Faculté
des Sciences

UMONS
Université de Mons

UNIVERSITÉ DE MONS

FACULTÉ DES SCIENCES
DÉPARTEMENT D'INFORMATIQUE
SERVICE D'INFORMATIQUE THÉORIQUE

MÉMOIRE DE MASTER

Implémentation d'algorithmes de décision pour la fonctionnalité et la sous-séquentialité de transducteurs finis.

Directrice

Mme Véronique BRUYÈRE

Auteur

Hugo ALLARD

Année académique 2014 - 2015

Remerciements

Je remercie ma directrice de mémoire, *Véronique Bruyère*, pour son aide de par ses explications et la relecture d'une partie de mon travail.
Je tiens également à remercier mon ami, *Maxime Deplasse*, pour sa relecture et ses remarques pertinentes.
Finalement, je remercie mes parents pour leur soutien.

Implémentation d'algorithmes de décision pour la fonctionnalité et la sous-séquentialité de transducteurs finis.

Hugo Allard

Résumé

Dans ce mémoire, nous étudions les transducteurs finis, et en particulier ceux qui réalisent des relations fonctionnelles et des fonctions sous-séquentielles. Ce sont des classes de transducteurs jouissant de bonnes propriétés de clôture et de décidabilité. Les transducteurs sous-séquentiels étant déterministes, ils sont d'un intérêt particulier pour l'implémentation. En effet, ce sont des transducteurs qui peuvent effectivement être implémentés avec une mémoire bornée. Nous voyons ensuite que la fonctionnalité et la sous-séquentialité sont elles-mêmes des propriétés décidables en temps polynomial pour les transducteurs non-déterministes, et proposons une implémentation d'algorithmes permettant de le faire.

Mots clés. Automate fini ; Transducteur fonctionnel ; Transducteur sous-séquentiel ; Déterminisation ; Implémentation

Table des matières

Chapitre 1

Introduction

Les transducteurs finis (FSTs) sont une extension des automates finis qui, en plus d'accepter ou refuser un mot d'entrée dans un langage rationnel, génèrent un mot de sortie également dans un langage rationnel. Ils réalisent donc une relation binaire entre deux langages rationnels. Les transducteurs offrent un modèle plus puissant que les automates mais, alors que les automates jouissent de très bonnes propriétés de clôture et de décidabilité, ce n'est la plupart du temps pas le cas pour les transducteurs. En effet, la classe des automates non-déterministes et la classe des automates déterministes sont équivalentes, closes pour toute opération booléenne et les problèmes de vide, inclusion, équivalence et universalité sont tous décidables. Pour les transducteurs finis, on peut principalement considérer deux classes : la classe des transducteurs non-déterministes qui étendent les automates non-déterministes et la classe des transducteurs déterministes qui étendent les automates déterministes. Ces deux classes ne sont pas équivalentes et on peut facilement prouver que les transducteurs non-déterministes sont plus expressifs. Malheureusement, la classe des transducteurs non-déterministes n'est pas close pour l'intersection et le complément et de plus, les problèmes d'inclusion et d'équivalence sont indécidables.

Cependant, lorsque l'on considère certains transducteurs non-déterministes, ceux qui définissent une relation fonctionnelle, les problèmes d'inclusion et d'équivalence deviennent décidables. Ces transducteurs sont d'autant plus intéressant que la fonctionnalité est une propriété décidable en temps polynomial.

Une autre classe de transducteurs intéressante, celle des transducteurs sous-séquentiels, est une extension des transducteurs déterministes qui associe un mot de sortie à chaque état final. Ces transducteurs sont dès lors plus expressifs que les transducteurs déterministes mais gardent de bonnes propriétés de décidabilité.

Il est également décidable en temps polynomial s'il existe un transducteur sous-séquentiel équivalent à un transducteur non-déterministe.

Ce mémoire propose une implémentation extensible des transducteurs finis et d'algorithmes de décision pour la fonctionnalité et la sous-séquentialité en temps polynomial, tels que proposés par Béal et al. [BCPS03], ainsi que d'un algorithme générant un transducteur sous-séquentiel équivalent à un transducteur non-déterministe passé en entrée, tel que proposé par Béal et Carton [BC02].

Le chapitre 2 fixe les notations utilisées dans ce mémoire et introduit les notions importantes sur les langages formels avant de détailler les modèles d'automates et de transducteurs. Il se clôture par une comparaison des propriétés de décidabilité entre les différentes classes de transducteurs. Le chapitre 3 présente une méthode de décision de l'ambiguïté des automates à partir de leur carré cartésien et montre comment étendre cette technique pour caractériser les transducteurs fonctionnels. Le chapitre 4 détaille les caractérisations des fonctions et transducteurs sous-séquentiels présentées par Choffrut [Cho77] et montre comment la technique employée au chapitre 3 peut également être appliquée pour les caractériser. Le chapitre 5 aborde les détails de l'implémentation d'un petit framework pour les transducteurs finis ainsi que des algorithmes permettant de vérifier les caractérisations présentées aux chapitres 3 et 4 en temps polynomial.

Chapitre 2

Terminologie

Ce chapitre introduit les notions importantes à la compréhension de ce mémoire et fixe les notations qui seront utilisées par la suite. La section 2.1 présente les concepts de base que sont les alphabets et les langages, la section 2.2 détaille les automates, qui sont le modèle sous-jacent des transducteurs qui sont, eux, discutés à la section 2.3. Enfin, la section 2.4 décrit plusieurs problèmes de décisions pour les automates et les transducteurs, montrant ainsi l'intérêt des classes de transducteurs fonctionnels et sous-séquentiels.

Les notations et la terminologie étant souvent différentes d'un auteur à l'autre, elles ont ici été fixées à partir de plusieurs sources. Les définitions et notations des alphabets, mots et langages proviennent essentiellement de Berstel [Ber79] et Hopcrof et al. [HMU06], les définitions d'automates et de transducteurs sont celles utilisées par Servais [Ser11].

2.1 Alphabets et mots

Un *alphabet* Σ est un ensemble fini et non-vide dont les éléments sont appelés *symboles*. Un *mot* u sur un alphabet Σ est une séquence de symboles de Σ.
Soit $u = a_1 \dots a_n$ avec $a_i \in \Sigma \ \forall i \leqslant n$. On note $|u| = n$ la *longueur* du mot u, c'est-à-dire le nombre de symboles le composant. On note $|u|_a$ avec $a \in \Sigma$ le nombre d'occurrences du symbole a dans le mot u. Le *mot vide*, de longueur nulle, est noté $\varepsilon : |\varepsilon| = 0$.
Soient $u = a_1 \dots a_n$ et $v = b_1 \dots b_m$ deux mots sur Σ. Leur *concaténation*, notée $u \cdot v = a_1 \dots a_n b_1 \dots b_m$, est le mot obtenu en faisant suivre les symboles de u par les symboles de v. La concaténation de k copies d'un même mot u est notée

$u^k = u \cdot u \cdot \ldots \cdot u$, en particulier $u^0 = \varepsilon$. Lorsqu'il est clair dans le contexte, l'opérateur $\cdot$ peut être omis et $u \cdot v$ devient uv. La concaténation est associative est admet ε comme élément neutre.

Soit $n \in \mathbb{N}$. $\Sigma^n = \{a_1 a_2 \ldots a_n \mid \forall i \leqslant n : a_i \in \Sigma\}$ est l'ensemble de tous les mots de longueur n sur Σ. En particulier, $\Sigma^0 = \{\varepsilon\}$ est l'ensemble contenant uniquement le mot vide.
L'*étoile de Kleene*, notée *, est un opérateur unaire sur un alphabet Σ tel que $\Sigma^* = \bigcup_{n \geqslant 0} \Sigma^n$. C'est-à-dire l'ensemble de tous les mots sur Σ.
Le *plus de Kleene*, noté $^+$, est tel que $\Sigma^+ = \bigcup_{n \geqslant 1} \Sigma^n$. C'est-à-dire l'ensemble de tous les mots non-vides sur Σ.

Définition 1 (Monoïde)**.** *Un ensemble $(M, \bullet)$ est un* monoïde *si $\bullet$ est une loi de composition associative sur M admettant un élément neutre $1_M \in M$.*

Soient $(A, \circ)$ et $(B, \bullet)$ deux monoïdes de neutre respectif 1_A et 1_B. Une application $h : A \to B$ est un *morphisme* de monoïde si :

1. $\forall x, y \in A : h(x \circ y) = h(x) \bullet h(y)$
2. $h(1_A) = 1_B$

Un *isomorphisme* est un morphisme qui admet un inverse qui est lui-même un morphisme. Deux structures sont dites *isomorphes* s'il existe un isomorphisme de l'une vers l'autre. Un monoïde M est dit *libre* s'il existe un isomorphisme de M à A^* où A est un alphabet.

Σ^* muni de la concaténation forme donc le monoïde libre $(\Sigma^*, \cdot)$ et Σ^+ muni de la concaténation forme quant à lui le *semi-groupe* $(\Sigma^+, \cdot)$.

Soient $u, v \in \Sigma^*$ deux mots sur Σ. On note $u \preceq v$ si u est *préfixe* de v, c'est-à-dire $v = uw$ avec $w \in \Sigma^*$. On note $u^{-1}v$ le mot obtenu en retirant le préfixe u de v : $u^{-1}v = w$. Le mot $w = u \wedge v \in \Sigma^*$ est le *plus long préfixe commun* de u et v, c'est-à-dire que $w \preceq u$, $w \preceq v$ et pour tout $w' \in \Sigma^*$ tel que $w' \preceq u$ et $w' \preceq v$ on a $w' \preceq w$. En particulier, le mot $(u \wedge v)^{-1}v$ est le mot obtenu de v en lui retirant son plus long préfixe commun avec u. La *distance préfixe* entre u et v est calculée comme $||u, v|| = |u| + |v| - 2|u \wedge v|$.

Soient Σ et Δ deux alphabets, u et $v = a_1 \ldots a_n \in \Sigma^*$ deux mots sur Σ. L'image de v par un morphisme $h : \Sigma^* \to \Delta^*$ est $h(v) = h(a_1 \cdot \ldots \cdot a_n) = h(a_1) \cdot \ldots \cdot h(a_n)$. On a donc également $h(u \cdot v) = h(u) \cdot h(v)$ et $h(\varepsilon) = \varepsilon$.

Soit Σ un alphabet. Un ensemble de mots sur Σ, $L \subseteq \Sigma^*$ est appelé un *langage* sur Σ. De la même manière, un ensemble de langages sur Σ est appelé une *classe*

de langages $\mathcal{L} \subseteq 2^{\Sigma^*}$.

Définition 2 (Langages rationnels)**.** *Soit M un monoïde, on définit l'ensemble des* parties rationnelles *de M, notée $Rat(M)$, comme le plus petit ensemble E des parties de 2^M contenant les singletons de M et clos pour les* opérations rationnelles *(l'union, la concaténation et l'étoile de Kleene) :*

1. $\varnothing \in E, \forall m \in M \quad \{m\} \in E$,
2. $\forall X, Y \in E, X \cup Y \in E, X \cdot Y \in E, X^* \in E$.

Un langage rationnel *sur un alphabet Σ est un élément de $Rat(\Sigma^*)$.*

Le langage vide sur un alphabet Σ est le langage $\{\varepsilon\}$ composé uniquement du mot vide.

Soient Σ et Δ deux alphabets. Une *transduction* $t : \Sigma^* \to \Delta^*$ est une relation $R \subseteq \Sigma^* \times \Delta^*$ entre des mots sur Σ et des mots sur Δ. Si $(u, v) \in R$, on dit que v est la *transduction* ou *l'image* de u par R. On note $R(u) = \{v \in \Delta^* \mid (u, v) \in R\}$ l'ensemble de toutes les transductions de u par R. Cette notation s'étend aux langages : $R(L) = \bigcup_{u \in L} R(u)$ est l'ensemble de toutes les transductions par R des mots de L.
On note $dom(R) = \{u \in \Sigma^* \mid \exists v \in \Delta^* : (u, v) \in R\}$ le *domaine* de R et $Im(R) = \{v \in \Delta^* \mid \exists u \in \Sigma^* : (u, v) \in R\}$ l'ensemble *image* de R.
Une transduction est dite *rationnelle* si sa relation R est une partie rationnelle de $\Sigma^* \times \Delta^*$. Elle est dite *fonctionnelle* si pour tout mot $u \in \Sigma^*$, il existe au plus un mot $v \in \Delta^*$ tel que $(u, v) \in R$. Afin d'alléger le vocabulaire, une relation fonctionnelle pourra plus simplement être appelée «fonction» par la suite.

Une *classe de transductions* $\mathcal{T}$ sur $\Sigma^* \times \Delta^*$ est un ensemble de transductions sur $\Sigma^* \times \Delta^* : \mathcal{T} \subseteq 2^{\Sigma^* \times \Delta^*}$.

2.2 Automates

Un *automate fini* est un modèle mathématique composé d'un ensemble d'états pouvant être initiaux et/ou finaux, et d'une relation de composition acceptant certains triplets (état courant, symbole courant, état suivant) qui forme un ensemble de transitions. Il prend en entrée un mot sur un certain alphabet, le lit de gauche à droite, symbole par symbole, et passe d'un état à un autre en sélectionnant une transition compatible avec le symbole lu à cette étape. Selon la configuration finale de l'automate, le mot d'entrée peut soit être accepté soit être refusé. L'ensemble de tous les mots acceptés par l'automate forme alors un langage L, on dit que l'automate *reconnaît* L. Il faut différencier les automates déterministes, qui

ne se trouvent que dans un seul état à un moment donné et les automates non-déterministes, qui peuvent se trouver dans plusieurs états en même temps.

Il convient maintenant de définir ces notions plus rigoureusement.

Définition 3 (ε-NFA). *Un automate fini asynchrone, aussi appelé automate à ε-transitions (ε-NFA pour* ε-transition Nondeterministic Finite Automaton*) est un quintuplet $\mathscr{A} = (\Sigma, Q, I, F, \delta)$ où*

- Σ *est l'alphabet d'entrée,*
- Q *est l'ensemble fini des états,*
- I *est l'ensemble des états initiaux,*
- $F \subseteq Q$ *est l'ensemble des états finaux,*
- $\delta \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \times Q$ *est la relation de transition.*

Soit $\mathscr{A}$ un ε-NFA. Un *chemin* ρ sur $\mathscr{A}$ avec l'entrée $u = a_1 \dots a_n$ de l'état q_0 à l'état q_n est une séquence de transitions de la forme $\rho = (q_0, a_1, q_1) \dots (q_{n-1}, a_n, q_n)$ telle que pour tout $i \in \{1, \dots, n\}$, on a $a_i \in \Sigma \cup \{\varepsilon\}$ et $(q_{i-1}, a_i, q_i) \in \delta$. Le chemin est dit *acceptant* si $q_0 \in I$ et $q_n \in F$. Un mot u est *accepté* par un automate $\mathscr{A}$ s'il existe un chemin acceptant sur $\mathscr{A}$ avec l'entrée u. On note $\rho^{acc}(\mathscr{A}, u)$ l'ensemble des chemins acceptants sur $\mathscr{A}$ avec l'entrée u. Le langage $L(\mathscr{A}) = \{u \in \Sigma^* \mid \rho^{acc}(\mathscr{A}, u) \neq \varnothing\}$ est l'ensemble des mots acceptés par $\mathscr{A}$, c'est-à-dire le langage *reconnu* par $\mathscr{A}$. Deux automates sont *équivalents* s'ils reconnaissent le même langage.
Soient $q_0, \dots, q_n \in Q$, $u = a_1 \dots a_n \in \Sigma^*$, on écrit la transition (q_0, a_1, q_1) comme $q_0 \xrightarrow{a_1} q_1$, on écrit le chemin $(q_0, a_1, q_1) \dots (q_{n-1}, a_n, q_n)$ comme $q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \dots \xrightarrow{a_{n-1}} q_{n-1} \xrightarrow{a_n} q_n$ ou comme $q_0 \xrightarrow{u} q_n$ s'il n'y a pas d'ambiguïté sur l'étiquette de la flèche. On définit la taille d'un automate comme suit : $|\mathscr{A}| = |Q| + |\delta|$.

Un automate peut être visualisé par une table de transition qui représente la relation δ. Les lignes correspondent aux états et les colonnes correspondent aux entrées. Dans ce cas, un état initial est marqué d'une flèche et un état final est marqué d'un *. Soient $q \in Q$, $a \in \Sigma \cup \{\varepsilon\}$, la case de la table de transition correspondant à (q, a) est l'ensemble $\{p \in Q \mid (q, a, p) \in \delta\}$.
Un automate peut également être représenté par un diagramme de transition qui est un graphe tel que

- Tout état de Q est représenté par un nœud.
- Pour tout $q, p \in Q$, $a \in \Sigma \cup \{\varepsilon\}$, si $(q, a, p) \in \delta$ alors le graphe possède une flèche du nœud q au nœud p.
- Il y a un arc sans origine pointant vers les états initiaux.
- Tout état final $q \in F$ est marqué par un double cercle ou par un arc sortant sans nœud destination.

	a	b	ε
$\rightarrow q_0$	$\{q_0\}$	$\{q_1, q_2\}$	$\varnothing$
$*q_1$	$\{q_1\}$	$\varnothing$	$\varnothing$
q_2	$\{q_1\}$	$\{q_0\}$	$\{q_0\}$

Table de transitions

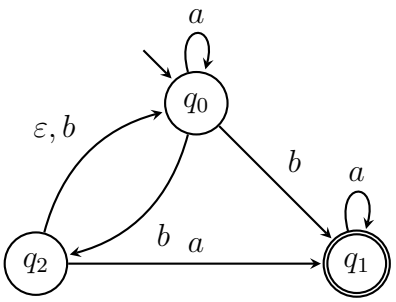


Diagramme de transitions

FIGURE 2.1 – Automate fini asynchrone

La figure 2.1 montre un automate asynchrone sous ces deux formes.

Soit $q \in Q$ un état d'un ε-NFA, q est dit *accessible* s'il existe un chemin allant d'un état initial jusque l'état q. On dit que q est *co-accessible* s'il existe un chemin allant de q à un état final. Un état est dit *utile* s'il est à la fois accessible et co-accessible. Un automate est *émondé* si tous ses états sont utiles. Il est possible d'émonder un automate en temps linéaire, un algorithme est donné à la section 5.3.

Définition 4 (NFA). *Un ε-NFA $\mathscr{A} = (\Sigma, Q, I, F, \delta)$ tel que $\delta \subseteq Q \times \Sigma \times Q$ est appelé* Automate fini non déterministe temps réel *ou Automate fini non déterministe (NFA pour* Nondeterministic Finite Automaton*).*

Un NFA est donc un ε-NFA sans ε-transition. Il est prouvé que la classe des ε-NFA et celle des NFA ont la même expressivité et qu'il existe un algorithme retirant les ε-transitions en temps polynomial. L'algorithme se base sur la fermeture transitive pour détecter les états atteignables par des ε-transitions et ensuite les supprimer. Une preuve et plus de détails sont donnés par Hopcroft et al. [HMU06]. Une propriété intéressante est que puisqu'un NFA lit exactement un symbole dans chaque état, la longueur d'un chemin pour un mot en entrée est égal à la taille du mot. De plus, pour un NFA $\mathscr{A} = (\Sigma, Q, I, F, \delta)$ et un mot d'entrée u, il y a au maximum $|\delta|^{|u|}$ chemins possibles.

