

**FONCTIONNALITES INTEX  
DANS L'OUTIL D'AIDE A LA TRADUCTION  
LEXPRO CD DATABANK**

AGATA CHROBOT  
*Université de Marne-la-Vallée*  
*LCI, Jouy-en-Josas*

Nous présentons une application industrielle de certaines fonctionnalités du système de traitement de grands corpus Intex dans le domaine d'aide à la traduction. Il s'agit du logiciel LexPro CD Databank réalisé et commercialisé par la société LCI (Jouy-en-Josas, France) qui est partenaire du LADL.

### **1. LexPro CD Databank**

LexPro CD Databank (que nous appelons LexPro par la suite) est une des plus grandes bases de données terminologiques accessibles sur ordinateur PC. Elle contient près de 120 dictionnaires traditionnels de traduction mis sur le support informatique, soit environ 5 millions de termes en 11 langues (les plus nombreux en français, anglais et russe) dans une trentaine de domaines techniques. La version 3.0 de ce logiciel comprend entre autres les modules de lemmatisation de termes et de leur correction orthographique. Pour la version suivante, le module d'extraction terminologique est sous élaboration. Les trois modules mentionnés ont été conçus à partir de certains programmes d'Intex ce que nous décrivons ci-dessous.

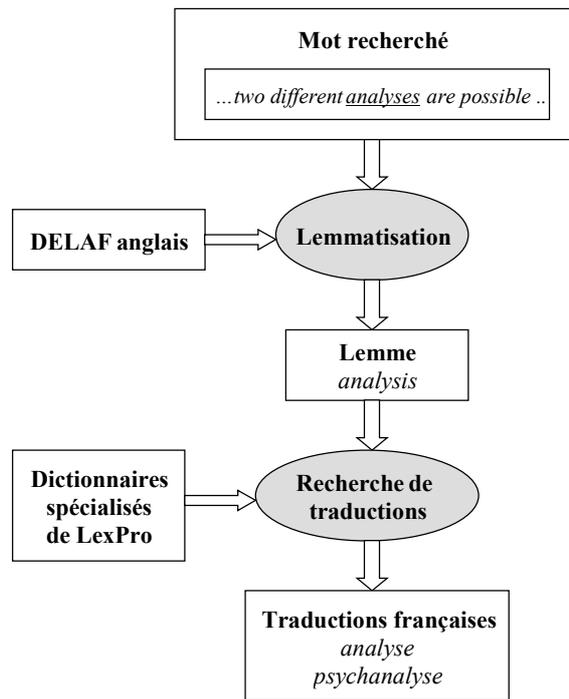
### **2. Lemmatisation**

Les dictionnaires terminologiques de LexPro sont sauvegardés sur un CD Rom sous format d'une base de donnée Access. Les termes, simples et composés, y figurent sous leurs formes de base (les infinitifs pour les verbes, les singuliers pour les noms etc.). Or, il est souvent utile de pouvoir rechercher un terme à partir d'une de ses formes fléchies car ce sont ces formes-là qui apparaissent dans des textes. En particulier, en utilisant un programme de traitement de texte, nous voudrions obtenir la traduction des termes inconnus sans devoir entrer leurs formes de base.

Ceci est possible dans LexPro grâce au module de lemmatisation. Le mot en question est d'abord recherché dans le dictionnaire DELAF<sup>1</sup> (ou DELACF) de la langue source. Le DELAF, comme ceci a lieu dans le système DELA, ne contient que les mots fléchis avec leurs étiquettes grammaticales (le lemme, la catégorie, les traits morphologiques). Il est converti en un automate fini de la même façon que ceci a lieu dans le système Intex (voir Silberztein 1997). La consultation de cet automate se fait en temps linéaire en fonction de la longueur du mot recherché (par un module fondé sur le programme *dicodlf* d'Intex). Ainsi, nous accédons au lemme du mots recherché qui permet ensuite d'obtenir sa traduction vers la langue cible souhaitée dans le (les) dictionnaire(s) terminologique(s) sélectionné(s). Un exemple de ce traitement et présenté sur la figure Fig.1.

---

<sup>1</sup> Pour les principes de la méthodologie DELA, voir par exemple Courtois, B., Silberztein M. (éd.) 1990. *Dictionnaire électroniques de français. Langue Française, septembre 1990.* Larousse,



**Fig.1.** Recherche de termes avec lemmatisation dans LexPro

### 3. Correction orthographique

Grâce au module de correction orthographique de LexPro, l'utilisateur peut rechercher des termes dans la base terminologique même s'il les a mal orthographiés. La méthode élaborée est indépendante de la langue concernée, c'est-à-dire elle fonctionne pour toute langue pour laquelle l'on dispose d'un dictionnaire DELAF.

