National Library of Canada
Canadian Theses Service
Ottawa, Canada
K1A 0N4

Bibliothèque nationale du Canada
Service des thèses canadiennes

NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.

Canada

NL-339 (r. 88/04) c

Permission has been granted to the National Library of Canada to microfilm this thesis and to lend or sell copies of the film.

The author (copyright owner) has reserved other publication rights, and neither the thesis nor extensive extracts from it may be printed or otherwise reproduced without his/her written permission.

L'autorisation a été accordée à la Bibliothèque nationale du Canada de microfilmer cette thèse et de prêter ou de vendre des exemplaires du film.

L'auteur (titulaire du droit d'auteur) se réserve les autres droits de publication; ni la thèse ni de longs extraits de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation écrite.

ISBN 0-315-56433-4

ORDINOTRAD:
A Machine Translation System
based on Case Grammar

THESIS
submitted to the
School of Graduate Studies and Research
of the
University of Ottawa

---

in partial fulfilment of the requirements
for the degree of
MASTER OF ARTS IN TRANSLATION

---

by
Victor Loewen
under the supervision of
Professor Brian Harris
&
Dr. Stanislaw Szpakowicz

School of Translators and Interpreters
University of Ottawa
1988

© Victor Loewen, Ottawa, Canada, 1989

UNIVERSITÉ D'OTTAWA
UNIVERSITY OF OTTAWA

Table of Contents

LIST OF COMMONLY USED ABBREVIATIONS

AI	Artificial Intelligence
CG	Case Grammar
M	Modality
MT	Machine Translation
NLP	Natural-Language Processing
NP	Noun Phrase
SL	Source language
TL	Target language
V	Verb
VP	Verb Phrase

In the program code, abbreviations can usually be determined from the context or from comments pertaining to a particular rule.

ABSTRACT

This thesis describes the design and implementation of a machine-translation system based on Fillmore and others' case grammar model whose purpose is to generate a representation of the deep syntactic and semantic structure of a sentence through the use of labelled relations. The system described here contains a Prolog implementation of a prototype machine-translation system which generates English translations of French sentences. The passage from French text to English text passes through morphological and syntactic analyses, case grammar (semantic) analysis, transfer of morphological, syntactic and semantic information to English-language equivalents, and, finally, syntactic and morphological generation governed by case frames. The system runs on a Macintosh Plus (2MB RAM) using Advanced A.I. Systems Prolog Version M1.15.

Key words: machine translation, case grammar, Prolog, Macintosh, French, English.

La présente thèse décrit la conception et la réalisation d'un système de traduction automatique reposant sur différents modèles de grammaire casuelle, dont celui de Fillmore. Ces modèles visent à établir une représentation de la structure syntaxique et sémantique profonde d'une phrase par le biais de relations portant des étiquettes. Le système décrit ici est un prototype réalisé en Prolog et traduisant des phrases du français à l'anglais. Le processus de traduction se décompose comme suit : analyse morphologique et syntaxique, analyse sémantique (selon la grammaire casuelle), transfert de l'information morphologique, syntaxique et sémantique vers l'anglais, puis génération syntaxique et morphologique à partir de cadres sémantiques casuels. Ce système fonctionne sur un Macintosh Plus (muni de 2 méga-octets de mémoire vive) avec AAIS Prolog version M1.15, un logiciel évolué d'intelligence artificielle.

Mots clés : traduction automatique, grammaire casuelle, Prolog, Macintosh, français, anglais

ACKNOWLEDGEMENTS

I wish to give special recognition to Dr. Stanislaw Szpakowicz, thesis supervisor and "Prologician" par excellence, for his correctly aimed criticisms of the linguistic and programming elements of this work and for his genuine enthusiasm in the project.

I also wish to give special recognition to Professor Brian Harris, thesis supervisor, for bringing machine translation to the School of Translators and Interpreters and for his insightful, helpful comments and criticisms on the content of this thesis.

I thank Dr. J.-P. and Mrs. M. P. Vinay for their kind permission to use the word *Ordinotrad*, taken from "Regards sur l'évolution des théories de la traduction depuis vingt ans," *META* 20.1 (1975): 7-27.

A special "thank-you" to all the professors at ETI/STI who participated in the development of my translation skills and understanding of the theory of translation.

I also thank my wife Lorena who, as a computer widow for nearly two years, eagerly helped me set deadlines and try to meet them, and who, as a living, loving companion, willingly sacrificed a pleasant, prairie life for the sake of academia.

I dedicate this thesis to the glory of God who created me and in whom I live, move and exist.

INTRODUCTION

This thesis discusses the rationale for using the case grammar approach in a machine-translation system, describes the system and its machine implementation, and presents some actual results. Its main purpose is twofold: to confirm the feasibility of implementing case grammar principles and, therefore, semantics in a machine translation system which also relies heavily on syntax; and to demonstrate the feasibility of running a complete but small-scale MT system on a micro-computer. The name of the system is Ordinotrad.

Chapter 1 contains a general description of the design, implementation (how linguistic data is represented and moved about within the system) and results of Ordinotrad. This chapter is intended for both linguistics- and computer-oriented readers. Chapters 2 and 3 describe case grammar, the linguistic "heart" of the system, and present the case labels used in it. These chapters were prepared before programming was begun and have been revised subsequently in light of programming. Chapter 4 is primarily technical and is meant for Macintosh users, providing a practical description of how a user can run the system (as it currently exists), how the data is manipulated (i.e., how data structures are built), and the interdependencies of the various program components. Chapter 5 evaluates what has been accomplished, what is promising, what can be handled, what cannot be handled, and what can be added easily or with difficulty where the system has room for development.

The appendices contain (1) a summary description of the case list used in this system, (2) the taxonomy used in the system, i.e., the way "something" is viewed in Ordinotrad, (3) sample results including interim data structures, (4) the program itself (material intended primarily for Prolog programmers), and (5) a glossary of key terms.

Hardware requirements to run the system as it is described herein are a Macintosh Plus with two megabytes of RAM and two 800K drives.

Arrangements may be made with the author to obtain the program in electronic form for research purposes only. This offer does not include a copy of AAIS Prolog M1.15 which is needed to run Ordinotrad. A user licence for this software may be obtained for a fee from AAIS, P.O. Box 39-0360, Mountain View, California 94039-0360; (415) 961-1121.

Chapter 1

Ordinotrad

GOALS

Two complementary goals underlie the work reported in this thesis: (1) to create a complete, small-scale machine-translation system on a micro-computer capable of generating French-to-English translations of typical administrative correspondence, and (2) to emphasize the use of case grammar principles in controlling the analysis and generation of text within a natural-language processing system.

It should be noted at the outset that the machine-translation system described herein is neither a first for micro-computers nor entirely original. For example, it would seem that natural language parsers for French abound, in view of the number of large-scale MT systems in use. Even so, obtaining an existing parser is virtually impossible, given the evident reluctance of system developers to publish program code. As a result, the parser contained herein should not be construed as a re-invention but rather as a re-creation based on what researchers and developers generally agree a parser should do.

I also do not claim that case grammar provides the only or even the best linguistic framework to be used in a machine-translation system. To do so would require comparative analyses of other systems using other or no semantic procedures. As a translator, however, I felt that case grammar with its labelled relations was intuitively applicable to understanding the translation process and also that it lent itself particularly well to computer programming. Some existing machine-translation systems show that machine translation in certain contexts need not incorporate semantic considerations. However, I felt that case grammar could be useful in that it could provide a simple, easily manipulated structure for representing linguistic data. Even so, the current version of Ordinotrad does rely greatly on syntactic analysis.

With the above-mentioned twofold aim then, the work was done on a micro-computer and hence within its limitations.[1] More importantly, however, it focussed on both semantic and syntactic controls for the purposes of analyzing French sentences (in order to construct meaning-based syntactic representations of them) and generating English

[1] The work was begun using a 512K Macintosh computer, then moved to a Macintosh Plus, which was subsequently upgraded to 2 megabytes of RAM.

sentences. The acceptability of the translations produced as output (i.e., whether a translator would consider them worth revising) is assumed to be the test of whether or not the analysis and generation together have succeeded.

The relationships between syntax and semantics in automatic text analysis are not examined, although material is provided for such studies as a by-product of the analysis procedures. It is assumed, in this work, that neither a strictly semantic nor a strictly syntactic approach to text analysis is worth pursuing when the goal is translation. This assumption has been made not so much on the basis of researched material, although it is mentioned by Boguraev,[2] as on personal translation experience[3] where I find that both types of information are generally required and that a translation is seldom produced strictly from semantic information and never from syntactic information alone. Moreover, a reading of translation theory textbooks[4] seems to indicate that translation is meaning-based, since it is *textual meaning* or the *message* that is the transferred element in a translation. In Ordinotrad, then, semantic analysis is emphasized, as it is used on the one hand to verify that conventional syntactic labels have been correctly applied to sentence components, and on the other hand to establish the relationships between a predicate (lexicalized as a verb) and its arguments (lexicalized as noun phrases or other sentence components).

The relative sizes of the parser and case analyzer in Ordinotrad seem to contradict the emphasis I am placing on semantics, as the former is much larger than the latter in terms of the number of rules. Nonetheless, it is the position and not the size of the different modules which reflects their importance, for the semantic judgements are made concerning the results of syntactic parsing. Moreover, the relative size of the two modules could indicate that syntax is a complex network of functional relations between sentence components, and that semantics is a simple but powerful network of the meaningful relations between them. In this case the bonus is that what is more powerful is also easier to organize and implement in an automated system.

LIMITS

The machine-translation system described in this work is complete in itself, although quite small-scale. All the elements customarily found in a translation system are there: in terms of procedure, there are input and output interfaces, and modules for

---

[2] Branimir Konstantinov Boguraev, *Automatic Resolution of Linguistic Ambiguities, Technical Report No. 11* (University of Cambridge, Computer Laboratory, August 1979), 0.18-0.19.

[3] Since April 1983, when I began to keep records, I have translated over 610 000 words.

[4] Cf. Larson, Newmark, Nida, Pergnier, etc.

analysis and generation; in terms of databases, there are monolingual dictionaries and a bilingual transfer dictionary.

Even so, Ordinotrad is very limited. The dictionaries cover only a small fragment of the French and English lexicons. Two points need to be made in this regard. First, there is a module that allows Ordinotrad to acquire new French and English words and their corresponding equivalents. This module is quite cumbersome to use, as the user must be familiar with the taxonomy used by the system in order to formulate the intensional meanings of words. As the module for acquiring new words is not directly related to the twofold purpose of this work and as it was used in developing the system only to ensure uniformity in dictionary entries, it is not as optimal as it could be.[5] Second, with regard to the French and English words that are currently defined in the dictionaries, I created the entries as a system developer and not as a lexicographer.[6] A cursory reading of the monolingual dictionaries will reveal shortcomings in the definitions for all words and in the case frames for events. The implication of this for meaning-based translation systems is that lexicography is an integral part of system development, though distinguishable from analysis and generation,[7] and any weakness on the part of the developer in this area would have to be overcome with input from professional lexicographers.

Another limitation of Ordinotrad is the sentence structures the parser is capable of recognizing. The system as it is handles only basic French structures (see the examples in Appendix 3), yet since it is modular, further development of the parser will not require re-programming the system as a whole. For a complete description of the syntactic limitations, see the printout of the "Parser" file in Appendix 4.

The generation module requires more attention, as some word forms are incorrectly constructed. Resolving generation problems would quite likely require a larger system, for a database of verb forms would have to be created for the irregular verbs along with the appropriate rules.

In general then, Ordinotrad's limitations are in terms of "breadth of language" and "complexity of sentence structures." If it is to increase in utility, the dictionaries and parser

---

[5] The assumption here is that the word-acquisition module should be part of a complementary dictionary management system, and not the translation system itself. Therefore, proper development of the module is, in terms of practice, part of a different domain of MT and, in terms of theory, reliant on a linguistic field not touched on in this work, namely lexicography.

[6] These two areas of activity are not necessarily mutually exclusive. It is just that I created definitions to meet my needs.

[7] This distinction is based on the fact that the system dictionaries can be expanded without making any changes to analysis or generation routines.

will have to be expanded, and if it is to improve in quality, the generator will have to be augmented.

RESULTS

The system was designed to translate typical administrative letters, for example, the one provided by Clas-Horguelin in their secretarial handbook on using French in business.[8] A typical letter which the system can translate will include names, dates and other such *mentions*[9] in addition to the sentences in the body of a letter. With regard to sentences, the system was designed to translate a wide range of sentence structures in addition to those contained in the above letter: for example, sentences containing pre-verbal particles (e.g., *il me la donne; il m'en a adressé trois*), or others containing ellipsed sentences (e.g., *nous vous remercions et sommes heureux*).

Appendix 3 contains a copy of the above-mentioned letter and a list of these sentences (reflecting basic sentence structures) along with the translations provided by the system. Interim results, that is, the products of parsing and case analysis, are also provided for some sentences in this letter that reflect different sentence structures.

The goal at first was to achieve an 80 percent rate of successful translations. However, given the relatively few sentences used to test the system[10] and the difficulty of establishing objective yardsticks for evaluating translations, I opted for a simpler, more subjective "success point." Would I as a human translator consider the output worth revising?

---

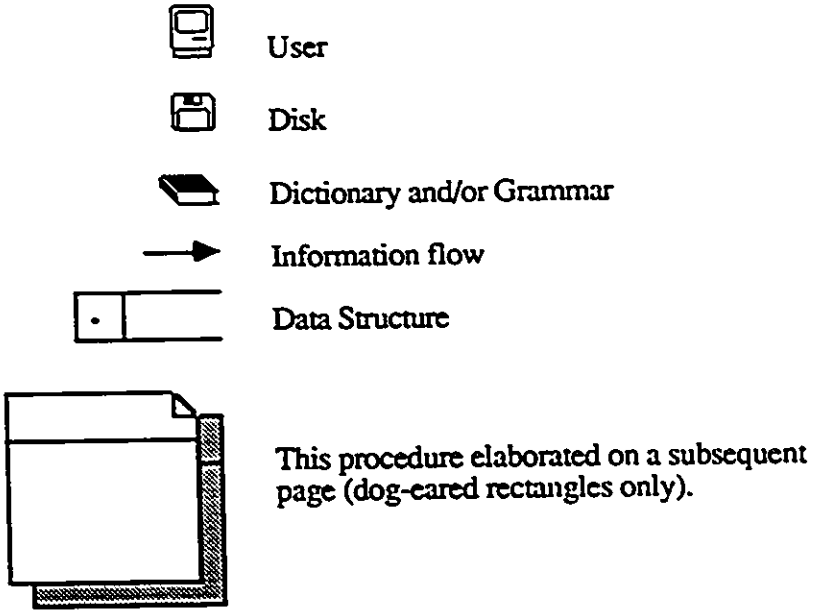
[8] André Clas & Paul A. Horguelin, *Le Français, langue des affaires*, 2nd edition (Montreal: McGraw-Hill, 1979), 244.

[9] A mention is distinguished from a sentence by its lack of central event and modalities. These elements are discussed further in Chapter 2.

[10] Reliable assessment of output would probably require a larger sampling of sentences. Eighty percent success is interpreted to mean fewer than 20 corrections required per 100 words.

SYSTEM OVERVIEW

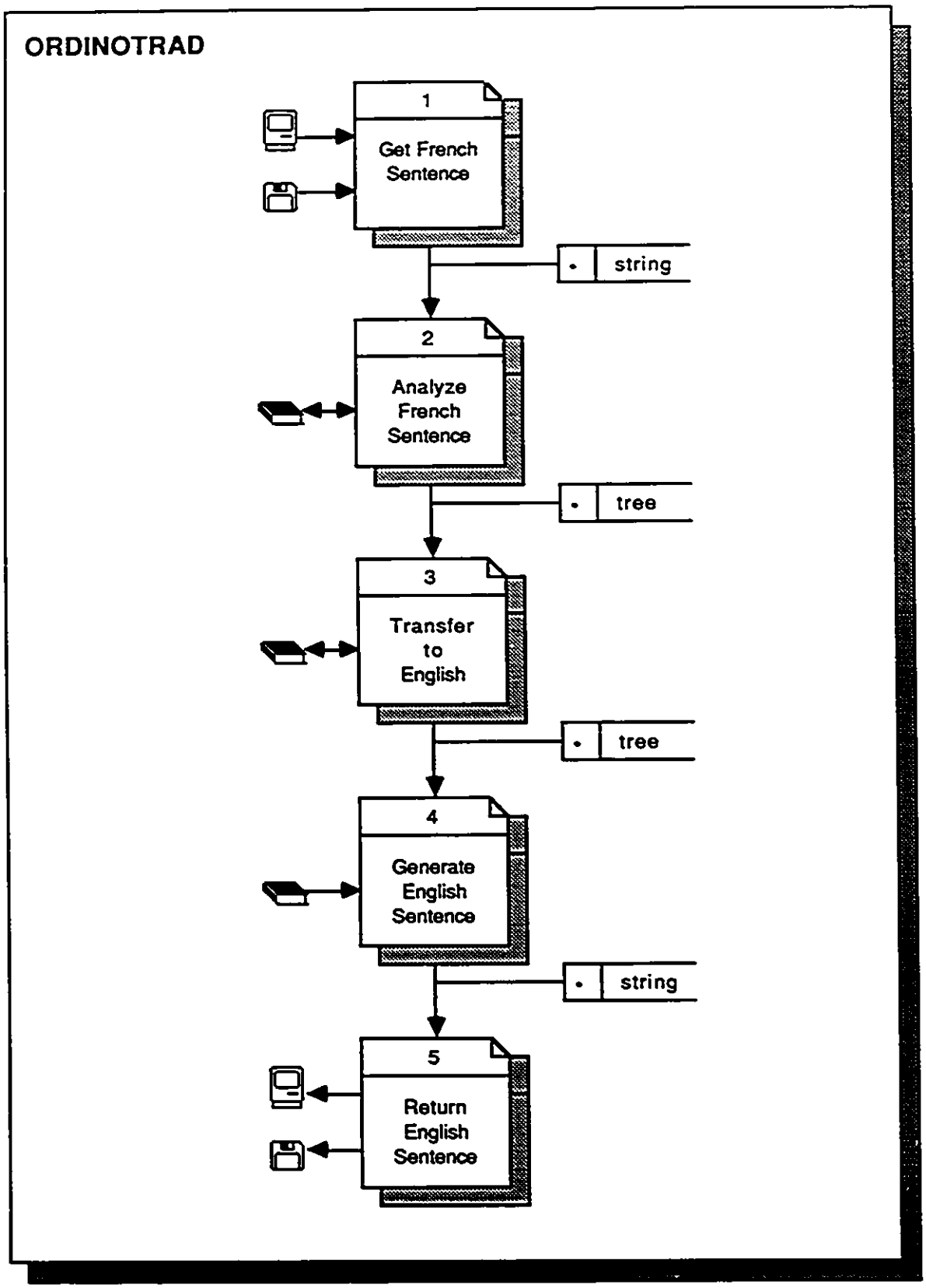
The diagrams on the following pages offer an overview of Ordinotrad. The rectangles represent the various procedures (their program equivalents use roughly the same wording). The meaning of the icons used in these diagrams is as follows:



