UNIVERSITÉ DE NEUCHÂTEL
FACULTÉ DES LETTRES ET SCIENCES HUMAINES

CORRECTION AUTOMATIQUE DES ERREURS MORPHOLOGIQUES ET SYNTAXIQUES PRODUITES À L'ÉCRIT EN LANGUE SECONDE

THÈSE

présentée à la Faculté des Lettres et Sciences Humaines
de l'Université de Neuchâtel
pour obtenir le grade de docteur ès lettres

par

ETIENNE CORNU

1997

Imprimerie de l'Evole SA, Neuchâtel

UNIVERSITÉ DE NEUCHÂTEL
FACULTÉ DES LETTRES ET SCIENCES HUMAINES

CORRECTION AUTOMATIQUE DES ERREURS MORPHOLOGIQUES ET SYNTAXIQUES PRODUITES À L'ÉCRIT EN LANGUE SECONDE

THÈSE

présentée à la Faculté des Lettres et Sciences Humaines
de l'Université de Neuchâtel
pour obtenir le grade de docteur ès lettres

par

ETIENNE CORNU

1997

Imprimerie de l'Evole SA, Neuchâtel

La Faculté des lettres et sciences humaines de l'Université de Neuchâtel, sur les rapports de MM. François Grosjean, professeur à l'Université de Neuchâtel, Bernard Py, professeur à l'Université de Neuchâtel et Eric Wehrli, professeur à l'Université de Genève, autorise l'impression de la thèse présentée par M. Etienne Cornu, en laissant à l'auteur la responsabilité des opinions énoncées.

Neuchâtel, le 18 juin 1997 Le doyen : Anton Näf

Remerciements

Je remercie sincèrement Monsieur le Professeur François Grosjean, directeur de cette thèse, pour ses conseils, ses encouragements et sa grande disponibilité.

J'exprime également mes remerciements à Messieurs les Professeurs Bernard Py et Eric Wehrli, membres du jury, pour leurs suggestions et propositions dans la dernière phase de rédaction.

Mes remerciements vont aussi à tous mes collègues du laboratoire de traitement du langage et de la parole de l'Université de Neuchâtel, pour les nombreuses discussions que nous avons eues lors du projet ARCTA, et plus particulièrement à Corinne Tschumi, Lysiane Grosjean, Natalie Kübler, Cornelia Tschichold et Franck Bodmer, qui ont contribué au développement des automates. Par ailleurs, je tiens à remercier Corinne Tschumi pour la lecture critique de cette thèse.

Un grand merci également à Catherine Liecbti et Susan Duciaume, pour avoir rédigé les phrases-test utilisées lors de l'évaluation de mes logiciels.

Je remercie encore John Chandioux et Annette Grimaila, qui ont bien voulu partager avec moi leur savoir et leur expérience du monde de la commercialisation des correcteurs grammaticaux, ainsi que Dominique Maret et Ian Johnson pour leur contribution au début du projet ARCTA.

Mes remerciements vont enfin à toute ma famille, qui m'a encouragé et aidé lors de la rédaction de cette thèse.

Table des matières

Introduction

Le langage écrit permet aux êtres humains d'exprimer des idées et des faits et de les communiquer à des tiers sous une forme ordonnée et permanente. Du point de vue du lecteur, un texte écrit comprend d'une part un contenu intrinsèque, c'est-à-dire le message qui est transmis, et d'autre part une forme, caractérisée par l'usage, le style et la mise en page. Lorsqu'il prend connaissance d'un texte, le lecteur s'attend à y trouver des informations correctes, ceci sous une forme répondant à une certaine norme. Les erreurs, qu'elles touchent au contenu ou à la forme, peuvent rendre difficile la compréhension du texte, provoquer une fausse interprétation, ou encore déconsidérer l'auteur du texte aux yeux du lecteur.

Ces dernières années, avec l'essor de l'ordinateur et son utilisation pour le traitement de texte, les chercheurs ont mis au point des outils informatiques destinés à corriger les erreurs. S'ils ne permettent pas de vérifier ou de corriger le contenu du message, ce qui en soi n'est probablement pas souhaitable, ils apportent une aide à l'utilisateur pour le style, la ponctuation, l'orthographe, la grammaire et parfois aussi le choix des mots.

De nos jours, de plus en plus de personnes se trouvent dans la situation où elles doivent s'exprimer par écrit dans une langue seconde, c'est-à-dire une langue qu'elles ne maîtrisent souvent pas aussi bien que leur langue première. Ces personnes produisent donc des erreurs qu'il faut pouvoir corriger. L'objectif de notre recherche consiste à montrer comment les erreurs en langue seconde peuvent être détectées et corrigées automatiquement à l'aide d'outils informatiques. Nous nous concentrons plus particulièrement sur les erreurs morphologiques (en relation avec la forme des mots) et syntaxiques (en relation avec la structure des phrases), tout en faisant en sorte que les algorithmes et les formalismes développés puissent traiter d'autres types d'erreurs. Nous accordons une importance particulière à la qualité de la détection. Les logiciels doivent traiter les erreurs fréquentes et être fiables, en ce sens qu'ils doivent non seulement détecter et corriger une proportion importante d'erreurs, mais aussi réduire au minimum les cas où une erreur est "repérée" alors qu'elle n'est pas présente. Nous nous limitons à l'étude et au développement d'algorithmes de détection et de correction des erreurs, et n'abordons pas les aspects

appliqués. Nous espérons toutefois que les résultats de notre travail pourront être utilisés dans des domaines tels que l'enseignement des langues assisté par ordinateurs.

Notre travail comporte six chapitres. Dans le chapitre 1, nous présentons tout d'abord les recherches effectuées sur les erreurs par la linguistique appliquée et décrivons l'origine et la nature des erreurs en langue première et seconde. Nous étudions ensuite les conséquences de ces recherches pour la détection et la correction automatique d'erreurs.

Dans le chapitre 2, nous décrivons les efforts accomplis à ce jour dans le traitement automatique d'erreurs. Nous présentons tout d'abord un aperçu des algorithmes de détection et de correction utilisés actuellement par les logiciels. Nous décrivons également les caractéristiques de quelques correcteurs monolingues et bilingues encore au stade de la recherche ou déjà commercialisés. Nous soulignons, en particulier, les problèmes rencontrés par les correcteurs monolingues dans le cadre bilingue, et nous en concluons qu'ils ne sont pas adaptés à cette situation.

Dans le chapitre 3, nous présentons un premier correcteur que nous avons développé sur la base du formalisme de la grammaire lexicale et fonctionnelle (LFG). Ce logiciel traite les erreurs commises en français par des anglophones. Il permet de corriger simultanément plusieurs erreurs dans une même phrase et, en plus, corrige des erreurs même si une analyse syntaxique ne peut pas être effectuée sur l'ensemble de la phrase. Le principal avantage de ce logiciel réside dans le fait qu'il n'utilise pas de

règles de détection d'erreurs mais uniquement des règles de correction. Ceci est rendu possible grâce à l'utilisation d'un algorithme d'unification non affecté par la présence d'erreurs.

Le chapitre 4 présente l'évaluation du logiciel LFG, basée sur des phrases-test écrites par une personne indépendante. Nous constatons que le logiciel réussit à corriger un certain nombre d'erreurs, mais que les algorithmes utilisés dans cette approche sont difficilement utilisables sur le plan commercial. En effet, d'une part nous devons faire face à une progression exponentielle des données que le logiciel doit conserver simultanément en mémoire. D'autre part, nous constatons que le fait même d'utiliser l'unification comme mécanisme central pour la détection et la correction des erreurs nous limite dans le genre d'erreurs que nous pouvons traiter à l'aide de l'approche LFG.

Dans le chapitre 5, nous proposons une seconde approche pour la correction d'erreurs en langue seconde. Ce nouveau logiciel, qui traite d'erreurs de francophones qui écrivent en anglais, se caractérise par le fait que les moyens linguistiques-informatiques mis en oeuvre sont adaptés aux différents types d'erreurs à traiter, tout en faisant référence à un seul formalisme. Les mécanismes de détection utilisés par le logiciel sont établis à partir de modèles basés sur un corpus d'erreurs. Le correcteur repose sur des détecteurs de séquences (automates) qui représentent des suites de mots, de syntagmes et de signes, eux-mêmes constituant une violation d'une ou plusieurs règles de la langue cible. Cette représentation par automates offre deux avantages importants: elle rend possible la

formalisation d'un grand nombre de types d'erreurs (contrairement à l'approche LFG) et elle permet le passage rapide d'une typologie linguistique d'erreurs à un ensemble de règles utilisables par le logiciel. Le choix des deux langues (inversé par rapport au logiciel LFG) a été dicté par le fait que nous avons réalisé le logiciel basé sur des automates dans le cadre du projet ARCTA (aide à la rédaction et à la correction de textes anglais de francophones), produit d'une collaboration entre le laboratoire de traitement du langage et de la parole de l'Université de Neuchâtel et une entreprise privée. Nous estimons que les résultats obtenus lors des évaluations, et les conclusions que nous en tirons, ne dépendent en principe pas du couple de langues choisi, car les deux approches étudiées dans cette thèse s'appliquent aussi bien à l'anglais langue seconde qu'au français langue seconde.

Enfin, le chapitre 6 rend compte de l'évaluation de ce nouveau logiciel. En plus de phrases-test rédigées par des personnes indépendantes, nous avons soumis au logiciel un ensemble de phrases extraites d'un corpus d'erreurs, phrases qui contiennent également des erreurs qui ne sont pas traitées par le logiciel. Le but ici est d'illustrer la fiabilité du système. Nous verrons que cette approche obtient de bons résultats et que les algorithmes utilisés se prêtent particulièrement bien au traitement d'une gamme très étendue d'erreurs, mais que le logiciel rencontre toutefois certains problèmes lorsque la structure des phrases est complexe.

Chapitre 1

Les erreurs

Idéalement, un système de correction automatique en langue seconde intégrerait l'ensemble des règles de cette langue et toutes les données contextuelles relatives au texte qu'il doit traiter. Un tel système n'est cependant pas prêt de voir le jour, étant donné la capacité restreinte des ordinateurs actuels, les limites des algorithmes d'analyse linguistique et la pauvreté de la représentation des connaissances linguistiques et encyclopédiques. Cette situation nous contraint à informer le système des erreurs que nous voulons traiter, ceci indépendamment de la méthode utilisée pour traiter celles-ci. Il faut dès lors recueillir des renseignements

précis quant à la nature et la fréquence des erreurs, ainsi que les données nécessaires pour leur détection et leur correction. C'est dans cette optique que nous décrivons, dans les sections 1.1 et 1.2 de ce chapitre, les recherches en linguistique appliquée qui expliquent l'origine et la nature des erreurs en langue première et en langue seconde. Dans la troisième section (1.3.), nous examinons les conséquences de ces travaux pour la détection et la correction automatiques.

1.1. Les erreurs en langue première

Chacun de nous est régulièrement confronté à des textes écrits ou des énoncés oraux entachés d'erreurs. Un enfant qui dit ses premiers mots, un mode d'emploi mal traduit d'une langue à l'autre, une faute d'orthographe dans un journal: dans tous ces exemples de la vie quotidienne, le récepteur du message considère que l'émetteur a commis une erreur. On peut toutefois se demander quels sont les critères qui nous amènent à porter ce genre de jugement. En fait, la question suivante se pose: qu'est-ce qu'une erreur?

"Le petit Robert" (Robert, 1989) définit l'erreur comme une "chose fausse, erronée par rapport à une norme (différence par rapport à un modèle ou au réel)". Cette définition nous invite à comparer, d'une part, l'élément de langage produit par une personne, et d'autre part la *norme*, constituée d'un ensemble de données de référence admises comme correctes, principalement celles contenues dans les dictionnaires et les livres de grammaire. Il n'existe toutefois pas qu'une norme et une seule, fût-ce au

sein d'une même langue: ce que certains considèrent comme une erreur n'en est peut-être pas une pour d'autres. Nous parlons alors d'un *usage*, qui décrit les règles et les habitudes, écrites et orales, en vigueur dans une communauté.

Les exemples suivants illustrent des variations possibles dans l'usage d'une langue:

- l'orthographe anglaise de certains mots n'est pas partout identique: *couleur* s'écrit *colour* en Grande-Bretagne et au Canada, mais *color* aux Etats-Unis;
- l'argot ne s'utilise en principe pas dans la correspondance, ni dans les documents officiels, mais est de plus en plus toléré dans les messages transmis par courrier électronique;
- en français, on ne se sert pratiquement plus aujourd'hui du subjonctif imparfait.

L'erreur peut dès lors se définir comme un écart par rapport à un usage bien défini de la langue.

Dans notre travail, nous ne traiterons pas des erreurs en rapport avec les variations d'usage d'une communauté à l'autre, mais nous nous concentrerons uniquement sur des normes précises, soit celles décrites, par exemple, dans la "Grammaire structurale du français" (Dubois, 1967-1969) pour le français et le "Collins Cobuild English Grammar" (Sinclair, 1990) pour l'anglais.

La norme à laquelle nous allons nous référer étant ainsi déterminée, nous pouvons analyser les écarts entre celle-ci et les expressions des locuteurs. Pour cela, il est nécessaire de présenter deux notions centrales à l'étude des erreurs, soit la *compétence* et la *performance* du locuteur, que Chomsky (1965) définit comme suit:

> *Compétence*: connaissance des règles linguistiques.
>
> *Performance*: utilisation de la connaissance linguistique, par exemple dans la production et la perception de la parole.

Corder (1973), entre autres, s'est intéressé aux causes des erreurs:

> Les *erreurs de compétence* sont le reflet de l'état du système interne du locuteur et proviennent ainsi de connaissances incorrectes, que le locuteur possède et utilise dans sa production langagière.
>
> Les *erreurs de performance* sont dues, par exemple, à l'inattention ou à la fatigue (le terme anglais qu'il utilise pour définir ce type d'erreurs est *mistake*, qui se traduit par *faute*).

Dans la langue parlée, Fromkin (1988) observe deux types d'erreurs (sans distinguer les erreurs de compétence et les erreurs de performance): celles liées aux unités linguistiques (changement d'ordre, insertion ou omission de mots, par exemple) et celles en relation avec les règles grammaticales (application erronée d'une règle, par exemple). Ces erreurs apparaissent au

niveau phonologique (comme dans **ponkutation* au lieu de *computation*, erreur du premier type), morphologique (**he swimmed in the pool* au lieu de *he swam in the pool*, erreur du second type), syntaxique (**She made him to do the assignment over*, où le *to* est superflu, erreur du premier type) et sémantique (**that's a horse of another color* au lieu de *... of another race*, erreur du premier type).

Nous trouvons dans la langue écrite des types d'erreurs similaires, ainsi que des erreurs typographiques. L'insertion, l'inversion, la substitution ou l'omission de caractères apparaissent fréquemment lorsque le texte est écrit sur une machine (Yannakoudakis et Fawthrop, 1982). Ce sont là typiquement des erreurs de performance. Les exemples suivants illustrent ce genre d'erreurs:

* *alller*	Correct: *aller*	(insertion)
* *lse*	Correct: *les*	(inversion)
* *manher*	Correct: *manger*	(substitution)
* *bonour*	Correct: *bonjour*	(omission)

Les erreurs de compétence se retrouvent évidemment aussi dans la forme écrite, comme, par exemple, l'utilisation du mauvais accord du participe passé en français.

1.2. Les erreurs en langue seconde

Nous avons vu dans la section précédente que la majorité des erreurs en

L1 étaient liées à la performance. En langue seconde (L2), ce type d'erreur apparaît naturellement aussi. On y observe toutefois un grand nombre d'erreurs de compétence, qui reflètent les connaissances linguistiques qu'un apprenant possède de la L2, connaissances qui sont forcément différentes de celles d'un locuteur natif. Dans les sections qui suivent, nous présenterons les recherches en linguistique appliquée effectuées ces quarante dernières années, afin de mieux comprendre ces erreurs de compétence en langue seconde.

1.2.1. Analyse contrastive

Initialement, les recherches sur l'acquisition des langues secondes avaient pour objectif principal d'identifier des moyens pour en améliorer l'enseignement. Dans cette optique, Lado (1957) fut l'un des premiers à se rendre compte des différences entre les mécanismes qui régissent l'acquisition d'une langue première et ceux mis en oeuvre lors de l'apprentissage d'une langue seconde. Il a ainsi souligné que les apprenants d'une langue seconde subissaient fortement l'influence de leur langue première. Il a en outre affirmé que l'apprentissage était rendu difficile là où les structures des deux langues divergeaient et qu'il était facilité là où ces structures se ressemblaient. D'après Lado et les tenants de l'analyse contrastive, les apprenants ont tendance à commettre des erreurs dans le premier cas, mais pas dans le second. Ces erreurs, appelées interférences, se produisent lorsque l'apprenant applique une règle de L1 pour s'exprimer en L2. On parle également dans ce cas de transfert négatif. Examinons quelques interférences fréquentes.

a. Au niveau orthographique, le mot *adresse* s'écrit en français avec un seul 'd', alors qu'en anglais il s'écrit avec deux 'd' (*address*). Sous l'influence du français, les francophones risquent d'écrire **adress* en anglais.

b. Au niveau morphologique, Tarone, Cohen et Dumas (1983) constatent que certains apprenants francophones appliquent en anglais la structure du possessif du français. L'exemple donné est **the book of Jack*, basé sur *le livre de Jack*, au lieu de *Jack's book*.

c. Au niveau syntaxique, les différences entre l'ordre des mots de deux langues entraînent souvent des interférences, comme dans l'exemple suivant, où l'ordre nom-adjectif utilisé en français a été appliqué à l'anglais, qui utilise l'ordre inverse:

Français:	*un mur gris*
Anglais:	*a grey wall*
Interférence:	***a wall grey***

d. Au niveau lexical, nous retrouvons les faux-amis, tels que le mot français *librairie*, souvent traduit en anglais par *library* (qui signifie *bibliothèque*) au lieu de *bookstore*. Tarone et al. (1983) mentionnent également une erreur, commise en français L2 par des apprenants anglophones, dans laquelle le verbe *to know* est traduit par *savoir* au lieu de *connaître* (*to know* possède les deux sens en anglais). L'exemple suivant illustre cette erreur:

Anglais:	*I know Jean.*
Français:	*Je connais Jean.*
Interférence:	**Je **sais** Jean.*

Un nombre important de contre-exemples à ces principes ont toutefois été proposés et démontrent que les phénomènes de transfert négatif et positif ne sont pas aussi systématiques qu'admis initialement. Ainsi, des erreurs furent constatées chez les apprenants là où la théorie ne le prévoyait pas. Le contraire a également été démontré: certains apprenants n'ont rencontré aucune difficulté dans des cas où les structures des deux langues étaient différentes. Dulay et Burt (1972), en particulier, rejettent l'analyse contrastive et avancent l'hypothèse de la construction créative: l'apprentissage d'une langue seconde n'est pas influencé par la langue première, mais suit les mêmes principes que l'acquisition de la langue première chez l'enfant. Ces auteurs se basent sur une étude des erreurs anglaises commises par des enfants âgés de cinq à huit ans, de langue première espagnole, dans laquelle ils montrent que la plupart des erreurs sont dues au développement, et sont semblables à celles d'enfants acquérant une langue première. Mais l'utilisation de textes rédigés par des enfants, ainsi que la manière de classer les erreurs, ont suscité de nombreuses critiques à l'encontre de l'hypothèse de la construction créative (voir Sheen, 1980, entre autres). Les chercheurs s'accordent aujourd'hui à affirmer que le transfert constitue la cause d'une partie des erreurs commises par l'apprenant d'une langue seconde.

Les fondements de l'analyse contrastive ont subi une certaine évolution durant ces dernières années. Ainsi, selon Py (1984), l'analyse constrastive permet davantage d'interpréter les erreurs commises par les apprenants que de les prédire. L'analyse de phénomènes tels que l'interférence a également évolué depuis le travail de Lado. Comme le dit Py, ce ne sont pas les différences de structure entre deux langues qui entraînent ces phénomènes (quoiqu'elles les facilitent), mais c'est l'apprenant qui les provoque par les stratégies qu'il utilise.

1.2.2. Analyse des erreurs et interlangue

Suite aux doutes suscités par l'analyse contrastive, les recherches en linguistique appliquée se sont concentrées sur l'explication des mécanismes utilisés par les apprenants pour acquérir une langue seconde et pour communiquer dans cette langue. Les erreurs, jusque-là considérées comme gênantes et sans intérêt, ont pris une place importante dans les recherches. D'après Corder (1967), elles permettent idéalement à l'enseignant de déterminer le stade d'apprentissage de l'étudiant et à l'étudiant de tester ses hypothèses au sujet de la langue qu'il apprend. Elles démontrent surtout l'existence d'un *système linguistique* interne, que l'apprenant construit lors de l'acquisition d'une langue seconde et qui contient toutes les connaissances et les règles qu'un apprenant possède de la langue seconde (voir, par exemple, Nemser (1971), Selinker (1972), et Klein (1986)). Nemser le décrit comme un *système approximatif*, situé entre le système idéal de L1 et celui de L2, qui évolue et illustre ainsi les différentes étapes de l'apprentissage. Selinker, quant à lui, a créé le terme d'*interlangue* pour définir ce système linguistique. Selinker décrit en outre

le concept de *fossilisation*: des éléments linguistiques, des règles et des sous-systèmes sont conservés dans leur interlangue par les apprenants d'une langue seconde, indépendamment de leur âge et du type d'enseignement auquel ils sont soumis. Selon lui, les apprenants atteignent un stade où certaines parties de l'interlangue sont figées et n'évoluent plus. Plusieurs facteurs concourent à cette fossilisation; on s'accorde à dire qu'elle apparaît principalement lorsque l'apprenant estime avoir atteint un niveau de compétence qui lui permet de communiquer dans la vie de tous les jours (Grosjean, 1982).

Aussi connue sous le nom de grammaire intériorisée, l'interlangue d'un apprenant de langue seconde contient trois types de règles:

1. Les règles de L2, dont la présence indique que l'apprenant a acquis la compétence d'un locuteur natif.

2. Les règles de L1 qui ne se retrouvent pas dans L2, dont l'utilisation constitue ce que nous appelons des interférences (voir la section précédente).

3. Les règles n'appartenant ni à L1, ni à L2, construites lors de l'apprentissage d'une langue, seconde ou non. L'utilisation de telles règles provoque des erreurs *intralangues*.

Les chercheurs ont identifié un certain nombre de types d'erreurs intralangues (voir entre autres Richards (1971), Grosjean (1982) et Tarone et al. (1983)). Le type le plus connu est la *surgénéralisation* de règles de

L2, illustrée ci-dessous aux différents niveaux linguistiques.

a. Au niveau morphologique, nous observons, par exemple, le cas où la règle de construction du prétérit, valable pour les verbes réguliers, est appliquée par erreur à un verbe irrégulier:

Anglais:	*I saw him.*
Erreur:	***I seed** him.*

b. Au niveau syntaxique, la structure des phrases interrogatives est parfois utilisée dans le discours indirect (Tarone et al., 1983):

Anglais:	*I don't know what it is.*
Erreur:	**I don't know what **is it**.*

c. Au niveau sémantique, le sens des mots est parfois mal interprété (Tarone et al., 1983):

Anglais correct:	*He is handsome.*
Erreur:	**He is **pretty**.*

Parmi les autres types d'erreurs intralangues recensés, nous trouvons la simplification, caractérisée par l'omission de mots, comme dans **Ich Mädchen* (*moi fille*) (Grosjean, 1982) et la surélaboration, dans laquelle l'apprenant produit des phrases compliquées en voulant trop bien faire, comme dans **Buddy, that's my foot **which** you're standing on* (correct:

... *my foot you're standing on*) (Tarone et al., 1983).

La surgénéralisation, la simplification, et la surélaboration font partie de l'ensemble des *stratégies de communication* que l'apprenant utilise lorsqu'il s'exprime en L2 (Selinker, 1972; Faerch et Kasper 1983; entre autres). Parmi celles-ci, il existe également des stratégies qui ne conduisent pas à des erreurs. Ainsi, dans les cas de l'évitement lexical, l'apprenant n'emploie pas certains mots par manque de connaissances du vocabulaire (Blum-Kulka et Leventson, 1983). Par exemple, certains apprenants préfèrent utiliser des mots dont l'orthographe ne présente pas de difficulté (comme *get* au lieu de *receive*).

En parallèle avec les recherches à caractère psycholinguistique décrites jusqu'ici, certains chercheurs se sont intéressés aux aspects pédagogiques de l'analyse des erreurs. Il s'agit principalement de recenser les erreurs et de les classer suivant des paramètres tels que la partie du discours où l'erreur apparaît (nom, verbe, adjectif, etc.), le niveau linguistique où elle se situe (morphologique, syntaxique, phonétique, etc.), le mécanisme qui l'explique (interférence, surgénéralisation, etc.), ceci dans le but de mieux connaître les erreurs produites par des ensembles spécifiques d'apprenants et ainsi de pouvoir les prédire. Comme le souligne Py (1975), les deux approches, psycholinguistique et pédagogique, sont étroitement liées: les études psycholinguistiques et pédagogiques "... non seulement ne s'excluent pas, mais devraient se présupposer les unes des autres".

Porquier (1977) formule diverses critiques à l'encontre de l'approche

pédagogique. Il remarque en particulier que les corpus utilisés se basent en général sur une population trop restreinte et un mélange inadéquat de sources, comme la dictée et la rédaction, l'écrit et l'oral, ou la traduction et les tests. Porquier met également en lumière de nombreux problèmes liés aux critères de classification des erreurs. Il montre, par exemple, qu'il est souvent difficile de déterminer avec exactitude le mot sur lequel l'erreur se situe et le mécanisme qui entre en jeu (ex. substitution ou addition).

L'approche pédagogique a malgré tout apporté une contribution certaine à l'enseignement, non pas grâce à ses résultats directs, mais par les questions qu'elle a soulevées en relation avec la compréhension et la classification des erreurs (Besse et Porquier, 1984).

1.2.3. Recherches récentes

Les recherches sur l'interlangue en linguistique appliquée se sont concentrées ces dernières années sur l'étude des facteurs qui l'influencent, comme l'âge de l'apprenant, l'environnement social et la connaissance d'autres langues secondes. Ainsi, Scarcella et Lee (1989) ont analysé les erreurs commises par des étudiants coréens, en fonction de leur durée de résidence aux Etats-Unis. Ils n'ont pas constaté de différence majeure, au niveau de la compétence morphologique et syntaxique, entre les étudiants qui vivent aux Etats-Unis depuis moins de 3 ans et ceux qui y résident depuis plus de 5 ans. Par contre, ils ont remarqué que les apprenants du second groupe commettaient nettement moins d'erreurs lexicales que ceux du premier, phénomène qu'ils ont attribué au fait que les étudiants en

question avaient été exposés aux éléments lexicaux dans des contextes plus variés. Scarcella et Lee en ont conclu que les règles morphologiques et syntaxiques peuvent être acquises indépendamment des connaissances lexicales.

Dans un effort de synthèse, Py (1993) propose que l'ensemble des éléments qui entrent en jeu lors de l'acquisition d'une langue seconde soient pris en compte dans ce qu'il nomme le "territoire de l'apprenant". Ce dernier comprend trois pôles: le système (l'interlangue), les tâches (les activités communicatives) et les normes. Py souligne que ces trois pôles sont indissociables, puis décrit les situations où l'apprenant privilégie, de manière consciente ou inconsciente, un pôle par rapport à un autre. Lorsque la priorité est donnée au système, par exemple, l'apprenant conçoit la langue seconde comme un ensemble de règles (vocabulaire + grammaire). Lorsque la priorité est donnée à la tâche, c'est l'accomplissement de la tâche qui prime et non la qualité linguistique du message.

Des recherches importantes ont en outre été effectuées dans le domaine de la *grammaire universelle*. Initialement introduite par Chomsky (1965), cette notion repose sur l'hypothèse que le langage est inné chez les êtres humains. Chomsky associe les principes linguistiques à des paramètres, dont les valeurs sont établies par l'enfant grâce à l'expérience qu'il acquiert au cours de l'apprentissage de sa langue première. Il met notamment en évidence un paramètre tenant compte des différences entre les langues qui acceptent un sujet nul et celles qui n'en acceptent pas (en

espagnol, par exemple, où *Il pleut* se dit *Llueve*). Flynn (1988), dans le but d'intégrer la construction créative et l'analyse contrastive, montre comment la notion de grammaire universelle permet d'expliquer les phénomènes liés à l'acquisition des langues secondes. Ainsi, lorsque l'apprenant n'a pas établi de valeur pour un paramètre, parce qu'il n'a pas été exposé aux données appropriées par exemple, l'apprentissage de la L2 s'effectue comme celui de la L1 et entraîne les effets décrits par la théorie de la construction créative. Par contre, si une valeur a déjà été attribuée à un paramètre et qu'elle diffère entre les deux langues, l'apprenant doit la "modifier", ce qui produit les effets décrits par la théorie de l'analyse contrastive. Même si plusieurs travaux ont donné des arguments en faveur du rôle de la grammaire universelle dans l'apprentissage d'une langue seconde (Flynn et O'Neil, 1988), certains chercheurs pensent que la grammaire universelle n'est pas accessible aux apprenants adultes d'une langue seconde (voir White, 1992, pour une discussion de ceci).

1.3. Conséquences pour le traitement automatique des erreurs

Les études que nous venons de décrire fournissent des renseignements importants sur la nature des erreurs en langue seconde. Dans cette section, nous évoquerons les conséquences de ces recherches pour le traitement automatique d'erreurs.

Comme nous l'avons vu, les erreurs apparaissent à tous les niveaux linguistiques: orthographique ou phonologique (selon la modalité),

morphologique, syntaxique, et sémantique. Or, la détection et la correction d'erreurs à un niveau donné requièrent souvent des informations en provenance d'un autre niveau. Par exemple, une erreur morphologique d'accord du participe passé conjugué avec *avoir* doit être détectée par l'examen de la position du complément d'objet direct (donnée syntaxique). Nous verrons également que le niveau des données linguistiques requis pour la correction d'une erreur peut ne pas être le même que celui requis pour sa détection.

Nous examinons ci-dessous les données relatives aux différents niveaux du traitement de l'écrit et leur application à l'analyse, à la détection et à la correction d'erreurs à ces niveaux.

1.3.1. Orthographe

Pour les êtres humains, reconnaître une faute d'orthographe est une tâche relativement aisée. Nous connaissons de nombreux mots et sommes constamment exposés aux nouveautés introduites dans notre langue, comme, par exemple, les emprunts à une autre langue et les nouveaux termes scientifiques.

Plusieurs types de connaissances nous permettent de déterminer de manière "automatique" la meilleure solution pour corriger ou remplacer un mot mal orthographié:

- Lors de l'utilisation de l'ordinateur, l'insertion, l'inversion, la substitution et l'omission de caractères apparaissent fréquemment. Dans

le cas de la substitution d'un caractère par un autre, la connaissance de la configuration du clavier utilisé permet parfois de reconnaître quelle touche a été actionnée à la place d'une autre.

- Phonétique: il arrive parfois que l'orthographe de sons identiques ou proches soit utilisée de manière erronée (*au* au lieu de *eau* ou *o*, *en* au lieu de *an*, par exemple).
- Analyse comparative de L1 et L2: grâce à celle-ci, les interférences possibles de L1 peuvent être identifiées a priori.

La présence d'un mot réel ne signifie pas toujours qu'il soit orthographié correctement dans le contexte de la phrase. En effet, il se peut que l'utilisateur ait substitué par erreur un homophone au mot correct. Un exemple cité fréquemment est celui de *there*, *they're* et *their*, qui se prononcent tous de la même façon. Pour détecter et corriger l'erreur, nous devons faire appel à des informations supplémentaires (syntaxiques dans ce cas).

Les ordinateurs n'ont à leur disposition qu'un nombre fini de mots de référence, soit le *dictionnaire d'analyse*. Signalons au passage que l'on estime en général qu'un dictionnaire d'analyse valable doit comprendre un minimum de cent mille mots. Le fait qu'un mot ne se trouve pas dans ce dictionnaire n'implique pas toujours qu'il soit faux, car le dictionnaire est forcément incomplet; des algorithmes morphologiques peuvent être appliqués, dans certains cas, pour déterminer s'il s'agit vraiment d'une erreur (voir la section suivante).

1.3.2. Morphologie

La morphologie traite de la forme des mots. Lorsque nous écrivons, nous appliquons des transformations morphologiques de deux sortes: les flexions, qui permettent de construire le pluriel des noms et conjuguer les verbes, par exemple, et les dérivations, qui produisent de nouveaux mots, comme *chanteur*, créé à partir de *chanter*. Dans le premier cas, des traits morphologiques, comme 'singulier', 'pluriel', 'présent' et 'troisième personne' sont d'habitude associés aux transformations morphologiques.

Une analyse morphologique nous permet d'établir les transformations que les mots ont subies, et donc de déterminer la valeur des traits morphologiques lorsque ceux-ci sont définis. Elle s'opère d'habitude en décomposant les mots en racine et affixe(s). Par exemple, le nom *claviers* se compose de la racine *clavier* et du suffixe *-s*, qui est la marque du pluriel. L'analyse morphologique s'applique également pour identifier les dérivations: l'adjectif *unfriendly* se compose de la racine *friend*, du préfixe *un-* et du suffixe *-ly*.

L'analyse morphologique est utile dans plusieurs situations. Considérons tout d'abord le cas où le mot analysé n'a pas été construit correctement. La décomposition en racine et affixe(s) nous permet d'émettre une hypothèse sur le sens que l'utilisateur voulait donner au mot et donc d'identifier une solution possible pour remplacer le mot incorrect. Dans le cas de **eated*, par exemple, l'analyse morphologique produit la racine, *eat*, et le suffixe *-ed*, marque habituelle du prétérit. Il se peut donc que la solution soit *ate*, prétérit de *to eat*. Toutefois, comme nous l'avons vu

précédemment, il peut exister d'autres candidats possibles, comme *heated*, *seated*, *eaten* et *eater*. Ainsi, même si une erreur de ce type peut être détectée grâce à une décomposition morphologique, sa correction passe par une analyse à un plus haut niveau linguistique (syntaxique ou sémantique, par exemple). Il est en effet impossible, sans information supplémentaire, de déterminer quelle solution est la bonne, *ate*, *heated*, *seated*, *eater*, ou *eaten*.

L'analyse morphologique est également utilisée pour identifier les traits des mots, comme 'pluriel', 'féminin' et 'présent'. Ces traits jouent un rôle principalement dans les règles d'accord et dans celles applicables au temps des verbes. Ici, des connaissances supplémentaires (syntaxiques, en particulier) sont souvent nécessaires pour détecter et pour corriger les erreurs. Dans le cas de l'accord sujet-verbe, par exemple, le sujet et le verbe principal doivent être connus avant que la détection puisse s'effectuer. Etant donné que les ordinateurs utilisent fréquemment des dictionnaires d'analyse qui contiennent les mots sous toutes leurs formes morphologiques, l'extraction des traits morphologiques ne leur pose pas de gros problèmes, si ce n'est que le manque de données contextuelles crée des ambiguïtés dans les valeurs des traits. Ainsi, le mot *mange* possède un grand nombre d'interprétations morphologiques: 1ère ou 3ème personne du singulier de l'indicatif présent ou du subjonctif présent, ou encore 2ème personne du singulier de l'impératif. Si ces ambiguïtés ne posent pas de problèmes lors de la détection d'erreurs, elles peuvent par contre entraîner une explosion exponentielle des données que le logiciel doit garder en mémoire lors d'une analyse syntaxique. Le chapitre 3 décrit ce phénomène

plus en détail.

1.3.3. Syntaxe

Alors que la morphologie traite de la structure des mots, la syntaxe traite de la manière de combiner les mots ou les syntagmes pour former des phrases. Les connaissances syntaxiques que l'on obtient à l'aide d'une analyse comprennent en général les éléments suivants:

- Catégorie syntaxique de chaque mot (nom, adjectif, verbe, etc.).
- Découpage en syntagmes (nominaux, verbaux, prépositionnels, etc.).
- Assignation de fonctions (sujet, complément direct, etc.).

Dans les paragraphes précédents, nous avons souligné à plusieurs reprises l'importance de la syntaxe dans la détection et la correction d'erreurs orthographiques et morphologiques. Bien évidemment, la syntaxe joue un rôle encore plus important dans la correction d'erreurs syntaxiques, qui apparaissent fréquemment en langue seconde. Considérons ainsi quelques exemples où des connaissances syntaxiques permettent de détecter ou de corriger une erreur:

(1) * *They wanted to make* ***there*** *bed.* (correct: *their*)
(2) * *The boy* ***eated*** *the whole cake.* (correct: *ate*)
(3) * *Ils sont* ***parti*** *hier soir.* (correct: *partis*)
(4) * *A* ***wall grey*** *surrounds the park.* (correct: *grey wall*)

Dans (1), la connaissance de la structure des compléments de *to want* et *to make* permet d'établir que *there* devrait être remplacé par *their*. Dans (2), le fait que *eated* soit précédé et suivi d'un syntagme nominal nous indique qu'il devrait probablement s'agir d'un verbe. Cependant, les informations syntaxiques ne sont pas suffisantes pour déterminer si *eated* doit être remplacé par *ate* (à la suite de la mauvaise construction du prétérit), *heated* ou *seated* (à la suite d'une omission de la première lettre). Dans (3), les connaissances syntaxiques nous indiquent que *parti* est un participe passé conjugué avec *être* et dont le sujet est *ils*. *Parti* doit donc être accordé au pluriel. Dans (4), les informations syntaxiques montrent que *a wall grey* est probablement un syntagme nominal et que l'adjectif *grey* y occupe la mauvaise place par rapport au nom.

Les logiciels qui procèdent à une analyse syntaxique rencontrent en général tous les mêmes difficultés, que ce soit lors de la détection d'erreurs ou lors d'autres tâches (par exemple, interrogation de bases de données, traduction, etc.):

- La quantité d'informations syntaxiques à disposition lors de l'analyse ne couvre pas toutes les structures de phrases possibles.
- La complexité des règles syntaxiques fait que le nombre de solutions partielles qui doivent être considérées simultanément peut dépasser les ressources de l'ordinateur.
- Certaines ambiguïtés ne peuvent être levées qu'à l'aide d'informations de niveaux linguistiques plus élevés (sémantique, pragmatique).

- La présence d'erreurs dans le texte peut fausser l'analyse ou la rendre impossible. Par exemple, les mauvaises règles d'analyse syntaxique peuvent être appliquées lorsque deux mots sont inversés.

Etant donné ces difficultés, certains choix doivent être faits lors de la programmation d'algorithmes d'analyse. Par exemple, on préfère parfois sacrifier la qualité des informations obtenues afin de réduire le temps d'exécution.

1.3.4. Sémantique et pragmatique

La sémantique et la pragmatique traitent de la signification et du sens des mots et des énoncés. La signification du mot *librairie*, par exemple, regroupe des informations telles que le fait qu'il s'agit d'un magasin, qu'on y achète des livres, qu'elle a des portes, des étalages, des caisses enregistreuses, et ainsi de suite. Afin de déterminer la signification d'une phrase, nous combinons les sens des mots et des syntagmes en créant des relations sémantiques entre eux. Ainsi, la phrase *Elle a acheté deux livres à la librairie* est cohérente car les relations entre les mots et syntagmes satisfont aux conditions imposées par leur sens. Cette combinaison réduit également au passage les ambiguïtés dues aux multiples interprétations de chaque mot. En effet, les autres sens de *livres* (monnaie et unité de masse) peuvent être écartés car ils sont sans rapport avec le sens du reste de la phrase.

Considérons la phrase *?Elle a acheté deux saucisses à la librairie*. Il existe ici un conflit possible entre le sens de *saucisses* d'une part et *acheter* et *librairie* d'autre part. En effet, on ne peut en principe pas acheter de saucisses dans une librairie. Cette phrase contient donc une erreur. S'il nous est relativement facile de détecter ce type d'erreurs, leur correction est souvent impossible car elle requiert des informations qui ne sont pas contenues dans la phrase même.

Il existe toutefois des situations où la correction d'erreurs sémantiques est possible. Considérons ainsi la phrase *?Elle a emprunté deux livres de la librairie*, dans laquelle nous pouvons également identifier une erreur (conflit entre le sens de *emprunter* et celui de *librairie*). Si nous savons que l'auteur de la phrase est anglophone, nous pouvons émettre l'hypothèse qu'il y a eu confusion de la part de l'auteur au sujet du sens de *librairie*, car *librairie* est un faux-ami de *library*, mot anglais signifiant *bibliothèque*. Nous pouvons donc corriger l'erreur en remplaçant *librairie* par *bibliothèque*.

Les connaissances sémantiques sont également utiles pour la correction d'erreurs à d'autres niveaux. Dans l'exemple cité plus haut, **the boy eated the whole cake* (erreur morphologique dans **eated*), il nous était impossible avec la seule syntaxe de déterminer si *eated* devait être remplacé par *ate* (prétérit de *to eat*) *heated* (prétérit de *to heat*) ou *seated* (prétérit de *to seat*). La connaissance de la signification de *to eat*, de *to heat* de *to seat* et de *cake* nous permet de déduire que *eated* doit être remplacé par *ate* ou par *heated*, et non pas par *seated*.

Pour que l'ordinateur parvienne à détecter et corriger les erreurs décrites ci-dessus, il devrait disposer d'un ensemble très complet de données sémantiques et pragmatiques. Il devrait connaître tous les contextes dans lesquels *acheter*, *saucisses*, *livres*, *emprunter*, *librairie*, *to eat* et *to seat* sont susceptibles d'apparaître. En fait, il n'existe actuellement pas de moyens de spécifier toutes les connaissances nécessaires, ni de les introduire dans un ordinateur.

Conclusion

Comme nous l'avons vu, les erreurs en langue seconde ne sont pas toujours systématiques, bien qu'elles apparaissent avec une certaine régularité et qu'il existe souvent des explications logiques à leur présence. Un logiciel de correction automatique doit donc tenir compte du fait que les textes à corriger contiennent des erreurs pour lesquelles il n'est pas programmé. Nous remarquons également que les erreurs apparaissent à tous les niveaux linguistiques. Alors que nous avons en principe accès, en tant qu'êtres humains, à toutes les informations requises pour détecter et corriger les erreurs à ces niveaux, les données contenues dans les logiciels de correction automatique ne suffisent souvent pas à accomplir cette tâche convenablement.

Dans le prochain chapitre, nous étudions le fonctionnement de plusieurs logiciels de correction automatique, certains déjà commercialisés, d'autres

au stade de projet. Nous analysons en particulier les mécanismes qu'ils utilisent pour tenir compte des difficultés inhérentes au traitement automatique et notamment à la correction d'erreurs en langue première et en langue seconde.

Chapitre 2

La correction automatique

Ce chapitre décrit les efforts accomplis jusqu'à ce jour pour traiter automatiquement les erreurs dans les textes rédigés à l'aide de l'ordinateur. Nous présentons tout d'abord un aperçu des algorithmes de détection et de correction utilisés actuellement par les logiciels (section 2.1.). Nous décrivons ensuite quelques correcteurs monolingues, soit au stade de recherche, soit commercialisés (section 2.2.). Nous soumettons un texte écrit en français par un anglophone à un correcteur monolingue commercial et mettons en évidence les problèmes que ce dernier rencontre. Le correcteur choisi est "Le correcteur 101", de Machina Sapiens. Nous terminons ce chapitre par une étude des correcteurs bilingues (au stade de projet et commerciaux) développés ces dernières années (section 2.3.).

Nous soumettons également un texte à un correcteur bilingue commercial. Pour ce faire, nous avons choisi un texte écrit en anglais par un francophone, ainsi que le correcteur "Grammatik anglais - pour francophones".

2.1. La correction automatique

La grande majorité des logiciels de traitement de texte commerciaux actuels comprennent une série d'outils destinés à assister l'utilisateur lors de la rédaction d'un texte. Nous distinguons deux types d'outils: les outils de référence, tels un dictionnaire des synonymes et un conjugueur, et les outils de correction automatique, comme le correcteur orthographique et le correcteur grammatical. Dans le premier type d'outils, l'utilisateur dirige la recherche des informations qu'il désire obtenir en sélectionnant des mots ou des commandes parmi des listes affichées par le logiciel. Dans le second type, qui nous intéresse ci-dessous, c'est le logiciel qui détermine, grâce à ses algorithmes, les informations à présenter à l'utilisateur.

2.1.1. Correction orthographique et grammaticale

Parmi les outils de correction automatique, les correcteurs orthographiques furent les premiers à voir le jour et restent les plus utilisés. Le fait qu'ils traitent en général chaque mot du texte de manière isolée, sans tenir compte des autres mots qui l'entourent, rend leur application plutôt limitée. Il leur est ainsi normalement impossible de détecter d'autres erreurs que celles où un mot est mal orthographié. De plus, la majorité ne peut pas identifier les cas d'orthographe incorrecte lorsque le mot en question est

parfaitement valable hors contexte. Par exemple, dans l'expression *une voie de chemin de fer, voie* pourrait s'écrire *voix, vois*, ou encore *voit*, sans qu'un correcteur orthographique traditionnel puisse détecter l'erreur. La popularité dont jouissent ces correcteurs encore aujourd'hui provient en grande partie du fait qu'ils remplissent de manière très satisfaisante le rôle que leur attribuent les utilisateurs. En effet, ils sont plutôt rapides, complets (il existe des logiciels qui s'appuient sur des dictionnaires spécialisés dans des domaines très variés) et offrent la plupart du temps à l'utilisateur des solutions acceptables pour remplacer les mots incorrects. La difficulté principale dans le développement d'un correcteur orthographique réside d'ailleurs dans l'identification et le tri des solutions. Relevons qu'il n'existe pas, à notre connaissance, de correcteurs orthographiques commerciaux munis d'algorithmes d'identification et de tri des solutions conçus spécialement pour les utilisateurs de langue seconde. Ces outils pourraient, par exemple, déterminer s'il s'agit d'une interférence de la langue première ou d'une surgénéralisation d'une règle de la langue seconde.

Comme leur nom l'indique, la fonction des correcteurs grammaticaux consiste à détecter et corriger les erreurs grammaticales contenues dans un texte. Alors que les correcteurs orthographiques interprètent le texte comme une suite de mots isolés, les correcteurs grammaticaux traitent les textes phrase par phrase. La définition de ce qu'est une phrase varie bien entendu suivant l'interprétation des signes de ponctuation que font les différents produits. Ces différences n'entrent toutefois pas en ligne de compte dans notre étude des correcteurs grammaticaux.

La définition d'une "erreur grammaticale" varie suivant les logiciels. Certains s'en tiennent au sens strict du terme; ils se concentrent sur la correction des règles grammaticales qui font fréquemment l'objet d'erreurs, comme l'accord du participe passé et l'accord entre le sujet et le verbe. D'autres logiciels s'assurent également du respect des conventions stylistiques et typographiques. Ils vérifient, par exemple, que les phrases débutent par une majuscule et qu'elles ne comptent pas plus d'un nombre déterminé de mots. Nous nous intéressons ici uniquement à la première approche, c'est-à-dire que nous nous limitons à l'aspect purement grammatical de la correction automatique.

Comme nous allons le voir dans le reste de ce chapitre, les correcteurs grammaticaux (commerciaux ou au stade de projet) ne détectent en général pas les erreurs sémantiques et pragmatiques. D'une part, les erreurs de ce type sont relativement peu fréquentes, sinon inexistantes, dans un cadre monolingue (voir section 1.1). D'autre part, la grande quantité de connaissances requises pour un traitement complet à ces niveaux dépasse largement les capacités des ordinateurs d'aujourd'hui.

2.1.2. Fonctionnement de la correction grammaticale

Pour des raisons évidentes, les algorithmes utilisés par les correcteurs grammaticaux commerciaux ne sont pas du domaine public. Nous pouvons toutefois en décrire les principes de base, d'une part en consultant la documentation fournie avec les produits, et d'autre part en examinant leur comportement face à des phrases spécialement choisies. Notre but se

limitant à étudier le fonctionnement des correcteurs de manière générale, nous ne nommons pas ces produits dans la description qui suit.

Nous pouvons distinguer dans le travail effectué par les correcteurs grammaticaux quatre opérations principales, exécutées l'une après l'autre ou en parallèle: découpage, analyse, détection et correction.

- Découpage et étiquetage

  Le logiciel recherche tout d'abord les bornes d'une phrase et divise celle-ci en mots et signes de ponctuation. Chaque mot est ensuite recherché dans un dictionnaire d'analyse et les traits qui lui sont propres en sont extraits. Suivant les cas, ces traits comprennent la forme canonique du mot, les catégories syntaxiques (les mots en ont souvent plusieurs), les informations morphologiques (nombre, genre, personne) et la structure de ses compléments s'il s'agit d'un nom ou d'un verbe.

- Analyse

  Le logiciel construit des données supplémentaires qui facilitent la détection, comme l'arbre syntaxique de la phrase.

- Détection d'erreurs

  Le logiciel utilise des règles de détection d'erreurs en se reposant sur les données produites lors des deux premières phases.

- Correction

  Le logiciel applique des algorithmes de correction afin de déterminer ce qu'il faut modifier dans le texte, opération qu'il effectuera lui-même ou qu'il conseillera à l'utilisateur.

Les sections suivantes décrivent le mode de fonctionnement de quatre types de correcteurs grammaticaux qui utilisent ces quatre opérations principales à différents niveaux de complexité. Il est important de noter que les auteurs de certains produits ont décidé d'ajuster leurs algorithmes afin de réduire les fausses détections, quitte à réduire également le nombre d'erreurs détectées. Bien que directement lié aux possibilités fournies par les algorithmes de détection, ce genre d'optimisation varie d'un produit à l'autre, suivant les caractéristiques désirées par ses concepteurs. Dans les descriptions ci-après, nous mentionnons les méthodes qui permettent d'effectuer ces optimisations, sans toutefois apporter plus d'informations sur leur mise en oeuvre dans les produits en question.

a. Mise en correspondance

Pour les produits qui se basent sur la mise en correspondance, la détection d'erreurs s'effectue à l'aide de règles permettant d'identifier des suites de mots particuliers. Ces règles sont en général constituées d'un ensemble de conditions qui, lorsqu'elles sont remplies, déclenchent l'affichage d'un message d'erreur. Dans ce type de produit, la phase d'analyse est normalement inexistante ou réduite au minimum. Elle peut par exemple avoir pour fonction de regrouper des prépositions composées (comme *au-dessus de*). Ce mode de fonctionnement requiert un nombre très important de règles, car celles-ci doivent tenir compte de toutes les situations

possibles. Par exemple, les règles de détection d'erreurs d'accord sujet-verbe ne connaissent probablement pas la nature du sujet de chaque verbe principal. Le logiciel doit donc supposer que le sujet se trouve directement avant un verbe et inclure des règles spécifiques pour les cas où le sujet est formé d'un pronom personnel, d'un syntagme nominal simple (article + nom) ou d'un syntagme nominal plus complexe.

Les produits de ce type se prêtent particulièrement bien à la détection d'erreurs dont le domaine ne dépasse pas quelques mots et où les mots concernés occupent des positions adjacentes. Voici quelques exemples:

- Accord des adjectifs de couleur. Exemple: **La maison blanc.*
- Accord du participe passé (les cas complexes ne sont pas traités). Exemple: **Ils sont parti.*
- Accord sujet-verbe (idem). Exemple: *Ils regarde les oiseaux.*
- Emploi des auxiliaires (*avoir* au lieu de *être*, par exemple). Exemple: **Il a devenu tout rouge.*
- Accord du substantif avec le déterminant. Exemple: **Les enfant.*
- Elision. Exemple: **Lorsque il est venu.*
- Emploi et orthographe des adjectifs numéraux. Exemple: **dix huit échantillons* (trait d'union manquant).
- Suite impossible de catégories syntaxiques. Exemple: **Il ma donne un livre.*
- Emploi des prépositions, lorsque celles-ci se trouvent directement après le verbe. Exemple: **Il a failli de rater le train.*

Pour éviter la détection d'erreurs lorsque le texte n'en contient pas, les règles contenues dans ce type de correcteur doivent être très précises. De plus, les messages d'erreur sont souvent relativement flous pour les cas où le logiciel peut se tromper. L'étape de correction est donc directement liée à celle de détection: à chaque règle de détection correspond un message d'erreur, dans lequel tous les paramètres proviennent directement de la règle de détection.

b. Ilots syntaxiques

Les produits qui procèdent par îlots syntaxiques utilisent également des règles de mise en correspondance pour détecter les erreurs. Ils s'appuyent par contre sur une base linguistique plus complète, construite lors de la phase d'analyse. Celle-ci fait appel à des algorithmes simples, mais rapides et efficaces, qui exécutent un certain nombre d'opérations. Par exemple:

- Désambiguïsation syntaxique. Lorsqu'un mot possède plus d'une catégorie syntaxique, celles sans rapport avec le contexte sont éliminées.
- Construction des bornes des syntagmes nominaux.
- Construction des groupes verbaux.
- Construction des bornes des propositions relatives.
- Traitement des conjonctions.
- Identification du sujet et des compléments de chaque verbe principal.

Ce type de produit ne tente pas d'analyser toute la phrase. Dans des cas simples, comme *Il dort*, par exemple, *Il* est bien reconnu comme sujet du verbe principal *dort*, ce qui suffit pour compléter un niveau d'analyse. Par contre, dans des cas plus complexes, l'ensemble de la phrase n'est pas considéré comme un tout, mais plutôt comme une série d'éléments disjoints.

Les solutions produites lors de l'analyse ne sont pas uniques. Par exemple, le logiciel garde les syntagmes nominaux simples et les syntagmes nominaux complexes dans une liste, afin de pouvoir utiliser les deux, suivant les situations. Autre exemple, le logiciel vérifie l'accord du nom dans chaque syntagme nominal simple et utilise les syntagmes complexes lors de la vérification de l'accord sujet-verbe.

La gamme d'erreurs traitées est comparable à celle des produits de la première catégorie, comme le sont les méthodes utilisées pour diminuer les cas de fausse détection et les méthodes de correction. Nous pouvons toutefois constater que les produits faisant appel aux îlots syntaxiques cernent mieux les erreurs et donnent généralement des messages d'erreur moins flous, grâce aux connaissances supplémentaires obtenues lors de la phase d'analyse.

c. Analyse syntaxique simple

Les programmes qui exécutent une analyse syntaxique simple combinent souvent la phase d'analyse et celle de détection. L'analyse s'effectue à l'aide d'un ensemble de règles syntagmatiques, qui représentent les contraintes syntaxiques de la langue. Par exemple, certaines règles

syntagmatiques spécifient qu'un syntagme nominal peut être formé d'un article suivi d'un adjectif, puis d'un nom (comme dans *la jeune fille*), qu'un groupe verbal peut être formé d'un verbe suivi d'un syntagme nominal (comme dans *conduit la voiture*), et qu'une phrase peut être formée d'un syntagme nominal suivi d'un groupe verbal (comme dans *la jeune fille conduit la voiture*). Lorsqu'elles sont appliquées successivement au texte à analyser, elles permettent d'en construire des arbres syntaxiques complets ou partiels, couvrant respectivement une phrase entière ou des segments de celle-ci. Un produit typique contient de plusieurs centaines à plusieurs milliers de règles syntagmatiques.

Cette méthode ne nécessite pas obligatoirement une désambiguïsation syntaxique des mots. Dans certains logiciels, par exemple, la règle SN=article+adjectif+nom indique simplement que les trois mots doivent posséder, entre autres, ces catégories syntaxiques. Ainsi dans le syntagme nominal *la belle ferme*, *la* peut être un article ou un pronom, *belle* peut être un adjectif ou un nom, et *ferme* un nom ou un verbe. La règle ci-dessus est satisfaite car les trois mots possèdent chacun la catégorie syntaxique requise.

Lorsqu'aucune analyse complète n'est possible, c'est-à-dire lorsque les règles syntagmatiques ne permettent pas de regrouper la phrase entière, le logiciel découpe la phrase en segments pour lesquels une analyse partielle a été faite et exécute les algorithmes de détection et de correction d'erreurs pour chaque segment séparément. Le fonctionnement des logiciels se distingue dans ce cas de la méthode par îlots syntaxiques. En effet, il existe ici une notion de hiérarchie entre les éléments de la phrase (représentée par

les arbres syntaxiques), alors que la structure utilisée lors du traitement par îlots est principalement plate.

Pour détecter les erreurs grammaticales, les logiciels recourant à une analyse syntaxique simple reposent principalement sur des contraintes morphologiques accompagnant la plupart des règles syntagmatiques. Par exemple, dans la règle SN=article+adjectif+nom, les contraintes du français imposent un genre et un nombre identiques aux trois composantes. Une erreur est présente dans le texte lorsque l'une des contraintes n'est pas satisfaite. Ce moyen de détection est particulièrement adapté au traitement des erreurs morpho-syntaxiques, comme les erreurs d'accord du participe passé et les erreurs d'accord en genre et en nombre des noms et des adjectifs. Etant donné cette approche, les logiciels ne traitent que peu d'erreurs en relation avec la structure syntaxique des phrases, car leurs mécanismes de détection se basent sur une structure syntaxique correcte. Dans certains cas, des règles d'analyse spéciales détectent des erreurs spécifiques. Ceci permet à ces logiciels de corriger des erreurs comme l'omission de l'accent sur la préposition *à*, la confusion entre *quand* et *quant*, ainsi que la confusion entre participe et infinitif, comme dans **Il a regarder le garçon dessiné un oiseau.*

Ce type de logiciels utilise normalement un algorithme de correction relativement simple: chaque règle d'analyse qui contient des conditions morpho-syntaxiques ou qui recherche une erreur spécifique contient également un message d'erreur. Le mécanisme de correction introduit les parties de la phrase concernées dans le message à des endroits réservés puis affiche le message à l'écran.

d. Analyse syntaxique avancée

Les produits de ce type se basent également sur une analyse complète de la phrase. Les informations supplémentaires auxquelles ils font appel leur permettent toutefois de traiter des erreurs plus complexes. Entre autres, les informations relatives à la structure des compléments de chaque verbe jouent un rôle très important dans ces produits lors de l'analyse déjà. Ceci contraste avec les autres approches, où ces informations ne sont utilisées que lors de la détection. Ces données supplémentaires contribuent à produire des analyses plus poussées (contenant davantage d'informations nécessaires à la détection d'erreurs) et plus complètes (recouvrant un maximum de la phrase), ce qui permet la détection d'une gamme d'erreurs plus étendue. Elles élargissent également le champ d'application du logiciel sur les erreurs morpho-syntaxiques, car l'analyse complexe de la phrase permet d'établir certaines dépendances à distance. Ainsi, certaines erreurs d'accord et de temps des verbes, hors de portée des autres algorithmes, sont ici détectées avec succès. La correction est également améliorée grâce à l'analyse plus poussée. En effet, le haut degré de certitude à propos des données à disposition permet des messages d'erreur plus précis.

Les deux exemples ci-dessous illustrent bien la différence entre les possibilités de détection des produits qui utilisent une analyse syntaxique simple et celles de ceux qui se servent d'une analyse plus complexe:

(1) **Après avoir* ***bouclée*** *sa malle, il partit.*

(2) **Ceux qui ont* ***dansés*** *seront appelés.*

Seul le produit utilisant une analyse syntaxique complexe a détecté avec succès ces deux erreurs d'accord du participe passé.

Ce quatrième type de logiciel utilise en principe le même algorithme de correction que les logiciels qui appliquent une analyse syntaxique simple: les règles d'analyse contiennent d'habitude un message qui est affiché à l'écran en cas d'erreur.

Il est difficile de connaître la gamme complète des erreurs corrigées par ce correcteur avancé. En effet, le seul logiciel (à notre connaissance) qui effectue une analyse complexe ne donne pas une liste des erreurs traitées, mais indique simplement qu'il tente d'appliquer les règles du français telles qu'elles sont décrites dans "Le bon usage" (Grévisse, 1988).

Le mode de fonctionnement par analyse complexe souffre d'inconvénients majeurs par rapport aux trois autres types de produits que nous avons présentés. D'une part, la complexité des algorithmes d'analyse le rend nettement plus lent (un facteur de 10 n'est pas exagéré). D'autre part, les messages d'erreur produits lorsque le logiciel a analysé la phrase de façon incorrecte sont parfois inappropriés. La section 2.3 illustre d'ailleurs ce genre de situations.

Nous avons étudié jusqu'ici les principes de base des correcteurs grammaticaux. Ces principes, y compris les modes de fonctionnement décrits ci-dessus (mise en correspondance, îlots syntaxiques, analyse syntaxique simple et analyse syntaxique avancée) sont utilisés à différents

degrés dans les correcteurs commerciaux, comme dans ceux encore au stade de projet. Dans les deux sections qui suivent, nous examinons de manière plus spécifique le fonctionnement et les caractéristiques (types d'erreurs, langues auxquelles ils s'appliquent) de plusieurs correcteurs monolingues et bilingues. Nous mettons également en évidence leurs qualités et leurs défauts à l'aide d'une évaluation simple.

2.2. Les correcteurs monolingues

Nous avons jugé important d'inclure ici la description d'un nombre de correcteurs de langue première, car les méthodes qu'ils utilisent sont également applicables à la correction en langue seconde. Nous présentons premièrement deux projets non-commerciaux (2.2.1.), qui ont l'avantage d'être transparents quant à leur fonctionnement. Nous analysons ensuite les caractéristiques et les performances de quelques correcteurs commerciaux disponibles aujourd'hui (2.2.2.).

2.2.1. Deux projets non-commerciaux

Dans cette section, nous décrivons deux correcteurs monolingues non-commerciaux: "Critique", qui fut l'un des premiers correcteurs grammaticaux d'une certaine envergure, et le système de Vosse (1992), qui tient compte des problèmes rencontrés lors du traitement de textes réels.

a. Critique

"Critique" (Jensen, Heidorn, Miller et Ravin, 1983), développé par IBM pour son usage interne, fut l'un des premiers logiciels de correction automatique pouvant traiter les textes réels. Il a été conçu pour assister les utilisateurs qui écrivent des lettres ou des rapports en anglais, indépendamment du fait que l'anglais soit leur langue première ou non. Critique détecte et corrige certaines erreurs orthographiques et morpho-syntaxiques, ainsi que celles en relation avec le style. Parmi les erreurs de style traitées par Critique, on notera celles devenues répandues dans les logiciels commerciaux, telles l'utilisation de jargon, les phrases trop longues, trop courtes ou trop complexes, ou encore l'abondance de passifs. La détection d'erreurs morpho-syntaxiques, quant à elle, porte entre autres sur l'accord sujet-verbe, sur l'accord du nom avec ses déterminants, les cas des pronoms et la construction des formes verbales.

