Université Paris 7

UFR d'Informatique Fondamentale

Centre d'Etude et de Recherche en Informatique Linguistique
Laboratoire d'Automatique Documentaire et Linguistique

ANALYSE SYNTAXIQUE TRANSFORMATIONNELLE DU FRANCAIS PAR TRANSDUCTEURS ET LEXIQUE-GRAMMAIRE

Emmanuel ROCHE

Thèse de Doctorat d'Informatique Fondamentale

Directeur: Maurice Gross

Janvier 1993

Jury

M. Crochemore
J. Désarménien
M.Gross
F. Günthner
D. Perrin
J-M. Staeyert

REMERCIEMENTS

Je veux exprimer en premier lieu ma reconnaissance au professeur Maurice Gross qui m'a encouragé tout le long de ce travail et qui, avant tout, m'a fait aimer cette discipline.

Cette étude n'aurait jamais pu être menée à terme sans le bénéfice de l'environnement exeptionnel du LADL et du CERIL et sans le contact stimulant de leurs informaticiens et linguistes. En particulier les discussions avec Eric Laporte, Max Silberztein, Anne Monceaux, David Clemenceau, Dominique Revuz, Marc Zipstein et Mehryar Mohri, et les conseils bienveillants de Morris Salkoff et Christian Leclère m'ont guidé dans l'éclaircissement de plusieurs questions.

Chacun des membres du jury m'a fourni une aide précieuse, et je suis honoré de la présence de tous.

1. INTRODUCTION

L'utilisation systématique d'ordinateurs pour la production, le stockage et la diffusion des textes augmentent chaque jour le besoin d'outils adaptés à leur traitement. Parmi ceux-ci l'analyse grammaticale, ou syntaxique, tient une place particulière. Tout d'abord, l'analyse lexicale d'un texte conduit à le représenter avec un grand nombre d'ambiguïtés introduites par la consultation des dictionnnaires (M. Silberztein 1993); elle est donc un point de passage obligé pour un grand nombre d'autres applications, par exemple celles qui mettent en jeu des représentations sémantiques dérivées des structures syntaxiques. Malgré les efforts qui lui sont consacrés depuis plusieurs dizaines d'années, l'analyse syntaxique automatique ne donne aujoud'hui que des résultats encore trop approximatifs pour être véritablement utilisables. La principale difficulté vient de la complexité de la langue elle-même, difficulté aujourd'hui encore largement sous-estimée et que seule une étude linguistique profonde permet d'appréhender.

A de rares exceptions près, le cheminement de l'analyse automatique suit la création de modèles de grammaires formelles (GB, HPSG, etc.) (voir P. Miller, T. Torris 1990) censées refléter les mécanismes internes de la langue. Cependant, ces grammaires, construites pour résoudre un nombre d'exemples, certes de plus en plus étendu, mais toujours très petit, se heurtent toutes rapidement au problème de la représentation de cas linguistiques nouveaux. Cette évolution a eu la conséquence sociologique fâcheuse de séparer des informaticiens, disposant d'outils formels et informatiques, et les linguistes accumulant les descriptions. Elle a également entraîné des discussions formelles hors de propos du point de vue de l'étude ou du traitement de la langue. Cependant, la confrontation des programmes à des exemples concrets met facilement en évidence la distance à parcourir, et progressivement sont apparus des grammaires à vocation formelle plus modeste (*TAG lexicalisés* essentiellement, voir A. Joshi 1987 ou A. Abeillé 1991) qui permettent de mieux serrer les faits linguistiques. Nous pensons cependant que ce type de formalisme est encore trop riche, c'est-à-dire qu'il semble difficile à justifier complètement par des faits linguistiques. Nous avons donc utilisé le formalisme le plus simple et le plus neutre possible, celui des listes, listes que sont les dictionnaires morphologiques et syntaxiques.

Ces dictionnaires forment cependant des listes trop importantes pour être manipulées directement. La notion d'automate, et plus particulièrement celle de transducteur, permet de pallier à cet inconvénient en factorisant un très grand nombre d'informations communes à plusieurs entrées. Ce formalisme a permis par exemple de représenter une analyse morphologique du français incluant la morphologie dérivationnelle, et portant sur les 700.000 entrées d'un dictionnaire de formes fléchies par un transducteur de 1,3 MO (voir D. Revuz 1991, D. Clemenceau 1992, D. Clemenceau E. Roche 1992 et E. Roche 1992d). De même on a représenté un dictionnaire de mots composés de 150.000 entrées avec un automate. Nous avons représenté par un transducteur un dictionnaire syntaxique de plus de 2.000.000 d'entrées. Ce dictionnaire, dont la taille est appelée à augmenter très notablement est appelé *DELSYN*, le transducteur qui le représentera *f_DELSYN*.

Par ailleurs, de telles représentations conduisent à des programmes d'analyse extrêmement simples, l'analyse d'une phrase *Ph* consistant simplement à appliquer la fonction *f_DELSYN* cycliquement à un automate représentant *Ph* jusqu'à l'obtention d'un point fixe. L'application

cyclique d'une transduction peut en effet effectuer une analyse descendante d'identification de l'élément prédicatif de la phrase (le verbe en général) puis passer à l'analyse de chacun des arguments de cet élément (le sujet et les compléments le plus souvent). Mais une transduction peut également simuler une analyse de type ascendante qui reconnaît des segments de plus en plus grand ; enfin nous verrons qu'une transduction peut également appliquer des règles de grammaires locales dans une analyse qu'on peut dire *transversale* par rapport aux deux types précédents. Il résulte de ces trois possibilités que l'union de ces trois types de transductions permet l'utilisation d'un transducteur unique (*f_DELSYN* ici) effectuant l'analyse dans ces trois directions, simultanément de manière très naturelle.

Ce travail trouve sa source dans l'entreprise de description systématique du français menée depuis plus de vingt ans par les linguistes du LADL. Cette description consiste à étudier l'ensembles des phrases simples qui sont considérées comme des éléments syntaxiques et sémantiques minimaux. Cette liste de phrases simples, donc chaque entrée d'un dictionnaire syntaxique, constitue un ensemble appelé lexique-grammaire. Ce travail s'est fait sur la base théorique de Zellig Harris et de Maurice Gross 1968 1975. Notre travail a donc en partie constitué à regrouper l'ensemble de ces données pour les inclure dans une structure intégrée, autrement dit dans un seul dictionnaire (*DELSYN*) et à les traduire en automates et transducteurs. Il a fallu pour cela écrire un système (*PUPITRE*) de création et de manipulation d'automates et de transducteurs. Cet ensemble de commandes qui viennent s'ajouter aux commandes générales du système a la particularité d'accepter des objets de grande taille (nous avons rencontré des automates de l'ordre du million d'états) et de proposer une variété de structures de représentations. *PUPITRE* permet par exemple de manipuler des automates dont l'alphabet est un ensemble ordonné de centaines de milliers de mots (et non plus de simples caractères).

Par ailleurs, la précision des données linguistiques employées (12.000 verbes simples et 20.000 expressions figées) permet l'application de transformations syntaxiques au sens de Zellig Harris sur les résultats d'analyse. Par exemple, il est possible de mettre en évidence la phrase active correspondante à une phrase passive ou d'expliciter, lors de certaines pronominalisations, l'argument à l'origine d'un pronom. Cela permet aussi, dans le cas de certains groupes nominaux, de mettre en évidence des phrases sous-jacentes et ainsi de proposer un début d'analyse sémantique, rigoureusement fondée sur des faits syntaxiques formalisés. Le terme *transformationnel* donné dans le titre se justifie également par le fait qu'il est obligatoire, dans l'analyse de certaines phrases, d'appliquer des transformations à des stades intermédiaires de l'analyse, ces problèmes sont rarement abordés dans les descriptions d'analyseurs automatiques mais ils sont souvent incontournables.

LE PLAN

Nous prendrons tout d'abord (dans le chapitre 2) un exemple de phrase et nous montrerons comment cette phrase s'analyse grâce à la donnée des phrases simples qui la constituent. Cet exemple nous permettra également d'introduire le formalisme des automates et des transducteurs et de montrer comment se fait l'analyse par l'application cyclique d'un transducteur. A ce stade, la totalité du formalisme nécessaire à l'analyse aura été introduit.

Nous donnerons au chapitre 3 les détails de l'analyse morphologique, avec en particulier les descriptions des dictionnaires employés. Cela nous permettra également d'expliciter les structures sur lesquelles s'appliquera l'analyse syntaxique proprement dit.

Le chapitre 4, point central de cette étude, montre quelles sont les étapes à suivre pour construire le dictionnaire syntaxique *DELSYN*. Nous montrerons comment des exemples de phrases de plus en plus complexes conduisent à l'enrichissement de ce dictionnaire. Chaque étape sera constitué :

a. d'un exemple de phrase à analyser,

b. d'une présentation plus générale du problème linguistique,

c. de la solution adoptée pour l'analyse, c'est-à-dire du type de structure à inclure dans *DELSYN*,

d. d'une analyse d'une phrase de ce type et

e. de la présentation d'un extrait de *DELSYN* et du schéma de construction.

Le chapitre 5 montre que l'efficacité (en durée) de l'analyse peut être grandement améliorée par la notion de grammaire locale dont nous montrerons qu'elle s'intègre parfaitement dans le processus global de l'analyse.

Nous discuterons enfin rapidement au chapitre 6 de la manière dont les programmes sont implémentés, discussion qui sera reprise et précisée dans l'annexe principale qui est en fait le manuel d'utilisation du système *PUPITRE* de manipulation d'automates et de transducteurs.

2. PRESENTATION GENERALE DE L'ANALYSE

Cette section présente directement un processus d'analyse automatique. Nous y donnons l'idée générale de manière complète sans nous attarder cependant sur aucun détail. Tous les problèmes, ici simplement évoqués, seront repris au cours de la discussion.
Nous décrivons l'analyse d'un exemple simple au moyen d'une grammaire réduite à l'indispensable pour notre exemple. Le processus d'analyse apparaîtra d'emblée. Nous introduisons le formalisme que nous utiliserons, celui des transductions et des automates.

2.1. PRESENTATION SUR UN EXEMPLE

Le but est de construire une fonction qui, à une séquence de caractères représentant la phrase :

(1) *Que Pierre fasse cela agace Jean*

associe son analyse, c'est-à-dire, en première approximation :

(2) $(Que\ ((Pierre)_{Nhum0}\ (fasse)_{V0}\ (cela)_{N1})_{P}\ )_{QuP0}\ (\ gace)_{V0}\ (Jean)_{Nhum1}$ [1]

On a placé des parenthèses avec des étiquettes attachées à chaque parenthèse fermante. Plus précisément, cette notation est équivalente à dire que la phrase principale est de structure

$$QuP_0\ V_0\ Nhum_1$$

où V_0 =: *agace*, $Nhum_1$=: *Jean* et QuP_0 =: *Que P* où *P* est également une phrase. Cette seconde phrase *P* s'analyse à son tour en une structure

$$Nhum_0\ V_0\ N_1$$

où V_0 =: *fasse*, $Nhum_0$ =: *Pierre* et N_1 =: *cela.*

La grammaire dont on dispose pour analyser cette phrase est une liste de structures de phrases et, dans notre exemple, cette liste est celle de la figure 2.1a :

$$QuP_0\ V_0\text{-}\langle agace\rangle\ Nhum_1$$

$$Nhum_0\ V_0\text{-}\langle faire\rangle\ N_1$$

Liste E1 Premier exemple de DELSYN

Figure 2.1a

[1] Les étiquettes des parenthèses sont placées à droite de ces parenthèses en indice.

L'idée est alors de considérer cette liste, ou grammaire, comme une fonction. La première structure : $QuP_0\ V_0$-<*agace*> $Nhum_1$ définira la fonction f qui a une séquence du type :

(3) $u_1\ u_2\ ..\ u_n\ agace\ v_1\ ..\ v_p$

où les u_i et v_i sont des mots quelconques, associera la séquence :

(4) $(que\ (u_1\ ..\ u_n))_{QuP0}\ (agace)_{V0}\ (v_1\ ..\ v_p)_{Nhum1}$.

Ainsi, f associera (5) à (1) :

(5) $(que\ (Pierre\ fasse\ cela))_{QuP0}\ (agace)_{V0}\ (Jean)_{Nhum1}$

On introduit un marqueur [P] qui délimite entre deux de ses occurrences consécutives une phrase à analyser. La fonction que l'on vient de définir se précise donc de la manière suivante, à toute séquence :

(7) $u_1\ ..\ u_n$ [P] $que\ v_1\ ..\ v_p\ agace\ w_1\ ..\ w_p$ [P] $x_1\ ..\ x_q$

on associe la séquence :

(8) $u_1\ ..\ u_n\ (que\ ([P]v_1\ ..\ v_p\ [P]))_{QuP0}\ (agace)_{V0}\ (w_1\ ..\ w_p)_{Nhum1}\ x_1\ ..\ x_q$.

Revenons à notre grammaire E1, elle définit une fonction f dont on donne à présent la définition complète. Cette fonction f peut en effet se définir par les sept types de fonctions partielles de la figure 2.1b.

(9) m_1 [P] *que* m_2 *agace* m_3 [P] m_4
$\rightarrow m_1$ (*que* ([P] m_2 [P])P)$_{QuP0}$ (*agace*)$_{V0}$ ([N] m_3 [N])$_{Nhum1}$ m_4

(10) m_1 [P] m_2 *fasse* m_3 [P] m_4
$\rightarrow m_1$ ([N] m_2 [N])$_{Nhum}$ (*fasse*)$_{V0}$ ([N] m_3 [N])$_{N1}$ m_4

(11) m_1 [N] *Pierre* [N] $m_2 \rightarrow m_1$ *Pierre* m_2

(12) m_1 [N] *Jean* [N] $m_2 \rightarrow m_1$ *Jean* m_2

(13) m_1 [N] *cela* [N] $m_2 \rightarrow m_1$ *cela* m_2

(14) $m_1 \rightarrow m_1$ *si* m_1 *ne contient ni* [P], [N]

(15) $m_1 \rightarrow \varnothing$ *dans tous les autres cas*

m_1, m_2, m_3 *et* m_4 *sont des séquences quelconques non vides de mots.*

Définition de la fonction f Première définition de f_DELSYN

Figure 2.1b

Maintenant revenons à la phrase de départ (1) que nous écrivons

(1) [P] *Que Pierre fasse cela agace Jean* [P]

ce qui se comprend comme "La séquence entre [P] et [P] est à analyser comme une phrase".

Notons $S_1 = \{(1)\}$ l'ensemble contenant l'unique élément (1). Si on applique une fois f à S_1 (c'est-à-dire à (1)) on peut vérifier que l'on obtient l'ensemble S_2 des séquences (2) et (3) :

$$S_2 = \{(2),(3)\} = f(S_1)$$

(2) (*que* ([P] *Pierre fasse cela* [P])$_P$)$_{QuP0}$ (*agace*)$_{V0}$ ([N] *Jean* [N])$_{Nhum1}$

(3) ([N] *que Pierre* [N])$_{Nhum0}$ (*fasse*)$_{V0}$ ([N] *cela agace Jean* [N])$_{N1}$

En effet (1) a la forme d'un élément de départ pour les fonctions partielles (9) et (10), l'image de (1) par (9) donne (2) et l'image de (1) par (10) donne (2).

Si on réapplique f à S_2 on obtient

$$S_3=\{(4)\}=f(S_2)=f^2(S_1)$$

(4) (*que* (([N] *Pierre* [N])$_{Nhum0}$ (*fasse*)$_{V0}$ ([N] *cela* [N])$_{N1}$)$_{P}$)$_{QuP0}$ (*agace*)$_{V0}$ (*Jean*)$_{Nhum1}$

En effet les associations (10) et (12) (reconnaissance du groupe nominal *Jean*) s'appliquent en même temps à (2) pour donner (4). Par ailleurs [N] *que Pierre* [N] vérifie la condition de (15) et donc (3) a une image vide par (15) et donc par *f*. Cela s'interprète en disant que l'analyse de la phrase avec *fasse* comme verbe principal a échoué.

Si on applique *f* à S_3 on obtient le singleton S_4 suivant:

$$S_4 = \{(5)\} = f(S_3) = f^3(S_1)$$

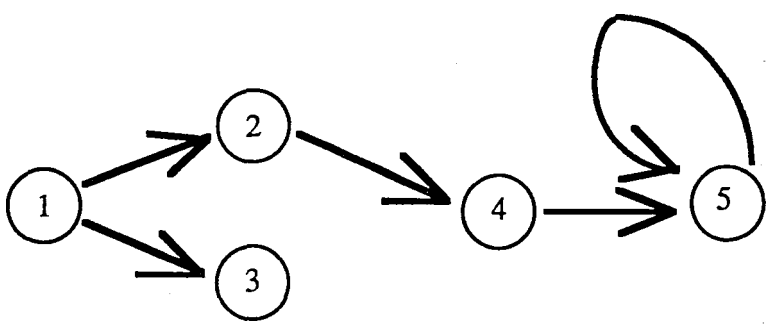
(5) (*que* ((*Pierre*)$_{Nhum0}$ (*fasse*)$_{V0}$ (*cela*)$_{N1}$)$_{P}$)$_{QuP0}$ (*agace*)$_{V0}$ (*Jean*)$_{Nhum1}$

par les associations (11) et (13).

Enfin *f* appliqué à S_4 redonne S_4 (association (14)) qui est donc le point fixe de $(f^n(S_1))_n$. On remarquera que ce point fixe existe pour toute séquence finie de mots et que l'on définit donc intuitivement la fonction d'analyse F par :

$F(S)$ est le point fixe de $(f^n(S))_n$.

L'analyse de notre exemple peut donc se schématiser en la série de transitions de la figure 2.1c

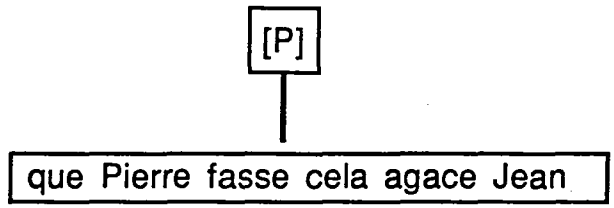


Vue générale des applications successives de f

Figure 2.1c

2.2. REPRESENTATION PAR ARBRES

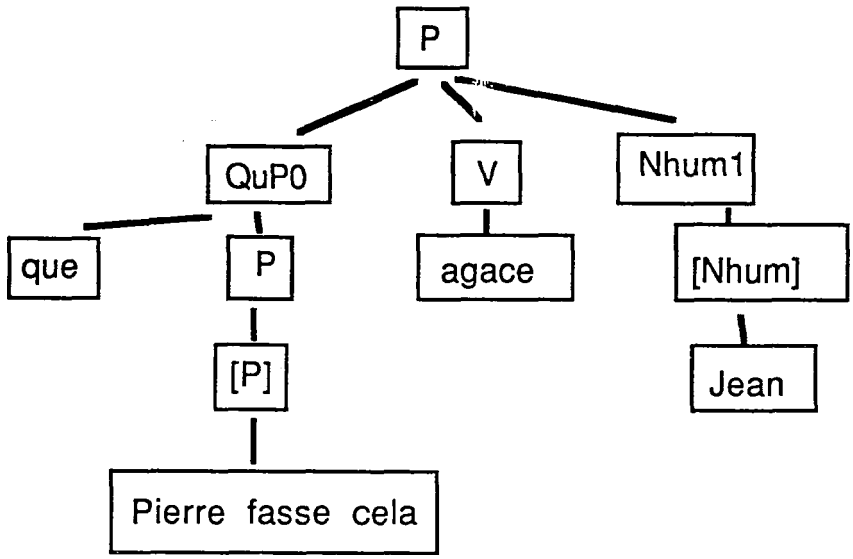
Le processus que nous venons de décrire et qui consiste à rajouter des parenthèses au fur et à mesure de l'analyse peut aussi se voir comme un processus de construction d'arbre. Cela le rapproche des méthodes d'analyse traditionnellement employées. Reprenons le même élément de départ (1) qui devient :



arbre(1') ≡ (1)

Figure 2.2a

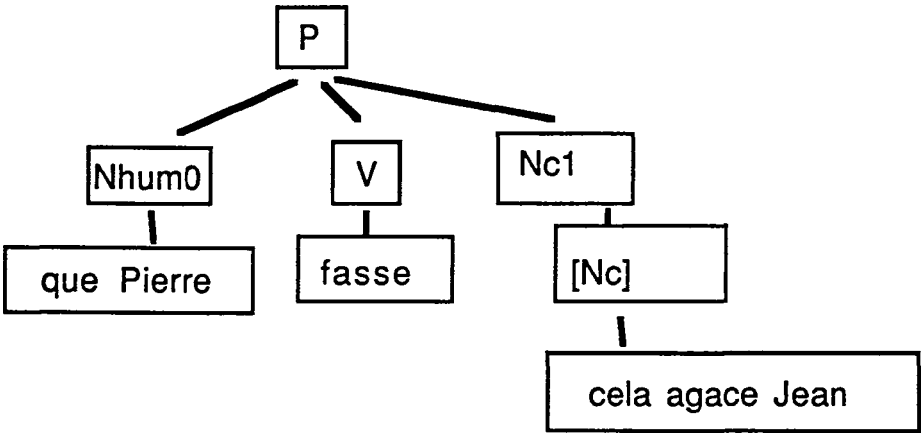
Cet arbre (1') correspondant à (1). On interprète alors les arbres avec une racine de type [P] ou [N] comme ayant comme des points d'interrogations typés. Ici par exemple, le symbole [P] sera le point d'interrogation typé signifiant "est-ce que la feuille ci-dessous est bien une phrase ?".



arbre (2') ≡ (2)

Figure 2.2b

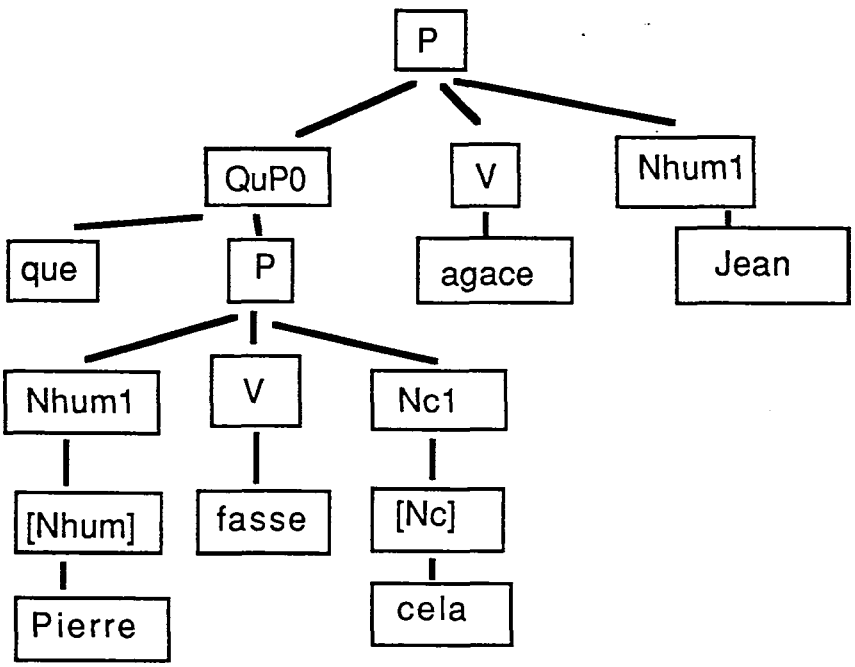
L'arbre (2') correspond à (2). La structure de phrase a été proposée mais il reste à analyser la *que*-phrase et un groupe nominal de type *Nhum*.



arbre (3') ≡ (3)

Figure 2.2c

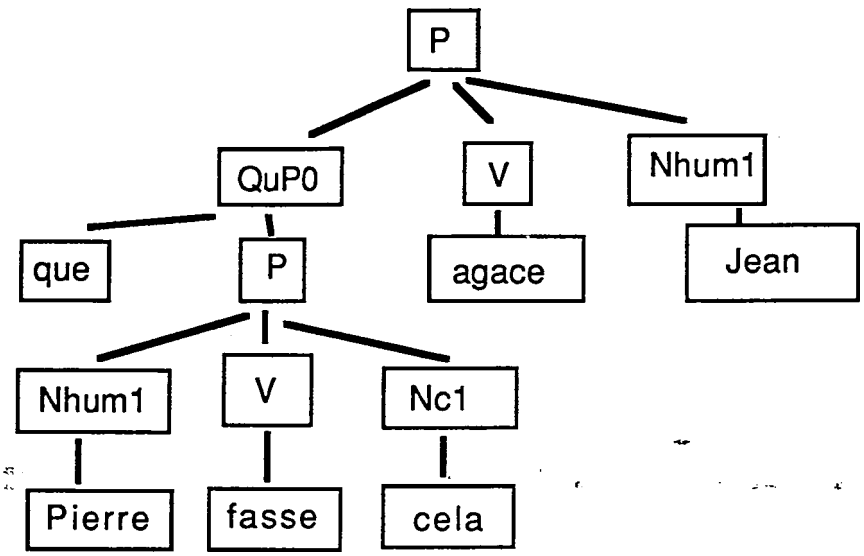
Pour cette phrase, une structure incorrecte est proposée. Elle sera détruite à l'étape suivante de l'analyse pour la raison que, par exemple, *que Pierre* ne peut constituer un groupe nominal.



arbre (4') ≡ (4)

Figure 2.2d

Cet arbre est l'image de (2') par *f*, une structure pour la *que*-phrase à été proposée et *Jean* a été reconnu comme un groupe nominal. *Pierre* et *cela* restent à analyser.



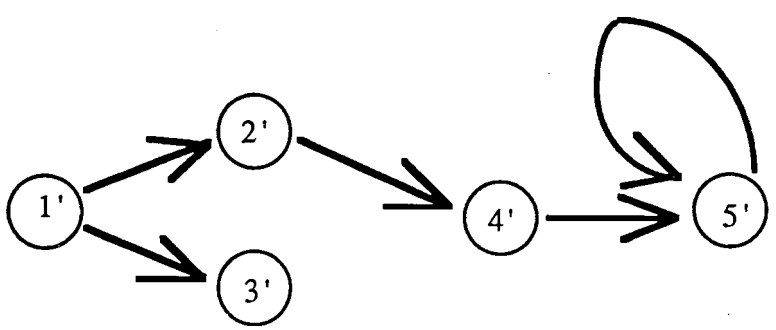
arbre (5') ≡ (5)

Figure 2.2e

L'arbre (5'), strictement équivalent à (5), est donc le résultat de l'analyse.

Cette approche éclaire le mécanisme d'analyse suivit en 2.1 et met en évidence une stratégie *Top-Down* . Cependant, cette représentation se heurte rapidement au problème de la gestion des ambiguïtés.

De la même manière que précédemment nous pouvons résumer l'analyse dans un schéma exactement parallèle à celui du paragraphe précédent.



Vue générale du processus d'analyse

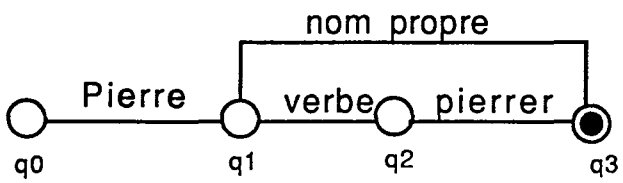
Figure 2.2f

2.3. REPRESENTATION PAR AUTOMATES ET TRANSDUCTEURS

Nous allons maintenant expliquer pourquoi nous n'utilisons pas des familles d'arbres mais plutôt le formalisme de représentation des automates et des transducteurs. Nous commencerons par donner un bref aperçu de leur définition avant de détailler le formalisme que nous utiliserons tout au long de cet étude, l'exemple 2.1.

2.3.1. Définitions

Remarquons tout d'abord si on recherche la forme graphique *Pierre* de notre exemple dans un dictionnaire, le résultat est ambigu. Le mot peut être soit le nom propre masculin soit une forme conjuguée du verbe *pierrer*. Une façon naturelle de représenter cette ambiguïté est d'utiliser un graphe tel que celui de la figure 2.3a :



Automate A_1

Figure 2.3a

Ce type de graphe est appelé AUTOMATE. Un automate est définit par un quintuplet $(Alph,Q,i,F,d)$ où l'alphabet *Alph* de l'automate est l'ensemble des mots qui peuvent apparaître sur des jonctions (appelées transitions), pour A_1,

$$Alph = \{Pierre, verbe, pierrer, nom\ propre\}.$$

Q est l'ensemble des états (notés par des cercles), pour A_1 on a $Q = \{q_0, q_1, q_2, q_3\}$. i est l'état initial ($i = q_0$ pour A_1) et F est l'ensemble des états terminaux ($F=\{q_3\}$ pour A_1). Enfin, d est la fonction[2] de transition qui fait passer d'un état à un autre en fonction des étiquettes, plus précisément, d est une fonction de Q^*Alph dans Q qui pour A_1 vaut

$d(q_0, Pierre) = q_1$
$d(q_1, verbe) = q_2$
$d(q_1, nom\ propre) = q_3$
$d(q_2, pierrer) = q_3$
et $d(q_i,m) = \varnothing$ *sinon.*

Cet automate définit l'ensemble L_1={*Pierre verbe pierrer, Pierre nom propre*} de la manière

[2]On se limite pour l'instant aux automates déterministes.

suivante: Si on prend la séquence

(1) *Pierre verbe pierrer*

et que l'on se positionne sur l'état initial $i = q_0$, en comparant les transitions avec le premier mot de la séquence on note que $d(q_0, Pierre) = q_1$, on peut donc se positionner sur q_1. On compare alors le deuxième symbole *verbe* avec les transitions de q_1 ce qui nous fait passer à q_2. De la même manière on passe à q_3. A ce point on a lu toute la séquence (1) et on est positionné sur q_3 qui est un élément de l'ensemble F des états terminaux. Par définition la séquence est bien élément de L_1. Avec la séquence de trois symboles

(2) *Pierre verbe nom*

en revanche on reste bloqué à l'état q_2, cette séquence n'est donc pas élément de L_1. On peut ainsi vérifier que l'ensemble L_1 est bien définit par l'automate, on dira que L_1 est le langage (ou l'ensemble) reconnu par A_1 et on notera $L_1=L(A_1)$. Nous nous limiterons ici à cette définition rapide.

Les automates sont encore insuffisants pour modéliser une fonction comme la fonction f de la section 2.1, il est nécessaire d'introduire un formalisme un peu plus riche : celui des TRANSDUCTEURS. L'automate est adapté à la représentation de certains ensembles et nous allons voir que certaines fonctions se représentent facilement par des transducteurs. Avant de prendre cette fonction f, prenons la fonction plus simple f_2 qui à la séquence :

(3) *Pierre / mange*

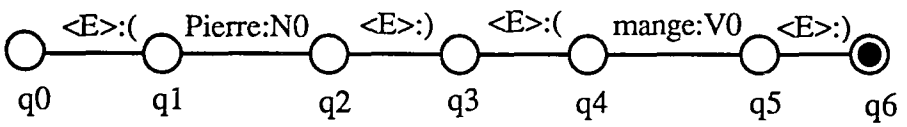
associe la séquence parenthésée (4) :

(4) $(N_0)(V_0)$

dont nous précisons la décomposition en mots au moyen du séparateur de mots '/':

(5) $(/N_0/)/(/V_0/)$

Cette définition équivaut à introduire l'automate de la figure 2.3b.



Transducteur T_1

Figure 2.3b

L'automate a ceci de particulier que chaque étiquette est un couple de mots, c'est ce type d'automate qu'on appel transducteur. Un transducteur est défini par un sextuplet

(Alph$_1$*, Alph*$_2$*, Q, i, F, d)* où Q, i et F ont la même signification que pour les automates et où *Alph1* est l'alphabet gauche, c'est à dire l'ensemble des étiquettes gauches donc des mots qui apparaissent à gauche des ':'. Pour T_1, $Alph_1$ = {*<E>, Pierre, mange*}. De la même manière $Alph_2$ est l'alphabet droit, donc l'ensemble des labels droits, pour T_1, $Alph_2$ vaut {(, N_0,), V_0}. La fonction de transition *d* associe à chaque couple état-lettre un ensemble de couple état d'arrivée-lettre émise. Plus précisément *d* : *Q*Alph*$_1$*->P(Alph*$_2$**Q)* est définie pour T_1 par

$d(q_0, <E>)$ = { ('(', q_1) }
$d(q_1, Pierre)$ = { (N_0, q_2) }
$d(q_2, <E>)$ = { (')', q_3) }
$d(q_3, <E>)$ = { ('(', q_4) }
$d(q_4, mange)$ = { (V_0, q_5) }
$d(q_5, <E>)$ = { (')', q_6) }

On peut appliquer un transducteur à une séquence de mots de la manière suivante : prenons la séquence s_1 = (3) et le transducteur T_1.; s_1 est équivalent à s_2 :

s_2 *<E>/Pierre/<E>/<E>/mange/<E>*

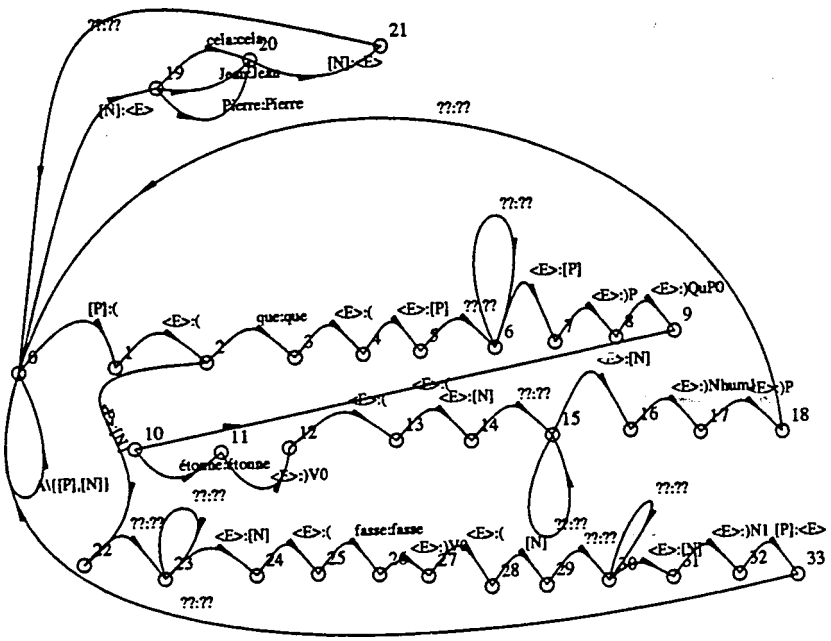
On a simplement inséré dans s1 le mot vide noté *<E>*. Si on se positionne sur l'état initial $i = q_0$ de T_1 avec s_2 et qu'on compare l'étiquette gauche de la transition avec le premier mot de s_2 (c'est à dire "<E>") on constate qu'ils sont identiques, on peut donc, comme pour un automate passer à l'état suivant. Cependant on mémorise le label gauche dans *r* qui vaut donc "(". On réitère cette opération en q_1 en comparant le label gauche *Pierre* avec le deuxième mot de s_2, *Pierre* également. On passe donc en q_3 avec en plus *r* = "(/ N_0". On réitère cette opération jusqu'au dernier mot de s_2, ce qui mène en q_6 qui est terminal, s_2 (donc s_1) était bien élément de l'ensemble de départ et l'image est mémorisée dans

$$r = \text{"(/ } N_0 \text{ /) / (/ } V_0 \text{ /)"}.$$

On vérifie ainsi que T_1 définie bien la fonction f_2.

2.3.2. Analyse de l'exemple par transducteur

Reprenons maintenant la fonction *f* définie en 2.1. Cette fonction est également une transduction car elle est représentable par le transducteur T_2 de la figure 2.3c.

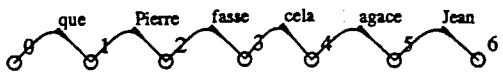


Transducteur T_2 **Première représentation de *f_DELSYN* par transducteur**

Figure 2.3c

Ce transducteur est certes plus complexe que T_1 mais il respecte le même formalisme, la manière d'appliquer le transducteur est la même. Nous utilisons en plus les transitions "??:??" qui indiquent que n'importe quel mot est accepté comme label gauche et que le label droit stocké est alors le même que le label gauche proposé. Par exemple *??:??* appliqué à *Pierre* donne *Pierre*. De plus la transition A\{[P],[N]}:id signifie que tout mot de l'alphabet sauf [P] et [N] est accepté comme étiquette gauche, l'émission stockée dans *r* est alors le mot lui même. L'état initial de ce transducteur est q_0 et $F = \{q_0\}$.

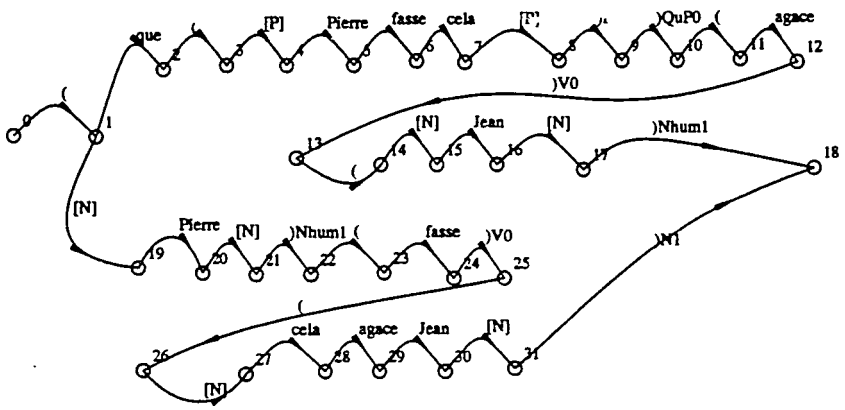
Plutôt que d'appliquer le transducteur à la séquence (1) nous allons l'appliquer à l'automate équivalent A_2 de la figure 2.3d



Automate A_2 représentant la phrase (1)

Figure 2.3d

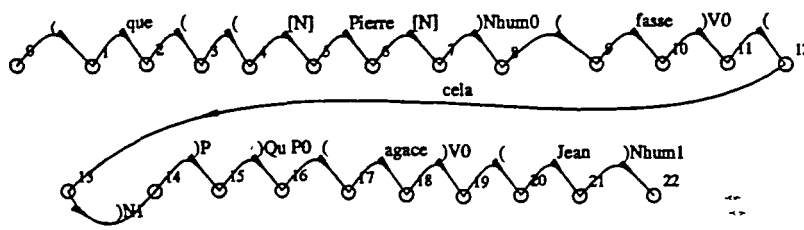
L'application de T_2 à A_2 consiste à appliquer la transduction T_2 à chacun des chemins représentés par A_2 (il n'y en a qu'un ici), le résultat est alors un ensemble de séquences qui peut également se représenter par un automate.



Automate $A_4 = T_2(A_2) = f(S_1)$

Figure 2.3e

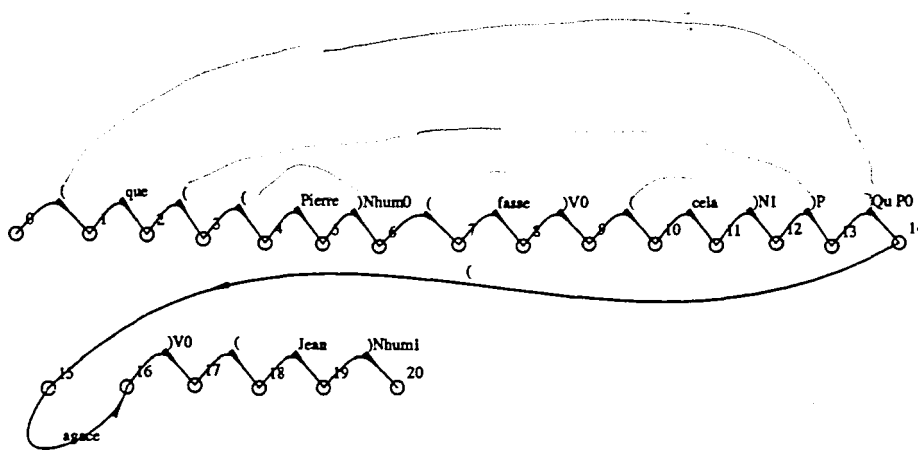
On a alors $f(L(A_2) = L(A_4) = L(T(A_2))$ qui montre que la définition ensembliste de la fonction correspond à l'application du transducteur à l'automate. Par rapport à la section 2.1 nous avons calculé $S_2 = f(S_1)$ puisque $S_1 = L(A_2)$. De même qu'en 2.1 et 2.2 on réitère l'opération pour obtenir l'automate A_5 tel que $L(A_5) = S_3 = f^2(S_1)$ de l'automate $T_2{}^2(A_2) = T_2(A_4)$ de la figure 2.3f.



Automate $A_5=T_2^2(A_2)=T_2(A_4)$ de $f^2(S_1)$

Figure 2.3f

ıtère encore T_2 pour obtenir $A_6 = T_2(A_5)$ de la fig..e 2.3g



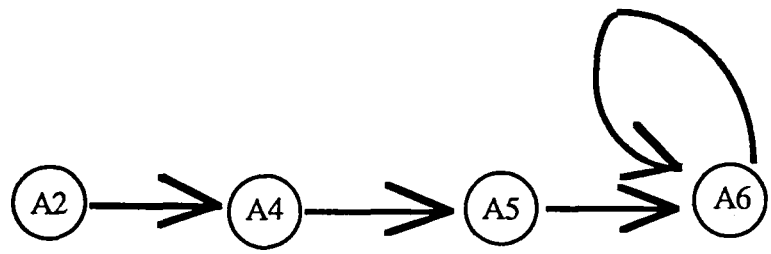
Automate $A_6=T_2^3(A_2)$ correspondant à $f^3(S_1)$

Figure 2.3g

On peut vérifier que $T_2(A_6) = A_6$ et par conséquent A_6 représente l'analyse de S_1 recherchée. La transduction T_2 représente donc la grammaire et le mécanisme nécessaire à l'application de cette grammaire à des phrases. La section 3 montrera non seulement que l'utilisation de transduction permet une analyse simple et rapide, mais qu'il est également facile d'utiliser ce formalisme pour écrire les grammaires.

De la même manière que pour les deux exemples précédents, on peut résumer l'analyse dans un

schéma tel que celui de la figure 2.3h.



Processus d'analyse sur l'automate A_2 de la phrase

Figure 2.3h

2.4. EXTENSION DE LA TRANSDUCTION A LA RECOPIE

Ce paragraphe décrit une extension des transductions à une analyse de certaines structures plus complexes. Cette présentation est plus technique que les précédentes, elle peut être ignorée en première lecture.

Pour l'instant, les structures que nous proposons sont traduites en des fonctions qui sont des transductions, c'est-à-dire qui se représentent par des graphes comme celui donné de la figure 2.3c. Cela implique en particulier que la famille des formes analysées no comporte pas de changement d'ordre dans ses éléments. Par exemple, il n'est pas possible de construire de cette manière une fonction qui associerait deux formes :

$$a\ u_1\ ..\ u_n\ b\ c$$

$$a\ u_1\ ..\ u_n\ b\ u_1\ ..\ u_n\ c$$

où $u_1\ ..\ u_n$ est une séquence de mots quelconque. Aussi, prenons l'exemple des verbes qui admettent une infinitive sujet, comme *satisfaire* dans

(1) *manger des bananes satisfait Paul.*

dont la structure est :

(2) $V_1{}^1W\ V_0\ N_1$

Le premier argument du verbe, *manger des bananes,* est une phrase dont le verbe est à l'infinitif et dont le sujet est absent. Mais le verbe principal (i.e. *satisfaire*) impose que le sujet de cette infinitive soit également le complément d'objet direct de ce verbe. Ici donc *Paul* doit être le sujet de *manger des bananes.*
La stratégie que nous avons adoptée jusqu'à présent consiste à proposer d'abord la structure de

la phrase principale, c'est-à-dire (2) puis à analyser de manière indépendante chacun des arguments. Dans cette deuxième étape on est donc conduit à vérifier que *manger des bananes* est une infinitive bien formée, cela suppose entre autre que l'on est capable d'en identifier le sujet. Or ce sujet est a priori détaché de ce segment. C'est-à-dire que la première passe d'analyse de (1) donne :

(3) (*manger des bananes*)$_{V1}{}^{0}{}_{W}$ (*satisfait*)$_{V0}$ (*Paul*)$_{N1}$

et que la deuxième passe cherche à analyser :

(4) (*manger des bananes*)$_{V1}{}^{0}{}_{W}$

de manière indépendante du reste, ce qui est impossible puisque le sujet est inaccessible. La solution consiste, lors de la première passe, à associer à (1) non pas (3) mais (5) :

(5) (*#Paul# manger des bananes*)$_{V1}{}^{0}{}_{W}$ (*satisfait*)$_{V0}$ (*Paul*)$_{N1}$

qui conduit la seconde passe à analyser :

(6) (*#Paul# manger des bananes*)$_{V1}{}^{0}{}_{W}$

comme une infinitive. Cette analyse se fait alors comme l'analyse d'une phrase habituelle en considérant simplement le verbe à l'infinitif comme un verbe conjugué.
Une fonction qui permet de suivre ce cheminement serai donc du type :

$$m_1\ [\mathrm{P}]\ u_1\ ..\ u_n\ \textit{satisfait}\ v_1\ ..\ v_p\ [\mathrm{P}]\ m_2$$
$$\rightarrow$$
$$m_1\ [\mathrm{Pinf}]\ v_1\ ..\ v_p\ u_1\ ..\ u_n\ [\mathrm{Pinf}]\ \textit{satisfait}\ [\mathrm{N}]\ v_1\ ..\ v_p\ [\mathrm{N}]\ m_2$$

où m_i, u_i, v_i sont des mots quelconques.

Nous sommes donc conduit à étendre le formalisme des transduction à de telles fonction.Cette extension nous fournira l'ensemble des structures et des opérations nécessaires pour la conduite de l'ensemble de l'analyse. Mis à part cette extension, nous n'introduirons plus aucun formalisme supplémentaire.

Prenons un exemple d'une telle fonction, nous noterons les chemins du graphe qui la représente de la manière suivante:

m:m' /pi:<E>/ 1:<E>/ a:b/c:d/fpi:<E>/n:n/ po:<E>/1:<E>/a:b/c:e/fpo:<E>/p:p'

où les symboles *pi, fpi, po, fpo* ont des significations particulières. *pi* marque le début de la zone qui sera à recopier, le nombre qui suit (*1* ici) donne un nom à cette zone, la fin de cette zone est marquée par le symbole *fpi*. Prenons l'exemple de la séquence :

$$S = m\ a\ c\ n\ p$$

la première lettre $S[1]=m$ se voit associé *m'*. Les symboles *pi fpi* et le nombre *1* sont dans un premier temps ignorés. On associe donc *b d* à *a c* pour obtenir

$$S_1 = m'\ b\ d$$

puis de la même manière :

$$S_2 = m'\ b\ d\ n'$$

po marque le début de l'emplacement auquel est recopié la séquence qui apparaît entre *pi* et *fpi* . Le *1* qui suit *po* indique que les données à recopier sont celles entre *pi 1* et *fpi*. Par conséquent, la suite de l'application entre *po 1* et *fpo* signifie que l'on doit prendre pour symbole d'entrée les même que ceux apparus entre *pi 1* et *fpi*, c'est-à-dire *a c*. On leur applique la transduction notée entre *po 1* et *fpo*, pour obtenir *b e*. On est donc au point où l'image vaut

$$S_3 = m'\ b\ d\ n\ b\ e$$

et il reste à appliquer *p:p'* pour obtenir le résultat final

$$S_4 = m'\ b\ d\ n\ b\ e\ p'$$

Reprenons maintenant le problème de la représentation de la fonction qui à (1) associe (5). Si on inclut des marqueurs morphologiques dans (1) pour noter que *manger* est une forme infinitive, que *satisfait* est une forme conjuguée et que *Paul* est un nom, on obtient (7) :

(7) *V-inf manger des bananes V-t satisfait N Paul*

La fonction suivante qui respecte le formalisme que nous venons d'exposer :

<E>:(/<E>:#/po:<E>/1:<E>/N:N/??:??/fpo:<E>/<E>:#/V-inf:V-inf/??:??/<E>:)$V_1{}^0W$*/V-t:V-t/ ??:?? /<E>:(/ pi:<E>/ 1:<E>/ N:N/ ??:??/ fpi:<E>/ <E>:)N1*

s'applique à (7) et donne très exactement (8) :

(8) *(#N Paul# V-inf manger des bananes)*$_{V1}{}^0{}_W$ *V-t satisfait (N Paul)*$_{N1}$

sur laquelle l'analyse peut se poursuivre.

2.5. LES TRANSFORMATIONS

Nous allons maintenant voir que le formalisme précédent permet d'appliquer effectivement des transformations syntaxiques sur le résultat des analyses. Reprenons l'exemple de la phrase :

(1) *Que Pierre fasse cela agace Jean*

dont nous avons vu que l'analyse est (nous simplifions un peu) :

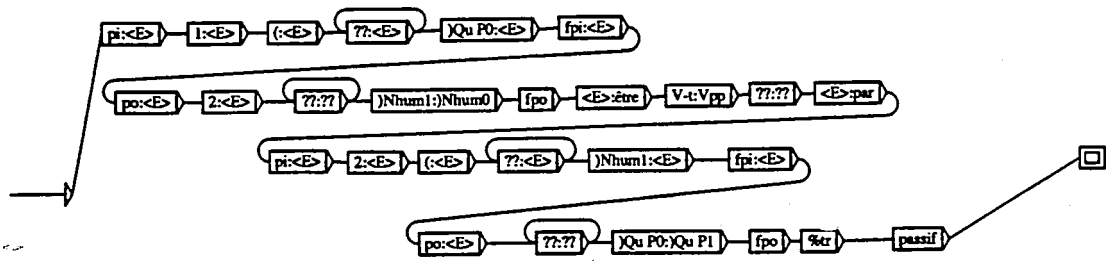
(2) *(que Pierre V-t faire cela)*$_{Qu\ P0}$ *V-t agacer (Jean)*$_{Nhum1}$

La phrase (1) est reliée par la transformation passive à la phrase :

Nhum$_0$ *être Vpp-<agacer-0> par le fait Qu* P_1 :=

(3) *Jean est agacé par le fait que Pierre fasse cela*

On peut vérifier que la fonction représentée sur la figure 2.5a



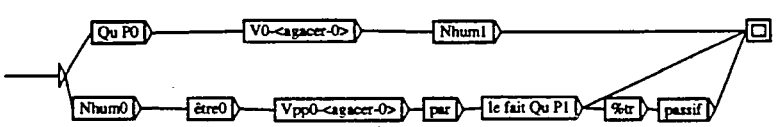
Extrait de la transformation passive *f_passif*

Figure 2.5a

s'applique au résultat d'analyse (2) pour donner (5) :

(5) $(Jean)_{Nhum0}$ *être Vpp agacer par le fait* $(que\ Pierre\ V\text{-}t\ faire\ cela)_{Qu\ P1}$ %tr passif

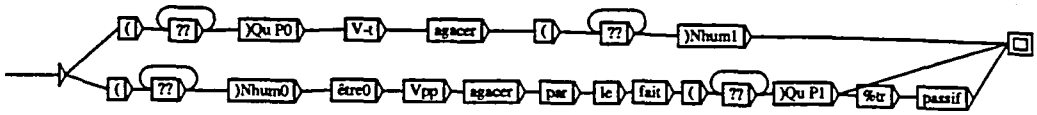
Par ailleurs (4) est répertoriée comme une structure possible de *agacer*, c'est-à-dire que l'automate de la figure 2.5b :



Extrait des structures de agacer, automate inclus dans DELSYN

Figure 2.5b

est équivalent, à un changement de notation près, à l'automate *SYN*(*agace*) de la figure 2.5c.



Extrait de l'automate *SYN(agacer)* ⊆ *DELSYN* des structures du verbe *agacer*

Figure 2.5c

On peut alors vérifier que (5) est bien une structure correcte pour le verbe *agacer* en faisant l'intersection de (5) avec *SYN(agacer)*, c'est à dire que nous cherchons l'ensemble des séquences communes à (5) et à *SYN(agace)*. Le résultat de cette intersection est ici la séquence (5), qui est bien le résultat de la transformation passive de (2).

La transformation passive est donc définie par le couple *(DELSYN f_passif)* où *DELSYN* est la liste des structures possibles (par exemple 2.5c). On dira que S_2 est la transformée de S_1 si les trois points suivants sont vérifiés :

1. S_1 appartient aux structures d'un verbe, plus précisément $S_1 \in DELSYN$
2. $S_2 \in f\_passif\ (S_1)$
3. $S_2 \in DELSYN$

ce qui conduit à une définition formelle d'une transformation syntaxique.

2.6. LE LEXIQUE SYNTAXIQUE : DELSYN

Dans notre exemple, la fonction f construite pour l'analyse provient des données contenues dans E1:

$QuP_0\ V_0$-<*étonne*> $Nhum_1$
$Nhum_0\ V_0$-<*faire*> $Nhum_1$

C'est la liste des structures de phrases simples permises par notre grammaire. Nous allons montrer que cette liste peut-être étendue à l'ensemble des phrases simples du français. Nous appelerons cette liste lexique : en effet elle décrit un ensemble en extension sans associer d'information à chaque entrée de ce lexique[3]. Un extrait d'un tel lexique, que nous appelerons *DELSYN*, est représenté par la figure 2.6a :

[3]Nous appellerons dictionnaire un lexique qui comporte des informations pour les entrées.

QuP_0 V_0-<étonner-0> $Nhum_1$
$Nhum_0$ V_0-<étonner-0> $Nhum_1$
IL_1 LE_1 V_0-<étonner-0>
V^1W V_0-<énerver-0> $Nhum_1$
$Nhum_0$ V_0-<énerver-0> $Nhum_1$ contre $Nhum_2$
$Nhum_0$ $Vsup_0$-<faire-3> Npred-<attention-1> à $Nhum_1$
$Nhum_0$ V_0-<dire-1> QuP_1 à $Nhum_2$
$Nhum_0$ V_0-<dire-1> QuP_1 de $Nhum_2$

Extrait de DELSYN

Figure 2.6a

Le coeur du problème, une fois que l'on a décidé du formalisme utilisé, consiste à trouver les structures qui doivent faire partie de ce dictionnaire et celles qui au contraire n'ont pas de raison de s'y trouver. Pour ces dernières en effet, l'analyse des phrases qu'elles représentent peut être faite par l'utilisation conjointe de structures plus simples présentes par ailleurs dans *DELSYN*.

