Les transducteurs à sorties variables

Denis Maurel[1], Jan Daciuk[2]

[1] Université François-Rabelais de Tours – Laboratoire d'Informatique
denis.maurel@univ-tours.fr

[2] Université Polytechnique de Gdańsk
jandac@eti.pg.gda.pl

Résumé

Dans le traitement automatique du langage naturel, les dictionnaires électroniques associent à chaque mot de l'information. La représentation informatique la plus efficace de ces dictionnaires utilise des machines à nombre fini d'états (automates ou transducteurs). Dans cet article, nous nous inspirons des algorithmes de construction directe d'un automate déterministe minimal pour proposer une nouvelle forme de transducteur. Cette nouvelle forme permet un calcul rapide des sorties associées aux mots, tout en étant plus compacte quant au nombre de transitions et de sorties distinctes, comme le montrent nos expérimentations.

Mots-clés : automates à nombre fini d'états, transducteurs, dictionnaires électroniques.

Abstract

In natural language processing, dictionaries usually associate additional information with lexical entries. The most effective representation of dictionaries makes use of finite-state machines – either automata (recognizers) or transducers. In this paper, we draw our inspiration from algorithms to directly build the minimal deterministic automaton and we propose a new form of a transducer. This new form outperforms existing transducers in terms of speed while computing outputs and in terms of size calculated on the basis of the number of transitions and different outputs, as shown in our experiments.

Keywords: finite state automata, transducers, electronic dictionaries.

1. Introduction

L'utilisation des automates et transducteurs à nombre fini d'états dans le cadre du Traitement automatique des langues, déjà recommandée dans les années quatre-vingts par Maurice Gross (Gross, 1989), est maintenant largement diffusée (Roche et Schabes, 1997). Les machines à nombre fini d'états permettent un accès rapide à l'information, proportionnel à la longueur de l'élément cherché. Elles sont utilisées dans différents systèmes, citons, parmi d'autres, Intex (Silberztein, 1993), Unitex (Paumier, 2003), Nooj (Silberztein, 2004), Xerox Finite-State Tool (Beesley et Karttunen, 2003)… et servent à toute sorte d'application : segmenteurs, analyseurs morphologiques (Kaplan et Kay 1994), étiqueteurs (Roche et Schabes, 1995), etc.

Une des applications les plus importantes est la modélisation des dictionnaires électroniques. Les dictionnaires peuvent être de simples listes de mots, mais fréquemment on veut associer à chaque mot une information (une partie du discours, le lemme, le genre, le nombre, le cas…). Cette information constitue ce qu'on appelle, dans le cadre des machines à nombre fini d'états, une sortie du transducteur.

Plusieurs modèles de sorties sont proposés, depuis le hachage parfait aux différents types de transducteurs. Nous présentons ici une idée originale : tenir compte de la construction de l'automate à partir du dictionnaire pour placer les sorties sur les nouvelles transitions des états divergents. Nous prendrons comme exemple la construction d'un automate représentant un dictionnaire électronique.

Après avoir présenté brièvement trois modèles de machines finies dans la section 2, nous définirons précisément ce que nous appelons un transducteur à sorties variables (section 3) et nous comparerons les tailles de ces différents modèles à la section 4.

2. Trois modèles de machines finies

Dans cette partie, nous allons illustrer différents procédés existants pour générer l'information à partir d'un dictionnaire électronique. Les figures ci-dessous représenteront un dictionnaire de sept mots : *rade* (*Nfs*), *rate* (*Nfs*), *ride* (*Nfs*), *rite* (*Nms*), *rude* (*Amfs*), *ruse* (*Nfs*) et *ruses* (*Nfp*). Les mots du dictionnaire seront systématiquement terminés par le caractère de fin de mots, #, ce qui évite la présence sur l'automate d'états finaux intermédiaires.