Pour son analyse grammaticale, Critique s'appuie sur un dictionnaire de quelques 130'000 mots et 300 règles syntagmatiques écrites en LISP dont voici quelques exemples:

(1) NOUN -> NP (HEAD=NOUN)

Un syntagme nominal (NP) peut être constitué d'un nom (NOUN) seul.

(2) NP (~PRON) NP (~PRON) -> NP

Deux syntagmes nominaux qui ne sont pas des pronoms et qui se suivent peuvent également former un syntagme nominal.

Certaines règles syntagmatiques contiennent également des contraintes morphologiques appliquées lors de l'analyse. Par exemple, la règle de construction d'un syntagme nominal à partir d'un déterminant et d'un nom vérifie que le nombre (singulier, pluriel) des constituants est le même.

L'analyse se fait en recherchant des suites de mots ou de syntagmes correspondant à la partie gauche de la règle. Lorsqu'une suite est identifiée, le membre de droite est ajouté à la liste des éléments reconnus et il peut être utilisé comme membre de gauche d'une autre règle. L'analyseur applique ces règles jusqu'à ce qu'il obtienne une phrase complète. Il arrive bien sûr que des ambiguïtés surviennent lors de l'analyse, et il existe donc plusieurs façons d'appliquer ces règles pour une même phrase. Dans ce genre de situation, Critique ne suit qu'une voie à la fois grâce à des règles spéciales qui éliminent les ambiguïtés. Dans la situation contraire, c'est-à-dire lorsque la phrase n'a pas pu être analysée complètement, Critique applique un algorithme, nommé *parse fitting*, qui tente de construire une phrase complète à partir des éléments produits lors de la première analyse. En cas de succès, la phrase est jugée correcte et le traitement passe à la phrase suivante. Lors d'un ou de plusieurs échecs, Critique relâche certaines contraintes liées aux règles d'analyse et tente une nouvelle fois de construire une phrase complète. Si cette nouvelle tentative réussit, le logiciel constate que la phrase contient une erreur, identifie celle-ci grâce aux contraintes modifiées précédemment, applique les contraintes pour déterminer la correction à effectuer et affiche un message d'erreur lié à la règle d'analyse dont les contraintes ont échoué.

Deux aspects de Critique rendent son utilisation pour la correction en langue seconde difficile. D'une part, les erreurs pour lesquelles il est programmé ne sont pas celles que commettent les utilisateurs dont l'anglais n'est pas la langue première. D'autre part, le fait que seules les contraintes morpho-syntaxiques soient l'objet d'une vérification écarte la possibilité de développer des règles détectant d'autres erreurs relativement fréquentes chez les utilisateurs de langue seconde, comme celles relatives à l'ordre des mots ou des compléments.

Une version de Critique a été commercialisée par IBM en 1989 en tant qu'option du système d'exploitation VM. Nous ne possédons malheureusement pas de renseignements quand au succès de ce logiciel.

b. Correcteur du néerlandais

Ce logiciel (Vosse, 1992) détecte et corrige les erreurs typographiques (comme les fautes de frappe), orthographiques et morpho-syntaxiques en néerlandais. Ces dernières comprennent en particulier les erreurs d'accord, d'homophones (mots ayant une même prononciation mais une orthographe différente) et de construction d'expressions idiomatiques et de noms composés (en néerlandais, certains noms composés s'écrivent en un seul mot et d'autres sont séparés par des espaces). Contrairement à pratiquement tous les logiciels décrits dans ce chapitre, son vocabulaire important (quelques 90'000 mots), le nombre de règles d'analyse qu'il contient (près de 500) et les algorithmes qu'il utilise le rendent particulièrement adapté au traitement des

textes réels. Une version de ce correcteur a d'ailleurs été commercialisée à la fin de 1992.

Ecrit dans le langage 'C', le logiciel fait appel à une méthode d'analyse semblable à celle utilisée par Critique (voir section 2.2.1.a.) et Scripsi (voir section 2.3.1.d): un ensemble de règles syntagmatiques est appliqué au texte de manière ascendante (de bas en haut). Les erreurs sont détectées à l'aide de deux mécanismes:

a. Un correcteur orthographique identifie les alternatives aux mots inconnus et les remplace dans le texte. L'analyseur traite ensuite cette situation de la même manière que lorsqu'un mot possède plus d'une catégorie syntaxique: avec un branchement, toutes les voies sont suivies en parallèle.

b. Un algorithme basé sur l'unification applique des contraintes sur les traits morphologiques. Vosse note ici une différence entre la méthode qu'il utilise et celle de Schwind dans son correcteur de l'allemand langue seconde, anglais langue première (voir section 2.3.1). Alors que Schwind utilise des règles d'erreurs pour identifier les cas où l'unification échoue, l'algorithme de Vosse laisse simplement passer ces cas, tout en prenant note du fait qu'une erreur d'unification s'est présentée à l'endroit en question. Vosse remarque à juste titre que son algorithme se prête davantage au traitement de textes réels. La mémoire requise lors de l'analyse ayant tendance à croître de manière exponentielle avec le nombre de règles, leur réduction au strict nécessaire peut faire la différence entre un analyseur rapide et un analyseur lent.

Vosse propose une méthode intéressante pour déterminer la solution à choisir lorsque l'analyse syntaxique identifie plus d'une structure. Au lieu de choisir celle qui contient le moins d'erreurs, le logiciel additionne les poids attribués par l'auteur aux différentes erreurs et ne garde que la solution avec le poids total le plus faible. La valeur du poids ne dépend que de la fréquence estimée de l'erreur en question.

Pour corriger les erreurs typographiques et orthographiques, le logiciel de Vosse détermine tout d'abord la liste des candidats pour remplacer le mot qui contient une erreur. Le logiciel effectue ensuite une série d'analyses en essayant chaque mot de la liste à la place du mot erroné. La meilleure analyse détermine la correction à suggérer à l'utilisateur. Pour corriger les erreurs morphologiques, le logiciel utilise une méthode différente suivant que l'erreur est détectée lors de l'unification ou par l'intermédiaire d'une règle d'erreur (une règle syntagmatique, identique aux règles d'analyse, mais qui recherche une erreur spécifique). Dans le premier cas, les données produites lors de l'unification permettent de déterminer la forme correcte du mot contenant l'erreur. Dans le second cas, un message associé à la règle d'erreur est affiché à l'écran.

Bien que le logiciel semble bien adapté au traitement de textes réels, il n'a pas été conçu pour les textes écrits dans une langue seconde. Comme le note Vosse, les erreurs contenues dans les textes en néérlandais langue première sont plus orthographiques et morpho-syntaxiques que syntaxiques. Les méthodes de détection qu'il a développées reflètent cette situation et se prêtent peu à la détection d'erreurs syntaxiques rencontrées fréquemment dans les textes écrits dans une langue seconde.

2.2.2. Produits commercialisés

Comme nous l'avons déjà constaté, le fonctionnement interne des logiciels commercialisés ne se trouve pas dans le domaine public. Nous pouvons par contre nous baser sur des évaluations effectuées sur ces produits pour déterminer ce dont ils sont capables. Deux paramètres nous intéressent plus particulièrement:

1. Le taux de détection: le rapport entre le nombre d'erreurs détectées correctement et le nombre d'erreurs présentes.
2. Le taux de fausse détection: le rapport entre le nombre de fois où le logiciel a détecté une erreur alors qu'il n'y en avait pas et le nombre de fois où le logiciel a détecté une erreur (qu'elle soit réelle ou non).

Par exemple, si, dans un texte qui contient 50 erreurs, le logiciel en a détecté 40 correctement et a fait 20 fausses détections, le taux de détection est de 80% (40/50). Le logiciel ayant identifié au total 60 erreurs (40 réelles et 20 fausses), le taux de fausse détection est de 33% (20/60). Ainsi, un taux de fausse détection de 33% indique qu'un message sur trois présentés à l'utilisateur par le logiciel n'est pas pertinent.

Examinons les résultats de quelques évaluations de correcteurs monolingues effectuées ces dernières années. Tous les correcteurs (français et anglais) soumis aux évaluations prétendent être capables de corriger les fautes de grammaire comme celles que nous avons décrites

dans la section 2.1.2. (règles d'accord, utilisation de certaines prépositions, élision, etc.).

a. Dinnematin et Sanz (1990) présentent une des premières évaluations faite sur des correcteurs grammaticaux du français, dans laquelle ils soumettent 100 phrases de huit à dix mots contenant toutes une faute grammaticale, à deux logiciels: Hugo Plus obtient un taux de détection de 15% et GramR obtient un taux de 4%. Les taux de fausse détection ne figurent pas dans cette évaluation.

b. Rabinovitz (1991) utilise 150 phrases contenant des erreurs de ponctuation, de majuscules manquantes, de grammaire et de style pour évaluer six correcteurs de l'anglais. Les résultats sont les suivants:

	Taux de détection [%]	Taux de fausse détection [%]
PowerEdit	51	17
Correct Grammar	43	11
Grammatik IV	30	10
Electric Webster	29	37
Editor	27	23
RightWriter	13	38

Rabinovitz constate également que PowerEdit et CorrectGrammar produisent les messages d'erreur les plus adéquats.

c. Sanz (1992) évalue quatre correcteurs grammaticaux du français. En plus des phrases-test, il soumet à chaque correcteur une lettre-type contenant onze erreurs parmi les plus répandues, ainsi qu'une dictée de Pivot dans laquelle dix erreurs ont été insérées. Sanz ne décrit que les taux de détection, qui sont les suivants:

	Taux de détection		
	Phrases [%]	Lettre-type [%]	Dictée [%]
Sans-Faute	81*	0	40
Grammatik	53	18	50
Hugo Plus	26	0	20
GramR	29	0	30

* Des doutes existent quant à la validité de ce résultat. Les auteurs du logiciel l'ont semble-t-il programmé spécialement pour détecter les erreurs contenues dans le corpus de test.

d. Laenzlinger (1992) reprend les logiciels évalués par Sanz (1992) et leur soumet 190 phrases, dont 143 contiennent une erreur de syntaxe et 47 sont tout à fait grammaticales. Ces phrases ne dépassent pas six ou sept mots, mais elles contiennent des structures relativement complexes. Les résultats sont les suivants:

	Taux de détection [%]	Taux de fausse détection [%]
Sans-Faute	34	6
Grammatik	48	8
Hugo Plus	18	33
GramR	13	17

Nous constatons que les taux de détection peuvent varier entre 0 et 53% suivant les produits et les types de tests effectués. La différence est particulièrement nette entre les taux obtenus sur des phrases-test, dans lesquelles des erreurs ont été introduites, et les taux obtenus sur le seul exemple de texte continu, c'est-à-dire les lettres-types présentées dans (c). Il est en effet inquiétant de constater que, dans un cas représentatif de la vie de tous les jours, un produit a détecté deux erreurs sur onze alors que les trois autres n'en ont détecté aucune. De plus, les taux de fausse détection, en moyenne de l'ordre de 20%, nous montrent que l'utilisateur ne peut pas avoir pleinement confiance dans le logiciel. Il s'ensuit que chaque message d'erreur affiché à l'écran doit être évalué afin de déterminer si le texte contient vraiment une erreur.

Même s'ils sont révélateurs, les taux de détection et de fausse détection ne représentent pour les produits commerciaux que deux facteurs de comparaison parmi d'autres. Par exemple, un taux de fausse détection élevé et une vitesse d'exécution lente peuvent rendre un correcteur inutilisable. De plus, la précision des messages à l'attention de

l'utilisateur, la convivialité, la plate-forme matérielle sur laquelle fonctionne le logiciel et le prix de vente sont autant de paramètres dont les producteurs de logiciel doivent tenir compte lors du choix des méthodes de détection. Des études approfondies portant sur l'ensemble de ces facteurs n'ayant pas encore été effectuées, nous pouvons néanmoins citer les commentaires exprimés par certains auteurs d'évaluations qui, plus que les chiffres mêmes, nous donnent des indications importantes sur la qualité des correcteurs grammaticaux commerciaux en général:

- Dobrin (1990):

> "Questions remain about where and how [Correct Grammar] can be useful. It is not terribly useful on final versions of good prose; the makers assert that it is more useful on edited prose, where there are many inadvertent errors."

- Rabinovitz (1991):

> "The sad truth is that [the grammar checkers] aren't very effective tools. ... Grammar checkers may be most valuable as tools to create style rules."

- Eglowstein (1991)

> "The grammar checkers are handy for helping you locate errors, but you still have to fix them yourself. Just as a paint program won't

make you an artist, a grammar and style checker won't make you a professional writer."

- John Chandioux, auteur de GramR, dans une conférence donnée le 8 novembre 1994 au SCIB '94 (Salon Canadien de l'Informatique/ Bureautique), est encore plus pessimiste en s'exclamant:

"La détection grammaticale ne sert à rien !"

Malgré ces remarques, nous constatons que des progrès importants ont été effectués ces dernières années dans la correction monolingue. Laenzlinger remarque ainsi que les taux de détection se sont bien améliorés entre une première évaluation (Dinnematin et Sanz, 1990) et une seconde (Sanz, 1992), qui portent sur des versions plus récentes des mêmes produits. De plus, les recherches comme celles de Vosse nous montrent qu'une analyse basée sur un vrai formalisme grammatical permet d'obtenir de bons résultats lorsqu'elle est appliquée à la détection et la correction d'erreurs morpho-syntaxiques. Les logiciels tels que celui de Vosse ont bien entendu une génération d'avance sur la grande majorité des produits commerciaux actuels, et le fait qu'ils sont développés en tenant compte des contraintes du monde réel (matériel, temps d'exécution) va certainement dans la bonne direction.

2.2.3. Evaluation d'un correcteur monolingue sur des phrases L2

Les évaluations de correcteurs grammaticaux que nous avons décrites jusqu'ici ont toutes eu lieu dans des situations monolingues: les phrases soumises aux logiciels contiennent des erreurs produites fréquemment par des utilisateurs dans leur langue première. Vu que ces produits ont pour fonction de détecter et corriger les erreurs grammaticales en général et que, à première vue, une erreur est une erreur, on peut se demander si leur performance varie face à des textes contenant des erreurs produites en langue seconde. Nous avons ainsi choisi un correcteur monolingue du français, le Correcteur 101 de Machina Sapiens, et lui avons soumis une série de phrases-test contenant des erreurs produites fréquemment en français langue seconde par des natifs de l'anglais. Ce logiciel est l'un des plus avancés à l'heure actuelle. A notre connaissance, il est le seul qui fasse appel à une analyse complexe du texte telle que nous l'avons décrite dans la section 2.1.2.d.

Les phrases-test sont celles que nous avons développées pour l'évaluation du logiciel LFG décrit dans les chapitres 3 et 4. Elles contiennent des erreurs qui appartiennent aux onze catégories suivantes:

(1) Accord sujet-verbe

Exemple: * *Le cri des minarets **faisaient** peur aux enfants.*

Correction: *faisait*.

(2) Accord du syntagme nominal

Exemple: * *Ces **quelque** livres étaient horriblement chers.*

Correction: *quelques*.

(3) Participe passé: Confusion entre infinitif et participe passé

Exemple: * *Nous avons **veiller** toute la nuit.*

Correction: *veillé*.

(4) Participe passé: Accord avec 'être'

Exemple: * *Ils sont **sorti** ensemble pendant des années.*

Correction: *sortis*.

(5) Participe passé: Accord avec 'avoir'

Exemple: * *Les services que tu lui as **rendu** étaient les bienvenus.*

Correction: *rendus*.

(6) Mode dans les propositions

Exemple: * *Il faudra que j'**écris** pour la convaincre.*

Correction: *écrive*.

(7) Structure des compléments du verbe

Exemple: * *Il donne la lettre **sa soeur**.*

Correction: *à sa soeur*.

(8) Position des adjectifs

Exemple: * *Il a cassé la **rouge tasse**.*

Correction: *la tasse rouge*.

(9) Les prépositions 'à' et 'de': Préposition + infinitif

Exemple: * *Ils ont réussi **de** sauver le chien.*

Correction: *à*.

(10) Les prépositions 'à' et 'de': Confusion entre 'a' et 'à'

Exemple: * *Il ne demandait qu'**a** partir.*

Correction: *à*.

(11) Les prépositions 'à' et 'de': Préposition + 'le', 'la', 'les'

Exemple: * *L'heure d'arrivée **du les** trains est toujours précise.*

Correction: *des*.

(Les erreurs des catégories 1, 2, 3, 4, 5 et 10 sont également produites par des natifs du français.)

Cbaque catégorie est représentée par 15 phrases, 10 qui contiennent une erreur et 5 qui n'en contiennent pas. Ces dernières reflètent toutefois les difficultés de la catégorie à laquelle elles appartiennent. Une description plus complète de chaque catégorie ainsi que les instructions suivies par l'auteur des phrases se trouvent dans l'annexe 3.

Les résultats de l'évaluation, présentés dans le tableau 2-1, montrent que le correcteur monolingue a obtenu un score presque parfait pour les 5 catégories relatives aux erreurs d'accord (1 à 5) avec un taux de détection moyen de 98%. La seule erreur non détectée se trouvait dans **Il y a un chat sur **le petit** terrasse* (correction: *la petite terrasse*). Le correcteur a également bien traité la catégorie (10), 'Confusion entre 'a' et 'à'', avec un score de 70%, et la catégorie (6), 'Mode dans les propositions', avec un

score de 80%. Ces très bons résultats s'expliquent en grande partie par le fait que les francophones commettent assez souvent des erreurs dans ces 7 catégories, erreurs que le logiciel s'attend donc à rencontrer. L'analyse syntaxique effectuée sur la phrase entière par ce correcteur n'est pas non plus étrangère à la qualité des résultats obtenus ici.

Catégorie	Erreurs détectées correctement [%]	* Erreurs détectées incorrectement [%]	Erreurs non-détectées [%]	Nombre de non-erreurs détectées
1. Accord S-V	100	0	0	0
2. Accord SN	90	0	10	0
3. Infinitif-PP	100	0	0	0
4. PP-être	100	0	0	0
5. PP-avoir	100	0	0	0
6. Mode	80	0	20	0
7. Structure	0	20	80	1
8. Adjectifs	0	10	90	0
9. Prép.+inf.	0	30	70	0
10. Conf. à-a	70	0	30	1
11. Prép.+art.	30	50	20	1

* Nous avons compté une détection incorrecte (deuxième colonne) lorsque le logiciel avait détecté une erreur mais avait proposé la mauvaise correction.

Tableau 2-1 - Résultats de l'évaluation du logiciel monolingue "Correcteur 101"

En effet, 6 des 7 catégories en question représentent des erreurs où seule la morphologie d'un mot doit être corrigée. L'analyse syntaxique peut donc s'opérer normalement et le logiciel ne rencontre aucun problème pour trouver les composantes qui entrent en jeu lors de la vérification des règles morpho-syntaxiques (catégories 1 à 6), comme la nature de l'auxiliaire, la position et les traits du sujet et du complément d'objet direct, et la position des verbes principaux de chaque phrase. Pour les confusions entre *a* et *à*, le logiciel possède vraisemblablement une règle d'analyse spécifique qui transforme une forme en l'autre lorsque l'analyse syntaxique produit des résultats anormaux. Ici aussi, c'est une analyse complète qui permet de traiter l'erreur avec succès.

Malheureusement, le correcteur a obtenu des résultats médiocres pour les 4 autres catégories. Seules 3 erreurs, appartenant toutes à la catégorie (11) 'Préposition + 'le', 'la', 'les' ', ont été détectées correctement. Le correcteur n'a détecté aucune erreur correctement dans les phrases des catégories (7) 'Structure des compléments du verbe', (8) 'Position des adjectifs', et (9) 'Préposition + infinitif'. Les 4 catégories en question possèdent deux points communs: d'une part, leurs erreurs se rencontrent relativement peu fréquemment, sinon jamais, chez les francophones, et d'autre part elles relèvent plutôt de la structure de la phrase que de la morphologie des mots. Voici quelques exemples de phrases dans lesquelles le correcteur a détecté une erreur sans l'identifier correctement (notons que la solution proposée est parfois pire que l'erreur d'origine):

Catégorie 7, structure des compléments du verbe

(1) * *Il **donne la lettre sa soeur**.*

Correction proposée: **Il donne là lettrée sa soeur*

(2) * *Il **rend visite sa mère**.*

Correction proposée: **Il rend visitée sa mère*

Catégorie 9, les prépositions 'à' et 'de': Confusion entre 'à' et 'de'

(3) * *Ils ont oublié **à** appeler.*

Correction proposée: **Ils ont oublié à appelé.*

(4) * *Elle m'a chargé **à** vous avertir.*

Correction proposée: **Elle m'a charge à vous avertir.*

Catégorie 11, les prépositions 'à' et 'de': préposition+'le','la','les'

(5) * *L'heure d'arrivée **du les trains** est toujours précise.*

Correction proposée: **L'heure d'arrivée due les trains sont toujours précis.*

Des résultats de cette évaluation, nous pouvons tirer les conclusions suivantes:

1. Le fait de tenter une analyse complète de la phrase ne suffit pas à détecter toutes les erreurs. Les auteurs du correcteur affirment que celui-ci vérifie les règles de grammaire contenues dans "Le Bon Usage"

(Grévisse 1988). Ceci ne suffit visiblement pas pour les erreurs produites en français langue seconde.

2. Les effets secondaires d'une analyse complète défaillante peuvent être désastreux.
3. Nous pouvons ainsi émettre de sérieux doutes quant à l'utilisation de correcteurs monolingues dans des situations bilingues.

2.3. Les correcteurs bilingues

Les correcteurs bilingues se distinguent des correcteurs monolingues par le fait qu'ils détectent et corrigent les erreurs commises spécifiquement par des utilisateurs qui écrivent dans une langue seconde. Dans cette section, nous décrivons les caractéristiques de cinq correcteurs bilingues non-commerciaux à l'état de prototype. Nous présentons également trois produits commerciaux et en soumettons un à une évaluation.

2.3.1. Cinq projets non-commerciaux

Un certain nombre de systèmes non-commerciaux de correction automatique en langue seconde ont vu le jour ces dernières années. Ils font partie pour la plupart de projets dont le but est l'enseignement des langues assisté par ordinateur (ELAO). Leur objectif consiste donc non seulement à détecter et à corriger les erreurs, mais aussi à expliquer les raisons pour lesquelles ces erreurs ont été identifiées comme telles. Les sections qui suivent décrivent les caractéristiques et le fonctionnement de quelques-uns de ces logiciels.

a. Automated German Tutor

Ce logiciel (Weischedel, Voge et James, 1978), développé aux universités de Delaware et de Californie à Irvine, a été conçu pour vérifier le contenu de textes produits par des étudiants anglophones apprenant l'allemand. Ces textes consistent en des phrases rédigées par l'utilisateur en réponse à des questions pré-définies posées par le logiciel. Outre les fautes d'orthographe, le système détecte les erreurs syntaxiques et morpho-syntaxiques de deux types: les interférences de l'anglais relatives à l'ordre des mots et les fautes d'accord. Le logiciel contient également un module dont la tâche consiste à déterminer si la réponse de l'utilisateur est en rapport avec la question posée et si cette réponse est correcte.

Le système de Weischedel et al. est programmé en LISP et utilise des réseaux de transition augmentés (*Augmented Transition Networks*, ou ATN; voir Woods 1970) lors de l'analyse syntaxique du texte. Le module de traitement sémantique utilise le langage Micro-Planner. Le vocabulaire reconnu par le logiciel se limite à quelque 200 mots.

L'algorithme d'analyse syntaxique (ATN) fonctionne d'une manière classique. Le sous-ensemble de la grammaire de l'anglais reconnu par le logiciel est représenté par des noeuds reliés entre eux par un réseau d'arcs. Ce réseau contient un noeud initial et un noeud terminal, qui correspondent au début et à la fin d'une phrase. L'objectif de l'analyse consiste à trouver un chemin allant du noeud initial au noeud terminal, tout en faisant correspondre les mots de la phrase aux arcs du réseau. A chaque arc sont attachées des conditions qui, lorsqu'elles sont remplies, permettent la traversée de l'arc. Ces

conditions s'appliquent notamment à la catégorie syntaxique des mots et à leurs traits morphologiques.

Le logiciel détecte les erreurs syntaxiques grâce à des arcs supplémentaires qui correspondent à des structures de phrase incorrectes. Ainsi, il existe une suite d'arcs qui accepte la phrase correcte "*Ich habe das Fleisch gegessen*" et une autre qui accepte "**Ich habe gegessen das Fleisch*", structure incorrecte fréquemment utilisée par des apprenants anglophones de l'allemand. Un message d'erreur adéquat est lié à cette seconde série d'arcs, ce qui permet au logiciel d'informer l'utilisateur. Le logiciel détecte les erreurs morpho-syntaxiques grâce aux conditions morphologiques liées aux arcs. Lorsque celles-ci ne correspondent pas aux traits morphologiques du mot courant, le logiciel en prend note et continue la traversée du réseau. Comme dans le cas des erreurs syntaxiques, un message d'erreur lié aux conditions de l'arc est affiché s'il le faut. Il arrive bien entendu qu'il existe plusieurs chemins menant du noeud initial au noeud terminal pour une même phrase. Dans ce cas, le logiciel ne garde que celui qui contient le moins d'erreurs et affiche les messages appropriés.

Le logiciel n'utilise aucune information syntaxique pour détecter les erreurs sémantiques. Il s'appuie, par contre, sur une série de règles d'inférence appliquées à la réponse de l'utilisateur. Les informations obtenues par l'application de ces règles sont comparées à un ensemble minimum de faits attendus pour la question correspondante. Le logiciel affiche un message d'erreur lorsque la comparaison échoue.

Bien que tenant compte des erreurs en allemand langue seconde, le logiciel possède certaines lacunes qui le rendent difficilement adaptable aux textes réels et aux erreurs qu'ils contiennent. En particulier, le fait que l'algorithme oblige un parcours complet à travers le réseau d'arcs afin de détecter des erreurs l'empêche de pouvoir traiter les constructions syntaxiques inattendues.

b. French Grammar Analyser

French Grammar Analyser (FGA) (Barchan, Woodmansee et Yazdani, 1986) détecte et corrige quatre types d'erreurs produites en français par des natifs de l'anglais:

- Erreurs lexicales sans référence au contexte. L'exemple qui est donné est l'utilisation erronée de *y* dans **Le homme le y donne.*
- Erreurs d'accord entre le sujet et le verbe.
- Structure des phrases. Le seul cas mentionné est celui de la mauvaise utilisation des pronoms en relation avec les adverbes de négation, comme dans **Il n'y a pas quelqu'un.* (correct: *Il n'y a personne*).
- Fautes d'orthographe (**garcon* au lieu de *garçon*).

FGA est un logiciel écrit en Prolog et dérivé d'une première version, "French Robust Grammar checker", ou FROG (Imlah et du Boulay, 1985). Les règles utilisées lors de l'analyse syntaxique sont similaires à celles employées par Critique. La différence entre FGA et Critique réside dans l'application de ces règles. Alors que Critique analyse la phrase de bas en haut, c'est-à-dire en remontant à partir des mots jusqu'à un élément qui recouvre toute la phrase,

FGA applique l'algorithme inverse: il part de la règle de plus haut niveau et la décompose en ses éléments notés à droite de la flèche (voir les exemples dans la section 2.2.1.a. qui décrit Critique), jusqu'à ce que tous les mots soient couverts. Barchan et al. décrivent les avantages et les inconvénients de cette méthode. Parmi les premiers, on peut prévoir les erreurs en introduisant des règles d'erreurs comparables aux arcs du logiciel de Weischedel et al. (voir 2.3.1.a). Lorsque l'analyseur rencontre une règle qui correspond à une erreur, il en informe immédiatement l'utilisateur en affichant un message associé à la règle. Pour corriger les erreurs morphologiques, FGA utilise l'algorithme standard qui consiste à obtenir les traits de l'élément de référence (comme le genre et le nombre du sujet dans le cas des erreurs d'accord sujet-verbe), à les appliquer au mot erroné (le verbe dans ce cas) et à présenter le résultat à l'utilisateur. Pour corriger les autres erreurs, FGA affiche le message associé à la règle d'erreur qui a été déclenchée après y avoir inséré des mots de la partie du texte qui contient l'erreur.

Le fait d'informer l'utilisateur dès qu'une erreur est détectée permet au logiciel de traiter les phrases qui possèdent une structure inconnue. Par exemple, le logiciel ne contient pas toutes les règles requises pour analyser correctement la phrase "**Le grande homme volera les printemps le samedi.*". L'erreur d'accord en début de phrase peut tout de même être détectée grâce à ce mécanisme. Parmi les inconvénients figure le fait qu'aucune erreur ne peut être détectée dans la partie de la phrase où l'analyse a échoué. De plus, la grammaire et le vocabulaire limités de FGA rendent difficile l'évaluation de la qualité des mécanismes développés.

c. Intelligent Language Tutoring System

Comme le système de Weischedel et al., Intelligent Language Tutoring System (ILTS) (Schwind, 1988 et 1995) corrige les erreurs en allemand langue seconde. Aucune précision n'est donnée au sujet de la langue première ciblée. Les erreurs suivantes sont prises en compte:

- Erreurs morphologiques, comme la mauvaise application de traits tels que le nombre, le genre et le cas aux noms et aux adjectifs.
- Erreurs d'accord à l'intérieur du syntagme nominal et entre le sujet et le verbe.
- Erreurs syntaxiques de deux types: mots manquants ou superflus (comme parfois les prépositions), et permutation de mots ou de syntagmes (comme dans **Er zurückkommt* au lieu de *Er kommt zurück*).
- Erreurs sémantiques, comme dans **Das Heft arbeitet*, provenant de confusions au sujet de la signification des mots.

Développé en Prolog, ILTS utilise une grammaire de métamorphose lors de l'analyse syntaxique (Schwind, 1988). Le format des règles appartenant à cette grammaire diffère quelque peu de celui utilisé par Critique et par FGA, mais le fonctionnement de la grammaire reste en gros le même, avec des règles de réécriture. Comme pour FGA et le logiciel de Weischedel et al., ILTS contient un ensemble de règles ajoutées à la grammaire de l'allemand qui représentent une suite erronée de mots ou d'éléments de la phrase.

Les erreurs d'accord sont traitées à l'aide d'une variante de la méthode d'unification (Karttunen, 1984). Lors de l'unification, chaque membre d'un syntagme nominal ou verbal ajoute une liste de traits à l'ensemble qui correspond au syntagme nominal ou verbal. Les traits sont constitués d'un attribut (nombre, personne, etc.) et d'une valeur (singulier, pluriel, troisième, etc.). Par exemple, l'article allemand *der* possède trois groupes de traits, un pour chaque interprétation:

A1. Art-cat=défini, cas=génitif, nombre=pluriel
A2. Art-cat=défini, genre=féminin, cas=génitif ou datif, nombre=singulier
A3. Art-cat=défini, genre=masculin, cas=nominatif, nombre=singulier

Le nom *Lehrer* (*maître*) possède deux groupes de traits:

B1. Genre=masculin, cas=non génitif, nombre=singulier
B2. Genre=masculin, cas=non datif, nombre=pluriel

L'unification consiste à réunir ces traits, groupe par groupe. Dans l'exemple ci-dessus, seules l'unification A3+B1 et l'unification A1+B2 réussissent, car aucune valeur n'est contradictoire. ILTS détecte une erreur dans une phrase ou dans une sous-partie de phrase lorsque l'unification échoue. L'identification de l'erreur, qui produit les informations nécessaires à la correction, a lieu en analysant les raisons de l'échec de l'unification.

d. Scripsi

Scripsi (Catt, 1988) s'applique à des textes anglais écrits par des natifs du français. C'est le seul système qui, à notre connaissance, traite des surgénéralisations morphologiques, lexicales et syntaxiques en plus des interférences. Les catégories d'erreurs traitées sont les suivantes:

Erreur morphologique

- Surgénéralisation de la règle de construction du prétérit à des verbes irréguliers (**writed* au lieu de *wrote*).

Erreurs syntaxiques

- Règles structurales incorrectes, comme dans **She wonders does he sleep* au lieu de *She wonders if he sleeps.*
- Sous-catégorisation erronée des verbes. Exemple: **He disobeys to his father*. Correct: *He disobeys his father.*

D'après Catt, un des principaux avantages de Scripsi réside dans la séparation entre les règles grammaticales de l'anglais, utilisées lors de l'analyse syntaxique, et les règles appliquées par le logiciel pour détecter les erreurs.

Scripsi, écrit en Prolog, ne contient qu'un vocabulaire réduit et relativement peu de règles d'analyse et de détection. Par contre, les algorithmes que le logiciel utilise démontrent un réel souhait, de la part de son auteur, de tenir compte des théories développées dans le cadre des recherches en linguistique appliquée. Ainsi, existe-t-il deux ensembles de règles: celles qui représentent

la grammaire de l'anglais et celles qui reflètent les problèmes que rencontrent, selon les théories linguistiques, les apprenants francophones de l'anglais.

La détection d'erreur dans Scripsi s'opère en deux phases: l'analyse lexicale et l'analyse syntaxique. Le but de la première phase consiste à assigner à chaque mot du texte d'entrée un ensemble de traits morphologiques, syntaxiques et sémantiques. Lorsque le logiciel rencontre un mot inconnu, il applique des règles de surgénéralisation morphologique pour tenter de reconstruire le mot original et la transformation morphologique voulue par l'utilisateur. Lors de la seconde phase de détection, le logiciel fait appel aux deux ensembles de règles syntaxiques que nous avons présentées plus haut. Les règles lexicales, qui font partie de l'ensemble de la langue première (le français) contiennent par exemple le fait que le verbe *désobéir* est suivi de la préposition *à* puis d'un syntagme nominal (*désobéir à quelqu'un*). L'équivalent de cette règle dans l'ensemble langue seconde (l'anglais) indique que *disobey* est suivi simplement d'un syntagme nominal (*to disobey someone*). Les règles syntagmatiques, quant à elles, indiquent l'ordre des mots dans les phrases. L'ensemble de l'anglais contiendra, par exemple, une règle signalant qu'un adjectif se place d'habitude avant le nom, alors que l'équivalent dans l'ensemble du français précise qu'un adjectif se place après le nom.

Lors de l'analyse du texte, les règles de l'ensemble de l'anglais sont activées en premier. Lors d'une impasse, le système revient en arrière et applique les règles de l'ensemble du français. Une fois que l'analyseur est parvenu à la fin de la phrase, Scripsi choisit la solution comportant le moins d'erreurs, c'est-à-dire celle où le minimum de règles de l'ensemble du français sont utilisées.

Scripsi présente ensuite à l'utilisateur les messages d'erreur associés à ces règles. Scripsi utilise le même algorithme que Critique si aucune analyse n'est possible: une nouvelle analyse est effectuée sans tenir compte de certaines contraintes morphologiques (accord du nom et du verbe, par exemple).

Le problème principal de Scripsi réside dans le fait que le succès de la détection des erreurs dépend du succès de l'analyse syntaxique, qui elle-même dépend d'un dictionnaire d'analyse extrêmement complet. Ainsi, les informations liées à chaque verbe doivent contenir toutes ses possibilités de sous-catégorisation. Quand celles-ci manquent, l'analyse se termine sans aucune action de la part du logiciel.

L'importance que Scripsi attribue aux règles syntaxiques en général s'avère problématique dans d'autres situations. Par exemple, dans la phrase ci-dessous,

> * *She **died** the shirt.* (*Elle a teint la chemise.*)
> (correct: *She dyed the shirt.*)

Scripsi ne détecte pas l'orthographe incorrecte de *died*, qui s'écrit ici *dyed*. Scripsi identifie par contre une erreur de sous-catégorisation du verbe *to die*, qui se construit sans complément d'objet direct, et affiche le message d'erreur correspondant.

e. Francophone Stylistic Grammar Checker

Le Francophone Stylistic Grammar Checker (FSGC) (Brehony et Ryan, 1994) détecte les erreurs dans les textes français écrits par des anglophones. Le logiciel est écrit dans le langage 'C' et fonctionne sous Microsoft Windows. Brehony et Ryan ne mentionnent pas les catégories d'erreurs détectées et corrigées par leur logiciel qui repose entièrement sur un dictionnaire d'analyse basé sur les grammaires de liage (Sleator et Temperley, 1991). A chaque mot du dictionnaire correspond un ensemble complet de contraintes de liage. L'analyse d'une phrase s'effectue en faisant correspondre les contraintes des mots adjacents tout en appliquant les règles de bonne formation propres à ce type de grammaire.

Pour détecter les erreurs morpho-syntaxiques et syntaxiques, les auteurs ont ajouté aux mots des contraintes de liage qui représentent les erreurs pertinentes à ces mots. Une fois l'analyse terminée, le logiciel signale une erreur dans le texte à l'endroit où une telle contrainte de liage a été utilisée. Le logiciel ne considère que la première solution obtenue lors de l'analyse. Etant donné que celle-ci n'est pas toujours la meilleure, le logiciel détecte des erreurs également lorsque le texte n'en contient pas. Ceci représente un inconvénient majeur pour un logiciel destiné à aider les utilisateurs lors de la rédaction de textes en langue seconde, d'autant plus que la nature du formalisme utilisé semble faire obstacle à une amélioration de ce comportement. Malgré ces défauts, les auteurs de FSGC comparent favorablement les caractéristiques du logiciel (temps d'exécution, mémoire requise) à celles des correcteurs décrits plus haut.

Pour terminer la présentation de ces cinq projets non commerciaux, et sans entrer dans les détails architecturaux de chacun d'eux, nous pouvons identifier quelques caractéristiques importantes qui leur sont communes:

1. Chacun de ces systèmes utilise une méthode d'analyse grammaticale qui rend nécessaire la construction des structures syntaxiques pour la phrase entière. Si l'analyse échoue, aucune détection globale n'est possible (sauf pour les fautes d'orthographe dans certains cas).

2. Les erreurs sont dans chaque cas prévues à l'avance. Les règles qui permettent leur détection font partie de l'ensemble des règles utilisées lors de l'analyse grammaticale, au même titre que les règles d'analyse habituelles. Ainsi, dans tous les systèmes décrits ici, une fois l'analyse terminée, le logiciel vérifie si des règles qui correspondent à des erreurs ont été utilisées. Dans l'affirmative, des règles de correction spécifiques sont appliquées. Le système Scripsi ne constitue pas une exception: si les règles d'erreurs sont détachées des règles normales du point de vue de l'organisation des bases de données du logiciel, elles n'en sont pas moins appliquées en même temps.

3. Tous les systèmes sauf ILTS possèdent des informations concernant la langue première des personnes qui l'utilisent et des erreurs en langue seconde qu'elles commettent. Les interférences constituent la majorité des erreurs traitées.

4. Aucune recherche n'a été effectuée à notre connaissance pour déterminer la fréquence relative des erreurs corrigées. Il est donc possible que les systèmes ci-dessus ne corrigent pas les erreurs que les utilisateurs commettent vraiment,

ou qu'ils appliquent les règles de détection dans un ordre inadéquat (voir l'exemple présenté dans la section qui décrit Scripsi).

5. Aucune évaluation formelle de la qualité des systèmes n'est présentée. Ce type d'information est crucial pour déterminer l'utilité véritable d'un logiciel de correction automatique, comme nous le verrons dans les prochains chapitres.

2.3.2. Produits commercialisés

Il n'existe à ce jour que très peu de correcteurs grammaticaux développés spécifiquement pour des textes écrits dans une langue seconde. Nous en citerons trois: Bilingual PC Proof version 3.0 et son successeur WinProof, de Lexpertise Linguistic Software, et Grammatik anglais pour francophones, de Reference Software. Parmi ces trois logiciels, qui corrigent l'anglais de francophones, deux sont construits autour d'un correcteur monolingue ayant fait ses preuves sur le marché (WinProof est basé sur CorrectGrammar et Grammatik anglais pour francophones sur Grammatik anglais), complété par des régles destinées à la détection et à la correction d'erreurs fréquentes commises par les utilisateurs ciblés. Tschichold (1991) pour Bilingual PC Proof et le manuel de l'utilisateur pour Grammatik nous donnent quelques renseignements au sujet des erreurs traitées par ces correcteurs:

- Faux amis. Par exemple, en anglais, *library* ne signifie pas *librairie* mais *bibliothèque*.
- Certains mots avec plusieurs traductions en anglais peuvent prêter à confusion. (*faire* se traduit en anglais par *to do* et *to make*)

- Prépositions aprés certains verbes et adjectifs. Par exemple, **To marry with someone* au lieu de *To marry someone.*
- Utilisation du pluriel pour des noms qui ne s'utilisent qu'au singulier, comme dans **a lot of informations* (Correct: *information*).
- Traductions littérales (*Tout compris* ne devrait pas être traduit par *everything comprised* mais par *all inclusive*)
- Position des adverbes, comme dans **I eat always too much.* (correct: *I always eat too much.*)

Les évaluations faites sur certains correcteurs grammaticaux en langue seconde (Liechti, 1991; Tschichold, 1991; Mandeville 1992; Tschumi et al., 1996) révèlent que ces logiciels sont loin d'être parfaits. Ainsi, Mandeville fait la remarque suivante à propos de l'utilisation de correcteurs de l'anglais langue seconde par des utilisateurs francophones:

> "Les logiciels vérificateurs-correcteurs et les dictionnaires électroniques sont ... davantage des outils d'aide à la rédaction que des substituts à la bonne connaissance de l'anglais."

Les taux de détection obtenus ne dépassent guère 20 à 30%, et les taux de fausse détection atteignent un niveau que beaucoup d'utilisateurs jugeraient inacceptable. Par exemple, Liechti (1991) cite le chiffre de 85% de fausses détections pour son étude de Bilingual PC Proof. Dans l'évaluation de Tschumi et al. (1996), WinProof obtient un taux de détection de 34% sur un corpus spécialement construit pour tester les correcteurs langue seconde. Tschumi (1996) note une nette amélioration de

ce produit par rapport à Bilingual PC Proof, mais constate également que le taux de fausse détection obtenu lors de cette évaluation est encore très élevé.

Nous constatons qu'il existe un grand écart entre les correcteurs non-commerciaux et les produits commercialisés. Ceci est probablement dû au fait qu'étant donné la gamme d'erreurs plus étendue, la correction en langue seconde requiert des algorithmes d'analyse, de détection et de correction plus sophistiqués et plus difficilement adaptables aux besoins commerciaux que la correction en langue première.

2.3.3. Evaluation d'un correcteur bilingue

Afin d'illustrer les possibilités des correcteurs bilingues commerciaux, nous avons soumis une série de tests à Grammatik anglais pour francophones, version 4.2. pour DOS, de Reference Software. D'une part nous avons utilisé les phrases-test écrites spécialement pour l'évaluation du logiciel automates (voir chapitres 5 et 6), celles-ci contiennent des erreurs syntaxiques et morpho-syntaxiques produites fréquemment en anglais langue seconde par des francophones. D'autre part, nous avons soumis au logiciel un texte, intitulé *Epidemics and History*, écrit par un élève francophone du gymnase cantonal de Neuchâtel lors des examens de maturité. Ce texte fait partie du corpus langue seconde du laboratoire de traitement du langage et de la parole de l'université de Neuchâtel.

Les phrases-test contiennent des erreurs qui appartiennent aux catégories suivantes:

(1) Accord sujet-verbe

Exemple: **Time, money and effort **was** needed.*

Correction: *were*.

(2) Construction du passif

Exemple: ***You are **want** on the phone.*

Correction: *wanted*.

(3) Choix de la préposition dans les compléments de temps

Exemple: ***I have not been feeling well **since** five days.*

Correction: *for*.

(4) Utilisation du temps continu

Exemple: * *Everyone had **been liking** her.*

Correction: *liked*.

(5) Ordre sujet-verbe dans le discours indirect

Exemple: * *He asked where **was I** going.*

Correction: *I was*.

(6) Forme du verbe dans les propositions infinitives

Exemple: * *Charles would like Diana __ stop drinking.*

Correction: *to stop drinking*.

(7) Accord du nom

Exemple: * *Each **applicants** has five choices.*

Correction: *applicant*.

(8) Temps des verbes avec compléments temporels

Exemple: * *Settlers **were** coming here for centuries.*

Correction: *had been*.

(9) Positions des adverbes par rapport aux verbes

Exemple: * *Charles **sits always** in the same chair.*

Correction: *always sits*.

(Ces erreurs sont décrites en détail au chapitre 6.)

Chaque catégorie est représentée par 15 phrases, 10 qui contiennent une erreur et 5 qui n'en contiennent pas. Ces dernières reflètent toutefois les difficultés de la catégorie à laquelle elles appartiennent. Une description plus complète de chaque catégorie ainsi que les instructions suivies par l'auteur des phrases se trouvent dans l'annexe 7. Les résultats de l'évaluation sont présentés dans le tableau 2-2.

Nous constatons que le correcteur bilingue n'a détecté que la moitié des erreurs d'accord (20% des erreurs Accord S-V et 80% des erreurs Accord N). Ceci provient probablement de l'utilisation de règles de mise en correspondance lors de la détection (au lieu de l'analyse de toute la phrase sur laquelle le correcteur monolingue évalué dans la section 2.2.3. peut compter).

Catégorie	Erreurs détectées correctement [%]	Erreurs détectées incorrectement [%]	Erreurs non-détectées [%]	Nombre de non-erreurs détectées
1. Accord S-V	20	0	80	2
2. Passif	50	0	50	0
3. Prép. Temp.	0	0	100	1
4. Cont/simple	0	0	100	3
5. Ordre S-V	0	0	100	1
6. Prop. Inf.	20	0	80	2
7. Accord N	80	0	20	1
8. Compl.temp	0	10	90	2
9.Ordre adv-V	80	0	20	1

Tableau 2-2 - Résultats de l'évaluation du correcteur bilingue "Grammatik anglais pour francophones"

En ce qui concerne les erreurs uniques à l'anglais langue seconde, le correcteur bilingue n'a obtenu de bons résultats que pour les erreurs locales: 50% pour la catégorie (2), 'Construction du passif', et 80% pour la catégorie (9), 'Position des adverbes'. Il n'a détecté que 20% d'erreurs dans la catégorie (6), 'Forme du verbe dans les propositions infinitives', et aucune erreur dans les 4 autres catégories.

Les exemples suivants illustrent quelques phrases pour lesquelles le correcteur a détecté l'erreur avec succès:

Catégorie 2, Construction du passif

* *Frog's legs are rarely* ***eat*** *in England.*

Correction: *eaten.*

* *I was* ***spoke*** *to in a language I could not understand.*

Correction: *spoken.*

Catégorie 9, Position des adverbes par rapport aux verbes

* *Charles* ***sits always*** *in the same chair.*

Correction: *always sits.*

* *Tom* ***seldom is*** *among the first to arrive.*

Correction: *is seldom.*

Nous remarquons que dans ces phrases, les mots en relation directe avec l'erreur occupent des positions adjacentes. Les erreurs sont donc locales et se prêtent particulièrement bien à un traitement par règles de mise en correspondance.

Ci-après, nous présentons quelques exemples de phrases pour lesquelles le correcteur n'a pas détecté d'erreur:

Catégorie 3, Choix de la préposition dans les compléments de temps

* *I have not been feeling well* ***since*** *5 days.*

(au lieu de: *for*)

* *My parents will be staying at our place* ***during*** *two weeks.*

(au lieu de: *for*)

Catégorie 4, Utilisation du temps continu

* *I **am wanting** a breath of air.*

(au lieu de: *want*)

* *Everyone **had been liking** her.*

(au lieu de: *had liked*)

Catégorie 5, Ordre sujet-verbe dans le discours indirect

* *She does not know what **were we** talking about.*

(au lieu de: *we were*)

* *He asked where **was I** going.*

(au lieu de: *I was*)

Catégorie 6, Forme du verbe dans les propositions infinitives

* *Charles would like Diana **stop** drinking.*

(au lieu de: *to stop*)

* *I only want **that you** have a good time.*

(au lieu de: ... *want you to*)

Catégorie 8, Temps des verbes avec compléments temporels

* *I **was asking** you about these doors for months.*

(au lieu de: *had been asking...*)

* *Settlers **were coming** here for centuries.*

(au lieu de: *had been coming...*)

En contraste avec les deux premières catégories, les erreurs précitées englobent un grand nombre d'éléments de la phrase. Dans la catégorie 8 (temps des verbes avec compléments temporels), par exemple, c'est la

présence de la préposition *for* qui détermine le temps du verbe principal. Ces deux éléments se trouvent aux extrémités opposées de la phrase, ce qui empêche le correcteur de détecter ce type d'erreurs.

Lorsque nous lui avons soumis le texte *Epidemics and History* (présenté ci-dessous), le correcteur n'a détecté que 6 erreurs sur les 105 erreurs effectives telles qu'elles ont été identifiées par des enseignants d'anglais.

[1] Epidemics and History

[2] The 11th and 12th centuries were plague-free and had not great political disease. The food production progressed a lot so the population had tripled from 25 millions to 75 in Europe. In 1300 a great famine began and many people moved to towns producing crowding and poverty. So the rat population progressed rapidly. In the towns plague, spreading by a flea, affect rats. If there are no more rats, these fleas infect humans. So we understand in these conditions of poverty and crowding, how the epidemic could rapidly progreas.

[3] The origin of plague is in Asia but bubonic plague arrived in Marseille in 1348. Rapidly the Europe was infected. The effects were terrible: one-third of the population died, and frequent waves of epidemic kept this lower number while 150 years; food prices and wages increased so the landowners couldn't pay all their manors. This was the end of this system.

[4] The epidemic had also religion effect. Many people didn't believe in Church any more and turned to superstitious religion. This phenomen was a great contribution to the Reformation.

[5] A new way in medicine research appeared to try to stop the epidemic: new tests, concepts of contagion and quarantine were accepted, and the first hospitals were established. The scientific area began.

[6] An other effect is the importation of plague in others continents. As the Spanish, who imported the smallpox in Central America. The Spanish survived because they were immunized but the Indians not, and Indian population regressed rapidly.

[7] Epidemic had also effects in military battles. As Napoleon in Russia. The typhus reduced his army from 500.000 to 3000. So the power of Napoleon in Europe were mostly destroy by the typhus than by military battles.

[6] Since the 19th century, epidemics has no more such effects. Governments have made a lot against the plagues. Food and drug are controlled and the very poor areas of the cities are cleaned.

Parmi les erreurs détectées, nous notons les exemples suivants:

* *The epidemic* ***had also*** *religion effect.*

Correction: *also had.*

(ordre verbe-adverbe)

* *So the power of Napoleon in Europe* ***were mostly destroy*** *by the typhus than by military battles.*

Correction: *was mostly destroyed.*

(accord de *were* et construction du passif, *was destroyed*)

A part quelques suggestions relatives à l'utilisation des chiffres (d'après le logiciel, *12th century* aurait par exemple dû être mis en toutes lettres: *twelfth century*), le correcteur n'a commis qu'une faute grave en suggérant *Marseillaise*, au lieu de *Marseilles*, pour remplacer *Marseille*.

Les conclusions que nous pouvons tirer de ces résultats sont les suivantes:

1. Le correcteur bilingue, qui utilise semble-t-il des règles de mise en correspondance, est capable de détecter certaines erreurs locales avec un taux de succès relativement satisfaisant.
2. Par contre, il est incapable de détecter des erreurs globales.

3. Les taux de détection très faibles obtenus pour les phrases-test et pour le texte réel montrent bien que ce détecteur, conçu pour les utilisateurs francophones écrivant en anglais, n'est pas à la hauteur de la tâche.

Conclusion

Dans ce chapitre, nous avons décrit des correcteurs grammaticaux, soit en cours de développement, soit déjà commercialisés. Nous avons présenté les algorithmes qu'ils utilisent pour analyser les textes, détecter et corriger les erreurs. Nous constatons que les correcteurs monolingues commerciaux détectent au maximum une erreur sur deux et produisent un nombre de fausses détections relativement élevé. Les correcteurs commerciaux les plus avancés et ceux au stade de projet commencent à recourir, avec succès semble-t-il, à des algorithmes plus sophistiqués qui font appel à une analyse avancée de chaque phrase. Les types d'erreurs pour lesquelles ce genre d'approche est le mieux adapté, les erreurs d'accord principalement, ne recouvrent toutefois qu'une partie réduite des erreurs produites en langue seconde. De plus, et comme nous l'a montré l'évaluation du correcteur monolingue dans un cadre bilingue, l'approche par analyse avancée mène parfois à des situations où le correcteur se trompe sur la nature de l'erreur et propose à l'utilisateur des corrections insatisfaisantes sinon étranges. Dans notre aperçu des correcteurs bilingues, nous avons constaté que ces lacunes avaient été partiellement comblées dans les logiciels en cours de développement. En effet, ces logiciels semblent tenir compte des erreurs faites dans une langue seconde par des utilisateurs d'une langue première spécifique. D'une part, ils sont programmés pour détecter et corriger certaines erreurs que ces utilisateurs font

fréquemment, des interférences et sugénéralisations notamment. D'autre part, les algorithmes d'analyse auxquels ils font appel possèdent semble-t-il des moyens de compléter l'analyse du texte lorsque celui-ci contient des erreurs, et donc de rassembler certaines informations indispensables pour leur correction. Nous constatons toutefois que les logiciels que nous examinons possèdent une structure relativement rigide et n'opèrent que dans des mondes relativement fermés. Par exemple, ils doivent procéder dans la plupart des cas à une analyse complète de la phrase, les erreurs qu'ils traitent doivent être prévues à l'avance et nous n'avons pas connaissance d'évaluations formelles auxquelles ils auraient été soumis. Nous remarquons donc qu'il faut encore adapter ces logiciels au monde réel. Ceci explique en partie le manque de performance des produits commercialisés qui, comme nous l'avons montré, ne détectent et ne corrigent qu'une fraction des erreurs présentes.

Dans le chapitre qui suit, nous examinons une approche qui fait appel au formalisme de la grammaire lexicale et fonctionnelle (lexical functional grammar, ou LFG). Nous adaptons ce formalisme et certains algorithmes d'analyse aux besoins du traitement des erreurs en langue seconde. Bien que cette approche tente d'accomplir une analyse complète de chaque phrase, le correcteur que nous présentons est capable de traiter des phrases qui contiennent plusieurs erreurs et dont la structure syntaxique n'a été analysée que de manière incomplète. De plus, la détection d'erreurs se fait en grande partie lors de l'analyse même et ne nécessite que peu de règles particulières.

Chapitre 3

Un correcteur basé sur le formalisme LFG

Le formalisme de la Lexical Functional Grammar (LFG), introduit par Bresnan et Kaplan (voir, entre autres, Bresnan, 1982), se révèle particulièrement attrayant pour deux raisons principales. D'une part, le processus d'unification, sur lequel la LFG se base pour construire une représentation du texte, s'adapte particulièrement bien à la détection et à la correction des erreurs. D'autre part, l'addition de contraintes qui permettent la détection de nouvelles erreurs se fait directement dans le lexique et à l'intérieur des règles syntagmatiques, et n'affecte ni ce qui reste des données, ni le processus d'unification. Il est clair que l'emploi du formalisme LFG à lui seul ne permet pas de résoudre tous les problèmes liés à la correction de texte écrit en langue seconde. Il forme toutefois une base

linguistique solide, sur laquelle pourront s'appuyer des algorithmes complémentaires avec des chances de réussite optimales.

La première section de ce chapitre (3.1.) donne un aperçu du logiciel de correction que nous avons développé grâce à l'aide de la LFG. Ce système permet de déterminer dans quelle mesure le formalisme choisi peut contribuer à l'identification et la correction d'erreurs produites à l'écrit en langue seconde. Sous sa forme actuelle, le logiciel détecte certaines erreurs fréquentes produites en français par des personnes anglophones. Le chapitre continue par une introduction à la LFG, où sont décrits en particulier les mécanismes de la LFG utilisés par le logiciel (section 3.2.). La section 3.3. présente certaines difficultés rencontrées par l'approche LFG en présence d'erreurs, ainsi que les outils ajoutés à son formalisme pour traiter ces erreurs. Finalement, la dernière section (3.4.) décrit en détail le logiciel de correction dans lequel ces nouveaux mécanismes sont intégrés.

3.1. Aperçu du logiciel

Deux éléments majeurs entrent en ligne de compte dans un système d'analyse automatique de texte : le formalisme grammatical et les algorithmes de traitement du texte. Emirkanian et Bouchard (1989) utilisent par exemple l'analyseur de Tomita (1987) et la grammaire syntagmatique généralisée (Gazdar, Klein, Pullum et Sag, 1985) pour construire un correcteur de fautes d'orthographe d'usage. Leur système montre qu'une analyse avancée du texte et un formalisme grammatical précis permettent de corriger certaines erreurs morphologiques du français (****gérir** / guérir*, ****pharmatie** / pharmacie*).

Le logiciel décrit dans ce chapitre reprend ces éléments sous une autre forme (analyseur ascendant classique et LFG respectivement), et les utilise pour procéder à la correction d'erreurs morpho-syntaxiques d'anglophones en français. Il laisse de côté des aspects fréquemment rencontrés lors de l'analyse de texte écrit en langue seconde tels que les confusions de faux-amis ou d'homonymes, qui nécessitent d'importantes bases de données. Le système présume que ce type d'aide à la rédaction est effectué dans une phase préliminaire, qui filtre en quelque sorte le texte de certaines inconnues.

On peut diviser les opérations effectuées par le logiciel en trois phases, chacune opérant sur les résultats de la phase précédente. Le programme débute par l'analyse syntaxique du texte qui lui est soumis. Celle-ci crée une série d'arbres syntaxiques à partir du lexique et d'une liste de règles de réécriture. Chaque arbre correspond à une interprétation différente du texte à ce niveau. Ces arbres sont ensuite soumis à la deuxième phase, qui bâtit des structures fonctionnelles à l'aide de règles opérant sur des valeurs fonctionnelles telles que sujet, complément d'objet direct, et sur des attributs, genre et nombre par exemple. Là aussi, plusieurs solutions sont souvent obtenues en partant d'un seul arbre syntaxique. Lors de la troisième et dernière phase, celle de la correction, les solutions de l'analyse fonctionnelle sont classées d'après le genre et le nombre d'erreurs qu'elles contiennent. Etant donné que le système ne possède pas de mécanisme qui lui permettent d'éliminer les solutions redondantes ou inappropriées telles qu'elles surviennent fréquemment lors de l'analyse, seule la meilleure des solutions est conservée et les erreurs qui y figurent sont corrigées. Ceci constitue certainement une des principales limitations du logiciel, car il

semble évident que les cas où plus d'une correction possible se présente sont les plus fréquents.

Voici quelques exemples de types d'erreurs que le logiciel peut corriger actuellement :

- ordre adjectif-nom

* *Il préfère la **rouge maison**.*

(basé sur: *He prefers the red house*)

- cas des pronoms

* *Il **la** donne une pomme.*

(basé sur: *He gives her an apple*)

- genre et nombre à l'intérieur d'un syntagme nominal

* *Il préfère **le** maison **blanc**.*

(basé sur: *He prefers the white house*)

- l'utilisation du subjonctif

* *Je veux qu'il **vient**.*

(basé sur: *I want him to come*)

- l'accord du participe passé

* *Les fleurs qu'il m'a **offert** sont belles.*

3.2. Introduction à la LFG

La grammaire lexicale fonctionnelle (LFG) a été introduite à la fin des années 1970 par Bresnan et Kaplan (voir Bresnan, 1982). Elle diffère des autres grammaires principalement par le fait qu'une grande partie des informations relatives à la spécification de la langue est contenue dans le lexique. Des règles syntagmatiques spécifient toutes les constructions possibles de la langue, et l'information concernant l'occurrence d'un mot dans ces constructions est spécifiée uniquement dans le lexique. Les principes de la LFG proviennent en grande partie de l'ouvrage "The Mental Representation of Grammatical Relations", de Bresnan (1982).

On trouvera dans Sells (1987) un résumé de la LFG dans le cadre d'une comparaison entre la LFG, la théorie de gouvernement et liage de Chomsky et la "Generalized Phrase Structure Grammar" de Gazdar, Klein, Pullum et Sag (1985). Sabah (1988), présente également un aperçu de la LFG avec des exemples en français. Alors que la description de Sells montre surtout une perspective théorique de la LFG, Sabah se concentre sur les aspects plus en rapport avec l'intelligence artificielle, tels que la construction et la forme des f-structures, qu'il identifie avec les 'frames'. Du côté informatique, Frey et Reyle (1983) proposent un analyseur LFG complet en Prolog, qui applique le contrôle fonctionnel et les liaisons à distance. L'article se concentre plutôt sur une explication de la théorie de la LFG, relativement nouvelle à l'époque, que sur le fonctionnement interne de l'analyseur. Block et Haugeneder (1986) ont également construit un analyseur LFG en Prolog. L'accent de leur description est mis sur l'explication des mécanismes utilisés par l'analyseur pour traiter les liaisons à

distance. Eisele et Dorre (1986) présentent surtout la manière dont le lexique et les règles de réécriture sont représentés dans leur système, ainsi que les fonctions qui procèdent à l'unification. Enfin, Block et Hunze (1986) décrivent une méthode qui est indépendante du langage de programmation pour construire en parallèle des c-structures et des f-structures. Les connaissances de ces deux éléments y sont utilisées pour diriger l'analyse en l'empêchant de poursuivre des voies sur lesquelles des erreurs ont été détectées.

3.2.1. Aperçu général

La LFG utilise deux types de données pour décrire les constructions grammaticales : les structures de constituants (c-structures) et les structures fonctionnelles (f-structures). Les c-structures contiennent les informations relatives à l'ordre des mots et à la structure des phrases. Elles sont gouvernées par des règles de réécriture indépendantes de contexte, telles qu'on les rencontre dans la plupart des grammaires, par exemple:

```
P   →  SN  SV
SV  →  V   SN
```

Comme indiqué plus haut, les règles n'imposent sur leurs éléments aucune condition autre que l'appartenance à une certaine classe de mots. Ainsi, des phrases non grammaticales comme **Le garçon pleut le bonheur* possèdent des c-structures bien formées. Afin d'éliminer ce genre de constructions, la LFG introduit la notion d'équations fonctionnelles, qui imposent des contraintes sur les composants fonctionnels de la phrase. Ces équations, aussi appelées annotations, apparaissent à deux niveaux. D'une part, elles sont attachées à chaque élément des règles de réécriture, et d'autre part elles accompagnent les entrées lexicales

pour limiter les domaines dans lesquels ces entrées se manifestent. Ainsi, à chaque phrase sera associé un ensemble d'équations fonctionnelles, appelées descriptions fonctionnelles, qui proviennent des règles de réécriture représentant la phrase et des entrées du lexique contenues dans la phrase. La résolution du système d'équations permet alors d'obtenir un ensemble de structures fonctionnelles (ou f-structures), qui consistent chacune en des listes de paires (attribut,valeur).