2.7. CONSTRUCTION DE DELSYN

Nous verrons que pour construire un tel dictionnaire il est possible d'utiliser la représentation par tables syntaxiques du lexique-grammaire. Le principe de construction d'un tel dictionnaire est double :

1. Transformer des tables syntaxiques décrivant un ensemble d'entrées en une série d'automates équivalents. On fait alors l'union de ces automates pour obtenir un automate représentant une liste comme celle de la figure 2.6a.

2. Construire des transductions qui, appliquées aux automates précédents, engendrent des structures à ajouter à *DELSYN*.

Le résultat de ce travail est donc un dictionnaire (ou lexique syntaxique) tel que nous en avons représenté un sur la figure 2.6a.

La seconde partie du travail consiste à transformer ce dictionnaire en une transduction équivalente. Cette transduction, *f_DELSYN*, du type de la fonction de la figure 2.3c, permet une analyse des phrases qui suit la procédure de 2.1, 2.2 et 2.3. Le passage de *DELSYN* à *f_DELSYN* se fait par une transduction *ABREV* (pour abréviations) qui traduit chaque symbole (*$Nhum_0$* par exemple) en une partie de transduction.

Nous détaillerons ces étapes dans le paragraphe 4. Pour chaque exemple de phrase abordé nous donnerons la liste des structures à ajouter à *DELSYN* et la procédure à suivre pour transformer les tables syntaxiques en cette liste de structures ainsi que les transductions à construire pour engendrer les structures encore manquantes. Puis nous donnerons des extraits pertinents de la transduction *ABREV* qui permet de passer de *DELSYN* à *f_DELSYN*. Nous reprendrons alors l'analyse d'une phrase de ce type par l'application de *f_DELSYN*.

3. ANALYSE MORPHOLOGIQUE

On ne dispose pour l'instant que de *DELSYN*. Il nous faut donc rapprocher des structures du type

$$QuP_0\ V_0\text{-}\langle agacer\text{-}0\rangle\ Nhum_1$$

de phrases réelles. Un problème particulier est de rapprocher les formes canoniques des verbes qui apparaissent dans les structures des formes fléchies telles qu'on les trouvent dans les textes. Ainsi il faut pouvoir dire que *agacions* est une forme fléchie du verbe dont l'infinitif est *agacer*. Il faut également pouvoir établir que *agacions* est une forme conjuguée de la première personne du pluriel à l'imparfait. Un travail similaire est à effectuer pour les noms et les adjectifs, autrement dit, il faut pouvoir reconnaître toutes les formes fléchies au pluriel et/ou au féminin. Nous verrons que cette étape est essentiellement réalisée par des dictionnaires qui décrivent ces phénomènes de flexion de manière exhaustive. Le module d'analyse morphologique se compose d'un dictionnaire de mots simples, d'un dictionnaire de mots composés et d'un module d'analyse de mots complexes tels que *redémobilisation* qui relèvent de la morphologie dérivationnelle. Nous montrons que l'utilisation d'un dictionnaire de taille importante, en plus de l'exactitude des analyses qu'il procure, permet une grande rapidité de traitement.

3.1. LE DICTIONNAIRE DES MOTS SIMPLES

Pour identifier les mots simples d'un texte, par exemple les mots *Pour*, *mots*, *identifier* de ce début de phrase, on se propose de les rechercher dans un dictionnaire. Un tel dictionnaire contient donc des mots tels que *grand*, *identifier*, *pour* que l'on trouve déjà comme entrée des dictionnaires éditoriaux classiques, mais également des mots tels que *mots* ou *mangions* qui ne se trouverons pas comme entrées explicites d'un dictionnaire classique. De tels mots sont retrouvés par les utilisateurs humains par référence à d'autres entrées, ici *manger* et *mot*. Pour relier de tels mots, inconnus, à des des entrées connues, l'utilisateur dispose parfois de tables qui permettent de vérifier que le verbe se conjugue bien comme il le pensait. Ce qui est possible à un utilisateur humain est difficile ou impossible à une machine si la description n'est pas complète. On veut donc que *mangions* soit déjà dans le dictionnaire et de plus, on veut pouvoir dire que c'est une forme conjuguée à la première personne de l'imparfait du verbe *manger*. Il faut donc que dans ce dictionnaire figure un article du type:

mangions,manger.I1p

où les symboles *I 1* et *p* signifient respectivement imparfait, première personne et pluriel. Ce dictionnaire s'appelle le *DELAF* (Dictionnaire Electronique du LADL pour les formes fléchies), et on se reportera à B. Courtois 1989 pour les détails de sa construction ainsi que de celle du *DELAS* (voir ci-dessous).
Pour le français, le *DELAF* contient environ 700.000 entrées et il est donc impossible de le décrire à la main. Pour obtenir le *DELAF* on a construit un dictionnaire de formes non fléchies (ou canoniques) qui ne contient par exemple qu'un seul représentant du verbe *manger* qui sera

la forme infinitive *manger* elle-même. De plus l'entrée de ce verbe contiendra un code, ici V5 qui se réfère à une table de conjugaison. Cette table décrit la manière exacte dont la forme canonique doit être modifiée pour obtenir chaque conjugaison, à chaque personne et à chaque temps-mode. Par exemple, la partie de la table V5 relative à l'imparfait de l'indicatif contient les informations suivantes:

1 : 1ais
2 : 1ais
3 : 1ait
4 : 2ions
5 : 2iez
6 : 1aient

1ais indique par exemple qu'en enlevant 1 lettre à la fin de la forme canonique *manger* (donc le 'r') et en y ajoutant *ais*, on obtient la forme conjuguée *mangeais* de la première personne du singulier. Ainsi, chaque nom, chaque adjectif et chaque verbe sera suivi d'un code de flexion qui permet d'obtenir ses formes fléchies. Ce dictionnaire appelé *DELAS* (pour Dictionnaire Electronique du LADL pour les formes Simples) contient par exemple les entrées suivantes:

manger,V5
étonner,V3
mot,N1
pour, Pré,N2[4]

Le *DELAS* (B. Courtois 1989) contient pour le français environ 80.000 entrées. Un programme automatique de flexion permet de fléchir chaque entrée et d'obtenir ainsi le *DELAF*.

Il faut remarquer que le *DELAF* ainsi constitué contient un grand nombre d'ambiguïtés, c'est à dire que plusieurs entrées peuvent avoir des clés d'accès identiques. Par exemple, la partie du *DELAS* qui contient le verbe et le nom suivant :

présider,V3
président,N32

se fléchit en un DELAF qui contient :

préside,présider.V3:P1s:P3s:S1s:S3s:Y2s
président,présider.V3:P3p
présidents,président.N32:mp
président,président.N32:ms

et qui contient donc deux entrées avec *président* pour clé d'accès.

Rechercher par programme un mot d'un texte dans le dictionnaire consiste alors à trouver toutes les entrées correspondant à ce mot dans la liste *DELAF* telle que nous venons de la présenter. Une autre solution consiste à factoriser les clés d'accès dans le *DELAF*, ainsi le même extrait devient-il :

---

[4] *pour* est en effet un nom dans *il y a du pour*.

préside,présider.V3:P1s:P3s:S1s:S3s:Y2s
président,présider.V3:P3p,président.N2:ms
présidents,président.N32:mp

Avec un tel dictionnaire nous disposons des données nécessaire à l'analyse morphologique que nous allons détailler maintenant.

3.2. ANALYSE MORPHOLOGIQUE PAR *DELAF*

Etant donné un texte, l'analyse morphologique consiste à associer à chaque mot simple, c'est-à-dire à chaque séquence de caractères encadrée de séparateurs (tels que le blanc, le tiret ou des signes de ponctuation), ses informations présentes dans le *DELAF*. Par exemple la phrase :

Il le passe à son ami

sera analysée en :

il,Pro
le,Pro,Dét
passe,passe;N1:ms,passe:N21:fs,passer.V3:P1s:P3S:S1s:S3s:Y2s
à,Prép
son Pro,son.N1:ms
ami,ami.N32:ms,ami.A32:ms

Cette analyse est donc obtenue par simple consultation du dictionnaire *DELAF* pour chacun des mots du texte. On se rappelera cependant que le *DELAF* contient environ 700.000 entrées et qu'il faut donc une procédure efficace pour chercher chaque mot dans cet ensemble. Nous ne rentrerons pas ici dans tous les détails de cette procédure et du codage du dictionnaire, on pourra se reporter à D. Revuz 1991 pour la description des algorithmes utilisés et à E. Roche 1992 pour des expériences diverses de représentation de dictionnaires par automates et transducteurs.

Mais l'idée simple consiste à représenter le *DELAF* comme un arbre sur les clés d'accès et à écrire à chaque feuille l'information correspondante. Soit le fragment de dictionnaire :

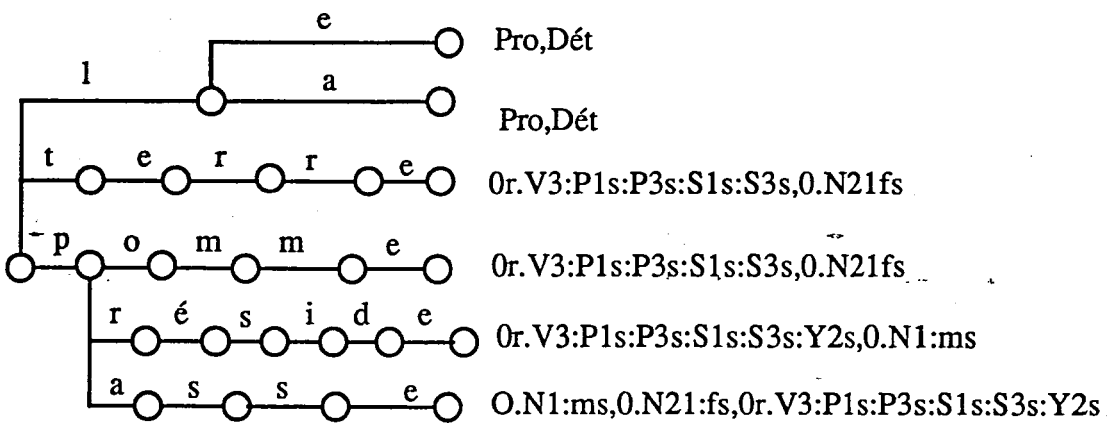
pomme, pommer.V3:P1s:P3s:S1s:S3s,pomme.N21fs
terre, terrer.V3:P1s:P3s:S1s:S3s,0.N21fs
préside,président.V3:P1s:P3s:S1s:S3s:Y2s,préside.N1:ms
le,Pro,Dét
la,Pro,Dét
passe,passe;N1:ms,passe:N21:fs,passer.V3:P1s:P3s:S1s:S3s:Y2s

Si on code les formes canoniques (par exemple *présider*) par rapport à la clé d'accès en nombre de lettres à retrancher et en lettres à ajouter (0 et 0r) ce même dictionnaire devient :

pomme, 0r.V3:P1s:P3s:S1s:S3s,0.N21fs
terre, 0r.V3:P1s:P3s:S1s:S3s,0.N21fs
préside,0r.V3:P1s:P3s:S1s:S3s:Y2s,0.N21:fs,0.N1:ms
le,Pro,Dét
la,Pro,Dét

passe,0;N1:ms,0:N21:fs,0r.V3:P1s:P3s:S1s:S3s:Y2s

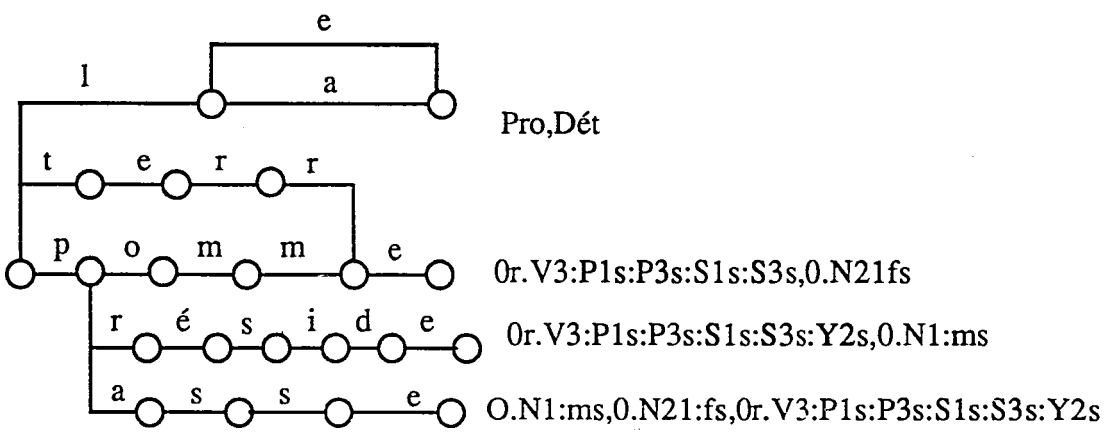
Ce dictionnaire peut alors se représenter par l'arbre :



Représentation d'un dictionnaire par arbre

Figure 3.2a

dans lequel on peut remarquer que plusieurs feuilles portent la même information. On peut alors fusionner ces feuilles et on obtient l'automate de la figure 3.2b :



Représentation d'un dictionnaire par automate

Figure 3.2b

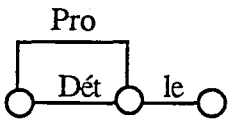
Formellement, l'arbre était déjà un automate acyclique et on peut minimiser le nombre d'états (ou de noeuds d'un tel automate (D. Revuz 1991). Ce qui rend possible une telle opération est que le nombre d'informations différentes qui peuvent apparaître sur les feuilles (par exemple *.Pro,.det* ou *0.N2:ms,0.N3:fs,0r.V3:P1s:P3S:S1s:S3s:Y2s*) est réduit par rapport aux nombres d'entrées. Ainsi pour les 700.000 entrées du français seulement 8.400 feuilles différentes sont à considérer.

Revenons à la phrase *il le passe à son ami*, si on se limite à la séquence de caractères *le passe*, l'analyse se conduit de la manière suivante :

On dispose donc de l'automate du *DELAF* figure 3.2b. On se place sur l'état numéro q_0 qui est l'état initial et on regarde la lettre *l* débutant le mot *le*. Une transition portant l'étiquette *l* existe qui part de l'état q_0. Cette transition conduit à l'état q_1. On regarde alors la lettre *e* qui mène de la même manière à l'état q_2 . Le caractère qui suit *e* est le blanc qui est un des séparateurs, on regarde alors si il y a une entrée sur l'état q_2. L'état q_2 portant l'information *.Pro,.Dét* on a l'analyse du premier mot: *le,.Pro,.Dét*. Pour le mot suivant, *passe*, on se repositionne sur l'état initial q_0 et l'examen successif des caractères mène à l'état q_3 qui porte l'information *0.N2:ms,0.N3:fs,0r.V3:P1s:P3S:S1s:S3s:Y2s*. On a donc fournie par des articles de dictionnaire l'analyse :

le,Pro,Dét
passe,0.N2:ms,0.N3:fs,0r.V3:P1s:P3S:S1s:S3s:Y2s

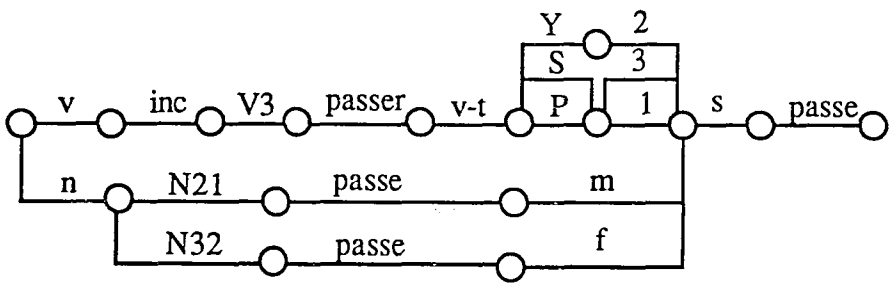
Cependant cette information n'est pas facilement exploitable. Une manière équivalente de représenter cet information est d'utiliser des automates acycliques. *le,.Pro,.Dét* sera représenté par l'automate de la figure 3.2c:



Représentation des informations morphologiques liées au mot *le*.

Figure 3.2c

et *passe,0.N2:ms,0.N3:fs,0r.V3:P1s:P3S:S1s:S3s:Y2s* par l'automate de la figure 3.2s :



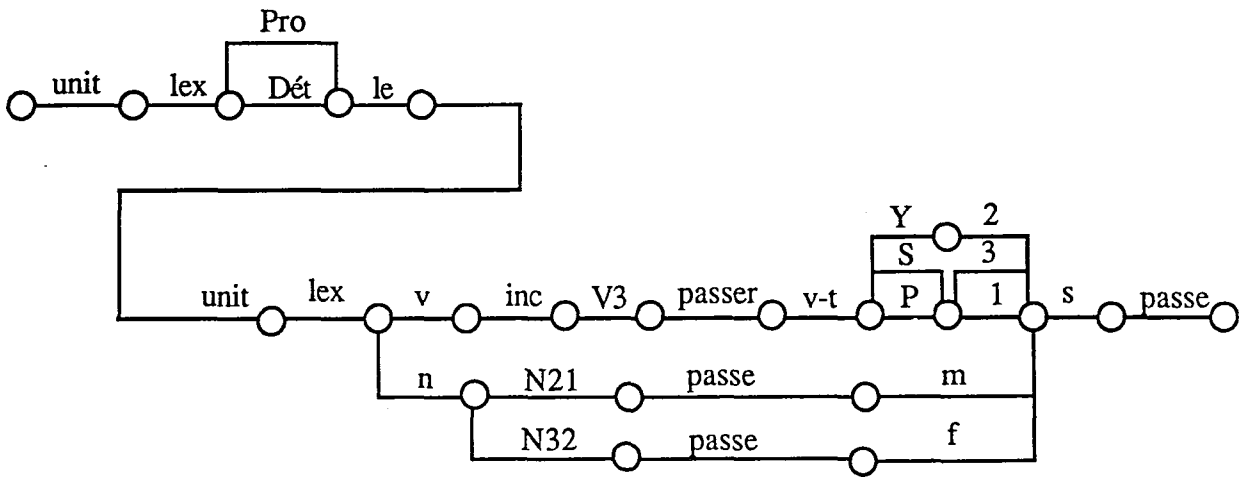
Représentation des informations morphologiques liées à la forme graphique *passe*

Figure 3.2d

Cet automate est équivalent aux articles de dictionnaires originaux dans la mesure où l'ensemble des chemins de l'automate c'est-à-dire l'ensemble des séquences acceptées :

v inc V3 passer v-t P 1 s passe
v inc V3 passer v-t P 3 s passe
v inc V3 passer v-t S 1 s passe
v inc V3 passer v-t S 3 s passe
v inc V3 passer v-t Y 2 s passe
n inc N21 passe m s passe
n inc N32 passe f s passe

représente bien l'ensemble des informations morphologiques disponible pour la forme graphique *passe*. Une manière de représenter les informations de la séquence *le passe* est alors de concaténer les deux automates (on inclus la séquence *unit lex* pour marquer le début de l'information relative à un mot) en un seul :



Représentation des informations morphologiques de la séquence *le passe*.

Figure 3.2e

L'ensemble des chemins de cet automate donne alors l'ensemble des informations morphologiques de la séquence :

unit lex pro le unit lex v inc V3 passer v-t P 1 s passe
unit lex pro le unit lex v inc V3 passer v-t P 3 s passe
unit lex pro le unit lex v inc V3 passer v-t S 1 s passe
unit lex pro le unit lex v inc V3 passer v-t S 3 s passe
unit lex pro le unit lex v inc V3 passer v-t Y 2 s passe
unit lex pro le unit lex n inc N21 passe m s passe
unit lex pro le unit lex n inc N32 passe f s passe
unit lex Dét le unit lex v inc V3 passer v-t P 1 s passe
unit lex Dét le unit lex v inc V3 passer v-t P 3 s passe
unit lex Dét le unit lex v inc V3 passer v-t S 1 s passe
unit lex Dét le unit lex v inc V3 passer v-t S 3 s passe
unit lex Dét le unit lex v inc V3 passer v-t Y 2 s passe
unit lex Dét le unit lex n inc N21 passe m s passe
unit lex Dét le unit lex n inc N32 passe f s passe

Cet automate est le résultat de l'analyse morphologique des mots simples.

3.3. DICTIONNAIRE DES MOTS COMPOSES

Nous nous sommes jusqu'à présent attachés à reconnaître des séquences de caractères ne contenant aucun séparateur. Il peut cependant être crucial de reconnaître des mots composés tels que *pomme de terre* ou *cousin germain. Pomme de terre* est en effet une unité lexicale minimale. Cette séquence devant être reconnue comme une unité, on doit reconnaître que la séquence *pommes de terre* représente la forme au pluriel de *pomme de terre* tandis que par exemple la séquence *pommes de terres* ne sera pas une unité lexicale (car elle n'est pas correctement orthographiée). Il faut donc disposer d'un dictionnaire qui contiennent par exemple les articles :

pomme de terre, pomme de terre.Nfs
pommes de terre, pomme de terre.Nfp
cousin germain, cousin germain.Nms
cousins germains,cousin germain.Nmp
cousine germaine,cousin germain.Nfs
cousines germaines, cousin germain.Nfp

Un tel dictionnaire est appelé *DELACF*, sa compilation est décrite intégralement dans M. Silberztein 1989, 1993 . Pour constituer un tel dictionnaire il a fallu d'abord écrire le dictionnaire des formes non fléchies, le *DELAC*, qui contient par exemple les données suivantes :

pomme de terre,NDN --+-
cousin germain, NA ++

où NDN et NA sont des classes morphosyntaxiques et les signes binaires "+" et "-" correspondent à des informations grammaticales sur la flexion.

Le *DELACF* contient 150.000 entrées ce qui pose le même problème de facilité d'accès que

pour les mots simples. Pour reconnaître les mots composés d'un texte il faut en effet rechercher toutes les séquences consécutives de deux, trois ou quatre mots simples. Il importe donc de consulter de manière très efficace ce dictionnaire. La stratégie utilisée diffère légèrement de celle employée pour la consultation du DELAF.

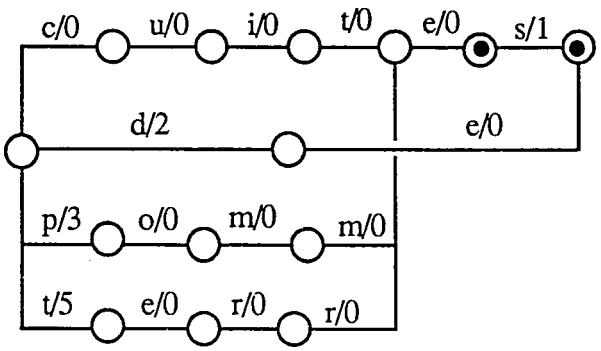
Prenons l'exemple de DELACF le plus (la liste des formes fléchies) soit :

pomme de terre
pommes de terre
terre cuite
terres cuites

Dans ce *DELACF* apparaissent les mots simples suivants :

- *cuite*
  cuites
  de
  pomme
  pommes
  terre
  terres

On peut appliquer à ce lexique la même technique qu'au *DELAF* (dictionnaire fléchi des mots simples), c'est-à-dire utiliser un automate pour représenter cet ensemble. La structure est cependant légèrement plus complexe, on utilise le transducteur de la figure 3.3a :



T_1 **Transducteur index du lexique précédent**

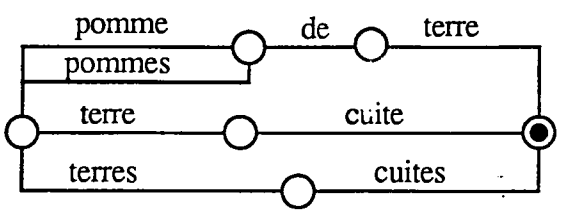
Figure 3.3a

Le but de ce transducteur est d'associer à chaque mot simple un numéro d'ordre unique. Le lexique était classé par ordre alphabétique et on veut que le transducteur associe 0 à *cuite*, 1 à *cuites*, 2 à *de*, 3 à *pomme*, 4 à *pommes* etc. Chaque état a ses transitions classées dans l'ordre croissant. Par exemple, pour l'état q_0, l'ordre des transition est le suivant: *c d p* et *t* et on les a de haut en bas. Si nous recherchons le mot *de* dans ce lexique, il nous faut prendre la transition *d*, une fois engagé dans ce chemin on sait que le mot recherché sera plus grand que tous les

mots qui commencent par *a,b* et *c*, donc ici le mot recherché sera plus grand que *cuite* et *cuites* il y a donc déjà deux mots qui précèdent *de* dans le lexique. On a donc indiqué sur la transition ce nombre de mots c'est à dire 2. La lettre suivante est donc les mots qui commencent par *da db dc dd* précèdent *de*, comme il n'y en pas dans le lexique on émet le symbole 0 au passage dans la transition. On sait alors que le mot fait bien parti de notre lexique et que le nombre de mots qui précèdent de est 2+0 = 2 donc *de* est le mots numéro 2 (c'est à dire le troisième). De même, pour *pommes*, le parcours du transducteur correspond à l'émission des symboles 3+0+0+0+0+1. Ce transducteur représente donc la fonction :

cuite	->	*0*
cuites	->	*1*
de -	>	*2*
pomme	->	*3*
pommes	->	*4*
terre	->	*5*
terres	->	*6*

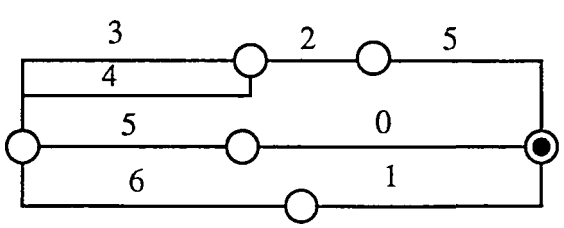
Remarquons maintenant que notre liste de mots composés se représente par l'automate suivant :



Lexique de mots composé

Figure 3.3b

D'autre part, comme on a un transducteur qui associe un numéro unique à chaque mot simple du lexique (et réciproquement), on voit que cet automate est isomorphe à l'automate suivant :



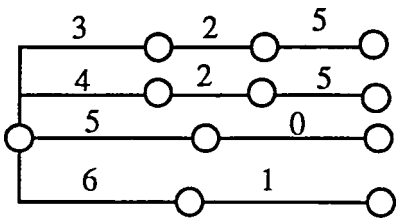
Lexique de mots composés

Figure 3.3c

Maintenant, pour vérifier que *terre cuite* appartient bien à notre liste de mots composés, on

utilise la procédure suivante. On prend d'abord *terre* et on consulte le transducteur des mots simples qui associe le symbole 5 à *terre*. On sait alors que l'on peut effectuer une transition à partir de l'état initial de l'automate, on parcours alors le transducteur avec *cuite*, ce qui donne 0 qui nous mène dans l'automate jusqu'à un état terminal. Le mot fait donc bien parti de notre liste.

Finalement, le *DELACF* sera représenté par le couple (T_1, A_2) où A_2 vaut :



DAG A_2 d'accès d'un dictionnaire de mots composés

Figure 3.3d

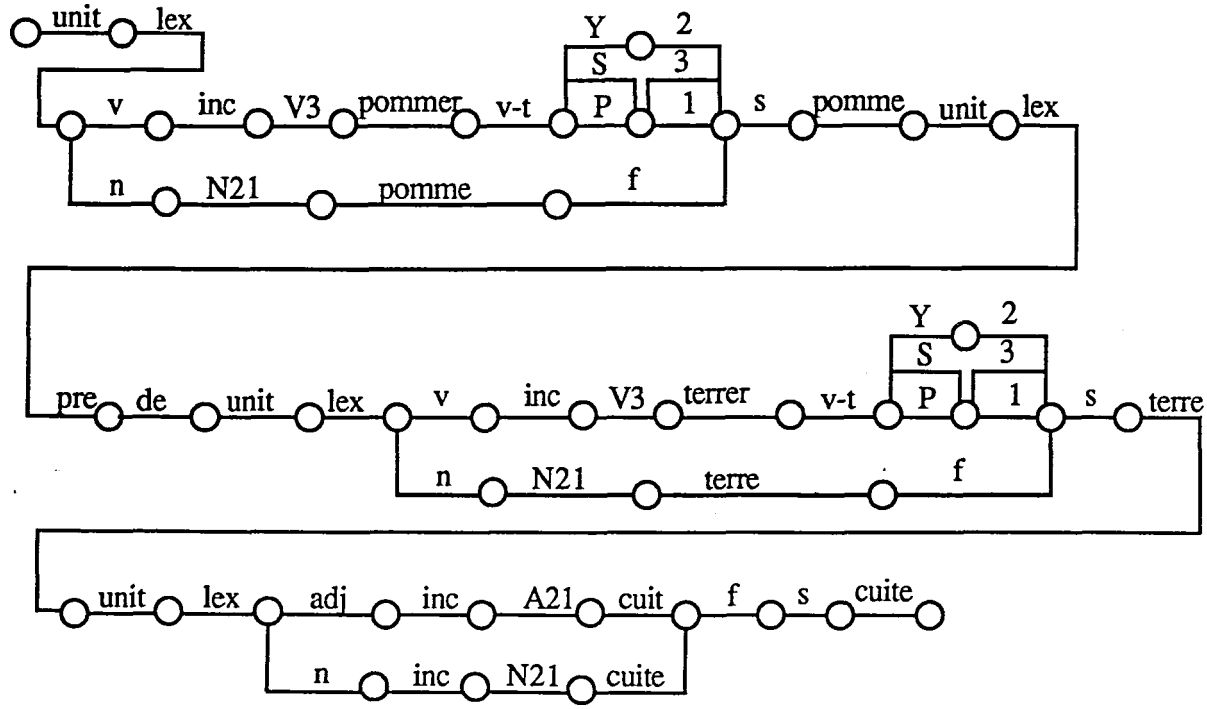
et l'analyse morphologique de la séquence *pomme de terre cuite* comporte les étapes suivantes :

A. L'analyse morphologique pour les mots simples donne l'automate de la figure 3.3e.

B. La consultation du DELACF pour les séquences *pomme de*, *pomme de terre*, *de terre* et *terre cuite* donne les information suivantes :

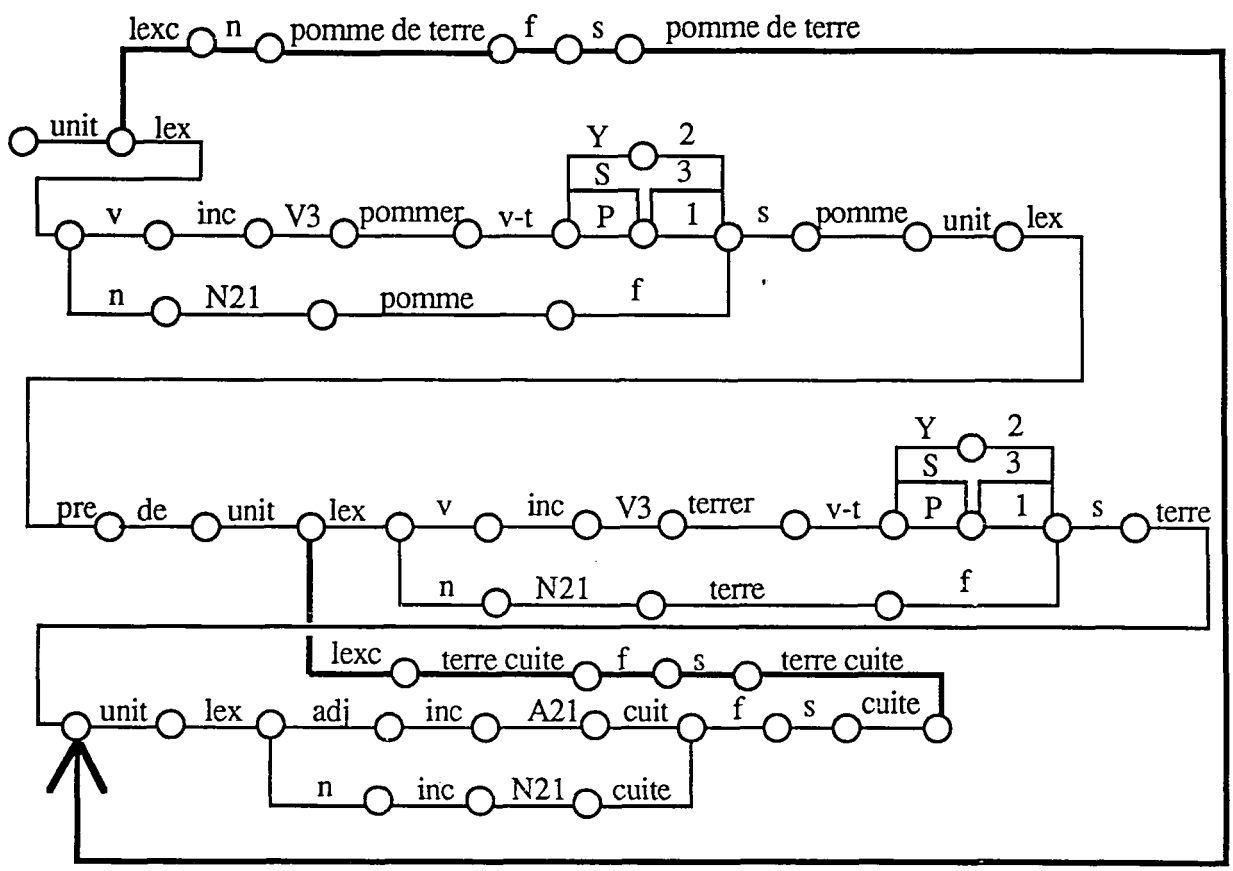
pomme de terre,pomme de terre, Nfs
terre cuite,terre cuite,Nfs

C. Les informations de (B) sont ajoutées à l'automate obtenu en A pour obtenir l'automate de la figure 3.3f.



Représentation de l'analyse morphologique en mots simples de *pomme de terre cuite*

Figure 3.3e



Représentation de l'analyse morphologique en mots simples et composés de

pomme de terre cuite

Figure 3.3f

3.4. MORPHOLOGIE DERIVATIONNELLE

Jusqu'à ce point nous ne sommes capables que d'analyser des mots explicitement décrit dans des dictionnaires. Une expérience récente (D. Clemenceau 1991) montre que l'emploi de tels outils permet déjà la reconnaissance de 98% des mots du texte. Trois grande catégories apparaissent parmi les formes non reconnues : les mots mal orthographiés, les noms propres et des mots tels que *coïnculpabilité* qui sont construits par ajouts de préfixes et de suffixes à des mots connus par ailleurs (ici *inculper* ou *inculpable*). Les noms propres posent des problèmes spécifiques que nous ne traiterons pas ici, disons simplement qu'ils justifient la constitution d'autres dictionnaires en général spécifiques à un domaine particulier. Un traitement adapté est également à employer pour les fautes d'orthographe, nous l'aborderons à la section 4.6.

3.4.1. Traitement morphologique de la morphologie dérivationnelle

Nous ne développerons pas non plus tous les problèmes liés à des mots tels que *coïnculpabilité*, les problèmes de la morphologie dérivationnelle (nous renvoyons à D. Clemenceau 1993 pour l'étude complète), cependant, nous appuyant sur un travail effectué avec D. Clemenceau (D. Clemenceau E. Roche 1992), nous donnons ici le moyen de reconnaître une grande partie de ces mots. De plus, dans un nombre appréciable de cas, les moyens employés pour la constitution du dictionnaire syntaxique *DELSYN* permettent naturellement de décrire les propriété syntaxique de ces mots ; nous verrons plus loin par quels moyens.

De plus le problème des mots manquant dans le dictionnaire illustre bien les possibilités offertes par le système de manipulation et de constructions d'automates et de transducteurs *PUPITRE*. Dans l'annexe 3 du manuel d'utilisation du système joint à cette étude on peut d'ailleurs trouver le script complet permettant la fabrication d'un transducteur unique d'analyse de ces mots à partir du dictionnaire *DELAF* et de règles dites Two-Level d'analyse morphologique.

Le programme d'analyse morphologique doit donc procéder aux analyses suivantes:

(1) *mangerons*

doit être analysé en :

(2) *manger,V1&F1p*[5]

comme le permettait déjà la consultation du *DELAF*,

[5]F signifie futur, 1 première personne et p pluriel.

(3) *chevaux*

doit être analysé en

(4) *cheval,N&mp*

et

(5) *débureautisations*

doit être analyser en

(6) *dé-*bureau,N*iser,V*ation,N*fp*

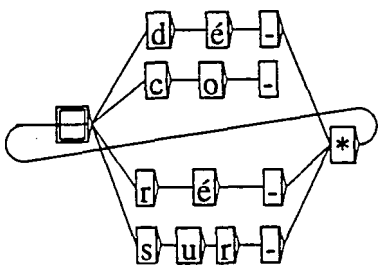
c'est à dire que le mot (5) est reconnu comme étant construit à partir d'un nom du *DELAS*, *bureau,N* sur lequel s'ajoute le suffixe *iser* qui transforme le nom en verbe, puis le suffixe *ation* qui retransforme le verbe en un nom sur lequel s'ajoute le préfixe *dé* et où le *s* final est reconnu comme étant une flexion de ce dernier nom.

L'approche a consisté à construire un transducteur unique qui effectue chacune des transformations (1) -> (2), (3) -> (4) et (5)->(6). Nous avons procédé comme suit :

- nous avons tout construit l'automate des décrivant les séquences :

(7) *(E+Pref*) mot (E+Suf*)*

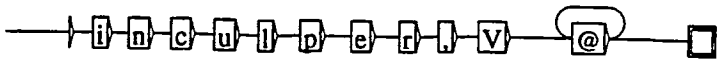
où *Pref* est l'ensemble des préfixes, *mot* une forme canonique du *DELAS* et *Suf* un ensemble de suffixes. Pour cela nous avons concaténé l'automate des préfixes *Préfixes(A)* dont nous donnons un extrait à la figure 3.4a :



Extrait de *Préfixes(A)*

Figure 3.4a

avec l'automate du *DELAS* dont nous donnons un extrait en 3.4b.

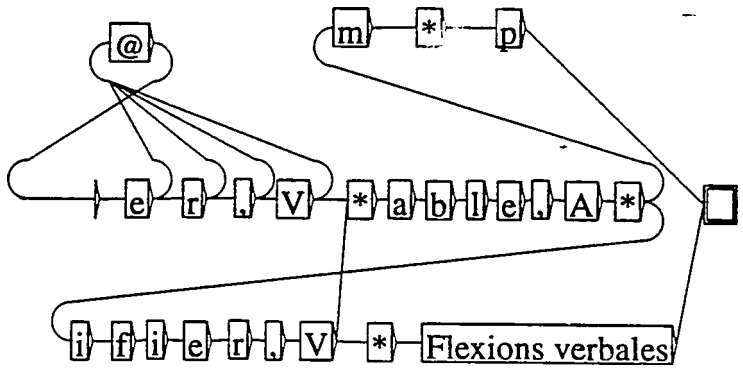


Extrait de *DELAS(A)*

Figure 3.4b

Ce dernier automate a une boucle *A* sur l'état terminal pour accepter tous les mots de type *motA** où *mot* est dans *DELAS*.

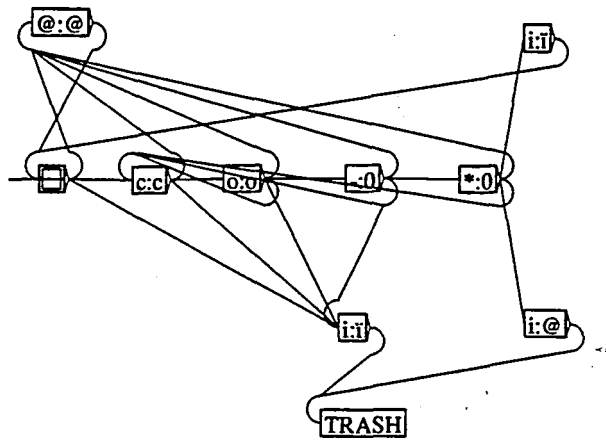
- nous avons alors pris l'intersection du résultat avec un automate de suffixes tel que celui de la figure 3.4c :



Extrait de Suffixes(A)

Figure 3.4c

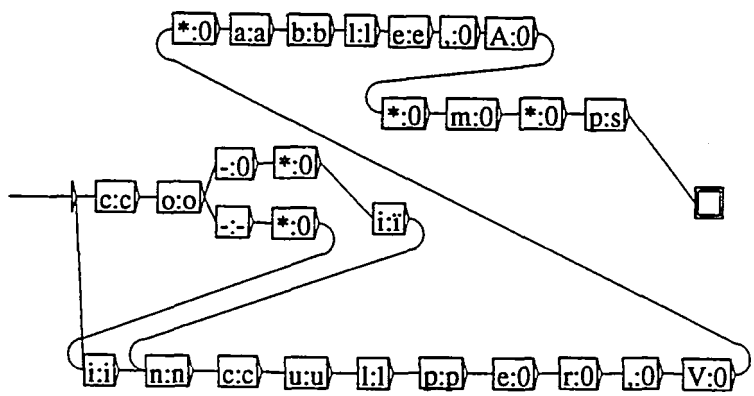
A ce point nous avons bien obtenu l'automate $A_1 = (Préfixes(A).DELAS(A)) \cap Suffixes(A)$ décrivant la structure (7). Se pose alors le problème de relier ces listes à des formes fléchies pouvant apparaître dans des textes. Le formalisme utilisé pour décrire ce type d'heuristiques est celui de la morphologie à deux niveaux (Two-Level Morphology) introduites par K. Koskenniemi 1983, 1984 pour analyser la morphologie du finlandais, formalisme repris par L. Kartunnen avec le formalisme des transducteurs. Ce formalisme consiste en des règles de réécriture équivalentes à un transducteur, nous donnons par exemple dans la figure 3.4d le transducteur qui convertit une séquence *co_*i* en *coï* ou *co-i* pour l'application d'un préfixe *co* à un mot tel que *inculper* (donc *coïnculper* ou *co-inculper*).



Extrait du transducteur *Règles* d'application des règles morphologiques

Figure 3.4d

Du transducteur de la figure 3.4d ou peut déduire le transducteur qui, à tout mot de A_1 associe sa forme graphique, il suffit de construire tout d'abord le transducteur *Transduc*(A_1) qui a tout mot de A_1 associe n'importe quelle séquence de caractère puis d'en faire l'intersection (intersection au sens des automates) avec le transducteur *Règles* de 4.4d. Cela donne alors le transducteur *Transduc*((*Préfixes*(*A*).*DELAS*(*A*)) ∩ *Suffixes*(*A*)) ∩ *Règles* dont nous donnons un extrait à la figure 3.4e :



Extrait de *Transduc*((*Préfixes*(*A*).*DELAS*(*A*)) ∩ *Suffixes*(*A*)) ∩ *Règles*

Figure 3.4e

On peut vérifier que ce dernier transducteur effectue l'analyse de (5) en (6).

Il nous faut maintenant construire le transducteur qui permet d'une part l'analyse de (1) en (2),

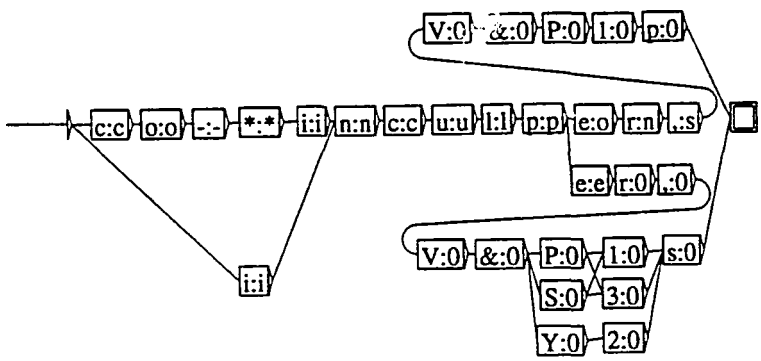
de (3) en (4) et d'autre part l'analyse de mots tels que *coïnculpions* dont la conjugaison est décrite dans le *DELAF* mais où il faut également reconnaître l'ajout du préfixe *co*. Pour cela nous reprenons le dictionnaire *DELAF* qui se présente de la manière suivante pour les deux entrées *inculpons* et *inculpe* :

inculpons,inculper,V&P1p
inculpe,inculper,V&P1s&P3s&S1s&S3s&Y2s

que nous pouvons représenter par une succession de lignes non ambiguës:

inculpons	<->	*inculper,V&P1p*
inculpe	<->	*inculper,V&P1s*
inculpe	<->	*inculper,V&P3s*
inculpe	<->	*inculper,V&S1s*
inculpe	<->	*inculper,V&S3s*
inculpe	<->	*inculper,V&Y2s*

Cette équivalence est strictement équivalente à un transducteur que nous pouvons concaténer avec le transducteur *Id(Préfixes(A))* qui à tout préfixe de *Préfixe(A)* associe lui-même. Un extrait du résultat de cette concaténation est donné à la figure 3.4f :



Extrait de $T_2 = Id(Préfixe).DELAF$

Figure 3.4f

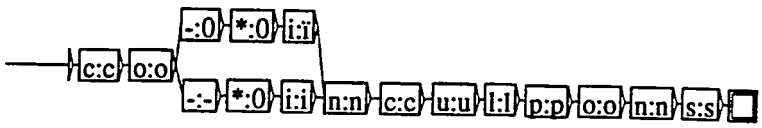
Ce dernier transducteur effectue par exemple l'association (8) suivante :

(8) *co_*inculpons* <-> *co_*inculper,V&P1p*

Par ailleurs, prenons l'automate $DELAF(A)$ qui représente la liste des mots fléchies et concaténons-le avec l'automate des préfixes $Préfixes(A)$ de la figure 3.4a en un automate $A_3 = Préfixes(A) . DELAF(A)$. Si nous prenons l'intersection du transducteur $Transduc(A_3)$ qui à tout mot de A_3 associe n'importe quelle séquence de A^* avec le transducteur des règles, nous obtenons le transducteur

$T_3 = Transduc(A_3) \cap Règles = Transduc(Préfixes(A).DELAF(A)) \cap Règles$

dont nous donnons un extrait à la figure 3.4g :



Extrait de T_3 = *Transduc(Préfixes(A).DELAF(A)) ∩ Règles*

Figure 3.4g

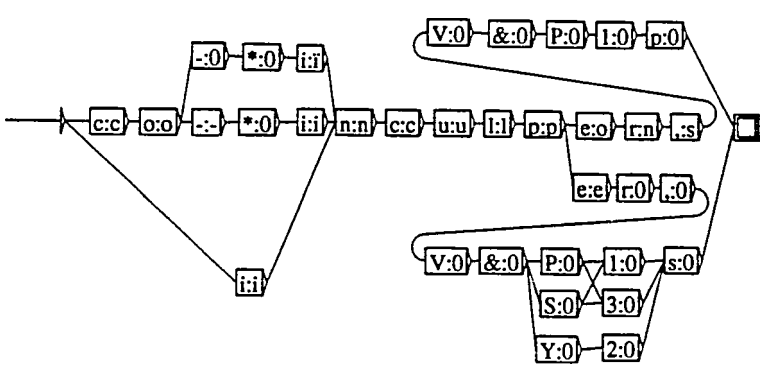
et qui réalise par exemple les associations (9) et (10) :

(9)	*co_*inculpons*	<->	*coïnculpons*
(10)	*co_*inculpons*	<->	*co-inculpons*

Si nous effectuons maintenant la composition de T_3 avec T_2 nous obtenons donc le transducteur $T_4 = T_3$ o T_2 qui compose (8) avec (9) et (10) pour aboutir aux deux associations suivantes :

(11)	*co_*inculper,V&P1p*	<->	*coïnculpons*
(12)	*co_*inculper,V&P1p*	<->	*co-inculpons*

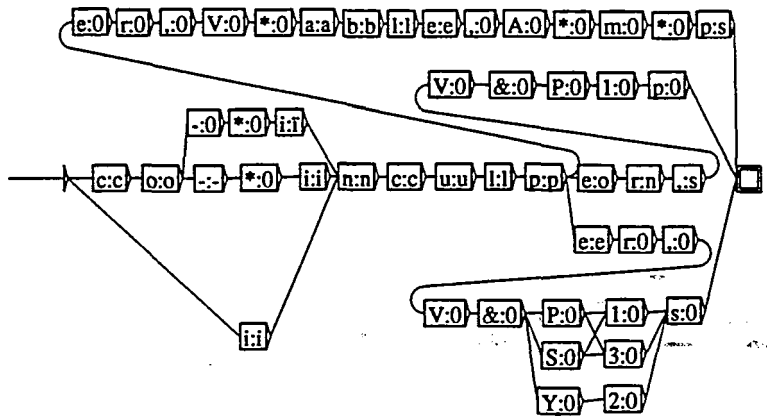
Nous donnons un extrait de ce transducteur à la figure 3.4h :



Extrait de T_4 = *Transduc(Préfixes(A).DELAF(A)) ∩ Règles* o *Id(Préfixe).DELAS*

Figure 3.4h

Finalement, si nous effectuons l'union des transducteurs T_4 avec celui de 3.4e, nous obtenons un transducteur unique qui fournit bien toutes les analyses que nous nous proposions de traiter. Un extrait de ce transducteur est donné à la figure 3.4j



Extrait du transducteur final d'analyse morphologique

Figure 3.4i

Ce transducteur comporte 123.000 états et 258.000 transitions, sa taille en mémoire est de 1,3MO et il permet une analyse de 1.100 mots par seconde.

3.4.2. Traitement syntaxique de la morphologie dérivationnelle

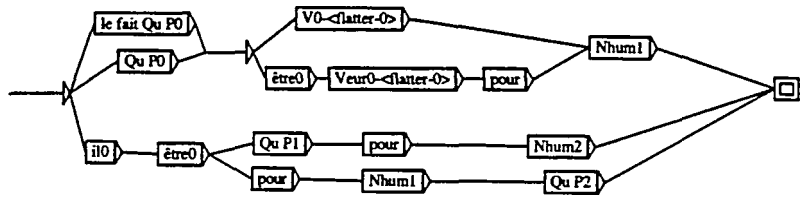
D. Clemenceau 1993 montre que le travail de description systématique des phrases simples du français doit et peut être repris systématiquement pour les mots dérivés par préfixation ou suffixation. Nous n'aborderons que très succinctement ce problème.

Traiter syntaxiquement un mot dérivé syntaxiquement signifie deux choses :

- donner certaines structures de phrases dans lequel il apparaît et,

- donner les transformations reliant ces phrases simples aux phrases simples d'un mot présent dans le dictionnaire.

Prenons le cas du mot *flatteur* présent dans le dictionnaire *DELAF* mais que le transducteur analyse aussi comme composé du verbe *flatter* et du préfixe *eur*. Le verbe *flatter* possède un emploi décrit dans la table 4 (M. Gross 1975), cette table donne par exemple les structures de la figure 3.4j pour ce verbe :



Extrait de l'automate des structures du verbe *flatter* de la table 4

Figure 3.4j

Cet automate montre en particulier qu'il existe une relation transformationnelle entre les deux phrases

(9) *Que Paul ait dit cela flatte Luc*
(10) *Que Paul ait dit cela est flatteur pour Luc*

de structures respectives :

(11) *Qu P_0 V_0-<flatte> $Nhum_1$*
(12) *Qu P_0 est $Veur_0$-<flatter-0> $Nhum_1$*

Cependant, pour que l'analyse de (10) fournisse (12) il faut préalablement établir que *flatteur* est une adjectivation de *flatter*. Or le transducteur donne comme analyse de *flatteur* :

(13) *flatter,V*eur,A*ms*

qui justement indique ce lien morphologique. Par conséquent, en utilisant parallèlement le transducteur d'analyse morphologique que nous venons de décrire et les données syntaxiques connues dans le lexique-grammaire. On peut faire l'analyse d'une phrase comme (10) et la relier à (9).

3.5. PRETRAITEMENT TYPOGRAPHIQUE DES TEXTES

Un programme d'analyse automatique est appelé à fonctionner sur des textes tels qu'ils sont disponibles. Le plupart du temps ces textes proviennent de systèmes de traitement de textes et il n'y a aucune raison pour que de tels sources soient déjà clairement découpées en phrases et que chaque mot apparaisse sous la forme exacte dans laquelle il est répertorié dans le dictionnaire. En particulier les mots peuvent être mal orthographiés ou en majuscules. Dans d'autres cas les textes disponibles sont issus de la reconnaissance optique de caractères (O.C.R) et certaines lettres peuvent ne pas avoir été reconnues. Dans tous ces cas nous allons montrer que le formalisme des automates et des transducteurs procurent deux avantages très important dans les traitements :

a. Un grand nombre des ambiguïtés que génèrent ces problèmes peuvent être levées rapidement au cours d'un prétraitement qui ne fait pas intervenir la syntaxe.

b. Même si l'ambiguïté persiste après ce prétraitement le processus d'analyse syntaxique reste exactement le même, c'est à dire que ni la grammaire ni le programme n'ont à être modifiés. Le résultat de l'analyse est alors correct si la donnée de départ avant prétraitement était syntaxiquement non ambiguë.

Nous allons maintenant illustrer ces deux points sur le problème du prédécoupage des textes en phrases et sur le problème de la reconnaissance optique de caractères.

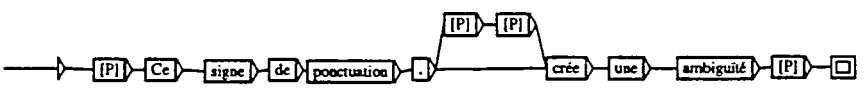
3.6. PREDECOUPAGE DES TEXTES EN PHRASES

L'analyseur automatique est appelé à traiter des textes qui n'ont pas été préparés pour lui, le premier problème consiste donc à segmenter le flot de mots en une succession de phrases. Ce découpage peut quelques fois être très compliqué, le fait de rechercher les points n'est évidement qu'une première approximation qui se révèle souvent fausse. Le point peut en effet apparaître également dans des abréviations: *I.B.M*, ou à la fin d'un mot tronqué. En fait la segmentation en phrase n'est validé qu'une fois l'analyse menée à son terme.