ORDINOTRAD

The overall translation process is comprised of five procedures, as shown in the following diagram.[11]

11 For a technical description of how to run Ordinotrad, see Chapter 4.


ORDINOTRAD
1
Get French Sentence
string
2
Analyze French Sentence
tree
3
Transfer to English
tree
4
Generate English Sentence
string
5
Return English Sentence

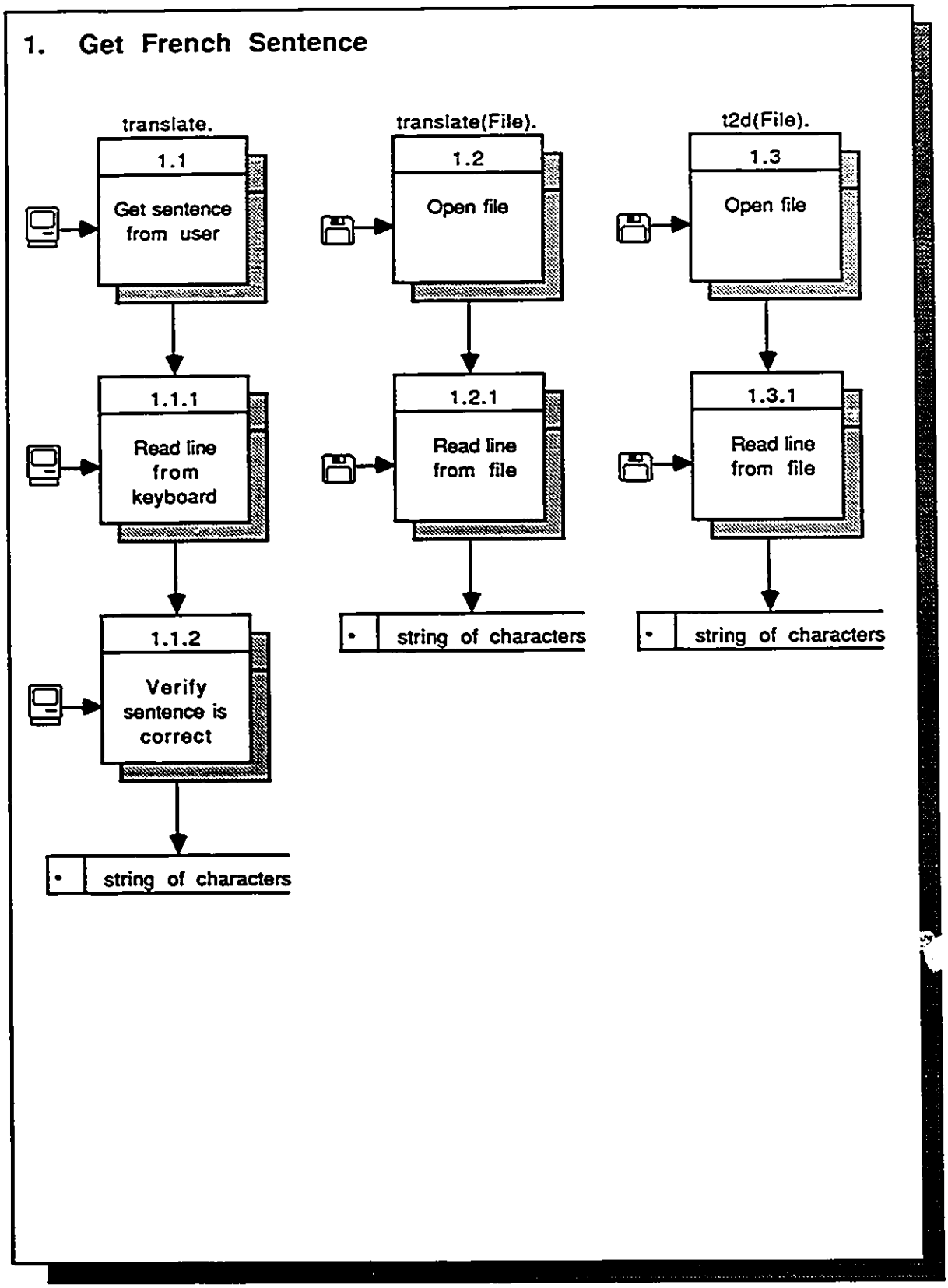
Get French Sentence

There are two possible sources for the original French sentence to be translated: the user who types in a sentence, or files on disk prepared in advance. It should be noted that there are no restrictions on input, which can range from a postal code to a complex sentence, and may or may not conclude with some sort of punctuation. However, input still has to make sense to the parser and case processor if it is to be processable.

The first column on the following page reflects the interactive process where the user provides the input. The other two columns show the procedures if input is taken from a prepared file. The difference between the latter two approaches is that the first one will display results for each sentence and query the user if it should continue with the next sentence, while the second one will display results for the user, record the results in a file on a disk, and continue non-stop with subsequent sentences.

The words at the top of each column are the corresponding Prolog commands the user would enter to have the translation process begun.[12]

12 Throughout the program I have tried to use predicates in Prolog that are easily recognizable. In the case of "t2d", which should be read as "translate to disk", I have opted for an acronym to facilitate entering that command.


1. Get French Sentence
translate.
1.1
Get sentence from user
1.1.1
Read line from keyboard
1.1.2
Verify sentence is correct
string of characters
translate(File).
1.2
Open file
1.2.1
Read line from file
string of characters
t2d(File).
1.3
Open file
1.3.1
Read line from file
string of characters

Analyze French Sentence

Once Ordinotrad has a sentence string to work with, it immediately proceeds to analyze it. If the word-acquisition module has been invoked by the user, the user may be asked to define new words. Otherwise, the user is not involved until the process is complete. It is possible to interrupt the process momentarily, but the user cannot force Ordinotrad to backtrack to previous procedures once they are completed.

The analysis procedure is subdivided into preprocessing, morphological analysis, parsing, and case analysis. Preprocessing takes the string of characters (including spaces and punctuation) and breaks it up into words, numbers and symbols. Details on the definition of "word," "number," and "symbol" are provided in the "Preprocessor" file in Appendix 4. These lexical units are annotated as to their type and are arranged into a list.

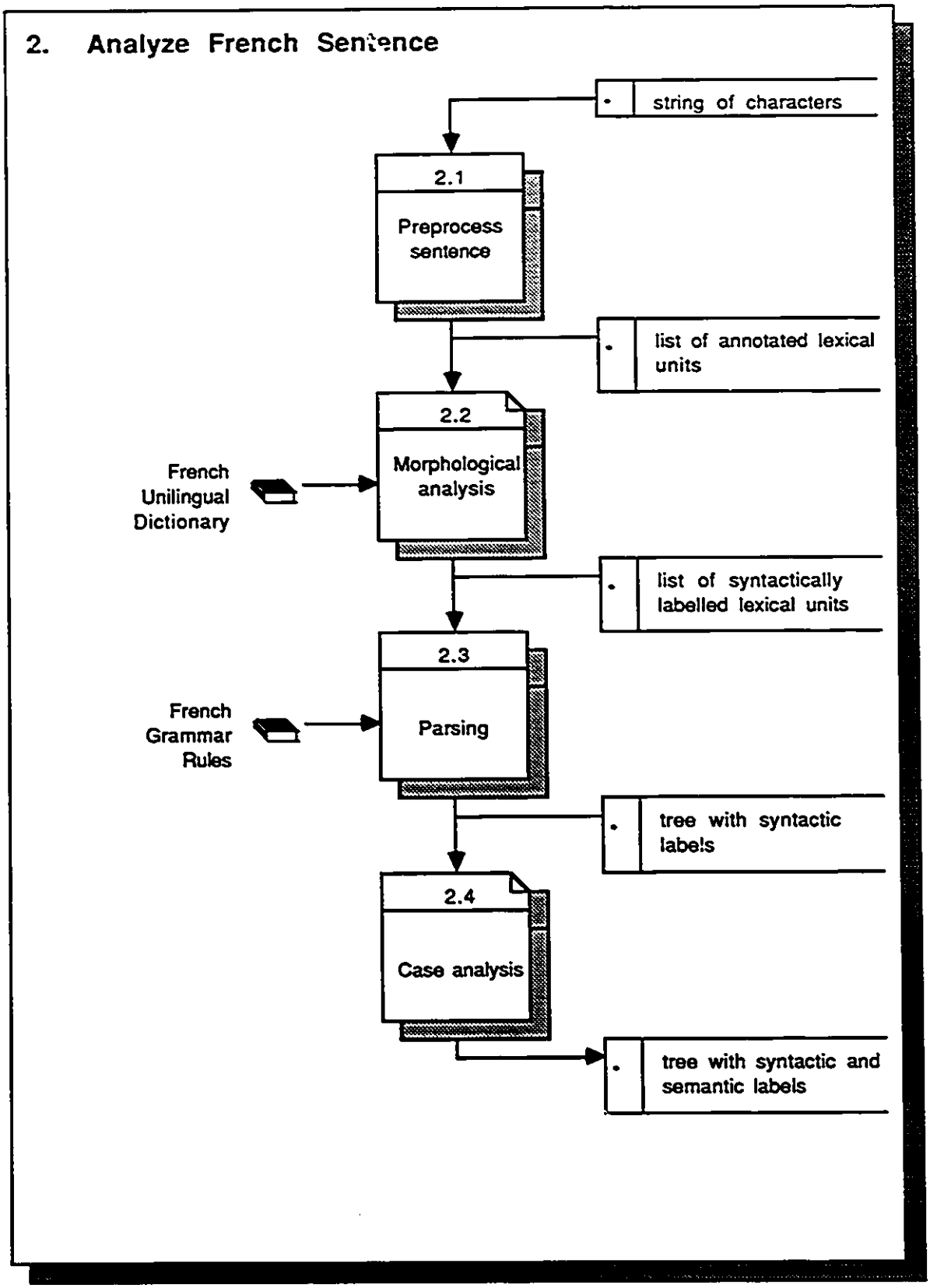
Morphological analysis, described in more detail below, takes the list of given words and replaces each word with a small packet of information about the word: its dictionary form, morpho-syntactic information about the word and its inflections, and semantic information about the dictionary form.

The list of syntactically labeled lexical units is then forwarded to the parser, which identifies propositions[13] and mentions and describes the syntactic relations between their components in detail.

The case processor takes the results of parsing and establishes case relations between events and their arguments.

---

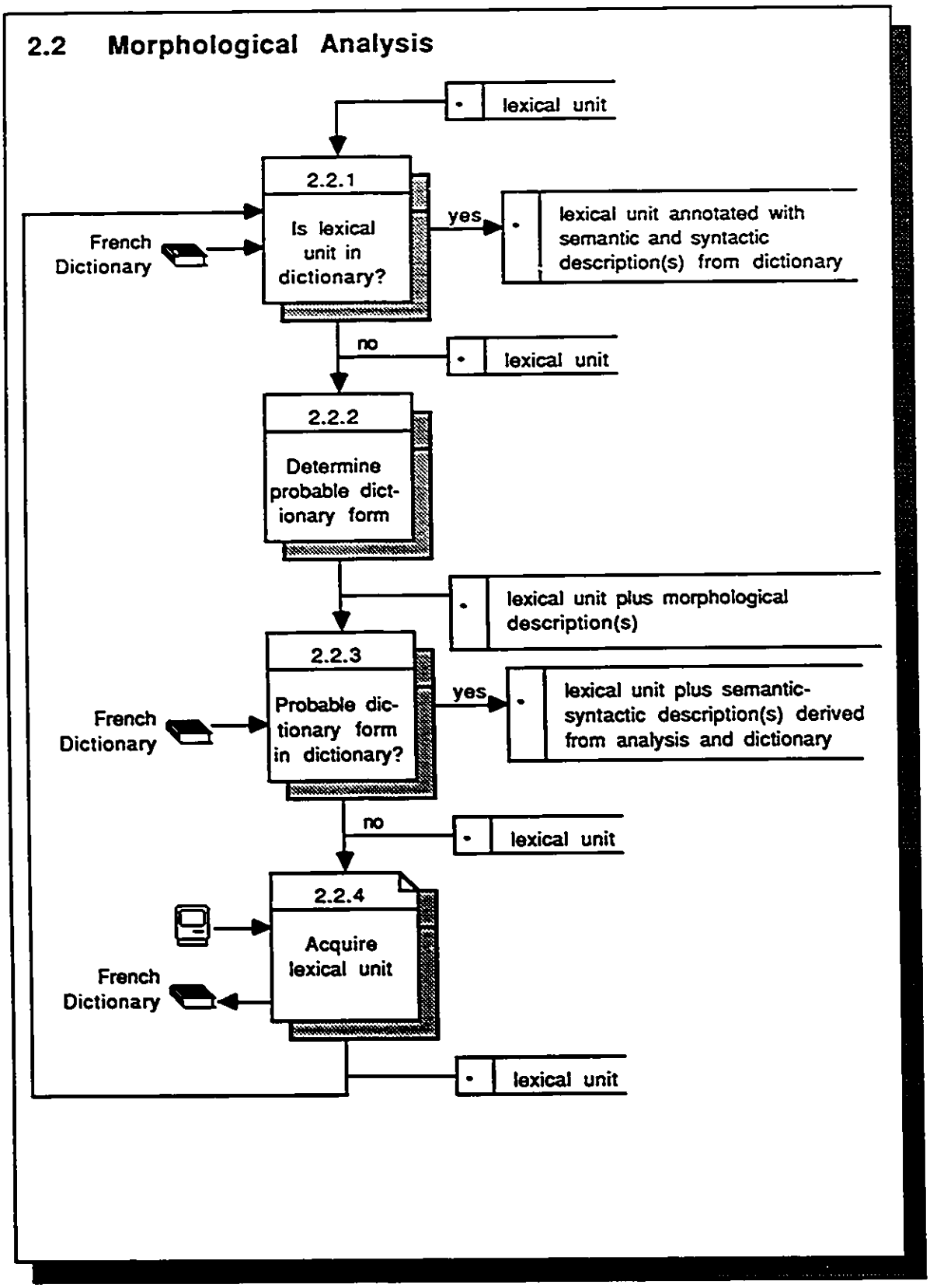
13 As will be discussed in Chapter 2, a sentence can be rewritten as "modality plus proposition" (S -> M + P); a surface structure, the sentence, is a manifestation of a deep structure, the proposition. In this thesis, therefore, the term "proposition" is roughly synonymous with "sentence." A distinction is made, however, between the two terms, for one proposition can underly two entirely different sentences: the original French one and the English translation.


2. Analyze French Sentence
string of characters
2.1
Preprocess sentence
list of annotated lexical units
2.2
Morphological analysis
French Unilingual Dictionary
list of syntactically labelled lexical units
2.3
Parsing
French Grammar Rules
tree with syntactic labels
2.4
Case analysis
tree with syntactic and semantic labels

Morphological Analysis

Checks are made to see if the words in the list (as they are found in the input) are found in the French monolingual dictionary. If not, they are then analyzed morphologically to determine probable dictionary forms.[14] The dictionary is then checked for these forms. Successful analyses at this point replace the original word in the list with the dictionary form and the appropriate morphological-syntactic information discovered about the input form. If words are unrecognizable, the word-acquisition module is invoked to have the user inform Ordinotrad about the word, provided the module has been loaded into memory. If it hasn't, the user is informed that processing cannot continue due to shortcomings in the dictionary and the process is unceremoniously aborted. Ordinotrad will not process a sentence containing a word it cannot recognize.

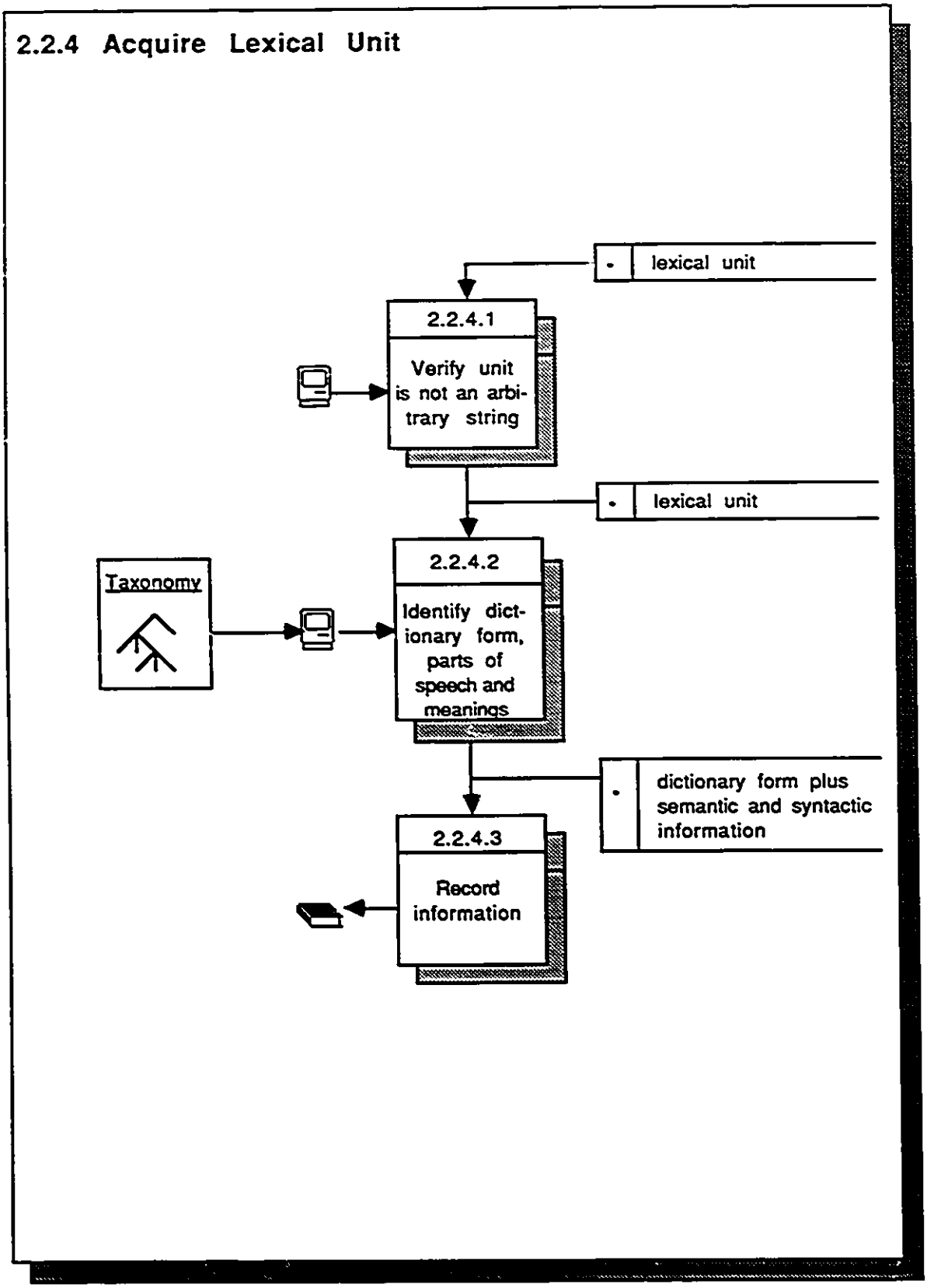
[14] Ordinotrad tests the following assumptions in the order given: the input word is in the dictionary; the input word is feminine, and the masculine singular form is in the dictionary; the input word is plural, and the masculine singular form is in the dictionary; the input word is feminine plural, and the masculine singular form is in the dictionary; the input word is a conjugated form of a verb whose infinitive form is in the dictionary; the input word is unknown. For more details, see the "Morphological Analysis" file in Appendix 4.


