UER MATHEMATIQUES

UNIVERSITE DE NANCY I

ANALYSE DE TEXTES
DE
MOYEN - FRANCAIS

Rapport de DEA

SOUVAY GILLES CRIN JUIN 1986

REMERCIEMENTS

Je remercie toutes les personnes qui m'ont aidé et soutenu au cours de cette année pour la réalisation de ce projet :

- l'équipe de l'URFA pour sa gentillesse et sa bonne humeur au cours de nos réunions qui a facilité la compréhension et la conception de l'analyseur

- J.M. PIERREL parrain de ce stage, et qui m'a guidé tout au long du semestre

- J.C. DERNIAME qui a bien voulu servir de ~~boîte aux lettres~~ l'intermédiaire entre l'URFA et le CRIN

- toutes les personnes du CRIN qui m'ont aidé à travailler sur VAX.

PLAN DU RAPPORT

INTRODUCTION

CHAPITRE 1 LES CONNAISSANCES ET L'ANALYSEUR

1. Présentation.

1.1. l'équipe de l'URFA.
1.2. les caractéristiques du Moyen-Français
1.3. constitution de la banque de données textuelle

2. Les connaissances.

2.1. recherche de la forme canonique
- segmentation
- règles de graphies

2.2. recherche de la solution
- une solution
- structure du lexique
- l'analyse

3. Le modèle de l'analyseur.

4. Le travail à réaliser.

CHAPITRE 2 LA MISE EN OEUVRE DE L'ANALYSEUR

1. La représentation des connaissances.

1.1. les désinences
1.2. les règles de graphie

1.2.1. le compilateur de règles II.1 et III

a) la syntaxe
b) l'analyseur syntaxique
c) la génération
- présentation
- le format d'entrée d'une règle
- la génération de la partie condition
- la génération de la partie action

d) les contrôles

1.2.2. le compilateur de règles II.2

a) la syntaxe
b) l'analyseur syntaxique
c) la génération
d) les contrôles

1.3. le lexique

a) la syntaxe
b) l'analyseur syntaxique
c) la génération
d) les contrôles

1.4. la table des paradigmes

2. La réalisation

2.1. le programme principal
2.2. les moteurs d'inférence
2.2.1. but
2.2.2. les règles
2.2.3. le moteur d'inférence
- la base de faits
- la stratégie
- l'algorithme

2.3. les fonction annexes

CHAPITRE 3 LES RESULTATS

1. Aide à l'analyse des résultats.

1.1. étude des règles
1.2. étude de l'analyseur

2. Résultats au niveau des règles

2.1. le niveau II
2.2. le problème du niveau III

3. Résultats de l'analyseur

CONCLUSION

ANNEXE

INTRODUCTION

Le C.N.R.S. a mis en oeuvre la réalisation d'un dictionnaire du Moyen-Français qui est la langue du XIVe au XVe siècle. Dans ce cadre l'équipe de l'URFA-URL10 (INaLF) a été chargée d'alimenter une base de données textuelle.

Le traitement d'un texte nécessite d'attribuer à chaque mot une analyse et un lemme qui permettront d'accéder aux informations lexicales et morphologiques réunies dans la base.

Les linguistes de l'URFA ont acquis une expérience en matière d'analyse et ont décidé d'optimiser un traitement déjà informatisé. Ils ont donc élaboré le modèle d'un analyseur et fait l'inventaire des connaissances dont il a besoin.

Dans une première partie je présenterai ce modèle avec les connaissances dont il se sert, je détaillerai ensuite la réalisation de chacune des composantes qui le constitue et enfin nous analyserons les résultats obtenus.

URFA : *Unité de Recherche sur le Français Ancien*
URL10 : *Unité de Recherche Linguistique N 10*
INaLF : *Institut National de la Langue Française*

CHAPITRE 1 LES CONNAISSANCES ET LE MODELE

1. Présentation

1.1. **L'équipe de l'URFA** avec qui j'ai travaillé est constituée de cinq personnes ; quatre linguistes : O.DERNIAME , M.HENIN , S.MONSONEGO , H.NAIS et un informaticien : J.GRAFF.

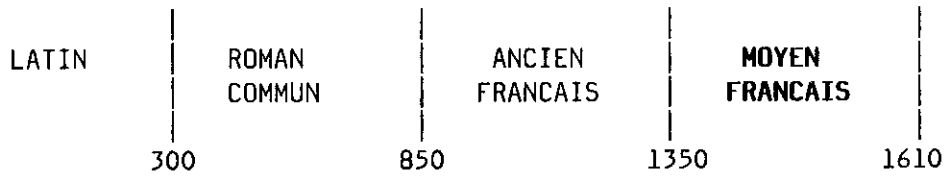
Le travail effectué par le groupe consiste à traiter des **textes littéraires médiévaux**. La saisie des textes, l'indexation et la lemmatisation des mots de ces textes se fait de manière informatique. Récemment, les activités de l'équipe ont été intégrées dans un vaste programme de travail suscité par le C.N.R.S au sein de l'Institut National de la Langue Française : la création d'un **Dictionnaire du Moyen-Français**. L' URFA a donc été chargée d'alimenter une base de données textuelle qui permettra de constituer les articles du dictionnaire.

L'équipe, afin d'améliorer son rendement se consacre d'autre part à automatiser et à optimiser les traitements. Afin de l'aider, elle a fait appel au C.R.I.N et JM.PIERREL après entretien a jugé qu'une collaboration pourrait être utile pour les deux parties, et c'est ainsi qu'un sujet de DEA a été proposé et choisi par un étudiant.

Depuis fin janvier, nous nous sommes régulièrement rencontrés. Parmi la dizaine de réunions que nous avons tenues, nous avons consacré deux à trois séances à cerner le problème. Ensuite cinq séances ont été nécessaires pour présenter les différentes connaissances et leur utilisation. L'équipe se servant déjà de l'informatique pour ses traitements avait formalisé une grande partie du problème au niveau des connaissances. Il fallut préciser la chronologie et la hiérarchie de l'utilisation des informations. Enfin les dernières réunions ont été consacrées à apporter des modifications à partir des premiers résultats fournis par le programme.

1.2. les caractéristiques du Moyen-Français.

Situons tout d'abord la langue :



Ce schéma [GUIRAUD 63] montre que le Moyen-Français est une **période detransition** dans l'histoire de la langue. Actuellement c'est encore une langue mal connue du fait de l'inexistence d'un dictionnaire. Le Moyen-Français traité dans ce sujet concerne le XIVe et XVe exclusivement.

Elle oppose un obstacle considérable aux traitements informatisés du fait du **grand nombre des variantes** et des types auquels elles appartiennent. Les homographies sont nombreuses et l'identification des mots est donc délicate. De plus entre le début de la période et sa fin le **système flexionnel évolue.**

Exemple de graphies :

- dans un même texte on peut trouver pour :

AGNEL : AIGNEL, AGNELS, AIGNELS, AGNEAUS, AIGNEAUS, AGNEAUX, AIGNEAUX, AGNEAULS, AGNEAULZ, AIGNEAULZ, AGNEAULX, AINGNEL ...

CONVENRAI : CONVENRAY, CONVENRAI , COUVENRAY, COVENRAI, COVENRAY, CONVENDRAI, CONVENDRAY, COUVENDRAI, COUVENDRAY, COVENDRAI, COVENDRAY, CONVIENDRAI ...

- PRINSE peut être une graphie de PRINCE ou le participe de PRENDRE

L'identification d'un mot se fera par une série d'étapes que nous exposerons plus loin qui permettra de se ramener à un mot générique.

1.3 constitution de la banque de données textuelle.

Elle est composée de textes enregistrés in extenso et de la lemmatisation de chacun des mots du texte. A chaque mot on associe une **entrée** dans le dictionnaire et une **analyse** succinte qui précise sa classe grammaticale et sa forme (temps-mode-personne pour les verbes, masculin-féminin pour

les substantifs) .

Comme nous l'avons vu précédement avec la graphie PRINSE, il peut y avoir plusieurs analyses pour un mot. Le but ne va pas être de déterminer l'analyse effective dans le texte mais de fournir l'**ensemble des propositions possibles.** Il est entendu que parmi les analyses proposées par le système il doit se trouver la solution. Il sera à la charge du linguiste de valider la bonne hypothèse.

Le traitement d'un texte va s'effectuer en quatre temps :
- En local un texte est saisi et subit un **pré-traitement** où l'on isole chacun de ses mots.
- Le fichier de mots est ensuite envoyé à l'**analyseur** implanté au CIRCE à ORSAY où chaque mot sera lemmatisé.
- Les résultats sont renvoyés en local où l'on **passe en revue** chaque analyse.
- Après ce traitement la banque textuelle est enrichie d'un texte. Il reste à **mettre à jour** les fichiers de l'analyseur à partir des modifications effectuées par le linguiste (introduction de nouvelles bases, de nouvelles significations de désinences ...).

cf figure 1

Actuellement l'analyseur travaille de façon très simple. Toutes les flexions déjà rencontrées d'un mot sont contenues dans le lexique avec une analyse immédiate. En se reportant au mot AGNEL on se rend compte qu'il y a une grande perte de place mémoire. Le nouvel analyseur qui doit être constitué aura deux avantages.

La **prolifération** de graphies aura **disparu**, on conservera une ou deux formes génériques pour une série de mot.

D'autre part le système pourra proposer des analyses sur des mots qu'il n'aura jamais traités auparavant. En effet à partir du moment où un substantif ou un adjectif aura été rencontré et que l'on aura mis à jour le lexique, on pourra reconnaître sa forme au féminin ou au pluriel. De même avec les verbes une fois les formes génériques entrée que nous appelerons base, la base rentrée on pourra analyser toutes ses formes fléchies.

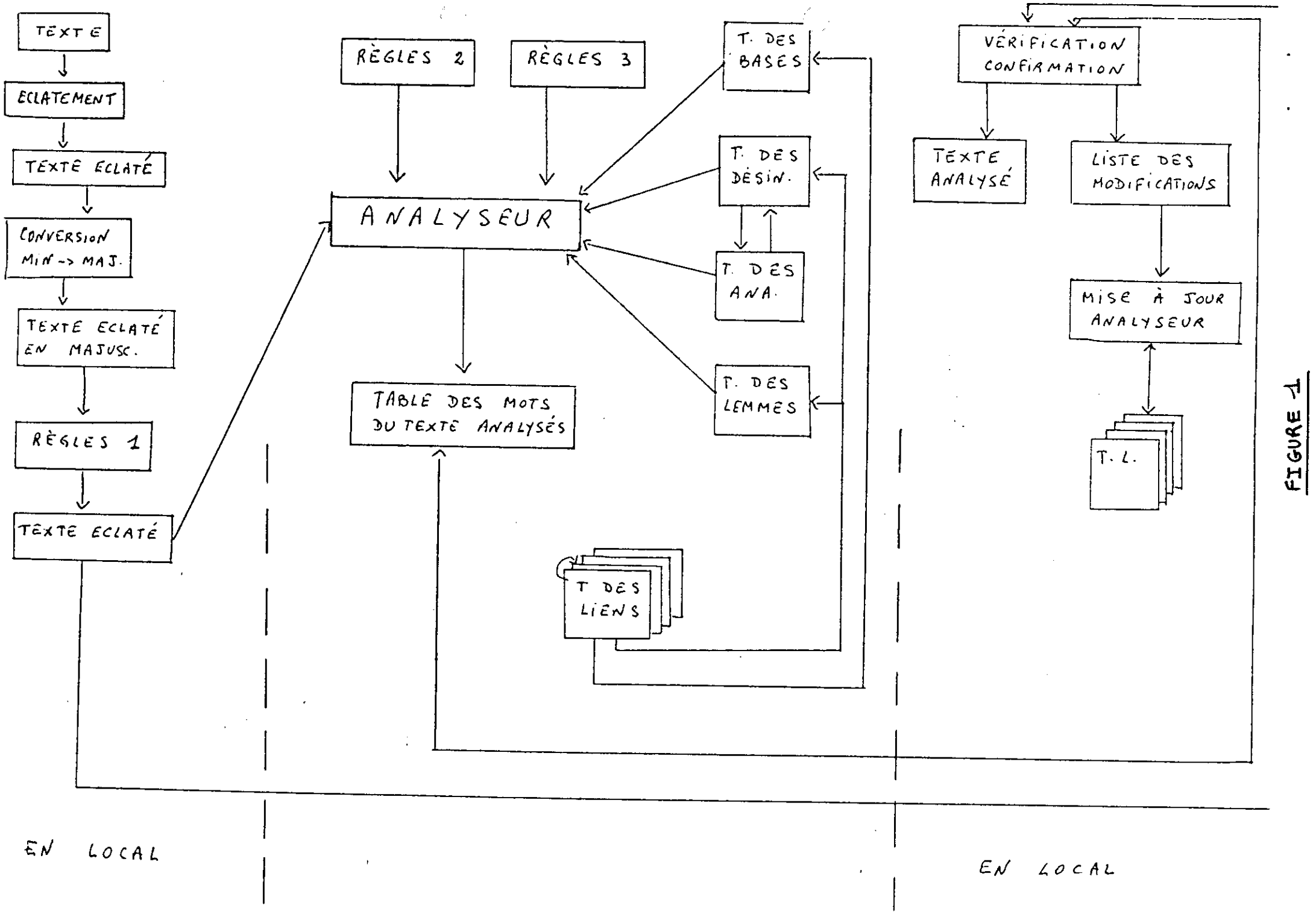

TEXTE
ECLATEMENT
TEXTE ECLATÉ
CONVERSION MIN -> MAJ.
TEXTE ECLATÉ EN MAJUSC.
RÈGLES 1
TEXTE ECLATÉ
RÈGLES 2
RÈGLES 3
T. DES BASES
T. DES DÉSIN.
T. DES ANA.
T. DES LEMMES
ANALYSEUR
TABLE DES MOTS DU TEXTE ANALYSÉS
T DES LIENS
VÉRIFICATION CONFIRMATION
TEXTE ANALYSÉ
LISTE DES MODIFICATIONS
MISE À JOUR ANALYSEUR
T. L.
EN LOCAL
EN LOCAL

FIGURE 1

2. Les connaissances.

Les connaissances dont se sert l'analyseur sont de deux types. Il y a celles qui ramenent à la forme générique et celles qui déterminent la ou les solutions.

2.1. recherche de la forme générique.

Il y a deux manières complémentaires pour atteindre la base. On peut le faire en segmentant le mot en base-désinence et en appliquant des règles de graphie sur le mot complet.

* la **segmentation** est assez simple à mettre en oeuvre. Il suffit a priori de connaître toutes les désinences possibles et ensuite de chercher toutes celles qui sont présentes dans le mot en cours de traitement. On a distingué trois types de régimes flexionnels : le système **invariable**, le système **nominal** et le système **verbal**.
Le système invariable a la particularité de ne pas posséder de désinence.
Le système nominal a un nombre de désinences limité : ø, S, E, ES.
Les désinences du système verbal sont beaucoup plus nombreuses et sont en cours d'inventaire. Je peux citer pour exemple ø, E, ES, T, ONS, OMES, {S, ES, ENT rien que pour le présent de l'indicatif.

Il y a possibilité d'inclusion d'une désinence dans une autre. Ceci aménera à envisager plusieurs hypothèses :

ex. pour AIMENT on étudiera AIMEN-T et AIM-ENT.

* **les règles de graphie** : Leur application est beaucoup plus complexe. Elles consistent à supprimer ou à ajouter des groupes de lettres lorsque certaines conditions sont remplies.
Si certaines règles étaient appliquées avec brutalité, on créerait plus d'ambiguïtés qu'on n'en résoudrait. Si par exemple on veut faire tomber S devant consonne, on réécrira CHATEL pour CHASTEL ; or CHATEL est la forme courante pour CHEPTEL au XIVe siècle.
D'autres règles, comme la réécriture d'Y en I, n'ont pratiquement pas d'inconvénient. D'autre-part, certaines règles ne s'appliquent qu'après segmentation. Une notion de type de règle a donc été introduite.

Les différents types de règles sont les suivants :

- les **règles I** . Elles s'appliquent de manière systématique, la forme d'origine est donc perdue. Ce sont des règles qui a priori ne conduisent pas à des ambiguïtés. Pour exemple la réécriture de Y en I.

- les **règles II** . Au contraire des précédentes, elles peuvent produire des formes ambiguës. Elles ont été établies de manière pragmatique. Dans tous les cas on conserve la forme initiale.
Elles se divisent en deux catégories selon qu'elles portent sur un mot avant ou après segmentation.
+ les **règles II de niveau 1** sont utilisées avant segmentation. Elles peuvent être appliquées à n'importe quel endroit dans le mot. Un exemple de règle II.1 a été donné précédemment - S tombe devant consonne - .
+ les **règles II de niveau 2** servent à modifier la finale d'une base en fonction de la désinence. Il faut donc que le mot ait été segmenté. Une règle II.2 dit que si la finale de la base est V et que la désinence est E alors V se transforme en F. Appliqué sur VIV-E cela permet de trouver VIF-E.
Les règles II.2 pour le système nominal et le système verbal sont séparées. En effet l'application de la règle précédente dans l'hypothèse verbale ne permettrait jamais de fournir un résultat valable.

- les **règles III.** Ce sont des règles d'annulation qui entrent en jeu lorsque l'on n'a pas trouvé de solution. Elles se subdivisent en trois catégories :

+ les **règles III.1** reviennent sur les productions des règles I.
+ les **règles III.2** reviennent sur les productions de II.1
+ les **règles III.3** sont utilisées si les règles II.2 n'ont pas permis de délivrer de solution.
Elles ne doivent être mises en oeuvre qu'en **dernière limite** si l'on n'a rien trouvé.

1.2.2 recherche de solution.

* une **solution** se compose d'une analyse et d'un lemme. Une **analyse** comporte trois colonnes. La première colonne correspond à la classe

morphologique. La deuxième colonne représente dans le cas des noms le genre et dans le cas des verbes le temps-mode. La troisième colonne indique la personne dans le cas des verbes ou note le changement de classe d'un mot dans un texte.

cf figure 2

exemples d'analyse

FC3

verbe parfait 3ème personne du singulier

BF

adjectif féminin

FAA

infinitif substantivé

* le **lexique** est l'élément déterminant pour trouver une solution. Il permet de savoir si un mot existe et de connaître les désinences qui sont compatibles avec sa ou ses bases afin de déterminer l'analyse.

Chaque élément du lexique possède cinq composantes :

cf figure 3

- la **base** : c'est la forme générique du mot. Il faut parcourir l'ensemble des bases pour déterminer si un mot existe ou non.

- le **type** : cette composante permet de différencier deux grands ensembles de mots, les **invariants** des autres. Par invariant on entend tout mot qui ne possède pas de paradigme.

- la **classe** : ce champ correspond à la classe morphologique.

- le **lemme** : ce champ contient un pointeur vers le lemme associé à la base.

- le **paradigme** : ce champ pointe vers l'ensemble des flexions possibles de la base. Dans le cas où le type de la base est invariant ce champ contient

Analyse 1ère colonne : classe morphologique

A	Substantif
B	Adjectif (qualificatif, ordinal, possessif et quelques indéfinis)
C	Cardinal
D	Déterminant
E	Pronom
F	Verbe
G	Article contracte
H	Préposition
I	Conjonction
J	Adverbe
K	Nom propre
L	Interjection
M	Mot contracte autre qu'article contracte
N	Adjectif/Déterminant
P	Adjectif/Pronom
Q	Déterminant/Pronom
R	Préposition/Adverbe
S	Conjonction/Adverbe
V	Mot en langue étrangère isolé dans une phrase en français
Y	Mot en langue étrangère dans une phrase dans cette langue
Z	Mot grammatical fréquent porteur d'ambiguïtés multiples (classes et lemmes)

Les mots des classes K, V et Y ne sont pas lemmatisés et sont donc exclus des tables de l'analyseur.

F ≡ verbes

A B N P Q ≡ noms

autres - invariable

sauf VAY

FIGURE 2.1

Analyse 2ème colonne

Système nominal genre : Féminin

Masculin

X douteux ou commun

A	F M X	genre douteux, seulement si le contexte ne permet pas de déterminer le genre du substantif
B	F M X	genre commun, systématique pour les adjectifs en -e, pour les adjectifs épicènes seulement si le contexte ne permet pas de déterminer le genre.
D	F M X	pour les formes où la marque de genre n'est pas apparente (ex. : SES)
G	M X	même règle que pour D (ex. : DES)
N	X	seulement NOSTRE et VOSTRE dans cette catégorie actuellement
P	F M X	
Q	F M X	
C E K M Z		ne reçoivent pas de code de genre

ni bien sûr les classes d'invariables H, I, J, L, R, S (sauf si code en 3e colonne).

Analyse 3e colonne

Colonne utilisée pour noter le changement de classe d'un mot dans un texte.

On peut donc y trouver la plupart des codes de 1ère colonne (codes de classe morphologique) et

T pour l'emploi allégorique

Exemples :

AFT	Substantif féminin en emploi allégorique
BMA	Adjectif substantivé
BMJ	Adjectif employé adverbialement
FQB	Participe passé adjectif
FAA	Infinitif substantivé
AFL	Substantif employé comme exclamation
LMA	Exclamation employée comme substantif, etc.

FIGURE 2.3

Analyse du verbe

2ème colonne : temps-mode

3ème colonne : personne (de 1 à 6)

F A	infinitif
B	participe présent
Q	participe passé
C 1 à 6	parfait
D 1 à 6	indicatif présent
E 1 à 6	subjonctif présent
F 2, 4, 5	impératif
G 1 à 6	indicatif imparfait
H 1 à 6	futur
I 1 à 6	conditionnel
K 1 à 6	subjonctif imparfait

FA, FB et FQ peuvent recevoir un code dans la 3e colonne, cf. codes de 3e colonne. L'absence de code en 3e colonne d'un mot analysé FQ indique qu'il s'agit d'une forme composée du verbe.

FIGURE 2.4

Table des Bases. ≡ Lexique

Base	Type	Classe	Pointeur Lemme	Pointeur Para. ou Analyse	
FU	I	F	x estre 1	-	C3
VIF	-	B	x vif	4	-
DE*	-	H	x de	-	-
MOIS	I	A	x mois	-	M
MUR	-	A	x mur	1	-
GENT	-	A	x gent 1	3	-
GENT	-	B	x gent 2	4	-
GRANT	-	B	x grant	6	-
ENKOIRES	-	J	x encore	-	-

* DE est dans le sous-ensemble Invariable de par sa classe morphologique.

FU et MOIS ont le type I indiqué car ce sont de "faux" invariables.

FIGURE 3

l'analyse immédiate du mot.

* l'accès à la deuxième et troisième colonne se fait par consultation de la **table des paradigmes** . En ligne on trouve le numéro de paradigme, en colonne la désinence et à l'intersection la proposition qui peut être donnée. **cf figure 4**

* l'**analyse** est donc obtenue à partir de ces éléments. Si le mot est présent dans le lexique on recherche son numéro de paradigme. En consultant la table des paradigmes on obtient à partir de la désinence l'analyse effective du mot. Il peut n'y avoir aucune solution si la case est vide, dans le cas d'une désinence incompatible.

exemple de solution

VIF-E → BF vif
FU → FC3 estrel
ENKOIRES → J encore

3. Le modèle de l'analyseur.

L'analyseur travaille sur les formes issues des règles I. Les mots sont d'abord soumis aux règles II de niveau 1 (production de formes parallèles), puis soumis aux essais, par segmentation, de recherche de désinences, avec comparaison aux trois sous-ensembles de bases : bases invariables, bases nominales, bases verbales, et application d'éventuelles règlesII de niveau 2. On procède alors à la vérification des compatibilités de la base et de l'éventuelle désinence testée, si celle-ci donne un résultat positif on obtient une ou plusieurs propositions selon que la désinence pointe sur une ou plusieurs analyses et la base sur un ou plusieurs lemmes. Si les règles II.2 proposent plusieurs hypothèses de bases parallèles, on les teste successivement et on recommence toute l'opération si les règles III ont proposé plusieurs hypothèses de formes parallèles.

Cette description du fonctionnement de l'analyseur tirée des documents fournis par l'URFA et les précisions obtenues au cours des réunions a

Paradigmes nominaux

	Ø	S	E	ES	exemples
1	M	M			le mur
2	F	F			la dame
3	M F X	M F X			le, la gent
4	M	M	F	F	bon
5	X	X			autre
6	M F X	M F X	F	F	grant
7	M		F	F	gros

M masculin
F féminin
X douteux ou commun

FIGURE 4

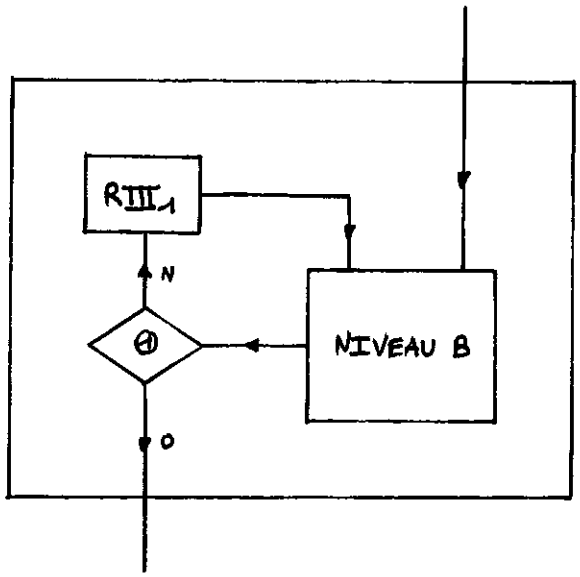
permis d'obtenir les **schémas de fonctionnement** de la **figure 5.**

4 Le travail à réaliser

Après que le problème eut été exposé, nous avons décidé que je devrais construire l'analyseur pour tout ce qui n'était pas verbal. Il devrait être implanté pour l'instant sur le VAX du C.R.I.N. . Cette première version devrait permettre de tester la validité du modèle et d'affiner les connaissances utilisées.