2.1. Le transducteur de hachage

Introduit par Dominique Revuz (1991) et retrouvé ensuite indépendamment par Cláudio Leonardo Lucchesi et Tomasz Kowaltowski (1993), le transducteur de hachage associe à chaque mot sa position dans l'ordre « alphabétique » ... Ou plutôt dans un ordre prédéfini sur les caractères, ce qui n'est pas exactement la même chose[1]. L'essentiel étant d'associer à un mot une unique sortie dans un tableau numéroté de 0 à n-1. Pour cela, on associe à chaque transition un nombre entier de manière à ce que la somme des sorties des transitions parcourues nous donne la position du mot dans la liste triée. Le transducteur de hachage a pour automate sous-jacent l'automate déterministe minimal.

Sur la Figure 1, le premier mot, *rade*, est associé à la valeur 0+0+0+0+0=0 et le mot *rite* à la valeur 0+2+1+0+0=3.

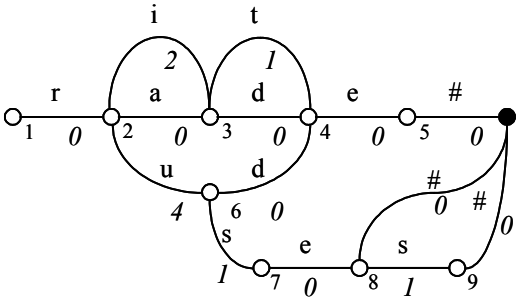

Figure 1. Un transducteur de hachage

Cette représentation est idéale lorsque les sorties sont toutes très différentes et volumineuses. L'inconvénient est la maintenance d'un tel automate. En effet, l'ajout ou la suppression d'un mot est possible sans difficultés, mais elle change l'index ! Voir à ce sujet (Daciuk *et al.*, 2005).

[1] Il ne s'agit pas, hélas, d'un tri alphabétique classique, mais d'un tri lexicographique (c'est-à-dire sur les caractères). En français, un grand nombre de règles interviennent dans le tri alphabétique (en particulier, l'interclassement de lettres majuscules et minuscules, de lettres accentuées ou non, de ligatures, etc.) (Labonté, 1998). Le tri se complique encore dans le cas des noms propres (Tran *et al.*, 2005) !

2.2. Le transducteur à multiterminaux

Lui aussi proposé par Dominique Revuz dans sa thèse (1991), le transducteur à multiterminaux (appelé aussi *Machine de Moore*) est utilisé par les logiciels Intex et Unitex. Il s'agit de placer l'information sur des états terminaux. L'automate obtenu est déterministe, mais la minimisation ne concerne qu'une partie des mots ayant la même sortie. Cette représentation est, au contraire de la précédente, très intéressante pour un nombre d'étiquettes très inférieur au nombre de mots. La Figure 2 présente un tel automate (rappelons que nous avons remplacé les états terminaux par des transitions finales, mais le résultat est identique).

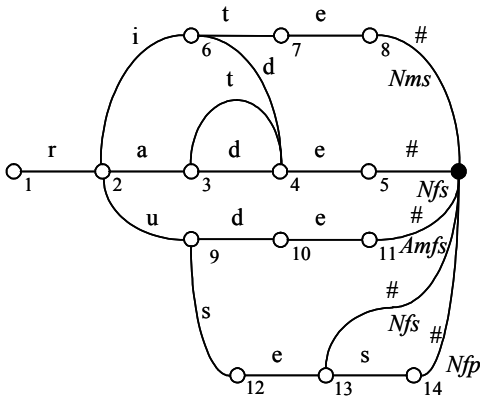

Figure 2. Un transducteur à multiterminaux

Remarquons sur cette figure la présence de deux transitions portant la même sortie (*Nfs*), puisque le mot *ruse* est contenu dans le mot *ruses*.