Prenons l'exemple de la phrase

(1) *Ce signe de ponctuation . crée une ambiguïté.*

dans laquelle le point crée une frontière de phrases possible. Comme le mot suivant n'est pas en majuscule le module de prétraitement renvoie le résultat que (1) correspond soit à une phrase, soit à deux phrases. Si le travail du module de prétraitement consiste à rajouter des marqueurs ([P]) de début et de fin de phrases, le résultat du prétraitement peut être l'automate suivant



Résultat du prédécoupage en phrases de (1)

Figure 3.6a

Ce résultat ne pose pas de problème particulier au module d'analyse syntaxique puisque ce dernier travail non pas sur des séquences de mots mais déjà sur des automates. L'analyse ultérieur du découpage en deux phrases va échouer, c'est à dire que le chemin correspondant a pour point fixe l'ensemble vide. Par conséquent, le résultat de l'analyse de 3.6a donne l'analyse du découpage en une phrase. La seule modification est une différence de taille dans les automates intermédiaires et donc une perte d'efficacité.

3.7. RECONNAISSANCE OPTIQUE DE TEXTES

Un très grand nombre de texte ne sont disponibles qu'imprimés sur papier. Même si le texte

existe sur support électronique il est encore difficile de se le procurer. De plus des incompatibilités de format rendent difficile la récupération de certains fichiers. Ce besoin a mené l'industrie vers la solution OCR (Optical Character Recognition) qui consiste à utiliser des scanners pour photographier et recomposer les documents en séquences de caractères. En raison du grand nombre de polices de caractères et de la mauvaise qualité de certain documents, ces procédés sont encore très faillibles.

Quand un programme d'OCR ne reconnaît pas un caractère, il le marque d'un signe particulier (*, par exemple) ou, faute plus grave, il pourra confondre un caractère avec un autre. Dans les deux cas, la donnée que l'analyseur syntaxique aura à traiter est ambiguë. En raison de la représentation par automates que nous avons adoptée pour représenter les phrases, il est possible au module d'analyse morphologique de traiter ces ambiguïtés, il en lève d'ailleurs un grand nombre, puis au module d'analyse syntaxique de poursuivre son analyse. La solution de la reconnaissance d'un caractère n'est alors quelque fois trouvée que lors de l'analyse syntaxique. Prenons l'exemple de la séquence :

(1) *il me croit*

où la lecture optique ne reconnaît pas le *e* de *me*. Dans ce cas le résultat de l'OCR sera écrit :

(2) *il m* croit*

où "*" représente a priori n'importe quel caractère. Cela veut dire que (2) est équivalent à l'automate suivant

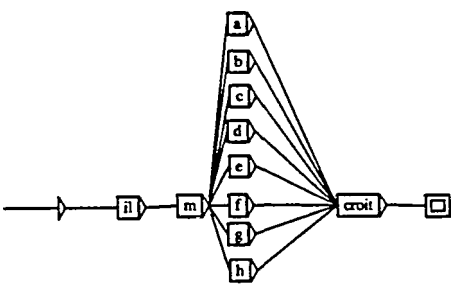


Figure 3.7a

où *m* peut être suivit de n'importe quelle lettre de l'alphabet (nous ne les avons pas toutes représentées).

L'application de l'analyse morphologique à 3.7a donne un second automate (figure 3.7b) dans lequel les ambiguïtés sont lexicales :

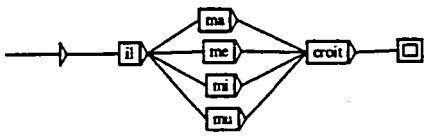


Figure 3.7b

Ce dernier automate est alors la donnée d'entrée de l'analyseur et toutes les interprétations sauf (1) seront rejetées.

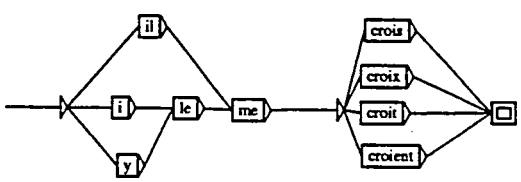
Le point important est que le module d'analyse morphologique reste inchangé, aucune modification n'est nécessaire pour traiter ce type de problème.

3.8. ANALYSE DE TRANSCRIPTIONS PHONETIQUES

De la même façon que dans les exemples précédents, un système de reconnaissance vocale proposera un ensemble de propositions lexicales qui risque d'être fortement ambigu. Reprenons l'exemple de la phrase :

(1) *Il me croit*

On peut supposer que le système de reconnaissance vocale propose un automate représentant les ambiguïtés lexicales sous la forme de l'automate de la figure 3.8a :



Sortie d'un système de reconnaissance vocale

Figure 3.8a

Le module d'analyse syntaxique s'applique de la même manière que dans les cas précédents et rejette naturellement les séquences syntaxiquement incorrectes pour ne proposer que l'analyse de (1). Dans cette optique, le module d'analyse syntaxique peut être un des composant d'un système de dictée automatique.

4. DELSYN

Pour décrire le traitement des différents problèmes syntaxiques et la mise en forme des données, nous décrivons l'analyse de phrases de complexité progressive, ce qui augmente la couverture des phrases analysées. Nous partons donc d'une situation très simplifiée pour nous rapprocher progressivement de situations plus réalistes. A la fin de chaque section nous donnerons un échantillon du dictionnaire syntaxique qui reflètera les structures abordées à ce stade.

4.1. ANALYSE DE *PIERRE AGACE JEAN.*

La première analyse que nous allons effectuer est celle de la phrase :

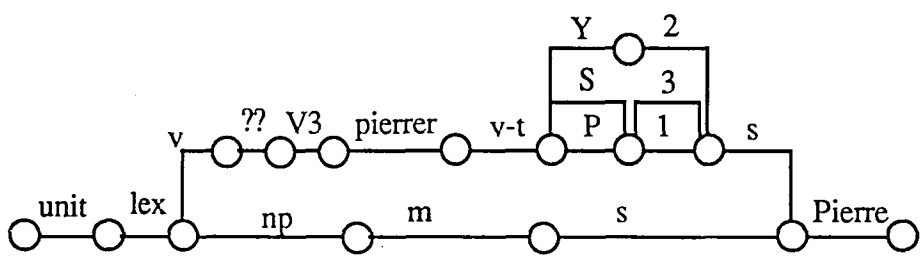
(1) *Pierre agace Jean.*

et nous allons la suivre dans tous ses détails, de l'analyse morphologique au résultat final, en passant par la constitution de la grammaire nécessaire à cette analyse.

Chacun des noms propres *Pierre* et *Jean* peut constituer un groupe nominal de type humain, nous n'approfondirons pas l'analyse de ces structures et nous nous concentrons sur la structure globale de la phrase. La connaissance syntaxique nécessaire à l'analyse de cette phrase tient alors dans la structure suivante :

(2) $Nhum_0\ V_0$-<*agacer-0*> $Nhum_1$

Nous détaillons l'analyse morphologique de ce premier exemple. Pour chaque mot la consultation du *DELAF* donne les trois automates des figures 4.1a, 4.1b et 4.1c:

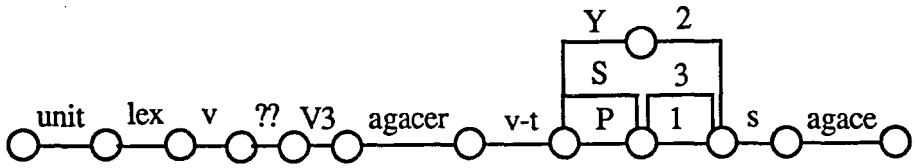


morpho(Pierre)

Figure 4.1a

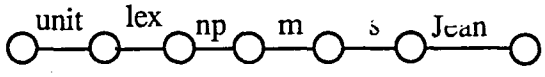
Pierre est le nom propre mais également une forme conjuguée du verbe *pierrer* (dans la phrase

Luc pierre le chemin).



morpho(agace)

Figure 4.1b



morpho(Jean)

Figure 4.1c

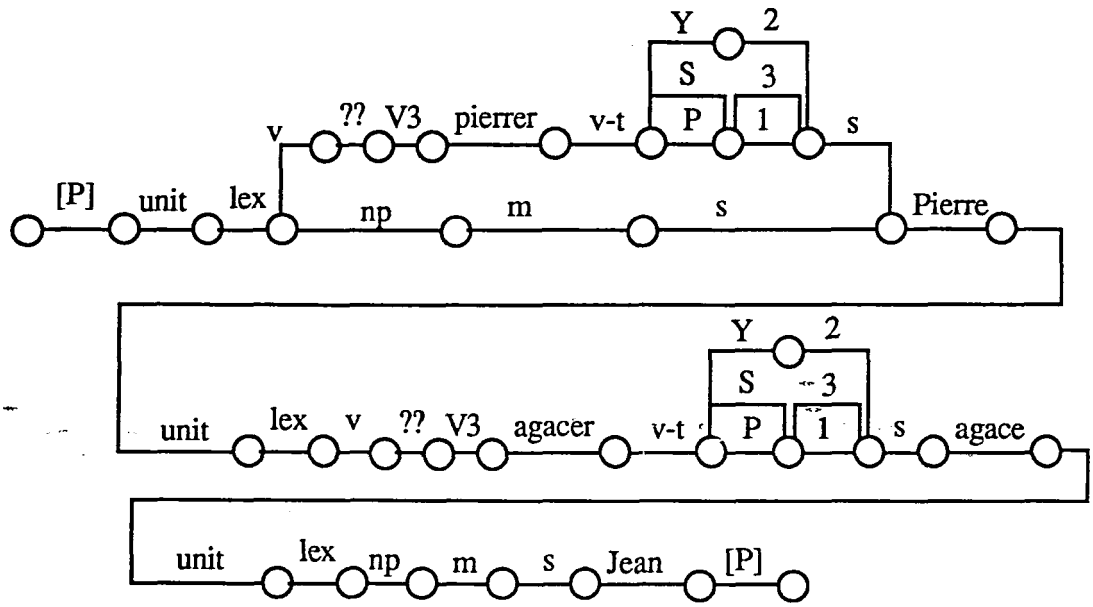
On ajoute le symbole [P] au début et à la fin des automates représentant l'analyse morphologique, cela donne l'automate

$$AUTPH = morpho((1)) = morpho(Pierre\ agace\ Jean)$$

qui est la concaténation :

[P].*morpho*(*Pierre*).*morpho*(*agace*).*morpho*(*Jean*).[P]

et que nous représentons sur la figure 4.1d :



AUTPH = *morpho*((1)) = *morpho(Pierre agace Jean)* Automate représentant l'analyse morphologique de la phrase (1)

Figure 4.1d

Par ailleurs, la structure S_1, qui est pour le moment le seul élément de notre dictionnaire syntaxique (donc de notre grammaire): $DELSYN = \{S_1\}$, soit :

S_1 : $Nhum_0\ V_0$-<*agacer-0*> $Nhum_1$

est explicitée en la suite de symboles S_2 suivante

S_2 : [P] $Nhum_0\ V_0$- *0 -- agacer* -$V_0\ Nhum_1$ [P]

On transforme ensuite S_2 en une transduction équivalente à *DELSYN* et que nous notons *f_DELSYN* et qui va pouvoir s'appliquer à l'automate de la phrase. Pour cela, on définit préalablement la transduction *ABREV* qui transforme la structure S_2 en cette transduction *f_DELSYN* : la transduction *ABREV* sera généralisée à tous les verbes et à toutes les structures, nous ne la détaillons que pour cet exemple.

ABREV sera la transduction qui prend une structure comme S_2 et transforme chaque symbole en une transduction. Ainsi le symbole $Nhum_0$ sera ainsi transformé en le transducteur de la figure 4.1f qui insère les symboles [N] et les parenthèses sur les séquences représentant l'analyse morphologique d'une phrase. *ABREV* est définie par la liste des associations suivantes:

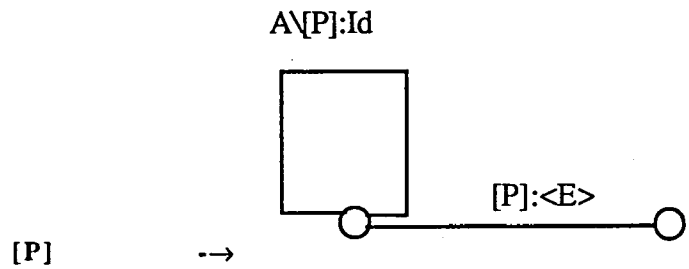


image de *[P]* par *ABREV*

Figure 4.1e

C'est à dire que le symbole [P] de la structure S_1 de *DELSYN* sera transformé chaque fois qu'il sera rencontré,en l'automate à deux états et deux transitions de la figure 4.1c.

Nhum0 →

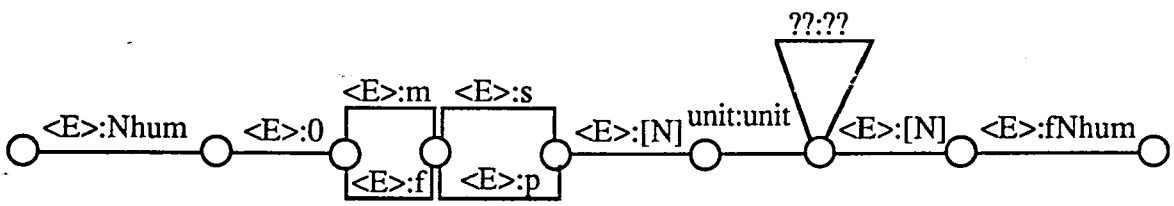


Image *ABREV(Nhum$_0$)* de *Nhum$_0$* par la transduction *ABREV*

Figure 4.1f

Nhum1 →

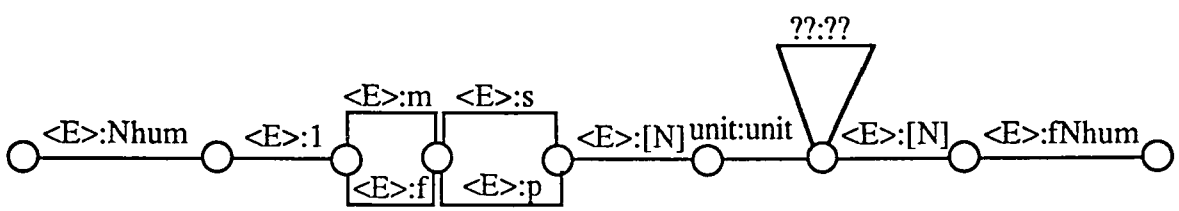


Image de *Nhum$_1$* par *ABREV*

Figure 4.1g

V0- →

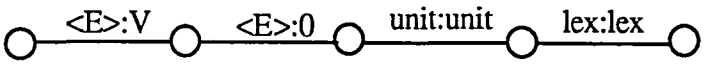


Image de V_0- par *ABREV*

Figure 4.1h

0 →

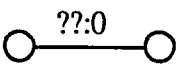


Image de *0* par *ABREV*

Figure 4.1i

-- →

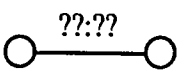


Image de -- par *ABREV*

Figure 4.1j

-V0 ->

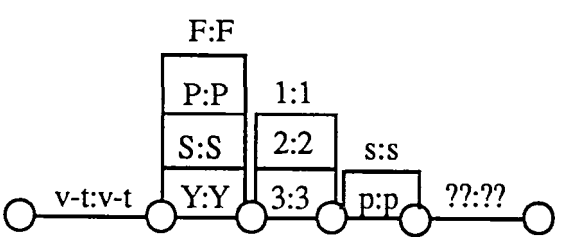


Image de -V0 par ABREV

Figure 4.1k

X ->

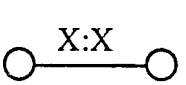
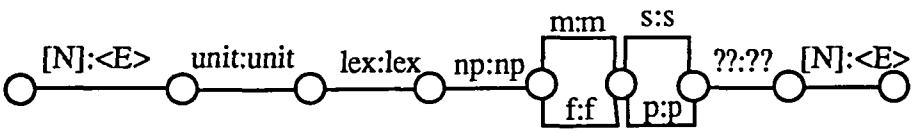


image des autres symboles par ABREV

Figure 4.1l



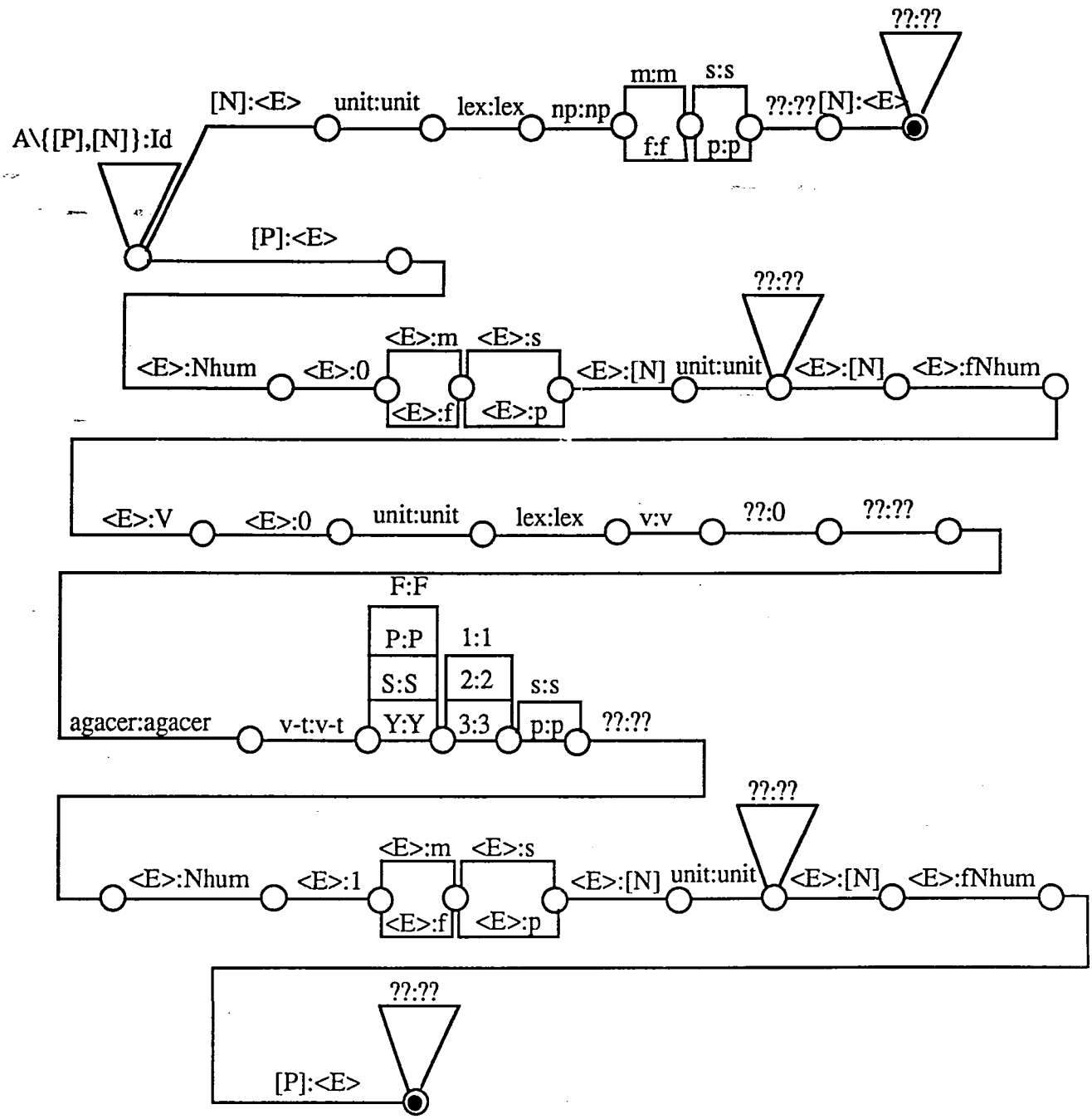
pour N

Transducteur représentant le fait qu'un nom propre peut être un groupe nominal

Figure 4.1m

Cette série de transformations définit la transduction *ABREV* qui appliqué à la structure S_2 donne le transducteur $T_1 = ABREV(S_2)$, c'est à dire aussi

$$f\_DELSYN = ABREV(DELSYN).$$



***$T_1 = ABREV(S_2) = f\_DELSYN = ABREV(DELSYN)$* Transducteur d'analyse.**

Figure 4.1n

Ce transducteur, *f_DELSYN*, représente donc notre grammaire.

Nous pouvons maintenant commencer l'analyse puisque nous disposons de l'analyse morphologique de la phrase morpho(Pierre agace Jean) et du transducteur représentant la grammaire f_DELSYN. La première étape de l'analyse consiste à appliquer la grammaire sur le texte, c'est à dire à calculer :

$$AUTPH_2 = T_1(AUTPH) = f\_DELSYN(morpho(Pierre\ agace\ Jean))$$

La seconde étape consiste à réappliquer la grammaire f_DELSYN sur le résultat de l'étape précédente pour obtenir :

$$f\_DELSYN(AUTPH_2) = AUTPH_3 = f\_DELSYN^2(AUTPH)$$

qui est le résultat de l'analyse puisque l'on a obtenu le point fixe :

$$f\_DELSYN(AUTPH_3) = AUTPH_3.$$

On rappelle qu'à ce stade la grammaire, c'est à dire *DELSYN*, est réduite à un élément:

Nhum:0 V0-<agacer-0> Nhum1

Premier extrait de DELSYN (1 élément)

Figure 4.1o

4.2. ANALYSE DES PHRASES SIMPLES CONSTRUITES AVEC AGACER

Le verbe *agacer* peut apparaître comme verbe principal dans des phrases d'une grande variété, comme le montre les exemples :

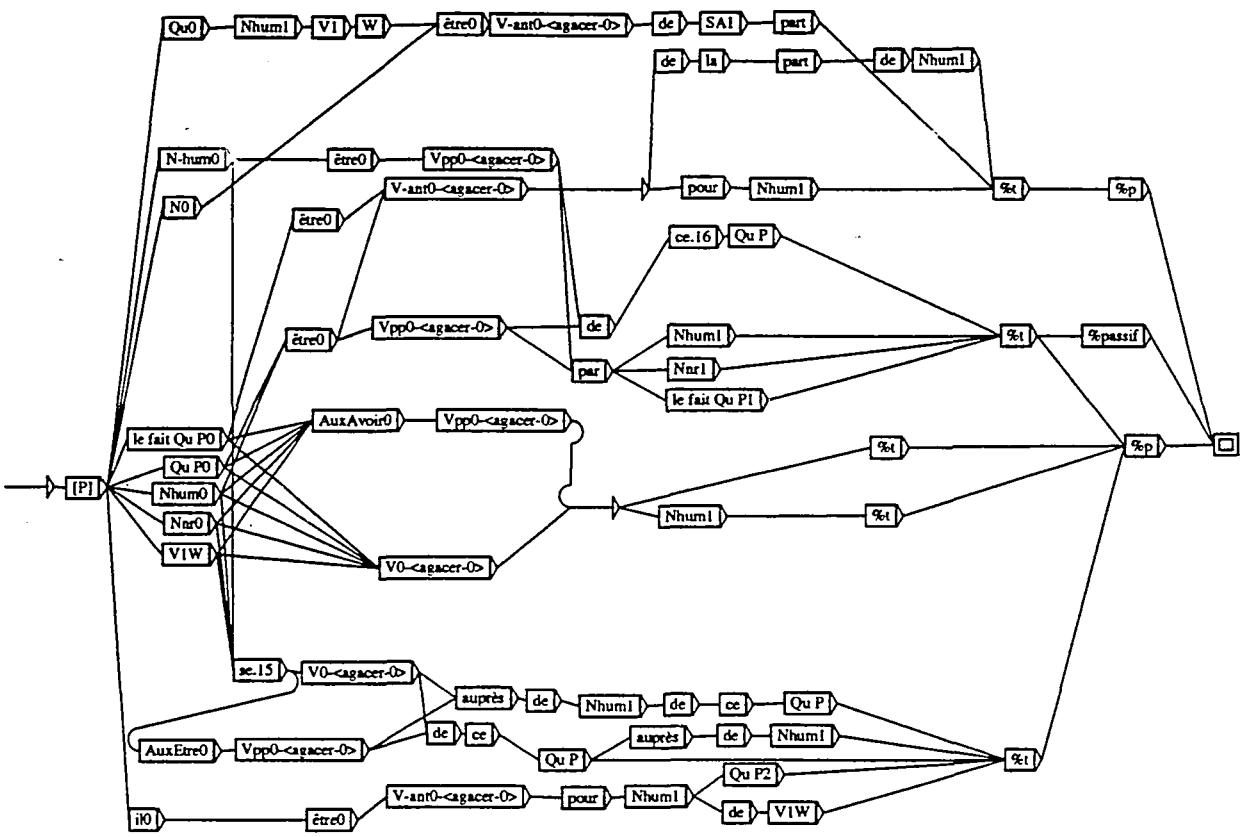
(1) *Pierre agace Luc*
(2) *Que Pierre fasse cela agace Luc*
(3) *Le fait que Pierre fasse cela agace Luc*
(4) *La nouvelle agace Luc*
(5) *Entendre ce brouhaha agace Luc*
(6) *Pierre agace*
(7) *Pierre s'agace de ce que Luc fasse cela*
(8) *Pierre s'agace auprès de Jean de ce que Luc fasse cela*
(9) *Luc est agacé de ce que Luc fasse cela*
(10) *Luc est agacé par Jean*

Ces phrases sont des illustrations de la liste structures suivantes (qui proviennent de la description de l'entrée *agacer* du lexique-grammaire (table 4 de Maurice Gross 1975).

(11) $Nhum_0\ V_0$-<*agacer-0*> $Nhum_1$
(12) $Qu\ P_0\ V_0$-<*agacer-0*> $Nhum_1$

(13) *Le fait Qu* P_0 V_0-<*agacer-0*> $Nhum_1$
(14) Nnr_0 V_0-<*agacer-0*> $Nhum_1$
(15) $V_1{}^0W$ V_0-<*agacer-0*> $Nhum_1$
(16) $Nhum_0$ V_0-<*agacer-0*>
(17) $Nhum_0$ *se* V_0-<*agacer-0*> *de ce Qu* P_1
(18) $Nhum_0$ *se* V_0-<*agacer-0*> *auprès de* $Nhum_1$ *de ce Qu* P_1
(19) $Nhum_0$ *être* Vpp_0-<*agacer-0*> *de ce Qu* P_1
(20) $Nhum_0$ *être* Vpp_0-<*agacer-0*> *par* $Nhum_1$

Analyser les phrases (1) à (10) revient à ajouter les structures (11) à (20) dans *DELSYN*. Cependant, cette liste de structures, au vu des propriétés syntaxiques notées dans lexique-grammaire est encore incomplète, la liste complète peut être donnée sous la forme de l'automate *SYN(agacer-0)*) de la figure 4.2a.



SYN(agacer-0)

Figure 4.2a

Pour obtenir notre grammaire, nous appliquons le transducteur *ABREV*, non plus à une seule structure comme nous l'avons fait à la section précédente, mais directement à tout l'automate du

verbe. Le transducteur *ABREV* appliqué à *SYN(agacer-0)* donne alors le transducteur *f_DELSYN* (que nous ne montrons plus car il ne tient plus sur une page). L'application de ce transducteur analyse alors les phrases (1) à (10) de manière identique à l'analyse détaillée au paragraphe précédent.

A ce point, *DELSYN* contient donc par exemple les éléments de la figure 4.2b

$Nhum_0\ V_0$-<*agacer-0*> $Nhum_1$
Qu $P_0\ V_0$-<*agacer-0*> $Nhum_1$
Le fait Qu $P_0\ V_0$-<*agacer-0*> $Nhum_1$
$Nnr_0\ V_0$-<*agacer-0*> $Nhum_1$
$V^1{}_0W\ V_0$-<*agacer-0*> $Nhum_1$
$Nhum_0\ V_0$-<*agacer-0*>
$Nhum_0$ *se* V_0-<*agacer-0*> *de ce Qu* P_1
$Nhum_0$ *se* V_0-<*agacer-0*> *auprès de* $Nhum_1$ *de ce Qu* P_1
$Nhum_0$ *être* Vpp_0-<*agacer-0*> *de ce Qu* P_1
$Nhum_0$ *être* Vpp_0-<*agacer-0*> *par* $Nhum_1$

Extrait de DELSYN (101 éléments)[6]

Figure 4.2b

4.3. ANALYSE DES MEMES PHRASES POUR 400 VERBES DE MEME TYPE

Nous n'avons traiter que le verbe *agacer*. A cet effet, nous avons construit un automate *SYN(<agace-0>)* qui représente la liste de ses structures de bases et un transducteur *ABREV* qui permet de transformer l'automate de ce verbe en un autre transducteur *f_DELSYN* qui simule la grammaire. Il nous a cependant fallu écrire à la main l'automate du verbe *agacer*. Comme on a pu le constater sur la figure le représentant, cet automate est assez complexe et en donner la représentation graphique peut prendre du temps. Il est donc difficilement envisageable de répéter cette opération sur chacun des 400 verbes de même type (décrits dans la table 4) et a fortiori sur chacun des 12.000 verbes distributionnels du français répertoriés dans le lexique-grammaire. Nous allons montrer ici qu'il est possible d'exploiter les tables syntaxiques pour générer de manière semi-automatique les automates de chacun des verbes. Nous montrerons ensuite qu'un tel ensemble d'automates peut conduire à une représentation compacte et unifiée de *DELSYN*.

Avant d'aller plus avant, nous allons décrire sommairement le lexique-grammaire des verbes tel que nous l'utilisons pour cet échantillon de verbes. Les 12.000 verbes simples du français sont divisés en deux sous-ensembles de taille équivalente, le premier est l'ensemble des verbes à construction complétive (dans lequel figure *agacer*), c'est à dire les verbes dont un des arguments (sujet ou complément) peut être une complétive (une *que*-phrase). Ce sont des séquences du type *que P* ou *que Psubj* où *P* est une phrase. Par exemple la phrase suivante

(1) *Jean dit à Pierre qu'il faut partir*

[6] A ce stade, le dictionnaire DELSYN est représenté par un automate de 52 états et de 72 transitions.

est de structure

(2) $Nhum_0\ V_0$-<*dire-0*> *à* $Nhum_1$ *Qu P*

où la complétive *que P* vaut *qu'il faut partir* .

Le second sous-ensemble est l'ensemble des verbes qui n'admettent pas de complétive comme argument.

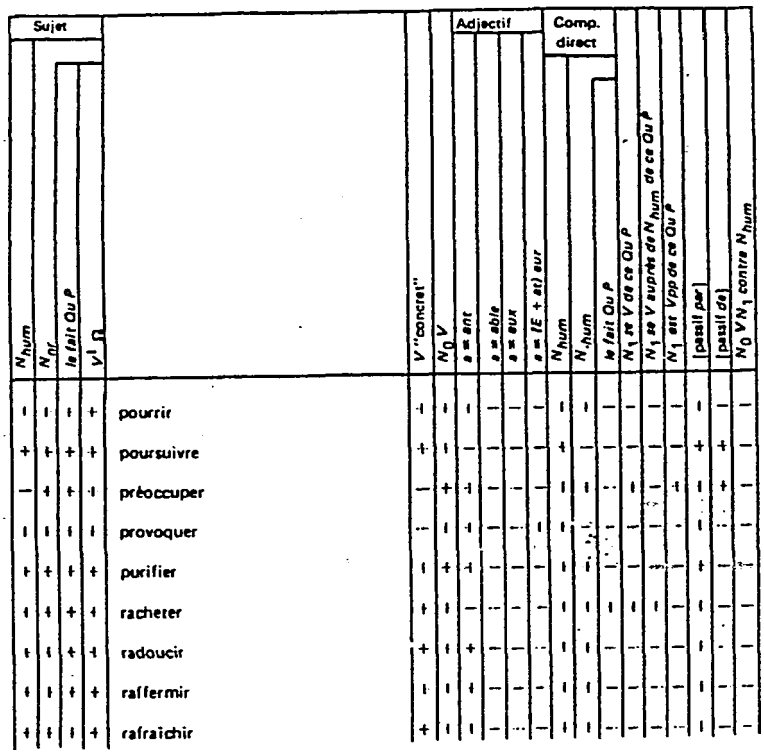
Dans chacun de ces sous-ensembles, les verbes sont regroupés par tables syntaxiques. En général, les verbes d'une même table ont en commun une propriété appelée propriété caractéristique de la table. Par exemple *agacer* figure dans la table *4* dont la propriété caractéristique est que les verbes de la table admettent la structure :

(3) *Qu* $P_0\ V_0\ Nhum_1$.

comme pour la phrase

(4) *Qu* $P_0\ V_0\ Nhum_1$ =:
Que Luc dise cela agace Jean

A l'intérieur de chaque table est établie la liste des structures de phrases dans lesquelles peuvent apparaître ces verbes, pour chaque verbe on établit alors exactement l'ensemble de ses structures possible. Concrètement chaque table est représentée par une matrice binaire dont nous donnons un exemple dans la figure suivante.

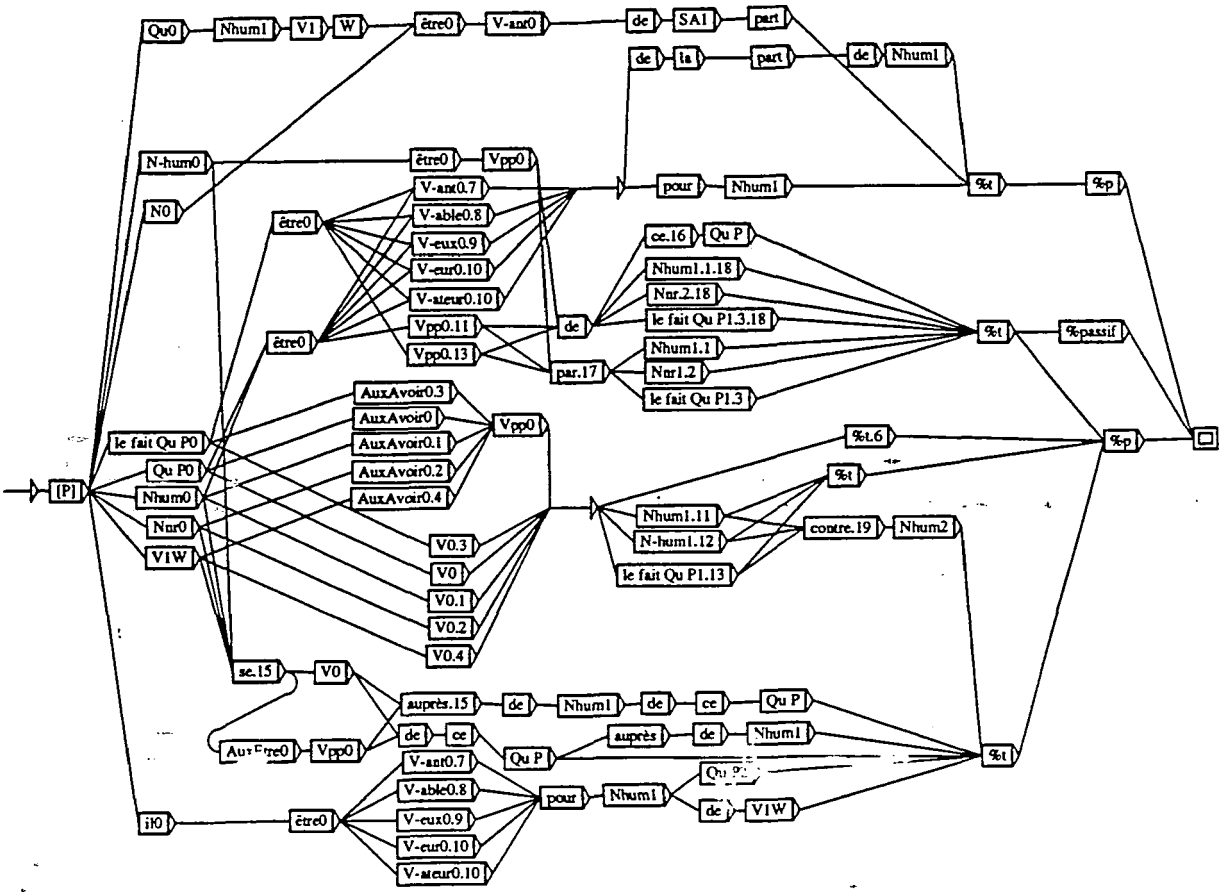


Echantillon de la table 4 (M. Gross 1975)

Figure 4.3a

Les lignes de cette matrice sont indicées par les verbes de cette table et les colonnes par les différentes structures possible. Si le verbe de ligne *i* peut apparaître dans une phrase dont la structure est celle la colonne *j* alors un signe + est noté à l'intersection (i,j). Un signe - est noté dans le cas contraire.

Nous allons décrire brièvement la manière de transformer ces données en un ensemble d'automates (un par verbe). La description complète de ce procédé se trouve dans E. Roche 1989. L'idée consiste à définir un AUTOMATE DE REFERENCE DE LA TABLE que nous noterons *Aref*. Cet automate est en fait l'automate d'un verbe de la table qui vérifierait chacune des propriétés, c'est à dire qui pourrait apparaître dans chacune des structures décrites dans la table. De plus pour chaque propriété, c'est à dire pour chaque colonne, nous donnons un ensemble de transition de cet automates qu'il faudra supprimer dans le cas où cette propriété n'est pas vérifiée. Examinons par exemple l'automate de référence *Aref*(4) de la table 4 qui est donné à la figure 4.3b.



***Aref*(4)** **Automate de référence de la table 4**

Figure 4.3b

Dans cet automate, certaines transitions, par exemple *V0.1*, sont étiquetées par un symbole suivi d'une série de nombres séparés par des points. Ces nombres réfèrent aux colonnes de la table. Pour obtenir l'automate du verbe *agacer*, il nous faut regarder la liste des propriétés marquées du signe -. Le fait que le complément puisse être un substantif non humain, c'est à dire la colonne numéro 12, est par exemple marqué négativement. Cela signifie qu'il faut détruire toutes les transitions marquées 12 dans l'automate de référence. En parcourant l'ensemble des propriétés marquées du signe - et en supprimant les transitions correspondantes, on obtient l'automate du verbe. Pour l'exemple du verbe *agacer*, après réduction et minimisation de l'automate on obtient *SYN*(<*agace*-0>) du paragraphe précédent.

Les programmes qui permettent d'effectuer cette opération font tous partis du système *PUPITRE*. Nous donnons dans l'annexe numéro 4 du manuel de l'utilisateur de *PUPITRE* le programme qui effectue cette opération.

Nous sommes maintenant au point où nous avons, pour chacun des 400 verbes de la table *4*, l'automate qui représente les structures. Nous ne pouvons pas utiliser directement une telle donnée. Mais comme nous avons un ensemble d'automates dans lesquels figurent explicitement chaque élément lexical, et en particulier chaque verbe, nous pouvons faire l'union de tous ces

automates. Nous rappelons ici que les automates représentent des ensemble de séquences de mots, donc qu'il est possible de définir une union ensembliste sur ces automates: l'automate A_3 union de deux automates A_1 et A_2 est l'automate qui vérifie $L_3 = L_1 \cup L_2$ où L_1, L_2 et L_3 sont respectivement les langages (c'est à dire les ensembles de séquences) reconnu par A_1, A_2 et A_3. L'algorithme qui permet de construire un tel automate est donné en annexe, avec les autres algorithmes utilisés.

Nous avons donc un automate unique qui représente l'ensemble des structures des 400 verbes de la table *4*. Cet automate définit également *DELSYN* au stade actuel de notre présentation. Ainsi, un échantillon de *DELSYN* est donné par la figure suivante :

Nhum$_0$ V_0-<*agacer-0*> *Nhum*$_1$
Qu$'P_0$ V_0-<*abasourdir-0*> *Nhum*$_1$
Le fait Qu P_0 V_0-<*abattre-0*> *Nhum*$_1$
Nnr$_0$ V_0-<*agacer-0*> *Nhum*$_1$
$V^1{}_0W$ V_0-<*afflige-0*> *Nhum*$_1$
Nhum$_0$ V_0-<*agacer-0*>
Nhum$_0$ *se* V_0-<*étonner-0*> *de ce Qu* P_1
Nhum$_0$ *se* V_0-<*apeurer-0*> *auprès de Nhum*$_1$ *de ce Qu* P_1
Nhum$_0$ *être* *Vpp*$_0$-<*attriste-0*> *de ce Qu* P_1
Nhum$_0$ *être* *Vpp*$_0$-<*attendri-0*> *par Nhum*$_1$

Extrait de *DELSYN* (39.872 éléments)[7]

Figure 4.3c

Comme dans les cas précédent, *ABREV(DELSYN)* donne le transducteur de la grammaire. Il faut noter que l'analyse de l'exemple 4.1 (1) reste absolument inchangé et que chaque verbe décrit dans ce début de dictionnaire donne des phrases qui peuvent être analysées de la même manière. Pour cette raison nous ne répétons pas l'exercice complet d'analyse effectué en 4.1 puisqu'il est inchangé.

Nous résumons dans le schéma de la figure 4.3d les étapes de la construction du dictionnaire syntaxique *DELSYN* dont un extrait figure en 4.3c et du transducteur équivalent *f_DELSYN*. Les données sont les tables d'un côté et les automates de références (il n'y en a pour l'instant qu'un) de l'autre. L'opération noté *CONSTRUCTION* est celle que nous venons de décrire et qui génère chacun des automates de verbe (*SYN*(*verbe*$_1$), *SYN*(*verbe*$_2$),..). L'union de ces automates donne l'automate *DELSYN* qui peut à son tour être transformé en *f_DELSYN* par l'application de la transduction *ABREV* qui explicite chacune des abréviations.

[7]Représenté par un automate de 135 états de 1603 transitions.

Schéma de construction de DELSYN et f_DELSYN pour les phrases simples d'une table

Figure 4.3d

4.4. ANALYSE DE PHRASES AVEC AUXILIAIRES ET TEMPS COMPOSES

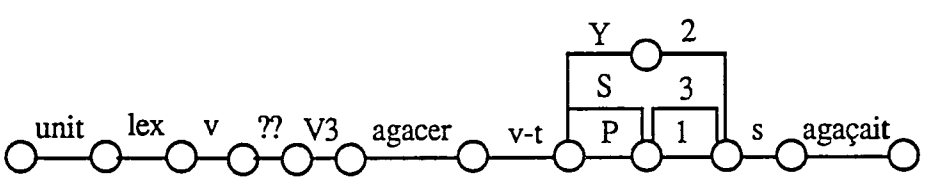
Au présent stade de l'analyse, DELSYN permet aussi bien l'analyse de :

(1) *Pierre agace Paul*

que celle de :

(2) *Pierre agaçait Paul.*

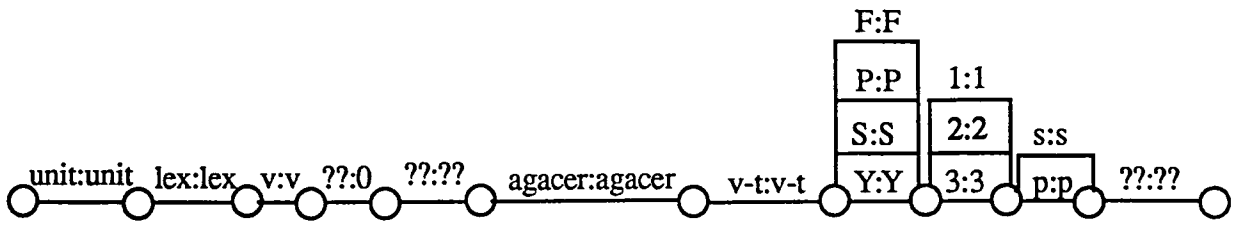
En effet, le module d'analyse morphologique, de *agaçait* fournit l'automate :



morpho(agaçait)

Figure 4.4a

dans lequel figure à la fois le mot et son analyse morphologique. D'un autre côté le transducteur *f_DELSYN* contient la séquence:



Transducteur $T_1 \subseteq f\_DELSYN$

Figure 4.4b

Dans cette séquence, qui vient de la grammaire, seule la forme canonique est donnée explicitement. L'application T_1 à *morpho(agaçait)* rétablit la forme-graphique *agaçait* (le ??:?? signifiant input peut être n'importe quelle lettre de l'alphabet et que l'output de la transition est alors égale à son input).

Cependant la même technique ne permet pas de reconnaître la même phrase conjugué à un temps composé comme :

(3) *Pierre a agacé Paul.*

Une analyse plus fine des temps du verbe (M. Gross 1968) montre qu'il faut considérer aussi des phrases comme :

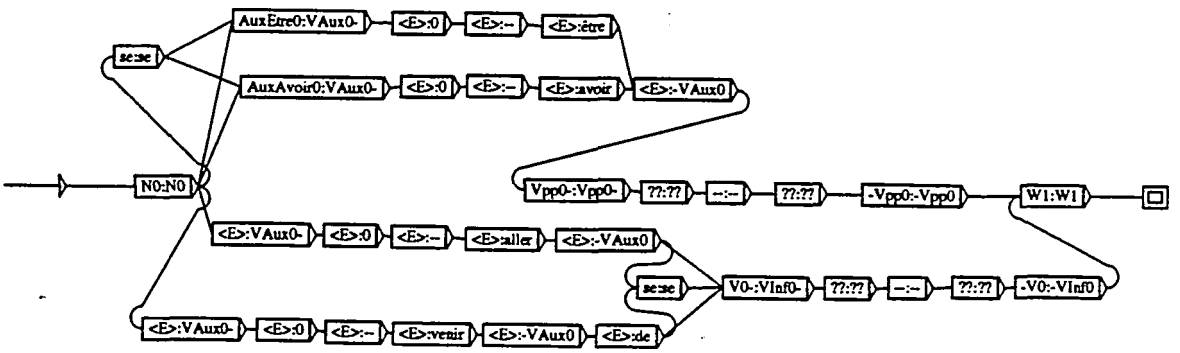
(4) *Pierre vient d'agacer Paul.*
(5) *Pierre va agacer Paul*

comme appartenant à la conjugaison de la phrase en *agacer*. La solution que nous avons adoptée consiste à ajouter à DELSYN des structures du type :

(6) $Nhum_0$ $AuxAvoir_0$ Vpp_0-<*agacer-0*> $Nhum_1$
(7) $Nhum_0$ $AuxVenir_0$ *de* $Vinf_0$-<*agacer-*$_0$> $Nhum_1$
(8) $Nhum_0$ $AuxAller_0$ $Vinf_0$-<*agacer-0*> $Nhum_1$

En fait, on veut reprend les structures déjà décrites et on leur adjoint ces structures à conjugaison composée. Une méthode générale consisterait à enrichir l'automate référence de la table (celui de la figure *4.3b* de la section précédente). Ainsi la liste des structures générées contiendrait les structures *(6)* à *(8)*. Deux inconvénients majeurs apparaissent alors. Le premier est d'ordre purement pratique, l'automate de référence est déjà très complexe, le graphique n'est pas facile à vérifier, des transitions peuvent manquer où être mal dirigées sans que cela soit très visible , le compliquer encore pose donc le problème de la correction de la description. Le deuxième inconvénient est plus linguistique, la conjugaison composée est un phénomène très général et il n'est pas naturel de la répéter de manière indépendante pour chaque classe de verbe.

Notre solution consiste à construire une fonction qui, à une structure à un temps simple, associe la même structure à un temps composé. Cette fonction est une transduction qui peut se représenter par le transducteur *CONJUG_TRANS* de la figure 4.4.c :



transducteur *CONJUG_TRANS*

Figure 4.4c

Dans ce transducteur N_0 et W_1 sont des abréviations qui doivent être transformées,

- pour N_0, en l'ensemble des types de sujet possibles (*Nhum*$_0$, *Qu* P_0, ..) et,

- pour W_1, en l'ensemble des séquences de compléments possibles (*Nhum*$_1$, *Nhum*$_1$ *contre Nhum*$_2$, ..).

Si on considère uniquement la séquence :

(9) *Nhum*$_0$*:Nhum*$_0$ *<E>:VAux*$_0$*- <E>:0* ..

cette séquence associe à la structure

(10) *Nhum*$_0$ *V*$_0$*- 0 - agacer -V*$_0$ *Nhum*$_1$

la structure :

(11) *Nhum*$_0$ *VAux*$_0$*- 0 -- aller -VAux*$_0$ *Vpp*$_0$*- 0 -- agacer -Vpp*$_0$ *Nhum*$_1$

qui correspond bien à :

(12) *Nhum1 aller Vpp0-<agacer-0> Nhum1*

Ainsi, l'application du transducteur *CONJUG_TRANS* à chaque structure lui associe la même structure conjuguée à chacun des temps composé. C'est à dire, pour (11) on obtient la liste des structures :

(13) $Nhum_0$ $VAux_0$- *0 -- avoir -*$VAux_0$ Vpp_0- *0 -- agacer -*Vpp_0 $Nhum_1$
(14) $Nhum_0$ $VAux_0$- *0 -- aller -*$VAux_0$ $Vinf_0$- *0 -- agacer -*$Vinf_0$ $Nhum_1$
(15) $Nhum_0$ $VAux_0$- *0 -- venir -*$VAux_0$ *de* $Vinf_0$- *0 -- agacer -*$Vinf_0$ $Nhum_1$

Si on applique ce transducteur à tout le dictionnaire *DELSYN,* on obtient l'ensemble de toutes les structures (c'est-à-dire pour l'instant les structures des verbes de la table 4) à tous les temps composés. En faisant l'union du résultat de cette transduction avec le dictionnaire original on obtient une nouvelle version du dictionnaire *DELSYN* dans laquelle figure toutes les structures de ces verbes aux temps simples et composés. Un nouvel extrait de *DELSYN* sera alors:

$Nhum_0$ V_0-<*agacer-0*> $Nhum_1$
$Nhum_0$ $VAux_0$-<*avoir*> Vpp_0-<*agacer-0*> $Nhum_1$
$Nhum_0$ $VAux_0$-<*aller*> $Vinf_0$-<*agacer-0*> $Nhum_1$
Qu P_0 V_0-<*abasourdir-0*> $Nhum_1$
Qu P_0 $VAux_0$-<*venir*> *de*$Vinf_0$-<*abasourdir-0*> $Nhum_1$
Le fait Qu P_0 V_0-<*abattre-0*> $Nhum_1$
Nnr_0 V_0-<*agacer-0*> $Nhum_1$
$V^1{}_0W$ V_0-<*afflige-0*> $Nhum_1$
$Nhum_0$ $VAux_0$-<*venir*> *de se* V_0-<*agacer-0*> *de ce Qu* P_1
$Nhum_0$ *se* $VAux_0$-<*être*> Vpp_0-<*agacer-0*> *de ce Qu* P_1

Extrait de *DELSYN* (403.225 éléments)[8]

Figure 4.4d

Comme dans le paragraphe précédent, nous pouvons résumer les étapes de la construction de *DELSYN* et *f_DELSYN* en un schéma qui s'enrichira au fur et à mesure que nous étendrons le domaine de la description. Ce schéma est donné à la figure 4.4e

[8] 450 états et 3462 transitions.

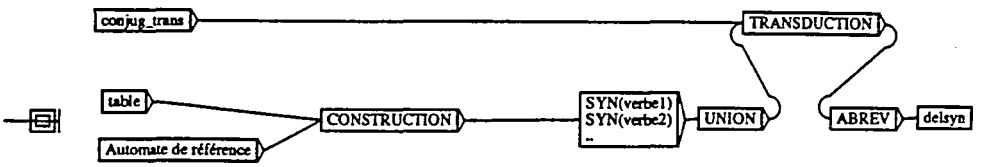


Schéma de construction de *DELSYN* et *f_DELSYN* à la fin de 4.4

Figure 4.4e

4.5. VERBES MODAUX

4.5.1. Présentation

Dans des phrases telles que

(1) *Jean n'arrête pas de parler*
(2) *Il continue de pleuvoir*

les verbes ne *arrêter Nég* et *continuer de* se comporte de manière proche de *venir de* et *aller* que nous avons décrits dans les conjugaisons composées. C'est à dire qu'ils s'appliquent sur des phrases bien formées comme

(3) *Jean parle*
(4) *Il pleut*

Ces verbes sont décrits dans la table *1* de M.Gross 1975. Les verbes de cette table ont ainsi pour point commun d'apparaître dans des phrases de structures

(5) $N_0\ V_0\ W$

où W est une liste (éventuellement vide) de compléments pour donner des phrases de structures

(6) $N_0\ U\ prep\ V_1{}^0\ W$

Ces verbes ont cependant des comportements sensiblement différents. En effet, un verbe tel que *continuer à* peut apparaître dans n'importe quelle phrase de structure (3) alors que *s'empresser de* de la phrase

(7) *Luc s'empresse de partir*

ne s'insèrera que dans le cas où N_0 est un substantif humain. De même certains verbes admettent une pronominalisation de l'infinitive *V0W* en une des particule préverbale *y*; *en* ou *le* comme dans

(8) *Pierre s'efforce de continuer*
(9) *Pierre s'y efforce*
(10) *Pierre ose partir*
(11) *Pierre l'ose*

L'ensemble de ces propriétés est décrit dans la table 1.

Nous allons voir, dans les deux paragraphes 4.5.2 et 4.5.3 qui suivent, qu'il existe deux manières d'analyser de tels verbes. La première consiste à traiter cette table comme la table *4* à la section 4.3, c'est à dire qu'elle consiste à écrire un automate de référence pour la table puis à en déduire un automate pour chaque verbe, cela donne alors une liste de structures du type

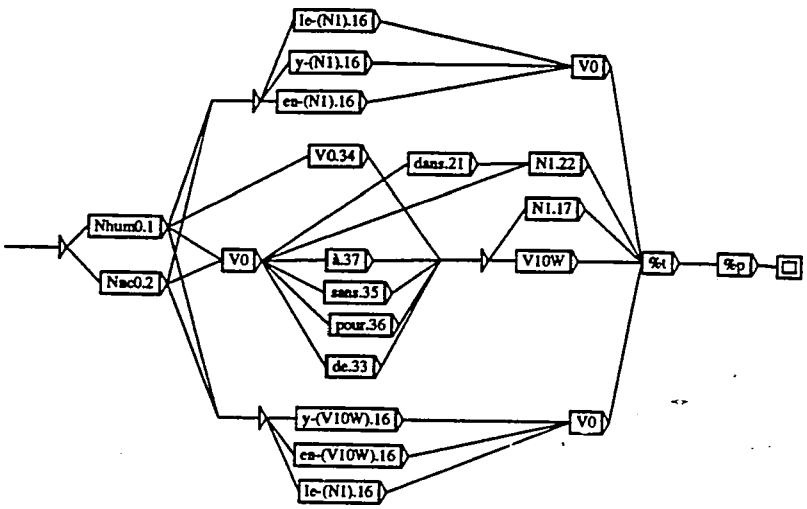
(12) N_0 *U-<continuer-0> à* V_1^0 *W*

Dans l'analyse de texte il faut alors rétablir le lien entre N_0 et le verbe V_1^0. [9]La deuxième méthode consiste à traiter chaque verbe de cette table comme aller et venir de au paragraphe précédent. Cette deuxième méthode, que nous avons finalement retenue, soulève certains problèmes que nous discutons plus bas.