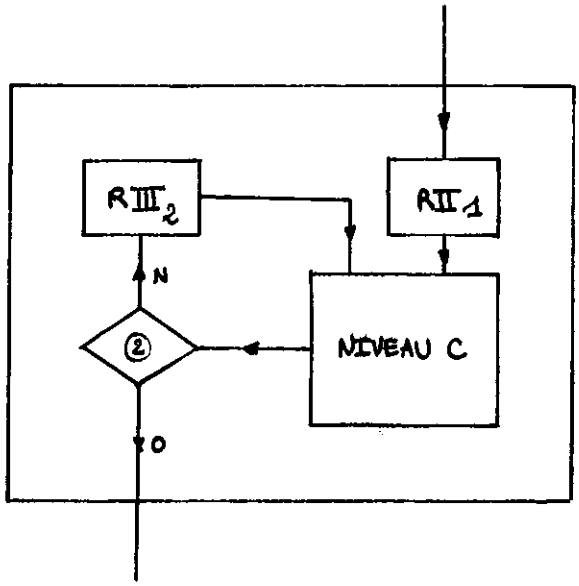
FIGURE 5.1

NIVEAU A



① il y a une solution ou plus de $RIII_1$

NIVEAU B



② il y a une solution ou plus de $RIII_2$

NIVEAU C

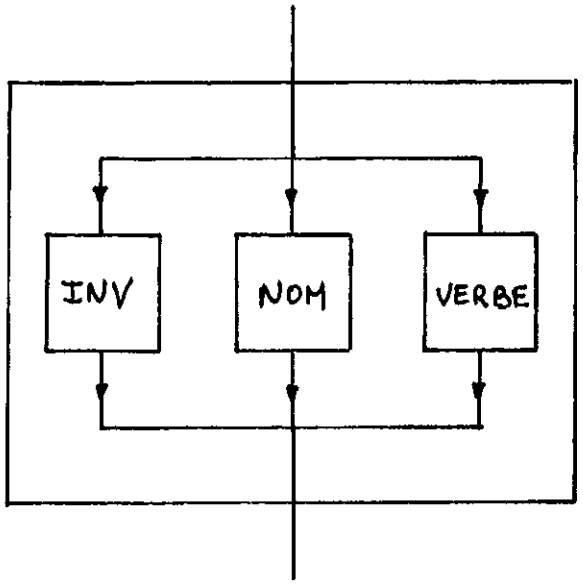
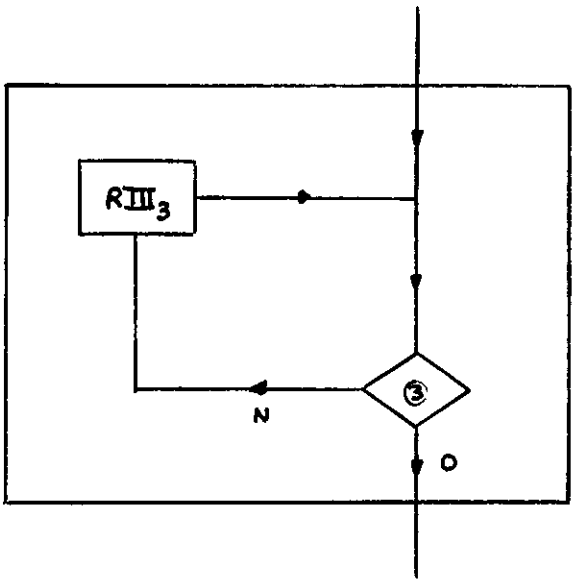


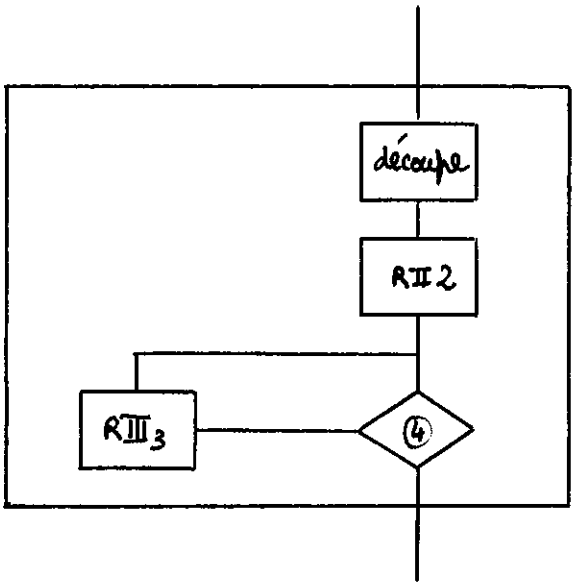
FIGURE 5.2

INV



③ solution ou plus de RIII3

NOM / VERBE



④ solution ou plus de RIII3

CHAPITRE 2 LA MISE EN OEUVRE DE L'ANALYSEUR

Dans ce chapitre nous allons aborder l'analyseur d'une manière plus informatique. Nous étudierons tout d'abord la manière dont les connaissances ont été représentées puis nous passerons aux principaux algorithmes qui composent l'analyseur.

1. La représentation des connaissances.

On peut distinguer deux types principaux de connaissances. Il y a celles qui sont quasiment figées et qui n'évolueront pratiquement pas comme l'ensemble des désinences et la table des paradigmes dans le système nominal, et celles qu'il faudra constamment mettre à jour tel le lexique ou pouvoir modifier à volonté comme les règles de graphie.

On peut remarquer aussi que le premier type d'informations est assez limité quantitativement contrairement au second. Ces deux critères vont influer sur le mode de représentation.

1.1. les désinences.

Comme nous l'avons vu dans le chapitre précédent, le nombre de désinences possibles pour le système nominal est réduit à quatre. Les désinences sont donc contenues dans une variable de type liste.

1.2. les règles de graphies

Nous avons vu dans l'introduction du paragraphe 2.1, que ces règles doivent pouvoir être modifiées à volonté. Ceci a pour conséquence qu'elles doivent être facilement accessibles à l'utilisateur.

La représentation adoptée est la règle de production. L'ensemble des règles d'un type et d'un niveau sera regroupé dans un fichier texte. Afin qu'elles puissent être comprises par l'analyseur, il va falloir effectuer une **compilation** de ces règles. Le fichier résultat pourra être lu par le programme pendant une phase d'initialisation. La moindre modification nécessite la re-compilation de l'ensemble du fichier.

Nous allons étudier l'ensemble du processus de la compilation en commençant par la syntaxe des règles, nous poursuivrons par l'analyseur syntaxique associé et enfin nous aborderons la représentation interne des règles en étudiant la génération et enfin nous traiterons des contrôles à effectuer.

Nous regarderons le compilateur des règles de niveau II.1 et III en premier et ensuite les différences qui existent pour le compilateur II.2.

1.2.1. les compilateurs de règles de niveau II.1 et de niveau III.

a) la syntaxe des règles

Le but de ce paragraphe est de faire ressortir les traits caractéristiques de la syntaxe des règles de production.

Chaque règle a la forme suivante

numéro : **si** condition **alors** action

La partie condition peut éventuellement être vide d'où la forme :

numéro : action

La partie **condition** est formée de prémisses liées entre elles par les opérateurs booléens et, ou, non. Les prémisses utilisent les quatre primitives : précédé de, suivi de, initial, final; les chaînes de caractères ; les ensembles génériques : voyelle, consonne .

Il existe deux types d'**actions** possibles, la chute d'un groupe de lettres et la réécriture d'une séquence de lettres en une autre séquence.

La syntaxe détaillée des règles est fournie en annexe sous deux aspects :

- une forme en terme de graphe type syntaxe du Pascal [MAURICE-80]

- une forme en terme de grammaire algébrique en respectant les conventions imposées par YACC [JOHNSON-78].

b) l'analyseur syntaxique.

L'analyse syntaxique de la base de règles a été effectuée à l'aide de **YACC** [JOHNSON-78] qui est un analyseur LR(1). Il permet de construire facilement le corps de l'analyseur. Il reste à fournir une fonction d'initialisation, l'analyseur lexical et les actions sémantiques associeés à chaque règle.

* l'**initialisation** consiste à initialiser les différentes variables du programme comme le fichier source , le fichier objet ...

* l'**analyseur lexical** est constitué d'un **automate d'état fini** dont les états de satisfaction servent à reconnaître les unités syntaxiques lues. Il se compose de huit états dont quatre de satisfaction et un d'erreur. Il peut lire sept types de lettres (les caractères alphabétiques, les symboles, espace, fin de ligne, fin de fichier, les chiffres, le point-virgule) . Les symboles @ | { } et / sont considérés comme des caractères alphabétiques accentués.
Les états de satisfaction sont les suivants :

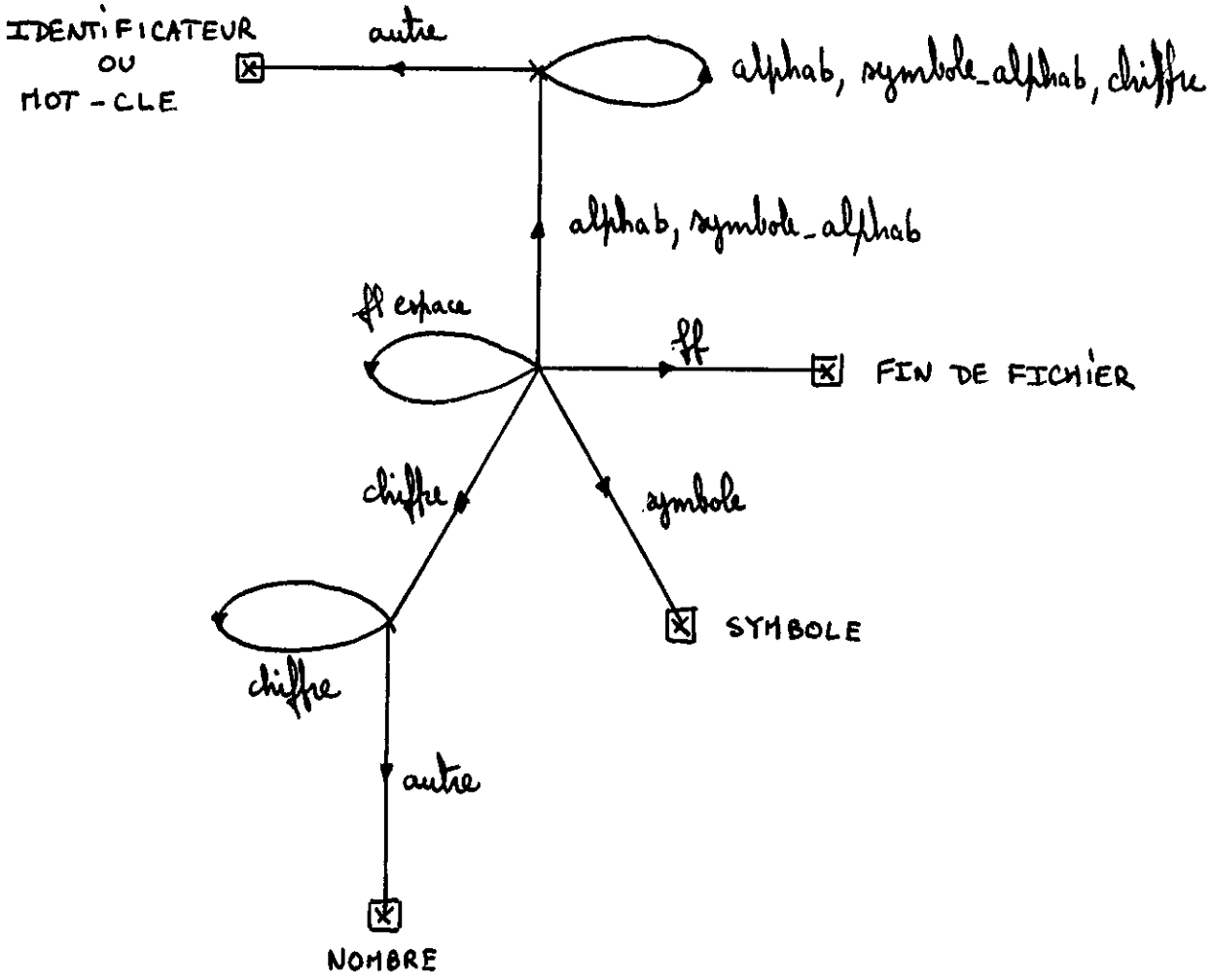
etat -1 : on a reconnu un identificateur ou un mot-clé
etat -2 : on a reconnu un symbole
etat -3 : on a reconnu la fin de fichier
etat -4 : etat d'erreur
etat -5 : on a reconnu un nombre

Le graphe de cet automate est représenté sur la **figure 6.**

* On attache à chaque règle de la grammaire une **action sémantique.** L'ensemble des actions va permettre de construire l'**arbre syntaxique** de la base de règles.

Les noeuds de l'arbre seront du type suivant :

GRAPHE DE L'AUTOMATE LEXICAL



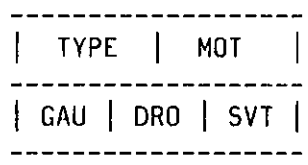
□ : état de satisfaction

fl : fin de ligne

ff : fin de fichier

symbole_alphab : symboles représentant les caractères accentués.

FIGURE 6



TYPE : conserve le type du noeud

MOT : ce champ est utilisé pour conserver une chaîne de caractères dans les noeuds de type IDF (identificateur) .

On remarque d'après la grammaire algébrique, que l'on a besoin au maximum de trois fils pour un noeud donné , d'où les trois champs :

GAU : pointe vers le fils gauche.

DRO : pointe vers le fils droit.

SVT : sert à la fois à pointer sur le troisième fils ou vers un frère dans le cas de liste (liste de règles LREG, liste de chaînes CHAI ...).

exemples de noeud **cf figure 7**

Chaque action sémantique consiste à créer le noeud associé à la règle et à le lier à chacun de ses fils ou frères.

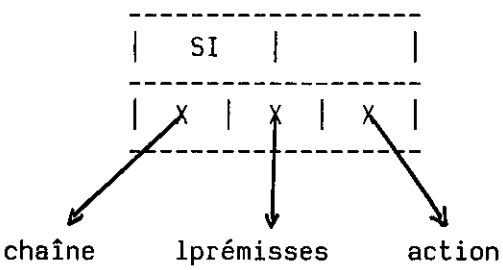
c) la génération

Elle va consister à parcourir l'arbre syntaxique et selon le cas du noeud à effectuer une certaine action.

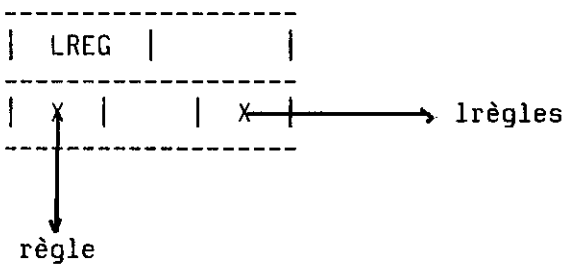
* **présentation**

Le code engendré sera un programme LISP ou plus précisément **l'affectation d'une variable**. La variable contiendra l' **ensemble de règles.**

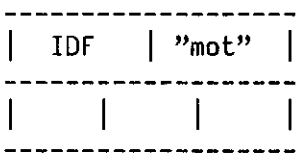
règle : chaîne ':' SI lpremisses ALORS action



lrègles : lrègles règle



chaîne : IDF



premisse : ENTRE generique

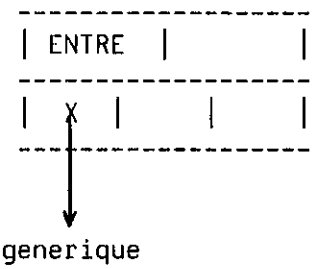


FIGURE 7

Chaque règle possède deux composantes : son numéro et son corps. Le **numéro** est utilisé pour tracer l'historique du système. Le **corps** constitue la partie utile de la règle. Il a lui aussi deux composantes : la **condition** de déclanchement et l'**action** à effectuer. Chaque corps de règle est en fait un morceau de fonction qui sera évalué par l'interprêteur LISP. On fournit à cette fonction le mot à traiter sous un format particulier et elle renvoie en résultat le mot obtenu après application de la règle.

* **le format d'entrée d'une règle**

Le mot qui est traité par une règle est formaté en **décomposition.** Une décomposition est un triplet **(alpha milieu beta)** , où alpha représente la partie initiale, milieu le groupe de lettres en cours de traitement et beta la partie finale du mot. Cette décomposition est effectuée par le moteur d'inférence qui régit les règles.

* **génération de la partie condition**

A chacune des différentes prémisses, on a associé un appel de fonction. Le travail à effectuer selon les types de noeud prémisses consistera à engendrer l'**appel de la fonction** avec ses différents paramètres.

Commençons par présenter la relation prémisse → fonction :

- SUIVI DE tyype → suivi_de
- PRECEDE DE tyype → precede_de
- INITIAL → initiale
- FINAL → finale
- ENTRE générique → precede_de et suivi_de

- SUIVI DE générique SAUF expression FINAL : l'appel associé est formé de la composition de la fonction suivi_de et d'une comparaison sur la finale.
- pour les quatre derniers types de prémisses qui concernent des tests sur les finale et initiale indépendemment de l'endroit où l'on se trouve dans le mot, les fonctions associées dépendent de la constatation que FINALE = tyype revient à tester PRECEDE DE tyype et INITIALE = tyype étant équivalent à SUIVI DE tyype, seuls les paramètres différant.

Les différents **paramètres** que l'on peut fournir sont les suivant :

- pour precede_de et initiale, il faut transmettre la partie [beta] de la décomposition

- pour suivi_de et finale, il faut transmettre la partie **alpha**

- pour precede_de et suivi_de il faut aussi transmettre

+ un **entier** qui indique le type d'expression que l'on trouve dans prémisse (cela correspond à la règle tyype de la grammaire). Ce numéro permet de plus de connaître le nombre de paramètres.

+ les chaînes de caractères, ensembles, ensemble générique qui sont exprimés par le noeud "tyype".

Les différentes prémisses sont liées entre-elles par les fonctions booléennes et,ou,non.

La condition contient une partie supplémentaire correspondant au champ milieu qui doit être détecté pour déclencher l'évaluation des prémisses.

exemple de parties conditions :

```
R30 : si suivi de consonne alors EIN -> AIN
(and (equal milieu '(E I N))
     (suivi_de 3 beta consonne))

R02 : si suivi de J alors AI -> A
(and  (equal milieu '(A I))
      (suivi_de   1 beta  'J))
```

*** génération de la partie action**

La partie action doit renvoyer le mot après application de la règle. Si elle ne s'applique pas on renvoie l'atome LISP nil. Le résultat est obtenu par concaténation :

- dans le cas chaîne$_1$ -> chaîne$_2$: chaîne$_1$ est contenue dans milieu. On fait la concaténation alpha + chaîne$_2$ + beta.

- dans le cas chaîne TOMBE . On fait la concaténation alpha + beta.

d) les contrôles

Les contrôles à effectuer sont de deux types. Il y a ceux qui sont liés au sens et portent sur les prémisses, et ceux liés à l'exécution.

* **contrôles liés au sens** : ce sont les plus faciles à mettre en oeuvre. On vérifie que la prémisse courante a un sens ; par exemple - suivi de consonne # A - n'a pas de sens. Ce genre d'erreur n'empêche pas le bon fonctionnement du programme.

* **contrôles liés à l'exécution** : ces contrôles sont beaucoup plus complexes et les erreurs qu'ils doivent détecter peuvent provoquer le bouclage du système. Lorsque la grammaire sous-jacente à l'ensemble de règle engendre un sous-langage de type a^n la règle peut s'appliquer indéfiniment sur le mot produit.

Voici trois exemples d'ensemble de règles permettant d'illustrer ces propos et d'exposer ce qui a été fait pour l'éviter.

- exemple 1

RO1 : A -> AA

On voit tout de suite que tout mot α A β peut se réécrire α A^n β. Lorsque l'on a une action X -> Y il faut donc vérifier que la chaîne du membre gauche n'est pas sous-chaîne du membre droit.

- exemple 2

RO1 : si entre consonnes alors A -> AA

Cette règle ne provoque pas de bouclage. Le mot obtenu après application de la règle α AA β ne comporte plus de A entre consonnes.

- exemple 3

RO1 : A -> B

RO2 : B -> AA

Cet ensemble de règles peut se ramener à l'exemple 1. Si l'on rencontre un mot α A β on aura non seulement α B β mais aussi la chaîne α AA β qui pourra être dérivée indéfiniment.

La version 0 du compilateur n'est pas capable de détecter le bouclage de l'exemple3.

Pour l'exemple 2 nous avons appliqué la méthode de l'exemple 1 sans tenir compte des prémisses. Pour cet exemple un message d'avertissement est donc envoyé à l'utilisateur. Toutefois la compilation continue.

Pour la version 0 l'utilisateur aura donc à vérifier les règles qu'il introduit dans le système.

Puisque les méthodes de **préventions** n'ont pas été mises en oeuvre lorsque de telles règles apparaîssent, elles sont détectées à l'execution. La **guérison** consiste à laisser tomber un mot qui aura été jugé trop long pour faire partie des données.

1.2.2. le compilateur de règles II.2

Les principes développés auparavant sont toujours valables pour ces règles. La différence provient de la syntaxe.

a) la syntaxe

La forme d'une règle est identique :

numéro : **si** condition **alors** action

La **partie condition** n'est jamais vide. Elle se compose de deux parties : la désinence et la finale. La composante **désinence** vient du fait que les règles II2 s'appliquent après segmentation. Quelle que soit la règle, on fait une hypothèse sur la désinence. La composante **finale** comme son nom l'indique porte sur la finale de la base.

La **partie action** peut être de trois types différents. On enlève ou transforme un groupe de lettres comme pour les règles précédentes, mais on a aussi la possibilité en plus d'en ajouter un.

b) l'analyseur syntaxique est identique à quelques détails près.

c) pour la **génération**, la différence réside dans le format d'entrée. La décomposition à effectuer est un triplet **(debut fin désinence)** où début représente l'initiale et fin la finale.
Pour la partie condition il faut prévoir d'introduire une fonction qui teste la désinence et pour la partie action AJOUTER chaîne revient à produire la forme debut + fin + chaîne.

d) les **contrôles** à effectuer sont de même nature.

1.3. le lexique

L'implantation actuelle du lexique n'est que **provisoire**. Elle doit permettre de tester l'analyseur sur un nombre réduit de mots par rapport à la version future.

Toutefois afin de pouvoir éprouver la valeur de l'analyseur sur des mots intéressants, il faut pouvoir mettre à jour facilement le lexique sans passer par la représentation interne. C'est pourquoi, comme pour les règles de graphie, un mécanisme de compilation a été utilisé.

a) la syntaxe du fichier source dans lequel va se trouver le vocabulaire est simple. Pour chaque lettre de l'alphabet, on regroupe dans un même bloc la liste des mots commençant par cette lettre. Pour chaque mot on doit préciser ses attributs (base, type, classe, lemme et paradigme). La grammaire algébrique et la dernière version du lexique sont fournies en annexe 4.

b) l'analyseur syntaxique est le même que celui utilisé pour les règlesII.2 mis à part la table des mots-clés et quelques autres détails.

c) la génération va consister à produire du code LISP où l'on affecte la variable *#:mf:lettreX* à la liste des mots du bloc de lettre initiale *X*.

d) étant donné que l'on a une version provisoire du lexique, le seul **contrôle** effectué vérifie que la définition d'un mot se trouve dans le bloc correspondant à sa lettre initiale.

2.1.4. la table des paradigmes

La taille étant réduite et ne devant pas évoluer est représentée par une variable de type table à deux dimensions.

2 La réalisation

Nous allons étudier dans cette partie les différents algorithmes qui composent le système.

2.1 le programme principal

Le programme principal est composé de plusieurs **fonctions imbriquées** les une dans les autres. Une partie de ces fonctions correspond aux modules qui apparaissent dans les schémas de la **figure 5** (niveauA, niveauB, decoupe ...). Les autres permettent de simplifier l'écriture des précédentes en divisant le problème mais surtout, comme nous le verrons dans le chapitre 3, seront utilisés pour suivre le déroulement pas à pas du programme.

2.2. les moteurs d'inférence

Ils méritent d'être regardés en détail. Ils ont tous la même structure. La différence entre le moteur II.1 et II.2 provient du format d'entrée pour les règles, mais les stratégies sont identiques.

La stratégie provisoirement adoptée pour le moteur RIII est identique, nous en verrons les raisons dans le chapitre 3.

Nous allons décrire ici ce que fait le moteur de niveau II.1

2.2.1. Le but

Le moteur de niveau II.1 a été conçu pour fournir à partir d'un mot la liste des **graphies dérivées**
parmi lesquelles doit être présente la forme qui est conservée dans le lexique ou qui permettra d'y arriver. Le moteur II.1 fournira en plus un historique de son fonctionnement.

Le système se compose d'une base de connaissances qui est un ensemble donné de règles et d'un programme d'inférence.

2.2.2. les règles

La base est constituée pour l'instant d'un peu moins de soixante-dix **règles de production.**

Chaque règle a la forme suivante :

numéro : **si** condition **alors** action

cf paragraphe 2.1.2 et annexe.

Il n'y a pas priorité d'une règle par rapport à une autre. Elles sont toutes au même niveau. L'ordre dans lequel elles doivent être fournies est donc indifférent.

2.2.3. le moteur d'inférence

* La **base de faits** qui est manipulée par le moteur est constituée de mots. Elle posséde deux composantes :

- **les mots à traiter**

- **les mots traités**

Ainsi on évite de boucler lorsque le système posséde des règles inverses l'une de l'autre. On ne traite une forme produite que si elle n'est pas présente dans la liste des mots qui ont déjà été traités.

exemple de boucle :

R62 : si entre consonnes alors ER -> AR

R63 : si entre consonnes alors AR -> ER

* **La stratégie** utilisée est toute simple. On prend dans la liste des mots à traiter le premier élément et on essaye d'appliquer successivement chacune des règles. Si l'on produit une ou plusieurs nouvelles formes, on met à jour la liste des mots traités.

Il existe pourtant une possibilité de bouclage du système lorsque la grammaire sous-jacente à la base de connaissances engendre un langage de type a^n. Pour éviter ceci deux types de méthodes peuvent être utilisés :

- la **prévention** : on vérifie a priori que la grammaire n'engendre pas de langage de ce type lors de la phase d'adaptation des règles au moteur.

- la **guérison** : lorsque l'on détecte un bouclage, on arrête le processus.

La méthode actuellement utilisée est la guérison. Si l'on rencontre un mot "trop long" (plus d'un certain nombre de caractères) on passe au suivant. Le mécanisme de prévention existe à un niveau très bas.

cf paragraphe 1.2.1 d)

* **algorithme**

Maintenant que le fonctionnement du moteur a été décrit, nous allons présenter l'algorithme correspondant.

Il manipule les **listes**. Nous allons donc utiliser les **opérations classiques** sur ces structures : estvide , appartient , adjqueue , suptête , tête.

On utilise aussi deux fonctions spécifiques à l'application :

- **appliquereg** : on passe en argument la règle et le mot courants. On obtient en résultat la liste des productions de la règle.

- **numéro** : on passe en argument la règle courante et on obtient en résultat le numéro de cette règle.

Les variables portent un nom explicite, et ne posent donc pas de problème d'interprétation. Les listes utilisées sont :

- **mots-à-traiter** : contient les mots qui doivent être traités. la tête de cette liste est représentée par **mot_courant**.

- **règlesIII** : contient l'ensemble des règles de niveau II.1. Un élément de cette liste a deux composantes : le numéro de la règle et la règle effective.

- **règles** : contient les règles qui n'ont pas encore été appliquées. La tête de cette liste est représentée par **règle courante**.

- **production** : contient les productions obtenues en appliquant la règle courante. La tête de cette liste est représentée par **production_courante**.

- **historique** : contient pour le mot d'origine, toutes les actions qui ont été effectuées sur ce mot et ses dérivés. Un élément de cette liste est donc composé ainsi : < mot , no_règle , resultat > .

- **mots_traités** : contient les mots que l'on a traités auparavant.

On obtient alors l'algorithme de la **figure 8.**

2.3. les fonctions annexes

* **le lexique** est géré par **hcode** sur la première lettre.

* **la compatibilité** d'une base et d'une désinence se fait à partir des informations du lexique. Lorsqu'une base est présente dans le lexique, on regarde d'abord si elle n'est pas invariante. On accède ensuite à l'analyse contenue dans la table des paradigmes à l'aide du champ paradigme.

* **la segmentation** consiste à parcourir toutes les désinences et à renvoyer en résultat tous les découpages base-désinence qui ont été possibles.

```
RESULTATS : HISTORIQUE , MOTS_TRAITES

JUSQU'A  ESTVIDE ( MOTS_A_TRAITER )  FAIRE

   MOT_COURANT <-- TETE ( MOTS_A_TRAITER )
   REGLES <-- REGLESII1

   JUSQU'A ESTVIDE ( REGLES ) FAIRE

      REGLE_COURANTE <-- TETE ( REGLES )
      PRODUCTION <-- APPLIQUEREG ( REGLE_COURANTE , MOT_COURANT )

      JUSQU'A  ESTVIDE ( PRODUCTION ) FAIRE

         PRODUCTION_COURANTE <-- TETE ( PRODUCTION )

         SI
            NON ( APPARTIENT ( PRODUCTION_COURANTE , MOTS_A_TRAITER ) )
         ALORS

            MOTS_A_TRAITER <-- ADJQUEUE ( PRODUCTION_COURANTE ,
                                          MOTS_A_TRAITER)
            HISTORIQUE <-- ADJQUEUE ( < MOT_COURANT ,
                                        NUMERO ( REGLE_COURANTE ) ,
                                        PRODUCTION_COURANTE > ,
                                        HISTORIQUE )
         FSI

         PRODUCTION <-- SUPTETE ( PRODUCTION )

      FIN_JUSQU'A

      REGLES <-- SUPTETE ( PRODUCTION )

   FIN_JUSQU'A

   MOTS_TRAITES <-- ADJQUEUE ( MOT_COURANT , MOTS_TRAITES )

   MOTS_A_TRAITER <-- SUPTETE ( MOTS_A_TRAITER )

FIN_JUSQU'A


DONNEES : REGLESII.1 , MOTS_A_TRAITER
```

FIGURE 8

CHAPITRE 3 LES RESULTATS

1. Aide à l'analyse des résultats

Afin de mieux pouvoir interpréter les résultats, deux programmes ont été réalisés. Le premier concerne les règles et le second l'analyseur.

