \cap F_\sigma$. Cette forme particulière est celle des « *automates intrinsèquement faibles* ». Il s'agit d'automates plus généraux que les automates faibles déterministes, mais qui acceptent la même classe de langages. Nous argumenterons ensuite que *toutes* les opérations que nous appliquons à des automates faibles qui représentent des ensembles pouvant être définis dans la structure $\langle \mathbb{R}, Z, +, \leq \rangle$, *doivent* donner des automates intrinsèquement faibles qu'il est systématiquement possible de convertir en automates faibles.

7.4 Automates intrinsèquement faibles

7.4.1 Définitions

Avant toute chose, précisons la notion de cycle que nous avons déjà employée informellement :

Définition 7.1. Un *cycle* dans le graphe d'un automate A est une suite finie non vide de transitions adjacentes de même orientation $(u_1, \ldots, u_q)$ avec $q \geq 1$. Les deux états aux extrémités du cycle doivent coïncider. Un cycle est dit « accepteur » s'il passe par au moins un état de F. Il est dit « non accepteur » s'il ne passe par aucun état accepteur.

Définition 7.2. Un automate de Büchi (éventuellement non déterministe) est dit *intrinsèquement faible* si chacune de ses composantes fortement connexes accessibles contient soit exclusivement des cycles accepteurs, soit exclusivement des cycles non accepteurs.

Exemple. Dans la figure 7.1 de la page 77, les automates (a), (b) et (c) sont intrinsèquement faibles. Par contre, (d) ne l'est pas : le cycle $1 \xrightarrow{b} 0 \xrightarrow{a} 1$ est non accepteur tandis que le cycle $1 \xrightarrow{b} 2 \xrightarrow{b} 1$ est accepteur. De même, (e) n'est pas intrinsèquement faible (il suffit de regarder les cycles $1 \xrightarrow{b} 2 \xrightarrow{a} 1$ et $1 \xrightarrow{b} 2 \xrightarrow{b} 0 \xrightarrow{a} 1$), tout comme (f) (cycles $0 \xrightarrow{a} 0$ et $1 \xrightarrow{b} 1$). □

Il est facile de constater que tout automate intrinsèquement faible peut être directement converti en un automate faible : une composante fortement connexe devient acceptrice ou non selon qu'elle contient ou non un état accepteur. Il s'agit d'une conséquence directe du corollaire 3.4. Si une composante fortement connexe ne

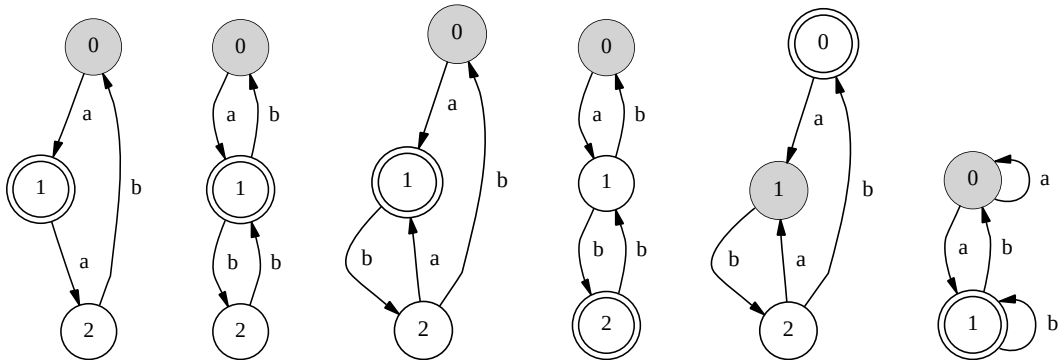


Fig. 7.1 – *Automates intrinsèquement faibles.*

contient aucun cycle, une course ne pourra jamais boucler dans cette composante : on peut donc en choisir librement le statut.

7.4.2 Topologie

Nous prouvons maintenant le théorème suivant :

Théorème 7.1. Un automate de Büchi déterministe accepte un langage de la classe $G_\delta \cap F_\sigma$ si et seulement si il est intrinsèquement faible.

À cet effet, nous allons définir une propriété particulière de certains $\omega-$langages :

Définition 7.3. Le langage $L \subseteq \Sigma^\omega$ a la propriété *d'oscillation dense* si, pour des mots $w_1, w_2, w_3, \ldots$ et des distances non nulles $\varepsilon_1, \varepsilon_2, \varepsilon_3, \ldots$, on a que $\exists w_1 \forall \varepsilon_1 \exists w_2 \forall \varepsilon_2$ $\exists w_3 \forall \varepsilon_3 \ldots$ d'une manière telle que $d(w_i,\ w_{i+1}) < \varepsilon_i$ pour tout $i \geq 1$, que $w_i \in L$ pour tout naturel i impair et que $w_i \notin L$ pour tout naturel i pair.

Un schéma de principe pour illustrer cette propriété est donné à la figure 7.2 de la page 78. Celle-ci affirme que pour tout choix des rayons ε_i, il y a toujours moyen de définir une suite de mots qui oscille à l'infini entre L et L^c. On peut concevoir cette propriété comme un jeu : quelle que soit la distance que donnera le joueur au cours de la suite (aussi petite soit-elle), il sera toujours possible de trouver un nouveau mot dont la distance avec le dernier mot de la suite est inférieure à la distance donnée par le joueur et qui appartient à L si et seulement si le dernier mot de la suite n'appartient pas à L. Une conséquence de cette propriété est que dans un tel langage oscillant, on peut toujours trouver deux mots arbitrairement proches l'un de l'autre, l'un d'eux appartenant à L et l'autre n'appartenant pas à L.

Lemme 7.2. Un automate de Büchi déterministe $A = (Q,\ \Sigma,\ \delta,\ \{s_0\},\ F)$ qui n'est pas intrinsèquement faible a la propriété d'oscillation dense.

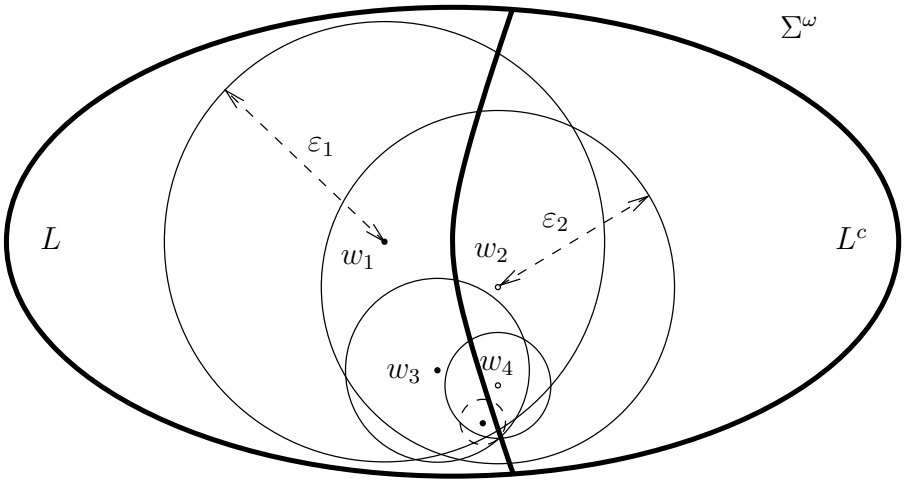


Fig. 7.2 – *Propriété d'oscillation dense pour un $\omega-$langage L.*

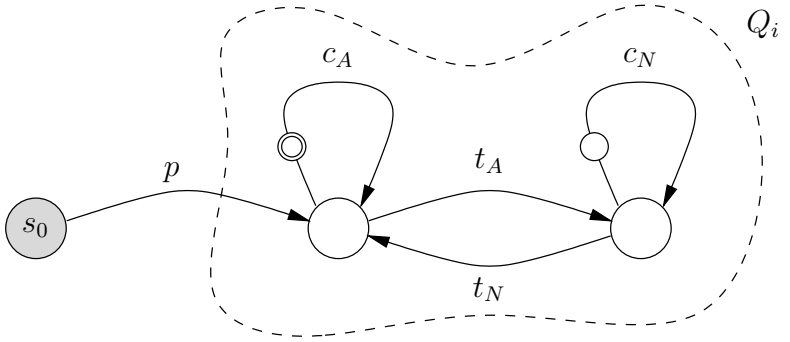


Fig. 7.3 – *Agencement des éléments du lemme 7.2.*

Preuve. Pour cette démonstration, le lecteur peut s'aider de la figure 7.3. Puisque par hypothèse, l'automate n'est pas intrinsèquement faible, il existe une composante fortement connexe Q_i admettant à la fois un cycle accepteur et un cycle non accepteur. Soit p le mot fini permettant d'atteindre au départ de s_0 le premier état du cycle accepteur de la composante fortement connexe fautive. Soit c_A le mot qui étiquette ce cycle et c_N le mot qui étiquette le cycle non accepteur. Posons également t_A (resp. t_N) le mot qui permet de passer du premier état du cycle accepteur (resp. non accepteur) au premier état du cycle non accepteur (resp. accepteur). Ce passage est possible, puisque nous sommes dans une composante fortement connexe.

Il faut maintenant définir une suite oscillante. Posons comme premier mot de la suite : $w_1 = p \cdot c_A^\omega \in L$. Pour alléger les notations, nous allons décomposer tout terme w_i de la suite en un mot fini u_i et un mot infini v_i qui pourra être soit c_A^ω, soit c_N^ω. En d'autres termes, $w_i = u_i \cdot v_i$ pour tout $i \in \mathbb{N}$, avec $u_i \in \Sigma^*$ et $v_i \in \{c_A^\omega, c_N^\omega\}$. Pour w_1, on pose $u_1 = p$ et $v_1 = c_A^\omega$.

Imaginons que nous soyons à un indice m dans la suite oscillante qui a débuté par w_1. Le joueur va donner une distance ε_m et nous allons devoir trouver un mot w_{m+1} dans ce rayon qui doit ou non appartenir à L selon que m est pair ou impair. Nous allons supposer que m est pair (le raisonnement est similaire si m est impair).

Le dernier terme de la suite est $w_m = u_m \cdot v_m$ où $v_m = c_N^\omega$. Il nous faut donner un mot w_{m+1} qui appartient au langage et qui partage avec w_m un préfixe commun suffisamment long pour que $d(w_m,\, w_{m+1}) < \varepsilon_m$. Nous définissons alors $w_{m+1} = u_{m+1} \cdot c_A^\omega$, où $u_{m+1} = u_m \cdot c_N^k \cdot t_N$ avec $k \in \mathbb{N}$ tel que $2^{-(|u_m|+k\cdot|c_N|)} < \varepsilon_m$. On peut par exemple poser $k = \max\{0,\, \lfloor -\frac{\log_2 \varepsilon_m + |u_m|}{|c_N|} \rfloor + 1\}$. On a bien que $w_{m+1} \in L$ et que $d(w_m,\, w_{m+1}) < \varepsilon_m$. Ceci étant valable pour tout m, le langage L est pourvu de la propriété d'oscillation dense. □

Lemme 7.3. Un $\omega-$langage L qui possède la propriété d'oscillation dense ne peut être accepté par un automate faible déterministe.

Preuve. Supposons le contraire, c'est-à-dire qu'il existe un automate faible déterministe A qui accepte L. Considérons le premier mot w_1 de la suite oscillante. Ce mot est accepté : la course ρ_1 correspondante doit donc finir par boucler dans une composante fortement connexe Q_{i_1} acceptrice. Puisque ε_1 peut être choisie librement, elle peut être prise assez petite pour que la course ρ_2 de A sur w_2 puisse atteindre la composante Q_{i_1} avant de commencer à différer de ρ_1 (on a utilisé l'hypothèse selon laquelle A est déterministe, c'est pourquoi la course sur le préfixe commun à w_1 et w_2 est unique). Puisque $w_2 \notin L$, la course ρ_2 doit finir par quitter la composante Q_{i_1} pour atteindre une composante Q_{i_2} qui n'est pas acceptrice et y boucler. On recommence un raisonnement similaire pour w_3 : puisque ε_2 peut être choisie arbitrairement petite, on la prend suffisamment petite pour que la course ρ_3 de A sur w_3 atteigne la composante Q_{i_2}. De là, ρ_3 devra nécessairement quitter Q_{i_2} pour boucler dans une composante acceptrice Q_{i_3}.

En itérant ce raisonnement, on déduit que A doit posséder un nombre infini de composantes fortement connexes. Effectivement, le graphe des composantes fortement connexes est *acyclique* (sinon, on pourrait agglomérer les composantes qui se situent dans le cycle pour reformer une nouvelle composante fortement connexe). On ne peut donc jamais retomber sur une composante déjà répertoriée. Ceci imposerait que l'ensemble Q des états de A soit infini dénombrable, ce qui n'est bien sûr pas permis. □

Grâce à ces deux lemmes, on peut maintenant démontrer le théorème 7.1 :

Preuve. ($\Leftarrow$) Tout automate intrinsèquement faible déterministe peut être converti en un automate faible déterministe. Il n'y a alors plus qu'à appliquer le corollaire 6.13 pour conclure qu'il accepte un $\omega-$langage de la classe $G_\delta \cap F_\sigma$.

($\Rightarrow$) Imaginons que l'automate de Büchi déterministe ne soit pas intrinsèquement faible. Le premier lemme affirme alors que le langage accepté possède la propriété d'oscillation dense. Par le second lemme, un tel langage ne pourra pas être accepté par un automate faible déterministe. Or, il est possible de montrer qu'un langage $G_\delta \cap F_\sigma$ est nécessairement accepté par un automate faible déterministe (voir chapitre A, section A.4). Il y a contradiction. Par conséquent, l'automate est nécessairement intrinsèquement faible. □

7.5 Les RVA en pratique

7.5.1 Manipulation des RVA

Ce théorème concernant les automates intrinsèquement faibles a d'importantes conséquences. Il permet en effet de prouver que *toute* manipulation d'un RVA restreint à $\langle \mathbb{R}, Z, +, \leq \rangle$ donnera *toujours* un automate intrinsèquement faible qu'il est facile de convertir sous une forme faible déterministe. De ce fait, les RVA restreints pourront toujours être vus comme des automates faibles déterministes.

Afin de montrer ce résultat, réalisons un raisonnement par induction pour toutes les opérations définies sur les RVA au chapitre 4. Dans la section 7.2, nous avons déjà dit que les *RVA élémentaires* ou générés depuis une *équation* sont par construction déterministes faibles. Ce n'est pas le cas des *inéquations*. Voyons donc comment ramener l'automate faible non déterministe A représentant une inéquation à son équivalent déterministe :

1. On déterminise l'automate faible A : ce faisant, le tableau de la figure 6.8 nous apprend que l'on obtient un automate co-Büchi $\mathcal{C}$.
2. La proposition 6.9 nous permet de convertir $\mathcal{C}$ en un automate de Büchi déterministe $\mathcal{B}$ qui accepte exactement le complément du langage.
3. Retirons de $\mathcal{B}$ les encodages non valides, ce qui peut se faire grâce à une opération d'intersection avec le RVA représentant $\mathbb{R}^n$, qui est déterministe : ainsi, on conserve le déterminisme de $\mathcal{B}$.
4. Si A représentait l'ensemble défini par la formule φ, $\mathcal{B}$ représente dès lors celui défini par la formule $\neg\varphi$. Donc, $\mathcal{B}$ représente une formule de $\langle \mathbb{R}, Z, +, \leq \rangle$. On sait ainsi que $L_\omega(\mathcal{B}) \in G_\delta \cap F_\sigma$.
5. De ce fait, le théorème 7.1 affirme que l'automate de Büchi déterministe $\mathcal{B}$ est intrinsèquement faible. Convertissons-le en un automate faible déterministe $\mathcal{W}$ grâce à l'algorithme donné à la fin de la section 7.4.1.
6. Complémentons $\mathcal{W}$: on obtient toujours par la figure 6.8 un automate faible déterministe. Là encore, éliminons par une intersection les encodages invalides. On obtient l'automate faible déterministe A'.
7. A' représente exactement l'ensemble défini par la formule $\neg\neg\varphi$, qui est équivalente à φ. A' est donc l'automate faible déterministe recherché.

Remarque 7.4. Nous venons en fait de donner une procédure de déterminisation des RVA. □

Ceci clôture l'étude des cas de base : tout RVA de base peut être mis sous forme faible déterministe. Étudions les cas inductifs et prouvons que les opérations qui ont été définies sur les RVA au chapitre 4 peuvent être modifiées de manière à conserver la forme déterministe faible.

Le résultat est trivial pour l'*union*, l'*intersection* et le *produit cartésien* : c'est une conséquence immédiate de la figure 6.8. La *différence* ne pose pas d'avantage de problèmes. Soient A_1 et A_2 les deux RVA dont on veut réaliser la différence, que l'on sait être sous forme faible déterministe. Il n'y a qu'à compléménter A_2, ce qui reste dans la forme faible déterministe ; puis on en prend l'intersection avec A_1. Or, on sait que l'intersection préserve la forme faible déterministe.

Il reste à traiter le cas des quantifications. Imaginons que nous voulions *projeter* le RVA A représentant un ensemble défini par la formule φ. On procède comme suit :

1. On applique la construction de projection : on obtient ainsi un automate faible non déterministe $\mathcal{W}$.
2. Comme nous l'avons dit à la section 4.3.3, il faut compléter le langage de l'automate faible ainsi obtenu pour qu'il accepte *tous* les encodages valides des vecteurs représentés.
3. Cet automate faible non déterministe représente la formule $(\exists x \in \mathbb{R})\varphi$. Il est dès lors permis d'appliquer la construction de déterminisation que nous avons développée dans le cadre des RVA représentant une inéquation.

La *quantification universelle* d'un RVA représentant une formule φ se ramène à une séquence complémentation – projection – complémentation. On sait donc que l'on conserve la forme faible déterministe, puisque que le complémentation d'un RVA se résume à la différence entre $\mathbb{R}^n$ et le RVA considéré. Il y a néanmoins moyen d'accélérer le calcul, en utilisant la séquence suivante d'opérations :

1. On applique la construction de complémentation d'un automate faible : on obtient alors un automate faible déterministe, que l'on projette. Là encore, il faut compléter le langage de l'automate faible de manière à ce qu'il accepte tous les encodages des vecteurs représentés. On obtient finalement un automate faible non déterministe $\mathcal{W}$.
2. On complémente cet automate faible. L'automate qui en résulte est un automate de Büchi déterministe $\mathcal{B}$, comme le montre la figure 6.8.
3. Par une intersection avec $\mathbb{R}^n$, on retire de $\mathcal{B}$ les encodages invalides, ce qui préserve le déterminisme. $\mathcal{B}$ représente alors la formule $\neg(\exists x \in \mathbb{R})\neg\varphi$: $\mathcal{B}$ accepte par conséquent un langage $G_\delta \cap F_\sigma$.
4. $\mathcal{B}$ est dès lors intrinsèquement faible et il suffit de le convertir en un automate faible.