Nous admettons, comme ceci a lieu dans les travaux de référence du domaine de la correction automatique de l'orthographe (Damerau 1964, Oflazer 1996), que chaque faute de frappe résulte d'un mot correct par application d'une ou plusieurs opérations élémentaires sur des lettres. Nous distinguons 4 opérations élémentaires:

- a) l'insertion d'une lettre : *certificat* > \**certificlat*;
- b) l'omission d'une lettre : *immunisation* > \**immunisatio* ;
- c) le remplacement d'une lettre : *cornichon* > \**kornichon*;
- d) l'inversion de deux lettres voisines : *enveloppe* > \**evnveloppe*.

Selon des études statistiques, la quasi totalité des fautes de frappes dans des textes est due à une, éventuellement à deux, de ces opérations à la fois.

#### 3.1. Exemple

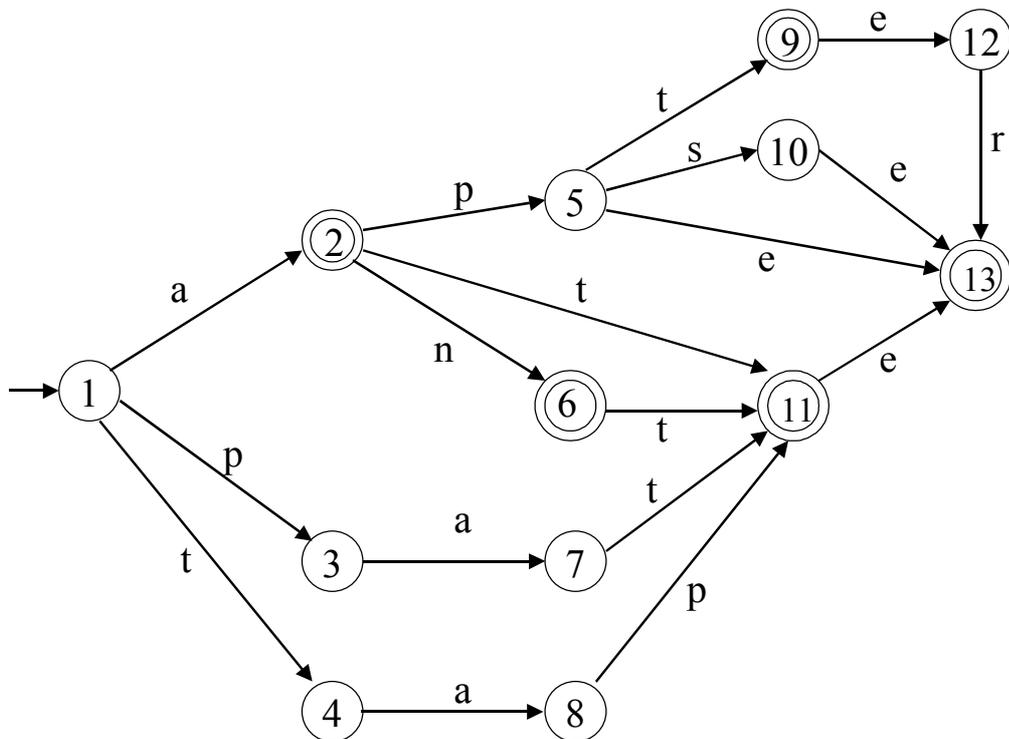
L'interprétation de l'origine d'une faute peut être ambiguë. Par exemple, pour le mot erroné anglais

- (1) \*apte

il existent au moins 7 corrections possibles:

- (2) *apter* : omission de *r*
- apt* : insertion de *e*
- ape* : omission de *t*
- apse* : remplacement de *s* par *t*
- ate* : insertion
- ante* : remplacement de *n* par *p*
- pate* : inversion de *p* et *a*

L'algorithme de correction est fondé sur celui de la recherche d'un mot dans un automate fini (le même qui a été utilisé pour la lemmatisation). D'abord nous recherchons le mot tel quel dans l'automate et, s'il n'a pas pu être trouvé, nous faisons le retour en arrière (*back-tracking*). A chaque fois que nous retournons à un état visité précédemment, nous essayons de trouver une autre continuation du chemin en admettant l'une des quatre opérations mentionnées ci-dessus. Examinons un extrait du dictionnaire DELAF anglais, Fig. 2, contenant entre autres les sept corrections (2) du mot erroné (1).