Définition 5 (DFA). *Un NFA $\mathscr{A} = (\Sigma, Q, I, F, \delta)$ tel que $|I| = 1$ et $((q, a, q') \in \delta \wedge (q, a, q'') \in \delta \Rightarrow q' = q'')$ est dit* déterministe *(DFA pour Deterministic Finite Automaton).*

Un automate déterministe (DFA) est donc un NFA possédant un seul état initial et tel qu'à chaque symbole lu, il n'existe qu'une seule transition sortant de l'état courant qui soit compatible. La relation de transition est donc fonctionnelle

et en conséquence il existe au plus un chemin possible par mot en entrée.

Il est prouvé que tout NFA, et donc tout ε-NFA, est équivalent à un DFA. De plus, il existe un algorithme prenant en entrée un NFA et construisant un DFA équivalent dont la taille est exponentiellement plus grande. Cette procédure est appelée *déterminisation* et se base sur la *construction des sous-ensembles*. Une brève description de l'algorithme est donné ici, plus d'informations ainsi qu'une preuve de l'équivalence de la classe des NFA et celle des DFA sont données par Hopcroft et al. [HMU06].

Soit $\mathscr{N} = (\Sigma, Q_N, I, F_N, \delta_N)$ un NFA, l'algorithme construit un DFA $\mathscr{D} = (\Sigma, Q_D, q_0, F_D, \delta_D)$ tel que $L(\mathscr{D}) = L(\mathscr{N})$ comme suit :

- $q_0 = I$, l'unique état initial de $\mathscr{D}$ contient tous les états initiaux de $\mathscr{N}$.
- $Q_D = 2^{Q_N}$, c'est-à-dire l'ensemble des sous-ensembles de Q_N. Certains de ces $2^{|Q_N|}$ états ne sont pas utiles mais $\mathscr{D}$ peut être émondé.
- $F_D = \{S \in Q_D \mid S \cap F_N \neq \varnothing\}$, c'est-à-dire tous les états de $\mathscr{D}$ qui possèdent au moins un état final de $\mathscr{N}$.
- $\delta_D(S, a) = \bigcup_{p \in S} \delta_N(p, a)$, pour tout $S \subseteq Q_N$ et pour tout $a \in \Sigma$. C'est-à-dire que $\delta_D(S, a)$ est l'union de tous les états accessibles dans $\mathscr{N}$ depuis l'état p avec l'entrée a.

La figure 2.2 montre la construction d'un DFA par cet algorithme. Le DFA résultant est émondé par souci de clarté.

Les classes ε-NFA, NFA et DFA ont donc la même expressivité et il existe des algorithmes pour passer de l'une à l'autre.

Définition 6 (Langage reconnaissable)**.** *Un langage est dit* reconnaissable *s'il existe un atomate fini le reconnaissant. On note* $Rec(\Sigma^*)$ *l'ensemble des langages reconnaissables sur* Σ^**.*

Théorème 1 (Kleene)**.** *Soit* Σ *un alphabet. Un langage sur* Σ *est rationnel si et seulement si il est reconnu par un automate :*

$$Rat(\Sigma^*) = Rec(\Sigma^*).$$

Les trois classes d'automates finis sont équivalentes et caractérisent les *langages rationnels.*

Définition 7. *Soit* $k \in \mathbb{N}$*, un* ε*-NFA* $\mathscr{A}$ *est* k-ambigu *si et seulement si pour tout mot* $u \in \Sigma^*$ *on a* $|\rho^{acc}(\mathscr{A}, u)| \leqslant k$*.*

Dans le cas où $k = 1$, c'est-à-dire un automate 1-ambigu, on dit que l'automate

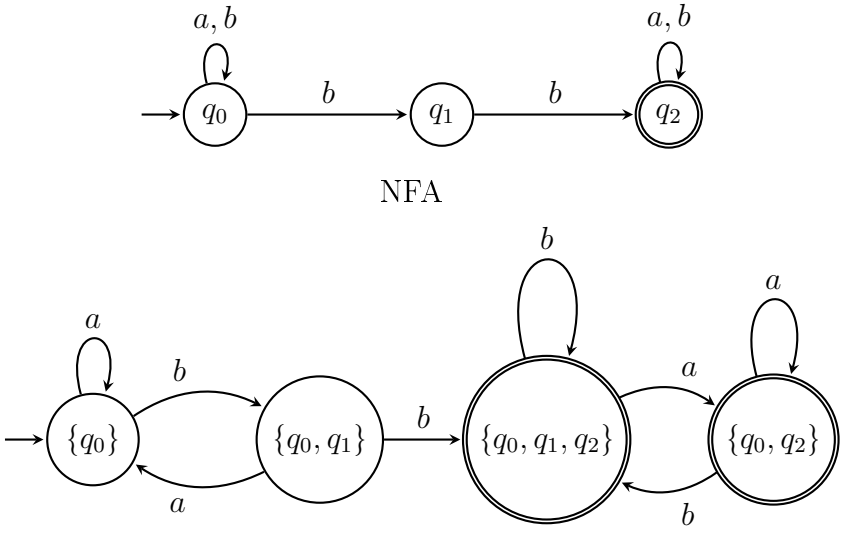


FIGURE 2.2 – Déterminisation par construction des sous-ensembles

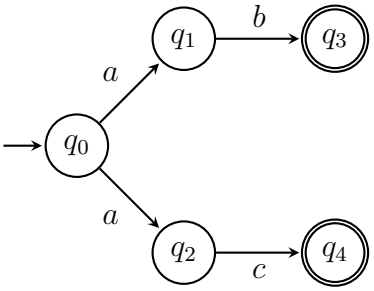


FIGURE 2.3 – Automate non-ambigu et non-déterministe

est *non-ambigu*. Un DFA est forcément non-ambigu mais un automate non-ambigu peut ne pas être déterministe. La figure 2.3 montre un tel automate reconnaissant les mots *ab* et *ac*, clairement cet automate est non-déterministe mais il n'existe qu'un seul chemin par mot. Il est dès lors non-ambigu.

2.3 Transducteurs

Un *transducteur fini* est essentiellement un automate fini augmenté d'un morphisme de sortie qui associe un mot à chaque transition de l'automate. L'automate fini sur lequel se base le transducteur est appelé *automate sous-jacent*. Soient

$\mathscr{A} = (\Sigma, Q, I, F, \delta)$ l'automate sous-jacent d'un transducteur, Ω le morphisme qui lui est associé et $\rho = (q_0, a_0, q_1) \dots (q_{n-1}, a_n, q_n)$ avec $q_i \in Q, a_i \in \Sigma \cup \{\varepsilon\}, \forall i \leqslant n$, un chemin sur l'automate sous-jacent. La sortie de ce transducteur pour ce chemin est alors $\Omega(\rho) = h((q_0, a_0, q_1) \dots (q_{n-1}, a_n, q_n)) = h((q_0, a_0, q_1)) \dots h((q_{n-1}, a_n, q_n))$, c'est-à-dire la concaténation des sorties de chacune des transitions du chemin.

Définition 8 (ε-NFT)**.** *Un transducteur fini asynchrone de Σ^* à Δ^* (ε-NFT pour ε-transitions Nondeterministic Finite Transducer) est une paire $\mathscr{T} = (\mathscr{A}, \Omega)$ où $\mathscr{A} = (\Sigma, Q, I, F, \delta)$ est un ε-NFA qui est l'automate sous-jacent d'entrée, et $\Omega : \delta \to \Delta^*$ est un morphisme appelé* sortie.

Soient $\mathscr{T} = (\mathscr{A} = (\Sigma, Q, I, F, \delta), \Omega : \delta \to \Delta^*)$ un ε-NFT, $u \in \Sigma^*$ et $\rho = t_1 \dots t_n \in \rho^{acc}(\mathscr{A}, u)$. On dit que $v = \Omega(\rho) = \Omega(t_1 \dots t_n) = \Omega(t_1) \dots \Omega(t_n) \in \Delta^*$ est la sortie du chemin et l'image de u par $\mathscr{T}$. On définit l'*automate sous-jacent de sortie* comme $\mathscr{A}' = (\Delta, Q, I, F, \delta')$ où $\delta' = \{(q, \Omega(q, a, p), p) \mid (q, a, p) \in \delta\}$, c'est-à-dire que l'étiquette de chaque transition dans l'automate sous-jacent d'entrée est remplacée par l'image de cette transition par le morphisme de sortie.

Théorème 2. *Une transduction est rationnelle si et seulement si il existe un transducteur fini la réalisant.*

Un transducteur $\mathscr{T}$ de Σ^* à Δ^* peut donc être vu comme une paire d'automates, un automate d'entrée et un automate de sortie reconnaissant respectivement deux langages rationnels $L_{in} \subseteq \Sigma^*$ et $L_{out} \subseteq \Delta^*$. Il réalise une transduction rationnelle $R \subseteq L_{in} \times L_{out}$. Le domaine ($\mathrm{Dom(T)} = L_{in}$), resp. l'image ($\mathrm{Im(T)} = L_{out}$), du transducteur est le domaine, resp. l'image, de la transduction reconnue par le transducteur. Le transducteur peut être représenté comme un automate dont les transitions sont étiquetées par des paires (entrée, sortie) $\in (\Sigma \cup \{\varepsilon\}) \times \Delta^*$. On écrit $q \xrightarrow{a/b} p$ une transition de q à p prenant a en entrée et b en sortie. De la même manière qu'avec les automates, cette notation peut s'étendre aux chemins. La transduction reconnue par $\mathscr{T}$ est donc $R(\mathscr{T}) = \{(u, v) \in \Sigma^* \times \Delta^* \mid q \xrightarrow{u/v} p : q \in I, p \in F\}$

On notera également *NFT*, resp. *DFT*, la classe des transducteurs non-déterministes, resp. des transducteurs déterministes, c'est-à-dire les transducteurs finis dont l'automate sous-jacent d'entrée appartient à la classe NFA, resp. DFA. On note R(ε-NFT), resp. R(NFT), resp. R(DFT), la classe de transductions reconnues par un ε-NFT, resp. NFT, resp. DFT.

Dans ce cas, les NFTs et les DFTs sont dits *temps réel* car la première composante de chaque transition est un symbole de Σ et non un mot de Σ^*. Pour tout transducteur il existe un transducteur temps-réel équivalent [BCPS03] et on ne considère par la suite, sans perte de généralité, que les transducteurs temps réel.

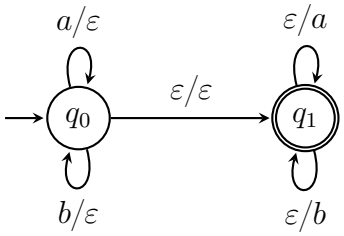


FIGURE 2.4 – ε-NFT $\mathscr{T}_\infty$ sur $\Sigma = \Delta = \{a, b\}$

Exemple 1 ([Ser11])**.** *Soient $\Sigma = \Delta = \{a, b\}$. Le transducteur $\mathscr{T}_\infty$, visible à la figure 2.4, définit la transduction $R(\mathscr{T}_\infty) = \Sigma^* \times \Delta^*$ qui est la transduction universelle sur Σ. Ce transducteur est un ε-NFT.*

L'exemple 1 montre un ε-NFT qui associe tout mot de Σ^* à tout mot de Δ^*. Chaque mot d'entrée a alors une infinité d'images. Or pour un NFT $\mathscr{T}$, il y a au plus $n = |\delta|^{|u|}$ chemins possibles pour un mot d'entrée u (avec δ comme relation de transition de l'automate sous-jacent d'entrée de $\mathscr{T}$), il y a donc au plus n images pour une entrée donnée. De la même manière, pour un DFT, il existe au plus un chemin par entrée et donc une seule image possible. On a donc les inclusions strictes R(DFT) $\subset$ R(NFT) $\subset$ R(ε-NFT).

Un transducteur dont l'automate sous-jacent d'entrée est k-ambigu, resp. non-ambigu est lui-même dit *k-ambigu*, resp. *non-ambigu*. La classe des transducteurs *fonctionnels* est l'ensemble de tous les transducteurs réalisant une transduction fonctionnelle.

Un DFT est forcément fonctionnel. Cependant, un NFT non-déterministe peut réaliser une transduction fonctionnelle. Pour certains de ces NFT fonctionnels, il n'existe d'ailleurs pas de DFT équivalent. La figure 2.5 montre un transducteur fonctionnel $\mathscr{T}$ tel que $R(\mathscr{T}) = \{(xy^{2n}x, ab^n a^n b) \mid n \in \mathbb{N}\} \cup \{(xy^{2n}z, bb^n a^n a) \mid n \in \mathbb{N}\}$. Clairement, l'automate sous-jacent d'entrée est non-déterministe puisque les deux transitions sortant de q_0 on la même étiquette d'entrée. Cependant, l'entrée des transitions entrant dans q_5 étant différente, les mots d'entrée seront forcément différents selon qu'ils passent par q_1 ou q_2, le transducteur est donc fonctionnel. $\mathscr{T}$ n'est pourtant pas déterminisable. Une preuve du caractère non déterminisable est donné au chapitre 4 et l'application de l'algorithme de déterminisation sur cet exemple est montrée à la section 5.7.

De la même manière, un NFT non-ambigu est forcément fonctionnel mais un NFT ambigu peut malgré tout réaliser une transduction fonctionnelle. Dans ce cas tous les chemins acceptants sur une certaine entrée doivent produire la même

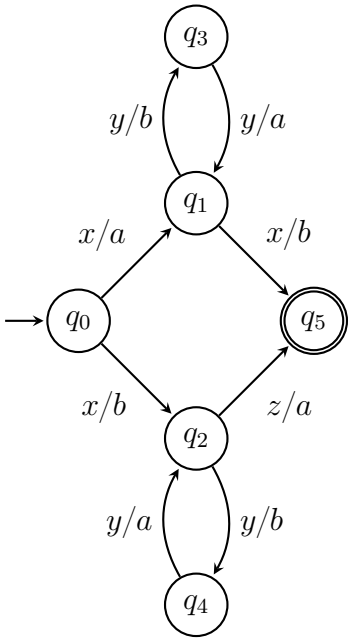


FIGURE 2.5 – NFT fonctionnel mais non déterminisable [AM03]

sortie. La figure 2.6 montre un tel transducteur simple pour lequel il n'y a que deux chemins réussis distincts : le chemin $q_0 \xrightarrow{x/a} q_1 \xrightarrow{x/bc} q_2$ et le chemin $q_0 \xrightarrow{x/ab} q_2 \xrightarrow{x/c} q_3$. Clairement, les deux chemins prennent la même entrée, l'automate sous-jacent est donc ambigu, mais ces deux chemins produisent la même sortie, le transducteur est donc fonctionnel.

Les NFT non-ambigus sont plus puissants que les DFT et il a été prouvé que toutes les transductions fonctionnelles peuvent être réalisées par un NFT non-ambigu.[EM65]

Il est possible d'étendre les DFT en ajoutant un morphisme de sortie pour

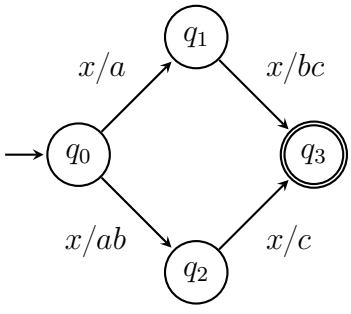


FIGURE 2.6 – NFT ambigu mais fonctionnel

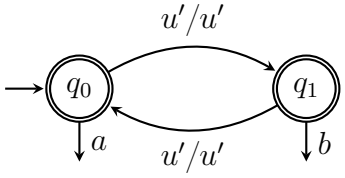


FIGURE 2.7 – Transducteur sous-séquentiel reconnaissant la transduction de l'exemple 2 avec u' un symbole quelconque de Σ

les états finaux. Dans ce cas, un mot de sortie supplémentaire est ajouté à la suite du mot produit pour l'entrée. Un tel transducteur est appelé *sous-séquentiel*. Ce terme fait référence aux transducteurs *séquentiels* qui sont les transducteurs déterministes en entrée. La distinction entre les termes séquentiels et DFT provient du fait que les transducteurs peuvent être déterministes en entrée et déterministes en sortie. Séquentiel signifie donc déterministe en entrée et évite l'ambiguïté du terme déterministe. Certains auteurs ajoutent une contrainte supplémentaire aux transducteurs séquentiels en ne considérant que les transducteurs déterministes en entrée dont tous les états sont finaux. Dans le cadre de ce mémoire, séquentiel et DFT seront équivalents.

Une transduction est dite *sous-séquentielle* si et seulement si il existe un transducteur sous-séquentiel la réalisant.

Définition 9. *Un transducteur sous-séquentiel $\mathscr{T}$ de Σ à Δ est une paire $(\mathscr{T}', \Omega_f)$ où $\mathscr{T}' = (\mathscr{A} = (\Sigma, Q, I, F, \delta), \Omega)$ est un DFT et Ω_f est un morphisme de F à Δ^*.*

La sortie d'un chemin réussi $\rho = q_0 \xrightarrow{u/v} q_n$ sur un transducteur sous-séquentiel $\mathscr{T}$ est donc $v \cdot \Omega_f(q_n)$. Les transducteurs sous-séquentiels sont plus expressifs que les DFT. En effet, l'exemple 2 montre une relation reconnaissable par un transducteur sous-séquentiel mais pas par un transducteur séquentiel. De fait, on peut prouver que pour une transduction séquentielle f et deux mots $u, v \in Dom(f)$ tels que $u \prec v$ alors $f(u) \prec f(v)$. Hors, pour l'exemple 2, un mot $w \in \Sigma^*$ et un symbole $x \in \Sigma, f(w) \not\prec f(wx)$.

La figure 2.7 montre un transducteur sous-séquentiel réalisant la transduction de l'exemple 2. Par la suite, un état final q d'un transducteur sous-séquentiel sera représenté par double cercle duquel sort un flèche sans destination, étiquetée par $\Omega_f(q)$.

Exemple 2 ([Ser11]). $R_{odd} = \{(u, ua) \mid u \in \Sigma^* \wedge |u| \textit{ est impair}\} \cup \{(u, ub) \mid u \in \Sigma^* \wedge |u| \textit{ est pair}\}$

Inversement, chaque transduction déterministe est forcément sous-séquentielle puisqu'il suffit de considérer un morphisme qui associe le mot vide à chaque état final.

2.4 Problèmes de décision

Soient $\mathscr{A}_1, \mathscr{A}_2$ deux automates finis sur Σ et $u \in \Sigma^*$ un mot sur Σ. Il est intéressant de considérer plusieurs problèmes de décision :

Appartenance revient à tester $u \in L(\mathscr{A}_1)$

Vide revient à tester $L(\mathscr{A}_1) = \varnothing$

Universalité revient à tester $L(\mathscr{A}_1) = \Sigma^*$

Équivalence revient à tester $L(\mathscr{A}_1) = L(\mathscr{A}_2)$

Inclusion revient à tester $L(\mathscr{A}_1) \subseteq L(\mathscr{A}_2)$

La plupart des problèmes de décision sont décidables pour les automates.