4.5.2. Traitement de la table 1 par automate de référence

La table 1 décrit pour chaque verbe l'ensemble des structures dans lesquelles il apparaît. Ces structure, comme pour la table 4, peuvent être réunies dans un automate unique tel que celui de la figure 4.5a. L'automate de cette figure est ici légèrement simplifié, on ne tient par exemple pas compte des auxiliaires et des formes particulière que peuvent prendre les verbes (le *se* intrinsèque comme dans *s'empresser de* où une forme négative comme dans *ne finir Nég de*).

[9] dans V10 le 1 signifie qu'il s'agit d'un autre verbe que le verbe V0 principal et le 0 signifie que le sujet de ce verbe est le sujet N0 du verbe principal.



Automate de référence de la table 1

Figure 4.5a

Cet automate décrit bien entendu la structure caractéristique (7), mais aussi les pronominalisations possibles et des structures anotées telles que :

(13) $N_0\ V_0\ Prép\ N_1$
(14) $N_0\ V_0\ N_1$

qui rendent compte de phrases telles que :

(15) *Pierre commence la partie*
(16) *Pierre achève la partie*

Ces phrases sont plus complexes à analyser, on aimerait probablement relier (13) à des formes comme :

(17) *Pierre commence à jouer (?) la partie*

Mais nous ne donnons ici que les formes de surface.

La liste des structures de ces verbes une fois obtenue, il reste à voir comment une phrase telle que (7) peut être analysée. La comparer à

(18) N_0 *se* V_0-*<empresser> de* $V_1{}^0$-*inf W*

n'est pas suffisant, il faut également pouvoir comparer

(19) $N_0\ V_0$-*<partir>*

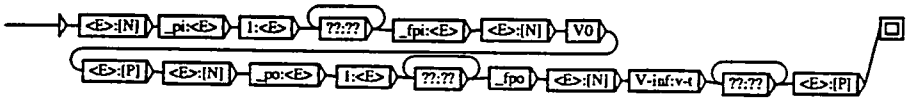
et la séquence obtenue par suppression de *s'empresser* de dans (7), c'est-à-dire :

(20) *Pierre partir*

Dans ce but, la première passe de l'analyse, c'est à dire la première application de la transduction *f_DELSYN* (équivalente à *DELSYN*) associera (21) à (7) :

(21) ([N] *Pierre* [N])$_{N0}$ (*s'empresser de*)$_{V0}$ ([P] (#[N] *Pierre* [N] #) *partir* [P])$_{V1}{}^{0}{}_{W}$

La phrase est alors parenthésée et le sujet *Pierre* est recopié. La structure (19) peut alors être comparée à *(#[N] Pierre [N] #) partir* lors de la deuxième application de *f_DELSYN*. Ceci se formalise par l'équivalence entre (15) et le transducteur de la figure 4.5b :



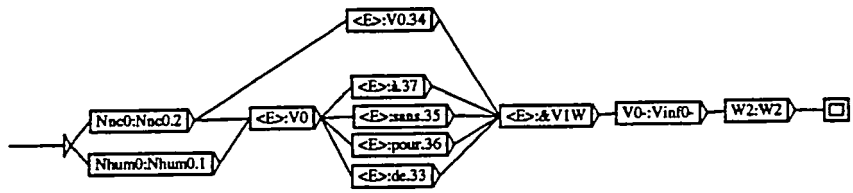
Formalisation de la recopie de N_0

Figure 4.5b

Tous ce qui apparaît entre *_pi 1* et *_fpi* sera recopié entre *_po 1* et *_fpo*.

4.5.3. Représentation des verbes de la table 1 comme des auxiliaires

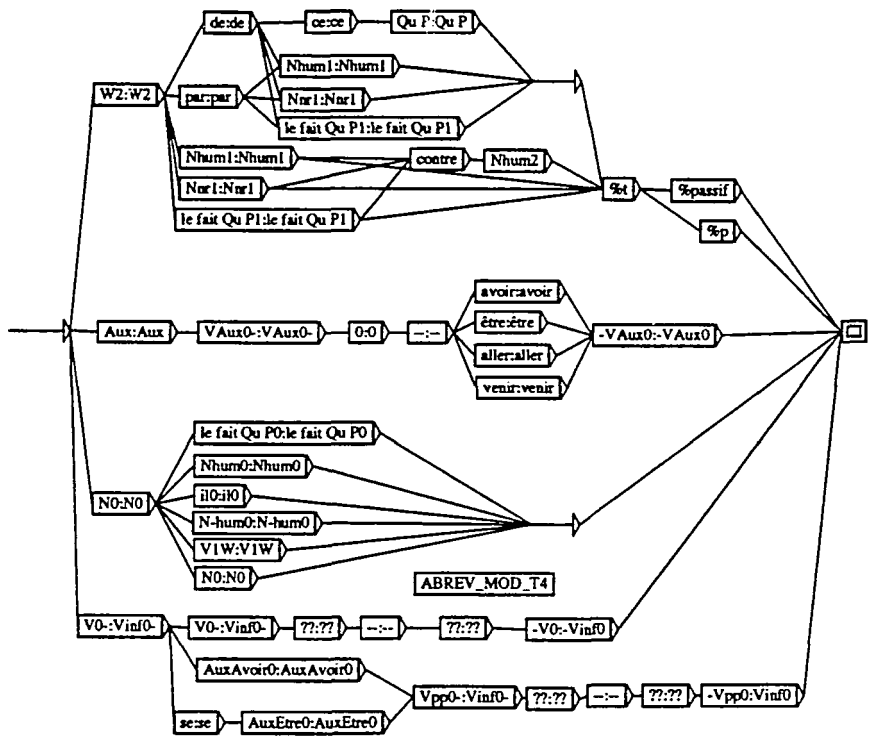
La seconde méthode consiste à écrire pour chaque verbe de cette table un transducteur comme celui de 4.4 pour les auxiliaires. Pour obtenir ces transducteurs on construit un transducteur de référence pour la table, dans lequel, comme avec les automates de références de 4.3, on indique les transitions à supprimer si la propriété est marquée moins. Ainsi, le transducteur (simplifié) est donné à la figure 4.5c.



Transducteur de référence simplifié de la table 1

Figure 4.5c

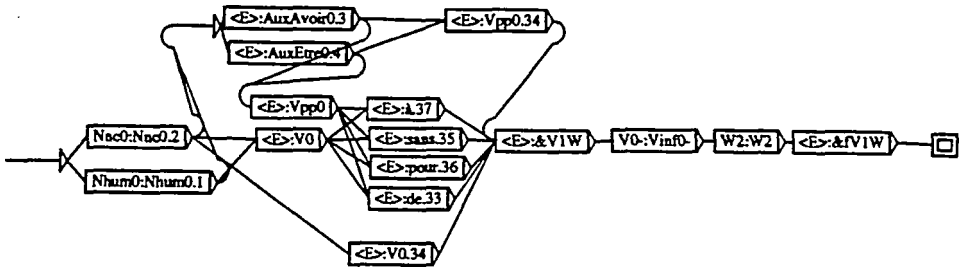
Le verbe *s'empresser de* est par exemple employé avec la préposition *de*, il est marqué - pour la colonne 37 (rajoutée dans un prétraitement) qui indique *à* comme préposition. La transition *à.37* est donc supprimée du transducteur. Les étiquettes de ce transducteur sont en général des abréviations qu'il faut rendre explicite. W_2:W_2 indique par exemple l'ensemble des complément possibles, pour la table 4 cet ensemble est donné par le transducteur *abrev1_modaux* de la figure 4.5d.



abrev1_modaux

Figure 4.5d

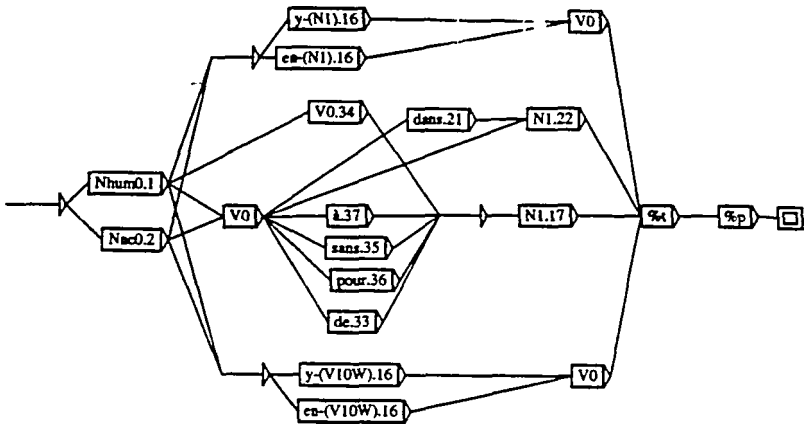
Il faut préciser le transducteur 4.5c pour prendre en compte les conjugaisons complexes du verbe modal, cela donne donc finalement le transducteur de la figure 4.5e :



Transducteur de référence de la table 1

Figure 4.5e

Cette approche ne permet cependant plus de reconnaître les structures (11) et (12), il faut donc garder un automate qui décrira ces structures à l'exclusion des structures de 4.5d. L'automate de référence qui permet cette reconnaissance est donné à la figure 4.5f.



Automate de référence de la table 1 pour les structures autres que N0 V0 Prep V10W

Figure 4.5f

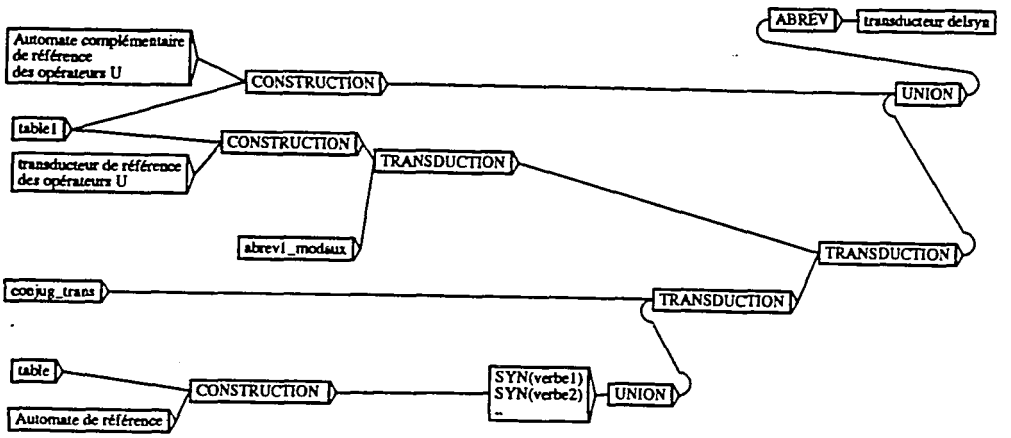
A ce point un extrait de *DELSYN* peut être la liste suivante

Nhum$_0$ V_0-<*agacer-0*> *Nhum*$_1$
Nhum$_0$ U_0-<*continuer-0*> *à* V_0-<*agacer-0*> *Nhum*$_1$
Nhum$_0$ *VAux*$_0$-<*avoir*> *Vpp*$_0$-<*agacer-0*> *Nhum*$_1$
Nhum$_0$ *VAux*$_0$-<*aller*> *Vinf*$_0$-<*agacer-0*> *Nhum*$_1$
Qu P_0 V_0-<*abasourdir-0*> *Nhum*$_1$
Qu P_0 *n'en* U_0-<*finir-0*> *pas de* V_0-<*abasourdir-0*> *Nhum*$_1$
Qu P_0 *VAux*$_0$-<*venir*> *deVinf*$_0$-<*abasourdir-0*> *Nhum*$_1$
Le fait Qu P_0 V_0-<*abattre-0*> *Nhum*$_1$
Nnr$_0$ V_0-<*agacer-0*> *Nhum*$_1$
$V^1{}_0W$ V_0-<*afflige-0*> *Nhum*$_1$
Nhum$_0$ *VAux*$_0$-<*venir*> *de se* V_0-<*agacer-0*> *de ce Qu* P_1
Nhum$_0$ *se VAux*$_0$-<*être*> *Vpp*$_0$-<*agacer-0*> *de ce Qu* P_1

Extrait de DELSYN et 4.5 (727.000 éléments)[10]

Figure 4.5g

et le procédé de fabrication peut être résumé dans le schéma de fabrication de la figure 4.5h :



Chaîne de construction de DELSYN

Figure 4.5h

4.8. PHRASES AVEC NEGATION

4.8.1. Les structures du type N_0 *ne* V_0 *pas* W

[10]A partir de ce stade de la présentation les chiffres dépendent des choix de solution. Nous donnons le nombre d'éléments dans le dictionnaire *DELSYN* tel que nous l'utilisons. Ici, une représentation complètement développée donne un dictionnaire de plus de 40.000.000 d'entrées représenté par un automate de 1200 états et 7600 transitions, dictionnaire qui reste donc parfaitement manipulable.

Nous en sommes au point où, pour l'ensemble des verbes décrits, nous pouvons analyser les phrases explicitement décrites et leurs formes conjuguées à tous les temps. Mais nous ne sommes capables de n'analyser que des phrases affirmatives. Une phrase telle que :

(1) *Paul n'agace jamais Paul*

n'est pas analysable sans l'ajout de la structure

(2) $Nhum_0$ *ne* V_0-<*agacer-0*> *jamais* $Nhum_1$.

et comme on veut aussi pouvoir analyser les autres phrases à négation :

(3) *Paul n'a jamais agacé Paul*
(4) *Paul ne va pas agacer Luc*
(5) *Paul ne vient pas d'agacer Luc*
(6) *Paul ne vient pas d'être agacé par Luc*

il faut tenir compte de l'enrichissement des temps composés de la section précédente.

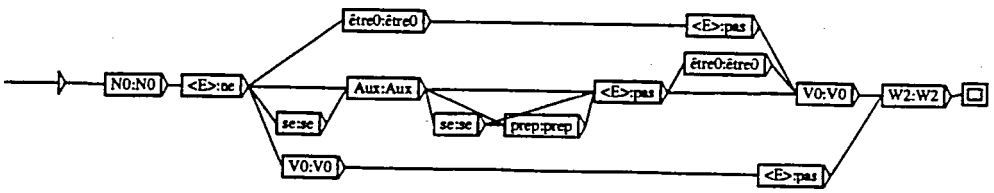
Pour inclure les négations, nous suivons l'étude des négations de J-T. Julia 1991 dans laquelle l'ensemble des structures de phrases avec négation se présente déjà sous la forme de listes de structures. Nous montrons ici que cette description possède un équivalent formel naturel dans les transducteurs, ce qui permet d'unifier le traitement des négations avec le reste de l'analyse.

Nous donnerons ici un échantillon des problèmes rencontrés.

Si nous commençons par nous limiter à la négation simple *ne ... pas*, la structure générale est la suivante :

(7) N_0 *ne* V_0 *pas* W

mais il faut tenir compte, comme le montre les exemples (3) à (4), de l'existence possible d'auxiliaires. Le transducteur T_1 de la figure 4.8a est ainsi celui qui appliqué à *DELSYN*, donne la liste des structures de type (7) qui incorpore les temps composés.



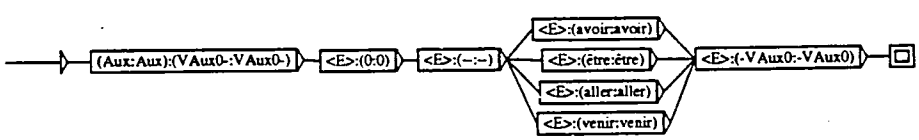
Transducteur T_1

Figure 4.8a

Il faut noter que T_1 comporte des abréviations (N_0, *Aux* par exemple) et qu'on ne peut donc véritablement l'appliquer directement à *DELSYN*. Pour résoudre ce problème, on introduit un transducteur *ABREV_NEGATION* qui rend explicite l'ensemble des abréviations utilisées dans le transducteur de la négation. Ainsi l'abréviation *Aux* est en fait équivalent à la liste suivante :

VAux$_0$*- 0 -- avoir -VAux*$_0$
VAux$_0$*- 0 -- être -VAux*$_0$
VAux$_0$*- 0 -- venir -VAux*$_0$
VAux$_0$*- 0 -- aller -VAux*$_0$

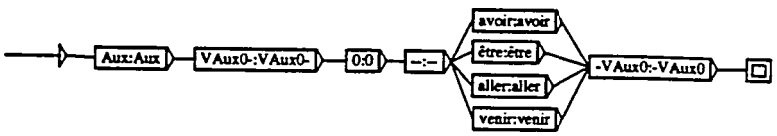
ce qui peut se modéliser par le transducteur T_2 de la figure 4.8b qui associe à *Aux* cette liste de séquences.



Transducteur T_2 d'explicitation de l'abréviation *Aux*

Figure 4.8b

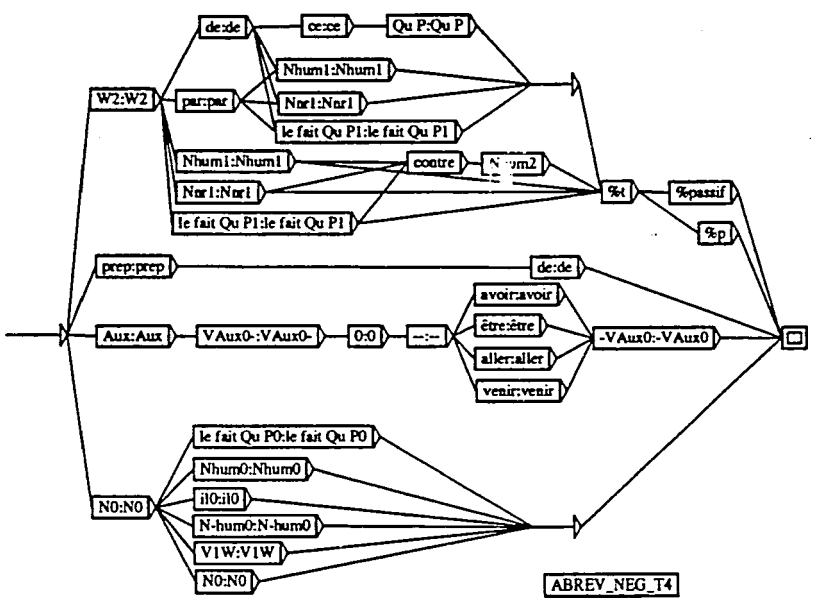
T_2 n'est pas d'une lecture ni d'une constitution aisée, nous avons donc choisi de représenter ce type particulier de transduction par des automates. Ces transducteurs sont en effet particuliers car les séquences de l'ensemble de départ sont réduites à un mot. Ainsi, l'automate A_3 de la figure 4.8c représente-il la même transduction T_2. On définit ainsi la fonction qui, à chaque étiquette portée par l'état initial (donc ici l'étiquette *Aux:Aux* uniquement), associe l'ensemble des séquences commençant à l'état pointé par cette transition (donc ici *VAux*$_0$*-:VAux*$_0$*- 0:0 --:-- avoir:avoir -VAux*$_0$*:-VAux*$_0$).



Définition de la transduction T_2 par un automate

Figure 4.8c

Pour compléter les abréviations utilisées dans T_1, il nous faut utiliser la transduction définie par l'automate de 4.8d. Cette transduction, appelée *ABREV_NEG_T4*, a été construite pour s'appliquer spécifiquement à toutes les structures de la table 4.



Définition de la transduction *ABREV_NEG_T4* par un automate

Figure 4.8d

Ainsi l'ensemble des structures de type (7) sera obtenu par l'application successive de deux transductions. Tout d'abord, $T_3 = ABREV\_NEG\_T_4(T_1)$ définit la transduction qui doit s'appliquer à *DELSYN* pour créer les structures des phrases avec négation. Ensuite, $T_3(DELSYN) = ABREV\_NEG(T_1)(DELSYN)$ donnera l'ensemble des structures de type (7).

Le transducteur T_1 pose encore un problème de compatibilité avec la description que nous avions faite, dans T_1 nous ne tenons aucun compte, par exemple, du type d'auxiliaire qui est employé avec le verbe. Mais comme T_1 ne s'applique qu'à des structures correctes la description des temps composés faite précédemment est conservée, seule les structure correctes

sont conservées. Par exemple dans *DELSYN* nous avons des séquences du type *venir de* et non pas de type *aller de*, quand T_1 s'applique à *DELSYN*, même s'il avait pris en compte la possibilité que *aller de* existe, la fonction ne s'appliquerait qu'à *aller* ou *venir de* et ne donnerait donc que des négations correctes. Il est important de remarquer que les structures incorporées dans *DELSYN* sont des structures correctes et que nous ne sommes pas obligé, au moment de décrire les négations, de refaire les description examinées aux sections précédentes.

4.8.2. Les structures de la négation *ne ... pas*

Maintenant que nous avons montré que la description de (7) donnée en T_1 permet véritablement d'inclure cet ensemble de structures à notre lexique syntaxique *DELSYN*, il nous faut voir comment étendre cette description, donc comment étendre T_1, pour obtenir une description de l'ensemble des négations.

La négation simple *pas* peut apparaître dans d'autres structures que (7), les phrases :

(8) *Pas un n'est arrivé l'heure*
(9) *Pas un de ces étudiants n'est arrivé à l'heure*
(10) *Luc ne donne à Luc pas une de ses affaires*

montrent qu'il faut également tenir compte de structures du type

(11) *pas* UN_0 *ne* V_0 *W*
(12) *pas* UN_0 *N ne* V_0 *W*
(13) N_0 *ne* V_0 *Prép* N_2 *pas* UN_1 N_1

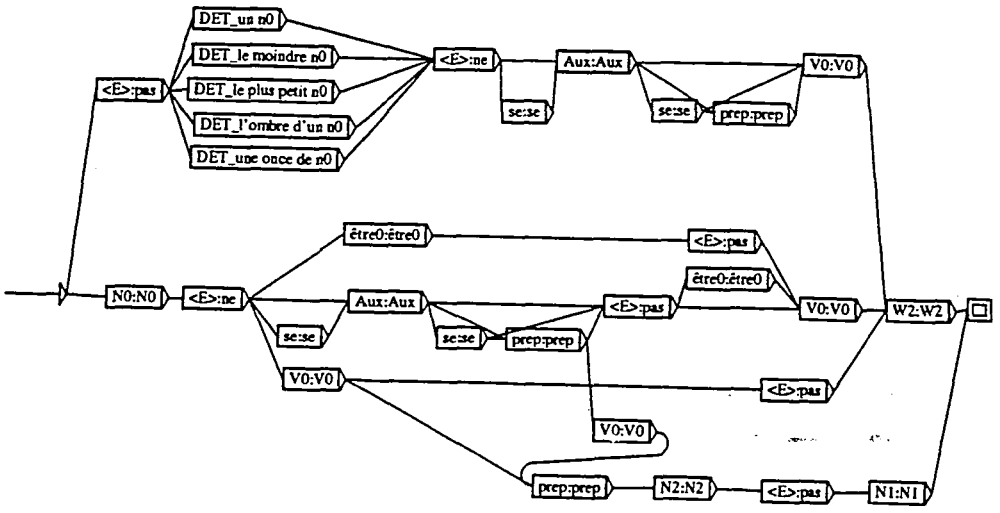
qui sont plus difficiles à représenter car le groupe nominal qui suit *pas* est contraint. Le déterminant *le* seul est par exemple interdit:

(14) * *Pas l'étudiant n'est arrivé à l'heure*

alors qu'il devient possible quand il fait parti d'un déterminant plus complexe comme dans :

(15) *Pas le moindre étudiant n'est arrivé à l'heure*

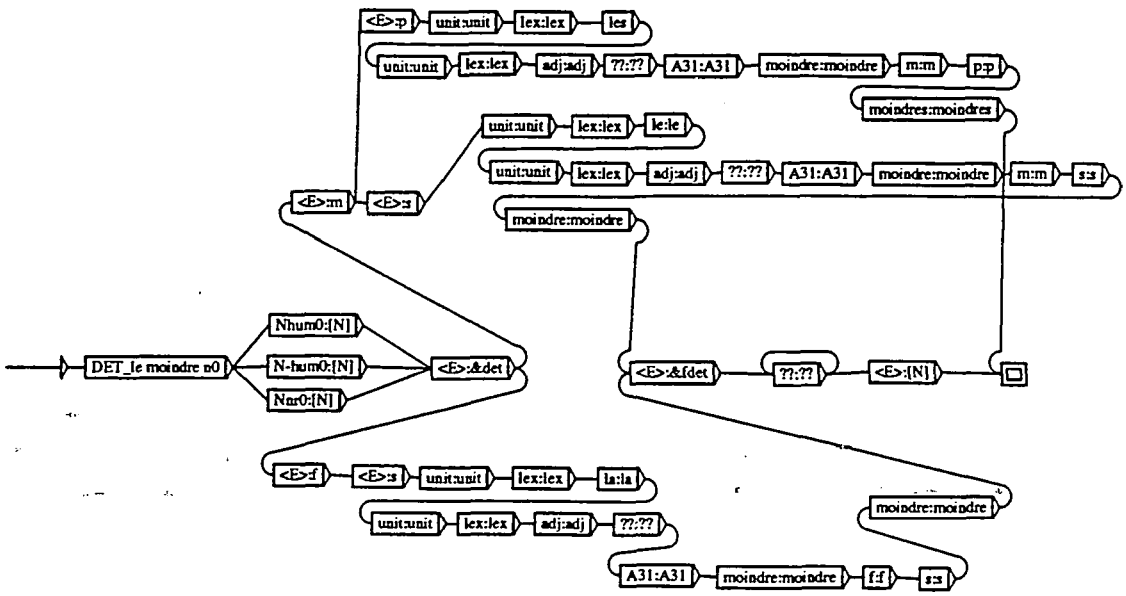
Nous représentons ces structures par le transducteur de la figure 4.8e :



Transducteur des structures de la négation simple *ne ... pas*

Figure 4.8e

Se pose ici le problème de la représentation des contraintes que la structure globale de la phrase (ici structure avec négation) impose sur la structure interne des groupes nominaux qui la constitue. On doit alors analyser *DET_le_moindre* n_0 comme un groupe nominal dont le déterminant est *le moindre*. L'utilisation des transducteurs permet de préciser la structure interne d'un groupe nominal. L'automate d'explicitation de *DET_le_moindre* n_0 de la figure 4.8f indique ainsi que dans l'automate 4.8e il faut remplacer *DET_le_moindre* n_0 par tout ce qui suit la transition de l'état initial (de 4.8f) pointée par l'étiquette *DET_le moindre* n_0, donc par l'automate (un transducteur en fait) dont l'état initial à trois transitions étiquetées $Nhum_0$*:[N]*, *N-hum*$_0$*:[N]* et Nnr_0*:[N]*.



Automate d'explicitation de l'abréviation *DET_le moindre n0*

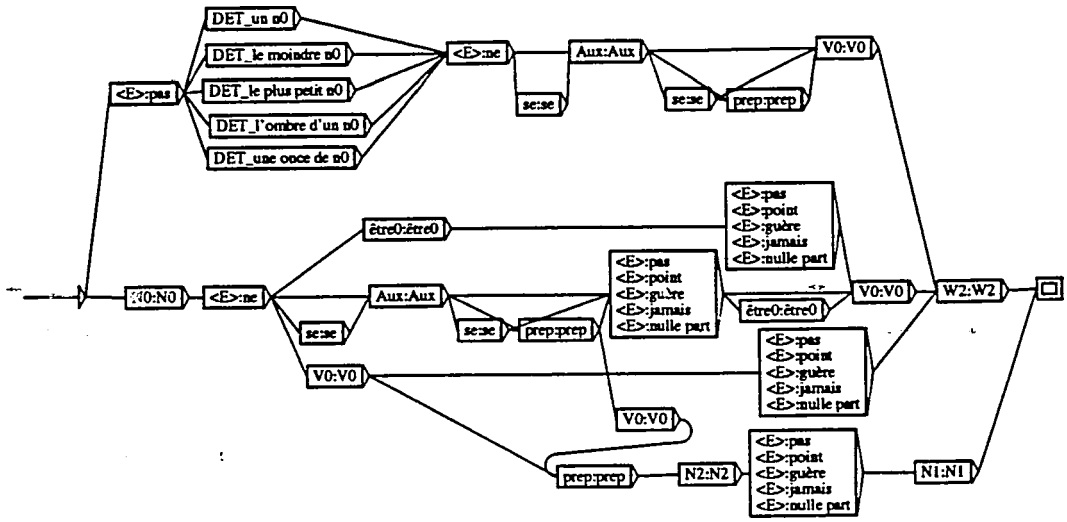
Figure 4.8f

On peut alors vérifier que (15) se voit donc associée, par *f_DELSYN*, la forme :

(16) *pas* ([N] *&det le moindre &fdet étudiant* [N]) *n'est arrive à l'heure*

La transduction f_DELSYN a donc reconnu la négation et a marqué le déterminant à l'intérieur du groupe nominal. On poursuit sur (16) l'analyse du groupe nominal entre [N] et [N]. Par le même procédé, (14) n'a pas d'image et n'est donc pas analysée comme une phrase correcte.

Le transducteur de la figure 4.8e s'étend naturellement aux négations simples avec *point, plus, guère, jamais, nulle part* que nous représentons à la figure 4.3fg.



Transducteur des négations simples avec *pas, point, plus, guère, jamais, nulle part.*

Figure 4.8g

4.8.3. Négations en *ne ... rien* et *ne ... personne*

Parmi les structures données dans J.T.Julia 1991 pour les négations *ne. ... rien* et *ne ... personne* on trouve par exemple :

(16) N_0 *ne* V_0 *rien Modif_X*

où *Modif_X* est un modifieur contraint de *rien* comme dans

(17) *Je n'ai rien du tout*
(18) *Je n'ai rien de tout cela*
(19) *Je n'ai rien de ce que Luc a acheté*
(20) *Je n'ai rien du tout de ce que j'aurais dû avoir*

Il faut également tenir compte de phrases telles que :

(21) *Rien de tout cela n'aurait dû arriver*

De manière parallèle, on a pour la négation *ne ... personne* une structure comme :

(22) N_0 *ne* V_0 *personne Modif_X Prep* N_2

qui est celle des phrases :

(23) *Je n'ai vu personne*
(24) *Je n'ai vu personne que je connaisse*
(25) *Je n'ai vu personne de ma connaissance*
(26) *Je n'ai vu personne de connu*
(27) *Je n'ai vu personne d'entre vous*

qui posent de nouveaux problèmes d'analyse. Il faut en effet reconnaître une proposition relative dans (24). Une solution qui peut être envisagée consiste à inclure dans *DELSYN* les structures qui permettent d'associer aux phrase (23) à (26) les analyses partielles suivantes :

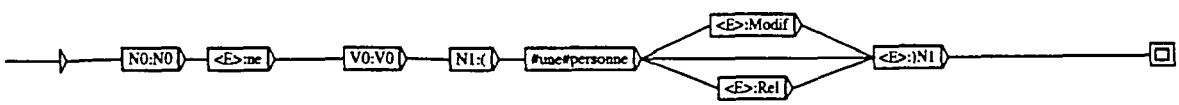
(28) *Je n'ai vu* ([N] *#une# personne* [N])$_{N1}$
(29) *Je n'ai vu* ([N] *#une# personne que je connaisse* [N])$_{N1}$
(30) *Je n'ai vu* ([N] *#une# personne de ma connaissance* [N])$_{N1}$
(31) *Je n'ai vu* ([N] *#une# personne de connu* [N])$_{N1}$

Les étapes suivantes de l'analyse auront alors à analyser les séquences entre [N] comme des groupes nominaux indépendants.

Pour permettre cette analyse, il faut construire un transducteur tel que celui de la figure 4.8h qui s'appliquera à l'ensemble des structures sans négation de *DELSYN*. Si nous prenons l'exemple de la structure :

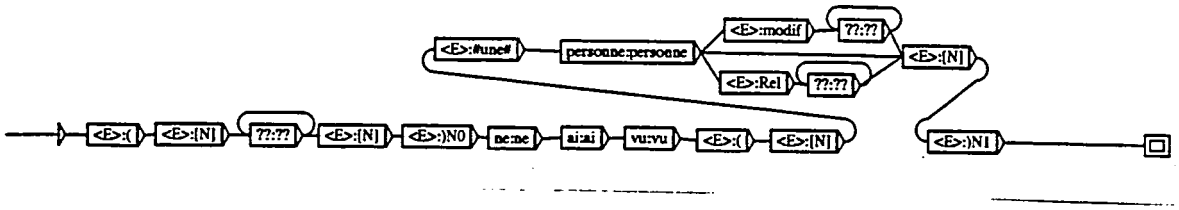
(32) N_0 V_0-<*voir-0*> N_1

qui fait partie de *DELSYN*, l'application du transducteur de 4.8h donne, après la transformation des abréviations, un transducteur tel que celui de la figure 4.8i qui, inclus dans *f_DELSYN*, permet les analyses (28) à (31).



Extrait du transducteur d'introduction de la négation *ne ... personne*.

Figure 4.8h



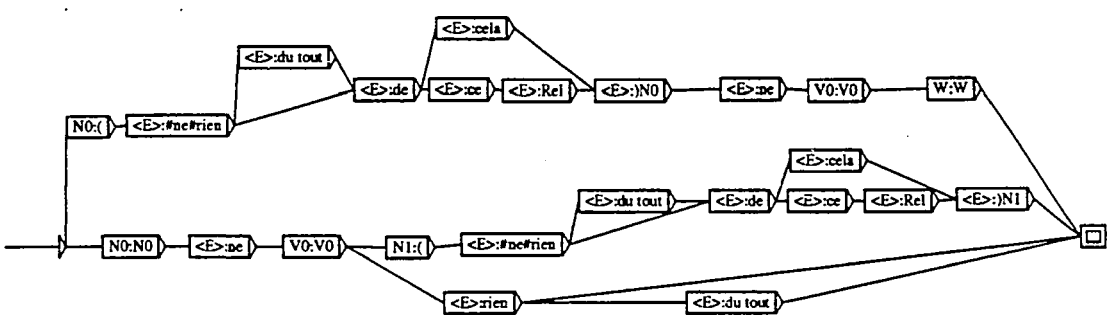
Extrait de *f_DELSYN* permettant une analyse de N_0 *voir* N_1 avec une négation *ne .. personne*

Figure 4.8i

Cette solution pose cependant d'autres problèmes. Elle suit en fait implicitement une analyse linguistique qui propose les phrases abstraites (28) à (31) comme source transformationnelle de (23) à (26). D'une part, cette analyse n'est pas bien justifiée (l'analyse de (27) est difficile de cette manière), d'autre part la solution n'est plus applicable pour la négation *ne ... rien* puisque la séquence *rien du tout* par exemple n'est possible qu'avec la particule préverbale *ne*. Il serait donc incorrect d'inclure *rien du tout* dans la description générale des groupes nominaux (indépendamment de toute négation). La solution que nous avons adoptée consiste à associer à (19) l'analyse partielle suivante :

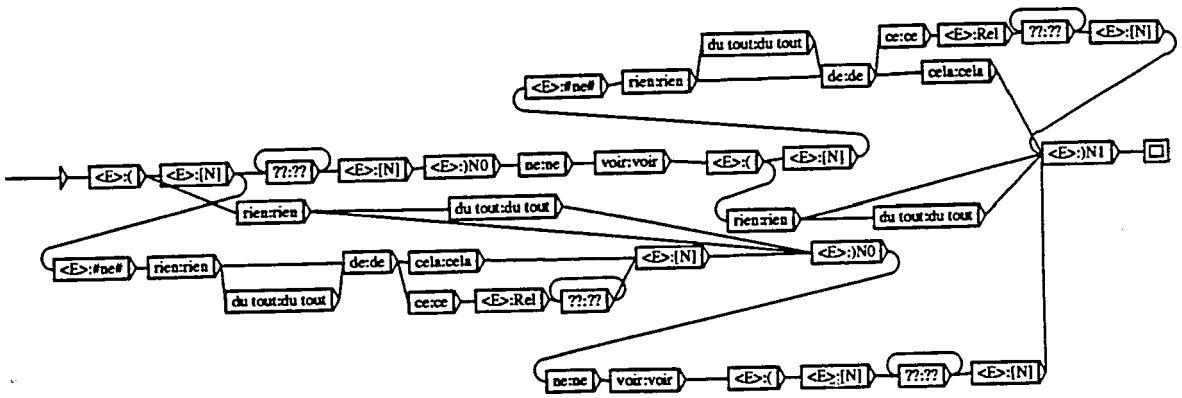
(33) *Je ne ai* ([N] *#ne# rien de ce que Luc a acheté* [N])

dans laquelle la suite de l'analyse découpera la séquence placée entre les [N] comme un groupe nominal dont le déterminant indécomposable sera *#ne# rien*. Plus concrètement, le transducteur de la figure 4.8j (équivalent à (16)) introduira la négation *ne ... rien* et s'appliquera aux structures sans négation de *DELSYN* pour donner les nouvelles structures qui enrichiront *DELSYN*. De ces dernières on déduit une transduction (donc une partie de *f_DELSYN*) telle que celle de la figure 4.8k qui permet l'analyse des phrases (17) à (20).



Extrait du transducteur d'introduction de la négation *ne ... rien* dans les structures de *DELSYN*

Figure 4.8j



Extrait de *f_DELSYN* permettant l'analyse des phrases avec la négation *ne ... rien*

Figure 4.8k

Il est important de remarquer que le choix des solutions pour l'analyse des phrases de ce paragraphe suivant de très près les problèmes soulevés par une analyse purement linguistique menée indépendamment de tout formalisme. Cela contribue à montrer que ce formalisme, minimal, permet de trouver des représentations dictées par une analyses précises des données et non par des propriétés propres au formalisme. L'utilisation des automates et des transducteurs n'introduit donc ici aucun problème périphérique propre à la représentation choisie.

4.8.4. Composition de deux négations

Les éléments précédents peuvent se combiner entre eux dans des phrases comme

(28) *Paul ne dit jamais rien*
(29) *Il n'y a guère que Paul qui soit au courant*
(30) *Luc n'est plus jamais revenu ici*

qui s'analyse respectivement selon les structures suivantes:

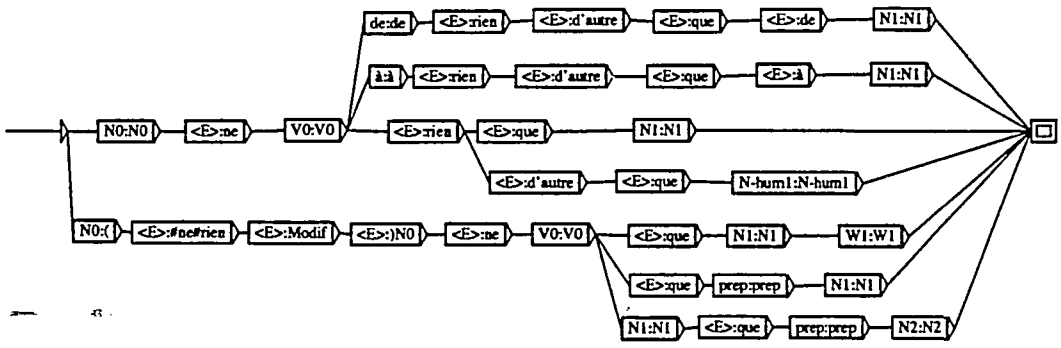
(31) N_0 *ne* V_0 *jamais rien*
(32) N_0 *ne* V_0 *guère que* N_1
(33) N_0 *ne* V_0 *plus jamais*

La composition de ces semi-négations est fortement contrainte et apparemment imprévisible (voir encore J.T.Julia 91) et la description de ces contraintes se fait naturellement par des automates. Prenons la composition *que rien* observée par exemple dans :

(34) *Rien n'est juste que ce résultat*
(35) *Je n'ai acheté rien qu'une tarte*
(36) *Elle ne dit rien que des bêtises*
(37) *Je ne voit rien d'autre de ce que j'ai apporté qu'un gâteau.*

Il faut donc décrire explicitement ces structures dans un automate tel que celui de la

figure 4.81 :



Extrait du transducteur des structures avec *rien* et *que*.

Figure 4.81

Cette combinatoire est fortement contrainte et apparemment peu prévisible, on ne voit pas comment en rendre compte de manière plus simple que par ce type de représentation.

4.8.5. Négations figées

En plus des cas de composition de semi-négations comme celles de 4.8.4, on trouve des négations qu'on peut appeler figées telles que *ne ... pas ... pour autant* dans des phrases comme :

(38) *Luc n'a pas acheté ce livre pour autant*
(39) *Luc n'a pas acheté pour autant ce livre*
(40) *Lyc n'a pas pour autant acheté ce livre*
(41) *Luc n'a, pour autant, pas acheté ce livre*
(42) *Luc, pour autant, n'a pas acheté ce livre*
(43) *Pour autant, Luc n'a pas acheté ce livre*

dans lesquelles *pour autant* a la mobilité d'un adverbe, mais ne peut apparaître qu'en présence de la négation *ne ... pas* (ou d'autres négations).

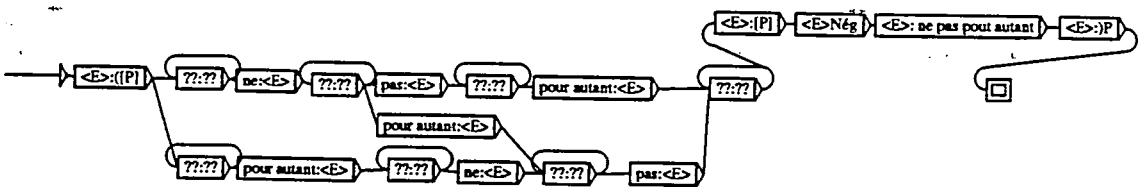
Formellement, il n'y a pas de problème nouveau de représentation pour chacune de ces structures, mais on se rend compte que lister explicitement dans *DELSYN* toutes les structures des phrases (38) à (43), c'est-à-dire :

(44) *Nhum$_0$ ne Aux pas Vpp-<acheter-0> N$_1$ pour autant*
(45) *Nhum$_0$ ne Aux pas Vpp-<acheter-0> pour autant N$_1$*
(46) *Nhum$_0$ ne Aux pas pour autant Vpp-<acheter-0> N$_1$*
(47) *Nhum$_0$ ne Aux pour autant pas Vpp-<acheter-0> N$_1$*
(48) *Nhum$_0$ pour autant ne Aux pas Vpp-<acheter-0> N$_1$*

(49) *Pour autant* $Nhum_0$ *ne Aux pas Vpp-<acheter-0>* N_1

pose très vite un problème de taille, puisqu'il faut disposer des structures équivalentes pour les autres 12.000 verbes répertoriés. Même la forte factorisation que permet l'utilisation des automates ou des transducteurs trouve ici ses limites.

La solution de Zellig Harris consiste à considérer la négation comme un opérateur agissant sur la phrase. On considère donc que le prédicat de la phrase n'est plus le verbe mais *ne..pas..pour autant.* Cela conduit à inclure le transducteur de la figure 4.8m dans *f_DELSYN.*



Premier essai d'analyse des phrases en *ne ... pas ... pour autant.*

Figure 4.8m

qui appliqué à (38) donne (50) = *f_DELSYN*(38)

(50) ([P] *Luc a acheté ce livre* [P] *Nég ne pas pour autant 0 Pred*)$_P$

qui signifie que la phrase placée entre "(" et ")P" a pour prédicat la forme *pas ... pour ... autant* qui porte sur la séquence *[P] Luc a acheté ce livre [P].*Cette dernière est à analyser comme une phrase par les procédés déjà décrits.

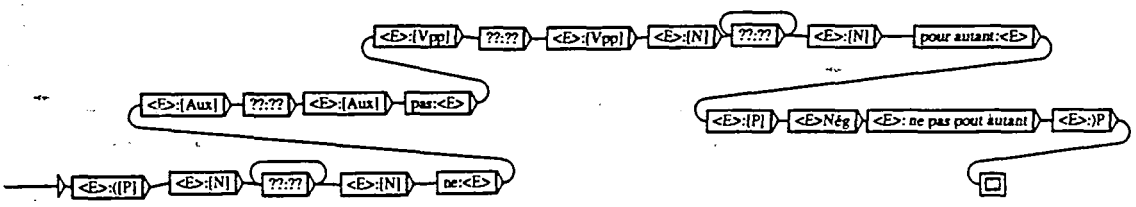
Ce premier transducteur ne rend cependant pas compte intégralement des informations contenues dans les structures (44) à (49). Il se contente d'effacer les trois séquences *ne pas pour autant* quelle que soit leur position et l'ordre dans lequel elles apparaissent. En particulier, les points d'insertion de *ne pas pour autant* ne sont plus spécifiés, le transducteur analyse donc correctement une phrase telle que :

(51) **Luc ne pour autant a pas acheté ce livre*

qui devrait être rejetée. (50) doit en effet contenir l'information que *Luc* devra être reconnu comme sujet de la phrase affirmative, que *a* en est l'auxiliaire, etc. Nous associerons donc à (38) non pas (50) mais (52)

(52) ([P] [N] *Luc* [N] [Aux] *a* [Aux] [Vpp] *acheté* [Vpp] [N] *ce livre* [N] [P] *Pred pas pour autant 0 Pred*)$_P$

ce qui se fait par le transducteur de la figure 4.8j :



Extrait de f_DELSYN pertinent à l'analyse des phrases avec *ne..pas..pour autant.*

Figure 4.8j

dont on voit qu'il reflète exactement l'automate des structures (44) à (49), il correspond donc à l'intuition la plus simple qu'on puisse avoir pour décrire les propriétés d'une telle négation. La forme (52) indique non seulement quelle phrase (sans négation) reste à analyser, mais aussi elle indique selon quelle structure elle devra être analysée.

Nous sommes donc revenu à la situation générale des verbes où nous décrivons un ensemble de structures ((43) à (46)) que nous intégrons à *DELSYN* et que nous pouvons transformer en un transducteur (*f_DELSYN*) qui, appliqué aux phrases, permet leur analyse.

Notons également que cette solution est utilisable pour toutes les négations, mais qu'une petite perte d'efficacité en résulte, il faut a priori un cycle de plus pour finir l'analyse.

4.8.6. Un point sur la chaîne de construction de *DELSYN* et *f_DELSYN*

Au stade que nous avons atteint dans cette présentation, celui de l'analyse des phrases simples construites sur des verbes de la table 4, avec des conjugaisons composées, des modaux et des négation, nous donnons un échantillon de *DELSYN* :

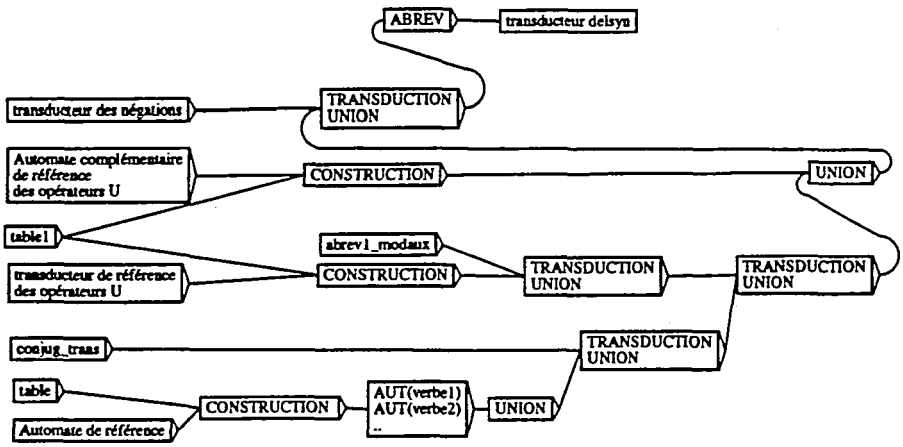
$Nhum_0$ ne V_0-<agacer-0> pas $Nhum_1$
$Nhum_0$ ne $VAux_0$-<avoir> jamais Vpp_0-<agacer-0> que $Nhum_1$
$Nhum_0$ ne $VAux_0$-<aller> guère $Vinf_0$-<agacer-0> $Nhum_1$
Qu P_0 ne V_0-<abasourdir-0> pas $Nhum_1$
$Nhum_0$ V_0-<agacer-0> $Nhum_1$
$Nhum_0$ $VAux_0$-<avoir> Vpp_0-<agacer-0> $Nhum_1$
$Nhum_0$ $VAux_0$-<aller> $Vinf_0$-<agacer-0> $Nhum_1$
Qu P_0 V_0-<abasourdir-0> $Nhum_1$
Qu P_0 $VAux_0$-<venir> de$Vinf_0$-<abasourdir-0> $Nhum_1$
Le fait Qu P_0 V_0-<abattre-0> $Nhum_1$
Nnr_0 V_0-<agacer-0> $Nhum_1$
$V^1{}_0W$ V_0-<afflige-0> $Nhum_1$
$Nhum_0$ $VAux_0$-<venir> de se V_0-<agacer-0> de ce Qu P_1
$Nhum_0$ se $VAux_0$-<être> Vpp_0-<agacer-0> de ce Qu P_1

Extrait de DELSYN (8.300.000 éléments 2400 états 7200 transitions)[11]

Figure 4.3k

et le schéma de construction peut se résumer dans la figure 4.3l :

[11] A partir de maintenant nous ne donnerons plus que la taille de l'automate qui représente le dictionnaire, c'est la quantité qui se rapproche le plus de la quantité d'informations stockées. Mais pour comparer la taille de notre grammaire avec la nombre de règles d'une grammaire d'unification c'est bien le nombre d'éléments du dictionnaire qu'il faut considérer.



Construction de DELSYN et f_DELSYN jusqu'à 4.3

Figure 4.3l

4.9. PRONOMINALISATION

Nous analysons maintenant une phrase telle que :

Jean ne s'en agace guère

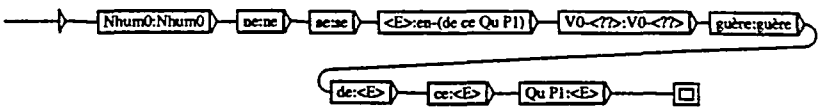
où, en plus de la négation, il nous faut pouvoir reconnaître qu'une pronominalisation en *en* est possible et que la particule préverbale *en* s'insère entre *se* et *agace*. De plus il est intéressant de pouvoir exprimer que ce *en* peut provenir d'un complément *de* N_1 =: *de ce Qu* P_1 de la structure :

(1) *Nhum*$_0$ *ne se* V_0-<*agacer*> *guère de ce Qu* P_1.

Nous allons décrire la pronominalisation de *de ce Qu*P_1 par une fonction, *f_ppv*, qui associe à la structure (1) :

(2) *Nhum*$_0$ *ne se en-(de ce Qu* P_1*)* V_0-<*agacer*> *guère.*

Cette transduction qu'on peut appeler fonction de pronominalisation et que l'on note *f_ppv* est donc la fonction qui associe à une structure quelconque dont les arguments ne sont pas pronominalisés, une structure qui est pronominalisée. Nous donnons sur figure suivante l'extrait de cette transduction pertinent à la création de (2) à partir de (1).



Extrait de *f_ppv* pertinent à la transformation (1) ≡ (2)

Figure 4.9a

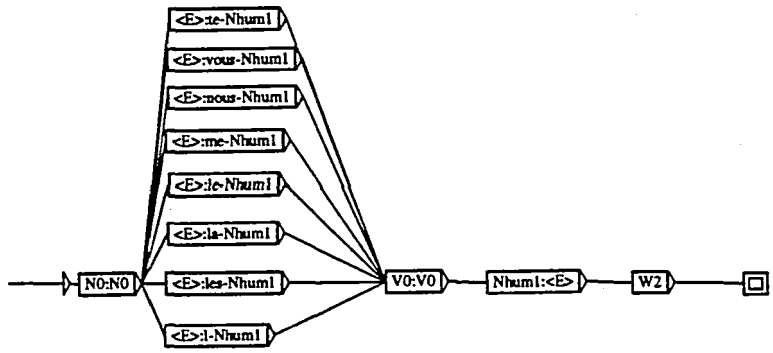
Ce transducteur est cependant plus complexe à construire que les deux précédents. Dans le cas de la conjugaison et même de la négation, il était possible de construire un transducteur général qui peut s'appliquer à chaque automate *SYN(verbe)* de chaque verbe simple. La fonction *f_ppv* dépend de la forme des arguments du verbe. Comme il est difficile de décrire dans un seul transducteur toute la combinatoire des formes d'arguments pour les 12.000 verbes, on construira un transducteur spécifique à chaque table. cela permet d'être sûr que toutes les structures sont traitées. Nous noterons *f_ppv_T*$_4$ la transduction construite pour les verbes de la table 4.

La première étape de la construction de *f_ppv_T*$_4$ consiste donc à examiner l'ensemble des structures d'une table et, pour chacune d'entre elles, à donner la liste des pronominalisations possibles. Reprenons donc les automates de la table 4, l'automate de référence de la figure 4.3b montre qu'une structure

(3) $N_0\ V_0\ Nhum_1$

peut se rencontrer. La pronominalisation de $Nhum_0$ peut engendrer *il, elle, elles, ils, je, tu, ,il, nous, vous* ou *on* qui ne modifie pas l'ordre des arguments, mais qui introduit la contrainte nouvelle que les insertions sont bloquées entre le sujet et le verbe.