**Fig.2.** Extrait d'un automate DELAF anglais

Les états terminaux sont marqués par un double rond. La consultation commence dans l'état initial numéro 1. La recherche du mot *apte* nous amène à l'état 12 qui n'est pas terminal et où aucune transition n'est plus possible puisque la séquence entière a été lue. L'automate étant déterministe le retour en arrière n'est pas nécessaire pour s'assurer que la séquence recherchée n'existe pas dans le dictionnaire. A ce moment-là, nous commençons la « consultation modifiée » qui consiste à chercher des mots semblables en admettant l'une des

quatre opérations élémentaires directement après l'une des cinq positions possibles dans le mot d'origine (le début du mot inclu):

	<i>a</i>	<i>p</i>	<i>t</i>	<i>e</i>
0	1	2	3	4

A la fin de la séquence d'entrée (position 4) une seule hypothèse de source de l'erreur est possible: la dernière lettre, i.e. celle qui devrait se trouver juste après le *e* a été omise. Nous essayons toutes les transitions sortant de l'état courant 12 et menant à un état terminal. Il y en a une seule: (12,*r*,13) et elle donne le premier candidat pour la correction: *apter*. Pour continuer la recherche il faut retourner de l'état 12 à l'état 9 (et de la position 4 dans le mot à la position 3), où nous testons trois interprétations possibles de l'erreur:

- La lettre *e* a été insérée à tort. Puisque le suffixe de *apte* suivant la lettre *e* est vide et l'état courant est terminal, nous obtenons un nouveau candidat: *apt*.
- Une lettre *a* a été omise à tort entre *t* et *e*. Nous essayons toutes les transitions partant de l'état courant 9 et arrivant à un état à partir duquel il est possible de reconnaître le suffixe *e*. Puisque la seule transition débutant dans l'état 9 et celle par laquelle nous venons de retourner, nous n'obtenons aucun nouveau candidat.
- La lettre correcte à la position 4 a été remplacée à tort par la lettre *e*. Nous essayons toute transition qui mène de l'état 9 à un état terminal. Il n'y en a aucune, nous n'obtenons aucun nouveau candidat.

A l'étape suivante nous retournons de nouveau en arrière jusqu'à l'état 5 (position 2 dans le mot) et nous testons 4 hypothèses:

- La lettre *t* a été insérée à tort. Nous cherchons toutes les transitions de l'état 5 à un état terminal en lisant le suffixe *e*. Il y en a une: (5,*e*,13), elle donne un nouveau candidat: *ape*.
- Les lettres *t* et *e* ont été inversées à tort. Nous essayons de reconnaître le suffixe *et* en partant de l'état courant 5, ce qui n'est pas possible.
- Une lettre *a* a été omise à tort entre les lettres *p* et *t*. Nous cherchons toutes les transitions qui mènent de l'état courant 5 à un état à partir duquel il est possible de reconnaître le suffixe *te*. Une telle transition n'existe pas.
- La lettre correcte à la position 3 a été remplacée à tort par la lettre *t*. Nous cherchons tout état accessible de l'état 5 à partir duquel il est possible de reconnaître le suffixe *e*. L'état 10 remplit cette condition et le candidat obtenu est *apse*.

De la même façon, en retournant de l'état 5 à l'état 2 (position 1 dans le mot) nous testons les quatre hypothèses possibles dont deux sont vérifiées:

- La lettre *p* a été insérée à tort. En passant par les transitions (2,*t*,11) et (11,*e*,13) nous obtenons un nouveau candidat *ate*.
- La lettre correcte à la position 2 a été remplacée à tort par la lettre *p*. En passant par les transitions (2,*n*,6), (6,*t*,11) et (11,*e*,13) nous obtenons le candidat *ante*.

Finalement, en retournant de l'état 2 à l'état 1 (position 0 dans le mot) nous obtenons *pate* par l'hypothèse de l'inversion des lettres *a* et *p* vérifiée par le chemin (1,*p*,3), (3,*a*,7), (7,*t*,11), (11,*e*,13).

### 3.1. Algorithme

Soit `search(word, state)` la procédure standard de recherche dans un automate de la séquence `word` à partir de l'état `state`. Au début cette procédure est appelée avec le paramètre `word` égal au mot recherché et le paramètre `state` égal à l'état initial `s`. Après chaque transition, l'appel récursif de `search` prend comme premier paramètre le suffixe

restant et comme deuxième paramètre l'état d'arrivé de la transition précédente. La procédure se termine avec succès si l'on arrive à un état terminal après avoir lu entièrement la séquence d'entrée. La procédure échoue dans l'un des deux cas:

- la séquence d'entrée a été lue entièrement mais nous ne nous trouvons pas dans un état terminal;
- la séquence d'entrée n'a pas été lu entièrement mais aucune transition de l'état courant n'est possible pour le suffixe restant.