Les f-structures sont représentées de la manière suivante:

```
[ attribut-1    valeur-1 ]
[ attribut-2    valeur-2 ]
[ attribut-3    valeur-3 ]
[                        ]
[ ...           ...      ]
```

Par exemple, pour le syntagme nominal *un garçon* on aura:

```
[ PRED          'garçon' ]
[ GENRE         MASCULIN ]
[ NOMBRE        SINGULIER ]
[ DEFINI        +        ]
```

Les attributs de ces f-structures peuvent avoir trois types de valeurs:

1. Un symbole atomique, tel que MASCULIN, SINGULIER, TROISIEME ou une valeur booléenne, telle que '+'.
2. Une forme sémantique, 'manger<(↑SUJET)(↑OBJ)>' ou 'garçon' par exemple. Seul l'attribut PRED peut avoir ce genre de valeur.
3. Une autre f-structure. Ce sont typiquement les fonctions grammaticales telles que SUJET et OBJ qui ont ce genre de valeurs.

Revenons aux équations fonctionnelles et considérons tout d'abord le premier cas où ces équations apparaissent, celui des règles de réécriture. La LFG associe à chaque symbole non terminal de ces règles un certain nombre d'équations portant sur des fonctions grammaticales et sur des caractéristiques syntaxiques telles que le genre, le nombre et la personne. Les deux règles mentionnées plus haut deviennent, lorsqu'on leur ajoute ces équations :

$$\begin{array}{lcccc} P & \rightarrow & SN & & SV \\ & & (\uparrow SUBJ)=\downarrow & & \uparrow=\downarrow \end{array}$$

$$\begin{array}{lcccc} SV & \rightarrow & V & & SN \\ & & & & (\uparrow OBJ)=\downarrow \end{array}$$

Dans la première règle, l'équation (↑SUBJ)=↓ stipule que le sujet de la phrase est la f-structure du syntagme nominal qui se trouve dans cette position à l'intérieur de la phrase. On remarquera la présence des deux symboles, les flèches ↑ et ↓, qui représentent les f-structures mères et filles, respectivement. Ainsi, dans la première équation, ↑ correspond à la f-structure de la phrase (P) et ↓ à la f-structure du syntagme nominal. L'équation associée au groupe verbal SV, ↑=↓, indique que toutes les informations contenues dans la f-structure du groupe verbal sont projetées dans la f-structure de la phrase. L'absence d'équations, comme pour les symboles terminaux des règles, indique comme dans le cas précédent que tous les attributs du noeud sont projetés dans la f-structure du niveau supérieur.

Le second endroit où les équations fonctionnelles apparaissent est le lexique. A chaque entrée lexicale correspond une liste d'équations fonctionnelles décrivant les conditions dans lesquelles l'entrée peut apparaître, par exemple :

```
le          ARTICLE     (↑GENRE) = MASCULIN,
                        (↑NOMBRE) = SINGULIER,
                        (↑DEFINI) = + .

les         ARTICLE     (↑NOMBRE) = PLURIEL,
                        (↑DEFINI) = + .

garçon      NOM         (↑PRED) = 'garçon',
                        (↑GENRE) = MASCULIN,
                        (↑NOMBRE) = SINGULIER,
                        (↑PERSONNE) = TROISIEME.
```

Chaque entrée du lexique introduit un ensemble de valeurs de ce type dans la f-structure à laquelle elle appartient . Etant donné que les f-structures rassemblent des données de plusieurs sources, ces valeurs représentent des conditions. En effet, comme on le verra plus loin, il existe dans la LFG des règles qui spécifient que les contributions de chaque source à l'intérieur d'une f-structure soient compatibles. Par exemple, si le mot *le* introduit le couple (↑GENRE) =MASCULIN dans une f-structure, alors pour toutes les autres contributions à la f-structure la valeur de l'attribut GENRE doit aussi être MASCULIN. (On notera que dans les conditions attachées à *les* ne figurent pas de contraintes pour le genre.)

Afin d'illustrer ces formalismes, analysons la phrase *Le garçon aime les pommes* à l'aide de la grammaire et du lexique suivants:

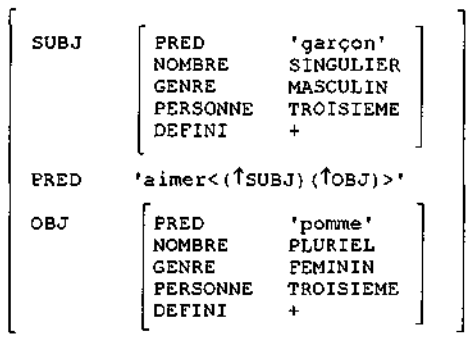
Grammaire:

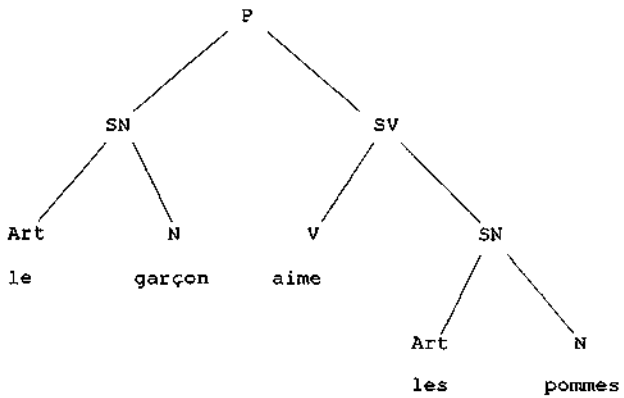
```
1.  P  ->    SN          SV
           (↑SUBJ)=↓     ↑=↓

2.  SV ->    V           SN
                        (↑OBJ)=↓

3.  SN ->    Art   N
```

Lexique:

```
aime        VERBE
            (↑PRED) = 'aimer<(↑SUBJ)(↑OBJ)>'
            (↑PERSONNE) = TROISIEME
            (↑NOMBRE) = SINGULIER
            (↑TEMPS) = PRESENT

pommes      NOM
            (↑PRED) = 'pomme'
            (↑GENRE) = FEMININ
            (↑NOMBRE) = PLURIEL
            (↑PERSONNE) = TROISIEME

le          ARTICLE
            (↑GENRE) = MASCULIN
            (↑NOMBRE) = SINGULIER
            (↑DEFINI) = +

les         ARTICLE
            (↑NOMBRE) = PLURIEL
            (↑DEFINI) = +

garçon      NOM
            (↑PRED) = 'garçon'
            (↑GENRE) = MASCULIN
            (↑NOMBRE) = SINGULIER
            (↑PERSONNE) = TROISIEME
```

La structure fonctionnelle et l'arbre syntaxique sont représentés ci-dessous:





Les informations contenues dans une structure fonctionnelle proviennent la plupart du temps de plusieurs sources. Dans la f-structure qui représente *le garçon,* par exemple, l'attribut DEFINI provient de l'article tandis que les attributs PRED et PERSONNE proviennent du nom. Les deux autres attributs, GENRE et NOMBRE, sont contenus à la fois dans la définition de l'article et dans celle du nom. L'opération qui combine les attributs de plusieurs sources dans une f-structure, comme dans notre exemple, s'appelle l'unification. Sa tâche consiste en quelque sorte à réunir les données de plusieurs sources tout en vérifiant que leurs attributs soient compatibles. Par exemple, prenons deux f-

structures A et B, où A contient l'ensemble de paires <attribut, valeur> $\{<a_1,val(a_1)>, ..., <a_n,val(a_n)>\}$ et B contient l'ensemble $\{<b_1,val(b_1)>, ..., <b_m,val(b_m)>\}$. S'il existe des attributs a_i et b_j tels que $a_i=b_j$ et $val(a_i)\neq val(b_j)$, alors l'unification échoue. Dans le cas contraire, l'unification produit une nouvelle f-structure qui contient la réunion des deux ensembles (voir Kay, 1985).

Pour reprendre notre exemple, considérons les f-structures qui représentent l'article *le* (f1) et le nom *garçon* (f2). L'unification des deux f-structures produit le résultat suivant (f3):

```
f1   [ GENRE      MASCULIN  ]
     [ NOMBRE     SINGULIER ]
     [ DEFINI     +         ]

     +

f2   [ PRED       'garçon'  ]
     [ PERSONNE   TROISIEME ]
     [ GENRE      MASCULIN  ]
     [ NOMBRE     SINGULIER ]

     =

     [ GENRE      MASCULIN  ]
     [ NOMBRE     SINGULIER ]
     [ DEFINI     +         ]
     [ PRED       'garçon'  ]
     [ PERSONNE   TROISIEME ]
```

Si le syntagme nominal était *la garçon*, l'attribut GENRE de l'article ne correspondrait plus à celui du nom, et l'unification échouerait.

3.2.2. Le lexique

Le lexique contient deux types de données : la sous-catégorisation et les annotations fonctionnelles. La sous-catégorisation définit l'environnement fonctionnel de certains mots, et les annotations fonctionnelles permettent d'introduire des règles d'accord, par exemple.

a. Sous-catégorisation

La sous-catégorisation est spécifiée dans le lexique à l'aide de l'attribut PRED. La liste des arguments de PRED définit une application entre des rôles thématiques (agent, patient, but, etc.) et des fonctions grammaticales. Par exemple :

```
                                        agent  patient
regarder   (↑PRED) = 'regarder<(↑SUBJ)(↑OBJ)>'
```

Cette valeur de l'attribut PRED indique que *regarder* peut apparaître dans une phrase avec un sujet et un complément d'objet direct, ces fonctions représentant respectivement l'agent et le patient de l'action 'regarder'.

La LFG utilise les fonctions grammaticales SUBJ, OBJ, OBJ2, OBJ-OBL, COMP, XCOMP, POSS, ADJUNCT, XADJUNCT, TOPIC et FOCUS (Nous conservons ici la terminologie anglaise de la LFG pour des raisons de clarté). Afin de mieux comprendre l'utilisation de ces fonctions, des exemples de phrases ainsi qu'une discussion sommaire des fonctions les plus importantes sont présentés ci-dessous.

- SUBJ désigne le sujet grammatical de la phrase, comme dans l'exemple suivant:

La fille a été renversée par une voiture.

- OBJ indique le complément d'objet direct :

La voiture a renversé la fille.

- OBJ2 indique un complément d'objet second représenté dans la structure de constituants par un syntagme nominal, tel qu'on le trouve en allemand et en anglais:

The man handed the baby a toy.

- COMP et XCOMP désignent des propositions relatives. On indique avec COMP les propositions relatives dans lesquelles toutes les fonctions sous-catégorisables sont présentes dans la c-structure. Pour les fonctions XCOMP, il existe un lien de contrôle entre un élément extérieur à la fonction et une position grammaticale non réalisée dans la c-structure. Les deux exemples ci-dessous illustrent la différence entre les deux fonctions :

a. *Jean veut manger cette pomme.* (XCOMP)
b. *Jean veut que tu manges cette pomme.* (COMP)

Dans (b), toutes les fonctions appartenant à la sous-catégorisation de *manger* sont présentes dans la c-structure, ce qui n'est pas le cas dans (a).

Les deux sous-catégorisations du verbe *vouloir* sont respectivement :

```
a. 'vouloir<(↑SUBJ)(↑XCOMP)>'
b. 'vouloir<(↑SUBJ)(↑COMP)>'
```

b. Annotations fonctionnelles

Comme nous l'avons vu au début de ce chapitre, les annotations fonctionnelles contenues dans le lexique permettent d'insérer des données dans les f-structures, ces données étant composées de paires (attribut,valeur). Une fois présents dans les f-structures, ces attributs agissent comme conditions sur les valeurs des mêmes attributs provenant d'autres entrées lexicales. Ainsi, les attributs déjà mentionnés, tels que NOMBRE, PERSONNE et GENRE, rendent compte des règles d'accord entre verbe et sujet d'une part, et entre nom et déterminant d'autre part. L'exemple ci-dessous illustre le second cas:

```
voiture    (↑PRED) = 'voiture'
           (↑NOMBRE) = SINGULIER
           (↑GENRE) = FEMININ

les        (↑NOMBRE) = PLURIEL

le         (↑NOMBRE) = SINGULIER
           (↑GENRE) = MASCULIN

la         (↑NOMBRE) = SINGULIER
           (↑GENRE) = FEMININ
```

Lorsqu'ils sont insérés dans une f-structure, les attributs de *voiture* sont compatibles avec ceux de *la* et *les*, mais pas avec ceux de *le*.

Le cas de l'accord entre le verbe et le sujet s'avère un peu plus compliqué. Reprenons l'exemple *Le garçon aime les pommes*:

```
[ SUBJ      [ PRED              'garçon'      ]
              NOMBRE            SINGULIER
              GENRE             MASCULIN
              PERSONNE          TROISIEME
              DEFINI            +             ]

  PRED 'aimer<(↑SUBJ)(↑OBJ)>'

  OBJ       [ PRED              'pomme'       ]
              NOMBRE            PLURIEL
              GENRE             FEMININ
              PERSONNE          TROISIEME
              DEFINI            +             ]
```

Nous constatons que le verbe *aimer* et ses attributs (PRED, SUBJ, OBJ) apparaissent dans la f-structure du niveau supérieur, tandis que le sujet *le garçon* possède sa propre f-structure. Les équations fonctionnelles attachées au mot *aime*, qui permettent de spécifier que le sujet doit être à la troisième personne du singulier, sont écrites de la manière suivante dans ce cas:

```
aime        (↑PRED) = 'aimer<(↑SUBJ)(↑OBJ)>'
            (↑SUBJ.PERSONNE) = TROISIEME
            (↑SUBJ.NOMBRE) = SINGULIER
```

D'une manière similaire, les équations fonctionnelles peuvent introduire des contraintes dépendantes de la langue, comme pour l'emploi du subjonctif dans les propositions relatives en français :

Le directeur souhaite que tu prennes tes vacances avant Pâques.

```
souhaiter       (↑PRED) = 'souhaiter<(↑SUBJ)(↑COMP)>'
                (↑COMP.MODE) = SUBJONCTIF
```

Il arrive également que l'équation fonctionnelle doive spécifier, en plus, une condition à remplir par la f-structure. Examinons la phrase *Le verre se brise* et le lexique suivant:

```
brise (↑PRED) = 'briser<(↑SUBJ)>'
                (↑REFLEXIF) =c +
                ...

se              (↑REFLEXIF) = +
```

On remarquera que la seconde équation de l'entrée lexicale *brise* contient le symbole '=c', qui signifie que la f-structure doit contenir l'attribut REFLEXIF avec la valeur '+', en provenance d'une autre source que *brise*, en l'occurrence *se*. Dans l'absence de cette condition, une phrase non grammaticale telle que **Le verre __ brise* serait acceptée par la grammaire.

Cette notation s'utilise beaucoup dans la LFG, comme par exemple dans l'entrée lexicale du pronom anglais *he*, tel que présenté dans Sells (1987) :

```
he          (↑PRED) = 'PRO'
            (↑PERS) = 3
            (↑NUM) = SG
            (↑GEN) = MASC
            (↑CASE) =c NOM
```

La dernière équation attachée à *he* indique que l'attribut 'cas' et la valeur 'nominal' qui lui correspond doivent être introduits par une autre source dans la f-structure. Dans le cas de l'anglais, c'est l'équation fonctionnelle attachée à la position sujet

dans la c-structure qui produit cette valeur. Pour les positions autres que sujet, la c-structure n'introduit pas de valeur pour l'attribut 'cas'.

3.2.3. Règles de bonne formation des f-structures

Un des buts d'une grammaire telle que la LFG consiste à mettre en place un formalisme et des mécanismes permettant de déterminer la grammaticalité d'un texte. Ainsi, dans la LFG, un texte ne peut pas posséder de c-structure si la suite des mots de ce texte ne satisfait pas aux règles de réécriture. Pour les f-structures, Bresnan introduit deux conditions : l'unicité et la consistance.

a. Unicité

La règle d'unicité pose deux conditions pour la bonne formation d'une f-structure: elle doit être complète, c'est-à-dire qu'elle comporte toutes les fonctions grammaticales régies par les prédicats, et elle doit être cohérente, c'est-à-dire que toutes les fonctions grammaticales qu'elle comporte sont régies par des prédicats.

La première condition permet de rejeter des phrases non grammaticales telles que

**Le camion transporte ___.*

```
regarder :    (↑PRED) = 'transporter<(↑SUBJ)(↑OBJ)>'
```

car la fonction OBJ, qui fait partie des arguments de la sous-catégorisation, n'est pas présente dans la f-structure.

La condition de cohérence rejette des phrases telles que:

La fille **pleure à son frère.*

```
pleurer :  (↑PRED) = 'pleurer<(↑SUBJ)>'
```

car le complément *à son frère* pourrait avoir été introduit dans la f-structure comme un objet oblique OBJ-but, et la sous-catégorisation de *pleurer* ne contient pas de telle fonction.

b. Consistance

La condition de consistance stipule que les attributs d'une f-structure doivent avoir des valeurs uniques. Ainsi, on ne peut pas avoir dans la même f-structure les valeurs NOMBRE=SINGULIER et NOMBRE=PLURIEL simultanément, comme dans l'exemple ci-dessous :

Les **fille mangent des pommes.*

les introduit l'attribut NOMBRE=PLURIEL, tandis que *fille* introduit NOMBRE=SINGULIER.

3.3. La détection et la correction en LFG

Dans cette section, nous décrivons les problèmes qu'une mise en oeuvre sur ordinateur du formalisme de la LFG peut rencontrer. Nous présentons ensuite les méthodes que nous avons utilisées pour résoudre ces problèmes.

3.3.1. Les problèmes de la LFG pour traiter les erreurs de langue seconde

Le rôle de l'unification au sein de l'analyse fonctionnelle dans le système est de présenter à l'algorithme de correction une série de structures fonctionnelles qui correspondent à toutes les solutions de l'analyse. Malheureusement, l'unification, telle qu'elle est décrite et employée dans les analyseurs normaux, permet seulement de trouver si une phrase est grammaticale ou non. Comme nous allons le voir plus loin, certaines opérations effectuées lors de l'unification détruisent même des données indispensables à l'identification et à la correction d'erreurs. Afin de parer à ces inconvénients, des modifications à l'algorithme d'unification sont introduites dans le logiciel et décrites dans les sections qui suivent.

a. Problèmes de l'analyse fonctionnelle

Les données nécessaires à l'unification sont les suivantes. Premièrement, on présume que tous les mots ont été identifiés (trouvés dans le dictionnaire). Dans le cas contraire, on peut faire appel à un correcteur morphologique qui fournit une liste de possibilités pour remplacer le mot inconnu. Dans les deux cas, une liste de définitions est disponible pour chaque mot du texte d'entrée. Elle contient un enregistrement non seulement pour toutes les classes auxquelles un mot appartient, mais également pour les différentes interprétations du mot au sein de chaque classe. Par exemple, le mot *ferme* peut représenter un nom ou un verbe, et ce verbe peut être à la première ou à la troisième personne du présent ou du subjonctif. Dans l'exemple:

La belle ferme le voile

l'analyse fournit les informations suivantes :

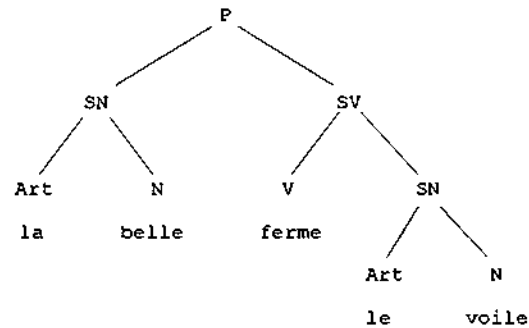
la	2 solutions :	article et pronom
belle	2 solutions :	adjectif et nom
ferme	6 solutions :	nom et verbe
		-présent, 1ère et 3e pers. du singulier
		-subjonctif, idem
		-impératif, 2e pers. du singulier
le	2 solutions :	article et pronom
voile	6 solutions :	identique à *ferme*

(*ferme* peut également être un adjectif. Pour des raisons de clarté, nous nous limitons à l'interpréter ici comme nom ou comme verbe.)

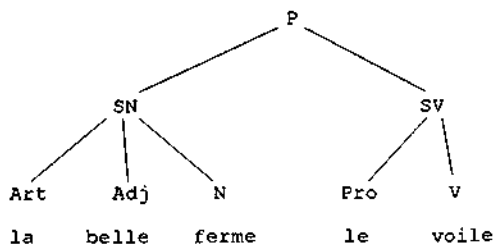
On suppose également que la première phase de l'analyse a produit une série d'arbres syntaxiques correspondant aux différentes solutions de l'analyse syntaxique. Sans informations sémantiques, il est rare qu'un texte contenant une ou plusieurs erreurs ne possède qu'un arbre syntaxique correspondant.

Ces arbres sont au nombre de deux pour notre exemple :

1ére solution :



2ème solution :



On remarque donc que même pour une phrase de quelques mots, le nombre de solutions morpho-syntaxiques peut être assez grand. Pour l'analyseur, ces solutions sont autant de chemins parallèles à parcourir. Si le texte contient une ou plusieurs erreurs, l'analyse de chaque alternative rencontre des problèmes différents. En effet, une des difficultés de l'analyse d'un texte agrammatical réside dans le fait que le processus d'analyse ne peut déterminer immédiatement la cause de l'erreur lorsqu'il en rencontre une. Le seul moyen de parvenir au résultat (la correction d'erreurs), consiste alors à mettre en oeuvre un mécanisme au sein de l'unification permettant de surmonter les erreurs et ainsi poursuivre tous les

chemins jusqu'au bout. Ce n'est que lorsque toutes ces alternatives ont été examinées qu'il est possible de les évaluer et d'en choisir la meilleure afin de réaliser les corrections.

Pour illustrer ce problème, modifions l'exemple de la façon suivante :

**Les belles ferme les voile.*

Les classes des mots restant les mêmes, l'analyse syntaxique produira également les deux arbres décrits plus haut. Pour le premier arbre (SN-V-SN), l'unification échoue pour deux raisons : l'attribut NOMBRE du verbe et du sujet ne correspondent pas, et l'attribut NOMBRE de l'article *les* et du nom *voile* ne correspondent pas. Pour le second arbre (SN-Pro-V), l'unification échoue pour deux autres causes : le nom *ferme* est au singulier alors que l'article et l'adjectif sont au pluriel, et le nombre du verbe *voile* ne correspond pas à son sujet. Définir laquelle des deux solutions choisir est très difficile, sinon impossible. L'important est toutefois que le système soit capable de reconnaître l'existence de ces deux solutions, afin d'offrir à l'utilisateur le choix de la correction à effectuer.

b. Problèmes de représentation des règles grammaticales

Ce n'est pas seulement le mécanisme central de l'unification qui est en question. Le formalisme de la LFG, tel qu'il a été défini par Bresnan, ne permet pas de représenter de manière adéquate toutes les règles grammaticales. Prenons l'exemple de l'accord du participe passé et examinons les règles grammaticales qui s'appliquent dans les cas simples.

A. Participe passé employé avec l'auxiliaire *avoir*.

1. Lorsqu'il n'y a pas de complément d'objet direct, le participe passé reste invariable. Exemple: *Ils avaient couru comme des fous.*

2. Lorsque le complément d'objet direct se trouve après le verbe, le participe passé reste aussi invariable. Exemple: *Les enfants ont dévoré tous les gâteaux.*

3. Lorsque le complément d'objet direct se trouve placé avant le verbe, le participe passé s'accorde en genre et en nombre avec lui. Exemple: *Les fleurs que je t'ai offertes sont belles.*

B. Participe passé employé avec l'auxiliaire *être* (on exclut le cas du verbe pronominal).

1. Le participe passé s'accorde en genre et en nombre avec le syntagme nominal sujet. Exemple: *Les feuilles des arbres étaient tombées.*

C. Participe passé du verbe pronominal.

1. Lorsque le pronom est le complément d'objet direct du verbe, le participe passé s'accorde en genre et en nombre avec le sujet. Exemple: *Elles se sont baignées dans la rivière.*

2. Lorsque le pronom est le complément d'objet indirect du verbe, le participe passé ne s'accorde ni en genre ni en nombre avec le sujet. En revanche, le participe passé s'accordera avec le complément d'objet direct s'il est placé avant le verbe. Exemple: *Elles se sont lavé les mains. Tu ne peux imaginer les choses que je me suis dites.*

Appelons PP_NOMBRE et PP_GENRE les attributs définissant la forme du participe passé. Nous remarquons premièrement que les valeurs PP_GENRE et PP_NOMBRE dépendent d'une part de l'auxiliaire et d'autre part de la position du complément d'objet direct. La dépendance de l'auxiliaire serait représentée habituellement par des attributs et des valeurs contenues dans les entrées lexicales de ces verbes, alors que la dépendance de la position serait spécifiée dans les annotations de c-structures, car ce sont elles seules qui fournissent les conditions sur l'ordre des mots. L'accord du participe passé des verbes pronominaux pose encore plus de problèmes, car les attributs PP_GENRE et PP_NOMBRE dépendent encore du cas du pronom. En résumé, les dimensions suivantes jouent un rôle dans l'accord du participe passé :

1. L'auxiliaire
2. La position et la présence du complément d'objet direct
3. Le cas du pronom pour les verbes pronominaux

Or, il s'avère que l'unification et le formalisme de la LFG ne supportent que l'analyse dans une seule dimension. Les valeurs des attributs proviennent soit du lexique, soit des annotations des c-structures, et ces valeurs sont sans équivoque

(c'est-à-dire qu'il n'existe pas de moyens de spécifier des conditions pour ces valeurs).

c. Problèmes de la correction automatique

L'analyse du texte se heurte ensuite à un autre problème: il n'est pas possible de différencier les données fournies par l'analyse fonctionnelle sans règles supplémentaires. En effet, la deuxième phase du logiciel, l'analyse fonctionnelle, est sujette aux mêmes problèmes que l'analyse syntaxique. Plusieurs solutions découlent bien souvent de cette phase à partir d'un seul arbre syntaxique. Pour illustrer cette situation, considérons la phrase ci-dessous et le lexique qui y est attaché.

Tu **dessine une fleur.*

```
tu          PRONOM      PERSONNE=DEUXIEME
                        NOMBRE=SINGULIER.

dessine     VERBE       PERSONNE=TROISIEME
                        NOMBRE=SINGULIER
                        TEMPS=PRESENT
                        MODE=INDICATIF.

dessine     VERBE       PERSONNE=DEUXIEME
                        NOMBRE=SINGULIER
                        MODE=IMPERATIF.

etc.
```

L'analyse fonctionnelle fournit deux solutions, une pour chacune des interprétations de *dessine*. La première indique l'incompatibilité de la personne

entre le sujet et le verbe. La deuxième solution, quant à elle, indique une incompatibilité pour le mode du verbe, car il est évident que l'impératif ne peut pas être présent à cet endroit de la phrase (cette condition est introduite au niveau des annotations de c-structures et n'est donc pas visible dans le lexique).

Considérons encore l'exemple de la phrase **Il la donne une fleur.* Dans cette phrase, le pronom *la* possède le mauvais cas: accusatif au lieu de datif. Il ne s'agit pas de l'erreur que l'unification découvre dans le texte. Tout d'abord, les annotations de c-structures donnent au pronom la fonction de complément d'objet direct, ce qui enfreint la condition d'unicité des f-structures (voir la section 3.2.). De plus, la sous-catégorisation du verbe *donner* n'est pas satisfaite, car aucun élément ne remplit la fonction de complément d'objet indirect. On a donc les deux erreurs détectées par l'unification:

1- Complément d'objet direct défini deux fois.
2- Sous-catégorisation non satisfaite.

Les deux situations décrites ci-dessus (la multiplicité des solutions partielles de l'unification et le cas où deux erreurs sont découvertes pour une seule faute dans le texte) peuvent bien sûr se produire en même temps, ce qui multiplie encore le nombre de possibilités dont le mécanisme de correction doit tenir compte. Afin de permettre au système de distinguer toutes ces solutions et de corriger toutes les erreurs du texte, il faut donner au linguiste la possibilité de contrôler le comportement du logiciel par des règles, celles-ci opérant sur le contenu des f-structures de chaque solution et sur le type d'erreurs qu'elles contiennent. Ces règles ne doivent pas seulement rendre possible la classification des erreurs, mais

leur fonction doit aussi inclure la spécification de l'algorithme à exécuter pour corriger ces erreurs.

Reprenons le cas où la phrase contient un pronom dont le cas est incorrect (exemple 11). Si le linguiste sait que ces deux erreurs vont apparaître dans cette situation, il peut spécifier la règle suivante :

SI La sous-catégorisation n'est pas satisfaite
ET L'objet direct est défini deux fois
ET Un des objets directs est un pronom

ALORS Chercher dans le lexique un pronom ayant les mêmes caractéristiques que celui dans le texte sauf pour le cas qui doit être DATIF.
ET Remplacer le pronom du texte par le nouveau.

Il faut noter que le cas traité par cette règle est très spécifique. Toutefois, étant donné que les erreurs produites en langue seconde sont souvent prévisibles, il sera facile pour le linguiste de spécifier le comportement du logiciel en réponse aux cas qu'il connaît.

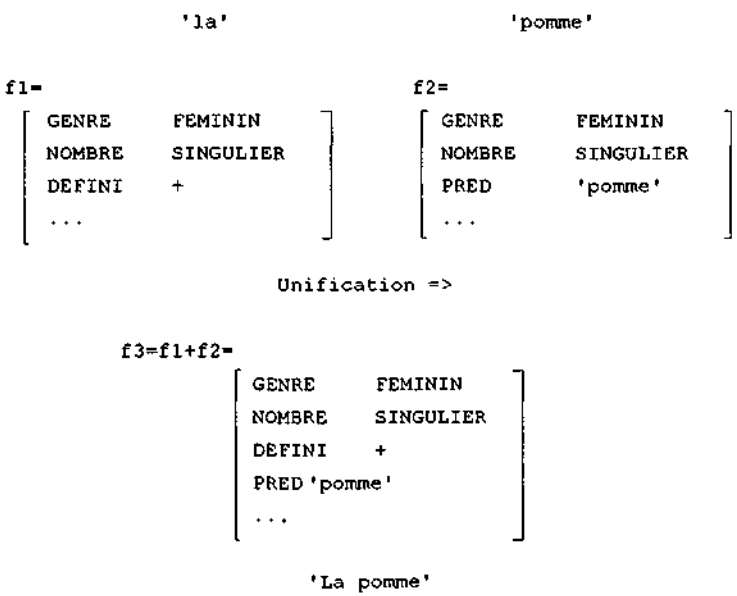
Nous avons montré, dans cette section, un certain nombre de problèmes auquels se heurtera un logiciel de correction automatique basé sur la LFG. Dans la section qui suit, nous décrivons les recherches que nous avons effectuées dans le but d'identifier des solutions à ces problèmes.

3.3.2. Solutions aux problèmes

Cette section décrit le symbolisme et les algorithmes utilisés dans le logiciel de correction afin de tenir compte des problèmes soulevés dans la sous-section précédente. De nouvelles notations ont été introduites pour compléter la LFG et permettre au linguiste de spécifier le comportement du logiciel dans certaines situations. Les algorithmes, quant à eux, représentent des extensions aux mécanismes rencontrés habituellement dans les logiciels de traitement automatique de texte, tels que l'analyseur fonctionnel. D'autres formalismes, tel que celui qui rend possible la spécification de la méthode de correction du texte, sont présentés dans la section suivante (3.4), celle qui décrit le logiciel.

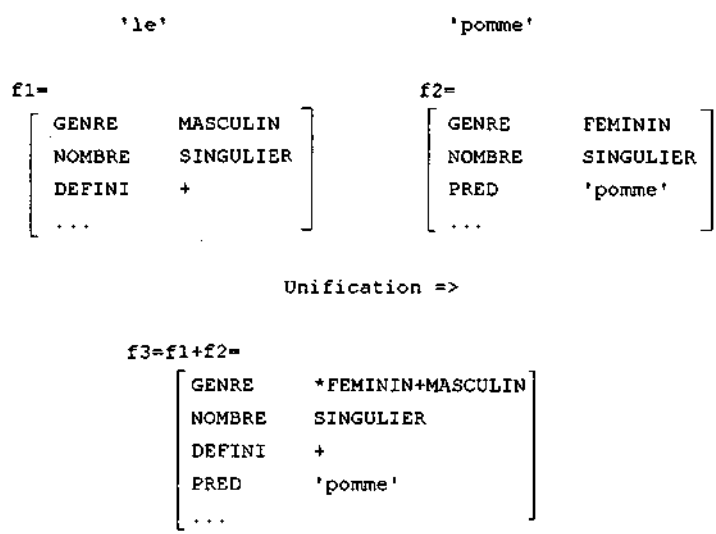
a. Valeurs de référeoce et uoificatioo ooo destructrice

Le mécanisme central de la LFG qui permet de construire les structures fonctionnelles est l'unification. Elle entre en jeu lorsque l'analyse syntaxique a produit un certain nombre d'arbres dont les feuilles sont des éléments du lexique. Comme mentionné dans l'introduction à la LFG (section 3.2), l'unification combine les attributs de plusieurs f-structures. Si deux f-structures à unifier contiennent le même attribut et que celui-ci a la même valeur, l'unification n'insère dans la nouvelle f-structure qu'une seule copie de l'attribut et de sa valeur. L'exemple ci-dessous illustre cette propriété.



Lorsque les valeurs d'un attribut diffèrent d'une f-structure à l'autre, la LFG présume que le texte n'est pas grammatical et l'unification s'arrête. Si le logiciel de correction d'erreurs utilisait ce procédé, il ne pourrait pas distinguer entre les solutions générées par les étapes précédentes (contenant toutes des erreurs) et la correction serait impossible.

Modifions quelque peu l'exemple ci-dessus en remplaçant *la* par *le*:



Nous constatons que les attributs GENRE de *le* et *pomme* ne correspondent pas. Afin de pouvoir continuer l'analyse, la valeur par défaut de cet attribut doit être connue. La solution consiste à indiquer dans le lexique que la valeur d'un attribut pour un mot prend le dessus sur la valeur du même attribut pour un autre mot. Nous avons donc décidé que le GENRE d'un nom a priorité sur le GENRE d'un article et que le NOMBRE de l'article a priorité sur le NOMBRE du nom. L'opération exacte se fait de la façon suivante. Lorsque la valeur d'un attribut est spécifiée dans le lexique, on remplace le signe '=' par '=r', ce qui indique que la valeur à droite prend le dessus dans une f-structure.

Par exemple:

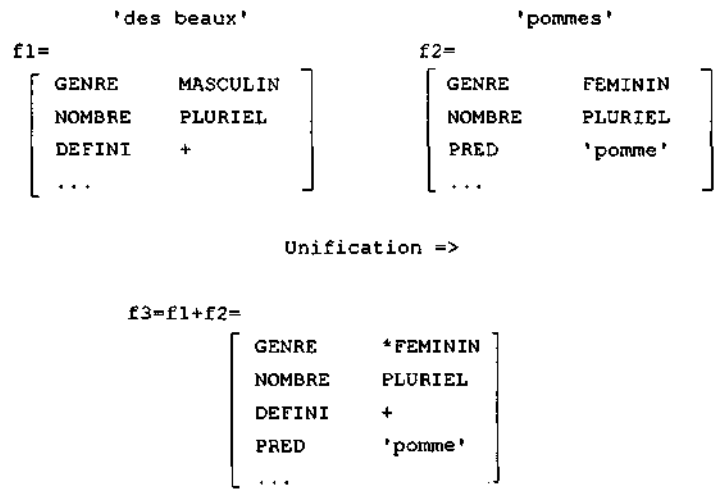
```
la          ARTICLE     GENRE = FEMININ
                        NOMBRE =r SINGULIER
                        DEFINI = +

pomme       NOM         GENRE =r FEMININ
                        NOMBRE = SINGULIER
                        PRED = 'pomme'
```

Après l'unification de *le* et *pomme*, la f-structure devient :

```
f3=f1+f2=
          [ GENRE     FEMININ   ]
          | NOMBRE    SINGULIER |
          | DEFINI    +         |
          | PRED      'pomme'   |
          [ ...                 ]

          *'Le pomme'
```

Si notre exemple intervient dans une phrase telle que ***le** pomme que tu as mangé n'était pas bonne*, la valeur de référence pour l'attribut GENRE permet non seulement de corriger l'article, mais aussi le mauvais accord du participe passé *mangé*.

Considérons maintenant le cas où un adjectif est présent entre l'article et le nom:



Bien que l'analyse puisse continuer normalement grâce aux valeurs de référence, la correction de *beaux* posera un problème car la f-structure f1 représente les données de deux mots dont l'unification s'est effectuée avec succès. Les informations contenues dans f1 ne permettent en effet pas de connaître la source de l'erreur. Pour parer à cet inconvénient, on ajoute à chaque ligne des f-structures un champ contenant l'élément lexical ou la c-structure d'où provient l'attribut. De plus, tous les attributs d'un élément lexical sont inclus dans la f-structure.

Pour permettre une meilleure visualisation des nouvelles informations, le logiciel utilise deux représentations de chaque structure fonctionnelle. La représentation

externe est celle qui est utilisée pour l'unification et qui a une forme identique aux f-structures habituelles. La représentation interne, quant à elle, contient toutes les données nécessaires à la correction du texte.

Les deux représentations de la f-structure f3 du dernier exemple ont la forme suivante:

```
f3(externe)=
              [ GENRE     FEMININ  ]
              [ NOMBRE    PLURIEL  ]
              [ DEFINI    +        ]
              [ PRED      'pomme'  ]
              [ ...                ]
```

```
f3(interne)=

    [ GENRE     MASCULIN   source=f1, incorrect ]
    [ GENRE     FEMININ    source=f2            ]
    [ NOMBRE    PLURIEL    source=f1            ]
    [ NOMBRE    PLURIEL    source=f2            ]
    [ DEFINI    +          source=f1            ]
    [ PRED      'pomme'    source=f2            ]
    [ ...                                       ]
```

La tâche incombe alors au linguiste de s'assurer que les valeurs de référence sont placées au bon endroit dans le lexique, de façon à ce qu'il n'y ait pas de conflit lors de l'unification.

b. Règles d'environnement lexical

Nous avons vu dans la section 3.3.1 que le formalisme de la LFG ne permettait pas de représenter certaines conditions grammaticales. La complexité des règles à appliquer dans ces cas requiert l'addition de nouvelles règles, appelées 'règles d'environnement lexical', que nous présentons ci-dessous. Elles permettront de spécifier de nouvelles contraintes grammaticales d'une manière compatible avec le formalisme LFG habituel. Ces règles ont les caractéristiques suivantes :

1. Elles prennent comme arguments les valeurs d'attributs existants dans le réseau des f-structures. En somme, ces règles s'appliquent lorsque l'unification est terminée.
2. Elles génèrent de nouvelles valeurs dans ces f-structures.
3. Leur but est d'introduire dans la f-structure la valeur requise d'un attribut.
4. Elles peuvent être introduites dans le lexique aussi bien que dans les annotations des c-structures.

Voyons comment ces règles sont spécifiées dans le cas de l'accord du participe passé. Tout d'abord, elles n'interviennent que lorsque les auxiliaires *être* ou *avoir* sont utilisés dans le texte, ce qui implique qu'elles doivent être attachées aux entrées lexicales de ces auxiliaires. Ensuite, leur domaine d'application se limite à la f-structure dont le participe passé est la tête ainsi qu'aux f-structures qui en dépendent. L'application de ces règles requiert également la définition des attributs COD-POSITION, COD-PRONOM et COI-PRONOM. Le premier est

introduit dans les annotations des c-structures et les deux autres dans les entrées lexicales de pronoms. S'il n'y a pas de complément d'objet direct dans la phrase, COD-POSITION n'est pas défini. Il en va de même pour les deux autres attributs dans les cas où le COD ou le COI n'est pas présent.

Les règles d'environnement lexical A1 jusqu'à C2 deviennent ainsi :

```
A1:   (attachée à l'auxiliaire 'avoir')
      SI          COD-POSITION = INDEFINI
      ALORS       PP_NOMBRE =r SINGULIER
        ET        PP_GENRE =r MASCULIN

A2:
      SI          COD-POSITION = APRES-VERBE
      ALORS       PP_NOMBRE =r SINGULIER
        ET        PP_GENRE =r MASCULIN

A3:
      SI          COD-POSITION = AVANT-VERBE
      ALORS       PP_NOMBRE =r COD.NOMBRE
        ET        PP_GENRE =r COD.GENRE

B1:   (attachée à l'auxiliaire 'être')
      SI          VERBE-PRONOMINAL = -
      ALORS       PP_NOMBRE =r SUJET.NOMBRE
        ET        PP_GENRE =r SUJET.GENRE
```

Et ainsi de suite.

c. Examen de la puissance du nouveau symbolisme

Les formalismes que nous avons vus ci-dessus représentent des outils dont le linguiste se sert pour décrire le comportement du logiciel lors de l'analyse et la

correction. Leur potentiel réside dans le fait qu'ils permettent de spécifier le comportement de l'analyseur en présence d'erreurs au niveau syntaxique. Le domaine de ces outils se limite à l'application très stricte de deux types de règles: les règles de grammaire de la langue seconde qui ne peuvent pas être spécifiées à l'aide du formalisme habituel de la LFG, et les cas reconnus où l'utilisateur produit des erreurs prévisibles au niveau syntaxique. Utilisées dans un logiciel de correction en langue seconde, ces règles ne s'appliquent que lorsque toutes les erreurs lexicales ont été traitées et corrigées. Malgré ces restrictions, l'introduction des nouveaux concepts offre de nouvelles possibilités aux correcteurs automatiques. En particulier, les valeurs de référence et les règles d'environnement lexical permettent de spécifier à l'intérieur même des données linguistiques le comportement de l'analyseur en cas d'erreur. Il est envisageable que dans de nombreux cas l'utilisation de ces mécanismes réduise l'interaction nécessaire entre le logiciel et l'utilisateur lors de la détection d'erreurs et la correction. En effet, le manque d'informations concernant le sens du texte empêche le logiciel de déterminer la nature de l'erreur la plupart du temps, ce qui ne lui permet pas ensuite de la corriger automatiquement. A l'aide des mécanismes ci-dessus, le logiciel pourra dans de nombreux cas présenter à l'utilisateur des alternatives ou des corrections bien plus précises.

3.4. Description du logiciel

Le logiciel tel qu'il a été développé exploite la théorie LFG modifiée comme indiqué ci-dessus afin de détecter et corriger certaines erreurs dans des textes écrits par des anglophones. Malgré le nombre limité de règles de grammaire qu'il possède, le type d'erreurs que le logiciel traite est relativement grand. Il permet

de vérifier et de corriger les types suivants (on trouvera dans le chapitre 4 des exemples pour chacun des cas) :

- Le genre et le nombre à l'intérieur du groupe du nom, y compris les déterminants et les adjectifs.
- L'accord du verbe avec le sujet.
- L'accord du participe passé, y compris dans les phrases relatives.
- L'emploi des prépositions.
- Le cas des pronoms.
- L'ordre des mots (adjectif-nom ou nom-adjectif).
- L'utilisation du subjonctif dans les propositions relatives.
- L'oubli de l'adverbe *ne* dans une négation.
- L'usage des prépositions *à* et *de* devant les propositions subordonnées.

Un aspect souvent négligé des logiciels consiste à leur donner le moyen de dialoguer de manière intelligente avec les humains. D'un côté, il doit être possible au linguiste de spécifier le comportement du système à l'aide d'un outil autre qu'un langage de programmation. De l'autre côté, celui de l'utilisateur, le problème consiste à décider s'il faut continuellement offrir à cet utilisateur tous les choix possibles de correction de son texte, ce qui peut devenir ennuyeux, ou s'il faut prendre la meilleure solution et modifier le texte sans son intervention. Dans le premier cas, l'optique choisie est de donner au linguiste un outil aussi proche du formalisme habituel que possible. Par exemple, les règles de réécriture et les annotations sont représentées dans le logiciel dans un format très proche de celui trouvé dans la littérature. En outre, toutes les règles sont contenues dans des fichiers de données facilement accessibles et indépendants du programme même

(leur modification ne requiert pas la recompilation du programme). L'interface avec le linguiste dans le logiciel se présente actuellement sous une forme qui favorise l'accès aux données internes du programme. A ce stade des recherches, il est en effet indispensable qu'il ait la possibilité d'examiner les structures créées lors de l'exécution de l'analyse, tels que les arbres syntaxiques, les structures fonctionnelles et la classification des erreurs. Par exemple, le programme affiche un certain nombre de messages indiquant les stades par lesquels il passe. Bien que la correction soit effectuée dans le texte, le linguiste retient toujours un contrôle très étroit du comportement du logiciel et des messages qui sont affichés. L'interaction avec l'utilisateur, quant à elle, est un aspect peu développé dans le logiciel; elle se limite à indiquer une seule possibilité d'altération du texte pour chaque erreur. Ceci provient du fait que c'est l'aspect linguistique-informatique de la correction d'erreurs qui nous intéresse ici, et non pas l'interface.

3.4.1. Structure générale

Le logiciel se compose de deux éléments bien séparés : les données linguistiques et les algorithmes de traitement de ces données. Les premières contiennent le lexique, les règles de réécriture, les règles d'environnement lexical et les règles de correction. Quant aux algorithmes de traitement des données, ils se répartissent en trois sous-éléments: l'analyse syntaxique, l'analyse fonctionnelle et la correction. Dans les logiciels d'analyse automatique utilisant le formalisme de la LFG (voir la revue bibliographique dans la section décrivant la LFG (3.2.)), les deux premières étapes sont habituellement accomplies en parallèle, c'est-à-dire que le logiciel construit les c-structures et les f-structures en même temps (la

troisième étape étant inexistante dans les logiciels dont le but est l'analyse seule). Cette façon de faire accélère considérablement le processus d'analyse, mais ne permet malheureusement pas de tenir compte des phrases qui contiennent des erreurs. Ces analyseurs acquièrent leur rapidité également en écartant certaines possibilités dès qu'une erreur se produit; ils évitent ainsi de poursuivre des chemins conduisant à des constructions non-grammaticales.

Le logiciel décrit ici exécute les trois étapes en série. Les règles de réécriture sont tout d'abord utilisées pour construire les c-structures. Cette opération est exécutée à l'aide d'un analyseur ascendant, qui se base sur trois types de données : le dictionnaire, les flexions (informations orthographiques) et les règles de réécriture sans leurs annotations. Ensuite, les annotations attachées aux noeuds des c-structures et les règles d'environement lexical sont utilisées par l'unification pour produire d'une part les f-structures correspondant aux différentes interprétations du texte, et d'autre part une liste d'erreurs trouvées dans le texte. Enfin, lors de la troisième étape, le contenu des f-structures permet d'accepter la phrase ou de déterminer la nature des erreurs et éventuellement de les corriger. Cette dernière étape est divisée en trois phases: la classification des erreurs, l'identification de celles-ci et la correction, qui génère une liste de modifications à apporter au texte.

Les sections qui suivent décrivent le contenu et le fonctionnement des composantes du logiciel tel qu'il a été développé.

3.4.2. Données linguistiques

a. Le lexique

Le lexique représente la base de données des mots qui sont reconnus par le programme. Il comporte deux parties. La première, le dictionnaire, contient la liste des mots avec leurs attributs. La seconde, le fichier des flexions, contient les informations morphologiques des mots du dictionnaire. Les informations contenues dans le dictionnaire indiquent pour chaque mot à quelle classe il appartient et quel est son modèle morphologique. A ces données s'ajoutent des valeurs dépendantes de sa classe, telles que le cas pour les pronoms et la sous-catégorisation pour les verbes. La spécification dans le logiciel de ce dernier élément s'avère particulièrement importante car il arrive bien souvent que des verbes possèdent plusieurs possibilités pour cet attribut. Considérons l'exemple du verbe *aimer* dans les trois environnements suivants:

a. *Elle aime les fleurs.*
b. *Elle aime recevoir des fleurs.*
c. *Elle aime que tu lui offres des fleurs.*

Les entrées lexicales qui leur correspondent ont la forme suivante:

```
a. aimer        CLASSE = VERBE
                (↑PRED) = 'aimer<(↑SUBJ)(↑OBJ)>'

b. aimer        CLASSE = VERBE
                (↑PRED) = 'aimer<(↑SUBJ)(↑XCOMP)>'
                (↑XCOMP SUBJ) = (↑SUBJ)

c. aimer        CLASSE = VERBE
                (↑PRED) = 'aimer<(↑SUBJ)(↑COMP)>'
```

Ces trois formes du verbe *aimer* peuvent être spécifiées séparément dans le dictionnaire de la façon suivante :

```
aimer       CLASSE = VERBE
            [ (↑PRED) = 'aimer<(↑SUBJ)(↑OBJ)>' ]
            [ (↑PRED) = 'aimer<(↑SUBJ)(↑XCOMP)>'
              (↑XCOMP SUBJ) = (↑SUBJ) ]
            [ (↑PRED) ='aimer<(↑SUBJ)(↑COMP)>' ]
```

De cette manière, la représentation des différentes formes de sous-catégorisation peut contenir pour chacune d'elle une liste de conditions fonctionnelles indépendantes des autres formes.

Le fichier des flexions, quant à lui, décrit les différentes constructions de chaque mot du dictionnaire : le pluriel des noms, le pluriel et le féminin des adjectifs, les différents temps et personnes des verbes, par exemple. Il permet également de définir comment former les substantifs à partir des verbes.

Le fichier des flexions contient trois niveaux :

1. Les modèles lexicaux, sur lesquels pointe chaque mot du dictionnaire. Ils permettent d'éviter d'avoir à spécifier séparément l'orthographe de chaque mot du dictionnaire. Il existe par exemple un seul modèle lexical pour les verbes réguliers en -er (*aimer*). La fonction des modèles lexicaux est d'associer les différentes racines des mots, spécifiées dans le dictionnaire, avec des terminaisons.

2. Les modèles de terminaisons, sur lesquels pointent les modèles lexicaux, qui contiennent les chaînes de caractères à ajouter aux racines pour former les mots.

3. Les modèles d'attributs, sur lesquels pointent les modèles de terminaisons, qui contiennent des listes d'attributs, comme le genre, le nombre ou la personne, associés avec les terminaisons.

Considérons l'exemple du verbe *partager*, qui possède deux racines, *partag-* et *partage-*. Son modèle lexical, comme tous les verbes qui se terminent en *-ger*, est le verbe *manger*. Le dictionnaire contiendra ainsi la ligne suivante :

```
partager   morph=(manger,partag,partage)
```

qui indique que le modèle lexical du verbe *partager* est *manger*, et que ses deux racines sont *partag-* et *partage-*.

Le modèle lexical *manger* contient les données suivantes :

```
manger      temps=présent, mode=indicatif,
            t-modèle=pres-reg, racines=1,1,1,2,2,1 ;
            temps=imparfait, mode=indicatif,
            t-modèle=imp-reg, racines=2,2,2,1,1,2 ;
```

Les six valeurs qui suivent l'attribut 'racines' indiquent les numéros des racines à associer avec les terminaisons des modèles 'pres-reg' et 'imp-reg'. Mis à part cette sélection entre la première et la seconde racine, les terminaisons de *manger* sont identiques à celles de *aimer*. Il sera donc possible d'utiliser les mêmes modèles.

Regardons de plus près le modèle de terminaisons 'pres-reg', commun au présent de tous les verbes réguliers en *-er*, et le modèle 'imp-reg', commun à l'imparfait :

```
pres-reg   a-modèle=conjug,
           term=(e,es,e,ons,ez,ent) .

imp-reg    a-modèle=conjug,
           term=(ais,ais,ait,ions,iez,aient).
```

Ces modèles contiennent deux types d'information. La partie de droite contient d'une part les terminaisons pour chaque personne de la conjugaison, et d'autre part le nom du modèle d'attributs contenant la définition de la liste des terminaisons, dont voici un exemple :

```
conjug      nombre=singulier, personne=première ;
            nombre=singulier, personne=deuxième ;
            nombre=singulier, personne=troisième ;
            nombre=pluriel,   personne=première ;
            nombre=pluriel,   personne=deuxième ;
            nombre=pluriel,   personne=troisième .
```

Pour un adjectif, par exemple, le modèle de terminaisons contient quatre termes, et le modèle d'attributs indique que ces quatre valeurs correspondent au singulier et au pluriel du masculin et du féminin.

b. Les règles de réécriture

Le fichier des règles de réécriture renferme les règles syntagmatiques de la grammaire ainsi que les annotations fonctionnelles qui leur correspondent (voir l'introduction à la LFG, section 3.2.). Le contenu et la forme des règles de réécriture sont conformes à ceux définis pas la LFG. Chaque règle spécifie comment un symbole de niveau supérieur (syntagme verbal, syntagme nominal, par exemple) se décompose en une liste de symboles de niveau inférieur (nom,

adjectif). Ainsi, un syntagme nominal, membre de gauche de la règle de réécriture, se décompose en un déterminant, un adjectif et un nom, membres de droite. De plus, l'ordre dans lequel ces derniers apparaissent dans les règles définit l'ordre des mots qui leur correspondent dans le texte.

c. Les règles d'environnement lexical

Les règles d'environnement lexical permettent d'ajouter des contraintes sur les éléments du lexique dans le cas où le formalisme de la LFG ne suffit pas. Lors du traitement du texte, ces règles sont appliquées après la phase d'unification proprement dite. Elles prennent comme arguments les attributs de la f-structure dans laquelle elles interviennent et produisent de nouveaux attributs. Il va de soi que les attributs nouvellement introduits doivent également obéir aux contraintes de bonne formation des f-structures. L'utilisation de ces règles est décrite dans la section 3.3.2.b.

d. Les règles de correction

Les règles de correction interviennent une fois que l'algorithme de classification des erreurs a trouvé la meilleure solution parmi les structures fonctionnelles produites par l'analyse fonctionnelle. L'utilisation de ces règles se fait d'une manière similaire aux règles d'environnement lexical, dans le sens que toutes deux consistent en des conditions sur les attributs des f-structures dans lesquelles elles s'appliquent. Le but de ces règles est de déterminer quelle procédure de correction activer pour corriger l'erreur. A chaque attribut susceptible de causer une erreur correspond un ensemble de règles qui vérifient certaines conditions dans la f-structure courante. Lorsqu'il traite une erreur, le logiciel parcourt toutes

les règles attachées à l'attribut qui a causé l'erreur. Si toutes les conditions d'une régle sont remplies, la procédure de correction qui lui correspond est exécutée. Considérons par exemple la phrase incorrecte **Il la donne une fleur.* L'analyse fonctionnelle de cette phrase produit une erreur pour l'attribut OBJ, qui est défini deux fois. La règle qui permet de déterminer qu'il faut remplacer *la* par *lui* a la forme suivante.

```
Conditions
      PRED              = 'donner'
      OBJ               = *2        (défini deux fois)
      OBJ.OBJ-PRONOM    = +   (un des OBJ est un pronom)
Actions
      PROCEDURE-51
```

Il faut se rappeler que l'analyse fonctionnelle permet de déterminer uniquement le fait que l'attribut qui a provoqué l'erreur est OBJ. Les conditions contenues dans les règles de correction doivent donc vérifier tous les autres paramètres entrant en jeu.

e. Les procédures de correction

Deux caractéristiques différencient les procédures de correction des autres données du logiciel. Tout d'abord, il n'existe pas aujourd'hui de formalisme ou de théorie adapté à la spécification des données relatives à la correction de texte. Ensuite, et comme leur nom l'indique, les procédures de correction ne sont pas de simples listes de conditions que certains éléments doivent remplir. Elles contiennent au contraire des suites d'opérations à effectuer dans le texte, telles que le remplacement, l'insertion ou la suppression d'un mot.

A première vue, l'utilisation d'un langage de programmation pour décrire ces opérations semble une bonne solution. Cette façon de faire pose toutefois des problèmes importants. Il n'est en effet pas concevable de devoir reconstruire le logiciel après chaque modification de la base de données des procédures. De plus, la correction du texte peut se réduire à l'exécution d'une suite d'opérations de base. Que celles-ci soient spécifiées dans le langage de programmation ou dans un formalisme propre revient alors pratiquement au même.

C'est pourquoi un symbolisme a été défini, qui permet au linguiste de spécifier les opérations de base à effectuer pour la correction d'un mot ou d'une construction fausse. On peut distinguer deux types d'actions que les procédures doivent exécuter dans le texte: celles qui impliquent le remplacement d'un mot (erreurs morphologiques principalement), et celles qui modifient l'ordre des mots dans la phrase. La différence entre les deux réside dans la connaissance par le logiciel de l'emplacement dans la phrase des mots à modifier. Dans le premier cas, la position du mot à remplacer est connue car elle a été encodée dans la f-structure du mot lors d'une des phases précédentes de l'analyse. Le second cas, par contre, est beaucoup plus délicat. Les exemples illustrent le problème de l'insertion de l'adverbe *ne* dans une phrase.

a. * *Il ___ aime pas les pommes.*
b. * *L'enfant qui ___ aime pas les pommes ...*
c. * *Il ____ lui a pas donné l'argent.*

Dans (a), l'adverbe s'insère entre le sujet et le verbe, dans (b), entre le pronom relatif et le verbe, et dans (c) entre le sujet et le pronom.

Les données accessibles aux procédures de correction (celles qui sont comprises dans les f-structures) ne suffisent pas à déterminer l'emplacement de l'adverbe dans la phrase, car il n'existe pas de lien entre ces f-structures et les c-structures, ces dernières contenant les informations requises. Le logiciel ne contient pas encore le symbolisme nécessaire à la spécification de ce genre de liens et le rôle des procédures de correction dans ce genre de situation se limite à imprimer un message du genre "Introduire l'adverbe *ne* avant le verbe *aimer*".

Le cas du remplacement d'un mot par un autre s'avère beaucoup plus simple à traiter. Les procédures de correction contiennent alors des listes d'instructions qui spécifient la valeur des attributs à assigner au nouveau mot. Deux situations peuvent se présenter. Dans la première, toutes les informations nécessaires à l'identification du nouveau mot sont contenues dans la f-structure même. Dans la seconde situation, les valeurs de certains attributs doivent être obtenues à partir d'autres f-structures.

Examinons tout d'abord le premier cas. Celui-ci se présente lors du traitement de la phrase **les grand maisons sont inhabitées.* La f-structure qui décrit **les grand maisons* contient les valeurs de référence pour les attributs GENRE et NOMBRE, qui proviennent du nom et de l'article respectivement. Les valeurs de ces attributs, ajoutées aux autres déjà présentes dans la f-structure, permettent d'identifier automatiquement la forme correcte de l'adjectif dans le lexique. La procédure de correction ne contient alors que la ligne suivante :

```
AUTOMATIC_CORRECTION
```

Pour illustrer le second cas, reprenons la phrase **Il la donne une fleur*. Etant donné que l'erreur apparaît dans la f-structure du niveau supérieur mais que les données du mot à corriger, *la*, se trouvent dans la f-structure (OBJ) à un niveau inférieur, la procédure de correction doit combiner les attributs des deux f-structures pour spécifier entièrement le nouveau mot (*lui*). Le linguiste dispose des opérations suivantes pour spécifier ces attributs:

```
BUILD_REFERENCE_FRAME
SET CAS TO OBJ2
SET PERSONNE TO OBJ.PERSONNE
SET NOMBRE TO OBJ.NOMBRE
SET CLASSE TO PRONOM
FIND_CORRECT_FORM
```

La première opération (BUILD_REFERENCE_FRAME) indique au logiciel d'initialiser une liste d'attributs de référence. SET permet de spécifier la valeur et la source des attributs et de les insérer dans cette liste.
FIND_CORRECT_FORM effectue une recherche dans le lexique du mot qui correspond aux attributs dans la liste de référence.

3.4.3. L'analyse syntaxique

Le but de l'analyse syntaxique consiste à produire un arbre syntaxique pour chaque interprétation du texte. Lorsque le texte contient une erreur, la nature des analyseurs les plus répandus les empêche de construire même des solutions partielles de ces arbres. Une des exceptions est l'analyseur ascendant. Cet analyseur bâtit les arbres syntaxiques par itérations successives qui lui permettent de traiter chaque groupe de mots indépendamment des autres, le tout n'étant rassemblé que vers la fin de l'analyse. Ainsi, il assigne tout d'abord à chaque mot

une ou plusieurs catégories syntaxiques. Il identifie ensuite les règles de réécriture de la grammaire pour lesquelles les suites de mots correspondent aux membres de droite des règles. Le succès de l'analyse dépend de la présence d'un arbre syntaxique dont les feuilles contiennent tous les mots de la phrase et dont la racine de l'arbre correspond au symbole maximal.

Alors que l'analyseur ATN (Winograd, 1983) opère de manière descendante, l'analyseur ascendant part des catégories syntaxiques des mots qu'il rencontre. Le travail principal d'un tel système consiste à chercher parmi les règles de réécriture celles dont le membre de droite correspond à une suite de catégories syntaxiques provenant du texte à analyser. Par exemple, si nous avons la règle de réécriture suivante,

```
SN  →  Article  Adjectif  N
```

un analyseur opérant en profondeur n'utilise cette règle que s'il existe dans le texte une suite article-adjectif-nom. Un analyseur opérant en largeur cherche un syntagme nominal (SN), et pour le trouver, cherche tout d'abord un article, puis un adjectif, et enfin un nom.

D'une manière plus formelle, l'analyseur ascendant exécute les étapes suivantes :

1. Initialiser la liste des règles de réécriture actives. Une règle active est une règle dont les membres de droite n'ont pas tous été trouvés dans le texte.

2. Pour chaque mot du texte:

2.1. Initialiser la liste des symboles actifs. Un symbole actif est la catégorie syntaxique d'un mot ou le membre de droite d'une règle de réécriture venant d'être identifié.

2.2. Déterminer les classes syntaxiques auxquelles le symbole peut appartenir (nom, adjectif, etc.).

2.3. Ajouter chaque classe à la liste des symboles actifs.

2.4. Accomplir pour chaque symbole actif de la liste les opérations suivantes:

2.4.1. Chercher dans la liste des règles de la grammaire celles dont le premier symbole correspond au symbole actif. Insérer celles-ci dans la liste des règles actives.

2.4.2. Chercher dans la liste des règles actives celles dont le prochain symbole correspond au symbole actif. Les faire passer au prochain symbole puis les insérer à nouveau dans la liste des règles actives.

2.4.3. Chercher parmi les règles actives celles qui ont épuisé tous leurs symboles, et effectuer leur réduction, c'est-à-dire insérer dans la liste des symboles actifs le membre de gauche dans la définition de la règle de réécriture.

3. L'analyse a réussi si le symbole maximal (S) se trouve dans la liste des règles actives et que tous les symboles des règles correspondantes ont été utilisés.

L'exemple suivant illustre ce mécanisme. Considérons la phrase *Les jeunes enfants aiment le chocolat*, ainsi que la grammaire et le lexique suivants:

Grammaire:

```
1.  S   →  SN  SV
2.  SN  →  Art  N
3.  SN  →  Art  Adj  N
4.  SV  →  V  SN
```

Lexique:

```
les          Art
le           Art
enfants      N
jeunes       N | Adj
aiment       V
chocolat     N
```

Chaque élément de la liste des règles de réécriture actives contient quatre valeurs: le numéro de la règle, un numéro indiquant le nombre de symboles utilisés et le domaine d'application de la règle (une valeur pour le début, une valeur pour la fin). Le nombre d'enregistrements contenus dans cette liste est remis à zéro lors de l'étape 1.