1.1. étude des règles

Afin de tester les règles sans passer par l'analyseur, un programme qui utilise directement les moteurs d'inférence a été construit.

A partir d'un menu on peut **tester le niveau de règle que l'on désire.** Une fois le niveau choisi, il suffit d'entrer au clavier le mot que l'on veut traiter et le programme envoie à l'écran toutes les formes produites par les règles. On peut au besoin demander l'historique du fonctionnement. Afin de faciliter l'examen, il est possible d'envoyer les résultats et l'historique dans un fichier texte.

1.2. étude de l'analyseur

Lorsque l'on traite un mot l'analyseur indique s'il a trouvé une solution et dans ce cas la propose. Ceci ne permet pas de connaître la manière dont il l'a obtenue.

Il va falloir conserver toutes les étapes intermédiaires. Nous avons vu que l'analyseur était constitué de fonctions imbriquées. A chaque niveau, les fonctions vont renvoyer deux choses : les **résultats** obtenus par les niveaux internes et l'**historique** de leur obtention. Ceci revient pour l'historique, à construire un **arbre** dont la racine est la fonction principale de l'analyseur, les noeuds les fonctions internes et les feuilles les fonctions d'accès au lexique et de test de compatibilité.

On peut ainsi associer une **grammaire algébrique** au déroulement du programme.

L'arbre obtenu n'est pas directement interprétable. On va le parcourir et le mettre sous une **forme lisible**. Mon approche a été indirecte. Ce n'est pas directement l'arbre qui permettra d'obtenir la forme lisible. Il sera sauvegardé

dans un fichier. Ensuite en utilisant les mécanismes de compilation présentés dans le chapitre 2, on engendrera la forme.

2. résultats au niveau des règles

2.1 les mots obtenus par le moteur ont permis d'affiner les règles de **niveau II**. Les affinements ont réduit le nombre de formes.

Ils consistaient à restreindre le champ d'application des prémisses.

exemple 1

RII1.R41 : si initiale et suivi de consonne alors O -> OU

est devenu:

RII1.R41 : si initiale et
suivi de consonne sauf [K,L,M,N,NK,S,T] final
alors O -> OU

exemple 2

RII1.R4 : si precede de consonne et suivi de voyelle alors / -> S
RII1.R4 : si precede de consonne et suivi de voyelle alors S/ -> S

Ces deux règles appliquées sur le mot CHANS/ON produisaient 6 formes pour 22 règles appliquées. Nous avons réduit en précisant que la consonne devait être différente de S dans R4, à 3 formes pour 9 règles.

exemple 3

RII2.R8 : si désinence = S alors ajouter [T,F,K,L,P]

produisait beaucoup de formes inutiles d'où la restriction :

RII2.R8 : si desinence = S et finale # [E,U,L,D,K,F,P,G] alors
ajouter
[T,F,K,L,P]

Le fait d'exclure E supprime l'application sur des mots d'une règle qui ne leur était pas destinée. Les autres lettres empêche le déclenchement de

cette règle lorsque l'une des autres s'applique.

2.2 le problème des règles III

Il provient des règles elles-mêmes. Le niveau III.1 est destiné à l'origine à revenir sur les production des règles I. Mais cela ne permet pas toujours d'obtenir une analyse. De nouvelles règles ont été proposées. Mais elles s'appliquent dans un grand nombre de cas :

ex : R20 : si entre consonne alors E -> I

Il faudrait revoir la stratégie et pour des problèmes de temps je n'ai pu le faire.

Nous avons donc supprimé ces nouvelles règles et adopté provisoirement la statégie des autres moteurs d'inférence.

3. résultats de l'analyseur

Le projet a permis de se rendre compte que l'analyseur avait le **comportement prévu** et délivrait les résultats pourvu que la base soit connue du lexique.

Pour illustrer ce paragraphe nous proposons la **figure 9** et en annexe 6 une série de résultats.

(THIESMOIN ((THIESMOIN RII1 ((THIESMOIN () () ()) (((THIESMOIN ()) (((
THIESMOIN ()) RII2 () ()))))) (((THIESMOIN ()) (((THIESMOIN ()) RII2 ()))))))
()) (TIESMOIN RII1 ((TIESMOIN () () ()) (((TIESMOIN ()) (((TIESMOIN ()) RII2 (
) ()))))) (((TIESMOIN ()) (((TIESMOIN ()) RII2 ()))))) ()) (THIEMOIN RII1 ((
THIEMOIN () () ()) (((THIEMOIN ()) (((THIEMOIN ()) RII2 ()))))) (((THIEMOIN
()) (((THIEMOIN ()) RII2 ()))))) ()) (TIEMOIN RII1 ((TIEMOIN () () ()) (((
TIEMOIN ()) (((TIEMOIN ()) RII2 ()))))) (((TIEMOIN ()) (((TIEMOIN ()) RII2 (
)))))) ()))) ((THESMOIN RIII1 ((THESMOIN RII1 ((THESMOIN () () ()) (((
THESMOIN ()) (((THESMOIN ()) RII2 ()))))) (((THESMOIN ()) (((THESMOIN ())
RII2 ()))))) ()) (TESMOIN RII1 ((TESMOIN () () ()) (((TESMOIN ()) (((
TESMOIN ()) RII2 ()))))) (((TESMOIN ()) (((TESMOIN ()) RII2 ()))))) ()) (
THEMOIN RII1 ((THEMOIN () () ()) (((THEMOIN ()) (((THEMOIN ()) RII2 ())))))
(((THEMOIN ()) (((THEMOIN ()) RII2 ()))))) ()) (TEMOIN RII1 ((TEMOIN () ()
()) (((TEMOIN ()) (((TEMOIN ()) RII2 ((BM temoin))))))) (((TEMOIN ()) (((
TEMOIN ()) RII2 ())))))))))))

ARBRE FORME BRUTE

FIGURE 9.1

```
(THIESMOIN
    ((THIESMOIN
        RII1
        ((THIESMOIN () () ())
         (((THIESMOIN ()) (((THIESMOIN ()) RII2 () ())))
         (((THIESMOIN ()) (((THIESMOIN ()) RII2 () ())))
        ())
     (TIESMOIN
         RII1
         ((TIESMOIN () () ())
          (((TIESMOIN ()) (((TIESMOIN ()) RII2 () ())))
          (((TIESMOIN ()) (((TIESMOIN ()) RII2 () ())))
         ())
     (THIEMOIN
         RII1
         ((THIEMOIN () () ())
          (((THIEMOIN ()) (((THIEMOIN ()) RII2 () ())))
          (((THIEMOIN ()) (((THIEMOIN ()) RII2 () ())))
         ())
     (TIEMOIN RII1
              ((TIEMOIN () () ())
               (((TIEMOIN ()) (((TIEMOIN ()) RII2 () ())))
               (((TIEMOIN ()) (((TIEMOIN ()) RII2 () ())))
              ()))
    ((THESMOIN
        RIII1
        ((THESMOIN
            RII1
            ((THESMOIN () () ())
             (((THESMOIN ()) (((THESMOIN ()) RII2 () ())))
             (((THESMOIN ()) (((THESMOIN ()) RII2 () ())))
            ())
         (TESMOIN RII1
                  ((TESMOIN () () ())
                   (((TESMOIN ()) (((TESMOIN ()) RII2 () ())))
                   (((TESMOIN ()) (((TESMOIN ()) RII2 () ())))
                  ())
         (THEMOIN RII1
                  ((THEMOIN () () ())
                   (((THEMOIN ()) (((THEMOIN ()) RII2 () ())))
                   (((THEMOIN ()) (((THEMOIN ()) RII2 () ())))
                  ())
         (TEMOIN RII1
                 ((TEMOIN () () ())
                  (((TEMOIN ()) (((TEMOIN ()) RII2 ((BM temoin)) ())))
                  (((TEMOIN ()) (((TEMOIN ()) RII2 () ())))
                 ())))))
```

ARBRE FORMATE A L'AIDE DE "PRETTYF"

FIGURE 9.2

```
THIESMOIN       --> RIIl
                THIESMOIN       invariable
                                THIESMOIN       ECHEC
                                nom
                                THIESMOIN-0     --> II2
                                                THIESMOIN-0     ECHEC
                                verbe
                                THIESMOIN-0     --> II2
                                                THIESMOIN-0     ECHEC
                --> RIIl
                TIESMOIN        invariable
                                TIESMOIN        ECHEC
                                nom
                                TIESMOIN-0      --> II2
                                                TIESMOIN-0      ECHEC
                                verbe
                                TIESMOIN-0      --> II2
                                                TIESMOIN-0      ECHEC
                --> RIIl
                THIEMOIN        invariable
                                THIEMOIN        ECHEC
                                nom
                                THIEMOIN-0      --> II2
                                                THIEMOIN-0      ECHEC
                                verbe
                                THIEMOIN-0      --> II2
                                                THIEMOIN-0      ECHEC
                --> RIIl
                TIEMOIN         invariable
                                TIEMOIN         ECHEC
                                nom
                                TIEMOIN-0       --> II2
                                                TIEMOIN-0       ECHEC
                                verbe
                                TIEMOIN-0       --> II2
                                                TIEMOIN-0       ECHEC
| rIIIl
v
THESMOIN        --> RIIl
                THESMOIN        invariable
                                THESMOIN        ECHEC
                                nom
                                THESMOIN-0      --> II2
                                                THESMOIN-0      ECHEC
                                verbe
                                THESMOIN-0      --> II2
                                                THESMOIN-0      ECHEC
                --> RIIl
                TESMOIN         invariable
                                TESMOIN         ECHEC
                                nom
                                TESMOIN-0       --> II2
                                                TESMOIN-0       ECHEC
                                verbe
                                TESMOIN-0       --> II2
                                                TESMOIN-0       ECHEC
                --> RIIl
                THEMOIN         invariable
                                THEMOIN         ECHEC
                                nom
                                THEMOIN-0       --> II2
                                                THEMOIN-0       ECHEC
                                verbe
                                THEMOIN-0       --> II2
                                                THEMOIN-0       ECHEC
                --> RIIl
                TEMOIN          invariable
                                TEMOIN          ECHEC
                                nom
                                TEMOIN-0        --> II2
                                                TEMOIN-0        BM temoin
```

FIGURE 9.3a

verbe

TEMOIN-0 --> II2

TEMOIN-0 ECHEC

FORMATAGE PAR COMPILATEUR

FIGURE 9.3 b

CONCLUSION

Le comportement du système est satisfaisant. Sa programmation a permis de montrer que son comportement correspondait à ce qui avait été prévu.

Il existe tout de même un problème au niveau des règles III qui ont été réduites en nombre. Une étude poussée de la stratégie du moteur d'inférence devrait permettre de le résoudre.

L'implantation du lexique est à revoir et pourrait très bien former une entité d'une base de données dans laquelle on trouverait aussi les lemmes et les paradigmes. Mais cela ne peut être fait tant que l'étude du système verbal n'aura pas été entreprise.

La structure de l'analyseur accepte déjà l'hypothèse verbale et propose une analyse pour les verbes invariants (ex : FU). Il reste à entrer les désinences, les règles II2 et surtout les paradigmes.

[GUIRAUD-63] Pierre GUIRAUD
"Le Moyen-Français"
Que sais-je ? P.U.F.

[JOHNSON-78] Stephen C. JOHNSON
"YACC : Yet Another Compiler Compiler"

[MAURICE-78] Pierre MAURICE
"Pascal 80 : Manuel d'utilisation"
Service de synthèse et d'orientation de la recherche en informatique.

[BEEKER-82] Etienne BEEKER
"GASP : Générateur d'Analyseur Syntaxique Pascal"

[LAURIERE-85] Jean-Louis LAURIERE
"Représentation et utilisation des connaissances"
T.S.I

[O'HARE-82] G.M O'HARE D.A. BELL
"The coexistence approch to knowledge representation"
Expert Systems Octobre 85 Vol2,N4

[CHAILLOUX-83] Jérome CHAILLOUX
"LeLisp80 : le manuel de référence"
I.N.R.I.A.

[KERNIGHAN-84] Brian W. KERNIGHAN et Denis RITCHIE
"Le langage C"
MASSON 84

ANNEXE 1

1. Le vocabulaire.

2. Les règles

2.1. généralités
2.2. syntaxe des règles II.1
2.3. syntaxe des règles II.2
2.4. syntaxe des règles III

SYNTAXE DES REGLES

Nous allons décrire dans cette partie la forme sous laquelle devront être introduites les régles afin d'être comprises du programme.

1 Le vocabulaire

Les éléments du vocabulaire des régles sont :

* les chaînes :

elles sont composées d'une suite de lettres majuscules.

elles ont dix caractéres significatifs.

les caractéres accentués sont codés de la facon suivante :

lettre	é	è	à	ç	ù
représentation	{	}	@	/	|

* les nombres :

ils sont composés d'une suite de chiffres décimaux.

ils ont dix caractéres significatifs.

le nombre 0 représente la désinence vide.

* les symboles spéciaux :

+ * , [ ] = # -> espace

alors	ajouter	consonne	de	desinence	entre	final	initial
non	ou	precede	rien	sauf	si	sinon	suivi	tombe	voyelle

les mots-réservés (symboles spéciaux alphabétiques) doivent être tapés en minuscule pour être reconnus.

la construction " ; suite de caractéres " est un commentaire. Elle peut apparaître n'importe où dans un programme. Tout le texte qui se trouve après le point-virgule est ignoré jusqu'à la fin de la ligne.

le caractére espace peut apparaître n'importe où et en nombre quelconque sauf dans une chaîne , un nombre ou un symbole spécial.

les symboles * + sont équivalents respectivement à ET et OU.

2 Les régles

2.1 généralités

Toutes les régles d'un niveau doivent se trouver dans un même fichier texte.

La syntaxe va être décrite au moyen de diagrammes qui utilisent les conventions suivantes.

(NOM) : le symbole nom apparaît dans le texte.

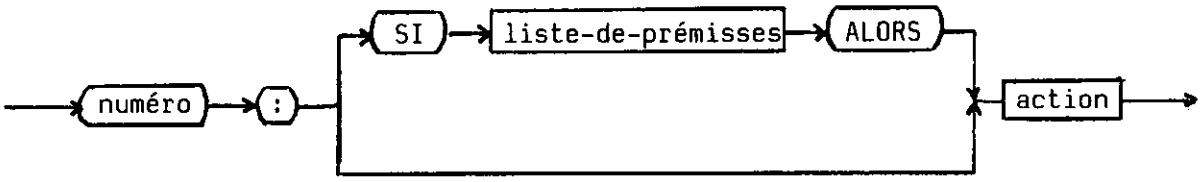
[nom] : il existe une description de nom sous forme de diagramme

La forme générale d'une régle est :

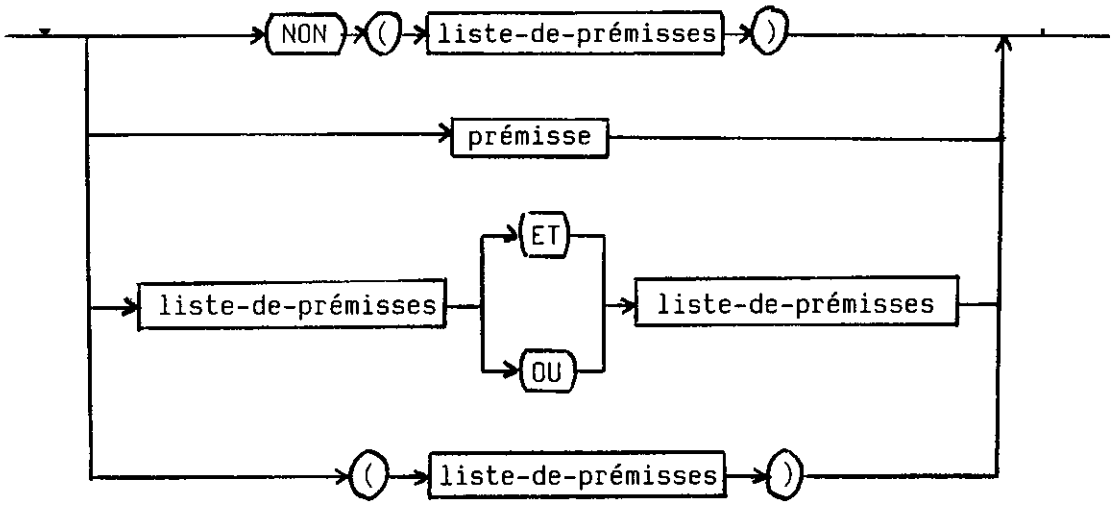
si condition alors action.

2.2 Syntaxe des régles de réécriture de niveau II.1

REGLE



LISTE DE PREMISSES



L'opérateur ET est prioritaire par rapport à l'opérateur OU. Le parenthésage permet de modifier cette priorité. Il est conseillé de ne pas composer des expressions trop complexes afin d' éviter une interprétation différente de celle que l'on voulait exprimer. Il vaut mieux avoir plusieurs régles avec une liste de prémisses réduite pour la lisibilité du problème.

Examinons le cas de la régle suivante :

RX : si initial ou precede de consonne et suivi de consonne alors ER -> AR

Cette régle est interprétée comme :

RX : si initial ou (precede de consonne et suivi de consonne) alors ER -> AR

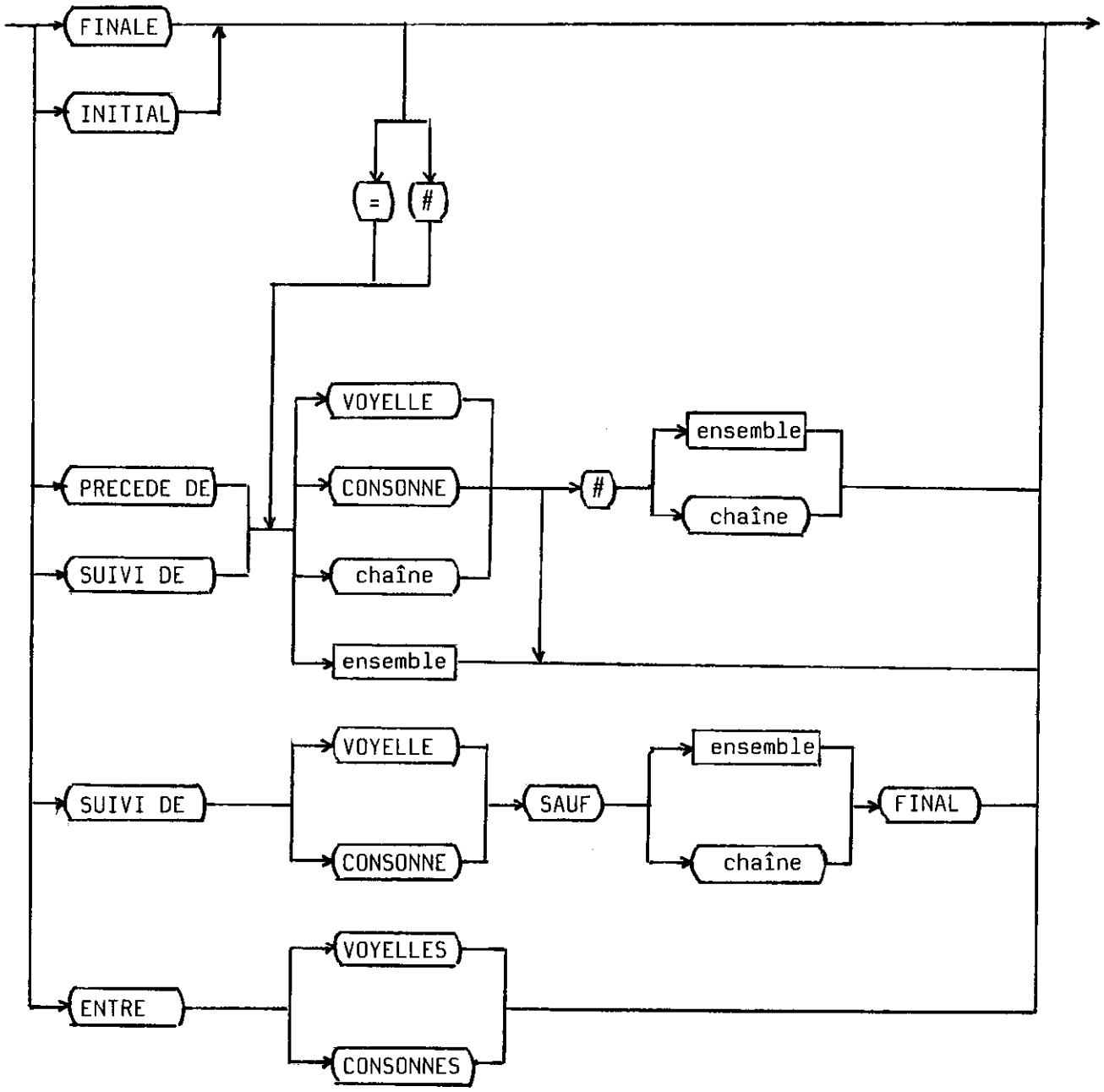
alors que l'on voulait exprimer :

RX : si (initial ou precede de consonne) et suivi de consonne alors ER -> AR

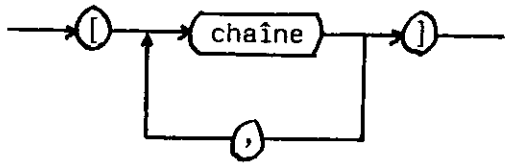
La solution pour éviter ce genre d'erreur, est de créer deux régles :

RX1 : si initial et suivi de consonne alors ER -> AR
RX2 : si entre consonne alors ER -> AR

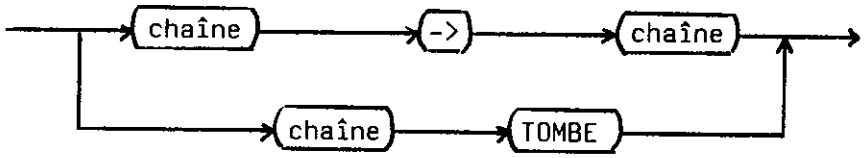
PREMISSE


FINALE
INITIAL
=
#
PRECEDE DE
SUIVI DE
VOYELLE
CONSONNE
chaîne
ensemble
#
ensemble
chaîne
SUIVI DE
VOYELLE
CONSONNE
SAUF
ensemble
chaîne
FINAL
ENTRE
VOYELLES
CONSONNES

ENSEMBLE



ACTION



exemple de regles II.1

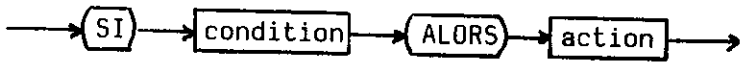
```
; ceci est un exemple de régles II.1

R01 : X -> Y
R02 : X tombe
R03 : si precede de consonne alors XX -> YY
R04 : si entre voyelles alors T -> SS
R05 : si suivi de ION alors S -> Z
R06 : si precede de voyelle # [A,E,I] alors T -> K
R07 : si initial et suivi de consonne alors X -> Y
R08 : si non( suivi de consonne ) ou initial alors S -> ST
R09 : si suivi de I # ION alors X -> Z
```

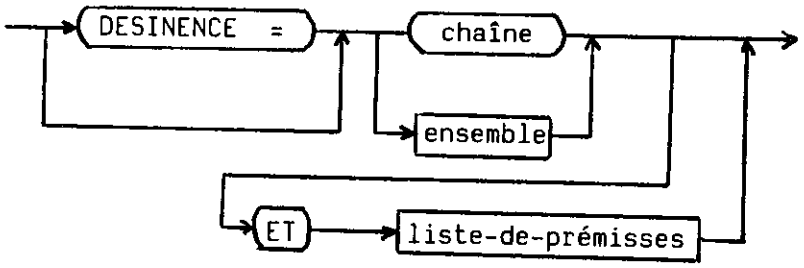
2.3 syntaxe des régles de niveau II.2

La syntaxe de ces régles reprend une grande partie des descriptions présentées pour les régles de niveau II.1. Il faudra donc se reporter à celles-ci si nécéssaire.

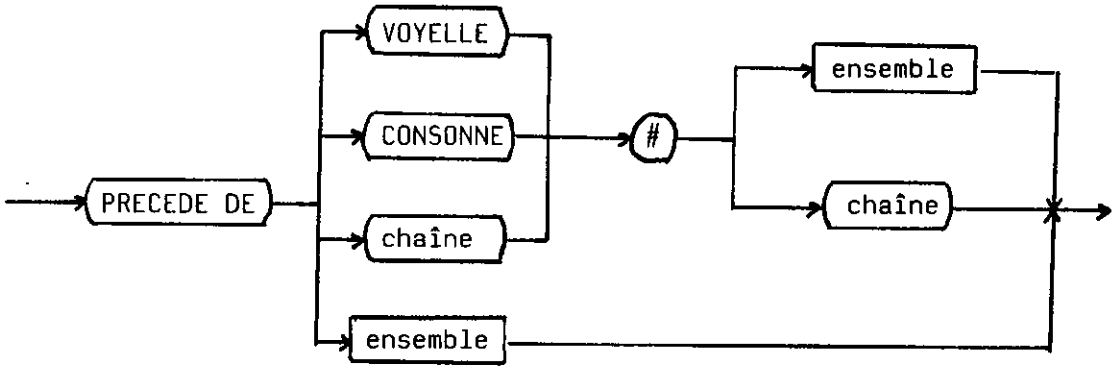
REGLE



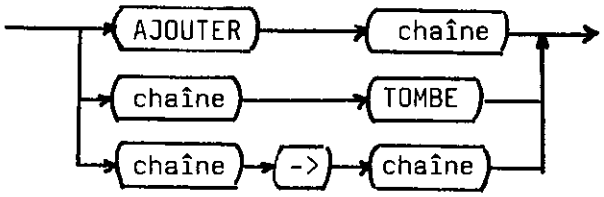
CONDITION



PREMISSE



ACTION



exemple de regles II.2

R01 : si desinence = O alors ajouter T
R02 : si desinence = [O,S] alors X tombe
R03 : si desinence = E et precede de consonne alors XX -> X

2.4 syntaxe des régles de niveau III

La syntaxe des régles de niveau III est identique à celle de niveau II.1.