2.2 Morphological Analysis
lexical unit
2.2.1
Is lexical unit in dictionary?
French Dictionary
yes
lexical unit annotated with semantic and syntactic description(s) from dictionary
no
lexical unit
2.2.2
Determine probable dictionary form
lexical unit plus morphological description(s)
2.2.3
Probable dictionary form in dictionary?
French Dictionary
yes
lexical unit plus semantic-syntactic description(s) derived from analysis and dictionary
no
lexical unit
2.2.4
Acquire lexical unit
French Dictionary
lexical unit

Acquire Lexical Unit

The input to the word-acquisition module is the word in its original form. The output, which is added to the dictionary, is the dictionary form of the lexical unit together with semantic and syntactic descriptions.

As can be noted in the diagram on the preceding page, morphological analysis involves a loop back to the beginning of the process once a word has been acquired. The reason for this is that word acquisition, in this system, does not include morphological analysis of input. It simply queries the user for the dictionary form and builds the dictionary entry accordingly.


2.2.4 Acquire Lexical Unit
lexical unit
2.2.4.1
Verify unit is not an arbitrary string
lexical unit
Taxonomy
2.2.4.2
Identify dictionary form, parts of speech and meanings
dictionary form plus semantic and syntactic information
2.2.4.3
Record information

Parsing

In terms of the parser's capabilities, propositions are independent (typically subject+predicate), subordinate (typically that+subject+predicate) or ellipsed (no surface subject) propositions, *or any number and mixture of these propositions*. Mentions are the various elements of a letter which are not recognized as propositions (e.g., "encl.", "À l'attention de M. X", phone numbers, postal codes, and so forth). For a list of annotated lexical units to be considered a proposition, the arrangement of its members must pass tests for well-formedness: subject-verb agreements, noun-adjective agreements (see the "Parser - agreements" file in Appendix 4 for an actual example), a maximum of two pre-verbal particles in the right order, and so forth.

The grammar rules used to parse the French sentences are procedural in nature: the parser scans a sentence from left to right, building a tree using predefined syntactic structures if the words meet the conditions placed upon the structures.[15] The tree therefore consists in a hierarchy of syntactic labels which are assigned to the various sentence components, and the elements of the original list become the terminal branches of the tree.

All syntactic types may be comprised of conjoined instances of the same type. For example, one or more conjoined noun phrases can function as a single noun phrase. A noun phrase in itself may be comprised of a single pronoun or of zero or more head nouns with zero or more modifiers.[16] It can be noted in the results in Appendix 3 that a noun phrase is always assumed to consist of at least two noun phrases, of which the second may be a "zero" noun phrase.[17] The use of "zero" instances has also been applied to other

---

15 Because of the diversity of paths that can be followed by the parser, no diagram has been provided here. The reader is encouraged to consult the "parser" files in Appendix 4, where the grammar rules are given.

16 There must be at least one noun or modifier (in particular an ordinal adjective) to allow for instances such as "il en a trois.". See Appendix 3 for more details on this example.

17 The reason for this structure is to facilitate programming. Once a noun phrase, for example, is found, a check is made to see if the next sentence component is also a noun phrase. For example, the sentence component "toutes les lettres" would form part of a tree in the following manner. Note that the "zero" instance is added here for the sake of clarity. It is generally not shown by the system.

```
np
    np
        noun
            lu(low,noun-(lettre,f,pl,common,count,[nonselfmoving,submatter]))
            modifiers
                lu(low,adj-(tout,indef,f,pl,[quantity]))
                lu(low,art-(les,def,f,pl))
    np
        ø
```

For a description of how conjunctions and series of equivalent types of syntactic units are handled, see the note on page 3 of the "parser" file in Appendix 4.

syntactic types (e.g., prepositional phrases, etc.) and its use is always reflected in tree structures by what may appear to be a redundant syntactic label.

It may be noted that the present grammar-rule definition of "proposition" in the parser allows for incomplete sentences, for the parser accepts subordinate propositions as propositions in themselves, even if there is no independent proposition. This situation is more serendipitous than intentional; yet as a translator, I have not infrequently had to deal with incomplete sentences.

Examples of syntactic trees showing the relationships between sentence components are provided in Appendix 3. In addition to the usual labels (noun phrase, event, adjective, etc.) found in syntactic trees, the parser also determines the modalities of the independent, subordinate or ellipsed sentences. In this way, the tense (present, future, etc.), mood (indicative, subjunctive, conditional, infinitive, participle), voice (active, passive), attitude (affirmative, negative, doubt) and form (declarative, interrogative, exclamatory/imperative) are separated from the proposition or "meaning" of the sentence before it undergoes analysis by the case processor. The assumption here is that the principles of case grammar may be applied without reference to time frames, moods, voices, attitudes and forms.[18]

Case Analysis

The case processor takes the syntactic tree produced by the parser and inserts a level of case grammar labels which reflect the relationships between sentence components and the appropriate, controlling "event."[19] In cases where there are no events, input is treated as a "mention" and case processing is skipped entirely. Labels (e.g., Agent, Affected, Beneficiary, etc.) are attached once checks have been made with respect to the order of input and to the intensional meaning of the sentence components. The way in which labels are determined is explained in detail in Chapter 2. An assumption underlying the application of case labels here is that they are common to French and English, and that the list of labels used for semantic analysis of French input will therefore be useful in the generation of English output.

Case processing succeeds only if the tree it is given is well-formed in terms of semantic consistency: i.e., the sentence components playing the roles required by the event

---

[18] This point may be debatable, as there are undoubtedly instances in which tense or mood affect meaning. The separation is made for the time being, as semantic analysis seems to succeed without such information. This is not to deny the possibility of taking a sentence's modalities into consideration when establishing the semantic relationships it contains.

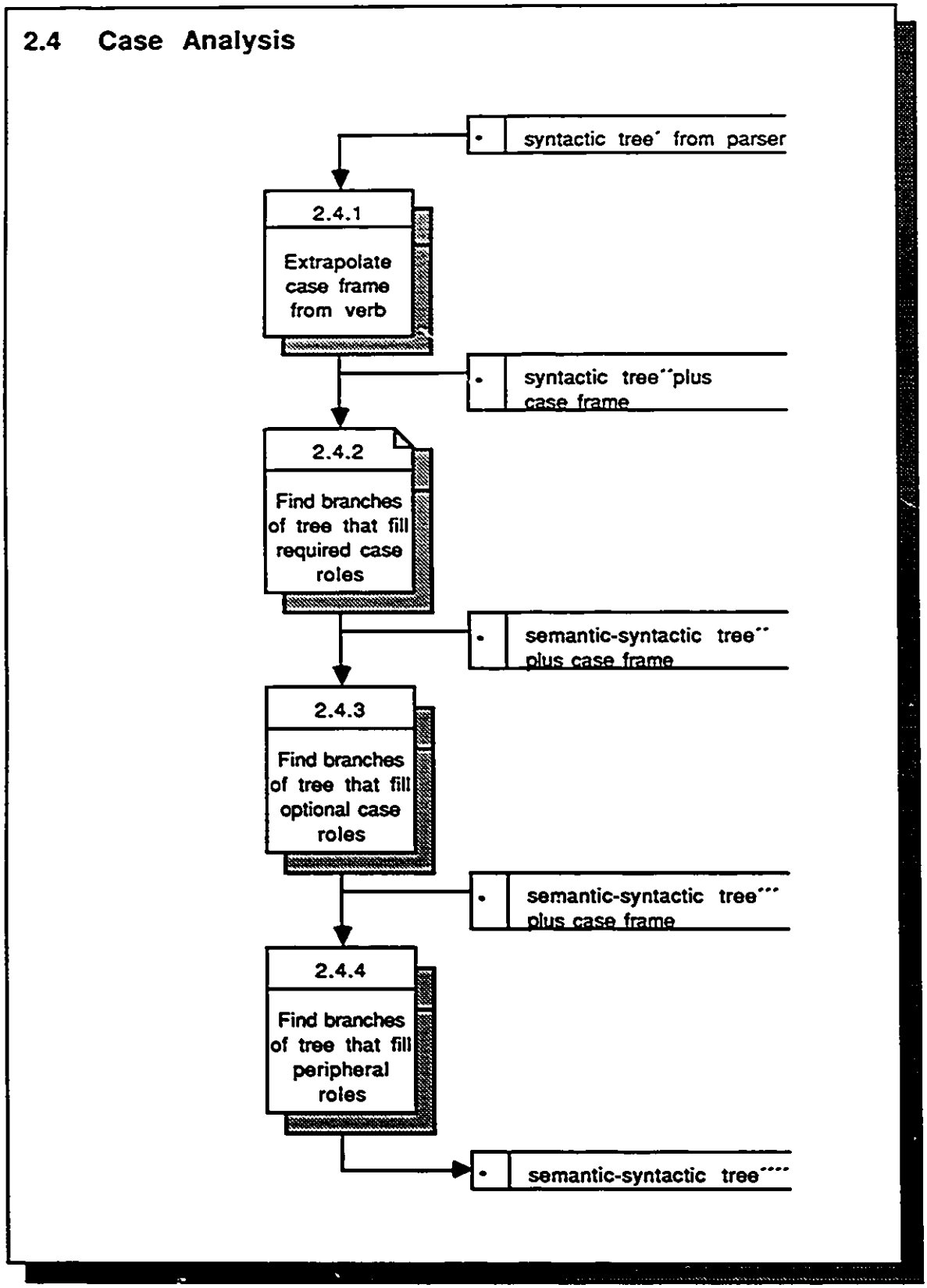
[19] See Chapter 2 for a full discussion on "events" and "case labels."

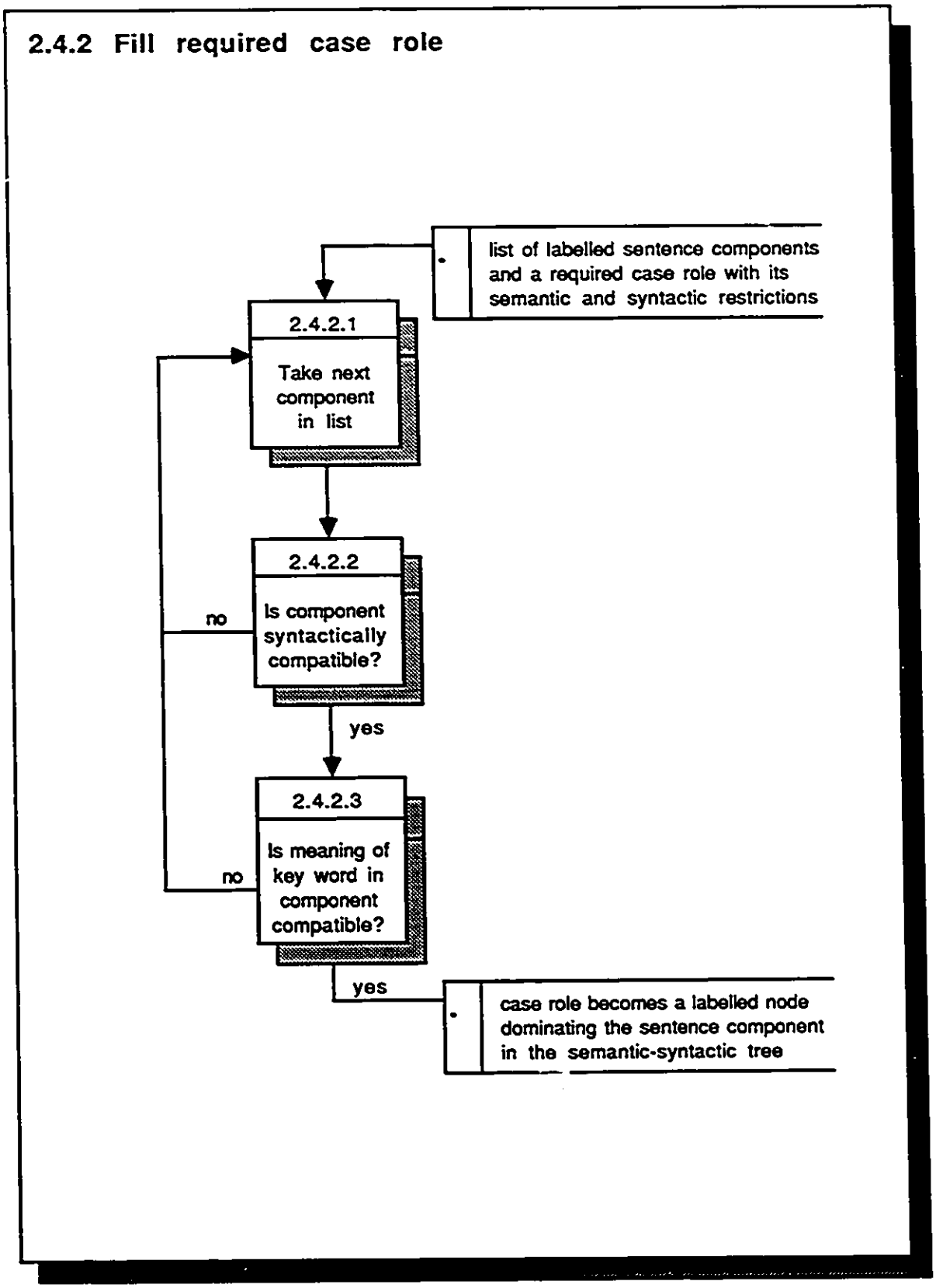
meet the semantic requirements of the roles. Case processing relies on the case frames associated with each event. If an event is described as requiring a certain case role to be filled and there are no sentence components that can fill it, the sentence is considered ill-formed and the process is aborted. With regard to optional roles, there may not be sentence components available to fill them. In such instances, the "fill optional roles" procedure is simply skipped. Peripheral roles[20] are also not filled if there are no sentence components available to fill them.

In the case of mentions, which do not have events (e.g., an address), case analysis is also bypassed and the translation takes into account only the words and their syntactic relationships.

The final output of analysis is a tree structure containing syntactic and semantic information on sentence components and on the relationships between them. This tree representation serves as input for the transfer and generation modules.

20 These roles do not distinguish events and therefore do not form part of a case frame. They include: Location, Time, Manner, Cause. See Chapter 2 for a discussion of these roles.


2.4 Case Analysis
syntactic tree' from parser
2.4.1
Extrapolate case frame from verb
syntactic tree'' plus case frame
2.4.2
Find branches of tree that fill required case roles
semantic-syntactic tree'' plus case frame
2.4.3
Find branches of tree that fill optional case roles
semantic-syntactic tree''' plus case frame
2.4.4
Find branches of tree that fill peripheral roles
semantic-syntactic tree''''


2.4.2 Fill required case role
list of labelled sentence components and a required case role with its semantic and syntactic restrictions
2.4.2.1
Take next component in list
2.4.2.2
Is component syntactically compatible?
no
yes
2.4.2.3
Is meaning of key word in component compatible?
no
yes
case role becomes a labelled node dominating the sentence component in the semantic-syntactic tree

Transfer & Generation

In this system, semantics and case grammar analysis are also part of sentence generation. The case frame of the target-language verb controls generation in that it controls the rearrangement of word groups. The arrangement of words within word groups during transfer is controlled by the syntactic nature of the words themselves (e.g., use of articles with mass nouns), and the case frames do not enter into play at that point.

In generating word groups (e.g. a prepositional phrase), the English words are chosen from lists of possibilities provided in the transfer dictionary. These choices are made taking into consideration the syntactic unit (e.g., an adjective may be replaced with an adjective or a verbal) and the meaning of the French input.[21] Selections are then capitalized and pluralized, if necessary, and then re-ordered within smaller word groupings first (e.g., a noun phrase), taking into account the compatibility of their syntactic markers[22] to produce semantically and syntactically well-formed word groupings. Prepositions are added if the event-case relation combination calls for them. Once individual word groupings have been established, they too are put in order for the output sentence. Surface-syntactic function markers (e.g., subject, object) are assigned to the word groupings taking into account their case relations to their respective event.

The "flow" of the generation procedures always begins with the smallest units provided by the transfer dictionary.[23] From there, noun phrases are built up, taking into account the determiners and modifiers of the head noun. It may be that a modifier of a head noun is itself a subordinate proposition (e.g., nous avons reçu la lettre *que vous nous avez adressée*). In such cases, the equivalent subordinate proposition in its entirety would be generated before the noun phrase was constructed. If subordinate propositions themselves contain other subordinate propositions, the latter too would have to be generated before the "higher up" proposition. Once all the sentence components have been generated individually (i.e., noun phrases, prepositional phrases, etc.[24]), the verb is generated in

---

21 If, for example, a French word points to several English words in the transfer dictionary, only the English word with the same meaning as that linked to the French word in the original context (i.e., selected during case processing) is retained.

22 A future development would introduce semantics into the process of putting adjectives with nouns in order to avoid nonsensical structures (for example, the system translation of the mention "meubles de bureau" is "furniture of office"). Assumptions could be made to the effect that where the head noun is an artifact, and the qualifier is a prepositional phrase containing a "physical-space" type of noun, "physical-space" becomes an attribute of the artifact (i.e., put "office" ahead of "furniture").

23 The system is not currently designed to handle transfers in which French idiomatic expressions point to standardized equivalents in English, e.g., "en guise de conclusion" -> "in conclusion".

24 See the diagram on page 25 for an indication of what sentence components are generated separately from each other (see also the "Generator - sentence components" file in Appendix 4).

accordance with the modalities of the sentence. With the "parts" ready, the generator then re-orders them based on the case relations existing between the sentence components, the event, and the sentence modalities.