Lors de l'étape 2.1, le nombre de symboles actifs est remis à zéro et le prochain mot est lu. Dans notre exemple, il s'agit de *les*.

Etape 2.2 : une consultation du lexique fournit la classe 'Art' pour ce mot.

Etape 2.3 : la liste des symboles actifs est initialisée avec cette valeur.

Etape 2.4.1 : on extrait le premier symbole actif de la liste; il s'agit de 'Art'. La grammaire contient deux règles de réécriture qui commencent par ce symbole. On insère donc deux éléments dans la liste des règles actives:

```
1- règle=2, pos=1, [0,1]   SN → Art * N
2- règle=3, pos=1, [0,1]   SN → Art * Adj N
```

(L'étoile indique la position courante à l'intérieur de la règle de réécriture. Le domaine [0,1] indique que la règle commence avant le premier mot et se trouve actuellement après le premier mot).

Bien qu'étant actifs, ces éléments ne sont pas considérés pour l'étape 2.4.2 car ils viennent d'être créés.

Aux étapes 2.4.2 et 2.4.3, aucun élément ne satisfait aux conditions.

Nous aurons donc dans la liste des règles actives les éléments suivants :

```
1- règle=2, pos=1, [0,1] SN → Art * N
2- règle=3, pos=1, [0,1] SN → Art * Adj N
```

Le prochain mot est ensuite lu ; il s'agit de *jeunes*, qui peut représenter soit un adjectif (Adj), soit un nom (N). La liste des symboles actifs est alors initialisée avec ces deux classes. Aucune règle de réécriture de la grammaire ne commençant avec ces symboles, les étapes 2.4.1 pour les deux classes seront vides. Par contre, lors de l'étape 2.4.2, on remarque que la règle active (1) attend N comme prochain symbole. On ajoute donc l'élément suivant à la liste :

```
3- règle=2, pos=2, [0,2]   SN → Art N *
```

L'étape 2.4.3 détermine que la règle (2) n'attend plus de symboles, et ajoute donc à la liste des symboles actifs la classe SN.

Les étapes 2.4.2 et 2.4.3 se comportent de manière similaire pour 'Adj', ce qui donnera les listes suivantes:

règles:

```
1- règle=2, pos=1, [0,1] SN → Art * N
2- règle=3, pos=1, [0,1] SN → Art * Adj N
```

règles actives:

```
3- règle=2, pos=2, [0,2] SN → Art N *
4- règle=3, pos=2, [0,2] SN → Art Adj * N
```

symboles actifs:

```
SN [0,2].
```

(les règles (1), (2) ne sont plus actives parce qu'elles ont été remplacées par (3) et (4))

Le symbole SN ayant été ajouté à cette dernière liste, il faut le soumettre aux étapes 2.4.1 à 2.4.3, ce qui produira un nouvel élément :

```
5- règle=1, pos=1, [0,2]   S → SN * SV
```

Avec le prochain mot, *enfants*, on peut faire passer la règle de réécriture (4) au prochain symbole et donc insérer une nouvelle règle active:

règles actives:

```
5- règle=1, pos=1, [0,2]   S → SN * SV
6- règle=3, pos=3, [0,3]   SN → Art Adj N *
```

symboles actifs:

```
SN [0,3]
```

La règle 2.4.2 détermine que la règle (1) commence par ce symbole, ce qui ajoute l'élément:

```
7- règle=1, pos=1, [0,3]   S → SN * SV
```

On voit ainsi que la liste des règles actives peut contenir plus d'une instance de la même règle. Dans notre cas, la règle active en (5) attend un SV commençant au troisième mot alors que la règle active en (7) en attend un commençant au quatrième mot.

Le traitement du quatrième et du cinquième mot (*aiment* et *le*) n'ajoute que les règles actives suivantes à la liste :

```
 8- règle=4, pos=1, [3,4]   SV → V * SN
 9- règle=2, pos=1, [4,5]   SN → Art * N
10- règle=3, pos=1, [4,5]   SN → Art * Adj N
```

Les choses se compliquent avec le dernier mot, *chocolat*. Tout d'abord, la règle active (9) utilise le nom, ce qui n'est pas le cas pour (10). Nous avons donc:

règles actives:

```
5- règle=1, pos=1, [0,2]   S → SN * SV
7- règle=1, pos=1, [0,3]   S → SN * SV
```

```
 8- règle=4, pos=1, [3,4]   SV → V * SN
11- règle=3, pos=2, [4,6]   SN → Art N *
```

symboles actifs :

```
SN [4,6]
```

A ce moment-là, il est possible de faire passer la règle active (8) au prochain symbole car elle attend un SN commençant après le quatrième mot. Cette opération créé une nouvelle règle et un nouveau symbole actif :

```
12- règle=4, pos=2, [3,6]   VP → V NP * SV [3,6]
```

La liste des règles actives montre que deux règles de réécriture, (5) et (7), attendent un SV. Le domaine du nouveau SV correspond avec le domaine attendu en (7) mais pas en (5). La seule solution consiste alors à faire passer (7) au prochain symbole pour obtenir

```
13- règle=1, pos=2, [0,6]   S → SN SV *
```

Cette règle active satisfait les conditions nécessaires à la réalisation de l'étape 3, et la phrase est acceptée. Les règles de réécriture suivantes permettent de constituer l'arbre syntaxique :

```
13- règle=1, pos=2, [0,6]   S → SN SV *
12- règle=4, pos=2, [3,6]   SV → V SN *
11- règle=3, pos=2, [4,6]   SN → Art N *
 6- règle=2, pos=3, [0,3]   SN → Art Adj N *
```

Dans le modèle informatique, chaque règle active contient en plus un pointeur vers les règles actives correspondant à ses symboles. Dans notre exemple, on aura:

```
13-  S   → SN(6)   SV(12)
12-  SV  → V(-)    SN(11)
11-  SN  → Art(-)  N(-)
 6-  SN  → Art(-)  Adj(-)  N(-)
```

3.4.4. L'analyse fonctionnelle

L'analyse fonctionnelle accepte en entrée une liste d'arbres produits par l'analyse syntaxique, chaque arbre correspondant à une interprétation possible du texte. Les feuilles de ces arbres représentent les noeuds terminaux des c-structures, c'est-à-dire des éléments lexicaux. Les données produites par l'analyse fonctionnelle consistent en un ensemble de f-structures, auxquelles sont attachées des listes d'erreurs découvertes lors de l'unification des données fonctionnelles. L'unification exécute la combinaison et la comparaison des attributs des différents mots ou groupes de mots contenus dans une phrase. C'est lors de cette opération que les règles telles que celle de l'accord du verbe avec son sujet sont appliquées. Kaplan et Bresnan (1982) décrivent un algorithme pour l'unification des f-structures qui opère de la façon suivante :

1. Le système construit une liste d'équations fonctionnelles à partir des annotations des c-structures et des annotations lexicales. Dans chaque annotation, les métavariables ↑ et ↓ sont remplacées par des numéros de f-structures, f1,f2...fn.

2. La f-structure principale (celle de la phrase entière) est construite pas à pas en unifiant les fx.

Cette méthode ne convient toutefois pas très bien à l'analyse de phrases contenant des erreurs, car des opérations particulières doivent être effectuées lorsqu'une erreur est détectée dans le texte lors de l'unification. Prenons l'exemple de la phrase **Tu aime de dessiner des fleurs*. Cette phrase contient deux erreurs, la première étant le mauvais accord entre le sujet et le verbe et la seconde l'utilisation de la préposition *de* dans la subordonnée. La forme *aime* du verbe *aimer* peut représenter un cas parmi cinq présents:

1- la première personne du singulier du présent.
2- la troisième personne du singulier du présent.
3 et 4- idem pour le subjonctif.
5- la deuxième personne du singulier de l'impératif.

Le verbe *aimer* possède également au moins trois formes de sous-catégorisation :

```
1- 'aimer<(↑SUBJ)(↑OBJ)>'        (il aime les fleurs)
2- 'aimer<(↑SUBJ)(↑COMP)>'       (il aime que tu dessines)
3- 'aimer<(↑SUBJ)(↑XCOMP)>'      (il aime dessiner)
```

Parmi ces quinze possibilités, aucune ne reflète la structure de la phrase. La seule solution que l'analyseur possède dans ce cas est de traiter toutes ces possibilités séparément, et de ne considérer à la fin que la meilleure. Ainsi, même si l'unification détecte des valeurs incompatibles pour certains attributs, l'analyse de chaque possibilité doit continuer jusqu'au bout afin que chacune puisse être évaluée. Pour tenir compte de ces exigences, un nouvel algorithme d'unification a été développé. Celui-ci opère en trois étapes successives, dont voici la description, suivie d'un exemple d'application de l'algorithme.

Première étape

L'analyseur parcourt tous les arbres syntaxiques de haut en bas et construit pour chacun d'eux un réseau préliminaire de structures fonctionnelles. Le processus commence au noeud maximal de chaque arbre (S, par exemple). D'après les spécifications de l'analyseur syntaxique, ce noeud correspond au symbole de gauche d'une règle de réécriture. Tous les sous-arbres qui correspondent aux symboles à droite de la règle sont alors parcourus les uns après les autres. Avant d'entrer dans un sous-arbre, l'analyseur exécute toutes les annotations attachées au symbole de la règle. Lorsque le symbole de la règle est un noeud terminal, ses annotations sont exécutées et introduites dans la f-structure. Si l'exécution des annotations entraîne une erreur d'unification, cette erreur est enregistrée dans une table liée à la f-structure en question. Dans le cas où plusieurs possibilités s'offrent, comme pour un verbe ayant plus d'une forme de sous-catégorisation, les différentes formes ne sont pas insérées dans la f-structure, mais une liste de toutes les valeurs possibles est attachée à cette f-structure. Cette liste sera traitée lors de la deuxième étape. Notons qu'il est souvent nécessaire de parcourir les arbres syntaxiques plus d'une fois lors de cette première étape, car certaines annotations peuvent faire référence à des attributs ou des f-structures qui n'existent pas encore.

Deuxième étape

L'analyseur parcourt les f-structures produites lors de la première étape pour déterminer les ambiguïtés qui s'y trouvent. Celles-ci sont introduites dans une liste de pointeurs attachée à chacune de ces solutions. Les ambiguïtés possibles sont les suivantes :

- L'analyse syntaxique donne plusieurs solutions.
- Un mot peut avoir plusieurs formes de sous-catégorisation.
- Un mot peut avoir plusieurs catégories syntaxiques (par exemple, *ferme* peut être un nom, un adjectif ou un verbe).
- Un mot peut avoir plusieurs formes à l'intérieur d'une classe (par exemple, *ferme* peut être la première ou la troisième personne du singulier du présent ou du subjonctif, ou l'impératif).

Troisième étape

La f-structure établie lors de la première étape est répétée pour chaque combinaison possible des ambiguïtés. Les attributs des éléments ambigus sont ensuite introduits (unifiés) dans cette f-structure au niveau où ils appartiennent. On obtient ainsi autant de solutions qu'il y a de combinaison de ces ambiguïtés. Là aussi, il se peut que l'introduction de nouveaux attributs dans la f-structure provoque des erreurs. Celles-ci sont alors ajoutées à la liste attachée à la f-structure créée lors de la première étape.

Voyons ces trois étapes plus en détail à l'aide de l'exemple de la phrase ci-dessous, ainsi que de la grammaire et du lexique qui y sont associés.

Les **belle ferme le voile.*

Grammaire:

```
S  ->      SN                 SV
        (↑SUBJ)=↓             ↑=↓
   (↑NOMBRE)=(↓NOMBRE)
 (↑PERSONNE)=(↓PERSONNE)

SN ->      Art  N

SN ->      Art  Adj  N

SV ->      V'       (SN)
                  (↑OBJ)=↑

V' ->     (CL)          V
       (↑(↓CAS))=↓
```

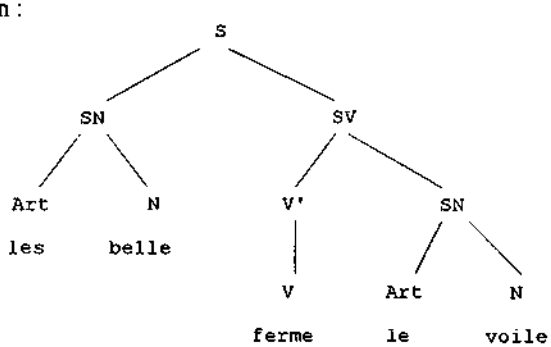
Lexique:

```
la          Art     (↑GENRE) = MASCULIN
                    (↑NOMBRE) =r SINGULIER

le          Art     (↑GENRE) = MASCULIN
                    (↑NOMBRE) =r SINGULIER

le          CL      (↑GENRE) = MASCULIN
                    (↑NOMBRE) = SINGULIER
                    (↑CAS)=OBJ
                    (↑PRED)='Pro'

les         Art     (↑NOMBRE) =r PLURIEL

belle       Adj     (↑GENRE) = FEMININ
                    (↑NOMBRE) = SINGULIER

belle       N       (↑GENRE) =r FEMININ
                    (↑NOMBRE) = SINGULIER
                    (↑PERSONNE) =r TROISIEME
                    (↑PRED) = 'belle'

ferme       V       (↑NOMBRE) = SINGULIER
                    (↑PERSONNE) = PREMIERE
                    (↑MODE) = INDICATIF
                    (↑TEMPS) = PRESENT
                    (↑PRED) = 'fermer<(↑SUBJ)(↑OBJ)>'

ferme       V       (↑NOMBRE) = SINGULIER
                    (↑PERSONNE) = TROISIEME
```

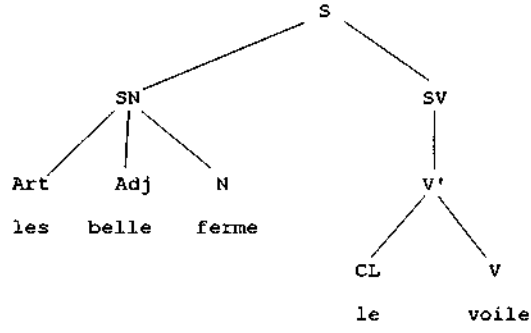
```
                        (↑MODE) = INDICATIF
                        (↑TEMPS) = PRESENT
                        (↑PRED) = 'fermer<(↑SUBJ)(↑OBJ)>'

ferme       N           (↑NOMBRE) =r SINGULIER
                        (↑GENRE) = FEMININ
                        (↑PERSONNE) =r TROISIEME
                        (↑PRED) = 'ferme'

voile       V           (↑NOMBRE) = SINGULIER
                        (↑PERSONNE) = PREMIERE
                        (↑MODE) = INDICATIF
                        (↑TEMPS) = PRESENT
                        (↑PRED) = 'voiler<(↑SUBJ)(↑OBJ)>'

voile       V           (↑NOMBRE) = SINGULIER
                        (↑PERSONNE) = TROISIEME
                        (↑MODE) = INDICATIF
                        (↑TEMPS) = PRESENT
                        (↑PRED) = 'voiler<(↑SUBJ)(↑OBJ)>'

voile       N           (↑NOMBRE) = SINGULIER
                        (↑GENRE) =r MASCULIN
                        (↑PERSONNE) =r TROISIEME
                        (↑PRED) = 'voile'
```

L'analyse syntaxique produit les arbres suivants, qui sont au nombre de deux pour notre exemple.

1ère solution :



2ème solution :



Chaque interprétation de cette phrase contient des erreurs. Dans la première, *belle* devrait être au pluriel et *ferme* à la troisième personne du pluriel (si l'on accepte les priorités définies dans la section 3.3.2.a). Dans la seconde, *belle* (adjectif) et *ferme* (nom) devraient être au pluriel et *voile* à la troisième personne du pluriel. Voyons comment l'analyseur découvre ces erreurs et considérons les deux interprétations séparément. La seule ambiguïté dans la première interprétation concerne le verbe *ferme*, qui peut avoir la première ou la troisième personne du singulier. La première étape de l'analyse produit ainsi les f-structures suivantes:

```
f4  [ NOMBRE      SINGULIER                    ]
    | PERSONNE    PREMIERE                     |
    | MODE        INDICATIF                    |
    | TEMPS       PRESENT                      |
    [ PRED        'fermer<(↑SUBJ)(↑OBJ>'       ]

f5  [ NOMBRE      SINGULIER                    ]
    | PERSONNE    TROISIEME                    |
    | MODE        INDICATIF                    |
    | TEMPS       PRESENT                      |
    [ PRED        'fermer<(↑SUBJ)(↑OBJ>'       ]
```

```
f1   [ SUBJ      f2   [ NOMBRE     *PLURIEL  ]
                        GENRE      FEMININ
                        PERSONNE   TROISIEME
                        PRED       'belle'   ]

       OBJ       f3   [ NOMBRE     SINGULIER ]
                        GENRE      MASCULIN
                        PERSONNE   TROISIEME
                        PRED       'voile'   ]

       NOMBRE          PLURIEL
       PERSONNE        TROISIEME

       AMBIGUITES      f4,f5                 ]
```

(Nous remarquons la présence de l'attribut AMBIGUITES dans la f-structure f1. Les valeurs associées à cet attribut, f4 et f5, représentent des f-structures qui, lorsqu'elles sont séparément unifiées avec f1, produisent deux solutions différentes de l'analyse)

De plus, une erreur a été enregistrée :

Dans f2, l'attribut NOMBRE a plus d'une valeur.

Le seul objectif de la deuxième étape étant de relever les ambiguïtés, on peut passer à la troisième étape, qui tente d'unifier toutes ces ambiguïtés dans la f-structure principale. Cette opération engendre d'autres erreurs :

Pour f4:

Dans f1, l'attribut PERSONNE a plus d'une valeur.

Dans f1, l'attribut NOMBRE a plus d'une valeur.

Pour f5:

Dans f1, l'attribut NOMBRE a plus d'une valeur.

Considérons maintenant la seconde interprétation de la phrase. La première étape de l'analyse fournit les f-structures décrites ci-dessous:

```
f4  [ NOMBRE      SINGULIER                    ]
    [ PERSONNE    PREMIERE                     ]
    [ MODE        INDICATIF                    ]
    [ TEMPS       PRESENT                      ]
    [ PRED        'voiler<(↑SUBJ)(↑OBJ>'       ]

f5  [ NOMBRE      SINGULIER                    ]
    [ PERSONNE    TROISIEME                    ]
    [ MODE        INDICATIF                    ]
    [ TEMPS       PRESENT                      ]
    [ PRED        'voiler<(↑SUBJ)(↑OBJ>'       ]

f6  [ SUBJ   f7   [ NOMBRE     *PLURIEL ]      ]
    [             [ GENRE      FEMININ  ]      ]
    [             [ PERSONNE   TROISIEME]      ]
    [             [ PRED       'ferme'  ]      ]
    [                                           ]
    [ OBJ    f8   [ NOMBRE     SINGULIER]      ]
    [             [ GENRE      MASCULIN ]      ]
    [             [ PERSONNE   TROISIEME]      ]
    [             [ PRED       'Pro'    ]      ]
    [                                           ]
    [ NOMBRE      PLURIEL                       ]
    [ PERSONNE    TROISIEME                     ]
    [                                           ]
    [ AMBIGUITES  f9, f10                       ]
```

Une erreur a été enregistrée :

Dans f7, l'attribut NOMBRE a plus d'une valeur.

L'unification des f-structures f9 et f10 engendre d'autres erreurs:

Pour f9:

Dans f6, l'attribut PERSONNE a plus d'une valeur.

Dans f6, l'attribut NOMBRE a plus d'une valeur.

Pour f10:

Dans f6, l'attribut NOMBRE a plus d'une valeur.

L'analyse fonctionnelle de cette phrase produit ainsi le tableau des erreurs suivant:

Solution	Arbre No	Ambiguité No	Erreurs	
1	1	1	3	NOMBRE (f2) PERSONNE (f1) NOMBRE (f1)
2	1	2	2	NOMBRE (f2) NOMBRE (f1)
3	2	1	3	NOMBRE (f6) PERSONNE (f5) NOMBRE (f5)
4	2	2	2	NOMBRE (f6) NOMBRE (f5)

Ceci conclut l'analyse fonctionnelle de la phrase.

3.4.5. La correction

La correction s'effectue en trois étapes : la détection d'erreurs, l'identification de la cause de ces erreurs, et enfin l'édition du texte pour modifier les constructions

et les mots erronés. Il est bien clair que l'édition du texte ne peut se faire automatiquement dans un correcteur 'réel'. Il a toutefois été choisi de mettre en oeuvre cette opération pour certaines erreurs (voir section 4.2.), afin de démontrer qu'il existe des cas où le logiciel connaît assez de données sur le texte pour le corriger lui-même. Lors de la première étape, les solutions produites par l'analyse fonctionnelle sont classées d'après la priorité affectée par le linguiste (au moment du développement des règles) aux attributs sur lesquels l'unification a échoué. Par exemple, l'attribut PRED possède la priorité la plus élevée et l'attribut COD-POSITION la priorité la plus basse. La meilleure de ces solutions est choisie pour être traitée par les deux étapes suivantes. S'il existe une solution sans erreur, le logiciel signale le succès de l'analyse et il s'arrête. La deuxième étape consiste à détecter la source des erreurs, c'est-à-dire à trouver quel mot ou construction fausse a déclenché l'erreur. Ceci se fait principalement à l'aide des valeurs de référence que le linguiste donne aux attributs dans le lexique et dans les annotations aux c-structures. (Par exemple, le genre et le nombre du sujet représentent la valeur de référence par rapport au genre et au nombre du verbe. Voir l'annexe 1 pour plus de détails.) Une fois la source de l'erreur détectée, la troisième étape consiste à modifier le texte soumis en remplaçant, supprimant ou insérant des mots.

Voyons ces étapes plus en détail et examinons d'abord les cas où les erreurs d'unification peuvent surgir.

1. Deux attributs ne sont pas compatibles (le GENRE d'un mot ne correspond pas au GENRE d'un autre).
2. Un attribut requis (=c) n'est pas défini (voir la section sur la LFG).

3. La sous-catégorisation de l'élément de tête ne remplit pas les conditions de bonne formation.
4. L'attribut spécial AUTOMATIC_ERROR se trouve dans la f-structure. Celui-ci peut être introduit n'importe où par le linguiste pour indiquer que l'utilisation d'une certaine construction est fausse.

Dans chacun de ces cas, la définition de l'erreur se résume à la spécification de l'attribut qui l'a causée et à la f-structure dans laquelle elle apparaît. L'analyse fonctionnelle produit ainsi les données suivantes pour chaque interprétation du texte :

<nombre d'erreurs>
Erreur 1 : <nom de l'attribut> <numéro de la f-structure>
Erreur 2 : <nom de l'attribut> <numéro de la f-structure>
...

La classification utilise une liste d'attributs classés suivant leur priorité pour évaluer les différentes interprétations. Chaque solution est parcourue et l'attribut causant l'erreur qui a la plus haute priorité est gardé. La meilleure solution est celle qui possède l'attribut avec la priorité la plus basse. La priorité d'un attribut est définie par sa position à l'intérieur de la liste suivante:

PRED, OBJ, OBJ2, SUBJ, COMP, XCOMP, COMP-PREP,
ADJ-POSITION, PREP-NOMBRE, PREP-GENRE,
PERSONNE, NOMBRE, GENRE, PP_NOMBRE, PP_GENRE,
MODE, MODEV, TEMPS, AUTOMATIC_ERROR, COD-POSITION.

Ainsi, PRED a la priorité la plus haute et COD-POSITION la plus basse.

Considérons la phrase incorrecte **Il a manger le pomme.* L'analyseur fonctionnel fournit deux solutions avec les erreurs qui leur correspondent :

Solution 1 :	f-structure 9-0	MODEV
	f-structure 9-5	GENRE
Solution 2 :	f-structure 12-0	MODEV
	f-structure 12-5	GENRE
	f-structure 12-12	PRED

Soit l'ordre des priorités suivant, de la plus haute à la plus basse : PRED - GENRE - MODEV. Dans la première solution, l'attribut ayant la plus haute priorité est GENRE et dans la seconde il s'agit de PRED. De ces deux attributs, c'est GENRE qui a la priorité la plus basse, ce qui fait que la première solution est choisie pour la phase suivante.

La deuxième phase consiste à chercher dans les f-structures contenant une erreur la source même de cette erreur. Ceci se fait à l'aide des règles de correction décrites plus haut. Enfin, chaque règle de correction dont les conditions sont remplies appelle une procédure de correction qui va effectuer les modifications dans le texte. Contrairement à la première phase de la correction, qui est effectuée par le logiciel même, les deux dernières phases sont entièrement sous le contrôle du linguiste, qui spécifie les règles et les procédures de correction.

Les annexes 1 et 2 présentent quelques extraits de données linguistiques qui figurent dans le logiciel LFG ainsi qu'un exemple d'une session interactive avec celui-ci.

Conclusion

Nous avons montré dans ce chapitre que l'utilisation de la LFG comme formalisme grammatical et d'un analyseur ascendant pour l'analyse automatique formaient une bonne base pour un logiciel d'identification et de correction d'erreurs produites à l'écrit en français par des utilisateurs anglophones. Le champ d'application de ce logiciel reste toutefois limité, car seuls les textes dont tous les mots sont connus par le lexique et dont la structure obéit aux règles de réécriture de la grammaire peuvent être traités. Si l'on considère en outre la quantité de données requise par le lexique et les règles de réécriture pour qu'elles couvrent une partie substantielle de la langue, il paraît peu probable qu'un logiciel reposant entièrement sur les principes de la LFG et destiné au grand public puisse un jour être réalisé.

On peut toutefois concevoir la possibilité d'utiliser un dictionnaire informatique commercial contenant un sous-ensemble de l'information requise par la LFG. Un autre dictionnaire, spécifique aux erreurs à traiter, pourrait alors compléter le premier afin d'introduire davantage de contraintes et donc identifier certaines erreurs. Cette alternative fera certainement l'objet de recherches supplémentaires dans un proche avenir, car il a été démontré ici qu'une analyse du texte au niveau fonctionnel permettait l'identification d'un grand nombre d'erreurs, spécialement dans les cas où

deux éléments dépendants d'une phrase sont séparés par plusieurs mots (accord du participe passé, par exemple).

La spécificité et la taille du lexique et des règles de réécriture ne constituent pas le seul problème du logiciel. Nous avons vu que le nombre de solutions produites par l'analyse pouvait devenir très grand, ce qui va poser des difficultés au logiciel sans que son importance soit mesurable immédiatement. D'une part, le logiciel devra avoir la capacité de contenir toutes ces solutions en mémoire et il devra être assez rapide pour que l'utilisateur ne s'ennuie pas. D'autre part, il est possible que la multiplication des solutions rende le problème trop complexe, et qu'il n'existe pas de moyen de différencier et d'évaluer toutes ces solutions. Ce phénomène n'a toutefois pas encore pu être examiné, ceci par manque d'informations dans le lexique et également à cause de l'utilisation de données sous forme non compressée qui exigent une quantité de mémoire atteignant les limites de l'ordinateur sur lequel le logiciel a été développé.

Le chapitre qui suit présente l'évaluation du logiciel basé sur la LFG. Dans celle-ci, des phrases-test écrites par une personne neutre ont été utilisées pour déterminer les qualités et les défauts de l'approche proposée.

Chapitre 4

Evaluation du correcteur basé sur le formalisme LFG

L'évaluation du logiciel LFG avait comme but de répondre aux questions suivantes: premièrement, est-ce que le logiciel détecte et corrige les erreurs pour lesquelles il est programmé? Deuxièmement, dans quelle mesure peut-on généraliser les mécanismes mis en oeuvre dans le logiciel et les utiliser pour détecter et corriger une grande variété d'erreurs? Plutôt que de porter sur un grand nombre de types d'erreurs identifiées grâce à une typologie complète, l'évaluation s'est faite sur une quantité limitée d'erreurs syntaxiques et morpho-syntaxiques jugées idéales à la démonstration des capacités du logiciel. Nous avons ainsi constitué un corpus d'évaluation ne contenant que des erreurs appartenant à certaines catégories pré-définies,

comme par exemple l'accord du participe passé et l'utilisation de certaines prépositions.

Les résultats intrinsèques de l'évaluation (les taux de détection et de non-détection) ont permis d'établir si les mécanismes mis en place permettaient en effet de détecter et corriger les erreurs des différentes catégories. L'analyse des causes de malfonctionnement du logiciel, quant à elle, a mis en évidence l'apport, positif ou négatif, de chaque composante. Cette analyse approfondie des résultats a permis d'identifier les limites de l'utilisation d'un formalisme grammatical comme la LFG pour la correction automatique d'erreurs.

Ce chapitre débute par une présentation du processus d'affinage des algorithmes que nous avons développés (section 4.1.). Il s'ensuit une description des erreurs sur lesquelles porte l'évaluation et des règles de détection et de correction développées (section 4.2.), ainsi qu'un aperçu du processus d'évaluation (section 4.3.). Nous terminons le chapitre par une analyse des résultats, suivie d'une discussion plus approfondie des avantages et inconvénients de l'approche LFG (4.4.).

4.1. Affinage des algorithmes

Avant d'entamer l'évaluation même, nous avons exécuté une phase préliminaire, dans laquelle les algorithmes ont été revus et corrigés après avoir été appliqués à un corpus d'affinage. La phase d'affinage des algorithmes est importante pour les logiciels de correction automatique car

elle permet l'identification de problèmes liés aux mécanismes de détection ainsi que l'amélioration de la robustesse du logiciel dans des cas imprévus.

Le corpus utilisé pour l'affinage du logiciel LFG consiste en une douzaine de textes français produits par des enfants anglophones de 14 à 15 ans, qui ont suivi 8 années dans une école au Canada anglophone où l'enseignement se donne uniquement en français dans toutes les branches (programme d'immersion). L'ensemble des textes compte environ 3500 mots.

Avant de pouvoir être soumis au logiciel LFG, ces textes ont tout d'abord dû être épurés des fautes d'orthographe qu'ils contenaient. Ceci a été fait à l'aide du vérificateur orthographique du logiciel Wordperfect. Le but de cette opération n'était pas de corriger chaque mot, mais plutôt de n'avoir dans les textes que des mots épelés correctement. Nous avons par exemple remplacé **imposible* par *impossible*, mais pas *arrive* par *arrivent* lorsque ce verbe devait être à la troisième personne du pluriel. Nous avons ainsi simulé les opérations d'un correcteur orthographique standard. Après cette phase de pré-traitement il restait environ 400 erreurs syntaxiques et morpho-syntaxiques dans ces textes. La phase suivante a consisté à mettre en oeuvre dans le logiciel les mécanismes de détection et de correction pour un certain nombre de catégories d'erreurs apparaissant fréquemment dans le corpus d'affinage. Nous avons ensuite soumis le corpus au logiciel et effectué les modifications suivantes:

1. Ajustement des règles syntagmatiques.
2. Insertion de nouveaux traits dans le lexique.

3. Amélioration des algorithmes d'analyse, surtout en ce qui concerne la quantité de données conservées simultanément en mémoire.
4. Dans certains cas, les suggestions de corrections ont été remplacées par de simples mises en garde mettant en évidence le mot présumé erroné.

La phase d'affinage a en outre permis d'obtenir quelques renseignements intéressants sur les caractéristiques du logiciel:

a. Malgré le fait que le logiciel ait été programmé pour traiter des difficultés contenues dans le corpus d'affinage, seule la moitié des erreurs syntaxiques et morpho-syntaxiques a pu être détectée. De plus, ceci est sans compter les opérations effectuées lors de la phase préliminaire, où les mots ayant une mauvaise orthographe furent corrigés manuellement. Il est facile d'imaginer que la présence de telles fautes lors d'un traitement basé sur une analyse complète du texte (comme c'est le cas ici) diminuerait de beaucoup le nombre de détections justes.

b. Le logiciel semble assez robuste, dans la mesure où même des phrases contenant un nombre impressionnant d'erreurs ont pu être traitées et entièrement corrigées.

4.2. Catégories d'erreurs et règles de détection

Cette section décrit les 11 catégories d'erreurs que nous avons choisies pour l'évaluation. Nous présentons, pour chaque catégorie, les règles

grammaticales liées au type d'erreur traité et les algorithmes de détection et de correction que nous avons mis en oeuvre. Nous illustrons ces algorithmes à l'aide des informations suivantes selon les cas: entrées lexicales, règles syntagmatiques, règles lexicales, règles et procédures de correction.

La liste ci-dessous indique les 11 catégories d'erreurs:

1. Accord sujet-verbe
2. Accord du syntagme nominal
3. Participe passé: confusion entre infinitif et participe passé
4. Participe passé: accord avec 'être'
5. Participe passé: accord avec 'avoir'
6. Mode dans les propositions
7. Structure des compléments du verbe
8. Position des adjectifs
9. Les prépositions 'à' et 'de': préposition + infinitif
10. Les prépositions 'à' et 'de': confusion entre 'a' et 'à'
11. Les prépositions 'à' et 'de': préposition + 'le', 'la', 'les'

La détection d'erreurs dans le texte se fait toujours suivant la même méthode, qui consiste à vérifier les trois conditions de bonne formation des structures fonctionnelles (voir chapitre 3). En pratique, ceci se résume à indiquer une erreur dans deux cas:

1. Un trait a des valeurs ambiguës (condition d'unicité).

2. Il n'y a pas de correspondance directe entre les fonctions grammaticales requises par la sous-catégorisation du verbe et celles qui se trouvent dans la structure fonctionnelle (celle-ci n'est ni complète ni cohérente).

Seule la catégorie qui traite de la structure des compléments utilise le second critère de détection. Toutes les autres se basent sur des traits ayant des valeurs multiples et différentes.

Deux possibilités s'offrent pour la correction: soit un message de mise en garde est affiché, soit une correction est effectuée. Dans ce second cas, la procédure de correction va rechercher dans le lexique la forme correcte du mot contenant l'erreur. Puis elle remplace la valeur du trait ayant provoqué l'erreur par la valeur de référence obtenue dans la structure fonctionnelle.

4.2.1. **Accord sujet-verbe** (Accord S-V)

a. Description

Cette catégorie traite des erreurs d'accord du verbe avec son sujet en genre et en nombre, comme dans les exemples suivants:

* *Je ne **peut** pas prendre l'auto.*
* *La fille qui regarde les fleurs **n'aiment** pas le soleil.*

La forme impérative du verbe n'est pas traitée.

b. Algorithmes de détection

La détection d'erreurs d'accord entre le sujet et le verbe se fait à l'aide des traits NOMBRE et PERS. Ceux-ci sont projetés de la f-structure du sujet vers celle de la proposition, par l'entremise des annotations à la règle syntagmatique suivante:

```
S  ->      SN                  SV
          (↑SUBJ)=↓            ↑=↓
       (↑NOMBRE)=(↓NOMBRE)
         (↑PERS)=(↓PERS)
```

L'équation de droite (↑=↓) indique que toutes les informations du syntagme verbal sont projetées au niveau supérieur. Ces informations comprennent les traits NOMBRE et PERS du verbe principal. Une erreur est détectée en cas de conflit entre la valeur des traits provenant du sujet et du verbe principal. Etant donné que le logiciel enregistre la provenance de chaque trait, il peut appliquer la valeur requise par le verbe au(x) mot(s) erroné(s) dans la f-structure du sujet.

La procédure de correction utilisée est commune à toutes les erreurs relatives aux traits morphologiques. Elle fait appel à deux fonctions, AUTOMATIC_CORRECTION et PRINT_ALL_SOLUTIONS, qui accomplissent les opérations suivantes:

1. Le trait ne satisfaisant pas à la condition d'unicité est identifié dans les f-structures.
2. La valeur de référence du trait est obtenue.

3. Le générateur morphologique applique la valeur de référence à tous les éléments ayant contribué au trait (par unification) et affiche les suggestions de corrections.

4.2.2. Accord du syntagme nominal (Accord SN)

a. Description

Cette catégorie traite des erreurs d'accord en genre et nombre des articles, des adjectifs et des noms à l'intérieur du syntagme nominal. Par exemple:

* *La chien a mangé mon soulier.*

Les nombres complexes (ayant plus d'un terme) ainsi que les participes ne sont pas traités.

b. Algorithmes de détection

Les deux traits entrant en ligne de compte pour cette erreur sont GENRE et NOMBRE. Chaque composante du syntagme nominal, comme les articles, les adjectifs et les noms qu'il contient, introduit séparément ces traits dans la f-structure du syntagme nominal lorsqu'ils sont définis dans l'entrée lexicale correspondante. L'algoritlume de correction est le même que celui décrit dans la section précédente.

4.2.3. Confusion infinitif-participe passé (Infinitif-PP)

a. Description

Cette catégorie traite des confusions entre la forme infinitive et le participe passé des verbes en *-er*. Exemple:

* *Le chien a* ***manger*** *mon soulier.*

b. Algorithmes de détection

La phase d'analyse construit deux types d'arbres syntaxiques pour une phrase du type sujet-verbe1-verbe2-compléments. Le premier type correspond à la forme où 'verbe1' est un auxiliaire et donc 'verbe2' devrait être au participe. Dans le second type, 'verbe1' est le verbe principal de la phrase et 'verbe2' fait partie d'une proposition infinitive (comme dans *Il va manger la pomme*). Lorsque 'verbe1' est un auxiliaire, le trait MODE de 'verbe2' doit être PARTICIPE et l'autre arbre syntaxique est éliminé. Dans le second cas, MODE doit être INFINITIF. Si ces valeurs de MODE ne correspondent pas à celles de 'verbe2', une erreur est détectée et corrigée en allant chercher dans le lexique 'verbe2' au mode de référence.

Comme pour les deux catégories précédentes, la procédure de correction des traits morphologiques est exécutée.

4.2.4. Accord du participe passé avec être (PP-être)

a. Description

Cette catégorie traite des erreurs d'accord du participe passé conjugué avec le verbe être. Par exemple:

* *La petite fille est* ***tombé*** *de l'arbre.*

Les verbes pronominaux sont aussi traités, sauf le cas où le pronom réfléchi est le complément d'objet indirect et que le complément d'objet direct se trouve avant le verbe, comme dans la phrase suivante:

Tu ne peux imaginer les choses que je me suis dites.

b. Algorithmes de détection

Comme nous l'avons vu au chapitre 3, la valeur des traits PP_GENRE et PP_NOMBRE est déterminée en deux temps. Tout d'abord, les traits du participe, présents dans le lexique, sont introduits dans la f-structure lors de l'unification. Ensuite, une règle lexicale est appliquée suivant la nature de l'auxiliaire, *être* ou *avoir*. Cette règle détermine la position du complément d'objet direct si nécessaire et introduit une valeur de référence pour PP_GENRE et PP_NOMBRE dans la f-structure. Considérons, par exemple, le cas des verbes pronominaux aux temps composés ayant un complément d'objet direct, comme *se baigner* et *se rencontrer*. La règle ci-dessous traite de cette situation:

```
REGLE 2.1
conditions:
      AUXILIAIRE = ETRE
      OBJ = *
      OBJ.PRONOM-TYPE = *
actions:
      PP_GENRE =r SUBJ.GENRE
      PP_NOMBRE =r SUBJ.NOMBRE
```

Les trois conditions indiquent que l'auxiliaire doit être *être*, le complément d'objet direct doit être défini (OBJ=*), et qu'il doit s'agir d'un pronom (OBJ.PRONOM-TYPE=*). Les traits OBJ et OBJ.PRONOM-TYPE font partie des règles de réécriture.

Les deux actions ajoutent le genre et le nombre du sujet dans la f-structure.

La correction se fait de la même manière que pour les quatre catégories précédentes.

4.2.5. Accord du participe passé avec avoir (PP-avoir)

a. Description

Cette catégorie traite les cas usuels d'accord du participe passé utilisé avec l'auxiliaire *avoir*, comme dans les exemples suivants:

* *Les enfants ont* ***mangés*** *des pommes.*

* *Il regarde les photos que j'ai* ***pris****.*

b. Algorithmes de détection

La détection et la correction de cette erreur utilisent le même algorithme que la catégorie précédente (PP-être). La règle lexicale ci-dessous permet d'affecter la bonne valeur des attributs PP_GENRE et PP_NOMBRE dans une situation bien précise:

```
REGLE 1.2
conditions:
      AUXILIAIRE = AVOIR
      MODEV = PARTICIPE-PASSE
      COD-POSITION = APRES-VERBE
actions:
      PP_GENRE =r MASCULIN
      PP_NOMBRE =r SINGULIER
```

Cette règle introduit PP_GENRE=MASCULIN et PP_NOMBRE=SINGULIER dans le cas où l'auxiliaire est *avoir* et l'objet direct est situé après le verbe principal. Un conflit entre les deux valeurs de ces traits entraîne une erreur qui est corrigée selon la méthode habituelle, décrite plus haut.

4.2.6. Mode dans les propositions relatives (Mode)

a. Description

Cette catégorie traite des erreurs d'utilisation du subjonctif et de l'indicatif dans les propositions subordonnées complément d'objet direct non infinitives introduites par la conjonction *que*. Exemple:

* *Je veux que vous venez à la maison après le match.*

Les cas non traités comprennent ceux où une des expressions telles que *ne ... que* et *seulement* modifient le mode requis dans la proposition relative.

b. Algorithmes de détection

Chaque verbe ayant une sous-catégorisation comprenant la fonction COMP introduit dans la f-structure de COMP la valeur requise du trait MODE. Considérons l'entrée lexicale de *vouloir*:

```
vouloir     (↑PRED) = 'vouloir<(↑SUBJ)(↑COMP)>'
            (↑COMP MODE) =c SUBJONCTIF
```

La structure verbale à l'intérieur de COMP introduit à son tour le trait MODE dans la f-structure. La condition d'unicité permet ensuite, lors de l'unification, d'indiquer une erreur. La correction se fait d'après la méthode habituelle.

4.2.7. Structure des compléments (Structure)

a. Description

Cette catégorie traite des erreurs de sous-catégorisation des verbes. Exemple:

* *Il téléphone toujours ___ sa soeur.*

b. Algorithmes de détection

Une erreur en relation avec la structure des compléments d'un verbe est détectée lorsqu'il n'y a pas correspondance entre les fonctions requises par la sous-catégorisation et celles présentes dans la f-structure (celle-ci doit être complète et cohérente). Le logiciel ne produit un message d'erreur que lorsque certaines conditions supplémentaires sont remplies. Considérons, par exemple, la phrase suivante:

* *Il la donne un cadeau.*

L'entrée lexicale de *donner* contient l'information suivante:

```
(↑PRED) = 'donner<(↑SUBJ)(↑OBJ)(↑OBJ2)>'
```

La f-structure créée lors de l'analyse de la phrase ci-dessus contient SUBJ et deux fois OBJ (*la* et *un cadeau*). Cette erreur sur le trait PRED déclenche une série de procédures de correction qui opèrent sur d'autres paramètres de la f-structure. Les conditions ci-dessous s'appliquent à l'erreur en question:

```
condition:
      PRED = 'donner'
      OBJ = *2
      OBJ2 = 0
      OBJ.PRED = 'PRO'
exécute:
      PROC011
```

Lorsque le complément d'objet direct est défini deux fois (OBJ=*2), le complément d'objet indirect n'est pas défini et lorsqu'un des objets directs est un pronom, la procédure de correction PROC011 est appelée. Cette procédure a le format suivant:

```
PROCEDURE PROC011
  BUILD_REFERENCE_FRAME ;
  USE OBJ.GENRE ;
  USE OBJ.NOMBRE ;
  USE OBJ.PERSONNE ;
  SET CAS TO DATIF ;
  SET CLASSE TO PRONOM ;
  SET PRONOM-TYPE TO PERSONNEL ;
  FIND_CORRECT_FORM ;
  PRINT CORRECT_FORM ;
  END
```

Les commandes contenues dans cette procédure de correction indiquent que le logiciel doit utiliser le genre, le nombre et la personne de la f-structure du complément d'objet direct (*la*), ainsi que les valeurs spécifiques pour les trois traits CAS, CLASSE et PRONOM-TYPE. Le logiciel doit ensuite rechercher dans le dictionnaire (FIND_CORRECT_FORM) le mot qui satisfait à l'ensemble des conditions (*lui*) et l'afficher (PRINT CORRECT_FORM).

Des algorithmes de correction s'appliquent à d'autres situations qui sont spécifiques au verbe *donner*, comme par exemple pour traiter les erreurs du type **Il donne la fille une pomme*, où le complément n'est pas un pronom. Vu que la transformation nécessaire pour corriger l'erreur (*Il donne une pomme à la fille*) n'est pas morphologique, le logiciel n'affiche qu'un message de mise en garde qui indique que la sous-catégorisation du

verbe semble être erronée. Dans tous les autres cas où une erreur est obtenue avec PRED=*donner* et qu'il n'existe pas de procédure spécifique, le logiciel n'affiche aucun message d'erreur.

4.2.8. Position des adjectifs (Adjectifs)

a. Description

Cette catégorie traite des erreurs de position des adjectifs de couleur et de forme par rapport aux noms à l'intérieur des syntagmes nominaux.
Exemple:

* *Les présidents soupent autour d'une* ***ronde table.***

Notons que le logiciel ne permet qu'un adjectif dans chaque syntagme nominal.

b. Algorithmes de détection

Une erreur en rapport à la position relative d'un nom et d'un adjectif est détectée à l'aide du trait ADJ-POSITION. Ce trait est introduit dans la f-structure de deux manières. D'une part, chaque adjectif qui ne peut être placé, soit avant, soit après un nom, contient dans son entrée lexicale le trait ADJ-POSITION avec la valeur AVANT-NOM ou APRES-NOM. Ce trait est également introduit dans la f-structure par l'intermédiaire de la règle syntagmatique suivante:

```
SN  ->   Art       (Adj)        N        (Adj)
                (↑ADJ-POSITION)        (↑ADJ-POSITION)
                  =AVANT-NOM             =APRES-NOM
```

Lorsque la condition d'unicité sur le trait ADJ-POSITION n'est pas satisfaite, le logiciel affiche une mise en garde.

4.2.9. **Préposition+infinitif** (Prép.+inf.)

a. Description

Cette catégorie traite de la mauvaise utilisation des prépositions *à* et *de* après certains verbes et devant l'infinitif, comme l'illustrent les exemples suivants:

* *Il a clairement refusé à signer la pétition.*
* *Il espère de récupérer son argent.*

b. Algorithmes de détection

Chaque verbe ayant la fonction XCOMP dans sa liste de sous-catégorisation introduit une valeur du trait COMP-PREP dans la proposition infinitive, comme dans l'exemple ci-dessous, qui est extrait du lexique:

```
promettre       (↑PRED) = 'promettre<(↑SUBJ)(↑XCOMP)>'
                (↑XCOMP COMP-PREP) =r PREP-DE
```

La valeur PREP-DE indique que la proposition infinitive doit débuter par *de*, comme dans la phrase *Il promet de venir ce soir*. La préposition *de* introduit également la valeur PREP-DE dans la f-structure lorsqu'elle est utilisée dans ce contexte, alors que la préposition *à* quant à elle introduit la

valeur PREP-A, ce qui permet de détecter une erreur par l'intermédiaire de la condition d'unicité.

L'algorithme de correction lié à COMP-PREP, illustré ci-dessous, détermine tout d'abord la valeur de référence introduite par le verbe puis affiche un message indiquant la préposition correcte qu'il aurait fallu utiliser:

```
GROUPE 10
 condition:
  ATTR_ORDRE = COMP-PREP

REGLE 10.1
 condition:
   COMP-PREP = NO-PREP
 execute:
   PROC10A

REGLE 10.2
 condition:
   COMP-PREP = PREP-A
 execute:
   PROC10B

REGLE 10.3
 condition:
   COMP-PREP = PREP-DE
 execute:
   PROC10C
```

La commande GROUPE indique que les règles qui suivent s'appliquent lorsque la condition d'unicité n'est pas satisfaite pour le trait COMP-PREP. Dans chacune des trois règles 10.1, 10.2 et 10.3, la valeur de

référence introduite par le verbe principal déclenche une procédure de correction spécifique qui affiche un message d'erreur.

4.2.10. Confusion a-à (Conf. à-a)

a. Description

Cette catégorie traite des cas où l'accent est omis de la préposition *à*. Exemple:

* *Il hésite **a** écrire ses mémoires.*

b. Algorithmes de détection

Pour détecter si l'accent a été oublié à tort sur la préposition *à*, le logiciel remplace chaque occurrence de *a* par *à* et détermine si la nouvelle solution est 'meilleure' (produit moins d'erreurs) que sans l'accent. Les traits introduits par la nouvelle forme de *a* sont ceux de *à* plus un trait ERREUR_AUTOMATIQUE. La présence de ce trait force l'appel d'une procédure de correction qui affiche un message indiquant qu'il faut ajouter un accent.

4.2.11. Préposition+article défini (Prép.+art.)

a. Description

Cette catégorie traite des erreurs suivantes:

- *au* suivi de *le*, *la* ou *les*. Exemple:

* *Elle a donné des chocolats **au le** maître.*

- *à* suivi de *le* ou *les*. Exemple:

  * *Elle a donné des chocolats **à les** enfants.*

- *du* suivi de *la* ou *les*.

  * *Ce renard vient **du la** forêt.*

b. Algorithmes de détection

Les prépositions *de*, *du*, *des*, *à*, *au* et *aux* introduisent chacune les traits NOMBRE, GENRE et ACCORD-DU ou ACCORD-A dans la f-structure des syntagmes prépositionnels suivant les cas. La condition d'unicité détermine ensuite, lors de l'unification, si ces traits correspondent à ceux introduits par l'article et le nom qui font partie du syntagme nominal. La procédure de correction habituelle est lancée en cas d'erreur.

La présence du trait ACCORD-A ou ACCORD-DU fait que les procédures de correction appelées sont celles qui traitent des problèmes liés à l'utilisation des prépositions et non pas celles qui corrigent l'accord à l'intérieur du syntagme nominal (catégorie 2), comme le montre l'exemple ci-dessous:

```
GROUPE 4
 condition:
  ATTR_ORDRE = GENRE

REGLE 4.1
 condition:
   ACCORD-AU = OUI
 execute:
   PROC17A
```

La condition initiale isole les erreurs en relation avec le trait GENRE. La condition ACCORD-AU=OUI déclenche, si elle est vérifiée, la procédure de correction PROC17A qui, comme les autres procédures de correction de l'utilisation de la préposition avec l'article défini, affiche une mise en garde indiquant une mauvaise utilisation de la préposition en question.

4.3. Le processus d'évaluation

Le choix d'un corpus d'évaluation est une opération assez complexe. D'une part on peut créer pour l'occasion des phrases dont la structure obéit aux règles grammaticales mises en oeuvre dans le logiciel, comme les règles syntagmatiques ou les annotations lexicales de la LFG. Si cette méthode est largement utilisée pour illustrer certains algorithmes de traitement automatique du langage (analyseurs syntaxiques et sémantiques, par exemple), la nature même de notre problème nous oblige à rechercher le plus possible des situations réalistes, excluant par conséquence ce genre de phrases. L'opposé consiste à choisir des phrases extraites de documents divers, tels que des épreuves d'examens ou des lettres, produits en français par des anglophones. On aurait ainsi pu étendre le corpus d'affinage décrit dans la section précédente. L'avantage principal de ce choix est que ces phrases contiennent une grande variété de mots, de structures de phrases et de cas d'erreurs. L'utilisation de tels textes dans cet environnement limité de manière intentionnelle, formé d'un nombre réduit de règles syntagmatiques et d'erreurs détectables, crée toutefois un certain nombre de problèmes. En particulier, les textes doivent, comme pour le corpus d'affinage, être épurés des erreurs non détectables. C'est-à-dire que les erreurs qui ne sont pas accompagnées de règles de détection dans le

logiciel doivent être identifiées manuellement et corrigées. La nature subjective de cette opération la rend donc peu souhaitable. De plus, il se peut que certaines erreurs détectables par le logiciel ne se produisent que rarement dans les textes choisis, ce qui empêcherait l'évaluation des mécanismes liés à ces erreurs.

De ces deux extrêmes se dégagent un certain nombre de caractéristiques désirables pour le corpus d'évaluation. Celles-ci se retrouvent dans la méthode d'évaluation que nous avons choisie. La procédure consiste à utiliser des phrases qui illustrent chaque type d'erreurs et qui ont été écrites pour l'occasion par une personne n'ayant pas connaissance des règles internes du logiciel. Cette procédure se décompose en étapes successives dont voici les détails.

1ère étape: les types d'erreurs ont été regroupés dans les 11 catégories présentées dans la section 4.2.

2ème étape: nous avons constitué un document pour guider l'auteur du corpus d'évaluation. Ce document comprend une description des tâches à accomplir ainsi qu'une série de contraintes imposées aux phrases-test. Ces contraintes sont de deux sortes:

a- contraintes liées à la structure des phrases

- pas de phrases interrogatives et impératives
- les syntagmes nominaux peuvent contenir:
    - un nom

- un ou deux noms propres (ex. *Robert Leblanc*)
- un article
- un adjectif qualificatif (ex. *rond, bleu*)
- un adverbe de quantité (ex. *très*)
- un adjectif possessif ou démonstratif (ex. *mon, ces*)
- un pronom
- un complément du nom du type syntagme prépositionnel ou pronom relatif suivi d'une proposition) (ex. *La copine de mon frère. La bague que tu as presque achetée*)

b- contraintes liées aux erreurs contenues dans les phrases

Des fiches de travail ont été constituées pour chaque type d'erreurs. Celles-ci contiennent une description du type de l'erreur ainsi que les contraintes qui s'appliquent lors du choix des phrases. Pour chaque erreur on trouve également des exemples de cas traités par le logiciel ainsi que de cas non traités.

Voici l'exemple de la fiche correspondant à la catégorie 3:

```
==========================================

CATEGORIE 3
Participe passé 1: confusion entre infinitif et
participe passé

Description
     Confusion entre la forme infinitive et le
     participe passé des verbes en -er.
```

Cas traités

- Utilisation de l'infinitif au lieu du participe passé.

  * *Il a **refuser** de manger sa soupe.*

- Utilisation du participe passé en '-é' dans une proposition infinitive.

  * *Il a refusé de **mangé** sa soupe.*

Cas non traités

- Lorsque le participe n'est pas accompagné d'un auxiliaire.

  * *Il montrait un regard **égarer**.*

3ème étape: nous avons demandé à une personne francophone de constituer un corpus pour les types de phrases indiquées dans la première étape tout en suivant les contraintes de la deuxième étape (voir Annexe 3 pour les instructions données à cette personne). Pour chaque catégorie d'erreurs, l'auteur du corpus d'évaluation devait choisir 15 phrases réparties comme suit:

A- 5 phrases extraites d'un ouvrage spécialisé (voir ci-dessous) et illustrant spécifiquement l'erreur en question.

B- 5 phrases libres (inventées) contenant l'erreur en question. Elles pouvaient éventuellement provenir d'un corpus.

C- 5 phrases libres contenant la difficulté à traiter mais sans erreur.

L'ouvrage spécialisé pouvait être :

- Une grammaire comparative de l'anglais et du français.
- Un livre du genre "Les 1001 erreurs en français, à l'usage des anglophones".
- Un livre de grammaire française.

L'auteur des phrases a également été chargé de constituer une liste des mots utilisés dans les phrases (sous leur forme canonique).

4ème étape: un lexique LFG a été créé d'après les mots de la liste produite lors de la 3ème étape. Ce nouveau lexique comprenait les éléments suivants:

- informations pour l'analyse morphologique.
- traits syntaxiques et fonctionnels requis par l'analyse grammaticale.
- traits relatifs aux erreurs à détecter.

Le corpus a été soumis au logiciel (sans analyse ni correction) afin de déterminer si le lexique contenait bien tous les mots du corpus. Cette vérification a permis d'identifier quelques mots (moins d'une dizaine) oubliés dans la liste produite par l'auteur du corpus ainsi que quelques structures morphologiques initialement omises, comme par exemple le conditionnel de certains verbes.

5ème étape: l'évaluation proprement dite s'est déroulée en soumettant le corpus d'évaluation au logiciel. Cette opération, effectuée par une personne extérieure au projet, a duré une quinzaine de minutes.

4.4. Résultats et discussion

La classification des messages du logiciel (mises en garde et correction) a été adaptée à la méthode d'analyse présentée par Grosjean (Grosjean, 1988). Nous considérons les cas indiqués dans le tableau 4-1.

A. ERREUR
- I. DETECTEE
  - a. correctement
    - 1) mise en garde (MG)
    - 2) suggestion de correction (SC)
  - b. incorrectement
    - 1) mise en garde (MG)
    - 2) suggestion de correction (SC)
- II. NON DETECTEE ('miss')

B. NON-ERREUR
- I. DETECTEE ('false alarm' ou 'bruit')
  - a. incorrectement
    - 1) mise en garde (MG)
    - 2) suggestion de correction (SC)

Tableau 4-1 - Classification des erreurs lors de l'évaluation

Notre analyse des résultats a été effectuée d'après une procédure couramment utilisée (Palmer et Finin 1990), qui consiste à analyser le comportement du logiciel d'une part en tant que *boîte noire*, et d'autre part en tant que *boîte de verre*. Dans la première approche, seule la performance intrinsèque du système compte. Dans la seconde, on étudie le fonctionnement de chaque partie du logiciel individuellement, ce qui permet d'identifier celles qui fonctionnent correctement et celles qui font défaut. Dans notre cas, nous examinons dans une première phase les erreurs détectées par le logiciel, et dans une seconde phase les problèmes de détection qu'il a rencontrés.

4.4.1. Erreurs détectées

L'évaluation a donné les résultats indiqués dans le tableau 4-2 ci-dessous. Etant donné que l'évaluation portait sur dix erreurs par catégorie, les pourcentages des cinq premières colonnes divisés par 10 représentent dans chaque cas le nombre d'erreurs détectées ou non.

A première vue, on peut considérer les résultats de l'évaluation comme tout à fait satisfaisants. Un taux de détection correcte de 70% (16% + 54%) et un nombre relativement faible de non-erreurs (20 sur 1043 mots) sont comparables à ceux des meilleurs logiciels de correction grammaticale en L1 ou en L2. A titre d'exemple, une étude de produits commerciaux pour l'anglais L1 (Rabinovitz 1991) indique un taux de détection d'erreurs grammaticales avoisinant 50% pour les meilleurs et 10% pour les moins bons.

Catégorie	Erreurs détectées correctement A.I.a		Erreurs détectées incorrectement A.I.b		Erreurs non-détectées A.II	Nombre de non-erreurs détectées B.I.a	
	MG [%]	SC [%]	MG [%]	SC [%]	[%]	MG	SC
1. Accord S-V	-	70	-	-	30	-	1
2. Accord SN	-	80	-	10	10	-	5
3. Infinitif-PP	-	100	-	-	-	-	8
4. PP-être	-	60	-	20	20	-	-
5. PP-avoir	-	80	-	-	20	-	-
6. Mode	-	40	-	-	60	-	-
7. Structure	-	20	10	-	70	2	-
8. Adjectifs	100	-	-	-	-	2	-
9. Prép.+inf.	-	60	-	-	40	1	-
10. Conf. à-a	-	80	-	10	10	-	1
11. Prép.+art.	80	-	-	-	20	-	-
Total	16%	54%	1%	4%	25%	5	15

MG = mise en garde SC = suggestion de correction

Tableau 4-2 - Résultats de l'évaluation des phrases-test

Toutefois, les contraintes imposées à la rédaction des phrases-test nous obligent à la prudence dans la comparaison avec des produits commerciaux. D'une part, nos phrases ne contiennent qu'un nombre limité de types d'erreurs et les structures grammaticales des phrases sont assez

restreintes comparées aux textes soumis aux logiciels commerciaux. D'autre part, les phrases-test du corpus ne contiennent chacune qu'une seule erreur. Nous n'avons pas étudié le fonctionnement du logiciel dans les cas difficiles où plusieurs erreurs apparaissent dans une phrase.

Considérons tout d'abord les erreurs détectées correctement par le logiciel. Elles se décomposent en mises en gardes (16%) et en suggestions de correction (54%). Cette répartition des messages du logiciel est le résultat direct de la différence entre les procédures de correction utilisées pour les différents types d'erreur. Pour les catégories 8 (position des adjectifs) et 11 (prép+article), le logiciel n'était pas programmé pour rechercher la forme correcte du texte mais uniquement pour afficher une mise en garde. Pour toutes les autres catégories, la procédure de correction allait chercher dans la structure fonctionnelle de la phrase les traits requis pour reconstruire la forme morphologique correcte ou identifier le mot à introduire.

Le fait qu'il ait été possible dans 9 catégories sur 11 de faire appel à des procédures suggérant une correction s'explique de plusieurs façons. Tout d'abord, la nature des erreurs de ces catégories se prête aisément à la correction car elles correspondent dans la plupart des cas à des règles de grammaire strictes, comme l'accord du participe passé ou la confusion 'à-a'. Ensuite, l'accès aux données nécessaires pour la correction est rendu possible par l'utilisation d'un formalisme grammatical avancé comme la LFG. En effet, les structures fonctionnelles créées lors de l'analyse du texte contiennent toutes les informations requises pour identifier l'erreur et retrouver les valeurs de référence telles que la position de l'auxiliaire ou le nombre et la personne du sujet.

Considérons maintenant l'aspect 'détection' du logiciel (A.I.a dans le tableau 4-2), sans distinguer entre mise en garde et suggestion de correction, et analysons ces résultats par catégorie.

a. Accord SN et Infinitif-PP (catégories numéros 2 et 3)

Si on compare le taux de détection correcte avec le nombre de non-erreurs produites pour chaque catégorie (colonne de droite du tableau 4-2), on constate une concentration des non-erreurs dans ces catégories pour lesquelles la détection fonctionne le mieux. En effet, leur taux de détection est élevé (80% pour Accord SN et 100% pour Infinitif-PP), mais elles se partagent près des deux tiers des non-erreurs. On en déduit que les règles utilisées pour détecter ces deux types d'erreurs se déclenchent trop facilement. Dans certains cas elles permettent de détecter correctement une erreur mais dans beaucoup d'autres elles se déclenchent sans qu'aucune erreur ne se trouve dans le texte.