La recherche modifié commence au moment où la recherche standard a été bloquée sans que la séquence d'entrée soit reconnue. Soit  $[l_1 l_2 \dots l_{w1}]$  le mot d'origine et  $w1$  sa longueur. Soit  $w_p = 0, 1, \dots, w1$  la position courante dans le mot (i.e. la position de la dernière lettre qui a été lue avec succès). Soit  $st$  l'état courant (celui à partir duquel aucune transition n'est plus possible). Nous supposons alors que l'une des quatre opérations possibles a eu lieu:

- L'insertion incorrecte d'une lettre à la position  $w_p+1$  (si  $w_p < w1$ ). Nous omettons la lettre  $l_{w_p+1}$  et essayons de reconnaître le suffixe  $[l_{w_p+2} \dots l_{w1}]$  à partir de l'état courant  $st$ , c'est-à-dire nous appelons  $search([l_{w_p+2} \dots l_{w1}], st)$  au lieu de  $search([l_{w_p+1} l_{w_p+2} \dots l_{w1}], st)$  qui aurait été appelé dans la recherche standard. Si cet appel se termine avec succès le candidat proposé pour la correction est la séquence  $[l_1 \dots l_{w_p} l_{w_p+2} \dots l_{w1}]$ .
- L'inversion incorrecte des lettres  $l_{w_p+1}$  et  $l_{w_p+2}$  (si  $w_p < w1-1$ ). Nous essayons de reconnaître le suffixe inversé  $[l_{w_p+2} l_{w_p+1} \dots l_{w1}]$  à partir de l'état courant  $st$ , c'est-à-dire nous appelons  $search([l_{w_p+2} l_{w_p+1} \dots l_{w1}], st)$ . Si ceci se termine avec succès nous proposons le candidat  $[l_1 \dots l_{w_p} l_{w_p+2} l_{w_p+1} \dots l_{w1}]$ .
- L'omission incorrecte d'une lettre à la position  $w_p+1$ . Pour chaque transition qui mène de l'état  $st$  à l'état  $st_s$  par une lettre  $l$ , nous essayons de reconnaître le suffixe  $[l_{w_p+1} l_{w_p+2} \dots l_{w1}]$  à partir de l'état  $st_s$ , c'est-à-dire nous appelons  $search([l_{w_p+1} l_{w_p+2} \dots l_{w1}], st)$ . Si la tentative se termine avec succès, nous proposons le candidat  $[l_1 \dots l_{w_p} l l_{w_p+1} \dots l_{w1}]$ .
- Un remplacement incorrect d'une lettre à la position  $w_p+1$  (si  $w_p < w1$ ). Pour chaque transition qui mène de l'état  $st$  à l'état  $st_s$  par une lettre  $l$ , nous essayons de reconnaître le suffixe  $[l_{w_p+2} l_{w_p+3} \dots l_{w1}]$  à partir de l'état  $st_s$ , c'est-à-dire nous appelons  $search([l_{w_p+2} l_{w_p+3} \dots l_{w1}], st)$ . Si la tentative se termine avec succès, nous proposons le candidat  $[l_1 \dots l_{w_p} l l_{w_p+2} \dots l_{w1}]$ .

### 3.3. Erreurs multiples dans un mot

Même si dans la plupart de fautes de frappe une seule opération élémentaire sur des lettres entre en jeu, il arrive qu'il faut en supposer deux ou plus, comme dans *\*ingeneer* (2 remplacements dans le mot correct *engineer*), *\*comitee* (2 omissions dans *committee*), *\*authentification* (2 insertions dans *authentication*), *\*inflexion* (1 omission et 1 remplacement dans *inflection*). Remarquons que tout mot fini peut être ramené à tout autre mot fini par un nombre fini d'opérations élémentaires (la prise en compte seulement des insertions et des effacements serait également suffisante). L'algorithme décrit ci-dessus est adaptable à tout nombre d'opérations admises, mais nous pensons qu'il n'est pas pratique de dépasser le seuil de deux opérations, pour deux raisons:

- nous risquons d'obtenir des candidats trop éloignés du mots d'origine;
- à partir de 3 opérations la recherche risque de durer trop longtemps pour l'application interactive telle que LexPro.