7.5.2 Conséquences

Les RVA représentant des formules définies dans la structure $\langle \mathbb{R}, Z, +, \leq \rangle$ peuvent systématiquement être convertis en des automates faibles déterministes. Les algorithmes de manipulation sont dès lors grandement simplifiés par rapport aux algorithmes adaptés aux automates de Büchi généraux, ce qui autorise l'utilisation des RVA en pratique.

En particulier, les opérations de déterminisation et de complémentation ont une complexité $2^{\mathcal{O}(n)}$, ce qui est similaire aux opérations correspondantes sur les automates sur mots finis. De ce fait, la manipulation de RVA ne semble pas beaucoup plus difficile que celle d'automates sur mots finis.

Afin de concrétiser ce résultat, nous avons développé une implémentation informatique qui permet le traitement des RVA. Cette implémentation est présentée au chapitre suivant. Avant cela, nous allons d'abord voir comment appliquer les RVA à la logique, puis nous allons exposer deux raffinements que l'on peut apporter à la méthode générale. À partir d'ici, quand nous parlerons de RVA, nous sous-entendrons qu'ils sont restreints à la théorie additive des entiers et des réels.

7.5.3 Décider $\langle \mathbb{R},\ Z, +\ , \leq \rangle$

En transposant ces résultats en termes logiques plutôt que ensemblistes, nous avons une procédure effective pour décider la théorie additive des entiers et des réels.

En effet, toutes les formules atomiques sont représentables par des RVA élémentaires comme nous l'a appris la section 4.2. De plus, la disjonction et la conjonction se ramènent respectivement à une union et à une intersection sur les RVA. L'adjonction d'une nouvelle variable se traite par un produit cartésien. Les quantifications existentielle et universelle se traitent par les opérations correspondantes sur les RVA.

Pour décider si une formule admet un modèle (i.e. une interprétation des variables libres qui la rend vraie), il suffit de construire le RVA correspondant en partant de ses sous-formules les plus internes. Quand celui-ci a été construit, on regarde s'il accepte le langage vide. Si c'est le cas, la formule n'est pas satisfaisable (i.e. elle est toujours fausse). Sinon, la formule admet au moins un modèle : pour en construire un, il suffit de trouver une composante fortement connexe acceptrice Q_i. Le vecteur dont l'encodage est le mot défini par une course atteignant Q_i puis bouclant dans cette composante, est un modèle de la formule.

7.6 Raffinements

7.6.1 Sérialisation

En pratique, on utilise un schéma d'encodage légèrement différent de celui que nous avons présenté au chapitre 2. Au lieu de lire un chiffre simultanément pour toutes les composantes du vecteur (i.e. de manière *synchrone*), on lit les chiffres des composantes successivement, par ordre croissant de position. Cet encodage peut être qualifié de « sérialisé » ou d'« entrelacé ».

Formellement, l'alphabet devient $\Sigma' = \{0, \ \ldots \ , r-1, \star\}$ et le $\omega-$langage des encodages valides pour les vecteurs à n composantes est :

$$V' = (\underbrace{\{0, r-1\} \cdot \ \cdots \ \cdot \{0, r-1\}}_{n}) \cdot (\underbrace{\Sigma' \cdot \ \cdots \ \cdot \Sigma'}_{n})^* \star (\underbrace{\Sigma' \cdot \ \cdots \ \cdot \Sigma'}_{n})^\omega$$

Exemple. L'encodage haut du vecteur (13,5; 2,7), déjà considéré à la fin du chapitre 2, admet en base décimale l'écriture sérialisée :

$$00\ 10\ 32\ \star\ 57\ (00)^\omega$$

où nous avons pris soin de laisser un espace entre chaque cycle de l'encodage. □

Tout RVA synchrone $\mathcal{R}$ peut facilement être converti en un RVA sérialisé $\mathcal{R}'$. Pour chaque transition $p \xrightarrow{\vec{d}} q$ dans $\mathcal{R}$, où $\vec{d}$ est un vecteur de n chiffres, il suffit de retirer cette transition, d'ajouter des états inédits $p_2, \ldots, p_n$ à $\mathcal{R}$ et d'ajouter les transitions $p \xrightarrow{d_1} p_2 \xrightarrow{d_2} \ldots \xrightarrow{d_{n-1}} p_n \xrightarrow{d_n} q$ au graphe du RVA $\mathcal{R}'$. Les transitions étiquetées par $\star$ ne sont pas modifiées.

Réciproquement, il est possible de traduire un RVA sérialisé $\mathcal{R}'$ en un RVA synchrone $\mathcal{R}$. Pour ce faire, il suffit de modifier les transitions de manière adéquate. Initialement, l'ensemble des transitions de $\mathcal{R}$ est vide. S'il est possible de passer dans $\mathcal{R}'$ d'un état p à un état q en lisant un mot formé par n chiffres $w = d_1 d_2 \ldots d_n \in \Lambda^n$, on ajoute la transition $p \xrightarrow{\vec{d}} q$ à $\mathcal{R}$ où on a défini $\vec{d} = (d_1; \ldots; d_n)$. D'autre part, pour chaque transition $p \xrightarrow{\star} q$ appartenant à $\mathcal{R}'$, on ajoute la même transition $p \xrightarrow{\star} q$ à $\mathcal{R}$. Remarquons que cette procédure peut rendre inaccessibles dans $\mathcal{R}$ certains états qui l'étaient dans $\mathcal{R}'$.

On en déduit que ces deux schémas d'encodage ont la même expressivité et donc que tous les théorèmes que nous avons vu sur les RVA synchronisés restent valables dans le monde des RVA sérialisés.

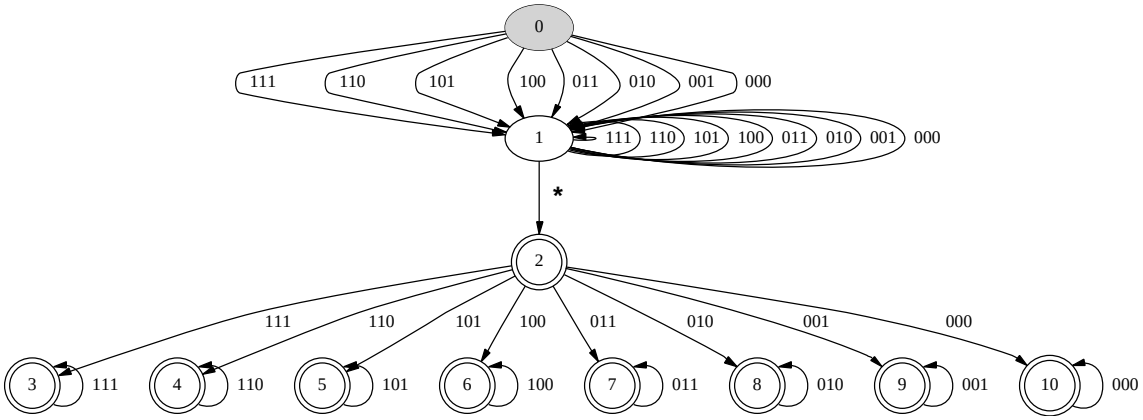
En revanche, les RVA sérialisés peuvent être plus concis que les RVA synchronisés car ils peuvent partager des structures communes à certaines transitions. En contrepartie, les algorithmes permettant de les manipuler sont légèrement plus complexes que ceux adaptés aux RVA synchronisés[2].

Exemple. La figure 7.4 de la page 84 expose les versions synchronisée et sérialisée du même RVA représentant $\mathbb{Z}^3$. Il apparaît clairement que le RVA sérialisé repris en (b) est plus compact que celui repris en (a). □

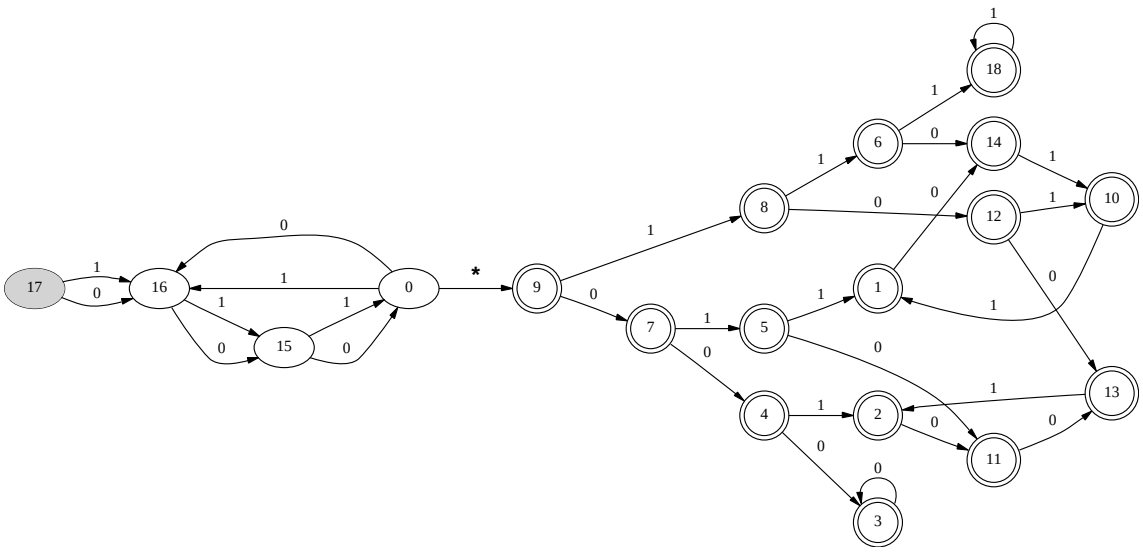
7.6.2 Minimisation

Nous avons montré que nous pouvons nous limiter à l'emploi des automates faibles pour manipuler effectivement les formules de la structure logique $\langle \mathbb{R}, Z, +, \leq \rangle$ et nous avons précisé les algorithmes permettant d'y arriver.

2. Pour être plus précis, il faut modifier les algorithmes de produit cartésien et de projection.



(a)



(b)

Fig. 7.4 – *Sérialisation de RVA et concision.*

Il reste toutefois un problème potentiel. Effectivement, à chaque opération, le nombre d'états est susceptible d'augmenter. Ainsi, à chaque intersection ou union, il est possible que l'automate résultant ait $\mathcal{O}(n_1 \cdot n_2)$ états. Au bout de quelques opérations, le risque existe que l'automate soit trop grand pour tenir en mémoire ou pour être traité en un temps raisonnable. Or, même si un ensemble final est simple, il est souvent obtenu par de longues séquences d'opérations.

Toutefois, rien ne dit que *tous* les états de l'automate soient nécessaires. Il devient en effet probable que certaines parties de l'automate soient redondantes après un certain nombre de manipulations. On voudrait donc pouvoir *minimiser* le nombre d'états de l'automate sans modifier le langage qu'il accepte afin d'obtenir une représentation concise du langage accepté.

Un autre intérêt de la minimisation des automates est qu'elle induit une *canonicité* pour la représentation d'un langage. Les tests d'égalité et d'inclusion entre deux langages peuvent alors être accélérés si l'on se contente de réaliser un test superficiel sur la structure des automates.

Les algorithmes permettant de minimiser le nombre d'états d'automates sur mots finis sont bien connus [Hop71, HU79]. Ils procèdent sur des automates déterministes et sont relativement efficaces, car de complexité $\mathcal{O}(n \cdot \log n)$. Malheureusement, à notre connaissance, peu d'algorithmes applicables aux $\omega-$automates sont venus étoffer les techniques de minimisation. La raison en est principalement qu'en général, l'unicité de l'automate minimal n'est pas garantie.

Récemment toutefois, il a été prouvé que les langages de $G_\delta \cap F_\sigma$ sont suffisamment bien formés pour admettre une forme normale [MS97]. Plus récemment encore, un algorithme permettant de minimiser un automate faible déterministe a été développé [Löd01]. L'idée est de réduire l'automate à une forme normale capable de supporter une application directe de l'algorithme de minimisation classique pour les automates sur mots finis. Cette normalisation est réalisée en un temps linéaire en fonction de la taille des automates. En fait, seul l'ensemble F des états accepteurs est modifié par cette procédure. L'algorithme proposé est donc remarquable, puisque sa complexité globale est identique à celle de la minimisation des automates sur mots finis et qu'il utilise des algorithmes bien connus.

Les idées principales ainsi que l'algorithme de minimisation proprement dit sont exposés à l'annexe B. Malheureusement, les résultats d'exactitude de cet algorithme sont relativement complexes et sortent du cadre de ce mémoire. Le lecteur intéressé pourra se référer à [Löd01]. Nous n'en dirons pas plus dans ce chapitre.

Chapitre 8

Implémentation et résultats

Au cours des chapitres précédent, nous avons développé une théorie nous permettant de manipuler efficacement les RVA, moyennant une légère restriction sur leur expressivité.

Nous avons tout d'abord vu comment encoder des vecteurs de réels sous la forme de mots infinis au chapitre 2, mots infinis qui peuvent être représentés sous la forme d'$\omega-$automates, comme nous l'a indiqué le chapitre 3. Ces résultats, une fois mis en commun, nous ont permis de représenter des sous-ensembles de $\mathbb{R}^n$ sous la forme de RVA au chapitre 4. Malheureusement, les RVA sont très difficiles à manier en pratique : c'est pourquoi nous avons cherché au chapitre 5 à mieux comprendre la structure des ensembles qu'ils représentent. Cette caractérisation s'est faite en termes topologiques, car la topologie des $\omega-$langages acceptés par des automates est un domaine très bien compris, comme nous avons pu le constater au chapitre 6. Finalement, le chapitre 7 nous a montré que les automates faibles déterministes, forme très particulière d'automates de Büchi, nous suffisent pour manipuler les RVA restreints à la structure $\langle \mathbb{R}, Z, +, \leq \rangle$. Or, les algorithmes applicables aux automates faibles sont d'une complexité comparable à ceux qui s'appliquent aux automates sur mots finis. C'est pourquoi il est probable que les RVA soient utilisables en pratique.

Dans le but d'évaluer le comportement réel des RVA restreints, nous avons réalisé une implémentation des algorithmes permettant leur manipulation. Cette implémentation s'est intégrée au sein d'un outil expérimental déjà existant dont le noyau permettait la manipulation d'automates sur mots finis. Pour ce faire, nous avons ajouté à cet outil toutes les primitives nécessaires à la manipulation d'automates de Büchi, co-Büchi et faibles. L'outil ainsi modifié permet donc non seulement la gestion des RVA, mais est également prêt pour intégrer d'autres méthodes basées sur les automates sur mots infinis.

Dans ce dernier chapitre, nous présentons cet outil ainsi que quelques résultats expérimentaux.

8.1 L'outil LASH

8.1.1 Présentation

L'outil LASH, pour *Liège Automata-based Symbolic Handler* [LASH], est un programme principalement rédigé en C qui a pour mission de vérifier des programmes à espace d'états infini par des méthodes symboliques basées sur les automates. Il exploite à cet effet les résultats proposés dans [Boi98]. LASH est actuellement en cours de développement à l'Université de Liège et possède les éléments suivants :

(i) un noyau permettant la gestion d'automates sur mots finis ;

(ii) la représentation et la manipulation symbolique d'ensembles finis ou infinis de vecteurs d'entiers sous la forme de NDD (*Number Decision Diagrams*), ainsi que de contenus de files FIFO non bornées sur un alphabet fini sous la forme de QDD (*Queue Decision Diagrams*) ;

(iii) un mécanisme spécifique permettant d'analyser en un temps fini des espaces d'états infinis grâce au concept de *méta-transitions* ;

(iv) la capacité de traiter des systèmes décrits dans les formalismes de haut niveau PROMELA [Hol91] et IF [BGGM00, Lat00].

8.1.2 Extensions

Dans le cadre de ce travail, nous avons apporté à LASH deux nouvelles fonctionnalités :

(i) la modification du noyau de manière à ce qu'il puisse manipuler certaines classes d'automates sur mots infinis, à savoir les automates de Büchi, co-Büchi et faibles (y compris la procédure de minimisation correspondante) ;

(ii) la création d'un *package* permettant la manipulation des RVA restreints à la théorie additive des entiers et des réels, conformément à ce qui a été présenté dans ce document au chapitre 7.

Ces fonctionnalités ont été intégrées dans LASH de manière à minimiser l'ampleur des modifications[1]. Le code source du *package* pour les RVA peut être trouvé à l'adresse :

`http://www.montefiore.ulg.ac.be/~jodogne/tfe/rva.tar.gz`

Cette implémentation ouvre la voie à l'analyse de certaines classes de systèmes hybrides, dont les variables peuvent être à la fois réelles et entières [BBR97]. Il devient en effet possible grâce aux RVA d'agglomérer des ensembles d'états (potentiellement infinis) de tels systèmes sous la forme d'un automate.

1. Nous avons également ajouté à LASH un module permettant l'exportation graphique d'automates par le biais de l'outil « Graphviz » développé par Bell Labs [GViz]. En vérité, de nombreux graphes d'automates qui ont été présentés dans ce document ont été générés par Graphviz.

Remarque 8.1. Notre implémentation a permis de déceler et de corriger une imperfection dans l'algorithme de minimisation des automates faibles proposé dans la version préliminaire de l'article [Löd01]. □

8.2 Résultats expérimentaux

8.2.1 Génération d'ensembles périodiques

La relation $Z(x)$ permet de générer des ensembles périodiques de réels. Nous allons présenter dans cette section trois exemples d'ensembles périodiques et montrer les RVA correspondants. Toutes les figures auxquelles nous ferons référence sont groupées à la fin de ce chapitre, à partir de la page 91.