Cette observation est quelque peu inquiétante: un ajustement du logiciel, qui consisterait par exemple à renforcer les contraintes pour les catégories produisant de nombreuses non-erreurs, entraînerait sans nul doute une diminution du nombre de détections correctes, et la performance globale du logiciel en serait probablement affectée négativement. La décision d'effectuer ou non ces ajustements dépend principalement du but recherché: couverture d'un grand nombre d'erreurs, comme c'est le cas ici, ou précision dans la détection. Cette décision ne peut toutefois pas être prise avant de connaître la performance du logiciel sur l'ensemble des

erreurs à traiter. Etant donné que nous ne traitons que de 11 catégories d'erreurs, nous ne nous pencherons pas sur cette question.

b. Adjectifs et Prép.+art. (catégories numéros 8 et 11)

Les phrases-test relatives à la position des adjectifs possèdent une structure relativement simple, principalement à cause des contraintes imposées lors de la rédaction des phrases elles-mêmes. Voici quelques-unes des phrases-test typiques:

* *Une **bleue tapisserie** pendait au mur.*
* *Du **noir drap** ornait la table.*
* *Elle portait un **blanc bonnet**.*
* *Il vit une **rouge voiture**.*
* *Il a cassé la **rouge tasse**.*
* *Elle voulait un **ovale miroir**.*

Pour ces phrases, comme d'ailleurs pour celles relatives à la catégorie 'Prép.+article', une simple identification d'une suite adjectif-nom ou nom-adjectif aurait suffi à détecter l'erreur. Dans ces cas, la mise en oeuvre, très coûteuse en temps de calcul et en espace mémoire, d'une détection basée sur une analyse syntaxique n'est donc pas justifiée.

c. PP-avoir (catégorie numéro 5)

Le fait que cette catégorie ait obtenu un très bon score (80%) sans provoquer de non-erreurs est assez encourageant, surtout si on examine de plus près la nature des phrases-test utilisées, dont voici quelques exemples:

a. * *Ceux qui ont **dansés** seront appelés.*

(le logiciel a indiqué avec raison qu'il fallait remplacer *dansés* par *dansé*)

b. * *L'argent que tu lui as **donnée** était de trop.*

(succès ici aussi: *donnée* -> *donné*)

C'est dans ce genre d'erreurs que l'utilisation de la LFG se met en valeur, car dans ces deux cas nous avons des paramètres très divers qui déterminent l'accord du participe passé:

- L'identification du complément d'objet direct.
- La détermination de la position relative de ce complément.
- Le rattachement du pronom relatif au syntagme nominal. (*que* avec *l'argent* dans le cas (b))

d. Conf à-a (catégorie numéro 10)

Le succès relatif de cette catégorie (80% de détection, 1 non-erreur) s'explique par le fait que la structure syntaxique qui correspond aux cas où l'erreur est présente n'a souvent pas de cohérence grammaticale. La méthode utilisée pour détecter ce type d'erreur consiste à encoder le mot 'a' dans le dictionnaire du logiciel comme verbe et comme préposition. Si la structure fonctionnelle qui correspond à la première possibilité n'est pas complète mais que la seconde l'est, l'erreur est indiquée.

Considérons l'exemple suivant:

** Il rentre **a** la maison.*

Lorsque *a* est un verbe, *a la maison* constitue un syntagme verbal (*la maison* est le complément) et *rentre a la maison* aussi ([V [V SN]]) avec la fonction XCOMP. Mais *rentrer* n'accepte pas cette sous-catégorisation. Si on remplace *a* par *à*, *a la maison* devient un syntagme prépositionnel et la sous-catégorisation de *rentrer* est satisfaite. La détection et la correction de ce type d'erreur dépend donc directement des données qui concernent la sous-catégorisation de chaque verbe. Comme on le verra plus loin, ceci peut manifestement poser des problèmes importants lors de la construction du lexique.

e. Mode et Structure (catégories numéros 6 et 7)

On se trouve ici dans le cas opposé à celui des catégories 2 (accord SN) et 3 (confusion infinitif-PP), où les taux de détection était les meilleurs mais où le nombre de non-erreurs était également élevé. Les catégories Mode et Structure sont celles qui ont produit les taux de détection les plus faibles tout en ne produisant en tout que 2 non-erreurs. Pour ces catégories, les règles de détection sont donc trop rigides: elles ne se déclenchent que dans des situations spécifiques déterminées à l'avance.

Considérons les deux exemples suivants, extraits des phrases-test:

1. * *Vous obéissez ___ Pierre.*
2. * *Elle veut **son frère venir**.*

Dans le premier cas, la sous-catégorisation du verbe *obéir* indique qu'un complément d'objet indirect est requis, alors que la phrase ne contient qu'un complément d'objet direct. Une règle qui vérifie cette situation est comprise dans le logiciel qui peut détecter l'erreur avec succès et produire le message d'erreur suivant:

```
Attention: le verbe obéir s'utilise avec un objet
indirect au lieu d'un objet direct.
```

Dans le second cas, le logiciel est incapable de construire les structures nécessaires et l'erreur n'est pas détectée (comme nous le verrons dans la section 4.4.2).

Le fait qu'il soit nécessaire de prévoir chaque situation particulière pour ces deux catégories d'erreurs rend l'utilisation du logiciel particulièrement difficile dans un cadre plus étendu que celui des phrases-test. Si on voulait éviter la multiplication des non-erreurs, il faudrait en effet programmer un grand nombre de règles de détection et de correction. On peut alors se demander si l'utilisation d'une grammaire comme la LFG est justifiée pour corriger ces erreurs: il serait plus simple de rechercher directement les constructions incorrectes dans le texte, plutôt que de construire à grands frais les structures fonctionnelles et syntaxiques et ensuite de déterminer quelles conditions n'ont pas été remplies.

f. Autres

Les catégories 1 (Accord S-V), 4 (PP-être) et 9 (Prép.+inf.) ont connu des taux de succès relativement moyens (70%, 60% et 60% respectivement). Il

est difficile d'évaluer ces résultats sans entrer dans les détails de l'architecture du logiciel, ce qui est fait dans la section qui suit. On peut tout de même dire que le traitement de ces trois catégories d'erreurs requiert une analyse complète de la phrase, ce qui explique en partie le fait qu'elles soient parmi celles qui ont eu le moins de succès.

Dans la section qui suit, nous analysons les causes des malfonctionnements du logiciel. Le but de cette étude est de mettre en évidence les éléments du logiciel qui opèrent de manière satisfaisante et ceux qui nécessitent des améliorations.

4.4.2. Problèmes de détection

Comme nous l'avons décrit dans le chapitre 3, le logiciel exécute un certain nombre d'opérations lors du traitement d'un texte, opérations que nous pouvons regrouper ici en quatre étapes majeures:

1. Création d'arbres syntaxiques à l'aide de règles de réécriture (règles syntagmatiques).
2. Unification (création des structures fonctionnelles) en utilisant les informations lexicales propres à la LFG et les annotations aux règles de réécriture.
3. Filtrage des solutions des étapes 1 et 2 pour ne garder qu'une seule structure fonctionnelle.
4. Identification des erreurs dans les structures fonctionnelles et correction de celles-ci, qui se fait par l'affichage de messages de mise en garde ou de suggestion d'un mot à remplacer par un autre.

Afin d'évaluer le fonctionnement de chacune de ces étapes, considérons les 53 cas où le logiciel s'est trompé dans son diagnostic, afin de produire des non-détections, des détections incorrectes et des non-erreurs.

a. Arbres syntaxiques: 24 cas

Le logiciel a construit un arbre syntaxique incomplet ou erroné 24 fois. Ce problème n'était pas dû à un malfonctionnement de l'analyseur syntaxique (un analyseur ascendant en l'occurrence), mais plutôt au fait que les règles syntagmatiques utilisées par l'analyseur ne permettaient pas de représenter certaines structures des phrases-test, comme les structures verbales complexes (5 cas), l'utilisation de l'infinitif passé (*avoir marché*, 5 cas) et les temps surcomposés (4 cas). Les exemples suivants illustrent ces problèmes.

i. Structures verbales complexes

(1) * *C'est moi qui est Guillot.*
(accord sujet-verbe)

(2) * *Monsieur Durand m'a parlé: le professeur, quoiqu'**agrégée**, est un imbécile.*
(accord du SN)

Dans (1), le logiciel n'a pas détecté la faute dans l'accord du second verbe *être*. Dans (2), la mauvaise personne du participe n'a pas non plus été détectée. Dans ce dernier cas, l'analyseur a décomposé la phrase en quatre parties: *Monsieur Durand m'a parlé*, *le professeur*, *quoiqu'agrégée*, et *est*

un imbécile. Toutes, sauf la troisième partie, ont été identifiées comme des phrases ou des syntagmes valides et le reste du traitement s'est déroulé avec succès dans chaque cas. La troisième partie, quant à elle, n'a pas été traitée, car la suite conjonction-participe passé ne se trouvait pas dans les règles syntagmatiques.

ii. Infinitif passé

(3) * *Après avoir* ***bouclée*** *sa malle, il partit.*
(Accord du PP avec avoir)

(4) * *Avoir* ***connus*** *des étrangers est important.*
(Accord du PP avec avoir)

Ici aussi, le logiciel a décomposé les phrases en plusieurs parties. Le fait que chacune d'elles soit traitée séparément a empêché la détection des erreurs d'accord. Dans (4), le logiciel, a utilisé *des étrangers est important* comme unité de traitement et a suggéré de remplacer *est* par *sont* et de mettre un *-s* à *important*.

iii. Temps surcomposés

(5) * *Il s'est senti* ***concernée*** *par cette histoire.*
(accord du PP avec être)

(6) *Les auteurs ont été admirés.*
(accord du PP avec être)

Dans ces deux cas, le logiciel a assimilé le troisième verbe avec le début d'une proposition subordonnée infinitive. Ainsi, dans (5), *senti* a été identifié comme le verbe principal et *concernée par cette histoire* comme son complément. Il en a résulté dans les deux cas un message du logiciel indiquant que le participe devait être remplacé par l'infinitif (*concerner* et *admirer* dans nos cas).

b. Eléments LFG: 26 cas

Des données inadéquates dans le logiciel ont provoqué 26 erreurs. Dans le cas présent, un lexique LFG incomplet en est en grande partie responsable.

Considérons les exemples suivants:

(1) * *Nous nous sommes* ***habillé*** *en vitesse.*
(accord du PP avec être)

Le verbe pronominal *s'habiller* n'a pas été reconnu comme tel et le logiciel n'a pas détecté l'erreur d'accord. La cause de ce problème est le second *nous* auquel le logiciel n'a pas assigné la bonne fonction.

(2) * *Je regrette que vous* ***attendez*** *toujours.*
(mode dans les propositions subordonnées)

Le logiciel a bien trouvé que *attendez* devrait être mis au subjonctif, mais il a aussi indiqué par erreur que *attendre* s'utilisait avec un complément d'objet direct.

(3) * *Je suis contente qu'elle **vient**.*

(mode dans les propositions subordonnées)

(4) * *Il rend visite _ sa mère.*

(structure des compléments)

Ce sont des expressions (*être contente*, *rendre visite*) qui déterminent le mode de la subordonnée (dans 3) et la sous-catégorisation (dans 4). Ce genre de données lexicales n'était pas mis en oeuvre dans le logiciel.

Les exemples ci-dessus mettent en évidence deux types de problèmes:
1- La complexité de l'utilisation des données fonctionnelles de la phrase. D'une part (exemple 1), il n'est pas toujours possible d'ajouter de nouvelles interprétations fonctionnelles aux mots du lexique sans causer d'effets secondaires. L'exemple de *nous* est représentatif. Il se trouve dans le lexique en tant que pronom nominatif, accusatif, datif et personnel. La quantité de solutions que le logiciel doit ainsi gérer simultanément lorsqu'il traite une phrase comme l'exemple (1) l'empêche de détecter l'erreur avec succès. Dans l'exemple (2), il manque au lexique la forme 'attendre<(↑ SUBJ)>'. Mais sa présence dans le lexique rendrait la détection d'une erreur comme **Il attend sur le train* impossible car *sur le train* serait traité comme un complément circonstanciel échappant à la sous-catégorisation. Seules des informations sémantiques attachées à *train* permettraient de distinguer **Il attend sur le train* de *Il attend sur le banc*.

2- La structure du lexique. Il manque à notre logiciel un mécanisme permettant de regrouper des expressions telles que celles des exemples (3) et (4), *être contente* et *rendre visite*. Si on compare la sous-catégorisation de *rendre* et de *rendre visite* par exemple, on constate qu'elles sont différentes:

```
'rendre <(↑SUBJ)(↑OBJ)(↑OBJ2)>'
'rendre_visite <(↑SUBJ)(↑OBJ2)>'
```

Deux possibilités s'offrent pour traiter ce genre de problème: exécuter une phase de pré-traitement qui identifie les expressions comme *rendre visite* ou remplacer dans les structures fonctionnelles les traits de *rendre* par ceux de *rendre visite* lorsque le mot *visite* apparaît. Si la seconde solution est préférable du point de vue de l'homogénéité des données, le fait de remplacer certains traits par d'autres n'est pas compatible avec le processus d'unification. Quant à la première solution, elle est faisable mais elle n'entre pas dans le cadre de notre étude de la LFG.

c. **Filtrage: 2 cas**

Nous avons identifié deux cas, des non-détections, où les critères de filtrage des structures fonctionnelles n'ont pas fonctionné. Dans chacun d'eux, le logiciel n'a pas pu distinguer entre les erreurs contenues dans le texte et celles provoquées par l'unification des fausses interprétations de certains mots, là où l'analyseur avait suivi le mauvais chemin. (On se souvient en effet que le logiciel construit une structure fonctionnelle pour chaque combinaison des différentes interprétations des mots.)

d. Correction: 1 cas

Le seul cas où le mécanisme de correction n'a pas fonctionné est le suivant:

* *Cette homme ne vient pas souvent.*

Le logiciel a proposé *ce* comme correction à la place de *cet*. Malgré le fait que l'erreur avait bien été détectée, c'est-à-dire que le trait 'masculin' devait s'appliquer à *cette*, il a manqué trois éléments au logiciel:

1. Un trait dans le lexique indiquant que le mot *homme* débute par un *-h* muet.
2. Un trait dans le lexique attaché à *cet* indiquant qu'il s'utilise au lieu de *ce* devant un mot comme *homme*.
3. Un mécanisme à l'intérieur de la procédure de correction qui décide entre *ce* et *cet*.

Conclusion

L'analyse des résultats présentée ci-dessus nous a permis d'obtenir les renseignements suivants au sujet du logiciel LFG:

1. Les catégories d'erreurs pour lesquelles le logiciel fonctionne le mieux sont aussi celles qui produisent le plus de non-erreurs.

2. Les algorithmes fonctionnent très bien dans les cas où les données linguistiques du texte à traiter sont connues (voir l'accord du participe

passé avec *avoir*). Lorsque ces données manquent, le logiciel commet de nombreuses erreurs.

On pourrait alors en conclure qu'il suffit d'ajouter des règles syntagmatiques et les données manquantes au lexique pour obtenir un taux de succès avoisinant les 100%. Certes, ceci permettrait probablement d'éviter les erreurs du logiciel dans des cas bien précis, mais on peut douter que de telles opérations n'auraient que des effets positifs sur le fonctionnement du logiciel, ceci pour les raisons suivantes:

- Nous observons une progression exponentielle des données que le logiciel doit conserver simultanément en mémoire. Il faudrait donc revoir les algorithmes d'analyse syntaxique, d'unification et de filtrage.
- Même si les ressources informatiques étaient illimitées, le fait d'ajouter de nouvelles règles et de nouvelles données lexicales entraînerait certainement d'autres non-erreurs et non-détections.
- Chaque nouveau corpus de phrases-test mettrait à jour de nouvelles lacunes dans les données.

Nous avons également constaté certaines limites dans le formalisme de la LFG, plus particulièrement dans le processus d'unification, ceci malgré les modifications que nous lui avons apportées. Considérons, par exemple, les erreurs de la catégorie 8, relatives au mauvais placement de l'adjectif par rapport au nom. Nous utilisons le trait ADJ-POSITION pour détecter les erreurs de cette catégorie. Ce trait a deux valeurs possibles, APRES-NOM et AVANT-NOM. Nous avons dû limiter les syntagmes nominaux des

phrases-test contenant un seul adjectif, car deux ou plusieurs adjectifs pourraient introduire des valeurs contradictoires dans la f-structure du syntagme nominal, ce qui provoquerait naturellement la détection non voulue d'une erreur. L'utilisation de l'unification, même si elle permet de détecter et de corriger certaines erreurs de manière adéquate, limite donc le genre d'erreurs que nous pouvons traiter à l'aide de cette approche.

Tous ces problèmes rendent donc l'utilisation de la LFG hasardeuse dans des correcteurs commerciaux surtout si elle est employée seule pour la détection. Les algorithmes présentés ici ne s'appliqueraient de manière efficace qu'après l'élimination des autres erreurs dans le texte, point que nous reprendrons dans la conclusion générale de la thèse.

Chapitre 5

Un correcteur basé sur l'approche par automates

En contraste avec l'approche LFG décrite dans les chapitres précédents, nous présentons ici une méthode qui ne fait pas appel à une analyse grammaticale complète du texte à corriger. Cette nouvelle approche repose sur l'utilisation d'automates, à savoir de règles de détection dont la structure s'apparente à celle des réseaux de transition augmentés . Ceux-ci se limitent à la capacité d'automates à états finis avec registres (Woods, 1970; Winograd, 1983). Cette approche peut nous permettre de résoudre certains problèmes rencontrés avec la méthode basée sur la LFG. Etant donné que nous ne procédons plus à une analyse complète du texte, nous pensons pouvoir éviter les cas où des données incomplètes lors de l'analyse provoquent des malfonctionnements. De plus, le formalisme des

automates nous permet de détecter et de corriger une gamme d'erreurs bien plus étendue.

Le formalisme a été développé dans le cadre du projet ARCTA (Aide à la rédaction et à la correction de textes anglais de francophones) au laboratoire de traitement du langage et de la parole de l'Université de Neuchâtel grâce à une subvention de la CERS. Le but du projet consistait à élaborer le prototype d'un outil qui facilite l'écriture en anglais de francophones. Le choix de l'anglais et du français provient d'une part des connaissances et intérêts linguistiques des membres de l'équipe travaillant sur le projet et d'autre part des contraintes liées à la collaboration avec une entreprise privée.

Dans ce chapitre, nous présentons tout d'abord un aperçu du logiciel développé dans le cadre du projet et le rôle que jouent les automates dans le logiciel (section 5.1.). Nous décrivons ensuite le formalisme des automates (section 5.2.), la méthode utilisée pour organiser logiquement leur exécution (section 5.3.), ainsi que les algorithmes de correction (section 5.4.). Nous montrons enfin comment les automates peuvent être utilisés pour détecter un nombre très varié de types d'erreurs (5.5.).

5.1. Le projet ARCTA

Comme nous l'avons indiqué ci-dessus, la langue seconde traitée par le correcteur ARCTA est l'anglais. Toutefois, nous ne pensons pas que cette différence avec le correcteur LFG affecte les résultats de nos recherches, car ce ne sont principalement que les règles grammaticales qui changent d'une langue à l'autre. Dans chaque langue apparaissent des erreurs du même type: orthographe, choix des mots, ordre des mots,

choix des prépositions, etc. Nous croyons donc que les mécanismes de détection et de correction, présentés ici pour le traitement de l'anglais écrit par des francophones, s'appliquent tout aussi bien lorsque les langues sont inversées.

Si notre but consiste à développer le côté algorithmique de la détection et de la correction en langue seconde, il est important de souligner tout le travail purement linguistique effectué par les autres membres de l'équipe pour préparer l'ensemble des données relatives aux erreurs. Ce travail a consisté, entre autres, à construire le corpus d'erreurs, à étudier la nature de celles-ci, à en établir une typologie, et à utiliser les mécanismes présentés ci-dessous afin de les tester et de les affiner.

Le logiciel de correction fait partie d'un prototype dont la fonction est de fournir un environnement complet à un utilisateur écrivant un texte en langue seconde. Nous décrivons ce prototype dans la section 5.1.1. et l'architecture du correcteur dans la section 5.1.2.

5.1.1. Le prototype ARCTA

Le prototype ARCTA, développé sous Microsoft Windows, comprend un ensemble d'outils informatiques qui fournissent à l'utilisateur une grande variété d'informations relatives aux problèmes fréquemment rencontrés par les locuteurs francophones lorsqu'ils écrivent en anglais. Ces outils regroupent un correcteur grammatical, que nous décrivons dans la section suivante, une série d'aides accessibles en temps réel et un identificateur de problèmes potentiels. L'utilisateur accède directement à chacun de ces outils depuis la fenêtre d'édition du texte par l'intermédiaire de menus.

Les aides comprennent les modules suivants:

- Un dictionnaire bilingue anglais-français (simulé dans le prototype).
- Un dictionnaire monolingue anglais (également simulé dans le prototype).
- Un dictionnaire des synonymes anglais (simulé).
- Une série de fiches grammaticales, qui contiennent des règles d'utilisation de l'anglais. Elles décrivent, par exemple, l'emploi de *since* et *for*.
- Des listes de mots difficiles, comme les faux-amis et d'autres mots prêtant à confusion pour les locuteurs francophones.
- Un conjugueur.
- Un petit traducteur d'expressions figées.

L'identificateur de problèmes potentiels met en évidence les mots du texte qui sont une source fréquente d'erreurs chez les francophones. L'utilisateur peut obtenir des renseignements au sujet de l'emploi de chaque mot simplement en sélectionnant le mot à l'aide de la souris. L'avantage de ce mécanisme est qu'il permet de prendre en charge un certain nombre de catégories d'erreurs que le détecteur grammatical ne peut pas traiter (à cause des limites de ses algorithmes, par exemple). Considérons, par exemple, le cas des faux-amis. Le correcteur grammatical n'a pas accès aux données sémantiques de chaque phrase et ne peut donc pas détecter la plupart des erreurs de faux-amis de manière satisfaisante. Une solution (utilisée par la plupart des correcteurs commerciaux) consisterait à faire que le correcteur grammatical affiche un avertissement à chaque fois qu'il rencontre un faux-ami, même si celui-ci est utilisé correctement, ce qui lasserait nombre d'utilisateurs. Au lieu de cela, il a été décidé que le correcteur grammatical du prototype ARCTA ne traiterait pas du tout les faux-amis. Il laisse le soin à l'identificateur de problèmes potentiels d'en signaler chaque occurrence de manière passive, donc sans inonder l'utilisateur de messages

d'erreurs inutiles.

Une description plus complète de ces outils ainsi qu'une évaluation du prototype ARCTA se trouvent dans Tschumi et al. (1996).

5.1.2. Architecture du correcteur ARCTA

Le logiciel de détection d'erreurs comprend des modules qui reflètent un certain nombre d'étapes, celles-ci allant de la lecture d'un fichier à traiter jusqu'à la correction même. Ces étapes sont illustrées dans la figure 5-1.

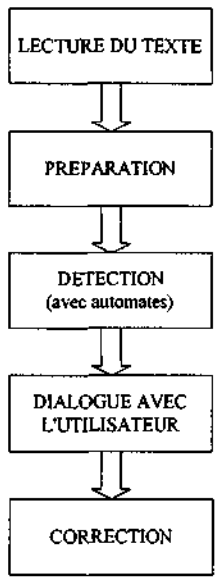


Figure 5-1. Architecture du correcteur grammatical ARCTA.

Deux méthodes de lecture du texte sont disponibles: l'utilisateur peut soit taper le texte à traiter dans une fenêtre d'édition, soit spécifier le nom d'un fichier à charger dans cette même fenêtre.

La phase de préparation, exécutée ensuite, consiste à construire pour chaque phrase du texte toutes les données requises pour la détection d'erreurs. Les opérations effectuées lors de cette deuxième phase sont les suivantes:

- Segmentation: découpage du texte en phrases, puis en unités lexicales: mots, chiffres, ponctuation.

- Etiquetage: identification des unités lexicales et attribution de traits syntaxiques à celles-ci. Les traits sont obtenus du dictionnaire d'analyse CELEX (obtenu du Consortium for Lexical Research) qui contient environ 5000 formes canoniques et leurs formes fléchies.

- Regroupement des expressions complexes: les prépositions composées, comme *a lot of* (*beaucoup de*), ainsi que certains mots composés, sont réunis en unités lexicales simples.

- Désambiguïsation syntaxique: le dictionnaire d'analyse produit dans certains cas plusieurs catégories syntaxiques pour un même mot. Ainsi, *can* peut être un verbe (*pouvoir*) ou un nom (*boîte*). Afin de faciliter la détection d'erreurs, le module de désambiguïsation syntaxique développé par Bodmer (1994) utilise les mots avoisinants pour éliminer toutes les catégories, sauf la plus probable.

- Identification des syntagmes nominaux simples: cette phase regroupe les

syntagmes nominaux simples (sans préposition), afin qu'ils puissent être utilisés en tant qu'entités distinctes lors de la détection (exemples: *my sister*, *a large house*, etc.).

La phase de préparation produit donc les résultats suivants:

1. Les unités lexicales, chacune d'elles étant accompagnée de la forme fléchie et la forme canonique, la catégorie syntaxique, les traits morphologiques (nombre, personne, temps) et certaines informations relatives à sa position au sein de la phrase.

2. Une liste de syntagmes nominaux simples.

L'exemple ci-dessous présente l'ensemble de données lexicales produites par la phase de préparation pour le texte *The clothes which we have*:

```
FF=The      FC=the     morph=X              CL=ART  3=1   pos=1
FF=clothes  FC=clothe  morph=p              CL=N    1=1   pos=2
FF=which    FC=which   morph=X              CL=PRON 5=70  pos=3
FF=we       FC=we      morph=X              CL=PRON 5=41  pos=4
FF=have     FC=have    morph=e1S+e2S+eP+i   CL=V    1=17  pos=5
```

Une fois la phase de préparation terminée, le système exécute la phase de détection qui consiste à activer les automates dans un ordre qui est déterminé par un agenceur. Ces automates sont répartis en trois groupes aux fonctions différentes. Les automates du premier groupe ne font qu'extraire des informations importantes du texte (automates de pré-traitement); ceux du deuxième groupe permettent de sélectionner des structures de phrases particulières, tandis que les automates du troisième groupe

ont pour fonction de détecter les erreurs. L'organisation logique des automates est décrite en détail dans la section 5.3.

Les données qui entrent en jeu lors de la détection sont les suivantes:

- La liste des unités lexicales (voir ci-dessus).
- La position des syntagmes nominaux simples à l'intérieur de la phrase.
- Les listes de mots ayant des propriétés communes (voir section suivante).
- Les automates.
- L'agenceur, qui décrit l'ordre dans lequel exécuter les automates.
- Les registres qui contiennent diverses informations, dont la position de l'erreur.

Lorsqu'un automate identifie une erreur, la phase de dialogue avec l'utilisateur est mise en route (voir Figure 5-1). Celle-ci consiste à rassembler les informations obtenues lors de l'exécution des automates de pré-traitement, à présenter l'erreur à l'utilisateur et à obtenir de celui-ci une décision quant à l'action à effectuer ensuite. Suivant le type d'erreur, l'utilisateur peut choisir de l'ignorer ou d'effectuer la correction manuellement ou automatiquement. Le formalisme des automates permet également un dialogue en deux temps: le système demande, par exemple, à l'utilisateur de confirmer le sens qu'il voulait donner à sa phrase puis, suivant sa réponse, il lui propose de faire une correction liée au sens voulu. Lorsque l'utilisateur choisit la correction automatique, le système utilise, pour modifier le texte, un mécanisme de correction rattaché à l'automate qui a détecté l'erreur. Ceci permet entre autres de remplacer automatiquement un mot par un autre, d'intervertir un ou plusieurs mots ou encore d'insérer ou d'effacer des mots. Le reste de ce chapitre est consacré uniquement à l'explication des mécanismes associés à la phase de

détection. Pour la description des autres parties du logiciel, consulter "Le programme ETI" (Comu, 1991a) et "Le logiciel ARCTA" (Comu, 1991b).

5.2. Formalisme des automates

Nos automates s'apparentent aux réseaux de transition augmentés présentés par Winograd (Winograd, 1983). De manière générale, un automate peut être décrit comme une série de conditions et d'actions qui s'appliquent à des suites d'éléments du texte, comme des mots, des signes de ponctuation ou des syntagmes nominaux. Un automate 'passe' ou 'réussit' si une suite d'éléments dans le texte satisfait aux conditions contenues dans l'automate.

Considérons l'automate suivant:

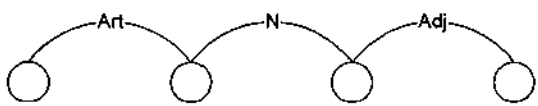


Figure 5-2 Automate ORDRE_ADJECTIF_NOM

et appliquons-le à la phrase **the house white was sold yesterday* (*La maison blanche a été vendue hier*). L'automate passe sur la suite **the house white* car ces trois mots correspondent exactement aux trois arcs.

On représente les automates de la manière suivante:

AUTOMATE <nom de l'automate>

<liste d'arcs>

La spécification d'un automate débute par le mot clé AUTOMATE, suivi du nom par lequel l'automate sera identifié dans le logiciel. Le nom de l'automate est suivi d'une liste d'arcs qui contiennent des conditions à respecter.

L'automate suivant reprend l'exemple présenté plus haut:

```
AUTOMATE 1
[CAT=ART] @[CAT=N] [CAT=ADJ]
```

Les conditions rattachées aux trois arcs sont notées entre les crochets '[' et ']'. Chacune d'elles détermine la catégorie syntaxique du mot à rechercher: article, nom et adjectif, respectivement[1]. Le signe '@', placé devant le deuxième arc, indique l'ancrage de l'automate: lors de l'exécution d'un automate par le logiciel, la première étape consiste à identifier dans la phrase un mot correspondant aux conditions indiquées dans l'arc d'ancrage. Il ne peut y avoir qu'un seul ancrage dans un automate. Une fois cet arc trouvé dans une phrase, l'exécution de l'automate se fait d'abord vers la droite, en direction de la fin de la phrase, puis vers la gauche à nouveau à partir de l'ancrage. Le fait de pouvoir spécifier le point de départ d'un automate nous donne un certain nombre d'avantages. En particulier, ceci permet de lier les automates aux types d'erreurs qu'ils détectent. Ainsi, l'automate ci-dessus, dont le but est d'identifier une erreur dans l'ordre adjectif-nom, sera ancré sur les noms; un autre automate, dédié à la vérification de l'accord sujet-verbe, pourra être

[1]Une liste complète des abréviations utilisées par le formalisme se trouve dans l'annexe 4. Ces abréviations sont en principe dérivés du français (comme SN pour syntagme nominal) sauf dans certains cas où nous avons repris la terminologie anglaise du dictionnaire CELEX.

ancré sur les verbes.

Winograd n'utilise pas d'ancrages dans son formalisme. Les réseaux d'arcs qu'il utilise possèdent chacun un noeud initial et un noeud terminal, qui correspondent respectivement au début et à la fin d'une phrase ou d'un syntagme. L'analyse consiste alors à trouver un chemin, parmi les arcs, allant du noeud initial au noeud terminal en couvrant toute la phrase. Ceci contraste avec notre approche, qui permet également le parcours des arcs dans le sens inverse, c'est-à-dire de droite à gauche.

Le formalisme que nous avons développé pour la définition des automates les rend particulièrement adaptés à la détection d'erreurs. Voici quelques aspects importants:

- En plus des données comprises dans le dictionnaire, comme la catégorie syntaxique et les informations morphologiques de chaque mot, les automates ont accès à des listes de mots regroupés selon des caractéristiques communes. On utilise habituellement ces listes en relation avec la forme canonique (FC) ou fléchie (FF) des mots. Nous avons établi, par exemple, une liste de tous les noms à aspect temporel (liste NTEMP: *afternoon*, *Christmas*, etc.) et une liste de verbes ne se mettant en principe pas à la forme continue (liste VNEVERCONT: *to seem*, *to last*).
- Les registres, notés $A, $B, ... $Z, permettent de stocker les traits afin que ceux-ci puissent être utilisés ailleurs dans l'automate. La notion de registre que nous utilisons est empruntée directement au formalisme de Winograd, que nous avons quelque peu adapté à nos besoins. En particulier, ils jouent ici un rôle principal dans le processus de correction. Les registres sont propres à chaque automate et leur valeur est détruite lorsque l'automate ne passe pas. Lorsque l'automate passe et indique une erreur, les valeurs des registres sont gardées et utilisées ensuite par

le module de dialogue avec l'utilisateur et par le module de correction.

- Il est possible de créer de nouveaux traits pour les mots et les syntagmes.
- La position et les caractéristiques des syntagmes nominaux peuvent se révéler très utiles pour la détection et la correction de certaines erreurs. Comme chez Winograd, nous avons inclus dans le formalisme la possibilité de faire référence à ces syntagmes.

Une description détaillée du formalisme utilisé pour définir les automates se trouve dans l'annexe 4.

5.3. Agencement

Les automates permettent d'identifier des suites spécifiques de mots dans le texte. Comme nous l'avons vu, certains automates détectent des suites légales (comme les formes verbales, par exemple), alors que d'autres identifient des situations correspondant à des erreurs. C'est dans le but de spécifier la fonction de chaque automate au sein du mécanisme global de détection et de correction que nous définissons leur *agencement*.

Nous présentons les détails de l'agencement dans les sous-sections 5.3.1 à 5.3.3. La sous-section 5.3.4. décrit un exemple de son application pour la détection du 's' manquant à la troisième personne du singulier des verbes. Nous présentons enfin un outil spécialement développé pour tester les automates et leur agencement, le VELO (vérificateur de logique des automates), qui nous a permis d'accomplir un travail préalable sur les automates avant de les inclure dans le correcteur même (section 5.3.5).

5.3.1. Aperçu

Considérons, par exemple, l'erreur relativement fréquente qui consiste à omettre le -s à la troisième personne du singulier des verbes anglais, comme dans la phrase suivante:

> * *With a modern office, productivity* ***increase****.* (correct: *increases*)
> (*Dans un bureau moderne, la productivité s'accroît*)

Avant qu'on puisse affirmer que le 's' manque à *increase*, il est nécessaire de déterminer,

> (1) Où se trouve le verbe principal ?
> (2) Quel est le sujet du verbe ?
> (3) Quels sont la personne et le nombre du sujet ?

Ce n'est qu'une fois ces informations connues qu'il est possible d'enclencher l'automate spécifique à cette erreur, c'est-à-dire celui qui compare la personne et le nombre du sujet et du verbe principal. Pour refléter cette situation, nous avons décomposé l'agencement des automates en deux phases distinctes: une phase dite de *pré-traitement*, qui consiste à rassembler certaines informations concernant le texte, et une phase de *détection*, où le système utilise ces données pour identifier les erreurs.

L'agencement des automates dans notre exemple aura ainsi l'allure suivante (nous supposons que les syntagmes nominaux ont été regroupés lors d'une phase

précédente où les automates n'entrent pas en jeu):

PRETRAITEMENT

{ automates d'identification des formes verbales }

{ automates d'extraction des traits des syntagmes }

DETECTION

{ automates de détection du 's' manquant }

Chaque ligne ci-dessus comprend une liste d'automates qui s'occupent chacun d'une situation particulière, que ce soit lors du pré-traitement ou lors de la détection. Ainsi, les automates de la liste qui identifient les formes verbales traitent des différents temps, du présent simple au conditionnel passé continu (*I would not have been drinking*).

L'agenceur d'automates (le module du logiciel qui s'occupe de les appliquer au texte) les exécute dans l'ordre dans lequel ils se trouvent dans la liste. Cet ordre est d'une grande importance, car il permet un classement rigoureux des automates: le plus complexe est exécuté en premier, le plus simple en dernier. L'exemple des formes verbales illustre parfaitement cette situation. Dans la phrase au conditionnel passé continu citée plus haut, si *have* n'était pas identifié initialement comme un auxiliaire de la forme verbale, l'automate d'une autre forme verbale pourrait l'utiliser et détecter, par exemple, le conditionnel (*would not have*) ou le présent simple (*have*) à cet endroit.

Afin d'empêcher des automates 'simples' d'identifier par erreur des formes plus complexes, il est donc nécessaire de les placer plus loin dans la liste. Pour rendre cette organisation en listes utilisable, le logiciel doit en outre appliquer la règle

suivante:

> Lorsqu'un automate 'passe', il empêche tous les autres automates de la liste d'utiliser les mots qui correspondent à ses arcs.

Ainsi, dans la phrase suivante,

> *I would not have been drinking if they had been here.*
> *(Je n'aurais pas été en train de boire s'ils avaient été là)*

l'automate d'identification du conditionnel passé continu est exécuté en premier et détecte la forme verbale au début de la phrase. Par conséquent, les autres automates des formes verbales ne peuvent pas s'arrêter sur un mot de la première forme; ils continuent donc et l'un d'eux détecte la forme verbale dans la seconde partie de la phrase. Cette règle d'exclusion du domaine d'exécution s'applique non seulement au niveau du pré-traitement, comme l'a montré l'exemple, mais aussi au niveau de la détection. Nous avons ainsi le fonctionnement général suivant pour l'agenceur:

1. pour chaque liste du niveau de pré-traitement
1.1. exécuter chaque automate de la liste
1.2. si l'automate passe, exclure son domaine d'application
2. pour chaque liste du niveau de détection
2.1. exécuter chaque automate de la liste
2.2. si l'automate passe, exclure son domaine d'application et indiquer une erreur.

5.3.2. Repérage

La nature de certaines erreurs, principalement celles relatives aux structures verbales, requiert un niveau d'automates intermédiaire. Les automates de ce niveau, appelé niveau de *repérage*, ont la fonction d'identifier certaines structures dans le texte afin de déclencher d'autres automates. Ce mécanisme permet de répartir la détection d'une erreur sur deux ou plusieurs automates, pour séparer la partie qui fait une analyse locale de celle qui recherche l'erreur, par exemple. Considérons à nouveau les problèmes d'accord entre le sujet et le verbe. Le processus de détection peut comprendre deux phases: identifier le verbe et le sujet avec lequel il s'accorde, puis déterminer si leur genre et leur nombre correspondent. La section 5.3.4., ci-après, illustre ce fonctionnement.

La sélection des automates qui recherchent l'erreur se fait à l'aide de registres, auxquels les automates de repérage assignent des valeurs suivant les structures qu'ils rencontrent. Ainsi, la séquence d'événements ci-dessous est exécutée de la manière suivante:

1- La liste d'automates de repérage est activée.
2- Les automates de cette liste déterminent la valeur de registres pré-définis.
3- Les conditions attachées à chaque liste de détection sont vérifiées.
4- Lorsque ces conditions sont remplies, la liste d'automates de détection correspondante est activée.

On aura par exemple les automates suivants:

- liste de repérage:

AUTOMATE_R1 action: valeur1 => $R

AUTOMATE_R2 action: valeur2 => $R

AUTOMATE_R3 action: valeur3 => $R

listes de détection:

si $R = valeur1, alors exécuter la liste

AUTOMATE_D11

AUTOMATE_D12

AUTOMATE_D13

si $R = valeur2, alors exécuter

AUTOMATE_D21

si $R = valeur3, alors exécuter la liste

AUTOMATE_D31

AUTOMATE_D32

Cette combinaison d'automates de repérage et de détection s'utilise, par exemple, pour détecter certaines structures verbales erronées. Le niveau de repérage comprend alors les différents automates responsables de l'identification des compléments, alors qu'au niveau de la détection sont testés, suivant les cas, les prépositions ou l'ordre de ces compléments.

Afin d'assurer que les automates de détection opérent sur le même domaine de la phrase que les automates de repérage qui les précèdent, l'agenceur utilise les mêmes ancrages pour les deux niveaux. Ceux-ci, identifiés au niveau du repérage, sont conservés et utilisés pour la détection.

5.3.3. Valeurs créées par les automates

Nous avons vu plus haut que certaines actions à l'intérieur des automates pouvaient affecter des valeurs à des registres et à des variables globales. Afin d'assurer l'exécution régulière du processus de détection, certaines règles s'appliquent à ces nouvelles valeurs suivant que les automates qui les définissent réussissent ou échouent. La liste suivante parcourt tous les types de données utilisables dans les automates et décrit les opérations effectuées par l'agenceur dans chaque cas.

a. Traits originaux

Les traits originaux sont ceux dont la valeur ne peut pas être modifiée lors de l'exécution des automates. Ces traits comprennent:

- Les informations extraites du dictionnaire d'analyse (nombre, genre, catégories syntaxiques, etc.).
- Les listes de traits.
- Les traits créés lors de la phase de préparation, comme les bornes des syntagmes simples et la position de chaque mot.

Ces traits ne sont pas affectés par les différentes phases de l'agencement.

b. Nouveaux traits

L'opération ':=', utilisée à l'intérieur d'un arc, permet d'assigner un nouveau trait à un mot. De même, l'opération '::=' permet d'assigner un nouveau trait à un syntagme.

Si l'automate qui les définit échoue, l'agenceur retire ces traits des données relatives aux mots et aux syntagmes. Si l'automate passe, les traits sont conservés jusqu'à la

fin de la détection.

c. Registres

L'agenceur détruit en général les valeurs des registres, même si l'automate réussit. Il existe un seul cas où les registres ne sont pas affectés: lors du passage d'une phase de repérage à une phase de détection. En effet, l'agenceur utilise les valeurs de ces registres pour déterminer la ou les listes d'automate(s) de détection à exécuter.

d. Bornes des SN complexes

Certains automates qui apparaissent au niveau du pré-traitement regroupent les syntagmes nominaux simples en syntagmes complexes (du type <SN> + *of* + <SN>, qui devient <SN-complexe>). D'autres regroupent des prépositions avec certains types de syntagmes nominaux pour créer des syntagmes prépositionnels. Etant donné que l'opération qui consiste à ajouter un syntagme à la liste est la dernière effectuée par un automate, et que cet automate a donc réussi, ces informations ne sont jamais effacées.

5.3.4. Exemple

L'exemple suivant présente l'agencement des automates pour la détection du 's' manquant à la troisième personne du singulier des verbes.

```
PRETRAITEMENT
! premier groupe: traits des SN
 (   TRAITSN_PRONOM_1 TRAITSN_PRONOM_2 TRAITSN_N_1   )
! deuxième groupe: formes verbales
 (   FUT_CONT_ACT  FUTANT_CONT_ACT
     FUTANT_SIMP_ACT  FUT_SIMP_ACT
```

```
      COND_CONT_ACT  CONDP_CONT_ACT
      CONDP_SIMP_ACT  COND_SIMP_ACT
      PRES_CONT  PRET_CONT
      PC_CONT_ACT  PC_SIMP_ACT
      PQP_CONT_ACT  PQP_SIMP_ACT
      PRES_SIMP  PRET_SIMP )
DETECTION
  REPERAGE
! repérage de l'erreur
      ( VACC_MANQ_ES_1_R  VACC_MANQ_ES_1A_R
        VACC_MANQ_ES_2_R  VACC_MANQ_ES_3_R  )
! détection de l'erreur
        ($V=R1)  VACC_MANQ_ES
  END
END
```

Le niveau de pré-traitement comprend deux groupes d'automates: le premier détermine le nombre et la personne des syntagmes nominaux et le second construit les formes verbales.

L'automate ci-dessous, par exemple, s'applique lorsque le syntagme se termine par un nom. Il ajoute deux traits au syntagme.

```
AUTOMATE TRAITSSN_N_1
  <SN @[CAT=N, EPOS=ELAS, PERS_SN::=P3, NBR_SN::=NBR]>
```

Explication:

- L'automate s'applique aux syntagmes nominaux uniquement.
- A l'intérieur du syntagme, il recherche un nom qui se trouve en dernière position. (vérifié à l'aide de la condition EPOS=ELAS).
- L'automate affecte deux traits globaux au syntagme: PERS_SN prend la valeur P3 (troisième personne) et NBR_SN prend le nombre du nom.

L'automate ci-dessous identifie le passé composé simple actif dans une phrase, comme dans *I have forgotten.*

```
AUTOMATE PC_SIMP_ACT
  @[FC='have', TNS=PRES]
    (AFF=>$S)
   [FF='not', NEG=>$S]*1
   [CAT=ADV]*
   [CAT=V, TNS=PART, TNS_FV:=PCOMP, VOIX_FV:=ACT,
              ASPECT_FV:=SIMP, STATUT_FV:=$S]
```

Explication:

- L'automate est ancré sur une forme de *have* au présent.
- Il saute tous les adverbes jusqu'à ce qu'il rencontre un verbe au participe passé.
- La valeur par défaut AFF (affirmatif) est enregistrée dans le registre $S.
- Si l'automate rencontre l'adverbe *not*, il change la valeur du registre $S.
- Quatre traits sont affectés au verbe principal: le temps (TNS_FV), la voix (VOIX_FV), l'aspect (ASPECT_FV) et le statut (STATUT_FV), affirmatif ou négatif.

Dans la phase de détection, nous remarquons la présence d'une liste de repérage et d'un seul automate de détection (VACC_MANQ_ES). Les automates de cette première liste identifient la structure du texte qui précède le verbe principal et ils affectent tous la valeur R1 au registre $R.

Considérons l'exemple de l'automate VACC_MANQ_ES_2_R:

```
AUTOMATE VACC_MANQ_ES_2_R
  (R1=>$R)
  <SN (PERS_SN=P3, NBR_SN=SING)>
  [FC='which' | FC='that' | FC='who']
 @[CAT=V, (TNS=PRES | TNS=PRET)]
```

Cet automate identifie des structures comme *the man who came*.

Explication:

- L'automate est ancré sur un verbe au présent ou au prétérit.
- Il recherche ensuite, à sa gauche, un des pronoms *which*, *that* ou *who*, puis un syntagme nominal à la troisième personne du singulier.
- La valeur R1 assignée au registre $R en cas de succès permet de déclencher l'automate de détection VACC_MANQ_ES.

L'automate de détection a la forme suivante:

```
AUTOMATE VACC_MANQ_ES
  @[MORPH/=M_3S]
```

Explication:

- La seule fonction de l'automate consiste à vérifier que l'ancrage n'est pas à la troisième personne du singulier.

5.3.5. VELO: Un outil de développement des automates

Le logiciel VELO (VErificateur de LOgique des automates) a été développé dans le cadre du projet ARCTA afin d'aider les linguistes-informaticiens à vérifier les automates qu'ils ont écrits. De manière générale, la personne qui écrit un automate introduit dans VELO le nom du fichier contenant la description des automates à tester ainsi que les textes qui font partie du corpus à utiliser. VELO applique les automates au corpus en suivant un des trois modes d'exécution disponibles: phrase par phrase, texte par texte ou sur le corpus dans son entier. D'autres options permettent, entre autres, de sélectionner les automates à vérifier ou de spécifier le genre d'information affiché à l'écran.

L'avantage du logiciel VELO est de produire des renseignements relatifs au succès ou à l'échec des automates sur certaines phrases. Les résultats comprennent, par exemple, des suites de mots correspondant au domaine d'un automate lorsque celui-ci a réussi, ou les ancrages trouvés pour chaque automate dans chaque phrase même si ceux-ci ont échoué.

Les modules qui composent VELO proviennent en grande partie du correcteur ARCTA décrit dans la section précédente. Ces modules sont les suivants:

a- Interface avec un corpus étiqueté

Le corpus soumis à VELO est composé de textes pour lesquels chaque mot a été préalablement associé à un certain nombre de traits tirés d'un dictionnaire. La fonction du module d'interface avec le corpus consiste à lire et interpréter les informations attachées à chaque mot et à les préparer pour qu'elles puissent être utilisées par les automates.

b- Interface avec l'utilisateur

Le module de dialogue avec l'utilisateur est caractérisé par une représentation assez simple, car une des contraintes imposées à VELO est d'être portable entre les différentes plate-formes informatiques. L'utilisateur entre des commandes, pour charger des fichiers ou sélectionner des options par exemple, et celles-ci sont directement exécutées par VELO.

c- Analyseur d'automates

Afin de faciliter la création et la modification des automates, le fichier qui les contient suit pas à pas le formalisme des automates décrit dans la section 5.2. La tâche de l'analyseur d'automates consiste à lire le fichier et à interpréter le formalisme spécifique aux automates. Une fois lus par le logiciel, les automates se trouvent sous une forme qui optimise leur temps d'exécution.

d- Processeur d'automates

Le processeur d'automates est le véritable coeur du logiciel. Suivant les options sélectionnées, il détermine les automates à exécuter, identifie le texte à analyser, recherche les ancrages dans chaque phrase et, à partir de ces ancrages, parcourt le texte dans les deux sens afin de déterminer si les mots du texte correspondent aux conditions attachées aux arcs.

Le logiciel utilise un ensemble de fichiers dans lesquels se trouvent les informations qu'il utilise lors de l'exécution. Ces fichiers contiennent des données sous forme textuelle uniquement, ce qui permet au linguiste-informaticien de les consulter et de les modifier facilement. Les fichiers sont les suivants:

a- Symboles

Ce fichier contient la liste de tous les traits et valeurs qui peuvent être utilisés dans les automates.

b- Automates

Le fichier des automates contient la description des automates d'après le formalisme décrit dans la section 5.2. de ce chapitre.

c- Agenceur

L'agenceur définit l'ordre d'exécution des automates. Le format de ce fichier est décrit dans la section 5.3. de ce chapitre.

d- Description du corpus

Le corpus se compose d'une série de textes étiquetés, séparés les uns des autres et ayant chacun un titre propre. Le fichier de description contient une liste des textes présélectionnés pour être soumis aux automates. Ceci permet d'éviter de devoir utiliser tout le corpus lorsque les erreurs sur lesquelles le linguiste-informaticien travaille ne se trouvent que dans une partie de celui-ci. Le corpus utilisé dans le cadre du projet ARCTA est décrit dans la section 6.2.1.

e- Textes étiquetés

Avant de pouvoir être traité par VELO, chaque mot est recherché dans le dictionnaire et les traits qui lui correspondent sont introduits dans un fichier. Un autre logiciel, le programme ETI, effectue cette opération.

f- Listes de mots ayant certains traits

Il est parfois utile de pouvoir regrouper une série de mots ayant un même trait afin de tester ce trait sans devoir modifier le contenu du dictionnaire. Ces listes sont décrites dans l'annexe 4.

g- Fichier de commandes

Le fichier contient les commandes qui sont exécutées lors de chaque activation du logiciel VELO. Ces commandes permettent de charger les fichiers décrits ci-dessus ou de faire démarrer l'exécution des automates, par exemple.

La description d'une session interactive avec le programme VELO se trouve dans l'annexe 5.

5.4. Correction

Nous avons vu, dans la section 5.2, que le formalisme permettait d'enregistrer des traits dans des registres lors de l'exécution des automates. Ces traits comprennent en particulier la forme canonique ou fléchie des mots ainsi que la position de mots-clés à l'intérieur de la phrase. Les informations contenues dans les registres sont primordiales au module de correction, car elles permettent de définir où se trouve l'erreur dans la phrase et quels sont les mots affectés. Dans

le cas d'une erreur d'accord entre le sujet et le verbe, par exemple, le contenu des registres décrit non seulement le verbe mal accordé, mais aussi le sujet complet tel que les automates l'ont identifié.

Les registres ne sont toutefois pas les seuls éléments que le module de détection transmet au module de correction. En effet, nous avons jugé utile d'attacher aux automates de détection toutes les informations requises pour que la correction puisse s'effectuer dans les meilleures conditions. Ces informations sont de trois types:

1. Le message d'erreur, qui est affiché à l'écran lorsque l'automate détecte une erreur. En plus des explications au sujet de l'erreur en question, le message peut contenir des références au contenu de registres et à des segments de phrases dont les bornes sont également définies par des registres. De plus, des caractères de contrôle insérés dans le message permettent d'en souligner des parties ou de les mettre en valeur sur l'écran.

2. Le numéro de la procédure de correction. Afin de faciliter la tâche du module de correction, nous avons identifié des algorithmes de correction communs à un ensemble d'erreurs, comme le remplacement d'un mot par un autre, l'effacement ou l'insertion d'un mot, ou encore la permutation de mots ou de groupes de mots. L'auteur des automates de détection identifie dans chaque cas la procédure que le module de correction doit exécuter et associe le numéro de celle-ci à l'automate. Chaque procédure de correction s'attend à rencontrer certaines valeurs dans des registres. Ainsi, la procédure d'effacement d'un mot ou d'un groupe de mots utilise le contenu des registres $L et $R pour déterminer la position du segment à effacer à l'intérieur de la phrase. La procédure d'insertion

ajoute un mot sélectionné par l'utilisateur dans une liste (voir point 3 ci-dessous). Le registre $1 contient le point d'insertion du nouveau mot.

3. Une liste de mots, qui contient des suggestions à présenter à l'utilisateur lorsque la correction implique l'insertion ou le remplacement d'un mot ou d'une suite de mots dans le texte. Comme pour les messages d'erreurs, les listes peuvent contenir des références à des registres.

L'automate suivant illustre comment ces trois éléments sont utilisés pour la correction:

```
AUTOMATE VSUIV_COMPL_THAT
  @[CAT=V, +V_NOT_THAT, FF=>$A]
   [FF='that', CURPOS=>$L]
   <SN (FIRSTPOS=>$B, LASTPOS=>$C)[EPOS=ELAS, CAT/=PRON]>
   [CAT=V, ~MODAUX, CURPOS=>$R, FC=>$D]
|
   MESSAGE "Le verbe \u\$A\v ne se construit d'habitude"
      " pas avec \uthat\v. \nUtilisez plutôt la"
      "forme avec l'infinitif complet, comme dans:\n"
      "\uWe expect your brother to come tomorrow.\v"
   SCHEMA 2
   LISTE "\($B-$C) to \$D"
}
```

Explication:

- L'automate détecte des erreurs comme l'utilisation erronée de *that* dans la phrase ***We expect that your brother comes tomorrow.*** (Correct: *...your brother to come tomorrow.*)
- L'automate est ancré sur un verbe appartenant à la liste V_NOT_THAT de

verbes qui ne se construisent pas avec *that*, comme *to expect*. La forme fléchie du verbe est assignée au registre $A.

- L'automate recherche ensuite un syntagme nominal qui n'est pas un pronom (pour éviter les cas tels que **He expected that he comes*..., qui serait corrigé par cet automate par **He expected he to come*..., au lieu de *...him to come*...), et recherche également un verbe non modal (pour éviter les cas tels que **He expected that your brother will come*..., qui serait corrigé par **He expected your brother to will come*...).

Examinons tout d'abord comment les données de correction sont enregistrées durant l'exécution de l'automate:

- Premier arc	forme fléchie du verbe principal => registre $A.
- Deuxième arc	position de *that* => registre $L
- Troisième arc	bornes du syntagme => registres $B et $C
- Quatrième arc	position du verbe => registre $R
	forme canonique du verbe même => registre $D

Les trois paramètres de correction ont les valeurs suivantes:

1. Dans le message d'erreur, tout le texte situé entre les caractères de contrôle '\u' et '\v' apparaît en souligné. La chaîne '\$A' est remplacée par la valeur du registre, c'est-à-dire le verbe principal. Des sauts de lignes sont également insérés lorsque le message contient le caractère de contrôle '\n'.

2. Le schéma numéro 2, 'remplacement', est activé.

3. Dans la liste de mots présentée à l'utilisateur, '\($B-$C)' est remplacé par le syntagme nominal identifié dans le deuxième arc et '\$D' est remplacé par le verbe qui suit le syntagme.

Ainsi, si la phrase contenue dans le texte est la suivante:

> * *I expect **that** the home team **wins** tonight.*
> Correct: *I expect the home team to win tonight.*

Le message d'erreur devient:

> Le verbe expect ne se construit d'habitude pas avec that.
> Utilisez plutôt la forme avec l'infinitif complet, comme dans:
> We expect your brother to come tomorrow.

Le schéma de remplacement est exécuté, ce qui active la touche 'Remplacer' sur laquelle l'utilisateur peut appuyer s'il désire effectuer l'opération. Dans ce cas, l'algorithme remplace le segment de phrase défini par les registres $L et $R (*that the home team wins*) par l'élément de la liste (*the home team to win*), sélectionné par défaut lorsque la liste ne contient qu'un élément.

5.5. Souplesse des approches disponibles

Le formalisme et l'agencement des automates permettent à ceux-ci de s'appuyer sur un certain nombre de données pour détecter les erreurs. En particulier, les automates peuvent faire appel aux traits syntaxiques des mots, à des listes de mots ayant une ou plusieurs caractéristiques communes, à la position et au contenu des syntagmes

nominaux simples et complexes, ainsi qu'aux formes verbales contenues dans le texte (ces dernières informations étant construites par les automates eux-mêmes). Le problème qui se pose alors est de savoir comment réaliser, à l'aide de ces nombreuses ressources, des automates de détection adaptés aux erreurs à détecter. Les différentes approches que nous présentons ci-dessous correspondent chacune à un choix de moyens à mettre en jeu. Ainsi, l'approche la plus simple (suite de mots isolés) ne demande aucun mécanisme spécial. Dans d'autres cas, on fait appel aux syntagmes nominaux et aux formes verbales. Ces divers automates contiennent des suites de conditions attachées aux connaissances linguistiques obtenues lors d'une phase de pré-traitement syntaxique. Nous présumons donc que le texte ne contient plus de mots inconnus du logiciel et que les problèmes relatifs à ceux-ci ont été résolus lors de la phase de traitement lexical.

Les approches que nous introduisons ici représentent des échantillons isolés le long d'un continuum allant du cas le plus simple, celui d'un automate recherchant une suite de mots, au cas le plus compliqué, qu'il est encore difficile d'imaginer. Nous avons toutefois jugé important de présenter ces différentes approches afin d'illustrer la richesse du système et la gamme étendue des erreurs qu'il peut traiter.

Les exemples qui servent à illustrer les niveaux d'approche sont en grande partie extraits de la typologie des erreurs établie par le laboratoire de traitement du langage et de la parole de l'université de Neuchâtel.

5.5.1. Suite de mots

Cette situation inclut toutes les suites contiguës de mots divers, ces suites ne représentant pas d'entités syntaxiques (comme les syntagmes nominaux, par

exemple). L'erreur même peut se trouver au début, au milieu ou à la fin de cette suite, comme l'illustrent les exemples suivants:

a. * *He is travelling **for** business.* (Correct: *on*)
b. * *It is the best **which** we **have** seen.* (Correct: *... the best we...*)
c. * *He is not as small **that** you think.* (Correct: *as*)

Dans chacun de ces exemples, c'est une suite bien précise de mots qui conduit à une erreur. Un des avantages de cette approche est que son utilisation ne requiert que des connaissances relatives à chaque mot de la suite et des connaissances extraites directement du dictionnaire d'analyse. L'inconvénient majeur se situe ici dans le fait que la suite entière est codée dans l'automate. Si l'erreur à traiter apparaît dans un contexte différent, celle-ci ne sera pas identifiée. Mais ce problème peut également être un avantage, car ce type d'automate ne se déclenchera pas par erreur.

L'automate suivant correspond à cette approche:

```
AUTOMATE 16
[FC='as'] @[CAT=ADJ] [FC='that']
```

Explication:

- L'automate est ancré sur un adjectif.
- Il recherche le mot *as* à gauche de l'ancrage et *that* à droite.
- Il se déclenche pour des suites de mots comme * *as small **that***. (Correct: *as small as*)

5.5.2. Erreur à l'intérieur d'un syntagme nominal

Ici il est nécessaire pour les automates d'opérer dans un domaine syntaxique, comme les syntagmes nominaux. Lorsque c'est le cas, les arcs n'en traversent jamais les limites.

L'exemple suivant illustre l'avantage de travailler à l'intérieur d'un syntagme:

> * *He didn't like his **grandmothers** house.*
> Correct: *He didn't like his grandmother's house.*

Si un syntagme nominal qui comprend *his grandmothers house* peut être identifié dans la phase préliminaire, alors la vérification de l'erreur peut se faire de manière relativement simple par l'automate suivant:

```
AUTOMATE 17
<SN  [CAT=N, NBR=PLUR]   @[CAT=N, EPOS=ELAS] >
```

Explication:

- L'automate est ancré sur un nom qui se trouve en dernière position à l'intérieur d'un syntagme nominal. La condition EPOS=ELAS vérifie ceci.
- Il recherche un nom au pluriel à gauche de l'ancrage.

Comme nous l'avons vu plus haut, l'identification des syntagmes nominaux n'est pas une opération simple. Si l'algorithme possède des lacunes, ce qui n'est pas toujours évident, les automates qui se basent sur les syntagmes risquent de ne pas fonctionner

de la manière attendue.

5.5.3. Point de référence à l'extérieur du syntagme nominal

Il arrive souvent qu'un SN soit correct en lui-même, mais que la présence en dehors de celui-ci de certains mots le rende incorrect. Considérons l'exemple ci-dessous:

* *He broke **the** leg skiing.*
Correct: *He broke his leg skiing.*

Le syntagme nominal *the leg* est affecté par la présence d'un élément extérieur: le mot *broke*. On distingue donc ici deux groupes de conditions: celles à l'extérieur des contextes et celles à l'intérieur. Les conditions extérieures aux SN peuvent apparaître à gauche ou à droite de celui-ci et sont spécifiées de manière habituelle.

Quant aux conditions internes aux contextes, elles peuvent être de deux types. D'une part, les conditions qui s'appliquent au contexte dans son ensemble, et d'autre part les conditions attachées aux mots qui forment le contexte. Ces dernières sont spécifiées entre les symboles '[' et ']'. Les informations globales, relatives au SN dans son ensemble, peuvent comprendre, par exemple, le nombre d'un syntagme nominal ou le temps de l'environnement du verbe.

Conclusion

Dans ce chapitre, nous avons décrit un formalisme de détection d'erreurs en langue seconde basé sur des automates. Ce formalisme possède les caractéristiques suivantes:

- Il donne accès aux données linguistiques requises pour la détection et la correction d'une gamme étendue d'erreurs.
- Il permet une organisation hiérarchisée des règles de détection.
- Il donne au linguiste-informaticien une grande souplesse lors du choix de l'approche à utiliser pour traiter les différents types d'erreurs.
- Il offre la possibilité de construire, lors de la phase de détection, toutes les données nécessaires à la correction.

Dans le chapitre qui suit, nous soumettons un ensemble de phrases-test à des automates conçus pour détecter des erreurs syntaxiques et morpho-syntaxiques de neuf types, allant de la mauvaise construction du passif à l'ordre verbe-adverbe incorrect. Nous analysons les résultats et mettons en évidence les avantages et les inconvénients de l'approche par automates.

Chapitre 6

Evaluation du correcteur basé sur l'approche par automates

Comme ce fut le cas pour le logiciel basé sur la LFG, nous avons soumis le logiciel basé sur l'approche par automates à une évaluation afin de déterminer la qualité des algorithmes que nous avons développés. Un certain nombre de différences entre les deux logiciels nous ont toutefois poussés à modifier quelques aspects de l'évaluation. D'une part nous avions à disposition lors du développement du logiciel ARCTA un important corpus d'erreurs comportant quelques 25'000 mots et nous avons jugé utile de l'incorporer à l'évaluation. D'autre part, le fait que les automates traitent les phrases sans contraintes

syntaxiques nous a permis d'utiliser des phrases-test représentatives de textes réels.