L'adaptation de l'algorithme décrit dans la section précédente consiste à définir le nombre maximal d'erreurs admises (`max_err`) et la suivie du nombre (`prec_err`) d'opérations qui ont déjà été supposées sur le chemin menant de l'état initial à l'état courant. A chaque moment où la consultation est bloquée nous essayons d'admettre l'une des 4 opérations seulement si le nombre d'opérations déjà admise ne dépasse par le seuil (`prec_err < max_err`). Nous ne recherchons que les candidats les plus proches du mot d'origine. C'est-à-dire que nous ne réexploitons pas les chemins qui ont été parcourus avec  $k$  modifications pour rechercher des candidats avec le nombre de modifications supérieur à  $k$ . Dans notre exemple (section 3.2) ceci signifie qu'après avoir trouvé 7 candidats avec une seule erreur nous ne cherchons plus à trouver les candidats avec 2 erreurs: *tape* (deux inversions de *t*), *pat* (l'inversion de *p* et l'omission de *e*) ou *ant* (remplacement de *p* par *n* et l'omission de *e*).

### 3.4. Complexité de l'algorithme

La consultation modifiée de l'automate n'est plus déterministe car plusieurs corrections sont possibles pour un mot mal orthographié, comme ceci a été le cas de l'exemple *\*apte*. Tous les chemins possibles avec une (ou deux) modifications doivent être explorés, alors la complexité du programme n'est plus linéaire. Son calcul exact est difficile car elle dépend non seulement de la longueur du mot d'entrée et de la taille de l'automate, mais aussi du nombre de mots contenu dans l'automate qui ont des infixes communs avec le mot d'origine. Dans le pire des cas, i.e. dans la situation (théorique) où tous les mots contenus dans le lexique sont des candidats possibles pour la correction, la consultation modifiée exige l'exploration de l'automate entier.

### 3.5. Travaux apparentés

Notre stratégie de l'interprétation d'une faute de frappe emploie quatre opérations élémentaires sur des lettres qui permettent de passer d'un mot à un autre. Ces opérations ont été proposées d'abord par F. J. Damerau (1964) et ensuite appliquées pour différentes langues, par exemple par K. Oflazer (1996) pour le turque, par J. Daciuk (1998) pour le polonais.

La méthode de correction orthographique par consultation modifiée de l'automate que nous avons proposée s'approche de celle obtenue par K.Oflazer (1996).

## 4. Extraction terminologique

A partir des ressources lexicographiques très riches dont nous disposons - les dictionnaires de la langue générale du LADL et la base terminologique LexPro - nous avons mis en forme un prototype d'un système de recherche de termes dans des textes spécialisés. Il peut être classée comme outil d'enrichissement (plutôt que d'extraction) terminologique car il exploite les ressources déjà existantes et les enrichie. Les principes de la méthode ont été décrits par Chrobot (1999) et nous n'en donnons ici qu'un résumé.

Voici les caractéristiques de notre méthode de recherche de nouveaux termes:

- Elle est fortement fondée sur les ressources lexicographique, dans la mesure où leurs qualité et complétude ont l'importance fondamentale pour les résultats de l'extraction.
- Elle est un outil d'aide à la traduction car son intérêt principal est d'assister un traducteur technique dans la tâche de création du glossaire d'un texte à traduire.
- Elle est indépendante de la taille du corpus (donc non statistique). Ceci est lié à la spécificité du domaine de la traduction, où les textes à traiter sont de tailles très variables.

Un corpus de très grande taille étant rarement disponible, l'emploi de toute méthode fondée sur un calcul statistique est exclu.

- Elle se limite à la recherche de termes complexes, c'est-à-dire constitués d'au moins deux mots simples séparés par un blanc ou un caractère de ponctuation (voir la définition du mot composé dans Intex).
- Elle a été élaborée et testée pour l'anglais (avec un DELAS et un DELAC généraux de 90000 et 60000 mots respectivement) dans le domaine de l'informatique (avec un DELAS et un DELAC spécialisés de 27000 et 57000 termes respectivement), mais son adaptation est envisageable pour toute langue pour laquelle l'on dispose d'un dictionnaire DELAF général, et pour tout domaine couvert par les dictionnaires techniques existants.
- Elle utilise la technique de la recherche d'un patron syntaxique dans un texte étiqueté. Une grande partie des algorithmes employés (l'identification des items du texte, l'indexation, l'étiquetage du texte, la recherche du patron) ont été récupérés du système Intex.