Proposition 1 ([Ser11])**.** *Les problèmes suivants sont décidables :*

- *Le vide et l'appartenance sont décidables en temps polynomial pour ε-NFA et NFA et DFA.*
- *L'universalité, l'inclusion et l'équivalence sont PSpace-complets pour ε-NFA et NFA. Décidables en temps polynomial pour DFA.*
- *Pour un certain k, on peut vérifier si un NFA est k-ambigu en temps polynomial. Une description de l'algorithme pour décider l'ambiguïté est donné au chapitre 3.*

De la même manière que pour les automates, on s'intéresse aux problèmes de décisions pour les transducteurs. Soient $\mathscr{T}_1, \mathscr{T}_2$ deux transducteurs finis de Σ^* à Δ^*, et $u \in \Sigma^*, v \in \Delta^*$ deux mots.

Appartenance revient à tester $(u, v) \in R(\mathscr{T}_1)$

Vide revient à tester $R(\mathscr{T}_1) = \varnothing$

Universalité revient à tester $R(\mathscr{T}_1) = \Sigma^* \times \Delta^*$

Équivalence revient à tester $R(\mathscr{T}_1) = R(\mathscr{T}_2)$

Inclusion revient à tester $R(\mathscr{T}_1) \subseteq R(\mathscr{T}_2)$

	Vide Appartenance	Equivalence Inclusion	Universalité
ε-NFT	PTime	indéc.	indéc.
NFT	PTime	indéc.	-
DFT	PTime	PTime	-

TABLE 2.1 – Problèmes de décisions pour les transducteurs finis

La relation définie par un transducteur $\mathscr{T}$ est vide si et seulement si $Dom(\mathscr{T})$ est vide. L'appartenance d'une paire $(u, v) \in \Sigma^* \times \Delta^*$ se teste en vérifiant l'appartenance de v dans l'image de u par $\mathscr{T}$. Il est possible de créer un NFA reconnaissant l'image de u par $\mathscr{T}$ en temps polynomial.

Proposition 2 ([Ser11]). *Le vide et l'appartenance sont décidables en temps polynomial pour ε-NFT.*

Griffiths a prouvé [Gri68] par une réduction au problème de correspondance de Post que l'équivalence, l'inclusion et l'universalité sont indécidables pour ε-NFT et NFT.

Théorème 3 ([Gri68]). *L'équivalence et l'inclusion sont indécidables pour ε-NFT et NFT. L'universalité est indécidable pour ε-NFT.*

Ces résultats sont répértoriés à la table 2.1. L'équivalence et l'inclusion, deux problèmes importants, sont tous deux indécidables pour NFT. Cependant, le théorème 4 montre que pour un sous-ensemble de NFT, les NFT fonctionnels, ces problèmes deviennent décidables.

Théorème 4 ([dS09]). *L'équivalence et l'inclusion des NFT fonctionnels sont PSpace-complets.*

La preuve du théorème 4 se base sur le théorème 5 et ce dernier rend le théorème précédent d'autant plus intéressant.

Théorème 5 ([Sch75]). *La fonctionnalité est une propriété décidable pour les transducteurs finis.*

Schutzenberger [Sch75] a prouvé que la fonctionnalité pour un NFT est décidable en NPTime. Ce résultat a par la suite été amélioré par Gurari et Ibarra [GI83] en temps polynomial. Une autre procédure a été présentée en 2003 par Béal et al. [BCPS03]. Cette dernière est détaillée au chapitre 3 et son implémentation à la section 5.5.

Théorème 6 ([Cho77]). *La sous-séquentialité est une propriété décidable pour les fonctions rationnelles.*

Choffrut [Cho77] a prouvé que les NFT sous-séquentialisables respectent une certaine *propriété de jumelage* qui a été prouvée décidable en temps polynomial [WK95]. Une procédure pour décider la sous-séquentialité [BCPS03] et pour construire un transducteur sous-séquentiel équivalent à un NFT réalisant une transduction sous-séquentielle [BC02] ont été présentées par Béal et al.. Ces dernières sont détaillées au chapitre 4 et leur implémentation aux section 5.6 et 5.7

Chapitre 3

Transducteurs fonctionnels

Ce chapitre présente la procédure décrite par Béal et al. [BCPS03] pour décider la fonctionnalité d'un NFT. Cette méthode se base sur la construction du carré du transducteur de la même manière que pour décider l'ambiguïté d'un automate. Le carré du transducteur est ensuite multiplié par le semi-automate d'une action particulière et une propriété sur le transducteur résultant permet de décider la fonctionnalité.

La première section présente le produit cartésien de deux automates et décrit la procédure permettant de décider l'ambiguïté à partir du carré d'un automate. La seconde section introduit la notion d'action d'un monoïde sur un ensemble et présente une action particulière qui sera utilisée par la suite. Enfin, la troisième section décrit la procédure proposée par Béal et al. pour décider la fonctionnalité. L'implémentation est détaillée à la section 5.5.

3.1 Ambiguïté d'un automate

Soient $\mathscr{A}' = (\Sigma, Q', I', F', \delta')$ et $\mathscr{A}'' = (\Sigma, Q'', I'', F'', \delta'')$ deux automates sur Σ. Le produit cartésien de $\mathscr{A}'$ et $\mathscr{A}''$ est l'automate $\mathscr{C} = \mathscr{A}' \times \mathscr{A}'' = (\Sigma, Q' \times Q'', I' \times I'', F' \times F'', \delta_{\mathscr{C}})$ où $\delta_{\mathscr{C}}$ est la relation de transition définie comme

$$\delta_{\mathscr{C}} = \{((p', p''), a, (q', q'')) \mid (p', a, q') \in \delta' \wedge (p'', a, q'') \in \delta''\}.$$

En particulier, le carré d'un automate est son produit cartésien avec lui-même. Soit $\mathscr{A}^2 = \mathscr{A} \times \mathscr{A} = (\Sigma, Q \times Q, I \times I, F \times F, \delta_{\mathscr{A}^2})$ le produit cartésien de l'automate $\mathscr{A} = (\Sigma, Q, I, F, \delta)$ avec lui-même. La relation de transition du carré est définie comme

$$\delta_{\mathscr{A}^2} = \{((p,r),a,(q,s)) \mid (p,a,q),(r,a,s) \in \delta\}.$$

On appelle *diagonale* de $\mathscr{A} \times \mathscr{A}$ le sous-automate $\mathscr{D}$ qui possède la diagonale $Q_{\mathscr{D}} = \{(q,q) \mid q \in Q\} \subseteq Q \times Q$ comme ensemble d'états. Les états et les transitions de $\mathscr{A}$ et $\mathscr{D}$ sont en bijection, les deux automates sont donc équivalents.

Lemme 1 ([BP$^+$85])**.** *Un automate $\mathscr{A}$ est non-ambigu si et seulement si la partie émondée de $\mathscr{A} \times \mathscr{A}$ est égale à $\mathscr{D}$.*

Démonstration. Par définition, $\mathscr{A}$ est ambigu si et seulement si il existe deux chemins réussis ρ' et ρ'' de même étiquette $f = a_1 a_2 \dots a_n$:

$$\rho' := q'_0 \xrightarrow{a_1} q'_1 \xrightarrow{a_2} \dots \xrightarrow{a_n} q'_n \text{ et } \rho'' := q''_0 \xrightarrow{a_1} q''_1 \xrightarrow{a_2} \dots \xrightarrow{a_n} q''_n,$$

c'est-à-dire si et seulement si il existe un chemin réussi ρ dans $\mathscr{A} \times \mathscr{A}$:

$$\rho := (q'_0, q''_0) \xrightarrow{a_1} (q'_1, q''_1) \xrightarrow{a_2} \dots \xrightarrow{a_n} (q'_n, q''_n),$$

dans lequel pour au moins un i, $0 \leqslant i \leqslant n$, on a $q'_i \neq q''_i$. Et donc si et seulement si il existe un état utile dans $\mathscr{A} \times \mathscr{A}$ qui ne soit pas dans $\mathscr{D}$. □

Proposition 3. *L'ambiguïté d'un automate fini est un problème décidable.*

Le déterminisme est également décidable à partir du carré de l'automate en modifiant la définition.

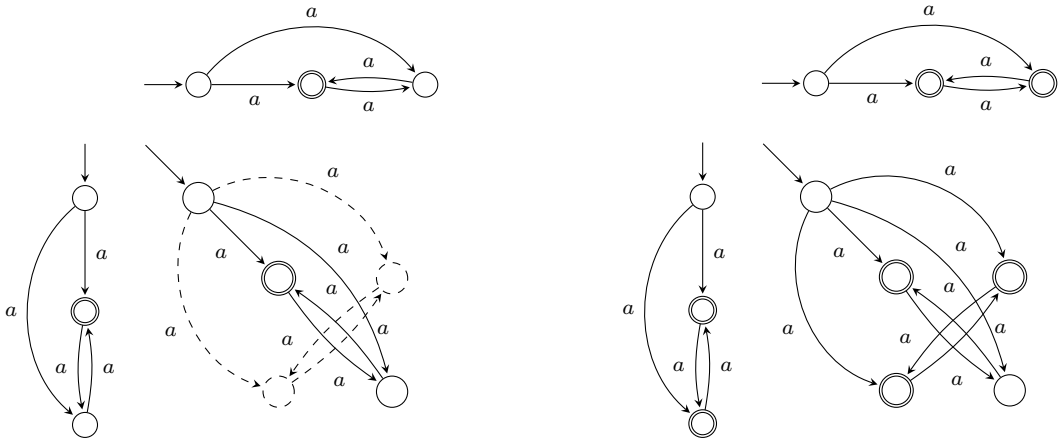
Lemme 2. *Un automate émondé $\mathscr{A}$ est déterministe si et seulement si la partie accessible de son carré $\mathscr{A} \times \mathscr{A}$ est égale à $\mathscr{D}$.*

La figure 3.1 illustre la construction du carré d'un automate dans le cas ambigu et le cas non-ambigu. Les transitions et les états discontinus représentent les états et transitions non co-accessibles. Dans le cas non-ambigu on a bien que la partie émondée du carré est égale à la diagonale.

3.2 Produit par une action

Définition 10 (Action (droite))**.** *Soient $(M, \bullet)$ un monoïde de neutre 1_M et X un ensemble. Une action (droite) de M sur X est un triplet (X, M, ϕ) où ϕ est une application $\phi : X \times M \to X$ compatible avec l'opération $\bullet$ du monoïde et son neutre :*

1. $\forall a, b \in M$ *et* $x \in X : \phi(\phi(x,a),b) = \phi(x,(a \bullet b))$,



Cas non-ambigu Cas ambigu

FIGURE 3.1 – ([BCPS03]) Construction du carré émondé. En lignes discontinues, la partie non co-accessible.

2. $\forall x \in X, \phi(1_M, x) = x$.

Lorsque le monoïde et l'ensemble associés à l'action sont clairs dans le contexte, il peut être fait référence à l'action par l'application seule.
Une action de M sur X, (X, M, ϕ), peut alors être vue comme un automate $\mathscr{G}_\phi = (M, X, \phi)$ où M est le monoïde d'où proviennent les étiquettes des transitions de $\mathscr{G}_\phi$, X est l'ensemble des états de $\mathscr{G}_\phi$ et ϕ est la fonction de transition les reliant. Cet automate est appelé un *semi-automate* car il ne comporte aucun état initial et aucun état final. Cependant, il peut parfois être intéressant de considérer un élément particulier de X comme état initial. Soit $\phi_w : X \to X$ l'application définie récursivement comme :

$$\begin{cases} \phi_{1_M}(x) = x \\ \phi_a(x) = \phi_M(x, a) \text{ avec } a \in M \\ \phi_{aw}(x) = \phi_w(\phi_a(x)) \text{ avec } a \in M, w \in M^* \end{cases}$$

alors l'ensemble engendré par $\{\phi_w x \mid w \in M^*\}$ est clos par composition de ϕ_w qui est associative et admet le neutre ϕ_{1_M}. Il forme dès lors un monoïde appelé *monoïde de transition* de l'action ϕ. Le monoïde de transition est isomorphe à $\mathbb{Z}$ lorsque $|M| = 1$ et isomorphe à $M^* \times M^*$ lorsque $|M| \geqslant 2$.

Il est donc possible de multiplier un automate par une action sous sa forme de semi-automate. Soient $\mathscr{A} = (M, Q, \delta, I, F)$ un automate fini émondé, $(X, M, \phi : X \times M \to X)$ une action de M sur X et $x_0 \in X$ un élément particulier de X. Alors

on calcule le produit cartésien de $\mathscr{A}$ avec le semi-automate de l'action comme suit :

$$\mathscr{A} \times \mathscr{G}_\phi = (M, Q \times X, \Phi, I \times \{x_0\}, F \times X)$$

avec pour ensemble de transitions

$$\Phi = \{((p,x), m, (q, \phi(x,m))) \mid x \in X \wedge (p,s,q) \in \delta\}.$$

L'automate $\mathscr{G}_\phi$ est souvent infini puisque l'ensemble X et le monoïde M sont eux-mêmes souvent infinis. On considère donc le produit d'un automate par une action, noté $\mathscr{A} \times \phi$, la partie accessible de $\mathscr{A} \times \mathscr{G}_\phi$.

La projection sur la première composante induit une bijection entre les transitions de $\mathscr{A}$ dont l'origine est p et les transitions de $\mathscr{A} \times \phi$ dont l'origine est (p,x). On a par induction sur la longueur des chemins :

$$(p,x) \xrightarrow[\mathscr{A}\times\phi]{m} (q,t) \Rightarrow t = \phi(x,m).$$

L'information $x \in X$ associée à chaque état de $\mathscr{A} \times \phi$ est appelée la *valeur* de l'état. Lorsque la projection sur la première composante associe exactement un état de $\mathscr{A} \times \phi$ à chaque état de $\mathscr{A}$, on dit que le produit $\mathscr{A} \times \phi$ est une *valuation*.

Soit Δ^* un monoïde libre de neutre ε, on définit un ensemble $H_\Delta \subset \Delta^* \times \Delta^*$ qui contient les paires de $\Delta^* \times \Delta^*$ dont un des membres est le mot vide ε auxquelles est adjoint un zéro noté $\mathbf{0}$ afin de le distinguer du zéro de $\mathbb{Z}$. Bien que la définition soit générale, il est ici volontairement fait référence au vocabulaire et aux notations des alphabets car c'est l'utilisation qui en sera faite.

$$H_\Delta = (\Delta^* \times \varepsilon) \cup (\varepsilon \times \Delta^*) \cup \{\mathbf{0}\}$$

On définit une application $\psi : \Delta^* \times \Delta^* \to H_\Delta$ comme

$$\forall u, v \in \Delta^* : \psi(u,v) = \begin{cases} (v^{-1}u, \varepsilon) \text{ si } v \prec u \\ (\varepsilon, u^{-1}v) \text{ si } u \prec v \\ \mathbf{0} \text{ sinon} \end{cases}$$

On a donc

$$\psi(u,v) = (\varepsilon, \varepsilon) \text{ si et seulement si } u = v \tag{3.1}$$

A partir de l'application ψ on définit l'*action de retard* relative à Δ comme $\omega_\Delta : H_\Delta \times (\Delta^* \times \Delta^*) \to H_\Delta$ telle que

1. $\forall (f,g) \in H_\Delta \backslash \{\mathbf{0}\} : \omega_\Delta((f,g),(u,v)) = \psi(fu, gv)$
2. $\omega_\Delta(\mathbf{0},(u,v)) = \mathbf{0}$.

Cette action de $(\Delta^* \times \Delta^*)$ sur H_Δ dit à quel point la première composante u est en avance ou en retard sur la seconde v ou si u et v ne sont pas préfixes d'un mot commun.

3.3 Décider la fonctionnalité

Il est également possible de définir le carré cartésien d'un transducteur. Par définition, le produit cartésien d'un transducteur $\mathscr{T} = (\mathscr{A}, \Omega)$ de Σ^* à Δ^* avec lui-même est le transducteur $\mathscr{T} \times \mathscr{T}$ de Σ^* à $\Delta^* \times \Delta^*$:

$$\mathscr{T} \times \mathscr{T} = (\mathscr{A}^2, \Omega \otimes \Omega)$$

avec

$$\Omega \otimes \Omega : \delta_{\mathscr{A}^2} \to \Delta^* \times \Delta^* : ((p,r), a, (q,s)) \mapsto (\Omega(p,a,q), \Omega(r,a,s))$$

où $\delta_{\mathscr{A}^2}$ est la relation de transition de $\mathscr{A} \times \mathscr{A}$ qui est l'automate sous-jacent d'entrée du carré du transducteur. Si la partie émondée de $\mathscr{A}^2$ est réduite à sa diagonale alors $\mathscr{A}$ est non-ambigu et $\mathscr{T}$ est fonctionnel.

Il est cependant possible qu'un transducteur réalisant une transduction fonctionnelle possède un automate sous-jacent d'entrée ambigu comme celui de la figure 3.2. Dans ce cas, il faut trouver une propriété sur les sorties nécessaire à la fonctionnalité. Dans l'exemple de la figure 3.2, pour l'entrée aaa il existe deux chemins réussis distincts

$$\rho' := q_0 \xrightarrow{a/x} q_1 \xrightarrow{a/x^4} q_2 \xrightarrow{a/x} q_3$$

et

$$\rho'' := q_0 \xrightarrow{a/x^3} q_2 \xrightarrow{a/x} q_3 \xrightarrow{a/x^2} q_0$$

Ce transducteur possède donc bien un automate sous-jacent d'entrée ambigu. La concaténation des étiquettes de sortie de ρ' et ρ'' est cependant identique. Naturellement, pour que le transducteur soit fonctionnel, cela doit être le cas de tous les chemins acceptant le même mot d'entrée. Il est aisé de se convaincre de la fonctionnalité du transducteur à la figure 3.2.