2.3. Le transducteur

C'est Mehryar Mohri (1994) qui a proposé de minimiser les transducteurs en déplaçant les sorties le plus à gauche possible, quitte à les scinder en morceaux. Les sorties sont donc à concaténer pour obtenir la sortie finale (voir **Erreur ! Référence non valide pour un signet.** où le *N* a été placé en amont du *fs* de *ride* et du *ms* de *rite*, tout comme le *Nf* en amont du *s* de *ruse* et du *p* de *ruses*).

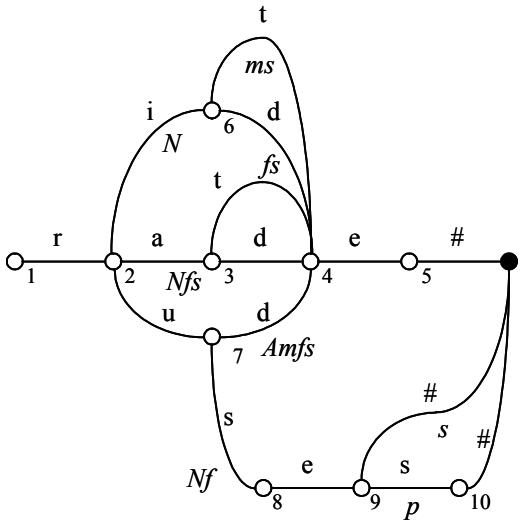

Figure 3. Un transducteur

3. Le transducteur à sorties variables

3.1. La construction directe de l'automate déterministe minimal

Dans les années quatre-vingt-dix, l'algorithme classique de construction d'un automate déterministe minimal (Hopcroft, 1971) exigeait trop de place en mémoire pour permettre la modélisation de vrais dictionnaires de langue. Les automates représentant des dictionnaires étant en principe acycliques, plusieurs algorithmes spécifiques ont été proposés, en particulier celui de Dominique Revuz (1992) qui utilise une pseudo-minimisation préliminaire. Un peu plus tard, plusieurs personnes ont parallèlement implanté des algorithmes de construction directe de l'automate déterministe minimal : citons Jan Daciuk (1998) et Stoyan Mihov (1998, 1999), dans le cas d'une liste de mots triée ; Jan Daciuk (1998) ainsi que Bruce et Richard Watson (Daciuk *et al.*, 1998), J. Aoe, K. Morimoto et M. Hase (1992), Dominique Revuz (2000), dans le cas d'une liste de mots non triée. L'algorithme pour une liste de mots triée a été étendu à la construction d'un transducteur (Mihov et Maurel, 2000) et, plus récemment, à un automate cyclique (Carrasco et Forcada, 2002).

D'autres algorithmes existent : citons celui de Bruce Watson, qui range préliminairement les mots par longueur décroissante (Watson, 1998, 2003).

3.2. La construction d'un transducteur à sorties variables

Dans cet article, nous utilisons une version légèrement augmentée de l'algorithme de construction à partir d'une liste de mots triée.

Illustrons notre algorithme à partir de notre exemple et de la Figure 4. Lors de la construction du premier mot, *rade* (*Nfs*), nous plaçons la sortie sur la première transition. La construction du deuxième mot, *rate* (*Nfs*), commence par une lecture du *r* et du *a* sur le premier automate, puis vient la création d'une deuxième transition sur l'état 3 qui devient divergent. C'est cette transition qui va porter la sortie. Et ainsi de suite… Nous suivons donc exactement l'algorithme de construction, en ajoutant l'insertion de la sortie associée au mot sur la première transition créée.

Dans la phase de minimisation partielle[2], qui suit l'insertion d'un nouveau mot, celle-ci prend en compte les étiquettes et les sorties ainsi placées. Ainsi, toujours sur la Figure 4, l'état 6 ne sera pas fusionné avec l'état 3, à cause de la présence de sorties différentes.