ANNEXE 2

Les règles II de niveau 1 et les règles III

1. Les règles

1.1. les règles II.1
1.2. les règles III.1
1.3. les règles III.2
1.4. les règles III.3

2. Le grammaire algébrique

3. Le compilateur.

3.1. le programme principal	**princ.c**
3.2. l'analyseur lexical	**lex.c**
3.3. l'analyseur syntaxique	**y.tab.c**
3.4. la construction de l'arbre	**arbre.c**
3.5. la génération	**generation.c**

Regles I

1 BB -> B

2 DD -> D

3 FF -> F

4 GG -> G

5 MM -> M

6 NN -> N

7 PP -> P

8 TT -> T

9 ' (apostrophe) -> E sauf C', Ç', CH, M', L', S', T'

10 PH -> F

11 Y -> I

12 C -> K Si suivi de A, O, S, U, Z, et en Finale

13 SÇ -> S à l'initiale

14 M -> N Si suivi de cons. ≠ de B, M, P

15 MP -> M Si suivi de N
MP -> N Si suivi de S, T

16 Q -> K
QU -> K

17 X -> S devant cons. sauf C, S
X -> S en Finale sauf six, Fix, Dix
AX -> AUS en Finale
EX -> EUS "
OX -> OUS "

18 G -> J Si suivi de I, É, È

19 GE -> JE Si suivi de E, ~~suivi de finale~~
GE -> J Si suivi de voy. ≠ E

20 GU -> G Si suivi de voy.

21 Z -> S à l'intérieur et en Finale sauf -EZ à la Finale

22 É -> E sauf en Finale et suivi de S

23 NGN -> GN'

24 NM -> M

25-26 N -> M Si suivi de B, P

27 È -> E sauf suivi de S

28 ?S -> ES en Finale, ES à l'intérieu

```
;  LISTE DES REGLES II.1
; ----------------------

R1  : AA -> A
R2  : si suivi de J alors AI -> A
R3  : si precede de voyelle et suivi de voyelle alors / -> SS
R4  : si precede de consonne # S et suivi de voyelle alors / -> S
R5  : si precede de voyelle et suivi de [E,I,{] alors C -> SS
R6  : si precede de consonne # [S,C] et suivi de [E,I,{] alors C -> S
R7  : si suivi de ION alors CC -> X
R8  : si suivi de voyelle # ION alors CC -> C
R9  : si finale alors CH -> K
R10 : si suivi de ION alors CT -> X
R11 : si finale alors EZ -> {S
R12 : si initiale alors ILL -> IL
R13 : si precede de consonne alors ILL -> IL
R14 : II -> I
R15 : si precede de [A,E,O,U] alors LL -> L
R16 : si precede de voyelle et suivi de voyelle alors S/ -> SS
R17 : si precede de consonne et suivi de voyelle alors S/ -> S
R18 : si initiale et suivi de [E,I,{] alors SC -> S
R19 : si precede de voyelle et suivi de [E,I,{] alors SC -> SS
R20 : si precede de consonne et suivi de [E,I,{] alors SC -> S
R21 : si precede de consonne alors SS -> S
R22 : si precede de voyelle et suivi de ION alors T -> SS
R23 : si precede de consonne # C et suivi de ION alors T -> S
R24 : si precede de voyelle et suivi de MENT alors U -> L
R25 : si precede de voyelle et suivi de consonne # L alors UL -> U
R26 : si precede de voyelle et finale alors UL -> U
R27 : si initiale et suivi de consonne # [R,S] alors AD -> A
R28 : si precede de voyelle et suivi de consonne # [L,R] alors B tombe
R29 : si precede de voyelle et suivi de consonne # [C,H,L,R] alors C tombe
R30 : si suivi de consonne alors EIN -> AIN
R31 : si precede de voyelle et suivi de consonne # [L,R] alors F tombe
R32 : si non ( initial ) et suivi de consonne # [H,L,N,R] alors G tombe
R33 : si initiale alors H tombe
R34 : si precede de consonne # C alors H tombe
R37 : si precede de [A,O] alors IGN -> GN
R38 : si suivi de V alors KO ->KON
R39 : si suivi de V alors KOU -> KON
R40 : si precede de voyelle # U et suivi de consonne # L alors L -> U
R41 : si precede de consonne et
        suivi de consonne sauf [K,L,M,N,NK,S,T] final alors O -> OU
R42 : si precede de consonne et suivi de [A,E,{] alors O -> OU
R43 : si initiale et
        suivi de consonne sauf [K,L,M,N,NK,S,T] final alors O -> OU
R44 : si initiale et suivi de [A,E,{] alors O -> OU
R45 : si initial et suivi de consonne # [M,N] alors O -> AU
R46 : si non ( initial ) et suivi de consonne # [L,R] alors P tombe
R47 : RR -> R
R48 : si non ( precede de TRE ) et suivi de consonne # [C,/,S] alors S tombe
R49 : si precede de consonne et suivi de consonne alors UM -> OM
R50 : si initiale et suivi de consonne alors UM -> OM
R51 : si precede de consonne et suivi de consonne alors UN -> ON
R52 : si initial et suivi de consonne alors UN -> ON
R53 : si precede de voyelle et suivi de voyelle alors X -> SS
R54 : si precede de consonne et suivi de consonne alors EM -> AM
R55 : si precede de consonne et suivi de consonne alors AM -> EM
R56 : si initial et suivi de consonne alors EM -> AM
R57 : si initial et suivi de consonne alors AM -> EM
R58 : si precede de consonne et suivi de consonne alors EN -> AN
R59 : si precede de consonne et suivi de consonne alors AN -> EN
R60 : si initial et suivi de consonne alors EN -> AN
R61 : si initial et suivi de consonne alors AN -> EN
R62 : si precede de consonne et suivi de consonne alors ER -> AR
```

R63 : si precede de consonne et suivi de consonne alors AR -> ER
R64 : si initial et suivi de consonne alors ER -> AR
R65 : si initial et suivi de consonne alors AR -> ER
R66 : si precede de consonne # [H,J,L,M,N,R,S,V] et
suivi de consonne # [H,R,S] alors ER -> RE
R67 : si precede de consonne # [G,H,J,L,M,N,R,S] et
suivi de consonne # [R,S] alors RE -> ER
R68 : si entre voyelles alors H tombe
R69 : si non (precede de TRE) et suivi de [CH,CL,CR] alors S tombe
R70 : si suivi de consonne alors CC -> C

; REGLES DE NIVEAU III1
; ---------------------

R01: si precede de consonne et suivi de consonne sauf IES final alors IE ->E
R02 : si precede de consonne et final alors I{ -> {
R03 : si precede de consonne et final alors I{S -> {S
R04 : si precede de voyelle et suivi de [E,I,{] alors C -> CH
R05 : si entre voyelles alors CH -> SS
R06 : si precede de consonne et suivi de voyelle alors CH -> S
R07 : si initial et suivi de [E,I,{] alors CH -> C
R08 : si final alors EZ -> ES
R09 : si suivi de [E,I,{] alors J -> G
R10 : si entre voyelles alors K -> SS
R11 : si precede de consonne et suivi de voyelle alors K -> S
R12 : si initial alors SK -> S
R13 : si entre voyelles alors S -> SS