Pour qu'un transducteur soit fonctionnel, il est nécessaire que toute sortie finale pour une même entrée soit identique. L'intuition pour les transducteurs possédant un automate sous-jacent d'entrée ambigu est que la sortie peut être produite à une «vitesse» différente selon le chemin emprunté. Par exemple, la lecture de a dans le chemin ρ' produit x en sortie alors que dans le chemin ρ'' elle produit x^3. Le chemin ρ'' est donc en avance sur ρ'. Ensuite, la lecture de aa produit x^5 sur

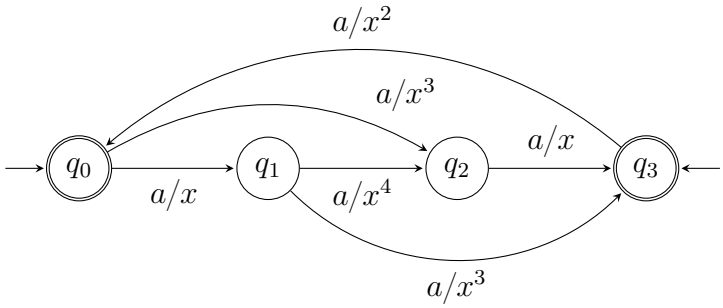


FIGURE 3.2 – ([BCPS03]) Transducteur fonctionnel possédant un automate sous-jacent d'entrée ambigu.

le chemin ρ' et x^4 sur le chemin ρ'' qui est maintenant en retard. Enfin, après la lecture de aaa, les deux chemins ont produit la même sortie x^6.
Il est nécessaire qu'à chaque étape du calcul, une sortie produite pour une certaine entrée soit *comparable* avec toutes les sorties produites sur la même entrée. Soient u un mot accepté par l'automate sous-jacent d'entrée d'un transducteur fonctionnel $\mathscr{T}$, v', v'' deux images produites par $\mathscr{T}$ après la lecture d'un mot $u' \prec u$. On a nécessairement que v' et v'' sont comparables, c'est-à-dire $v' \prec v''$, $v'' \prec v'$ ou $v' = v''$ et naturellement toutes les sorties pour une même entrée doivent être identiques. L'action de retard permet de vérifier cela.

Un transducteur $\mathscr{T} \times \mathscr{T}$ de Σ^* à $(\Delta^* \times \Delta^*)$ peut être vu comme un automate sur le monoïde $M = \Sigma^* \times (\Delta^* \times \Delta^*)$ qu'il est possible de multiplier par l'action de retard $(H_\Delta, M, \omega_\Delta)$.

Théorème 7 ([BCPS03]). *Un transducteur $\mathscr{T}$ de Σ^* à Δ^* est fonctionnel si et seulement si le produit de la partie émondée $\mathscr{U}$ du carré cartésien $\mathscr{T} \times \mathscr{T}$ par l'action de retard ω_Δ avec $(\varepsilon, \varepsilon)$ comme élément particulier de H_Δ, est une valuation de $\mathscr{U}$ telle que la valeur de chaque état final est $(\varepsilon, \varepsilon)$.*

Démonstration. **La condition est suffisante** Soient v la valuation définie par le produit de $\mathscr{U}$ par ω_Δ, ρ' et ρ'' deux chemins réussis distincts de $\mathscr{T}$:

$$\rho' := q'_0 \xrightarrow{a_1/u'_1} q'_1 \xrightarrow{a'_2/u'_2} \dots \xrightarrow{a_n/u'_n} q'_n$$

et

$$\rho'' := q''_0 \xrightarrow{a_1/u''_1} q''_1 \xrightarrow{a_2/u''_2} \dots \xrightarrow{a_n/u''_n} q''_n$$

On a $v(q'_0, q''_0) = (\varepsilon, \varepsilon)$ et $\omega_\Delta(v(q'_0, q''_0), (u'_1 \dots u'_i, u''_1 \dots u''_i)) = v(q'_i, q''_i)$ pour tout i et donc $\omega_\Delta(v(q'_0, q''_0), (u'_1 \dots u'_n, u''_1 \dots u''_n)) = v(q'_n, q''_n) = (\varepsilon, \varepsilon)$ puisque (q'_n, q''_n) est un état final de $\mathscr{T} \times \mathscr{T}$.
De plus, par (3.1) on a que $u'_1 \dots u'_n = u''_1 \dots u''_n$ et donc que $\mathscr{T}$ est fonctionnel.

La condition est nécessaire Deux cas sont possibles :

1. Le produit de $\mathscr{U}$ et ω_Δ est une valuation mais il existe un état final (r, r') de $\mathscr{U} \times \omega_\Delta$ dont la valeur diffère de $(\varepsilon, \varepsilon)$. C'est-à-dire qu'il existe un chemin réussi :
$$(i', i'') \xrightarrow[\mathscr{T}\times\mathscr{T}]{f/(u',u'')} (r', r'')$$
avec
$$\omega_\Delta((\varepsilon, \varepsilon), (u', u'')) \neq (\varepsilon, \varepsilon).$$
Dès lors, par (3.1) on a $u' \neq u''$ et $\mathscr{T}$ n'est pas fonctionnel.
2. Le produit de $\mathscr{U}$ et ω_Δ n'est pas une valuation, c'est-à-dire qu'il existe deux chemins réussis :
$$(i', i'') \xrightarrow[\mathscr{T}\times\mathscr{T}]{f_1/(u_1',u_1'')} (p', p'') \xrightarrow[\mathscr{T}\times\mathscr{T}]{f_2/(u_2',u_2'')} (r', r'')$$
et
$$(j', j'') \xrightarrow[\mathscr{T}\times\mathscr{T}]{g_1/(v1',v_1'')} (p', p'') \xrightarrow[\mathscr{T}\times\mathscr{T}]{f_2/(u_2',u_2'')} (r', r'')$$
avec
$$u = \omega_\Delta(\varepsilon, \varepsilon), (u_1', u_1'') \neq \omega_\Delta((\varepsilon, \varepsilon), (v_1', v_1'')) = v.$$
Dès lors les deux égalités $u_1'u_2' = u_1''u_2''$ et $v_1'u_2' = u_1''v_2''$ ne peuvent être vraies en même temps donc $\mathscr{T}$ n'est pas fonctionnel.

□

La figure 3.3 montre le produit du carré cartésien d'un transducteur fonctionnel $\mathscr{T}$ de $\{a\}^*$ à $\{x\}^*$ par l'action de retard et illustre le théorème 7. Puisque dans ce cas $\Delta = \{x\}$ ne possède qu'une lettre, le monoïde de transition de l'action de retard est isomorphe à $\mathbb{Z}$. En effet, puisque $|\Delta| = 1$, tous les mots de Δ^* sont forcément comparables et $\mathbf{0} \notin H_\Delta$, on peut donc considérer les applications :

$$h : H_\Delta \to \mathbb{Z} : (u, v) \mapsto \begin{cases} |u| \text{ si } v = \varepsilon, \\ -|v| \text{ si } u = \varepsilon, \\ 0 \text{ si } u = v = \varepsilon \end{cases}$$

et

$$h^{-1} : \mathbb{Z} \to H_\Delta : n \mapsto \begin{cases} (x^n, \varepsilon) \text{ si } n > 0, \\ (\varepsilon, x^n) \text{ si } n < 0, \\ (\varepsilon, \varepsilon) \text{ si } n = 0 \end{cases}$$

Il est aisé de se convaincre que h et h^{-1} sont inverse l'une de l'autre. Et soient $x \in \Delta$ et $n > m \in \mathbb{Z}$ (le cas $n < m$ est symétrique), on a bien

$$h(\omega_\Delta((x^n, \varepsilon), (\varepsilon, x^m))) = h((x^n, \varepsilon)) - h((\varepsilon, x^m)) = n - m, \text{ et } h(\varepsilon, \varepsilon) = 0$$

ainsi que

$$h^{-1}(n-m) = \omega_\Delta(h(n), h(m)) = (x^{n-m}, \varepsilon), \text{ et } h^{-1}(0) = (\varepsilon, \varepsilon).$$

Dès lors, h et h^{-1} sont bien des isomorphismes et H_Δ est isomorphe à $\mathbb{Z}$. Il est donc possible d'identifier la valeur des états et d'étiqueter les transitions du transducteur produit par $\mathbb{Z}$ plutôt que respectivement par H_Δ et $\Delta^* \times \Delta^*$. De plus, l'entrée étant toujours a, les transitions peuvent être représentées sans ambiguïté par leur sortie uniquement, et une sortie de la forme (x^n, x^m) peut être codée comme $n-m$. Afin de ne pas surcharger d'avantage la figure, l'étiquette $n-m$ d'une transition est décrite par la nature de la flèche la représentant : en pointillés pour 0, une simple flèche pour 1, une grosse flèche pour 2, une double flèche pour 3 et des flèches discontinues pour les valeurs négatives correspondantes.

La section 5.5 détaille la procédure proposée par Béal et al. pour décider la fonctionnalité en temps polynomial [BCPS03] et en propose une implémentation.

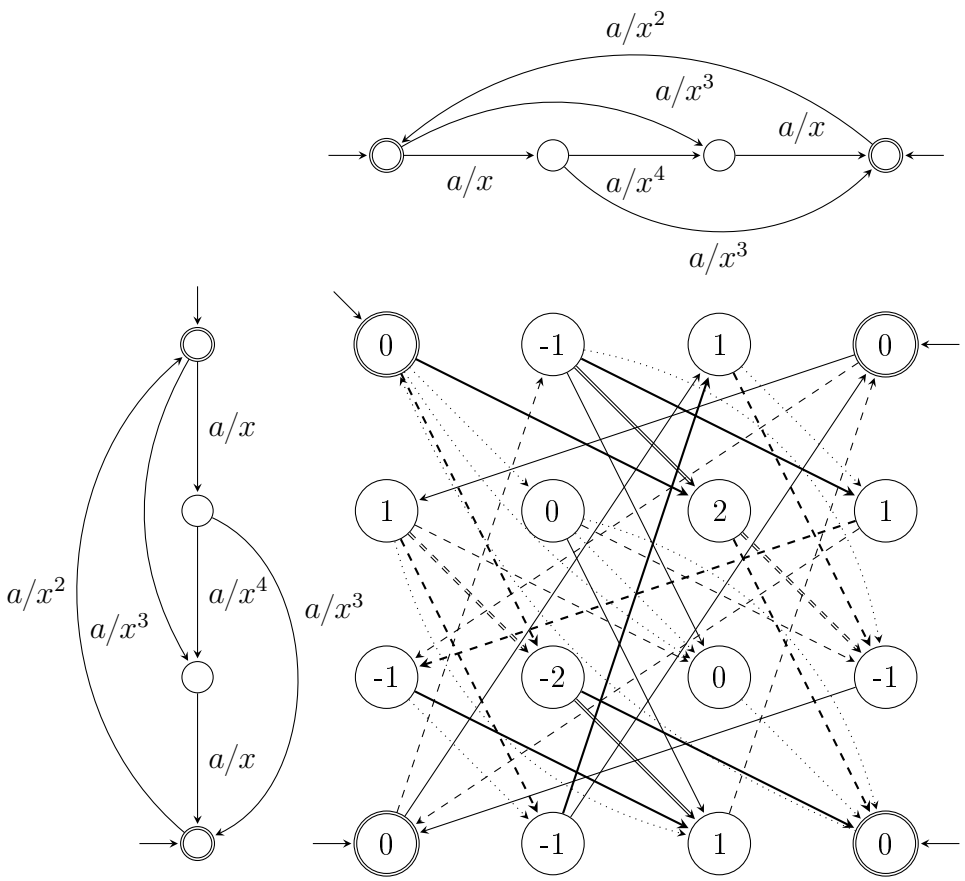


FIGURE 3.3 – ([BCPS03]) Produit du carré cartésien du transducteur de la figure 3.2 par l'action de retard ω_Δ

Chapitre 4

Transducteurs sous-séquentiels

Choffrut a présenté [Cho77] en 1977 deux caractérisations des fonctions sous-séquentielles ; la première se base sur la fonction sous-séquentielle elle-même alors que la seconde se base sur une propriété d'un transducteur la réalisant. En prouvant l'équivalence de ces caractérisations, il a prouvé qu'il était décidable si un transducteur non sous-séquentiel réalise une fonction sous-séquentielle. Par la suite, Béal et al. ont proposé [BCPS03] une nouvelle caractérisation basée sur le produit du carré cartésien du transducteur et de l'action de retard, et vérifiable en temps polynomial.

4.1 Fonction uniformément divergente

Pour rappel, la distance entre deux mots u et v est $||u, v|| = |u| + |v| - 2|u \wedge v|$. Dès lors, si $u = hu'$ et $v = hv'$ avec $h = u \wedge v$ on a clairement

$$||u, v|| = |u'| + |v'|. \tag{4.1}$$

Définition 11. *Une fonction partielle $f : \Sigma^* \to \Delta^*$ est* uniformément divergente [1] *si et seulement si pour tout entier n, il existe un entier N plus grand que la distance entre les images par f de deux mots dont la distance est plus petite que n :*

$$\forall n \in \mathbb{N}, \exists N \in \mathbb{N}, \forall u, v \in Dom(f) \quad ||u, v|| \leqslant n \Rightarrow ||f(u), f(v)|| \leqslant N.$$

C'est-à-dire que le ratio des distances entre des points et leur images doit rester borné dans le domaine de la fonction lorsque la distance devient arbitrairement

1. Le terme *fonction à variation bornée* est aussi souvent utilisé.

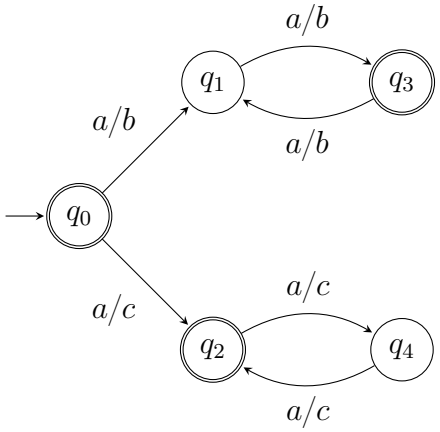


FIGURE 4.1 – Transducteur réalisant la fonction de l'exemple 3

grande.

Théorème 8 ([Cho77]). *Une fonction rationnelle est sous-séquentielle si et seulement si elle est uniformément divergente.*

Exemple 3 ([BC02]). *La relation* $R = \{(u, b^n) \mid u \in \{a\}^* \wedge |u| = n \text{ est pair}\} \cup \{(u, c^n) \mid u \in \{a\}^* \wedge |u| = n \text{ est impair}\}$ *réalisée par le transducteur de la figure 4.1 est la fonction* $f : \{a\}^* \to \{b, c\}^*$ *qui associe* a^n *à* b^n *si* n *est pair et* a^n *à* c^n *si* n *est impair. Cette fonction n'est pas sous-séquentielle, en effet, elle n'est pas uniformément divergente puisque que pour un entier* n *on a*

$$||a^{2n}, a^{2n+1}|| = 1 \text{ alors que } ||b^{2n}, c^{2n+1}|| = 4n + 1.$$

4.2 Propriété de jumelage

Définition 12. *Deux mots* u *et* v *sont dits* conjugués *s'il existe un mot* t *tel que* $ut = tv$.

La condition de jumelage est une propriété sur les états d'un transducteur et par extension, sur le transducteur entier. Deux états q et q' d'un transducteur sont *jumelés* si et seulement si pour chaque paire de chemins

$$i \xrightarrow{u/u_1} q \xrightarrow{v/v_1} q,$$

$$i' \xrightarrow{u/u_2} q' \xrightarrow{v/v_2} q',$$

où i et i' sont des états initiaux, alors soit les sorties des cycles sont vides, soit la sortie d'un des chemins avant le cycle est préfixe de la sortie de l'autre chemin avant le cycle et les sorties des deux cycles sont conjuguées par le mot restant après avoir retiré le préfixe commun des u_x (le préfixe commun étant la plus petite des deux sorties). Plus rigoureusement :

1. $v_1 = v_2 = \varepsilon$ ou
2. Il existe un mot fini w tel que
   (a) $u_2 = u_1 w$ et $v_1 w = w v_2$ ou
   (b) $u_1 = u_2 w$ et $v_2 w = w v_1$.

De plus, de $wv_2 = v_1 w$ et $wv_1 = v_2 w$, il vient naturellement que $|v_1| = |v_2|$. Puisque le transducteur est forcément fonctionnel, les deux chemins doivent donner la même sortie pour la même entrée, c'est-à-dire $u_1 v_1^r = u_2 v_2^r$. Dès lors, le cas 2 est équivalent aux deux conditions suivantes

1. $|v_1| = |v_2|$,
2. $u_1 v_1^r = u_2 v_2^r$.

On dit qu'un transducteur $\mathscr{T}$ satisfait la condition de jumelage si toute paire d'états de $\mathscr{T}$ est jumelée.

L'intuition derrière la propriété de jumelage est qu'à la lecture d'une entrée uvw compatible, après uv, avec deux chemins $i \xrightarrow{u,u_1} q \xrightarrow{v/v_1} p$ et $i' \xrightarrow{u,u_2} q' \xrightarrow{v/v_2} p'$ il est nécessaire de stocker le retard $\omega_\Delta(u_1 v_1, u_2 v_2)$ jusqu'à ce que l'ambiguïté puisse être levée. C'est le cas si le mot w suivant n'est compatible qu'avec un seul des deux chemins considérés. Or dans le cas d'un cycle, c'est-à-dire si $q = p$ et $q' = p'$, le retard $\omega_\Delta(u_1 v_1^r, u_2 v_2^r)$ peut croître avec r. Le retard stocké devient alors arbitrairement grand, ce qui nécessiterait une mémoire infinie.

Proposition 4 (Choffrut)**.** *Soit $f : \Sigma^* \to \Delta^*$ une fonction réalisée par un transducteur $\mathscr{T}$. Les trois propositions suivantes sont équivalentes :*

— *La fonction f est sous-séquentielle.*
— *La fonction f est uniformément divergente.*
— *$\mathscr{T}$ satisfait la condition de jumelage.*

Exemple 4. *Il est maintenant possible de montrer que le NFT de la figure 2.5 ne réalise pas une fonction sous-séquentielle et qu'il n'est donc pas déterminisable. En effet, il y a deux chemins*

$$q_0 \xrightarrow{x/a} q_1 \xrightarrow{yy/ba} q_1,$$

$$q_0 \xrightarrow{x/b} q_2 \xrightarrow{yy/ba} q_2,$$

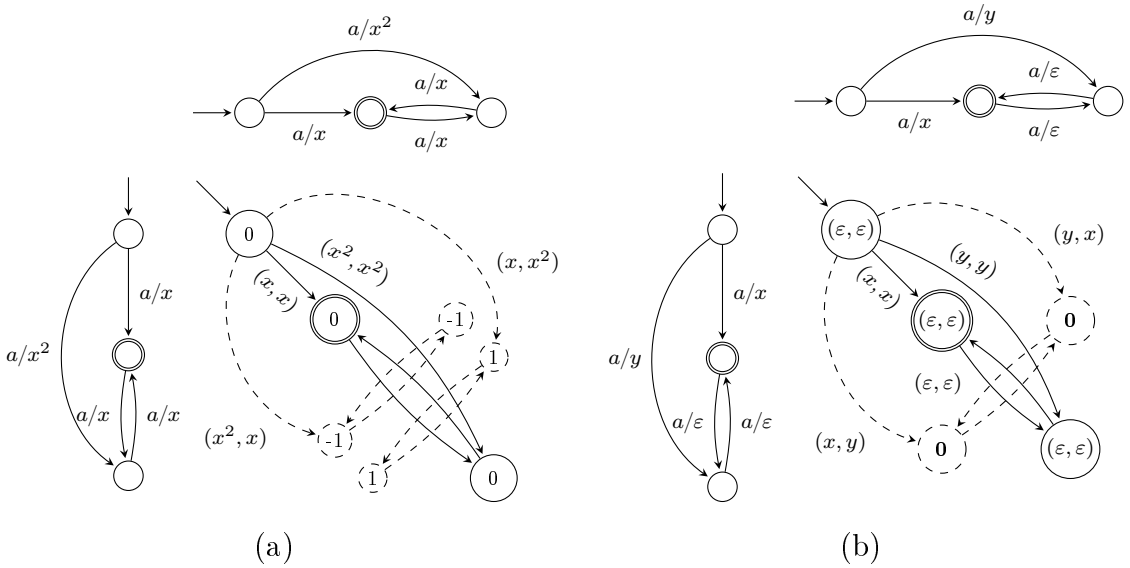


FIGURE 4.2 – Deux transducteurs réalisant des fonctions sous-séquentielles. [BCPS03]

et bien qu'on ait $|ba| = |ba|$, clairement $a(ba)^r \neq b(ba)^r$. Le transducteur ne satisfait pas la condition de jumelage et la fonction qu'il réalise n'est donc pas sous-séquentielle.