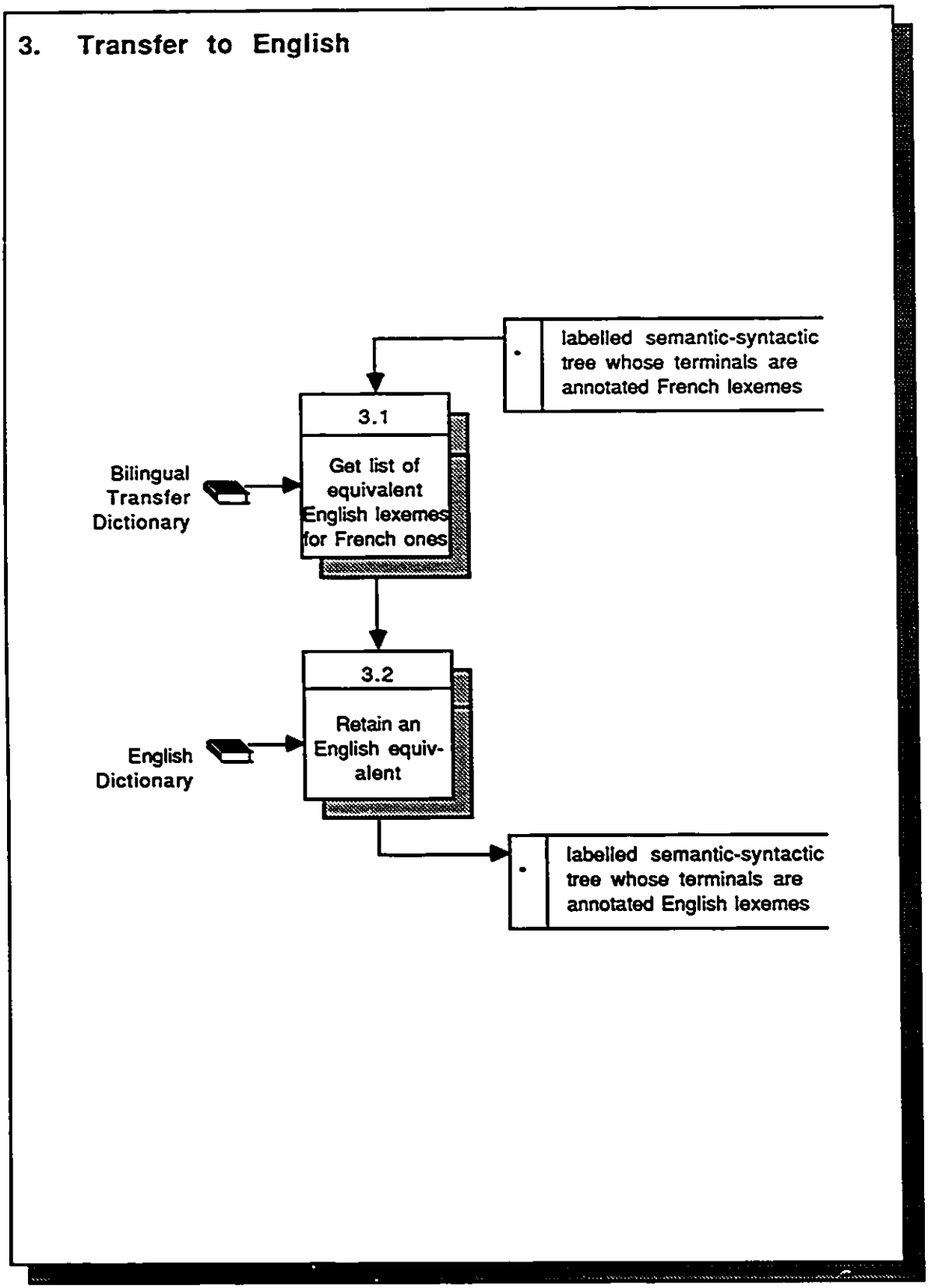
When all components have been generated and ordering has been completed, the labels are removed and the tree is reduced to a list of words.

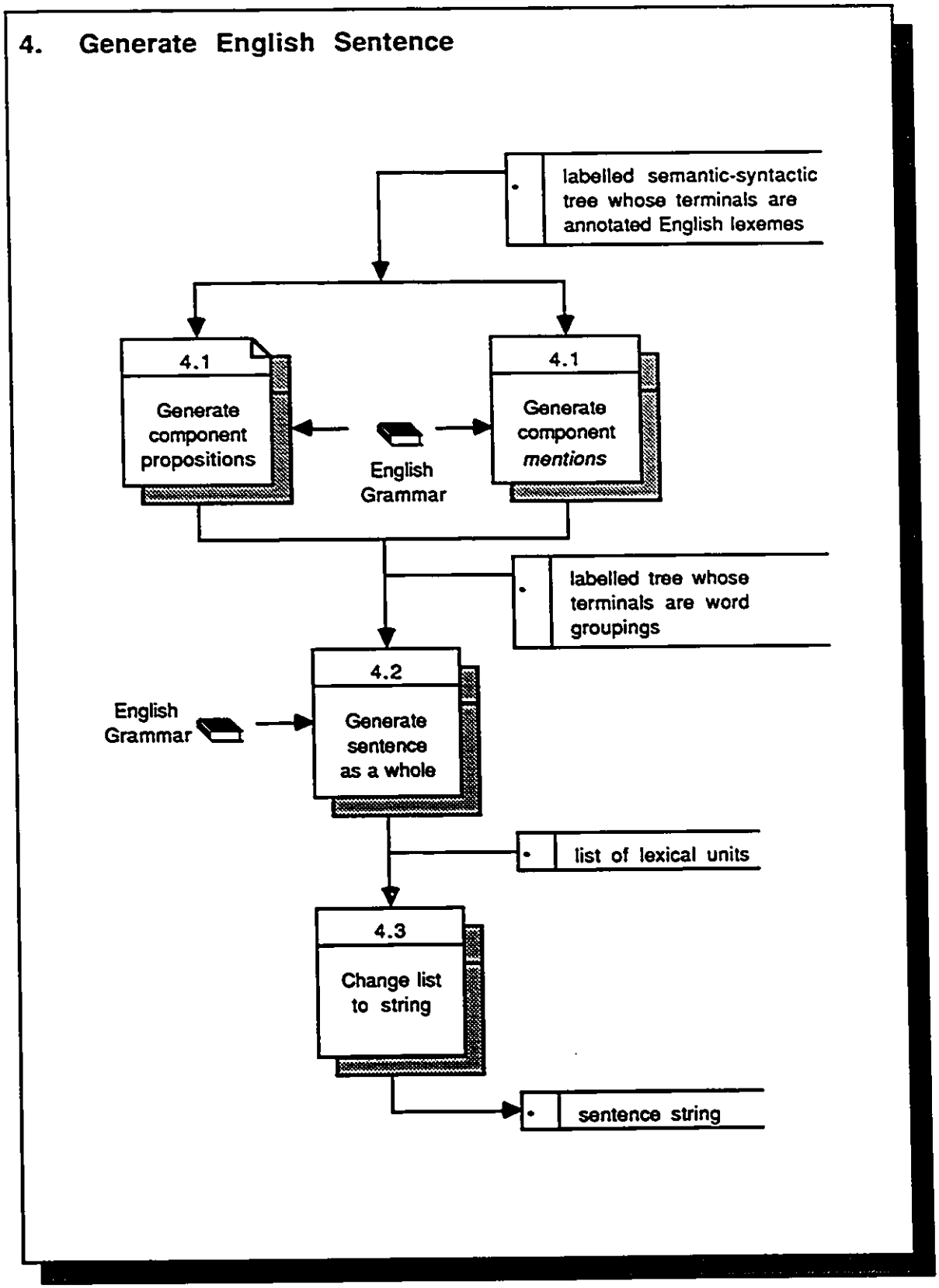
The final processing step involves inserting spaces between words, as well as punctuation and symbols where appropriate, and capitalizing the first word of the sentence. The final list includes all the characters which are converted to a string for output.

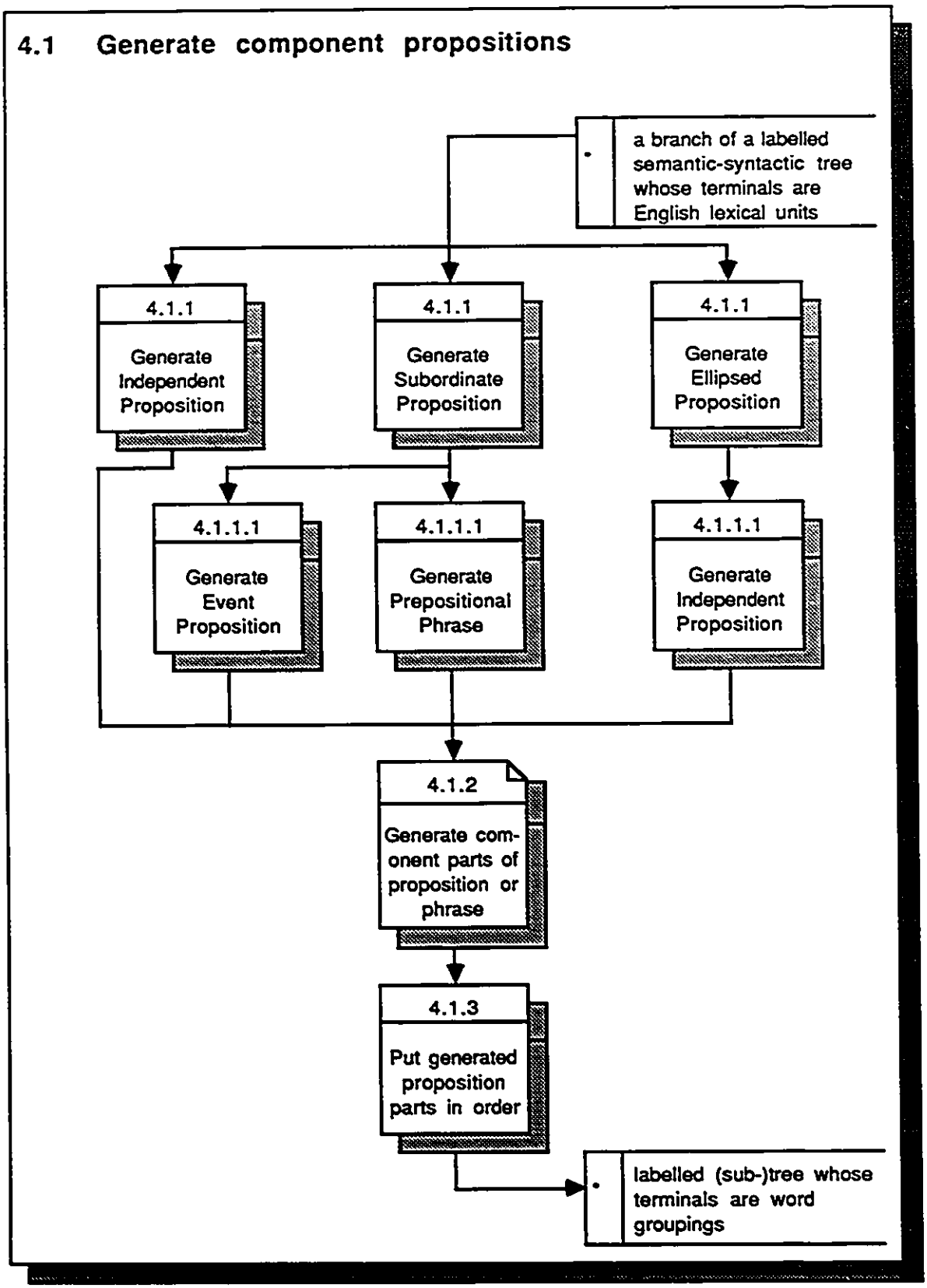
As can be seen in the final diagram, the output string is displayed on the screen for the user if the original command was "translate." or "translate(Filename)." or is recorded in a pre-specified file on disk for possible post-editing, spell checking and formatting with a word-processor, if the original command was "t2d(Filename).".[25]

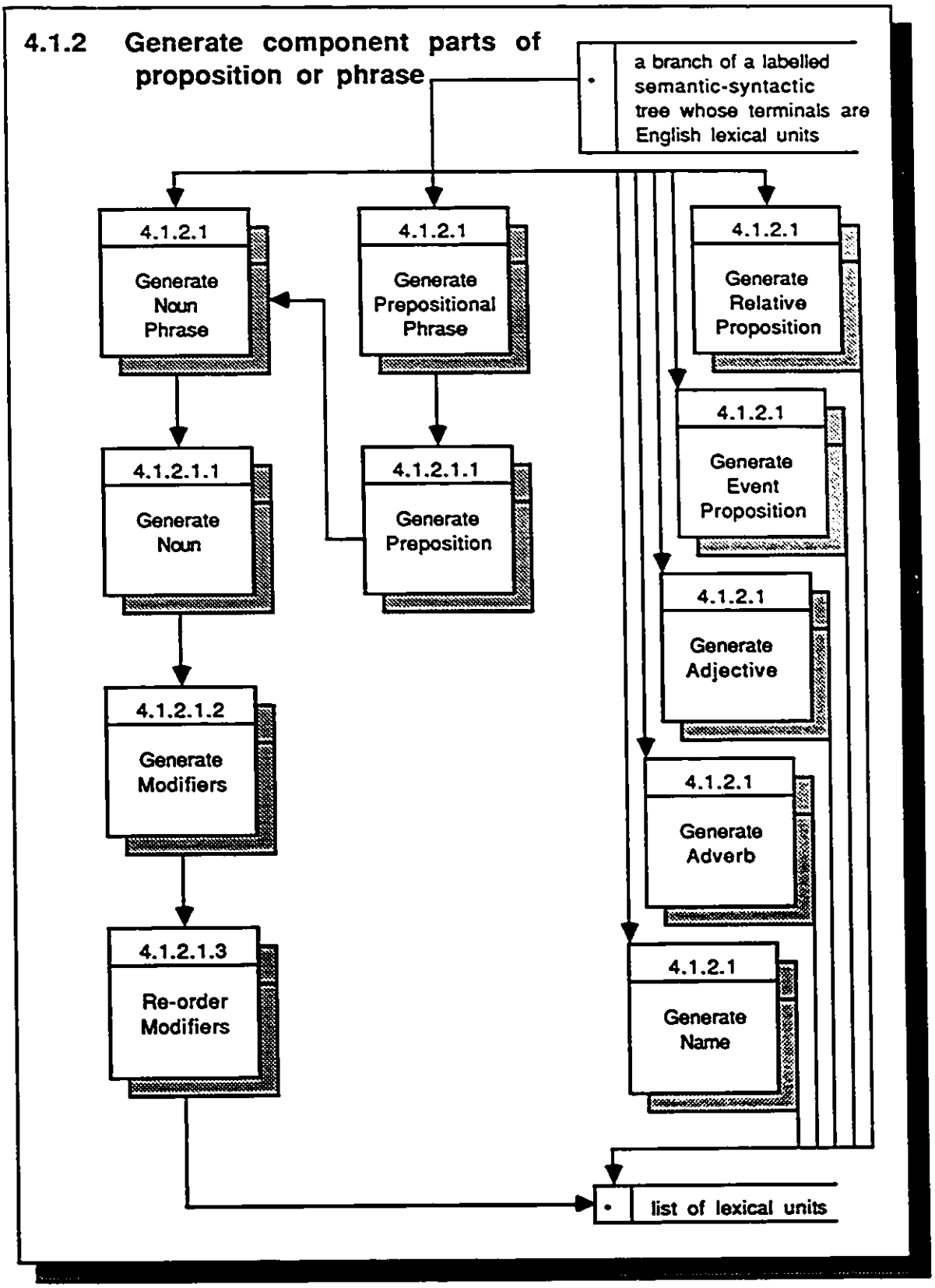
---

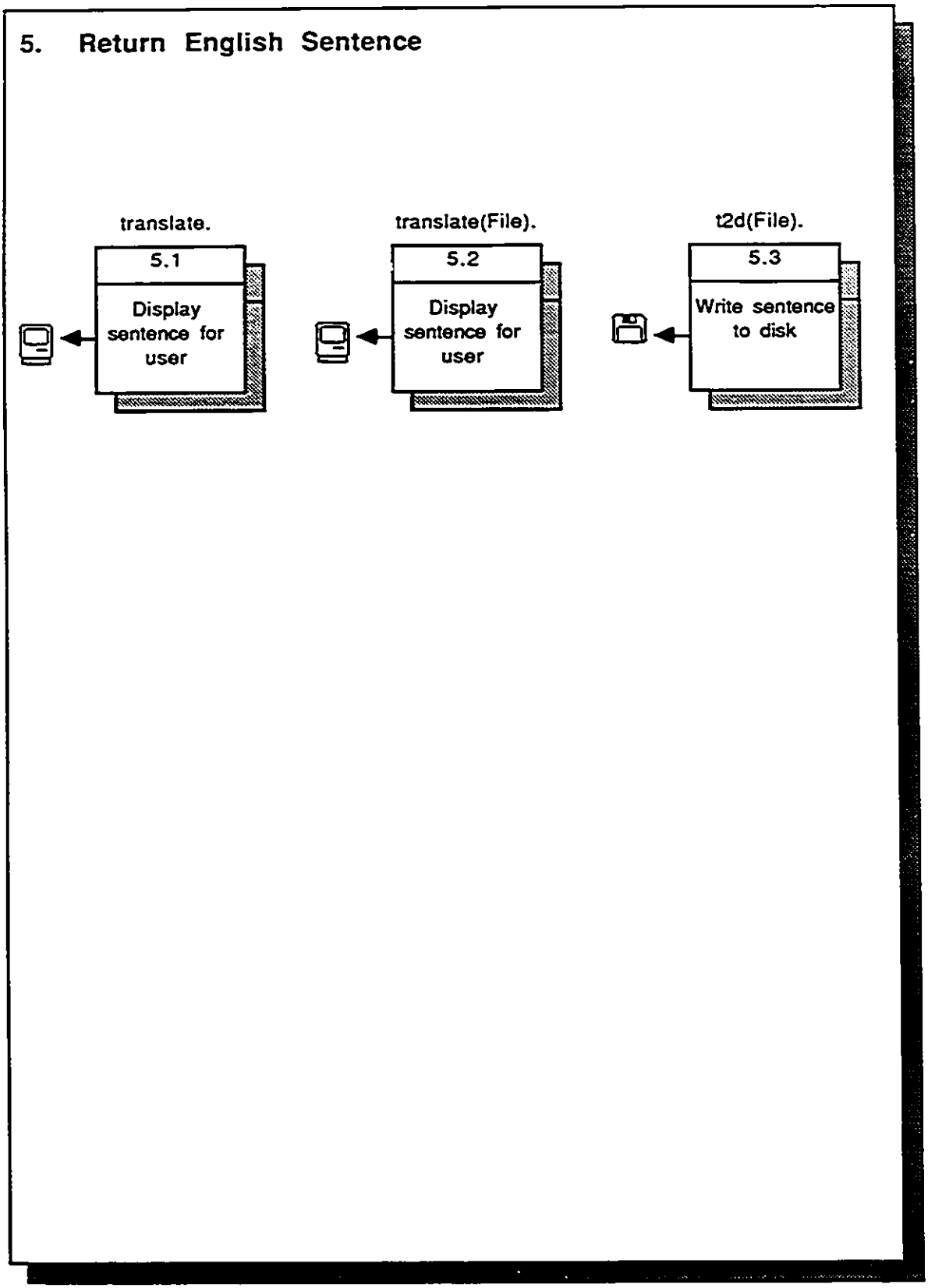
25 See pages 6 and 7.