$Nhum_1$ peut se pronominaliser et introduire les particules préverbales (ppv) *me, te, le, la les, nous, vous, lui*, ce qui se formalise par le transducteur suivant:



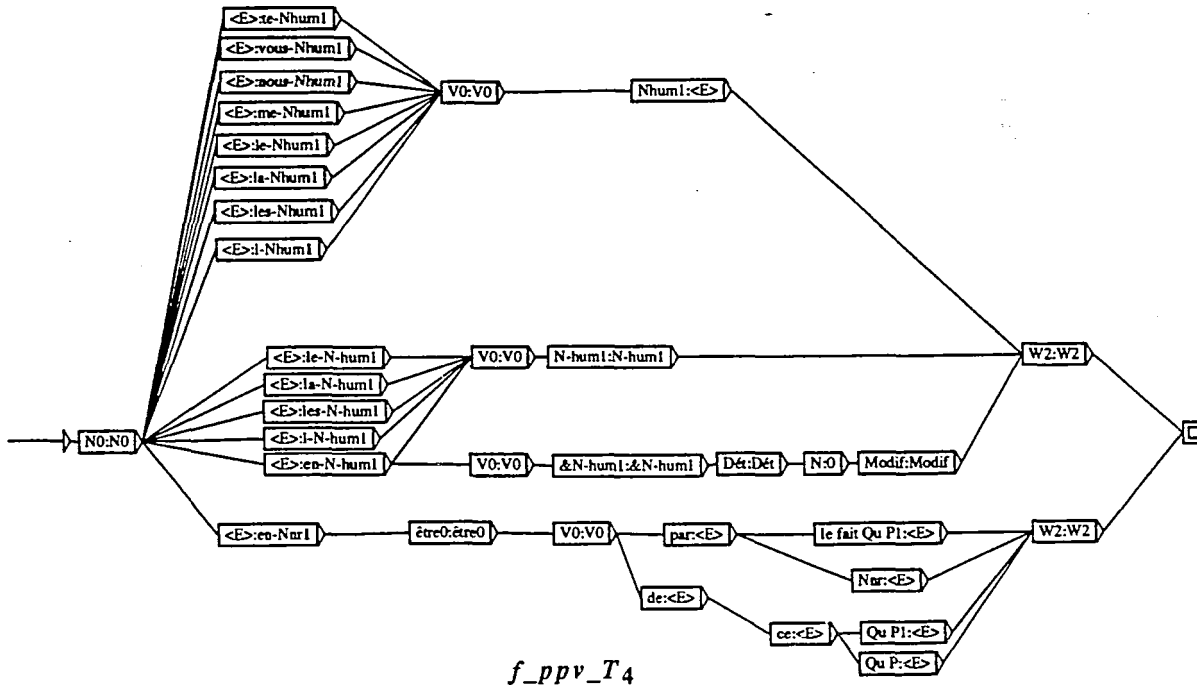
Extrait de f_ppv_T4

Figure 4.9b

On peut vérifier que (4) $\subseteq f\_ppv\_T_4((3))$

(4) N_0 *le-Nhum*$_1$ V_0

Par conséquent $f\_ppv\_T_4((3))$ sera inclus dans *DELSYN*. Le même principe donne pour l'ensemble des structures répertoriées à la table 4 le transducteur $f\_ppv\_T_4$ de la figure 4.9c :

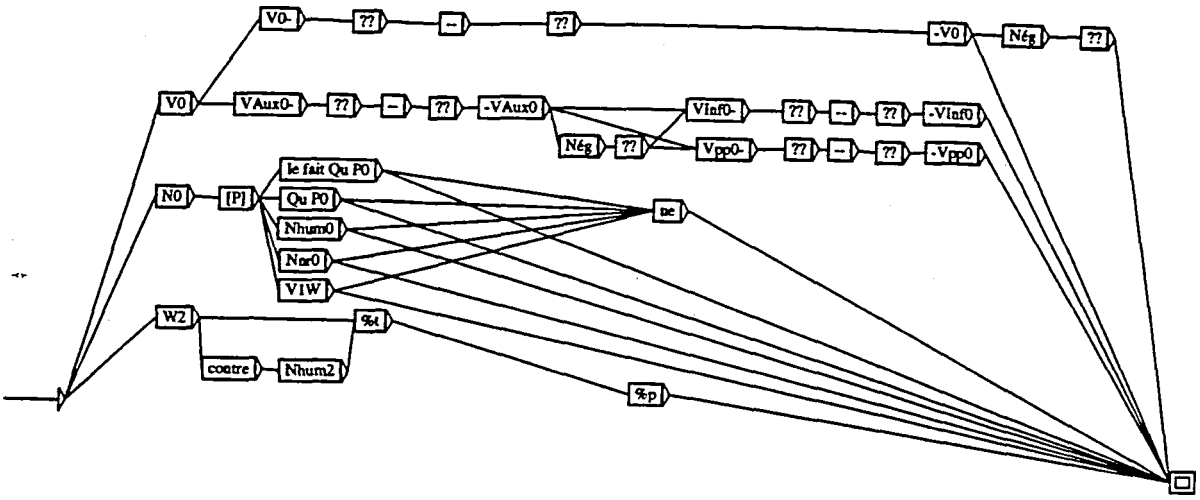


f_ppv_T4

Figure 4.9c

Comme pour les négations, le transducteur $f\_ppv\_T_4$ utilise des abréviations que l'on spécifie dans l'automate *ABREV_ppv* de la figure 4.9d. Par exemple nous avons noté N_0:N_0 pour dire que le sujet, quelque soit sa forme, est conservé. N_0:N_0 doit donc être remplacé par *Nhum*$_0$:*Nhum*$_0$, *Qu* P_0:*QU* P_0, etc. En d'autres termes, tous les couples de type *A*:*A* (donc

N_0:N_0) où A est l'étiquette d'une transition de l'état initial qui pointe vers l'état q_i doit être remplacé par les séquences u_1:u_1 .. u_n:u_n où u_1 .. u_n est l'étiquette d'un chemin de q_i à un état terminal. *ABREV_ppv* définie ainsi une transduction d'une manière plus simple à représenter que la notation standard, c'est le procédé qui a déjà été employé pour les négations.



ABREV_ppv

Figure 4.9d

Le transducteur qu'on applique véritablement à *DELSYN* sera donc *ABREV_ppv* (*f_ppv_T*$_4$) et par conséquent, l'enrichissement de *DELSYN* par les structures avec pronominalisation consiste à remplacer *DELSYN* par *DELSYN* $\cup$ *ABREV_ppv(f_ppv_T*$_4$*)DELSYN*.

Le transducteur décrit donc non seulement la combinaison des particules préverbales telle qu'un automate simple peut les représenter mais également les sources de ces pronoms. En particulier, on a construit une grammaire locale des particules préverbales spécifique à chaque verbe. Ce type de donnée peut être exploité séparément dans une grammaire locale (voir chapitre 5).

Au stade de la présentation nous donnons un nouvel extrait du dictionnaire syntaxique *DELSYN* :

$Nhum_0$ ne V_0-<agacer-0> pas $Nhum_1$
$Nhum_0$ ne le_$Nhum_1$ V_0-<agacer-0>
$Nhum_0$ ne $VAux_0$-<avoir> jamais Vpp_0-<agacer-0> que $Nhum_1$
$Nhum_0$ ne $VAux_0$-<aller> guère $Vinf_0$-<agacer-0> $Nhum_1$
Qu P_0 ne V_0-<abasourdir-0> pas $Nhum_1$
$Nhum_0$ V_0-<agacer-0> $Nhum_1$
$Nhum_0$ $VAux_0$-<avoir> Vpp_0-<agacer-0> $Nhum_1$
$Nhum_0$ $VAux_0$-<aller> $Vinf_0$-<agacer-0> $Nhum_1$
Qu P_0 V_0-<abasourdir-0> $Nhum_1$
Qu P_0 $VAux_0$-<venir> de$Vinf_0$-<abasourdir-0> $Nhum_1$
Qu P_0 $VAux_0$-<venir> de le-$Nhum_1$ $Vinf_0$-<abasourdir-0>
Le fait Qu P_0 V_0-<abattre-0> $Nhum_1$
Nnr_0 V_0-<agacer-0> $Nhum_1$
$V^1{}_0W$ V_0-<afflige-0> $Nhum_1$
$Nhum_0$ $VAux_0$-<venir> de se V_0-<agacer-0> de ce Qu P_1
$Nhum_0$ se $VAux_0$-<être> Vpp_0-<agacer-0> de ce Qu P_1

Extrait de DELSYN (4.700 états et 11.300 transitions)

Figure 4.9e

et la construction de *DELSYN* peut se résumer par le schéma suivant :

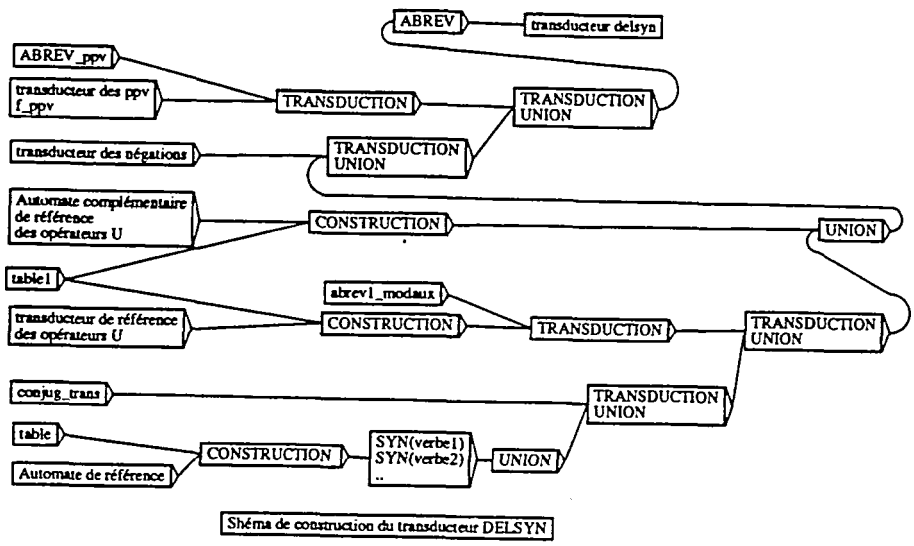


Schéma de construction de *DELSYN* jusqu'au pronominalisation

Figure 4.9f

où l'on voit que la dernière étape consiste à appliquer *ABREV_ppv* sur *f_ppv_T4* puis à appliquer la transduction résultat sur *DELSYN* puis de faire l'union de ce dernier résultat avec le *DELSYN* d'origine.

4.10. PHRASES FIGEES

Parmi les phrases à analyser figurent des phrases telles que :

(1) *Dieu seul le sait*

L'analyse de cette phrase comme phrase libre donnera, dans le meilleur des cas, une structure de type

(2) N_0 *Adv le-ppv-QuP*$_1$ V_0-<*savoir-3*>

dont le verbe principal est *savoir*, le sujet de type de type humain (?) et le complément d'objet une Que-phrase pronominalisée en *le*. Cette analyse cache complètement que la phrase est à interpréter comme un seul bloc. On doit donc stocker (1) explicitement dans *DELSYN*.

A côté de phrases complètement figées telles que (1) ou le proverbe :

(3) *Tant va la cruche à l'eau qu'elle se casse*

figurent des phrases telles que :

(4) *Dieu sait si cela a été difficile*
(5) *Luc prend la poudre d'escampette*

de structures :

(6) *Dieu sait si P*
(7) *Nhum*$_0$ *prendre la poudre d'escampette*

(6) explicite le fait que pratiquement n'importe quelle phrase peut apparaître en position P. (7) montre que si la partie *prendre la poudre d'escampette* est figée (invariante), le sujet peut être n'importe quel substantif de type humain. Par ailleurs, la phrase (7) (mais pas (6)) peut se conjuguer à un temps composé et/ou peut se voir appliquer une négation, il faut donc décrire cette variabilité avec la même précision que pour les phrases libres.

Un lexique-grammaire de ces structures a été construit par M. Gross 1989a de la même manière que pour les phrases libres. Les expressions figées sont regroupées par table, dans chaque table la liste des sous-structures et des transformations de chaque expression est donnée. La transformation de ces tables en liste de structures de *DELSYN* n'est cependant pas aussi simple que pour les phrases libres, nous allons en 4.10.1 reprendre le procédé qui permet cette génération.

4.10.1. Transformation des tables en liste de structures

Considérons l'expression *jouer une vilaine farce à quelqu'un* (<C1IPN-445>)[12] que l'on trouve dans :

(8) *Luc joue une vilaine farce à Léa*
(9) *Luc lui joue une vilaine farce*
(10) *Une vilaine farce a été jouée à Léa*
(11) *Une vilaine farce a été jouée par Luc à Léa*

qui s'analysent respectivement dans les structures

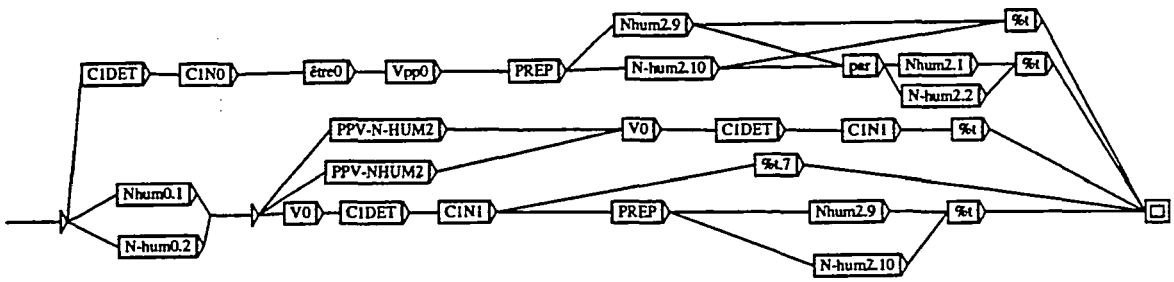
(12) $Nhum_0$ *jouer une vilaine farce à* $Nhum_2$
(13) $Nhum_0$ *lui-ppv*$Nhum_2$ *jouer une vilaine farce*
(14) *Une vilaine farce être jouée à* $Nhum_{21}$
(15) *Une vilaine farce être jouée par* $Nhum_1$ *à* $Nhum_2$

Cette expression est décrite dans la table C1IPN des phrases à sujet libres, à premier argument figé avec le verbe et à second argument libre. De plus, dans un grand nombre de cas, par exemple dans :

(16) *Luc a jeté un regard qui en disait long à Léa* *(<C1IPN-423>)*

l'argument figé peut être modifié ici par la concordance des temps. Comme pour les phrases libres on peut donner un automate de référence qui donne l'ensemble des structures: automate de la figure 4.10a.

[12]Nous rappelons qu'un verbe tel que *voler* n'a pas d'existence en tant que tel dans DELSYN. On distingue les structures de *voler-0* intransitif (*l'oiseau vole*) des structures de *voler-1* transitif (*des diamants ont été volés*). L'identité d'un verbe est ici sa forme canonique plus un numéro d'emploi, une autre manière de noter sans ambiguïté l'identité d'un verbe, ou ici d'une expression figée consiste à donner le nom de la table dans lequel il est décrit et son numéro d'ordre dans cette table. Ici <C1IPN-445> signifie que l'expression est la la 445eme de la table C1IPN de M. Gross 1989a. On pourra également se reporter à P. Vasseux 1979 pour une présentation plus précise de ces problèmes d'identité des entrées des tables syntaxiques.



Automate de référence de la table C1IPN

Figure 4.10a

Cependant nous avons fait figurer des transitions telles que C1DET (le déterminant de l'argument figé) dont l'interprétation dépend du mot donné dans la colonne 5 (voir l'extrait de la table de la figure 4.10b.

N0 N-hum	N0 N-hu		N1 N0pc			N0 V N1		N2 Nhum	N2 N-hu	Prep Nhum	Prep N-hum	Sur N2=dc	Passif	
+	-	établir	-	une	tête de pont	+	Loc	-	-	-	-	-	+	277
-	+	éveiller	-	de vieux	demons	+	en	-	-	-	-	-	-	279
-	+	éveiller	-	des vieux	demons	+	en	-	-	-	-	-	-	278
+	+	éveiller	-	un	echo	+	en	-	-	-	-	-	+	282
-	+	éveiller	-	un	soupcon	-	Loc	-	-	-	-	-	-	280
-	-	éveiller	-	un	souvenir	+	en	-	-	-	-	-	-	281
-	+	exercer	-	une	domination	+	sur	-	-	-	-	-	-	284
-	+	exercer	-	un	pouvoir	+	sur	-	-	-	-	-	+	283
-	-	exercer	-	une	pression	+	sur	-	-	-	-	-	+	286
+	+	exercer	-	une	surveillance	+	sur	-	-	-	-	-	+	285
+	-	exposer	-	un	cas	-	à	-	-	-	-	-	+	287
+	+	faire	-	un certain	accueil	-	à	-	-	-	-	-	+	494
+	-	faire	-	des	affaires	+	avec	-	-	-	-	-	+	293

Extrait de la table C1IPN de Maurice Gross 1989

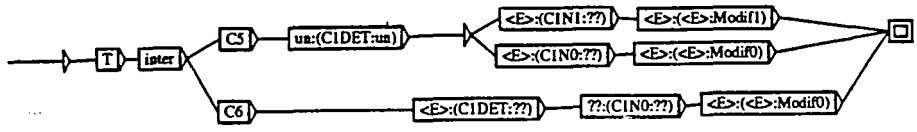
Figure 4.10b

En fait, si le déterminant *UN* figure dans la colonne 5, on veut appliquer à l'automate du verbe, le transducteur de la figure 4.10c (le déterminant UN =: *un+une+des* signale la liberté d'apparition des séquences Det-Modif pour le nom figé).

Transducteur à appliquer à l'automate si *UN* est le déterminant de l'argument figé

Figure 4.10c

De manière plus précise, on veut associer à C1DET C1N1 des séquences qui dépendent à la fois de la valeur de la colonne C_5 et de la colonne C_6. Cette précision est donné par le graphe de la figure 4.10d.



Transducteur qui détermine la valeur des séquences C1DET C1N1 et C1DET C1N0

Figure 4.10d

Ce graphe énonce le procédé à suivre pour obtenir l'automate d'un verbe :

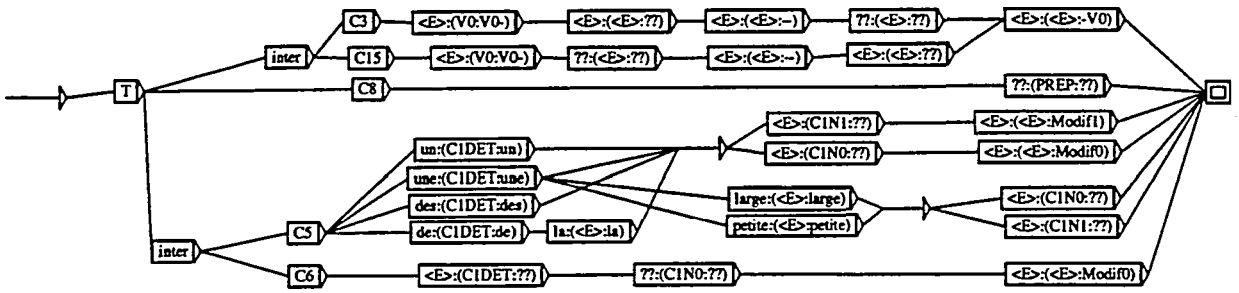
a. on applique à la colonne C_5 le transducteur pointé par C_5, il associe ainsi à *UN* le transducteur T_1 4.10c.

b. on applique au nom de la colonne C_6 le transducteur pointé par C_6 dans 4.10e, le résultat est le transducteurT_2,

c. on fait l'intersection de T_1 avec T_2 (comme spécifié par l'étiquette *inter* de 4.10e),

d. on applique le résultat à l'automate complet du verbe.

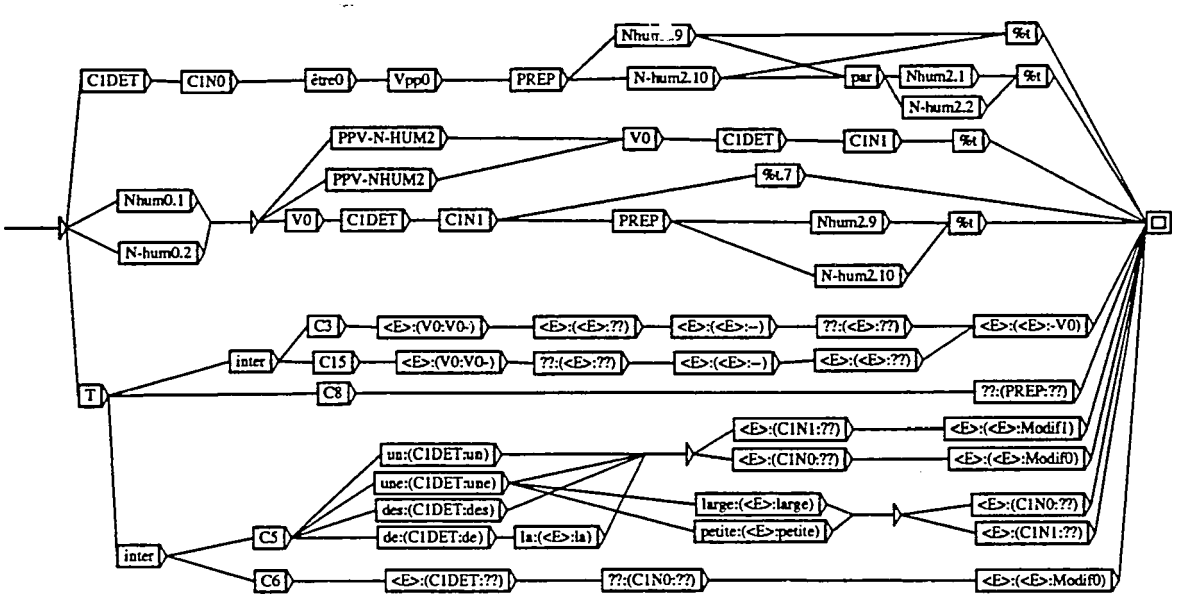
De manière plus complète, le transducteur de la figure 4.10e rend compte des emplacements où les éléments lexicaux des colonnes devront être insérés.



Procédé d'interprétation des colonnes de C1IPN pour l'automate de référence

Figure 4.10e

Tous ces éléments peuvent être réunis dans un seul graphe qui constituera le graphe (ou l'automate) de référence de la table. Ce graphe permet alors, comme pour les verbes libres, de générer l'ensemble des structures de cette table. Pour la table C1IPN, on obtient donc finalement le graphe de la figure 4.10f.

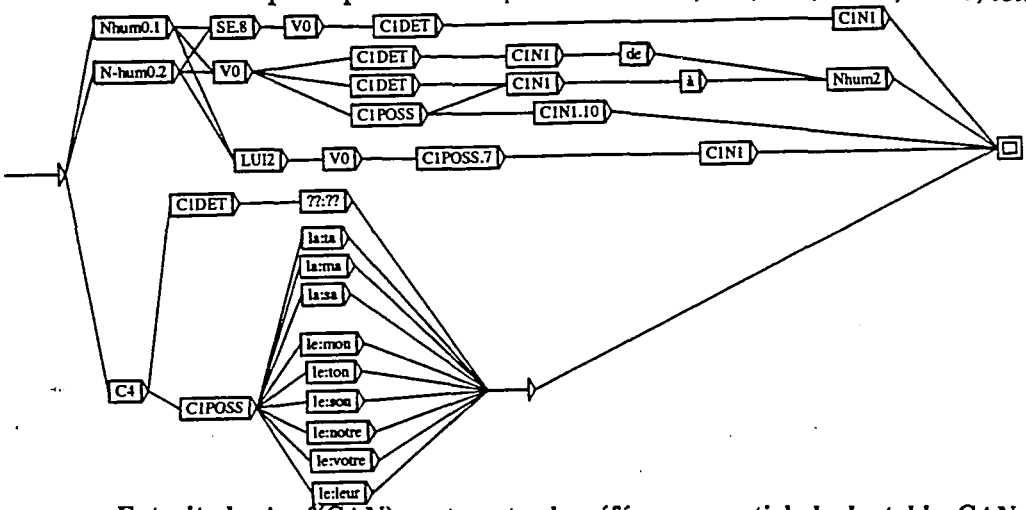


Aref(C1IPN) **Automate de référence de la table C1IPN**

Figure 4.10f

Ce procédé permet de résoudre d'autres problèmes: prenons la table CAN où le premier argument figé peut s'employer avec un possessif coréférent au sujet (*il perd sa tête*). Avant de traiter le problème de la coréférence, il nous faut rendre compte de l'accord possessif-N_1. Cela est possible grâce au transducteur associé à C_4 (colonne dans laquelle est écrit le déterminant de

N_1) de la figure 4.10e, on y voit que si le est le déterminant de *C1N1* indiqué dans C_4 alors *C1POSS* devra être remplacé par l'un des possessifs *mon, ton, son, notre, votre, leur.*



Extrait de Aref(CAN), automate de référence partiel de la table CAN

Figure 4.10g

4.10.2. Différents problèmes d'analyses des phrases figées

Nous sommes maintenant capable de convertir les données disponibles, c'est à dire les tables syntaxiques, en une liste de structures qui viendra compléter *DELSYN*. Cependant, un certain nombre d'expressions posent des problèmes nouveaux. *Prendre ses jambes à son coup* exige par exemple que soit géré l'accord entre le sujet de la phrase et chacun des possessifs *ses* et *son*. En effet, il faut distinguer une phrase avec un sujet singulier tel que

(17) *Il prend ses jambes à son cou*

d'une phrase avec sujet pluriel telle que

(18) *Ils ont tous pris leur jambes à leur cou.*

Nous noterons

(18) N_0 V_0-<*prendre-7*> SES_0 *jambes à* SON_0 *cou*

la structure de cette expression.

Par ailleurs la plupart des opérations, telles que la négation et le pronominalisation des arguments que nous avons appliqué aux structures de phrases simples sont à réexaminer ici.

4.11. PHRASES A VERBE SUPPORT DE NOM

Il semble que certains verbes puissent difficilement subir le même traitement que des verbes tels que *agacer* dont les structures sont entièrement décrites dans une table. Prenons le verbe *faire*, si on regarde la phrase :

(1) *Luc fait une promenade*

il semblerait qu'il faille ajouter :

(2) N_0 *V0-<faire>* N_1

à notre liste de structures. Par ailleurs une phrase telle que :

(3) *Luc fait une blague à Max*

semble de même imposer la structure :

(4) N_0 V_0*-<faire>* N_1 *à* $Nhum_2$

Cependant la phrase :

(5) * *Luc fait une promenade à Max*

est interdite alors qu'elle respecte bien la structure (4). L'intuition est alors que la structure des phrases (1) et (3) est contrainte respectivement par *faire une promenade* et *faire une blague*. Une étude systématique de ce type de phrase (J. Giry-Schneider 1987) montre qu'il existe un grand nombre de noms qui, apparaissant à droite de *faire*, contraignent la structure de la phrase. Les propriétés de la phrase venant du nom, et non du verbe, il est préférable de décrire les structures de phrases nom par nom plutôt que d'essayer de rattacher ces structures au verbe. Le verbe, *faire* dans nos exemples, n'est présent ici que pour supporter les marques de temps et de nombre, cela apparaît clairement dans la phrase :

(6) *La blague de Luc à Max était de mauvais goût*

où le verbe *faire* a disparu mais où la phrase sous-jacente (3) est encore perçue très clairement. Ce verbe est alors appelé *verbe support* (ou verbe support de nom) et le nom est appelé nom prédicatif. Les structures que nous voulons ajouter à DELSYN pour rendre compte (1) et (3) seront donc :

(7) $Nhum_0$ *Vsup-<faire-1> une Npred-<promenade-0>*[13]
(8) $Nhum_0$ *Vsup-<faire-1> une Npred-<blague-0>* *à* $Nhum_1$

Cette classification est parfaitement naturelle du fait que ces phrases sont souvent reliées par une relation de nominalisation avec des phrases sans verbe support. La phrase (1) est par exemple équivalente à :

(9) *Luc se promène*

que l'on analysera selon la structure :

(10) $Nhum_0$ *se* V_0*-<promener-0>*

---

[13] *Vsup* et *Nprédicatif* signifient respectivement verbe support et nom prédicatif.

Précisons également que *faire* peut être un verbe support comme dans les exemples que nous venons de donner mais qu'il a aussi d'autres emplois. La phrase :

(11) *Luc fait une tarte aux pommes*

comporte un autre verbe *faire*. Nous examinerons également plus loin l'analyse de phrases telles que

(12) *La situation fait craindre le pire*

4.11.1. Verbe support *faire*

Les structures des phrases avec verbe support *faire* sont décrites dans des tables (FN, FNN FNA, .. J. Giry-Schneider 1987) que nous allons convertir en listes explicites de structures de manière similaire aux phrases libres et figées. Ces phrases posent cependant des problèmes particuliers.

Remarquons tout d'abord que comme pour les phrases libre, il convient de spécifier la nature du sujet, essentiellement s'il est humain ou non-humain. La grande majorité des structures admet certes des $Nhum_0$ en position sujet ((1) et (3) par exemple), mais certaines phrases, telles que :

(13) *Le cinéma ne fait plus recette*

n'admettent que des substantifs non humains. On distinguera, d'après les tables, les structures :

(14) $Nhum_0$ *faire det N W*
(15) *N-*hum_0 *faire det N W*

Les exemples mentionnés ont tous des structures relativement simples, mais comme pour les verbes simples, des infinitives et des complétives peuvent apparaître comme arguments, soit comme sujet:

(16) *Que cela soit arrivé a fait l'étonnement de tous*

soit comme second argument comme dans

(17) *Luc a fait l'annonce que ils peuvent commencer*

Ces structures ne posent pas de difficultés particulières de représentation, par rapport à l'exemple de la table 4, on notera ainsi (16) et (17) respectivement par :

(18) $QuP\ Vsup_0$-*<faire-0> le Npréd-<étonnement-0> de* $Nhum_1$
(19) $Nhum_0\ Vsup_0$-*<faire-0> le Npréd-<annonce-0>* QuP_1

Cependant, ces phrases font très souvent intervenir des relations d'argument entre le nom prédicatif et un argument de la phrase, ces relations permettent d'ailleurs de distinguer les

emplois libres de *faire* des emplois à verbe support (Cf encore J. Giry-Schneider 1987). Ceci se voit dans la phrase :

(20) *Luc fait un compliment à Léa*

où *compliment* est le compliment de Luc et non de Léa d'autre comme le montre l'interdiction de la phrase

(21) **Luc fait le compliment de Léa*

La relation peut aussi opérer entre le nom prédicatif et un argument complément comme le montre l'emploi du possessif dans :

(22) *Luc fait ses devoirs à Max*

Cette relation de coréférence rend l'emploi d'infinitives telles que *de partir* dans :

(23) *Luc fait l'erreur de partir*

très fréquentes (comme le montre les colonnes marquées N_0 *faire à* N_1 *le N de Comp* dans FNANN ou de V_0 *Comp* dans FNAN). Le sujet de l'infinitive est en effet également l'argument partageant une relation avec le nom prédicatif (*Nhum*$_0$ dans notre exemple). Il conviendra donc aussi de noter dans *DELSYN* des structures telles que :

(24) *Nhum*$_0$ *Vsup*$_0$-*<faire-0> le Npred-<erreur-0> de* $V_1{}^0W$

dans lesquelles le 1 de $V_1{}^0W$ indique qu'il s'agit d'un autre verbe que le verbe principal et le 0 que le sujet de ce verbe à l'infinitif est le sujet de la principale (soit *Nhum*$_0$). L'analyse de ces phrases suivra donc également les étapes de l'analyse de la phrase :

(25) *Luc parle de partir*

Par conséquent le transducteur *f_DELSYN* associera à (23) :

(26) $(Luc)_{Nhum0}$ $(fait)_{Vsup}$ *la* $(erreur)_{Npred}$ *de* ([P] (*Luc*) *partir* [P])$_{V1{}^0W}$

dans laquelle la sous séquence *[P] (Luc) partir [P]* doit être analysée comme une phrase.

Si les propriétés précédentes rapprochent l'analyse des phrases à verbe support de l'analyse des phrases libres, le fait d'avoir un nom comme prédicat de la phrase introduit de nouveaux paramètres. En effet ces noms doivent être introduits par des déterminants (éventuellement vides) et peuvent éventuellement être modifiés. Si dans une phrase telle que :

(27) *Luc fait le con*

le déterminant *le* est figé, dans l'exemple suivant au contraire.

(28) *Luc a fait banqueroute* (*FN*)

le nom prédicatif *banqueroute*, décrit dans la table *FN*, est précédé d'un déterminant vide

inhabituel dans les formes libres, d'un déterminant indéfini avec modifieur portant sur le nom ou d'un déterminant défini également avec modifieur comme le montre :

(29) *Luc a fait une banqueroute inattendue*
(30) *Luc a fait la banqueroute qu'on lui prédisait*

Il convient donc de décrire explicitement chacune des structures. Pour les phrases (28) à (30) nous pouvons tout d'abord proposer respectivement les structures suivantes :

(31) $Nhum_0$ $Vsup_0$-*<faire-0> Npred-<banqueroute-0>*
(32) $Nhum_0$ $Vsup_0$-*<faire-0> une Npred-<banqueroute-0> Modif*
(33) $Nhum_0$ $Vsup_0$-*<faire-0> la Npred-<banqueroute-0> Rel*

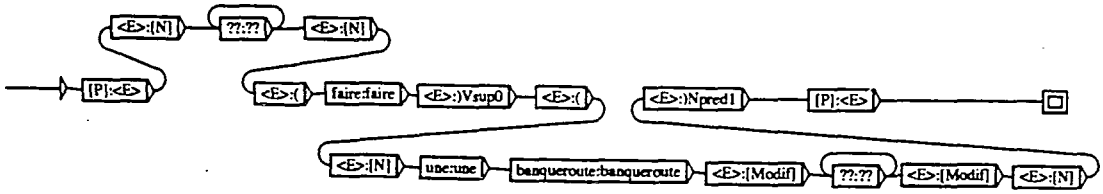
où *Modif* signifie modifieur (un adjectif ou une relative par exemple) et *Rel* relative. Si l'analyse de (27) par comparaison avec (31) ne pose pas de problèmes particulier, la comparaison de (30) avec (33) est plus complexe. L'application de la transduction *f_DELSYN* doit en effet détecter le fait que *banqueroute* est le nom prédicatif et que *la banqueroute qu'on lui prédisait* s'analyse comme une groupe nominal. La première passe de l'analyse de (30), c'est à dire la première application de la transduction *f_DELSYN*, donne donc :

(34) $(Luc)_{Nhum0}$ $(a\ fait)_{Vsup}$ ([N] $(la)_{det}$ $(banqueroute)_{Npred}$ $(qu'on\ lui\ prédisait)_{Rel}$ [N])

Nous verrons plus bas comment la transduction, lors d'une seconde passe analyse

(35) [N] $(la)_{det}$ $(banqueroute)_{Npred}$ $(qu'on\ lui\ prédisait)_{Rel}$ [N]

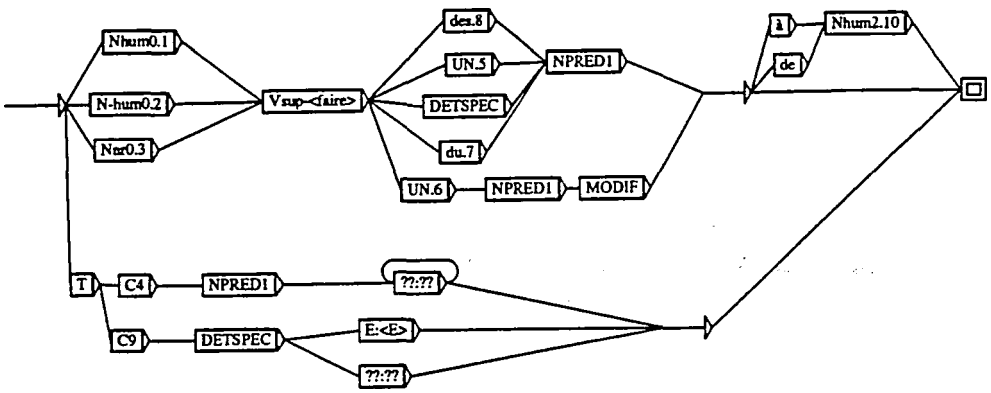
Le transduction associée à (33) sera aussi une sous transduction *f_DELSYN*.



Transduction construite à partir de la structure (32)

Figure 4.11a

Finalement, il convient de donner une structure différente pour chaque type de déterminant. Pour avoir cette liste de manière exhaustive nous avons procédés comme pour les verbes simples : nous avons écrit un automate de référence pour chaque table, d'où on déduit l'automate particulier de chaque forme. Ainsi l'automate de référence de la table FN est-il celui de la figure 4.11b. Nous avons repris les mêmes conventions que pour les phrases figées pour traduire des valeurs telles que $Poss_0$ (possessif coréférent au sujet) en des mots explicites.



Automate de référence de la table FN de J. Giry-Schneider 1987

Figure 4.11b

Comme pour les verbes libres, il faut appliquer successivement aux automates dérivés de 4.11b les transductions qui donnent les structures aux temps composés, les négations et également les formes interrogatives. La description des formes interrogatives doit être modifiée puisque si le dialogue

(36) *Que fait Paul ?*
(37) *Une mousse au chocolat.*

est parfaitement correct, (le verbe faire n'est ici pas le verbe support), le couple suivant

(38) **Que fait Paul sur l'état des lieux?*
(39) **Une remarque d'ordre général.*

n'est pas acceptable au contraire de la phrase en principe source :

(40) *Paul fait une remarque d'ordre général sur l'état des lieux.*

Cette propriété met en évidence que le nom prédicatif, même s'il a superficiellement la forme d'un groupe nominal général, se comporte de manière particulière, du moins pour certaines transformations. La pronominalisation du nom prédicatif, quand elle s'applique comme pour n'importe quel nom, pose paradoxalement un problème particulier. Ainsi le nom *affaire* de :

(40) *Luc fait des affaires profitables*

peut se pronominaliser en (41) ou (42) :

(41) *Luc en fait des profitables.*
(42) *Des affaires profitables, Luc en fait régulièrement.*

A priori des structures telles que

(43) *$Nhum_0$ en Vsup-<faire-0> des Modif1*
(44) *$Nhum_0$ en Vsup-<faire-0>*

rendent bien compte du phénomène comme pour les phrases libres. Cependant se pose ici le problème de l'utilité du résultat de l'analyse. Jusqu'à présent, on associe à chaque niveau d'analyse une structure de *DELSYN* qui donne, non seulement la structure, mais également l'identité du prédicat (*<agacer-0>* dans le premier exemple traité dans cette étude). Seul ce résultat permet d'envisager l'application de transformations ou une quelconque analyse sémantique ultérieure. Or dans le cas de (43) ou (44) le prédicat qui permet d'identifier les transformations appliquées ou que l'on peut encore appliquer, c'est à dire le nom prédicatif *Npred-<affaires-0>* a disparu. Il est ainsi impossible de déterminer quelle est la table qui a généré la structure. Il convient donc de noter sur la particule préverbale le nom prédicatif dont elles sont issues. Ce sera donc les structures (45) et (46) qui seront inclues dans *DELSYN*.

(45) *$Nhum_0$ en-(Npred-<affaire-0>) Vsup-<faire-0> des Modif*
(46) *$Nhum_0$ en-(Npred-<affaire-0>) Vsup-<faire-0>*

Bien sûr l'analyse de la phrase (41) sera très ambiguë puisqu'elle se verra associée aussi bien (45) que (47)

(47) *$Nhum_0$ en-(Npred-<gaffes-0>) Vsup-<faire-0> des Modif*

et avec elles toutes les pronominalisations possibles des autres nom prédicatifs. On aboutit donc à une analyse où l'ambiguïté peut dépasser la centaine d'analyses mais n'est-ce pas l'objectif de cette phase de l'analyse : proposer l'ensemble exact des analyses syntaxiquement possibles à un éventuel module de traitement ultérieur.

Par ailleurs, comme pour certaines phrases figées, la description précise des structures permises permet de lever des problèmes de coréférence sur le possessif. Par exemple l'équivalence :

(48) *Luc fait les courses de Marie*
=(49) *Luc fait ses courses à Marie*

inclus le *ses* de (49) comme coréférent à *Marie*. On peut donc, comme pour les expressions figées noter la structure correspondante à (49) :

(50) *$Nhum_0$ Vsup-<faire-0> SES_2 Npred-<courses-1> à $Nhum_2$*

qu'une transformation reliera à (51) et (52) :

(51) *$Nhum_0$ Vsup-<faire-0> les Npred-<courses-1> de $Nhum_2$*
(52) *$Nhum_0$ Vsup-<faire-0> les Npred-<courses-1> pour $Nhum_2$*

La structure (52), qui correspond par exemple à la phrase suivante:

(53) *Luc fait les courses pour Marie*

permet de mettre en évidence un problème nouveaux dans le processus d'analyse, problème que nous retrouverons fréquemment. Le groupe *pour Marie* est en général analysé comme adverbe (ou groupe adverbial ou complément circonstanciel). Nous n'aborderons que plus tard le problème de l'analyse des phrases avec adverbe mais il est possible d'en discuter certains aspects ici.

L'adverbe est en général analysé comme un argument inessentiel du verbe, c'est à dire qu'il pourrait apparaître accollé à n'importe quelle phrase simple. Cette attitude impliquerait l'inutilité d'inclure la description des adverbes dans la description de chaque phrase simple. Or la relation entre (53) et (48) = (49) est impossible à décrire si l'on suit cette démarche. Par conséquent, la structure (54) de (53) :

(54) *$Nhum_0$ $Vsup_0$-<faire-0> les $Npred_1$-<courses-1> pour $Nhum_2$*

notera explicitement l'adverbe. La position adoptée consiste donc à noter les adverbes d'une phrase quand ceux-ci sont reliés à une autre structure ((50) et (51) ici) par une transformation connue. Ce choix conduit alors à étudier des phrases comme (54) et (55)

(54) *Luc fait une toilette matinale rapide (FNA)*
(55) *Luc fait rapidement sa toilette matinale*

où l'adverbe *rapidement*, qui porte sur la phrase, est en relation avec un modifieur du nom prédicatif dans (54). Ce phénomène, appelé descente de l'adverbe par Z. Harris, est d'ailleurs un des critères de reconnaissance des phrases à verbe support. La description de (54) étant une structure qui intègre le modifieur, il convient donc que la structure (56) de (55) tienne compte de l'adverbe. De plus, comme la phrase (55) accepte d'autres adverbes (*hier* par exemple) non spécifiques, nous marquerons la particularité de cet adverbe par le fait qu'il provient d'un adjectif modifiant $Npred_1$:

(56) *$Nhum_0$ $Vsup_0$-<faire-0> Adv-Adj1 une $Npred_1$-<toilette matinale-0>*

Il faut de plus décrire l'ensemble des positions possibles de cet adverbe, ce qui conduit aux formes :

(57) *$Nhum_0$ $Vsup_0$-<faire-0> une $Npred_1$-<toillette matinale-0> $Adv\text{-}Adj_1$*
(58) *$Adv\text{-}Adj_1$, Nhum $Vsup_0$-<faire-0> une $Npred_1$-<toilette matinale-0>*
(59) *$Nhum_0$, $Adv\text{-}Adj_1$, $Vsup_0$-<faire-0> une N_1pred-<toilette matinale-0>*

Pour que la structure (56) permette d'analyser (55) il reste à être capable de reconnaître *rapidement* comme relié à l'adjectif *rapide*. Il faut donc avoir la liste exacte de ces couples (adverbes-adjectifs). A notre connaissance, cette description n'est pas encore disponible. Il existe cependant un moyen d'approcher de manière précise l'analyse de phrases telles que (55). Rappelons que le transducteur présenté au paragraphe 3.4 sur la morphologie dérivationnelle

permet l'analyse de mots construits par suffixations de mots connus. Par exemple, le suffixation en *ment* peut donner des adverbes à partir d'adjectifs. Par conséquent, comme *rapide,Adj* fait parti du dictionnaire initial *DELAS*, la composition *rapide,A*ement,Adv* génère la forme *rapidement,Adv*. Par ailleurs l'adverbe *rapidement* figure dans le *DELAS*, par conséquent l'analyse de *rapidement* par le transducteur morphologique donne

rapidement,Adv
*rapide,A*Ement,Adv*

La solution consiste donc à considérer que toutes les formes ayant cette double analyse peuvent apparaître en position *Adv-Adj1*. Pour ce faire il suffit d'appliquer le transducteur trans.morpho sur l'ensemble des adverbes du DELAS.

Notons par ailleurs que (54) est équivalente à :

(60) *Luc accomplit sa toilette matinale*

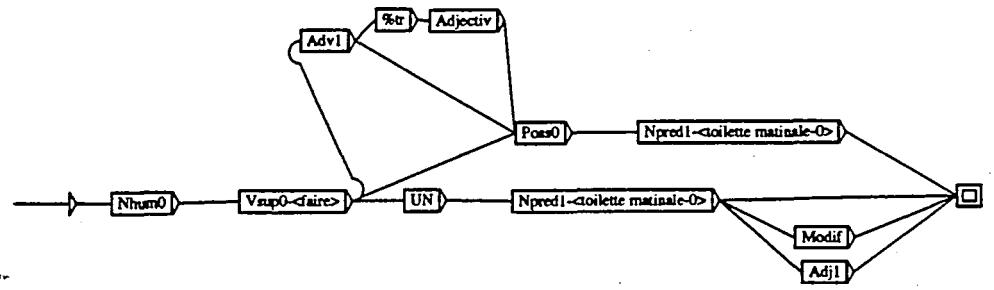
dans laquelle le verbe support a été remplacé par une de ses variantes. Un des problèmes particulier rencontré dans la représentation de ces phrases est le nombre important de verbes supports équivalents à *faire*. Il faut que ces variantes soient notées explicitement car elles sont spécifiques au nom. Ainsi on a :

(61) *Luc (commet = faire) une erreur*

mais pas

(62) * *Luc (commet = fait) sa toilette matinale*

On trouvera dans M. Mohri 1993 des tables de description des variantes de verbes supports .A ce point l'automate représentant les structures de *toilette matinale* sera le suivant:



Automate des structure de faire sa toilette matinale(FNA)

Figure 4.11c

Nous n'avons pas encore abordé l'analyse des groupes nominaux complexes, mais dans des cas comme :

(63) *L'insulte de Jean à Max n'est pas passée inaperçue.*

l'étude précédente est à la fois indispensable et suffisante pour mettre en évidence la structure du groupe nominal.

4.13. PHRASES A VERBE OPERATEUR

4.13.1. Présentation

La phrase

(1) *Luc a mis Pierre en retard*

est a priori de structure superficielle :

(2) $Nhum_0$ Aux_0 *Vpp-<mettre-1>* $Nhum_1$ *en* N_2

Cependant, une telle structure n'explicite pas le fait que N_2 est relié à $Nhum_1$ de manière particulière : *retard* est relié à *Pierre* par la phrase :

(3) *Pierre est en retard*

et, en fait, (1) s'analyse classiquement comme l'application sur (3) d'un opérateur au sens de Z.Harris, ici *Luc a mis* qui, dans son application, provoque l'effacement du verbe support *être* de (3). Cet opérateur a une certaine généralité puisqu'il s'applique à un très grand nombre de phrases de structure :

(4) *$Nhum_0$ Vsup-<être-0> Prep Npréd W*

de phrases comme par exemple

(5) *Luc a mis Pierre en congé*
(6) *Luc a mis Pierre aux arrêts*
(7) *Luc a mis Pierre en colère*

ou de phrases avec une partie figée comme :

(7) *Luc a mis Pierre dans de beaux draps*

ou de phrases avec un argument supplémentaire introduit par la phrase initiale à verbe support :

(8) *Luc a mis Pierre au courant des dernières nouvelles*

où la phrase sous-jacente est

(9) *Pierre est au courant des dernières nouvelles*

de structure

(10) *$Nhum_0$ $Vsup_0$-<être-0> au $Npréd_1$-<courant-0> de N_2*

et où le prédicat <courant-0> admet aussi la structure plus générale

(11) *$Nhum_0$ $Vsup_0$-<être-0> au $Npréd_1$-<courant-0> de ce Qu P*

Cette dernière structure se retrouve dans la phrase :

(12) *Luc a mis Pierre au courant de ce que son ami doit venir*

qui montre bien que la structure globale de la phrase ne peut être déduite d'une simple description des structures superficielles de *mettre*.

Il existe deux méthodes, l'une implicite et l'autre totalement explicite, pour représenter les structures de ces phrases puis pour les analyser. Nous les discutons successivement dans les deux paragraphes qui suivent. Cette discussion provient d'un travail commun avec M. Mohri, on pourra se reporter à M. Mohri 1993 pour d'autres aspects de cette question.

4.13.2. Représentation *implicite*

La première solution, qui est celle que nous utilisons pour l'instant, consiste à reconstruire explicitement la phrase de structure (4) sur laquelle l'opérateur s'applique dans le processus même de l'analyse. Nous notons alors la structure de (1), nous pas (2) mais

(13) $Nhum_0$ Vop_0-<*mettre-0*> #$Nhum_{10}$ $Vsup_{10}$-<*être-0*> $Prep_{21}$ $Npred_{21}$ W_2# $Nhum_1$ $Prep_2$ $Npred_2$ W_2

qui est la structure qui sera entrée dans *DELSYN*. Dans (13), la phrase d'origine à verbe support est placée entre les séparateurs #, entre ces deux # nous notons également la renumérotation des arguments qui permet de se référer à une structure exactement identique à un élément de *DELSYN*. Cette représentation est appelée implicite car la phrase à verbe support est présente de manière implicite : on ne spécifie pas quel nom prédicatif $Npred_2$ elle comporte.

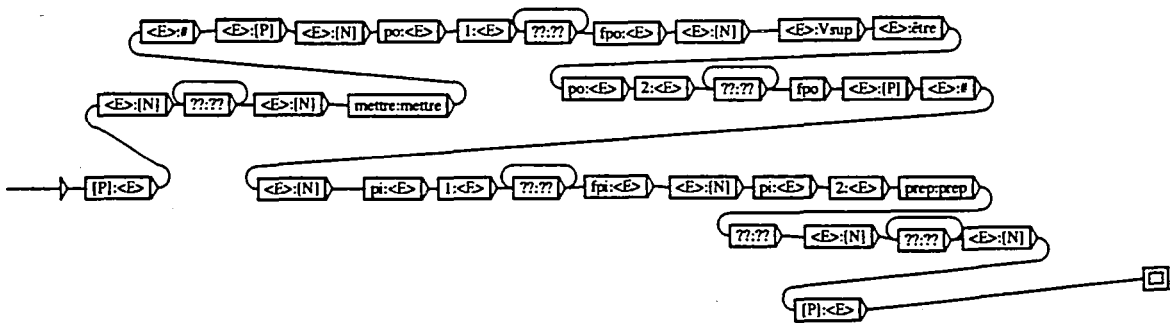
L'analyse de la phrase (1) consiste alors à

1. faire correspondre $Nhum_0$, $Nhum_1$, $Prep_2$ et $Npred_2$ avec les séquences correspondantes du texte,

2. recopier entre les séparateurs # $Nhum_1$ $Prep_2$ et $Npred_2$ et les renuméroter et

3. analyser la phrase entre # de manière indépendante.

Plus précisément, le transducteur qui effectue cette analyse est celui de la figure 4.13a :



Transducteur d'analyse équivalent à *(13)*

Figure 4.13a

Dans l'hypothèse où cette solution est retenue, le transducteur de la figure 4.13a est inclus dans f_DELSYN et l'analyse de (1) se fait de la manière suivante:
(1) est associée à :

(14) [N] *Luc* [N] *a mis* # [P] *Pierre Vup est prep en Npréd retard* [P] # *Pierre en retard*

puis l'analyse se poursuit sans changement, on vérifie que la séquence entre [P] et [P] est bien

une phrase répertoriée dans *DELSYN*.

4.13.3. Représentation explicite

L'avantage de la représentation précédente est que l'application de l'opérateur est visible et que l'on peut utiliser la description des phrases N_0 *être Prép* N_1 déjà faite ailleurs. Cependant, une telle analyse ne rend pas compte d'interdictions comme

(15) * *Luc met Pierre en avance*

qui suggère que l'opérateur, comme en fait tout opérateur, ne s'applique pas sur des structures abstraites du type *Nhum*$_0$ *Vsup Prep Npred W* dans lesquelles aucun élément lexical n'apparaît, mais plutôt sur un autre opérateur, sur une structure de phrase dans laquelle on spécifie le prédicat. Ainsi la structure à stocker n'est plus (14) mais :

(15) *Nhum*$_0$ *Vsup*$_0$-*<mettre-0>* # *Nhum*$_1$ *être en retard* # *Nhum*$_1$ *en Npred*$_2$-*<retard-0>*

Dès lors la phrase entre # n'a plus de raison d'être et (15) devient

(16) *Nhum*$_0$ *Vsup*$_0$-*<mettre-0>* *Nhum*$_1$ *en Npred-<retard-0>*

que nous appelons la représentation explicite (dans laquelle le nom prédicatif est explicitement mentionné). Dans le cas où nous choisissons cette représentation, c'est (16) qui devient un élément de *DELSYN*. Il devient alors plus cohérent, mais c'est purement terminologique, d'appeler le verbe verbe non pas verbe opérateur mais support comme dans les structures (15) et (16).

Notons que pour ce choix, *DELSYN* contient, en plus de (16) les structures :

(16) *Nhum*$_0$ *Vsup*$_0$-*<mettre-0>* *Nhum*$_1$ *en Npred-<retard-0>*
(17) *Nhum*$_0$ *Vsup*$_0$-*<mettre-0>* *Nhum*$_1$ *en Npred-<congé-0>*
(18) *Nhum*$_0$ *Vsup*$_0$-*<mettre-0>* *Nhum*$_1$ *en Npred-<colère-0>*
(19) *Nhum*$_0$ *Vsup*$_0$-*<mettre-0>* *Nhum*$_1$ *C*$_2$-*<dans de beaux draps-0>*
(20) *Nhum*$_0$ *Vsup*$_0$-*<mettre-0>* *Nhum*$_1$ *courant Npred-<courant-0> de ce Qu P*

mais pas

(21) **Nhum*$_0$ *Vsup*$_0$-*<mettre-0>* *Nhum*$_1$ *en Npred-<avance-0>*

ce qui rend compte de l'interdiction (15).