R36 : si precede de V alors UI -> I

R38 : JU -> JEU
R39 : si suivi de voyelle alors K -> CH

R42 : W -> V
R43 : si precede de K alors I -> UI
R44 : si suivi de [A,O] alors G -> J
R45 : si suivi de voyelle # U alors G -> GU

```
; REGLES DE NIVEAU III.2
; ----------------------

; pas de regle de ce niveau pour l'instant
```

```
; REGLES DE NIVEAU III.3
; ----------------------

RO1 : si suivi de consonne alors { -> E
```

```
ERRORS : 103,101,101,

        %{
        # include <stdio.h>

        # define LGUNI 10

        typedef struct noeud {
                int type;
                char *mot[LGUNI];
                struct noeud *gau;
                struct noeud *dro;
                struct noeud *skNOEUD;

        NOEUD *racine;
        char retient[LGUNI];

        %}
        %start  axiome

        %union { NOEUD *res ; }

        %token  IDF  ALORS  CONSONNE  DE  ENTRE  FINAL  INITIAL NON
        %token  PRECEDE  RIEN  SAUF SI  SINON  SUIVI TOMBE  VOYELLE
        %token  LREG  REEC  GENEXP  ENSEXP  SANSP  CHAI  CHAIDF NOMBRE
        %token  FIN NFIN INI NINI

        %type <res> lregles  regle   lpremisses  premisse  tyype  ensemble
        %type <res> lchaine  chaine  action  expression  generique

        %left '+'
        %left '*'

        %%

        axiome          : lregles
                                { racine=$1; }
                        | /* vide */
                        ;
        lregles         : lregles regle
                                { $$=creer_noeud(LREG,"",$2,NULL,$1); }
                        | regle
                                { $$=creer_noeud(LREG,"",$1,NULL,NULL); }
                        ;
        regle           : chaine ':' SI lpremisses ALORS action
                                { $$=creer_noeud(SI,"",$1,$4,$6); }
                        | chaine ':' action
                                { $$=creer_noeud(SANSP,"",$1,$3,NULL); }
                        ;
        lpremisses      : premisse
                        | lpremisses '+' lpremisses
                                { $$=creer_noeud('+',"",$1,$3,NULL); }
                        | lpremisses '*' lpremisses
                                { $$=creer_noeud('*',"",$1,$3,NULL); }
                        | '(' lpremisses ')'
                                { $$=$2; }
                        | NON '(' lpremisses ')'
                                { $$=creer_noeud(NON,"",$3,NULL,NULL); }
                        ;
        premisse        : SUIVI DE tyype
                                { $$=creer_noeud(SUIVI,"",$3,NULL,NULL); }
                        | PRECEDE DE tyype
                                { $$=creer_noeud(PRECEDE,"",$3,NULL,NULL); }
                        | INITIAL
                                { $$=creer_noeud(INITIAL,"",NULL,NULL,NULL); }
                        | FINAL
                                { $$=creer_noeud(FINAL,"",NULL,NULL,NULL); }
                        | ENTRE generique
                                { $$=creer_noeud(ENTRE,"",$2,NULL,NULL); }
                        | SUIVI DE generique SAUF expression FINAL
                                { $$=creer_noeud(SAUF,"",$3,$5,NULL); }
                        | FINAL '=' tyype
                                { $$=creer_noeud(FIN,"",$3,NULL,NULL); }
                        | FINAL '#' tyype
                                { $$=creer_noeud(NFIN,"",$3,NULL,NULL); }
                        | INITIAL '=' tyype
                                { $$=creer_noeud(INI,"",$3,NULL,NULL); }
                        | INITIAL '#' tyype
                                { $$=creer_noeud(NINI,"",$3,NULL,NULL); }
                        ;
        tyype           : chaine
                        | chaine '#' expression
                                { $$=creer_noeud(CHAIDF,"",$1,$3,NULL); }
                        | generique
                        | generique '#' expression
                                { $$=creer_noeud(GENEXP,"",$1,$3,NULL); }
                        | ensemble
                        ;
        ensemble        : '[' lchaine ']'
                                { $$=$2; }
                        ;
        lchaine         : lchaine ',' chaine
                                { $$=creer_noeud(CHAI,"",$3,NULL,$1); }
                        | chaine
                        ;
        chaine          : IDF
```

```
                        { $$=creer_noeud(IDF,retient,NULL,NULL,NULL); }
                ;
action          : chaine '-' '>' chaine
                        { $$=creer_noeud(REEC,"",$1,$4,NULL); }
                | chaine TOMBE
                        { $$=creer_noeud(TOMBE,"",$1,NULL,NULL); }
                ;
expression      : chaine
                | ensemble
                ;
generique       : VOYELLE
                        { $$=creer_noeud(VOYELLE,"",NULL,NULL,NULL); }
                | CONSONNE
                        { $$=creer_noeud(CONSONNE,"",NULL,NULL,NULL); }
                ;

%%
# include "princ.c"
```

```
# define LGMAX 100
# define ERREUR 300

/* TABLE DES MOTS-CLES */

char nom_source[9];     /* nom physique du fichier source */
char nom_objet[9];      /* nom physique du fichier objet  */
FILE *source,*objet;    /* fichiers source et objet       */
int indice=0;           /* indice du caractere analyse de la ligne courante */
int nolig=0;            /* numero de ligne */
char lsource[LGMAX];    /* ligne source */
char lvide[LGMAX];
char lobjet[LGMAX];     /* ligne objet courante */
int ff=1 ;              /* detection de la fin de fichier */
char unite[LGUNI];            /* unite syntaxique */
int  nberr;             /* nombre d'erreurs de syntaxe */
int  maxii1;            /* plus grande longueur de membre gauche d'action */
char niveau;            /* variable qui indique le niveau des regles      */

/* LGUNI prevu a 10 */

struct mot_cle {char nom[LGUNI];
                int  svt;
                int  code;} tmc[27]=
       {"          ",0,0,
        "alors     ",0,ALORS,
        "entre     ",0,ENTRE,
        "consonne  ",8,CONSONNE,
        "de        ",0,DE,
        "et        ",2,'*',
        "final     ",7,FINAL,
        "finale    ",0,FINAL,
        "consonnes ",0,CONSONNE,
        "initial   ",10,INITIAL,
        "initiale  ",0,INITIAL,
        "          ",0,0,
        "          ",0,0,
        "          ",0,0,
        "non       ",0,NON,
        "ou        ",0,'+',
        "precede   ",0,PRECEDE,
        "          ",0,0,
        "rien      ",0,RIEN,
        "si        ",23,SI,
        "tombe     ",0,TOMBE,
        "voyelles  ",0,VOYELLE,
        "voyelle   ",21,VOYELLE,
        "sauf      ",24,SAUF,
        "sinon     ",25,SINON,
        "suivi     ",0,SUIVI,
        "          ",0,0};

# include "/proj/dialogue/souvay/commun/arbre.c"
# include "/proj/dialogue/souvay/commun/lex.c"
# include "/proj/dialogue/souvay/commun/gencommun.c"
# include "generation.c"

/*************************************************************************/
/*  PROGRAMME PRINCIPAL                                                  */
/*************************************************************************/


main ()
   {char c;
    char ligne[LGMAX];
    int i,longueur;
    static NOEUD* malloc();

    entete();
    for (i=0;i<LGMAX;++i) {
        lvide[i]=' ';
        lobjet[i]=' ';
                          }

    if (yyinit()==1) {
        racine=malloc(sizeof(NOEUD));
   printf("------------------------------------------------ \n\n");
        printf("analyse syntaxique en cours \n");
        yyparse();
        if (nberr==0) {
           printf("analyse syntaxique correcte.\n\n") ;
/*           printf("edition arbre o/n : ");
           c=getchar();
           if (c=='o'|c=='O') {printf("edition de l'arbre\n\n");
                               ecr_arb(racine);}*/
   printf("------------------------------------------------ \n\n");
```

```
            printf("generation en cours\n");
            generation(racine,niveau);
            printf("\n");

   printf("------------------------------------------------ \n\n");

            printf("fin de travail \n");
}
         else printf("%d erreur(s) de syntaxe\n",nberr);
}}


/*****************************************************************************/
/* ENTETE DE PROGRAMME                                                       */
/*****************************************************************************/

entete()

{  char c;

   home();
   printf("Compilateur de regle                  Version 0 \n");
   printf("------------------------------------------------ \n\n");

   printf("(a) regles II.1     (b) regles III.1 \n");
   printf("(c) regles III.2    (d) regles III.3 \n\n");
   printf("taper votre choix : ");
   c='e';
   while ( (c != 'a') && (c != 'b') && (c != 'c') && (c !='d') )
        { c=getchar(); }
   niveau=c;
   printf("\n");
   printf("------------------------------------------------ \n\n");
}

/*****************************************************************************/
/* EFFACEMENT DE L'ECRAN                                                     */
/*****************************************************************************/

home()

{printf("\033E");}
```

```
/*****************************************************************************/
/* TRANSFORMATION EN MAJUSCULE DES CARACTERES                                */
/*****************************************************************************/

majus(c)
char c;
   {if (c>='A' && c<='Z') {
        return(c-'a'+'A'); }
    else {
        return(c);}
   }


/*****************************************************************************/
/*  ANALYSEUR LEXICAL                                                        */
/*****************************************************************************/

yylex()

  {int i;
   int etat=0;          /* etat courant de l'automate */
   int action,lettre;   /* numero de l'action a effectuer et lettre lue */
   int lg=0;            /* longueur de l'unite syntaxique */
   char c;              /* caractere courant */

   static int teta[3][8] = {
                            {1,-2,0,0,-3,-4,2,0},{1,-1,-1,-1,-1,-4,1,-1},
                            {-5,-5,-5,-5,-5,-4,2,-5}
                           } ;
                        /* automate d'analyse */
   static int tact[3][8] = {
                            {1,2,0,3,6,4,1,3},{1,5,5,5,5,4,1,5},
                            {7,7,7,7,7,4,1,7}
                           } ;
                        /* table des actions */


/*   strcpy(unite,"          "); */
   for (i=0;i<LGUNI;++i) { unite[i] = ' ' ; }
   while (etat>=0)
      {c=lsource[indice];
       c=(c == '|')?c='?':c;
       lettre=det_lettre(c);
       action=tact[etat][lettre];
       etat=teta[etat][lettre];
       switch(action)
          {case 0:++indice;break;
           case 1:if (++lg<LGUNI) {
                     unite[lg-1]=c;}
                  ++indice;
                  break;
           case 2:++indice;
                  return (c);
           case 3:if (ff==0) {
                     return(EOF);}
                  else {
                     indice=0;
                     lire(lsource,LGMAX);
                     break; }
           case 4:++indice;
                  return(YYERRCODE);
           case 5:strcpy(retient,unite);
                  return(calcul());
           case 6:return(EOF);
           case 7:strcpy(retient,unite);
                  return(NOMBRE);}
       } /* while */ }

/*****************************************************************************/
/*  RECHERCHE DE LA LIGNE SUIVANTE                                           */
/*****************************************************************************/

lire(ligne,taille)
char ligne[];
int taille;
   {int c,i;

    ++nolig;
    for (i=0;i<LGMAX-1;++i) { ligne[i]=' '; }
    for (i=0;i<taille-1 && (ff=(c=getc(source))!=EOF) && c!='\n';++i)
        ligne[i]=c;
    if (c=='\n') {
       ligne[i]=c;
       ++i;
                }
```

```
    if (ff==0)
    ligne[i]='\0';
    return(i);}

/**********************SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS(T0;
/*  DETERMINATION DE LA LETTRE                                          */
/*************************************************************************/

det_lettre(c)
char c;
  {if ((c>='A' && c<='Z') || (c>='a' && c<='z'))
     { return(0);}
   else
      if (c>='0' && c<='9')
         { return(6); }
      else
         switch(c)
            {case ':':  return(1);
             case '+':  case '*':        case '=':
             case '#':  case '[':        case ']':        case ',':
             case '(':  case ')':        case '>':        case '-':return(1);
             case '\n': return(3);
             case ' ':  return(2);
             case ';':  return(7);
/*            case '\': */
             case '/':       case '@':   case '|':
             case '{':  case '}':       case '_':        case '?':
                                                                  return(0);
             default:return((ff==0)?4:5);}
}

/*************************************************************************/
/*  RECHERCHE DU CODE DU MOT-CLE                                        */
/*************************************************************************/

calcul()
   {int hcode=unite[0]-'a'+1;
    int fin=0;
    int trouve=0;

    if (hcode<1 || hcode>26) { return(IDF); }
    while (fin==0) {
        trouve=(strcmp(tmc[hcode].nom,unite)==0);
        if (!trouve) {
           hcode=tmc[hcode].svt;}
        fin=((hcode==0) || trouve || (tmc[hcode].nom[0] !=unite[0]));
                   }
    if (trouve !=0) {
       return(tmc[hcode].code);}
    else {return(IDF);}
   }

/*************************************************************************/
/*  INITIALISATION DE L'ANALYSEUR LEXICAL                               */
/*************************************************************************/

yyinit ()
{   printf("nom du fichier source : ");
    scanf("%8s",nom_source);
    printf("\n");
    if ((source=fopen(nom_source,"r"))==NULL) {
        printf("fichier inexistant\n");
        return(0); }
    else {printf("nom du fichier objet  : ");
          scanf("%8s",nom_objet);
          printf("\n");
          objet=fopen(nom_objet,"w");
          lire(lsource,LGMAX);
          return(1);}
   }

/*************************************************************************/
/* AIDE A LA MISE AU POINT                                              */
/*************************************************************************/

editer(code)
int code;
   {printf("----------------------------------\n");
    printf("%s\n",lsource);
    lvide[indice]='^';
    printf("%s\n",lvide);
    lvide[indice]=' ';
    printf("%s --> %d \n",unite,code);}
```

```
# line 2 "gII1"
# include <stdio.h>

# define LGUNI 10

typedef struct noeud {
          int type;
          char *mot[LGUNI];
          struct noeud *gau;
          struct noeud *dro;
          struct noeud *svt;} NOEUD;

NOEUD *racine;
char retient[LGUNI];


# line 19 "gII1"
typedef union  { NOEUD *res ; } YYSTYPE;
# define IDF 257
# define ALORS 258
# define CONSONNE 259
# define DE 260
# define ENTRE 261
# define FINAL 262
# define INITIAL 263
# define NON 264
# define PRECEDE 265
# define RIEN 266
# define SAUF 267
# define SI 268
# define SINON 269
# define SUIVI 270
# define TOMBE 271
# define VOYELLE 272
# define LREG 273
# define REEC 274
# define GENEXP 275
# define ENSEXP 276
# define SANSP 277
# define CHAI 278
# define CHAIDF 279
# define NOMBRE 280
# define FIN 281
# define NFIN 282
# define INI 283
# define NINI 284
#define yyclearin yychar = -1
#define yyerrok yyerrflag = 0
extern int yychar;
extern short yyerrflag;
#ifndef YYMAXDEPTH
#define YYMAXDEPTH 150
#endif
YYSTYPE yylval, yyval;
# define YYERRCODE 256

# line 111 "gII1"

# include "princ.c"
short yyexca[] ={
-1, 1,
        0, -1,
        -2, 0,
        };
# define YYNPROD 37
# define YYLAST 237
short yyact[]={

  13,  46,  35,  20,  56,   5,  67,  28,  46,  27,
  24,  23,   5,  36,  66,  34,   8,  44,   4,  32,
   4,  30,  45,   7,  60,  10,  54,  24,  23,  40,
  24,  23,  24,  26,  56,  57,  42,   9,   3,  58,
  10,   6,  12,   2,   1,  31,   0,  29,   0,   0,
   0,   0,   0,   0,  53,   0,   0,  48,   0,   0,
  37,   0,  11,  65,  59,  47,  49,  50,  51,  52,
   0,   0,   0,  61,  61,  61,  25,  33,  62,  62,
  62,  63,  64,   0,  68,  43,  38,  39,   0,  41,
   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0,   0,   0,   0,   0,   0,   5,   0,  35,
```

```
   0,   0,   0,   0,   5,   0,   0,   0,   0,   0,
   0,   0,  34,   0,   0,   0,   0,   0,   0,   0,
   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,  19,  18,  17,  14,  16,  22,   0,   0,  21,
  15,   0,   0,   0,   0,   0,  55 };
short yypact[]={

-245,-1000,-245,-1000, -35,-1000,-1000,-252, -40,-1000,
 -42, -32,-1000, -40,  -7,-251,-253, -14, -16,-257,
 -49,-1000,-245, -40, -40, -12, -40, -90, -90, -90,
 -90, -90, -90,-1000,-1000,-1000,-245,-1000, -10,-1000,
-1000, -15,-1000, -31,   0,-1000,-245,-1000,  -1,-1000,
-1000,-1000,-1000,-1000,-1000, -83, -83, -83, -30,-1000,
-256,-1000,-1000,-1000,-1000,-1000,-245,-1000,-1000 };
short yypgo[]={

   0,  44,  43,  38,  62,  42,  36,  22,  39,  17,
  37,  24,  57 };
short yyr1[]={

   0,   1,   1,   2,   2,   3,   3,   4,   4,   4,
   4,   4,   5,   5,   5,   5,   5,   5,   5,   5,
   5,   5,   6,   6,   6,   6,   6,   7,   8,   8,
   9,  10,  10,  11,  11,  12,  12 };
short yyr2[]={

   0,   1,   0,   2,   1,   6,   3,   1,   3,   3,
   3,   4,   3,   3,   1,   1,   2,   6,   3,   3,
   3,   3,   1,   3,   1,   3,   1,   3,   3,   1,
   1,   4,   2,   1,   1,   1,   1 };
short yychk[]={

-1000,  -1,  -2,  -3,  -9, 257,  -3,  58, 268, -10,
  -9,  -4,  -5,  40, 264, 270, 265, 263, 262, 261,
  45, 271, 258,  43,  42,  -4,  40, 260, 260,  61,
  35,  61,  35, -12, 272, 259,  62, -10,  -4,  -4,
  41,  -4,  -6, -12,  -9,  -7,  91,  -6, -12,  -6,
  -6,  -6,  -6,  -9,  41, 267,  35,  35,  -8,  -9,
 -11,  -9,  -7, -11, -11,  93,  44, 262,  -9 };
short yydef[]={

   2,  -2,   1,   4,   0,  30,   3,   0,   0,   6,
   0,   0,   7,   0,   0,   0,   0,  14,  15,   0,
   0,  32,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0,   0,  16,  35,  36,   0,   5,   8,   9,
  10,   0,  12,  24,  22,  26,   0,  13,  24,  20,
  21,  18,  19,  31,  11,   0,   0,   0,   0,  29,
   0,  33,  34,  25,  23,  27,   0,  17,  28 };
#ifndef lint
static char yaccpar_sccsid[] = "@(#)yaccpar     4.1     (Berkeley)      2/11/83";
#endif not lint

#
# define YYFLAG -1000
# define YYERROR goto yyerrlab
# define YYACCEPT return(0)
# define YYABORT return(1)

/*      parser for yacc output  */

#ifdef YYDEBUG
int yydebug = 0; /* 1 for debugging */
#endif
YYSTYPE yyv[YYMAXDEPTH]; /* where the values are stored */
int yychar = -1; /* current input token number */
int yynerrs = 0;  /* number of errors */
short yyerrflag = 0;  /* error recovery flag */

yyparse() {

        short yys[YYMAXDEPTH];
        short yyj, yym;
        register YYSTYPE *yypvt;
        register short yystate, *yyps, yyn;
        register YYSTYPE *yypv;
        register short *yyxi;

        yystate = 0;
        yychar = -1;
        yynerrs = 0;
        yyerrflag = 0;
        yyps= &yys[-1];
        yypv= &yyv[-1];
```

```
yystack:    /* put a state and value onto the stack */

#ifdef YYDEBUG
        if( yydebug  ) printf( "state %d, char 0%o\n", yystate, yychar );
#endif
                if( ++yyps> &yys[YYMAXDEPTH] ) { yyerror( "yacc stack overflow" ); return(1); }
                *yyps = yystate;
                ++yypv;
                *yypv = yyval;

yynewstate:

        yyn = yypact[yystate];

        if( yyn<= YYFLAG ) goto yydefault; /* simple state */

        if( yychar<0 ) if( (yychar=yylex())<0 ) yychar=0;
        if( (yyn += yychar)<0 || yyn >= YYLAST ) goto yydefault;

        if( yychk[ yyn=yyact[ yyn ] ] == yychar ){ /* valid shift */
                yychar = -1;
                yyval = yylval;
                yystate = yyn;
                if( yyerrflag > 0 ) --yyerrflag;
                goto yystack;
                }

yydefault:
        /* default state action */

        if( (yyn=yydef[yystate]) == -2 ) {
                if( yychar<0 ) if( (yychar=yylex())<0 ) yychar = 0;
                /* look through exception table */

                for( yyxi=yyexca; (*yyxi!= (-1)) || (yyxi[1]!=yystate) ; yyxi += 2 ) ; /* VOID */

                while( *(yyxi+=2) >= 0 ){
                        if( *yyxi == yychar ) break;
                        }
                if( (yyn = yyxi[1]) < 0 ) return(0);   /* accept */
                }

        if( yyn == 0 ){ /* error */
                /* error ... attempt to resume parsing */

                switch( yyerrflag ){

                case 0:   /* brand new error */

                        yyerror( "syntax error" );
                yyerrlab:
                        ++yynerrs;

                case 1:
                case 2: /* incompletely recovered error ... try again */

                        yyerrflag = 3;

                        /* find a state where "error" is a legal shift action */

                        while ( yyps >= yys ) {
                           yyn = yypact[*yyps] + YYERRCODE;
                           if( yyn>= 0 && yyn < YYLAST && yychk[yyact[yyn]] == YYERRCODE ){
                              yystate = yyact[yyn];  /* simulate a shift of "error" */
                              goto yystack;
                              }
                           yyn = yypact[*yyps];

                           /* the current yyps has no shift onn "error", pop stack */

#ifdef YYDEBUG
                           if( yydebug ) printf( "error recovery pops state %d, uncovers %d\n", *yyps, yyps[-1] );
#endif
                           --yyps;
                           --yypv;
                           }

                        /* there is no state on the stack with an error shift ... abort */

        yyabort:
                        return(1);


                case 3:  /* no shift yet; clobber input char */

#ifdef YYDEBUG
```

```
                        if( yydebug ) printf( "error recovery discards char %d\n", yychar );
#endif

                        if( yychar == 0 ) goto yyabort; /* don't discard EOF, quit */
                        yychar = -1;
                        goto yynewstate;   /* try again in the same state */

                        }

                }

        /* reduction by production yyn */

#ifdef YYDEBUG
                if( yydebug ) printf("reduce %d\n",yyn);
#endif
                yyps -= yyr2[yyn];
                yypvt = yypv;
                yypv -= yyr2[yyn];
                yyval = yypv[1];
                yym=yyn;
                        /* consult goto table to find next state */
                yyn = yyr1[yyn];
                yyj = yypgo[yyn] + *yyps + 1;
                if( yyj>=YYLAST || yychk[ yystate = yyact[yyj] ] != -yyn ) yystate = yyact[yypgo[yyn]];
                switch(yym){

case 1:
# line 35 "gII1"
{ racine=yypvt[-0].res; } break;
case 3:
# line 39 "gII1"
{ yyval.res=creer_noeud(LREG,"",yypvt[-0].res,NULL,yypvt[-1].res); } break;
case 4:
# line 41 "gII1"
{ yyval.res=creer_noeud(LREG,"",yypvt[-0].res,NULL,NULL); } break;
case 5:
# line 44 "gII1"
{ yyval.res=creer_noeud(SI,"",yypvt[-5].res,yypvt[-2].res,yypvt[-0].res); } break;
case 6:
# line 46 "gII1"
{ yyval.res=creer_noeud(SANSP,"",yypvt[-2].res,yypvt[-0].res,NULL); } break;
case 8:
# line 50 "gII1"
{ yyval.res=creer_noeud('+',"",yypvt[-2].res,yypvt[-0].res,NULL); } break;
case 9:
# line 52 "gII1"
{ yyval.res=creer_noeud('*',"",yypvt[-2].res,yypvt[-0].res,NULL); } break;
case 10:
# line 54 "gII1"
{ yyval.res=yypvt[-1].res; } break;
case 11:
# line 56 "gII1"
{ yyval.res=creer_noeud(NON,"",yypvt[-1].res,NULL,NULL); } break;
case 12:
# line 59 "gII1"
{ yyval.res=creer_noeud(SUIVI,"",yypvt[-0].res,NULL,NULL); } break;
case 13:
# line 61 "gII1"
{ yyval.res=creer_noeud(PRECEDE,"",yypvt[-0].res,NULL,NULL); } break;
case 14:
# line 63 "gII1"
{ yyval.res=creer_noeud(INITIAL,"",NULL,NULL,NULL); } break;
case 15:
# line 65 "gII1"
{ yyval.res=creer_noeud(FINAL,"",NULL,NULL,NULL); } break;
case 16:
# line 67 "gII1"
{ yyval.res=creer_noeud(ENTRE,"",yypvt[-0].res,NULL,NULL); } break;
case 17:
# line 69 "gII1"
{ yyval.res=creer_noeud(SAUF,"",yypvt[-3].res,yypvt[-1].res,NULL); } break;
case 18:
# line 71 "gII1"
{ yyval.res=creer_noeud(FIN,"",yypvt[-0].res,NULL,NULL); } break;
case 19:
# line 73 "gII1"
{ yyval.res=creer_noeud(NFIN,"",yypvt[-0].res,NULL,NULL); } break;
case 20:
# line 75 "gII1"
{ yyval.res=creer_noeud(INI,"",yypvt[-0].res,NULL,NULL); } break;
case 21:
# line 77 "gII1"
{ yyval.res=creer_noeud(NINI,"",yypvt[-0].res,NULL,NULL); } break;
case 23:
# line 81 "gII1"
```

```
{ yyval.res=creer_noeud(CHAIDF,"",yypvt[-2].res,yypvt[-0].res,NULL); } break;
case 25:
# line 84 "gII1"
{ yyval.res=creer_noeud(GENEXP,"",yypvt[-2].res,yypvt[-0].res,NULL); } break;
case 27:
# line 88 "gII1"
{ yyval.res=yypvt[-1].res; } break;
case 28:
# line 91 "gII1"
{ yyval.res=creer_noeud(CHAI,"",yypvt[-0].res,NULL,yypvt[-2].res); } break;
case 30:
# line 95 "gII1"
{ yyval.res=creer_noeud(IDF,retient,NULL,NULL,NULL); } break;
case 31:
# line 98 "gII1"
{ yyval.res=creer_noeud(REEC,"",yypvt[-3].res,yypvt[-0].res,NULL); } break;
case 32:
# line 100 "gII1"
{ yyval.res=creer_noeud(TOMBE,"",yypvt[-1].res,NULL,NULL); } break;
case 35:
# line 106 "gII1"
{ yyval.res=creer_noeud(VOYELLE,"",NULL,NULL,NULL); } break;
case 36:
# line 108 "gII1"
{ yyval.res=creer_noeud(CONSONNE,"",NULL,NULL,NULL); } break;
                }
                goto yystack;  /* stack new state and value */

        }
```

```
/*************************************************************************/
/* ERREUR DE SYNTAXE                                                     */
/*************************************************************************/

yyerror()
   {printf("*** erreur de syntaxe ligne %d\n",nolig);
    printf("%s\n",lsource);
    lvide[indice-1]='^';
    printf("%s\n",lvide);
    lvide[indice]=' ';
    nberr++; }


/*************************************************************************/
/*  CREATION D'UN NOEUD DE L'ARBRE SYNTAXIQUE                            */
/*************************************************************************/

NOEUD *creer_noeud(type,mot,gau,dro,svt)

NOEUD *gau,*dro,*svt;
int type;
char *mot[10];
   {NOEUD *p;
    static NOEUD* malloc ();

    p=malloc(sizeof(NOEUD));
    p->type = type;
    if (type == 257) strcpy(p->mot ,retient);
    else strcpy(p->mot,"");
    p->gau = gau;
    p->dro = dro;
    p->svt = svt;
    return(p); }

/*************************************************************************/
/* EDITION DE L'ARBRE SYNTAXIQUE                                         */
/*************************************************************************/

ecr_arb(p)
NOEUD *p;
   {if (p!=NULL)
       {printf("-----------------------------------------------\n");
        printf("type=%d    mot=>%s<\n",p->type,p->mot);
        printf("gau =%d    dro=%d        svt=%d\n",
            (p->gau)->type ,(p->dro)->type,(p->svt)->type);
        ecr_arb(p->gau);
        ecr_arb(p->dro);
        ecr_arb(p->svt);}}
```

```
/*************************************************************************/
/* GENERATION DES REGLES LISP                                            */
/*************************************************************************/

generation(n,c)

NOEUD *n;
char  c;

{  copie(1,23,"; LISTE DES REGLES II.1");
   sortie(1,23);
   sortie(1,1);
   switch(c)
     {case 'a':copie(1,24,"(setq #:mf:reglesII1 '( ");
               break;
      case 'b':copie(1,24,"(setq #:mf:reglesIII1 '( ");
               break;
      case 'c':copie(1,24,"(setq #:mf:reglesIII2 '( ");
               break;
      case 'd':copie(1,24,"(setq #:mf:reglesIII3 '( ");
               break;}
   sortie(1,24);
   genlregles(n);
   sortie(1,1);
   copie(1,2,"))");
   sortie(1,2);
   sortie(1,1);
   switch (c)
     {case 'a':copie(1,21,"(setq #:mf:maxII1   )     ");
               break;
      case 'b':copie(1,21,"(setq #:mf:maxIII1  )     ");
               break;
      case 'c':copie(1,21,"(setq #:mf:maxIII2  )     ");
               break;
      case 'd':copie(1,21,"(setq #:mf:maxIII3  )     ");
               break;}
   if (maxii1<10)
      {lobjet[20]='0'+maxii1;}
   else
      {lobjet[19]='0'+(maxii1/10);
       lobjet[20]='0'+(maxii1%10);}
   sortie(1,21);
}


/*************************************************************************/
/* GENERATION AU NIVEAU DE lregles                                       */
/*************************************************************************/

genlregles(n)

NOEUD *n;

   {if (n != NULL) {
       switch(n->type)
          {case  LREG:genlregles(n->svt);
                      genregle(n->gau);
                      break;
              default:genregle(n); }}}



/*************************************************************************/
/* GENERATION POUR regle                                                 */
/*************************************************************************/

genregle(n)

NOEUD *n;

   {switch(n->type)
       {case     SI:gensi(n);
                    break;
        case  SANSP:gensansp(n);
                    break;
       }}

/*************************************************************************/
/* GENERATION POUR reecriture DANS LE CAS OU IL N'Y A PAS DE PREMISSE    */
/*************************************************************************/

gensansp(n)

NOEUD *n;
```

```
   {char  *mg;
    char  *md;
    int lg,i;

    mg=((n->dro)->gau)->mot;
    md=((n->dro)->dro)->mot;
    controle3(mg,md,(n->gau)->mot);
    sortie(1,1);
    copie(1,6,"(     ");
    copie(3,10,(n->gau)->mot);
    sortie(1,12);sortie(1,1);
    lg=lgchaine(mg);
    maxii1=(lg>maxii1)?lg:maxii1;
    copie(1,41,"(if (equal milieu '(                    ");
    for (i=21;i<21+2*lg-1;i+=2) { lobjet[i]= *mg++; }
    copie(21+2*lg-1,2,"))     ");
    sortie(1,41);
    switch((n->dro)->type)
         {case   REEC:lg=lgchaine(md);
                      copie(5,50,
                      "(list (append alpha '(                          ");
                      for(i=27;i<27+2*lg-1;i+=2) { lobjet[i]= *md++; }
                      copie(27+2*lg-1,10,") beta))))     ");
                      sortie(5,51);
                      break;
          case  TOMBE:copie(6,28,"(list (append alpha beta))))       ");
                      sortie(6,28);
                      break;
              default:errpgt(2,(n->dro)->type);

}}


/*************************************************************************/
/* ERREUR DE PROGRAMMATION                                               */
/*************************************************************************/

errpgt(i,type)
int i,type;

   {switch(i)
       {case 1:printf("erreur de programmation genregles   : ");
               break;
        case 2:printf("erreur de programmation gensansp    : ");
               break;
        case 3:printf("erreur de programmation genpremisse : ");

       }
     printf("%d \n",type);
   }


/*************************************************************************/
/* GENERATION POUR si alors                                              */
/*************************************************************************/

gensi(n)
NOEUD *n;
   {char  *mg;
    char  *md;
    int lg,i;

    mg=((n->svt)->gau)->mot;
    md=((n->svt)->dro)->mot;
    controle3(mg,md,(n->gau)->mot);
    sortie(1,1);
    copie(1,2,"( ");copie(3,10,(n->gau)->mot);
    sortie(1,12);
    sortie(1,1);
    lg=lgchaine(mg);
    maxii1=(lg>maxii1)?lg:maxii1;
    copie(1,47,"(if (and  (equal milieu '(                    ");
    for (i=27;i<27+2*lg-1;i+=2) { lobjet[i]= *mg++; }
    copie(27+2*lg-1,2,")) ");
    sortie(1,47);
    genlpremisses(n->dro,11,(n->gau)->mot);
    copie(5,1,")");
    sortie(5,1);
    if ((n->svt)->dro==NULL)
       {copie(5,28,"(list (append alpha beta))))  ");
        sortie(5,28);}
    else
       {lg=lgchaine(md);
        copie(5,50,"(list (append alpha '(                          ");
        for (i=27;i<27+2*lg-1;i+=2) { lobjet[i]= *md++; }
```

```
        copie(27+2*lg-1,10,") beta)))) ");
        sortie(5,51);}
    }


/*************************************************************************/
/* GENERATION POUR lpremisses                                            */
/*************************************************************************/

genlpremisses(n,alignement,nomreg)

NOEUD *n;
int alignement;
char *nomreg;

   {switch(n->type)
       {case NON:
        case '+':
        case '*':genetounon(n,alignement,nomreg);
                 break;
         default:genpremisse(n,alignement,nomreg);}}


/*************************************************************************/
/*  GENERATION POUR lpremisses ET/OU lpremisses                          */
/*************************************************************************/

genetounon(n,alignement,nomreg)

NOEUD *n;
int alignement;
char *nomreg;

   {switch(n->type)
       {case '*':copie(alignement,5,"(and   ");
                 sortie(alignement,5);
                 break;
        case '+':copie(alignement,5,"(or    ");
                 sortie(alignement,5);
                 break;
        case NON:copie(alignement,5,"(not   ");
                 sortie(alignement,5);
                 break;
        }
    genlpremisses(n->gau ,alignement+5,nomreg);
    if (n->type != NON ) genlpremisses(n->dro ,alignement+5,nomreg);
    copie(alignement,1,")");
    sortie(alignement,1);}



/*************************************************************************/
/* GENERATION POUR premisse                                              */
/*************************************************************************/

genpremisse(n,alignement,nomreg)

NOEUD *n;
int alignement;
char *nomreg;

{int i,j,lg;
 char *m;
 NOEUD *s;

   switch(n->type)
       {case SUIVI:
        case PRECEDE:gensuipre(n,alignement,nomreg,0);
                     break;
        case FINAL:copie(alignement,13,"(finale beta)");
                   sortie(alignement,13);
                   break;
        case INITIAL:copie(alignement,16,"(initiale alpha)");
                     sortie(alignement,16);
                     break;
        case ENTRE:copie(alignement,5,"(and ");
                   spconst(alignement+5,PRECEDE,'3',0);
                   genvoycons((n->gau)->type,alignement+5,0);
                   sortie(alignement,34);
                   spconst(alignement+5,SUIVI,'3',0);
                   genvoycons((n->gau)->type,alignement+5,nomreg,0);
                   sortie(alignement+5,29);
                   lobjet[alignement]=')';
                   sortie(alignement,1);
                   break;
       case SAUF:copie(alignement,35,"(and (suivi_de  3 beta               ");
```

```
                genvoycons( (n->gau)->type,alignement+5,0);
                sortie(alignement,35);
                gensfchai(n->dro,alignement+5);
                lobjet[alignement]=')';
                sortie(alignement,1);
                break;
        case  FIN:s=creer_noeud(PRECEDE,"",n->gau,NULL,NULL);
                gensuipre(s,alignement,nomreg,1);
                break;
        case NFIN:copie(alignement,5,"(not     ");
                sortie(alignement,5);
                s=creer_noeud(PRECEDE,"",n->gau,NULL,NULL);
                gensuipre(s,alignement,nomreg,1);
                lobjet[alignement]=')';sortie(alignement,1);
                break;
        case  INI:s=creer_noeud(SUIVI,"",n->gau,NULL,NULL);
                gensuipre(s,alignement,nomreg,1);
                break;
        case NINI:copie(alignement,5,"(not     ");
                sortie(alignement,5);
                s=creer_noeud(SUIVI,"",n->gau,NULL,NULL);
                gensuipre(s,alignement,nomreg,1);
                lobjet[alignement]=')';sortie(alignement,1);
                break;
           default:errpgt(3,n->type);}}


/*****************************************************************************/
/* GENERATION POUR precede de et suivi de                                    */
/*                                                                           */
/* n       : noeud courant            alignement : marge gauche              */
/* nomreg : numero de la regle        simule : noeuds FIN NFIN INI NINI      */
/*****************************************************************************/

gensuipre(n,alignement,nomreg,simule)

NOEUD *n;
int alignement;
char *nomreg;
int simule;

   {NOEUD *p;
    int i,j,lg,idx;
    char  *m;

    idx=(simule==1)?42:20;
    p=n->gau;m=p->mot;
    switch(p->type)
       {case CHAI:spconst(alignement,n->type,'7',simule);
                genchai(n->gau,alignement+idx);
                lobjet[alignement]='}';
                sortie(alignement,1);
                break;
        case VOYELLE:
        case CONSONNE:spconst(alignement,n->type,'3',simule);
                    genvoycons(p->type,alignement,simule);
                    sortie(alignement,idx+9);
                    break;
        case GENEXP:controle1(n->type,(p->gau)->type,p->dro,nomreg);
                   spconst(alignement,n->type,'4',simule);
                   genvoycons((p->gau)->type,alignement,simule);
                   lobjet[alignement+idx+8]=' ';
                   genchai(p->dro,alignement+idx+9);
                   lobjet[alignement]=')';
                   sortie(alignement,1);
                   break;
        case IDF:if ((lg=lgchaine(p->mot))==1)
                   {spconst(alignement,n->type,'1',simule);
                    copie(alignement+idx,4,"' )  ");
                    lobjet[alignement+idx+1]= *m;
                    sortie(alignement,idx+4);}
                 else
                   {spconst(alignement,n->type,'2',simule);
                    copie(alignement+idx,2,"'(");
                    j=alignement+idx+2+(2*lg-1);
                    for (i=alignement+idx+2;i<j+(2*lg-1);i+=2)
                        { lobjet[i]= *m++; }
                    copie(j,2,"))  ");
                    sortie(alignement,idx+2+2*lg+1);}
                 break;
        case CHAIDF:controle2();
                    spconst(alignement,n->type,'5',simule);
                    copie(alignement+idx,2,"'(                          ");
                    lg=lgchaine(m=(p->gau)->mot);
                    j=alignement+idx+2+(2*lg-1);
                    for (i=alignement+idx+2;i<j;i+=2)
```

```
                    { lobjet[i]= *m++;}
                lobjet[j]=')';
                genchai(p->dro,j+2);
                lobjet[alignement]=')';
                sortie(alignement,1);
                break;

        }
    }

/*****************************************************************************/
/* GENERATION DE LA LIGNE POUR SUIVI DE ET PRECEDE DE                        */
/*                                                                           */
/* alignement : marge gauche      type : SUIVI ou PRECEDE                    */
/* nombre     : genre de precede_de ou suivi_de                              */
/* simule     : noeuds FIN NFIN INI NINI                                     */
/*****************************************************************************/

spconst(alignement,type,nombre,simule)

int alignement,type;
char nombre;

{  if (simule != 1)
     {if (type==SUIVI)
       copie(alignement,47,"(suivi_de     beta                              ");
      else
       copie(alignement,47,"(precede_de   alpha                             ");
     }
   else
     {if (type==SUIVI)
        copie(alignement,69,
  "(suivi_de     (append alpha milieu beta)                                  ");
      else
        copie(alignement,69,
  "(precede_de   (append alpha milieu beta)                                  ");
     }
  lobjet[alignement+12]=nombre;
}


/*****************************************************************************/
/* GENERATION DU MOT VOYELLE OU CONSONNE                                     */
/*                                                                           */
/*      voycons    : voyelle ou consonne                                     */
/*      alignement : marge gauche                                            */
/*      simule     : noeuds FIN NFIN INI NINI                                */
/*****************************************************************************/

genvoycons(voycons,alignement,simule)

int voycons,alignement;

{  int i;

   i=(simule==1)?42:20;
   {if (voycons==VOYELLE)
       copie(alignement+i,9," voyelle)   ");
    else
       copie(alignement+i,9,"consonne)   ");
   }
}


/*****************************************************************************/
/* generation pour CHAI                                                      */
/*****************************************************************************/

genchai(n,alignement)

NOEUD *n;
int alignement;

   {int i,lg;
    char *m;

    if (n!=NULL)
       if (n->type == CHAI)
          {genchai(n->svt,alignement);
           genchai(n->gau,alignement);}
       else
          {lg=lgchaine(m=(n->mot));
           copie(alignement,4,"'(                    ");
           for (i=alignement+2;i<alignement+2+(2*lg-1);i+=2)
               { lobjet[i]= *m++; }
           lobjet[alignement+2+(2*lg-1)]=')';
```

```
        sortie(5,alignement+2+(2*lg-1)-4);}
  }


/*************************************************************************/
/* CONTROLE D'ERREUR                                                     */
/*************************************************************************/

controle1(suipre,voycons,n,nomreg)
int suipre,voycons;
NOEUD *n;
char *nomreg;
  {}

controle2(){}


/*************************************************************************/
/*  CONTROLE DU NON BOUCLAGE DES REGLES    Version 0                     */
/*************************************************************************/

controle3(mg,md,nomreg)
char *mg,*md;
char *nomreg;


{ int lg1,lg2;

  if (strcmp(mg,md)==0)
     {printf("*** CII1.V0 : la regle %s n'effectue rien \n",nomreg);}
   else
     {lg1=lgchaine(mg);lg2=lgchaine(md);
      if ( (lg1<lg2) && ( inclu(mg,md)==1) )
         {printf("*** CII1.V0 : verifier que cette regle");
          printf(" ne provoque pas de bouclage \n");}
     }
}
inclu(mg,md)
char *mg,*md;
{ return((sousch(mg,md) == -1)?0:1);}

/*************************************************************************\
*                                                                         *
* sousch(s,t)                                                             *
* renvoie l'index (dans s) du premier carate`re de de t si t est une sous *
* chaine de s; sinon renvoie -1                                           *
* aq                                                                      *
*                                                                         *
\*************************************************************************/
sousch(s,t)
char s[],t[];
{int i,j,k;
        for (i = 0; s[i] != '\0'; i++){
                for (j=i, k=0; t[k] != '/0' && s[j]==t[k]; j++, k++);
                if (t[k] == '\0') return(i);
        }
        return(-1);
}

/*-------------------------------------------------------------------------*/
/*            GENERATION DE ensemble DANS PREMISSE SAUF                    */
/*-------------------------------------------------------------------------*/

gensfchai(n,alignement)

NOEUD *n;
int alignement;

{  int i,j,lg;
   char *m;

   if ( n != NULL )
      if ( n->type == CHAI )
         {gensfchai(n->svt,alignement);
          gensfchai(n->gau,alignement);}
      else
         {copie(alignement,40,
             "(nequal '(                                                ");
          lg=lgchaine(m=(n->mot));
          j=alignement+10+(2*lg-1);
          for ( i=alignement+10 ; i<j ; i+=2 ) { lobjet[i] = *m++ ; }
          copie(j,10,") beta)            ");
          sortie(alignement,40);}
}
```

```
/*****************************************************************************/
/* ECRITURE DANS LE FICHIER OBJET                                            */
/*****************************************************************************/

sortie(debut,lg)

int debut,lg;

   {int i;

/*    printf("%s\n",lobjet);   */
    for (i=1; i < (debut>5?(debut-5):1); i++) putc(' ',objet);
    for (i=(debut>5?(debut-5):1);i<debut+lg;i++) putc(lobjet[i],objet);
    putc('\n',objet);
    for (i=(debut>5?(debut-5):1);i<lg+debut;++i) lobjet[i]=' ';
   }


/*****************************************************************************/
/* COPIE D"UNE CHAINE DE CARACTERE DANS LA LIGNE OBJET                       */
/*****************************************************************************/

copie(debut,lg,chaine)

int debut,lg;
char chaine[];

     {int i;

      for (i=debut;i<=lg+debut;++i) lobjet[i]=chaine[i-debut];}


/*****************************************************************************/
/*                                                                           */
/*****************************************************************************/

lgchaine(mot)
char *mot;
   {int i=0;
    while ((*mot++ != ' ') | (i==LGUNI)){++i;}
    return(i);}
```