La fonction « palier »

En guise de premier exemple, les RVA sont à même de représenter l'ensemble $S_1 = \{(x,y) \mid y = \lfloor x \rfloor\}$. Cet ensemble est représenté à la figure 8.1. En d'autres termes, il est possible de représenter par un RVA la fonction « palier » introduite à la section 2.2.1. En effet :

$$S_1 = \{(x,y) \in \mathbb{R}^2 \mid Z(y) \wedge y \leq x \wedge x < y + 1\}$$

par définition de la fonction palier. On peut représenter cet ensemble S_1 par la combinaison suivante de RVA :

$$S_1 = (\mathbb{R} \times \mathbb{Z}) \cap \{(x,y) \mid y \leq x\} \cap \{(x,y) \mid y \leq x - 1\}^c$$

Notre implémentation des RVA fournit l'automate minimisé de la figure 8.2 comme résultat pour cette combinaison. Remarquons que l'automate possède bien la forme faible déterministe.

Dents de scie

Le second exemple est extrait de [BBR97]. Il s'agit de représenter la fonction « en dents de scie » de la figure 8.3. On voit que l'on a :

$$S_2 = \left\{ (x_1,x_2) \in \mathbb{R}^2 \mid \begin{array}{l} (\exists x_3 \in \mathbb{R}) \\ (\exists x_4 \in \mathbb{Z}) \end{array} \left(\begin{array}{l} (x_1 = x_3 + 2 \cdot x_4) \wedge \\ \left(\begin{array}{l} (x_3 \geq 0 \wedge x_3 \leq 1 \wedge x_2 = x_3) \vee \\ (x_3 \geq 1 \wedge x_3 \leq 2 \wedge x_2 = 2 - x_3) \end{array} \right) \end{array} \right) \right\}$$

Dans cette formule, x_4 sert à recopier horizontalement une portion de la dent de scie décrite par le couple de variables (x_3,x_2). Le RVA construit au départ de cette formule est donné à la figure 8.4.

Dallage périodique

Le but de ce troisième exemple est de représenter par un RVA le dallage périodique illimité de la figure 8.5, ensemble que nous nommerons S_3. Le lecteur peut vérifier que l'on a bien :

$$S_3 = \left\{ (x_1,x_2) \in \mathbb{R}^2 \mid \begin{array}{c} (\exists x_3,x_4 \in \mathbb{R}) \\ (\exists x_5,x_6 \in \mathbb{Z}) \end{array} \left(\begin{array}{c} (x_1 = x_3 + 2 \cdot x_5) \wedge \\ (x_2 = x_4 + 2 \cdot x_6) \wedge \\ (x_3 \geq 0 \wedge x4 \leq 1 \wedge x_4 \geq x_3) \end{array} \right) \right\}$$

Les variables entières x_5 et x_6 sont introduites pour imposer la périodicité respectivement sur l'axe horizontal et sur l'axe vertical, tandis que l'ensemble de $\mathbb{R}^2$ décrit par (x_3,x_4) correspond au triangle qui sera répété à l'infini. L'automate fourni par le module RVA est présenté à la figure 8.6.

8.2.2 Comportement de l'algorithme de déterminisation

L'opération la plus complexe que l'on est amené à effectuer sur des RVA est l'algorithme de déterminisation hérité des automates co-Büchi. Celui-ci pourrait induire une explosion exponentielle du nombre des états, comme nous l'avons vu à la section 6.3.3. Toutefois, nous avons déjà argumenté à la remarque 6.4 que cet algorithme est du même ordre de complexité que celui de déterminisation des automates sur mots finis. Il est donc intéressant de comparer la déterminisation des RVA avec celle des NDD, qui sont des automates sur mots finis représentant uniquement des sous-ensembles de $\mathbb{Z}^n$ et réputés offrir un très bon comportement moyen malgré un coût théorique élevé.

À cet effet, nous avons généré les RVA et les NDD représentant les mêmes sous-ensembles de $\mathbb{Z}^n$ correspondant aux solutions de systèmes aléatoires d'inéquations. Ensuite, nous avons réalisé une quantification sur ces ensembles afin d'introduire du non-déterminisme, puis nous avons appliqué les algorithmes de déterminisation et de minimisation. La figure 8.7 présente les résultats pour 300 systèmes de 6 inéquations avec 3 variables et des coefficients appartenant à l'intervalle $[-5,\ldots,5]$.

Deux conclusions peuvent être tirées de ce graphique :

1. Tous les points se situent sous la ligne d'égalité en pointillés. On en déduit que pour tous les systèmes qui ont été générés, le nombre d'états des RVA et des NDD décroît après une séquence projection – déterminisation, là où on aurait pu craindre une explosion. Un tel constat avait déjà été tiré dans le cadre strict des NDD [WB00].
2. Même si le nombre d'états d'un RVA est toujours supérieur au nombre d'états du NDD correspondant (un RVA doit représenter la partie fractionnaire des solutions ainsi que leurs encodages duals, au contraire des NDD), le comportement des RVA et des NDD est très similaire. On constate en effet que les nuages de points pour les RVA et les NDD partagent une même forme et se recouvrent fortement.

Il s'avère donc que (i) les quantifications ne sont pas aussi problématiques que ce à quoi on aurait pu s'attendre et que (ii) utiliser des RVA n'est pas beaucoup plus complexe que manipuler des NDD. Il est par conséquent raisonnable de penser que le bon comportement des NDD s'étend aux RVA.

Or, les RVA possèdent une expressivité largement supérieure aux NDD puisqu'ils peuvent représenter des réels en plus des entiers. Ceci indique donc que les RVA semblent être une méthode efficace pour représenter des ensembles d'entiers et de réels. Cette intuition demandera toutefois à être consolidée par des études de cas concrets.

Remarque 8.2. Notre implémentation nous a permis de découvrir une heuristique qui permet d'optimiser l'algorithme de déterminisation des automates co-Büchi. Elle consiste simplement à marquer préalablement comme accepteurs tous les états de l'automate qui n'appartiennent à aucun cycle. Ce faisant, on maximise le nombre d'états accepteurs sans changer le langage accepté. Quand on observe la construction de la figure 6.6 (à la page 69), ceci a pour effet de minimiser la taille des ensembles R puisqu'après avoir atteint un point de rupture, on réinitialise R à S duquel on a retiré les états accepteurs. De ce fait, si un point de rupture doit être atteint sur un chemin de l'automate déterminisé, il le sera plus vite, réduisant d'autant la taille de l'automate déterminisé. Bien que certains automates soient légèrement plus rapidement déterminisés si on ne l'applique pas, nous avons pu constater que cette heuristique a un impact globalement très positif sur la déterminisation. □

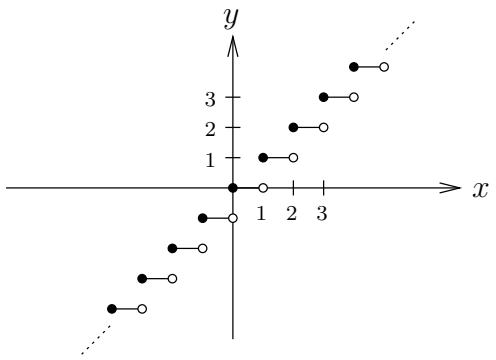


Fig. 8.1 – *Ensemble de $\mathbb{R}^2$ défini par la fonction « palier ».*

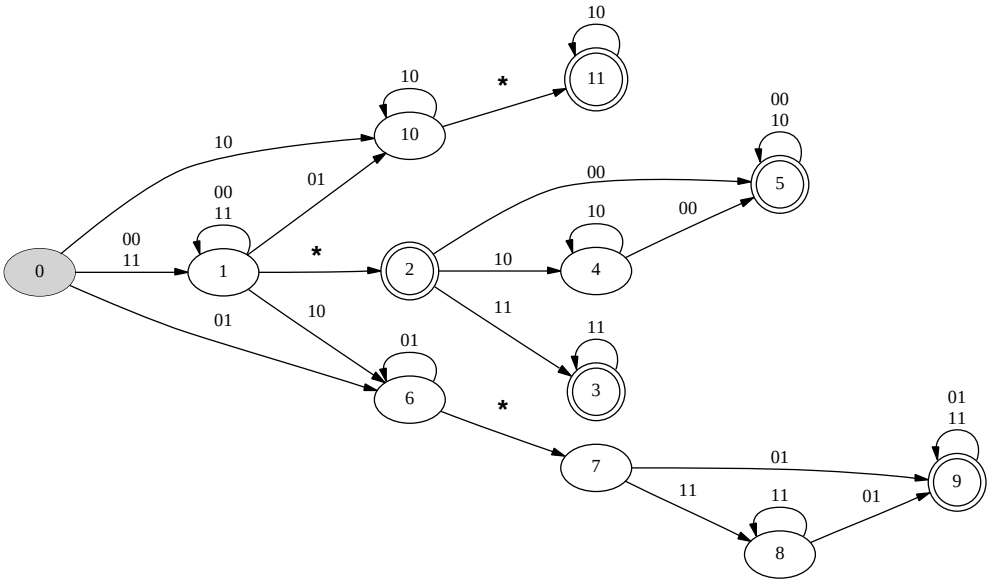


Fig. 8.2 – *RVA en base binaire représentant la fonction « palier ».*

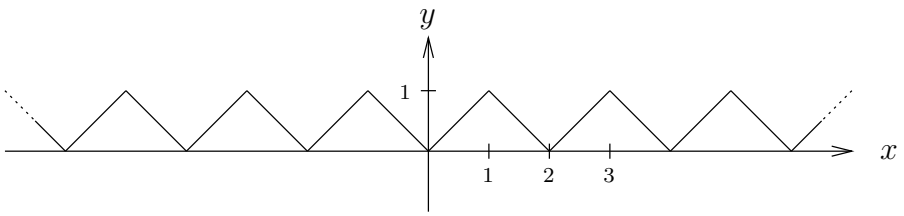


Fig. 8.3 – *Ensemble périodique en dents de scie.*

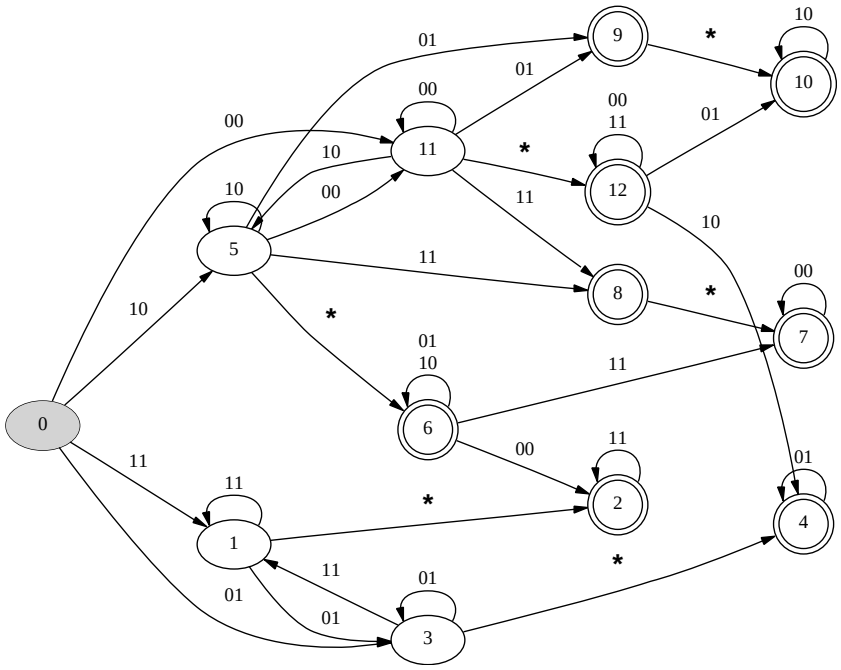


Fig. 8.4 – *RVA en base binaire représentant l'ensemble en dents de scie.*

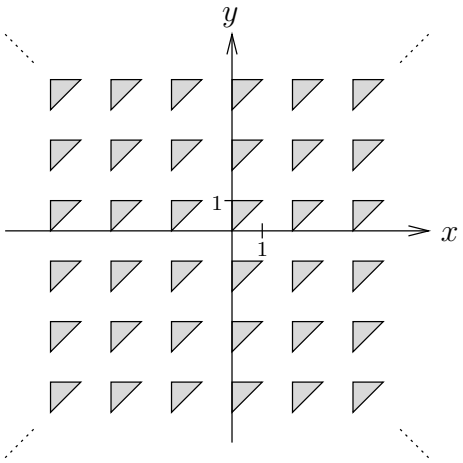


Fig. 8.5 – *Dallage périodique avec des triangles.*

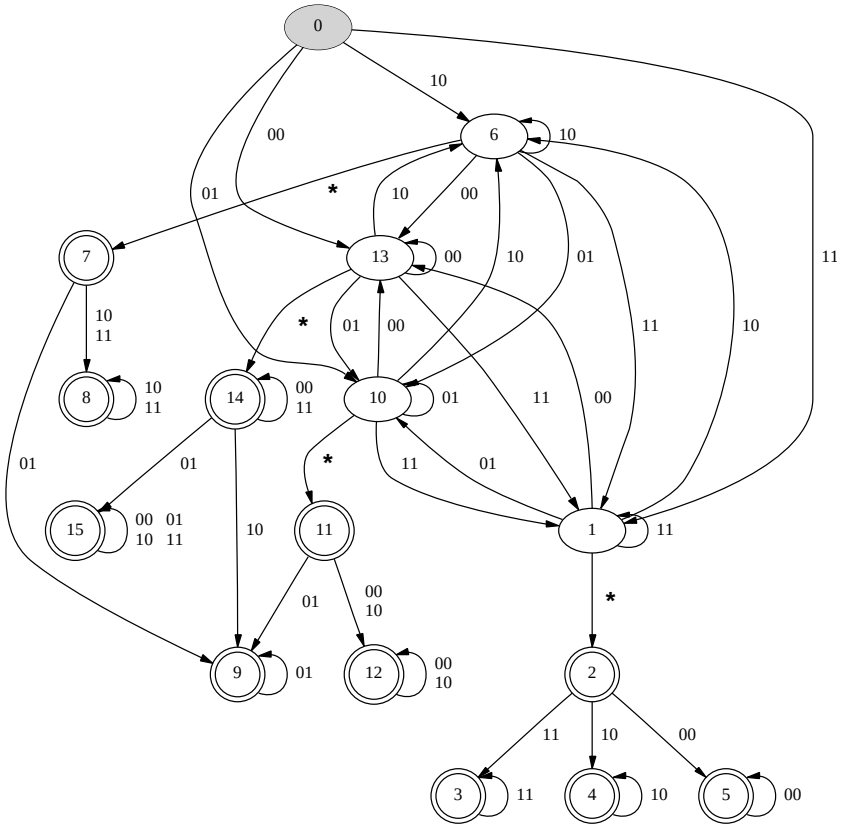


Fig. 8.6 – *RVA en base binaire représentant le dallage triangulaire.*

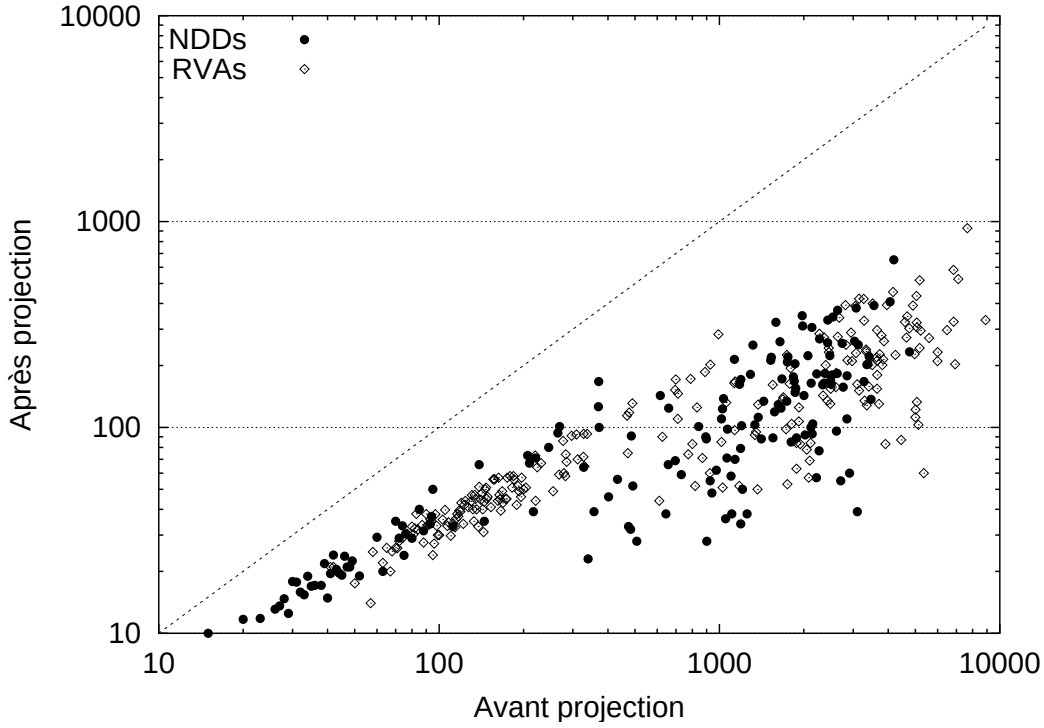


Fig. 8.7 – *Taille d'automates représentant des sous-ensembles de $\mathbb{Z}^3$, avant et après projection sur une des trois variables.*

Chapitre 9

Conclusion

Ce mémoire s'est attaché à fournir la preuve que les automates faibles suffisent pour manipuler les RVA représentant des formules de la structure $\langle \mathbb{R},\ Z,\ +\ ,\ \leq \rangle$. Ceci simplifie grandement les algorithmes à mettre en œuvre tout en les rendant considérablement plus efficaces, ce qui a permis l'implémentation d'un programme de manipulation de RVA. La minimisation des RVA devient également praticable.

Deux applications principales peuvent être dégagées. Premièrement, il devient possible de représenter des ensembles périodiques comportant à la fois des entiers et des réels, puis de réaliser des quantifications et des tests d'inclusion sur ces ensembles. À notre connaissance, ceci n'était à la portée d'aucun autre outil existant. Il est permis de penser que cette représentation implicite aura de larges applications dans le domaine de la vérification automatisée de systèmes hybrides.

Deuxièmement, on dispose d'un outil capable de décider efficacement la théorie additive des entiers et des réels, ce que les RVA généraux ne permettaient pas en pratique. Les algorithmes correspondants sont en effet très complexes et probablement hors de portée d'une implémentation pratique.