3. Transfer to English
labelled semantic-syntactic tree whose terminals are annotated French lexemes
3.1
Get list of equivalent English lexemes for French ones
Bilingual Transfer Dictionary
3.2
Retain an English equiv-alent
English Dictionary
labelled semantic-syntactic tree whose terminals are annotated English lexemes


4. Generate English Sentence
labelled semantic-syntactic tree whose terminals are annotated English lexemes
4.1
Generate component propositions
English Grammar
4.1
Generate component *mentions*
labelled tree whose terminals are word groupings
English Grammar
4.2
Generate sentence as a whole
list of lexical units
4.3
Change list to string
sentence string


4.1 Generate component propositions
a branch of a labelled semantic-syntactic tree whose terminals are English lexical units
4.1.1
Generate Independent Proposition
4.1.1
Generate Subordinate Proposition
4.1.1
Generate Ellipsed Proposition
4.1.1.1
Generate Event Proposition
4.1.1.1
Generate Prepositional Phrase
4.1.1.1
Generate Independent Proposition
4.1.2
Generate com-onent parts of proposition or phrase
4.1.3
Put generated proposition parts in order
labelled (sub-)tree whose terminals are word groupings


4.1.2 Generate component parts of proposition or phrase
a branch of a labelled semantic-syntactic tree whose terminals are English lexical units
4.1.2.1
Generate Noun Phrase
4.1.2.1
Generate Prepositional Phrase
4.1.2.1
Generate Relative Proposition
4.1.2.1
Generate Event Proposition
4.1.2.1
Generate Adjective
4.1.2.1
Generate Adverb
4.1.2.1
Generate Name
4.1.2.1.1
Generate Noun
4.1.2.1.1
Generate Preposition
4.1.2.1.2
Generate Modifiers
4.1.2.1.3
Re-order Modifiers
list of lexical units


5. Return English Sentence
translate.
5.1
Display sentence for user
translate(File).
5.2
Display sentence for user
t2d(File).
5.3
Write sentence to disk

SL AND TL MONOLINGUAL DICTIONARIES

The monolingual French and English dictionaries provide grammatical and semantic information for individual words. The information in the dictionaries is in descriptive form (i.e., not procedural) and is maintained separately from the rules governing processing. If a given lexical unit is polysyntactic or polysemantic[26] (e.g., sommes, faits), each possibility is provided in the dictionary. It is up to the parser and case processor to determine which is being used in the immediate context, in other words, which textual meaning is to be chosen from the list of structural meanings.

The dictionaries' entries as they exist at print-time of this document are listed in Appendix 4. Note that this is a very dynamic part of Ordinotrad, as every use of the system with the word-acquisition module enabled may lead to the creation of new entries, depending on the input. There is no database management system available for the dictionaries, though new entries can be created using the above-mentioned module built into Ordinotrad (which ensures consistency of format and the correct number of parentheses) or with a word-processor. Existing entries may be edited with a word-processor, and any change is likely to affect resulting translations.

Entries may become very complex because verbs with more than one intensional meaning require separate case frames for each meaning (cf. "servir" in the French dictionary). Other syntactic types, such as nouns, adjectives, adverbs, and so forth require much less complexity in their definition.

TRANSFER DICTIONARY

Entries in the transfer dictionary are basically pointers that link individual French lexical units[27] with a list of possible English lexical units. Any change in these entries will affect output for two reasons. First, order within the transfer dictionary entries is important, since the first equivalent encountered that meets the semantic requirements is the one that is retained in the translation. Therefore, changing the order may affect the translation. Second, the generation module of Ordinotrad does not start from a given meaning and try to find a word that matches it. To control generation, Ordinotrad depends

---

26 Cf. "structural meaning" in the glossary in Appendix 5.

27 In the current version of the system, lexical units are, for all intents and purposes, the same as words. The clearest indication of what is accepted as a lexical unit by the system is the contents of the transfer dictionary. It is hoped that the transfer dictionary will ultimately contain pointers from French idiomatic expressions to equivalent English ones. In such cases, these expressions would be dealt with as lexical units.

on the transfer dictionary to provide leads, i.e., possible SL-TL equivalents suggested by the user when the dictionary entries were created.[28]

[28] An interesting prospect would be to enable Ordinotrad to measure equivalence. It would then be less dependent on the competence of the user teaching it new words. To create such a meta-system would necessarily entail developing a detailed method for measuring equivalence, or at least setting up a scale of probabilities for making educated guesses about equivalence.

Chapter 2

Case Grammar

OVERVIEW

Data structures are an integral part of a natural-language processing system, particularly a machine-translation system, since they enable it to represent information about the structure of a sentence and about its meaning. Several natural-language processing systems use structures based on the principles of case grammar,[1] and there are some instances of their use in existing machine-translation systems.[2] As this work proposes to enable a machine to translate by implementing case grammar principles, it is appropriate at this point to give a clear description of them.

In this chapter we will briefly describe reasons for developments in case grammar, taking into consideration historical and present-day linguistic motivations. Underlying all these developments are two main elements of language: syntax and semantics. Their roles in a case grammar will also be considered. The main part of the chapter is concerned with a detailed description of case grammar, its component parts (case, case frame, case system), relations between component parts, and relations between component parts and surface syntactic structures.

A BRIEF HISTORY OF CASE GRAMMAR

Historically, 'cases' were first used in the study of classical languages with reference to categories of nouns (nominative, genitive, dative, etc.). It is only recently, in the last twenty years, that 'case' has been reconsidered as useful for descriptive generative linguistics, but this time the meaning of the term has been broadened. Formerly restricted to morphological surface analysis of categories of words, the term now encompasses 'deep relations' or relations among sentence components which mediate messages. As the structure of these deep relations became better understood, there arose a set of principles which helped to describe natural language in a structured fashion. This set of principles has been given the name 'Case Grammar.'

---

1 Bertram Bruce, "Case systems for natural language," *Artificial Intelligence* 6 (1975): 345.

2 Harold J. Somers, "The need for MT-oriented versions of Case and Valency in MT." *Proceedings of COLING 86*. Bonn (1986): 118.

The evolution of 'case' and 'case grammar' can perhaps be best traced by considering the reasons why they first interested linguists. Historical motivations for developing case grammar, not necessarily in chronological order, seem to include: understanding the use, system or evolution of affixes; a search for linguistic universals; and, above all, improvements to transformational grammar.

HISTORICAL MOTIVATIONS FOR DEVELOPING CASE GRAMMAR

According to Fillmore, the notion of case is traditionally reduced to studies of semantic relationships within a sentence, to studies of dependency relations between specific affixes and lexical-grammatical properties of neighbouring elements, or to statements concerning morphophonemic realizations of syntactic relations.[3] These areas of interest provide several approaches to studying case as a morphological component of words. There are those more traditional studies which focus on the semantic use of affixes, others focus on a particular system of such affixes for a given language, and a third variety of study considers the evolution of cases or of particular case affixes. Fillmore also indicated that generative grammarians had more recently introduced a variation on the semantic uses of affixes, namely to consider affixes (which they called case markers) as a surface structure reflex of various kinds of deep or surface-syntactic relations.[4] It is in the latter area and in that of the study of "formal and substantive syntactic universals"[5] that Fillmore introduced his concept of case as a labeled relation.[6]

Notably, Fillmore's presentation of "The Case for Case" was part of a conference on the role of universals in linguistic theory. He hypothesized that the uses of cases were comparable across languages, although he did note that case labels and case systems themselves may be different from language to language.[7] Even so, other linguists have reacted against the concept of structured language universals. Nida, for example, makes the point that "constructing an elaborate set of rules for determining various sets and degrees of correspondence between languages is possible, but such rules are not very

---

[3] Charles J. Fillmore, "The Case for Case," *Universals in Linguistic Theory*, Emmon Bach & Robert T. Harms, eds., (New York: Holt, Rinehart and Winston, 1968) 2, 3.

[4] Fillmore 5.

[5] Fillmore 2.

[6] This notion is discussed in detail later in the chapter. For now, a case can be understood to be the label given to a specific role played by some thing in a given Event, or, syntactically speaking, a label for the relation held between the word group and the verb.

[7] Fillmore 19.

useful...."[8] We note, however, that his argument is not against the existence of universals[9] nor against the possibility of writing rules describing the realization of universals, but against the usefulness of the rules, especially in translation, where "success...depends ultimately more on æsthetic sensitivity than upon logical deduction."[10]

Since the machine-translation system described herein does implement rules, one general point needs to be made. As a translator, I agree with Nida that translation is a complex linguistic, cultural and æsthetic activity and is not strictly mechanical or rule-governed[11] (when done by humans). Nonetheless, in any attempt to imitate the product of human translation using a machine, the process *is* mechanical and must therefore be rule-governed to a large extent. In this context we can find justification for researching universals and case grammar rules, but in no way is it possible to force the results of this research on human translation. If, in the course of research on machine translation, light is shed on particular sequences of activities followed by human translators, so much the better, but it is not necessarily so.

In addition to his search for linguistic universals, Fillmore also developed his 'case for case' in the context of transformational grammar. Yet according to Anderson, case grammars can be distinguished from previous transformational grammars in two respects. in deep structure representations, semantic-relation labels are used instead of part-of-speech labels; and the terms 'subject' and 'object' are used only in describing the surface structure which is derived from deep structure.[12] Because of these differences, case grammar provided a means for introducing semantics into transformational grammar, which was primarily syntactic up till then.[13] It should be noted, however, that case grammar, as developed by Fillmore, did not require *explicit* transformational links to a surface structure. It was only later, under descriptive-linguistic motivation, that such links were deemed

---

8 Eugene A. Nida, "Barriers and Bridges in Translating," *Building Bridges*, Proceedings of the 27th Annual Conference of the American Translators Association, October 1986, Karl Kummer, ed. (Medford, N.J.: Learned Information, Inc., 1986) 12.

9 Nida 10. Nida himself states in the same paper that certain functions of language and rhetoric, including formal and conceptual devices, are universals even though they may not mean the same across languages.

10 Nida 12.

11 Nida 12.

12 John M. Anderson, *On Case Grammar: Prolegomena to a Theory of Grammatical Relations*, Croom Helm Linguistic Series (London: Croom Helm Ltd.; Atlantic Highlands, N.J.: Humanities Press, 1977) 188.

13 Walter Anthony Cook, *Case Grammar: Development of the Matrix Model (1970-1978)* (Washington, D.C.: Georgetown University Press, 1979) 36.

necessary in order to provide a means to explain the acceptability[14] of the syntactic or semantic coherence of a sentence.

LINGUISTIC MOTIVATIONS FOR CASE

Fillmore's search for language universals was directed to the discovery of underlying relations which determine syntactic and semantic relations at a surface level.[15] Anderson makes the point more forcibly, claiming that a grammatical theory of case must at the very least provide a clear explanation of the relationships between case relations and case form (his terms for labeled deep-relationships and surface structure forms, respectively).[16] He even goes so far as to say that *all* discussion must focus on clarifying the relationships between surface and deep structures.[17] A primary linguistic motivation for developing case grammar then is not only to describe a surface structure in terms of a deep structure, but also to explain the determining relationships between them.

Two other linguistic reasons for developing case grammar are, according to Bruce, to find a way to account for the relative acceptability of sentences and to explain how an intelligent computer system might understand language by focussing on Events instead of syntactic constructions.[18] More precisely, the relative acceptability of sentences can be determined through the use of case structures in conjunction with selection constraints: a case structure dictates required and optional cases, and selection constraints place semantic restrictions on the word groups which fit into each particular relationship.[19]

Another linguistic motivation for case grammar may be to analyze a sentence so as to produce a structured representation of its syntactic components and the semantic relations between them. The utility of this is evident in the context of machine translation which, by the very nature of the computer, requires structured accounts of a sentence for the purposes of processing it further.

---

[14] In this work, a sentence is deemed to achieve 'acceptability' if it does not deviate from the norms of acceptability in a particular linguistic system. I have borrowed this concept from Gideon Toury, "Translated Literature: System, Norm, Performance," *In Search of a Theory of Translation.* Tel Aviv (1980): 48,49. See also Gideon Toury, "Interlanguage and its Manifestations," *Meta* 24.2 (1979): 227. Whereas Toury applies the concept of acceptability to texts as a whole within a literary/linguistic system, I apply it to individual sentences (through analysis of sentence components) within a linguistic system.

[15] Emmon Bach & Robert T. Harms, eds., *Universals in Linguistic Theory* (New York: Holt, Rinehart and Winston, 1968) vii.

[16] Anderson 11.

[17] Anderson 11.

[18] Bruce 329.

[19] Bruce 329.

That there are also broader linguistic motivations for developing case grammar is apparent in the fact that case structures are in some way part of many theories of language and cognition, such as Chomsky's current theory of language.[20]

LEXICOGRAPHICAL MOTIVATIONS FOR CASE

As an extension of 'measuring' the acceptability of a sentence, cases by their nature also provide a way of classifying sentence types and Events. Anderson finds that much of the interest raised by case grammar is due to the way it classifies certain lexical information and lexical items,[21] i.e. by the way lexical sentence components fit into a certain structure defined in terms of case combinations. For example, Events (verbs) can be sorted according to whether or not they require an agent, and things (nouns) can be classified according to whether or not they could fill the role of an agent. For his part, Fillmore finds that case grammar provides a means for producing descriptions of 'sentence types' which may even hold true across languages and that such descriptions, once organized, have the effect of classifying the verbs of a language according to the sentence type in which they may be used.[22] Cook observes that no such classification of all the verbs of the English language has been attempted, even though it appears to be feasible.[23]

It is interesting to note that Cook had stated in 1979 that case frames for verbs (a concept explained in detail below) could form part of lexical entries for verbs in a linguistically sophisticated lexicon[24] and that seven years later, Hirst emphasized "...the need for information about case structures, case flags and semantic markers..." in dictionaries in order for them to be complete and more useful to both human and mechanical language users.[25] It would seem then that by 1986 emphasis had shifted from justifying case grammar to developing applications for it.

Another lexical motivation for developing case grammar is derived not from case grammar itself, but from the possibility of doubling its 'classificatory power' by

---

20 Graeme Hirst, "Why dictionaries should list case structures," *Conference on Advances in Lexicology* (University of Waterloo, November 1986) 3.

21 Anderson 23.

22 Fillmore 21.

23 Cook 48.

24 Cook 48.

25 Hirst 1. According to Hirst, three arguments for including case structure information in dictionaries are: (1) NLP system developers need not duplicate efforts in creating lexicons of case structures; (2) people, second-language learners in particular, may find the information useful; and (3) it would force linguists to come up with a generally accepted theory of case and with a comprehensive description of the cases of a given language. Hirst 7, 8.

elaborating bare case structures so as to include selection constraints. Using the example given above as an illustration, Events could then be sorted according to whether or not they require an agent *and* according to the type of thing which can play the role of the agent. Even though the descriptions of Events in the dictionaries implemented in Ordinotrad have this more elaborate structure, no attempt has been made to classify the Events (there simply are not enough of them at this point to allow for drawing any tenable conclusions from sorting them). Within a machine-translation context, such lexicographical developments in case grammar—particularly case structures which include selection constraints—are of great utility in sentence analysis, quite apart from the usefulness of classifying the words of a given language for other purposes.

ARTIFICIAL INTELLIGENCE MOTIVATIONS FOR CASE

In order for artificial intelligence (AI) to begin to be implemented on a computer, the computer has to know something and, therefore, have a facility for representing knowledge internally. Representing knowledge requires a language tailored to both the computer and the knowledge. Such languages, usually called knowledge representation languages, come in many varieties, depending on the AI-system developers who design them to suit their immediate purposes. Hirst, who argues for recognition of case structures by lexicographers, as noted above, makes the generalization that case structures are incorporated implicitly or explicitly in AI applications whose knowledge representation language involves 'frames with slots.'[26] Typical AI applications are 'expert systems' which contain appropriate portions of the knowledge of human experts and apply them to problem solving. Less typical AI applications are language-oriented systems, such as translation, or natural-language querying of information contained in a database; these systems contain portions of 'linguistic knowledge' and apply them to sentence analysis and generation. In Ordinotrad, for example, 'linguistic knowledge' can be found in the dictionaries, in the short databases contained in the various files (cf. the 'letter' database in the Preprocessor file, or the 'vb_ending' database in the Morphological Analysis file in Appendix 4), and in the dynamic case-based structures generated during processing.

Whereas both language-oriented and problem-solving applications can make use of case structures, it would appear that such structures would be of more utility in processing natural languages, i.e., in creating and testing hypotheses about the use and meaning of

---

[26] Hirst 4.

individual words, sentence components, sentence modalities, sentences as a whole, and so forth. Hirst claims that "research in the last decade has shown the importance of case structures in language understanding.... My own research has shown the importance of case structures in resolving both word ambiguities and syntactic ambiguities."[27] A similar claim may be made for Ordinotrad, as resolution of semantic ambiguities depends on case information, and resolution of syntactic ambiguities on a mixture of syntactic and case information (e.g., the string *le* may be an article as in *le bail*, pronoun as in *il me le donne*, or attribute as in *je le suis*, where attribute reflects a subjective completion in the surface structure).

LINGUISTIC THEORY BEHIND CASE GRAMMAR

The processing of natural language by computer assumes that language as a system provides a means for mediating messages, and that this system can be organized and described; in other words, natural-language processing relies on a linguistic theory. The linguistic theory on which case grammar rests has three significant characteristics: two which Fillmore developed in detail, and one mentioned by Fillmore but explained by other linguists.