Cette notation permet également d'être plus cohérent avec la structure

(21) *Nhum*$_0$ *Vsup*$_0$-*<mettre-0>* *Nhum*$_1$ *en Npred-<garde-0>*

d'une phrase telle que

(22) *Luc met Max en garde*

qui ne peut pas s'analysée par la représentation implicite puisque la phrase

(23) **Max est en garde*

n'existe pas dans ce sens là et que, par conséquent, la structure :

(24) **Nhum$_0$ Vsup$_0$-<être-0> en Npred-<garde-0>*

n'est pas dans *DELSYN*.

Cette seconde solution, un peu plus coûteuse puisqu'elle exige le codage explicite d'une structure par nom prédicatif, est cependant cette qui doit être choisie à terme. La correction de la description, ce qui est la première exigence, nécessite cette précision. Par ailleurs cette seconde solution est également plus efficace puisqu'elle entraîne a priori un cycle d'analyse (d'application de *f_DELSYN*) en moins.

4.14. LES GROUPES NOMINAUX

A ce stade, les seuls groupes nominaux que nous sommes capable de traiter sont constitués d'une liste de noms propres et des séquences *Article Nom* où *Article* est un des article *le, la les un une des ce cette ces* et *Nom* est un nom simple ou composé (obtenu par consultation du DELACF), de plus il y a accord en genre et en nombre entre l'article et le nom. Nous allons tout d'abord étendre ces analyses aux groupes nominaux construit par relativation, par exemple :

le livre que j'ai acheté

Nous allons montrer comment utiliser la connaissance des phrases simples accessibles dans DELSYN pour décrire cette syntaxe. Nous montrerons ensuite que l'analyse des phrases à verbes supports faite dans les sections précédentes permet d'analyser des groupes nominaux tels que :

le cadeau de Jean à Marie

comme une réduction de :

le cadeau que fait Jean à Marie

qui s'analyse à son tour comme un groupe nominal avec relative.

Nous passerons ensuite à la description des déterminants complexes tels que *un grand nombre* dans le groupe nominal

un grand nombre de livres

4.15. PROPOSITIONS RELATIVES

Une phrase telle que

(1) *Luc donne un gâteau à Max*

donne lieu à des groupes nominaux de la forme

(2) *Luc qui a donné un gâteau à Max*
(3) *Max à qui Luc a donné un gâteau*
(4) *Un gâteau que Luc a donné à Max*
(5) *Un gâteau qu'a donné Luc à Max*

Si on veut analyser de tels groupes de manière cohérente avec nos autres analyses, il nous faut introduire dans DELSYN les structures :

(6) [N] $Nhum_0$ *qui* V_0-<*donner-0*> N_1 *à* $Nhum_2$
(7) [N] $Nhum_2$ *à qui* $Nhum_0$ V_0-<*donner-0*> N_1
(8) [N] N_1 *que* $Nhum_1$ V_0-<*donner*> *à* $Nhum_2$
(9) [N] N_1 *que* V_0-<*donner*> *Nhum1 à* $Nhum_2$

que nous marquons [N] pour spécifier qu'il s'agit de structures de nom et non pas de phrases.

On va procéder pour les relatives comme pour les temps composés ou les pronoms préverbaux, nous allons construire une transduction qui, appliquée aux structures de phrases de *DELSYN*, engendrera l'ensemble des structures possibles de relatives parmi lesquelles figurent les structures (6) à (9). Pour illustrer ce processus nous allons d'abord prendre la structure plus simple

(10) $Nhum_0$ V_0-<*agacer-0*> $Nhum_1$

qui ne donne lieu qu'à trois relativations :

(11) [N] $Nhum_0$ *qui* V_0-<*agacer-0*> $Nhum_1$
(12) [N] $Nhum_1$ *que* $Nhum_0$ V_0-<*agacer-0*>
(13) [N] $Nhum_1$ *que* V_0-<*agacer-0*> $Nhum_0$

La transduction de la figure 4.15a transforme bien (10) en (11),(12) et (13) :

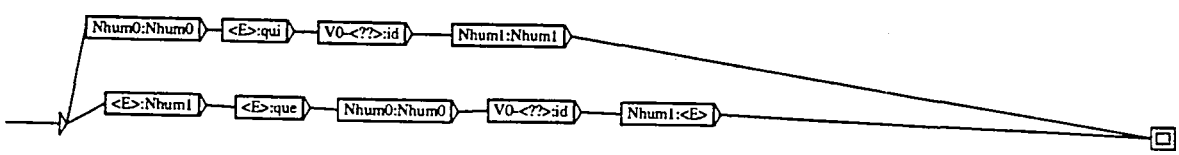


Figure 4.15a

Si on veut en plus tenir compte des temps composés il faut passer au transducteur de la figure

4.15b.

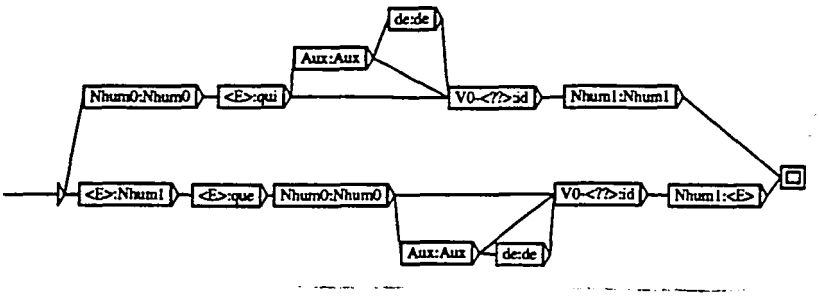


Figure 4.15b

On remarque qu'il n'est pas nécessaire d'ajuster la composition des auxiliaires, cette description est déjà faite dans *DELSYN* et le transducteur ne s'appliquera donc qu'aux structures correctes. Cette propriété de composition est encore plus visible dans le transducteur qui tient compte de l'introduction des négations de la figure 4.15c.

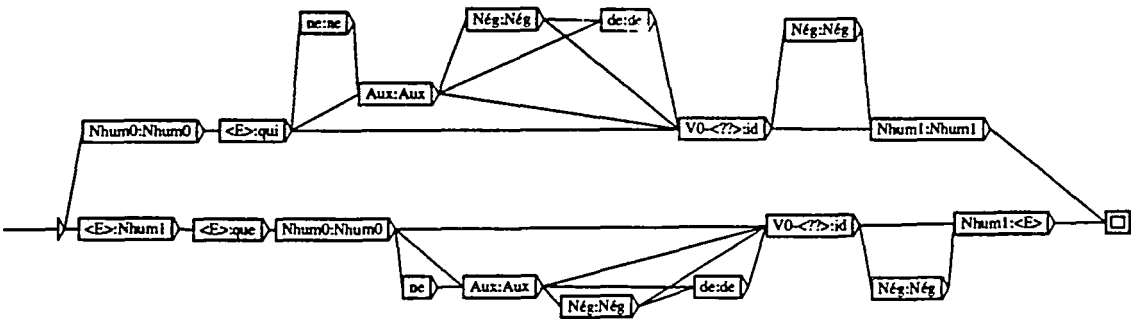


Figure 4.15c

Le négation est placée à tous les endroit possibles, ce qui permet dans le transducteur le chemin *Nhum*$_0$ *ne Aux Nég* V_0 *Nég Nhum*$_1$ où la négation apparaît deux fois (ce qui est inacceptable). En fait ce chemin n'apparaît pas dans le résultat de la transduction car il est de toute façon inexistant dans l'ensemble des structures de *DELSYN* (qui, nous le rappelons se doivent toutes d'être exactes quelle que soit l'étape de la construction de ce lexique).

4.17. DETERMINANTS

4.17.1. Présentation

Les paragraphes précédents ont permis l'analyse de certains groupes nominaux complexes, des constructions avec relatives telles que :

(1) *Le livre que Paul a acheté [est intéressant]*

ou de complément de nom comme :

(2) *La promenade de Luc [lui a fait du bien]*

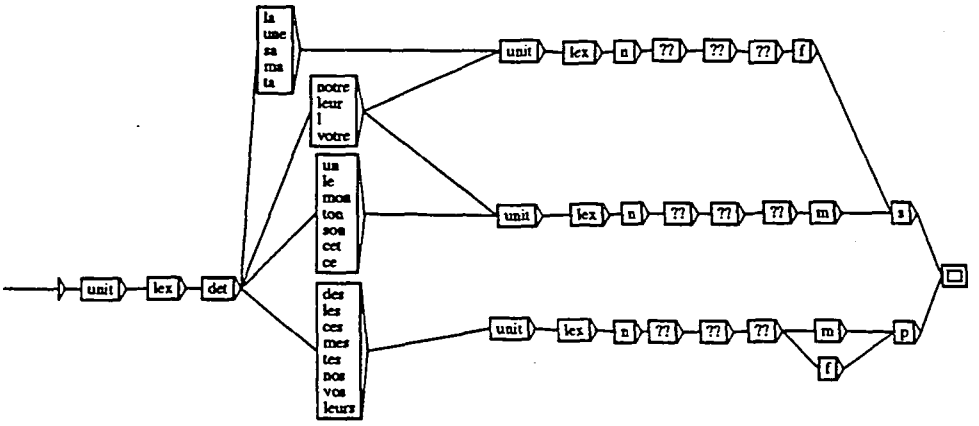
qui s'analysent comme des réductions de relatives et de verbe support: (2) étant équivalent à :

(3) *La promenade que fait Luc [lui fait du bien].*

Cependant, nous n'avons pas encore défini les formes que peut prendre un groupe nominal simple du type *un gâteau* dans :

(4) *Luc mange un gâteau*
(5) *Luc mange ce gâteau*
(6) *Luc mange un de ces gâteaux*

Commençons par rappeler que, parmi les formes les plus simples de groupes nominaux nous avons les formes *Dét N* où *Dét* est un déterminant défini ou indéfini. Les déterminants *Ddef* comportent essentiellement les articles définis *le la les l'*, les adjectifs démonstratifs *ce, cet, cette, ces* et les adjectifs possessifs *mon, ton, son, ma, ta, sa, mes, tes, ses, notre, votre, leur, nos, vos, leur* (nous reprenons la description de M. Gross 1985 sur laquelle nous nous appuyons dans ce paragraphe). La simple reconnaissance de ces groupes nominaux ne pose a priori pas de problème, il suffit de les représenter par l'automate :



Automate des groupes nominaux à déterminant défini simple

Figure 4.17a

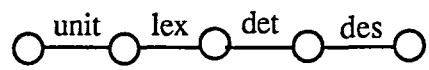
Cependant, pour reconnaître *le garçon* dans la phrase :

(5) *Pierre emprunte ce livre au garçon*

il faut reconnaître que *au* est une contraction de *à le*. La liste des principales contractions étant

(6) *à le* -> *au*
(7) *à les* -> *aux*
(8) *de le* -> *du*
(9) *de les* -> *des*

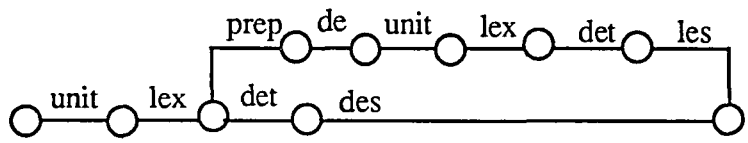
il faut disposer d'un mécanisme spécifique pour les reconnaître. Remarquons tout d'abord que la solution ne peut être de transformer systématiquement *du* en *de le*, *des* en *de les*, etc. puisque le déterminant indéfini *des* par exemple doit être conservé tel quel. La solution à ce problème est un prétraitement de la phrase au niveau de son analyse morphologique. Après analyse morphologique d'une séquence comme *des*, simplement représenté par l'automate suivant



Résultat de l'analyse morphologique de *des*

Figure 4.17b

nous ajoutons systématiquement la possibilité à *des* d'être une contraction de *de les* sans détruire la première interprétation, cela donne ainsi l'automate



Représentation d'une forme *des* trouvée dans un texte

Figure 4.17c

La suite de l'analyse n'est en rien affectée, simplement l'ambiguïté sera levée à un stade ultérieur de l'analyse.

Pour l'analyse de groupes nominaux à déterminants plus complexes, M. Silberztein 1993 a montré que l'utilisation d'automates permet la description de déterminants tels que *deux mille huit cent cinquante huit*. En effet en composant l'automate qui décrit l'ensemble des manières d'exprimer un nombre de 1 à 99 avec un automate qui décrit les manières de compter les centaines, on obtient l'automate qui permet de lister les nombres de 1 à 999. En itérant cette opération sur huit automates différents, on obtient alors les façons d'exprimer un nombre. Cet automate, dénommé *Dnum* définie de manière extensive l'ensemble des déterminants appelés numériques (notés *Dnum*).

La même méthode peut être employée pour certains autres phénomènes, ainsi on peut ainsi construire l'automate des fractions pour rendre compte d'une forme comme

(10) *les huit-cent cinquante trois vingt-cinquième de N*

Ces deux grammaires, même si elles sont extrêmement particulières, sont décrites d'une manière qui se généralise à d'autres déterminants. Remarquons que l'ensemble des déterminants *Dnum* peu être précédé par *aux environs de* ou *à peu près* (classés parmi les prédéterminants et notés *Pred* dans M. Gross 1977) comme :

(11) *Luc a acheté aux environs de trois livres*
(12) *Luc a acheté à peu près trois livres*

On voit donc qu'il faut enrichir l'automate des *Dnum* ou celui de la figure 4.17a. On devine cependant un risque, celui d'avoir à construire des automates tellement volumineux et tellement complexes qu'on ne pourra ni les stocker ni les vérifier. Trois stratégies sont possibles pour poursuivre ce type de traitement. La première consiste à construire quand même explicitement les automates, c'est à dire l'automate qui contient à la fois les *Dnum* et des séquences telles que *aux environs de Dnum*. La seconde méthode, comme elle est suggérée par exemple dans M. Silberztein 1993, consiste à travailler sur des automates récursifs (RTN : Recursive Transition Networks), l'automate n'est pas calculé explicitement : il contient des chemins tels que *aux environs de Dnum* où *Dnum* pointe lui même vers un autre automate qui décrit les séquences correspondantes. Nous avons opté pour une approche légèrement différente, que nous allons décrire maintenant et qui s'intègre bien dans le processus global d'analyse.

4.17.2. Analyse ascendante des déterminants

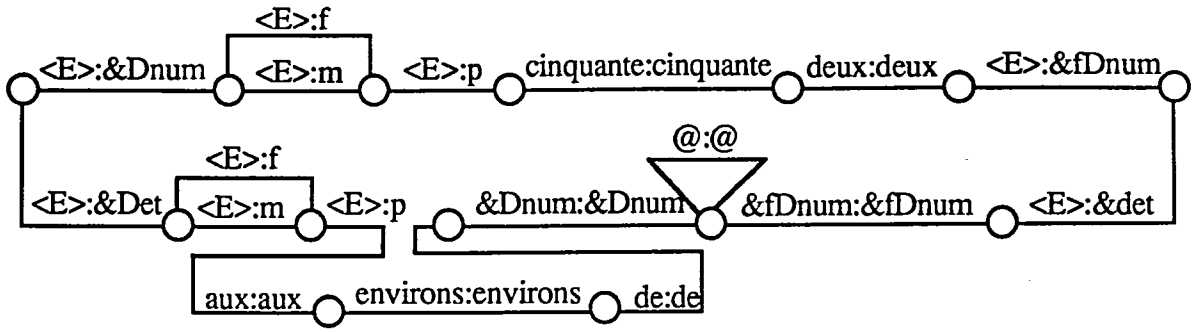
Soit la phrase :

(13) *Luc a acheté aux environs de cinquante deux titres*

Nous décrivons dans un unique transducteur les structures (14) et (15)

(14) *Dét* = *aux environ de Dnum*
(15) *Dnum* = *cinquante deux*

de la manière suivante



Extrait du transducteur T_1 des déterminant

Figure 4.17a

L'application du transducteur T_1 ci-dessus à (13) donne l'analyse partielle $T_1(13)$ = (16)

(16) *Luc a acheté aux environs de* &Dnum g p *cinquante deux* &fDnum *titres*

où *g* (genre) vaut *m* ou *f*. Dans (16) on a marqué l'emplacement de du déterminant numérique *cinquante deux*. Une seconde application de T_1 donne $T_1(16) = T_1^2(13)$ = (17)

(17) *Luc a acheté* &Dét g p *aux environs de* &Dnum g p *cinquante deux* &fDnum &fDet *titres*

dans laquelle le déterminant complet a été reconnu. Ce type d'analyse est bottom-up, c'est à dire du bas vers le haut. Certaines séquences sont reconnues, elles permettent à leur tour de marquer d'autres éléments plus complexes.

Remarquons que ce type d'analyse, de même que l'analyse générale, est effectué par un transducteur T_1 qui s'applique cycliquement. Mais rien n'empêche d'appliquer les deux transducteurs en parallèle, autrement dit d'appliquer l'union des deux transductions[14]. Nous calculons donc $T_1 \cup f\_DELSYN$ (transducteur général d'analyse) qui devient donc *f_DELSYN*. Nous sommes donc revenus à la méthode suivie depuis le début, c'est-à-dire l'application cyclique d'un seul transducteur *f_DELSYN* jusqu'à l'obtention d'un point fixe. La méthode est donc à la fois descendante et ascendante et elle s'arrête quand il y une jonction entre les deux.

Pour illustrer notre méthode, reprenons l'analyse de la phrase (13). Une première application de *f_DELSYN* associe à (13) dans un même temps une structure syntaxique globale et les marqueurs de détection du *Dnum*. Cela conduit (18) = *f_DELSYN*(13) (nous simplifions les parties non pertinentes à la discussion) :

(18) = *f_DELSYN*(13)
([N] *Luc* [N])$_{Nhum0}$ (*a acheté*)$_{V0}$ ([N] *aux environ de* &Dnum g p *cinquante deux* &fDnum unit n *titres* [N])$_{N1}$

Nous donnons sur la figure 4.17b la partie de *f_DELSYN* qui concerne l'analyse des groupes nominaux de la forme *Dét n*. Cette partie ne fait l'analyse complète du groupe nominal que si la

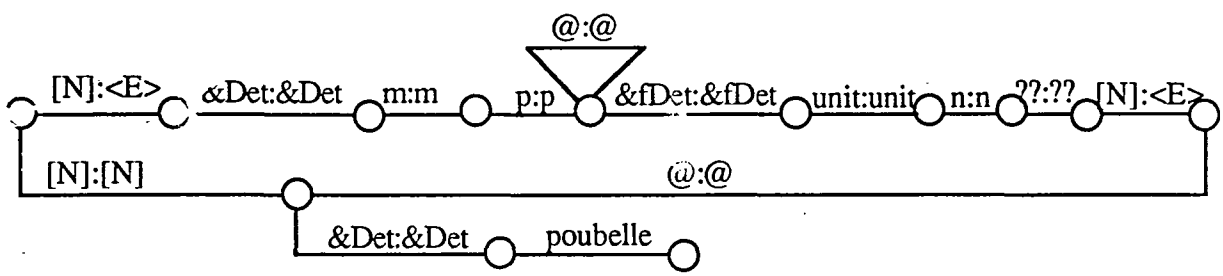
[14]Nous détaillons en 6 l'utilisation conjointes de deux types de transductions.

marque *&Dét* est déjà présente dans l'automate de la phrase (ce qui n'est pas encore le cas dans (18)). Dans le cas contraire, la fonction laisse la phrase invariante l'analyse reste à faire. Nous avons réuni dans le transducteur de la figure 4.17c le transducteur qui analyse (très partiellement) les déterminants et celui qui analyse les groupes nominaux. Ce transducteur est donc également la partie de *f_DELSYN* active dans l'analyse de notre exemple.

La deuxième application de *f_DELSYN* laisse identique le second groupe nominal : (transition *[N]:[N]* et @:@ de T_2, @:@ signifiant : les lettres en lecture sont toutes les lettres sauf celles marquées sur d'autres transitions, donc ici n'importe quel mot sauf *&Dét*. D'autre part l'output est égal à l'input). En revanche la partie de *f_DELSYN* correspondant à T_1 reconnaît le déterminant complet. On obtient donc

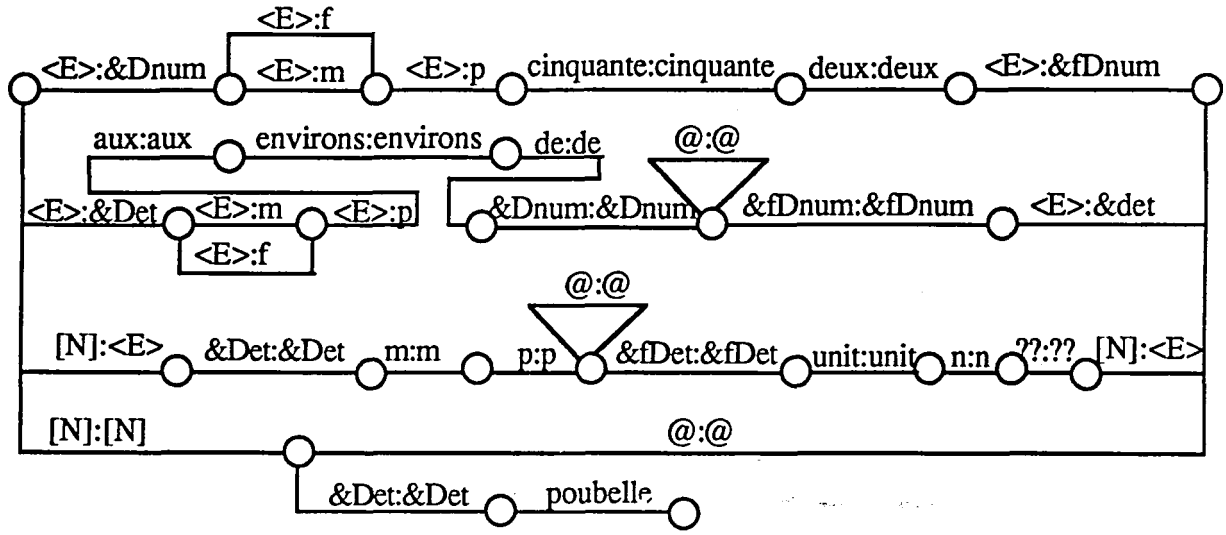
(19) $= f\_DELSYN^2(13)$

(*Luc*)$_{Nhum0}$ (*a acheté*)$_{V0}$ ([N] &Dét g p *aux environ de* &Dnum g p *cinquante deux* &fDnum &fDét unit n *titres*[N])$_{N1}$



Extrait du transducteur T_2 d'analyse des groupes nominaux

Figure 4.17b



Union des transducteurs 4.17a et 4.17b

Figure 4.17c

A ce stade le second groupe nominal est marqué *&Dét unit n*, il peut donc être analysé. Une nouvelle application de *f_DELSYN* donne alors :

$$f\_DELSYN^2(13) = (19)$$

$(Luc)_{Nhum0}$ $(a\ acheté)_{V0}$ (&Dét g p *aux environ de* &Dnum g p *cinquante deux* &fDnum &fDér unit n *titres*)$_{N1}$

qui est le résultat de l'analyse puisque le groupe nominal complet a été reconnu.

La description des prédéterminants, c'est-à-dire des séquences telles que *aux environs de* (décrites la table *Préd* de M. Gross 1977) se convertit ainsi partiellement en un automate que l'on incorpore à *DELSYN*.

Par ailleurs, les différents déterminants, tels qu'on les trouve dans les tables *Dadv*, *Dnom*, *Préd* et *Dadj*, se combinent entre eux de manière imprévisible comme on peut le voir dans les exemples

(20) *Beaucoup de livres*
(21) *Beaucoup plus de livres*
(22) *Beaucoup moins de livres*
(23) **Beaucoup peu de livres*

ou dans les exemples suivants, les déterminants *Dnom*, qui contiennent des noms :

(23) *un ensemble de livres*
(23) *une certaine quantité de livres*
(24) *une grosse quantité de livres*
(25) *un groupe d'amis*
(26) **un ensemble d'une certaine quantité de livres*

Une solution possible est de reprendre la technique employée pour les déterminants numériques et de répertorier dans des automates l'ensemble ce ces séquences. Bien sûr le problème est ici quantitativement et qualitativement très différent et nous allons voir en 4.17.3 que certains phénomènes, en particulier la mobilité des déterminants à l'intérieur de la phrase principale, ne peuvent être décrits de cette manière.

4.17.3. Problèmes de mobilité

Le problème de la combinatoire des déterminants, même s'il n'est pas résolu, n'est pas le principal problème que pose leur analyse. Certains d'entre eux ont une grande mobilité, à l'intérieur même du groupe nominal par exemple, comme entre :

(20) *J'ai acheté une grosse quantité de légumes*
(21) *J'ai acheté des légumes en grosse quantité*

où intervient la transformation *[Dnom p.]* (restructuration du déterminant nominal)

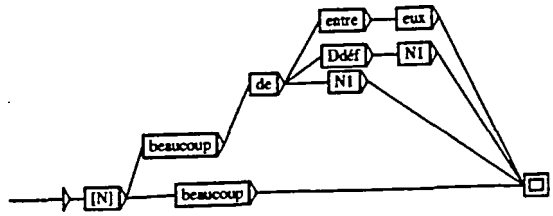
Dans la phrase :

(22) *Beaucoup de personnes viendront*

le groupe nominal sujet *beaucoup de personnes*, s'analyse comme le déterminant *beaucoup de* suivit du nom *personnes*. L'analyse de cette phrase ne pose a priori pas d'autres problèmes que ceux décrits en 4.17.2, cependant la pronominalisation de (22) en :

(23) *Beaucoup viendront*

montre qu'il faut également considérer la possibilité de pronominaliser le groupe nominal en ne conservant que le déterminant (bien entendue cette propriété est à étudier déterminant par déterminant). Notons qu'en grammaire traditionnelle on distingue *beaucoup* déterminant de *beaucoup* pronom comme s'il s'agissait de deux mots différents, la grande généralité du phénomène pour les *Dadv* conduit cependant à représenter les structures de (22) et (23) dans une même automate (voir figure 4.17d).



Deux structures de groupe nominal avec *Beaucoup*

Figure 4.17d

Un groupe nominal de la même forme, tel que *beaucoup de gâteaux* dans

(24) *Luc a mangé beaucoup de gâteaux*

posent de nouveaux problèmes qui rendent impossible la description des propriétés de *beaucoup* dans un automate tel que 4.17d ne décrivant que les structures d'un groupe nominal. En effet (24) est équivalent à :

(25) *Luc a beaucoup mangé de gâteaux*

Dans cette dernière phrase, le déterminant s'est détaché du groupe nominal pour apparaître entre l'auxiliaire et le participe passé, c'est d'ailleurs la seule place qu'il peut occuper. Cette propriété, très générale, est explicitement mentionnée dans les tables *Dadv, Dadj, Dnom* et *Préd* qui donnent non pas uniquement des structures de groupes nominaux mais bien des structures de phrases. Ainsi la possibilité pour *beaucoup* de se déplacer de cette manière est représentée par les deux structures :

(26) N_0 V_0 *beaucoup de N W*
(27) N_0 Aux_0 *beaucoup* V_0 *de N*

où, cette fois Aux_0 =:*avoir* + *être* comme dans

(28) *Luc s'est beaucoup acheté de gâteaux*

mais pas *venir de* ni *aller* comme le montre

(29) **Luc va beaucoup manger de gâteaux*
(30) **Luc vient de beaucoup manger de gâteaux*

Par ailleurs la pronominalisation de (24) donne

(31) *Luc en a mangé beaucoup*

qui est décrite par le fait que *beaucoup* admet la structure

(32) N_0 *en* V_0 *beaucoup*

Cette dernière phrase s'analyse en utilisant la description que nous avons déjà faite sur les pronominalisations d'arguments du verbe. La première application de *f_DELSYN* à (31) donne en effet :

(33) ([N] *Luc* [N]) *en-1* Aux_0 *a* $(mangé)_{Vpp0}$ ([N] *beaucoup # de en-1#* [N])

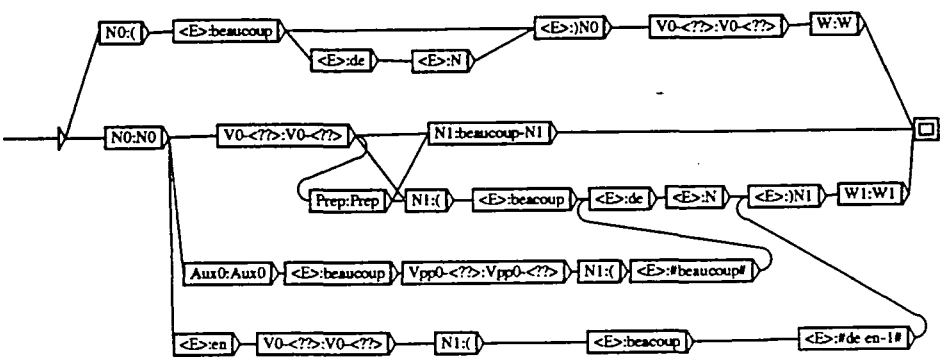
dans laquelle *[N] beaucoup # de en-1 # [N]* s'analyse comme groupe nominal.

Pour rendre compte de la structure (32) nous utilisons donc deux éléments:

a. une transduction qui associe (33) à (32),

b. une transduction qui reconnaît *beaucoup #de en-1#* comme un groupe nominal abstrait.

Finalement l'automate décrivant les comportements de *beaucoup* d'après la table *Dadv* est donné dans la figure suivante:



Automate des structure du déterminant beaucoup (Dadv)

Figure 4.17e

4.18. ADVERBES (COMPLEMENTS CIRCONSTANCIELS).

L'approche que nous avons choisie consiste à utiliser la description explicite des phrases simples, elle conduit naturellement à privilégier la description des arguments essentiels du verbe (sujets, compléments d'objet directs et compléments prépositionnels essentiels) aux dépens d'arguments inessentiels parmi lesquels, en grande partie, se trouve les adverbes. Nous allons voir que le traitement des adverbes posent des difficultés souvent plus grandes que l'analyse des phrases simples, toutes ces difficultés sont d'ailleurs loin d'être résolues. Avant d'engager la discussion sur leur traitement spécifique nous rappelerons qu'un grand nombre de cas

d'analyse d'adverbes est déjà inclus dans les analyses auxquelles nous avons procédé jusqu'à présent. Nous montrerons ensuite que la description d'un petit nombre de verbes conduit naturellement à la description d'un grand nombre d'adverbe et que cette description peut à son tour être utilisée pour l'analyse d'autres phrases. Nous verrons alors que l'enrichissement de la grammaire, c'est à dire de *DELSYN*, se fait de manière comparable à son enrichissement par la description de nouveaux verbes. La discussion qui suit et les données linguistiques proviennent essentiellement de Maurice Gross 1986b. Notons également que ce domaine, délaissé pendant longtemps par les linguistes, est très peu abordé en analyse automatique car en plus des difficultés rencontrées pour les phrases simples et qui persistent ici, un grand nombre de problèmes nouveaux apparaissent.

Nous avons divisé les adverbes en quatre groupes, chacun ayant un traitement qui lui est particulier :

a. Nous avons déjà traité certains adverbes de manière indirecte comme dans l'exemple de la phrase à verbe support traité en 4.10 :

Luc fait une toilette matinale rapide
Luc fait rapidement sa toilette matinale

La position et la nature de l'adverbe sont alors explicitement notées dans la structure de (2) qui est alors une structure enregistrée dans *DELSYN*. Nous verrons que dans un grand nombre de cas la description explicite de l'adverbe pour un verbe (ou nom prédicatif) est rendue obligatoire comme ici par l'existence de relations transformationnelles entre la phrase avec adverbe et une phrase sans adverbe décrite par ailleurs pour ce même verbe.

b. Nous verrons que la description de certains verbes supports tels que *avoir lieu, se produire*, définit une classe d'adverbes qu'il n'est pas aisé de représenter directement dans leur ensemble. Une description par automates de telles phrases permet de les reconnaître quand ils apparaissent dans des phrases. Cette partie prend essentiellement en compte des adverbes, comme les adverbes de temps, dont la portée est la phrase tout entière.

c. Certaines insertions (adjectifs, propositions relatives) ont des propriétés d'adverbes et portent plus précisément sur le sujet humain de la phrase comme dans

Paul, goguenard, est venu présenté ses excuses

d. Certains adverbes ont une double portée, à la fois sur la phrase et sur le sujet humain, par exemple

Luc a fait cela de son propre chef

Les trois dernières classes correspondent aux trois possibilités d'analyse des adverbes par introduction coréférentielle proposée dans M.Gross 1986b.

Ce découpage ce superpose avec une seconde partition qui sépare chacun des groupes en deux:

A. les adverbes figés ou qu'on peut tout du moins décrire par des automates explicites (les dates par exemples),

B. les groupes adverbiaux contenant des parties plus libres, par exemple des phrases complètes.

L'étude précise des adverbes montre que ces séparations sont souvent difficiles à établir et la méthode que nous proposons n'est qu'une première approximation. Mais en terme de réalisme d'analyse, une analyse automatique qui ne rechercherait pas les adverbes ne s'appliquerait pratiquement à aucun texte. Les descriptions présentées ici, aussi imparfaites soient elles, permettent souvent de débloquer des situations autrement condamnées.

4.18.1 Présentation

Alors que dans les structures de phrases simples que nous avons répertoriées les arguments sont en général contraints par le verbe, les adverbes semblent apparaître de manière indépendante des verbes. Par exemple, l'adverbe *la dernière fois* (nous appelons adverbes indifféremment adverbes simples comme *maintenant* ou les groupes adverbiaux comme *depuis le jour où tu l'as vu pour la dernière fois*) peut apparaître aussi bien dans

(1) *Luc a lu ce livre la dernière fois*

que dans

(2) *La dernière fois, Luc a dit tout ce qu'il pensait à Paul*

alors que les verbes sont très différents. Par ailleurs la mobilité de ces groupes est bien connue puisque (2) est équivalent aux phrases suivantes :

(3) *Luc, la dernière fois, a dit tout ce qu'il pensait à Paul*
(4) *Luc a, la dernière fois, dit tout ce qu'il pensait à Paul*
(5) *Luc a dit, la dernière fois, tout ce qu'il pensait à Paul*
(6) *Luc a dit tout ce qu'il pensait la dernière fois, à Paul*
(7) *Luc a dit tout ce qu'il pensait à Paul, la dernière fois*

Ce type de comportement pousse à construire une description des adverbes de manière indépendante de la description des autres phrases. Cependant, ces phénomènes, bien connus, cachent en fait une énorme diversité.

4.18.2. Rappel sur les adverbes déjà traités

Les phrases (1) à (7) donnent l'exemple d'un adverbe peu lié au verbe dans le sens où *dire* peut être remplacé par un très grand nombre de verbes aussi divers que *pleuvoir, manger, continuer*, etc. En revanche, dans certains cas que nous allons voir ici, les adverbes ont des liens particuliers avec certains verbes, les structures de *DELSYN* pour ces verbes décriront alors le comportement de l'adverbe en question. Cette situation recouvre au moins deux cas nettement différents :

a. Le cas des adverbes figés avec le verbe, ces adverbes n'apparaissent qu'avec un nombre restreint et bien définit de verbes comme *sous silence* dans :

(8) *Luc passe cet événement sous silence*

il est clair que la description sera faite avec l'expression *passer sous silence* (CNP2) et la nature syntaxique (adverbiale ?) de *sous silence* n'a en fait que peu d'importance.

b. Le cas des phrases avec adverbes reliées par transformations à d'autres phrases de *DELSYN*. Dans le chapitre sur les constructions à verbes supports nous avions par exemple mentionné le cas de la descente de l'adverbe dans des phrases telles que :

(9) *Luc fait rapidement sa toilette matinale*
(10) *Luc fait une toilette matinale rapide*

où la structure (11) de (10) :

(11) $Nhum_0$ $Vsup_0$-<*faire-0*> *UN* $Npréd$-<*toilette matinale-0*> *Modif*

est répertoriée dans *DELSYN*. Dès lors, il est obligatoire, pour des raisons de cohérence et pour pouvoir appliquer la transformation sur le résultat de l'analyse, d'introduire la description (12) de (9) dans *DELSYN* :

(12) $Nhum_0$ $Vsup_0$-<*faire-0*> $Adv\text{-}adj_1$ $Poss_0$ $Npréd$-<*toilette matinale-0*>

ce qui, du même coup, règle le cas de l'analyse des phrases avec adverbes telles que (9).

Nous ne reprenons pas le cas (a) qui est traité avec les phrases figées.

Si nous prenons l'adverbe de lieu *à Paris*, il semble parfaitement libre dans une phrase comme

(13) *A Paris, le temps est agréable*

alors que c'est clairement un argument essentiel du verbe *aller* (35L) dans la phrase :

(14) *Luc va à Paris*

ce qui se voit dans l'interdiction

(15) **Luc va*

La description (16) de (14) doit donc explicitement être introduite dans *DELSYN* :

(16) $Nhum_0$ V_0-<*aller-2*> *Loc* N_1

Elle provient d'ailleurs de tables syntaxiques du lexique-grammaire(table *35L*), elle est donc obtenue par le même processus que les structures des verbes de la table 4 (voir le chapitre 4.3).

A côté de ce cas, sont à considérer des phrases (exemple discuté dans J-P. Boons, A. Guillet, Ch. Leclère 1976a) telles que :

(16) *Luc caresse Léa*

(17) *Luc caresse Léa dans le dos*

qui laissent penser que *dans le dos* dans (17) doit être considéré comme un adverbe qui viendrait s'ajouter librement. Mais il faut relier (18) à (17) :

(18) *Luc caresse le dos de Léa*

de structure

(19) $Nhum_0\ V_0$-<*caresse-0*> N_1

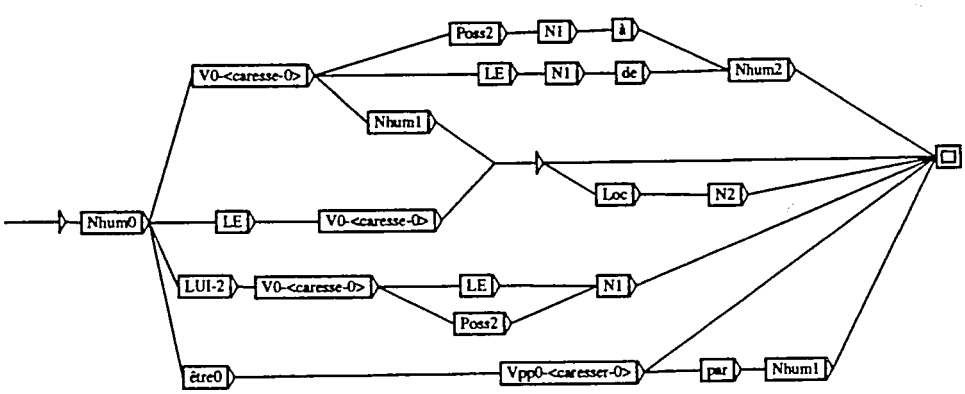
ou plus précisément

(20) $Nhum_0\ V_0$-<*caresse-0*> *DET* Npc_1 *de* $Nhum_1$

où Npc_1 dénote une partie du corps. La structure (20) devant bien évidemment être répertoriée dans *DELSYN* ce qui , comme dans le cas précédent, impose pour préserver la cohérence, d'introduire la structure (21) de (17) :

(21) $Nhum_0\ V_0$-<*caresse-0*> $Nhum_1$ *Prep* Npc_1

D'ailleurs, dans ce cas, dans *le dos* n'a pas plus de mobilité qu'un autre argument du verbe (c'était également vrai dans le cas de *(14)*). L'automate de la figure 4.18aa est ainsi celui du verbe *caresser* de la table 32CL (J-P. Boons, A. Guillet, Ch. Leclère 1976b). Cet automate se déduit, comme dans le cas des verbes de la table 4, d'un automate de référence *Aref(32CL)* et de l'entrée du verbe dans la table.



Automate des structures du verbe *caresser* de 32CL

Figure 4.18aa

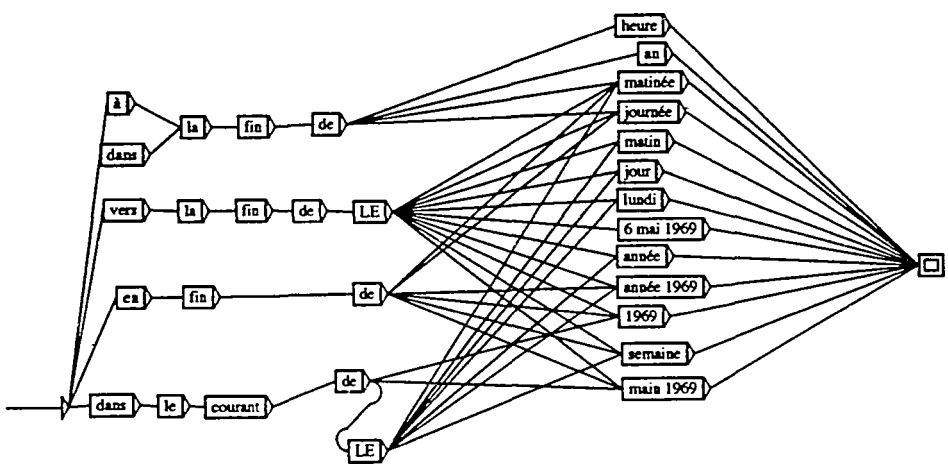
La donnée de cet automate, et donc des structures qu'il représente dans *DELSYN*, conduit à une analyse de ces adverbes semblable à celles des autres arguments essentiels du verbe.

4.18.3. La description des verbes support d'adverbes de portée phrastique

Prenons le verbe *se produire*, dont la structure de base est

(8) *Qu P se* V_0-*<produire-0> Prép* N_1

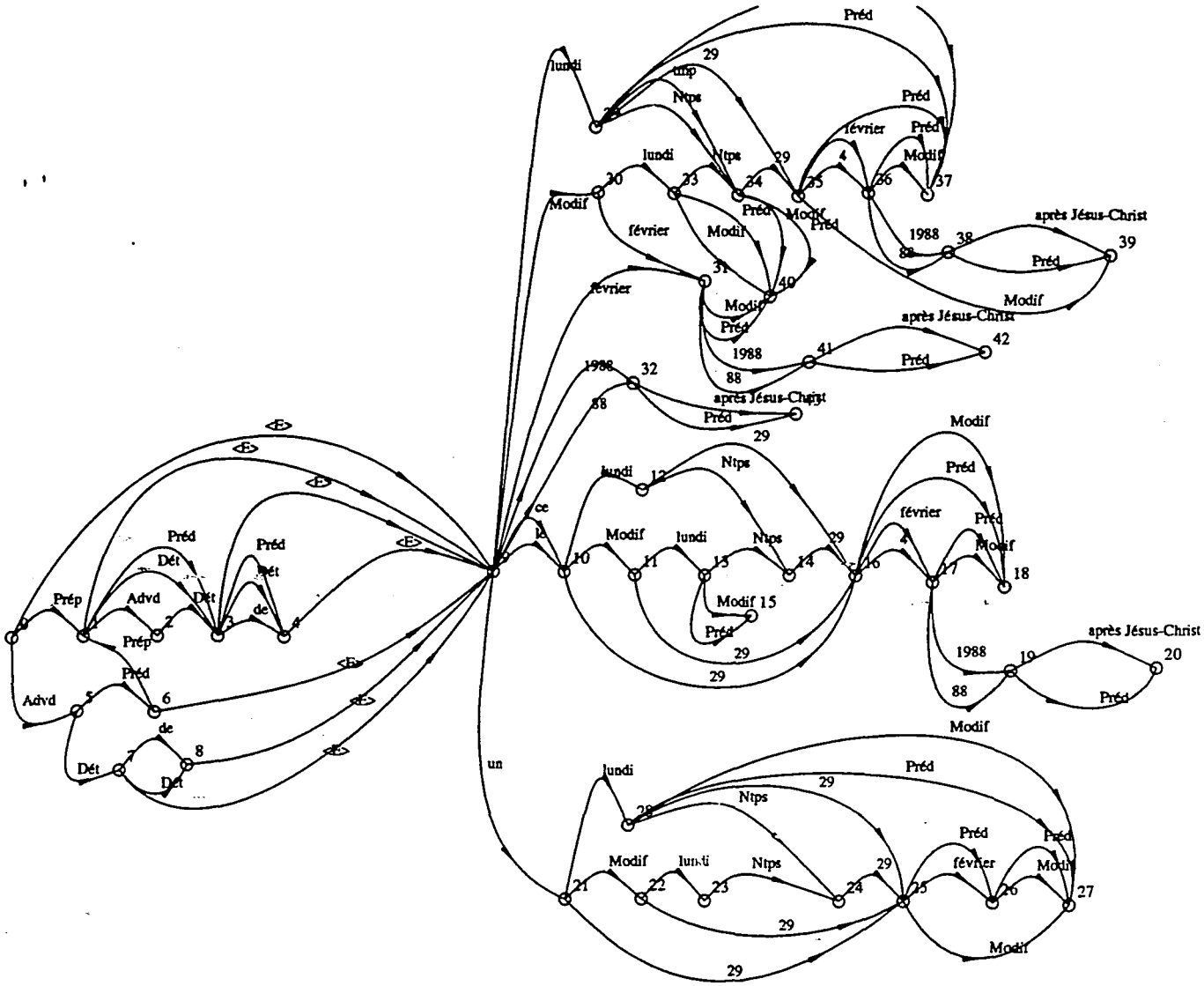
Le terme N_1 recouvre un grand nombre de formes telles que *Lundi dernier, le 6 juin, lundi en huit, cinq minutes avant que tu n'arrives, plus tard que prévu.* N_1 peut être introduit par une préposition qu'il est impossible de déduire du verbe, par exemple, *à la première heure, à la fin de l'histoire, de lundi à dans trois jours* . La description de la distribution de ce verbe, c'est-à-dire la description des formes prises par N_1, recouvre donc très largement la description des adverbes de temps. Il n'est pas question ici d'envisager une description complète de ces termes. L'automate de la figure suivant, construit à partir du tableau de quelques compléments de *se produire* (M. Gross 1986b : p229) donne un aperçu de cette complexité.



Automate de la distribution de Vsup-<produire-0>

Figure 4.18a

Ce premier automate, extrêmement partiel, met déjà en évidence la complexité d'un tel travail. Le problème de la lisibilité, et donc de la correction de l'automate, se pose très rapidement. D. Maurel 1990 a montré que la seule description des termes comprenant une date, telle que *le lundi 8 novembre* justifie une description minutieuse car elle est d'une grande complexité. La description finale est alors un automate décrivant l'ensemble exacte de ce type de séquences (nous discutons en 5.2 de l'utilisation des grammaires locales dans la construction d'un tel automate), un extrait de cet automate est donnée figure 4.18b.



Automate des dates de D Maurel 1989 (voir également 5.3)

Figure 4.18b

La méthode employée par D. Maurel 1990 consiste à construire un automate comme celui de 4.18b où chaque transition doit être remplacée par un ensemble de termes (par un transducteur *ABREV* particulier à cette grammaire) et où des contraintes opèrent localement. Nous verrons en 5 que cette méthode se formalise très bien avec des grammaires locales.
Cet automate décrit de manière précise un ensemble d'arguments possibles de *se produire*, cependant un nombre important de phénomènes restent non décrits. Ainsi, à côté d'adverbes qu'on peut décrire explicitement comme les précédents, ou d'adverbes figés tels que *au gré des circonstances* dans

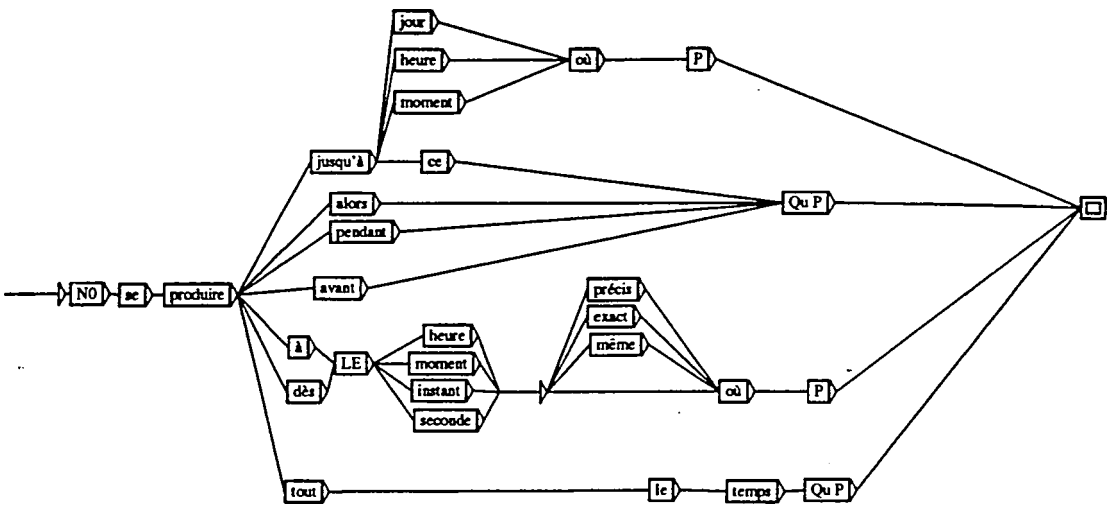
(9) *Cela s'est produit au gré des circonstances (PCDC)*

se trouvent des adverbes, ou des propositions subordonnées circonstancielles, contenant des phrases entièrement libres comme dans

(10) *Cela s'est produit avant que le film ne soit finit*

l'adverbe *avant que le film ne soit fini* ayant la structure *avant Qu P*.

Nous donnons à la figure 4.18c un ensemble de constructions de *se produire* avec des complétives et/ou des subordonnées.



Automate de structures phrastiques de *se produire*

Figure 4.18c

L'ensemble des structures de 4.18b et 4.18c sont à introduire explicitement dans *DELSYN*. Cette opération est à effectuer pour un certain nombre de verbes tels que *avoir lieu, se passer, remonter à*. Pour obtenir cet ensemble d'automates, dans l'état actuel des connaissances, il est impossible de procéder comme pour les verbes simples (de la table 4 en 4.3 par exemple). En effet ces verbes ont des comportements à la fois très différents et très complexes. Il faut donc construire explicitement chacun des automates. L'intérêt de cette description est bien sûr de rendre compte du fait que tous ces adverbes de temps peuvent apparaître dans la plupart des phrases, comme le montre les phrases (2) à (7).

Etant donné la description précédente, la question devient comment analyser les phrases (2) à (7) ?

Une solution consiste à marquer dans chaque structure de phrase la liste des positions où des adverbes peuvent apparaître. Ainsi, pour la phrase

(11) *Luc a annoncer à Paul qu'il devait partir*

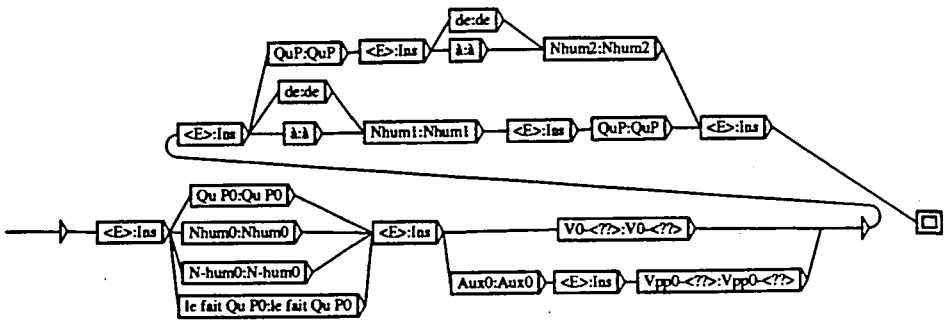
de structure

(12) $Nhum_0\ Aux_0\ Vpp_0$-<*annoncer-1*> *à* $Nhum_1\ QuP$

un adverbe peut apparaître entre chaque élément sauf entre la préposition *à* et $Nhum_1$. Nous marquons ces positions de la manière suivante:

(13) *Ins* $Nhum_0$ *Ins* Aux_0 *Ins* Vpp_0-<*annoncer-1*> *Ins à* $Nhum_1$ *Ins QuP Ins*

Pour obtenir ces structures, deux solutions sont a priori possibles, soit les noter directement dans les automates de référence, soit les déduire des structures, qui telles que *(12)*, sont déjà présentes dans *DELSYN*. Etant donné la grande généralité du phénomène, au moins pour les tables de verbes libres, nous avons construit un transducteur, *Ajoute_Insertions* dont nous donnons un extrait à la figure 4.18d. Ce transducteur associe aux structures de *DELSYN* ((12) par exemple) leur structure équivalente avec insertion ((13) pour (12)).