L'idée qui se cache derrière la méthode proposée peut être exprimée par l'hypothèse suivante:

*La création d'un nouveau terme se fait le plus souvent par une combinaison grammaticalement correcte de termes simples et composés déjà existants.*

Cela signifie que la terminologie à l'intérieur d'un domaine peut être très croissante mais son noyau lexical reste à peu près stable. Autrement dit, pour nommer des nouveaux concepts on emploie toujours les mêmes mots dans de nouvelles combinaisons. Prenons deux exemples de nouveaux termes trouvés dans un corpus informatique:

(3) *notification message timeout*

(4) *user free memory*

Dans le premier cas tous les trois composants simples du terme complexe étaient déjà recensés dans nos dictionnaires spécialisés. Dans le deuxième cas, le mot *free* ne constituait pas une entrée indépendante du dictionnaire de l'informatique mais aussi bien *user* que *free memory* s'y trouvaient. Dans les deux cas, des nouveaux termes ont été donc créés par combinaisons de termes existants.

Le schéma général de fonctionnement de notre méthode d'enrichissement terminologique est présenté sur la figure Fig. 3. Le texte est soumis à l'étiquetage, c'est-à-dire toutes les unités du texte sont recherchées dans des dictionnaires disponibles: le DELAF de mots grammaticaux, les DELAF général et spécialisé, et le DELACF général et spécialisé, le premier et les deux derniers de ces dictionnaires (boîtes grisées) étant consultés en mode prioritaire (voir Silberstein 1997). En sortie on obtient deux listes attribuées au texte: la liste des mots simples, généraux et spécialisés, y trouvés, et celle des mots composés, généraux et spécialisés. L'étiquetage se fait sans aucune désambiguïsation contextuelle, donc un mot peut obtenir zéro, une ou plusieurs étiquettes différentes.

Ensuite le texte avec les deux listes est parcouru par le patron syntaxique et toutes les suites contiguës qui matchent le patron sont incluses dans une liste de candidats-termes. Cette liste est proposée à l'utilisateur qui décide pour chaque candidat s'il doit être retenu ou rejeté. Les candidats retenus rentrent dans le glossaire du texte à traduire.

Le patron syntaxique utilisé pour l'extraction des candidats-termes (Fig. 4) reflète l'hypothèse présentée plus haut. Les étiquettes grammaticales apparaissant dans les nœuds du graphe sont conformes à la notation d'Intex. Le trait *Spec* qui accompagne les symboles des catégories grammaticales signifie que l'étiquette donnée provient du dictionnaire spécialisé (des mots

simples ou composés) et non pas du DELAF ou DELACF général. Analysons les trois branches supérieures du graphe. La recherche passe le plus souvent par l'étiquette