FILLMORE'S TWO ASSUMPTIONS

Fillmore bases his development of case grammar on two assumptions. His first assumption is that syntax takes precedence over morphology.[28] He makes this assumption not only to distinguish his work from research focussing on affixes and their use in surface structures, but also because he believes that surface forms of words are derived from syntactic concepts and not the other way around.[29] Even though Fillmore's assumption implies that syntax is central to his study, Cook finds that his deep structures "...are semantic in scope, and acquire syntactic structure through a core set of realization rules which map deep structures into surface structures."[30]

Fillmore's second key assumption is that covert categories are important.[31] Covert categories are, for Fillmore, "...those grammatical properties lacking obvious 'morphemic'

---

27 Hirst 5.
28 Fillmore 3.
29 Fillmore 3.
30 Cook 36, 37.
31 Fillmore 3.

realizations but having a reality that can be observed on the basis of selectional restraints and transformational possibilities."[32] In other words, there are relationships in sentences which are comprehended not visually or orally but in terms of the way the meaning of sentence components affect each other in a sentence. He cites two sentences which, under comparison, demonstrate a covert grammatical distinction:

1. John ruined the table.
2. John built the table.

In (1), the table receives the action and therefore exists prior to the action; in (2) the table is the result of the action.[33] It may be inferred from this example that covert categories[34] are more closely related to the semantic properties of an action than to the syntactic properties of the individual sentence components. This does not, however, diminish the importance of covert categories; in fact, Fillmore assumes their importance and proceeds to characterize them as 'case relationships.'

> I am going to suggest that there are many semantically relevant syntactic relationships involving nouns and the structures that contain them, that these relationships—like those seen in [sentences] 1 and 2—are in large part covert but are nevertheless empirically discoverable, that they form a specific finite set, and that observations made about them will turn out to have considerable cross-linguistic validity. I shall refer to these as 'case' relationships.[35]

It can be noted that assumptions similar to those of Fillmore have been used in developing Ordinotrad. On the one hand, syntax is considered central to sentence analysis, since the application of case labels depends on correct syntactic analysis. Syntactic analysis, in turn, relies somewhat on morphology. Yet it is the semantic aspect of the case grammar principles that ultimately provides direction for the generation module. Furthermore, covert categories—as described above—are assumed essential to the translation process within Ordinotrad. In fact, the finite (tentative) set of covert categories (case relation labels) described below is assumed to be valid for both French and English.[36]

---

32 Fillmore 3.

33 Fillmore 4.

34 In terms of the covert categories or case labels used in this thesis, 'the table' in the first sentence plays the role of 'affected,' and in the second, the role of 'result.'

35 Fillmore 5.

36 This assumption is, of course, tested in every individual translation by the system. Testing the applicability of the case set to other languages is, however, outside the scope of this thesis.

DEEP STRUCTURE VERSUS SURFACE STRUCTURE

Another aspect of linguistics underlying case grammar, and one mentioned by Fillmore, is the distinction between surface structure and deep structure and the corresponding need to distinguish terms used to describe each of these two phenomona. In the context of transformational grammar—which he was attempting to improve—Fillmore questions the validity of using the labels 'subject' and 'predicate', typically used in describing surface structures, to describe a division in the underlying form of all sentences.[37]

Fillmore finds, on the one hand, that the notions 'subject' and 'object' do not have a semantically constant value and, on the other, that if one surface sentence component does evidence some semantically relevant relationship to another component or to the sentence as a whole, this relationship could be expressed by means of another, more precise label. As a result, he states that for purposes of case grammar, all semantically relevant syntactic relationships between noun phrases and the sentences which contain them must be described in terms of these more precise labels with the result that the concepts of 'subject' and 'object' are relegated to surface structures only.[38]

The same surface-deep structure distinction is made by other linguists. Bruce, for example, provides us with the term 'surface case' to describe word groupings in a realized sentence: a surface case is a syntactic category for noun groups based on such things as word endings, word order, and accompanying prepositions.[39] One notable element of this definition is the word 'category,' since it maintains the traditional notion of case as a category. In order to describe deep structure, Bruce uses the term 'deep case' or simply 'case' (in opposition to 'surface case'). His definition of 'deep case,' as we shall note later, refers to a relation between the underlying roles played by sentence components. A key word of this definition is 'relation,' which, when compared with 'category' of surface cases, helps clarify the distinction between deep and surface structures.

---

[37] Fillmore 17.

[38] Fillmore 17. In terms of transformational grammar, this choice also had the effect of eliminating the verb-phrase category and of adding a rule, or system of rules, for determining which noun would become the deep-structure subject (NP1) in the transformational grammar sense.

In developing Ordinotrad, I found that the notion "sentence" too does not have a constant value in terms of an Event and its set of case relations. For example, a sentence may be a "compound sentence" and have two Events. In this thesis, therefore, I have chosen to use the concept of "proposition", where a proposition may be comprised of any number or combination of independent, subordinate or ellipsed propositions, each of them having only one Event. See note 51.

[39] Bruce 338.

Bruce also finds a dependency relation between surface and deep structures, the former being 'epiphenomena of the deep cases,' their number and relationships to each other being dependent on how they are derived from deep cases.[40] In other words, cases or underlying relations require the surface forms of language to mediate them, and surface structures themselves require these underlying relations to give them 'shape.'

Larson offers another perspective on the difference between surface and deep structures. She compares them respectively with 'form' and 'meaning', with 'grammatical, lexical, and phonological structures' and 'semantics,' arguing that analyzing only a surface structure does not provide all the information that can be extracted from a sentence.[41]

It can be generally seen therefore that distinguishing between surface and deep structures provides one way of understanding what constitutes a meaningful sentence and becomes one of the starting points for developing a case grammar.

SOME PRINCIPLES OF CASE GRAMMAR

Up to this point, we have been following the linguists' usage of the terms 'case,' 'case structure,' 'case system' and 'case grammar' in reference to deep structures and relations. Since the purpose of this thesis is to develop a machine-translation system based on the principles of case grammar, it is appropriate that we ourselves define our terms, beginning with the smallest unit, the 'case.'

THE NOTION OF 'CASE'

It is clear from the foregoing that Fillmore abandons the idea of case as an expression in the form of an affix on a noun. However, Fillmore's clearest statement on what he means by the term is borrowed: "I shall adopt the usage first proposed...by Blake (1930) of using the term *case* to identify the underlying syntactic-semantic relationship, and the term *case form* to mean the expression of a case relationship in a particular language—whether through affixation, suppletion, use of clitic particles, or constraints on word order."[42]

---

[40] Bruce 337.

[41] Mildred L. Larson, *Meaning-based Translation: A Guide to Cross-Language Equivalence* (Lanham, New York & London: University Press of America, 1984) 26.

[42] Fillmore 21. The reference to Blake is: Frank Blake, "A semantic analysis of case," *Curme volume of linguistic studies*, James Taft Hartfield, Werner Leopold, & A. J. Friedrich Ziegelschmid, eds. (Baltimore, 1930) 34-49. Identical with *Language Monograph No. 7*.

As opposed to the traditional notion of case as an affix on nouns, which has been mentioned above and which is duly mentioned by Bruce[43] and Anderson,[44] present-day users of cases (mostly in AI) have their own notion of what a case is. Bruce spends some time describing and discounting several of their definitions, which all begin with "A case is a property... [assigned, specified, distinguished, etc., in some way]." Bruce's own notion is: a case is a relation which is 'important' for an Event in the context in which it is described.[45] In the context of Bruce's discussion, 'important' should be understood as 'necessary for the purposes of distinguishing this Event from others.'

Through his notion of case as relation, Bruce supplements the descriptive definition provided by Fillmore: "by 'labeled relation' I mean the relation of an NP to a sentence, or to a VP, which is mediated by a pseudocategory label such as Manner, Extent, Location, Agent."[46] Bruce states:

> A (deep) case is a binary relation which holds between a predicate (usually, but not necessarily, realized as a verb) and one of its arguments. The motivation for deep cases is usually to provide a better account of grammaticality. Thus the links to surface structure need to be explicit and (usually) easily computable.[47]

From a terminological point of view, it seems unfortunate that the term 'case' was chosen since the term already had a historically grounded notion and reality to which it pointed. As such it is confusing unless the distinction between surface and deep levels is clearly made and maintained. Fillmore does raise the issue of whether he is justified in using the term 'case' for the kind of underlying syntactic-semantic relations that are being described (Fillmore 19), yet he chooses to keep it on grounds which could actually support the use of a different term. He states:

> It seems to me that if there are recognizable intrasentence relationships of the types discussed in studies of case systems (whether they are reflected in case affixes or not), that if these relationships can be shown to be comparable across languages, and that if there is some predictive or explanatory use to which assumptions concerning the universality of these relations can be put, then surely there can be no meaningful objection to using the word *case*, in a clearly understood deep-structure sense, to identify these relationships. The dispute on the term *case* loses its force in a linguistics which accepts the centrality of syntax (Fillmore 20).

[43] Bruce 327, 328.

[44] Anderson 9, 10.

[45] Bruce 333-336

[46] Fillmore 16.

[47] Bruce 338. Bruce also mentions the notion of "conceptual case" which allows for the existence of a case which need not be realized in a surface structure.

> A conceptual case, like a deep case, is a binary relation which holds between a predicate and one of its arguments. However, its existence is entirely independent of surface structure considerations. A conceptual case is postulated, not because it is apparent in surface language, but because it is a kind of information about actions which people attempt to communicate (p. 338).

To illustrate, if conceptual cases were implemented in Ordinotrad, they would be used to define "conceptual case frames" for the primitive Events indicated in the ontology (Appendix 2). For example, a conceptual case frame could be defined for the semantic primitive "mental interaction" using conceptual case labels referring to some "thinker" and some "subject matter." With this type of case frame available, it would then

For his part, Anderson takes the best of both worlds in choosing his terms. Like Fillmore, he uses the term 'case-form' to cover any form at the surface level that serves to express or realize a 'case-relation' at a deeper level. The term 'case-relation' is a composite of Fillmore's 'case'[48] and of Bruce's 'relation'[49] and is simply a label he applies to the semantic roles played by particular noun phrases in a sentence.[50] Anderson is to be commended for appending 'form' and 'relation' to the word 'case,' since these make explicit what the terms refer to.

For the purposes of this thesis, the following definition of case combines those of Bruce and Fillmore:

> a case is an underlying semantic-syntactic binary relation holding between a predicate embodying the event[51] of a proposition[52] and one of its arguments (a sentence component with both a syntactic and semantic role).

In other words, 'case' is a label for the relation between two sentence components, with one component always being the *Verb* and the other being a word group which plays a role in relation to the verb (in terms of the deep structures used in Ordinotrad, the same relation would be described as being between two *proposition* components, with one being the *Event* and the other a word group). Despite its terminological shortcoming mentioned above, the term 'case' will be used in this thesis, since it is the standard term, along with 'case role,' 'case relation,' and 'role.'

With this understanding of case as a relation, a choice has to be made of what labels to apply. Fillmore determines his choice of labels (described in detail in the next section) based on the way humans perceive the syntactic-semantic relations found among sentence components:

> The case notions comprise a set of universal, presumably innate, concepts which identify certain types of judgements human beings are capable of making about the events that are going on

---

be theoretically unnecessary to define case frames for all Events (e.g., think, learn, analyze, etc.) which may have been labelled with the "mental interaction" semantic primitive.

48 Fillmore 21.

49 Bruce 338.

50 Anderson 9.

51 Ordinotrad is not yet capable of handling coordinate or disjunctive instances of Events (e.g., John designed and built the table.) However, each independent, subordinate or ellipsed proposition is expected to have only one Event, although a sentence component (e.g., a noun phrase) may itself contain a relative proposition which has its own Event. See the remarks in this regard in the 'Parser' files listed in Appendix 4.

52 See page 60 for a discussion on the distinction between "proposition" and "sentence."

> around them, judgements about such matters as who did it, who it happened to, and what got changed.[53]

The terms traditionally used by linguists to label these judgements usually refer to roles, typically terms like 'Agent' for 'who did it,' 'Object' or 'Affected' or 'Patient' for 'who it happened to,' 'Beneficiary' or 'Recipient' for 'who it happened for,' and so forth. Establishing acceptable labels for these roles and finding the limit to their number is the next problem. The labels prepared by the different case grammarians are grouped together in lists called case lists, and these are examined below.

CASE LISTS AND INDIVIDUAL CASES

A considerable number of case lists for English (applicable to other languages if the cases are considered universal) have been proposed. Three criteria must be kept in mind when evaluating lists of cases: the 'acceptability' of individual labels, the 'adequacy' of individual labels, and the 'exhaustivity' of the list as a whole. The first criterion measures the degree to which a label seems intuitively to reflect the human judgement being made, the second requires that a label be univocal, and the third criterion requires one to ask if all categories of these human judgements have been taken into account. However, given the divergent, dynamic nature of language and the characteristic capriciousness of its users, it would appear difficult to ensure exhaustivity.

Hirst illustrates some of the difficulties encountered in drawing up a case list, especially if one tries to be too specific:

> It has been a matter of some considerable debate as to exactly what the cases of English are. On the one hand, we see many obvious regularities. Many verbs clearly involve a cognitive AGENT that performs an act, and a PATIENT on whom or which the act is performed. ... Other frequent regularities involve DESTINATION-LOCATION and SOURCE-LOCATION ... and EXCHANGE-ELEMENTS. ... But after a while things start to get a little murky, people's intuitions start to differ, and one ends up inventing dubious cases that seem unique to a particular problematic verb...[54]

---

53 Fillmore 24.
54 Hirst 3.

As a first step in considering the case lists proposed to date, one can identify the roles—however labeled—common to them all. Any such study should begin with a look at the first and successive lists drawn up by Fillmore. In 1968 he found the following cases were acceptable and adequate:

> *Agentive* (A), the case of the typically animate perceived instigator of the action identified by the verb.
>
> *Instrumental* (I), the case of the inanimate force or object causally involved in the action or state identified by the verb.
>
> *Dative* (D), the case of the animate being affected by the state or action identified by the verb.
>
> *Factitive* (F), the case of the object or being resulting from the action or state identified by the verb, or understood as part of the meaning of the verb.
>
> *Locative* (L), the case which identifies the location or spatial orientation of the state or action identified by the verb.
>
> *Objective* (O), the semantically most neutral case, the case of anything representable by a noun whose role in the action or state identified by the verb is identified by the semantic interpretation of the verb itself; conceivably the concept should be limited to things which are affected by the action or state identified by the verb. The term is not to be confused with the notion of direct object, nor with the name of the surface case synonymous with accusative.[55]

To illustrate these cases, we can take some of the examples provided by Fillmore in *A Case for Case* and add a few others in order to cover every instance.

1. John ruined the table.
2. John built the table.
3. John opened the door.
4. John opened the door with the key.
5. John opened the door for Mary.
6. The wind opened the door.
7. The door opened.
8. Open the door, please!
9. John opened a way into the next room.

---

55 Fillmore 24, 25.

An Agentive, John, can be recognized in (1), (2), (3), (4), (5), and (9). An Instrumental can be found in (4) and (6), 'the key' and 'the wind' respectively. In (5), Mary fills the Dative role. Sentence (9) illustrates both a Factitive, 'a way,' and a Locative, 'into the next room.' Finally, an Objective can be noticed in all sentences with 'the door.' The distinction between Objective and Factitive is more evident in sentences (1) and (2) respectively: 'the table' in (1) would be Objective, as it is affected by the action identified by the verb, whereas 'the table' in (2) would be Factitive in that it resulted from the action described by the verb.

The difficulty of constructing an exhaustive list of acceptable and adequate labels can be seen in the fact that Fillmore's successive lists (1970,[56] 1971[57]) changed both in labels and in number of labels. The notion of animate instigator remains the same but is labeled as *Agent* in his last list. *Instrumental* is kept in the second list but becomes *Instrument* in the third. The *Dative* of the first list is divided into *Experiencer* and *Benefactive* in the second list, the former affected by the action of the verb and the latter receiving it. In the third list, *Benefactive* becomes *Counter-Agent*, although *Experiencer* remains the same. The *Factitive* of the first list is subsumed by *Objective* in the second list and redefined as *Result* in the third (again in opposition to *Object*, now more broadly defined as the entity that moves or changes or whose position or experience is in consideration[58]). In the second list, *Locative* is distinguished from *Source* and *Goal*, allowing *Locative* to take a more peripheral role (i.e. part of the modality of a sentence). In the third list, it is dropped altogether.

Furthermore, the number of cases in the lists varies between six, ten, and eight, respectively. Entirely new additions in the second list are *Time* (the temporal orientation of the action described by the verb) and *Comitative* (the accompaniment role). Notably they are both dropped in the third, possibly because they, like *Locative*, are peripheral in nature.

Another type of case list, more abstract in its use of less intuitive labels, is offered by Celce-Murcia: *Causal Actant*, *Theme*, *Locus*, *Source*, and *Goal*.[59] Her *Theme* corresponds with Fillmore's *Objective*, her *Causal Actant* with his *Agent* and *Instrument* together, and her *Locus* with his *Locative*.

---

56 Cook 18.
57 Bruce 339. See Table 1, page 47.
58 Bruce 339.
59 Bruce 340.

A more sophisticated list was developed by Grimes for use as a basis for discourse analysis. He groups 'roles' (deep cases) into sets which describe motion and position or changes in state. The first set, called orientation roles, includes: *Object*, *Source*, *Goal*, *Range*, and *Vehicle*; the second set, called process roles, includes: *Patient*, *Material*, *Result*, and *Referent*. He also has three Agentive type roles—*Agent*, *Instrument*, *Force*—and a *Benefactive* role. Despite the fact that he identifies so many cases, Grimes suggests that the maximum number of role relations realized in any one instance of a surface structure sentence is eight.[60]

With such variations in numbers of cases in lists, it may seem a hopeless endeavour to come up with an exhaustive list. The lists described so far were based on what the linguists thought a priori would be required to describe a sentence. However, Cook took a different approach, preferring to start with a corpus from which a list of cases would be induced.

> In a recent analysis of a running text in English which examined a story by Ernest Hemingway, the first 300 clauses (simple sentences) yielded 100 verbs but less than a dozen case frames. These case frames were made up of the propositional cases such as Agent, Experiencer, Benefactive, Objective, and Locative. A series of modal cases also occurred, including Locative, Time, Manner, Cause, Purpose, and Accompaniment.[61]

The interesting part of his analysis is that even though he finds the usual labels acceptable, he has to add a few cases to account for what was in the text. We note, for example, that a distinction has to be made between *Experiencer* and *Benefactive* (as Fillmore did in his second and third lists), two cases which are usually lumped together under *Dative*. He also has two *Locatives*, one which is central and another which is peripheral. *Cause* and *Purpose* are also new introductions, whereas his *Accompaniment* would be the same as Fillmore's *Comitative*.

The same approach, i.e., searching for a more comprehensive case list through a detailed analysis of a large sample of English sentences, was taken by Sparck Jones and Boguraev.[62] Paralleling the criteria of acceptability and adequacy, they state:

---

[60] Bruce 341.

[61] Cook 48.

[62] Karen Sparck Jones & Branimir K. Boguraev, "A Note on a Study of Cases," *Computational Linguistics*, 13 (1987): 65.