4.3 Décider la sous-séquentialité

Plus récemment, Béal et al. ont présenté [BCPS03] une procédure de décision pour la sous-séquentialité basée sur le produit du carré du transducteur par l'action de retard. Cette caractérisation des transducteurs sous-séquentiels est très proche de la première formulation de la condition de jumelage.

Théorème 9 ([BCPS03])**.** *Un transducteur fonctionnel émondé $\mathscr{T} = (\mathscr{A} = (\Sigma, Q, I, F, \delta), \Omega : \delta \to \Delta^*)$ réalise une transduction sous-séquentielle si et seulement si le produit de la partie accessible $\mathscr{V}$ de $\mathscr{T} \times \mathscr{T}$ par ω_Δ a les deux propriétés suivantes :*

1. *Il est fini*
2. *Si un état a la valeur* **0** *dans $\mathscr{V} \times \omega_\Delta$ appartient à un cycle de $\mathscr{V}$, alors l'étiquette de ce cycle est $(\varepsilon, \varepsilon)$.*

La figure 4.2 montre deux cas où la fonction réalisée par les transducteurs est

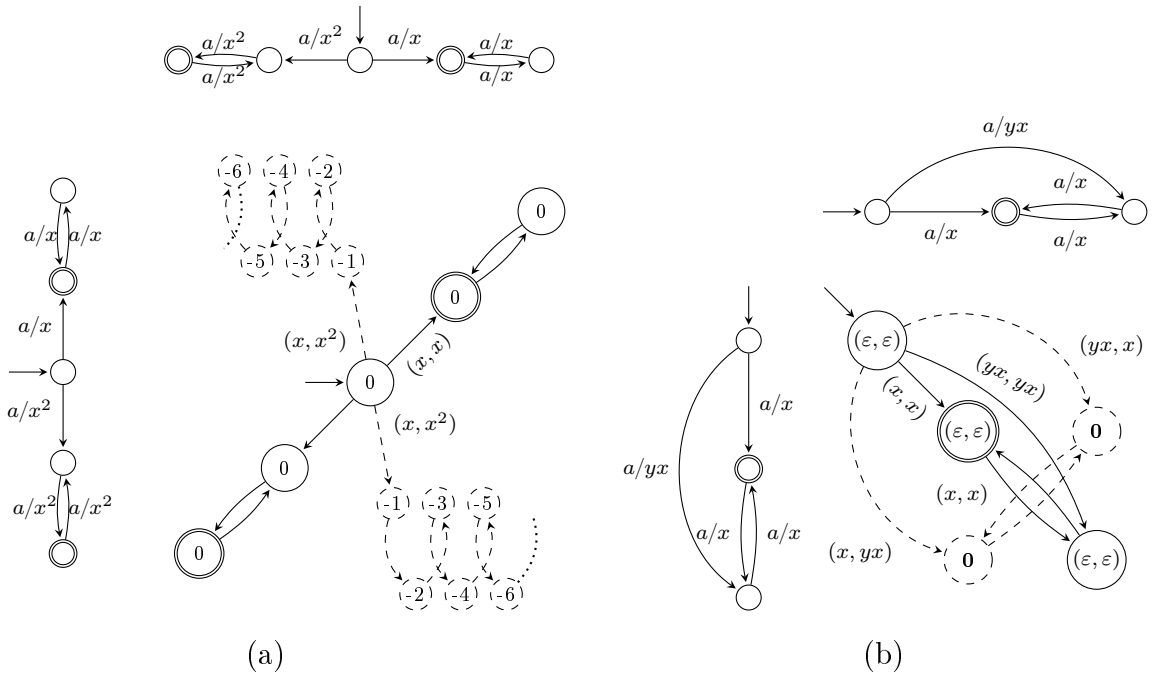


FIGURE 4.3 – Deux transducteurs réalisant des fonctions qui ne sont pas sous-séquentielles. [BCPS03]

sous-séquentielle. Dans le cas (a), la fonction est sous-séquentielle car le produit est fini et aucun état possède la valeur **0**. D'ailleurs la valeur est identifiée par $\mathbb{Z}$ puisque l'alphabet de sortie ne contient qu'un seul symbole. Dans le cas (b), la partie accessible du produit est également finie et la valeur **0** n'apparaît que dans des états appartenant à un cycle d'étiquette $(\varepsilon, \varepsilon)$.

La figure 4.3 montre quant à elle deux cas où la fonction réalisée par les transducteurs n'est pas sous-séquentielle. Dans le cas (a) car la partie accessible du produit est infinie. Dans le cas (b) car il existe dans la partie accessible du produit deux états de valeur **0** appartenant à un cycle qui est étiqueté par $(x^2, x^2) \neq (\varepsilon, \varepsilon)$.

Le lemme suivant est nécessaire pour prouver le théorème 9. Il permet une reformulation de la condition de jumelage grâce à l'action de retard.

Lemme 3 ([BCPS03]). *Soient* $(\varepsilon, w) \in H_\Delta \backslash \mathbf{0}$ *et* $(v_1, v_2) \in \Delta^* \times \Delta^* \backslash (\varepsilon, \varepsilon)$. *Alors l'ensemble* $X = \{\omega_\Delta((\varepsilon, w), (v_1, v_2)^r) \mid r \in \mathbb{N}\}$ *est fini et ne contient pas* **0** *si et seulement si* v_1 *et* v_2 *sont conjugués par un mot* t *et* $w = v_1^k t$ *pour un certain* k. *Si cette condition est vérifiée,* X *est un singleton.*

Démonstration. Si la condition est vérifiée, on a

$$\begin{aligned}
\omega_\Delta((\varepsilon,z),(v_1,v_2)) &= (\varepsilon, v_1^{-1}(wv_2)) \\
&= (\varepsilon, v_1^{-1}(v_1^k t v_2)) && \text{car } w = v_1^k t \\
&= (\varepsilon, v_1^{k-1} t v_2) \\
&= (\varepsilon, v_1^{k-1} v_1 t) && \text{car } v_1 t = t v_2 \\
&= (\varepsilon, w) && \text{car } v_1^{k-1} v_1 t = v_1^k t = w
\end{aligned}$$

A l'inverse, si $\mathbf{0} \notin X$, c'est-à-dire que $\omega_\Delta((\varepsilon,w),(v_1,v_2)^r) \neq \mathbf{0}$, alors on est forcément dans un des cas suivants :

1. $u = \varepsilon$; ou
2. $v = \varepsilon$ et w est préfixe d'une puissance de u ; ou
3. w est préfixe d'une puissance de v_1, c'est-à-dire $w = v_1^k t$ où t est un préfixe de v_1, et il doit exister un l tel que v_1^k est conjugué à v_2^l par t.

Il est clair que pour que X soit un singleton, il suffit de prendre $k = l$. □

La preuve du théorème 9 donnée par Béal et al. [BCPS03] est présentée ici. Il suffit de prouver que les conditions du théorème 9 sont vérifiées si et seulement si la fonction f réalisée par un transducteur $\mathscr{T}$ est uniformément divergente.

Démonstration du théorème 9. **Les conditions sont suffisantes**. Soient K une borne sur la longueur des sorties de $\mathscr{T}$ et L une borne sur la longueur des valeurs des états dans le produit $\mathscr{V} \times \omega_\Delta$. Soient $m, n \in Dom(f)$, on pose $h = m \wedge n$ donc $m = hm'$ et $n = hn'$. Il existe deux chemins réussis dans $\mathscr{T}$

$$i \xrightarrow{h/u} p \xrightarrow{m'/u'} t \text{ et } j \xrightarrow{h/v} q \xrightarrow{n'/v'} s.$$

Dès lors, il existe un chemin

$$(i,j) \xrightarrow{h/(u,v)} (p,q)$$

dans $\mathscr{V}$. Deux cas sont alors possibles :

Cas 1 : $\omega_\Delta((\varepsilon,\varepsilon),(u,v)) \neq \mathbf{0}$, alors

$$\begin{aligned}
||f(m), f(n)|| &= ||uu', vv'|| \\
&= |uu'| + |vv'| - 2|uu' \wedge vv'| \\
&= (|u| + |v| - 2|uu' \wedge vv'|) + |u'| + |v'| \\
&\leqslant L + |u'| + |v'| \\
&\leqslant L + K(|m'| + |n'|) \\
&= L + K||m, n|| && \text{par (4.1)}
\end{aligned}$$

Dans ce cas, f est bien uniformément divergente puisque L et K sont des constantes.

Cas 2 : $\omega_\Delta((\varepsilon,\varepsilon),(u,v)) = \mathbf{0}$, alors u et v ne sont pas comparables mais il est possible de décomposer $h = h_1 a h_2 h_3$ avec $a \in \Sigma$ et $h_1, h_2, h_3 \in \Sigma^*$ de telle sorte que la première partie des sorties soit comparable. Le chemin $(i,j) \xrightarrow{h/(u,v)} (p,q)$ devient alors

$$(i,j) \xrightarrow{h_1/(u_1,v_1)} (p_1,q_1) \xrightarrow{a/(x,y)} (p_2,q_2) \xrightarrow{h_2/(\varepsilon,\varepsilon)} (p_3,q_3) \xrightarrow{h_3/(u_2,v_2)} (p,q),$$

où la valeur de (p_1,q_1) est différente de $\mathbf{0}$, la valeur de (p_2,q_2) est égale à $\mathbf{0}$ et (u_2,v_2) est différent de $(\varepsilon,\varepsilon)$ si h_3 est différent de ε. Comme chaque état après (p_2,q_2) a une valeur de $\mathbf{0}$ et que par hypothèse la condition est vérifiée, le chemin

$$(p_3,q_3) \xrightarrow{h_3/(u_2,v_2)} (p,q)$$

ne peut pas contenir de cycle et sa longueur est donc bornée par le nombre d'états $|Q|^2$. On a alors

$$\begin{aligned} ||f(m),f(n)|| &= ||u_1 x u_2 u', v_1 y v_2 v'|| \\ &\leqslant L + (|u_2| + |v_2|) + (|u'| + |v'|) \\ &\leqslant L + K(|Q|^2+1) + K(|m'| + |n'|) \\ &= L + K(|Q|^2+1) + K||m,n|| \qquad \text{par (4.1)} \end{aligned}$$

Dans ce cas également la fonction f est uniformément divergente. Les conditions sont donc bien suffisantes.

Les conditions sont nécessaires. Deux cas sont possibles :

Cas 1 : il existe un cycle dans $\mathscr{V} \times \omega_\Delta$ dont tous les états ont la valeur $\mathbf{0}$ et dont l'étiquette est différente de $(\varepsilon,\varepsilon)$. Dans $\mathscr{V}$, il y a un chemin

$$(i,j) \xrightarrow{h_1/(u_1,v_1)} (p,q) \xrightarrow{h_2/(u_2,v_2)} (p,q)$$

tel que $\omega_\Delta((\varepsilon,\varepsilon),(u_1,v_1)) = \mathbf{0}$. Ce qui implique pour la distance

$$\begin{aligned} ||f(h_1 h_2^r m'), f(h_1 h_2^r n')|| &= ||u_1 u_2^r u', v_1 v_2^r, v'|| \\ &\geqslant r(|u_2| + |v_2|) + |u'| + |v'| \end{aligned}$$

peut être rendue arbitrairement grande avec r. Dans ce cas, f n'est pas uniformément divergente.

Cas 2 : le produit $\mathscr{V} \times \omega_\Delta$ est infini. Il existe donc au moins un chemin dans $\mathscr{V}$

$$(i,j) \xrightarrow{h_1/(u_1,v_1)} (p,q) \xrightarrow{h_2/(u_2,v_2)} (p,q)$$

qui devient un graphe infini dans $\mathscr{V} \times \omega_\Delta$. C'est-à-dire que

$$\omega_\Delta((\varepsilon, \varepsilon), (u_1, v_1)) = (x, y) \neq \mathbf{0}$$

et

$$\forall r \in \mathbb{N}\ \omega_\Delta((x, y), (u_2, v_2)^r) \neq \mathbf{0}.$$

Grâce au lemme 3 on a que $|u_2| \neq |v_2|$ et il existe un l tel que

$$|\omega_\Delta((x, y), (u_2, v_2)^r)| \geqslant (r - l)|(|u_2| - |v_2|)$$

et donc

$$\begin{aligned} ||f(h_1 h_2^r m'), f(h_1 h_2^r n')|| &= ||u_1 u_2^r u', v_1 v_2^r v'|| \\ &\geqslant [(r - l)|(|u_2| - |v_2|)|] - |(|u'| - |v'|)| \end{aligned}$$

qui peut être rendu arbitrairement grand. Dans ce cas, f n'est pas uniformément divergente. Les conditions sont donc bien nécessaires.

□

La section 5.6 détaille la procédure proposée par Béal et al. pour décider la sous-séquentialité en temps polynomial [BCPS03] et en propose une implémentation. La section 5.7 donne également une procédure exponentielle pour déterminiser un transducteur réalisant une fonction sous-séquentielle, c'est-à-dire créer un transducteur sous-séquentiel équivalent.

Chapitre 5

Implémentation des transducteurs

Ce chapitre détaille l'implémentation des transducteurs et des algorithmes introduits aux chapitres 3 et 4. La section 5.1 présente les structures de données utilisées, c'est-à-dire comment les modèles d'automates et transducteurs sont représentés en mémoire principale. La manière dont ils sont représentés en mémoire secondaire est discutée à la section 5.2. Les sections suivantes sont respectivement consacrées aux procédures d'émondage, de carré de transducteur, de décision de la fonctionnalité et de la sous-séquentialité, et de déterminisation.

Ce chapitre s'articule autour des différentes étapes d'un programme, fourni avec le code source en annexe B, qui prend en entrée un transducteur, teste sa fonctionnalité, sa sous-séquentialité et produit un automate sous-séquentiel équivalent le cas échéant. Il est régulièrement fait référence au transducteur $\mathscr{T}_1$ de la figure 5.1 afin d'illustrer le travail du programme à chaque étape.

Afin de rendre les pseudo-codes de ce chapitre plus lisibles, l'accès à une variable d'une instance peut se faire grâce au point (.) plutôt que par l'appel à une méthode «getter». Ainsi $t_a.dest.id$ est équivalent à $t_a.getDestination().getId()$ et fait référence à l'identifiant du nœud destination de la transition t_a.

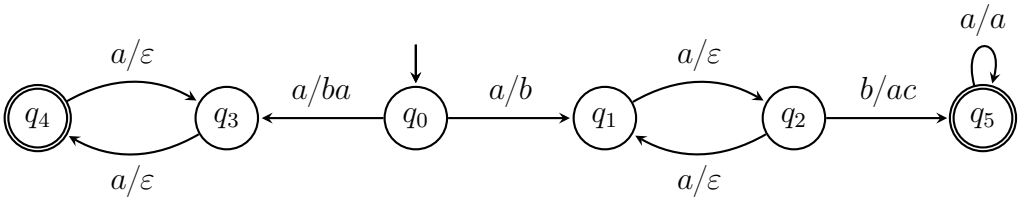


FIGURE 5.1 – NFT $\mathscr{T}_1$ inspiré d'un exemple de [BC02]

5.1 Structures de données

L'implémentation des transducteurs est ici proposée en JAVA car il s'agit d'un langage de programmation populaire et portable. De plus, le paradigme orienté objet permet facilement de représenter les relations entre les modèles (comme l'automate sous-jacent d'un transducteur par exemple) et autorise également via l'héritage une extensibilité aisée. Le polymorphisme est aussi très utile pour la généralisation des procédures. Nous décrivons ici les structures de données visibles dans le diagramme de classe à la figure 5.2, la classe **State** étant trop grande pour la figure, elle est détaillée à la figure 5.3. Les méthodes de type «getters» et «setters» sont volontairement omises par soucis de clarté.

La structure de base des automates et des transducteurs présentés ici est un graphe orienté.

Graph Le graphe est composé d'un tableau *states* [1] contenant tous les nœuds du graphe et d'un entier *size* qui est la taille de ce tableau, c'est-à-dire le nombre de nœuds du graphe. Il possède également une méthode *addState* prenant un nœud en paramètre et le plaçant dans le tableau *states* à la bonne place.

State Un nœud est caractérisé par un $id \in [0, size[$ unique (au sein du graphe) qui donne la place du nœud dans le tableau *states* du graphe auquel il appartient, tout en servant de discriminant. Un nœud possède également deux listes de transitions pour ses transitions entrantes et sortantes. Un entier *mark* permettant de marquer un état comme non-exploré, en cours d'exploration ou totalement exploré.

Transition Les transitions du graphe sont caractérisées par leur origine et leur destination.

Un automate étant un graphe dont certains nœuds (maintenant appelés états) peuvent être initiaux et/ou finaux et dont les transitions sont étiquetées par des symboles. Cela est concrétisé par l'héritage de graph à automata [2].

Automaton Un automate hérite des graphes le tableau d'états. Il possède également une méthode *trim*() qui émonde l'automate.

AState Un état d'automate peut être initial et/ou final, il possède donc deux variables booléennes à cet effet. Les champs d'instance *accessible* et *coaccessible* sont évidents et servent lors de l'émondage détaillé à la section 5.3.