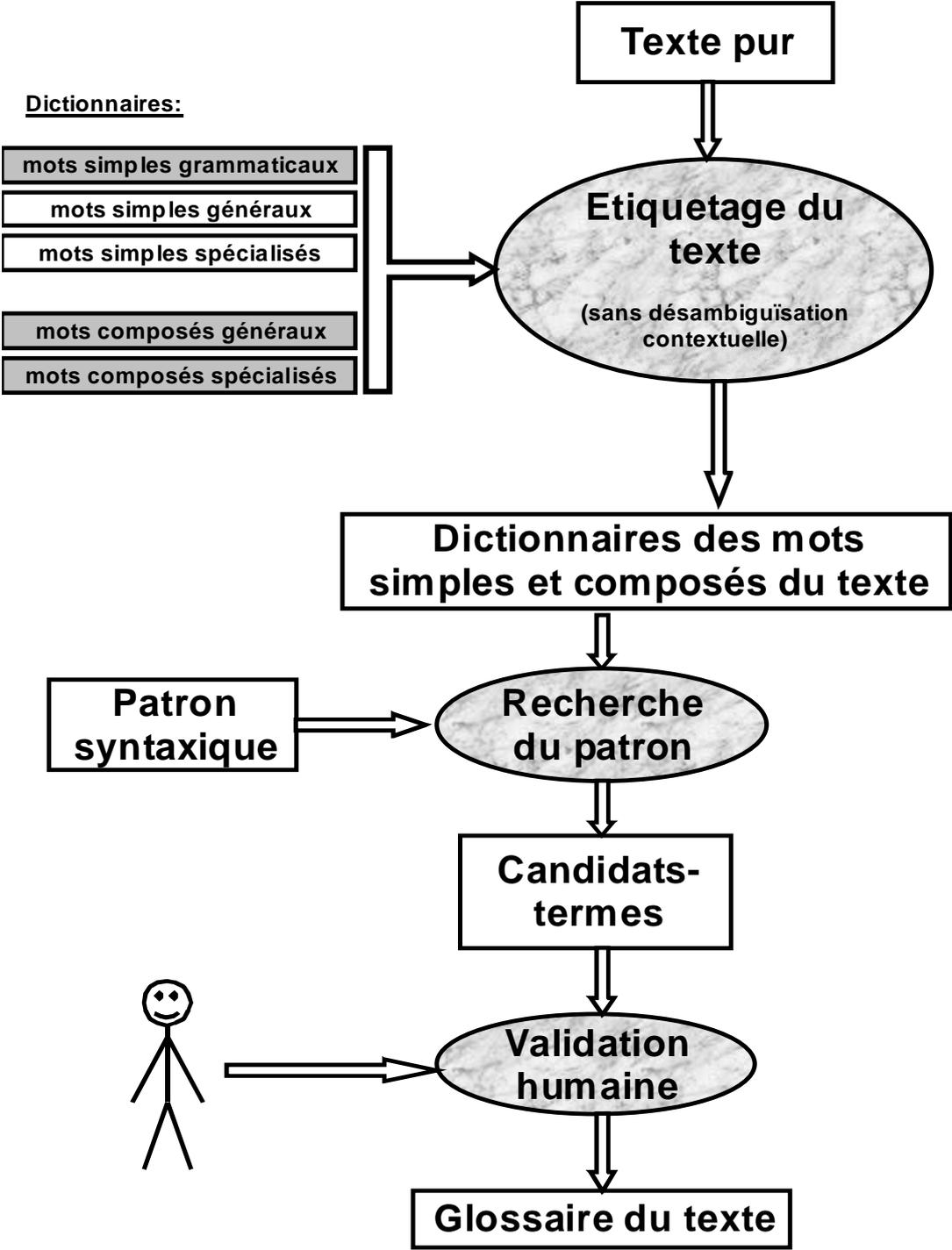
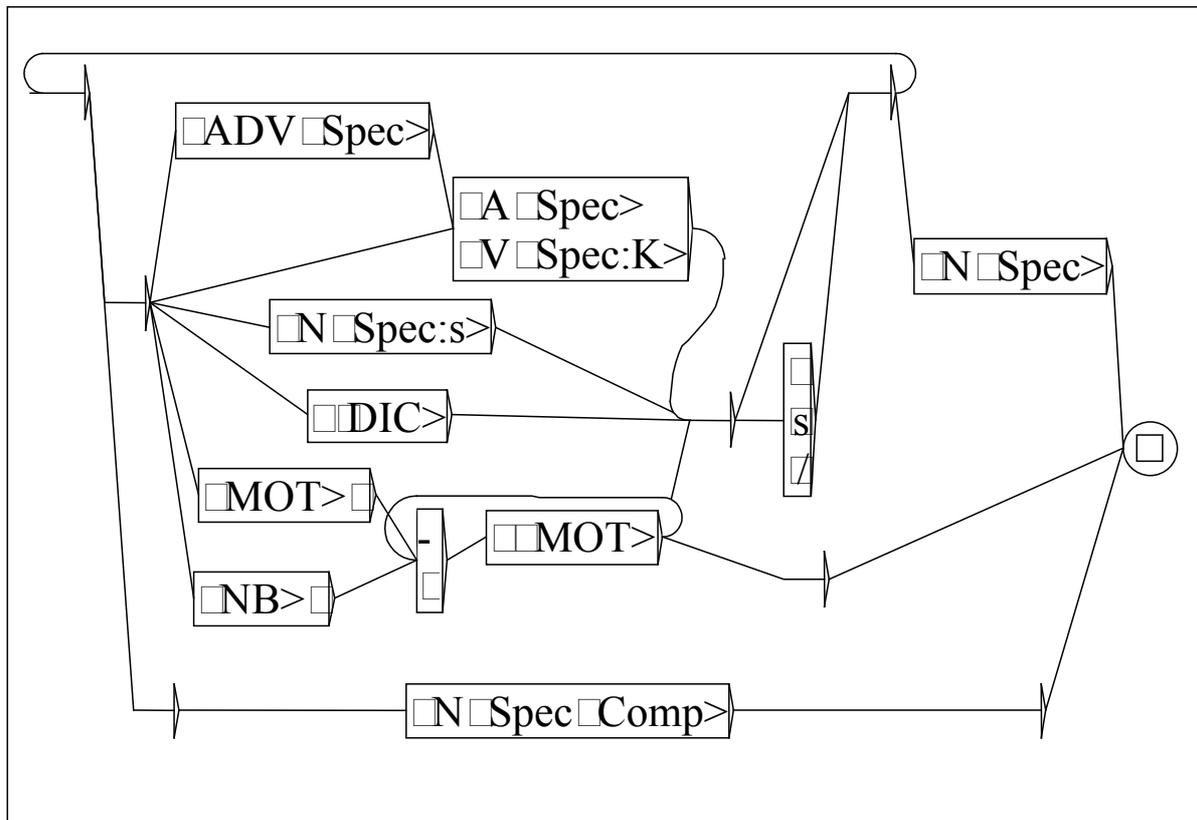


Fig.3. Schéma de fonctionnement de l'enrichissement terminologique

<N+Spec : s> car les termes du type □om□om...□om sont les plus nombreux en anglais. Par exemple

(5) data transmission control unit

est obtenu en passant une fois par le nœud <N+Spec:s> (pour reconnaître *data transmission* comme nom composé spécialisé au singulier) et une fois par <N+Spec> (pour *control unit*). Ainsi, nous disposons d'une méthode de recherche de nouvelles surcompositions.