En marge de ces thèmes centraux, l'application des RVA à d'autres domaines est envisageable. Par exemple, certaines bases de données temporelles ou de contraintes demandent de pouvoir représenter des ensembles périodiques de réels.

Du point de vue des performances, les premiers essais tendent à montrer que les RVA présentent un coût pratique très similaire à celui des NDD, qui sont réputés offrir un excellent comportement moyen malgré un coût théorique élevé. En outre, nous avons vu que les RVA correspondant à des contraintes périodiques non triviales restent relativement simples. Il y a donc de bonnes raisons de penser que les RVA sont exploitables en pratique.

Il faudra toutefois étayer cette affirmation par des études de cas réalistes. De plus, avant de pouvoir appliquer les RVA à la vérification symbolique de systèmes hybrides, il est nécessaire d'étendre les résultats arithmétiques permettant, grâce au concept de *méta-transition* [BW94], l'analyse en un temps fini de systèmes ne comportant que des variables entières. Cette étude reste encore très partielle pour les systèmes hybrides. Tout ceci ouvre la voie à de nouvelles idées de recherche.

Bibliographie

[ACH+95] R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, et S. Yovine. « The Algorithmic Analysis of Hybrid Systems ». *Theoretical Computer Science*, 138(1):3–34, février 1995.

[AHU74] A. V. Aho, J. E. Hopcroft, et J. D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.

[BBR97] B. Boigelot, L. Bronne, et S. Rassart. « An Improved Reachability Analysis Method for Strongly Linear Hybrid Systems ». Dans *Proc. 9th International Conference on Computer-Aided Verification*, volume 1254 de *Lecture Notes in Computer Science*, pages 167–178, Haifa, Israël, juin 1997. Springer-Verlag.

[BG96] B. Boigelot et P. Godefroid. « Symbolic Verification of Communication Protocols with Infinite State Spaces using QDDs ». Dans *Proc. Computer Aided Verification*, volume 1102 de *Lecture Notes in Computer Science*, pages 1–12, New-Brunswick, NJ, USA, juillet 1996. Springer-Verlag.

[BGGM00] M. Bozga, L. Ghirvu, S. Graf, et L. Mounier. « IF: A Validation Environment for Timed Asynchronous Systems ». Dans *Proceedings of Conference on Computer Aided Verification (CAV'00)*, LNCS, Chicago, juin 2000. Springer Verlag.

[BHMV94] V. Bruyère, G. Hansel, C. Michaux, et R. Villemaire. « Logic and p-recognizable Sets of Integers ». *Bulletin of the Belgian Mathematical Society*, 1(2):191–238, mars 1994.

[BJW01] B. Boigelot, S. Jodogne, et P. Wolper. « On the Use of Weak Automata for Deciding Linear Arithmetic with Integer and Real Variables ». Dans *Proc. International Joint Conference on Automated Reasoning (IJCAR)*, Lecture Notes in Computer Science, Siena, juin 2001. Springer-Verlag. À paraître.

[BL01] B. Boigelot et L. Latour. « Counting the Solutions of Preburger

Equations without Enumerating them », 2001. Soumis pour publication.

[Boi98] B. Boigelot. *Symbolic Methods for Exploring Infinite State Spaces.* Thèse de Doctorat, Université de Liège, 1998.

[Bou74] N. Bourbaki. *Éléments de Mathématiques, Topologie Générale.* Hermann, Paris, 1974.

[BRW98] B. Boigelot, S. Rassart, et P. Wolper. « On the Expressiveness of Real and Integer Arithmetic Automata ». Dans *Proc. 25th Colloquium on Automata, Programming, and Languages (ICALP)*, volume 1443 de *Lecture Notes in Computer Science*, pages 152–163. Springer-Verlag, juillet 1998.

[Bry92] R.E. Bryant. « Symbolic Boolean Manipulation with Ordered Binary-Decision Diagrams ». *ACM Computing Surveys*, 24(3):293–318, 1992.

[Büc62] J. R. Büchi. « On a Decision Method in Restricted Second Order Arithmetic ». Dans *Proceedings of the International Congress on Logic, Method, and Philosophy of Science*, pages 1–12, Stanford, CA, USA, 1962. Stanford University Press.

[BW94] B. Boigelot et P. Wolper. « Symbolic Verification with Periodic Sets ». Dans *Proc. 6rd Workshop on Computer Aided Verification*, volume 818 de *Lecture Notes in Computer Science*, pages 55–67, Stanford, juin 1994. Springer-Verlag.

[Car93] O. Carton. *Mots Infinis, ω-Semigroupes et Topologie.* Thèse de Doctorat, Université Paris 7, 1993.

[CVWY92] C. Courcoubetis, M. Y. Vardi, P. Wolper, et M. Yannakakis. « Memory Efficient Algorithms for the Verification of Temporal Properties ». *Formal Methods in System Design*, 1:275–288, 1992.

[dM95] P. A. de Marneffe. « *Introduction à l'Algorithmique I* ». Université de Liège, 1995. Notes de cours à l'usage des candidatures ingénieur et informatique (version 2.81).

[Dub95] D. P. Dubhashi. « Complexity of Logical Theories ». Lecture Series LS-95-5, BRICS, Department of Computer Science, University of Aarhus, septembre 1995.

[Eil74] S. Eilenberg. *Automata, Languages, and Machines (Volume A).* Academic Press, 1974.

[Éti86] J. Étienne. « *Géométrie Analytique — Notions de Géométrie Dif-*

férentielle ». Université de Liège, 1986. Notes de cours à l'usage des candidatures ingénieur.

[Éti97] J. Étienne. « *Analyse Mathématique* ». Université de Liège, 1997. Notes de cours à l'usage des candidatures ingénieur et informatique.

[GG90] P. Gochet et P. Gribomont. *Logique 1 : Méthodes pour l'Informatique Fondamentale*. Hermès, Paris, 1990.

[GH93] P. Godefroid et G. J. Holzmann. « On the Verification of Temporal Properties ». Dans *Proc. 13th IFIP WG 6.1 International Symposium on Protocol Specification, Testing, and Verification*, pages 109–124, Liège, mai 1993. North-Holland.

[Gri00] P. Gribomont. « *Logiques pour l'Intelligence Artificielle* ». Université de Liège, octobre 2000. Notes de cours à l'usage des licences en informatique.

[GViz] « The Graphviz – Graph Drawing Program ». AT&T & Lucent Bell Labs. Disponible à l'adresse : `www.research.att.com/sw/tools/graphviz`.

[Hen96] T. A. Henzinger. « The Theory of Hybrid Automata ». Dans *Proceedings, 11th Annual IEEE Symposium on Logic in Computer Science*, pages 278–292, New Brunswick, New Jersey, juillet 1996.

[Hol91] G. Holzmann. *Design and Validation of Computer Protocols*. Prentice-Hall International Editions, 1991.

[Hop71] J. E. Hopcroft. « An $n \log n$ algorithm for minimizing states in a finite automaton ». *Theory of Machines and Computation*, pages 189–196, 1971.

[HU79] J. Hopcroft et J. Ullman. *Introduction to Automata Theory, Languages and Computation*. Reading. Addisson-Wesley, 1979.

[Jod01] S. Jodogne. « Correspondance entre les Ensembles $F_\sigma \cap G_\delta$ de $\mathbb{R}^n$ et de Σ^ω ». Rapport Technique, Université de Liège, janvier 2001.

[Knu83] D. E. Knuth. « The `WEB` System of Structured Documentation ». Stanford Computer Science Report CS980, Université de Stanford, Stanford, CA, septembre 1983.

[KV97] O. Kupferman et M. Y. Vardi. « Weak Alternating Automata are not that Weak ». Dans *Proc. 5th Israeli Symposium on Theory of Computing and Systems*, pages 147–158. IEEE Computer Society Press, 1997.

[Lan69] L. H. Landweber. « Decision Problems for ω-Automata ». *Math. System Theory*, 3:376–384, 1969.

[LASH] « The Liège Automata-based Symbolic Handler (LASH) ». Disponible à l'adresse : `http://www.montefiore.ulg.ac.be/~boigelot/research/lash/`.

[Lat00] L. Latour. Development of a Compiling Front-end for the LASH tool. Mémoire de Diplôme d'Études Complémentaires, Université de Liège, 2000.

[Löd97] C. Löding. « Methods for the Transformation of $\omega-$Automata : Complexity and Connection to Second Order Logic ». Master's thesis, Université de Kiel, 1997. Revised Version.

[Löd99] C. Löding. « Optimal Bounds for the Transformation of $\omega-$Automata ». Dans *Proceedings of the 19th Conference on Foundations of Software Technology and Theoretical Computer Science*, volume 1738 de *Lecture Notes in Computer Science*, pages 97–109. Springer, décembre 1999.

[Löd01] C. Löding. « Efficient Minimization of Deterministic Weak $\omega-$Automata », 2001. Soumis pour publication.

[Man88] M. Mano. *Computer Engineering, Hardware Design.* Prentice Hall International, 1988.

[MH84] S. Miyano et T. Hayashi. « Alternating Finite Automata on ω-Words ». *Theoretical Computer Science*, 32:321–330, 1984.

[Mos84] A. W. Mostowski. « Regular Expressions for Infinite Trees and a Standard Form of Automata ». *Theoretical Computer Science*, 32:321–330, 1984.

[MS87] D.E. Muller et P.E. Schupp. « Alternating automata on infinite trees ». *Theoretical Computer Science*, 54,:267–276, 1987.

[MS97] O. Maler et L. Staiger. « On Syntactic Congruences for ω-Languages ». *Theoretical Computer Science*, 183(1):93–112, 1997.

[MSS86] D.E. Muller, A. Saoudi, et P.E. Schupp. « Alternating Automata, the Weak Monadic Theory of the Tree and its Complexity ». Dans *Proc. 13th Int. Colloquium on Automata, Languages and Programming.* Springer-Verlag, 1986.

[noweb] « The `noweb` Literate-programming Tool ». Par N. Ramsey. Disponible

à l'adresse : `http://www.eecs.harvard.edu/~nr/noweb`.

[Nuu95] E. Nuutila. « *Efficient Transitive Closure Computation in Large Digraphs* ». PhD thesis, Helsinki University of Technology, Espoo, Finlande, 1995.

[PP01] D. Perrin et J.-E. Pin. Infinite Words, 2001. Version préliminaire disponible à l'adresse : `http://www.liafa.jussieu.fr/~jep/Resumes/InfiniteWords.html`.

[Pri00] C. Prieur. *Fonctions Rationnelles de Mots Infinis et Continuité*. Thèse de Doctorat, Université Paris 7, 2000.

[Rou93] M. Roubens. « *Théorie des Graphes* ». Université de Liège, 1993. Notes de cours à l'usage de la seconde licence en sciences mathématiques.

[RS97] F. Reinhardt et H. Soeder. *Atlas des Mathématiques*. Librairie Générale Française, 1997.

[Saf89] S. Safra. « *Complexity of Automata on Infinite Objects* ». PhD thesis, Weizmann Institute of Science, Rehovot, Israël, 1989.

[Sta97] L. Staiger. $\omega-$Languages. Dans *Handbook of Formal Languages*, volume 3, pages 339–387. Springer-Verlag, Berlin, 1997.

[SVW87] A. P. Sistla, M. Y. Vardi, et P. Wolper. « The Complementation Problem for Büchi Automata with Applications to Temporal Logic ». *Theoretical Computer Science*, 49:217–237, 1987.

[Tan96] A. S. Tanenbaum. *Computer Networks*. Prentice-Hall International, 1996. Troisième édition.

[Tar72] R. E. Tarjan. « Depth-first Search and Linear Graph Algorithms ». *SIAM J. Computing*, 1(2):146–160, 1972.

[Tho90] W. Thomas. « *Automata on Infinite Objects* », volume A de *Handbook of Theoretical Computer Science*, Chapitre 4, pages 133–191. Elsevier, Amsterdam, J. V. Leeuwen édition, 1990.

[Var96] M. Y. Vardi. « An Automata-Theoretic Approach to Linear Temporal Logic ». Dans *Logics for Concurrency : Structure versus Automata*, volume 1043 de *Lecture Notes in Computer Science*, pages 238–266. Springer-Verlag, 1996.

[WB95] P. Wolper et B. Boigelot. « An Automata-Theoretic Approach to Presburger Arithmetic Constraints ». Dans *Proc. Static Analysis*

Symposium, volume 983 de *Lecture Notes in Computer Science*, pages 21–32, Glasgow, septembre 1995. Springer-Verlag.

[WB00] P. Wolper et B. Boigelot. « On the Construction of Automata from Linear Arithmetic Constraints ». Dans *Proc. 6th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, volume 1785 de *Lecture Notes in Computer Science*, pages 1–19, Berlin, mars 2000. Springer-Verlag.

[Wes78] C.H. West. « Generalized Technique for Communication Protocol Validation ». *IBM J. of Res. and Devel.*, 22:393–404, 1978.

[Wol91] P. Wolper. *Introduction à la Calculabilité*. InterEditions, 1991.

[Wol98] P. Wolper. « *The Algorithmic Verification of Reactive Systems* ». FUNDP (Namur), 1998. Conférences de la Chaire Francqui.

[WS75] K. Wagner et L. Staiger. « Finite Automata Acceptation of Infinite Sequences ». Dans A. Blikle, éditeur, *Mathematical Foundations of Computer Science, 3rd Symposium*, volume 28 de *Lecture Notes in Computer Science*, pages 69–72, Jadwisin, juin 1975. Springer.

Deuxième partie

Annexes

Annexe A

Démonstrations omises

Nous avons décidé de simplifier au possible l'exposé théorique de ce document. Certains théorèmes qui ont été mentionnés font eux-mêmes appel dans leur démonstration à des propositions qui n'ont pas d'utilité dans la suite de l'exposé. D'autres démonstrations sont assez techniques ou bien ne peuvent être résumées en quelques paragraphes. La présente annexe répertorie et démontre ces propositions intermédiaires, dont la connaissance n'est pas requise pour une bonne compréhension de ce mémoire.

A.1 Encodages duals

Preuve du théorème 2.1 (page 12). Soit un mot obtenu par les règles (2.4). Considérons les premiers termes S_n (avec $n \geq 1$) obtenus par la fonction de décodage appliquée à la partie fractionnaire $u_0u_1u_2\ldots$:

$$\begin{aligned} S_n = \sum_{i=0}^{n-1} \frac{u_i}{r^{i+1}} &= \frac{\lfloor rx \rfloor}{r} + \sum_{i=1}^{n-1} \frac{\lfloor r^{i+1}x \rfloor - r\lfloor r^i x \rfloor}{r^{i+1}} = \frac{\lfloor rx \rfloor}{r} + \sum_{i=2}^{n} \frac{\lfloor r^i x \rfloor}{r^i} - \sum_{i=1}^{n-1} \frac{\lfloor r^i x \rfloor}{r^i} \\ &= \frac{\lfloor rx \rfloor}{r} + \left(\frac{\lfloor r^n x \rfloor}{r^n} + \sum_{i=2}^{n-1} \frac{\lfloor r^i x \rfloor}{r^i} \right) - \left(\frac{\lfloor rx \rfloor}{r} + \sum_{i=2}^{n-1} \frac{\lfloor r^i x \rfloor}{r^i} \right) = \frac{\lfloor r^n x \rfloor}{r^n} \end{aligned}$$

Par ailleurs, si on se rappelle que $0 \leq y - \lfloor y \rfloor < 1$ pour tout y, on obtient :

$$0 \leq \frac{r^n x - \lfloor r^n x \rfloor}{r^n} < \frac{1}{r^n} \Leftrightarrow 0 \leq x - \frac{\lfloor r^n x \rfloor}{r^n} < \frac{1}{r^n} \qquad \text{pour tout } n > 1$$

Donc, la suite des sommes partielles S_n tend vers x. La limite d'une suite étant unique, on a :

$$d_{\mathbb{R}}(w) = \lim_{n \to \infty} S_n = \sum_{i=0}^{\infty} \frac{u_i}{r^{i+1}} = x$$

Ce qui prouve la première partie de l'énoncé. Supposons maintenant que x ait plusieurs écritures, dont deux sont :

$$x = d_{\mathbb{R}}(0 \star w' dt) = d_{\mathbb{R}}(0 \star w' d' t') \text{ avec } w' \in \Lambda^*,\ d \in \Lambda \text{ et } t,t' \in \Lambda^\omega \tag{A.1}$$

On peut supposer sans restriction que $d < d'$ (si $d = d'$, on étend le mot w' et si $d > d'$, on permute les deux écritures). Remarquons que pour tout $u \in \Lambda^*$ et $v \in \Lambda^\omega$:

$$d_{\mathbb{R}}(0 \star uv) = \frac{1}{r^{|u|}} \left(d_{\mathbb{N}}(u) + d_{\mathbb{R}}(0 \star v)\right) \tag{A.2}$$

En appliquant cette règle à (A.1), on obtient que $d_{\mathbb{R}}(0 \star dt) = d_{\mathbb{R}}(0 \star d't')$. En réappliquant la formule (A.2), on arrive à :

$$d + d_{\mathbb{R}}(0 \star t) = d' + d_{\mathbb{R}}(0 \star t') \Leftrightarrow d - d' = d_{\mathbb{R}}(0 \star t') - d_{\mathbb{R}}(0 \star t) \tag{A.3}$$

Si on se rappelle que $0 \leq d_{\mathbb{R}}(0 \star v) \leq 1$ pour tout mot $v \in \Lambda^\omega$, on obtient au pire $d - d' \geq -1$. Puisque $d - d' < 0$, on a finalement $d = d' - 1$: ceci impose que x ait au plus deux écritures. En injectant ce résultat dans (A.3), $d_{\mathbb{R}}(0 \star t') = d_{\mathbb{R}}(0 \star t) - 1$, d'où $d_{\mathbb{R}}(0 \star t) = 1$ et $d_{\mathbb{R}}(0 \star t') = 0$. Ceci n'est possible que si $t = (r-1)^\omega$ et si $t' = 0^\omega$. Les deux écritures obtenues sont exactement celles des équations (2.5). □

Remarque A.1. L'ouvrage [Eil74] étudie plus en profondeur les propriétés de l'encodage de parties fractionnaires. □

A.2 Topologie

A.2.1 Fermeture topologique

Nous avons omis dans le chapitre 5 une définition qui n'était pas utile à notre propos. Elle nous sera néanmoins nécessaire pour prouver les théorèmes qui vont suivre :

Définition A.2. La *fermeture* (aussi appelée *adhérence*) de A est l'intersection de tous les fermés contenant A (c'est donc un fermé), notée $\overline{A}$. Les points de la fermeture sont dits *adhérents*.