ANNEXE 3

Les règles II de niveau 2

1. Les règles

2. Le grammaire algébrique

3. Le compilateur.

3.1. le programme principal	**princ2.c**
3.2. l'analyseur lexical	**lex2.c**
3.3. l'analyseur syntaxique	**y.tab.c**
3.4. la génération	**gen2.c**

```
;  LISTE DES REGLES DE NIVEAU II.2 systeme nominal
; ------------------------------------------------

R1  : si [O,S] alors D -> T
R2  : si [O,S] alors G -> K
R3  : si [O,S] et precede de voyelle # [IA,EA] alors U -> L
R4  : si [O,S] et precede de consonne alors AU -> AIL
R5  : si [O,S] alors EAU -> EL
R6  : si [O,S] alors IAU -> EL
R7  : si [O,S] alors NG -> N
R8  : si S et finale # [E,U,L,D,K,F,P,G] alors ajouter [T,F,K,L,P]
R9  : si [E,ES] alors E -> {
R10 : si [E,ES] alors D -> T
R11 : si [E,ES] alors V -> F
R12 : si [E,ES] alors G -> K
R13 : si [E,ES] alors SS -> S
R14 : si desinence = [E,ES] alors LL -> L

R16 : si [E,ES] alors CH -> K
```

```
%{
# include <stdio.h>

# define LGUNI 10

typedef struct noeud {
        int type;
        char *mot[LGUNI];
        struct noeud *gau;
        struct noeud *dro;
        struct noeu *svt;} NOEUD;

NOEUD *inter,*racine;
char retient[LGUNI];

%}
%start  axiome

%union { NOEUD *res ; }

%token  IDF  ALORS  AJOUTER  CONSONNE  DE  DESINENCE ENTRE  FINAL  INITIAL
%token  NON  PRECEDE  RIEN  SAUF SI  SINON  SUIVI TOMBE  VOYELLE
%token  LREG  REEC  GENEXP  ENSEXP  SANSP  CHAI  CHAIDF
%token  FIN NFIN

%type <res> lregles  regle   lpremisses  premisse  tyype  ensemble
%type <res> lchaine  chaine  action  expression  generique desinence

%left '+'
%left '*'

%%

axiome          : lregles
                        { racine=$1; }
                | /* vide */
                ;
lregles         : lregles regle
                        { $$=creer_noeud(LREG,"",$2,NULL,$1); }
                | regle
                        { $$=creer_noeud(LREG,"",$1,NULL,NULL); }
                ;
regle           : chaine ':' SI desinence '*' lpremisses ALORS action
                        { inter=creer_noeud('*',"",$4,$6,NULL);
                          $$=creer_noeud(SI,"",$1,inter,$8); }
                | chaine ':' SI desinence ALORS action
                        { $$=creer_noeud(SANSP,"",$1,$4,$6); }
                ;
lpremisses      : premisse
                | lpremisses '+' lpremisses
                        { $$=creer_noeud('+',"",$1,$3,NULL); }
                | lpremisses '*' lpremisses
                        { $$=creer_noeud('*',"",$1,$3,NULL); }
                | '(' lpremisses ')'
                        { $$=$2; }
                | NON '(' lpremisses ')'
                        { $$=creer_noeud(NON,"",$3,NULL,NULL); }
                ;
premisse        : PRECEDE DE tyype
                        { $$=creer_noeud(PRECEDE,"",$3,NULL,NULL); }
                | FINAL '=' tyype
                        { $$=creer_noeud(FIN,"",$3,NULL,NULL); }
                | FINAL '#' tyype
                        { $$=creer_noeud(NFIN,"",$3,NULL,NULL); }
                ;
tyype           : chaine
                | chaine '#' expression
                        { $$=creer_noeud(CHAIDF,"",$1,$3,NULL); }
                | generique
                | generique '#' expression
                        { $$=creer_noeud(GENEXP,"",$1,$3,NULL); }
                | ensemble
                ;
ensemble        : '[' lchaine ']'
                        { $$=$2; }
                ;
lchaine         : lchaine ',' chaine
                        { $$=creer_noeud(CHAI,"",$3,NULL,$1); }
                | chaine
                ;
chaine          : IDF
                        { $$=creer_noeud(IDF,retient,NULL,NULL,NULL); }
                ;
action          : chaine '-' '>' chaine
                        { $$=creer_noeud(REEC,"",$1,$4,NULL); }
                | chaine TOMBE
                        { $$=creer_noeud(TOMBE,"",$1,NULL,NULL); }
                | AJOUTER expression
                        { $$=creer_noeud(AJOUTER,"",$2,NULL,NULL); }
                ;
expression      : chaine
                | ensemble
                ;
desinence       : expression
```

```
                | DESINENCE '=' expression
                        { $$=$3; }
generique       : VOYELLE
                        { $$=creer_noeud(VOYELLE,"",NULL,NULL,NULL); }
                | CONSONNE
                        { $$=creer_noeud(CONSONNE,"",NULL,NULL,NULL); }
                ;

%%
# include "princ2.c"
```

```
# define LGMAX 100
# define ERREUR 300

/* TABLE DES MOTS-CLES */

char nom_source[9];     /* nom physique du fichier source */
char nom_objet[9];      /* nom physique du fichier objet  */
FILE *source,*objet;    /* fichiers source et objet       */
int indice=0;           /* indice du caractere analyse de la ligne courante */
int nolig=0;            /* numero de ligne */
char lsource[LGMAX];    /* ligne source */
char lvide[LGMAX];
char lobjet[LGMAX];     /* ligne objet courante */
int ff=1 ;              /* detection de la fin de fichier */
char unite[LGUNI];              /* unite syntaxique */
int  nberr;             /* nombre d'erreurs de syntaxe */
int  maxii2;            /* longueur maximum d'une partie gauche d'action */

struct mot_cle {char nom[LGUNI];
                int  svt;
                int  code;} tmc[27]=
      {"          ",0,0,
       "alors     ",11,ALORS,
       "entre     ",0,ENTRE,
       "consonne  ",8,CONSONNE,
       "de        ",12,DE,
       "et        ",2,'*',
       "final     ",7,FINAL,
       "finale    ",0,FINAL,
       "consonnes ",0,CONSONNE,
       "initial   ",10,INITIAL,
       "initiale  ",0,INITIAL,
       "ajouter   ",0,AJOUTER,
       "desinence ",0,DESINENCE,
       "          ",0,0,
       "non       ",0,NON,
       "ou        ",0,'+',
       "precede   ",0,PRECEDE,
       "          ",0,0,
       "rien      ",0,RIEN,
       "si        ",23,SI,
       "tombe     ",0,TOMBE,
       "voyelles  ",0,VOYELLE,
       "voyelle   ",21,VOYELLE,
       "sauf      ",24,SAUF,
       "sinon     ",25,SINON,
       "suivi     ",0,SUIVI,
       "          ",0,0};

# include "/proj/dialogue/souvay/commun/arbre.c"
# include "lex2.c"
# include "gen2.c"
# include "/proj/dialogue/souvay/commun/gencommun.c"

/*************************************************************************/
/*  PROGRAMME PRINCIPAL                                                  */
/*************************************************************************/


main ()
   {int c;
    char ligne[LGMAX];
    int i,longueur;
    static NOEUD* malloc();

    entete();
    for (i=0;i<LGMAX;++i) {
        lvide[i]=' ';
        lobjet[i]=' ';
                          }

    if (yyinit()==1) {
         racine=malloc(sizeof(NOEUD));
   printf("------------------------------------------------ \n\n");
         printf("analyse syntaxique en cours \n");
         yyparse();
         if (nberr==0) {
            printf("analyse syntaxique correcte.\n\n") ;
/*            printf("edition arbre o/n : ");
            c=getchar();
            if (c='o'|c=='O') {printf("edition de l'arbre\n\n");
                               ecr_arb(racine);}*/
   printf("------------------------------------------------ \n\n");
            printf("generation des regles en cours \n\n");
            generation(racine);
            printf("\n");
```

```
   printf("------------------------------------------------ \n\n");
            printf("fin de travail \n");CR;
                                        }
         else printf("%d erreur(s) de syntaxe\n",nberr);
}}


/*************************************************************************/
/* ENTETE DE PROGRAMME                                                   */
/*************************************************************************/

entete()

{  home();
   printf("Compilateur des regles de niveau II.2 Version 0 \n");
   printf("------------------------------------------------ \n\n"); }


/*************************************************************************/
/* EFFACEMENT DE L'ECRAN                                                 */
/*************************************************************************/

home()

{printf("\033E");}
```

```
/*************************************************************************/
/* TRANSFORMATION EN MAJUSCULE DES CARACTERES                            */
/*************************************************************************/

majus(c)
char c;
   {if (c>='A' && c<='Z') {
        return(c-'a'+'A'); }
    else {
        return(c);}
   }


/*************************************************************************/
/*  ANALYSEUR LEXICAL                                                    */
/*************************************************************************/

yylex()
  {int etat=0;          /* etat courant de l'automate */
   int action,lettre;   /* numero de l'action a effectuer et lettre lue */
   int lg=0;            /* longueur de l'unite syntaxique */
   char c;              /* caractere courant */

   static int teta[2][8] = {
                            {1,-2,0,0,-3,-4,-5,0},{1,-1,-1,-1,-1,-4,1,-1}
                           } ;
                        /* automate d'analyse */
   static int tact[2][8] = {
                            {1,2,0,3,6,4,7,3},{1,5,5,5,5,4,1,5}
                           } ;
                        /* table des actions */


   strcpy(unite,"          ");
   while (etat>=0)
      {c=lsource[indice];
       lettre=det_lettre(c);
       action=tact[etat][lettre];
       etat=teta[etat][lettre];
       switch(action)
          {case 0:++indice;break;
           case 1:if (++lg<10) {
                     unite[lg-1]=c;}
                  ++indice;
                  break;
           case 2:++indice;
                  return (c);
           case 3:if (ff==0) {
                     return(EOF);}
                  else {
                     indice=0;
                     lire(lsource,LGMAX);
                     break; }
           case 4:++indice;
                  return(YYERRCODE);
           case 5:strcpy(retient,unite);
                  return(calcul());
           case 6:return(EOF);
           case 7:unite[0]=c;lg=1;++indice;
                  strcpy(retient,unite);
                  if (c=='0') { return(IDF); }
                  else { return(YYERRCODE);}}
       } /* while */ }

/*************************************************************************/
/*  RECHERCHE DE LA LIGNE SUIVANTE                                       */
/*************************************************************************/

lire(ligne,taille)
char ligne[];
int taille;
   {int c,i;

    ++nolig;
    for (i=0;i<LGMAX-1;++i) { ligne[i]=' '; }
    for (i=0;i<taille-1 && (ff=(c=getc(source))!=EOF) && c!='\n';++i)
        ligne[i]=c;
    if (c=='\n') {
       ligne[i]=c;
       ++i;
                 }
    if (ff==0)
    ligne[i]='\0';
    return(i);}
```

```
/*************************************************************************/
/*  DETERMINATION DE LA LETTRE                                           */
/*************************************************************************/

det_lettre(c)
char c;
  {if ((c>='A' && c<='Z') || (c>='a' && c<='z'))
     { return(0);}
   else
      if (c>='0' && c<='9')
         { return(6); }
      else
         switch(c)
            {case ':':  return(1);
             case '+':  case '*':       case '=':
             case '#':  case '[':       case ']':       case ',':
             case '(':  case ')':       case '>':       case '-':return(1);
             case '\n': return(3);
             case ' ':  return(2);
             case ';':  return(7);
             case '/':     case '@':    case '|':
             case '{':  case '}':return(0);
             default:return((ff==0)?4:5);}
}

/*************************************************************************/
/*  RECHERCHE DU CODE DU MOT-CLE                                         */
/*************************************************************************/

calcul()
   {int hcode=unite[0]-'a'+1;
    int fin=0;
    int trouve=0;

    if (hcode<1 || hcode>26) { return(IDF); }
    while (fin==0) {
        trouve=(strcmp(tmc[hcode].nom,unite)==0);
        if (!trouve) {
           hcode=tmc[hcode].svt;}
        fin=((hcode==0) || trouve || (tmc[hcode].nom[0] !=unite[0]));
                  }
    if (trouve !=0) {
       return(tmc[hcode].code);}
    else {return(IDF);}
   }


/*************************************************************************/
/* AIDE A LA MISE AU POINT                                               */
/*************************************************************************/

editer(code)
int code;
   {printf("-----------------------------------\n");
    printf("%s\n",lsource);
    lvide[indice]='^';
    printf("%s\n",lvide);
    lvide[indice]=' ';
    printf("%s --> %d \n",unite,code);}


/*************************************************************************/
/*  INITIALISATION DE L'ANALYSEUR LEXICAL                                */
/*************************************************************************/

yyinit ()
{   printf("nom du fichier source : ");
    scanf("%8s",nom_source);
    printf("\n");
    if ((source=fopen(nom_source,"r"))==NULL) {
        printf("fichier inexistant\n");
        return(0); }
    else {printf("nom du fichier objet  : ");
          scanf("%8s",nom_objet);
          printf("\n");
          objet=fopen(nom_objet,"w");
          lire(lsource,LGMAX);
          return(1);}
   }
```

```
# line 2 "gII2"
# include <stdio.h>

# define LGUNI 10

typedef struct noeud {
         int type;
         char *mot[LGUNI];
         struct noeud *gau;
         struct noeud *dro;
         struct noeud *svt;} NOEUD;

NOEUD *inter,*racine;
char retient[LGUNI];


# line 19 "gII2"
typedef union  { NOEUD *res ; } YYSTYPE;
# define IDF 257
# define ALORS 258
# define AJOUTER 259
# define CONSONNE 260
# define DE 261
# define DESINENCE 262
# define ENTRE 263
# define FINAL 264
# define INITIAL 265
# define NON 266
# define PRECEDE 267
# define RIEN 268
# define SAUF 269
# define SI 270
# define SINON 271
# define SUIVI 272
# define TOMBE 273
# define VOYELLE 274
# define LREG 275
# define REEC 276
# define GENEXP 277
# define ENSEXP 278
# define SANSP 279
# define CHAI 280
# define CHAIDF 281
# define FIN 282
# define NFIN 283
#define yyclearin yychar = -1
#define yyerrok yyerrflag = 0
extern int yychar;
extern short yyerrflag;
#ifndef YYMAXDEPTH
#define YYMAXDEPTH 150
#endif
YYSTYPE yylval, yyval;
# define YYERRCODE 256

# line 103 "gII2"

# include "princ2.c"
short yyexca[] ={
-1, 1,
        0, -1,
        -2, 0,
        };
# define YYNPROD 33
# define YYLAST 232
short yyact[]={

  14,  40,   8,  37,  22,  14,   5,  15,  28,  34,
  33,  14,   5,  31,  12,   4,  10,   4,  39,  57,
  17,   7,  58,  34,  33,  47,  34,  33,  34,  19,
  13,  27,  36,  60,  29,  49,  59,   9,   3,  51,
  26,   6,  18,  21,  38,  42,  43,  27,  20,   2,
   1,   0,  50,  50,  50,   0,   0,   0,   0,   0,
   0,   0,  30,   0,   0,   0,   0,   0,  52,  52,
  52,  35,  61,  44,  55,  56,  62,  63,   0,   0,
   0,   0,  45,  46,   0,  48,   0,   0,   0,   0,
   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0,   0,   0,   0,   0,   5,   0,   0,  54,
   0,   5,   0,   0,   0,   0,  11,   5,   0,   0,
```

```
   0,   0,   0,  53,   0,   0,   0,   0,   0,   0,
   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0,   0,  16,   0,  32,   0,   0,  25,  41,
  23,  24 };
short yypact[]={

-245,-1000,-245,-1000, -37,-1000,-1000,-268, -86, -35,
-1000, -41,-1000,-1000,-245, -36,-251, -80, -31,-1000,
 -33,-1000, -36,  -8,-258, -17,-1000, -44, -80,-1000,
-1000,-245,-251, -36, -36, -16, -36, -91, -91, -91,
 -43,-1000,-1000,-1000,-1000, -14,-1000,-1000, -19,-1000,
   1,  -2,-1000,-1000,-1000,-1000,-1000,-245,-1000, -80,
 -80,-1000,-1000,-1000 };
short yypgo[]={

   0,  50,  49,  38,  48,  43,  35,  30,  42,  14,
  40,  16,  39,  37 };
short yyr1[]={

   0,   1,   1,   2,   2,   3,   3,   4,   4,   4,
   4,   4,   5,   5,   5,   6,   6,   6,   6,   6,
   7,   8,   8,   9,  10,  10,  10,  11,  11,  13,
  13,  12,  12 };
short yyr2[]={

   0,   1,   0,   2,   1,   8,   6,   1,   3,   3,
   3,   4,   3,   3,   3,   1,   3,   1,   3,   1,
   3,   3,   1,   1,   4,   2,   2,   1,   1,   1,
   3,   1,   1 };
short yychk[]={

-1000,  -1,  -2,  -3,  -9, 257,  -3,  58, 270, -13,
 -11, 262,  -9,  -7,  91,  42, 258,  61,  -8,  -9,
  -4,  -5,  40, 266, 267, 264, -10,  -9, 259, -11,
  93,  44, 258,  43,  42,  -4,  40, 261,  61,  35,
  45, 273, -11,  -9, -10,  -4,  -4,  41,  -4,  -6,
  -9, -12,  -7, 274, 260,  -6,  -6,  62,  41,  35,
  35,  -9, -11, -11 };
short yydef[]={

   2,  -2,   1,   4,   0,  23,   3,   0,   0,   0,
  29,   0,  27,  28,   0,   0,   0,   0,   0,  22,
   0,   7,   0,   0,   0,   0,   6,   0,   0,  30,
  20,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,  25,  26,  21,   5,   8,   9,  10,   0,  12,
  15,  17,  19,  31,  32,  13,  14,   0,  11,   0,
   0,  24,  16,  18 };
#ifndef lint
static char yaccpar_sccsid[] = "@(#)yaccpar     4.1     (Berkeley)      2/11/83";
#endif not lint

#
# define YYFLAG -1000
# define YYERROR goto yyerrlab
# define YYACCEPT return(0)
# define YYABORT return(1)

/*      parser for yacc output  */

#ifdef YYDEBUG
int yydebug = 0; /* 1 for debugging */
#endif
YYSTYPE yyv[YYMAXDEPTH]; /* where the values are stored */
int yychar = -1; /* current input token number */
int yynerrs = 0;  /* number of errors */
short yyerrflag = 0;  /* error recovery flag */

yyparse() {

        short yys[YYMAXDEPTH];
        short yyj, yym;
        register YYSTYPE *yypvt;
        register short yystate, *yyps, yyn;
        register YYSTYPE *yypv;
        register short *yyxi;

        yystate = 0;
        yychar = -1;
        yynerrs = 0;
        yyerrflag = 0;
        yyps= &yys[-1];
        yypv= &yyv[-1];

 yystack:    /* put a state and value onto the stack */
```

```
#ifdef YYDEBUG
        if( yydebug  ) printf( "state %d, char 0%o\n", yystate, yychar );
#endif
                if( ++yyps> &yys[YYMAXDEPTH] ) { yyerror( "yacc stack overflow" ); return(1); }
                *yyps = yystate;
                ++yypv;
                *yypv = yyval;

 yynewstate:

        yyn = yypact[yystate];

        if( yyn<= YYFLAG ) goto yydefault; /* simple state */

        if( yychar<0 ) if( (yychar=yylex())<0 ) yychar=0;
        if( (yyn += yychar)<0 || yyn >= YYLAST ) goto yydefault;

        if( yychk[ yyn=yyact[ yyn ] ] == yychar ){ /* valid shift */
                yychar = -1;
                yyval = yylval;
                yystate = yyn;
                if( yyerrflag > 0 ) --yyerrflag;
                goto yystack;
                }

 yydefault:
        /* default state action */

        if( (yyn=yydef[yystate]) == -2 ) {
                if( yychar<0 ) if( (yychar=yylex())<0 ) yychar = 0;
                /* look through exception table */

                for( yyxi=yyexca; (*yyxi!= (-1)) || (yyxi[1]!=yystate) ; yyxi += 2 ) ; /* VOID */

                while( *(yyxi+=2) >= 0 ){
                        if( *yyxi == yychar ) break;
                        }
                if( (yyn = yyxi[1]) < 0 ) return(0);   /* accept */
                }

        if( yyn == 0 ){ /* error */
                /* error ... attempt to resume parsing */

                switch( yyerrflag ){

                case 0:   /* brand new error */

                        yyerror( "syntax error" );
                yyerrlab:
                        ++yynerrs;

                case 1:
                case 2: /* incompletely recovered error ... try again */

                        yyerrflag = 3;

                        /* find a state where "error" is a legal shift action */

                        while ( yyps >= yys ) {
                           yyn = yypact[*yyps] + YYERRCODE;
                           if( yyn>= 0 && yyn < YYLAST && yychk[yyact[yyn]] == YYERRCODE ){
                              yystate = yyact[yyn];  /* simulate a shift of "error" */
                              goto yystack;
                              }
                           yyn = yypact[*yyps];

                           /* the current yyps has no shift onn "error", pop stack */

#ifdef YYDEBUG
                           if( yydebug ) printf( "error recovery pops state %d, uncovers %d\n", *yyps, yyps[-1] );
#endif
                           --yyps;
                           --yypv;
                           }

                        /* there is no state on the stack with an error shift ... abort */

        yyabort:
                        return(1);


                case 3:  /* no shift yet; clobber input char */

#ifdef YYDEBUG
                        if( yydebug ) printf( "error recovery discards char %d\n", yychar );
```

```
#endif

                        if( yychar == 0 ) goto yyabort; /* don't discard EOF, quit */
                        yychar = -1;
                        goto yynewstate;   /* try again in the same state */

                        }

                }

        /* reduction by production yyn */

#ifdef YYDEBUG
                if( yydebug ) printf("reduce %d\n",yyn);
#endif
                yyps -= yyr2[yyn];
                yypvt = yypv;
                yypv -= yyr2[yyn];
                yyval = yypv[1];
                yym=yyn;
                        /* consult goto table to find next state */
                yyn = yyr1[yyn];
                yyj = yypgo[yyn] + *yyps + 1;
                if( yyj>=YYLAST || yychk[ yystate = yyact[yyj] ] != -yyn ) yystate = yyact[yypgo[yyn]];
                switch(yym){

case 1:
# line 35 "gII2"
{ racine=yypvt[-0].res; } break;
case 3:
# line 39 "gII2"
{ yyval.res=creer_noeud(LREG,"",yypvt[-0].res,NULL,yypvt[-1].res); } break;
case 4:
# line 41 "gII2"
{ yyval.res=creer_noeud(LREG,"",yypvt[-0].res,NULL,NULL); } break;
case 5:
# line 44 "gII2"
{ inter=creer_noeud('*',"",yypvt[-4].res,yypvt[-2].res,NULL);
                        yyval.res=creer_noeud(SI,"",yypvt[-7].res,inter,yypvt[-0].res); } break;
case 6:
# line 47 "gII2"
{ yyval.res=creer_noeud(SANSP,"",yypvt[-5].res,yypvt[-2].res,yypvt[-0].res); } break;
case 8:
# line 51 "gII2"
{ yyval.res=creer_noeud('+',"",yypvt[-2].res,yypvt[-0].res,NULL); } break;
case 9:
# line 53 "gII2"
{ yyval.res=creer_noeud('*',"",yypvt[-2].res,yypvt[-0].res,NULL); } break;
case 10:
# line 55 "gII2"
{ yyval.res=yypvt[-1].res; } break;
case 11:
# line 57 "gII2"
{ yyval.res=creer_noeud(NON,"",yypvt[-1].res,NULL,NULL); } break;
case 12:
# line 60 "gII2"
{ yyval.res=creer_noeud(PRECEDE,"",yypvt[-0].res,NULL,NULL); } break;
case 13:
# line 62 "gII2"
{ yyval.res=creer_noeud(FIN,"",yypvt[-0].res,NULL,NULL); } break;
case 14:
# line 64 "gII2"
{ yyval.res=creer_noeud(NFIN,"",yypvt[-0].res,NULL,NULL); } break;
case 16:
# line 68 "gII2"
{ yyval.res=creer_noeud(CHAIDF,"",yypvt[-2].res,yypvt[-0].res,NULL); } break;
case 18:
# line 71 "gII2"
{ yyval.res=creer_noeud(GENEXP,"",yypvt[-2].res,yypvt[-0].res,NULL); } break;
case 20:
# line 75 "gII2"
{ yyval.res=yypvt[-1].res; } break;
case 21:
# line 78 "gII2"
{ yyval.res=creer_noeud(CHAI,"",yypvt[-0].res,NULL,yypvt[-2].res); } break;
case 23:
# line 82 "gII2"
{ yyval.res=creer_noeud(IDF,retient,NULL,NULL,NULL); } break;
case 24:
# line 85 "gII2"
{ yyval.res=creer_noeud(REEC,"",yypvt[-3].res,yypvt[-0].res,NULL); } break;
case 25:
# line 87 "gII2"
{ yyval.res=creer_noeud(TOMBE,"",yypvt[-1].res,NULL,NULL); } break;
case 26:
# line 89 "gII2"
```

```
{ yyval.res=creer_noeud(AJOUTER,"",yypvt[-0].res,NULL,NULL); } break;
case 30:
# line 96 "gII2"
{ yyval.res=yypvt[-0].res; } break;
case 31:
# line 98 "gII2"
{ yyval.res=creer_noeud(VOYELLE,"",NULL,NULL,NULL); } break;
case 32:
# line 100 "gII2"
{ yyval.res=creer_noeud(CONSONNE,"",NULL,NULL,NULL); } break;
                }
                goto yystack;  /* stack new state and value */

        }
```

```
/*****************************************************************************/
/* GENERATION DES REGLES LISP                                                */
/*****************************************************************************/

generation(n)
NOEUD n;
{  copie(1,23,"; LISTE DES REGLES II.2N");
   sortie(1,23);
   sortie(1,1);
   copie(1,24,"(setq #:mf:reglesII2N '(  ");
   sortie(1,24);
   genlregles(n);
   sortie(1,1);
   copie(1,2,"))");
   sortie(1,2);
   sortie(1,1);
   copie(1,21,"(setq #:mf:maxII2   )     ");
   if (maxii2<10)
      {lobjet[20]='0'+maxii2;}
   else
      {lobjet[19]='0'+(maxii2/10);
       lobjet[20]='0'+(maxii2%10);}
   sortie(1,21);
}


/*****************************************************************************/
/* GENERATION AU NIVEAU DE lregles                                           */
/*****************************************************************************/

genlregles(n)
NOEUD *n;
   {if (n != NULL) {
       switch(n->type)
          {case  LREG:genlregles(n->svt);
                      genregle(n->gau);
                      break;
              default:genregle(n); }}}




/*****************************************************************************/
/* ERREUR DE PROGRAMMATION                                                   */
/*****************************************************************************/

errpgt(i,type)
int i,type;

   {switch(i)
       {case 1:printf("erreur de programmation genregles   : ");
           '    break;
        case 2:printf("erreur de programmation gensansp    : ");
               break;
        case 3:printf("erreur de programmation genpremisse : ");

       }
     printf("%d \n",type);
   }


/*****************************************************************************/
/* GENERATION POUR lpremisses                                                */
/*****************************************************************************/

genlpremisses(n,alignement,nomreg)

NOEUD *n;
int alignement;
char *nomreg;

   {switch(n->type)
       {case NON:
        case '+':
        case '*':genetounon(n,alignement,nomreg);
                 break;
         default:genpremisse(n,alignement,nomreg);}}


/*****************************************************************************/
/*  GENERATION POUR lpremisses ET/OU lpremisses                              */
/*****************************************************************************/

genetounon(n,alignement,nomreg)
```

```
NOEUD *n;
int alignement;
char *nomreg;

   {switch(n->type)
       {case '*':copie(alignement,5,"(and    ");
                 sortie(alignement,5);
                 break;
        case '+':copie(alignement,5,"(or     ");
                 sortie(alignement,5);
                 break;
        case NON:copie(alignement,5,"(not    ");
                 sortie(alignement,5);
                 break;
        }
    genlpremisses(n->gau ,alignement+5,nomreg);
    if (n->type != NON )  genlpremisses(n->dro ,alignement+5,nomreg);
    copie(alignement,1,")");
    sortie(alignement,1);}



/*************************************************************************/
/* GENERATION POUR premisse                                              */
/*************************************************************************/

genpremisse(n,alignement,nomreg)

NOEUD *n;
int alignement;
char *nomreg;

{NOEUD *s;

   switch(n->type)
       {
       case PRECEDE:gensuipre(n,alignement,nomreg);
                    break;
       case  FIN:s=creer_noeud(PRECEDE,"",n->gau,NULL,NULL);
                 gensuipre(s,alignement,nomreg,1);
                 break;
       case NFIN:copie(alignement,5,"(not     ");
                 sortie(alignement,5);
                 s=creer_noeud(PRECEDE,"",n->gau,NULL,NULL);
                 gensuipre(s,alignement,nomreg,1);
                 lobjet[alignement]=')';sortie(alignement,1);
                 break;
            default:errpgt(3,n->type);}}

/*************************************************************************/
/* GENERATION POUR precede de et suivi de                                */
/*                                                                       */
/* n      : noeud courant            alignement : marge gauche           */
/* nomreg : numero de la regle       simule : noeuds FIN NFIN INI NINI   */
/*************************************************************************/

gensuipre(n,alignement,nomreg,simule)

NOEUD *n;
int alignement;
char *nomreg;
int simule;

   {NOEUD *p;
    int i,j,lg,idx;
    char  *m;

    idx=(simule==1)?34:20;
    p=n->gau;m=p->mot;
    switch(p->type)
       {case CHAI:spconst(alignement,n->type,'7',simule);
                  genchai(n->gau,alignement+idx);
                  lobjet[alignement]=')';
                  sortie(alignement,1);
                  break;
        case VOYELLE:
        case CONSONNE:spconst(alignement,n->type,'3',simule);
                      genvoycons(p->type,alignement,simule);
                      sortie(alignement,idx+9);
                      break;
        case GENEXP:controle1(n->type,(p->gau)->type,p->dro,nomreg);
                    spconst(alignement,n->type,'4',simule);
                    genvoycons((p->gau)->type,alignement,simule);
                    lobjet[alignement+idx+8]=' ';
                    genchai(p->dro,alignement+idx+9);
```

```
                lobjet[alignement]=')';
                sortie(alignement,1);
                break;
        case IDF:if ((lg=lgchaine(p->mot))==1)
                {spconst(alignement,n->type,'1',simule);
                 copie(alignement+idx,4,"' )  ");
                 lobjet[alignement+idx+1]= *m;
                 sortie(alignement,idx+4);}
              else
                 {spconst(alignement,n->type,'2',simule);
                  copie(alignement+idx,2,"'(");
                  j=alignement+idx+2+(2*lg-1);
                  for (i=alignement+idx+2;i<j+(2*lg-1);i+=2)
                      { lobjet[i]= *m++; }
                  copie(j,2,"))  ");
                  sortie(alignement,idx+2+2*lg+1);}
              break;
        case CHAIDF:controle2();
                spconst(alignement,n->type,'5',simule);
                copie(alignement+idx,2,"'(                          ");
                lg=lgchaine(m=(p->gau)->mot);
                j=alignement+idx+2+(2*lg-1);
                for (i=alignement+idx+2;i<j;i+=2)
                    { lobjet[i]= *m++;}
                lobjet[j]=')';
                genchai(p->dro,j+2);
                lobjet[alignement]=')';
                sortie(alignement,1);
                break;

      }
   }


/*************************************************************************/
/* GENERATION DE LA LIGNE POUR SUIVI DE ET PRECEDE DE                    */
/*************************************************************************/

spconst(alignement,type,nombre,simule)

int alignement,type;
char nombre;
int simule;

{  if (simule==1)
      copie(alignement,60,
         "(precede_de    (append debut fin)                              ");
   else
      copie(alignement,47,"(precede_de   debut                           ");
   lobjet[alignement+12]=nombre;
}


/*************************************************************************/
/* GENERATION DU MOT VOYELLE OU CONSONNE                                 */
/*************************************************************************/

genvoycons(voycons,alignement,simule)

int voycons,alignement;

{int idx;

    idx=(simule==1)?34:20;
    if (voycons==VOYELLE)
       copie(alignement+idx,9," voyelle)");
    else
       copie(alignement+idx,9,"consonne)");
}


/*************************************************************************/
/* generation pour CHAI                                                  */
/*************************************************************************/

genchai(n,alignement)

NOEUD *n;
int alignement;

   {int i,lg;
    char *m;

    if (n!=NULL)
       if (n->type == CHAI)
          {genchai(n->svt,alignement);
```

```
          genchai(n->gau,alignement);}
       else
          {lg=lgchaine(m=(n->mot));
           copie(alignement,4,"'(                    ");
           for (i=alignement+2;i<alignement+2+(2*lg-1);i+=2)
               { lobjet[i]= *m++; }
           lobjet[alignement+2+(2*lg-1)]=')';
           sortie(5,alignement+2+(2*lg-1)-4);}
   }


/*****************************************************************************/
/* CONTROLE D'ERREUR                                                         */
/*****************************************************************************/

controle1(suipre,voycons,n,nomreg)
int suipre,voycons;
NOEUD *n;
char *nomreg;
   {}

controle2(){}


/*****************************************************************************/
/*  CONTROLE DU NON BOUCLAGE DES REGLES    Version 0                         */
/*****************************************************************************/

controle3(mg,md,nomreg)
char *mg,*md;
char *nomreg;

{  if (strcmp(mg,md)==0)
      {printf("*** CII1.V0 : la regle %s n'effectue rien \n",nomreg);}
   else
      if (inclu(mg,md)==1)
         {printf("*** CII1.V0 : verifier que cette regle");
          printf(" ne provoque pas de bouclage \n");}
}
inclu(mg,md)
char *mg,md;
{return(0);}


/*****************************************************************************/
/* GENERATION POUR desinence                                                 */
/*****************************************************************************/

gendesinence(n)

NOEUD *n;

   {if (n->type==CHAI)
       {copie(10,40,"(or                                          ");
        gendes(n->svt,14);
        gendes(n->gau,14);
        lobjet[10]=')';
        sortie(10,1);}
    else gendes(n,10);}

/*****************************************************************************/
/* FONCTION UTILISEE PAR gendesinence                                        */
/*****************************************************************************/

gendes(n,alignement)

NOEUD *n;
int alignement;

   {char *m;
    int lg,i;

    if (n!=NULL)
       if (n->type==CHAI)
          {gendes(n->svt,alignement);
           gendes(n->gau,alignement);}
       else
          {m=n->mot;
           if ( m[0] == '0')
              {copie(alignement,17,"(null desinence)  ");
               sortie(alignement,17);}
           else
              {copie(alignement,41,
                   "(equal desinence '(                    ");
               lg=lgchaine(m);
               for (i=alignement+19;i<alignement+19+(2*lg-1);i+=2)
```

```
                { lobjet[i]= *m++; }
              copie(alignement+19+(2*lg-1),2,"))   ");
              sortie(alignement,41);}}}


/*****************************************************************************/
/* GENERATION POUR si alors sinon                                            */
/*****************************************************************************/

gensialors(n)

NOEUD *n;

   {int lg,i;
    NOEUD *a,*d;
    char *m;

    sortie(1,1);
    copie(1,2,"( ");copie(3,10,(n->gau)->mot);
    sortie(1,12);sortie(1,1);
    d=(n->type==SANSP)?(n->dro):(n->dro)->gau;
    a=(n->svt);
    maxii2=( ( (a->gau)->type != CHAI ) &&
             ( (lg=lgchaine((a->gau)->mot)) > maxii2 ) )
          ?lg:maxii2;
    if (a->type  != AJOUTER)
       {copie(1,25,"(if (and (equal fin '(                    ");
        lg=lgchaine(m=(a->gau)->mot);
        for (i=23;i<23+(2*lg-1);i+=2) lobjet[i]= *m++;
        copie(23+(2*lg-1),2,"))  ");
        sortie(1,23+(2*lg-1)+2);}
    else
       {copie(1,10,"(if (and   ");
        sortie(1,10);}
    gendesinence(d,10);
    if (n->type == SI) genlpremisses((n->dro)->dro,10);
    lobjet[5]=')';sortie(5,1);
    genaction(a);
    copie(1,2,")) ");sortie(1,2);}


/*****************************************************************************/
/* GENERATION POUR regle                                                     */
/*****************************************************************************/


genregle(n)

NOEUD *n;

   {switch(n->type)
      {case SI:
       case SANSP:gensialors(n);
                  break;
       default:errpgt(1);}
   }


/*****************************************************************************/
/* GENERATION POUR genaction                                                 */
/*****************************************************************************/

genaction(n)

NOEUD *n;
{  int lg,i;
   char *m;

switch (n->type)
   {case TOMBE:copie(5,28,"(list `(,debut ,desinence))           ");
               sortie(5,28);
               break;
    case REEC :copie(5,60,
       "(list `(,(append debut '(                                 ");
               lg=lgchaine(m=(n->dro)->mot);
               for (i=30;i<30+(2*lg-1);i+=2)
                   { lobjet[i]= *m++; }
               copie(30+(2*lg-1),16,")) ,desinence))       ");
               sortie(5,60);
               break;
    case AJOUTER:copie(5,65,"(list  ");
                 sortie(5,5);
                 genajouter(n->gau);
                 copie(5,1,")  ");
                 sortie(5,1);
                 break;
```

```
}
}

/* ------------------------------------------------------------------------*/
/*  GENERATION POUR ajouter                                                */
/* ------------------------------------------------------------------------*/

genajouter(n)

NOEUD *n;

{int i,lg;
 char *m;

    {if (n->type == IDF)
        {copie(11,60,
     "`(,(append debut '(                                          ");
         lg=lgchaine(m=(n->mot));
         for (i=30;i<30+(2*lg-1);i+=2)
             { lobjet[i] = *m++; }
         copie(30+(2*lg-1),15,")) ,desinence)    ");
         sortie(11,60);}
     else
        {genajouter(n->svt);
         genajouter(n->gau);}
    }
}
```