> Our study was intended to establish both the justification for each case relation individually, by reference to a range of sentences, and the plausibility of the set of relations as a whole, by reference to a complete set of sentences.[63]

The result of their investigative study was a set of 28 cases[64] which, like Cook's list, maintains several of the usual labels, while adding many more—primarily by splitting up some of the usual labels into more specific ones (e.g., *Time* becomes: *After*, *Before*, *Time-location*, and *Time-span*). We note with interest that they have not provided a label for the concept of product or result, a concept which is not really covered by either of their Mental-object or Object labels as they describe them (see table below).

Sparck Jones and Boguraev also provide labels for relations which may hold between nominals (*Attribute*, *Comparison*, *Possessed-by*, *Reason*, and *State*) and not necessarily between a word group and its predicate. *Comparison*, *Possessed-by*, and *Reason* cannot form part of a case-frame for an Event since they do not assist in distinguishing Events, but simply account for relationships within a word group which, taken as a whole, has a relationship with the Event embodied by the predicate. In comparison, the *Attribute* and *State* cases account for relations between word groups (usually adjectival groups which either rename or modify a nominal), but may do so with or without the use of a linking verb.[65] In the system described in this thesis, these inter-nominal case relations are not implemented, though it would be possible to incorporate them in the analysis, transfer and generation of noun phrases if more semantic control of these processes proves necessary.

Cook also clearly stated that each case notion must be univocal and not 'analogous,' i.e., with complementary underlying notions which can be distinguished from one another. He gives the example of the analogous notion of Instrument which, "...in conjunction with an animate Agent, means 'a physical cause which comes in contact with an object', but with psychological verbs,...is 'the stimulus inducing a psychological event.'"[66] He adds that it may be possible to keep the same label, but then each variation of it in a particular

---

63 Sparck Jones & Boguraev 65.

64 Sparck Jones & Boguraev 66-67.

65 This strongly parallels Larson's 'state proposition' which has two parts, the topic and comment. More exhaustive in her analysis than Sparck Jones & Boguraev, Larson lists possible state relations, including: naming, ownership, location, classification, substance, partitive, depiction, identification, description, kinship role, social role, containership, existence, ambience, and time. Larson 194, 215. All these subtleties are not implemented in Ordinotrad, although they could be used in processing noun phrases (e.g., the 'substance' relation between *bibliothèque* and its head noun *gamme* in *une gamme complète de bibliothèques*).

66 Cook 19.

context would have to be defined[67] and kept in mind when the label is being used. So it would appear less confusing to use different labels.

The concept behind each label is not always the same from linguist to linguist, although there is a good deal of overlapping. It may be helpful at this point to tabulate the various case lists mentioned so far, including the lists provided by Larson and Sparck Jones & Boguraev which will be used as a basis for the case list to be used in this thesis. In the following table, each row contains labels for quasi-synonymous roles. The arrow (⇑) indicates that the notion of that row is subsumed by the label above it within that author's list.

[67] Cook 27.

TABLE 1
COMPARISON OF CASE LISTS

Fillmore 1968	Fillmore 1970	Fillmore 1971	Celce 1972	Grimes 1972	Cook 1978	Larson 1984	Sparck Jones & Boguraev 1987***
Agentive	Agentive	Agent	Causal-Actant	Agent	Agent	Agent	Agent
Instrumental	Instrumental	Instrument	⇑	Instrument		Instrument	Instrument
Dative	Benefactive	CounterAgent		Benefactive	Benefactive	Beneficiary	Recipient
	Experiencer	Experience		Patient	Experiencer	Affected	Subject
Objective	Objective	Object	Theme	Object Material	Objective	⇑	Mental-object Object
Factitive		Result		Result		Resultant	
	Source	Source	Source	Source	Locative**	⇑	Abstract-source Source
	Goal	Goal	Goal	Goal	⇑	Goal	Goal Activity Destination
Locative	Locative		Locus	Range	Locative*	Location	Abstract-location Location
	Time				Time	Time	After Before Time-location Time-span
	Comitative				Accompani-ment	Accompani-ment	Acompani-ment
				Vehicle			
				Referent			Direction
				Force	Cause	Causer	Force
					Manner	Manner	Manner
					Purpose		Abstract-destination
						Measure	Quantity

* This *Locative* colours the sentence as a whole, and is thus peripheral to the cases 'important' to a verb, i.e. needed to distinguish one Event from another.

** This *Locative* combines the spatial source and goal of the Event described by the verb.

*** The list drawn up by Sparck Jones & Boguraev contains other (inter-nominal) case labels without 'equivalents' in the other lists: Attribute, Comparison, Possessed-by, Reason, and State.

One further point needs to be introduced here before we list and define the cases used in this thesis, namely the distinction between central and peripheral cases. In his analysis of labels, Cook finds that some cases are central to a description of relations between sentence components and Events (he uses the terms 'nuclear' and 'diagnostic' to describe them). These are: *Agent*, *Event*, *Instrument*, *Object*, *Source*, and *Goal*. He finds other cases, such as *Locative*, *Time*, *Comitative*, and *Benefactive*, to be peripheral to the analysis of a sentence, for they add only "circumstantial detail of the verbal action."[68] The latter list, the peripheral cases, could be considered as part of the modality of a sentence,[69] in that they add to the meaning of the independent (main) proposition as a whole.[70] In terms of traditional French syntax, they are essentially *circonstancielles:* "Les propositions *circonstancielles* marquent les circonstances de temps, de lieu, de cause, de but, de conséquence, d'opposition (ou de concession), de condition, de comparaison."[71] These cases are not 'important' in that they do not lend themselves to being used as a means for classifying sentence types or verbs. In Table 1, all the cases listed from the *Locative* row and down could be considered as peripheral cases.

The list of cases used in this thesis is based for the most part on Larson's (1984) list,[72] which appears to be the most suitable for our purposes for two reasons: (1) her work being relatively recent, she has been able to make use of all the preceding research on cases and to weigh the advantages and disadvantages of each notion and term; and (2) as a translator and linguist working in a multilingual situation, she has found justification for the quasi-universality of case notions across languages.

In the following definitions for the case labels used in Ordinotrad, each label is described in terms of something which is capable of playing the role identified by the label. No effort is made at this point to specify what kind of thing (or nominalized Event) could fill a role (e.g., to say that an Agent must be a human). Such specificity depends on the individual frame for particular Events where semantic constraints are placed on each role.

---

68 Cook 19.

69 Fillmore 23.

70 In Ordinotrad, the modalities of a sentence are limited to its tense, mood, voice, attitude (affirmative, negative, doubt), and form (declarative, interrogative, imperative). As such they cannot be considered cases, for they do not describe any semantic relationships within a sentence. Cases are identified as either central or peripheral, and central cases may be either required or optional, depending on the particular Event. Peripheral cases are always considered optional.

71 Maurice Grevisse, *Le Bon Usage*, 11th ed. (Paris: Duculot, 1980): 1332.

72 Larson 199-204.

TABLE 2
CASE LIST USED IN ORDINOTRAD

Central Roles

The first part of the case list used in Ordinotrad consists of labels for roles important to the Event in a proposition.

The *Agent* is something which does an action described by an Event; that is, the doer.

The *Instrument* is something used to carry out an action described by an Event.

The *Beneficiary* is something that is advantaged or disadvantaged by an action or state described by an Event.

The *Affected* is something that undergoes, is affected by, or experiences an action described by an Event.[73] It may also be something that is renamed or modified by the Attribute of a state described by an Event.[74]

The *Result* is something which is produced by an action described by an Event.

The *Source* is something considered as the place where an Agent begins an action described by an Event or something considered as the place from which an Affected moves or is moved.

The *Destination* is something considered as the place where an Agent concludes an action described by an Event or something considered as the place to which an Affected moves or is moved.

---

[73] When an action described by an Event is a "mental" or "emotional" interaction (see the taxonomy in Appendix 2), the "Affected" may be something perceived as the focus or target of such action. Since it is proving intuitively awkward to apply the "Affected" label where there is no change, future developments in Ordinotrad should include a new case relation called "Concept" which would be something that is perceived as the focus or target of an interaction described by an Event (e.g., constater, prier). This would also require reconstructing the taxonomical divisions of "Event" into "state," "action," and "interaction" and deleting the node "physical interaction."

[74] For the purposes of programming Ordinotrad, an "Event" is considered to be either a state or non-state, despite the apparent contradiction in terms found in "stative Event." See the taxonomy in Appendix 2.

The *Attribute* is something which renames or modifies the Affected of a state described by an Event.[75]

Peripheral Roles

The second part of the list consists of labels for underlying relations between sentence components and the sentence as a whole. They do not have a specific role in relation to the Event indicated in a sentence but affect how the overall Event and filled case-roles are perceived.

Location identifies the spatial placement of an action or state described by an Event.

Time identifies the temporal placement of an action or state described by an Event. It indicates when an action began or ended, its point in time, or its duration.

Manner is the qualification of the way an action described by an Event was, is being, or will be accomplished.

Cause is something which instigates an action described by an Event but does not do it, an action to be done, or something which is perceived as the end to which an action is directed.[76]

It may be noticed in the above list that the *Subject* mentioned by Sparck Jones & Boguraev has been subsumed by *Affected* since this role in a 'state' situation (i.e., where the Event is a linking verb only) adequately covers *Subject* and since we wish to keep terms used to describe surface structure categories separate from those used to describe deep structure relations.

The *Source* and *Destination* case replace Larson's *Location*, which is considered peripheral. Even though Larson considers *Location* as 'important,' i.e. useful in a case frame for distinguishing Events, she does add that *Location* may be further subdivided into *source location*, *destination location*, and 'location at which an Event is occurring.'[77] What we have done is taken two aspects of her *Location* which may help describe the meaning of

---

[75] An Attribute therefore requires the existence of an Affected.

[76] "Aim" or "Purpose" would also be applicable labels, however I prefer "Cause" as a label since it seems intuitively to be on the same lexical level as the space, time, and manner coordinates.

[77] Larson 202.

an action and given them central roles, calling her 'source location' *Source* and her 'destination location' *Destination*.

For example, the action of the verb *move* requires the semantic roles 'from someplace' and 'to someplace,' i.e. *Source* and *Destination* for its 'transfer' meaning.[78] In comparison, the action of the verb *obtain* only allows for a *Source* role. For both verbs, however, where the action occurs cannot be tied in with the meaning of the verb—the action as a whole could occur by a house, on a boat, in space, and so forth. The *Location* role, therefore, adds to the meaning of the sentence as a whole, and not specifically to the meaning of the verb.

It can also be noticed from the chart that neither Larson nor I have kept the *Object* or *Objective* notion. I have dropped it for two reasons: first, *Beneficiary*, *Affected*, and *Result* adequately cover (and disambiguate) the notion of *Object* as defined in Fillmore (1968); and second, so that there is again no possibility of confusing *Object* with the same term used in describing surface structures.

The following examples taken from the corpus of French sentences used in developing Ordinotrad illustrate some of the above roles.

```
1.   Nous vous remercions de votre lettre.
     Event:       remercier
     Agent:       nous
     Beneficiary: vous
     Cause:       de votre lettre

2.   Nous sommes heureux de vous adresser la documentation
     demandée.
     Event:       être
     Affected:    nous
     Attribute:   heureux
     Cause:       de vous adresser la documentation demandée

3.   de vous adresser la documentation demandée[79]
     Event:       adresser
     Beneficiary: vous
     Affected:    la documentation demandée
```

[78] Note that each intensional meaning of an Event requires its own case frame, which may or may not be identical to those of other meanings. See, for example, the case frames provided for "servir" in the French dictionary listed in Appendix 4.

[79] This example is analyzed as an embedded subordinate proposition (particularly an "Event proposition" as defined in the "Parser - event" file in Appendix 4). It is clear from this example that the rules for analyzing the case structures of propositions are recursive.

4. En parcourant les dépliants ci-joints, vous constaterez que nous fabriquons une gamme complète de bibliothèques et rayonnages.
   Event: constater
   Agent: vous
   Affected: que nous fabriquons une gamme complète de bibliothèques et rayonnages[80]
   Manner: en parcourant les dépliants ci-joints

5. Nous fabriquons une gamme complète de bibliothèques et rayonnages.
   Event: fabriquer
   Agent: nous
   Result: une gamme complète de bibliothèques et rayonnages

6. en parcourant les dépliants ci-joints
   Event: parcourir
   Affected: les dépliants ci-joints

7. Il vous sera donc facile de choisir un modèle en fonction de vos besoins.
   Event: être
   Affected: de choisir un modèle en fonction de vos besoins
   Attribute: facile
   Beneficiary: vous

8. de choisir un modèle en fonction de vos besoins
   Event: choisir
   Affected: un modèle
   Manner: en fonction de vos besoins

9. Tous les modèles illustrés sont livrables dans un délai de huit jours.
   Event: être
   Affected: tous les modèles
   Attribute: livrables
   Time: dans un délai de huit jours

10. Nous assurons nous-mêmes le transport et l'installation.
    Event: assurer
    Agent: nous nous-mêmes[81]
    Affected: le transport et l'installation

---

[80] Note that "constater" is considered to be a "mental" type of action. The "Affected" in this case is what is perceived to be the target of "constater." (See note 73).

[81] "Nous-mêmes" is parsed as a right dislocation of an adjective modifiying the head noun "nous" and is therefore considered as part of the noun phrase filling the "Agent" requirement. The rules defining such right dislocations appear to be based more on the expected translation than on the cognitive structures underlying the French construction. This is one instance of "English" interference discussed in Chapter 5.

```
11.  Nous sommes ainsi en mesure de vous garantir une entière
     satisfaction.
     Event:       être
     Affected:    nous
     Attribute:   en mesure de vous garantir une entière
     satisfaction
     Manner:      ainsi

12.  (en mesure) de vous garantir une entière satisfaction
     Event:       garantir
     Beneficiary: vous
     Affected:    une entière satisfaction
```

On the assumption that the above case list is both acceptable and adequate in terms of the number of cases, the next important case grammar principle to consider is the 'case frame,' which makes direct use of cases and the relations between them. It will then be possible to see how case grammar provides a means for accounting for grammatical and semantic elements of sentences.

CASE FRAMES

Given a list of cases in which each case is a label for the relation between an Event and a sentence component, it is possible to describe every proposition (made up of an Event and the sentence components in relation to it) in a sentence in terms of a case frame. For the moment, however, the formation of a 'case frame,' its notation and its use need to be described. It should be noted that some authors use the term 'case structure' for the same notion.

Simply put, a case frame lists all the cases which must or can be used in relation with a given Event. When applied to a sentence, one finds that the Event is typically expressed by the main verb of the sentence. The case-roles, in turn, are expressed by noun phrases, prepositional phrases, embedded sentences, and so forth. The notation used below to describe a case frame is essentially the same as the notation used in the dictionaries in Ordinotrad. This notation is adapted from that of Fillmore and Anderson.

```
Case Frame for Underlying Relations
     Event [case1, case2, case3, …, casen]
```

```
Corresponding Surface Structure
     verb [sentence component1, sentence component2,
     sentence component3, …, sentence componentn]
```
[82]

Comparing the above generic case frame and its corresponding surface structure, we note that the Event slot is filled with the verb, and that each case slot is filled with the corresponding sentence component.[83] In order to discover a case frame for a given Event, we must analyze sentences whose verbs embody the Event.[84] To illustrate, we can consider various sentences in which the verb 'open' is the main verb and try to establish an appropriate case frame:

```
Surface Structure
     John opened the door.
Filled Case Frame
     open [John, the door]
Possible Case Frame for 'open'
     open [agent, affected]

Surface Structure
     John opened the door with the key.
Filled Case Frame
     open [John, the door, with the key]
Possible Case Frame for 'open'
     open [agent, affected, instrument]

Surface Structure
     John opened the door for Mary.
```

---

[82] The numbers in this notation do not imply that there is one-to-one correspondence, even though the order of the cases in the frame is deemed important.

[83] Note that the order of case labels in the frame is important: the arrangement of case labels will affect the labeling of sentence components, as the case processor in Ordiontorad seeks to satisfy their requirements in a left-to-right manner.

[84] The word-acquisition module of Ordinotrad does not use this inductive approach but relies on the intuition of the user to identify the required and optional case labels for a given Event. It is assumed that the user carries out the appropriate analyses mentally while informing Ordinotrad about case frames. In any event, the dictionaries are subject to subsequent verification and change.

```
Filled Case Frame
     open [John, the door, for Mary]
Possible Case Frame for 'open'
     open [agent, affected, beneficiary]

Surface Structure
     The wind opened the door.
Filled Case Frame
     open [the wind, the door]
Possible Case Frame for 'open'
     open [instrument, affected]

Surface Structure
     The door opened.
Filled Case Frame
     open [the door]
Possible Case Frame for 'open'
     open [affected]
```

Since the meaning of 'open' is the same in all six sentences, we can 'add up' the case frames to find that the general case frame is:

```
open [agent-optional, affected-required, beneficiary-
     optional, instrument-optional]
```[85]

A case role is optional if an understandable, grammatical sentence can be found with that particular case slot filled and if another sensible sentence can also be found with

---

85 cf. Fillmore 27.

It can be argued that the prepositions "with" and "for" in the second and third examples used to produce this case frame do not belong within the slots, as they are only markers. I have left them in simply to streamline the explanation. The reader will note that case frames in the dictionaries (see Appendix 4) include recommended prepositions for each case, although they are there only for the sake of sentence generation. If prepositions were to be separated from the lexical material filling the case role but still shown to be in relation to the case role, we could arrive at the following gener.l case frame for "open":

open [agent-optional-nil, affected-required-nil, beneficiary-optional-for, instrument-optional-with]

that case slot empty. A case role is required if that particular slot must be filled for the sentence to be meaningful. Considering the above example, it would appear that 'open' always takes an *Affected* and until shown otherwise, we will consider this case to be 'required.' Where a particular case role is not indicated in the frame as either required or optional, that role is deemed to be disallowed. In view of this, our case frame for 'open' may be incomplete, as the corpus of sentences on which the case frame is based is quite limited, yet for the given corpus of sentences it is complete.

In light of the foregoing, we could borrow Bruce's definition of 'case frame' (he uses the term 'case structure'): The case structure for a predicate (what we call an Event) is the set of cases allowed for that predicate. Usually the cases are marked in the frame as being either optional or required in the surface structure realization.[86]

With this definition of case frame in mind, it is possible to describe some characteristics of case frames, beginning with their classificatory power. If we were to replace the verb 'open' in the above frame with the variable 'Verb,' we would have a case frame which may be suitable for many verbs.

```
Verb [agent-optional, affected-required, beneficiary-
     optional, instrument-optional]
```

It would appear that verbs like 'break,' 'hit,' 'burn,' 'turn,' 'move,' and 'rotate,' among others, would fit into this frame (taking into account only one structural meaning for each), yet for some of them, it would be incomplete. The verb 'move,' for example, would call for an optional *Source* and *Destination*. Fillmore finds that case frames with their particular pattern of cases contain abbreviated statements called 'frame features' which "indicate the set of case frames into which the given verbs may be inserted. These frame features have the effect of imposing a classification on the verbs of a language."[87]

As mentioned above (pages 33, 34), this classificatory power could be increased through the addition of specific selection constraints to each case role in a frame. In Ordinotrad, the choice of constraints is important since they must be consistent with its understanding of things. This 'understanding of things' is contained in the commonsense inferences it makes about things (see the Taxonomy in Appendix 2 or its implementation in the Appendix 4). For example, we may stipulate that the thing which may be an agent for

---

[86] Bruce 338.