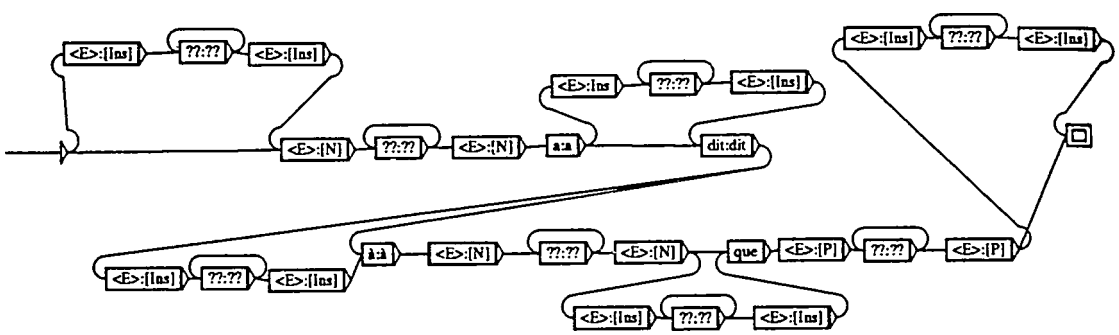
Extrait du transducteur *Ajoute_Inserstions*

Figure 4.18d

Ainsi on peut vérifier que *Ajoute_Insertions*((12)) = (13). La donnée de ces structures garantit que le résultat de l'analyse sera correct mais le problème de l'explosion combinatoire se pose. Voyons l'analyse de la phrase

(14) *A la première heure Luc a annoncé à Paul qu'il devait partir*

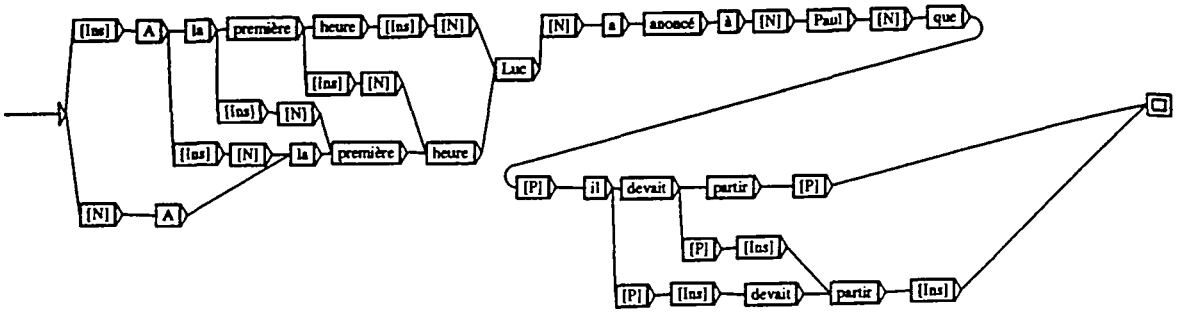
L'application de *Ajoute_Insertions* sur la structure (2) donne le transducteur suivant :



Premier schéma de transducteur (T_1) d'analyse des phrases avec adverbes

Figure 4.18e

En ne tenant compte que de la première insertion possible, le transducteur T_1 associe déjà à (14) l'ensemble représenté par l'automate de la figure 4.18f



Automate A_1 inclus dans $T_1(14)$

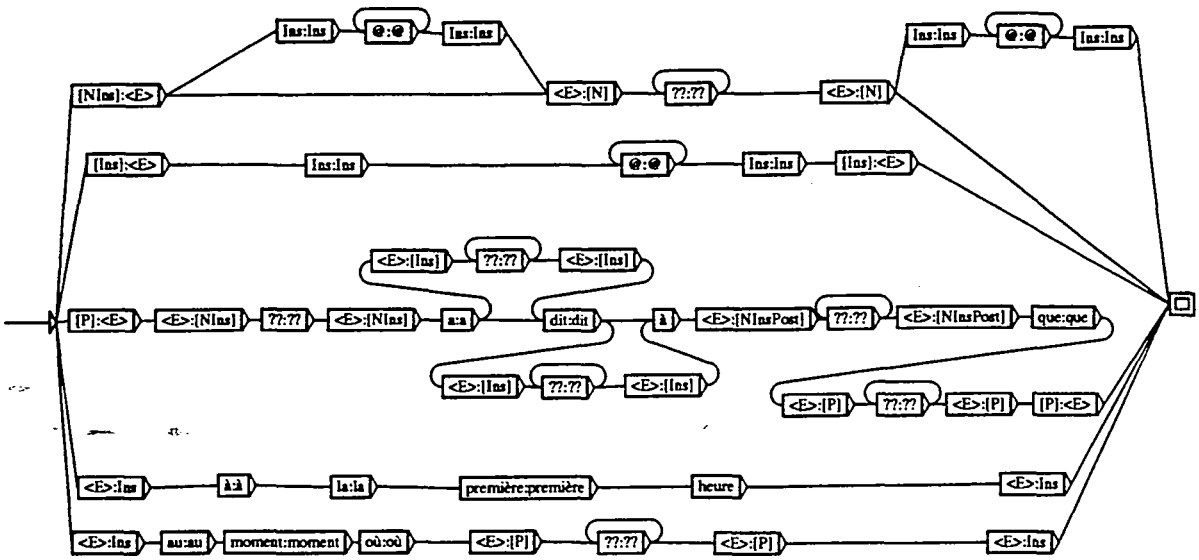
Figure 4.18f

Les séquences de cet automate représentent, dans la partie qui précède le verbe, l'ensemble des solutions pour découper 5 mots : *A la première heure Luc* en deux parties contiguës. Il y a cinq découpages possibles:

0 / A la première heure Luc
A la / première heure Luc
A la première / heure Luc
A la première heure / Luc
**A la première heure Luc / 0*

et le dernier n'est pas pris en compte puisque la deuxième partie (entre les marqueurs [N] et [N]) contient au moins un mot. Il est donc clair que si la séquence préverbale est beaucoup plus longue (plus de vingt mots n'aurait rien d'exceptionel) le nombre d'ambiguïtés à gérer deviendrait très grand. Trois des quatres analyses partielles de A_1 sont détruites dans la suite de l'analyse, le résultat n'est donc pas altéré, mais la rapidité du traitement risque de diminuer. Avec cette approche, la taille des automates intermédiaires (avant l'obtention du point fixe) risque d'être inutilement grande.

La solution adoptée consiste à exploiter la possibilité de mener conjointement une analyse descendante (top-down) et ascendante (bottom-up). Le découpage *Nom Adverbe* n'est fait qu'au moment où les adverbes figés ou partiellement figés ont été détectés. Nous remplaçons pour cela le transducteur T_1 par T_2 de la figure 4.18g.



Transducteur T_2 d'analyse pour les deux adverbes *à la première heure* et *au moment où P.*

Figure 4.18g

Reprenons l'analyse de (14), ou plus exactement de (15) :

(15) [P] *A la première heure Luc a annoncé à Paul qu'il devait partir* [P]

La première application de T_2 sur (15) donne à la fois le découpage suivant de la phrase :

(16) $(A\ la\ première\ heure\ Luc)_{NIns}\ a\ annoncé\ à\ (Paul)_{Nhum1}\ (qu'il\ devait\ partir)_{QuP}$

où *NIns* marque que le groupe à analyser est soit un groupe nominal, soit un groupe nominal accolé à un adverbe (en fait une insertion quelconque notée *Ins*) et le marquage de l'adverbe par

(17) *Ins A la première heure Ins Luc a annoncé à Paul qu'il devait partir*

Le résultat de la transduction est donc (18) = $T_2(15)$

(18) [NIns] *Ins A la première heure Ins Luc*[NIns] *a annoncé à* [N] *Paul* [N] *qu* [P] *il devait partir* [P]

La deuxième application doit donc analyser $(A\ la\ première\ heure\ Luc)_{Nins}$ comme la concaténation d'un adverbe et d'un nom mais elle dispose déjà du découpage. Cela donne, comme on peut le vérifier, le résultat suivant (19) = $T_2(18) = T_2^2(15)$

(19) Ins *A la première heure* Ins [N] *Luc* [N] *a annoncé à* $(Paul)_{Nhum1}$ (*que* [N] *il* [N] *devait* [$V_1{}^0W$] *#il # partir* [$V_1{}^0W$])$_{QuP}$

qui nous ramène à une situation connue. L'analyse a certes été largement facilitée par le fait que l'adverbe était explicitement décrit, mais on peut voir sur T_2 que la détection de segments incomplets d'adverbes, comme *au moment où,* permet également de détecter des points de séparation entre les arguments.

Comme précédemment, le transducteur T_2, qui sera donc une partie de *f_DELSYN*, n'aura pas à être écrit "à la main". T_2 est en effet obtenu par l'application des deux transducteurs sur la structure initiale (12), plus précisément $T_2 = ABREV(Ajoute\_Insertions((12)))$.

4.18.4. Analyse des phrases avec adverbes à portée sur le sujet

Dans une phrase telle que

(20) *Max, goguenard, est arrivé avec presqu'une heure de retard*

goguenard a la même mobilité que par exemple un adverbe de temps:

(21) *Goguenard, Max est arrivé avec presqu'une heure de retard*
(22) *Max est arrivé, goguenard, avec presqu'une heure de retard*
(23) *Max est arrivé avec presqu'une heure de retard, goguenard*

alors qu'il a clairement des propriétés différentes puisqu'il possède un sujet de type $Nhum_0$. *Goguenard* est en effet décrit par ailleurs comme admettant la structure

(24) $Nhum_0$ *être* Adj_0-<*goguenard-0*>

Plus généralement, on constate qu'un grand nombre de phrases peuvent remplacer (24) le rôle de source de ce type d'adverbe. Les couples suivants :

(25) *Luc est surpris par le résultat de l'enquête*
(26) *Surpris par le résultat de l'enquête, Luc est passé aux aveux*

(27) *Luc était sur ses gardes*
(28) *Luc a continué d'avancer, sur ses gardes*

montrent que des phrases libres aussi bien que figées peuvent être à l'origine de tels adverbes. Comme dans le cas des adverbes de temps, le cas des phrases figées est le plus favorable puisqu'alors une large partie de l'incertitude sur les frontières du groupe adverbial est levée par simple reconnaissance.

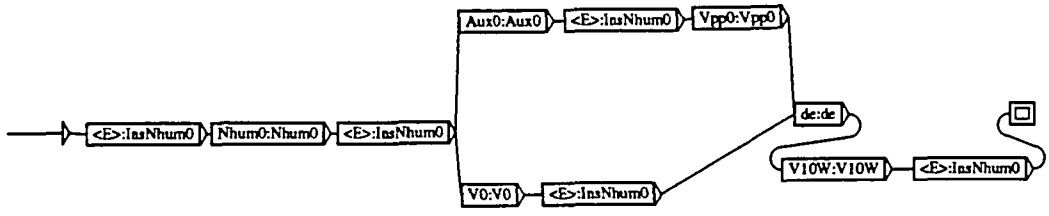
Dans l'état actuel du traitement et en première approximation, nous avons considéré que toute phrase de type

(29) $Nhum_0$ *être Prep W*

ou de type (comme (25))

(30) $Nhum_0$ *être-passif Vpp W*

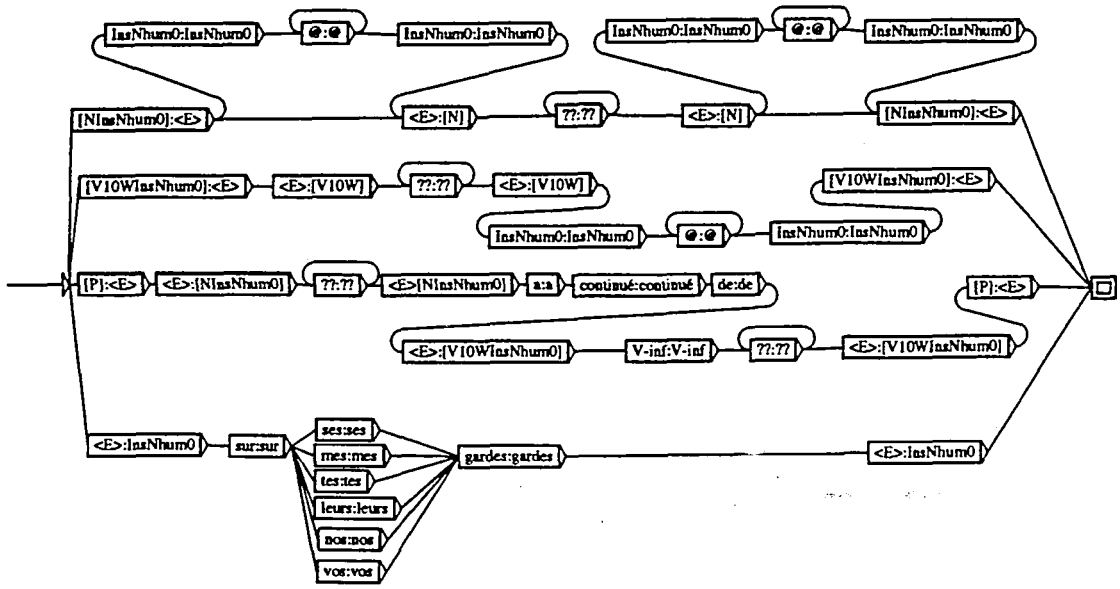
pouvait être la source d'un adverbe ou d'une insertion dans toute phrase à sujet libre de type $Nhum_0$. On peut dans un premier temps chercher à résoudre le problème de l'analyse de (28) comme en 4.18.3 ce qui, techniquement, s'effectuerait ainsi : le transducteur *Ajoute_InsNhum0* appliqué à *DELSYN* donne l'ensemble des structures avec les positions d'insertion. Nous donnons un extrait de ce transducteur à la figure 4.18h :



Extrait du transducteur *Ajoute_InsNhum*$_0$

Figure 4.18h

Par ailleurs un transducteur *Cherche_AdvNhum*$_0$ s'applique sur *DELSYN* pour associer à chaque phrase de type (29) ou (30) la partie adverbiale. Il n'est pas très intéressant de détailler *Cherche_AdvNhum0*, montrons simplement que le résultat de *ABREV*(*Cherche_AdvNhum*$_0$(*DELSYN*)) ∪ *ABREV*(*Ajoute_InsNhum*$_0$(*DELSYN*)) qui vient enrichir *DELSYN* donne un transducteur, dont nous donnons un extrait en *4.18i*. Ce transducteur effectue l'analyse de (28) en suivant le même principe que l'analyse des adverbes à portée phrastique donné en 4.18.4.



Extrait de *f_DELSYN* permettant l'analyse de *(28)*

Figure 4.18i

Ce transducteur s'applique de la manière suivante:

f_DELSYN(28) donne (31) = *f_DELSYN*(28)

(31) [NInsNhum$_0$] *Luc* [NInsNhum$_0$] *a continué d* [$V_1{}^0$WInsNhum$_0$]
#Luc # avancer InsNhum$_0$ *,sur ses gardes* InsNhum$_0$ [$V_1{}^0$WInsNhum$_0$]

où $V_1{}^0WInsNhum_0$ correspond à une infinitive qui peut être suivie d'un adverbe portant sur $Nhum_0$. Dans un même temps le marqueur $InsNhum_0$ indique que *sur ses gardes* a été détecté comme pouvant être un tel adverbe. On peut alors vérifier que (32) = *f_DELSYN(31)* = *f_DELSYN2(28)*

(32) [N] *Luc* [N] *a continué d* [$V_1{}^0$W] *#Luc # avancer* [$V_1{}^0$W] InsNhum$_0$, *sur ses gardes* InsNhum$_0$

(32) continue à s'analyser comme précédemment.

Cette analyse présente cependant un défaut, elle accepte aussi bien (28) que (33) :

(33) **Luc a continué de s'avancer, sur mes gardes.*

puisque l'analyse de l'adverbe se fait indépendamment du reste de la phrase. Il existe deux solutions pour remédier à ce problème: soit spécifier dans la structure générale de *P#Adv* (Phrase avec adverbe) les coréférences de manière explicite comme pour une expression figée du type *prendre ses jambes à son cou,* soit, et c'est ce que nous avons retenu, reconstituer explicitement la phrase sous-jacente :

(34) *Luc est sur ses gardes*

Donner dans le résultat de l'analyse les phrases sous-jacentes est un important gain de précision qui distingue les analyses transformationnelles des analyses distributionnelles . Par ailleurs l'analyse n'acceptera plus des séquences telles que (33).

Cette méthode conduit bien sûr à un transducteur *f_DELSYN* différent, nous en donnons un extrait représentatif correspondant à 4.18i dans la figure 4.18j. L'analyse de *(28)* se fait alors de la manière suivante:

à (28) *f_DELSYN* associe

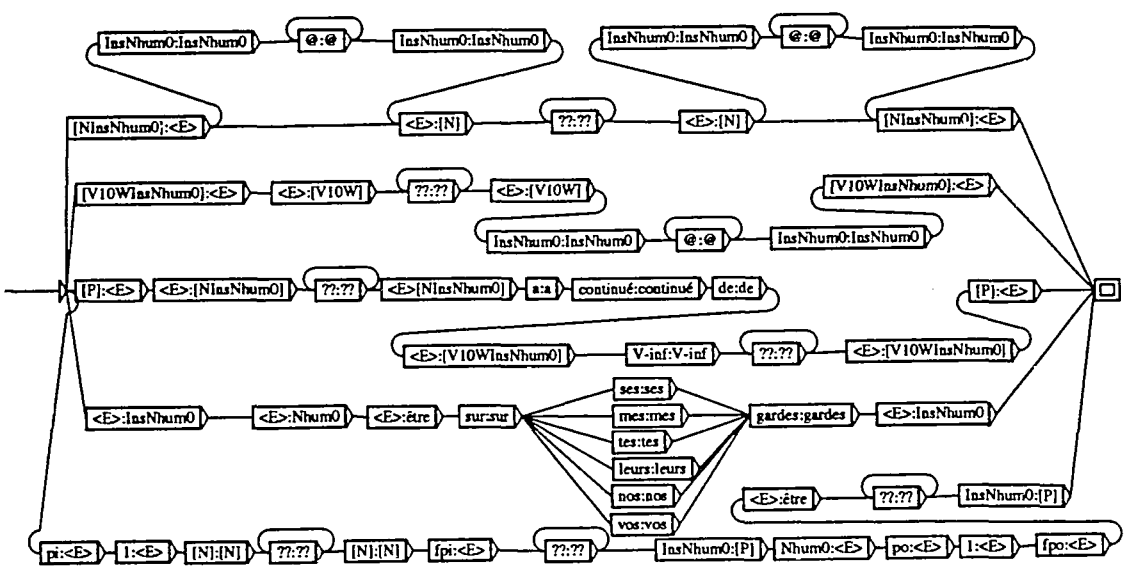
$$(35) = f\_DELSYN(28)$$

(35) [N] *Luc* [N] *a continué d* [$V_1{}^0$WInsNhum$_0$] # *Luc # avance* InsNhum$_0$ Nhum$_0$ *être sur ses gardes InsNhum0* [$V_1{}^0$WInsNhum$_0$]

qui ne diffère pas fondamentalement de (31). La seconde application du transducteur donne
$(36) = f\_DELSYN(35) = f\_DELSYN^2(28)$

(36) (*Luc*)$_{Nhum0}$ *a continué de* [$V_1{}^0$W] #*Luc*# *avance* [$V_1{}^0$W]
([P] *Luc être sur ses gardes*[P])$_{InsNhum0}$

dans laquelle la phrase (34), qui apparaît explicitement, pourra être analysée de manière indépendante.



Extrait de *f_DELSYN* permettant l'analyse transformationnelle de (28)

Figure 4.18j

4.18.5. Analyse des adverbes à double portée

M. Gross 1986b : p148 propose une analyse de phrases dont les adverbes ont une double portée, à la fois sur la phrase (Cf 4.18.3) et sur le sujet *Nhum0* (Cf 4.18.4). Par exemple la phrase

(37) *Max est venu de sa propre initiative*

est analysée à partir de (38) et (39)

(38) *Max est venu*
(39) *Que Max soit venu s'est fait de sa propre initiative*

Cet exemple n'est analysable automatiquement ni par la méthode suivie en 4.18.3 puisque le possessif *sa* doit être coréférent au sujet *Nhum0* de (39) ni par la méthode suivie en 4.18.4 puisque la phrase

(40) **Max est de sa propre initiative*

n'existe pas. La table PAC, à laquelle appartient l'adverbe, donne certes par ailleurs la phrase

(41) *Max agit de sa propre initiative*

mais elle ne permet pas d'analyser (37). Une solution consiste, comme au paragraphe précédent, à suivre l'analyse (37) = (38),(39) dans le processus même de l'analyse.

Nous supposons donc disposer dans *DELSYN* de la structure (42) de (39)

(42) *Qu Nhum$_0$ V_1^0 W se fait de Poss$_0$ propre initiative*

Notons que nous avons déjà rencontré ce type de structure dans la table *4*, on pourra se reporter à l'automate de référence de la table 4 *Aref*(4) donné en 4.3. La structure (42) provient alors l'automate dont nous donnons un extrait à la figure suivante

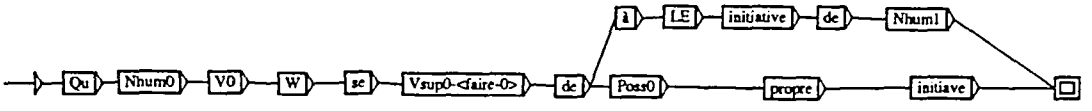
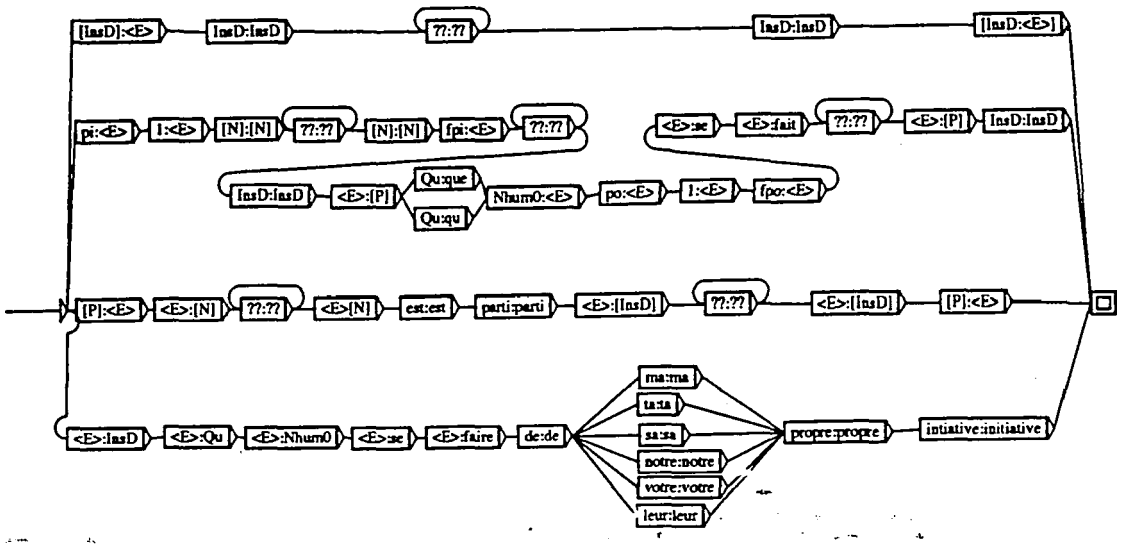


Figure 4.18k

Nous pouvons, à partir de 4.18j, enrichir *f_DELSYN* de manière à incorporer (42), ce qui donne un transducteur dont nous donnons l'extrait pertinent à la figure 4.18k



Extrait de *f_DELSYN* permettant l'analyse de *(37)*

Figure 4.18l

L'analyse de (37) se passe alors de la manière suivante: (37) se voit associée (43) = f_DELSYN(37)

(43) [N] *Max* [N] *est parti* [Ins] *InsD Qu Nhum*$_0$ *V*$_0$ *W se fait de sa propre initiative InsD* [Ins]

qui se voit à son tour associée (44) = *f_DELSYN*(43) = *f_DELSYN*^{2}(37)

(44) [N] *Max* [N] *est parti* InsD [P] *Que* (*max*)$_{Nhum0}$ (*être parti*)$_{V0}$ *se fait de sa propre initiative* [P] InsD

et l'analyse se poursuit par l'analyse indépendante de la phrase entre [P] et [P] d'après la structure (42).

Cet aperçu des possibilités d'analyse des phrases avec adverbes n'est qu'une première approximation de ce qu'il est possible de faire. Nous pensons qu'elle met déjà en évidence la nécessité d'une analyse véritablement transformationnelle, les exemples précédents montrant qu'une approche distributionnelle est fréquemment bloquée dans des situations où une information explicite (de coréférence par exemple) permet une analyse profonde.

5. GRAMMAIRES DE CONTRAINTES LOCALES

5.1. PRESENTATION

Nous avons vu que la simple consultation du dictionnaire des mots d'un texte donné conduit à des ambiguïtés pour un très grand nombre de mots. Nous avons vu que cela ne modifiait pas l'application de la grammaire, les ambiguïtés étant levées naturellement au cours de l'analyse si la grammaire est suffisamment précise. Cependant, si le but est uniquement de lever des ambiguïtés et si la rapidité de traitement est cruciale, il est possible de n'explorer qu'une partie bornée du contexte du mot. Ce traitement contextuel, et donc local, conduit à la construction de ce que nous appellerons grammaires de contraintes locales (ou grammaires locales).

Il existe un autre avantage à l'utilisation de grammaires locales: si on applique ces grammaires à la phrase avant qu'elle ne soit analysée complètement, on ne change pas le résultat final (les grammaires locales et la grammaire générale ne doivent certes pas être contradictoires), mais le traitement peut devenir sensiblement plus rapide. En effet, le fait de lever des ambiguïtés conduit à ne pas explorer des hypothèses d'analyse qui de toute façon se seraient révélées incorrectes.

Nous allons montrer comment une opération sur les automates, simple et formellement bien définie, permet d'appliquer un certain type de grammaire locale. Nous montrerons également, que, appliqué à des textes, cet outil constitue un filtre simple et efficace pour la constitution d'index. Construire des index de corpus permet alors de tester la correction de ces grammaires. Par ailleurs il existe plusieurs moyens de projeter la grammaire générale telle qu'elle est décrite dans DELSYN sur des grammaires locales. La même expérience conduit alors également à tester indirectement la grammaire générale.

D'autre part, si on remarque que les grammaires locales s'appliquent à des phrases et que la description de la grammaire générale (dans *DELSYN*) prend également la forme de phrases, on constate que l'on peut utiliser cet outil sur la grammaire, soit pour la vérifier, soit même pour la construire. Nous donnerons l'exemple de la grammaire des adverbes de dates tel qu'il est traité par D. Maurel 1989 et montrerons qu'elle consistait en fait une grammaire approximative mais contrôlable à la main, sur laquelle on applique une grammaire locale, le résultat étant une grammaire exacte.

Certaines précisions formelles sur ces opérations et leur description complète peuvent être trouvées dans E. Roche 1992a.

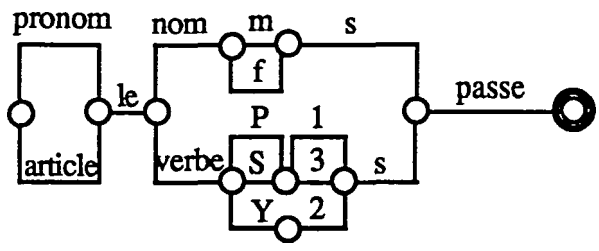
5.2. UN EXEMPLE

Prenons la séquence

le passe

Les deux mots sont ambigus, *le* peut être un pronom préverbal ou un article et *passe* est un verbe (*passer*) ou un nom. Le nom peut lui-même signifier *une passe* (de football) ou *un passe*

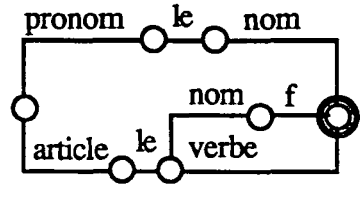
(une clé passe-partout). Par ailleurs le verbe *passe* porte une ambiguïté en temps (indicatif présent, subjonctif présent ou impératif présent) et en personne (première, deuxième ou troisième). L'analyse morphologique de cette séquence conduit donc à l'automate suivant (légèrement simplifié):



Représentation simplifiée de l'analyse morphologique de *le passe* : *morpho(le passe)*

Figure 5.2a

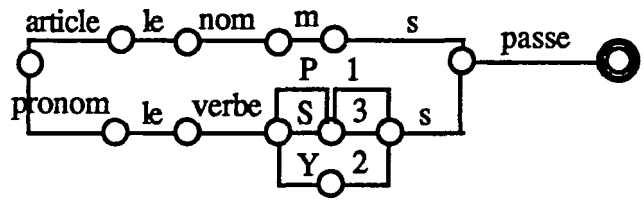
Par ailleurs, d'après ce que nous savons des pronoms préverbaux (voir section 5.6), le pronom *le* ne peut être suivit que d'un verbe ou d'un autre pronom préverbal. Par conséquence la séquence *ppv Nom* est interdite. D'autre part l'article défini masculin ne peut être suivit d'un verbe conjugué ni d'un nom féminin. Par conséquent les séquences *Article le verbe* et *Article le Nom f* sont interdites. Si on représente ces trois séquences dans un même automate on obtient la représentation suivante:



Automate représentant des séquences interdites : grammaire locale

Figure 5.2b

D'après ce second automate des séquences interdites, l'automate de la phrase doit être celui de la figure 5.2c :



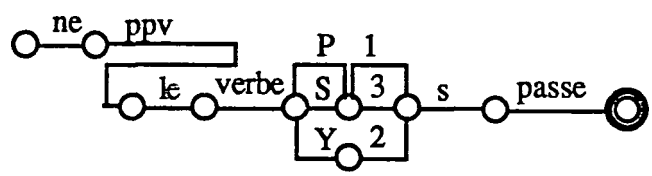
Représentation de le passe après application de la grammaire locale

Figure 5.2c

On peut vérifier que cet automate représente exactement l'automate de la phrase moins l'ensemble des séquences interdites. Le problème est donc d'obtenir l'automate de la figure 5.2c à partir ce ceux des figures 5.2a et 5 2b. Notons par ailleurs que si on étend le contexte et que *ne* précède la séquence, donc que la séquence à traiter est

ne le passe

alors, comme *ne* ne peut être suivit que d'un verbe conjugué ou d'un pronom préverbal (et en tout cas pas d'un article défini), la séquence peut être complètement désambiguée avec pour résultat l'automate suivant:



Résultat de l'application de la grammaire locale

Figure 5.2d

5.3. L'ALGORITHME

5.3.1. Définition formelle du problème

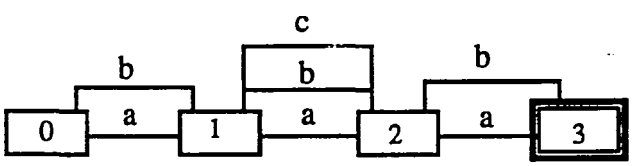
Commençons par préciser le problème:

Soit une phrase *ph*, l'automate A_1 = *morpho(ph)* qui le représente, tel que A_1 est défini par le 5-uple $(Alph, Q_1, i_1, F_1, d_1)$ où *Alph* est l'alphabet de l'automate, Q_1 l'ensemble d'états, i_1 l'état initial, F_1 l'ensemble des états finaux et d_1 la fonction de transition de $(Q_1 * Alph)$ sur Q_1

(l'automate A_1 est déterministe et acyclique). De la même manière l'ensemble des contraintes, c'est-à-dire l'ensemble des séquences interdites est défini par l'automate $A_2 = (Alph, Q_2, i_2, f_2, d_2)$ (l'ensemble des états terminaux est un singleton puisque si on interdit un préfixe on interdit lui même). Ces deux automates définissent chacun le langage qu'ils reconnaissent: $L_1 = L(A_1)$ et $L_2 = L(A_2)$. L_2 représente un ensemble de séquences interdites, donc si A représente l'automate après application des contraintes, A est le résultat que nous recherchons, alors A doit vérifier $L(A) = L_1 \setminus Alph^* L_2 Alph^*$. C'est-à-dire que L est la différence, au sens des ensembles, de L_1 et de l'ensemble de tous les mots qui ont au moins un de leurs facteurs dans L_2. Nous allons maintenant montrer que cette opération de type ensembliste peut se calculer directement sur les automates sans avoir à calculer $Alph^* L_2 Alph^*$ ce qui est toujours difficile car nous opérons avec des alphabets qui augmentent en taille dynamiquement. Nous noterons cette opération $A = A_1 f\text{-} A_2$ et l'appelerons *différence par facteurs.*

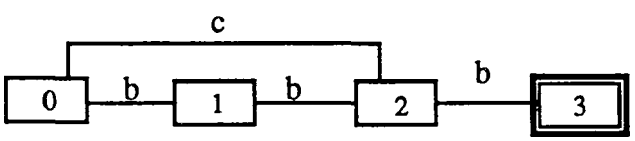
5.3.2. Description informelle de l'algorithme

Appliquons tout d'abord cet algorithme à un exemple de petite taille. A_1 est l'automate de la figure 5.2d et que A_2 est celui de la figure 5.2e , nous voulons construire l'automate $A_1 f\text{-} A_2$.



Automate A_1 équivalent à l'automate d'une phrase à désambiguer

Figure 5.2d



Automate A_2 équivalent à l'automate d'une grammaire locale

Figure 5.2e

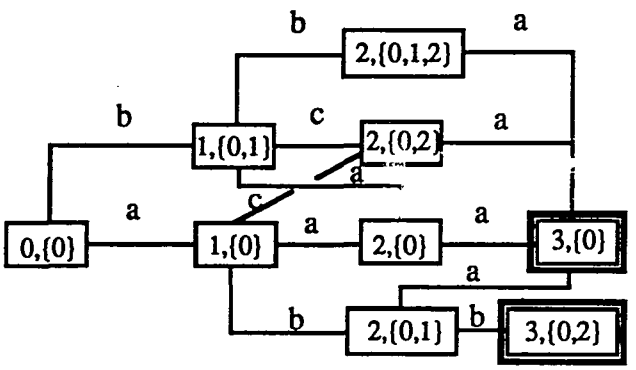
Chaque état de l'automate $A = A_1 f\text{-} A_2$ est étiqueté par un état de A_1 et un ensemble d'états de A_2. Plus concrètement, l'automate $A = A_1 f\text{-} A_2$ de la figure 5.2f est construit de la manière suivante:

L'état initial est étiqueté par *(0,{0})*, où le premier *0* correspond à l'état *0* de A_1 (que nous noterons 0_1). La lettre *a* conduit de 0_1 à l'état 1_1 de A_1 mais elle ne conduit à rien dans A_2, ce

qui nous amène à construire l'état (1,{0}). Cela signifie que, pour a, 0 mène à 1 dans A_1 mais que{0} ne mène à rien (donc à l'ensemble vide) dans A_2 , ensemble auquel nous ajoutons systématiquememt l'état initial pour marquer qu'une séquence peut commencer à ce nouvel état dans A_1.

D'un autre côté, $d_2(\{0\},b) = \{1\}$ auquel nous ajoutons, comme pour a, l'état 0; et donc, pour A, $d((0,\{0\}),b) = (d_1(0,b), \{0,d_2(0,b)\}) = (1,\{0,1\})$.

Pour chaque état que nous construisons, nous listons les états auxquels il peut correspondre dans A_2 et, pour chacun de ces états nous donnons leur image pour chaque lettre. Quand l'état de A a une de ses références dans Q_2 qui conduit à un état terminal de A_2, alors une séquence complète de A_2 a été reconnue et, par conséquence elle doit être effacée. C'est par exemple le cas si nous considérons l'état (2,{0,1,2}) dans A: $d_2(\{0,1,2\},b) = \{1,2,3\}$ où 3 est final, cet état n'a donc pas de transition pour la lettre b ce qui conduit à détruire le chemin bbb qui, en effet, était interdit par A_2.



Automate A résultat de l'application de A2 su A1: A = A1 f- A2

Figure 5.2f

5.3.3. Définition de l'automate résultat

Si $A_1 = (Alph, Q_1, i_1, F_1, d_1)$ et
$A_2 = (Alph, Q_2, i_2, f_2, d_2)$ sont deux automates déterministes, nous définissons tout d'abord la fonction F par :

$A = F(A_1, A_2)$ si $A = (Alph, Q, i, F, d)$ tel que:

$Q = Q_1 \times P(Q_2)$ (i.e le produit cartésien de Q_1 par l'ensemble des parties de Q_2)
$i = (i_1,\{i_2\})$
$F = \{(q,S)$ avec $q \in F_1$ et $S \in P(Q_2)\}$

Nous définissons une fonction intermédiaire G de $(P(Q_2)*Alph)$ sur $P(Q_2)$: Pour $S \in P(Q_2)$, $s \in Alph$, $G(S, s)$ est défini par: si $q \in S$ et $d(q, s) = f_2$ alors $G(S, s) = \varnothing$
sinon $G(S, s) = d(S, s) \cup \{i_2\}$.

La fonction de transition d de A est défini pour chaque état $q = (q_1, S)$ dans Q (où $q_1 \in Q_1$ et $S \in P(Q_2)$) et pour chaque lettre s dans *Alph* par :

si $(d_1(q_1, s) = \emptyset$ ou $G(S, s) = \emptyset)$ $d(q,s) = d((q_1,S),s) = \emptyset$
sinon

$$d(q,s) = d((q_1,S),s) = (d_1(q_1,s),G(S,s)).$$

On peut maintenant donner la proposition qui garantit que cette opéraion sur les automates est bien équivalente à l'opération sur les langages qu'ils représentent :

Proposition: $F(A_1,A_2)=A_1$ f- A_2

La preuve n'est pas difficile à établir et elle aide à vérifier que la définition est correcte. Nous allons tout d'abord montrer que $L(A_1)$ f- $L(A_2) \subseteq L(F(A_1,A_2))$ puis la réciproque. Nous donnons d'abord deux lemmes simples (nous utilisons la notation classique qui consiste à écrire *quq'* si la chaîne de caractères *u* est un chemin de l'état *q* à l'état *q'*).

Lemma 1: Si l'état (q,S) de A est atteint par la chaîne $x \in Alph^*$ (i.e. ix(q,S)) alors pour tout q' dans S on peut trouver une décomposition de la chaîne (ou du chemin) x=u.v tel que v est un chemin de A_2 de l'état initial i_2 à q' (i.e. i_2vq').

Nous montrons ce lemme par récurrence sur la longueur |x| de x.
1. Si |x|=0, alors $i\emptyset(q,S)$ avec $(q,S)=(i_1,\{i_2\})$. $i_2\emptyset i_2$ montre le lemme pour $q' \in S$.
2. Supposons que |x|>0 et que l'hypothèse de récurrence a été montrée pour toute chaîne x' telle que |x'|<|x|. Supposons que l'état (q,S) de A est atteint x (i.e. ix(q,S)). Comme |x|>0, x peut être écrit x'.a où *a* est une lettre ($a \in$ Alph). Cela signifie que i(x'.a)(q,S) ou que ix'(q',S')a(q,S) (l'état (q',S') de A est un prédécesseur de (q,S) pour a). Maintenant, par la définition de G et par la définition de la fonction de transition d, si $q" \in S$ cela signifie soit que $q"=i_2$ soit que il existe q"' dans S' tel que q"'aq". Dans le premier cas, $i_2\emptyset q"$ montre la proposition et dans le second cas, comme nous pouvons appliquer l'hypothèse de récurrence à x' et que x'=u.v and $i_2vq"'$, nous pouvons voir que $i_2(v.a)q"$, ce qui est la conclusion pour q". Comme nous avons trouvé un chemin $i_2pq"$ pour tout q" dans S nous avons montré la proposition pour x.

Lemma 2: Si ix(q,S) alors i_1xq (La preuve est évidente par induction sur |x|).

On peut maintenant montrer la proposition.

[[Soit $x \in L$, supposons que x n'est pas reconnu par A. Cela signifie que sa reconnaissance s'arrête après avoir parcouru un préfixe u (i.e. x=uav, iuq et $d(q,a)=\emptyset$, $q=(q_1,S)$). Par le lemme2, i_1uq_1 et comme $x \in L_1$, $d(q_1,a) \neq \emptyset$. Donc $G(S,a)=\emptyset$ ce qui signifie que $d(q_2,a)=f_2$ (for a q_2 in S). Par le Lemme 1, il existe une chaîne v' telle que v=v".v' et $i_2v'q_2$, donc $i_2(v'.a)f_2$, par conséquent $v'.a \in L_2$ ce qui contredit le fait que x n'a pas de facteur dans L_2. Par conséquent $x \in L(A)$.]]

[[Soit $x \in L(A)$ ce qui signifie $ix(f_1,S)$ où $f_1 \in F_1$. Par le lemme2, i_1xf_1, donc $x \in L_1$. Supposons que x a un facteur u.a dans L_2, x=x'.u.a.x", donc $ix'(q_1,S_1)u(q_2,S_2)a(q_3,S_3)x""(f_1,S)$. Mais $i_2 \in S_1$ et $i_2uq'_2af_2$, de plus comme A_2 est déterministe, $q'_2 \in S_2$, ce qui implique $G(S_2,a)=\emptyset$ qui contredit $(q_2,S_2)a(q_3,S_3)$. Donc x n'a pas de facteur dans L_2. Donc $x \in L$.]]

5.3.4. L'algorithme

L'algorithme suivant calcule $F(A_1,A_2)$ où A_1 et A_2 sont définis comme ci-dessus et par conséquent cet algorithme calcule $A_1 f\text{-} A_2$.

```
1.f[0] = (i1,{i2})
2.q = 0;
3.n = 1;
4.F = ∅;
5.do
6.    (x1,X2) = f[q];
7.    G = {i2};
8.    for each s ∈ Alph so that d1(x1,s)≠∅
9.       y1 = d1(x1, s);
10.      for each x'∈ X2
11.        if d2(x',s)=f2
12.           G=∅; goto 8;
13.        else
14.           G=G U {d2(x',s)};
15.      endfor
16.      if $q'<=(n-1) so that f[q']=(y1,G)
17.           d(q,s)=q'
18.      else
19.           f[n]=(y1,G);d(q,s)=n; n+=1;
20.           if y1∈F1 then F=F U {n};
21.   endfor
22.           q+=1;
23.while (q<n)
```

Algorithme de calcul de la différence par facteurs A1 f- A2

Figure 5.2g

5.3.3. Comparaison avec l'opération A_1 - *Alph** A_2 *Alph**

Comme $A_1 f\text{-} A_2$ est définie par l'opération A_1 - *Alph** A_2 *Alph** on peut penser que l'application de l'algorithme de différence classique aurait aussi bien pu s'appliquer. Cela appelle deux remarques.

a. L'automate *Alph** A_2 *Alph** est non déterministe et il n'est a priori pas possible de le déterminiser sans connaître *Alph* en entier (ce qui est impossible). On peut pallier cet inconvénient en utilisant des abréviations du type @ qui signifie *tout l'alphabet sauf un certain nombre de lettres* mais cela n'est pas facile à gérer non plus.

b. Il est possible d'utiliser *Alph* A_2 Alph** et de calculer le résultat de manière non déterministe mais on perd en efficacité et le résultat est non déterministe: il y a donc une déterminisation en plus à faire.

5.5. APPLICATION DES GRAMMAIRES LOCALES A LA CONSTITUTION DE LA GRAMMAIRE GENERALE

Les grammaires locales s'appliquent à des phrases. Par ailleurs nous avons vu que les grammaires que nous utilisons sont des listes de phrases. On peut donc appliquer ces grammaires locales aux grammaires générales. Si la grammaire générale est suffisamment précise, le résultat de l'application d'une grammaire locale ne doit rien changer. Mais cette technique peut être utilisée pour la fabrication de grammaires trop complexes pour être décrites directement sous forme d'automates comme nous allons le voir sur deux exemples. Le premier consiste à construire une grammaire des dates telle qu'elle a été faite par D. Maurel 1989, problème que nous avons déjà abordé. Le second exemple est plus particulier, il consiste à reprendre une description de certains déterminants numériques. Dans les deux cas il ne s'agit pas d'affiner la description linguistique, mais de montrer que la notion de grammaire locale permet de mieux comprendre la démarche suivie dans la constitution de ces descriptions.

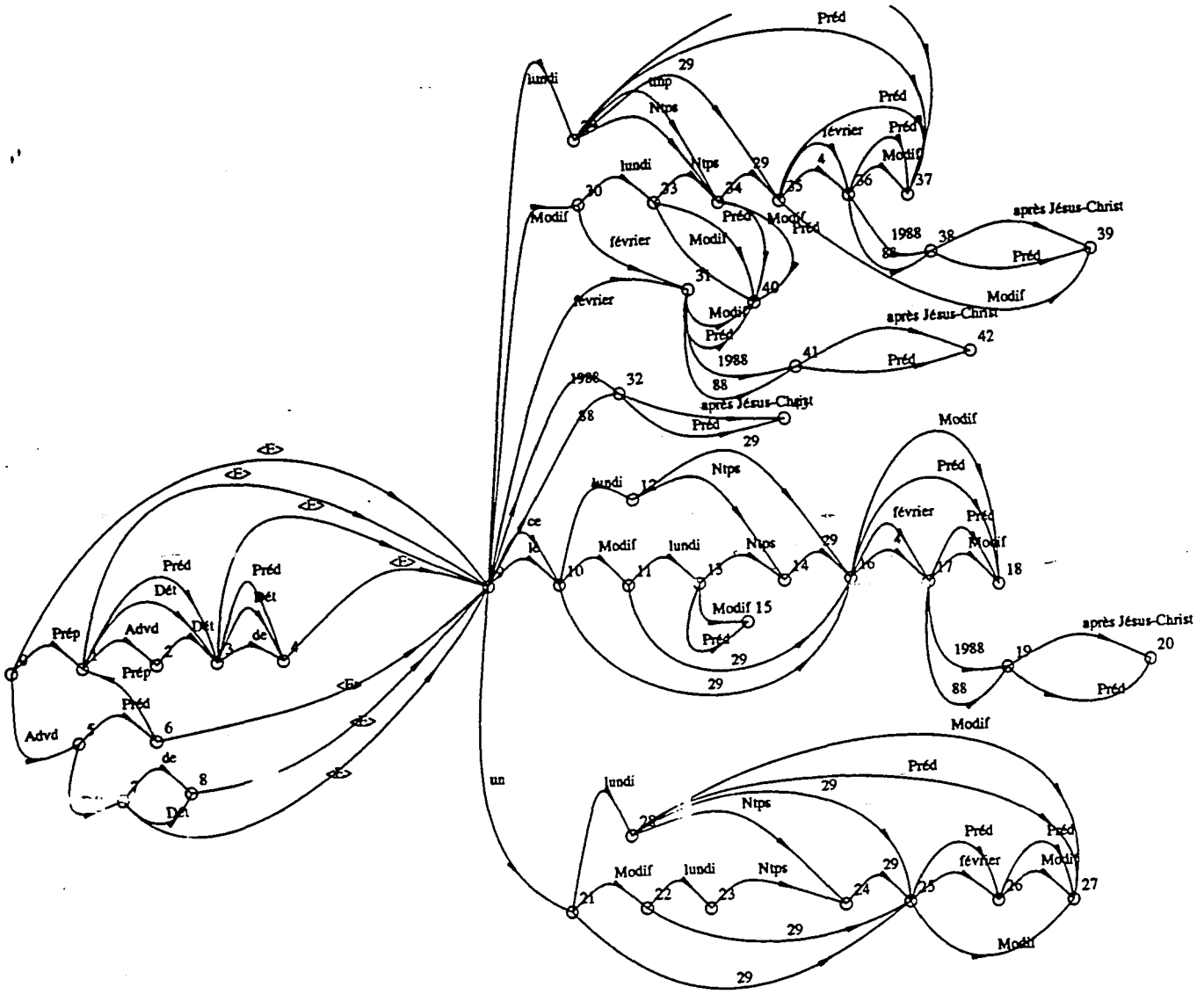
5.5.1. Grammaire des dates

Rappelons brièvement la méthode suivie dans D. Maurel 1989 pour construire un système de reconnaissance des dates, elle se décompose en deux étapes :

1. Description de l'ensemble des formes prises par les différents adverbes dans un automate tel que l'automate $Gram_1$ de la figure 5.5a.

2. Description des contraintes dans la composition des éléments.

L'automate qui décrit les adverbes de dates,et mentionné en 4.18.3, est donné à la figure 5.5a :



Un automate des adverbes de dates *Gram*$_1$ (D.Maurel 1989)

Figure 5.5a

Les contraintes sont données sous forme de matrices binaires telles que celle de la figure 5.5b :

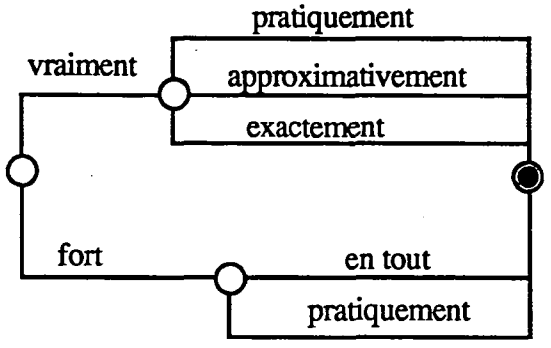
J en tout
K approximativement
L exactement
M pratiquement

	J	K	L	M
vraiment	+	-	-	-
aussi	+	+	+	+
fort	-	+	+	-

Extrait de la matrice de contraintes *Advd(Préd + Dét)* (D. Maurel 1989)

Figure 5.5b

Il existe alors deux méthodes pour utiliser ce couple (*Automate, Contraintes*) dans la reconnaissance des adverbes de dates dans les textes. La première consiste à détecter dans un texte toutes les séquences reconnues par l'automate puis vérifier les contraintes. La seconde consiste à construire un automate équivalent à la matrice en supprimant l'ensemble des chemins interdits par les contraintes. Ces deux méthodes ne sont pas très faciles à mettre en oeuvre car elles combinent deux formalismes sensiblement différents : celui des automates et celui des matrices de contraintes. On peut cependant constater que les matrices de contraintes sont équivalentes à l'automate représentant les séquences *a b* où le couple *(a,b)* est marqué "-" dans la matrice. L'automate décrit donc un ensemble de séquences interdites, une grammaire locale au sens où nous l'avons défini ci-dessus. Ainsi, la matrice de 5.5b est équivalente à l'automate suivant:

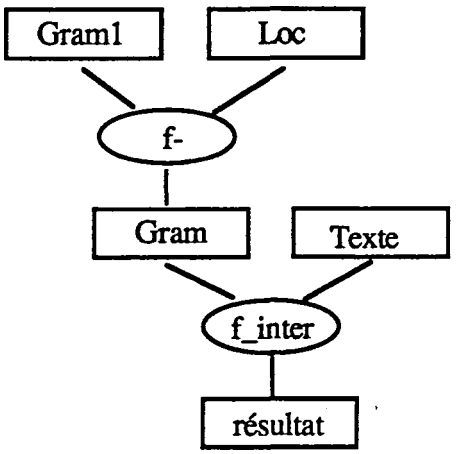


Automate de grammaire locale *Loc* équivalent à la matrice de la figure 5.5b

Figure 5.5c

La grammaire complète des dates *Gram* est alors définie par le couple *($Gram_1$, Loc)* où $Gram_1$ est un automate du type de celui de la figure 5.5a et *Loc* un automate décrivant des séquences interdites. Nous venons de nous ramener au cas des grammaires locales traité précédemment. Deux méthodes sont alors possibles pour utiliser ce couple. On peut tout d'abord calculer

l'automate exact de la grammaire complète *Gram* par la simple application de l'opération *Gram* = $Gram_1$ *f- Loc* puis appliquer *Gram* au texte. Cela peut se résumer par le schéma suivant

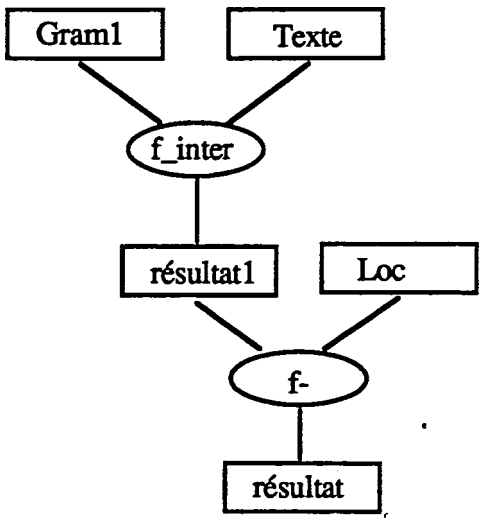


Calcul de *Gram*= $Gram_1$ *f- Loc* puis application au texte

Figure 5.5f

Dans cette figure nous avons représenté la recherche de séquences vérifiant à la fois l'automate $Gram_1$ et les contraintes. L'opération *f-* représente la différence par facteurs décrite ci-dessus, donc l'application de la grammaire locale. L'opération *f_inter* est l'opération de pattern matching sur l'automate représentant le texte, *f_inter* dénotant *intersection par facteurs*.

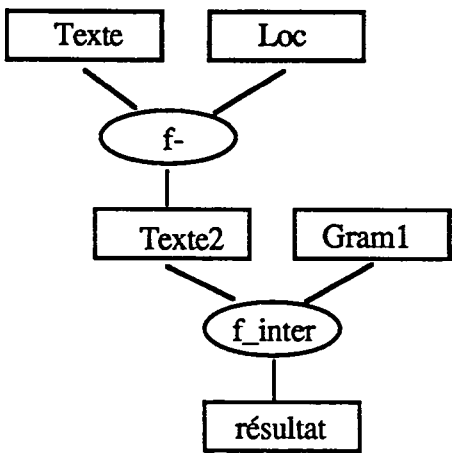
Une autre possibilité est de rechercher d'abord des séquences vérifiant *Gram1* puis d'en filtrer le résultat par la grammaire locale, ce qui correspond au schéma suivant:



Recherche des séquences de $Gram_1$ dans le texte puis filtrage par *Loc*

Figure 5.5g

Enfin, il est possible d'appliquer la grammaire locale de manière similaire à celle du paragraphe précédent, c'est à-dire en appliquant d'abord les contraintes locales au texte et en recherchant dans le résultat les séquences décrites dans $Gram_1$, ce qui se résume par le schéma suivant :



Application des contraintes locales puis recherche des séquences de $Gram_1$

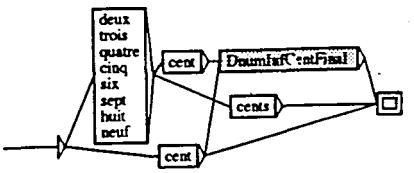
Figure 5.5h

Ces trois méthodes donnent des résultat équivalents, mais elles ont des efficacités différentes. La méthode la plus rapide pour le traitement des textes consiste à calculer d'abord la grammaire complète $Gram = Gram_1$ *f- Loc* (schéma 5.5f), calcul qui se fait indépendamment de tout texte, puis de rechercher *Gram* dans le texte. Il n'y a donc qu'une seule opération de recherche de séquences à faire sur le texte, l'inconvénient peut être que *Gram* occupe en mémoire une place beaucoup plus importante que le couple *(Gram1, Loc)*. Les deux autres méthodes au

contraire ont moins besoin de mémoire, mais deux opérations successives sont nécessaires une fois le texte disponible.