Ainsi que nous l'avons mentionné au chapitre précédent, la souplesse des automates fait qu'ils peuvent être utilisés pour détecter un nombre de catégories d'erreurs beaucoup plus important que le modèle LFG. Bien que l'envergure du projet, au cours duquel les automates et le logiciel furent développés, ait permis de traiter un grand nombre de ces catégories, nous nous sommes concentrés uniquement sur des erreurs liées à des phénomènes morpho-syntaxiques et syntaxiques.

Ce chapitre débute par une description des erreurs sur lesquelles porte notre évaluation et des règles de détection et de correction que nous avons développées (section 6.1.). Nous présentons ensuite le processus d'évaluation que nous avons utilisé (6.2.). Nous terminons le chapitre par une analyse des résultats, suivie d'une discussion des avantages et inconvénients de l'approche par automates (6.3.).

6.1. Catégories d'erreurs et règles de détection

Comme pour le logiciel LFG, nous avons sélectionné des catégories d'erreurs qui sont représentatives des problèmes rencontrés par des francophones écrivant en anglais, et qui illustrent particulièrement bien les possibilités de détection et de correction des automates. La liste ci-dessous contient les 9 catégories qui font l'objet de l'évaluation:

1. Accord sujet-verbe (Accord S-V)

2. Construction du passif (Passif)
3. Préposition dans les compléments de temps (Prép. temp.)
4. Utilisation du temps continu (Cont/simple)
5. Ordre sujet-verbe dans le discours indirect (Ordre S-V)
6. Forme du verbe dans les propositions infinitives (Prop. Inf.)
7. Accord du nom (Accord N)
8. Temps des verbes avec des compléments temporels (Compl. temp.)
9. Position des adverbes par rapport aux verbes (Ordre adv-V)

Nous avons ensuite développé et vérifié les automates en utilisant trois sources de textes:

- Le corpus ARCTA, décrit dans la section 6.2.1., qui nous a permis d'étudier les erreurs et d'affiner les automates.
- Un récit de Sir Arthur Conan Doyle, "The Adventure of the Speckled Band" (Sherlock Holmes), et un ensemble d'articles de revues anglaises, totalisant plus de 30'000 mots, qui nous ont permis de vérifier que les automates ne s'arrêtaient pas sur des non-erreurs.

Les sections qui suivent décrivent chaque catégorie d'erreurs, présentent des exemples d'automates et précisent leur agencement. Il est à noter que nous avons personnellement développé les automates des catégories 1 à 5 (voir annexe 6), alors que ceux des catégories 6 à 9 ont été élaborés par d'autres membres de l'équipe (C. Tschumi, N. Kübler, L. Grosjean et C. Tschichold). Chaque section indique également les codes qui correspondent aux catégories d'erreurs répertoriées dans la typologie établie par le laboratoire de traitement du langage et de la parole de l'université de Neuchâtel.

6.1.1. **Accord sujet-verbe** (Accord S-V)

Codes: VAccSing/Plur, VAccPlur/Sing

a. Description

Cette catégorie traite des erreurs d'accord entre le verbe et son sujet. Les automates recherchent l'oubli de '-s' ou de '-es' à la troisième personne du singulier du présent ainsi que l'insertion de '-s' ou de '-es' aux autres personnes du présent. Les deux phrases ci-dessous illustrent ces erreurs:

* *My mother* ***watch*** *TV after dinner.*
(Correct: *My mother watches TV after dinner.*)
* *Children* ***likes*** *chocolate.*
(Correct: *Children like chocolate.*)

Ne sont pas traités les cas où un 's' est ajouté par erreur à un verbe à l'infinitif, comme dans:

* *I saw the child* ***eats*** *chocolate.*
(Correct: *I saw the child eat chocolate.*)

b. Agencement

La figure 6-1 (page suivante) illustre l'organisation des automates. Ceux-ci sont regroupés en sept ensembles. Les trois premiers ensembles (101 à 103) recherchent le '-s' ou '-es' manquant:

- L'ensemble 101 traite des structures du type SN1+préposition+SN2+V, où l'accord du verbe se fait avec le premier SN.
- L'ensemble 102 traite des structures SN+V et SN+pronom relatif+V.
- L'ensemble 103 traite des phrases interrogatives.

```
101  ( VACC_MANQ_ES_PREP_1  VACC_MANQ_ES_PREP_2 )

REPERAGE
102    ( VACC_MANQ_ES_1_R  VACC_MANQ_ES_2_R
         VACC_MANQ_ES_3_R  VACC_MANQ_ES_4_R )
  {$V=R1} VACC_MANQ_ES_DET
END

103 ( VACC_MANQ_ES_Q_1A VACC_MANQ_ES_Q_1B VACC_MANQ_ES_Q_2
      VACC_MANQ_ES_Q_3A VACC_MANQ_ES_Q_3B )

REPERAGE 104 ( VACC_S_ES_1_R  VACC_S_ES_2_R
               VACC_S_ES_3_R  VACC_S_ES_4_R )
    {$V=R1} VACC_S_ES_DET
END

REPERAGE 105 ( VACC_S_ES_1W_R  VACC_S_ES_2W_R
               VACC_S_ES_3W_R  VACC_S_ES_4W_R )
    {$V=R1} VACC_S_ES_W_DET
END

106  (  VACC_S_ES_AND_1 VACC_S_ES_AND_2 VACC_S_ES_PREP_1
        VACC_S_ES_PREP_2 VACC_S_ES_PREP_1_W VACC_S_ES_PREP_2_W )

107  (  VACC_S_ES_Q_1A VACC_S_ES_Q_1B VACC_S_ES_Q_1C
        VACC_S_ES_Q_2  VACC_S_ES_Q_3A VACC_S_ES_Q_3B
        VACC_S_ES_Q_4A VACC_S_ES_Q_4B
        VACC_S_ES_Q_5A VACC_S_ES_Q_5B (
```

Figure 6-1. Agencement des automates de la catégorie Accord S-V.

Les quatre autres ensembles (104-107) recherchent un '-s' ou '-es' superflu:

- L'ensemble 104 traite des structures SN+V et SN+pronom relatif+V.
- L'ensemble 105 fait de même mais pour les cas où le verbe est *was*.
- L'ensemble 106 traite des structures du type SN1+préposition+SN2+V où l'accord du verbe se fait avec le premier SN, et des structures du type SN1+*and*+SN2+V où le verbe se met au pluriel.
- L'ensemble 107 traite des phrases interrogatives.

Les ensembles 102, 104 et 105 contiennent des groupes de repérage dont la fonction est de séparer la partie détection de l'erreur (listes de repérage) de la partie correction afin de réduire la complexité des automates de détection.

c. Automates

Examinons quelques automates de la catégorie Accord-SV:

Exemple 1

```
AUTOMATE VACC_MANQ_ES_3_R
  (R1=>$V)
    [FC/='and', FC/='nor', CAT/=PREP]
    <SN (PERS_SN=P3, NBR_SN=SING)>
    [FC='which' | FC='that' | FC='who']
   @[CAT=V, STATUT_FV/=INT, TNS=PRES, ~V_3P,
          ~V_HOMO_3S, MORPH/=M_3S, FF=>$A]
```

⇒ L'automate détecte les erreurs du type:

* *I think that the girl who **drive** this car should be more careful.*

(Correct: *I think that the girl who drives this car should be more careful*)

⇒ L'automate est ancré sur un verbe qui satisfait aux conditions suivantes:

- ne se trouve pas dans une phrase interrogative.
- est au présent.
- ne se trouve pas dans la liste V_3P, qui contient les verbes n'ayant pas de '-s' à la troisième personne du singulier au présent, comme *can* et *will*.
- ne se trouve pas dans la liste V_HOMO_3S, qui contient les verbes comme *to fell* et *to saw*, dont la forme de base est identique à celle d'autres verbes (*I fell* et *he saw*, dans notre exemple).
- n'est pas à la troisième personne du singulier.

⇒ La forme fléchie du verbe est mise dans le registre $A, dont la valeur est utilisée lors de la correction.

⇒ L'automate recherche ensuite un des pronoms relatifs *which*, *that* ou *who*, puis un syntagme nominal à la troisième personne du singulier.

⇒ La condition suivante s'assure du fait que ni *and*, *nor* ou une préposition ne précède le syntagme nominal.

⇒ Si toutes les conditions sont satisfaites, la valeur R1 est affectée au registre $V, ce qui permet le passage à l'automate de détection par l'entremise du mécanisme de repérage.

Exemple 2

```
AUTOMATE VACC_MANQ_ES_DET
   (+EXCLUDE)
     @[CAT=V]
     ^[FF/='?', POS=LAS]
     {
       SCHEMA 1
       MESSAGE "Le verbe \u\$A\v est à la mauvaise"
        " personne. Nous vous suggérons de le mettre "
```

```
        " à la troisième personne du singulier."
    }
```

⇒Cet automate de correction est exécuté lorsqu'un des automates de détection faisant partie de la liste de repérage a détecté une erreur (voir agencement).

⇒L'ancrage est automatiquement défini comme étant le même que celui de l'automate de détection.

⇒L'automate s'assure que la phrase ne se termine pas par un point d'interrogation.

⇒La condition '+EXCLUDE' empêche d'autres automates de détection de s'arrêter sur le même verbe.

Exemple 3

```
AUTOMATE VACC_MANQ_ES_Q_1A
  <SN @[CURPOS=1, +WH_PRON] >
 @[CAT=V , (+AUX | +MODAUX | FC='do'),
        TNS=PRES, ~V_3P,
        ~V_HOMO_3S, MORPH/=M_3S, FF=>$A ]
  [CAT=ADV]*
  <SN (PERS_SN=P3, NBR_SN=SING) >
  [CAT=ADV]*
  [CAT=V]
  [
       SCHEMA 1
       MESSAGE "Le verbe \u\$A\v est à la mauvaise"
        " personne. Nous vous suggérons de le mettre "
        " à la troisième personne du singulier."
  ]
```

⇒L'automate détecte l'erreur d'accord dans les phrases du type:

* *What toy **do** he prefer?*

(Correct: *What toy does he prefer?*)

⇒L'automate est ancré sur un verbe modal, auxiliaire ou sur *do*.

⇒Les mêmes conditions que dans l'exemple 1 ci-dessus s'appliquent au verbe.

⇒A gauche du verbe, l'automate recherche un syntagme nominal apparaissant au début des phrases interrogatives, comme *which color*.

⇒A droite du verbe, l'automate recherche un syntagme nominal à la troisième personne du singulier suivi d'un verbe (le verbe principal dans ce cas), tout en sautant les adverbes.

Exemple 4

```
AUTOMATE VACC_S_ES_2_R
  (R1=>$V)
  <SN @[EPOS=EFIR, POS=FIR] (PERS_SN/=P3 | NBR_SN=PLUR)>
 @[CAT=V, TNS=PRES, ~V_3P, MORPH=M_3S, FF=>$A]
```

⇒L'automate détecte l'erreur d'accord dans les phrases du type:

* *We **doesn't** wear shoes.*

(Correct: *We don't wear shoes.*)

⇒L'automate est ancré sur un verbe à la troisième personne du singulier du présent.

⇒L'automate recherche ensuite un syntagme nominal en début de phrase qui n'est pas à la troisième personne du singulier.

6.1.2. Construction du passif (Passif)

Code: VMorPassif

a. Description

Cette catégorie traite de l'utilisation du mauvais temps pour le verbe principal à l'intérieur d'une forme passive, comme dans l'exemple suivant:

* *The lost tape was **discover** under the bench.*
(Correct: *The lost tape was discovered under the bench.*)

Dans cet exemple, la forme de base *discover* est utilisée au lieu du participe passé *discovered.*

b. Agencement

La catégorie ne contient qu'un automate et ne nécessite donc pas d'agencement.

c. Automates

L'automate suivant détecte les erreurs de la catégorie:

```
AUTOMATE VMORPASSIF
  @[CAT=V, FC='be', CURPOS=>$L]
   [FF='not']*1
   [CAT=ADV, FF/='not', FC/='to']*1
   [CAT=V, TNS=BASE, ~V_PARTBASE, ~V_HOMO_3S,
         CURPOS=>$R, FF=>$A]
```

```
{
    SCHEMA 1
    MESSAGE
     "La forme passive du verbe \u\$A\v est "
     "peut-être mal formée dans la séquence "
     " \u\($L-$R)\v. Nous vous suggérons de la "
     "remplacer par le participe passé."
}
```

⇒L'automate est ancré sur une des formes du verbe *to be*. La position du verbe à l'intérieur de la phrase est mise dans le registre $L.

⇒L'automate recherche ensuite la forme de base d'un verbe tout en sautant les adverbes. Le verbe ne doit se trouver ni dans la liste V_PARTBASE, qui contient les verbes dont la forme de base ainsi que le participe passé sont identiques (comme *to come*), ni dans la liste V_HOMO_3S.

⇒La forme fléchie du second verbe est mise dans le registre $A et sa position dans le registre $R.

⇒Le schéma numéro 1 (message simple) est appelé lors de la correction.

6.1.3. Préposition dans les compléments de temps (Prép. temp.)

Code: EnoMgramPrepTemp

a. Description

Cette catégorie traite du mauvais choix des prépositions *while, for, during* et *since* dans les compléments de temps, comme dans les exemples ci-dessous:

* *He remained there **during** about twenty hours.*
(Correct: *He remained there for about twenty hours.*)

* *He has been sitting here **since** three hours.*
(Correct: *He has been sitting here for three hours.*)

Sont exclus de la catégorie les cas où le choix de la préposition dépend du verbe principal, comme dans l'exemple suivant:

* *He **lived** in Paris **since** the month of October.*
(Correct: *He lived in Paris during the month of October.*)

b. Agencement

La figure 6-2 illustre l'agencement des automates de la catégorie. Ceux-ci sont répartis dans deux ensembles, l'un traitant de la préposition *during* (301) et l'autre des prépositions *while*, *since* et *for* (302).

```
REPERAGE
  301     ( ENOPREP_DURING_R )
    ()      ENOPREP_DURING_D
END

302 ( ENOPREP_WHILE
      ENOPREP_SINCE_1
      ENOPREP_SINCE_2
      ENOPREP_FOR_2  )
```

Figure 6-2. Agencement de la catégorie Prép. Temp.

c. Automates

Dans le premier ensemble (301), un automate de détection identifie la suite *during*+SN temporel pluriel. Le succès de cet automate déclenche un autre

automate qui s'assure que le syntagme nominal ne débute pas par *the*, *this*, *that*, *these* ou *those*. Les deux automates sont représentés ci-dessous:

```
AUTOMATE ENOPREP_DURING_R
  @[FF='during', CURPOS=>$L, CURPOS=>$R]
   [FF='almost' | FF='nearly' | FF='about']*1
   <SN @[+HEADSN, +TEMP, NBR=PLUR]>

AUTOMATE ENOPREP_DURING_D
  @[]
   [FF='almost' | FF='nearly' | FF='about']*1
   <SN @[EPOS=EFIR, FF/='the', FF/='these', FF/='those',
            FF/='this', FF/='that']>
   {
    SCHEMA 2
    MESSAGE "La préposition \uduring\v ne semble pas"
         "correcte dans ce contexte. Nous vous "
         " suggérons de la remplacer par \ufor\v."
    LISTE "for"
  }
```

⇒Le premier automate est ancré sur la préposition *during*. La position de la préposition est mise dans les registres $L et $R.

⇒Le second automate contient les données utilisées pour la correction: le schéma 2 (remplacement) est exécuté et la liste comprenant *for* apparaît à l'écran. Si l'utilisateur sélectionne *for* ou appuie sur la touche 'Remplacer', *for* est inséré dans le texte à la place de *during*. L'exclusion de *the*, *these*, etc. du SN dans le second automate empêche celui-ci de s'arrêter sur des phrases correctes telles que *It rained during the week we were in Vancouver*.

Le second ensemble d'automates (302) traite de *while*, *since* et *for*. Ceux-ci recherchent la suite préposition+SN, comme dans l'exemple ci-dessous:

```
AUTOMATE ENOPREP_SINCE_1
  @[FF='since', CURPOS=>$L, CURPOS=>$R]
   <SN @[+HEADSN, +TEMP, NBR=PLUR]
        (TEMP_TYPE/=DATE, TEMP_TYPE/=TEMPS_PRECIS)>
  {
    SCHEMA 2
    MESSAGE "La préposition \usince\v ne semble "
        " pas correcte dans ce contexte. "
        "Nous vous suggérons de la remplacer par \ufor\v."
    LISTE "for"
  }
```

⇒L'automate détecte les erreurs du type:

* *He has been sitting here **since** three hours.*
(Correct: *He has been sitting here for three hours.*)

⇒Il recherche la préposition *since* suivie d'un syntagme nominal au pluriel, ayant le trait 'temporel', et ne représentant ni une date, ni un temps précis, mais une durée par exemple, comme dans *three hours*.
⇒Le mécanisme de correction est similaire au premier exemple de la catégorie.

6.1.4. Utilisation du temps continu (Cont/simple)

Code: VTempCont/Simp

a. Description

Cette catégorie traite de l'utilisation, à un temps continu, de verbes qui ne le permettent pas, comme *to love*, *to last*, *to seem* et *to like*. Sont exclus de la catégorie les cas où le contexte sémantique détermine si le verbe doit se mettre à un temps continu ou à un temps simple. Ainsi, l'erreur contenue dans la phrase suivante fait partie de la catégorie:

* *He **is seeming** quite upset these days.*
(Correct: *He seems quite upset these days.*)

Mais l'erreur ci-dessous n'est pas traitée:

* *A machine **is only doing** what it is told to do.*
(Correct: *A machine only does what it is told to do.*)

b. Agencement

La catégorie ne contient qu'un automate et ne nécessite donc pas d'agencement.

c. Automates

L'automate suivant détecte les erreurs de la catégorie:

```
AUTOMATE VTEMPCONT_1
  @[CAT=V, ASPECT_FV=CONT, +V_NEVERCONT, FC=>$A]
```

```
{
  SCHEMA 1
  MESSAGE "Le verbe \u\$A\v ne se met généralement "
        "pas à la forme continue. "
        "Nous vous suggérons de le mettre à la forme simple."
}
```

⇒L'automate recherche un verbe qui est le verbe principal d'une forme verbale à un temps continu et qui se trouve dans la liste des verbes qui ne se mettent en principe pas à la forme continue.

⇒Le schéma 1 (message simple) est exécuté.

6.1.5. Ordre sujet-verbe dans le discours indirect (Ordre S-V)

Code: EnoOrdInter/Indi

a. Description

Cette catégorie traite des erreurs d'inversion de l'ordre sujet-verbe dans le discours indirect utilisé avec les adverbes *when, how, where, what* et *why*. L'exemple suivant illustre une erreur:

* *He asked me when* ***had the incident occurred.***
(Correct: *He asked me when the incident had occurred.*)

Parmi les cas non traités, nous trouvons les phrases contenant un pronom interrogatif qui est aussi pronom relatif, comme *which* et *who*, et celles où le discours indirect libre est utilisé. Exemples:

* *He asked me who **did I see** last night.*

(Correct: *He asked me who I saw last night.*)

* *She asked me: "**you are** going to the movie?".*

(Correct: *She asked me: "are you going to the movie?".*)

b. Agencement

La figure 6-3 contient l'agencement pour la catégorie Ordre S-V. Celui-ci contient deux groupes d'automates, le premier (501) traite les phrases avec *how*, *where*, *when* et *why*, et le second (750) celles contenant *what*. Chaque groupe est formé d'un bloc de repérage dont les automates détectent l'erreur. La différence entre les trois automates contenus dans les blocs de repérage réside dans les structures verbales qu'ils recherchent. Dans le second groupe, nous utilisons l'interaction pour nous assurer que le SN est bien le sujet.

```
REPERAGE 501 ( ENOORD_INTERINDI_R1
               ENOORD_INTERINDI_R2
               ENOORD_INTERINDI_R3 )
     () ENOORD_INTERINDI_D
END

REPERAGE 750 ( ENOORD_INTERINDI_W1
               ENOORD_INTERINDI_W2
               ENOORD_INTERINDI_W3 )
     () ENOORD_INTERINDI_I
     ($Q=R1) ENOORD_INTERINDI_QI
     ($Q=R2) MERCI
END
```

Figure 6-3. Agencement des automates de la catégorie Ordre S-V.

c. Automates

L'automate suivant fait partie du premier bloc de repérage (501):

```
AUTOMATE ENOORD_INTERINDI_R1
  @[(CAT=PRON, +WH_PRON, FC/='which', FC/='who',
        FC/='that', FC/='what')
        | (FC='how') | (FC='where'), POS/=FIR]
   [CAT=V, CURPOS=>$L, CURPOS=>$R]
   <SN (FIRSTPOS=>$X, LASTPOS=>$Y)>
   ^[FF/='?', POS=LAS]
```

⇒L'automate est ancré sur *how*, *where*, *when* ou *why*.

⇒Il recherche ensuite un verbe, dont il enregistre la position dans les registres $L et $R, puis un syntagme nominal dont il enregistre la position dans les registres $X et $Y.

⇒Le dernier arc vérifie que la phrase n'est pas interrogative.

L'automate de correction est le suivant:

```
AUTOMATE ENOORD_INTERINDI_D
  @[]
  {
   SCHEMA 5
   MESSAGE "La séquence \u\($L-$R) \($X-$Y)\v semble "
    "faire partie d'une question indirecte, comme "
    " dans:\n"
    "\uWe do not know when you are leaving.\v\n"
    "Si c'est le cas, nous vous conseillons de permuter"
    "\u\($L-$R)\v et \u\($X-$Y)\v."
  }
```

⇒L'automate est exécuté lorsqu'un des trois automates du bloc de repérage passe.

⇒Le schéma 5 (permutation) est exécuté. Les deux segments de phrase à permuter sont définis dans les automates de détection par les registres $L et $R (pour le verbe) et $X et $Y (pour le sujet).

La présence de *what* requiert un traitement particulier au niveau de la correction. En effet, des phrases comme celle ci-dessous sont correctes:

I don't know what caused the accident.

Alors que la suivante contient une erreur:

** I don't know what **has he** done.* (Correct: ... *what he has done.*)

Les trois automates du bloc de repérage sont les mêmes dans le cas de *what* que dans les autres cas. Par contre, la phase de correction fait appel aux trois automates suivants:

```
AUTOMATE ENOORD_INTERINDI_I
  @[FC='what']
  {
    SCHEMA 6
    MESSAGE "Est-ce que dans cette phrase \u\($X-$Y)\v"
            " est le sujet du verbe \u\($L-$R)\v?"
  }
```

```
AUTOMATE ENOORD_INTERINDI_Q1
  @[FC='what']
  {
    SCHEMA 1
    MESSAGE "Dans ce cas nous vous suggérons de "
      "permuter \u\($X-$Y)\v et \u\($L-$R)\v"
  }

AUTOMATE MERCI
  @[ ]
  {
        SCHEMA 9
        MESSAGE "Merci! Rien à corriger."
  }
```

⇒Le premier automate, qui passe toujours, appelle le schéma d'interaction. L'utilisateur répond ensuite 'oui' ou 'non' en appuyant sur une touche. Cette action affecte la valeur R1 au registre $Q pour 'oui' et R2 pour 'non'.

⇒Le second automate est exécuté si $Q=R1.

⇒Le troisième automate est exécuté si $Q=R2.

6.1.6. Forme du verbe dans les propositions infinitives (Prop. Inf.)

Codes: VSuivCompl/Pron+Inf, VSuivF_ing/Inf, VSuivBase/Inf

Auteur: N. Kübler

a. Description

Cette catégorie traite de trois cas de mauvaise construction des formes verbales dans les propositions infinitives. Les types d'erreurs détectées par les automates sont les suivants:

1. *That* au lieu de l'infinitif pour les verbes *to want*, *to like* et *to love*, comme dans la phrase suivante:

> * *They wanted **that we came** earlier.*
> (Correct: *They wanted us to come earlier.*)

2. Oubli de 'to'. Exemple:

> * *It all seemed__be under control.*
> (Correct: *It all seemed to be under control.*)

3. Gérondif au lieu de l'infinitif. Exemple:

> * *I intend **going** to the movies tonight.*
> (Correct: *I intend to go to the movies tonight.*)

b. Agencement

L'agencement comprend deux listes: une pour les erreurs relatives à *that* (type 1) et une pour celles relatives à *to* (types 2 et 3):

```
601 ( VSUIV_COMPL_THAT_1 VSUIV_COMPL_THAT_2 )
602 ( TO_INF_V_1  TO_INF_V_2 )
```

c. Automates

Considérons, par exemple, l'automate suivant qui détecte les erreurs du premier type:

```
AUTOMATE VSUIV_COMPL_THAT_1
  (+EXCLUDE)
 @[CAT=V, +V_NOT_THAT, FF=>$A]
  [FF='that', CURPOS=>$L]
  <SN (FIRSTPOS=>$B, LASTPOS=>$C) @[+HEADSN, CAT/=PRON] >
  [CAT=V, ~MODAUX, CURPOS=>$R, FC=>$D]
  {
    SCHEMA 2
    MESSAGE "Le verbe \u\$A\v ne se construit "
        "généralement pas avec \uthat\v. Nous vous "
        "suggérons d'utiliser plutôt la forme avec "
        "l'infinitif complet, comme dans:\n"
        "\uWe expect you to come tomorrow.\v"
    LISTE "\($B-$C) to \$D"
  }
```

⇒L'automate est ancré sur un verbe appartenant à la liste des verbes qui ne se construisent généralement pas avec *that*, comme *to want*, *to like* et *to love*.

⇒Il recherche ensuite le mot *that*, suivi d'un syntagme nominal qui n'est pas composé d'un pronom et d'un verbe non modal.

L'automate ci-dessous traite des erreurs où le gérondif est utilisé au lieu de l'infinitif (type 3):

```
AUTOMATE TO_INF_V_1
   @[CAT=V, +TO_INF_V, FC=>$E]
```

```
    [CAT=V, TNS=ING, FF=>$D, FC=>$F, CURPOS=>$L,
CURPOS=>$R]
{
SCHEMA 2
MESSAGE "Le verbe \u\$E\v se construit avec \uto "
        "+ infinitif\v; nous vous suggérons de "
        "remplacer \u\$D\v par \uto \$F\v."
LISTE "to \$F"
}
```

⇒L'automate est ancré sur un verbe appartenant à la liste de ceux qui ne s'utilisent en principe pas avec le gérondif, comme *to hear*, *to seem* et *to intend*.

⇒L'automate recherche ensuite un verbe au gérondif.

6.1.7. Accord du nom (Accord N)

Codes: NAccDetPron, NAcc"all", NAccNomb, NAccQuan, NAccDiv

Auteurs: C. Tschumi et L. Grosjean

a. Description

Cette catégorie traite des erreurs d'accord du nom à l'intérieur du syntagme nominal, en particulier avec les déterminants *every*, *each*, *few*, *many*, *this*, *these*, et *all*. Exemple:

* *Every **players** has agreed to come that day.*

(Correct: *Every player has agreed to come that day.*)

b. Agencement

Cette catégorie comprend 46 automates. Ceux-ci sont répartis dans 9 ensembles, qui traitent chacun de problèmes spécifiques. Deux de ces ensembles font appel au mécanisme d'interaction, pour traiter d'erreurs relatives à *few* dans le premier cas et à *many* dans le second.

c. Automates

Les automates de cette catégorie ont tous la même structure:

⇒Ils opèrent à l'intérieur d'un syntagme nominal qui n'est pas du type SN+préposition+SN.
⇒Ils s'ancrent sur un déterminant.
⇒Ils recherchent le nom principal du syntagme et vérifient certaines conditions sur celui-ci.

Par exemple, l'automate suivant détecte les erreurs d'accord du nom avec *much*:

```
AUTOMATE NACC_PLUR_SING_1                    ! much jobs
  <SN (ENV_CL/=SN_COMPL)
     @ [FF='much', CURPOS=>$L, CURPOS=>$R]
     ^[CAT=N, FF=>$F, EPOS=ELAS, NBR=PLUR, CURPOS=>$Z] >
  {
    SCHEMA 2
    MESSAGE "Il semble qu'il y ait un problème d'accord"
      " dans \u\($L-$Z)\v. \uMuch\v est généralement "
      " suivi d'un nom singulier. Nous vous suggérons de "
      "mettre \u\$F\v au singulier, ou de remplacer "
      "\umuch\v par \umany\v."
    LISTE "many"
```

```
        FICHE Mogram_quant_MUCH
    }
```

⇒L'automate est ancré sur *much*.

⇒Il passe si le nom en dernière position à l'intérieur du syntagme est au pluriel.

6.1.8. Temps des verbes avec des compléments temporels (Compl. temp)

Code: VTempPret/Plupar"for"

Auteur: C. Tschumi

a. Description

Cette catégorie traite des erreurs relatives au temps des verbes ayant un complément de temps construit avec *for* ou *since*. Contrairement à la catégorie 3 (Prép. temp.), qui traite des erreurs à l'intérieur du complément de temps, cette catégorie traite des cas où le temps du verbe principal dépend du complément de temps. Exemple:

* *I live here for two years.*
(Correct: *I have been living here for two years.*)

b. Agencement

Les automates de cette catégorie sont répartis dans deux groupes, utilisant chacun un bloc de repérage et le mécanisme d'interaction avec l'utilisateur. Le premier groupe traite des verbes qui ne se mettent d'habitude pas à la forme progressive, et le second des verbes qui acceptent la forme progressive. Le mécanisme

d'interaction permet ici d'obtenir des informations de la part de l'utilisateur au sujet du sens du complément de temps.

c. Automates

L'automate suivant recherche une phrase qui contient à la fois un verbe à la forme continue et un complément de temps construit avec la préposition *for*.

```
AUTOMATE VTEMP_FOR_2_REP
    ^[FF='was', CURPOS=>$L]
   @[CAT=V, TNS_FV=PRET, ASPECT_FV=CONT, ~V_NEVERCONT,
       CURPOS=>$R, FC=>$E]
    ^[FF='for']
    <SN (SN_TYPE=SN_TEMP)>
```

⇒L'automate est ancré sur un verbe au prétérit appartenant à une forme verbale ayant l'aspect 'continu'. Le verbe doit également ne pas faire partie de la liste des verbes qui ne se mettent en principe pas à la forme continue.

⇒L'automate recherche ensuite la préposition *for* suivie d'un syntagme nominal ayant des caractéristiques temporelles.

⇒Lorsque cet automate passe, un automate d'interaction demande à l'utilisateur le sens du complément de temps. La réponse de l'utilisateur fait démarrer un automate de correction dont le seul but est d'afficher le message d'erreur approprié.

6.1.9. Position relative adverbes/verbes (Ordre adv-V)

Codes: AdvOrdFreq, AdvOrd

Auteur: C. Tschichold

a. Description

Cette catégorie traite des erreurs de mauvais positionnement des adverbes par rapport au verbe, comme dans l'exemple suivant:

> * *I go **always** to the zoo on sunny weekends.*
> (Correct: *I always go to the zoo on sunny weekends.*)

Les locutions adverbiales, comme *last night*, ne sont pas traitées.

b. Agencement

L'agencement des automates est illustré dans la figure 6-4.

```
REPERAGE
710   ( ADV_ORD_REP )
!          ()        ADV_ORD_0
          ($Q=R2)    (  ADV_ORD_1    ADV_ORD_2    ADV_ORD_3
                        ADV_ORD_4    ADV_ORD_5    ADV_ORD_6
                        ADV_ORD_7    ADV_ORD_8    ADV_ORD_9
                        ADV_ORD_10   )
END
```

Figure 6-4. Agencement des automates de la catégorie Ordre adv-V.

L'exécution des automates se fait en trois étapes. Premièrement, l'automate de repérage ADV_ORD_REP recherche un adverbe faisant partie de la liste des adverbes traités, comme *seldom*, *often*, et *normally*. En cas de succès, l'automate affecte la valeur R2 au registre $Q. Ensuite, l'automate ADV_ORD_0 identifie certaines structures de phrase à exclure, comme, par exemple, celles où l'adverbe est précédé d'une forme non-infinitive de *to be*. Si c'est le cas, l'automate affecte la valeur R1 au registre $Q. Enfin, la liste des automates de détection est activée si la valeur du registre est R2, c'est-à-dire si un adverbe a été trouvé et qu'il n'apparaît pas dans une structure de phrase exclue.

c. Automates

Considérons l'automate suivant, qui détecte l'erreur dans la phrase **So I now can go*:

```
AUTOMATE ADV_ORD_4
  <SN>
 @[CAT=ADV, +ADV_POS, FF=>$A, CURPOS=>$L, CURPOS=>$R]
  [CAT=V, (+MODAUX | +AUX)]+2
  [FF='not']*1
  [CAT=V]
{
  SCHEMA 1
  MESSAGE "L'adverbe \u\$A\v se trouve à la mauvaise"
        "place par rapport au verbe principal."
  FICHE adv_pos_categ
}
```

⇒L'automate est ancré sur un adverbe appartenant à la liste ADV_POS des adverbes traités.

⇒Il recherche ensuite, à droite de l'adverbe, un ou deux modaux ou auxiliaires, puis le mot *not*, et enfin le verbe principal.

6.2. Le processus d'évaluation

Une fois les automates écrits et vérifiés, nous avons effectué l'évaluation sur deux ensembles de phrases:

1. Le corpus ARCTA, contenant des textes écrits par des étudiants francophones du niveau secondaire.
2. Des phrases-test écrites spécialement pour l'occasion.

6.2.1. Corpus ARCTA

Le corpus ARCTA est formé de 72 textes qui contiennent en tout près de 1400 phrases. Les textes sont formés de compositions libres et de traductions de documents français (certains pouvant apparaître plusieurs fois). Les conclusions que nous pouvons tirer de cette phase de l'évaluation restent limitées étant donné que les phrases sont celles qui ont été utilisées tout au long du développement pour établir une typologie des erreurs, analyser leur contexte grammatical et créer les automates. Nous avons toutefois jugé important de faire cette première évaluation, car elle nous donne des indications quant au pourcentage d'erreurs traitées et elle illustre les cas où les automates identifient des erreurs qui n'en sont pas.

Les opérations suivantes ont été effectuées sur les phrases du corpus avant de les soumettre au logiciel:

1ère étape: deux enseignants d'anglais ont marqué et classé les erreurs dans le corpus. Le tableau ci-dessous donne la répartition des erreurs dans les différentes catégories qui nous intéressent:

1. Accord S-V	41 erreurs
2. Passif	4
3. Prép. temp.	23
4. Cont./simple	6
5. Ordre S-V	5
6. Prop. Inf.	9
7. Accord N	13
8. Compl. temp.	2
9. Ordre adv.-V	35

Tableau 6-5 - Répartition des erreurs dans le corpus ARCTA

2ème étape: des linguistes ont identifié les syntagmes nominaux simples dans chaque phrase d'après des règles précises.

3ème étape: un module de pré-traitement a découpé les textes en phrases et en mots, puis a affecté à chaque mot les informations le concernant présentes dans le dictionnaire d'analyse CELEX, comme, par exemple, les catégories syntaxiques et les traits morphologiques.

4ème étape: les linguistes ont éliminé les catégories syntaxiques superflues des mots qui en possédaient plus d'une.

5ème étape: nous avons soumis les phrases au logiciel.

Notons que nous avons choisi d'effectuer manuellement le marquage des syntagmes nominaux ainsi que la désambiguïsation plutôt que d'utiliser les modules décrits dans la section 5.1.2. Cette manière de procéder nous permet d'éviter les problèmes dus au malfonctionnement de ces modules.

6.2.2. Corpus de phrases-test

La seconde partie de l'évaluation s'est faite sur un ensemble de phrases écrites spécialement pour l'occasion. Comme pour le logiciel LFG, nous avons utilisé 15 phrases par catégorie d'erreur, 10 contenant une erreur et 5 ne contenant pas d'erreur mais illustrant la difficulté de la catégorie. Ces phrases ont été rédigées par une personne de langue anglaise connaissant le type d'erreurs que font les anglophones lorsqu'ils écrivent en français. Les instructions qui lui ont été données sont présentées dans l'annexe 7. Les phrases sont restées cachées jusqu'au moment de l'évaluation afin de ne pas influencer le développement des automates. Nous avons appliqué la procédure suivante pour obtenir ces phrases, celle-ci étant similaire à celle utilisée pour le logiciel LFG.

1ère étape: nous avons écrit un document contenant des instructions à l'intention de l'auteur des phrases-test (voir l'annexe 7). Ce document présente la démarche à suivre et décrit les catégories d'erreurs à traiter. Il décrit également les contraintes qui s'appliquent à la structure syntaxique des phrases-test. Pour le

logiciel basé sur les automates, ces contraintes sont minimes; elles spécifient uniquement que les syntagmes ne doivent pas contenir de termes d'une complexité sans relation avec les problèmes que le logiciel doit résoudre. Par exemple, l'auteur des phrases-test doit éviter des phrases telles que * *The Compaq 486/33 Mhz PC* ***contain*** *4 Mbytes of RAM.* (Correct: *contains*)

2ème étape: l'auteur a écrit les phrases-test et les a incluses dans un fichier d'ordinateur.

3ème étape: nous avons identifié les syntagmes nominaux simples dans chaque phrase d'après des règles utilisées pour le corpus ARCTA.

4ème étape: un module de pré-traitement a découpé les textes en phrases et en mots puis a affecté à chaque mot des informations extraites du dictionnaire d'analyse.

5ème étape: nous avons éliminé les catégories syntaxiques superflues des mots qui en possédaient plus d'une.

6ème étape: nous avons soumis les phrases au logiciel.

6.3. Résultats et discussion

Pour des raisons de cohérence, nous avons utilisé ici la même méthode de classification des messages du logiciel que pour le programme LFG (voir le chapitre 4 pour plus d'explications). Celle-ci est adaptée à partir de la méthode présentée par Grosjean (1988). Le logiciel n'ayant pas détecté d'erreurs

incorrectement, notre classification ne contient pas cette catégorie (A.I.b. dans le tableau 4.1). Nous considérons les cas représentés dans le tableau 6-6.

A. ERREUR
- I. DETECTEE
  - 1) mise en garde (MG)
  - 2) suggestion de correction (SC)
- II. NON DETECTEE ('miss')

B. NON-ERREUR
- I. DETECTEE ('false alarm', ou 'bruit')
  - 1) mise en garde (MG)
  - 2) suggestion de correction (SC)

Tableau 6-6 - Classification des erreurs lors de l'évaluation

La distinction entre mise en garde et suggestion de correction est la suivante:

- Nous avons compté une mise en garde chaque fois que le mécanisme d'interaction a été exécuté (voir la section 5.1.2, Architecture du correcteur ARCTA, pour une description de ce mécanisme. Voir aussi 6.1.5, Ordre sujet-verbe dans le discours indirect, pour son application au traitement d'une erreur). Son utilisation indique que l'auteur des automates a jugé qu'il n'était pas possible d'identifier l'erreur avec certitude sans entamer un dialogue avec l'utilisateur. Comme le montreront les résultats de l'évaluation, les mises en garde ne sont apparues que dans des cas où le texte contenait effectivement une erreur.

Nous pouvons donc traiter les mises en garde comme des détections correctes de l'erreur.

- Nous avons compté une suggestion de correction chaque fois qu'un automate a détecté une erreur sans faire appel au mécanisme d'interaction. Ce type d'automates est assez précis pour identifier l'erreur et déterminer la correction à appliquer.

Nous analysons les résultats de deux façons. D'une part, nous ne considérons que les taux de réussite du logiciel pour les phrases soumises à l'évaluation. D'autre part nous essayons de déterminer les raisons des échecs afin d'identifier les règles et les algorithmes défectueux.

6.3.1. Erreurs détectées

Le tableau 6-7 ci-dessous présente les résultats de l'évaluation sur le corpus ARCTA; le tableau 6-8 présente les résultats pour les phrases-test. Les chiffres dans la deuxième colonne représentent le nombre total d'erreurs de chaque catégorie. Les chiffres dans les troisième, quatrième et cinquième colonnes indiquent comment le logiciel a traité ces erreurs. Par exemple, le corpus ARCTA contient 41 erreurs de la catégorie Accord S-V. 10% de ces erreurs ont été détectées avec une mise en garde du logiciel (voir le tableau 6-6, A.I.1), 51% ont été détectées avec une suggestion de correction (A.I.2), et 39% n'ont pas été détectées (A.II). Les chiffres dans les deux dernières colonnes ('Nombre de non-erreurs détectées') représentent le nombre de non-erreurs détectées par les automates pour chaque catégorie (B.I.1 et B.I.2). Dans le corpus ARCTA, par exemple, les automates qui détectent les erreurs de la catégorie Accord S-V ont

identifié 13 erreurs là où il n'y en avait pas et ce parmi les quelques 1400 phrases du corpus.

Catégorie	Nombre d'erreurs	Erreurs détectées A.I		Erreurs non-détectées A.II	Nombre de non-erreurs détectées B.I	
		MG	SC		MG	SC
		[%]	[%]	[%]		
1. Accord S-V	41	10	51	39	-	13
2. Passif	4	-	100	-	-	-
3. Prép. temp.	23	-	87	13	-	-
4. Cont/simple	6	-	100	-	-	-
5. Ordre S-V	5	-	100	-	-	2
6. Prop. Inf.	9	-	78	22	-	-
7. Accord N	13	8	69	23	-	2
8. Compl.temp.	2	100	-	-	-	-
9. Ordre adv-V	35	-	86	14	-	3
Total	138	5 %	74 %	21 %	-	20

MG = mise en garde

SC = suggestion de correction

Tableau 6-7 - Résultats de l'évaluation sur le corpus ARCTA

Catégorie	Nombre d'erreurs	Erreurs détectées A.I		Erreurs non-détectées A.II	Nombre de non-erreurs détectées B.I	
		MG [%]	SC [%]	[%]	MG	SC
1. Accord S-V	10	10	50	40	-	-
2. Passif	10	-	80	20	-	-
3. Prép. temp.	10	-	60	40	-	-
4. Cont/simple	10	-	100	-	-	-
5. Ordre S-V	10	-	100	-	-	-
6. Prop. inf.	10	-	90	10	-	-
7. Accord N	10	10	90	-	-	1
8. Compl.temp.	10	70	-	30	-	-
9. Ordre adv-V	10	-	70	30	-	1
Total	90	10 %	71 %	19 %	-	2

MG = mise en garde

SC = suggestion de correction

Tableau 6-8 - Résultats de l'évaluation sur les phrases-test

Nous pouvons constater que les résultats sont tout à fait satisfaisants pour le corpus ARCTA qui a servi lors du développement des automates (Tableau 6-7), ainsi que sur les phrases-test cachées jusqu'au moment de l'évaluation (Tableau 6-8). Le faible rapport de non-erreurs détectées (sauf peut-être pour l'accord sujet-verbe) nous montre que les automates font très bien leur travail. Lorsqu'ils identifient une erreur, l'utilisateur peut avoir pleine confiance qu'ils ne se trompent pas. Nous remarquons aussi l'homogénéité des taux de détection: chaque catégorie a obtenu un taux de détection de plus de 50%, ce qui indique que les automates sont bien adaptés pour traiter des erreurs syntaxiques et morpho-syntaxiques en général.

Considérons le rapport qui existe entre les suggestions de correction (SC) et les mises en garde (MG) produites lorsque le logiciel identifie une erreur avec succès. Dans les deux séries de phrases, le nombre de suggestions de correction est nettement supérieur au nombre de mises en garde, ce qui montre que les automates permettent de bien cerner les erreurs. Ce n'est qu'à peu près une fois sur dix qu'une mise en garde est produite. La consultation des tableaux des résultats nous permet de plus de constater que les mises en garde apparaissent uniquement pour 3 catégories d'erreurs: Accord S-V, Accord N et Compl. temp.

Pour la catégorie Accord S-V, les mises en garde sont produites lorsque le texte contient la suite suivante:

SN + *and* ou préposition + SN + Verbe (3ème personne du singulier)

Comme dans **My mother and my father is ill.* (Correct: *are*)

Dans le cas de la catégorie Accord N, il s'agit du cas spécial de *all*, comme dans la phrase **You cannot say that **all prisoner** should be treated the same* (correct: *all prisoners*). Enfin, dans le cas de Compl. temp. le logiciel ne génère que des mises en garde, jamais de suggestions de correction, car l'auteur des automates a jugé qu'il n'était pas possible d'identifier l'erreur avec certitude sans entamer un dialogue avec l'utilisateur.

Considérons les taux de détection et le nombre de non-erreurs détectées dans les deux évaluations. Mis à part les non-erreurs détectées dans le corpus ARCTA par les automates de la catégorie Accord S-V (nombre qui n'est pas très important si l'on considère la taille du corpus, 1400 phrases), nous constatons une certaine constance dans les résultats. Ceci montre d'une part que la phase de développement des automates a été bien menée, car il n'existe pas de catégories d'erreurs pour lesquelles la première évaluation a donné des résultats nettement meilleurs que la seconde. D'autre part, nous pouvons en déduire que si les automates ont fonctionné relativement bien pour toutes les catégories d'erreurs, ils devraient alors se conduire de manière acceptable pour d'autres catégories de problèmes syntaxiques et morpho-syntaxiques.

Dans les sections qui suivent, nous examinons les résultats de l'évaluation catégorie par catégorie. Comme indiqué plus haut, le fait que les phrases du corpus ARCTA ont été utilisées lors du développement du logiciel nous empêche d'utiliser les résultats de la première évaluation pour déterminer la qualité des automates. Nous avons toutefois la possibilité d'extraire de ces résultats quelques indications relatives au nombre de situations d'erreurs couvertes par les automates et aux problèmes que ceux-ci rencontreront lorsque des phrases nouvelles leur sont soumises.

a. Accord S-V (catégorie numéro 1)

Bien qu'ayant obtenu le moins bon résultat, cette catégorie a tout de même réalisé un taux de détection de l'ordre de 60% sur chaque ensemble de textes. Les cas où l'erreur n'a pas été détectée représentent tous des situations où le sujet était séparé du verbe soit par une proposition relative, soit par une ou plusieurs virgules. Parmi les 13 non-erreurs détectées dans le corpus ARCTA, nous retrouvons 9 fois la même situation. En effet, le corpus contient 12 traductions du même texte; 9 de ces traductions contiennent l'extrait de phrase suivant, pour lequel une erreur est détectée pour *remind: The pictures of desolation that the fire left behind on the 18 of February remind us....* Nous pouvons en conclure que le logiciel détecte fort bien les erreurs d'accord sujet-verbe, mais que certaines contraintes devraient être introduites dans les automates afin d'éviter ce genre de situation. Sur le corpus de phrases-test, les automates n'ont pas détecté de non-erreurs, ce qui montre que les situations problématiques rencontrées lors de l'évaluation sur le corpus ARCTA ne se retrouvent pas de manière régulière.

b. Passif et Cont/Simple (catégories numéros 2 et 4)

Ces deux catégories traitent de problèmes liés à la construction des temps des verbes. Les bons résultats obtenus lors des deux évaluations (seules 2 erreurs non-détectées parmi 30, aucune non-erreur détectée) confirment la qualité du pré-traitement effectué par le logiciel pour identifier les structures verbales et leur assigner des traits. Ces résultats montrent également les avantages de l'utilisation des automates pour le traitement de textes réels. Nous constatons ainsi que la présence d'adverbes autour du verbe principal et de ses auxiliaires ne présente pas de problèmes de détection.

c. Prép. temp. et Compl. temp. (catégories numéros 3 et 8)

Ces deux catégories traitent de problèmes liés aux aspects temporels des syntagmes prépositionnels construits avec les prépositions *for*, *since* et *during*. Sur le corpus ARCTA, ces catégories ont obtenu de bons résultats: 20 détections sur 23 pour Prép. temp. et 2 sur 2 pour Compl. temp. Par contre, les résultats relativement moyens obtenus par ces catégories sur le corpus des phrases-test illustrent les difficultés liées à la tâche d'interpréter correctement les aspects temporels. Par exemple, les erreurs contenues dans les deux phrases ci-dessous n'ont pas été détectées:

(2) * *We did not see each other **during** a long time.* (Correct: *for)*

(3) * *Settlers **were** coming here for centuries.* (Correct: *have been)*

d. Ordre S-V (catégorie numéro 5)

Cette catégorie a connu un taux de détection parfait sur les deux ensembles de phrases. Comme la catégorie Accord S-V, celle-ci traite d'erreurs relatives au verbe et à son sujet. Ordre S-V a toutefois obtenu de meilleurs résultats, principalement à cause des structures syntaxiques plus simples rencontrées dans le discours indirect. L'exemple ci-dessous illustre bien ce phénomène:

(1) **She does not know what were we talking about.*

(Correct: *She does not know what we were talking about.*)

Il est difficilement envisageable de remplacer le pronom *we* par un sujet contenant une longue proposition relative sans rendre la phrase pratiquement incompréhensible.

e. Prop. Inf. (catégorie numéro 6)

Cette catégorie d'erreurs n'a connu que trois cas de malfonctionnement, deux sur le corpus ARCTA et un sur les phrases-test. Ce résultat est étonnant si l'on considère qu'elle traite de problèmes liés au verbe et à ses compléments et que les automates sont quelque peu limités dès que le domaine de l'erreur devient trop étendu. Nous constatons toutefois que la structure des phrases où se trouve une erreur de cette catégorie reste assez simple et toujours plus ou moins la même. Par exemple:

(4) * *I would like* ***asking*** *you one question.* (correction: *to ask*)
(5) * *It all seemed __ be such a silly mistake.* (correction: insérer *to*)

L'erreur dans la phrase ci-dessous n'a pas été détectée par contre à cause du complément (*Diana*) qui se trouve entre *like* et *stop*:

(6) * *Charles would like Diana __ stop drinking.* (correction: insérer *to*)

f. Accord N (catégorie numéro 7)

Toutes les erreurs relatives à cette catégorie ont été détectées avec succès dans les phrases-test, et 10 erreurs sur 13 ont été détectées dans le corpus ARCTA. Les automates permettent donc de traiter la structure des syntagmes nominaux simples correctement. Les exemples ci-dessous donnent une idée de la forme

générale, relativement simple, des phrases qui contiennent une erreur de cette catégorie:

(7) * *Each **seats** was covered with a white lace cover.* (Correction: *seat*)

(8) * *At school we had to wear these awful white cotton **hat**.*
(Correction: *hats*)

Par contre, une non-erreur a été détectée dans la phrase correcte suivante:

(9) *It is exactly the sort of scheme he will like.*

Le logiciel a indiqué que *scheme* devrait être au pluriel parce qu'il manquait la condition disant que *sort of* ou *type of* sont suivis du singulier.

g. Ordre adv-V (catégorie numéro 9)

Les résultats de cette catégorie sont relativement bons (86% sur le corpus ARCTA et 70% sur les phrases-test), surtout si l'on considère la variété et le nombre important d'erreurs contenues dans ces phrases (45 au total). Comme les catégories Cont/simple et Passif, celle-ci traite de problèmes locaux aux verbes. Alors que pour les deux premières catégories, les automates doivent éviter les adverbes, ici ils doivent déterminer si leur position est incorrecte. Les résultats confirment ainsi que les automates permettent de traiter de manière satisfaisante toutes les formes verbales.

6.3.2. Problèmes de détection

Pour cette analyse des résultats, nous ne considérons que les cas où le logiciel n'a pas fonctionné. Ils comprennent les 46 erreurs non détectées dans les deux évaluations ainsi que les 22 non-erreurs. Quatre raisons principales semblent expliquer le non-fonctionnement du logiciel:

a. La structure syntaxique de la phrase était trop compliquée: 24 cas.

Les automates étant élaborés pour identifier des suites de mots et de syntagmes, il n'est pas surprenant que leur utilisation ne permette pas de traiter de phrases trop complexes. Les exemples ci-dessous illustrent quelques situations difficiles rencontrées dans le corpus ARCTA et dans les phrases-test:

(10) * *What we hear through the music, what we see through the painting, **transmit** what we live and what we can not explain with words.* (correction: *transmits*)

(11) * *The suits that we bought from one of your competitive firm last year were not sold very well last winter.* (Une erreur d'accord sujet-verbe a été faussement indiquée pour *were*)

(12) * *Time, money and effort **was** needed.* (correction: *were*)

Note: nous avons conservé ici l'état original des phrases afin d'illustrer les nombreuses erreurs qu'elles contiennent.

b. Les automates étaient incomplets: 27 cas.

Ces cas sont présents dans la plupart des catégories d'erreurs. Ils illustrent les difficultés qui existent à traiter de textes réels et mettent en avant quelques problèmes liés aux automates. Ceux-ci doivent exécuter une analyse syntaxique locale et en même temps appliquer des conditions relatives aux erreurs à détecter. Ainsi, pas moins de 39 automates sont utilisés pour détecter les erreurs de la catégorie Accord S-V. Ces deux facteurs rendent leur développement parfois difficile et des oublis peuvent se produire.

Considérons quelques exemples de phrases pour lesquelles la structure syntaxique locale n'a pas été analysée correctement, ce qui a empêché la détection de l'erreur:

(13) * ... *his attitude and especially his expression **was** far more...*
(correction: *were*)

(14) * *I was **spoke** to in a language I could not understand.*
(correction: *spoken*)

Dans la phrase (13), l'automate de détection de l'erreur d'accord sujet-verbe n'a pas tenu compte du fait que l'adverbe *especially* pouvait se trouver entre les deux syntagmes nominaux liés par *and*. Dans la phrase (14), le passif est construit avec le prétérit au lieu du participe (*spoken*) alors que l'automate, faisant partie de la catégorie Passif, ne recherche que la forme de base. Cet automate aurait donc détecté l'erreur dans * *I was **speak** to in a language I could not understand* (correction: *spoken*).

Dans l'exemple suivant les automates ont détecté une erreur d'accord pour *remind*:

(15) * *The pictures of desolation that the fire left behind it on 18th February remind us of...*

Si *on 18th February* avait été reconnu comme syntagme prépositionel, l'automate de détection n'aurait pas identifié *18th February* comme le sujet de *remind* et n'aurait donc pas passé.

Les 27 cas de cette section se distinguent de ceux pour lesquels la structure syntaxique était trop compliquée par le fait qu'il s'agit ici de problèmes locaux, donc mieux contrôlables. Pour chacun de ces cas il est possible de modifier un automate, d'en ajouter un ou d'ajuster quelques conditions pour que la qualité de la détection soit améliorée.

c. Des informations contextuelles auraient été nécessaires: 12 cas.

Cette catégorie regroupe tous les cas de malfonctionnement pour lesquels des informations sémantiques, pragmatiques ou stylistiques étaient requises pour déterminer si des phrases ou bouts de phrase étaient corrects ou s'ils contenaient une erreur. Nous retrouvons fréquemment cette situation dans la catégorie d'erreurs Ordre adv-V. Par exemple:

(16) * *In a modern office, the secretary can choose a definite work, she can **only** work with word-processor operator or she can do all the correspondence with the foreign firms.*
(Correction: *... can work only...*)

Ici l'adverbe *only* devrait se trouver après le verbe *work* et non pas avant. Les données syntaxiques et morpho-syntaxiques auxquelles les automates ont accès ne suffisent pas à identifier cette erreur.

d. Les informations morphologiques ou syntaxiques attachées à un mot étaient incorrectes au départ: 5 cas.

Comme mentionné au début de ce chapitre, chaque phrase du corpus d'évaluation est passée par une étape de pré-traitement dans laquelle le module d'étiquetage a assigné à chaque mot une catégorie syntaxique et des traits morphologiques. Dans les 5 cas regroupés ici, le dictionnaire d'analyse contenait des données insuffisantes, ce qui a empêché certaines détections et a causé quelques fausses détections. Dans l'exemple (17),

(17) * *The child's clothing* ***look*** *new*. (Correction: *looks*)

le dictionnaire d'analyse a indiqué que *clothing* était un verbe. L'automate qui détecte les problèmes d'accord sujet-verbe n'a de ce fait pas pu passer. Dans (18),

(18) *Aids is going this way...*

Aids a été noté comme le pluriel de *aid*, ce qui a permis au même automate de passer.

Parmi l'ensemble des cas de non-fonctionnement du logiciel, nous pouvons distinguer entre ceux qui pourraient être résolus moyennant un travail

supplémentaire, et ceux pour lesquels le formalisme actuel est insuffisant. Ainsi, lorsque la structure syntaxique est trop complexe, la seule possibilité d'identifier les erreurs qui s'y trouvent consiste à entreprendre une analyse complète de la phrase, avec les conséquences que l'on connaît (voir chapitre 4). Les cas où des données sémantiques ou pragmatiques sont nécessaires ne peuvent pas non plus être traités de manière satisfaisante avec les automates uniquement. Par contre, on peut envisager d'améliorer le dictionnaire d'analyse et l'étape de préparation à l'exécution des automates, mais cela n'aura qu'un effet relativement peu important (5 malfonctionnements du logiciel uniquement sont dus à ces causes). Les problèmes liés au fait que certains automates sont incomplets requièrent plus d'attention; on pourrait par exemple étudier un corpus d'erreurs plus important ou établir une grammaire locale complète pour les domaines concernés.

Conclusion

L'évaluation a mis en évidence quelques problèmes liés aux automates. Nous avons vu que la combinaison, à l'intérieur d'un même automate, des conditions sur la structure locale et des conditions de détection pouvait entraîner certaines confusions et rendait les automates peu clairs. Nous avons également constaté que la détection d'erreurs sans analyse syntaxique complète de la phrase avait ses limites. Cette situation se retrouvera certainement pour d'autres catégories d'erreurs non traitées ici.

Malgré ces problèmes, les résultats que nous avons obtenus lors de l'évaluation sont bons. Un taux de détection de 80% sur 248 erreurs indique que le formalisme des automates est particulièrement adapté à la tâche de détection et de correction d'erreurs. Le fait que le logiciel n'ait produit que 22 fausses détections

sur plus de 1500 phrases montre que le formalisme permet d'inclure dans les automates des conditions qui éliminent pratiquement tous les cas sauf ceux où une erreur reconnue se trouve dans le texte.

L'utilisation de phrases réelles pour l'évaluation donne une vraie valeur à ces résultats. En effet, l'expérience montre la difficulté qui existe à rendre un logiciel capable de traiter des textes ordinaires alors qu'il était conçu pour de simples phrases à la structure et au contenu lexical limités. Il est intéressant de noter que le processus de détection n'a pas été affecté par des phrases réelles qui contenaient souvent plus d'une erreur, certaines de celles-ci n'étant même pas traitées par les automates. L'étape d'utiliser des textes ordinaires a donc été franchie ici en partie, même si nous avons préparé le corpus manuellement en désambiguïsant les catégories syntaxiques de chaque mot et en regroupant les syntagmes nominaux simples. Etant donné que nous ne pourrons pas éviter quelques problèmes lorsque ces algorithmes seront exécutés automatiquement, un certain nombre d'erreurs ne seront pas détectées. Mais c'est justement la manière d'opérer des automates (à savoir rechercher uniquement des suites de mots spécifiques dans les phrases, sans avoir à effectuer d'analyse complète) qui va empêcher bon nombre de fausses détections dues à des données incorrectes. Un autre avantage de cette approche réside dans le fait qu'il est possible d'ajouter des automates sans entraver le fonctionnement et la capacité de détection des automates existants. Nous pouvons ainsi envisager sans trop de problème de compléter les automates existants pour éliminer une partie des malfonctionnements identifiés lors de cette évaluation, et de développer d'autres automates pour compléter le logiciel.

Une solution aux problèmes qui demeurent consisterait à affiner les étapes qui mènent à la mise en oeuvre des automates de détection. Il serait par exemple possible d'effectuer une analyse syntaxique simple de la phrase. Des automates de détection supplémentaires pourraient ensuite identifier des structures spécifiques dans les données produites par cette analyse. Ceci permettrait de soulager les automates de détection de l'analyse locale et de destiner les automates au traitement d'erreurs plus complexes. En effet, l'efficacité des automates tient en partie à leur simplicité: nous n'effectuons pas d'analyse complète du texte et nous les simplifions délibérément par rapport aux ATN, en évitant la récursivité et le backtracking (c'est-à-dire que les automates ne reviennent jamais en arrière lorsqu'ils rencontrent un mot ou un syntagme qui ne correspond pas au prochain arc). Nous devrions avant tout prendre garde à ne pas enfreindre ces principes de base si nous voulons étendre ce formalisme dans le but de traiter des erreurs plus complexes. Si ces restrictions dans le fonctionnement des automates étaient abandonnées, nous rencontrerions probablement les mêmes problèmes qu'avec l'approche basée sur la LFG.

Conclusion générale

Dans ce travail, nous avons d'abord décrit la nature des erreurs faites en langue seconde et nous avons fait le point sur les recherches en linguistique appliquée et plus particulièrement en analyse des erreurs. Ces travaux nous ont fourni des renseignements quant à la nature des erreurs, leur origine et la manière de les détecter et corriger. Nous avons ensuite décrit les caractéristiques et le fonctionnement de plusieurs logiciels récents spécialisés dans la correction automatique. Nous avons pu mettre en évidence deux faits: d'une part, les algorithmes utilisés par les logiciels de laboratoire s'adaptent difficilement aux contraintes imposées par les ordinateurs personnels d'aujourd'hui; d'autre part, les logiciels

commerciaux actuels ne traitent pas de manière satisfaisante les erreurs en langue seconde; leur utilité est donc limitée.

Nous avons alors présenté deux approches de la correction automatique des erreurs en langue seconde. La première repose sur le formalisme de la grammaire lexicale et fonctionnelle (LFG) et la seconde sur l'approche par automates. Les résultats obtenus lors de l'évaluation du correcteur LFG nous ont indiqué que cette première approche permettait la détection d'erreurs complexes et multiples. Nous avons toutefois aussi constaté que les algorithmes élaborés étaient difficilement utilisables sur le plan commercial. Afin de rendre compte des structures de phrases inattendues que l'on rencontre dans des textes en langue seconde, ils doivent faire appel à un lexique et à un ensemble de règles syntagmatiques à la fois complexe et complet, ce qui entraîne une multiplication des solutions à conserver en mémoire et une lenteur excessive du système.

Nous avons ensuite présenté une méthode de détection d'erreurs basée sur l'approche par automates. La qualité des résultats obtenus lors de l'évaluation ainsi que la nature des algorithmes utilisés nous font penser que cette manière de faire pourrait s'appliquer avec succès au traitement d'une gamme très étendue d'erreurs en langue seconde, ceci en plus des erreurs syntaxiques et morphologiques qui ont fait l'objet de notre travail. En plus de son adéquation aux moyens informatiques à disposition aujourd'hui, l'approche par automates se caractérise par une bonne fiabilité, ce qui est démontré par le nombre relativement réduit de fausses détections que nous avons relevées lors de l'évaluation. L'approche par automates n'est toutefois pas dénuée de problèmes, spécialement dans les cas de phrases à structure complexe. Il faudrait donc apporter certaines

améliorations aux algorithmes de détection et de correction pour augmenter les performances du logiciel.