Une fois la construction terminée, lors de la lecture d'un mot, la variable de sortie prendra successivement toutes les valeurs de sortie rencontrées, la dernière étant la sortie associée au mot. Par exemple, lisons le mot *ruses* (*Nfp*) sur la Figure 4 : la variable de sortie associée prendra successivement les valeurs *Nfs*, *Amfs* , *Nfs* et *Nfp*, sa valeur finale.

[2] Dont la profondeur dépend du mot suivant, ce qui justifie le fait que la liste soit triée au départ de l'algorithme.

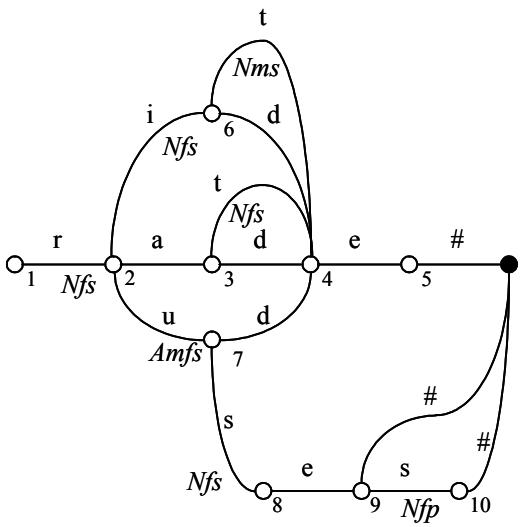

Figure 4. Un transducteur à sorties variables

3.3. Définition et algorithmes

Définition : Un *transducteur à sorties variables* est un *transducteur p-sous-séquentiel à nombre fini d'états* qui associe à chaque mot reconnu la dernière sortie rencontrée lors du parcours de l'automate.

C'est-à-dire un septuplet $T=(Q, L, S, q_0, q_1, \delta, \lambda)$ où Q est un ensemble fini non vide d'états, L un ensemble fini non vide de lettres (l'alphabet d'entrée), S un ensemble fini non vide d'entiers (les numéros de ligne du tableau des sorties), q_0 est un élément de Q (l'état initial), q_1 est un élément de Q (l'état terminal), δ est une fonction définie de $Q \times (L \cup \{\#\})$ dans Q (la fonction de transition) et λ est une fonction définie de $Q \times (L \cup \{\#\})$ dans $S \cup \{\varepsilon\}$ (la fonction de sortie des transitions).

Un mot $m=l_1 l_2 ... l_n$ est reconnu par T ssi $\exists\ p_0, p_1, ..., p_n \in Q$, tels que :

$\forall\ i=0, 1, ..., n-1, \delta(p_i, l_{i+1})=p_{i+1}$ et $\forall\ i=0, 1, ..., n-1, \lambda(p_i, l_{i+1})=s_{i+1}$ avec $p_0=q_0$ et $p_n=q_1$

et il a pour sortie $s= s_j$ avec $j=\text{Max}\{i=0, 1, ..., n-1 / \lambda(p_i, l_{i+1})\neq\varepsilon\}$

Algorithme :

```
{       Créer_l'automate_avec_le_premier_mot[3]
        suivant←Lire_un_mot_dans_le_dictionnaire
        FAIRE
        {       courant←suivant
                suivant←Lire_un_mot_dans_le_dictionnaire
                Lire_le_début_du_mot_courant_sur_l'automate
                Complèter_l'automate_par_la_fin_du_mot_courant[3]
                préfixe_plus_un←Calculer_la_fin_de_la_minimisation
                Minimiser_l'automate_jusqu'au_préfixe_plus_un
        }
        TANT_QUE (le_mot_suivant_existe)
}
```

[3] En plaçant la sortie sur la première transition.

4. Comparaison des résultats

Nous allons comparer les résultats obtenus sur deux dictionnaires français, la liste des noms des villes françaises et le dictionnaire Delaf (Courtois et Silberztein, 1990), qui est sans doute le dictionnaire électronique le plus complet pour les mots monolexicaux de la langue française. Ces dictionnaires sont présentés sur la Figure 5 (nombre de mots et de caractères, sans prendre en compte les sorties, mais avec le caractère de fin de mot). Nous ajoutons aussi à ce tableau le nombre d'états et de transitions de l'automate déterministe minimal correspondant.