Remarque A.3. Un ensemble A est fermé si et seulement si $A = \overline{A}$. □

Lemme A.1. $x \in \overline{A}$ si et seulement si tout voisinage de x rencontre A.

Preuve. ($\Rightarrow$) Supposons que $x \in \overline{A}$. Soit O un voisinage de x. Supposons en outre que $O \cap A = \phi$ (par l'absurde). Ainsi $O \subseteq E \setminus A$, donc $O^c \supset (E \setminus A)^c = A$. Or, O^c est un fermé par définition d'un voisinage, donc $O^c \supset \overline{A}$. Du coup, $x \in \overline{A} \subset O^c$, et $x \notin O$, ce qui est en contradiction avec l'hypothèse que O était un voisinage de x.

($\Leftarrow$) Réciproquement, supposons que tout voisinage de x rencontre A. Soit C un ensemble fermé contenant A. Supposons que $x \notin C$ (par l'absurde). $A \subseteq C$ implique que $A^c \supset C^c$; autrement dit, C^c ne rencontre pas A. Par ailleurs, $x \in C^c$. Or, C^c est un ouvert car on a supposé C fermé. Donc, C^c est un voisinage de x qui ne rencontre pas A, en contradiction avec l'hypothèse. Par conséquent, $x \in C$. Puisque C est arbitraire, on a que $x \in \overline{A}$. □

A.2.2 Unions et intersections dénombrables dans un espace métrique

Preuve du théorème 5.3 (page 53)[1]**.** Soit C un fermé de la topologie métrique induite par (E,d). Posons, pour tout $n > 0 : \Omega_n = \bigcup_{y \in C} B(y,1/n)$. Une boule ouverte étant un ouvert, chaque Ω_n est une union arbitraire d'ouverts et est ainsi ouvert. Posons également $X = \bigcap_{n>0} \Omega_n$. Vérifions que X égale C, auquel cas C sera bien une intersection dénombrable des ouverts Ω_n.

Visiblement, $C \subseteq X$ car pour tout $x \in C$, on a que $x \in \Omega_n$ pour tous les n (cf. remarque 5.10). Réciproquement, il faut prouver que $X \subseteq C$. Supposons que $x \in X$, mais que $x \notin C$. En d'autres termes, soit $x \in X \setminus C = X \cap C^c$ et montrons que l'on arrive à une contradiction. X et C^c sont des ouverts (C est fermé), donc x appartient à un ouvert. Par le théorème 5.2, x admet un voisinage entièrement compris dans $X \setminus C$. Puisque les boules forment une base pour ce voisinage, on peut supposer qu'il existe une boule ouverte centrée sur x et de rayon $\varepsilon > 0$ entièrement contenue dans ce voisinage. Posons $N = \lceil 2/\varepsilon \rceil$; de cette manière, $B(x,2/N) \subseteq B(x,\varepsilon) \subseteq X \setminus C$. Cela étant, pour tout $y \in X$, on obtient schématiquement que :

$$\begin{aligned} x \in B(y,1/N) &\Rightarrow d(x,y) < 1/N \\ &\Rightarrow \forall z \in B(y,1/N) : d(x,z) \leq \underbrace{d(x,y)}_{<1/N} + \underbrace{d(y,z)}_{<1/N} < 2/N \\ &\Rightarrow B(y,1/N) \subseteq B(x,2/N) \subseteq X \setminus C \\ &\Rightarrow y \notin C \text{ car } y \in B(y,1/N) \end{aligned}$$

Supposons maintenant que $x \in \Omega_N = \bigcup_{y \in C} B(y,1/N)$. Il existerait alors un $y \in C$ tel que $x \in B(y,1/N)$. Par le raisonnement ci-dessus, ceci impliquerait que $y \notin C$, ce qui est absurde. On en déduit que $x \notin \Omega_N$ et donc $x \notin X$. Il y a contradiction, puisque x a été supposé appartenant à X. On en déduit que l'hypothèse selon laquelle $x \notin C$ est fausse et donc que $X \subseteq C$, ce qui prouve la première partie de l'énoncé.

Maintenant, le complément d'un ouvert est un fermé, que l'on sait être une intersection dénombrable d'ouverts. Par dualité, tout ouvert est une union dénombrable de fermés (pour obtenir une démonstration plus formelle, on peut répéter la démonstration du théorème 5.1). □

A.2.3 Combinaisons booléennes

Les résultats de cette section sont inspirés de [GG90]. Ils exploitent le lien très étroit entre logique et théorie des ensembles. En effet, la disjonction logique « $\vee$ » peut être mise en correspondance avec l'union ensembliste « $\cup$ », la conjonction logique « $\wedge$ » avec l'intersection ensembliste « $\cap$ » et la négation logique « $\neg$ » avec la complémentation ensembliste « ^{c} ».

1. Je remercie Fabien Boniver pour son aide précieuse dans la démonstration de ce théorème.

Définition A.4. Une combinaison booléenne finie $\mathcal{B}(S_k)$ d'ensembles S_k est mise en *forme normale disjonctive* si elle est de la forme :

$$\bigcup_{i=1}^{n} \bigcap_{j=1}^{m_i} L_{ij}$$

où L_{ij} sont des « littéraux ensemblistes », à savoir un ensemble S_k ou son complément S_k^c. De même, $\mathcal{B}(S_k)$ est en *forme normale conjonctive* si elle est de la forme :

$$\bigcap_{i=1}^{n} \bigcup_{j=1}^{m_i} L_{ij}$$

Lemme A.2. Toute combinaison booléenne finie $X = \mathcal{B}(S_k)$ d'ensembles S_k peut se mettre en forme normale conjonctive (FNC).

Preuve. Pour mettre X en forme normale conjonctive, on procède comme suit :

1. On propage les occurrences de S^c vers l'intérieur grâce aux lois de De Morgan :

$$(A \cap B)^c = A^c \cup B^c$$

$$(A \cup B)^c = A^c \cap B^c$$

2. On élimine les doubles complémentations : $(A^c)^c = A$.
3. On distribue $\cup$ par rapport à $\cap$ pour éliminer tous les $\cap$ de l'intérieur des disjonctions :

$$A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$$
$$(A \cap B) \cup C = (A \cup C) \cap (B \cup C)$$

Par la troisième étape, toutes les intersections ont été propagées à l'extérieur des disjonctions. La seconde étape avait déjà éliminé les complémentations extérieures. Par conséquent, au niveau supérieur, il ne reste plus que des opérations d'intersection. L'associativité de l'intersection permet d'obtenir une conjonction de disjonctions de littéraux ensemblistes : nous avons bien obtenu une FNC. De plus, toutes les opérations que l'on a utilisées préservent l'équivalence des ensembles décrits[2]. Par conséquent, la formule obtenue en FNC est équivalente à X, la formule initiale. □

Lemme A.3. Toute combinaison booléenne finie $X = \mathcal{B}(S_k)$ d'ensembles S_k peut se mettre en forme normale disjonctive (FND).

Preuve. La preuve est quasiment similaire. Seule la troisième étape change. On utilise plutôt les formules :

$$A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$$

$$(A \cup B) \cap C = (A \cap C) \cup (B \cap C)$$

2. Le lecteur peut utiliser les diagrammes de Venne pour s'en convaincre.

qui ont cette fois pour effet de propager les unions vers l'extérieur. On conclut alors en utilisant l'associativité de l'union. □

Preuve du théorème 5.4 (page 53). Soit une combinaison booléenne finie $X \in \mathcal{B}(S_k)$ où S_k est soit un ouvert, soit un fermé pour tous les k. Le premier lemme implique que l'on peut écrire X en FNC :

$$X = \bigcap_{i=1}^{n} \bigcup_{j=1}^{m_i} L_{ij}$$

où L_{ij} est soit un ouvert, soit un fermé. En effet, si $L_{ij} = S_k$, L_{ij} est un ouvert ou un fermé par hypothèse sur les S_k. Si maintenant, $L_{ij} = S_k^c$, on utilise le fait que le complément d'un ouvert est un fermé et que le complément d'un fermé est un ouvert (par définition des fermés). Isolons les fermés des ouverts dans chaque conjonction :

$$X = \bigcap_{i=1}^{n} \left(\left(\bigcup_{j=1}^{f_i} F_{ij} \right) \cup \left(\bigcup_{j=1}^{g_i} G_{ij} \right) \right)$$

où F_{ij} sont des fermés en nombre f_i et où G_{ij} sont des ouverts en nombre g_i. Maintenant, le théorème 5.3 indique que tout fermé F_{ij} est une intersection dénombrable d'ouverts G'_{ijl} :

$$\begin{aligned} X &= \bigcap_{i=1}^{n} \left(\left(\bigcup_{j=1}^{f_i} \bigcap_{l=0}^{\infty} G'_{ijl} \right) \cup \left(\bigcup_{j=1}^{g_i} G_{ij} \right) \right) \\ &= \bigcap_{i=1}^{n} \bigcap_{l=0}^{\infty} \left(\left(\bigcup_{j=1}^{f_i} G'_{ijl} \right) \cup \left(\bigcup_{j=1}^{g_i} G_{ij} \right) \right) \end{aligned}$$

Le terme intérieur de cette dernière formule est une union finie d'ouverts. C'est donc un ouvert G''_{il} :

$$X = \bigcap_{i=1}^{n} \bigcap_{l=0}^{\infty} G''_{il} = \bigcap_{(i,l) \in [1..n] \times \mathbb{N}} G''_{il}$$

L'ensemble $[1..n] \times \mathbb{N}$ est dénombrable. Nous sommes par conséquent en présence d'une intersection dénombrable d'ouverts, et donc $X \in G_\delta$.

Considérons X^c. On sait maintenant que cette formule appartient à G_δ, puisque X^c est elle-même une combinaison booléenne finie des ensembles S_k. Par dualité, X appartient à F_σ, ce qui achève la démonstration. □

A.2.4 Topologie de $\langle \mathbb{R},\, 1, +\,, \leq \rangle$

Il est possible de prouver que la structure $\langle \mathbb{R},\, 1, +\,, \leq \rangle$ admet l'élimination des quantificateurs [Dub95]. En d'autres termes, toute formule φ de cette structure peut se réécrire sous la forme d'une formule φ' ne contenant plus aucune quantification. Nous savons aussi par la section 4.5.3 que chacune de ses formules peut se ramener à une forme dans laquelle les formules atomiques font appel aux seules relations $=(x,y)$, $+(x,y,z)$ et $\leq (x,y)$. Étudions donc les ensembles décrits par ces relations.

Lemme A.4. La relation $\leq (x,y)$ décrit un ensemble fermé dans la topologie naturelle de $\mathbb{R}^2$.

Preuve. Posons $O = \{(x,y) \in \mathbb{R}^2 \mid x > y\}$. Soit un point $(x,y) \in O$. Puisque l'inégalité $x > y$ est stricte, ce point ne peut se situer sur la droite formée par la bissectrice des premier et troisième quadrants. On calcule ensuite la distance d de (x,y) à cette droite [Éti86] : comme le point est distinct de la droite, $d > 0$. On construit alors la boule ouverte $B((x,y), d/2)$. Cette boule est incluse dans O par la définition de la distance d'un point à une droite. Par conséquent, (x,y) admet un voisinage entièrement inclus dans O. On déduit par le théorème 5.2 que O est ouvert et donc que son complémentaire, qui est l'ensemble que l'on cherche à caractériser, est fermé. □

Lemme A.5. La relation $= (x,y)$ décrit un ensemble fermé dans la topologie naturelle de $\mathbb{R}^2$.

Preuve. Dire que $x = y$ revient à dire que $x \leq y \wedge x \geq y$. L'ensemble S décrit est donc l'intersection des ensembles décrits par $\leq (x,y)$ et $\geq (x,y)$. Un raisonnement similaire au lemme précédent permet de prouver que l'ensemble décrit par $\geq (x,y)$ est fermé. S est ainsi l'intersection de deux ensembles fermés et est donc lui-même fermé. □

La caractérisation de la relation $+(x,y,z)$ est moins triviale. Nous allons à cet effet utiliser la notion de suite convergente :

Définition A.5. Une suite (x_n) de points de A *converge vers une limite* x s'il existe un $n_0 \in \mathbb{N}$ tel que $\forall \varepsilon > 0$, $\forall n \geq n_0 : x_n \in B(x, \varepsilon)$.[3]

Lemme A.6. Dans un espace métrique, un ensemble F est fermé si et seulement si la limite de toute suite convergente d'éléments de F est dans F.

Preuve. ($\Rightarrow$) La limite x d'une suite convergente de points de F est un point adhérent à F, car $x \in \overline{F}$ si et seulement si tout voisinage de x rencontre F (lemme A.1), ce qui est visiblement le cas pour un point limite. Or, puisque F est fermé, $F = \overline{F}$. Autrement dit, l'ensemble des points adhérents est l'ensemble lui-même. On en déduit que toute limite de toute suite convergente de F est dans F.

($\Leftarrow$) Si F n'est pas fermé, il existe un point adhérent x de F qui n'appartient pas à F. Comme x est adhérent, dans toute boule $B(x,1/m)$ avec $m > 0$, nous pouvons choisir un élément $x_m \in F$ et la suite (x_m) de F ainsi construite converge vers $x \notin F$, en contradiction avec l'hypothèse. F est donc nécessairement fermé. □

Lemme A.7. La relation $+(x,y,z)$ décrit un ensemble fermé F dans la topologie naturelle de $\mathbb{R}^3$.

3. La notion de convergence ne peut être définie que dans un espace métrique. Placée dans $\mathbb{R}^n$, elle équivaut à la classique notion de limite introduite dans les cours d'analyse [Éti97].

Preuve. Considérons une suite (x_m,y_m,z_m) convergente dans F vers un point (x,y,z). Comme nous sommes dans $\mathbb{R}^3$, nous pouvons exploiter tous les résultats des suites de points dans $\mathbb{R}^n$ (voir [Éti97]). Tout d'abord, cette suite converge vers (x,y,z) si et seulement si les suites (x_m), (y_m) et (z_m) convergent elles-mêmes respectivement vers x, y et z. Donc :

$$\lim_{m\to\infty} x_m = x, \quad \lim_{m\to\infty} y_m = y \quad \text{et} \quad \lim_{m\to\infty} z_m = z$$

La limite d'une somme étant la somme des limites, on a :

$$\lim_{m\to\infty} (x_m + y_m) = x + y$$

Puisque $(x_m,y_m,z_m) \in F$, on a $x_m + y_m = z_m$ pour tout $m > 0$, donc :

$$\lim_{m\to\infty} z_m = \lim_{m\to\infty} (x_m + y_m) = x + y$$

On savait déjà que $\lim_{m\to\infty} z_m = z$. La limite d'une suite étant unique, cela implique que $x + y = z$. Par conséquent, la limite de la suite (x_m,y_m,z_m) est $(x,y,y + z)$, ce qui est un point de F. Le lemme A.6 affirme alors que F est fermé. □

Grâce au lemme suivant, ces caractérisations restent correctes même si les formules qui ont décrit ces ensembles de base comportent plus de deux ou trois variables :

Lemme A.8. Les ouverts et les fermés de $\mathbb{R}^n$ restent des ouverts et des fermés quand on leur adjoint un ensemble de k nouvelles variables.

Preuve. Soit O un ouvert de $\mathbb{R}^n$. Si k nouvelles variables lui sont adjointes, il définit l'ensemble $O' = O \times \mathbb{R}^k$. Soit $x \in O'$. x peut s'écrire (y,z) où $y \in O$ et $z \in \mathbb{R}^k$. Puisque O est un ouvert, y admet autour de lui une boule ouverte entièrement incluse dans O de rayon ε. Soit maintenant la boule ouverte $V = B(x,\varepsilon) \subseteq \mathbb{R}^{n+k}$. Par définition de ε et puisqu'il n'y a aucune contrainte sur les k nouvelles variables, on sait que $V \subseteq O'$. Par conséquent, tout point x de O' admet un voisinage V entièrement inclus dans O', ce qui montre que O' est un ouvert [4].

Maintenant, soit F un fermé de $\mathbb{R}^n$ auquel on adjoint k nouvelles variables. Il définit alors l'ensemble $F' = F \times \mathbb{R}^k$. F est, par définition d'un fermé, le complément d'un ouvert O dans $\mathbb{R}^n$. On a dès lors successivement :

$$\begin{aligned} x = (y,z) \in F' &\Leftrightarrow y \in O^c \wedge z \in \mathbb{R}^k \Leftrightarrow y \in O^c \\ &\Leftrightarrow y \in O^c \vee z \notin \mathbb{R}^k \Leftrightarrow \neg(y \in O \wedge z \in \mathbb{R}^k) \end{aligned}$$

De cette dernière relation, on déduit que $F' = (O \times \mathbb{R}^k)^c$. Par le résultat ci-dessus, $O \times \mathbb{R}^k$ est un ouvert, donc F' est le complémentaire d'un ouvert, ce qui équivaut à dire que F' est un fermé. □

4. Si l'on avait pris soin de définir la *topologie produit*, qui fournit une topologie pour l'espace $E \times F$ où E et F sont deux espaces topologiques, ce théorème serait en fait une définition [RS97].

L'élimination des quantificateurs de la structure $\langle \mathbb{R},\, 1,\, +\,,\, \leq \rangle$ fait que toute formule φ se réduit à une formule φ' sans quantification, qui est dès lors une combinaison booléenne finie d'ensembles décrits par les relations $\leq (x,y)$, $=(x,y)$ et $+(x,y,z)$. Les lemmes précédents impliquent immédiatement le résultat :

Théorème A.9. Les formules de la structure $\langle \mathbb{R},\, 1, +\,, \leq \rangle$ décrivent des ensembles de la classe de Borel $\mathcal{B}(F) = \mathcal{B}(G)$. Par la figure 5.1 (page 54), ces ensembles sont en particulier dans la classe F_σ.