En nous basant sur l'ensemble des caractéristiques des deux approches, nous pensons qu'il serait possible d'obtenir de meilleurs résultats encore en les combinant, et ce afin de tirer profit des avantages de chacune d'elles. Nous pourrions envisager la procédure suivante:

1. Une phase de pré-traitement effectuée par des automates, comme décrit au chapitre 5. Il s'agit ici d'identifier les traits et les bornes des syntagmes nominaux et des formes verbales.

2. Une première phase de détection, également effectuée par des automates, afin de traiter tous les problèmes locaux, comme l'orthographe et l'ordre des mots. Ceci serait une phase de nettoyage en quelque sorte.

3. Une deuxième phase de détection, effectuée par un analyseur LFG tel que celui que nous avons décrit au chapitre 3, pour traiter l'ensemble des erreurs morphologiques. L'analyseur LFG ne devrait pas rencontrer les mêmes problèmes que ceux mis en évidence au chapitre 4, étant donné que les erreurs relatives à l'ordre des mots et aux structures incorrectes auront déjà été détectées et corrigées dans la première phase de détection.

4. Une troisième phase de détection, effectuée par des automates. Ces derniers utiliseraient les structures syntaxiques et fonctionnelles créées par l'analyseur LFG pour détecter et corriger les erreurs pour lesquelles le formalisme LFG n'est pas particulièrement bien adapté. Nous pensons ici à tous les problèmes de sous-catégorisation (voir Kübler, 1995).

Considérons, par exemple, la phrase suivante:

* *The students who obtained the best marks* ***was accuse to be*** *cheaters.*

Correct: *The students who obtained the best marks were accused of being cheaters.*

La phrase incorrecte contient trois erreurs. Premièrement, l'accord du nombre n'est pas respecté entre *the students* et *was*. Deuxièmement, le passif de *to accuse* n'est pas construit correctement. Enfin, le complément de *accuse* ne devrait pas prendre la forme *to* + infinitif, mais la forme *of* + -ing.

Après la phase de pré-traitement effectuée par les automates, les trois phases de détection entrent en jeu séquentiellement de la manière suivante:

1. L'automate spécifiquement destiné à identifier les erreurs de construction du passif (voir section 6.1.2.) détecte l'erreur d'utilisation de *accuse* sans être affecté par les autres erreurs contenues dans le texte. Cette erreur est corrigée par l'utilisateur et *accuse* est remplacé par *accused*. Aucun autre automate ne détecte une erreur dans cette phrase, car les automates de la section 6.1.1. qui se chargent des erreurs d'accord sujet-verbe n'entrent plus en jeu à ce stade.

2. Le logiciel LFG, dans lequel nous pouvons remplacer la grammaire et le lexique du français par leur équivalent anglais (voir Bresnan, 1982),

détecte avec succès l'erreur d'accord de *was*, grâce à l'analyse complète de la phrase qu'il effectue et ce que nous avons su tirer de l'évaluation de ce type d'erreurs. Une fois l'erreur corrigée, le logiciel n'en détecte pas d'autres car nous avons supprimé les règles de détection relatives à la sous-catégorisation des verbes.

3. Des automates opérant sur le contenu des structures fonctionnelles construites par le logiciel LFG identifient la présence du verbe *accuse* et du complément construit avec *to* + infinitif; l'erreur est donc détectée. Il est important de noter que les automates, à ce stade, ne recherchent plus des suites de mots ou de syntagmes; ils agissent ici indépendemment de la structure de surface de la phrase. Contrairement à ce qui se faisait jusqu'ici, il n'est donc plus nécessaire de créer des automates pour chaque structure différente. Dans notre exemple, le fait que le complément d'objet direct de *accuse* (*the students*) se trouve au début de la phrase plutôt que juste après le verbe n'affecte donc ni le développement des automates par le linguiste, ni leur exécution.

Cette procédure n'est bien sûr pas la seule possible. Nous estimons toutefois qu'elle pourrait être mise en oeuvre assez facilement, ce qui pourrait conduire à l'élaboration d'un correcteur grammatical en langue seconde de bon niveau.

Dans ce travail, nous nous sommes penchés uniquement sur les aspects mécaniques de la détection des erreurs. Ainsi, nous n'avons pas étudié les problèmes spécifiques liés à l'intégration de notre logiciel dans des systèmes de plus haut niveau et destinés à des applications bien précises. Des recherches supplémentaires seraient nécessaires pour identifier les

besoins des différents groupes d'utilisateurs, par exemple. Nous pourrions certes sélectionner des règles de détection pour différents registres (commercial, scientifique, formel, etc.), mais nous ne possédons pas encore de données relatives à la fréquence et aux types d'erreurs commises dans chacun de ces cas particuliers. A défaut de pouvoir cibler un utilisateur précis, nous pouvons envisager la présence d'un test qui permettrait au logiciel de déterminer le niveau de compétence de l'utilisateur. Une telle procédure aiderait ce dernier à déterminer si le logiciel répond vraiment à ses besoins. Suivant le cadre dans lequel le logiciel est intégré, il serait de plus possible de conserver un historique des erreurs détectées, afin d'évaluer le progrès de l'utilisateur en fonction des erreurs commises. Enfin, la manière de présenter les erreurs à l'utilisateur en fonction de ses besoins et de son niveau linguistique nécessiteraient également une recherche.

Ces différents aspects appliqués devraient être abordés dans une phase de commercialisation d'un logiciel de correction en langue seconde mais dépassent les objectifs que nous nous sommes fixés en début de thèse. En effet, nous nous sommes efforcé dans ce travail de développer les algorithmes de détection et de correction de deux approches différentes et d'en montrer les avantages et les inconvénients. Nous en avons conclu qu'une combinaison judicieuse de ces deux voies serait la meilleure solution.

Bibliographie

Ouvrages cités

Barchan J., B. Woodmansee et M. Yazdani (1986). A Prolog-based tool for French grammar analysis. *Instructional Science* 15, 21-48.

Besse, H. et R. Porquier (1984). *Grammaires et didactique des langues*. Paris: Hatier.

Block, H.-U. et R. Hunze (1986). Incremental construction of C- and F-structure in a LFG-parser. *Proceedings of COLING '86*, 490-493.

Block, H.-U. et H. Haugeneder (1986). The treatment of movement-rules in a LFG parser. *Proceedings of COLING '86*, 482-486.

Blum-Kulka, S. et E. Leventson (1983). Universals of lexical simplification. Dans Faerch, C. et G. Kasper (eds.). *Strategies in IL communication* (pp. 119-139). New York: Longman.

Bodmer, F. (1994). DELENE: un désambiguïsateur lexical neuronal pour textes en langue seconde. *TRANEL (Travaux neuchâtelois de linguistique)* 21, 247-263.

Brehony T. et K. Ryan (1994). Francophone stylistic grammar checking (FSGC) using link grammars. *Computer Assisted Language Learning* 7(3), 257-269.

Bresnan, J. (ed.) (1982). *The Mental Representation of Grammatical Relations*. Cambridge, MA: The MIT Press.

Catt, M. (1988). *Intelligent Diagnosis of Ungrammaticality in Computer-Assisted Language Instruction*. University of Toronto, Technical Report CSRI-218.

Chomsky, N. (1965). *Aspects of the Theory of Syntax*. Cambridge, MA: The MIT Press.

Chomsky, N. (1981). *Lectures on Government and Binding*. Dordrecbt: Foris.

Corder, S. P. (1967). The significance of learner's errors. *IRAL* V(2-3), 161-169.

Corder, S. P. (1973). *Introducing Applied Linguistics*. New York: Penguin.

Cornu, E. (1991a). *Le programme ETI*. Rapport interne du laboratoire de traitement du langage et de la parole. Université de Neuchâtel.

Cornu, E. (1991b). *Le logiciel ARCTA*. Rapport interne du laboratoire de traitement du langage et de la parole. Université de Neuchâtel

Dinnematin, S. et D. Sanz (1990). Sept correcteurs pour l'orthographe et la grammaire. *SVM*, Décembre, 118-130.

Dobrin, D. (1990). A new grammar checker. *Computer and the Humanities* 24, 67-80.

Dubois, J. (1967-1969). Grammaire structurale du français. Tome I Le verbe; Tome II La phrase et ses transformations. Paris: Larousse.

Dulay, H. C. et M. K. Burt (1972). Errors and strategies in child second language acquisition. *TESOL Quarterly* 8(2), 129-136.

Eglowstein H. (1991). Can a grammar checker improve your writing? *Byte*, Août, 238-242.

Eisele, K. et J. Dorre (1986). A lexical functional grammar system in Prolog. *Proceedings of COLING '86*, 551-553.

Emirkanian, L. et L.H. Bouchard (1989). La correction des erreurs d'orthographe d'usage dans un analyseur morpho-syntaxique du français. *Langue Française* (83), 106-122.

Faerch, C. et G. Kasper (eds.) (1983). *Strategies in Interlanguage Communications*. New York: Longman.

Flynn, S. (1988). Contrast and construction in a parameter-setting model of L2 acquisition. *Language Learning* 37(1), 19-62.

Flynn, S. et W. O'Neil (eds.) (1988). *Linguistic Theory in Second Language Acquisition*. Dordrecht/Boston/London: Kluwer Academic Publishers.

Frey, W. et U. Reyle (1983). A Prolog implementation of lexical functional grammar as a base for a natural language processing system. *Proceedings of the European ACL 1983*, 52-57.

Fromkin, V. A. (1988). Grammatical aspects of speech errors. Dans Newmeyer, F. (ed.). *Linguistics: The Cambridge Survey. II. Linguistic Theory: Extensions and Implications* (pp. 117-138). Cambridge: Cambridge University Press.

Gazdar, G., E. Klein, G.K. Pullum et I. Sag (1985). *Generalized Phrase Structure Grammar*. Cambridge, MA: Harvard University Press.

Grevisse, M. (1988). *Le bon usage*. Paris: Editions Duculot.

Grosjean, F. (1982). *Life with two Languages: An Introduction to Bilingualism*. Cambridge, MA/London: Harvard University Press.

Grosjean, F. (1988). *The Evaluation of Writing Tools.* Manuscrit, Laboratoire de traitement du langage et de la parole. Université de Neuchâtel.

Imlah, W. G. et J. B. H. du Boulay (1985). Robust natural language parsing in computer-assisted language instruction. *System* 13(2), 137-147.

Jensen K., G. E. Heidorn, L. A. Miller et Y. Ravin (1983). Parse fitting and prose fixing: getting a hold on ill-formedness. *American Journal of Computational Linguistics* 9(3-4), 147-160.

Kaplan R., et J. Bresnan. (1982). Lexical-functional grammar: a formal system for grammatical representation. Dans Bresnan (ed.). *The Mental Representation of Grammatical Relations* (pp. 173-281). Cambridge, MA: The MIT Press.

Karttunen, L. (1984). Features and values. *Proceedings of the International Conference on Computational Linguistics 1984*, 28-33.

Kay, M. (1985). Parsing in functional unification grammar. Dans Dowty, D., L. Karttunen et A. M. Zwicky (eds.). *Natural Language Parsing* (pp. 251-278). Cambridge: Cambridge University Press.

Klein, W. (1986). *Second language acquisition.* Cambridge: Cambridge University Press.

Kübler, N. (1995). *L'automatisation de la correction d'erreurs syntaxiques*. Thèse de doctorat. Université de Marne-la-Vallée.

Lado, R. (1957). *Linguistics across Cultures.* Ann Arbor: University of Michigan Press.

Laenzlinger, C. (1992). *Les correcteurs grammaticaux: linguistiquement satisfaisants?* Manuscrit. Université de Genève.

Liechti, C. (1991). *L'aide à la rédaction en langue seconde: évaluation d'un premier logiciel (Bilingual PC Proof).* Mémoire de licence. Université de Neuchâtel.

Mandeville, B. (1992). L'anglais assisté par ordinateur. *Info'PC.* Mars, 139-148.

Nemser, W. (1971). Approximative systems of foreign language learners. *IRAL* 9, 115-123.

Palmer, R. et T. Finin (1990). Workshop on the evaluation of natural language processing systems. *Computational Linguistics* 16(3), 175-181.

Porquier, R. (1977). L'analyse des erreurs, problèmes et perspectives. *Etudes de linguistique appliquée* 25, 23-43.

Py, B. (1975). A propos de quelques publications récentes sur l'analyse des erreurs. *Bulletin CILA* 22, 45-55.

Py, B. (1984). L'analyse contrastive: histoire et situation actuelle. *Le français dans le monde* 185, 32-37.

Py, B. (1993). L'apprenant et son territoire: système, norme et tâche. *Aile* 2, 9-24.

Rabinowitz, R. (1991). 15 writer's tools. *Byte*, September, 321-369.

Richards, J. C. (1971). A non-contrastive approach to error analysis. *English Language Teaching* 25, 206-219.

Robert, P. (1989). *Le petit Robert 1*. Paris: Dictionnaires Le Robert.

Sabah, G. (1988). *L'intelligence artificielle et le langage. I: représentation des connaissances*. Paris: Hermès.

Sanz, D. (1992). Grammaire: Quatre ténors à l'épreuve. *SVM*, Janvier, 100-108.

Scarcella, R. et C. Lee (1989). Different paths to writing proficiency in a second language? A preliminary investigation of ESL writers of short-term and long-term residence in the United States. Dans Eisenstein, M. (ed.). *The Dynamic Interlanguage. Empirical Studies in Second Language Variation* (pp. 137-153). New York/London: Plenum Press.

Schwind, C. (1988). Sensitive parsing: error analysis and explanation in an intelligent language tutoring system. *Proceedings of COLING '88*, 608-613.

Schwind, C. (1995). Error analysis and explanation in knowledge based language tutoring. *Computer Assisted Language Learning* 8(4), 295-324.

Selinker, L. (1972). Interlanguage. *IRAL* X(3), 209-231.

Sells, P. (1987). *Lectures on Contemporary Syntactic Theories*. Menlo Park, CA: CSLI.

Sheen, R. (1980). The importance of negative transfer in the speech of near-bilinguals. *IRAL* XVIII(2), 105-119.

Sinclair, J. (1990). *Collins Cobuild. English Grammar.* London and Glasgow: Collins.

Sleator, A. et B. Temperley (1991). *Parsing English with a Link Grammar*. Carnegie Mellon University Report CMU-CS-91-196.

Tarone E., A. Cohen et G. Dumas (1983). A closer look at some interlanguage terminology: a framework for communication strategies. Dans Faerch, C. et G. Kasper. (eds.), *Strategies in IL communication* (pp. 4-15). New York: Longman.

Tomita, M. (1987). An efficient augmented-context-free parsing Algorithm. *Computational Linguistics* 13 (1-2), 31-46.

Tschichold, C. (1991). *The evaluation of computer-assisted writing tools for non-native speakers of English*. Lizentiatsarbeit. Universität Basel.

Tschumi, C., F. Bodmer, E. Cornu, F. Grosjean, L. Grosjean, N. Kübler et C. Tschichold (1994). The ARCTA prototype: An English writing tool and grammar checker for French-speakers. *TRANEL (Travaux neuchâtelois de linguistique)* 21, 223-228.

Tschumi, C. (1996). *Evaluation de correcteurs en langue seconde*. Communication personnelle.

Tschumi, C., F. Bodmer, E. Cornu, F. Grosjean, L. Grosjean, N. Kübler, N. Lèwy et C. Tschichold (1996). *Un logiciel prototype pour aider les francophones à rédiger en anglais*. Manuscrit. Université de Neuchâtel.

Vosse, T. (1992). Detecting and correcting morpho-syntactic errors in real texts. *Proceedings of the Third Conference on Applied Natural Language Processing*, 111-118.

Weischedel, R., W. Voge et M. James (1978). An Artificial intelligence approach to language instruction. *Artificial Intelligence* 10, 225-240.

White, L. (1992). Universal grammar: is it just a new name for old problems? Dans Gass, S. et L. Selinker (eds.). *Language Transfer in Language Learning* (pp. 217-232). Amsterdam/Philadelphia: John Benjamins Publishing Company.

Winograd, T. (1983). *Language as a Cognitive Process. Volume 1: Syntax*. Reading, MA: Addison-Wesley.

Woods, W. A. (1970). Transition network grammars for natural language analysis. *CACM* 13(10), 591-606.

Yannakoudakis, E.J. et D. Fawthrop (1982). The Rules of spelling errors. *Information Processing and Management* 19(2), 87-99.

Autres ouvrages

Aho, A., R. Sethi et J. Ullman (1986). *Compilers, Principles, Techniques and Tools*. Reading, MA: Addison Wesley Publishing Company.

Allen, J. (1987). *Natural Language Understanding*. Menlo Park, CA: Benjamin Cummings.

Atwell, E. et S. Elliott (1987). Dealing with ill-formed English text. Dans Garside, R., G. Leech et G. Sampson (eds.). *The Computational Analysis of English* (pp. 120-138). London: Longman.

Berwick, R.C. et A.S. Weinberg. (1986). *The Grammatical Basis of Linguistic Performance*. Cambridge, MA: MIT Press.

Clément, D. (1988). Le lexique en vedette : son rôle dans le modèle de Bresnan "Lexical Functional Grammar". *DRLAV* 38, 93-118.

Cornu, E. (1992). Automatic correction of prose written in a second language. *Proceedings of SGAICO '92*, 1-7.

Davies, A., C. Criper et A.P.R. Howatt (eds.) (1984). *Interlanguage.* Edinburgh: Edinburgh University Press.

Dowty, D., L. Karttunen et A. M. Zwicky (eds.) (1985). *Natural Language Parsing.* Cambridge: Cambridge University Press.

Eisenstein, M. (ed.) (1989). *The Dynamic Interlanguage. Empirical Studies in Second Language Variation.* New York/London: Plenum Press.

Frith, M. (1978). Interlanguage theory: implications for the classroom. *McGill Journal of Education* 13, 155-165.

Gardent, C., G. Bès, P.-F. Jurie et K. Baschung (1989). Efficient parsing for French. *Proceedings of the 27th Annual Meeting of the ACL*, 280-287.

Granger, S. et F. Meunier (1994). Towards a grammar checker for learners of English. Dans Fries, V. et G. Tootie (eds.). *Proceedings of the 14th ICAME Conference on English Language Research on Computerized Corpora* (pp. 1-15). Amsterdam/Atlanta: Rodopi.

Grimshaw, J. (1982). On the lexical representation of romance reflexive clitics. Dans Bresnan, J. (ed.). *The Mental Representation of Grammatical Relations* (pp. 87-148). Cambridge, MA: The MIT Press.

Gross, M. (1968). *Grammaire transformationnelle du francais: le nom*. Paris: Larousse.

Gross, M. (1977). *Grammaire transformationnelle du francais: syntaxe du verbe*. Paris: Larousse.

Guillemin-Flescher, J. (1981). *Syntaxe comparée du français et de l'anglais*. Paris: Editions Ophrys.

Halvorsen, P.K. (1983). Semantics for lexical functional grammar. *Linguistic Inquiry* 14(4), 567-615.

Heidorn, G. E., K. Jensen, L.A. Miller, R.J. Bird et M.S. Chodorow (1982). The EPISTLE text-critiquing system. *IBM Systems Journal* 21(3), 305-326.

Jeker, F. (1992). *An error-detection system for grammar checkers*. Manuscrit. Université de Neuchâtel.

Kudo, I., et H. Nomura (1986). Lexical-functional transfer: a tranfer framework in a machine translation system based on LFG. *Proceedings of COLING '86*, 112-114.

Kübler, N. et E. Cornu (1994). Using automata to detect and correct errors in the written English of French-speakers. *TRANEL (Travaux neuchâtelois de linguistique)* 21, 235-245.

Le Compagnon, B. (1984). Interference and overgeneralization in second language learning: the acquisition of English dative verbs by native speakers of French. *Language Learning* 34(3), 39-67.

Levin, L., M. Rappaport et A. Zaenen (eds.) (1983). *Papers in Lexical Functional Grammar*. Bloomington IN: Indiana University Linguistics Club.

Mellish, C. (1989). Some chart-based techniques for parsing ill-formed input. *Proceedings of the 27th Annual Meeting of the ACL*, 102-109.

Netter, K. (1986). Getting things out of order, an LFG proposal for the treatment of German word order. *Proceedings of COLING '86*, 494-496.

Pusak, J. P. (1983). Answer-processing and error correction in foreign language CAI. *System* 11(1), 53-64.

Py, B. (1978). Analyse contrastive et analyse des erreurs. Dans Nickel, G. (ed.). *Error Analysis* (pp. 91-98). Stuttgart: Hochschulverlag.

Richard, D. et G. Lapalme (1986). Un système de correction automatique des accords du participe passé. *Technique et science informatiques* (4), 307-319.

Richards, J. C. (1974). Social factors, interlanguage, and language learning. *Language Learning* 22(2), 159-180.

Sager, N. (1981). *Natural Language Information Processing, A Computer Grammar of English and its Applications*. Reading, MA: Addison Wesley.

Slack, J. M. (1986). Distributed memory: a basis for chart parsing. *Proceeding of COLING '86*, 476-481.

Tremblay, J.-P. (1972). *Grammaire comparative du français et de l'anglais*. Québec: Les presses de l'université Laval.

Tschumi, C. et C. Tschichold (1994). Selecting English errors made by French-speakers for automatic correction. *TRANEL (Travaux neuchâtelois de linguistique)* 21, pp. 229-234.

Valceschini, N. (1995). Les limites de la détection grammaticale. *Micro Gazette* 9(2), 22-24.

Vinay, J.P. et J. Darbelnet (1957). *Stylistique comparée du francais et de l'anglais*. Saint-Laurent, Québec: Editions Beauchemin.

Weischedel, R.M. et L.A. Ramshaw (1987). Reflections on the knowledge needed to process ill-formed sentences. Dans S. Nirenburg (ed.). *Machine Translation* (pp. 155-167). Cambridge: Cambridge University Press.

Annexe 1. Logiciel LFG: extraits de données linguistiques

Cette section contient des extraits des données utilisées par le logiciel LFG:

a. Dictionnaire

b. Règles morphologiques

c. Règles de réécriture

d. Règles de correction

e. Règles lexicales

a. Dictionnaire

```
!
! ------------------------------------------------------------
!       déterminants
! ------------------------------------------------------------

%défaut classe=article, personne=troisième, art-type=défini .
le          morph=(art_def,le,la,les) .
l           nombre=singulier, prép-nombre=singulier .

%défaut classe=article, art-type=partitif .
de          .
du          .

%défaut classe=article, art-type=indéfini, personne=troisième
.
un          morph=(art_ind,un,des) .
! ------------------------------------------------------------

%défaut classe=adjectif, possessif=oui, personne=troisième .
notre     dst-personne=première, dst-nombre=pluriel,
              morph=(adjpob,notre,nos) .
votre     dst-personne=deuxième, dst-nombre=pluriel,
              morph=(adjpob,votre,vos) .
leur      dst-personne=troisième, dst-nombre=pluriel,
              morph=(adjpob,leur,leurs) .

! ------------------------------------------------------------
!       adjectifs
```

```
! ----------------------------------------------------------

%défaut classe=adjectif, qualificatif=oui, personne=troisième
.
agrégé     adj-position=après-nom, morph=(adj_e,agr,g,) .
amical     adj-position=après-nom, morph=(adj_alx,amic) .
bas        morph=(adj_ss,bas) .
bienvenu   adj-position=après-nom, morph=(adj_e,bienvenu) .

! ----------------------------------------------------------
!       noms
! ----------------------------------------------------------

%défaut classe=nom-propre, (^.personne) =r troisième  .
Durand     genre=r masculin, nombre=r singulier .
France     genre=r féminin, nombre=r singulier .
%défaut classe=nom, (^.personne) =r troisième  .
accent     morph=(mas_rég,accent) .
air        morph=(mas_rég,air) .
ami        morph=(mf_e,ami) .
année      morph=(fem_rég,année) .
arbre      morph=(mas_rég,arbre) .

! ----------------------------------------------------------
!       3- verbes
! ----------------------------------------------------------

%défaut classe=verbe, auxiliaire=avoir .
casser     [ (^.pred) = 'casser<(^.SUBJ)(^.OBJ)>' ] ,
           [ (^.pred) = 'casser<(^.SUBJ)(^.OBJ)>' ,
             (^.réfléchi) =r oui ] ,
           morph=(aimer,cass) .

lancer     morph=(placer,lan) ,
           [ (^.PRED) = 'lancer<(^.SUBJ)(^.OBJ)>' ] ,
           [ (^.PRED) = 'lancer<(^.SUBJ)(^.OBJ)(^.OBJ2)>' ] .

oublier    [ (^.pred) = 'oublier<(^.SUBJ)(^.OBJ)>' ] ,
           [ (^.pred) = 'oublier<(^.SUBJ)(^.XCOMP)>' ,
             (^.XCOMP.COMP-PREP) =r PREP-DE ,
             (^.XCOMP.SUBJ)=(^.SUBJ) ] ,
           morph=(aimer,oubli) .

sauver     [ (^.pred) = 'sauver<(^.SUBJ)>' ,
             (^.réfléchi) =r oui ] ,
           [ (^.pred) = 'sauver<(^.SUBJ)(^.OBJ)>' ] ,
           morph=(aimer,sauv) .

travailler  [ (^.pred) = 'travailler<(^.SUBJ)>' ] ,
           morph=(aimer,travaill) .

! ----------------------------------------------------------
!       pronoms
! ----------------------------------------------------------
!
%défaut classe=pronom, (^.CAS)=SUBJ, cas1=nominatif ,
          pronom-type=personnel, (^.PRED)='PRO' .
je          personne=r première, nombre=r singulier .
```

```
tu          personne=r deuxième, nombre=r singulier .
il          personne=r troisième, nombre=r singulier, genre=r
                masculin .
elle        personne=r troisième, nombre=r singulier, genre=r
                féminin .
! ------------------------------------------------------------
!       prépositions
! ------------------------------------------------------------

%défaut classe=préposition .
à           (^.OBJOBL) = OBJ2 ,
                [ (^.OBJ.PREP-NOMBRE)=SINGULIER,
                  (^.OBJ.PREP-GENRE)=FEMININ ] .
a           (^.OBJOBL) = OBJ2 ,
                AUTOMATIC_ERROR=1 ,
                [ (^.OBJ.PREP-NOMBRE)=SINGULIER,
                  (^.OBJ.PREP-GENRE)=FEMININ ] .
à           (^.OBJOBL) = ADJUNCT ,
                [ (^.OBJ.PREP-NOMBRE)=SINGULIER,
                  (^.OBJ.PREP-GENRE)=FEMININ ] .
a           (^.OBJOBL) = ADJUNCT ,
                AUTOMATIC_ERROR=1 ,
                [ (^.OBJ.PREP-NOMBRE)=SINGULIER,
                  (^.OBJ.PREP-GENRE)=FEMININ ] .
au          (^.OBJOBL) = ADJUNCT , ENV_RULE = 3 .
au          (^.OBJOBL) = OBJ2 , ENV_RULE = 3 .
aux         (^.OBJOBL) = ADJUNCT , ENV_RULE = 4 .
aux         (^.OBJOBL) = OBJ2 , ENV_RULE = 4 .

de          (^.OBJOBL) = OBJ2 .
des         (^.OBJOBL) = OBJ2 .
d           (^.OBJOBL) = OBJ2 .
du          (^.OBJOBL) = OBJ2 , [ (^.OBJ.ACCORD-DU)=OUI ,
                (^.OBJ.NOMBRE)=SINGULIER,
                (^.OBJ.GENRE)=MASCULIN ] .
du          (^.OBJOBL) = ADJUNCT, (^.COMP-PREP)=PREP-DE,
                [ (^.OBJ.ACCORD-DU)=OUI,
                  (^.OBJ.NOMBRE)=SINGULIER,
(^.OBJ.GENRE)=MASCULIN ] .
```

b. Règles morphologiques

```
%Amodèles
conjug  nombre=singulier, personne=première ;
        nombre=singulier, personne=deuxième ;
        nombre=singulier, personne=troisième ;
        nombre=pluriel, personne=première ; nombre=pluriel,
        personne=deuxième ; nombre=pluriel,
        personne=troisième .
noms    nombre = singulier, genre =r masculin ;
        nombre = pluriel, genre =r masculin ;
        nombre = singulier, genre =r féminin ;
        nombre = pluriel, genre =r féminin .

adject  nombre=singulier, genre=masculin ;
        nombre=pluriel, genre=masculin ; nombre=singulier,
        genre=féminin ; nombre=pluriel, genre=féminin .
```

```
artdéf  nombre=r singulier, genre=masculin,
           prép-nombre=singulier, prép-genre=masculin ;
        nombre=r singulier, genre=féminin ,
        prép-nombre=singulier, prép-genre=féminin ;
         nombre=r pluriel, prép-nombre=pluriel .

%Tmodèles
! indicatif présent
c1  conjug      e, es, e, ons, ez, ent .
c11 conjug      is, is, it, issons, issez, issent .
c31 conjug      ais,as,a,ons,ez,ont .
c41 conjug      s,s,t,ons,ez,nent .
c51 conjug      s,s,t,tons,tez,tent .
c61 conjug      s,s,,tons,tez,tent .
c71 conjug      s,s,t,ons,ez,ent .
c81 conjug      s,s,t,vons,vez,vent .
c91 conjug      ds,ds,d,dons,dez,dent .

! imparfait
c2  conjug      ais,ais,ait,ions,iez,aient .
c12 conjug      issais, issais, issait, issions,
                      issiez,issaient.
c22 conjug      eais, eais, eait, ions, iez, eaient .
c52 conjug      tais,tais,tait,tions,tiez,taient .
c82 conjug      vais,vais,vait,vions,viez,vaient .
c92 conjug      dais,dais,dait,dions,diez,daient .

! masculin régulier
c202 noms       ,s,0,0 .

! adjectif régulier      (ex: rouge)
c203 adject     ,s,,s .


%Lmodèles
! modele 6
aimer c1    modev=conjugué, mode=indicatif,
                temps=présent; 6,1,1,1,1,1,1 ;
      c2    modev=conjugué, mode=indicatif,
                temps=imparfait; 6,1,1,1,1,1,1 ;
      c3    modev=conjugué, mode=indicatif,
                temps=futur-simple; 6,1,1,1,1,1,1 ;
      c4    modev=conjugué, mode=subjonctif,
                temps=présent ; 6,1,1,1,1,1,1 ;
      c5    modev=participe-passé ; 4,1,1,1,1  ;
      c6    modev=participe-présent ; 1,1 ;
      c7    modev=infinitif ; 1,1 ;
      c8    modev=conjugué, mode=indicatif,
                temps=passé-simple ; 6,1,1,1,1,1,1 ;
      c9    modev=conjugué, mode=impératif,
                temps=présent ; 6,1,1,1,1,1,1 .
! modele 8
manger c1   modev=conjugué, temps=présent,
                mode=indicatif ; 6,1,1,1,1,1,1 ;
       c22 modev=conjugué, temps=imparfait,
                mode=indicatif ; 6,1,1,1,1,1,1 ;
       c3   modev=conjugué, temps=futur,
                mode=indicatif ; 6,1,1,1,1,1,1 ;
```

```
       c4  modev=conjugué, temps=présent,
             mode=subjonctif ; 6,1,1,1,1,1,1 ;
       c25 modev=participe-passé ; 4,1,1,1,1  ;
       c26 modev=participe-présent ; 1,1 ;
       c27 modev=infinitif ; 1,1 ;
       c9  modev=conjugué, mode=impératif,
             temps=présent ; 6,1,1,1,1,1,1 .
!
fem_rég c201 ; 4,1,1,1,1 .
mas_rég c202 ; 4,1,1,1,1 .
adj_rég c203 ; 4,1,1,1,1 .
```

c. Règles de réécriture

```
! ----------------------------------------------------------
! propositions
! ----------------------------------------------------------
!
<P> :== <GN>   (^.SUBJ)=*
               (*.FRAME-TYPE)=GN
               (*.CAS)=SUBJ
               (^.NOMBRE)=r (*.NOMBRE)
               (^.PERSONNE)=r (*.PERSONNE)
               (^.GENRE)=r (*.GENRE)
               (^.FRAME-TYPE)=PHRASE
         <GV>  ^=* .
!
<GV> :== <VERBE>
          <GN>0          (^.OBJ)=*
                         (*.FRAME-TYPE)=GN
                         (^.COD-POSITION)=APRES-VERBE
          <GP>0          (^.*.OBJOBL)=*
                         (*.FRAME-TYPE)=GP
          <GV1>0         (^.XCOMP)=*
                         (*.FRAME-TYPE)=PHRASE
                         (*.MODE)=r INFINITIF
                         (*.MODEV)=r INFINITIF .

! ----------------------------------------------------------
! groupe du nom
! ----------------------------------------------------------

<GN> :== <*ARTICLE>
          <GA>0                    (^.ADJ-POSITION) =r AVANT-NOM
          <*NOM>
          <COMPL-DU-NOM>0 .

<GN> :== <GA>                     (^.ADJ-POSITION) =r AVANT-NOM
          <*NOM>
          <COMPL-DU-NOM>0 .

! ----------------------------------------------------------
! verbes
! ----------------------------------------------------------

! temps simple
```

```
<VERBE> :== <*VERBE> <*ADVERBE>0 .

<VERBE> :== <*ADVERBE> <*VERBE> <*ADVERBE>0 .

! temps composé

<VERBE> :== <*VAUX>
            <*ADVERBE>0
            <*VERBE>          .
```

d. Règles de correction

```
! ------------------------------------------------------------
!     sous-catégorisation
! ------------------------------------------------------------

GROUPE 1
 C:
  ATTR_ORDRE = PRED .

GROUPE 1.1
 C:
  PRED = 'donner' .

REGLE 1.1.1
 C:
  OBJ = *2
  OBJ.PRED = 'PRO'
 E:
  PROC011 .

REGLE 1.1.2
 C:
  OBJ2 = *2
  OBJ2.PRED = 'PRO'
 E:
  PROC012 .

REGLE 1.1.3
 C:
  OBJ = *2
 E:
  PROC016 .

! ..............................

GROUPE 1.2
 C:
 PRED = 'téléphoner' .

REGLE 1.2.1
 C:
  OBJ = *
 E:
```

```
  PROC013 .

! ------------------------------------------------------------
! personne
! ------------------------------------------------------------

GROUPE 3
 C:
  ATTR_ORDRE = PERSONNE .

REGLE 3.1
 E:
   PROC03 .

! ------------------------------------------------------------
! genre
! ------------------------------------------------------------

GROUPE 4
 C:
  ATTR_ORDRE = GENRE .

! ------------------------------------------------------------
! cas de la préposition 'au' suivie d'un nom féminin
! ------------------------------------------------------------

REGLE 4.1
 C:
   ACCORD-AU = OUI
 E:
   PROC17A .

! ------------------------------------------------------------
! cas de la préposition 'du' suivie d'un nom féminin
! ------------------------------------------------------------

REGLE 4.2
 C:
   ACCORD-DU = OUI
 E:
   PROC17DU .
!
REGLE 4.3
 C:
   ACCORD-AU = 0
   ACCORD-DU = 0
 E:
   PROC04 .

! ------------------------------------------------------------
! nombre
! ------------------------------------------------------------

GROUPE 5
 C:
  ATTR_ORDRE = NOMBRE .

! cas normal
```

```
REGLE 5.1
 C:
   ACCORD-AU = 0
   ACCORD-DU = 0
 E:
   PROC05 .

! ---------------------------------------------------------
! cas de 'à' suivi du pluriel
! ---------------------------------------------------------

REGLE 5.2
 C:
   ACCORD-AU = OUI
 E:
   PROC17A .

! ---------------------------------------------------------
! participe passé
! ---------------------------------------------------------

GROUPE 6
 C:
  ATTR_ORDRE = PP_GENRE .

REGLE 6.1
 E:
   PROC06 .

! ---------------------------------------------------------

GROUPE 7
 C:
  ATTR_ORDRE = PP_NOMBRE .
!
REGLE 7.1
 E:
   PROC07 .

! ---------------------------------------------------------
! PROCEDURES
! ---------------------------------------------------------

PROCEDURE PROC01
   END

PROCEDURE PROC011
! --- Procédure de correction du verbe donner ---
  PRINT "Mot : >+" ;
  PRINT OBJ.PRED ;
  PRINT "<+" ;
  BUILD_REFERENCE_FRAME ;
  USE OBJ.GENRE ;
  USE OBJ.NOMBRE ;
  USE OBJ.PERSONNE ;
  SET CAS1 TO DATIF ;
  SET CLASSE TO PRONOM ;
```

```
  SET PRONOM-TYPE TO PERSONNEL ;
  FIND_CORRECT_FORM ;
  PRINT " , remplacer par >+" ;
  PRINT CORRECT_FORM ;
  PRINT "<" ;
  DISABLE_TEXT_OUTPUT ;
  END

PROCEDURE PROC012
! --- Procédure de correction du verbe donner ---
  PRINT "Mot : >+" ;
  PRINT OBJ2.PRED ;
  PRINT "<+" ;
  BUILD_REFERENCE_FRAME ;
  USE OBJ2.GENRE ;
  USE OBJ2.NOMBRE ;
  USE OBJ2.PERSONNE ;
  SET CAS1 TO ACCUSATIF ;
  SET CLASSE TO PRONOM ;
  SET PRONOM-TYPE TO PERSONNEL ;
  FIND_CORRECT_FORM ;
  PRINT " , remplacer par >+" ;
  PRINT CORRECT_FORM ;
  PRINT "<" ;
  DISABLE_TEXT_OUTPUT ;
  END

PROCEDURE PROC013
   PRINT "Attention: le verbe 'téléphoner' demande un objet
          indirect" ;
   END

PROCEDURE PROC014
   PRINT "Attention, erreur possible : le verbe +" ;
   PRINT PRED ;
   PRINT " s'utilise " ;
   PRINT "avec un objet direct" ;
   END

PROCEDURE PROC015
   PRINT "Erreur possible : le verbe +" ;
   PRINT PRED ;
   PRINT " s'utilise avec une préposition." ;
   END
```

e. Règles lexicales

```
! ------------------------------------------------------------
!  Accords du participe passé
! ------------------------------------------------------------
!
!  <sujet> <verbe> <cod>
!
GROUPE 1
 C:
  FRAME-TYPE = PHRASE .
```

```
!
! accord avec le verbe avoir
!
GROUPE 1.1
 C:
  AUXILIAIRE = AVOIR
  MODEV = PARTICIPE-PASSE .
!
! le cod se trouve après le verbe
!
REGLE 1.1.1
 C:
  COD-POSITION = APRES-VERBE
 G:
  PP_GENRE =r MASCULIN
  PP_NOMBRE =r SINGULIER .
!
! le cod se trouve avant le verbe
!
REGLE 1.1.2
 C:
  COD-POSITION = AVANT-VERBE
 G:
  PP_GENRE =r OBJ.GENRE
  PP_NOMBRE =r OBJ.NOMBRE .
!
! le cod n'est pas défini
!
REGLE 1.1.3
 C:
  COD-POSITION = 0
  OBJ = 0
 G:
  PP_GENRE =r MASCULIN
  PP_NOMBRE =r SINGULIER .
!
REGLE 1.1.4
 C:
  COD-POSITION = 0
  OBJ = *
  OBJ.COD-POSITION = AVANT-VERBE
 G:
  PP_GENRE =r OBJ.GENRE
  PP_NOMBRE =r OBJ.NOMBRE .
!
! ------------------------------------------------------------
!  au
! ------------------------------------------------------------
!
GROUPE 3
 C:
  OBJ = *
  ENV_RULE = 3 .            ! introduit par 'au'
!
! première partie, article défini absent
!
REGLE 3.1
 C:
```

```
  OBJ.PREP-GENRE = 0
  OBJ.PREP-NOMBRE = 0
 G:
  OBJ.GENRE = MASCULIN
  OBJ.ACCORD-AU = OUI .
!
! seconde partie, article défini présent => erreur
! automatique
!
REGLE 3.2
 C:
  OBJ.PREP-GENRE = 0
  OBJ.PREP-NOMBRE = *
 G:
  AUTOMATIC_ERROR = 2
  OBJ.GENRE = MASCULIN
  OBJ.ACCORD-AU = OUI .
!
REGLE 3.2
 C:
  OBJ.PREP-GENRE = *
  OBJ.PREP-NOMBRE = 0
 G:
  AUTOMATIC_ERROR = 2
  OBJ.GENRE = MASCULIN
  OBJ.ACCORD-AU = OUI .
!
REGLE 3.2
 C:
  OBJ.PREP-GENRE = *
  OBJ.PREP-NOMBRE = *
 G:
  AUTOMATIC_ERROR = 2
  OBJ.GENRE = MASCULIN
  OBJ.ACCORD-AU = OUI .
```

Annexe 2. Session avec le logiciel LFG

Dans cette section, nous présentons une session interactive avec le logiciel LFG. Nous lui soumettons la phrase suivante:

* *Elle a **lancer la** cendrier.*

Cette phrase contient deux erreurs: *lancer* devrait être au participe passé et non à l'infinitif, et l'article *la* devrait être remplacé par *le*.

Dans le compte rendu de la session interactive, les données introduites par l'utilisateur sont soulignées et nos explications apparaissent en italiques.

```
C:> MSLFG

*** Analyseur LFG ***

... symboles
... descriptions morphologiques
... dictionnaire
... règles syntaxiques
... règles de bonne formation
... règles de correction

Introduire le texte avec ponctuation.
Taper <RETURN> ou <ENTER> pour terminer.

1> Elle a lancer la cendrier .

1>>> Mot : >lancer<, remplacer par >lancé<
2>>> Mot : >la<, remplacer par >le<
```

Le logiciel a détecté les deux erreurs et affiche deux messages qui décrivent les erreurs et donnent des suggestions de correction.

```
2> /
```

'/' permet d'entrer le mode de commandes.

```
CMD> S 1L
```

'S 1L' affiche toutes les solutions de l'analyse. Pour chaque solution, le logiciel indique le numéro de la f-structure, le numéro dans la table de la racine de l'arbre syntaxique, et la liste des erreurs détectées.

```
Solutions

> f-structure  9   chart 102   : 2 erreur(s)
  #0 f-str  0 Attribut incompatible : MODEV
  #1 f-str  5 Attribut incompatible : GENRE

> f-structure 12   chart 102   : 3 erreur(s)
  #0 f-str  0 Attribut incompatible : MODEV
  #1 f-str  5 Attribut incompatible : GENRE
  #2 f-str 12 (orig=0) Sous-catégorisation non satisfaite.

> f-structure 25   chart 103   : 4 erreur(s)
  #0 f-str 28 Attribut incompatible : GENRE
  #1 f-str 27 (orig=19) Sous-catégorisation non satisfaite.
  #2 f-str 25 (orig=15) Sous-catégorisation non satisfaite.
  #3 f-str 25 (orig=15) Attribut incompatible:AUTOMATIC_ERROR

> f-structure 29   chart 103   : 4 erreur(s)
  #0 f-str 21 Attribut incompatible : GENRE
  #1 f-str 31 (orig=19) Sous-catégorisation non satisfaite.
  #2 f-str 29 (orig=15) Sous-catégorisation non satisfaite.
  #3 f-str 29 (orig=15) Attribut incompatible:AUTOMATIC_ERROR

> f-structure 33   chart 103   : 3 erreur(s)
  #0 f-str 21 Attribut incompatible : GENRE
  #1 f-str 35 (orig=19) Sous-catégorisation non satisfaite.
  #2 f-str 33 (orig=15) Attribut incompatible:AUTOMATIC_ERROR

> f-structure 37   chart 103   : 3 erreur(s)
  #0 f-str 21 Attribut incompatible : GENRE
  #1 f-str 39 (orig=19) Sous-catégorisation non satisfaite.
  #2 f-str 37 (orig=15) Attribut incompatible:AUTOMATIC_ERROR
```

```
CMD> T 102
```

La commande 'T 102' affiche l'arbre syntaxique ayant comme racine l'élément 102. Chaque ligne représente un noeud dans l'arbre et décrit la règle de réécriture utilisée (numéro 1 dans le premier noeud) ainsi que le domaine de la phrase auquel la règle s'applique (du premier au sixième mot). '102' représente le numéro de l'élément de la table apparaissant dans la première solution.

```
<P'> (1) [0,6]
  <P> (3) [0,5]
    <GN> (10) [0,1]
      <*PRONOM> (-) [0,1]
    <GV> (4) [1,5]
      <VERBE> (29) [1,3]
        <*VAUX> (-) [1,2]
        <*VERBE> (-) [2,3]
      <GN> (14) [3,5]
        <*ARTICLE> (-) [3,4]
        <*NOM> (-) [4,5]
  <PONCT> (41) [5,6]
    <*PONCTUATION> (-) [5,6]
```

CMD> F CR 9

La commande 'F CR 9' permet d'afficher le réseau de f-structures ayant comme parent la f-structure numéro 9. On remarque que dans la f-structure 9 le trait MODEV a deux valeurs différentes. C'est également le cas pour le trait GENRE dans la f-structure 11.

```
F-STRUCTURE 9
  FRAME-TYPE        PHRASE
  SUBJ              -> f-structure 10
  NOMBRE            SINGULIER
  PERSONNE          TROISIEME
  GENRE             FEMININ
  MODEV             PARTICIPE-PASSE
  AUXILIAIRE        AVOIR
  TEMPS             PRESENT
  MODE              INDICATIF
  ENV_RULE          1
  MODEV             INFINITIF
  OBJ               -> f-structure 11
  COD-POSITION      APRES-VERBE
  PONC-TYPE         P_POINT
  PRED              'lancer<(^.SUBJ)(^.OBJ)>'
  PP_GENRE          MASCULIN
  PP_NOMBRE         SINGULIER

F-STRUCTURE 10
  CAS               SUBJ
  PRED              'PRO'
  PERSONNE          TROISIEME
  NOMBRE            SINGULIER
  GENRE             FEMININ
  CAS1              NOMINATIF
  PRONOM-TYPE       PERSONNEL
  FRAME-TYPE        GN

F-STRUCTURE 11
  NOMBRE            SINGULIER
  GENRE             FEMININ
```

```
    PREP-NOMBRE       SINGULIER
    PREP-GENRE        FEMININ
    PERSONNE          TROISIEME
    ART-TYPE          DEFINI
    GENRE             MASCULIN
    FRAME-TYPE        GN
```

CMD> T 103

Cette commande affiche la seconde solution de l'analyse syntaxique. Nous remarquons que cet arbre correspond à la structure '[Elle] a [lancer la cendrier]' alors que l'arbre précédent correspondait à '*[Elle] a lancer [la cendrier]'.*

```
<P'> (1) [0,6]
  <P> (3) [0,5]
    <GN> (10) [0,1]
      <*PRONOM> (-) [0,1]
    <GV> (5) [1,5]
      <VERBE> (25) [1,2]
        <*VERBE> (-) [1,2]
      <GV> (4) [2,5]
        <VERBE> (25) [2,3]
          <*VERBE> (-) [2,3]
        <GN> (14) [3,5]
          <*ARTICLE> (-) [3,4]
          <*NOM> (-) [4,5]
  <PONCT> (41) [5,6]
    <*PONCTUATION> (-) [5,6]
```

CMD> F CR 25

Cette commande affiche la f-structure d'une des salutions produisant l'arbre syntaxique ci-dessus. Nous remarquons quatre erreurs: GENRE dans la f-structure 28, la sous-catégorisation non-satisfaite de 'avoir' dans la f-structure 25, la sous-catégorisation non-satisfaite de 'lancer' dans la f-structure 27, et une erreur générée par une règle lexicale dans la f-structure 25 (AUTOMATIC_ERROR).

```
F-STRUCTURE 25
  FRAME-TYPE        PHRASE
  SUBJ              -> f-structure 26
  NOMBRE            SINGULIER
  PERSONNE          TROISIEME
  GENRE             FEMININ
  ENV_RULE          6
  MODEV             CONJUGUE
  TEMPS             PRESENT
  MODE              INDICATIF
  AUXILIAIRE        AVOIR
  XCOMP             -> f-structure 27
  PONC-TYPE         P_POINT
```

```
    PRED              'avoir<(^.SUBJ)(^.OBJ)>'
    AUTOMATIC_ERROR 199

  F-STRUCTURE 26
    CAS               SUBJ
    PRED              'PRO'
    PERSONNE          TROISIEME
    NOMBRE            SINGULIER
    GENRE             FEMININ
    CAS1              NOMINATIF
    PRONOM-TYPE       PERSONNEL
    FRAME-TYPE        GN

  F-STRUCTURE 27
    MODEV             INFINITIF
    AUXILIAIRE        AVOIR
    OBJ               -> f-structure 28
    COD-POSITION      APRES-VERBE
    MODE              INFINITIF
    FRAME-TYPE        PHRASE
    COMP-PREP         NO-PREP
    PRED              'lancer<(^.SUBJ)(^.OBJ)>'

  F-STRUCTURE 28
    NOMBRE            SINGULIER
    GENRE             FEMININ
    PREP-NOMBRE       SINGULIER
    PREP-GENRE        FEMININ
    PERSONNE          TROISIEME
    ART-TYPE          DEFINI
    GENRE             MASCULIN
    FRAME-TYPE        GN
```

CMD> C_C

Cette commande affiche le contenu complet de la table.
- La première colonne contient le numéro de l'élément.
- La deuxième colonne contient le symbole de gauche de la règle de réécriture appliquée.
- La troisième colonne contient le numéro de cette règle.
- La quatrième colonne contient le domaine de la phrase utilisé par la règle.
- La cinquième colonne décrit le statut de la règle.
- Dans la sixième colonne, A indique un noeud terminal (un mot, par exemple) et le nombre qui suit représente son numéro dans la phrase; B indique une règle de réécriture et les nombres qui suivent représentent les prédécesseurs de la règle, qui sont aussi des indices dans le tableau.

```
  CHART
     1 :              <*PRONOM>          [0,1] C+  A: 0
     2 :                   <GN>  (10)    [0,1] C+  B: 1
     3 :                   <GN>  (11)    [0,1] C-9 B: 1
```

```
 4 :              <PRONOM>  (33)  [0,1] C-9 B: 1
 5 :                  <P>   (3)  [0,1]     B: 2
 6 :                 <GN>  (19)  [0,1]     B: 2
 7 :       <*PREPOSITION>        [1,2] C+  A: 1
 8 :       <*PREPOSITION>        [1,2] C+  A: 2
 9 :  <*PREPOSITION-LOC>         [1,2] C+  A: 3
10 :             <*VAUX>         [1,2] C+  A: 4
11 :            <*VERBE>         [1,2] C+  A: 5
12 :               <GP>  (20)    [1,2]     B: 7
13 :               <GP>  (20)    [1,2]     B: 8
14 :              <GV1>   (6)    [1,2]     B: 9
15 :            <VERBE>  (29)    [1,2]     B: 10
16 :            <VERBE>  (32)    [1,2]     B: 10
17 :            <VERBE>  (29)    [1,2]     B: 10 0
18 :            <VERBE>  (25)    [1,2]     B: 11
19 :            <VERBE>  (25)    [1,2] C+  B: 11 0
20 :               <GV>   (4)    [1,2]     B: 19
21 :               <GV>   (5)    [1,2]     B: 19
22 :               <GV>   (7)    [1,2]     B: 19
23 :               <GV>   (8)    [1,2]     B: 19
24 :               <GV>   (4)    [1,2]     B: 19 0
25 :               <GV>   (5)    [1,2]     B: 19 0
26 :               <GV>   (4)    [1,2]     B: 19 0 0
27 :               <GV>   (4)    [1,2] C+  B: 19 0 0 0
28 :                <P>   (3)    [0,2] C+  B: 2 27
29 :               <P'>   (1)    [0,2]     B: 28
30 :               <P2>   (2)    [0,2]     B: 28
31 :               <P'>   (1)    [0,2] C+  B: 28 0
32 :            <*VERBE>         [2,3] C+  A: 6
33 :            <VERBE>  (25)    [2,3]     B: 32
34 :            <VERBE>  (29)    [1,3] C+  B: 10 0 32
35 :            <VERBE>  (25)    [2,3] C+  B: 32 0
36 :               <GV>   (4)    [1,3]     B: 34
37 :               <GV>   (5)    [1,3]     B: 34
38 :               <GV>   (7)    [1,3]     B: 34
39 :               <GV>   (8)    [1,3]     B: 34
40 :               <GV>   (4)    [1,3]     B: 34 0
41 :               <GV>   (5)    [1,3]     B: 34 0
42 :               <GV>   (4)    [1,3]     B: 34 0 0
43 :               <GV>   (4)    [1,3] C+  B: 34 0 0 0
44 :               <GV>   (4)    [2,3]     B: 35
45 :               <GV>   (5)    [2,3]     B: 35
46 :               <GV>   (7)    [2,3]     B: 35
47 :               <GV>   (8)    [2,3]     B: 35
48 :               <GV>   (4)    [2,3]     B: 35 0
49 :               <GV>   (5)    [2,3]     B: 35 0
50 :               <GV>   (4)    [2,3]     B: 35 0 0
51 :               <GV>   (4)    [2,3] C+  B: 35 0 0 0
52 :                <P>   (3)    [0,3] C+  B: 2 43
53 :              <GV1>   (6)    [1,3] C+  B: 9 51
54 :               <GV>   (5)    [1,3] C+  B: 19 0 51
55 :               <P'>   (1)    [0,3]     B: 52
56 :               <P2>   (2)    [0,3]     B: 52
57 :               <P'>   (1)    [0,3] C+  B: 52 0
58 :                <P>   (3)    [0,3] C+  B: 2 54
59 :               <P'>   (1)    [0,3]     B: 58
60 :               <P2>   (2)    [0,3]     B: 58
61 :               <P'>   (1)    [0,3] C+  B: 58 0
```

```
 62 :              <*PRONOM>          [3,4] C+   A: 7
 63 :             <*ARTICLE>          [3,4] C+   A: 8
 64 :                  <GN>   (10)    [3,4] C-9  B: 62
 65 :                  <GN>   (11)    [3,4] C-9  B: 62
 66 :               <PRONOM>  (33)    [3,4] C+   B: 62
 67 :                  <GN>   (14)    [3,4]      B: 63
 68 :                  <GN>   (14)    [3,4]      B: 63 0
 69 :                <VERBE>  (27)    [3,4]      B: 66
 70 :                <VERBE>  (31)    [3,4]      B: 66
 71 :                <VERBE>  (27)    [3,4]      B: 66 0
 72 :                <VERBE>  (31)    [3,4]      B: 66 0
 73 :                 <*NOM>          [4,5] C+   A: 9
 74 :                  <GN>   (18)    [4,5]      B: 73
 75 :                  <GN>   (14)    [3,5]      B: 63 0 73
 76 :                  <GN>   (18)    [4,5] C+   B: 73 0
 77 :                  <GN>   (14)    [3,5] C+   B: 63 0 73 0
 78 :                   <P>    (3)    [4,5]      B: 76
 79 :                  <GN>   (19)    [4,5]      B: 76
 80 :                   <P>    (3)    [3,5]      B: 77
 81 :                  <GN>   (19)    [3,5]      B: 77
 82 :                  <GV>    (4)    [1,5]      B: 34 77
 83 :                  <GV>    (5)    [1,5]      B: 34 77
 84 :                  <GV>    (4)    [2,5]      B: 35 77
 85 :                  <GV>    (5)    [2,5]      B: 35 77
 86 :                  <GV>    (4)    [1,5]      B: 34 77 0
 87 :                  <GV>    (4)    [2,5]      B: 35 77 0
 88 :                  <GV>    (4)    [1,5] C+   B: 34 77 0 0
 89 :                  <GV>    (4)    [2,5] C+   B: 35 77 0 0
 90 :                   <P>    (3)    [0,5] C+   B: 2 88    ,
 91 :                 <GV1>    (6)    [1,5] C+   B: 9 89
 92 :                  <GV>    [5]    [1,5] C+   B: 19 0 89
 93 :                  <P'>    [1]    [0,5]      B: 90
 94 :                  <P2>    (2)    [0,5]      B: 90
 95 :                  <P'>    (1)    [0,5] C+   B: 90 0
 96 :                   <P>    (3)    [0,5] C+   B: 2 92
 97 :                  <P'>    (1)    [0,5]      B: 96
 98 :                  <P2>    (2)    [0,5]      B: 96
 99 :                  <P'>    (1)    [0,5] C+   B: 96 0
100 :       <*PONCTUATION>            [5,6] C+   A: 10
101 :              <PONCT>   (41)    [5,6] C+   B: 100
102 :                  <P'>    (1)    [0,6) C+   B: 90 101
103 :                  <P'>    (1)    [0,6) C+   B: 96 101
```

Annexe 3. Logiciel LFG: instructions pour construire les phrases-test

Remarques générales

A. Chaque phrase doit être représentative de la catégorie d'erreurs dans laquelle vous la placez et ne doit contenir aucune autre erreur.

B. Ne pas introduire des mots contenant des fautes d'orthographe. Par exemple :

Dans la catégorie (1), 'Accord du verbe avec le sujet', mettez le verbe à la mauvaise personne ou au mauvais nombre, mais ne modifiez pas l'orthographe pour qu'il ne se trouve plus dans la table de conjugaison. Ne modifiez pas non plus le temps (présent, imparfait) ou le mode (indicatif, subjonctif), sauf dans les cas où l'erreur à introduire l'exige.

C. N'utilisez pas le gérondif, l'impératif, ou la forme interrogative.

D. Vous pouvez utiliser des adverbes, mais ceux-ci doivent rester relativement simples. Evitez les locutions adverbiales.

E. Vous pouvez utiliser des compléments circonstanciels pour compliquer un peu les phrases. (Comme 'Toute la journée', 'Dans le train', 'Avec grande difficulté', etc.)

Structure des phrases

Forme globale : Phrases affirmatives uniquement. Les phrases interrogatives et impératives ne sont pas traitées.

La seule autre contrainte s'applique aux syntagmes nominaux, qui peuvent contenir :

- un nom
- un ou deux noms propres (ex. *Robert Leblanc*)
- un article
- un adjectif qualificatif (ex. *rond*, *bleu*)
- un adverbe de quantité (ex. *très*)
- un adjectif possessif ou démonstratif (ex. *mon*, *ces*)
- un pronom
- un complément du nom (du type syntagme prépositionnel ou pronom relatif suivi d'une proposition) (ex. *La copine de mon frère. La bague que tu as presque achetée*)

Liste des erreurs

Les pages qui suivent représentent des fiches de travail pour chaque catégorie d'erreurs. Chaque fiche contient une description du type d'erreur ainsi que les contraintes qui s'appliquent lors du choix des phrases. Assurez-vous de bien respecter ces contraintes.

Pour chaque erreur on trouvera des exemples de cas traités par le logiciel ainsi que de cas non traités. Etant donné qu'une liste exhaustive de ces derniers prendrait trop de place, ou est simplement impossible à constituer, référez-vous toujours aux contraintes décrites ci-dessus en cas de doutes.

CATEGORIE 1

Accord sujet-verbe

Description

Accord en nombre et en personne du verbe avec le sujet.

Cas traités

- Sujet simple : pronom ou syntagme nominal simple.
  * *Tu mange une pomme.*
  * *Les fleurs n'aime pas le soleil.*
- Sujet contenant une proposition subordonnée.
  * *La fille qui regarde les fleurs n'aiment pas le soleil.*

Cas non traités

- La forme infinitive ou impérative des verbes.

Remarques

Mettez le verbe à la mauvaise personne ou au mauvais nombre, mais ne modifiez pas l'orthographe pour qu'il ne se trouve plus dans la table de conjugaison.

CATEGORIE 2

Accord du syntagme nominal

Description

Accord en genre et en nombre des articles, adjectifs et du nom dans le syntagme nominal.

Cas traités

- Articles définis et indéfinis.
  * *La chien a mangé mon souller.*
- Adjectifs possessifs (mon, ma, leur) et démonstratifs (cet, cette).
  * *Il a reçu cette cadeau.*
- Adjectifs numéraux ordinaux simples, c'est-à-dire lorsqu'un seul mot est utilisé.
  * *Les premier bateaux sont déjà arrivés.*
- Adjectifs qualificatifs.

* *Ce garçon ne veut pas se baigner dans l'eau froid.*

- Noms.

* *Tous les enfant doivent essayer de nager.*

Cas non traités

- Les nombres complexes.

* *Elle a perdu cent vingt-trois franc aux courses.*

- Les participes à l'intérieur du syntagme nominal

* *Les enfants mangé par les abeilles...*

Remarques

N'introduisez pas de mots mal orthographiés pour tester la correction (éviter 'chevals', 'marteaus', etc.)

CATEGORIE 3

Participe passé: confusion entre infinitif et participe passé

Description

Confusion entre la forme infinitive et le participe passé des verbes en -er.

Cas traités

- Utilisation de l'infinitif au lieu du participe passé.

* *Il a refuser de manger sa soupe.*

- Utilisation du participe passé en '-é' dans une proposition infinitive.

* *Il a refusé de mangé sa soupe.*

Cas non traités

- Lorsque le participe n'est pas accompagné d'un auxiliaire.
  * *Il montrait un regard égarer.*

CATEGORIE 4

Participe passé: accord avec 'être'

Description

Accord du participe passé utilisé avec l'auxiliaire être.

Cas traités

- Forme usuelle : accord avec le sujet.
  * *La petite fille est tombé de l'arbre.*
- Accord des verbes pronominaux (accord avec le sujet ou invariable).
  * *Elles se sont baigné dans la rivière.*
  * *Elles se sont lavées les mains.*

Cas non traités

- Pour les verbes pronominaux : lorsque le complément d'objet direct se trouve avant le verbe.
  * *Tu ne peux imaginer les chases que je me suis dit.*

CATEGORIE 5

Participe passé: accord avec 'avoir'

Description

Accord du participe passé utilisé avec l'auxiliaire avoir.

Cas traités

- Forme usuelle : invariable.
  * *Les enfants ont mangés des pommes.*
- Accord avec le complément d'objet direct si celui-ci est placé avant le verbe.
  * *Il regarde les photos que j'ai pris.*

Cas non traités

- Les temps surcomposés.
  * *Cette poire, quand je l'ai eu mangé ...*
- Participe passé en rapport avec *l'* .
  * *Cette rue est moins bruyante que je ne l'avais crue.*

Ainsi que d'autres cas particuliers.

CATEGORIE 6

Mode dans les propositions

Description

Utilisation du subjonctif et de l'indicatif dans les propositions subordonnées complément d'objet direct non infinitives introduites par la conjonction 'que'.

Cas traités

- L'utilisation du subjonctif ou de l'indicatif est vérifiée dans les cas ... <verbe> ... que <proposition>.

  * *Je veux que vous venez à la maison après le match.*

Cas non traités

- Lorsqu'une négation est attachée au verbe principal et modifie ainsi le mode de la proposition subordonnée.