1. *states* étant l'implémentation de Q, *states* et Q feront référence à la même structure dans les algorithmes

2. Le terme Automata fait ici référence au package complet

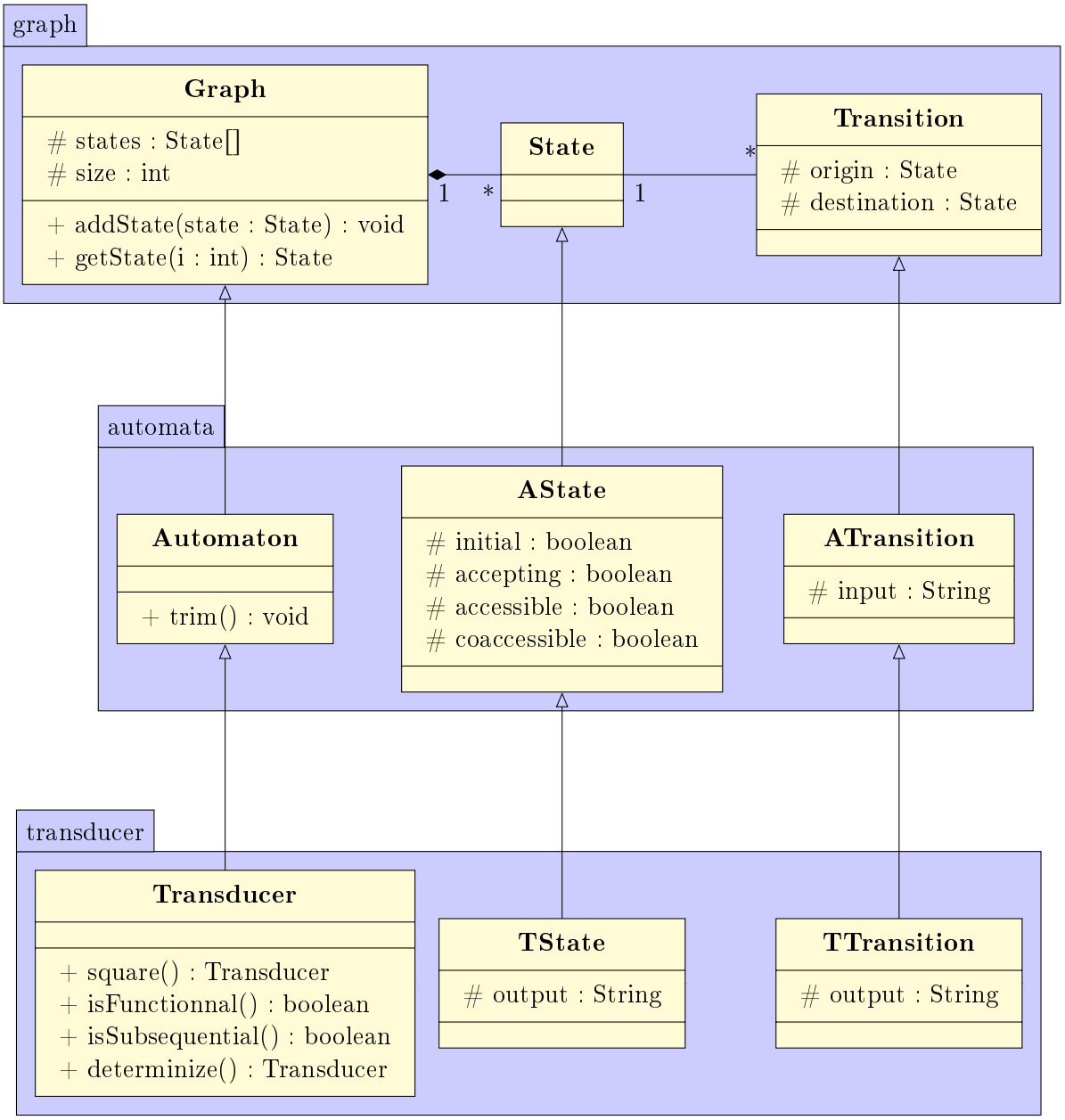


FIGURE 5.2 – Diagramme de classe

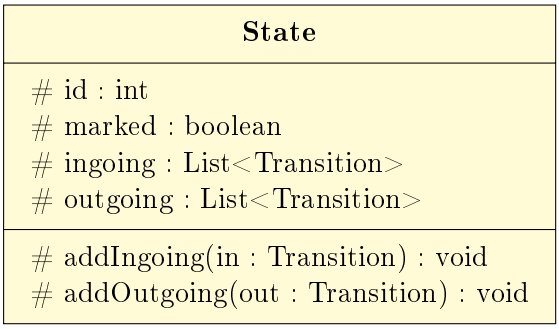


FIGURE 5.3 – Classe **State**

ATransition Les transitions d'un automate héritent leur origine et leur destination de celles du graphe et leur ajoutent une chaîne de caractère *input* représentant l'entrée de la transition.

Un transducteur possède un automate sous-jacent d'entrée dont les étiquettes sont «augmentées» d'une sortie. Cela est également concrétisé par l'héritage de automata à transducer.

Transducer Un transducteur hérite de l'automate le tableau d'états et la méthode d'émondage tout en ajoutant une méthode *square*() retournant le carré cartésien du transducteur détaillée à la section 5.4, une méthode *isFunctionnal*() décidant la fonctionnalité du transducteur et détaillée à la section 5.5, une méthode *isSubsequential*() décidant la sous-séquentialité et détaillée à la section 5.6 et une méthode *determinize*() qui retourne un transducteur sous-séquentiel équivalent et détaillée à la section 5.7.

TState Un état d'un transducteur peut produire une sortie, c'est le cas pour un transducteur sous-séquentiel, il possède donc une chaîne de caractère *output* à cet effet.

TTransition Les transitions d'un transducteur héritent leur origine, leur destination et leur entrée de celles de l'automate et leur ajoutent une chaîne de caractère *output* représentant la sortie de la transition.

5.2 Format de fichiers

Cette section détaille la manière dont les modèles d'automates et de transducteurs sont représentés en mémoire secondaire, c'est-à-dire le format de fichier contenant toutes les informations nécessaires à la description du modèle.

Le XML est choisi comme langage de description car il est facilement lisible par un être humain tout en étant très bien supporté par les langages de programmations dont le JAVA avec l'implémentation de l'API SAX. De plus, la nature extensible de XML permet à ce format d'évoluer en fonction des besoins et des modèles à décrire. Ainsi, un modèle basé sur un graphe aura une structure telle que présentée au listing 5.1 et devra être compatible avec le schéma XML présenté en annexe A. Le schéma XML vérifie la syntaxe du fichier d'entrée, il est bien sûr nécessaire qu'il ait également du sens au niveau sémantique. Ainsi, les nœuds doivent avoir un *id* compris entre 0 et *size*, le nombre de nœuds dans la balise $< nodes >$ doit être égal à l'attribut *size* de la structure et les attributs $from$ et *to* des transitions doivent faire référence à des nœuds.

```
<?xml version="1.0" encoding="UTF-8"?>

<transducer size="6"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="schema.xsd">

 <nodes>
  <node id="0" initial="true" />
  <node id="1" />
  <node id="2" />
  <node id="3" />
  <node id="4" accepting="true" />
  <node id="5" accepting="true" />
 </nodes>

 <transitions>
  <transition from="0" to="1" input="a" output="b" />
  <transition from="0" to="3" input="a" output="ba" />
  <transition from="1" to="2" input="a" output="" />
  <transition from="2" to="1" input="a" output="" />
  <transition from="2" to="5" input="b" output="ac" />
  <transition from="3" to="4" input="a" output="" />
  <transition from="4" to="3" input="a" output="" />
  <transition from="5" to="5" input="a" output="a" />
 </transitions>
</transducer>
```

Listing 5.1 – Fichier XML pour l'exemple de la figure 5.1

5.3 Émondage

L'algorithme pour l'émondage d'un automate est assez trivial ; il consiste en une lecture du graphe de l'automate depuis les états initiaux afin de marquer les états accessibles et une lecture inverse depuis les états finaux afin de marquer les états co-accessibles. Les états utiles sont alors les états qui ont été marqués lors des deux parcours. Une possibilité aurait été de créer un automate émondé équivalent lors du second parcours en copiant les états utiles et les transitions entre ces états. Cependant, puisque cette méthode implique une copie des états et des transitions, elle dépend de l'entrée sur laquelle l'algorithme est appliqué et il est nécessaire de redéfinir l'émondage pour les automates, les produits d'automates sur Δ^*, sur $(\Delta^* \times \Delta^*)$ (comme c'est le cas pour le carré des automates ou les transducteurs), sur $\Sigma^* \times (\Delta^* \times \Delta^*)$ (comme c'est le cas pour le carré des transducteurs) ou sur toute extension impliquant une modification des structures de données.
Puisque par définition l'émondage ne dépend que du sens des transitions et pas de l'étiquette de celles-ci, il est préférable de définir l'émondage pour les automates de manière suffisamment générale pour que l'algorithme soit applicable à tous les modèles possédant un automate sous-jacent, quelque soit le monoïde sur lequel il travaille. La procédure transforme donc l'objet sur laquelle elle est appliquée en supprimant les états et transitions non utiles. La structure en graphe étant commune à tous les modèles héritant des automates, cette procédure est générale. Cette généralisation vient donc au prix d'un calcul supplémentaire peu coûteux.

L'algorithme 1 détaille la procédure présentée ici. Le pseudo-code étant assez long, il est divisé en deux parties logiques.

La première partie se divise également en deux étapes. La première effectue un parcours en profondeur afin de marquer tous les états accessibles. Tous les états et toutes les transitions sont donc visités et, pour un automate $\mathscr{A} = (\Sigma, Q, I, F, \delta)$, la complexité en temps dans le pire des cas, c'est-à-dire lorsque tous les états sont accessibles, est $O(|Q| + |\delta|)$.

La seconde étape est un nouveau parcours en profondeur, inverse cette fois, qui permet de marquer les états co-accessibles parmi les états accessibles. Ce parcours inverse est rendu possible grâce au stockage des transitions entrantes. Les états ainsi marqués sont les états utiles et un compteur est utilisé pour comptabiliser leur nombre. La complexité en temps de cette partie est également $O(|Q| + |\delta|)$.

Cette dernière partie est le calcul supplémentaire engendré par la généralisation. Un nouveau tableau Q' est créé pour stocker les états utiles. Ce tableau est rempli en parcourant Q et en copiant la référence de chaque état utile dans Q' (l'*id* des états est mis à jour pour correspondre à leur nouvelle position). La complexité

Algorithme 1 Émondage d'un automate - Lecture des états utiles

Entrée: calcul sur $\mathscr{A} = (\Sigma, Q, I, F, \delta)$
Sortie: $-$ $\mathscr{A}$ est émondé

```
    // Lecture des états accessibles
 1: Soit P une pile
 2: pour chaque état s de 𝒜 faire
 3:     si s est initial alors
 4:         P.empiler(s)
 5:     fin si
 6: fin pour

 7: tant que P non vide faire
 8:     courant ← P.depiler()
 9:     Marquer courant comme accessible
10:     pour chaque Transition sortante t de courant faire
11:         si t.dest non accessible alors
12:             P.empiler(t.dest)
13:         fin si
14:     fin pour
15: fin tant que

    // Lecture des états co-accessibles
16: pour chaque état s de 𝒜 faire
17:     si s est final alors
18:         P.empiler(s)
19:     fin si
20: fin pour

21: Soit utiles un compteur
22: tant que P non vide faire
23:     courant ← P.depiler()
24:     Marquer courant comme co-accessible
25:     si courant n'a pas encore été compté alors
26:         utiles ← utiles + 1
27:     fin si
28:     pour chaque Transition entrante t de courant faire
29:         si t.origin non co-accessible alors
30:             P.empiler(t.origin)
31:         fin si
32:     fin pour
33: fin tant que
```

Algorithme 1 Émondage d'un automate - Suppression des états et transitions inutiles

Créer un tableau $usefulStates$ de taille $utiles$ // Q'
Soit j un compteur
pour chaque état s de $\mathscr{A}$ **faire**
    si s est utile **alors**
        $s.id \leftarrow j$
        $usefulStates[j] \leftarrow s$
        $j \leftarrow j + 1$
    fin si
fin pour
pour chaque état s dans $usefulStates$ **faire**
    pour chaque Transition entrante t de s **faire**
        si $t.origin$ non utile **alors**
            supprimer t
        fin si
    fin pour
    pour chaque Transition sortante t de s **faire**
        si $t.dest$ non utile **alors**
            supprimer t
        fin si
    fin pour
fin pour
Remplacer les état de $\mathscr{A}$ par $usefulStates$

est donc toujours $O(|Q|)$.
Ensuite, les transitions de chaque états de Q' sont parcourues et une transition est supprimée si elle ne mène pas à un état utile. Dans le pire des cas, c'est-à-dire que l'automate est déjà émondé, tous les états et transitions sont parcourus, cette étape revient à un nouveau parcours complet de l'automate, donc en $O(|Q| + |\delta|)$.

Au total, l'émondage est bien linéaire en la taille de l'automate d'entrée.

5.4 Carré

Pour rappel, le carré cartésien d'un transducteur $\mathscr{T} = (\mathscr{A}, \Omega)$ de Σ^* à Δ^* est le transducteur $\mathscr{T} \times \mathscr{T}$ de Σ^* à $\Delta^* \times \Delta^*$:

$$\mathscr{T} \times \mathscr{T} = (\mathscr{A}^2, \Omega \otimes \Omega)$$

avec

$$\Omega \otimes \Omega : \delta_{\mathscr{A}^2} \to \Delta^* \times \Delta^* : ((p, r), a, (q, s)) \mapsto (\Omega(p, a, q), \Omega(r, a, s))$$

où $\delta_{\mathscr{A}^2}$ est la relation de transition de $\mathscr{A} \times \mathscr{A}$ qui est l'automate sous-jacent d'entrée du carré du transducteur.

Il convient donc d'introduire de nouvelles structures pour supporter les nouveaux états de $\mathscr{T}^2$ composés d'une paire d'états et les nouvelles transitions dont la sortie est étiquetées dans $\Delta^* \times \Delta^*$. La figure 5.5 montre comment les structures de données de base sont étendues pour représenter le produit de transducteurs. Une *transition produite* hérite des transitions d'automates tout en leur ajoutant une sortie dans $\Delta^* \times \Delta^*$. Un *état produit* de $\mathscr{T}^2$ contient une paire d'états de $\mathscr{T}$ et une variable *valeur* permettant de simuler le produit par ω_Δ, l'état produit est initial,resp. final, si et seulement si les deux états de la paire sont initiaux, resp. finaux. Un *transducteur produit* possède une méthode *computeV()* qui calcule le sous-automate $\mathscr{V}'$ tel que décrit à la section 5.6.

Il existe une bijection entre les paires d'états de $\mathscr{T}$ et les états de $\mathscr{T}^2$. Il est donc possible d'identifier sans ambiguïté un état de $\mathscr{T}^2$ grâce à ses deux sous-états. Pour ce faire, les états du transducteur produit sont stockés dans un tableau de taille $|Q|^2$ de telle sorte que l'état (q_i, q_j) se trouve à l'index donné par la fonction de hachage :

$$hash(i, j) = i \times |Q| + j.$$

Le pseudo-code pour le calcul du carré d'un transducteur est donné par l'algorithme 2. La figure 5.4 montre le carré du transducteur de la figure 5.1. Seule la partie accessible nous intéresse par la suite et seule cette partie est représentée

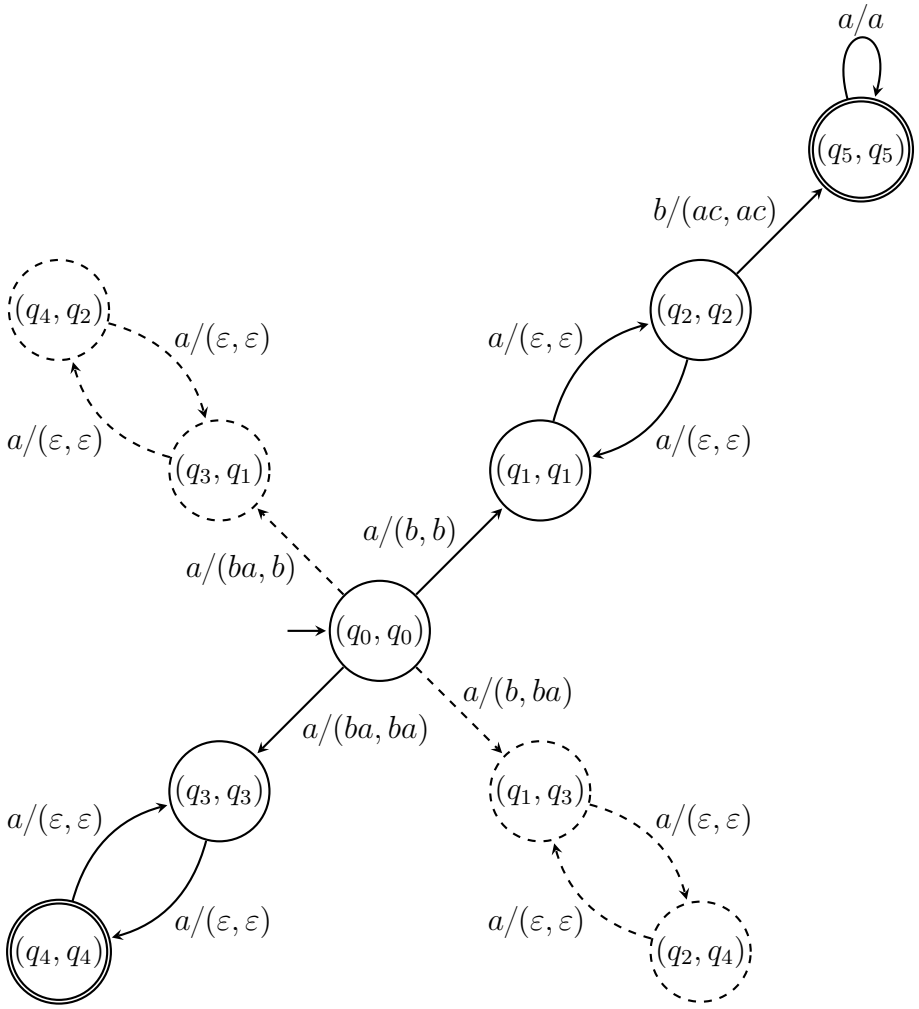


FIGURE 5.4 – Partie accessible de $\mathscr{T}_1 \times \mathscr{T}_1$

sur la figure. La partie accessible mais non co-accessible est en traits discontinus. Il est intéressant de noter que la partie émondée du carré est égale à la diagonale du carré. Ce qui prouve que l'automate sous-jacent est non-ambigu et que le transducteur est fonctionnel.

Le calcul des états de $\mathscr{T}^2$ se fait trivialement en $O(|Q|^2)$. Pour le calcul des transitions de $\mathscr{T}^2$, à chaque transition, on considère toutes les autres transitions donc la complexité est $O(|\delta|^2)$. Au total, le calcul du carré de fait en $O(|Q|^2 + |\delta|^2)$.