ANNEXE 4

Le lexique

1. La grammaire algébrique

2. Le contenu du lexique

```
%{
# include <stdio.h>

# define LGUNI 26

typedef struct noeud {
          int type;
          char *mot[LGUNI];
          struct noeud *gau;
          struct noeud *dro;
          struct noeud *svt;} NOEUD;

NOEUD *racine;
char retient[LGUNI];

%}
%start  axiome

%union { NOEUD *res ; }

%token  IDF  LETTRE NOMBRE
%token  LBLOC BLOC LBASE BASE CARACT PTR

%type <res> lblocs bloc entete lbases base caractere type categorie chaine ptr
%type <res> paradygme

%%

axiome          : lblocs
                        { racine=$1; }
                | /* vide */
                ;
lblocs          : lblocs bloc
                        { $$=creer_noeud(LBLOC,"",$2,NULL,$1); }
                | bloc
                ;
bloc            : entete lbases
                        { $$=creer_noeud(BLOC,"",$1,$2,NULL); }
                ;
entete          : LETTRE '=' chaine
                        { $$=$3; }
                ;
lbases          : lbases base
                        { $$=creer_noeud(LBASE,"",$2,NULL,$1); }
                | base
                ;
base            : chaine ',' caractere
                        { $$=creer_noeud(BASE,"",$1,$3,NULL); }
                ;
caractere       : type ',' categorie ',' ptr
                        { $$=creer_noeud(CARACT,"",$1,$3,$5); }
                ;
type            : IDF
                        { $$=creer_noeud(IDF,retient,NULL,NULL,NULL); }
                ;
categorie       : IDF
                        { $$=creer_noeud(IDF,retient,NULL,NULL,NULL); }
                ;
chaine          : IDF
                        { $$=creer_noeud(IDF,retient,NULL,NULL,NULL); }
                ;
ptr             : chaine ',' paradygme
                        { $$=creer_noeud(PTR,"",$1,$3,NULL); }
paradygme       : IDF
                        { $$=creer_noeud(IDF,retient,NULL,NULL,NULL); }
                | NOMBRE
                        { $$=creer_noeud(IDF,retient,NULL,NULL,NULL); }
                ;
%%
# include "princ.c"
```

```
lettre = A

A,_,H,a,_
ACLIN,_,B,aclin,4
ADRESSE,_,A,adresse,2
ADONKES,I,J,adonc,_
AD{S,_,I,J,ad}s
AFIN,I,J,Ta_fin,_
AGNEL,_,A,agnel,1
AIM,_,F,aimer,0
AINSOIS,I,J,ain/ois,_

ANSOIS,I,J,ain/ois,_
APEL,_,A,apel,1
APARTENANSE,_,A,apartenance,2
AUS,I,G,Tau,_
AUTRE,_,P,autre,5
AXION,_,A,accion,2

lettre = B

BREMENT,I,J,Tbri{ment,_
BREVEMENT,I,J,brievement,_

lettre = C

CEL,_,q,cel,4
CEST,_,Q,cest,4
CHEVEL,_,A,chevel,1
CREUS,_,B,crues,7
CRUEL,_,B,cruel,6

lettre = D

DAME,_,A,dame,2
DANSE,_,A,dance,2
DE,_,H,de,x
DERNIER,_,B,dernier,4
DERENIER,_,B,dernier,4
DESCROISSEMENT,_,A,descroissement,1
DESSI,I,J,deci,_

lettre = E

ECRIT,_,A,escrit,1
ENKOIRES,_,J,encore,_
ENUI,_,A,enui,1

lettre = F

FAUS,_,B,faus,7
FU,I,F,estre,C3

lettre = G

GARSON,_,A,gar/on,1

GENT,_,A,gent1,3
GENT,_,B,gent2,4
GERE,I,J,gaire,_
GERES,I,J,gaire,_
GRANT,_,B,grant,6
GROS,_,B,gros,7
```

```
HAUT,_,B,haut,4

lettre = I

ICEL,_,Q,icel,4
ICIST,I,Q,icest,M
ILEK,I,J,iluec,_
ILUEK,I,J,iluec,_

lettre = J

JEUNE,_,B,juene,5
JOUVENSEL,_,A,josvencel,1

lettre = K

KANT,I,I,cant1,_
KANT,_,P,cant2,6
KONOISSANSE,_,A,conoissance,2
KONTRAT,_,A,contract,1
KORS,I,A,cors1,M

lettre = L

LI{,_,B,li{1,4

lettre = M

MASSON,_,A,ma/on,1
MATIERE,_,A,matiere,2
MANIFESTATION,_,A,manifestacion,2
MOIS,I,A,mois,M
MOITI{,_,A,moiti{,2
MOUT,I,J,mout,_
MUR,_,A,mur,1

lettre = N

NASSION,_,A,nacion,2
NU,_,B,nu,4
NUL,_,P,nul,4

lettre = O

OCISION,_,A,ocision,2

lettre = P

PAROLE,_,A,parole,2
PORSION,_,A,porcion,2
PRINSE,_,A,prince,1
PROUESSE,_,A,proece,2


lettre = S

SEURT{,_,A,seurt{,2
SIEN,_,B,sien,4
SOUBSCRIPSION,_,A,soscription,2

lettre = V

VIF,_,B,vif,4
VIL,_,B,vil,4
```

VILE,_,A,vile,2
VIS,I,A,visl,M

ANNEXE 5

Les algorithmes de l'analyseur.

1. Les fonctions liées aux prémisses	**sp.ll**
2. Les moteurs II.1 et III	**acces.ll** **decp.ll**
3. Le moteur II.2	**deux.ll**
4. L'accès au lexique	**acclexi.ll**
5. L'analyseur	**mf.ll** **niveau.ll**

User souvay

File nvse/sp.ll

Path /proj/dialogue/souvay

Lundi 23 Juin 1986 11:42:0

imprimé au CRIN Campus Scientifique bp 239 54506 Vandoeuvre les Nancy

```
; ----------
;  SUIVI_DE
; ----------

(de suivi_de (nb beta mbgau . mbdro)
  (selectq nb
    (1  (equal  (initiale1 beta) mbgau))
    (2  (equal  (initialen (length mbgau) beta) mbgau))
    (3  (member (initiale1 beta)   mbgau))
    (4  (and    (member   (initiale1 beta) mbgau)
                (suipre45 beta mbdro 'initialen)))
    (5  (and    (nequal (initialen (length mbgau) beta) mbgau)
                (suipre45 beta mbdro 'initialen)))
    (7  (or     (equal (initialen (length mbgau) beta) mbgau)
                (suipre7 beta mbdro 'initialen)))))

; ------------
;  PRECEDE_DE
; ------------

(de precede_de (nb alpha mbgau . mbdro)
  (selectq nb
    (1  (equal  (finale1 alpha)  mbgau))
    (2  (equal  (finalen (length mbgau) alpha) mbgau))
    (3  (member (finale1 alpha)  mbgau))
    (4  (and    (member   (finale1 alpha) mbgau)
                (suipre45 alpha mbdro 'finalen)))
    (5  (and    (nequal (finalen (length mbgau) alpha) mbgau)
                (suipre45 alpha mbdro 'finalen)))
    (7  (or     (equal (finalen (length mbgau) alpha) mbgau)
                (suipre7 alpha mbdro 'finalen)))))

; --------
;  FINALE
; --------

(de finale (beta)
    (if (null beta) t))


; ----------
;  INITIALE
; ----------

(de initiale (alpha)
    (if (null alpha) t))

; ------------------------------------------------
;  FONCTIONS AUXILIAIRES POUR SUIVI_DE ET PRECEDE_DE
; ------------------------------------------------

(de suipre45  (albet mbdro suipre)
    (if (null mbdro)
        t
        (and (nequal (eval (cons suipre '((length (car mbdro)) albet)))
                     (car mbdro))
             (suipre45 albet (cdr mbdro) suipre))))

(de suipre7 (albet mbdro suipre)
    (if (null mbdro)
        ()
        (or (equal (eval (cons suipre '((length (car mbdro)) albet)))
                   (car mbdro))
            (suipre7 albet (cdr mbdro) suipre))))
```

User **souvay**

File **nvse/acces.ll**

Path **/proj/dialogue/souvay**

Lundi 23 Juin986 11:42:5

```
; ------------------------------------------------
;  EXTRACTION DE nb LETTRE(S) INITIALE(S) D'UN mot
; ------------------------------------------------

(de initialen (nb mot)
    (cond ((<= nb 0) ())
          ((null mot) '(*))
          (t   (cons (car mot)
                     (initialen (1- nb) (cdr mot))))))

(de initiale1 (mot)
    (car mot))

; ----------------------------------------------
;  EXTRACTION DE nb LETTRE(S) FINALE(S) D'UN mot
; ----------------------------------------------

(de finalen (nb mot)
    (reverse (initialen nb (reverse mot))))

(de finale1 (mot)
    (initiale1 (reverse mot)))

; --------------------
;  NEGATION DE MEMBER
; --------------------

(de nmember (element liste)
    (not (member element liste)))

; ---------------------
;  SUPPRESSION EN TETE
; ---------------------

(de suptete (liste)
    (cdr liste))

; ------------------------------------------------------------------------
;              ADJONCTION EN QUEUE DE ELEMENT DANS LISTE
; ------------------------------------------------------------------------

(de adjqueue (element liste)
    (if (nmember element liste)
        (reverse (cons element (reverse liste)))
        liste))

; ------------------------------------------------------------------------
;              ADJONCTION EN TETE DE ELEMENT DANS LISTE
; ------------------------------------------------------------------------

(de adjtete(element liste)
    (if (nmember element liste)
        (cons element liste)
         liste))

; ------------------------------------------------------------------------
;       ADJONCTION  DANS MOTS_A_TRAITER DES ELEMENTS DU LISTE_MOTS
;       NE SE TROUVANT NI DANS MOTS_A_TRAITER NI DANS MOTS_TRAITES
; ------------------------------------------------------------------------

(de ajoute (liste_mots mots_traites mots_a_traiter)
    (append mots_a_traiter
            (mapcan (lambda (mot)
                      (if (and (nmember mot mots_traites)
                               (nmember mot mots_a_traiter))
                          (list mot)))
                    liste_mots)))
```

User **souvay**

File **nvse/decp.ll**

Path **/proj/dialogue/souvay**

Lundi 23 Juin 1986 11:42:31

imprimé au CRIN Campus Scientifique bp 239 54506 Vandoeuvre les Nancy

```
; ----------------------------
;  CREATION DES DECOMPOSITIONS
; ----------------------------

(de decp (nb l1 l2 l3)
    (if (null l3)
        (list (list l1 l2 l3))
        (if l2
            (cons (list l1 l2 l3)
                  (decp nb
                        (append l1 (list (car l2)))
                        (append (cdr l2) (list (car l3)))
                        (cdr l3)))
            (let ((lg (length l3)))
                 (if (<= nb lg)
                     (decp nb
                           ()
                           (initialen nb l3)
                           (finalen (- lg nb) l3)))))))


; ------------------------------------------------------------------------
;  DECOMPOSITION POUR TOUTE LONGUEUR INFERIEURE OU EGALE A MAX
;------------------------------------------------------------------------


(de decposeint (nb liste)
    (if (= 0 nb)
        ()
        (cons (decposen nb liste)
              (decposeint (1- nb) liste ))))


(de decpose (liste max)
    (reverse (decposeint max liste)))

;------------------------------------------------------------------------
;                       DECOMPOSITION DE LONGUEUR N
;------------------------------------------------------------------------

(de decposen (nb liste)
    (decp nb () () liste))

;------------------------------------------------------------------------
;                   APPLICATION DE LA REGLE COURANTE
;------------------------------------------------------------------------


(de appliquereg (regle)
    `(lambda ((alpha milieu beta)) ,regle))


;------------------------------------------------------------------------
;       RECHERCHE DE LA LONGUEUR DU MILIEU D'UNE DECOMPOSITION
;------------------------------------------------------------------------

(de longueur (regle)
    (let ((signif (cadr regle)))
         (cond ( (equal (car signif) 'and)
                 (length (eval (car (last (cadr signif))))) )
               (t (length (eval (car (last signif)))) ) )))

;------------------------------------------------------------------------
;            MOTEUR D'INFERENCE POUR LES REGLES DE NIVEAU II1
;
;   mots_a_traiter = liste des mots a traiter
;   mots_traites   = liste des mots traites
;   reglesII1      = liste des regles de niveau II.1
;   historique     = historique des regles appliquees
;   mot_courant    = mot courant en cours de traitement
;   regles         = liste des regles a appliquer a un instant donne
;   production     = resultat de l'application de la regle courante
;
;------------------------------------------------------------------------

(de moteur_rII1 (mots_a_traiter reglesII1)

; initialisation de mots_traites et historique

    (let ( (mots_traites () ) (historique () )
           (regles ())  (mot_courant ())  (production ()) (decomposition ())
 )

; parcours de la liste des mots a traiter

         (while mots_a_traiter
             (setq mot_courant   (car mots_a_traiter))

             (setq decomposition (decpose mot_courant #:mf:maxII1))
             (if (> (length mot_courant) 26)
                 (print "ATTENTION ! le systeme boucle sur " mot_courant)
                 (setq regles reglesII1)
                 (while regles
```

```
                    (setq production
                          (mapcan (appliquereg (cadar regles))
                                  (nth (1- (longueur (cadar regles)))
                                       decomposition)))
; mise a jour de l'historique et de mots_a_traiter

                    (unless (null production)
(setq historique (cons `(,mot_courant ,(caar regles) ,production) historique))
(setq mots_a_traiter (ajoute production mots_traites mots_a_traiter)))
                    (nextl regles)))
            (setq mots_traites (adjtete mot_courant mots_traites))
            (nextl mots_a_traiter))
        `(,(reverse mots_traites) ,(reverse historique))))


; ------------------------------------------------------------------------
;         MOTEUR D'INFERENCE POUR LES REGLES DE NIVEAU III
; ------------------------------------------------------------------------


(de moteur_rIII (mots_a_traiter reglesIII max)

; initialisation de mots_traites et historique

   (let ( (mots_traites () ) (historique () )
          (regles ())  (mot_courant ())  (production ()) (decomposition ())
 )

; parcours de la liste des mots a traiter

        (while mots_a_traiter
            (setq mot_courant   (car mots_a_traiter))

            (setq decomposition (decpose mot_courant max))
            (if (> (length mot_courant) 26)
                (print "ATTENTION ! le systeme boucle sur " mot_courant)
                (setq regles reglesIII)
                (while regles

; application de la regle courante

                    (setq production
                          (mapcan (appliquereg (cadar regles))
                                  (nth (1- (longueur (cadar regles)))
                                       decomposition)))
; mise a jour de l'historique et de mots_a_traiter

                    (unless (null production)
(setq historique (cons `(,mot_courant ,(caar regles) ,production) historique))
(setq mots_a_traiter (ajoute production mots_traites mots_a_traiter)))
                    (nextl regles)))
            (setq mots_traites (adjtete mot_courant mots_traites))
            (nextl mots_a_traiter))
        `(,(reverse mots_traites) ,(reverse historique))))
```

User **souvay**

File **nvse/deux.ll**

Path **/proj/dialogue/souvay**

Lundi 23 Juin 1986 11:42:13

imprimé au CRIN Campus Scientifique bp 239 54506 Vandoeuvre les Nancy

```
; ------------------------------------------------------------------------
;  DECOUPAGE EN BASE/DESINENCE
; ------------------------------------------------------------------------

(de decoupe (mots desinences)

(mapcan
    (lambda (mot)
        (cons `(,mot ())
              (mapcan
                  (lambda (desinence)
                      (let ( (lgdes (length desinence)))
                           (if (equal (lastn lgdes mot) desinence)
                               (list `(,(firstn (- (length mot) lgdes) mot)
                                 ,desinence)))))
                   desinences)))
     mots))


; ------------------------------------------------------------------------
;  MOTEUR D'INFERENCE POUR LES REGLES DE NIVEAU II.2
; ------------------------------------------------------------------------

(de moteur_rII2 (mots_a_traiter reglesII2)
    (let ( (mots_traites ()) (historique ()) (mot_courant) (regles) )
         (while mots_a_traiter
            (setq mot_courant (car mots_a_traiter))
            (setq mots_traites (adjtete mot_courant mots_traites))
            (setq finalisation (finalise mot_courant #:mf:maxII2))
            (setq regles reglesII2)
            (while regles
                 (setq production
                      (apply  (appliqueregII2 (cadar regles))
                              (nth (longueurII2 (cadar regles))
                                   finalisation)))
                 (unless (null production)
                    (setq historique
                         (cons `(,mot_courant
                                 ,(caar regles)
                                 ,production) historique))
                    (mapcar (lambda (produit)
                              (setq mots_traites
                                    (adjtete produit mots_traites)))
                            production))
                 (nextl regles))
            (nextl mots_a_traiter))
         `(,(reverse mots_traites)  ,(reverse historique))))

; ------------------------------------------------------------------------
;                       FINALISATION D'UN MOT
; ------------------------------------------------------------------------

(de finalise ( ( base desinence ) max )
    (let ( (liste nil) )
         (mapcar (lambda (nombre)
                    (let ( (lgbase (length base)) )
                         `(,(firstn (- lgbase nombre) base)
                           ,(lastn nombre base)
                           ,desinence)))

                 (progn (while (>= max 0)
                               (setq liste (cons max liste))
                               (setq max (1- max)))
                        liste))))

; ------------------------------------------------------------------------
;  RECONSTITUTION DE LA REGLE COURANTE
; ------------------------------------------------------------------------

(de appliqueregII2 (regle)
    `(lambda ( debut fin desinence ) ,regle))

; ------------------------------------------------------------------------
;  RECHERCHE DE LA LONGUEUR DU MEMBRE GAUCHE D'UNE ACTION
; ------------------------------------------------------------------------

(de longueurII2 (regle)
    (let ( (signif (cadadr regle)) )
         (if (equal (nth 1 signif) 'fin)
             (length (eval (nth 2 signif)))
             0)))
```

User **souvay**

File **nvse/acclexi.ll**

Path **/proj/dialogue/souvay**

Lundi 23 Juin 1986 11:42:12

imprimé au CRIN Campus Scientifique bp 239 54506 Vandoeuvre les Nancy

```
; ------------------------------------------------------------------------
;  ACCES AU LEXIQUE
; ------------------------------------------------------------------------

(de acces_lexique (base)
    (let ( (code (cascii (car base))) )
    (let ( (hcode (selectq code
                           (123 26)
                           (125 27)
                           (40  28)
                           (47  29)
                           (124 30)
                           (t   (if (> code 96)
                                    (- code 97)
                                    (- code 65))))) )
    (let ( (rangee (nth hcode #:mf:lexique)) )
         `(, (chercher (implodech base) rangee))))))

; ------------------------------------------------------------------------
;  RECHERCHE DANS UNE RANGEE DU LEXIQUE
; ------------------------------------------------------------------------

(de chercher (base rangee)
    (if (null rangee)
        ()
        (if (equal base (caar rangee))
            (car rangee)
            (chercher base (cdr rangee)))))

; ------------------------------------------------------------------------
;  GENERATION DU LEXIQUE
; ------------------------------------------------------------------------

(de genlexique (n)
    (let ((symbole (gensymbole n)))

         (if (equal n 31)
             (if (boundp symbole)
                 (list symbole))
             (cons (if (boundp symbole) (eval symbole))
                   (genlexique (1+ n))))))

; ------------------------------------------------------------------------
;  GENERATION D'UN SYMBOLE
; ------------------------------------------------------------------------

(de gensymbole (n)
    (let ((lettre
                '(A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
                  { } @ |/| ?)))
         (symbol 'mf (concat "lettre" (nth (1- n) lettre)))))
```

User **souvay**

File **nvse/mf.ll**

Path **/proj/dialogue/souvay**

Lundi 23 Juin 1986 11:42:7

imprimé au CRIN Campus Scientifique bp 239 54506 Vandoeuvre les Nancy

```
(de mf ()

(tycls)
(tyco  0 0 '#"ANALYSE DE TEXTE DE MOYEN - FRANCAIS")
(tyco  0 1 '#"----------------------------------------")
(tyco 39 1 '#"----------------------------------------")
(tyco  0 5 '#"INITIALISATION EN COURS")
(tyflush)

; pas de conversion Majuscule -> minuscule

(setq #:system:read-case-flag t)

; chargement des fichiers de fonction
; -----------------------------------

(load "decp.ll")
(load "acces.ll")
(load "sp.ll")
(load "menu.ll")
(load "niveau.ll")
(load "acclexi.ll")
(load "deux.ll")

; initialisation des variables
; ----------------------------

; initialisation des ensembles generiques

(setq voyelle '(A E { } I O U ))

(setq consonne '(B C D F G H J K L M N P Q R S T V W X Z \ /))


; desinences nominales

(setq #:mf:desnom '(

(S)
(E)
(E S)
))

; desinences verbales

(setq #:mf:desverbe '(

(S)
(T)
(E N T)
))

; chargement des regles

(load "/proj/dialogue/souvay/mf/rII1.ll")
(load "/proj/dialogue/souvay/mf/rII2N.ll")
(load "/proj/dialogue/souvay/mf/rIII1.ll")
(load "/proj/dialogue/souvay/mf/rIII2.ll")
(load "/proj/dialogue/souvay/mf/rIII3.ll")
(setq #:mf:reglesII2V ())

; table des paradigmes

(setq #:mf:paradygmes '(

        ( (M) (M) () () )
        ( (F) (F) () () )
        ( (M F X) (M F X) () () )
        ( (M) (M) (F) (F) )
        ( (X) (X) () () )
        ( (M F X) (M F X) (F) (F) )
        ( (M) () (F) (F) )
))

(load "/proj/dialogue/souvay/nvse/para.ll")

; initialisation des categories

(setq #:mf:invariables '(C D E G H I J K L M R S Z))
(setq #:mf:nominal '(A B N P Q))
(setq #:mf:verbal '(F))

; variable qui retient les resultats de l'analyseur II.1

(setq #:mf:resII1  ())

; variable qui retient l'historique  de l'analyseur II.1

(setq #:mf:histoII1 ())

; initialisation du lexique

(load "/proj/dialogue/souvay/lex/lex.ll")
(setq #:mf:lexique (genlexique 1))
```