5.5.2. Grammaire des déterminants numériques

Nous avons donné un aperçu en 4.18.3 de la description des déterminants numériques *Dnum* proposée par M. Silberztein 1993. Une des raisons qui rendent la constitution d'un automate général difficile est l'existence de règles orthographiques telles que la mise au pluriel du mot *cent* qui prévoient que *cent* prend un *s* s'il est précédé d'un numéral pluriel (*deux, trois, ...*) et s'il n'est suivit d'aucun nombre. L'automate des nombres permettant de décrire les nombres de 1 à 1000 doit alors avoir la forme suivante :



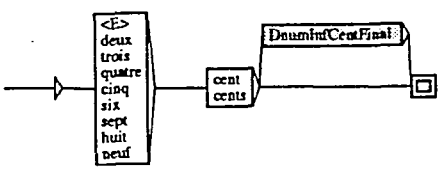
Automate A_1 *DnumCentFinal* (Max Silberztein 1992) décrivant les nombres de 1 à 1000

Figure 5.5i

où *DnumInfCentFinal* est un automate qui décrit les nombres de 1 à 99. En fait cet automate est équivalent à un couple :

(A_2, Loc_2) où A_2 décrit les séquences de A_1 sans tenir compte du nombre *cent* et

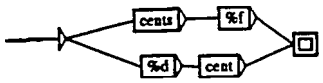
Loc_2 décrit les contraintes. A_2 est alors l'automate de la figure suivante



Automate A_2 ne rendant pas compte du nombre de *cent*

Figure 5.5j

et où l'automate Loc_2 décrivant les contraintes est le suivant :



Automate Loc_2 des contraintes de nombre sur le mot *cent*.

Figure 5.5k

Cette seconde représentation, (A_2, Loc_2) est équivalente à la première puisque $A_1 = A_2 f\text{-}Loc_2$ mais elle a deux avantages :

1. Elle permet de décomposer les problèmes et donc d'être plus sûr de la correction de la grammaire que l'on décrit. C'est particulièrement clair dans le cas des déterminants numéraux puisque l'automate A2 décrit des propriétés syntaxiques naturelles qu'on retrouve à l'oral par exemple alors que la règle sur le nombre de *cent* est purement orthographique (académique ?).

2. Ce dédoublement de la description permet également d'envisager des applications en correction orthographique. L'idée est de faire une analyse n'utilisant que A_2 puis de détecter les séquences de Loc_2. Ces séquences sont alors des déterminants numériques dans lesquelles l'accord de *cent* est incorrect.

Cette méthode permet aussi d'envisager le traitement de propriétés hors de la portée des seuls automates. Prenons le cas de la phrase :

(1) *Luc a acheté de cinq à onze mètres de tissu*

le complément est de structure *Dadv N* où *Dadv* vaut *de cinq à onze*. C'est-à-dire que la structure du déterminant est de $Dnum_1$ *à* $Dnum_2$ avec la contrainte supplémentaire que $Dnum_1$ doit représenter un nombre inférieur à $Dnum_2$. On retrouve cette contrainte dans une construction formellement identique, mais spécifique de certains verbes :

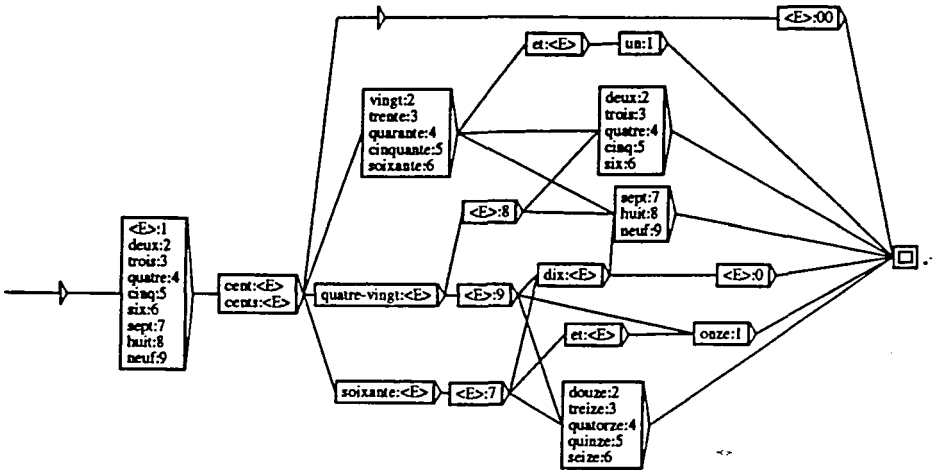
(2) *Le niveau a monté de cinq à onze mètres*

ce qui devient pour *baisser* une contrainte sur les déterminants inverse, comme on peut le voir dans

(3) *Le niveau a baissé de cinq à trois mètres*

Nous arrivons là aux frontières de la syntaxe mais il est cependant instructif de pousser l'exemple à son terme.

Plutôt que de représenter les *Dnum* par des automates nous allons utiliser des transducteurs qui associent aux déterminants numériques *Dnum* le nombre qu'ils représentent. Le transducteur de la figure suivante est une extension de l'automate représentant les *Dnum* :

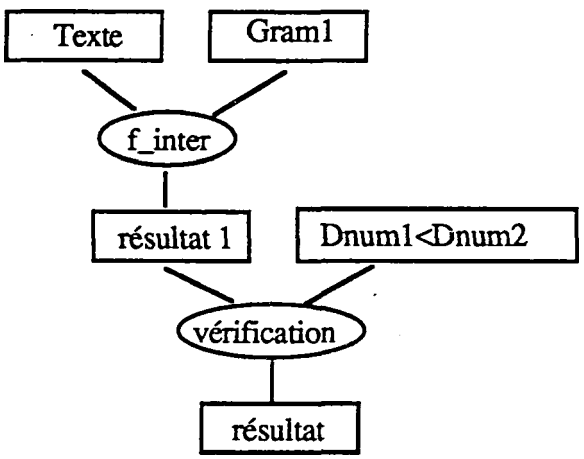


Extrait du transducteur d'association d'une valeur numérique à un *Dnum*

(extension de l'automtate *Dnum* de Max Silberztein 1992).

Figure 5.5l

On peut par exemple vérifier que la séquence *cent quatre-vingt onze* se voit associer la valeur numérique 191. Vérifier la correction d'une phrase telle que (1) consiste donc à appliquer le transducteur à $Dnum_1$ et $Dnum_2$ et vérifier la condition $Dnum_1 < Dnum_2$, ce qui se résume par le schéma de la figure suivante:



Analyse d'une phrase de la même structure que (1)

Figure 5.5m

5.6. TRANSDUCTIONS LOCALES

Nous allons voir qu'il est possible d'étendre la notion de grammaire locale par l'utilisation d'une notion simple de transduction locale.

5.6.1. Définition succincte

On peut définir une transduction de manière locale, c'est-à-dire qu'à partir d'une transduction f, on définit la transduction $F = Loc(f)$ qui, étant donné un mot u, transforme chacun de ses facteurs w tels que $w \in Def(f)$ en $f(w)$ et laisse le reste invariant. Cette définition intuitive et incomplète soulève un grand nombre de problèmes de définition et d'implémentation qu'il n'est pas question de traiter ici. Nous ne donnerons que quelques éléments utilisés de manière très approximative.

Définition

Soit f une transduction de A^* sur A^*, on définit l'*extension locale F de f* et on note $F = Loc(f)$, par :
pour tout $u \in A^*$ et pour toute décomposition de u en facteurs dans A^*
$u = a_1\ b_1\ a_2\ b_2\ ..\ a_n\ b_n\ a_{n+1}$
telle que pour tout i
$b_i \in Def(f)$ (ensemble de définition de f) et
$a_i \notin A^*\ Def(f)\ A^*$
alors pour tout $b'_1, .. b'_n$ tels que $b'_i \in f(b_i)$
$v = a_1\ b'_1\ a_2\ b'_2\ ..\ a_n\ b'_n\ a_{n+1} \in F(u)$

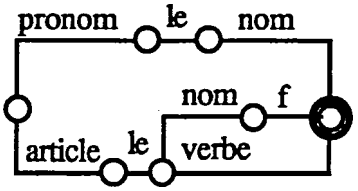
Il faut noter que le transducteur défini par $Id\,f\,Id$ (concaténation au sens des automates où Id est l'identité sur A^*) ne convient pas puisque si $v = f(u)$ alors $u\,v \in (Id\,f\,Id)(u\,u)$ ce qui n'est pas cohérent avec notre définition.

On élargit la possibilité d'écrire des transductions (sans en changer la définition) en ajoutant le symbole d'émission $\varnothing$ qui signifie que le chemin n'a pas d'image. Cela permet de distinguer les chemins qui n'ont pas d'image parce qu'ils ne sont pas décrits dans le transducteur de ceux qui n'ont pas d'image pour des raisons explicitement décrites dans le transducteur.

5.6.2 Application aux grammaires de contraintes locales et généralisations

Cette notion englobe la notion générale de grammaire locale puisque si A_1 est une grammaire de contraintes locales (un ensemble de séquences interdites) et que f_1 est la transduction qui à chaque élément de $L(A_1)$ associe l'ensemble vide alors $Loc(f_1)(phrase) = phrase\ f\text{-}\ A_1$ où *phrase* est l'automate de la *phrase*.

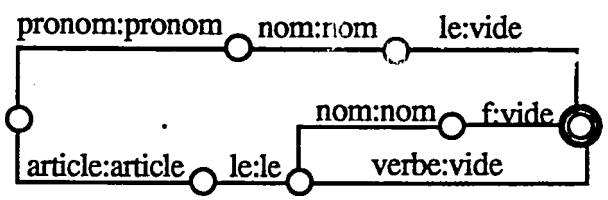
Plus précisément si nous reprenons la grammaire locale de notre exemple (contraintes sur la particule préverbale *le* et sur le déterminant *le*), c'est-à-dire l'ensemble des séquences interdites décrites par l'automate de la figure 5.6a :



***Loc*$_1$: Exemple de grammaire locale sous forme d'automate**

Figure 5.6a

Faire la différence par facteurs de cet automate avec le texte (*phrase f- Loc*$_1$) est équivalent à appliquer le transducteur de la figure 5.6b de manière locale :



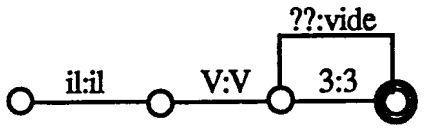
***Loc(Loc*$_1$*)*: Grammaire locale sous forme de transduction**

Figure 5.6b

dans $\varnothing$ laquelle est noté *vide*.

Nous allons maintenant voir que ce formalisme permet aussi d'englober d'autres types de grammaires locales. Prenons par exemple la règle qui consiste à dire que si *il* précède un verbe alors ce verbe est à la troisième personne (on ne précise pas le nombre dans cet exemple), ce type de règle a déjà été utilisée (voir M. Silberztein 1993) mais nous verrons que l'utilisation des transductions locales permet d'inclure dans un même transducteur ce type de règles et les règles d'interdiction présentées précédemment.

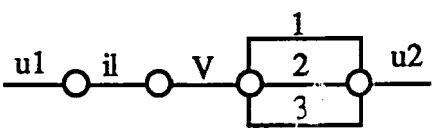
Cette règle peut bien sûr s'écrire de manière négative, c'est-à-dire qu'on peut mettre dans *Loc* l'ensemble des séquences *il V personne* où *personne* est différent de *3*. Plus précisément, il faut inclure dans *Loc* les séquences *il V 2* et *il V 1*. Cette démarche est cependant très peu naturelle et dans certains cas elle est impossible à appliquer car on ne connaît pas l'ensemble des traits qu'il faudrait supprimer. La solution consiste à utiliser le transducteur *Loc*$_2$ de la figure suivante:



***Loc2* : Transducteur de la règle un verbe suivant il est à la troisième personne**

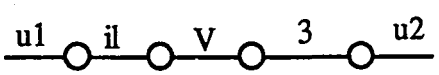
Figure 5.6c

pour lequel on peut vérifier qu'une séquence de type *u il V 2 v* a une image vide alors qu'une séquence de type *u il V 3 w* a pour image $\varnothing \cup \{u\ il\ V\ 3\ w\}$ c'est-à-dire *{u il V 3 w}*. Par conséquent si l'automate A de la figure 5.6d représente le résultat (très simplifié) de l'analyse morphologique d'une phrase, alors l'automate A_2 de la figure 5.6e représente l'application de Loc_2 sur A, autrement dit $A_2 = Loc(Loc_2)(A)$.



***A* : Extrait de l'automate représentant l'analyse morphologique d'une phrase**

Figure 5.6d



A_2 : Résultat de l'application sur A de Loc_2 : $A_2 = Loc(Loc_2)(A)$

Figure 5.6e

5.7 APPLICATION DES GRAMMAIRES DE CONTRAINTES LOCALES A LA CREATION D'INDEX

Nous avons jusqu'à maintenant considéré les grammaires de contraintes locales comme un des modules de l'analyse syntaxique globale. Nous allons voir que l'utilisation indépendante des grammaires locales permet par exemple d'envisager le perfectionnement des outils d'indexation automatique. Les expériences que nous allons donner permettent aussi d'évaluer

linguistiquement l'efficacité de ces grammaires locales.

Supposons que nous voulions enrichir le dictionnaire des mots composés du *DELACF* en dépouillant des textes (textes techniques par exemple) pour y rechercher les mots composés qui n'y sont pas. Pour cela nous nous proposons de rechercher des séquences de la forme syntaxique *Nom Adjectif* telles que *automates déterministes* qui représentent une part importante des noms composés.

Nous nous sommes livrés à l'expérience suivante : nous avons comparé trois programmes de recherche. La premier consiste simplement à consulter le dictionnaire DELAF pour chacun des mots simples du texte et à donner les séquences de deux mots successifs où le premier est un nom et le second un adjectif. Le second programme consiste à ajouter en plus une fonction qui vérifie l'accord en genre et nombre entre le nom et l'adjectif. Le troisième programme consiste à intercaler entre l'analyse morphologique et l'extraction des séquences un module d'application de grammaires locales. On peut résumer ces différentes approches par le schéma de la figure suivante

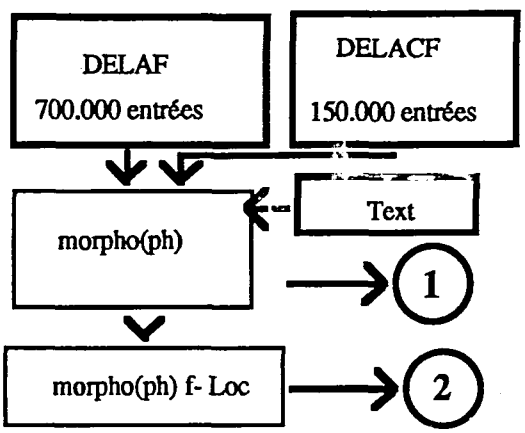


Schéma des programmes d'extraction des séquences N Adj, avant et après application d'une gramm

Figure 5.7a

Si on applique le premier programme à un corpus constitué au LADL-CERIL, on obtient le nombre d'occurrences répertorié à la première ligne du tableau de résultat 5.7b. Si on impose en plus qu'il y ait accord entre le nom et l'adjectif on obtient un nombre de séquences répertorié à la seconde ligne. On constate par exemple que pour le corpus complet plus de trois mille séquences ne vérifiaient pas l'accord genre nombre. Cela permet donc déjà d'éliminer un certain nombre de séquences incorrectes. Dans le troisième programme, qui correspond au cercle 2 de la figure 5.7a, on a appliqué la grammaire locale au texte avant d'y rechercher les séquences *Nom-Adjectif* (on a par exemple supprimé dans la représentation morphologique de la phrase des séquences du type *le(pro)-Nom*) On remarque alors que plus de la moitié de l'ensemble des séquences a été éliminé. La grammaire locale étant exacte, il n'y a pas de règles heuristiques, toutes les séquences éliminées étaient linguistiquement incorrectes, on est donc sûr qu'aucune séquence pertinente n'a été éliminée. En comparant le nombre de séquences trouvées avec grammaire locale avec le nombre correct (compté à la main pour les trois premiers textes et

indiqué à la ligne 4) on constate qu'on se rapproche très fortement d'une sortie exacte. Pour affiner ce résultat il faut appliquer l'analyse syntaxique générale. Cependant, pour ce type d'application, il peut être intéressant de se limiter à une grammaire locale qui s'applique de manière beaucoup plus rapide et qui donne déjà des résultats utilisables.

	Editorial	Article	Roman	Corpus
en octet	4185	13010	369292	1738115
en pages	1	4	100.	
N Adj	66 15"	227 40"	3334 19'	31532 1h25
N Adj accord	56 15"	198 40"	2651 19'	28234 1h25
N Adj acc.+Loc	13 20"	40 50"	1277 41'	11125 3h05
N Adj Nombre réel	10	34	1150	non compté

Recherche des séquences *N Adj* avec et sans grammaire locale

Figure 5.7b

5.8 PROJECTIONS DE LA GRAMMAIRE GENERALE EN DES GRAMMAIRES LOCALES

Nous avons vu que les grammaires locales sont intéressantes en tant que telles, c'est-à-dire indépendamment d'une analyse syntaxique générale, il est donc utile de chercher à les rendre le plus efficace possible. Pour cela deux méthodes sont possibles: soit les construire entièrement à la main (sous forme d'automates ou de transductions locales), soit utiliser la grammaire générale (donc *DELSYN* et *f_DELSYN*) et en déduire des règles qui s'appliquent localement. C'est cette seconde approche que nous allons présenter ici.

Supposons que nous recherchions dans un texte les séquences du type "verbe suivi du premier complément essentiel". Dans une phrase telle que

(1) *Luc dit à Paul qu'il va partir*

nous voulons extraire la séquence *dit à Paul* alors que dans la phrase

(2) *Luc mange dans la cuisine*

nous ne cherchons pas à reconnaître le complément de lieu *dans la cuisine* qui n'est pas spécifique au verbe. Rechercher ces couples verbe-complément peut être intéressant pour la recherche d'informations dans les textes techniques. Il est en effet bien plus précis de pointer vers des séquences structurées du type *appliquer un transducteur* que vers des couples de mots non ordonnés : (*appliquer, transducteur*) qui ne donnent qu'une idée vague du sujet du texte et

le plus souvent une idée fausse.

Pour cela il faut disposer, pour chaque verbe, d'une description de l'ensemble de leurs compléments essentiels. Cette information est contenue dans *DELSYN*, il suffit donc de l'en extraire. On peut procéder comme suit:

1. On écrit un petit automate A_1 qui contient par exemple des chemins du type *verbe (E+Prep) N ??* dans lesquels aucun élément lexical n'est spécifié. (figure 5.8b).

2. On recherche cet automate dans l'automate de *DELSYN* (intersection par facteurs).

3. On fait l'union de tous les résultats de ces recherches en un seul automate Loc_1 qui décrit une grammaire de contrainte locale spécifique.

4. On utilise cette grammaire locale comme précédemment pour rechercher les séquences qui nous intéressent.

Ceci peut se résumer dans la figure suivante:

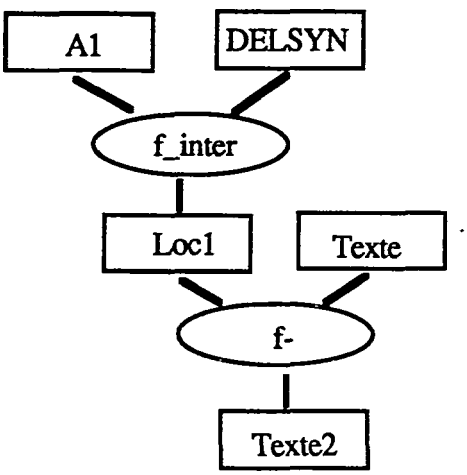
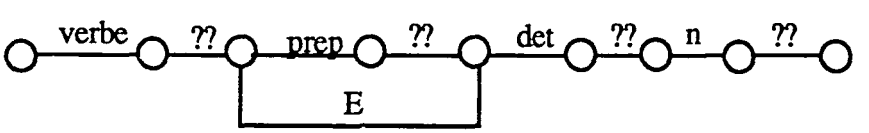


Schéma de l'utilisation d'une projection de *DELSYN*

Figure 5.8a



Automate A_1 de rechercher dans l'automate du dictionnaire syntaxique *DELSYN*

Figure 5.8b

6. IMPLEMENTATION DES PROGRAMMES

Nous ne décrirons pas tous les détails de l'implémentation qui se trouvent dans l'annexe :*PUPITRE*, Manuel de l'utilisateur, et plus particulièrement dans l'annexe 1 de ce manuel. Nous donnons ici un aperçu formel de l'application d'un cycle d'analyse, c'est-à-dire de l'application d'une transduction

6.1 ORGANISATION GENERALE DU PROGRAMME

L'application du programme d'analyse peut se résumer par le schéma suivant:

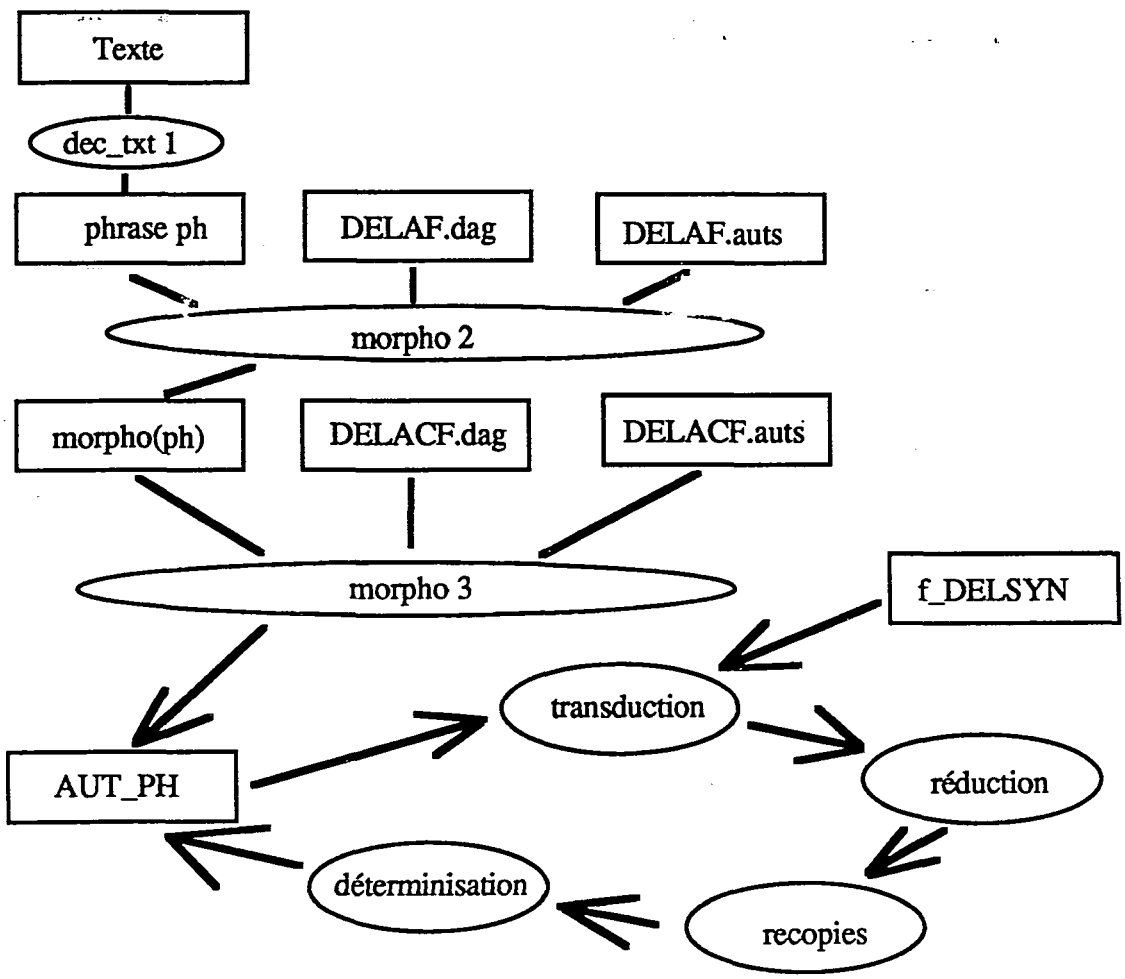


Schéma général du programme d'analyse syntaxique

Figure 6.1a

La première étape, notée *dec_txt 1* (qui correspond à la commande *_dec_txt* de *PUPITRE*) effectue un prédécoupage en phrases comme indiqué en 3.6.

La seconde étape, *morpho 2*, fournit l'analyse morphologique des mots simples d'après le dictionnaire *DELAF* comme décrit en 3.2. Cela consiste donc à utiliser le DAG *DELACF.dag* qui décrit l'ensemble des mots et leur associe le numéro d'un des automates répertorié dans la liste d'automates de mots DELACF.auts. Le résultat de cette analyse morphologique est l'automate *morpho(ph)* qui représente l'ensemble des ambiguïtés morphologiques.

La troisième étape, *morpho 3*, enrichit l'analyse morphologique précédente de la reconnaissance des mots composés. Cette reconnaissance se fait de la manière décrite en 3.3, elle consiste à rechercher l'informations stockée pour chaque entrée, sous la forme d'un automate de *DELACF.auts* à partir du DAG (à deux niveaux: voir 3.3) *DELACF.dag*.

A partir de cette étape, l'analyse morphologique peut être lancée. On va appliquer le cycle de transductions sur l'automate de la phrase *AUT_PH* jusqu'à obtenir un point fixe. Chaque cycle opère de la manière suivante :

1. On applique la transduction *f_DELSYN* à l'automate de la phrase.

2. Une réduction du résultat détruit les états non coaccessibles.

3. On recopie les zones indiquées par *_pi* comme indiqué en 2.4.

4. On effectue une déterminisation du résultat.

La dernière étape pourrait être omise, mais elle permet une recherche d'erreurs éventuelles plus aisée comme nous le verrons en 6.5.

Pour améliorer l'efficacité de l'analyse, on peut réduire le nombre d'ambiguïtés avant le début de l'analyse syntaxique par l'utilisation d'une grammaire locale. L'application de cette grammaire locale apparaît donc après l'analyse morphologique et avant le premier cycle d'analyse syntaxique. Ce processus est résumé dans le schéma de la figure 6.1b qui résume également l'état actuel de l'implémentation.

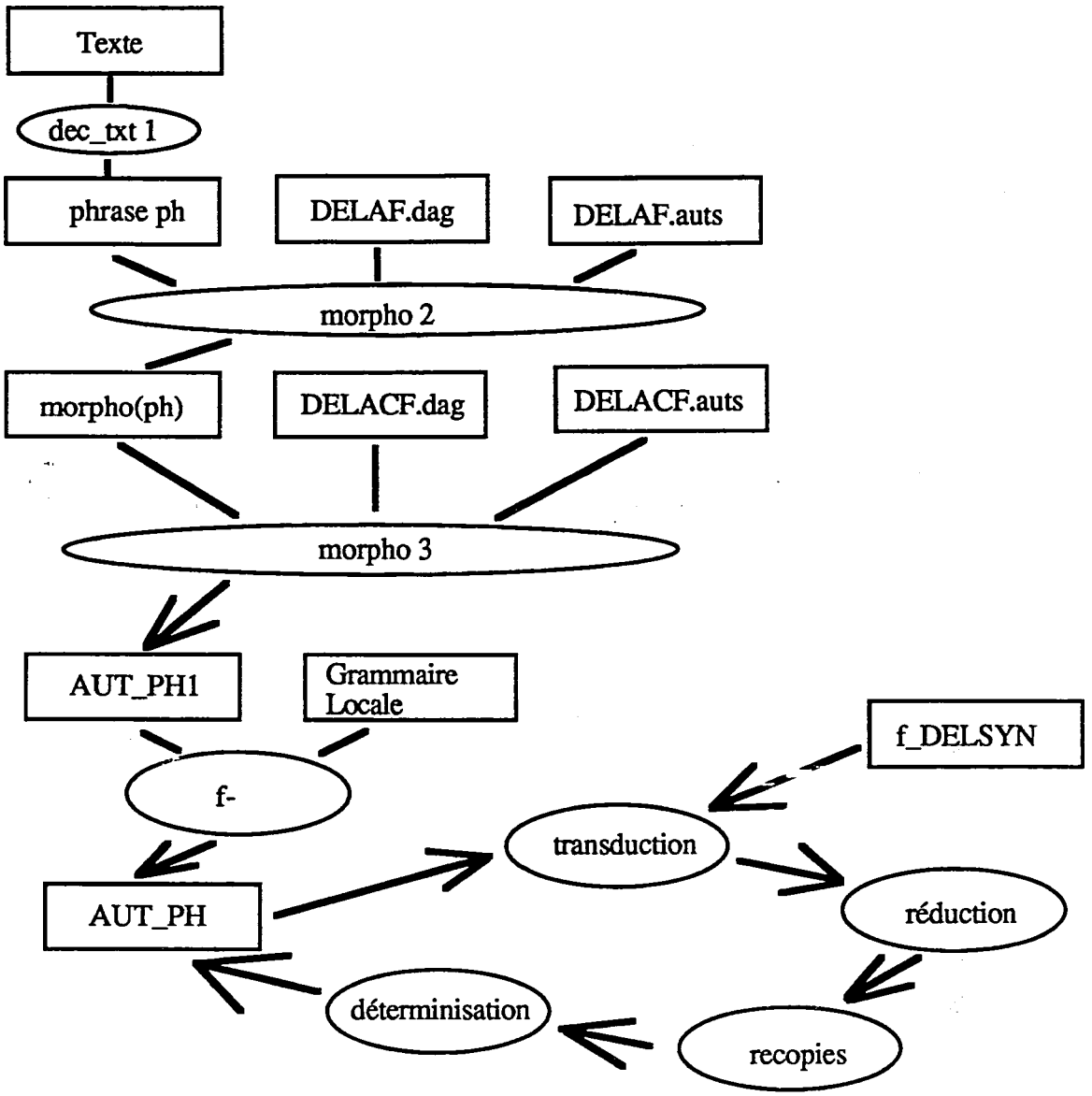


Schéma général de l'analyse syntaxique avec grammaire locale

Figure 6.1b

Nous avons vu qu'une généralisation des grammaires locales amène à considérer l'application de transductions locales à l'intérieur de chaque cycle d'analyse, nous avons représenté cette possibilité sur la figure 6.1c.

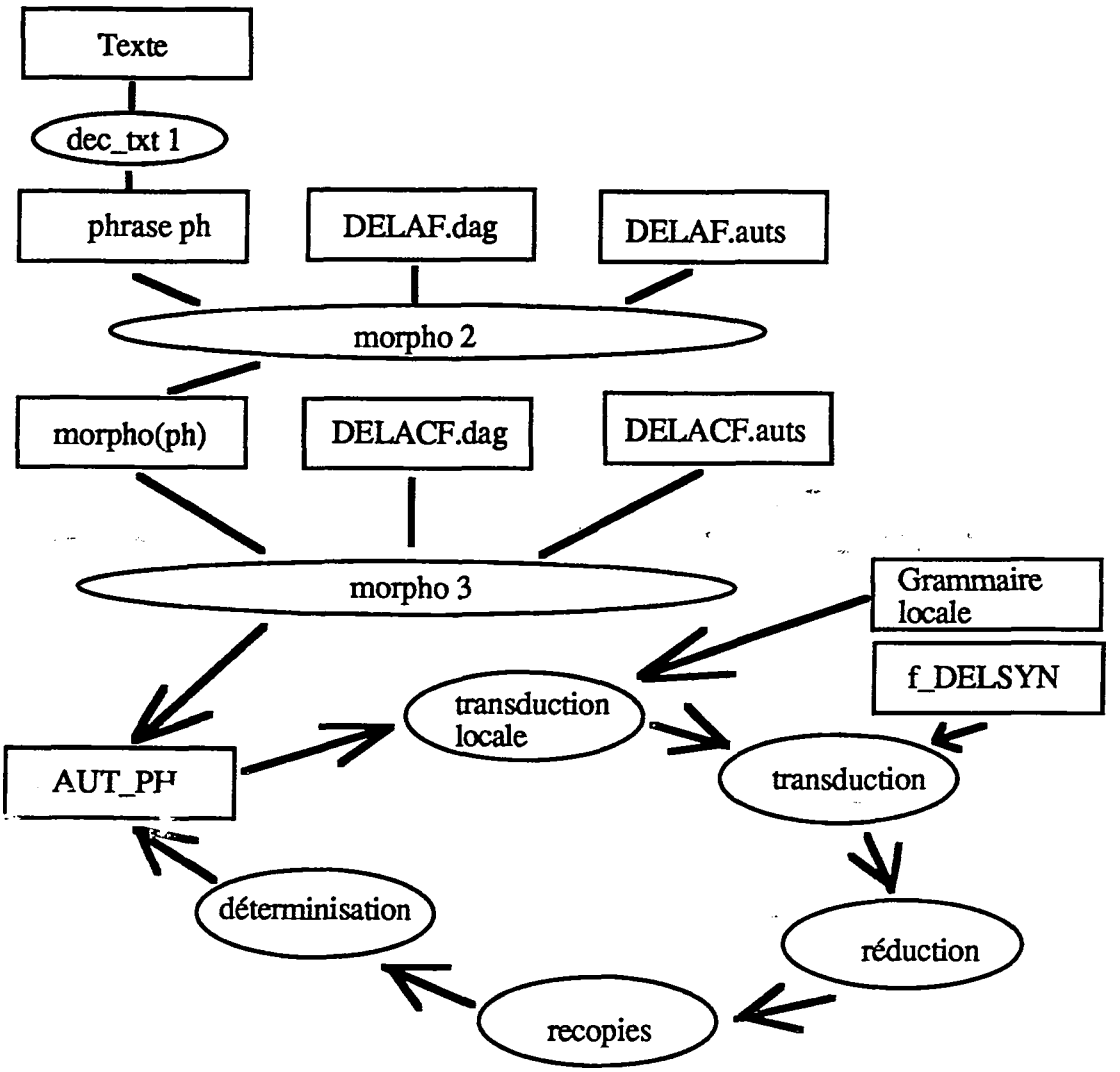


Schéma général de l'analyse syntaxique avec grammaire locale dans le cycle

Figure 6.1c

6.2 APPLICATION DE LA TRANSDUCTION

Appliquer une transduction T à un automate A correspond très exactement à calculer l'intersection de la projection gauche $g(T)$ de T avec A et à noter les labels droits au fur et à mesure. Cela s'énonce de la manière plus précise suivante :

On définit tout d'abord, comme en 3.4.1, l'opérateur *transduc* sur les automates qui à un automate A associe le transducteur *transduc(A)* qui à tout élément u de $L(A)$ associe A^*.
Soit T un transducteur et A un automate, on définit alors l'application de T sur $T(A)$
par $A_2 = T(A) = d(T \cap transduc(A))$
et on a bien $L(A_2) = T(L(A))$.

L'algorithme du calcul de $T(A_1)$ est alors le suivant:

Donnée: $A_1 = (Alph, Q_1, 0, F_1, d_1)$ et $T = (Alph, Q_2, 0, F_2, d_2)$
On calcul alors $A = T(A_1) = (Alph, Q, (0,0), F, d)$ où $Q = Q_1 * Q_2$

```
1.    q=0;
2.    Q[0]=(0,0);
3.    n=1;
4.    do{
5.          (q1,q2)=Q[q];
6.          si (q1∈ F1) et (q2∈ F2) alors
7.                Q[q]∈F;
8.          pour tout s∈ Alph tel que d2(q2,(ε,s))≠∅ alors pour tout
q'∈ d2(q2,(ε,s))
9.          {
10.               si ∃p<n tel que Q[p]=(q1,q')
11.                     d(Q[q],s)=Q[p];
12.               sinon
13.                     Q[n]=(q1,q');
14.                     d(Q[q],s)=Q[n];
15.                     n++;
16.         }
17.         pour tout s,s1∈ Alph tels que d1(q1,s)≠∅ et d2(q2,(s,s1))≠∅
18.         pour tout q1'∈ d1(q1,s) et q2'∈ d2(q2,(s,s1))
19.         {
20.               si ∃p<n tel que Q[p]=(q1',q2')
21.                     d(Q[q],s)=Q[p];
22.               sinon
23.                     Q[n]=(q1',q2');
24.                     d(Q[q],s)=Q[n];
25.                     n++;
26.         }
27.         q++;
28.   } while (q<n);
```

Cet algorithme appelle deux remarques importantes:

a. Pour tester les conditions des lignes 10 et 20 nous avons utilisé des fonctions de hachage. En principe, d'autres structures, telles que les arbres binaires, ou d'autres types d'arbres, pourraient aussi être employés, mais des tests montrent que ces solutions sont la plupart du temps trop gourmandes en mémoire et, par ailleurs les nombreuses allocations et désallocations successives ralentissent notablement l'algorithme.

b. Comme pour une intersection, chaque boucle consiste principalement à comparer les transitions d'un état du premier automate avec les transitions d'un état du second automate (ici le transducteur). Pour avoir une efficacité maximale, il faut donc utiliser, non pas des représentations en temps constant (à accès direct) mais une représentation en liste et une représentation en temps constant.

Développons cette seconde remarque. La comparaison entre un état q_1 de A_1 et un état q_2 de A_2

(ou T) consiste essentiellement à faire une comparaison du type suivant :

Pour chaque transition $d_1(q_1,s)$ de q_1 telle que $d_2(q_1,s)\neq\emptyset$ faire *quelque chose*,

c'est-à-dire regarder la LISTE des transitions de q_1 et à regarder, pour chacune d'entre elles si elle correspond à une transition de q_2. Il est par ailleurs clair qu'il n'est pas possible d'utiliser un tableau de transitions pour une représentation en temps constant puisque la taille de l'alphabet est bien trop importante (de l'ordre du million). Nous donnons dans l'annexe 1 du manuel de *PUPITRE* : *Notions de base sur les automates à états finis et leur représentation*, la possibilité de représenter ce type d'automate avec néanmoins un accès en temps constant. Cette dernière représentation a l'inconvénient d'être très rigide, il n'est pas possible d'y ajouter des transitions, elle ne peut donc être utilisée que pour la grammaire, c'est-à-dire *f_DELSYN*.

6.3 RECHERCHE D'ERREURS DANS LA GRAMMAIRE *DELSYN*

La recherche d'erreurs dans la grammaire se fait essentiellement de la manière suivante: on teste des phrases dont on attend une analyse et si l'analyse échoue ou propose des analyses incorrectes on génère les automates intermédiaires à chaque étape du processus comme indiqué sur les figures 6.1a 6.1b et 6.1c. Une fois ces automates générés on peut les visualiser par le viewer du programme *pauto* de *PUPITRE*. Ce programme commence par afficher l'état initial avec toutes ses transitions et donc tous les états vers lequel il pointe. On développe les états affichés, c'est-à-dire qu'on affiche leurs transitions en cliquant dessus. De cette manière on peut suivre les chemins qui semblent poser problème de manière rapide. On pourra se reporter à la description du programme *pauto* de *PUPITRE*.

6.4 VISUALISATION DES RESULTATS

Les automates qui sont le résultat de l'analyse comportent toutes les informations nécessaires : tous les traits morphologiques, toutes les références aux structures de DELSYN, et toutes les transformations appliquées au cours de l'analyse. Par conséquent il peut être difficile d'évaluer du premier coup d'oeil le résultat d'une analyse. La solution consiste à utiliser un type d'opération que nous appelerons projection (qui s'appliquent par la commande _proj de PUPITRE). Une transduction définit une projection de la manière suivante :

Si T est un transducteur et A un automate alors la projection de A par T est la liste des chemins de $T(A)$. Concrètement la projection $proj(T)(A)$ se calcul de la manière suivante :

1. Calcul de $A_2 = T(A)$
2. Déterminisation de A_2
3. Parcours de A_2

Si T est par exemple la transduction qui ne conserve que les mots d'origine et traduit les symboles de début d'argument en "(" et de fin d'argument en ")", on obtient l'analyse parenthésée de la phrase. Nous avons ainsi dessiner un nombre réduit de transductions prédéfinies qui permettent divers examens des résultats de l'analyse.

6.5 ETAT ACTUEL DE L'IMPLEMENTATION

Le programme *_analyse* de *PUPITRE* qui effectue l'analyse syntaxique suit le schéma de la figure 6.1b. Tout les exemples traités dans cette étude sont effectivement incorporés dans *DELSYN* et *f_DELYN*. Par conséquent tous les exemples sont analysés par le programme.

6.5.1 Taille du dictionnaire syntaxique

Nous avons donnés avec les extraits de *DELSYN* des paragraphes 4.1 à 4.9 l'évolution de la taille de ce dictionnaire. Le nombre d'éléments de ce dictionnaire dépend des choix faits au cours de sa construction.

Le dictionnaire syntaxique tel que nous l'utilisons actuellement et qui reflète les choix faits lors de cette présentation est représenté par un automate de 19.700 états et de 75.200 transitions. Cet automate a plus de 110.000.000 de chemins, chacun représentant une phrase différente. Ce nombre très élevé vient du très haut degré de lexicalisation et du fait que ce dictionnaire contient explicitement les phrases avec négations, pronominalisation, temps composés et verbes modaux. Si on n'applique aucune de ces opérations à l'intérieur du dictionnaire on obtient un dictionnaire *DELSYN* représenté par un automate 12.000 états et 51.000 transitions représentant 2.100.000 chemins. Ce dernier nombre est intéressant puisqu'il représente le nombre de manières différentes d'appliquer un opérateur au sens de Z. Harris. Ce nombre représente également le nombre de règles qu'il faudrait implémenter dans un système d'unification équivalent.

6.5.2 Temps d'exécution

Il est difficile de mesurer et de comparer les temps d'analyse tant les résultats sont différents. De plus nous n'avons pas encore utilisé la possibilité esquissée en 6.2 qui consiste à avoir un accès aux transitions en temps (presque) constant. Nous donnons ici des temps mesurés pour certains exemples. Des expériences beaucoup plus complètes restent à faire.

Les temps d'exécution que nous donnons comprennent l'analyse morphologique et syntaxique ainsi que la projection des résultats sous la forme d'expressions parenthésées. Le programme utilise les données suivantes :

- Le dictionnaire morphologique *DELAF* comprenant 700.000 entrées représenté par un automate de 80.000 états et 175.000 transitions occupant 1MO.

- Le dictionnaire morphologique *DELACF* des mots composés de 150.000 entrées représenté par un automate à deux niveaux occupant 2MO.

- L'ensemble des grammaires locales représenté par un automate occupant 400KO.

- Le dictionnaire *DELSYN* ayant les caractéristiques données en 6.5.1 occupant 500KO.

On obtient alors les temps d'analyse qui suivent :

Pierre agace Jean
Temps d'analyse : 0,16s[15]

Pierre est agacé de ce que Jean ne lui avait pas clairement dit ce qu'il voulait.
Temps d'analyse : 0,8s

Il fut très agaçant pour Paul de ne pas entendre distinctement ce que le professeur disait à une assemblée pourtant silencieuse.
Temps d'analyse : 1,7s

Depuis l'arrivée du bataillon canadien qui était chargé d'assurer la protection rapprochée de l'aéroport, de nombreux avions-cargos de différentes nationalités y atterrissent quotidiennement.
Temps d'analyse : 2,1s

Ces résultats appellent deux remarques :

- Des codages différents des transductions et une implémentation plus efficace des programmes devraient permettre (le travail reste à faire) de gagner un facteur de temps important (au moins 10 et probablement beaucoup plus).

- Nous n'avons présentés ci-dessus que des phrases où l'analyse a réussit, mais la couverture de la grammaire n'est pas encore complète. En particulier un nombre important de tables d'expressions figées et de tables de constructions à verbe support sont à incorporer à *DELSYN*. Par ailleurs certains problèmes difficiles dus aux conjonctions sont linguistiquement non résolus. Par conséquent l'analyse est encore incomplète pour un grand nombre de phrases.

[15]Temps mesurés sur un NeXT-Cube, processeur 68040 à 33Mhz avec 32MO de mémoire vive.

7. CONCLUSION

On aura pu constater qu'une description linguistique détaillée conduit naturellement à un procédé simple d'analyse automatique. Le programme d'analyse se réduit à l'application répétée d'un transducteur équivalent à un dictionnaire syntaxique, à une liste de structures simples.

Cette étude garantit qu'il est possible d'utiliser des données d'un volume important alors que la plupart des analyseurs proposés à ce jour n'opèrent que sur de petits échantillons de dictionnaire et de grammaire, sans aucune garantie que le passage à une couverture importante de texte soit réalisable avec les mêmes programmes.

La couverture du dictionnaire syntaxique est destinée à être augmentée dans une large part et il faudra établir si certains phénomènes syntaxiques plus complexes tels que la conjonction s'adaptent au formalisme présenté ici. Il n'existe aujourd'hui pas de réponse à ce problème, les faits linguistiques étant encore à établir.

L'application des transformations syntaxiques montre qu'il est possible d'utiliser ce formalisme pour la génération de textes. Par ailleurs des outils de traduction ou d'aide à la traduction automatique pourraient consister à écrire un dictionnaire syntaxique *DELSYN* bilingue. Des problèmes nouveaux seront à résoudre mais dans le cas des phrases simples où deux structures sont directement mises en relation, la traduction ne requiert pas d'informations linguistiques nouvelles. Soulignons qu'un tel dictionnaire suppose une précision de description d'un ordre de grandeur très supérieur à ce qui est disponible et même à ce qu'on envisage généralement aujourd'hui. Il ne s'agit plus de donner une traduction pour chaque mot simple (chaque verbe, chaque nom, etc.) mais pour chaque structure de phrase de ces mots, cela conduit à des listes d'une taille plus importante.

8. REFERENCES

Abeillé, Anne, 1988. *Parsing French with a Tree Adjoining Grammar*, Actes 12° COLING, Budapest.

Abeillé, Anne, 1991. *Une grammaire lexicalisée d'arbres adjoints pour le français,* Application à l'analyse automatique. Thèse de Doctorat de linguistique. Université Paris 7. Paris.

Aho, Alfred V., John E Hopcroft, Jeffrey D. Ullman, 1974. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 470p.

Aho, Alfred,Ravi, Sethi,Jeffrey D. Ullman, 1989. *Compilateurs Principes, techniques et outils,* Paris : Interéditions.

Berstel, Jean, 1979. *Transductions and Context-Free Languages*, Stuttgart, B.G.Teubner 277p.

Boons, Jean-Paul ; Alain Guillet ; Christian Leclère 1976a. *La structure des phrases simples en français. I Construction intransitives,* Genève : Droz, 377p.

Boons, Jean-Paul ; Alain Guillet ; Christian Leclère. 1976b. *La structure des phrases simples en français. II Construction transitives*: Rapport de recherche du LADL, N°6, 85p.,tables et index, 58p. Paris :Université Paris 7.

Champarnaud, Jean Marc, 1985. *Un système de manipulation des automates finis*, Paris : Publication du LITP,Paris: Université Paris 7: Paris.

Church, Kennteh, 1988. A stochastic parts program and noun phrase parser for unrestricted text, Actes 2° International Conference on applied Natural Language Processing,Austin.

Church, Kenneth, William Gale, Patrick Hanks, Donald Hindle, 1989. *Parsing, Word Associations and Typical Predicate-Argument Relations*. Internal report. Bell Laboratories, Murray Hill.

Clemenceau, David, 1992. *Problèmes de couverture lexicale.* Langue Française, *Productivité et créativité lexicale*, Paris.

Clemenceau, David, Emmanuel Roche 1992. *Enhancing a large scale dictionary with a two-level system.* Technical Report. Institut Gaspard Monge, Marne-la-Vallée.

Clemenceau, David, 1993. *Structuration du lexique et reconnaissance de mots dérivés dans les textes* Thèse de doctorat. Université Paris 7; Paris.(à paraître).

Courtois, Blandine, 1984, 1989. *DELAS : Dictionnaire Electronique du LADL pour les mots simples du français*, Rapport technique du LADL,Paris: Université Paris 7.

Danlos, Laurence, 1988. *Les expressions figées construites avec le verbe Etre Prep*, Languages

n°90.
Giry-Schneider, Jacqueline, 1978. *Les prédicats nominaux en français, les phrases simples à verbe support.* Genève-Paris Droz 396p.

Giry-Schneider, Jacqueline, 1987. *Les nominalisations en français, l'opérateur* faire *dans le lexique.* Genève-Paris Droz 340p.

Gross, Maurice, 1968. *Grammaire transformationnelle de français : 1) Syntaxe du verbe,* Paris : Larousse Cantilène, 183p.

Gross, Maurice, 1975. *Méthodes en syntaxe,* régime des constructions complétives, Paris : Hermann, 415p.

Gross, Maurice, 1977. *Grammaire transformationnelle de français : 2) Syntaxe du nom,* Paris : Larrouse Cantilène, 255p.

Gross, Maurice, 1986. *Grammaire transformationnelle de français : 3) Syntaxe de l'adverbe,* Paris : Cantilène, 669p.

Gross, Maurice, 1988. *La construction de dictionnaires électroniques.* Annales des télécommunications, tome 44, CNET.

Gross, Maurice, 1989a. *Les expressions figées. Une description des expressions françaises et ses conséquences théoriques.* Rapport du programme de recherches coordonnées, Informatique Linguistique. Université Paris 7, LADL. Paris, 109p.

Gross, Maurice, 1989b. *The use of finite automata in the lexical representation of natural language.* Electronic Dictionaries and Automata in Computational Linguistics. LITP Spring School on Theoritical Computer Science. Springer Verlag, Berlin-Heidelberg.

Guillet, Alain ; Christian Leclère 1981. *Restructuration du groupe nominal,* Formes syntaxiques et prédicats sémantiques, A. Guillet et C. Leclère éds., Langages N° 63, Paris : Larousse, pp 99-125.

Guillet, Alain ; Christian Leclère 1992. *La structure des phrases simples en français. Verbes à complément direct et complément locatif,* Genève : Droz.

Harris, Zellig, 1976. *Notes du cours de syntaxe,* Seuil, Paris

Harris, Zellig, 1991. *Theory of Language and Information.* Oxford University Press.

Jayez, J-H, 1982. *Compréhension automatique du langage naturel, le cas du groupe nominal en français,* Masson.

Joshi, A 1987. *Introduction to Tree Adjoining Grammar.* A. Manaster Ramer (ed), *The Mathematics of Language,* J. Benjamins.

Julia, Jean-Thierry, 1991. *Reconnaissance par automate du système de la négation en français.* Rapport de DEA. Université Paris 7 : Paris

Karttunen, Lauri; Ronald M. Kaplan, Annie Zaenen, 1992. *Two-level Morphology with Composition.*COLING-92. Proceedings of the Conference, Nantes.

Kay, Martin, 1985. *Parsing in functional unification grammar.*Studies in Natural Language Processing, Natural Language Parsing (p251-278). Cambridge University Press. Cambridge.

Karttunen, Lauri ; Martin Kay, 1985. *Parsing in a free word order.* Studies in Natural Language Processing, Natural Language Parsing (p279-306). Cambridge University Press. Cambridge.

Koskenniemi, Kimmo, 1990. *Finite-State Parsing and Disambiguation.* Coling-90. Proceedings of the conference. Helsinki.

Leclère, Christian, 1990. *Organisation du Lexique-Grammaire des verbes français.* Langue française 87. Dictionnaire électronique du français. Paris : Larousse.

Liang, Franklin Mark, 1983. *Word Hyphenation by Computer*, Stanford : report No STAN-CS-83-977.

Maurel, Denis, 1989. *Reconnaissance de séquences de mots par automate, Adverbes de date du Français.* Thèse de doctorat, Université Paris 7, Paris.
Harris, Zellig,1971. *Structures mathématiques du langage.* Dunod, Paris, 243p.

Miller, Philip, Thérèse Torris, 1990. Formalismes syntaxiques pour le traitement automatique du langage naturel. Hermès, Paris, 359p.

Meunier, Annie, 1981. *Nominalisations d'adjectifs par verbes supports.* Thèse de doctorat. Université Paris 7, Paris.

Mohri, Mehryar, 1993. Analyse et représentation par automates de structures syntaxiques composées. Thèse de Doctorat, Université Paris 7, Paris (A paraître).

Peireira, Fernando C.N. , Rebecca N. Wright. 1991. *Finite state approximation of phrase structure grammars*, 29th Meeting of the A.C.L, Proceedings of the conference. University of California, Berkeley.

Revuz, Dominique, 1991. *Dictionnaires et lexiques, méthodes et algorithmes.* Thèse de doctorat, Paris : Université Paris 7.

Roche Emmanuel, 1990. *Une représentation par automate fini des textes et des propriétés transformationnelles des verbes.* Rapport de DEA. A paraître dans Linguisticae Investigasiones 1992/2 Amsterdam.

Roche Emmanuel, 1992a. *Text disambiguation by finite state automata, an algorithm and experiments on corpora.* COLING-92. Proceedings of the Conference, Nantes.

Roche Emmanuel, 1992b. *Looking for syntactic patterns in texts.* COMPLEX'92. Papers in Computational Lexicography, Budapest.

Roche Emmanuel, 1992c. *Fast indexing methods for French texts.* ICEBOL 92. Proceedings

of the conference, Dakota State University, Madison.

Roche Emmanuel, 1992d. *Dictionary Compression Experiments*. Rapport Technique Institut Gaspard Monge.

Salkoff, Morris, 1973. *Une grammaire en chaîne du français, analyse distributionnelle*. Monographies de Linguistique Mathématique, Dunod, Paris 197 pages.

Salkoff, Morris, 1990. *Translation of Support Verb Constructions*. COLING-90. Proceedings of the Conference, Helsinki.

Salminen Airi; Frank W.M. Tompa, 1992. *PAT expressions: an algebra for text search*. COMPLEX'92. Papers in Computational Lexicography, Budapest.

Schabes, Y., A. Joshi 1988. *An Erley-type Algorithm for Tree Adjoining Grammars*, Actes 26° annual meeting of ACL, Buffalo.

Schabes, Yves 1990 *Mathematical and computational aspects of lexicalized grammars*, Thèse de PhD, Université de Pennsylvanie, Philadelphie.

Sedgewick, Robert, 1988. *Algorithms*. Addison-Wesley, 657p.

Silberztein, Max ,1989. *Dictionnaires électroniques et reconnaissance lexicale automatique*. Thèse de doctorat, Paris : Université Paris 7.

Silberztein, Max ,1993. *Dictionnaire électroniques et analyse automatique de textes : le système INTEX*. Masson, Paris.

Tarjan R.E., Yao,1979. *Storing a sparse table*. CACM 22.

Vasseux, Philippe, 1979. *Le système LEXSYN de gestion des données lexcico-syntaxiques du LADL*, Rapport de recherche du LADL, Paris.

9. ANNEXE: MANUEL D'UTILISATION DE PUPITRE