Algorithme 2 Calcul du carré d'un transducteur

Entrée: $\mathscr{T} = (\mathscr{A} = (\Sigma, Q, I, F, \delta), \Omega)$
Sortie: $\mathscr{T} \times \mathscr{T}$

1: $output \leftarrow new$ Transducer($|Q|^2$)

// Calcul de $Q \times Q$
2: **pour** $i \leftarrow 1$ **à** $|Q|$ **faire**
3: **pour** $j \leftarrow 1$ **à** $|Q|$ **faire**
4: $newID \leftarrow hash(states[i].id, states[j].id)$
5: $output.addState(newState(newID, states[i], states[j]))$
6: **fin pour**
7: **fin pour**

// Calcul des transitions
8: **pour** $i \leftarrow 1$ **à** $|Q|^2$ **faire**
9: $origin \leftarrow output.getState(i)$
10: **pour chaque** Transition sortante t_a de a **faire**
11: **pour chaque** Transition sortante t_b de b **faire**
12: **si** $t_a.input = t_b.input$ **alors**
13: $destId = hash(t_a.dest.id, t_b.dest.id)$
14: $dest \leftarrow output.getState(destId)$
15: Créer une transition $origin \xrightarrow{t_a.input/(t_a.output, t_b.output)} dest$
16: **fin si**
17: **fin pour**
18: **fin pour**
19: **fin pour**
retourner *output*

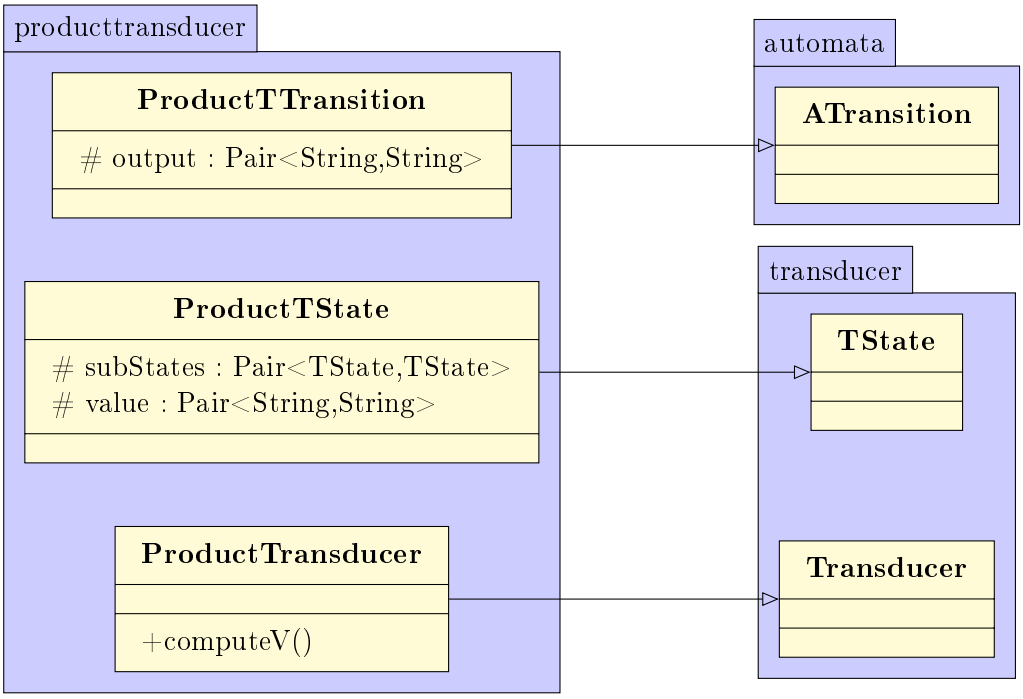


FIGURE 5.5 – Extension pour le produit de transducteurs

5.5 Fonctionnalité

Cette section présente l'algorithme proposé par Béal et al. [BCPS03] pour tester la fonctionnalité d'un NFT. Cet algorithme est basé sur le théorème 7 et est donc une preuve du théorème 5.

La multiplication par l'action ω_Δ est implémentée grâce à la variable *value* de la classe **ProductTState** qui sert à stocker la valeur des états de $\mathscr{U} \times \omega_\Delta$. Dans un premier temps, il est nécessaire de calculer $\mathscr{U}$, la partie émondée de $\mathscr{T} \times \mathscr{T}$. Tous les états initiaux de $\mathscr{U}$ reçoivent $(\varepsilon, \varepsilon)$ comme valeur. Ensuite, un parcours en profondeur est effectué et lorsqu'une transition $((p', p''), (u, v), (q', q''))$ où l'état (p', p'') a la valeur (f, g) est considérée, il y a trois cas possibles :

1. Si (q', q'') n'a pas encore été visité, alors on lui assigne la valeur $\omega_\Delta((f, g), (u, v))$.
2. Si (q', q'') a déjà été visité et sa valeur est différente de $\omega_\Delta((f, g), (u, v))$, alors l'algorithme s'arrête car $\mathscr{U} \times \omega_\Delta$ n'est pas une valuation de $\mathscr{U}$. Si (q', q'') est final alors sa valeur doit être égale à $(\varepsilon, \varepsilon)$ sinon $\mathscr{T}$ n'est pas fonctionnel et l'algorithme s'arrête.
3. Si (q', q'') a déjà été visité et que sa valeur est égale à $\omega_\Delta((f, g), (u, v))$, alors l'algorithme continue et considère une autre transition jusqu'à ce que

le point 2 soit rencontré ou que toutes les transitions aient été considérées.

Dans le cas où toutes les transitions ont été considérées, on a donc que $\mathscr{U} \times \omega_\Delta$ est une valuation et chaque état final a la valeur $(\varepsilon, \varepsilon)$, $\mathscr{T}$ est donc fonctionnel.

Pour rappel, la taille d'un automate est définie par $|Q| + |\delta|$. La taille d'une transition $p \xrightarrow{u/v} q$ d'un transducteur est la longueur de $|uv|$. On note K la taille de la plus longue transition. La somme de la taille de toutes les transitions de $\mathscr{T}$ est notée $\lceil \mathscr{T} \rceil$ et est bornée par $K|\delta|$. Le calcul du carré se fait en $O(|Q|^2 + |\delta|^2)$ et son émondage $\mathscr{U}$ est linéaire en sa taille donc également $O(|Q|^2 + |\delta|^2)$. Puisque chaque transition du carré possède deux mots de sortie, $\lceil \mathscr{U} \rceil$ est borné par $2K|\delta|^2$. Le calcul de la valeur d'un état de $\mathscr{U} \times \omega_\Delta$ peut être borné par $\lceil \mathscr{U} \rceil$.

La complexité totale est donc $O((|Q|^2 + |\delta|^2) \times \lceil \mathscr{U} \rceil)$.

La figure 5.6 montre le produit de la partie émondée $\mathscr{U}_1$ du carré de la figure 5.4 par l'action de retard. Les états ont été renommés par de simples lettres pour plus de lisibilité. Ainsi, l'état $(5, (\varepsilon, \varepsilon))$ est l'état (q_5, q_5) possédant la valeur $(\varepsilon, \varepsilon)$. Trivialement, le produit est bien une valuation et les états finaux ont bien la valeur $(\varepsilon, \varepsilon)$.

5.6 Sous-séquentialité

Cette section présente l'algorithme proposé par Béal et al. [BCPS03] pour tester la sous-séquentialité d'un NFT fonctionnel.

Soient $\mathscr{T} = (\mathscr{A} = (\Sigma, Q, I, F, \delta), \Omega : \delta \to \Delta^*)$ un transducteur et K la taille de la plus longue transition de $\mathscr{T}$. Le calcul de la partie accessible $\mathscr{V}$ de $\mathscr{T} \times \mathscr{T}$ se fait de la même manière que le premier parcours dans l'émonde, donc en $O(|Q| + |\delta|)$. Cependant, seule une partie de $\mathscr{V}$ est intéressante pour l'algorithme. En effet, au vu des conditions énoncées dans le théorème 9, on voit qu'il est possible de se limiter aux états qui sont co-accessibles à un cycle dont l'étiquette est différente de $(\varepsilon, \varepsilon)$. En effet, un cycle étiqueté par $(\varepsilon, \varepsilon)$ ne permet pas de faire croître arbitrairement le retard devant être stocké avant de lever l'ambiguïté, ce qui est l'intuition qui permet de caractériser les transducteurs non-déterminisables.

Soit $\mathscr{V}'$ le sous-automate de $\mathscr{V}$ qui ne contient que les états co-accessibles à un cycle dont la sortie est différente de $(\varepsilon, \varepsilon)$. Le calcul de $\mathscr{V}'$ se fait grâce à un parcours en profondeur dans lequel on stocke pour chaque état la sortie produite jusque-là. Un état est marqué comme en cours d'exploration quand il a déjà été

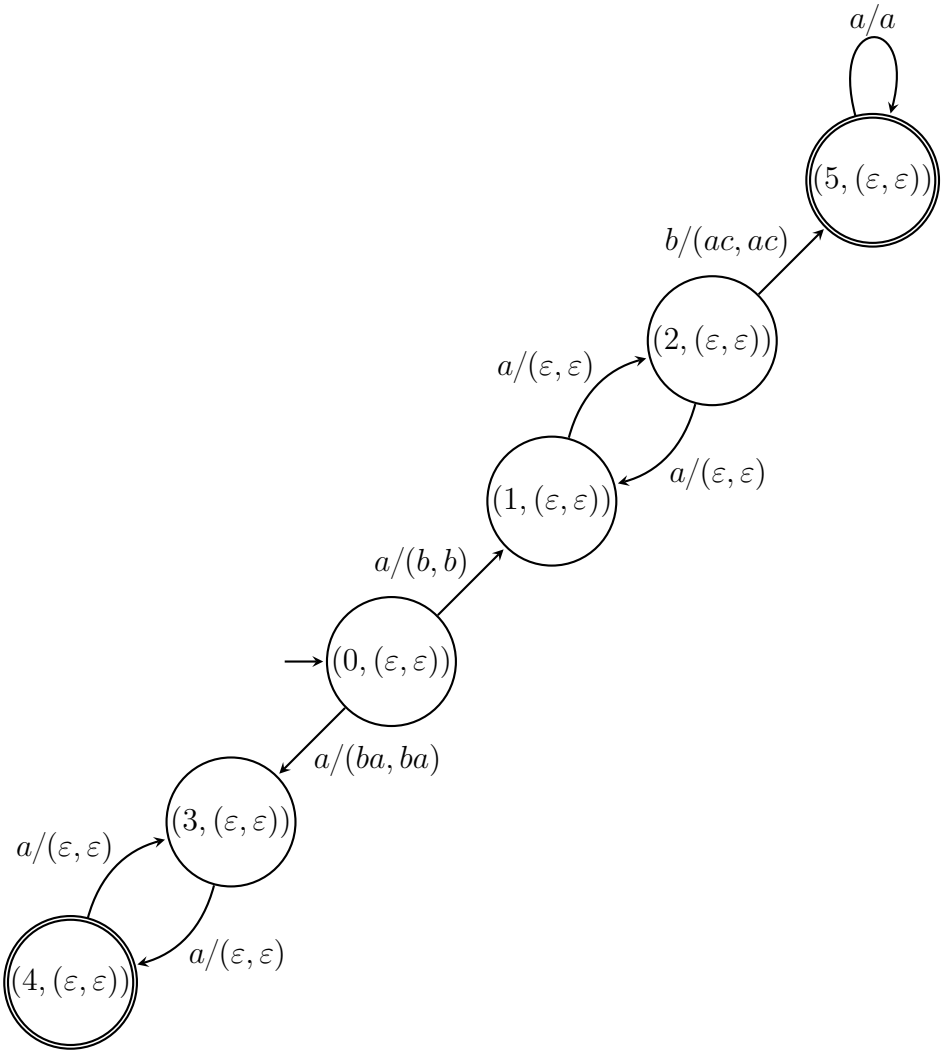


FIGURE 5.6 – Produit de $\mathscr{U}_1$ par l'action de retard

Algorithme 3 Teste la fonctionnalité d'un transducteur

Entrée: calcul sur $\mathscr{T} = (\mathscr{A}, \Omega)$
Sortie: **vrai** si $\mathscr{T}$ est fonctionne, **faux** sinon

```
1: 𝒰 = 𝒯 × 𝒯
2: 𝒰.trim()
3: Soit P une pile
4: pour chaque état s de 𝒰 faire
5:     si s est final alors
6:         Marquer s
7:         Donner la valeur (ε, ε) à s
8:         P.empiler(s)
9:     fin si
10: fin pour

11: tant que P non vide faire
12:     courant ← P.depiler()
13:     pour chaque Transition sortante t de courant faire
14:         Soit valeur une paire de mots
15:         valeur ← courant.getValue() + t.output
16:         Retirer le lcp des deux composantes de valeur
17:         suivant ← t.destination
18:         si suivant pas encore marqué alors
19:             Marquer suivant
20:             suivant.setValue(valeur)
21:             P.empiler(suivant)
22:         sinon si suivant.value ≠ valeur alors
23:             retourner faux
24:         sinon si (suivant est final et suivant.value ≠ (ε, ε) alors
25:             retourner faux
26:         fin si
          retourner vrai
27:     fin pour
28: fin tant que
```

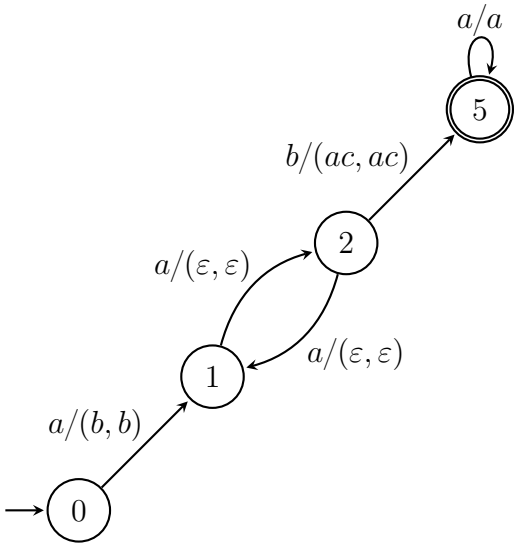


FIGURE 5.7 – Calcul de $\mathscr{V}'$ pour $\mathscr{T}_1 \times \mathscr{T}_1$

visité mais que tous ses successeurs n'ont pas encore été visités. Lorsque le parcours rencontre un état en cours d'exploration, un cycle est détecté et la différence entre la sortie produite lors de la première visite et la sortie produite lors de la seconde visite (après le cycle) permet de vérifier si le cycle est étiqueté par $(\varepsilon, \varepsilon)$. Lorsqu'un cycle non-vide est détecté, l'état est marqué comme co-accessible. Une fois le parcours en profondeur terminé, un second parcours, inverse et à partir des états marqués comme co-accessibles, est effectué pour marquer tous les états accessibles qui sont également co-accessibles à un cycle non-vide. Deux parcours en profondeur sont effectués et dans le pire des cas, la procédure est linéaire en la taille du carré : $O(|Q|^2 + |\delta|^2)$.

La figure 5.7 montre le calcul de $\mathscr{V}'$ pour $\mathscr{T}_1 \times \mathscr{T}_1$. Le seul cycle non-vide dans la partie accessible du carré est celui de l'état 5. Les états co-accessibles à un cycle non-vide sont donc les états $0, 1, 2, 5$.

Lemme 4 ([BCPS03])**.** *Si* $((p, q), (\varepsilon, w))$ *et* $((p, q), (\varepsilon, w'))$ *sont deux états de* $\mathscr{V}' \times \omega_\Delta$ *alors* w *et* w' *sont comparables ou la condition 2 du théorème 9 n'est pas respectée.*

Démonstration. Puisque (p, q) est dans $\mathscr{V}'$, il est co-accessible à un état (r, s) appartenant à un cycle étiqueté par $(u, v) \neq (\varepsilon, \varepsilon)$, c'est-à-dire qu'il existe un chemin $(p, q) \xrightarrow{f/(x,y)} (r, s)$ dans $\mathscr{V}'$. Si w et w' ne sont pas comparables, alors au moins un des ensembles

$$X = \{\omega_\Delta(\omega_\Delta((\varepsilon, w), (x, y)), (u, v)^n) \mid n \in \mathbb{N}\}$$

et

$$X' = \{\omega_\Delta(\omega_\Delta((\varepsilon, w'), (x, y)), (u, v)^n) \mid n \in \mathbb{N}\}$$

contient $\mathbf{0}$ et l'état $((r, s), \mathbf{0})$ est dans $\mathscr{V}' \times \omega_\Delta$

□

Lemme 5 ([BCPS03]). *Si $((p, q), (\varepsilon, w))$ est un état de $\mathscr{V}' \times \omega_\Delta$ alors $|w| \leqslant K|Q|^2$ ou les conditions du théorème 9 ne sont pas respectées.*

Démonstration. Le plus court chemin d'un état initial $((i, j), (\varepsilon, \varepsilon))$ à un état $((p, q), (\varepsilon, w))$ dans $\mathscr{V}' \times \omega_\Delta$ a une longueur inférieure à $|Q|^2$. Sinon, il contient obligatoirement un cycle et il est possible de le décomposer en

$$((i, j), (\varepsilon, \varepsilon)) \xrightarrow{f_1/(u_1,v_1)} ((r, s), h_1) \xrightarrow{f_2/(u_2,v_2)} ((r, s), h_2) \xrightarrow{f_3/(u_3,v_3)} ((p, q), (\varepsilon, w))$$

. Par le lemme 3, l'ensemble $X = \{\omega_\Delta(h_1, (u_2, v_2)^n) \mid n \in \mathbb{N}\}$ doit être un singleton et donc $h_1 = h_2$, il existe donc un chemin plus court.

Dès lors, la longueur de w est bornée par $K|Q|^2$.

□

Contrairement à l'algorithme de décision pour la fonctionnalité, on ne s'intéresse pas à la partie émondée mais à la partie accessible de $\mathscr{T} \times \mathscr{T}$, de plus, il n'est plus nécessaire que le produit par l'action de retard ω_Δ soit une valuation. Il faut donc stocker plusieurs valeurs pour chaque état mais il suffit de stocker la valeur la plus longue[3] de la forme (ε, w) et la valeur la plus longue de la forme (w, ε). En effet, dans la figure 4.2(a), les quatre états non co-accessibles sont en fait deux états de $\mathscr{V}$ qui reçoivent chacun deux valeurs (-1 et 1) lors du produit par l'action de retard et entre lesquels il y a un cycle qui ne fait pas croître ces valeurs. De la même manière, les boucles infinies de la figure 4.3(a) sont chacune un état pour lequel le produit par l'action de retard associe plusieurs valeurs, de plus en plus grandes.

Le calcul de $\mathscr{V}' \times \omega_\Delta$ se fait donc à l'aide de deux tableaux T_1 et T_2 indexés par $|Q|^2$ et stockant respectivement pour chaque état le w de la valeur la plus longue (ε, w) et de la valeur la plus longue de la forme (w, ε). Pour chaque transition $(p, q) \xrightarrow{a/(u,v)} (p', q')$ de $\mathscr{V}'$, on calcule $h' = \omega_\Delta(T_1[(p, q)], (u, v))$, resp. $h' = \omega_\Delta(T_2[(p, q)], (u, v))$:

1. Si $h' = \mathbf{0}$ alors la condition 2 du théorème 9 n'est pas vérifiée et l'algorithme s'arrête.

3. Comprendre w le plus long.

2. Si h' est de la forme (ε, w), resp. (w, ε), alors on regarde si w et $T_1[p', q']$, resp. $T_2[p', q']$, sont comparables. Vient alors deux possibilités :
   (a) S'ils ne sont pas comparables, alors le lemme 4 dit que les conditions du théorème 9 ne sont pas satisfaites et l'algorithme s'arrête.
   (b) S'ils sont comparables, alors on met à jour $T_1[p', q']$, resp. $T_2[p', q']$, avec le plus long des deux mots.
3. Si $h' = T_1[p', q']$, resp. $T_2[p', q']$ alors le cycle n'a pas fait évolué la valeur de (p', q') et il n'est pas nécessaire de continuer le parcours à partir de cet état puisqu'il a déjà été fait.
4. Si $h' = (\varepsilon, w)$, resp. (w, ε), et que $|w| > K|Q|^2$ alors le lemme 5 dit que les conditions du théorème 9 ne sont pas satisfaites et l'algorithme s'arrête.