A.2.5 Ouverts dans la topologie naturelle de Σ^ω

Preuve du théorème 6.1. Considérons d'abord une boule ouverte $B(u,\varepsilon) = \{v \mid d(u,v) < \varepsilon\}$. Soit k le plus grand entier tel que $2^{-k} \geq \varepsilon$ (i.e. $k = \lfloor -\log_2 \varepsilon \rfloor$). Puisque $d(u,v)$ ne peut être que de la forme 2^{-r}, on a : $B(u,\varepsilon) = B(u,\, 2^{-k})$. Dès lors :

$$B(u,\varepsilon) = \{v \mid d(u,v) < 2^{-k}\} = \{v \mid (\exists w \in \Sigma^\omega)(v = u_0 \cdots u_k \cdot w)\} = \{u_0 \dots u_k\} \cdot \Sigma^\omega$$

Comme les boules ouvertes forment la base de toute topologie métrique (définition 5.11), tout ouvert O peut s'écrire sous la forme d'une union arbitraire de boules ouvertes, soit $\{B_i\}_{i\in I}$ cette famille :

$$O = \bigcup_{i\in I} B_i = \bigcup_{i\in I} (\{u_{i,0} \dots u_{i,k_i}\} \cdot \Sigma^\omega) = \left(\bigcup_{i\in I} \{u_{i,0} \dots u_{i,k_i}\} \right) \cdot \Sigma^\omega = X \cdot \Sigma^\omega$$

où on a posé $X = \bigcup_{i\in I} \{u_{i,0} \dots u_{i,k_i}\}$. Clairement, $X \subseteq \Sigma^*$ puisqu'il s'agit d'une union de mots finis. De plus, tous les k_i sont supérieurs ou égaux à 0, donc X ne peut se réduire au seul mot vide, donc $X \subseteq \Sigma^+$. O possède bien la forme requise.

Réciproquement, tout ensemble O de la forme $X \cdot \Sigma^\omega$ peut s'écrire :

$$O = \bigcup_{x\in X} (x \cdot \Sigma^\omega) = \bigcup_{x\in X} B\left(x,\, 2^{-|x|+1}\right)$$

De ce fait, O est une union arbitraire de boules ouvertes, qui sont elles-mêmes des ouverts. Donc O est un ouvert. □

A.2.6 Opérateur limite

Nous avons vu à la section 3.4 que tous les $\omega-$langages ne peuvent pas être acceptés par des automates de Büchi déterministes. Il serait donc intéressant de pouvoir caractériser les langages qui peuvent l'être. Une telle caractérisation nous sera utile plus loin quand nous chercherons à faire le lien entre topologie et $\omega-$automates. Pour ce faire, on introduit un nouvel opérateur, appelé *opérateur limite*, semblable à l'opération L^ω, qui permet de générer un langage infini à partir d'un langage fini. Pour un langage de mots finis $L \subseteq \Sigma^*$, on pose :

$$\overrightarrow{L} = \{w \in \Sigma^\omega \mid w \text{ a une infinité de préfixes dans } L\}$$

Pour rappel, un mot fini u est préfixe d'un mot w (éventuellement infini) si et seulement si il existe un mot v tel que $w = u \cdot v$.

Exemple.

1. $L = (a \cdot b)^*$ décrit les mots qui sont formés d'une répétition de ab. Un mot infini $w \in \overrightarrow{L}$ doit donc posséder une infinité de préfixes qui sont une répétition de ab. Le langage décrit est celui des mots infinis qui sont une répétition infinie de ab, soit $\overrightarrow{L} = (a \cdot b)^\omega$.
2. Si $L = a^* \cdot b$, alors $\overrightarrow{L} = \phi$. En effet, si il existait un mot w dans $\overrightarrow{L}$, il existerait deux préfixes u et v de w appartenant à L tels que $u = a^k \cdot b$ soit lui-même préfixe de $v = a^l \cdot b$, avec $k < l$. Cela est impossible car $u_k = b \neq v_k = a$.
3. Soit $L = \{a,b\}^* \cdot b$. L décrit l'ensemble des mots qui se terminent par b. Donc, pour qu'un mot w appartienne à $\overrightarrow{L}$, il faut qu'il possède un nombre infini de préfixes qui se terminent par b. Ceci implique que w doit posséder un nombre infini de b (éventuellement non consécutifs). Par conséquent, $\overrightarrow{L}$ est le langage des mots contenant un nombre infini de b, c'est-à-dire $\overrightarrow{L} = (a^* \cdot b)^\omega$.

□

Théorème A.10. Soit $A = (Q, \Sigma, \delta, \{s_0\}, F)$ un automate déterministe. On a : $L_\omega(A) = \overrightarrow{L(A)}$. En d'autres termes, le langage accepté par un automate de Büchi déterministe est le même que le langage limite de cet automate vu comme un automate sur mots finis.

Preuve. Soit un mot $w \in L_\omega(A)$. Il existe alors une course $\rho = q_0 \xrightarrow{w_0} q_1 \xrightarrow{w_1} q_2 \dots$ de A sur w telle que $q_0 = s_0$ et telle qu'il existe une suite infinie (n_k) d'éléments pour lesquels $q_{n_k} \in F$. Considérons le mot $u_k = w_0 w_1 \dots w_{n_k}$. La course de A sur ce mot fini u_k se termine dans l'état q_{n_k} qui est par définition accepteur. Donc, $u_k \in L(A)$ avec $k = 0,1,2,\dots$. De plus, chaque mot u_k est préfixe de w puisque l'automate est déterministe et qu'alors, la course est unique. Donc, w possède un nombre infini de préfixes dans $L(A)$, ce qui signifie que $w \in L_\omega(A)$. Finalement, $L_\omega(A) \subseteq \overrightarrow{L(A)}$.

Réciproquement, si $w \in \overrightarrow{L(A)}$, w possède un nombre infini de préfixes de longueur croissante $u_k \in L(A)$. Puisque l'automate est déterministe, deux préfixes distincts partagent le même début de course. Donc, la suite des (u_k) définit une course sur w pour l'automate vu comme un automate de Büchi. De plus, puisque chaque u_k appartient à $L(A)$, le dernier état de la sous-course correspondante est accepteur. Comme les u_k sont en nombre infini, la course passe par un nombre *infini* d'états accepteurs. Finalement, $w \in L_\omega(A)$ et ainsi, $\overrightarrow{L(A)} \subseteq L_\omega(A)$. □

Corollaire A.11. Un $\omega-$langage X peut être accepté par un automate de Büchi déterministe si et seulement si il existe un langage $L \subseteq \Sigma^*$ reconnaissable par un automate sur mots finis et tel que $X = \overrightarrow{L}$.

Preuve. ($\Rightarrow$) Soit un automate de Büchi déterministe A qui accepte X. Le théorème A.10 affirme que $X = L_\omega(A) = \overrightarrow{L(A)}$. On peut donc poser $L = L(A)$ et A est l'automate sur mots finis recherché.

($\Leftarrow$) Réciproquement, on construit un automate sur mots finis A qui reconnaît le langage L (ceci est possible car L est par hypothèse reconnaissable). On déterminise ensuite A [HU79, Wol91]. Puis on regarde A comme un automate de Büchi. Par le théorème A.10, on a $L_\omega(A) = \overrightarrow{L(A)} = \overrightarrow{L} = X$. Donc, X est accepté par A vu comme un automate de Büchi. De plus, A est déterministe. □

A.2.7 Automates de Büchi déterministes

Dans cette section, nous allons exploiter les résultats de la section A.2.6 afin de caractériser les ensembles qui peuvent être acceptés par un automate de Büchi déterministe. Plus précisément, nous allons établir un lien entre la topologie des mots infinis et l'opérateur limite déjà introduit. Pour ce faire, il faut faire appel à la proposition :

Proposition A.12. Un $\omega-$langage X peut s'écrire $\overrightarrow{L}$ avec $L \subseteq \Sigma^*$ si et seulement si X est un langage G_δ.

Preuve.[5] ($\Rightarrow$) On suppose que $X = \overrightarrow{L}$. Posons $L_n = \{v \in L \mid |v| \geq n\}$ l'ensemble des mots de L ayant une longueur supérieure à n. Tout mot infini w de $\overrightarrow{L}$ possède un préfixe dans L de longueur supérieure à n et donc un préfixe dans L_n. De ce fait, le mot w appartient pour tout n à $L_n \cdot \Sigma^\omega$ (qui est un ouvert par le théorème 6.1). Donc, w appartient aussi à l'intersection dénombrable d'ouverts $Y = \bigcap_{n\geq 0} L_n \cdot \Sigma^\omega$. Par conséquent, $\overrightarrow{L} \subseteq Y$. Réciproquement, tout mot de cette intersection Y possède des préfixes dans L de longueur arbitrairement grande et appartient par conséquent au langage $\overrightarrow{L}$. Ceci montre que $X = \overrightarrow{L} = Y$ et donc que tout $\omega-$langage de la forme $\overrightarrow{L}$ est de la classe G_δ.

($\Leftarrow$) Réciproquement, supposons que X s'écrive $\bigcap_{n\geq 0} O_n$ où les O_n sont des ouverts. En posant $V_n = \bigcap_{k=0}^{n} O_k$, on a $X = \bigcap_{n\geq 0} V_n$. Chaque V_n est un ouvert (une intersection finie d'ouverts reste un ouvert) et est donc de la forme $L_n \cdot \Sigma^\omega$ où $L_n \subseteq \Sigma^+$ par le théorème 6.1. On peut même supposer que les langages L_n sont préfixes[6] puisque $L_n \cdot \Sigma^\omega = (L_n \setminus L_n \cdot \Sigma^+) \cdot \Sigma^\omega$: il suffit dès lors d'appliquer la règle $L_n := L_n \setminus L_n \cdot \Sigma^+$ étant donné que ce dernier ensemble est préfixe par construction. De plus, on sait que $V_{k+1} \subseteq V_k$ pour tout $k \in \mathbb{N}$. Posons maintenant $L = \bigcup_{n\geq 0} L_n \cdot \Sigma^n$ et vérifions que $X = \overrightarrow{L}$.

Soit $w \in X$. Puisque $X = \bigcap_{n\geq 0} L_n \cdot \Sigma^\omega = \bigcap_{n\geq 0} L_n \cdot \Sigma^n \cdot \Sigma^\omega$, le mot w possède un préfixe p_n dans chacun des langages $L_n \cdot \Sigma^n$. Or, $|p_n| \geq n$. Par conséquent, w possède une infinité de préfixes dans L et donc $w \in \overrightarrow{L}$. Réciproquement, si $w \in \overrightarrow{L}$, il possède

5. La démonstration qui suit est essentiellement extraite de [Car93] et de [PP01].

6. Un langage est dit *préfixe* s'il ne contient aucun mot qui est préfixe d'un autre de ses mots.

une suite infinie (p_n) de préfixes de longueur croissante dans L. Par définition de L, chaque p_i appartient à un certain langage $L_{n_i} \cdot \Sigma^{n_i}$. Comme on a supposé que L_{n_i} est un langage préfixe, aucun des préfixes précédant p_i ne peut appartenir à $L_{n_i} \cdot \Sigma^{n_i}$: tous les préfixes appartiennent nécessairement à des langages $L_i \cdot \Sigma^i$ distincts. Il s'ensuit que $w \in L_n \cdot \Sigma^n \cdot \Sigma^\omega = V_n$ pour une infinité de valeurs de n. Étant donné que $V_{k+1} \subseteq V_k$ pour tout k, on a nécessairement $w \in \bigcap_{n \geq 0} V_n = X$. □

Cette proposition, mise en commun avec le corollaire A.11 de la section A.2.6, a pour conséquence immédiate le théorème de Landweber énoncé à la page 65.

A.3 Déterminisation d'automates co-Büchi

Proposition A.13. Si on emploie la construction de la section 6.3.3, on a : $L_\omega(A') = L_\omega(A)$.

Preuve. $L_\omega(A') \subseteq L_\omega(A)$: Soient $w \in L_\omega(A')$ et ρ' une course acceptrice de A' sur w qui consiste en une suite infinie (S_n, R_n). Comme ρ' est acceptrice, il existe un indice n' à partir duquel on ne peut plus jamais avoir $R_k = \phi$ (i.e. n' représente l'indice du dernier point de rupture atteint par ρ'). On construit alors une course ρ de A sur w en choisissant $\rho(k) \in S_k$ pour tout $k \leq n'$ et $\rho(k) \in R_k$ pour tout $k > n'$, peu importe le choix. Il s'agit bien d'une course de A sur w par la définition, dans δ', de R' et de S'. Il reste à montrer que ρ est acceptrice. Par définition de R', on ne peut jamais avoir $R_k \cap F \neq \phi$. De ce fait, $\rho(k) \notin F$ pour tous les $k > n'$. Ainsi, $\mathrm{Inf}(\rho) \cap F = \phi$, ce qui prouve que ρ est acceptrice pour A et donc que $w \in L_\omega(A)$.

$L_\omega(A) \subseteq L_\omega(A')$: Soient $w \in L_\omega(A)$ et ρ une course acceptrice de w sur A. Définissons une suite (S_n) telle que S_0 est l'ensemble des états initiaux et que $S_{k+1} = T(S_k, w_k)$ pour tout $k \geq 0$. L'existence de la course ρ implique qu'aucun des ensembles S_k ne peut être vide. Quant à la suite (R_n), on la construit au départ de (S_n) en respectant les règles de δ'. Ce faisant, on a défini une course ρ' de A' sur w. Il reste à prouver que ρ' est acceptrice. Supposons que ce ne soit pas le cas. Il existe alors une suite infinie (r_n) d'indices tels que $R_{n_k} = \phi$. La définition de δ' implique alors qu'il existe au moins un état qui appartient à F dans l'intervalle $\rho(r_i) \ldots \rho(r_{i+1} - 1)$, pour tout $i \in \mathbb{N}$. Par conséquent, ρ contient en son sein une suite infinie d'états accepteurs, ce qui signifie que ρ n'est pas acceptrice : il y a contradiction. Donc, ρ' est nécessairement acceptrice et ainsi $w \in L_\omega(A')$. □

A.4 Automates faibles déterministes

En analysant la figure 6.7, nous avons suggéré que les automates faibles acceptent exactement les langages qui sont à la fois acceptés par un automate de Büchi et un

automate co-Büchi déterministes. La proposition suivante prouve cette affirmation [7] :

Proposition A.14. Soit $L \subseteq \Sigma^\omega$ un $\omega-$langage. L est accepté par un automate déterministe faible si et seulement si il peut être accepté par un automate de Büchi et un automate co-Büchi, tous deux déterministes.

Preuve. ($\Rightarrow$) Nous avons déjà démontré cette implication dans le corollaire 6.12.

($\Leftarrow$) Pour l'autre direction, soient $A_1 = (Q_1, \Sigma, \delta_1, \{s_0^1\}, F_1)$ un automate de Büchi déterministe pour lequel $L_\omega(A_1) = L$ et $A_2 = (Q_2, \Sigma, \delta_2, \{s_0^2\}, F_2)$ un automate co-Büchi déterministe pour lequel $L_\omega(A_2) = L$. Nous les supposerons complétés. Construisons alors l'automate $T = (Q, \Sigma, \{s_0\}, \delta, F)$ comme suit :

- $Q = Q_1 \times Q_2$,
- $s_0 = (s_0^1, s_0^2)$,
- pour tout $q_1 \in Q_1$, $q_2 \in Q_2$ et $\sigma \in \Sigma$, on pose :

$$\delta((q_1, q_2), \sigma) = \{(\delta_1(q_1, \sigma), \delta_2(q_2, \sigma))\}$$

Nous préciserons F plus loin. Visiblement, T est un automate déterministe par la forme de la fonction δ. Regardons maintenant T comme un automate de Büchi $\mathcal{B}$ pour lequel l'ensemble des états accepteurs est $F' = F_1 \times Q_2$. Regardons également T comme un automate co-Büchi $\mathcal{C}$ pour lequel l'ensemble des états accepteurs est $F'' = Q_1 \times F_2$. $\mathcal{B}$ fonctionne de manière identique à A_1 car la condition d'acceptation ne porte que sur la partie héritée de A_1 et car la définition de la fonction de transition n'a pas modifié les transitions possibles pour A_1. De même, $\mathcal{C}$ fonctionne de manière identique à A_2. Par conséquent, $L_\omega(\mathcal{B}) = L_\omega(\mathcal{C}) = L$. Cela implique que pour tout cycle $C \subseteq Q$, C est accepteur pour $\mathcal{B}$ si et seulement il est accepteur pour $\mathcal{C}$ (sinon, une course qui bouclerait dans ce cycle serait acceptrice pour l'un des automates et pas pour l'autre, ce qui contredirait l'égalité des langages). En d'autres termes, $C \cap F' \neq \phi$ si et seulement si $C \cap F'' = \phi$.

Maintenant, soit $S \subseteq Q$ une composante fortement connexe de T et soient C_1 et C_2 deux cycles de cette composante S. Nous pouvons en déduire que $C_1 \cap F' \neq \phi$ si et seulement si $C_2 \cap F' \neq \phi$. Imaginons en effet que $C_2 \cap F' = \phi$, ce qui équivaut par le raisonnement ci-dessus à $C_2 \cap F'' \neq \phi$. Construisons alors le cycle C qui consiste à parcourir une fois le cycle C_1, puis à rejoindre le cycle C_2, à parcourir une fois celui-ci, et enfin à revenir à C_1 (ce qui est possible, puisque nous sommes dans une composante fortement connexe S). Les conditions sur C_1 et C_2 affirment alors respectivement que $C \cap F' \neq \phi$ et que $C \cap F'' \neq \phi$. Ceci est en contradiction avec la relation $C \cap F' \neq \phi \Leftrightarrow C \cap F'' = \phi$. Un raisonnement identique tient pour l'autre sens de l'équivalence.

On en déduit que dans une même composante fortement connexe, soit tous les cycles rencontrent F', soit aucun. Si tous les cycles rencontrent F', tous les états de la composante peuvent devenir accepteurs sans changer le statut accepteur des cycles,

7. La démonstration qui suit reprend de larges extraits de [Löd97].

ni donc le langage accepté. Soient dès lors $Q_1, \ldots, Q_m$ les composantes fortement connexes de T ; on pose $F = \{q \in Q \mid (\exists i)(q \in Q_i \wedge Q_i \cap F' \neq \phi)\}$. L'automate T ainsi défini respecte bien la forme faible déterministe et accepte L. □

Remarque A.6. L'automate co-Büchi ne doit pas nécessairement être déterministe, puisqu'on peut toujours le déterminiser. □

Comme les automates de Büchi déterministes acceptent exactement les $\omega-$langages de la classe G_δ (théorème 6.8) et que les automates co-Büchi acceptent exactement la classe F_σ (corollaire 6.10), on en déduit directement le résultat :

Corollaire A.15. Un $\omega-$langage est de la classe $G_\delta \cap F_\sigma$ si et seulement si il est accepté par un automate faible déterministe.