	Dictionnaire		Automate	
	Mots	Caractères	États	Transitions
Villes	38 198	447 549	61 240	95 589
Delaf	682 418	7 238 120	67 995	177 465

Figure 5. Nos listes de mots et l'automate déterministe minimal correspondant

Nous associons aux noms des villes françaises, le numéro de département et aux mots du dictionnaire Delaf, le lemme[4], la catégorie syntaxique, la couche d'utilisation[5] et le code morphologique. Des exemples de sorties sont donnés sur la Figure 6. Le dictionnaire des noms des villes françaises ainsi constitué possède 1 550 sorties distinctes[6] et le dictionnaire Delaf, 7 185.

	Entrées	Sorties
Villes	Tours	37
Delaf	exemple exemples	0.N+z1:ms -1.N+z1:mp

Figure 6. Exemples de données

Présentons maintenant les résultats concernant la construction d'un transducteur. La Figure 7 présente, pour chaque construction, le nombre d'états et de transitions obtenus, ainsi que le nombre de transitions qui portent des sorties et le nombre de sorties distinctes présentes sur le transducteur. Les sorties distinctes sont placées dans un tableau. Ce sont les numéros de ligne de ce tableau qui sont réellement notée sur le transducteur. D'où l'importance de la taille de ce tableau, c'est-à-dire du nombre de sorties distinctes sur le transducteur. ?

Ces exemples prouvent, s'il était besoin, l'importance de la représentation des données. Si nous mémorisons nos automates sous la forme d'un tableau de transitions (Daciuk, 2000), nous obtenons évidemment que le tableau minimal est celui du hachage parfait, puisqu'il

[4] Le lemme est calculé : on indique le nombre de caractères à retirer au mot en entrée pour l'obtenir (avec, éventuellement des caractères à ajouter).

[5] Le vocabulaire du dictionnaire Delaf est divisé en trois couches : mots courants, précieux et techniques (Garrigues, 1993).

[6] Il ne faut pas s'étonner de trouver 1 550 sorties pour les numéros de département associés au noms de ville, alors qu'il n'existe que 99 numéros ! Cela est dû à l'importante homographie des noms de ville française. Par exemple, il existe 25 villes portant le nom de *Neuilly* ; celles-ci sont en fait distinguées par un nom polylexical plus large (*Neuilly-sur-Seine*, *Neuilly-sur-Marne*, etc.), mais trois restent avec un nom monolexical associé à une sortie triple : *27|58|89* (Belleil, Maurel, 1996).

correspond exactement à l'automate déterministe minimal. Cependant, celui-ci comprend un nombre important de transitions avec sortie (sur le tableau de la Figure 7, le nombre de transitions avec sortie des autres transducteurs représente respectivement pour les deux dictionnaires 4,4 % et 0,1 % du nombre de sorties du transducteur de hachage).

Pour + plusieurs nom + ,

Pour les trois autres constructions, le transducteur à sorties variables est celui qui possède le moins de transitions pour les deux dictionnaires (pour les noms de ville, le transducteur à multiterminaux a même un tiers de transitions en plus…). Paradoxalement peut-être, puisque le principe du transducteur classique est de permettre des factorisations, c'est aussi celui des deux transducteurs qui possède le moins de sorties distinctes. En fait, exactement le nombre de sorties distinctes présentes dans le dictionnaire, comme évidemment le transducteur à multiterminaux, bien moins performant en nombre de transitions comme cela a été déjà remarqué.

« moins que », « plus que » pour la comparaison

Notre représentation des dictionnaires semble donc légèrement meilleure que la représentation classique par transducteur. Elle évite en plus le calcul correspondant à la concaténation des sorties, calcul non trivial lorsqu'une sortie est multiple[7].