L'algorithme s'arrête toujours, soit parce qu'une condition d'arrêt est rencontrée et que $\mathscr{T}$ n'est pas sous-séquentiel, soit parce que tous les états de $\mathscr{V}'$ ont été visités et qu'aucun cycle ne fait croître la valeur des états qui le forment, dans ce cas $\mathscr{T}$ est sous-séquentiel.

L'algorithme effectue donc un parcours complet de $\mathscr{V}'$ et peut même visiter un état plusieurs fois si sa valeur croît. Le nombre total d'états visités est donc borné par $2|Q|^2 \times K|Q|^2 = 2K|Q|^4$, c'est-à-dire deux tableaux de taille $|Q|^2$ dont les éléments peuvent être mis à jour au plus $K|Q|^2$ fois. Le nombre total de transitions visitées est borné par $|\delta|^2 \times 2K|Q|^2$, c'est-à-dire toutes les transitions de $\mathscr{V}'$ (au pire $|\delta|^2$) pour lesquelles on peut calculer au plus $2K|Q|^2$ mises à jour de valeur. Puisque vérifier si deux mots sont comparables est borné par $K|Q|^2$, le temps total pour la procédure peut être borné par $2K|Q|^4|\delta|^2$.

L'algorithme 4 montre le pseudo-code pour cette procédure. On peut également voir dans la figure 5.6 en ne gardant que les états de $\mathscr{V}'$ que $\mathscr{T}_1$ réalise bien une transduction sous-séquentielle.

5.7 Déterminisation

Cette section présente la procédure de déterminisation proposée par Béal et Carton [BC02]. Cette procédure s'applique bien sûr aux transducteurs réalisant une fonction sous-séquentielle.

Soit $\mathscr{T} = (\mathscr{A} = (\Sigma, Q, I, F, \delta), \Omega : \delta \to \Delta^*)$ un transducteur temps réel réalisant une fonction sous-séquentielle f. L'algorithme qui suit produit un transducteur sous-séquentiel réalisant f, il est exponentiel en les états de $\mathscr{T}$, ce qui n'est pas surprenant puisque la déterminisation d'un automate est déjà exponentielle.

Algorithme 4 Teste la sous-séquentialité d'un transducteur

Entrée: calcul sur $\mathscr{T} = (\mathscr{A}, \Omega)$
Sortie: **vrai** si $\mathscr{T}$ est sous-séquentiel, **faux** sinon

Soit P une pile
Calculer $\mathscr{T}$
Calculer $\mathscr{V}'$ à partir de $\mathscr{T}$
pour chaque état s de $\mathscr{V}'$ **faire**
    si s est initial **alors**
        Ajoute s à P
    fin si
fin pour

tant que P non vide **faire**
    $courant \leftarrow P.depiler()$
    marquer *courant* comme exploré
    pour chaque transition sortante t de *courant* **faire**
        $suivant \leftarrow t.dest$
        si *suivant* est marqué comme co-accessible [4] **alors**
            $valeur \leftarrow \omega_\Delta(T_1[courant], t.output)$
            si $valeur = \mathbf{0}$ **ou** $valeur > K|Q|^2$ **alors**
                retourner faux
            sinon si *suivant* est non exploré **alors**
                Stocker *valeur* dans $T_1[suivant]$ ou $T_2[suivant]$
                $P.empiler(suivant)$
            sinon si *suivant* est déjà exploré **alors**
                si $\omega_\Delta(T_1[suivant], valeur) = \mathbf{0}$ **alors**
                    retourner faux
                fin si
                si $\omega_\Delta(T_2[suivant], valeur) = \mathbf{0}$ **alors**
                    retourner faux
                fin si
                $T_1[suivant] \leftarrow Max(T_1[suivant], valeur)$
                si le cycle a fait croître $T_1[suivant]$ **alors**
                    $P.empiler(suivant)$
                fin si
            fin si
        fin si
    fin pour
fin tant que

On définit un transducteur sous-séquentiel $\mathscr{D} = (\mathscr{T}', \Omega_f)$ comme suit : un état P de $\mathscr{D}$ est un ensemble de paires (q, w) avec $q \in Q$ un état de $\mathscr{T}$ et $w \in \Delta^*$ un mot sur l'alphabet de sortie. Soit $a \in \Sigma$ un symbole sur l'alphabet d'entrée, la paire (P, a) détermine un ensemble de paires $Q \times \Delta^*$:

$$R = \{(q', wu) \mid \exists (q, w) \in P \wedge q \xrightarrow{a/u} q' \in \delta\}.$$

Si R est vide, alors il n'y a pas de transitions sortant de P avec l'entrée a. Si R est non-vide, alors on pose v comme le plus long préfixe commun des mots wu pour $(q', wu) \in R$ et on crée un état P' dans $\mathscr{D}$ tel que

$$P' = \{(q', w') \mid (q', vw') \in R\}.$$

On crée alors une transition $P \xrightarrow{a/v} P'$ dans $\mathscr{D}$. L'unique état initial de $\mathscr{D}$ est l'ensemble $J = \{(i, \varepsilon) | i \in I\}$ où I est l'ensemble des états initiaux de $\mathscr{T}$. Le lemme suivant caractérise rigoureusement les transitions de $\mathscr{D}$.

Lemme 6 ([BC02]). *Soient $u \in Sigma^*$ un mot fini et $J \xrightarrow{u/v} P$ l'unique chemin de $\mathscr{D}$ sortant de l'état initial J et ayant pour entrée u. Alors l'état P est égal à*

$$P = \{(q, w) \mid \textit{ il existe un chemin } i \xrightarrow{u/vw} q \textit{ dans } \mathscr{T} \textit{ où } i \in I\}.$$

Démonstration. Par induction sur la longueur de u. On considère un chemin de $\mathscr{D}$

$$J \xrightarrow{u/v} P \xrightarrow{a/t} P',$$

où $a \in \Sigma$ est un symbole sur l'alphabet d'entrée. Soit $(q', w') \in P'$. Par la définition des transitions de $\mathscr{D}$, il y a une paire $(q, w) \in P$ et une transition $q \xrightarrow{a/t'} q'$ dans $\mathscr{T}$ telle que $tw' = wt'$.
Par hypothèse d'induction, il existe un chemin $i \xrightarrow{u/vw} q$ dans $\mathscr{T}$ et on a $vtw' = vwt'$.

□

Une conséquence du lemme 6 est que si les deux paires (q, w) et (q', w') appartiennent à un état P accessible depuis un état initial de $mathscrD$ et que q et q' sont des états finaux de $\mathscr{T}$ alors forcément $w = w'$. Dans le cas contraire, $\mathscr{T}$ ne serait pas fonctionnel. On peut alors définir les états finaux de $\mathscr{D}$ comme les états contenant au moins une paire (q, w) telle que q est un état final de $\mathscr{T}$. La fonction Ω_f associe alors la sortie w à l'état final contenant la paire (q, w).

Du lemme 6 et la définition de Ω_f découle la proposition suivante :

Proposition 5 ([BC02]). *Le transducteur sous-séquentiel $\mathscr{D}$ réalise la même fonction f que le transducteur $\mathscr{T}$.*

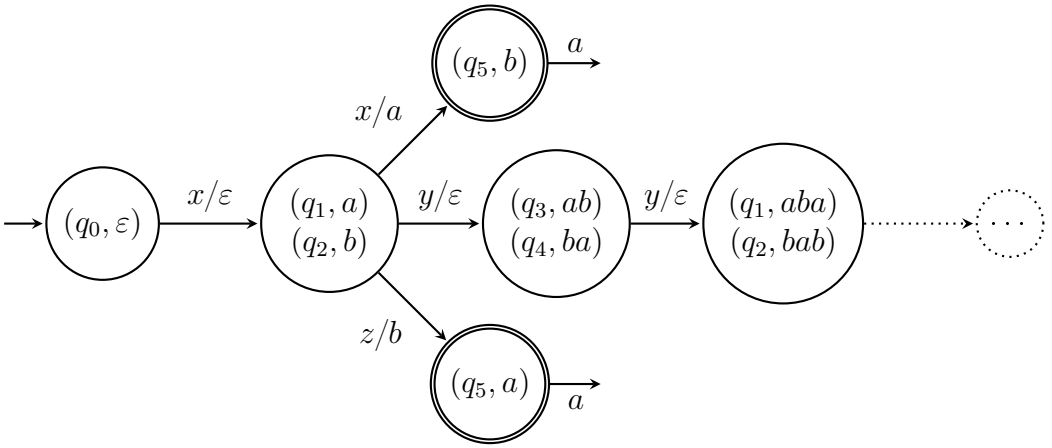


FIGURE 5.8 – Application de la procédure sur l'exemple de la figure 2.5 [AM03]

L'algorithme 5 reprend le pseudo-code pour la déterminisation. Le nombre d'états créés étant exponentiel voire infini dans le cas où la fonction réalisée par le transducteur n'est pas sous-séquentielle, les états sont stockés dans une structure de données dynamique, ici une liste *newStates*, avant d'être recopiés dans le tableau du transducteur sous-séquentiel de sortie. Une nouvelle classe **MetaState** héritant de **TState** est créée. Elle possède un ensemble de paires composées d'un élément **TState** et d'un élément **String**. L'algorithme crée un état *suivant* en calculant l'ensemble R qui lui est associé, ensuite il vérifie si l'état *suivant* a déjà été calculé lors d'un précédent appel, c'est-à-dire qu'il existe un cycle. Si c'est le cas, on crée une transition entre l'état courant et l'état équivalent à *suivant* qui est déjà dans *newStates*. Si ce n'est pas le cas, alors l'état *suivant* est ajouté à *newStates*, une transition entre l'état courant et *suivant* est créée et l'algorithme continue à partir de *suivant*.

La figure 5.8 montre l'application de la procédure de déterminisation sur l'exemple de la figure 2.5 qui ne réalise pas une fonction sous-séquentielle. Dans ce cas, un nombre infini d'états est créé. La figure 5.9 montre l'application de la procédure sur l'exemple de la figure 5.1. Dans ce cas, l'algorithme s'arrête lorsque plus aucun nouvel état n'est créé.

Algorithme 5 Déterminisation d'un transducteur

Entrée: calcul sur $\mathscr{T} = (\mathscr{A}, \Omega)$ réalisant une fonction sous-séquentielle f
Sortie: $\mathscr{D} = (\mathscr{T}', \Omega_f)$ sous-séquentiel réalisant f

```
1: Créer une liste newStates de MetaState
2: Créer un MetaState initial init
3: pour i ← 1 à |Q| faire
4:     si states[i] est initial alors
5:         init.addSsubstate((states[i], ε))
6:     fin si
7: fin pour

8: newStates.add(init)
9: computeNext(newStates, init)

10: pour chaque état s de newStates faire
11:     pour chaque paire (q, w) de s faire
12:         si q est final alors
13:             Marquer s comme final
14:             Donner w comme sortie à s
15:         fin si
16:     fin pour
17: fin pour
```

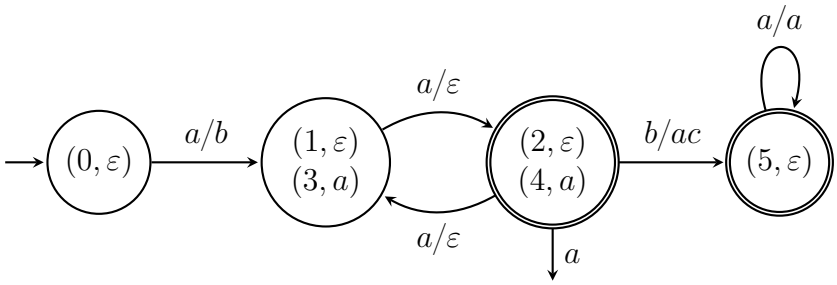


FIGURE 5.9 – Transducteur sous-séquentiel équivalent à $\mathscr{T}_1$

Algorithme 6 *ComputeNext*$(newStates, courant)$

Entrée: Tableau de MetaState *newStates*, MetaState *courant*
Sortie: Calcule l'état suivant

1: Créer une liste *pool* contenant toutes les transitions sortant des sous-états de *courant*
2: **tant que** *pool* est non vide **faire**
3: $input \leftarrow pool[0].input$
4: Créer un MetaState *suivant*
5: **pour chaque** transition t de *pool* **faire**
6: **si** $t.input = input$ **alors**
7: **pour chaque** paire (q, w) de *courant* **faire**
8: **si** $t.origin = q$ **alors**
9: Ajouter $(t.dest, q + t.output)$ à R
10: **fin si**
11: Retirer t de *pool*
12: **fin pour**
13: **fin si**
14: **fin pour**

15: Calculer *lcp* le plus long préfixe commun des w de $(q, w) \in R$
16: $R' \leftarrow \{(q, lcp^{-1}w) \mid (q, w) \in R\}$
17: Donner l'ensemble de paires R' à *suivant*

18: **pour chaque** état *state* de *newStates* **faire**
19: **si** $state = suivant$ **alors**
20: Ajoute *courant* à *newStates*
21: créer une transition $courant \xrightarrow{input/lcp} state$
22: **computeNext**$(newStates, next)$
23: **fin si**
24: **fin pour**
25: **si** *suivant* n'avait pas encore été créé **alors**
26: créer une transition $courant \xrightarrow{input/lcp} suivant$
27: **fin si**
28: **fin tant que**

Chapitre 6

Conclusion

Dans ce mémoire, nous avons étudiés les différentes classes de transducteurs classiques. En particulier, nous avons vu que les transducteurs fonctionnels et sous-séquentiels jouissaient de bonnes propriétés de décidabilité tout en étant plus expressifs que les DFTs. Ces classes de transducteurs sont donc particulièrement intéressantes en pratique. De plus, nous avons étudié les propriétés qui caractérisent les relations fonctionnelles et les fonctions sous-séquentielles, et nous avons vu qu'il existe donc des procédures pour décider ces propriétés.

L'étude des algorithmes de décision pour ces propriétés a mené à la création d'un petit framework pour les transducteurs. L'implémentation se veut le plus générale possible pour permettre d'étendre facilement les structures de base. Des extensions possibles sont les *transducteurs avec pile à comportement visible* tels que décrits par Servais [Ser11], les transducteurs pondérés très souvent utilisés par Mohri [AM03] ou les transducteurs finis *symboliques* [DV13].

Bibliographie

[AM03] Cyril Allauzen and Mehryar Mohri. Efficient algorithms for testing the twins property. *J. Autom. Lang. Comb.*, 8(2) :117–144, April 2003.

[BC02] Marie-Pierre Béal and Olivier Carton. Determinization of transducers over finite and infinite words. *Theoretical Computer Science*, 289(1) :225 – 251, 2002.

[BCPS03] Marie-Pierre Béal, Olivier Carton, Christophe Prieur, and Jacques Sakarovitch. Squaring transducers : an efficient procedure for deciding functionality and sequentiality. *Theoretical Computer Science*, 292(1) :45 – 63, 2003.

[Ber79] J. Berstel. *Transductions and context-free languages.* Leitfäden der angewandten Mathematik und Mechanik. Teubner, 1979.

[BP+85] Jean Berstel, Dominique Perrin, et al. *Theory of codes*, volume 22. Academic Press New York, 1985.

[Cho77] Christian Choffrut. Une caracterisation des fonctions séquentielles et des fonctions sous-séquentielles en tant que relations rationnelles. *Theoretical Computer Science*, 5(3) :325 – 337, 1977.

[dS09] Rodrigo de Souza. On the decidability of the equivalence for a certain class of transducers. In Volker Diekert and Dirk Nowotka, editors, *Developments in Language Theory*, volume 5583 of *Lecture Notes in Computer Science*, pages 478–489. Springer Berlin Heidelberg, 2009.

[DV13] Loris D'Antoni and Margus Veanes. Equivalence of extended symbolic finite transducers. Technical Report MSR-TR-2013-4, January 2013.

[EM65] Calvin C. Elgot and John E. Mezei. On relations defined by generalized finite automata. *IBM J. Res. and Develop.*, 9 :47–68, 1965.

[GI83] Eitan M. Gurari and Oscar H. Ibarra. A note on finite-valued and finitely ambiguous transducers. *Mathematical systems theory*, 16(1) :61–66, 1983.

[Gri68] Timothy V. Griffiths. The unsolvability of the equivalence problem for lambda-free nondeterministic generalized machines. *J. ACM*, 15(3) :409–413, 1968.

[HMU06] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation (3rd Edition)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2006.

[Sch75] Marcel Paul Schützenberger. Sur les relations rationnelles. *Automata Theory and Formal Languages*, 33 :209–213, 1975.

[Ser11] Frédéric Servais. *Visibly Pushdown Transducers*. PhD thesis, Université Libre de Bruxelles, 2011.

[WK95] Andreas Weber and Reinhard Klemm. Economy of description for single-valued transducers. *Inf. Comput.*, 118(2) :327–340, May 1995.

Annexe A

Schéma XML

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs:element name="graph" type="graph"/>
<xs:element name="automaton" type="graph"/>
<xs:element name="transducer" type="graph"/>

 <xs:complexType name="graph">
  <xs:sequence>
   <xs:element name="nodes">
    <xs:complexType>
     <xs:sequence>
      <xs:element name="node" type="node" maxOccurs="unbounded" />
     </xs:sequence>
    </xs:complexType>
   </xs:element>

   <xs:element name="transitions">
    <xs:complexType>
     <xs:sequence>
      <xs:element name="transition" type="transition" maxOccurs="
         unbounded" />
     </xs:sequence>
    </xs:complexType>
   </xs:element>
  </xs:sequence>

  <xs:attribute name="size" type="xs:positiveInteger" use="
     required"/>
 </xs:complexType>

 <xs:complexType name="node">
```

```
31    <xs:attribute name="id" type="xs:nonNegativeInteger" use="
          required"/>
32    <xs:attribute name="initial" type="xs:boolean"/>
33    <xs:attribute name="accepting" type="xs:boolean"/>
34    <xs:attribute name="output" type="xs:string"/>
35   </xs:complexType>
36
37   <xs:complexType name="transition">
38    <xs:attribute name="from" type="xs:nonNegativeInteger" use="
          required"/>
39    <xs:attribute name="to" type="xs:nonNegativeInteger" use="
          required"/>
40    <xs:attribute name="input" type="xs:string" use="required"/>
41    <xs:attribute name="output" type="xs:string"/>
42   </xs:complexType>
43 </xs:schema>
```

Listing A.1 – Schéma XML pour le format de fichier

Annexe B

Code source

Contient le code source complet, un programme exécutable prenant en entrée et effectuant les tests tels que décrits au chapitre 5 ainsi que plusieurs exemples au format XML.

Le programme fourni se lance avec la commande **java -jar Prog.jar**, il prend éventuellement comme arguments l'emplacement du fichier d'entrée et l'emplacement du fichier de sortie.