User souvay

File nvse/niveau.ll

Path /proj/dialogue/souvay

Lundi 23 Juin 1986 11:42:24

imprimé au CRIN Campus Scientifique bp 239 54506 Vandoeuvre les Nancy

```
; ------------------------------------------------------------------------
;  TRAITEMENT DU MOT ORIGINE
; ------------------------------------------------------------------------

(de niveauA (mot_origine)

    (let ( (prop (niveauB mot_origine)) )
         (if (car prop)
             `(,(car prop) ,`(,mot_origine ,(cadr prop) () ))
             (let ( (mots_a_traiter
                    (mapcar 'implodech
                            (car (moteur_rIII (list (explodech mot_origine))
                                              #:mf:reglesIII1
                                              #:mf:maxIII1)))))
                  (let ( (resultat (mapcar 'uneIII1 (cdr mots_a_traiter))))
                  (let ( (dro (copy (mapcar 'cadr resultat)))
                         (gau (mapcan 'car resultat)))
                       `(,gau
                         ,`(,mot_origine
                            ,(cadr prop)
                            ,dro
                           )
                        ))
                       )))))

; ------------------------------------------------------------------------
;  TRAITEMENT D'UN MOT ISSUS DES REGLES III1
; ------------------------------------------------------------------------

(de uneIII1 (mot_courant)

    (let ( (resultat (niveauB mot_courant)) )
         `(,(car resultat) ,`(,mot_courant RIII1 ,(cadr resultat)))
    )
)

; ------------------------------------------------------------------------
;  APPLICATION DES REGLES II1
; ------------------------------------------------------------------------

(de niveauB (mot_depart)
    (let ( (mots_a_traiter (moteur_rII1 (list (explodech mot_depart))
                                        #:mf:reglesII1)))
         (let ( (resultat (mapcar 'uneII1 (car mots_a_traiter))))
         (let ( (dro (copy (mapcar 'cadr resultat)))
                (gau (mapcan 'car  resultat)) )
              `(,gau ,dro)))))

; ------------------------------------------------------------------------
;  TRAITEMENT D'UN MOT ISSUS DES REGLES II1
; ------------------------------------------------------------------------

(de uneII1 (mot_courant)
    (let ( (resultat (niveauC mot_courant)) )
         (if (car resultat)
             `(,(car resultat) ,`(,(implodech mot_courant)
                                  RII1
                                  ,(cadr resultat)
                                  ()))
             (let ((mots_a_traiter (moteur_rIII (list mot_courant)
                                                #:mf:reglesIII2
                                                #:mf:maxIII2)))
                  (let ((resIII (mapcar 'uneIII2 (cdar mots_a_traiter))))
                  (let ((dro (copy (mapcar 'cadr resIII)))
                        (gau (mapcan 'car resIII)))
                       `(,gau
                         ,`( ,(implodech mot_courant)
                             RII1
                             ,(cadr resultat)
                             ,dro))))
                       ))))

; ------------------------------------------------------------------------
;  TRAITEMENT D'UN MOT ISSUS DES REGLES II2
; ------------------------------------------------------------------------

(de uneII2 (mot_courant)

    (let ( (analyse (acces_lexique (car mot_courant))) )
         (let ( (succes (mapcan (lambda (ana)
                                  (recherche ana mot_courant hypothese))
                                analyse)) )
              (if (and analyse succes)
                  `(,succes ,`( ,(basedesi (car  mot_courant)
                                           (cadr mot_courant))
                                RII2
                                ,succes
                                ()))
                  (let ( (atraiterIII (moteur_rIII (list (car mot_courant))
                                                   #:mf:reglesIII3
                                                   #:mf:maxIII3))
                         (desi (cadr mot_courant)))
                  (let ( (resultat (mapcar 'uneIII3 (cdar atraiterIII))) )
```

```
                    (gau (mapcar 'car resultat)) )
                   `(,gau
                     ,`( ,(basedesi (car  mot_courant)
                                    (cadr mot_courant))
                         RII2
                         ()
                         ,dro)))
                   ))))))

; ---------------------------------------------------------------------------
;  ON EFFECTUE LES HYPOTHESES
; ---------------------------------------------------------------------------

(de niveauC (mot)
    (let ( (resultat `(,(hypo_inv mot)
                       ,(ldecoupages 'nom mot)
                       ,(ldecoupages 'verbe mot))) )
    (let ( (dro (copy (mapcar 'cadr resultat)))
           (gau (mapcan 'car resultat)) )
         `(,gau ,dro))))

; ---------------------------------------------------------------------------
;  DECOUPAGE D'UN MOT
; ---------------------------------------------------------------------------

(de ldecoupages (hypothese mot)

(if (equal hypothese 'nom) (print "hypothese nom")
                           (print "hypothese verbe") )

    (let ( (reglesII2 (if (equal hypothese 'nom)
                          #:mf:reglesII2N
                          #:mf:reglesII2V))
           (desinence (if (equal hypothese 'nom)
                          #:mf:desnom
                          #:mf:desverbe)))

       (let ( (resultat (mapcar 'decoupage (decoupe (list mot) desinence))))
       (let ( (dro (copy (mapcar 'cadr resultat)))
              (gau (mapcan 'car resultat)) )
              `(,gau ,dro)))))

; ---------------------------------------------------------------------------
;  APPLICATION DES REGLES II2
; ---------------------------------------------------------------------------

(de decoupage (d(coupage)

    (let ( (mots_a_traiter (moteur_rII2 (list d(coupage) reglesII2)) )
         (let ( (resultat (mapcar 'uneII2 (car mots_a_traiter))) )
         (let ( (dro (copy (mapcar 'cadr resultat)))
                (gau (mapcan 'car resultat)) )
               `(,gau
                 ,`( ,`(,(implodech (car d(coupage))
                         ,(if (null (cadr d(coupage)) ()
                              (implodech (cadr d(coupage))))
                     ,dro))))))

; ---------------------------------------------------------------------------
;  TRAITEMENT D'UN MOT ISSUS DE III3 DANS HYPOTHESE NOM/VERBE
; ---------------------------------------------------------------------------

(de uneIII3 (mot)
    (let ( (analyse (acces_lexique mot)) )
         (if analyse
             (let ( (resultat (mapcan (lambda (ana)
                                        (recherche ana `(,mot ,desi) hypothese))
                                      analyse)) )
                 `(,resultat ,`(,(basedesi mot desi)
                                RIII3
                                ,resultat)))
             `( () ,`( ,(basedesi mot desi)
                       RIII3
                       () )) )))


; ---------------------------------------------------------------------------
;  RECHERCHE D'UNE ANALYSE
; ---------------------------------------------------------------------------

(de recherche ( (mot type categorie lemme parad) (base desinence) hypothese)

    (let ( (ensemble (if (equal hypothese 'nom)
                         #:mf:nominal
                         #:mf:verbal)) )
         (if (member categorie ensemble)
             (if (equal type 'I)
                 (if (null desinence)
                     `(,`(,(concat categorie parad)
                          ,lemme
                         )))
                 (let ( (resultat (compatible hypothese desinence parad)) )
                      (if resultat
                          (mapcar (lambda (cas)
```

```
                                    ,lemme))
                                resultat)))))))

; ------------------------------------------------------------------------
;  HYPOTHESE INVARIABLE
; ------------------------------------------------------------------------

(de hypo_inv (mot)

(print) (print (implodech mot))
(print "hypothese invariable")

    (let ( (analyse (acces_lexique mot)) )
         (if analyse
             (let ( (resultat (mapcan
                                    (lambda (ana)
        (if (member (nth 2 ana) #:mf:invariables)
                    `( ,`( ,(nth 2 ana)
                           ,(nth 3 ana) ) )))
                                    analyse)) )
                  `( ,resultat ,`(,(implodech mot)
                                  ()
                                  ,resultat
                                  () )))
             (let ( (mots_a_traiter (moteur_rIII (list mot)
                                                 #:mf:reglesIII3
                                                 #:mf:maxIII3)) )
             (let ( (resultat (mapcar 'vneIII3 (cdar mots_a_traiter))) )
             (let ( (dro (copy (mapcar 'cadr resultat)))
                    (gau (mapcan 'car resultat)) )
                  `(,gau
                    ,`(,(implodech mot) () () ,dro)))
)))))

; ------------------------------------------------------------------------
;  TRAITEMENT D'UN MOT ISSUS DE III3 DANS L'HYPOTHESE INVARIABLE
; ------------------------------------------------------------------------

(de vneIII3 (mot_courant)
    (let ( (analyse (acces_lexique mot_courant)) )
         (if analyse
             (let ( (resultat (mapcan

(lambda (ana)
        (if (member (nth 2 ana) #:mf:invariables)
            `( ,`(,(nth 2 ana)
                  ,(nth 3 ana)))))
                                    analyse)) )
                  `(,resultat ,`( ,(implodech mot_courant)
                                  RIII3
                                  ,resultat)))
             `(() ,`(,(implodech mot_courant) RIII3 ()))))))

; ------------------------------------------------------------------------
;  TEST DE LA COMPATIBILITE BASE/DESINENCE
; ------------------------------------------------------------------------

(de compatible (hypothese desinence paradigme)
    (let ( (ligne (nth (1- paradigme) #:mf:paradygmes)) )
         (if (equal hypothese 'nom)
             (if (null desinence)
                 (nth 0 ligne)
                 (let ( (mb (length (member desinence #:mf:desnom)))
                        (lg (length #:mf:desnom)) )
                      (nth (1+ (- lg mb)) ligne))))))


;------------------------------------------------------------------------
;                  CREATION DE ( BASE DESINENCE )
;------------------------------------------------------------------------

(de basedesi (base desinence)
    `( ,(implodech base)
       ,(if (null desinence)
            ()
            (implodech desinence))))
```

ANNEXE 6

Exemples d'exécutions

1. Exemples de production des moteurs.

2. Exemples d'analyses.

2.1 aide à l'analyse

- la grammaire algébrique
- le compilateur
- exemples

AIGNEL
AGNEL

(AIGNEL R37 AGNEL)

ENFANT
ENFENT
ANFANT
ANFENT

(ENFANT R59 ENFENT)
(ENFANT R60 ANFANT)
(ENFENT R58 ENFANT)
(ENFENT R60 ANFENT)
(ANFANT R59 ANFENT)
(ANFANT R61 ENFANT)
(ANFENT R58 ANFANT)
(ANFENT R61 ENFENT)

KOUVENRAI
KONVENRAI
KOUVANRAI
KOUNVENRAI
KONVANRAI
KOUNVANRAI

(KOUVENRAI R39 KONVENRAI)
(KOUVENRAI R58 KOUVANRAI)
(KONVENRAI R41 KOUNVENRAI)
(KONVENRAI R58 KONVANRAI)
(KOUVANRAI R39 KONVANRAI)
(KOUVANRAI R59 KOUVENRAI)
(KOUNVENRAI R58 KOUNVANRAI)
(KONVANRAI R41 KOUNVANRAI)
(KONVANRAI R59 KONVENRAI)
(KOUNVANRAI R59 KOUNVENRAI)

```
-------------------------
AIGNEAU-0
AIGNEAL-0
AIGNEL-0

(AIGNEAU-0 R3 (AIGNEAL-0))
(AIGNEAU-0 R5 (AIGNEL-0))
-------------------------
UNG-0
UNK-0
UN-0

(UNG-0 R2 (UNK-0))
(UNG-0 R7 (UN-0))
```

```
%{
# include <stdio.h>

# define LGUNI 26

typedef struct noeud {
          int type;
          char *mot[LGUNI];
          struct noeud *gau;
          struct noeud *dro;
          struct noeud *svt;} NOEUD;

NOEUD *racine;
char retient[LGUNI];

%}
%start  axiome

%union { NOEUD *res ; }

%token  IDF  LETTRE NOMBRE
%token  LBLOC BLOC LBASE BASE CARACT PTR

%type <res> lblocs bloc entete lbases base caractere type categorie chaine ptr
%type <res> paradygme

%%

axiome          : lblocs
                        { racine=$1; }
                | /* vide */
                ;
lblocs          : lblocs bloc
                        { $$=creer_noeud(LBLOC,"",$2,NULL,$1); }
                | bloc
                ;
bloc            : entete lbases
                        { $$=creer_noeud(BLOC,"",$1,$2,NULL); }
                ;
entete          : LETTRE '=' chaine
                        { $$=$3; }
                ;
lbases          : lbases base
                        { $$=creer_noeud(LBASE,"",$2,NULL,$1); }
                | base
                ;
base            : chaine ',' caractere
                        { $$=creer_noeud(BASE,"",$1,$3,NULL); }
                ;
caractere       : type ',' categorie ',' ptr
                        { $$=creer_noeud(CARACT,"",$1,$3,$5); }
                ;
type            : IDF
                        { $$=creer_noeud(IDF,retient,NULL,NULL,NULL); }
                ;
categorie       : IDF
                        { $$=creer_noeud(IDF,retient,NULL,NULL,NULL); }
                ;
chaine          : IDF
                        { $$=creer_noeud(IDF,retient,NULL,NULL,NULL); }
                ;
ptr             : chaine ',' paradygme
                        { $$=creer_noeud(PTR,"",$1,$3,NULL); }
paradygme       : IDF
                        { $$=creer_noeud(IDF,retient,NULL,NULL,NULL); }
                | NOMBRE
                        { $$=creer_noeud(IDF,retient,NULL,NULL,NULL); }
                ;
%%
# include "princ.c"
```

```
/*************************************************************************/
/* INITIALISATION DES MARGES                                             */
/*************************************************************************/

int marge1=1;
int marge2=17;
int marge3=33;
int marge4=49;
int tab1=50;
int tab2=66;


/*************************************************************************/
genlresultats(n)

NOEUD *n;

{  if (n->type == LRES)
      { genlresultats(n->svt);
        sortie(1,1);
        copie(1,50,"; ------------------------------------------------");
        sortie(1,50);
        genresultat(n->gau); }
   else genresultat(n);
}

/*************************************************************************/

genresultat(n)

NOEUD *n;

{  char *m;
   int i,j,lg;

   lg=lgchaine( m = n->gau->mot );
   j=marge1+lg;
   for (i=marge1;i<j;++i) { lobjet[i] = *m++; }
   if (j>marge2) { sortie(marge1,j); }
   genlisteIII1(n->dro);
   if ( (n->svt) != NULL ) {genlisteIII1(n->svt);}
}

/*************************************************************************/

genlisteIII1(n)

NOEUD *n;

{  if (n->type == LIII1)
      {genlisteIII1(n->svt);
       genuneIII1(n->gau);}
   else genuneIII1(n);
}

/*************************************************************************/

genuneIII1(n)

NOEUD *n;

{  char *m;
   int i,j,lg;

   copie(marge1,7,"| rIII1   ");
   sortie(marge1,7);
   lobjet[marge1]='v';sortie(marge1,1);
   lg=lgchaine(m=(n->gau)->mot);
   j=marge1+lg;
   for (i=marge1;i<j;++i) { lobjet[i] = *m++ ; }
   if (j>marge2) { sortie(1,lg); }
   genlisteII1(n->svt);
}

/*************************************************************************/

genlisteII1(n)

NOEUD *n;

{  if (n->type == LII1)
      {genlisteII1(n->svt);
       genuneII1(n->gau);}
   else genuneII1(n);
}
```

```
/*************************************************************************/

genuneIII1(n)

NOEUD *n;

{  char *m;
   NOEUD *g,*d;
   int i,j,lg;

   copie(marge2,8,"--> RII1   ");
   sortie(marge1,marge2-marge1+8);
   g=(n->gau);d=(n->dro);
   lg=lgchaine(m=(g->gau)->mot);
   j=marge2+lg;
   for (i=marge2;i<j;++i) { lobjet[i] = *m++; }
   if (j>marge3) {sortie(marge2,lg);}
   genprob(d->gau);
   if ( (d->dro) != NULL) { genlisteIII2(d->dro); }
}

/*************************************************************************/

genlisteIII2(n)

NOEUD *n;

{  if (n->type == LIII2)
      { genlisteIII2(n->svt);
        genuneIII2(n->gau);}
   else genuneIII2(n);
}

/*************************************************************************/

genuneIII2(n)

NOEUD *n;

{  char *m;
   int i,j,lg;

   copie(marge2,7,"| rIII2     ");
   sortie(marge2,7);
   lobjet[marge2]='v';sortie(marge2,1);
   lg=lgchaine(m=(n->gau)->mot);
   j=marge2+lg;
   for (i=marge2;i<j;++i) { lobjet[i] = *m++ ; }
   if (j>marge3) { sortie(1,j); }
   genprob(n->svt);
}

/*************************************************************************/

genprob(n)

NOEUD *n;

{  geninv(n->gau);
   gennom(n->dro);
   genverbe(n->svt);
}

/*************************************************************************/

genlisteIII3(n,marge,tab)

NOEUD *n;

{  if (n->type == LIII3)
      { genlisteIII3(n->svt,marge,tab);
        genuneIII3(n->gau,marge,tab);}
   else genuneIII3(n,marge,tab);
}

/*************************************************************************/

gennom(n)

NOEUD *n;

{  copie(marge3,3,"nom   ");
   sortie(marge3,3);
   genldecoupages(n);
}
```

```
/*************************************************************************/

genverbe(n)

NOEUD *n;

{  copie(marge3,5,"verbe   ");
   sortie(marge3,5);
   genldecoupages(n);
}

/*************************************************************************/

geninv(n)

NOEUD *n;

{  char *m;
   NOEUD *g,*d,*a;
   int i,j,k,lg;

   copie(marge3,10,"invariable     ");
   sortie(marge2,marge3-marge2+10);
   g=(n->gau);d=(n->dro);a=(d->gau);
   lg=lgchaine(m=( (g->gau)->mot ));
   j=marge3+lg;
   for (i=marge3;i<j;++i) {lobjet[i] = *m++ ; }
   gendecoupe(g->gau,marge3,tab1);
   if ( a->type == NIL )
      { copie(tab1,5,"ECHEC   ");
        sortie(marge3,tab1+5-marge3);
        if (d->dro != NULL) genlisteIII3(d->dro,marge3,tab1);}
   else genanalyse(a,marge3,tab1);
}

/*************************************************************************/

genuneIII3(n,marge,tab)

NOEUD *n;
int marge,tab;

{  char *m;
   int i,j,k,lg;

   copie(marge,7,"| rIII3     ");sortie(marge,7);
   lobjet[marge]='v';sortie(marge,1);
   gendecoupe(n->gau,marge,tab);
   if ( (n->svt)->type == NIL )
      { copie(tab,5,"ECHEC     ");
        sortie(marge,tab+5-marge);}
   else genanalyse(n->svt,marge,tab);

}

/*************************************************************************/

gendecoupe(n,marge,tab)

NOEUD *n;
int   marge,tab;

{  int i,j,k,lg;
   char *m;

   switch(n->type)
     { case DCP1 : lg=lgchaine(m = n->gau->mot );
                   j=lg+marge;
                   for (i=marge;i<j;++i) { lobjet[i] = *m++; }
                   lobjet[j]='-';
                   lg=lgchaine(m = n->dro->mot );
                   k=lg+j+1;
                   for (i=j+1;i<k;++i)   { lobjet[i] = *m++; }
                   if (k>tab-2) { sortie(marge,k-marge);}
                   break;
       case IDF  : lg=lgchaine(m = n->mot);
                   j=lg+marge;
                   for (i=marge;i<j;++i) {lobjet[i] = *m++ ; }
                   if (j>tab-2) { sortie(marge,lg); }
                   break;
       case DCP3 : lg=lgchaine(m = n->gau->mot);
                   j=lg+marge;
                   for (i=marge;i<j;++i) { lobjet[i] = *m++ ; }
                   copie(j,2,"-0          ");
                   if (j>tab-2) { sortie(marge,lg+2); }
```

```
                    break; }
}

/*************************************************************************/

genlisteII2(n)

NOEUD *n;

{  if ( n->type == LII2 )
      { genlisteII2(n->svt);
        genuneII2(n->gau);}
   else genuneII2(n);
}

/*************************************************************************/

genldecoupages(n)

NOEUD *n;

{  if ( n->type == LDEC )
      { genldecoupages(n->svt);
        gendecoupage(n->gau);}
   else gendecoupage(n);
}

/*************************************************************************/

gendecoupage(n)

NOEUD *n;

{  gendecoupe(n->gau,marge3,tab1);
   genlisteII2(n->dro);}


/*************************************************************************/

genuneII2(n)

NOEUD *n;

{  char *m;
   NOEUD *g,*d,*a;
   int i,j,lg;

   copie(marge4,7,"--> II2     ");
   sortie(marge3,marge4-marge3+7);
   g=(n->gau);d=(n->dro);a=(d->gau);
   gendecoupe(g->gau,marge4,tab2);
   if ( a->type == NIL )
      { copie(tab2,5,"ECHEC     ");
        sortie(marge4,tab2+5-marge4);
        if (d->dro != NULL) genlisteIII3(d->dro,marge4,tab2);}
   else genanalyse(a,marge4,tab2);
}

/*************************************************************************/

genanalyse(n,marge,tab)

NOEUD *n;
int marge,tab;

{  if (n->type == LANA)
      { genanalyse(n->svt,marge,tab);
        genana(n->gau,marge,tab); }
   else genana(n,marge,tab);
}

/*************************************************************************/

genana(n,marge,tab)

NOEUD *n;
int marge,tab;

{  int i,j,k,lg;
   NOEUD *g,*d;
   char *m;

   if (n->type != NIL)
      { g=n->gau;d=n->dro;
        lg=lgchaine(m = g->mot);
        j=tab+lg;
```

```
        for (i=tab;i<j;++i) { lobjet[i] = *m++ ; }
        lg=lgchaine(m = d->mot);
        k=j+lg+1;
        for (i=j+1;i<k;++i) { lobjet[i] = *m++ ; }
        sortie(marge,k-marge+1);
}}
```

```
AIGNEAU          --> RIIl
                 AIGNEAU          invariable
                                  AIGNEAU          ECHEC
                                  nom
                                  AIGNEAU-0        --> II2
                                                   AIGNEAU-0        ECHEC
                                                   --> II2
                                                   AIGNEAL-0        ECHEC
                                                   --> II2
                                                   AIGNEL-0         ECHEC
                                  verbe
                                  AIGNEAU-0        --> II2
                                                   AIGNEAU-0        ECHEC
                 --> RIIl
                 AGNEAU           invariable
                                  AGNEAU           ECHEC
                                  nom
                                  AGNEAU-0         --> II2
                                                   AGNEAU-0         ECHEC
                                                   --> II2
                                                   AGNEAL-0         ECHEC
                                                   --> II2
                                                   AGNEL-0          AM agnel
                                  verbe
                                  AGNEAU-0         --> II2
                                                   AGNEAU-0         ECHEC
```

```
APIAUS          --> RIIl
                APIAUS          invariable
                                APIAUS          ECHEC
                                nom
                                APIAUS-O        --> II2
                                                APIAUS-O        ECHEC
                                APIAU-S         --> II2
                                                APIAU-S         ECHEC
                                                --> II2
                                                APIAL-S         ECHEC
                                                --> II2
                                                APEL-S          AM apel
                                verbe
                                APIAUS-O        --> II2
                                                APIAUS-O        ECHEC
                                APIAU-S         --> II2
                                                APIAU-S         ECHEC
```

```
ENKOIRES        --> RIIl
                ENKOIRES        invariable
                                ENKOIRES         J encore
                                nom
                                ENKOIRES-0      --> II2
                                                ENKOIRES-0      ECHEC
                                ENKOIRE-S       --> II2
                                                ENKOIRE-S       ECHEC
                                ENKOIR-ES       --> II2
                                                ENKOIR-ES       ECHEC
                                verbe
                                ENKOIRES-0      --> II2
                                                ENKOIRES-0      ECHEC
                                ENKOIRE-S       --> II2
                                                ENKOIRE-S       ECHEC
                --> RIIl
                ANKOIRES        invariable
                                ANKOIRES         ECHEC
                                nom
                                ANKOIRES-0      --> II2
                                                ANKOIRES-0      ECHEC
                                ANKOIRE-S       --> II2
                                                ANKOIRE-S       ECHEC
                                ANKOIR-ES       --> II2
                                                ANKOIR-ES       ECHEC
                                verbe
                                ANKOIRES-0      --> II2
                                                ANKOIRES-0      ECHEC
                                ANKOIRE-S       --> II2
                                                ANKOIRE-S       ECHEC
```

```
FU              --> RIIl
                FU              invariable
                                FU              ECHEC
                                nom
                                FU-0            --> II2
                                                FU-0            ECHEC
                                                --> II2
                                                FL-0            ECHEC
                                verbe
                                FU-0            --> II2
                                                FU-0            FC3 estre
```

```
GRANTS          --> RII1
                GRANTS          invariable
                                GRANTS          ECHEC
                                nom
                                GRANTS-0        --> II2
                                                GRANTS-0        ECHEC
                                GRANT-S         --> II2
                                                GRANT-S         BM grant
                                                                BF grant
                                                                BX grant
                                                --> II2
                                                GRANTT-S        ECHEC
                                                --> II2
                                                GRANTF-S        ECHEC
                                                --> II2
                                                GRANTK-S        ECHEC
                                                --> II2
                                                GRANTL-S        ECHEC
                                                --> II2
                                                GRANTP-S        ECHEC
                                verbe
                                GRANTS-0        --> II2
                                                GRANTS-0        ECHEC
                                GRANT-S         --> II2
                                                GRANT-S         ECHEC
                --> RII1
                GRENTS          invariable
                                GRENTS          ECHEC
                                nom
                                GRENTS-0        --> II2
                                                GRENTS-0        ECHEC
                                GRENT-S         --> II2
                                                GRENT-S         ECHEC
                                                --> II2
                                                GRENTT-S        ECHEC
                                                --> II2
                                                GRENTF-S        ECHEC
                                                --> II2
                                                GRENTK-S        ECHEC
                                                --> II2
                                                GRENTL-S        ECHEC
                                                --> II2
                                                GRENTP-S        ECHEC
                                verbe
                                GRENTS-0        --> II2
                                                GRENTS-0        ECHEC
                                GRENT-S         --> II2
                                                GRENT-S         ECHEC
```

```
VIS             --> RII1
                VIS             invariable
                                VIS             ECHEC
                                nom
                                VIS-0           --> II2
                                                VIS-0           AM visl
                                VI-S            --> II2
                                                VI-S            ECHEC
                                                --> II2
                                                VIT-S           ECHEC
                                                --> II2
                                                VIF-S           BM vif
                                                --> II2
                                                VIK-S           ECHEC
                                                --> II2
                                                VIL-S           BM vil
                                                --> II2
                                                VIP-S           ECHEC
                                verbe
                                VIS-0           --> II2
                                                VIS-0           ECHEC
                                VI-S            --> II2
                                                VI-S            ECHEC
```

```
VIVES           --> RII1
                VIVES           invariable
                                VIVES           ECHEC
                                nom
                                VIVES-O         --> II2
                                                VIVES-O         ECHEC
                                VIVE-S          --> II2
                                                VIVE-S          ECHEC
                                VIV-ES          --> II2
                                                VIV-ES          ECHEC
                                                --> II2
                                                VIF-ES          BF vif
                                verbe
                                VIVES-O         --> II2
                                                VIVES-O         ECHEC
                                VIVE-S          --> II2
                                                VIVE-S          ECHEC
```