		États	Transitions	Transitions avec sortie	Sorties distinctes
Villes	Transducteur de hachage	61 240	95 589	95 589	35 382[8]
	Transducteur à multiterminaux	94 870	130 199	3 792	1 550
	Transducteur	62 692	98 026	35 443	1 564
	Transducteur à sorties variables	62 395	97 523	35 129	1 550
Delaf	Transducteur de hachage	67 995	177 465	177 465	637 282[8]
	Transducteur à multiterminaux	102 343	251 472	20 186	7 185
	Transducteur	99 529	259 139	81 623	9 013
	Transducteur à sorties variables	92 946	240 534	147 589	7 185

Figure 7. Taille respective des transducteurs

[7] Par exemple, le mot *fort* a pour catégorie adjectif, adverbe ou nom et est donc associé à une sortie triple : *0.A+z1:ms|0.ADV+z1|0.N+z1:ms* pour laquelle on aurait peut-être pu factoriser *0.* sur des transitions précédentes.

[8] Il ne s'agit pas véritablement ici du nombre de sorties différentes, mais de la plus grande sortie présente sur le transducteur.

5. Conclusion

Dans cet article, nous avons défini un nouveau modèle de transduction à nombre fini d'états, le transducteur à sorties variables. Après comparaison avec d'autres machines finies, nous remarquons qu'il permet une représentation des dictionnaires légèrement plus compacte en nombre de transitions et de sorties différentes. L'algorithme de construction du transducteur à sorties variables est basé sur l'algorithme de construction à partir d'une liste de mots triée et a donc une complexité similaire[9]. De plus, le calcul de la sortie associée à un mot reconnu est trivial, ce qui n'est pas toujours le cas d'un transducteur classique. Ceci à la fois dans l'algorithme de construction du transducteur, où il faut décomposer les sorties et dans l'algorithme de lecture où il faut au contraire les concaténer.

6. Remerciements

Les auteurs remercient le Professeur Éric Laporte pour l'utilisation du dictionnaire *Delaf*. La collaboration sur ce sujet entre les universités Politechnika Gdańska et François-Rabelais a débuté alors que Jan Daciuk était professeur invité à Blois.

Références

AOE J., MORIMOTO K., HASE M. (1992). « An Algorithm for Compressing Common Suffixes Used ». In *Trie Structures*. Trans. IEICE.

BEESLEY K.R., KARTTUNEN L. (2003). *Finite-State Morphology*. CSLI Publications.

BELLEIL C., MAUREL D. (1996). « Traitement informatique des ambiguïtés dans la reconnaissance des noms propres liés à la géographie ». In *Bulag* 21 : 29-50.

CARRASCO R.C., FORCADA M.L. (2002). « Incremental construction and maintenance of minimal finite-state automata ». In *Computational Linguistics* 28 (2) : 207-216.

COURTOIS B., SILBERZTEIN M. (1990). « Dictionnaires électroniques du français ». In *Langue française* 87 : 11-22.

DACIUK J. (1998). *Incremental Construction of Finite-State Automata and Transducers, and their Use in the Natural Language Processing*. Ph.D. dissertation, Technical University of Gdańsk.

DACIUK J. (2000). « Experiments with Automata Compression ». In *CIAA 2000*. LNCS 2088 : 105-112.

DACIUK J., MAUREL D., SAVARY A. (2005). « Dynamic Perfect Hashing with Finite-State Automata ». In *Intelligent Information Systems 2005* (*IIS 2005*). Gdansk.

DACIUK J., MIHOV S., WATSON B.W., WATSON R.E. (2000). « Incremental construction of Minimal Acyclic Finite-state Automata ». In *Computational Linguistics* 26 (1) : 3-16.