Remarque A.7. Il est également possible de montrer qu'un $\omega-$langage appartient à la classe F_σ si et seulement si il est accepté par un automate faible *non déterministe*. Cette preuve requiert une construction supplémentaire introduite dans [KV97]. Le lecteur intéressé peut en trouver une démonstration simplifiée dans [Löd97]. □

Annexe B

Minimisation d'automates faibles

Nous présentons dans ce chapitre deux algorithmes qui ont une place centrale dans l'implémentation de la librairie des RVA. Il s'agit de :

1. l'algorithme de Tarjan [Tar72] qui permet d'isoler les composantes fortement connexes d'un graphe (*strongly connected components*, en abrégé, les s.c.c) ;
2. l'algorithme de minimisation des automates faibles déterministes proposé dans le récent article [Löd01].

Les algorithmes proprement dits ont été mis en page par l'outil de programmation littéraire (*literate programming*) `noweb`, inspiré du langage `web` de Knuth [Knu83, noweb]. Le langage de programmation sous-jacent est le Pascal[1].

B.1 Coloriages

Le titre de cette section peut prêter à sourire... Il s'agit néanmoins d'un passage théorique nécessaire pour comprendre la minimisation des automates faibles. À l'instar des courses sur un mot infini, un coloriage est un autre formalisme permettant de décrire le langage accepté par un $\omega-$automate. Soit donc un automate faible déterministe $A = (Q, \Sigma, \delta, \{s_0\}, F)$.

Définition B.1. Un état $q \in Q$ est dit *de transit* s'il est le seul élément dans une composante fortement connexe et s'il n'est pas relié à lui-même. Un état est dit *récurrent* s'il n'est pas de transit. Plus formellement, q est récurrent si et seulement si il existe un mot $w \in \Sigma^* - \{\varepsilon\}$ tel que $\delta(q, w) = q$.

Le terme « état de transit » se comprend très bien : il s'agit d'un état dans lequel on ne peut jamais séjourner. Quand on y arrive, on est obligé d'en sortir au prochain

1. On a préféré `noweb` à `web` car il permet une inclusion aisée des fichiers générés dans un document LaTeX. La mise en forme du code est réalisée *via* un filtre dû à M. Kostas et spécialement adapté pour l'occasion au Pascal.

symbole lu sur le ruban.

Définition B.2. Une application $c : Q \mapsto \mathbb{N}$ est appelée *coloriage* de A si et seulement si :

- $c(q)$ est pair pour tout $q \in F$ récurrent,
- $c(q)$ est impair pour tout $q \notin F$ récurrent,
- $\forall \sigma \in \Sigma$, $\forall p,q \in Q : q \in \delta(p,\sigma) \Rightarrow c(p) \leq c(q)$.

On peut concevoir un coloriage comme l'attribution d'un numéro à chacun des états de l'automate. La troisième condition impose que tous les éléments d'une même composante fortement connexe soient coloriés de la même façon. Une composante acceptrice est coloriée par un nombre pair, une composante non acceptrice par un nombre impair et les états de transit peuvent être coloriés de manière indifférente (tant que leur coloriage respecte la troisième condition).

D'un point de vue intuitif, un coloriage définit un ordre partiel sur les composantes de l'automate : les composantes profondes ont un numéro (une couleur) supérieur aux composantes qui sont enfouies moins profondément dans le graphe des composantes fortement connexes. Ce faisant, on procède indirectement à une *mise en ordre* du graphe de l'automate [Rou93].

Remarque B.3. Plusieurs composantes connexes situées sur un même chemin dans le graphe des composantes peuvent avoir un coloriage identique, car l'égalité de la troisième condition n'est pas stricte. □

En fait, les coloriages permettent de définir sous une autre forme la condition d'acceptation des automates faibles. En effet, prenons la liberté d'écrire $c(\rho) = c(\rho(0))c(\rho(1))c(\rho(2))\cdots$ pour une course ρ de A. $c(\rho)$ est la suite des couleurs des états qu'emprunte la course ρ. Par la troisième condition, la suite $c(\rho)$ est nécessairement croissante. Par ailleurs, la définition des automates faibles implique qu'une course finira toujours par boucler dans une composante Q_i qui est soit acceptrice, soit non acceptrice. De ce fait, la couleur extrême de $c(\rho)$ sera respectivement paire ou impaire. Par conséquent, une course est acceptrice si et seulement si la couleur maximale reprise dans son coloriage est paire. Pour formaliser ce constat, on définit l'opération « occurrence » sur les courses, qui est très semblable à l'opération $\mathrm{Inf}(\rho)$:

$$\mathrm{Occ}(\rho) = \{p \in Q \mid (\exists i \in \mathbb{N})\rho(i) = p\}$$

Suite à cette définition, une course ρ d'un automate faible A est acceptrice si et seulement si $\max(\mathrm{Occ}(c(\rho)))$ est pair [2].

Nous allons maintenant poser une série de définitions relatives aux coloriages :

Définition B.4.

1. Un coloriage c est un $k-$coloriage s'il existe $k \in \mathbb{N}$ tel que pour tout $q \in Q$,

2. Il existe même une classe d'automates, les *automates à parité*, dont la description du langage accepté est basée exclusivement sur les coloriages [Mos84, Löd99].

on a : $c(q) \leq k$. En d'autres termes, toutes les couleurs des s.c.c. doivent être inférieures à une couleur k donnée.

2. Un coloriage c est $k-$maximal si c'est un $k-$coloriage tel que pour tout autre $k-$coloriage e et pour tout état $q \in Q$, on a : $e(q) \leq c(q)$.

Remarque B.5. Un $k-$coloriage c est aussi un $k'-$coloriage si $k \leq k'$. En effet, pour tout $q \in Q$, on a bien : $c(q) \leq k \leq k'$. □

B.2 Forme normale

Les coloriages vont nous permettre de définir la forme normale des automates faibles déterministes. Posons :

$$F_c = \{q \in Q \mid c(q) \text{ est pair}\}$$

Définition B.6. L'automate faible déterministe A est dit *en forme normale* si et seulement si $F = F_c$ pour un certain coloriage $k-$maximal c de A, où k est un naturel pair.

Théorème B.1. Si une forme normale existe pour A, elle est unique[3].

Preuve. Supposons que la forme normale ne soit pas unique, c'est-à-dire qu'il existe deux coloriages c et c' de A tels que $F_c \neq F_{c'}$, avec c $k-$maximal, c' $k'-$maximal, k et k' pairs. On peut supposer sans perte de généralité que $k \leq k'$ (sinon, on intervertit c et c'). Posons alors $n = k' - k$; puisque k et k' sont pairs, n l'est aussi. Définissons alors une application $e : Q \mapsto \mathbb{N} : q \mapsto c(q) + n$. Notons que $c(q) + n \in \mathbb{N}$ car $n \geq 0$. e est un $k'-$coloriage :

1. Soit $q \in F$ récurrent. $c(q)$ est pair car c'est un $k-$coloriage. Donc, $e(q) = c(q) + n$ est également pair car n est lui-même pair. De même, si $q \notin F$ et est récurrent, $c(q)$ est impair, ainsi que $e(q)$.
2. Soient $\sigma \in \Sigma$ et p, $q \in Q$ tels que $q \in \delta(p,\sigma)$. c étant un coloriage, on a $c(p) \leq c(q)$. D'où : $e(p) = c(p) + n \leq c(q) + n = e(q)$.
3. Par ailleurs, $c(q) \leq k$ puisque c est un $k-$coloriage, et donc $e(q) = c(q) + n \leq k + n = k + (k' - k) = k'$.

Puisque c' est $k'-$maximal, on en déduit finalement que $e(q) \leq c'(q) \Leftrightarrow c'(q) - n \geq c(q)$. Définissons alors une nouvelle application $d : Q \mapsto \mathbb{N} : d(q) \mapsto c'(q) - n$. On vient de voir que $c'(q) - n \geq c(q) \geq 0$, on a donc bien $d(q) \in \mathbb{N}$. d est un $k-$coloriage :

1. Soit $q \in F$ récurrent. $c'(q)$ est pair, donc $d(q)$ aussi car n est pair. Le même raisonnement tient pour tout $q \notin F$ récurrent, auquel cas $d(q)$ est impair.
2. Si $q \in \delta(p,\sigma)$, $c'(p) \leq c'(q)$, donc $d(p) = c'(p) - n \leq c'(q) - n = d(q)$.

3. L'idée générale de cette démonstration m'a été suggérée par Christof Löding.

3. c' étant $k'-$maximal, on a $c'(q) \leq k'$. Ainsi, $d(q) = c'(q) - n \leq k' - n = k' - (k' - k) = k$. Donc, $d(q) \leq k$.

Nous sommes aussi en mesure de prouver que d est $k-$maximal. Puisque c est $k-$maximal, il suffit de prouver que $d(q) \geq c(q)$ pour tout $q \in F$, et le résultat sera prouvé par transitivité. Or, nous savons justement que $d(q) = c'(q) - n \geq c(q)$ par l'existence de e.

Maintenant, puisque n est pair, on a que $d(q)$ est pair si et seulement si $c'(q)$ est pair. On conclut que $F_d = F_{c'}$. Nous avons supposé que $F_{c'} \neq F_c$, donc $F_d \neq F_c$. Cela signifie qu'il existe un état p tel que soit $d(p)$ est pair et $c(q)$ impair, soit $d(p)$ est impair et $c(q)$ pair. Dans les deux cas, on a : $d(p) \neq c(p)$. Nous avons alors deux possibilités :

1. $d(p) > c(p)$, ce qui contredit la $k-$maximalité de c,
2. $d(p) < c(p) \Leftrightarrow c(p) > d(p)$, ce qui contredit la $k-$maximalité de d.

Dans les deux cas, nous arrivons à une contradiction : l'hypothèse que nous avions faite est erronée. Par conséquent, la forme normale est unique. □

Remarque B.7. Il faut souligner le fait que la forme normale est unique grâce à la parité de k. Si nous avions laissé la possibilité que k soit impair, nous n'aurions pas pu mener à bien la démonstration car n n'aurait pas nécessairement été pair. □

B.3 Réduction à la forme normale

La réduction à la forme normale pour un automate faible déterministe nécessite donc le calcul d'un coloriage $k-$maximal de A, où k est un naturel pair quelconque. Dans cette section, nous présentons une méthode pour réaliser ce calcul. Celle-ci repose sur l'algorithme de Tarjan de découverte des s.c.c. Voici le schéma de l'entièreté du programme :

118 ⟨ * 118⟩≡
```
program normalisation;

const ⟨Constantes globales 119.2⟩
type ⟨Structures de données 119.3⟩
var ⟨Variables globales 120.1⟩

⟨Routines annexes 124.2⟩
⟨Algorithme de Tarjan 120.2⟩
⟨Algorithme de normalisation 129⟩

begin
   ⟨Programme de test 119.1⟩
end.
```
Fragment racine.

Le but de ce chapitre n'est pas de présenter un programme complet, mais seulement d'exposer les algorithmes. Par conséquent, nous ne précisons pas quel est le programme de test :

119.1 ⟨*Programme de test* 119.1⟩≡
```
{ L'utilisateur peut tester ici la librairie }
```
Code utilisé dans le fragment 118.

B.3.1 Représentation des graphes

Il nous faut spécifier sur quel type de données les algorithmes vont fonctionner. Par définition d'un coloriage, l'étiquetage des transitions de l'automate A ne nous intéresse pas : on peut se contenter de travailler sur des graphes dirigés et non sur des automates en toute généralité. En Pascal, il faut préciser la taille des tableaux. C'est pourquoi nous introduisons deux constantes qui représentent respectivement le nombre maximal de nœuds d'un graphe et le nombre maximal de successeurs d'un nœud :

119.2 ⟨*Constantes globales* 119.2⟩≡
```
Nnoeuds = 1000;
Nsucc = 500;
```
Code utilisé dans le fragment 118.

On définit alors un nœud du graphe, en prenant soin de laisser de la place pour quelques informations additionnelles qui seront utiles dans l'algorithme de normalisation :

119.3 ⟨*Structures de données* 119.3⟩≡
```
TNoeudID = 0..Nnoeuds-1;
TNoeud =
  record
    nbsucc : integer;   { Nombre de successeurs du nœud }
    succ : array[0..Nsucc-1] of TNoeudID;
    ⟨Données supplémentaires pour la normalisation 125.1⟩
  end;
```
Voir aussi dans les fragments 119.4 et 124.1.
Code utilisé dans le fragment 118.

Un graphe est alors défini ainsi :

119.4 ⟨*Structures de données* 119.3⟩+≡
```
TGraph =
  record
    n : integer;   { Nombre de nœuds du graphe }
    s : array[TNoeudID] of TNoeud;
  end;
```
Code utilisé dans le fragment 118.

B.3.2 Algorithme de Tarjan

Il existe une très étroite relation entre les s.c.c. et les coloriages d'un graphe puisque tous les éléments d'une s.c.c. reçoivent la même couleur. Le principe général de l'algorithme de normalisation est dès lors de reconnaître les s.c.c. du graphe, puis de leur attribuer une couleur. Cette recherche des s.c.c. peut être réalisée par l'algorithme de Tarjan [Tar72], déjà évoqué à la section 3.3.6 et dont nous allons présenter une version simplifiée due à [Nuu95]. L'exposé restera très intuitif. Le lecteur pourra ce référer à cette thèse pour une preuve formelle.

Dans sa version classique, l'algorithme de Tarjan attribue un même numéro unique à tous les nœuds situés dans une même s.c.c. :

120.1 ⟨*Variables globales* 120.1⟩≡
```
sccnum : array[TNoeudID] of integer;
```
Code utilisé dans le fragment 118.

Tant qu'on a pas trouvé la s.c.c. dans laquelle un nœud v doit se trouver, on aura par convention sccnum[v]= -1.

Recherche en profondeur d'abord

L'algorithme de Tarjan repose sur une recherche en profondeur d'abord classique. Cette recherche est appliquée successivement à tous les nœuds qui n'ont pas encore été traités :

120.2 ⟨*Algorithme de Tarjan* 120.2⟩≡
```
procedure tarjan(var g : TGraph);
var ⟨Variables de Tarjan 121.1⟩

   procedure visit(v : TNoeudID);
   var ⟨Variables de Visit 121.4⟩
   begin
      ⟨Prétraitement d'un nœud 121.5⟩
      ⟨Étape récursive 121.6⟩
      ⟨Détection d'une nouvelle s.c.c. 122.6⟩
   end;

begin
   ⟨Initialiser la recherche 121.2⟩
   ⟨Lancer l'exploration 121.3⟩
end;
```
Code utilisé dans le fragment 118.

Nous avons besoin comme variables d'un pointeur qui nous permet de passer en revue tous les nœuds du graphe, d'une structure de données nous permettant de mémoriser les nœuds déjà examinés et d'un compteur pour numéroter les s.c.c. :

121.1 ⟨*Variables de Tarjan* 121.1⟩≡
```
node : TNoeudID;
seen : TNoeudSet;
scccount : integer;
```
Voir aussi dans les fragments 121–23 et 125.2.
Code utilisé dans le fragment 120.2.

Nous pouvons maintenant préciser le fonctionnement global de l'algorithme :

121.2 ⟨*Initialiser la recherche* 121.2⟩≡
```
newset(seen);
scccount:= 0;
for node:= 0 to g.n-1 do
    sccnum[node]:= -1;
```
Voir aussi dans les fragments 122.3, 123.2, et 125.3.
Code utilisé dans le fragment 120.2.

121.3 ⟨*Lancer l'exploration* 121.3⟩≡
```
for node:= 0 to g.n-1 do
    if not appartient(node, seen) then
        visit(node);
```
Code utilisé dans le fragment 120.2.

121.4 ⟨*Variables de Visit* 121.4⟩≡
```
w, succ : TNoeudID;
i : integer;
```
Voir aussi dans les fragments 127.1 et 128.2.
Code utilisé dans le fragment 120.2.

121.5 ⟨*Prétraitement d'un nœud* 121.5⟩≡
```
insert(v, seen);
```
Voir aussi dans les fragments 122 et 123.3.
Code utilisé dans le fragment 120.2.

121.6 ⟨*Étape récursive* 121.6⟩≡
```
for i:= 0 to g.s[v].nbsucc-1 do
    begin
        succ:= g.s[v].succ[i];
        if not appartient(succ, seen) then
            visit(succ);
        ⟨Mise à jour d'informations 122.5⟩
    end;
```
Code utilisé dans le fragment 120.2.

Racine d'une composante fortement connexe

Il est possible de prouver que tous les nœuds appartenant à une même s.c.c. possèdent nécessairement un ancêtre commun u dans le processus de recherche en profondeur, u appartenant à cette même s.c.c. [Tar72]. Pour une s.c.c. donnée, le plus vieil ancêtre dans cette composante est appelé « racine » de la s.c.c. Elle est unique. L'algorithme de Tarjan se réduit ainsi à la recherche des racines des s.c.c. Pour ce faire, on crée une variable root qui retient une racine candidate pour chaque nœud en cours de visite :

121.7 ⟨*Variables de Tarjan* 121.1⟩+≡
```
root : array[TNoeudID] of TNoeudID;
```
Code utilisé dans le fragment 120.2.

Initialement, le nœud lui-même est la racine candidate. Cette approximation va être affinée au cours de l'algorithme :

122.1 ⟨*Prétraitement d'un nœud* 121.5⟩+≡
```
root[v]:= v;
```
Code utilisé dans le fragment 120.2.

Pour appliquer le critère, il faut pouvoir déterminer si un nœud est ancêtre d'un autre dans le processus d'exploration. C'est pourquoi il faut mémoriser l'ordre dans lequel les nœuds sont rencontrés au cours de la recherche en profondeur :

122.2 ⟨*Variables de Tarjan* 121.1⟩+≡
```
dfsnum : array[TNoeudID] of TNoeudID;
dfscount : integer;
```
Code utilisé dans le fragment 120.2.