  * *Je ne pense pas qu'il part ce soir.*

- La proposition subordonnée en tant que complément circonstanciel.

  * *J'irai te voir avant que tu pars.*

Remarques

Faites attention de bien introduire la distinction entre l'indicatif et le subjonctif. Celle-ci n'est pas toujours claire:

'regardiez' : deuxième personne, pluriel, subjonctif présent
'regardiez' : deuxième personne, pluriel, indicatif imparfait

Dans un cas comme celui-ci, le logiciel ne verra pas la distinction entre indicatif et subjonctif, parce qu'il n'y en a simplement pas. Choisissez

plutôt un verbe où la forme subjonctive ne se retrouve pas à un autre temps ou à un autre mode. (Toutes les personnes du singulier pour certains verbes comme 'écrire', 'dire', 'faire', etc.)

CATEGORIE 7

Structure des compléments du verbe

Description

Construction des compléments du verbe.

Cas traités

Seules les structures suivantes sont acceptées:

- des syntagmes nominaux

  (ex. *Il mange une banane*)

- des syntagmes prépositionnels

  (ex. *Il téléphone toujours à sa soeur*)

- une conjonction suivie d'une proposition

  (ex. *Il pense que tu vas venir*)

- une prèposition (à, de) suivie d'une proposition

  (ex. *Il me permet de prendre sa voiture*)

CATEGORIE 8

Position des adjectifs

Description

Les adjectifs de couleur et de forme se placent après le nom.

Cas traités

- Les syntagmes nominaux avec un seul adjectif qualificatif.

  * *Les présidents soupent autour d'une ronde table.*

Cas non traités

- Plus d'un adjectif qualificatif par syntagme nominal.

  * *Il a acheté une petite rouge voiture.*

- Les adjectifs composés.

  * *Sur son dessin, il y avait un bleu foncé ciel.*

CATEGORIE 9

Les prépositions 'de' et 'à': préposition + infinitif

Description

Utilisation des prépositions *à* et *de* après certains verbes et devant l'infinitif.

Cas traités

- ... <verbe> ... <préposition> <prop. infinitive>

  * *Il a clairement refusé à signer la pétition.*

- Les verbes qui s'utilisent sans préposition.

  * *Il espère de récupérer son argent.*

Cas non traités

- Les combinaisons telles que *être disposé*, *valoir mieux* et *aimer mieux*.

  * *Il est disposé de nous payer cette somme.*

CATEGORIE 10

Les prépositions 'à' et 'de': confusion entre 'a' et 'à'

Description

Omission de l'accent sur la préposition 'à'.

Cas traités

- Utilisation de la préposition 'à' sans accent à l'intérieur d'un syntagme prépositionnel ou devant une proposition infinitive.

  * *Il hésite a écrire ses mémoires.*

Cas non traités

Aucun en particulier.

Remarques

Le logiciel ne détecte pas l'erreur dans l'autre sens, c'est-à-dire lorsque le verbe avoir à la troisième personne du singulier est écrit avec un accent.

CATEGORIE 11

Les prépositions 'à' et 'de': préposition + 'le', 'la', 'les'

Description

Utilisation de 'au', 'aux', 'de' et 'des'.

Cas traités

- 'au' suivi de 'le', 'la' ou 'les'.

* *Elle a donné des chocolats au le maître.*

- 'à' suivi de 'le' ou 'les'.

* *Elle a donné des chocolats à les enfants.*

- 'du' suivi de 'la' ou 'les'.

* *Ce renard vient du la forêt.*

Cas non traités

- 'du' suivi de 'le'.

Annexe 4. Logiciel automates: formalisme

Nous présentons ci-dessous le formalisme utilisé pour la définition des automates.

a. Répétitions d'arcs

Des caractères de contrôle peuvent être placés après chaque arc. Ainsi:

*	Indique que les conditions s'appliquent sur un nombre indéterminé de mots qui se suivent. Ce nombre peut être égal à zéro. Parmi les options décrites ici, celle-ci est la seule qui soit mise en oeuvre dans le formalisme de Winograd.
+	Indique que les conditions s'appliquent à un ou plusieurs mots qui se suivent.
*n	Indique que les conditions s'appliquent 0, 1, ..., ou n fois.
+n	indique que les conditions s'appliquent 1, 2, ..., ou n fois.

L'automate ci-dessous, qui détecte une erreur d'accord du nombre entre le sujet et le verbe (ici entre *he* et *eat*), illustre l'utilisation des caractères de contrôle:

```
AUTOMATE 2
[FC='he']  [CAT=ADV]*  @[FF='eat']
```

Explication:

- FF signifie 'forme fléchie' et FC 'forme canonique'.
- L'automate est ancré sur le troisième arc, qui recherche le mot *eat*.
- Il recherche ensuite le mot *he* à la gauche de *eat* tout en sautant les adverbes.
- Le troisième arc, [FF='eat'], n'acceptera pas les déclinaisons comme *ate*, *eating* et *eats*.

Le signe '^' placé avant un arc indique que le processeur d'automates doit sauter tous les mots du texte qui ne satisfont pas aux conditions de l'arc. Le logiciel va donc chercher, en avant ou en arrière, le prochain mot correspondant à un tel arc.

Exemple:

```
AUTOMATE 3
@[FF='the']  ^[CAT=N]
```

Explication:

- L'automate est ancré sur le premier arc.
- Il détecte la présence de l'article *the* suivi dans la phrase d'un nombre indéterminé de mots puis d'un nom.

b. Opérations logiques

Le contenu des arcs consiste en une série de conditions s'appliquant aux traits du texte d'entrée. Celles-ci sont liées par des opérateurs logiques et des parenthèses:

- L'opération logique ET est représentée par une virgule.
- L'opération logique OU est représentée par la barre verticale '|'.

Nous pouvons ainsi améliorer l'automate qui détecte une erreur d'accord sujet-verbe de la manière suivante:

```
AUTOMATE 4
[CAT=N, NBR=SING]  [CAT=ADV]*  @[FF='eat'
     | FF='drink']
```

Explication:

- L'ancrage de l'automate se fait sur *eat* ou *drink*.
- Une fois l'ancrage trouvé, l'automate recherche un mot qui soit un nom et dont le nombre est singulier.
- Tous les adverbes situés entre le nom et le verbe *eat* ou *drink* sont sautés.

c. Listes de traits

On spécifie l'appartenance à une liste à l'intérieur d'un automate de la manière suivante:

+NTEMP indique que le mot doit appartenir à la liste

NTEMP.

~VNEVERCONT indique que le mot ne doit pas se trouver dans la liste VNEVERCONT.

L'automate ci-dessous utilise une liste de traits pour détecter les erreurs relatives à l'utilisation de verbes qui ne se mettent jamais à la forme continue.

```
AUTOMATE 5
@[CAT=V, ASPECT_FV=CONT, +V_NEVERCONT]
```

Explication:

- L'automate n'est formé que d'un arc.
- L'ancrage se fait sur un verbe faisant partie d'une forme verbale continue. Le trait ASPECT_FV est créé lors de la phase de pré-traitement.
- L'automate vérifie également que la forme canonique du verbe se trouve dans la liste V_NEVERCONT.

d. Registres

On distingue deux manières d'utiliser les registres: l'affectation d'une valeur d'un trait à un registre et le test du contenu de celui-ci. Pour affecter la valeur, l'opération suivante est utilisée:

<trait> => <registre>

On peut aussi affecter une valeur spécifique à un registre:

<valeur> => <registre>

Les exemples ci-dessous illustrent la différence entre ces deux possibilités:

TNS_FV => $T	met la valeur du trait TNS_FV pour le mot courant dans le registre $T. TNS_FV représente le temps d'une forme verbale et prend des valeurs telles que PRES (présent), FUT (futur) ou PCOMP (passé composé).
PRES => $T	met la valeur PRES (présent) dans le registre $T.

Pour tester le contenu du registre, le type d'opération qui correspond à son contenu est utilisé:

- Si le registre contient la valeur d'un trait symbolique (PERS, CAT, etc.), les opérateurs symboliques '=' et '/=' (différent de) sont utilisés.

- Si le registre contient une chaîne de caractères, on peut utiliser les opérations sur les listes ou les opérations qui comparent la chaîne de caractère à une valeur donnée.

Deux possibilités se présentent lorsqu'on fait référence à la valeur d'un registre:

a. Le registre est comparé à une valeur donnée et il est alors placé à gauche de l'opérateur:

$X = PRES

b. Le registre est comparé à la valeur d'un trait du mot courant et il est alors placé à droite de l'opérateur:

TNS_FV = $X

L'exemple suivant illustre l'utilisation de registres dans l'automate de détection d'erreurs de l'accord sujet-verbe:

```
AUTOMATE 6
 [CAT=N, NBR=SING, CURPOS=>$L]
 [CAT=ADV]*
@[CAT=V, TNS=PRES, MORPH/=M3S, CURPOS=>$R, FF=>$A]
```

Explication:

- L'ancrage de l'automate se fait sur un verbe au présent et qui n'est pas à la troisième personne du singulier.
- Une fois l'ancrage trouvé, l'automate recherche un mot qui soit un nom et dont le nombre est singulier.
- Tous les adverbes situés entre le nom et le verbe sont sautés.
- La position du nom à l'intérieur de la phrase est enregistrée dans le registre $L et celle du verbe dans le registre $R. Ceci permet au module de correction de connaître le segment de phrase où se trouve l'erreur et de l'afficher à l'écran.
- La forme fléchie du verbe qui contient l'erreur est enregistrée dans le registre $A afin que le module de correction puisse l'afficher.
- L'automate ne traite que les cas où le sujet précède directement le verbe (sans compter les adverbes). Il ne traite donc pas les situations plus complexes, comme ceux où une proposition relative est présente entre le sujet et le verbe.

e. Création dynamique de traits

Il est possible de stocker des valeurs au niveau des mots et des syntagmes:

1. Dans un mot, le stockage se fait à l'aide de l'opérateur ':='. Le nouveau trait est alors attribué à ce mot seul.
2. Dans un syntagme nominal, il se fait à l'aide de '::='. Le nouveau trait est attribué au SN en entier.

Un nouveau trait est défini en appliquant l'opération d'assignation soit sur une valeur symbolique (la valeur d'un trait) soit sur une chaîne de caractères de la manière suivante:

Pour une valeur symbolique: SUJ_NBR ::= SING. La valeur SING (singulier) est affectée au trait SUJ_NBR du syntagme nominal.

Lorsque la valeur se trouve dans un autre trait: SUJ_NBR ::= NBR. Le nombre du mot courant est affecté au trait SUJ_NBR du syntagme nominal.

Pour une chaîne de caractères: PREP_LINK := 'to' . Ici, la valeur *to* est affectée au trait PREP_LINK du mot courant.

Il est aussi possible d'attribuer la valeur d'un registre à un trait avec les opérations ':=' et '::='. Par exemple:

PREP_LINK := $R

f. Syntagmes nominaux

La position et les caractéristiques des syntagmes nominaux peuvent s'avérer très utiles pour la détection et la correction de certaines erreurs. Comme chez Winograd, nous avons inclus dans le formalisme des automates la possibilité de faire référence à ces syntagmes. Ainsi, les symboles '<' et '>' délimitent un environnement représentant un syntagme. Les conditions liées au syntagme sont placées entre ceux-ci:

<SN conditions>

Ces environnements s'utilisent comme des arcs; ils sont insérés dans les automates de la manière suivante:

```
AUTOMATE 7
 <SN [CAT=N, +UNC_N]>
@[CAT=V, TNS=PRES, MORPH/=M3S]
```

Explication:

- L'ancrage de l'automate se fait sur un verbe au présent et qui n'est pas à la troisième personne du singulier.
- L'automate recherche ensuite un syntagme à gauche de l'ancrage. Le syntagme doit être formé d'un nom non comptable (comme *courage*, *rain*).

Les conditions à l'intérieur des environnements peuvent être de deux types:

1- Des conditions sur les traits de l'environnement tels qu'ils ont été identifiés lors d'un pré-traitement syntaxique. On spécifie ces

conditions de la même manière qu'un arc et on les insère immédiatement après le début de l'environnement. Afin de distinguer ces traits on les insère entre des parenthèses rondes: '(' et ')'.

2- Des conditions sur les composants de l'environnement. On spécifie ces conditions en les entourant des signes '[' et ']', comme si on décrivait un arc.

Exemple:

```
AUTOMATE 8
<SN  (NBR_SN=SING)  [CAT=ART]  @[CAT=N]>
```

Explication:

- L'automate n'opère qu'à l'intérieur d'un syntagme nominal.
- Il identifie un syntagme nominal ayant le trait global 'singulier' et qui est formé d'un article suivi d'un nom.

Un environnement peut ne pas contenir de conditions spécifiques, comme dans l'automate suivant qui détecte deux SN séparés par *of*.

```
AUTOMATE 9
<SN>  @[FC='of']  <SN>
```

Explication:

- L'automate est ancré sur le mot *of*.
- Il recherche deux syntagmes nominaux, un de chaque côté de l'ancrage.

Il est possible de créer un environnement directement à partir d'un automate, et alors d'autres automates pourront y faire référence. Ceci se fait en trois étapes:

1- Identifier la borne à gauche du nouvel environnement. Pour ceci, on utilise le trait pré-défini CURPOS qu'on enregistre dans le registre $L (pour Left, gauche).

2- Identifier la borne à droite du nouvel environnement. On enregistre CURPOS dans le registre $R (pour Right, droite).

3- Enregistrer le nouvel environnement à l'aide du trait NEWSN, qu'il suffit d'utiliser dans un arc pour que l'action soit exécutée.

Exemples:

1- Définir un SN composé d'un article et d'un nom:

```
AUTOMATE 10
@[CAT=ART, CURPOS=>$L]
 [CAT=N, CURPOS=>$R]
 (+NEWSN)
```

Explication:

- L'automate est ancré sur un article. La position de l'article est enregistrée dans le registre $L.

- L'automate recherche ensuite un nom, dont la position est enregistrée dans le registre $R.
- L'exécution de l'arc qui contient le trait +NEWSN crée un syntagme nominal défini par les registres $L et $R.

2- Regrouper deux SN reliés par *of*:

```
AUTOMATE 11
    (+NEWSN)
    <SN   @[EPOS=EFIR, CURPOS=>$L]>
   @[FC='of']
    <SN   @[EPOS=ELAS, CURPOS=>$R]>
```

Explication:

- L'automate est ancré sur le mot *of*.
- Il recherche deux syntagmes nominaux, un de chaque côté de l'ancrage.
- Les conditions EPOS=EFIR et EPOS=ELAS existent respectivement pour le premier et le dernier mot d'un syntagme. Ceci permet d'enregistrer l'endroit où débute le syntagme nominal de gauche dans le registre $L et l'endroit où finit le syntagme nominal de droite dans le registre $R.
- L'exécution de l'arc qui contient le trait +NEWSN crée un syntagme nominal défini par les registres $L et $R.

Il est également possible d'assigner des traits globaux au SN nouvellement créé. Ceci se fait en plaçant des actions globales (opérateur ::=) juste après l'utilisation du trait NEWSN, comme dans l'exemple suivant:

```
AUTOMATE 12
   (+NEWSN, SN_NBR::=SING)
   <SN  @[EPOS=EFIR, CURPOS=>$L] (SN_NBR=SING) >
  @[FC='of']
   <SN  @[EPOS=ELAS, CURPOS=>$R]>
```

Explication:

- Le syntagme nominal de gauche doit être au singulier.
- L'exécution de l'arc qui contient le trait +NEWSN crée un syntagme nominal défini par les registres $L et $R. Le trait 'singulier' est affecté au nouveau syntagme.

g. Traits pré-définis

Outre les traits contenus dans le dictionnaire principal du logiciel et dans les listes, le module d'exécution des automates définit lui-même un certain nombre de traits. En voici quelques-uns:

POS — possède une des valeurs suivantes: FIR, MID ou LAS. Ce trait est défini pour chaque mot du texte, suivant qu'il est le premier mot de la phrase (FIR), le dernier (LAS) ou entre ces deux (MID).

EPOS — similaire à POS, ce trait s'utilise dans les environnements. Les valeurs de EPOS sont EFIR, EMID et ELAS.

CURPOS — contient la position du mot relative au début de la phrase. CURPOS vaut 1 pour le premier mot, 2 pour le deuxième mot, etc.

CONTR indique si le mot se trouve sous forme contractée dans le texte. Ce trait prend trois valeurs: 0, 1 et 2. La valeur 0 est donnée au mot qui n'est pas contracté. Cette condition est testée à l'aide des deux expressions suivantes:

-CONTR et CONTR = 0

Les valeurs 1 et 2 indiquent si on traite de la partie de gauche ou de droite d'un mot contracté. Prenons l'exemple de *I can't*, qui est décomposé en *I* + *can* + *not*. Les affirmations suivantes sont vraies pour ces trois mots:

I	-CONTR	CONTR=0
can	+CONTR	CONTR=1
not	+CONTR	CONTR=2

NEWSN Crée un environnement SN dont les bornes sont les valeurs des registres $R (borne de droite) et $L (borne de gauche). Si ces registres ne sont pas définis, les délimitations de l'automate sont utilisées à la place. Seule la forme +NEWSN est acceptée.

EXCLUDE Ce trait empêche tous les autres automates d'agir sur la même suite de mots. Considérons l'automate suivant:

```
AUTOMATE 13
@[CAT=ART]   [CAT=N]   (+EXCLUDE)
```

Explication:

- L'automate est ancré sur un article.
- Il détecte un article suivi d'un nom.
- Aucun autre automate ne peut utiliser l'article et le nom sur lesquels cet automate est basé.

MORPH Ce trait a 4 valeurs possibles:

M_1S M_2S M_3S M_P

indiquant la première, la deuxième et la troisième personne du singulier ainsi que toutes les personnes du pluriel.

h. Glossaire des abréviations utilisés dans les automates

Cette liste contient des abréviations de deux types:
- Un trait. Dans ce cas, ses valeurs possibles sont également indiquées.
- La valeur d'un trait. Dans ce cas, le trait auquel la valeur appartient est indiqué.

ACT Voix active de la forme verbale (trait: VOIX_FV)

ADJ	Adjectif (trait: CAT)
ADV	Adverbe (trait: CAT)
AFF	Positif (trait: STATUT_FV)
ART	Article (trait: CAT)
ASPECT_FV	Aspect de la forme verbale (valeurs: CONT, SIMP)
CAT	Catégorie syntaxique (valeurs: ADJ, ADV, etc.)
CONT	Continu (trait: ASPECT_FV)
CONTR	Forme contractée (valeurs: 0=pas de contraction, 1=partie de gauche, 2=partie de droite)
CURPOS	Position dans la phrase (valeurs: FIR, MID, LAS)
EFIR	Position initiale dans un SN (trait: EPOS)
ELAS	Position finale dans un SN (trait: EPOS)
EMID	Position ni initiale ni finale dans un SN (trait: EPOS)
EPOS	Position à l'intérieur d'un SN (valeurs: EFIR, EMID, ELAS)
FIR	Position initiale dans la phrase (trait: CURPOS)
FC	Forme canonique (valeurs: un mot)
FF	Forme fléchie (valeurs: un mot)
FUT	Futur (trait: TNS_FV)
INF	Infinitif (trait: MODE)
M1S	Première personne du singulier (trait: MORPH)
M2S	Deuxième personne du singulier (trait: MORPH)
M3S	Troisième personne du singulier (trait: MORPH)
MP	Pluriel (trait: MORPH)
MID	Position ni initiale ni finale dans la phrase (trait: CURPOS)
LAS	Position finale dans la phrase (trait: CURPOS)

MODE	Mode des verbes (valeurs: INF, PART)
MORPH	Morphologie (valeurs: M1S, M2S, M3S, MP)
NEG	Négatif (trait: STATUT_FV)
NEWSN	Trait utilisé pour créer un syntagme nominal.
PASSIF	Voix passive de la forme verbale (trait: VOIX_FV)
PCOMP	Passé composé (trait: TNS_FV)
PART	Participe (traits: TNS, MODE)
PERS	Personne (valeurs: P1, P2, P3)
PREP	Préposition (trait: CAT)
PRES	Présent (traits: TNS_FV et TNS)
PRET	Prétérit (trait: TNS)
PRON	Pronom (trait: CAT)
SING	Singulier (trait: NBR)
SIMP	Aspect simple (trait: ASPECT_FV)
SN	Syntagme nominal
STATUT_FV	Statut de la forme verbale (valeurs: AFF, NEG)
TNS	Temps d'un verbe (valeurs: PART, PRES, etc.)
TNS_FV	Temps de la forme verbale (valeurs: PRES, FUT, PCOMP, etc.)
V	Verbe (trait: CAT)
VOIX_FV	Voix de la forme verbale (valeurs: ACT, PASSIF)

Annexe 5. Session avec le logiciel de vérification des automates (VELO)

Dans cette annexe, nous présentons une session interactive avec le logiciel basé sur l'approche des automates. Nous lui soumettons la phrase suivante:

* *She asked where* ***was I*** *going.*

Cette phrase contient une erreur: *was* et *I* devraient être permutés.

Dans ce compte rendu, les données introduites par l'utilisateur sont soulignées et nos explications apparaissent en italiques.

```
C:> VELO2
```

Le logiciel affiche les fichiers qu'il charge et les commandes d'initialisation qu'il exécute. Celles-ci sont lues dans le fichier velo2.scr.

```
VELO Version 2.8.a

>> fichier des automates chargé: pret0194.aut.
>> fichier des automates chargé: erret2.aut.
>> fichier des automates chargé: part2.aut.
>> Désambiguïsation activée
>> fichier d'entrée chargé: demo1.tag.
>> fichier d'agencement chargé: erret2.age.
>> Listes chargées: pret0394.lst
>> Sortie: écran
>> Affichage des résultats: phrase entière
>> Affichage: résumé
```

```
>> Exécution: agenceur
>> Entrée: phrase par phrase
>> Nombre de détections avant arrêt: 1
>> Historique = 2
```

```
(1) e
```

La commande 'e' lit la prochaine phrase du fichier d'entrée (demo1.tag)

```
Phrase cat1.1.0
>> prochaine phrase chargée
```

```
(2) p
```

La commande 'p' affiche la phrase courante, y compris les bornes des syntagmes nominaux.

```
>> [ he ] asked where was [ I ] going .
```

```
(3) t
```

La commande 't' affiche les traits des mots de la phrase courante. Les traits contenus dans la seconde ligne proviennent du dictionnaire CELEX.

```
1 [0]:  FF='he'  FC='he'  CAT=PRON  morph=*
   Pers_PRON Pron_PRON

2 [1]:  FF='asked'  FC='ask'  CAT=V  morph=a1S+a2S+a3S+aP+pa
   Trans_V TransComp_V Intrans_V Ditrans_V

3 [2]:  FF='where'  FC='where'  CAT=PRON  morph=*
   Wh_PRON Pron_PRON

4 [3]:  FF='was'  FC='be'  CAT=V  morph=a1S+a3S
   Intrans_V Link_V

5 [4]:  FF='I'  FC='I'  CAT=PRON  morph=*
   Pers_PRON Pron_PRON

6 [5]:  FF='going'  FC='go'  CAT=V  morph=pe
   Intrans_V Link_V

7 [6]:  FF='.'  FC='.'  CAT=PCT  morph=*
```

```
(4) g
```

La commande 'g' lance l'agenceur. Le logiciel affiche un message indiquant que l'automate de détection ENOORD_INTERINDI_D s'est arrêté sur 'where'. Les syntagmes nominaux sont délimités par les signes '[' et ']' dans le texte où se trouve l'erreur.

```
-----> Automate ENOORD_INTERINDI_D déclenche sur le texte
    @> where <@

Dans: >> [ he ] asked where was [ I ] going .
```

(5) z t

La commande 'z t' affiche les traits calculés lors du pré-traitement. Dans notre cas, les deux pronoms possèdent le trait HEADSN car ils forment chacun la tête d'un syntagme nominal. Les deux verbes 'asked' et 'going' sont des verbes principaux et possèdent donc les traits de la forme verbale.

```
! Attributs pour mot 'he'
!    HEADSN = *OUI*
! Attributs pour mot 'asked'
!    TNS_FV = PRET
!    ASPECT_FV = SIMP
!    VOIX_FV = ACT
!    STATUT_FV = AFF
! Attributs pour mot 'I'
!    HEADSN = *OUI*
! Attributs pour mot 'going'
!    TNS_FV = PRET
!    ASPECT_FV = CONT
!    VOIX_FV = ACT
!    STATUT_FV = INT
```

(6) z e

La commande 'z e' affiche les traits des syntagmes nominaux tels qu'ils ont été calculés. Le premier SN est 'He' (troisième personne du singulier, le mot est en première position), le second est 'I' (première personne du singulier, le mot est en cinquième position).

```
! Attributs pour SN [0,1] type 0
!    NBR_SN = SING
!    PERS_SN = P3
!    LASTPOS = 1
!    FIRSTPOS = 1
! Attributs pour SN [4,5] type 0
!    NBR_SN = SING
!    PERS_SN = P1
!    LASTPOS = 5
!    FIRSTPOS = 5
```

(7) q

Annexe 6. Logiciel automates: règles

Cette section présente l'agencement et les automates correspondants aux catégories d'erreurs 1 à 5.

a. Agencement

```
PRETRAITEMENT

!
!  SN et SP
!

  (      HEAD_SN_1)

  (      NBR_SN_SING_QUANT          NBR_SN_GEN
         NBR_SN_SING_PRON           NBR_SN_PLUR_PRON )

  (      SP_ON_DEMAND               SP_FOR_EXAMPLE
         SP_ON_THE_CONTRARY         BY_THE_WAY
         OF_COURSE                  IN_FACT
         FOR_INSTANCE               IN_X_OPINION
         SINCE_THEN                 IN_THE_FUTURE
         IN_MY_LIFE )

         SN_TYPE_TEMP

         SN_TYPE_LIEU

  (      SN_TYPE_TEMP_1             SN_TYPE_TEMP_2
         SN_TYPE_TEMP_3             SN_TYPE_TEMP_4
         SN_TYPE_TEMP_5             SN_TYPE_TEMP_6
         SN_TYPE_TEMP_7             SN_TYPE_TEMP_8
         SN_TYPE_TEMP_9  )

  (      SP_TYPE_TEMP_1             SP_TYPE_TEMP_2
         SP_TYPE_TEMP_3             SP_TYPE_TEMP_4
         SP_TYPE_TEMP_5             SP_TYPE_TEMP_6
         SP_TYPE_TEMP_7             SP_TYPE_TEMP_8
         SP_TYPE_TEMP_9             SP_TYPE_TEMP_10
         SP_TYPE_TEMP_11 )

  (      SP_TYPE_LIEU_1 )

  (      END_OF_SN_TEMP             SN_TEMP_OF_SN
         SN_LIEU_OF_SN                     SN_OF_SN
         SN_BETWEEN_SN              SN_PREP_SN_LIEU
         SN_PREP_SN
         DAY_AND_NIGHT              NIGHT_AND_DAY )
```

```
  (     PERS_SN_1_2              PERS_SN_3 )

  (     SN_LASTPOS               SP_LASTPOS
        SN_FIRSTPOS              SP_FIRSTPOS )


!
! formes verbales
!

4 (     FUT_SIMP_PASS            FUT_CONT_ACT
        FUTANT_SIMP_PASS         FUTANT_CONT_ACT
        FUTANT_SIMP_ACT          FUT_SIMP_ACT
        FUT_SIMP_PASS_INT        FUT_CONT_ACT_INT
        FUTANT_SIMP_PASS_INT     FUTANT_CONT_ACT_INT
        FUTANT_SIMP_ACT_INT      FUT_SIMP_ACT_INT
        FUT_SIMP_NEG             FUT_SIMP_AFF

        COND_SIMP_PASS           COND_CONT_ACT
        CONDP_SIMP_PASS          CONDP_CONT_ACT
        CONDP_SIMP_ACT           COND_SIMP_ACT
        COND_SIMP_PASS_INT       COND_CONT_ACT_INT
        CONDP_SIMP_PASS_INT      CONDP_CONT_ACT_INT
        CONDP_SIMP_ACT_INT       COND_SIMP_ACT_INT
        COND_SIMP_NEG            COND_SIMP_AFF

        PC_SIMP_PASS             PC_CONT_ACT
        PC_SIMP_ACT              PC_SIMP_PASS_INT
        PC_CONT_ACT_INT          PC_SIMP_ACT_INT

        PRES_SIMP_PASS           PRES_CONT_PASS
        PRES_CONT_ACT            PRES_SIMP_PASS_INT
        PRES_CONT_PASS_INT       PRES_CONT_ACT_INT
        IMPER_NEG
        PRES_SIMP_ACT_AFF_1      PRES_SIMP_ACT_INT


        PQP_SIMP_PASS            PQP_CONT_ACT
        PQP_SIMP_ACT             PQP_SIMP_PASS_INT
        PQP_CONT_ACT_INT         PQP_SIMP_ACT_INT

        PRET_SIMP_PASS           PRET_CONT_PASS
        PRET_CONT_ACT            PRET_SIMP_PASS_INT
        PRET_CONT_PASS_INT       PRET_CONT_ACT_INT
        PRET_SIMP_ACT_AFF_1      PRET_SIMP_ACT_INT
        PRET_SIMP_AUX_ACT_NE     PRET_SIMP_AUX_ACT_IN
        PRET_SIMP_AUX_ACT

        PRES_SIMP_AUX_ACT_NE     PRES_SIMP_AUX_ACT_IN
        PRES_SIMP_AUX_ACT        PRES_SIMP_ACT_AFF_2

        IMPER_AFF

        PRET_SIMP_ACT_AFF_2
    )
```

```
DETECTION

! 1-ACCORD SUJET-VERBE

520   ( VACC_MANQ_ES_PREP_1  VACC_MANQ_ES_PREP_2 )

REPERAGE
520   ( VACC_MANQ_ES_1_R  VACC_MANQ_ES_2_R
        VACC_MANQ_ES_3_R  VACC_MANQ_ES_4_R )
      ($V=R1) VACC_MANQ_ES_DET
END

520   ( VACC_MANQ_ES_Q_1A  VACC_MANQ_ES_Q_1B  VACC_MANQ_ES_Q_2
        VACC_MANQ_ES_Q_3A  VACC_MANQ_ES_Q_3B )

REPERAGE
520   ( VACC_S_ES_1_R  VACC_S_ES_2_R
        VACC_S_ES_3_R  VACC_S_ES_4_R )
      ($V=R1) VACC_S_ES_DET
END

REPERAGE
520   ( VACC_S_ES_1W_R  VACC_S_ES_2W_R
        VACC_S_ES_3W_R  VACC_S_ES_4W_R )
      ($V=R1) VACC_S_ES_W_DET
END

520   ( VACC_S_ES_AND_1   VACC_S_ES_AND_2     VACC_S_ES_PREP_1
        VACC_S_ES_PREP_2  VACC_S_ES_PREP_1_W  VACC_S_ES_PREP_2_W )

520   ( VACC_S_ES_Q_1A  VACC_S_ES_Q_1B  VACC_S_ES_Q_1C
        VACC_S_ES_Q_2   VACC_S_ES_Q_3A  VACC_S_ES_Q_3B
        VACC_S_ES_Q_3B  VACC_S_ES_Q_4A  VACC_S_ES_Q_4B
        VACC_S_ES_Q_5A  VACC_S_ES_Q_5B )

! 2-CONSTRUCTION DU PASSIF

610   ( VMORPASSIF )

! 3-CHOIX DE LA PREPOSITION DANS LES COMPLEMENTS DE TEMPS

REPERAGE
620   ( ENOPREP_DURING_R )
      ()  ENOPREP_DURING_D
END

620   ( ENOPREP_WHILE    ENOPREP_SINCE_1
        ENOPREP_SINCE_2  ENOPREP_FOR_2  )

! 4-UTILISATION DU TEMPS CONTINU

620   ( VTEMPCONT_1 )

! 5-ORDRE SUJET-VERBE DANS LE DISCOURS INDIRECT

REPERAGE
750   ( ENOORD_INTERINDI_R1  ENOORD_INTERINDI_R2
        ENOORD_INTERINDI_R3 )
```

```
     () ENOORD_INTERINDI_D
END

REPERAGE
750  | ENOORD_INTERINDI_W1  ENOORD_INTERINDI_W2
       ENOORD_INTERINDI_W3 |
     () ENOORD_INTERINDI_I
     ($Q=R1) ENOORD_INTERINDI_Q1
     ($Q=R2) MERCI
END

END
```

b. automates de détection

```
!
! 1-ACCORD SUJET-VERBE
!
! La liste V_3P contient les formes canoniques de verbes n'ayant
! de -s à la 3e personne du singulier au présent.
!

! -------- MANQ --------

! premier cas, SN+V, avec le SN qui n'est pas précédé de 'and'
! ou 'of'

AUTOMATE VACC_MANQ_ES_1_R
  (R1=>$V)
    [FC/='and', FC/='nor', CAT/=PREP, CAT/=V]
    <SN (PERS_SN=P3, NBR_SN=SING)>
    @[CAT=V, STATUT_FV/=INT, (TNS=PRES | TNS=PRET),
          ~V_3P, ~V_HOMO_3S, MORPH/=M_3S, FF=>$A]

! début de la phrase

AUTOMATE VACC_MANQ_ES_2_R
  (R1=>$V)
    <SN @[EPOS=EFIR, POS=FIR] (PERS_SN=P3, NBR_SN=SING)>
    @[CAT=V, (TNS=PRES | TNS=PRET), MORPH/=M_3S, ~V_3P, ~V_HOMO_3S,
         FF=>$A]

! SN wh V

AUTOMATE VACC_MANQ_ES_3_R
  (R1=>$V)
    [FC/='and', FC/='nor', CAT/=PREP]
    <SN (PERS_SN=P3, NBR_SN=SING)>
    [FC='which' | FC='that' | FC='who']

    @[CAT=V, STATUT_FV/=INT, (TNS=PRES | TNS=PRET),
        ~V_3P, ~V_HOMO_3S, MORPH/=M_3S, FF=>$A]
```

```
AUTOMATE VACC_MANQ_ES_4_R
   (R1=>$V)
     <SN @[EPOS=EFIR, POS=FIR] (PERS_SN=P3, NBR_SN=SING)>
     [FC='which' | FC='that' | FC='who']
     @[CAT=V, STATUT_FV/=INT, (TNS=PRES | TNS=PRET),
         ~V_3P, ~V_HOMO_3S, MORPH/=M_3S, FF=>$A]

! PREP

AUTOMATE VACC_MANQ_ES_PREP_1
   (R1=>$V)
     (+EXCLUDE)
     [FC/='and', FC/='nor', CAT/=V, CAT/=PREP]
     <SN (PERS_SN=P3, NBR_SN=SING, FIRSTPOS=>$X, LASTPOS=>$Y)>
     [CAT=PREP]
     <SN>
     @[CAT=V, STATUT_FV/=INT, (TNS=PRES | TNS=PRET),
           ~V_3P, ~V_HOMO_3S, MORPH/=M_3S, FF=>$A]
     ^[FF/='?', POS=LAS]

AUTOMATE VACC_MANQ_ES_PREP_2
   (R1=>$V)
     (+EXCLUDE)
     <SN @[EPOS=EFIR, POS=FIR]
           (PERS_SN=P3, NBR_SN=SING, FIRSTPOS=>$X, LASTPOS=>$Y)>
     [CAT=PREP]
     <SN>
     @[CAT=V, STATUT_FV/=INT, (TNS=PRES | TNS=PRET),
           ~V_3P, ~V_HOMO_3S, MORPH/=M_3S, FF=>$A]
      ^[FF/='?', POS=LAS]

!
! Questions.
!
! Trois structures grammaticales sont recherchées:
!
!  1- SN-wh + modal + SN2 + V     (how did you sleep ?)
!                                 (which one did he see ?)
!         l'accord se fait entre SN2 et modal
!         l'accord ne se fait pas entre SN2 et V
!
!  2- SN-wh + non-modal           (which one comes at midnight ?)
!
!         l'accord se fait entre SN-wh et non-modal
!
!  3- modal + SN + V             (did you see the light ?)
!
!         l'accord se fait entre SN et modal
!         l'accord ne se fait pas entre SN et V
!
!  Pour les cas 1 et 3, les types de SN supportés sont:
!
!         SN simple
!         SN prep SN
!         SN 'and' SN
!
```

```
AUTOMATE VACC_MANQ_ES_Q_1A
  <SN @[CURPOS=1, (+WH_PRON | FC='how' | FC='where')] >
 @[CAT=V , (+AUX | +MODAUX | FC='do'),
       (TNS=PRES | TNS=PRET), ~V_3P, ~V_HOMO_3S,
         MORPH/=M_3S, FF=>$A ]
  [CAT=ADV]*
  <SN (PERS_SN=P3, NBR_SN=SING) >
  [CAT=ADV]*
  [CAT=V]

AUTOMATE VACC_MANQ_ES_Q_1B
  <SN @[CURPOS=1, (+WH_PRON | FC='how' | FC='where')] >
 @[CAT=V , (+AUX | +MODAUX | FC='do'),
       (TNS=PRES | TNS=PRET), ~V_3P, ~V_HOMO_3S,
        MORPH/=M_3S, FF=>$A ]
  <SN (PERS_SN=P3, NBR_SN=SING) >
  [CAT=ADV]*
  [CAT=PREP]
  <SN>
  [CAT=ADV]*
  [CAT=V]

AUTOMATE VACC_MANQ_ES_Q_2
  <SN @[POS=FIR, EPOS=EFIR, (+WH_PRON | FC='how' | FC='where') ]
        ^[CAT=N] (PERS_SN=P3, NBR_SN=SING) >
 @[CAT=V , ~AUX, ~MODAUX, FC/='do',
       (TNS=PRES | TNS=PRET), ~V_3P, ~V_HOMO_3S,
        MORPH/=M_3S, FF=>$A ]

AUTOMATE VACC_MANQ_ES_Q_3A
 @[CAT=V, CURPOS=1, (+AUX | +MODAUX | FC='do') ,
       (TNS=PRES | TNS=PRET), ~V_3P, ~V_HOMO_3S,
        MORPH/=M_3S, FF=>$A ]
  <SN (PERS_SN=P3, NBR_SN=SING) >
  [CAT=ADV]*
  [CAT=V]

AUTOMATE VACC_MANQ_ES_Q_3B
 @[CAT=V, CURPOS=1, (+AUX | +MODAUX | FC='do') ,
       (TNS=PRES | TNS=PRET), ~V_3P, ~V_HOMO_3S,
        MORPH/=M_3S, FF=>$A ]
  <SN (PERS_SN=P3, NBR_SN=SING) >
  [CAT=PREP]
  <SN>
  [CAT=ADV]*
  [CAT=V]


AUTOMATE VACC_S_ES_1_R
  (R1=>$V)
    [FC/='and', CAT/=PREP, CAT/=V]
    <SN (PERS_SN/=P3 | NBR_SN=PLUR)>
    @[CAT=V, STATUT_FV/=INT, TNS=PRES, ~V_3P, MORPH=M_3S, FF=>$A]

AUTOMATE VACC_S_ES_1W_R
  (R1=>$V)
    [FC/='and', CAT/=PREP, CAT/=V]
    <SN (PERS_SN=P3, NBR_SN=PLUR)>
```

```
    @[CAT=V, FF='was', FF=>$A]

! deuxième cas, SN+V au début de la phrase


AUTOMATE VACC_S_ES_2_R
  (R1=>$V)
    <SN @[EPOS=EFIR, POS=FIR]  (PERS_SN/=P3 | NBR_SN=PLUR)>
    @[CAT=V, TNS=PRES, ~V_3P, MORPH=M_3S, FF=>$A]

AUTOMATE VACC_S_ES_2W_R
  (R1=>$V)
    <SN @[EPOS=EFIR, POS=FIR]  (PERS_SN=P3, NBR_SN=PLUR)>
    @[CAT=V, FF='was', FF=>$A]

! troisième cas, SN+(which,that,who)+V

AUTOMATE VACC_S_ES_3_R
  (R1=>$V)
    [FC/='and', CAT/=PREP]
    <SN (PERS_SN/=P3 | NBR_SN=PLUR)>
    [FC='which' | FC='that' | FC='who']
    @[CAT=V, STATUT_FV/=INT, TNS=PRES, ~V_3P, MORPH=M_3S, FF=>$A]

AUTOMATE VACC_S_ES_3W_R
  (R1=>$V)
    [FC/='and', CAT/=PREP]
    <SN (PERS_SN=P3, NBR_SN=PLUR)>
    [FC='which' | FC='that' | FC='who']
    @[CAT=V, FF='was', FF=>$A]

AUTOMATE VACC_S_ES_4_R
  (R1=>$V)
    <SN @[EPOS=EFIR, POS=FIR] (PERS_SN/=P3 | NBR_SN=PLUR)>
    [FC='which' | FC='that' | FC='who']
    @[CAT=V, STATUT_FV/=INT, TNS=PRES, ~V_3P, MORPH=M_3S, FF=>$A]

AUTOMATE VACC_S_ES_4W_R
  (R1=>$V)
    <SN @[EPOS=EFIR, POS=FIR] (PERS_SN=P3, NBR_SN=PLUR)>
    [FC='which' | FC='that' | FC='who']
    @[CAT=V, FF='was', FF=>$A]


! quatrième cas, SN+of+SN+V

AUTOMATE VACC_S_ES_PREP_1
  (R1=>$V)
    (+EXCLUDE)
    [FC/='and', CAT/=V, CAT/=PREP]
    <SN ((PERS_SN/=P3 | NBR_SN=PLUR), FIRSTPOS=>$X, LASTPOS=>$Y)>
    [CAT=PREP]
    <SN>
    @[CAT=V, STATUT_FV/=INT, TNS=PRES, ~V_3P, MORPH=M_3S, FF=>$A]
    ^[FF/='?', POS=LAS]
```

```
AUTOMATE VACC_S_ES_PREP_2
  (R1=>$V)
    (+EXCLUDE)
    <SN @[EPOS=EFIR, POS=FIR]
         ((PERS_SN/=P3 | NBR_SN=PLUR), FIRSTPOS=>$X, LASTPOS=>$Y)>
    [CAT=PREP]
    <SN>
    @[CAT=V, STATUT_FV/=INT, TNS=PRES, ~V_3P, MORPH=M_3S, FF=>$A]
    ^[FF/='?', POS=LAS]

AUTOMATE VACC_S_ES_PREP_1_W
  (R1=>$V)
    (+EXCLUDE)
    [FC/='and', CAT/=V, CAT/=PREP]
    <SN (PERS_SN=P3, NBR_SN=PLUR, FIRSTPOS=>$X, LASTPOS=>$Y)>
    [CAT=PREP]
    <SN>
    @[CAT=V, FF='was', FF=>$A]
     ^[FF/='?', POS=LAS]

AUTOMATE VACC_S_ES_PREP_2_W
  (R1=>$V)
    (+EXCLUDE)
    <SN @[CURPOS=1] (PERS_SN=P3, NBR_SN=PLUR,
          FIRSTPOS=>$X, LASTPOS=>$Y)>
    [CAT=PREP]
    <SN>
    @[CAT=V, FF='was', FF=>$A]
     ^[FF/='?', POS=LAS]

AUTOMATE VACC_S_ES_AND_1
    (+EXCLUDE)
    [FC/='and', CAT/=V, CAT/=PREP]
    <SN (FIRSTPOS=>$X)>
    [FC='and']
    <SN (LASTPOS=>$Y)>
    @[CAT=V, STATUT_FV/=INT, TNS=PRES, ~V_3P, ~V_HOMO_3S,
        MORPH=M_3S, FF=>$A]
     ^[FF/='?', POS=LAS]

AUTOMATE VACC_S_ES_AND_2
    (+EXCLUDE)
    <SN @[EPOS=EFIR, POS=FIR] (FIRSTPOS=>$X)>
    [FC='and']
    <SN (LASTPOS=>$Y)>
    @[CAT=V, STATUT_FV/=INT, TNS=PRES, ~V_3P, ~V_HOMO_3S,
        MORPH=M_3S, FF=>$A]
     ^[FF/='?', POS=LAS]

! QUESTIONS

AUTOMATE VACC_S_ES_Q_1A
  <SN @|CURPOS=1, (+WH_PRON | FC='how' | FC='where')|
       (PERS_SN/=P3 | NBR_SN=PLUR) >
 @[CAT=V , (+AUX | +MODAUX | FC='do'),
       TNS=PRES, ~V_3P, MORPH=M_3S, FF=>$A]
  <SN (PERS_SN/=P3 | NBR_SN=PLUR) >
  [CAT=ADV]*
```

```
  [CAT=V]

AUTOMATE VACC_S_ES_Q_1B
  <SN @[CURPOS=1, (+WH_PRON | FC='how' | FC='where')] >
 @[CAT=V , (+AUX | +MODAUX | FC='do'),
        TNS=PRES, ~V_3P, MORPH=M_3S, FF=>$A]
  <SN (PERS_SN/=P3 | NBR_SN=PLUR) >
  [CAT=PREP]
  <SN>
  [CAT=ADV]*
  [CAT=V]

AUTOMATE VACC_S_ES_Q_1C
  <SN @[CURPOS=1, (+WH_PRON | FC='how' | FC='where')] >
 @[CAT=V , (+AUX | +MODAUX | FC='do'),
        TNS=PRES, ~V_3P, MORPH=M_3S, FF=>$A]
  <SN>
  [FF='and']
  <SN>
  [CAT=ADV]*
  [CAT=V]

! how many.

AUTOMATE VACC_S_ES_Q_2
  <SN @[CURPOS=1, +WH_PRON] ^[CAT=N] (PERS_SN/=P3 | NBR_SN=PLUR) >
 @[CAT=V , -AUX, +MODAUX, FC/='do',
        TNS=PRES, ~V_3P, MORPH=M_3S, FF=>$A]

AUTOMATE VACC_S_ES_Q_3A
 @[CAT=V, CURPOS=1, (+AUX | +MODAUX | FC='do'),
        TNS=PRES, ~V_3P, MORPH=M_3S, FF=>$A]
  <SN (PERS_SN/=P3 | NBR_SN=PLUR) >
  [CAT=ADV]*
  [CAT=V]

AUTOMATE VACC_S_ES_Q_3B
 @[CAT=V, CURPOS=1, (+AUX | +MODAUX | FC='do'),
        TNS=PRES, ~V_3P, MORPH=M_3S, FF=>$A]
  <SN (PERS_SN/=P3 | NBR_SN=PLUR) >
  [CAT=PREP]
  <SN>
  [CAT=ADV]*
  [CAT=V]

AUTOMATE VACC_S_ES_Q_3C
 @[CAT=V, CURPOS=1, (+AUX | +MODAUX | FC='do'),
        TNS=PRES, ~V_3P, MORPH=M_3S, FF=>$A]
  <SN>
  [FC='and']
  <SN>
  [CAT=ADV]*
  [CAT=V]

AUTOMATE VACC_S_ES_Q_4A
  <SN @[CURPOS=1, (+WH_PRON | FC='how' | FC='where')]
       (PERS_SN/=P3 | NBR_SN=PLUR) >
 @[CAT=V , (+AUX | +MODAUX | FC='do')]
```

```
  <SN (PERS_SN/=P3 ( NBR_SN=PLUR) >
  [CAT=ADV]*
  [CAT=V, TNS=PRES, ~V_3P, MORPH=M_3S, FF=>$A]

AUTOMATE VACC_S_ES_Q_4B
  <SN @[CURPOS=1, (+WH_PRON | FC='how' | FC='where')] >
 @[CAT=V , (+AUX | +MODAUX | FC='do')]
  <SN>
  [CAT=PREP | FF='and']
  <SN>
  [CAT=ADV]*
  [CAT=V, TNS=PRES, ~V_3P, MORPH=M_3S, FF=>$A]

AUTOMATE VACC_S_ES_Q_5A
 @[CAT=V, CURPOS=1, (+AUX | +MODAUX | FC='do')]
  <SN (PERS_SN/=P3 | NBR_SN=PLUR) >
  [CAT=ADV]*
  [CAT=V, TNS=PRES, ~V_3P, MORPH=M_3S, FF=>$A]

AUTOMATE VACC_S_ES_Q_5B
 @[CAT=V, CURPOS=1, (+AUX | +MODAUX | FC='do')]
  <SN (PERS_SN/=P3 | NBR_SN=PLUR) >
  [CAT=PREP]
  <SN>
  [CAT=ADV]*
  [CAT=V, TNS=PRES, ~V_3P, MORPH=M_3S, FF=>$A]

! ==== automates qui affichent les messages ====

AUTOMATE VACC_S_ES_W_DET
   (+EXCLUDE)
     @[CAT=V]
     ^[FF/='?', POS=LAS]
     {
       SCHEMA 1
       MESSAGE "Le verbe \u\$A\v est à la mauvaise personne. "
        "Nous vous suggérons de le remplacer par 'were'"
     }

AUTOMATE VACC_S_ES_DET
   (+EXCLUDE)
     @[CAT=V]
     ^[FF/='?', POS=LAS]
     {


       SCHEMA 1
       MESSAGE "Le verbe \u\$A\v est à la mauvaise personne. "
        "Nous vous suggérons de le mettre à la forme sans \u-s\v."
     }

AUTOMATE VACC_MANQ_ES_DET
   (+EXCLUDE)
     @[CAT=V]
     ^[FF/='?', POS=LAS]
     {
       SCHEMA 1
       MESSAGE "Le verbe \u\$A\v semble être à la "
```

```
      "mauvaise personne. "
      "Nous vous suggérons de le mettre à la troisième"
      "personne du singulier."
    }

! ------------------------------------------------------------
!
! 2-CONSTRUCTION DU PASSIF
!

AUTOMATE VMORPASSIF
  @[CAT=V, FC='be', CURPOS=>$L]
    [FF='not']*1
    [CAT=ADV, FF/='not', FC/='to']*1
    [CAT=V, TNS=BASE, ~V_PARTBASE, ~V_HOMO_3S, CURPOS=>$R, FF=>$A]
    [
      SCHEMA 1
      MESSAGE "La forme passive du verbe \u\$A\v est peut-être "
              "mal formée "
       "dans la séquence \u\($L-$R)\v. Nous vous suggérons de la "
       "remplacer par le participe passé."
    }


! ------------------------------------------------------------
!
! 3-CHOIX DE LA PREPOSITION DANS LES COMPLEMENTS DE TEMPS
!

AUTOMATE ENOPREP_DURING_R
    @[FF='during', CURPOS=>$L, CURPOS=>$R]
       [FF='almost' | FF='nearly' | FF='about']*1
       <SN @[+HEADSN, +TEMP, NBR=PLUR]>

AUTOMATE ENOPREP_DURING_D
    @[]
      [FF='almost' | FF='nearly' | FF='about']*1
      <SN @[EPOS=EFIR, FF/='the', FF/='these', FF/='those',
          FF/='this', FF/='that']>
  {
    SCHEMA 2
    MESSAGE "La préposition \uduring\v ne semble pas correcte"
        "dans ce contexte. "
        "Nous vous suggérons de la remplacer par \ufor\v."
    LISTE "for"
  ]

AUTOMATE ENOPREP_WHILE
   @[FF='while', CURPOS=>$L, CURPOS=>$R]
    <SN @[+HEADSN, +TEMP, NBR=PLUR]>
  [
    SCHEMA 2
    MESSAGE "La préposition \uwhile\v ne semble pas correcte"
        "dans ce contexte. "
        "Nous vous suggérons de la remplacer par \ufor\v"
        "ou \uduring\v."
    LISTE "for; during"
  }
```

```
AUTOMATE ENOPREP_SINCE_1
  @[FF='since', CURPOS=>$L, CURPOS=>$R]
   <SN @[+HEADSN, +TEMP, NBR=PLUR]
        (TEMP_TYPE/=DATE, TEMP_TYPE/=TEMPS_PRECIS) >
  {
    SCHEMA 2
    MESSAGE "La préposition \usince\v ne semble pas correcte"
        "dans ce contexte. "
        "Nous vous suggérons de la remplacer par \ufor\v."
    LISTE "for"
  }

AUTOMATE ENOPREP_SINCE_2
  @[FF='since', CURPOS=>$L, CURPOS=>$R]
    <SN @[EPOS=EFIR, [FC='a' | FC='an']]
        (TEMP_TYPE/=DATE, TEMP_TYPE/=TEMPS_PRECIS) >
  {
    SCHEMA 2
    MESSAGE "La préposition \usince\v ne semble pas correcte"
        " dans ce contexte. "
        "Nous vous suggérons de la remplacer par \ufor\v."
    LISTE "for"
  }

AUTOMATE ENOPREP_FOR_1
  @[FF='for', CURPOS=>$L, CURPOS=>$R]
   [FF='almost' | FF='nearly' | FF='about']*1
   <SN [SN_TYPE=SN_TEMP, [TEMP_TYPE=DATE |
TEMP_TYPE=TEMPS_PRECIS])>
  {
    SCHEMA 2
    MESSAGE "La préposition \ufor\v ne semble pas correcte"
        "dans ce contexte. "
        "Nous vous suggérons de la remplacer par \usince\v."
    LISTE "since"
  }

AUTOMATE ENOPREP_FOR_2
  @[FF='for', CURPOS=>$L, CURPOS=>$R]
   <SN (SN_TYPE=SN_TEMP)
        @[EPOS=EFIR, (FF='this' | FF='these' | FF='that')]
        ^[+HEADSN, +TEMP] >
  {
    SCHEMA 2
    MESSAGE "La préposition \ufor\v ne semble pas correcte"
        "dans ce contexte. "
        "Nous vous suggérons de la remplacer par \uduring\v."
    LISTE "during"
  }


! ------------------------------------------------------------
!
! 4-UTILISATION DU TEMPS CONTINU
!
```

```
AUTOMATE VTEMPCONT_1
  @[CAT=V, ASPECT_FV=CONT, +V_NEVERCONT, FC=>$A]
  {
    SCHEMA 1
    MESSAGE "Le verbe \u\$A\v ne se met généralement pas à"
            "la forme continue. "
            "Nous vous suggérons de le mettre à la forme simple."
  }

! ------------------------------------------------------------
!
!  5-ORDRE SUJET-VERBE DANS LE DISCOURS INDIRECT
!

AUTOMATE ENOORD_INTERINDI_R1        ! .....pour une FV d'un seul mot
  @[(CAT=PRON, +WH_PRON, FC/='which', FC/='who',
            FC/='that', FC/='what')
          | (FC='how') | (FC='where'), POS/=FIR]
   [CAT=V, CURPOS=>$L, CURPOS=>$R]
   <SN (FIRSTPOS=>$X, LASTPOS=>$Y)>
   ^[FF/='?', POS=LAS]

AUTOMATE ENOORD_INTERINDI_R2      !pour une FV de deux à trois mots
  @[(CAT=PRON, +WH_PRON, FC/='which', FC/='who',
            FC/='that', FC/='what')
          | (FC='how') | (FC='where'), POS/=FIR]
   [CAT=V, CURPOS=>$L]
   [FF='not']*1
   [CAT=V, CURPOS=>$R]
   <SN (FIRSTPOS=>$X, LASTPOS=>$Y) >
   ^[FF/='?', POS=LAS]

AUTOMATE ENOORD_INTERINDI_R3  ! pour une FV de trois à quatre mots
  @[(CAT=PRON, +WH_PRON, FC/='which', FC/='who',
            FC/='that', FC/='what')
          | (FC='how') | (FC='where'), POS/=FIR]
   [CAT=V, CURPOS=>$L]
   [FF='not']*1
   [CAT=V]
   [CAT=V, CURPOS=>$R]
   <SN (FIRSTPOS=>$X, LASTPOS=>$Y)>
   ^[FF/='?', POS=LAS]

AUTOMATE ENOORD_INTERINDI_D
  @[]
  {
    SCHEMA 5
    MESSAGE "La séquence \u\($L-$R) \($X-$Y)\v semble faire"
    " partie d'une question indirecte, comme dans:\n"
    "\uWe do not know when you are leaving.\v\n"
    "Si c'est le cas, nous vous conseillons de permuter"
    " \u\($L-$R)\v et \u\($X-$Y)\v."
  }

! La série d'automates ci-dessous traite du cas 'what', qui ne
! peut en principe pas être suivi directement par un verbe.
```

```
AUTOMATE ENOORD_INTERINDI_W1
  @[FC='what', POS/=FIR]
   [CAT=V, CURPOS=>$L, CURPOS=>$R, FF=>$A]
   <SN (FIRSTPOS=>$X, LASTPOS=>$Y)>
   ^[FF/='?', POS=LAS]

AUTOMATE ENOORD_INTERINDI_W2   !...pour une FV de deux à trois mots
  @[FC='what', POS/=FIR]
   [CAT=V, CURPOS=>$L]
   [FF='not']*1
   [CAT=V, CURPOS=>$R]
   <SN (FIRSTPOS=>$X, LASTPOS=>$Y) >
   ^[FF/='?', POS=LAS]

AUTOMATE ENOORD_INTERINDI_W3    ! ...pour une FV de trois à quatre
mots
  @[FC='what', POS/=FIR]
   [CAT=V, CURPOS=>$L]
   [FF='not']*1
   [CAT=V]
   [CAT=V, CURPOS=>$R]
   <SN (FIRSTPOS=>$X, LASTPOS=>$Y)>
   ^[FF/='?', POS=LAS]

AUTOMATE ENOORD_INTERINDI_I
  @[FC='what']
  {
    SCHEMA 6
    MESSAGE "Est-ce que dans cette phrase \u\($X-$Y)\v "
            "est le sujet du verbe \u\($L-$R)\v?"
  }

AUTOMATE ENOORD_INTERINDI_Q1
  @[FC='what']
  {
    SCHEMA 1
    MESSAGE "Dans ce cas nous vous suggérons de permuter "
            "\u\($X-$Y)\v et \u\($L-$R)\v?"

  }
```

Annexe 7. Logiciel automates: instructions pour construire les phrases-test

1 - Format général des phrases

Le logiciel ARCTA ne possède en principe pas de limites quant aux mots et aux structures de phrases qu'on peut lui soumettre. Il faut toutefois tenir compte du fait que vos phrases-test seront utilisées pour évaluer les aspects syntaxiques et morpho-syntaxiques de la détection d'erreurs en langue seconde. Vous devriez donc éviter les difficultés sans rapport avec cette fonction, comme dans la phrase ci-dessous:

* The Compaq 486/33 Mhz PCs contains 4 Mbytes of RAM.

qui contient des syntagmes nominaux au format complexe (The Compac 486/33 Mhz PCs', '4 Mbytes of RAM') ou des noms propres et des acronymes. Dans des cas comme ceux-là, le logiciel ne parviendra probablement pas à identifier l'erreur d'accord du verbe 'contain'; il est également possible que le logiciel détecte par erreur d'autres problèmes dans cette phrase.

2 - Liste des erreurs

Les pages qui suivent représentent des fiches de travail pour chaque catégorie d'erreurs. Chaque fiche contient une description du type d'erreur ainsi que les contraintes qui s'appliquent lors du choix des phrases. Assurez-vous de bien respecter ces contraintes.

Pour chaque erreur on trouvera des exemples de cas traités par le logiciel ainsi que de cas non traités. Etant donné qu'une liste exhaustive de ces derniers prendrait trop de place, ou est simplement impossible à constituer, référez-vous toujours aux contraintes décrites dans l'appendice A en cas de doutes.

CATEGORIE 1

Accord sujet-verbe

Description

Accord du verbe avec le sujet.

Cas traités

Phrases régulières

* *The children likes to eat chocolate.*

Cas non traités

- La forme infinitive ou impérative des verbes.

* *I saw the man steals the watch.*

Remarques

Mettez le verbe à la mauvaise personne ou au mauvais nombre, mais ne modifiez pas l'orthographe pour qu'il ne se trouve plus dans la table de conjugaison.

CATEGORIE 2

Construction du passif

Description

Mauvais temps du verbe principal dans la forme passive.

Cas traités

- Verbe principal ne se trouvant pas au participe.

  * *The lost tape was discover under the bench.*

Cas non traités

- Aucun.

CATEGORIE 3

Préposition dans les compléments de temps

Description

Mauvais choix de la préposition dans les compléments de temps.

Cas traités

- Confusion entre 'for', 'during' et 'since'.

  * *He remained there during about twenty hours.*

Cas non traités

- Lorsque le choix de la préposition dépend du verbe principal.

  * *He lived in Paris since the month of October.*

Remarque:

Ne considérer que les cas où le contenu du complément de temps détermine de manière univoque la préposition correcte.

CATEGORIE 4

Utilisation du temps continu

Description

Utilisation d'un temps continu pour les verbes ne le permettant pas.

Cas traités

- Verbes ne se mettant en principe pas aux temps continus.

  * *He is seeming quite upset these days.*

Cas non traités

- Lorsque le temps dépend du contexte pragmatique.

  * *A machine is only doing what it is told to do.*

CATEGORIE 5

Ordre sujet-verbe dans le discours indirect

Description

Inversion de l'ordre sujet-verbe dans le discours indirect.

Cas traités

- Adverbes 'when', 'how', 'where', 'what' et 'why'.
- Discours indirect lié.

  * *He asked me when had the incident occured.*

Cas non traités

- Pronoms interrogatifs qui sont aussi des pronoms relatifs, comme 'which', 'who'.

  * *He asked me who did I see last night.*

- Discours indirect libre.

  * *She asked me: "you are going to the movie?".*

CATEGORIE 6

Forme du verbe dans les propositions infinitives

Description

Mauvaise construction des formes verbales dans les propositions infinitives.

Cas traités

- 'that' au lieu de l'infinitif pour les verbes 'to want', 'to like' et 'to love'.

  * *They wanted <u>that</u> we came earlier.*

- Oubli de 'to'.

  * *It all seemed__be under control.*

- Gérondif au lieu de l'infinitif.

  * *I would like <u>going</u> to the movies tonight.*

Cas non traités

- Tous les autres cas.

<u>CATEGORIE 7</u>

Accord du nom

Description

Mauvais accord du nom après certains déterminants.

Cas traités

- Accord en nombre avec le déterminant, en particulier avec 'every', 'each', 'few', 'many', 'this', 'these', 'all'.

  * *<u>Every players</u> has agreed to come that day.*

Cas non traités

- Aucun.

CATEGORIE 8

Temps des verbes avec des compléments temporels.

Description

Mauvais temps du verbe ayant un complément de temps.

Cas traités

- Prépositions 'since' et 'for'.

  * *I live here for two years.*

Cas non traités

- Temps du verbe à l'intérieur du complément de temps.

  * *I have been living here since I am born.*

CATEGORIE 9

Position des adverbes par rapport aux verbes

Description

Mauvais positionnement des adverbes par rapport aux verbes.

Cas traités

- Avant/après le verbe.

  * *I go always to the zoo on sunny weekends.*

Cas non traités

- Locutions adverbiales.

  * *I last night went to the cinema.*