**Fig.4.** Patron de recherche de termes

Le chemin supérieur contenant les étiquettes <ADV+Spec>, <A+Spec> et <V+Spec:K> permet de prendre en compte les adjectifs, les adverbes et les participes passés, comme dans

(6) *locally attached arrays*

Le nœud <!DIC> signifiant « tout mot non reconnu par les dictionnaires » permet de tenir compte des néologismes. C'est ici que l'on voit l'importance d'un DELAF général très complet. Si nous n'en n'avions pas ou s'il était trop petit, certains mots généraux seraient acceptés à tort comme parties de nouveaux termes. Mais puisque notre DELAF général est très exhaustif, nous pouvons être presque sûrs qu'un mot qui n'a pas été reconnu par les dictionnaires ni généraux ni spécialisés est un néologisme du domaine concerné par le texte traité. Le plus souvent c'est un nom propre du domaine (nom d'un logiciel, d'une marque, d'un fabricant etc.) ou bien un nouveau concept du domaine, ou encore une faute de frappe. Par exemple l'analyse du nouveau terme

(7) *powerup initialization sequence*

passé une fois par le nœud <!DIC> (néologisme *powerup*), une fois par le nœud <N+Spec:s> (*initialization*) et une fois par le nœud <N+Spec> (*sequence*).

La première estimation de l'efficacité de notre méthode a été faite sur un petit corpus informatique de 8500 mots. Ses résultats indiquent le taux de rappel (nombre de bons

candidats proposés par rapport au nombre de tous les termes existant dans le texte) de 82% et celui de précision (la proportion de bons termes parmi tous les candidats proposés) de 53%. Si ces résultats se confirment ceci correspondra à peu près aux objectifs de l'application, car peu de termes vont manquer dans le glossaire du texte et l'utilisateur ne va devoir rejeter qu'un terme sur deux de la liste des candidats.

## Conclusion

L'application du système Intex dans le domaine de la terminologie et traduction a permis de confirmer l'utilité du modèle à états finis dans la description de la langue. Les fonctions de lemmatisation et de correction orthographique de termes y ont trouvés une mise au point naturelle et efficace. Les travaux sur le module d'enrichissement terminologique ont démontré que les nouveaux termes sont souvent créés à partir soit des termes déjà existants, soit des néologismes - les deux cas peuvent être repérés grâce à l'utilisation des lexiques généraux et spécialisés à large couverture.

Agata Chrobot  
LADL, Université Paris 7  
Tour Centrale, 9e étage  
2, pl. Jussieu  
75251 Paris Cedex 05  
France  
chrobot@ladl.jussieu.fr

## Références

- Chrobot, Agata. 1999. Enrichissement terminologique en anglais fondé sur des dictionnaires généraux et spécialisés. In Condamines, A., Enguehard, Ch. (eds.) *Terminologies nouvelles. Terminologie et intelligence artificielle, actes du colloque TIA 99, 10-11 mai 1999. 19-20 décembre 1999 - juin 1999*, pp. 78-88, Nantes: IRIN, Toulouse: CNRS.
- Daciuk, Jan. 1998. Incremental Construction of Finite-State Automata and Transducers, and Their Use in the Natural Language Processing. Thèse de doctorat, Gdańsk: Technical University of Gdańsk.
- Damerau, F. J. 1964. A technique for computer detection and correction of spelling errors. *Communications of the ACM*, 7(3), pp. 171-176.
- Oflazer, Kemal. 1996. Error-tolerant finite state recognition with applications to morphological analysis and spelling correction. *Computational Linguistics*, 22(1), pp. 73-89.
- Silberztein, Max. 1997. Manuel de référence du logiciel INTEX. Habilitation à diriger des recherches en informatique. LADL, Paris: Université Paris 7.

## Summary

We present an application of some Intex functions in a computer aided translation tool LexPro CD Databank (made by a French society LCI). It is a large multilingual terminological database which integrates some natural language features - lemmatisation of terms, spelling checking, and terminological extraction - based on finite-state model of the lexicon.