122.3 ⟨*Initialiser la recherche* 121.2⟩+≡
```
dfscount:= 0;
```
Code utilisé dans le fragment 120.2.

122.4 ⟨*Prétraitement d'un nœud* 121.5⟩+≡
```
dfsnum[v]:= dfscount;
dfscount:= dfscount+1;
```
Code utilisé dans le fragment 120.2.

Quand un successeur succ de v a été traité, il est possible que de nouvelles racines candidates aient été découvertes. La définition d'une racine nous apprend qu'une nouvelle racine pour v a été découverte si et seulement si la racine candidate de succ est devenue un ancêtre de la racine candidate de v. Le principe de la récursivité exige de propager cette découverte au nœud v, à condition que succ n'ait pas entre-temps été placé dans une s.c.c. (auquel cas v et succ ne figurent pas dans la même s.c.c.) :

122.5 ⟨*Mise à jour d'informations* 122.5⟩≡
```
if sccnum[succ]= -1 then
  if dfsnum[root[succ]] < dfsnum[root[v]] then
    root[v]:= root[succ];
```
Code utilisé dans le fragment 121.6.

Création d'une nouvelle composante

Si au terme du traitement du nœud v, la racine candidate se confond avec v, nous avons détecté une racine pour la s.c.c. à laquelle v appartient :

122.6 ⟨*Détection d'une nouvelle s.c.c.* 122.6⟩≡
```
if root[v]=v then
  begin
    ⟨Nouvelle s.c.c. de racine v détectée 123.4⟩
  end;
```
Code utilisé dans le fragment 120.2.

Quand une nouvelle racine v a été détectée, il faut pouvoir retrouver tous les nœuds qui appartiennent à la s.c.c. dont v est la racine. Pour mener à bien cette recherche, il faut se rappeler que l'on effectue une recherche en profondeur d'abord. De ce fait, les composantes fortement connexes les plus internes sont détectées en premier lieu dans le processus d'exploration. Ainsi, quand une s.c.c. est découverte, ses éléments sont ceux qui ont été atteints en dernier lieu dans l'exploration sans avoir été affectés préalablement à une s.c.c.

De ce fait, en parallèle avec l'exploration des nœuds, l'algorithme de Tarjan mémorise sur une pile auxiliaire les nœuds qui n'ont pas encore été affectés à une s.c.c. et ce, dans l'ordre dans lequel ils sont rencontrés :

123.1 ⟨*Variables de Tarjan* 121.1⟩+≡
```
stack : TStack;
```
Code utilisé dans le fragment 120.2.

123.2 ⟨*Initialiser la recherche* 121.2⟩+≡
```
newstack(stack);
```
Code utilisé dans le fragment 120.2.

123.3 ⟨*Prétraitement d'un nœud* 121.5⟩+≡
```
push(v, stack);
```
Code utilisé dans le fragment 120.2.

Quand une s.c.c. est découverte, les nœuds qui la composent sont au sommet de la pile. On n'a donc plus qu'à les retirer de celle-ci jusqu'à rencontrer le nœud v :

123.4 ⟨*Nouvelle s.c.c. de racine* v *détectée* 123.4⟩≡
```
⟨Initialiser des informations additionnelles 127.2⟩
repeat
    w:= pop(stack);
    sccnum[w]:= scccount;
    ⟨Recueil d'informations additionnelles 127.3⟩
until w=v;
scccount:= scccount+1;
⟨Traitement additionnel sur la s.c.c. 126.2⟩
```
Code utilisé dans le fragment 122.6.

Les fragments correspondant au traitement additionnel sur la s.c.c. sont introduits pour l'algorithme de normalisation. La complexité globale de l'algorithme de Tarjan est $\mathcal{O}(\max(|V|,|E|))$, où $|V|$ est le nombre de nœuds du graphe et $|E|$ est le nombre de transitions dans le graphe.

Structures de données annexes

Nous définissons ici toutes les structures de données qui ont été utilisées :

124.1 ⟨*Structures de données* 119.3⟩+≡

```
TNoeudSet = array[TNoeudID] of boolean;
TStack =
  record
    top : integer;
    pile : array[TNoeudID] of TNoeudID;
  end;
```

Code utilisé dans le fragment 118.

124.2 ⟨*Routines annexes* 124.2⟩≡

```
procedure newset(var s : TNoeudSet);
var i : integer;
begin
  for i:= 0 to Nnoeuds-1 do
    s[i]:= false
end;

function appartient(n : TNoeudID ;
  s : TNoeudSet) : boolean;
begin
  appartient:= s[n]
end;

procedure insert(n : TNoeudID ;
  var s : TNoeudSet);
begin
  s[n]:= true
end;

procedure newstack(var s : TStack);
begin
  s.top:= 0
end;

procedure push(n : TNoeudID ;
  var s : TStack);
begin
  s.pile[s.top]:= n;
  s.top:= s.top +1
end;

function pop(var s : TStack) : TNoeudID;
begin
  s.top:= s.top-1;
  pop:= s.pile[s.top]
end;

function isempty(s : TStack) : boolean;
begin
  isempty:= s.top=0
end;
```

Code utilisé dans le fragment 118.

B.3.3 Algorithme de coloriage

Informations additionnelles

Nous voulons maintenant pouvoir calculer un coloriage $k-$maximal pour un automate faible représenté sous la forme d'un graphe. Pour ce faire, il faut de toute façon connaître le statut accepteur ou non accepteur de chaque état. De plus, il faut disposer d'une zone mémoire dans laquelle stocker le coloriage des états :

125.1 ⟨*Données supplémentaires pour la normalisation* 125.1⟩≡
```
accept : boolean;
color : integer;
```
Code utilisé dans le fragment 119.3.

Choix de k

La première question à se poser est le choix de k. Dans le pire des cas, toutes les s.c.c. possèdent un seul état et voient une couleur distincte leur être attribuée. Il faut donc prévoir au moins n couleurs, où n est le nombre d'états de l'automate A. Puisqu'en outre, k doit être pair, on fixe k au plus petit entier pair supérieur ou égal à $n-1$.

Nous l'avons déjà mentionné à la section B.3.2 : tous les états d'une même s.c.c. doivent être coloriés de la même façon. C'est pourquoi l'algorithme de coloriage va s'intégrer directement au sein de l'algorithme de Tarjan. Le calcul de k peut dès lors se greffer dans l'initialisation de l'algorithme de Tarjan :

125.2 ⟨*Variables de Tarjan* 121.1⟩+≡
```
k : integer;
```
Code utilisé dans le fragment 120.2.

125.3 ⟨*Initialiser la recherche* 121.2⟩+≡
```
if (g.n mod 2) = 0
  then k:= g.n
  else k:= g.n-1;
```
Code utilisé dans le fragment 120.2.

Principe du coloriage

Pour obtenir un coloriage valide c, il faut nécessairement attribuer aux états récurrents une couleur paire s'ils sont accepteurs et une couleur impaire sinon. La définition des coloriages n'impose rien de tel sur les états de transit. Tous les états doivent néanmoins satisfaire la contrainte selon laquelle leur couleur doit être inférieure ou égale à la couleur de chacun de ses successeurs. Autrement dit, pour tout état v, à supposer que la couleur de tous ses successeurs ait été fixée, il faut toujours avoir :

$$c(\mathrm{v}) \leq l \ =_{def} \ \min\{c(\mathrm{succ}) \mid \mathrm{succ\ est\ un\ successeur\ de\ v}\}$$

Notons que l'on aura toujours $l \leq k$ puisque nous cherchons à définir un $k-$coloriage. Finalement, pour générer un coloriage maximal, il faut que chaque couleur attribuée soit la plus grande parmi tous les choix admissibles.

La politique de coloriage se base dès lors sur l'alternative suivante :

1. L'état fait partie d'une s.c.c. qui n'admet aucune s.c.c. pour successeur. Pour respecter la maximalité du coloriage, il faudra lui donner la valeur k ou $k-1$:
   (a) si l'état est de transit, il hérite de la couleur $k-1$, ce qui le rendra non accepteur : un tel état n'admet en effet aucune course acceptrice, il n'y a donc aucune raison de le laisser accepteur ;

(b) si l'état est récurrent, il admet des courses et on le colorie par k ou $k-1$ selon que l'état est accepteur ou non accepteur.

2. L'état fait partie d'une s.c.c. qui n'est pas une feuille dans le graphe des s.c.c. Pour respecter la maximalité du coloriage, il faudra lui donner la valeur l ou $l-1$:

   (a) si l'état est de transit, le choix est indifférent : le principe de maximalité veut dès lors qu'il hérite de la couleur l ;

   (b) si l'état est récurrent, on applique la règle suivante :

l pair ?	État accepteur ?	Couleur attribuée
Oui	Oui	l
Oui	Non	$l-1$
Non	Oui	$l-1$
Non	Non	l

En accord avec ce que nous venons de dire, nous pouvons calculer ainsi la couleur de chaque élément de la s.c.c. :

126.1 ⟨*Stockage dans* i *de la couleur de la s.c.c.* 126.1⟩≡

```
if leaf then begin
      if not transient and accept
         then i:= k
         else i:= k-1

   end else begin
      if transient
         then i:= l
         else if (l mod 2 = 0) xor accept
            then i:= l-1
            else i:= l
end;
```

Code utilisé dans le fragment 126.2.

Le traitement additionnel à réaliser consécutivement à la découverte d'une s.c.c. prend alors la forme suivante :

126.2 ⟨*Traitement additionnel sur la s.c.c.* 126.2⟩≡

```
⟨Stockage dans i de la couleur de la s.c.c. 126.1⟩
⟨Coloriage par i de tous les états de la s.c.c. 128.5⟩
```

Code utilisé dans le fragment 123.4.

Définition et calcul des variables auxiliaires

Dans le calcul de i, nous avons introduit les variables suivantes :

127.1 ⟨ *Variables de Visit* 121.4⟩+≡
```
l : integer;
transient : boolean;
accept : boolean;
leaf : boolean;
```
Code utilisé dans le fragment 120.2.

127.2 ⟨*Initialiser des informations additionnelles* 127.2⟩≡
```
l:= k;
transient:= true;
accept:= false;
leaf:= true;
```
Voir aussi dans le fragment 128.4.
Code utilisé dans le fragment 123.4.

Leur valeur n'est connue que quand on a parcouru une première fois la s.c.c. nouvellement détectée. Ainsi, en même temps que l'on parcourt la pile pour en extraire les éléments de la s.c.c., on met à jour ces variables comme suit :

127.3 ⟨*Recueil d'informations additionnelles* 127.3⟩≡
```
if g.s[w].accept then accept:= true;
if g.s[w].nbsucc > 0 then
    for i:= 0 to g.s[w].nbsucc-1 do
        begin
            succ:= g.s[w].succ[i];
            ⟨Mise à jour de transient, l et leaf 127.4⟩
        end;
```
Voir aussi dans le fragment 128.3.
Code utilisé dans le fragment 123.4.

Si succ désigne un élément de la s.c.c. qui vient d'être découverte, il faut mettre transient à false. Sinon, la s.c.c. détectée ne peut pas être une feuille dans l'arbre acyclique des composantes fortement connexes : on peut mettre leaf à false. De plus, la valeur de l est susceptible d'être modifiée dans ce second cas :

127.4 ⟨*Mise à jour de* transient, l *et* leaf 127.4⟩≡
```
if ⟨succ est-il dans la s.c.c. courante ? 127.5⟩
    then transient:= false
    else begin
        leaf:= false;
        ⟨Mise à jour de l 128.1⟩
    end;
```
Code utilisé dans le fragment 127.3.

Nous avons déjà constaté que les s.c.c. les plus internes sont détectées en premier lieu. Par conséquent, on a déjà attribué un numéro appartenant à l'intervalle $[0,scccount-1]$ à tous les états des s.c.c. plus profondes que la s.c.c. courante. On peut dès lors écrire :

127.5 ⟨succ *est-il dans la s.c.c. courante ?* 127.5⟩≡
```
(sccnum[succ]= -1) or (sccnum[succ]=scccount)
```
Code utilisé dans le fragment 127.4.

Par le même argument, si succ n'est pas dans la s.c.c. qui vient d'être découverte, on sait qu'on lui a déjà attribué une couleur. La mise à jour de l est par conséquent directe :

128.1 ⟨*Mise à jour de* l 128.1⟩≡
```
if g.s[succ].color<l then
  l:= g.s[succ].color
```
Code utilisé dans le fragment 127.4.

Coloriage de la composante

Une fois que l'on a stocké dans i la couleur de la s.c.c. nouvellement détectée, il faut en colorier tous les états. On se heurte ici à un problème, car on a déjà ôté ces états de stack. Il faut donc les mémoriser dans une zone mémoire annexe au fur et à mesure qu'on les dépile. Une pile auxiliaire fait parfaitement l'affaire :

128.2 ⟨*Variables de Visit* 121.4⟩+≡
```
tmp : TStack;
```
Code utilisé dans le fragment 120.2.

128.3 ⟨*Recueil d'informations additionnelles* 127.3⟩+≡
```
push(w, tmp);
```
Code utilisé dans le fragment 123.4.

128.4 ⟨*Initialiser des informations additionnelles* 127.2⟩+≡
```
newstack(tmp);
```
Code utilisé dans le fragment 123.4.

Le coloriage consiste simplement à extraire tous les éléments de la pile auxiliaire, puis à les colorier avec la couleur i :

128.5 ⟨*Coloriage par* i *de tous les états de la s.c.c.* 128.5⟩≡
```
while not isempty(tmp) do
  begin
    w:= pop(tmp);
    g.s[w].color:= i
  end;
```
Code utilisé dans le fragment 126.2.

Remarque B.8. En pratique, au lieu de créer une pile temporaire, on réalise un premier parcours de la s.c.c. sans dépiler stack, puis on réalise la seconde passe en dépilant. □

B.3.4 Algorithme de normalisation

Nous savons maintenant comment colorier un automate faible A. Pour convertir A dans sa forme normale, il reste à appliquer la règle $F := F_c$. En d'autres termes, il suffit de marquer comme accepteurs les états dont la couleur est paire :

129 ⟨*Algorithme de normalisation* 129⟩≡
```
procedure normalize(var A : TGraph);
var node : TNoeudID;
begin
   tarjan(A);
   for node:= 0 to A.n-1 do
      A.s[node].accept:= (A.s[node].color mod 2) = 0
end;
```
Code utilisé dans le fragment 118.

B.4 Procédure de minimisation

Au final, la procédure de minimisation pour un automate faible déterministe A consiste à :

1. Calculer un coloriage $k-$maximal c sur la représentation sagittale de A,
2. Remplacer l'ensemble des états accepteurs F par F_c,
3. Minimiser l'automate par un algorithme traditionnel de minimisation des automates sur mots finis.

Théorème B.2. [Löd01] Pour un automate faible déterministe A avec n états, on peut calculer un automate faible déterministe équivalent avec un nombre minimal d'états en un temps $\mathcal{O}(n \cdot \log n)$, en suivant la procédure décrite ci-dessus.

Preuve. La preuve d'exactitude de ce théorème fait appel à la notion de *congruence* [MS97] qui dépasse le cadre de ce mémoire. La raison intuitive pour laquelle on peut utiliser cette technique de coloriage est la suivante[4] : « Si un état de transit est équivalent à un de ses successeurs, alors ce doit être un de ceux à partir desquels il y a le plus grand nombre d'alternances entre états récurrents accepteurs et états récurrents non accepteurs ». Pour de plus amples détails, le lecteur intéressé peut se référer à [Löd01].

Nous allons néanmoins étudier la complexité de l'algorithme. L'étape (1) est réalisable en un temps $\mathcal{O}(n)$ par l'algorithme que nous venons de voir. En effet, $|V| = n$ et le déterminisme de A impose que le nombre de transitions $|E|$ soit borné par $n \cdot |\Sigma|$. Par conséquent, $\mathcal{O}(\max(|V|,|E|)) = \mathcal{O}(\max(n,n \cdot |\Sigma|)) = \mathcal{O}(n)$ puisque $|\Sigma|$ est une constante. L'étape (2) consiste en un simple parcours des états de A et est donc réalisable en un temps $\mathcal{O}(n)$. L'étape (3) prend un temps $\mathcal{O}(n \cdot \log n)$ [Hop71]. On voit que la complexité réside principalement dans l'étape (3) : l'algorithme est $\mathcal{O}(n \cdot \log n)$. □

Remarque B.9. L'algorithme de normalisation que nous avons développé ici se distingue de celui présenté dans [Löd01] car l'algorithme de coloriage a directement été intégré dans l'algorithme de Tarjan. □

4. Je cite ici un extrait d'un échange avec Christof Löding.

B.5 Référence

B.5.1 Liste des fragments

⟨ * 118⟩
⟨succ *est-il dans la s.c.c. courante ?* 127.5⟩
⟨*Étape récursive* 121.6⟩
⟨*Algorithme de normalisation* 129⟩
⟨*Algorithme de Tarjan* 120.2⟩
⟨*Coloriage par* i *de tous les états de la s.c.c.* 128.5⟩
⟨*Constantes globales* 119.2⟩
⟨*Données supplémentaires pour la normalisation* 125.1⟩
⟨*Détection d'une nouvelle s.c.c.* 122.6⟩
⟨*Initialiser des informations additionnelles* 127.2⟩
⟨*Initialiser la recherche* 121.2⟩
⟨*Lancer l'exploration* 121.3⟩
⟨*Mise à jour d'informations* 122.5⟩
⟨*Mise à jour de* l 128.1⟩
⟨*Mise à jour de* transient, l *et* leaf 127.4⟩
⟨*Nouvelle s.c.c. de racine* v *détectée* 123.4⟩
⟨*Programme de test* 119.1⟩
⟨*Prétraitement d'un nœud* 121.5⟩
⟨*Recueil d'informations additionnelles* 127.3⟩
⟨*Routines annexes* 124.2⟩
⟨*Stockage dans* i *de la couleur de la s.c.c.* 126.1⟩
⟨*Structures de données* 119.3⟩
⟨*Traitement additionnel sur la s.c.c.* 126.2⟩
⟨*Variables de Tarjan* 121.1⟩
⟨*Variables de Visit* 121.4⟩
⟨*Variables globales* 120.1⟩

B.5.2 Liste des identificateurs