DACIUK J., WATSON B. W., WATSON R.E. (1998). « Incremental construction of Minimal Acyclic Finite-state Automata and Transducers ». In *Proceedings of FSMNLP* : 48-56.

GARRIGUES M. (1993). *Méthode de paramétrage des dictionnaires et grammaires électroniques : Application à des systèmes interactifs en langue naturelle*. Thèse de doctorat en Sciences du langage, Université Paris VII.

GROSS M. (1989). « The Use of Finite Automata in the Lexical Representation of Natural Language ». In *Electronic Dictionaries and Automata in Computational Linguistics.* LNCS 377 : 34-50.

HOPCROFT J. (1971). « An *n* log *n* algorithm for minimizing states in a finite automaton ». In *International Symposium on Theory of Machines and Computations*. Haifa : 189-196.

[9] O(n) en espace et O(mlog(n)) en temps, avec n le nombre d'états et m le nombre de mots (Mihov, 1999).

KAPLAN R.M., KAY M. (1994). « Regular Models of Phonological Rule Systems ». In *Computational Linguistics*. 20 (3) : 331-378.

LABONTÉ A. (1998). *Technique de réduction – Tris informatiques à quatre clés*, Gouvernement du Québec, publié sur la Toile à l'adresse : http://www.lirmm.fr/~lafourca/Souk/trif/techniques-de-tri.html.

LUCCHESI C.L., KOWALTOWSKI T. (1993). « Applications of Finite Automata Representing Large Vocabularies ». In *Softw., Pract. Exper.*, 23 (1) : 15-30.

MIHOV S. (1998). « On-line algorithm for building minimal automaton Presenting Finite Language ». In *Annuaire de l'Université de Sofia St. Kl. Ohridski* 91 (1). Faculté de Mathématiques et Informatique.

MIHOV S. (1999). « Direct construction of minimal acyclic finite states automata ». In *Annuaire de l'Université de Sofia St. Kl. Ohridski*. 92 (2). Faculté de Mathématiques et Informatique.

MIHOV S., MAUREL D. (2000). « Direct construction of minimal acyclic sub- sequential transducers ». In *CIAA'2000*. LNCS 2088 : 217-229.

MOHRI M. (1994). « Minimization of Sequential Transducers ». In *CPM'94*. LNCS 807 : 151-163.

PAUMIER S. (2003). *De la Reconnaissance de Formes Linguistiques à l'Analyse Syntaxique*. Thèse de Doctorat en Informatique, Université de Marne-la-Vallée.

REVUZ D. (1991). *Dictionnaires et lexiques – Méthodes et algorithmes*, Thèse de Doctorat en Informatique, Université Paris 7.

REVUZ D. (1992). « Minimization of acyclic deterministic automata in linear time ». In *Theoretical Computer Science* 92 : 181-189.

REVUZ D. (2000). « Dynamic acyclic minimal automaton ». In *CIAA 2000*. LNCS 2088:226-232.

ROCHE E., SCHABES Y. (éds) (1997). *Finite state language Processing.* MIT Press, Cambridge, Mass./London,.

SILBERZTEIN M. (1993). *Dictionnaires électroniques et analyse automatique de textes – Le système INTEX*. Masson, Paris.

SILBERZTEIN M. (2004). « Nooj: A cooperative Object Oriented Architecture for NLP ». In *Cahiers de la MSH Ledoux*, *Série Archive, Bases, Corpus* 1 : 351-361.

TRAN M., MAUREL D., SAVARY A. (2005). « Implantation d'un tri lexical respectant la particularité des noms propres ». In *Lingvisticae Investigationes*, XXVIII-2 (à paraître).

WATSON B.W. (1998). « A fast new semi-incremental algorithm for the Construction of Minimal Acyclic DFAs ». In *WIA'98*, LNCS 1660 : 121-132.

WATSON B.W. (2003). « A new algorithm for the construction of minimal acyclic DFAs ». In *Science of Computer Programming*, 48-2-3 : 81-97.