[87] Fillmore 27.

'open' must be sentient (one of the classes of things identified in the taxonomy). This would allow both humans and animals to be agents, but not winds and bombs, since Ordinotrad is able to infer that humans and animals hold membership in the class of sentient things, and that winds and bombs do not. One possibility for the frame for 'open' would then be:

```
open [ agent - optional - sentient,
       affected - required - non_living_thing,
       beneficiary - optional - sentient,
       instrument - optional - physical_thing]
```

This stricter formulation of case frames could also lend itself to identifying personification, figures of speech, and so forth. Where the sentence components of a metaphorical sentence do not match the case roles required for the verb in that sentence, the components that do not 'fit' could be assigned to certain roles following a predetermined order of preference. It would have to be assumed, of course, that the sentence is semantically acceptable. For example, if a case frame could not be directly applied to a sentence, a rule could be made to the effect that if a sentence is an affirmative statement, the first word grouping (generally the surface subject) is to be assigned to the first required case-role, the second group to the second role, and so forth. If there are not enough sentence components to fill all the case-roles, the slots could be filled with an 'understood' label (or inferred from context). Personification would then be inferred where the semantic requirement was 'sentient' and the filler is a 'non_living_thing'; and figures of speech where an expected Event-Agent-Affected pattern is switched around. Despite this potential of case grammar, it is not implemented in Ordinotrad, as it goes well beyond the scope of this work.

A second characteristic of case frames is the dominance of the Event; therefore, it needs to be made clear what may be considered an 'Event.' According to the commonsense understanding of the world used in Ordinotrad (see the taxonomy depicted in Appendix 2), an Event can be a stative or nonstative Event. In comparison, Bruce treats 'Event' only as nonstative, but adds that special types of artificial Events could be introduced: "...events are of fundamental importance and it is often useful to see both

objects and state descriptions as special types of events."[88] Larson, for her part, restricts 'Event' to any verb that is not stative, and thus distinguishes between 'state propositions' and 'Event propositions.' She does not have one term to subsume the notions of both state and Event.

For the purposes of Ordinotrad, an Event will be understood as a concept which subdivides into stative Events and nonstative Events, the former being the category of linking verbs which describe states (to be, to feel (good), etc.) and the latter being action oriented. In contrast to Larson, therefore, we have the possibility of Event propositions which include both state and non-state propositions, and non-Event propositions where there is no Event (generally no verb) and where a nominal is central. Examples of the non-Event propositions abound in business and administrative correspondence: company names, addresses, dates, salutations, and so forth. In Ordinotrad, such non-Event propositions are labelled 'mentions' (see the Parser file) and are processed on a syntactic basis only, since they do not contain an Event for controlling case analysis.

A third characteristic of case frames, described by Hirst, revolves around the relationships between case frames and reality: "cases reflect the structure of reality in that the relationship between the verb and its cases depends on the action described by the verb and the roles that occur in that action."[89] Therefore, cases are proven 'correct' only in reference to reality: "case structures are constrained by reality; a verb cannot take a case inappropriate to its meaning."[90]

Insofar as case structures reflect reality and are constrained by reality, Hirst also finds what could be considered a fourth characteristic of case structures, namely that they are both semantic and syntactic.[91] With regard to the above case frame for 'open,' the frame was determined only from consistent uses of a particular sense of the verb and not from its uses in all its possible senses. In Ordinotrad, this semantic characteristic of case frames is reflected in the fact that a separate case frame is required for each structural meaning of a verb (see the comments in the word-acquisition module in Appendix 4 and the descriptions of data structures in Chapter 4). Furthermore, "case structures interact with and influence the syntactic form of sentences."[92] As briefly noted above, if a prepositional phrase in the surface structure begins with 'for,' we could test to see if the following noun

---

88 Bruce 331.
89 Hirst 4.
90 Hirst 4. Metaphors, as described above, would be the exception.
91 Hirst 4.
92 Hirst 4.

phrase would satisfy the *Beneficiary* or *Purpose* slot. Going in the opposite direction, from deep to surface structure, i.e. in generating a sentence, the syntactic form of a *Beneficiary* may require the preposition 'for,' depending on the case frame. In addition, case frames are based on syntax because the Event is generally expressed by a verb and the roles by word groupings (generally noun phrases, with or without prepositions).

THE SENTENCE IN CASE GRAMMAR

According to the way we have defined it so far, case grammar can be applied only to the sentence as linguistic unit and, with limitations, to noun phrases. Noun phrases are treated in terms of their relationship to the verb; nouns within a noun phrase are treated in relation to the head noun. Since the maximum scope of a case frame is the sentence, it is appropriate to understand how the sentence is treated in case grammar. Fillmore states:

> The sentence in its basic structure consists of a verb and one or more noun phrases, each associated with the verb in a particular case relationship. The 'explanatory' use of this framework resides in the necessary claim that, although there can be compound instances of a single case (through noun phrase conjunction), each case relationship occurs only once in a simple sentence.[93]

The last point, that each case relationship occurs only once in a sentence, is reflected in the non-repetition of cases in the same case frame. The parser allows for compound noun phrases, but the case processor treats them as a unit.

To be more precise or more structured, Fillmore further refines the way a sentence is viewed within case grammar:

> In the basic structure of sentences, then, we find what might be called the 'proposition' (P), a tenseless set of relationships involving verbs and nouns (and embedded sentences, if there are any), separated from what might be called the 'modality' (M) constituent. This latter will include such modalities on the sentence-as-a-whole as negation, tense, mood, and aspect. The exact nature of the modality constituent may be ignored for our purposes. It is likely, however, that certain 'cases' will be directly related to the modality constituent as others are related to the proposition itself, as for example certain temporal adverbs.[94]

---

93 Fillmore 21.
94 Fillmore 23.

The first grammar rule that Fillmore provides therefore is that a sentence is composed of a modality plus a proposition, abbreviated to:

$$S \Rightarrow M + P$$ [95]

Although case grammar rightly focuses on the propositional part of a sentence, a few things need to be stated with regard to the importance of a sentence's modality, Fillmore notwithstanding. Cook simply states that modality has to be brought back into the overall case grammar picture.[96] For Larson, a proposition achieves referential meaning only when its 'illocutionary force' (the 'form' modality in Ordinotrad, i.e., whether a sentence is declarative, interrogative or imperative) is added: i.e., communicated is hampered if the receiver of a proposition does not perceive any signal that the proposition is embodied in a statement, question or command.[97] Newmark, another translator, states "philosophers often point out that a communicative act consists of a modality plus a proposition, and the essence is in the proposition.... To a translator, ...the modality is more important than the proposition."[98] It may be pointless to consider whether modality is more important than the proposition, but if natural language processing, including machine translation, is to succeed, there must be a means for taking into account both the propositional meaning of a sentence and its modality. In Ordinotrad, the modalities of a sentence are considered to be: tense, mood, voice, attitude (affirmative, negative, doubt), and illocutionary form (declarative, interrogative, imperative). As these modalities do not reflect relationships within a proposition but characterize the surface sentence as a whole, they are separated from the propositional components of the sentence before case analysis is carried out.

Case grammar therefore focuses on the proposition of a sentence, in particular on describing its underlying structure. If we consider Larson's view of the proposition, we find that she does not treat the verb as the only possible central element: "A proposition is a...semantic unit consisting of concepts, one of which is central and the others directly

---

95 Fillmore 23, 24. "The arrow notation...should not to be interpreted as meaning that the proposal for case grammar requires an assumption of a left-to-right orientation of the constituent symbols of the rewriting rules." (24)

96 Cook 27.

97 Larson 196.

98 Peter Newmark, *Approaches to Translation* (Oxford: Pergamon Press, 1982) 103.

related to the central concept."[99] If it happens that an Event is the central element, she considers the whole proposition to be an Event proposition; if the focus is on a thing or on an attribute, it is a state proposition.[100] Her reasoning about propositions is consistent with that of case grammar, and therefore requires us to account for her distinction. In this thesis, we treat what she calls state propositions as Event propositions if they contain a stative Event (i.e., a linking verb) and as mentions (noun phrases) if there is no Event.

Fillmore's description of a proposition is as follows: "in their deep structure, the propositional nucleus of sentences in all languages consists of a V [verb] and one or more NP's [noun phrases], each having a separate case relationship to the P [proposition]."[101] The rule for the expansion of the proposition, Fillmore's second rule,[102] then is:

$$P \Rightarrow V + C_1 + C_2 + \ldots + C_n$$ (where each C is a noun in a case relationship)

Despite Fillmore's statement that the proposition consists of a verb and one or more noun phrases, he does not equate noun phrases with cases. He notices that some noun phrases have prepositions to suggest the case, whereas others do not. He therefore adds a third rule whereby the case consists of a Kasus (K) plus a noun phrase, where the K may be zero or a preposition,[103] or affixes in more inflected languages.

$$C \Rightarrow K + NP$$

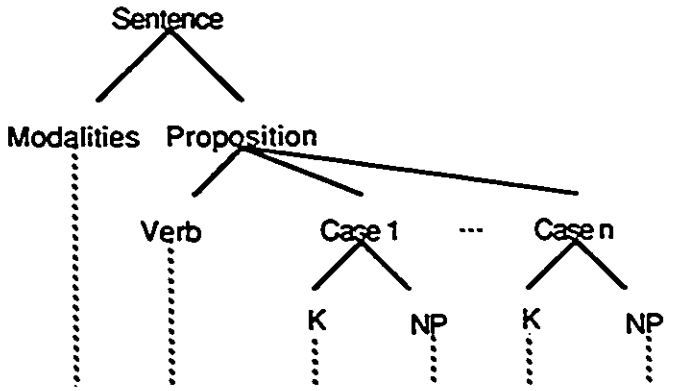
If a sentence (S) is embedded within a noun phrase, as may happen with a relative clause, Fillmore provides a fourth rule to allow a noun to be modified by such an embedded sentence (itself divisible into modality plus proposition, and so on).[104]

$$NP \Rightarrow N(S)$$

99 Larson 189.
100 Larson 191. Larson's state propositions may contain linking verbs.
101 Fillmore 51, 52.
102 Fillmore 24.
103 Fillmore 33.
104 Fillmore 49.

The relationship between all four grammar rules can be seen in Diagram 1. The vertical dotted lines would terminate with the lexical items found in the surface structure, with the possible exception of that leading from M, since some elements of modality do not have morphological representation in the surface structure but are inferred from syntactic arrangements (e.g. the imperative mood in English). More concrete examples are provided in Appendix 3 (see the material following "RESULTS OF CASE PROCESSING").

DIAGRAM 1
TREE SHOWING RELATIONSHIP
OF FILLMORE'S GRAMMAR RULES



According to Simmons, it is possible to elaborate on the definition of a noun phrase. He suggests that the last two rules be replaced by the following written in case rule syntax.[105]

$$NP \Rightarrow (PREP) + (DET) + (ADJ \mid N)^* + N + (S \mid NP)^*$$ [106]

This rule indicates that a noun phrase may consist of an optional preposition followed by an optional determinant followed by zero to several adjectives and/or noun modifiers followed by the noun which dominates the noun phrase. The main noun may

---

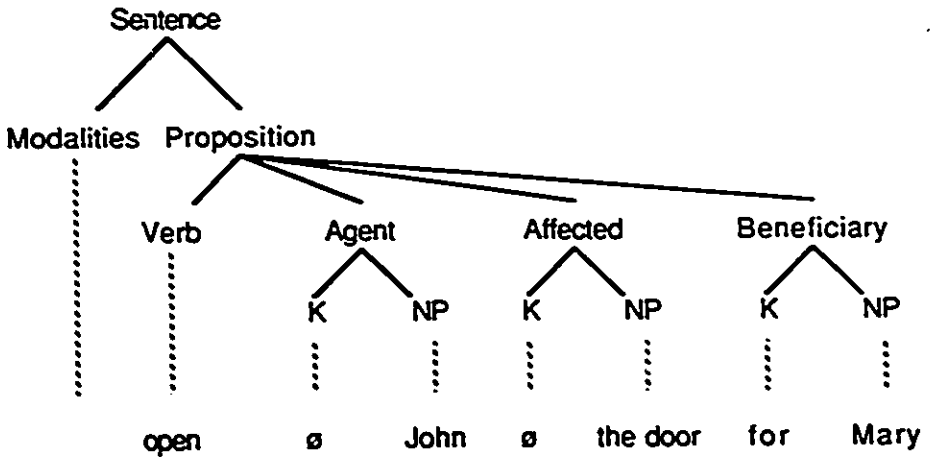
105 Robert F. Simmons, "Semantic Networks: Their Computation and Use for Understanding English Sentences," *Computer Models of Thought and Language*, Roger Schank & Kenneth Kolby, eds. (San Francisco: Freeman and Company, 1973) 63-113. Cited in Mary Dee Harris, *Introduction to Natural Language Processing* (Reston, Virginia: Reston Publishing Company, 1985) 184.

106 The parentheses indicate optionality, the vertical bar indicates "and/or", and the asterisk indicates that a constituent may be repeated.

also be followed by zero to several embedded sentences and/or noun phrases, each in turn subject to the same grammar rule. This rule is somewhat nonstandard in that the preposition is handled with the noun phrase.[107] Even so, the rule provides a clear means for representing a noun phrase which may include embedded sentences.

The following diagram shows how the sentence 'John opened the door for Mary.' could be represented by following these case grammar rules. As the grammar rule for the modality constituent of a sentence has not been identified, it is left blank. The cases of the case frame for 'open' are indicated, but only if there are corresponding lexical items in the surface structure.

DIAGRAM 2
REPRESENTATION OF A SENTENCE
USING CASE GRAMMAR RULES



To conclude this section on the sentence, it can be said that case grammar provides a very structured method for representing some of the syntactic and semantic information contained in a sentence. As will be seen in the following chapters, the structure of this representation lends itself to machine processing of natural language, both in describing

---

107 In Ordinotrad, prepositions are not considered elements of a noun phrase but are part of a prepositional phrase. The corresponding grammar rule is:

PP ⇒ PREP + NP

and representing surface structures and in providing possibilities for generating surface sentences.

CASE GRAMMAR AND GENERATION OF SURFACE STRUCTURES

In a machine-translation setting, the utility of case grammar shows itself not only in describing or representing surface structure sentences but also in generating acceptable ones. For example, if the surface structure of a French sentence were represented using the principles of case grammar, and if the structure were transferred with the appropriate lexical and structural transfers, we would have the deep structure of an as yet unrealized English surface structure sentence (provided we assume that the same case list is common to both languages).

The key problems in sentence generation are choosing the subject, the object and their appropriate prepositions, if any.[108] Fillmore finds that "every English sentence has a surface subject, if only formally so. For most combinations of cases there is a 'preferred' or 'unmarked' subject choice; for some there is no actual choice—the subject is uniquely determined."[109] Fillmore then proposes a rule for determining subjects for sentences in the active voice when they are not marked.[110] We adapt it here to coincide with the case list used in this work.

```
If there is an Agent, it becomes the subject; otherwise,
if there is an Instrument, it becomes the subject;
otherwise, the subject is the Affected or the
Beneficiary or the Result.
```

Cook too finds an order of preference among the cases for subject selection, but he finds that it is not the same for object selection. Using abbreviations for his list of labels (cf. Table 1, page 47), he identifies the subject selection sequence as A - E - I - O - S - G and the object selection hierarchy as E - O - G.[111] Adapting his hierarchies to the case list

---

[108] See the 'GENERATOR - SENTENCE ARRANGEMENT' file in Appendix 4 for how this aspect of generation is dealt with.

[109] Fillmore 33.

[110] Ordinotrad is capable of recognizing French passive constructions and in generating English passive constructions with the appropriate choice of subject. The system currently constructs English passives only when there is a passive in French. Future developments would incorporate the possibility of translating French actives with English passives.

[111] Cook 19.

implemented in this work, our subject selection in active sentences would follow

Agent - Affected - Instrument - (Beneficiary - Result) - Source - Destination

and our object selection

Affected - (Beneficiary - Result) - Destination.

As can be seen, there are similarities between the subject hierarchies suggested by Fillmore and Cook.

Another problem—more so for inflected langauges such as German than for English and French—is the selection of surface cases (affixes) which also signal the underlying case. The corresponding problem in French would be the selection of pronoun forms. Since Fillmore was concerned only with English, he did not treat this problem. Since the generation module of Ordinotrad is designed to produce English sentences, this aspect of sentence generation is likewise not treated in this work.

The final problem is that of determining prepositions. Fillmore states that prepositions are more dependent on structural features at the surface level than on elements of the deep structure. For example, certain prepositions are possible depending on which lexical unit is to become the subject; some verbs require certain prepositions to follow them; some nouns accept only certain prepositions; some constructions accept only certain prepositions, and so forth.[112] Nonetheless, Fillmore does come up with a rather long rule for determining which prepositions are to be used in generating a sentence if they are to be used at all. In considering it, the reader must keep in mind Fillmore's (1968) list of cases.

> The rules for English prepositions may look something like this: the A preposition is *by*; the I proposition is *by* if there is no A, otherwise it is *with*; the O and F prepositions are typically *zero;* the B preposition is *for;* the D preposition is typically *to;* the L and T (for Time) prepositions are either semantically nonempty (in which case they are introduced as optional choices from the lexicon), or they are selected by the particular associated noun [*on the street, at the corner* (=intersection of two streets), *in the corner* (of a room); *on Monday, at noon, in the afternoon*]. Specific verbs may have associated with them certain requirements for preposition choice that are exceptions to the above generalization.[113]

On the whole however, no preposition is exclusively related to only one case. In Ordinotrad, therefore, the frames for English Events suggest prepositions (with allowances

---

112 Fillmore 15.

113 Fillmore 32.

for a zero-preposition) associated with each case role. In this way, there is no need to produce a highly generalized rule such as that proposed by Fillmore, which may be more prone to error than is immediately apparent.

It can be seen then, that in principle case grammar provides a means to control generation of surface structure sentences. The actual process of producing surface sentences is dealt with in detail in the generation module, where the grammar rules implemented for realizing an English sentence are listed and explained.

CONCLUSION

The foregoing discussion on the principles of case grammar has only dealt with the topics usually considered central in a study of cases and case frames. Bruce raises a series of other topics which could also be studied to further discussion on the principles of case grammar.

> One important issue that has not been covered is the relationship between embedded sentences and case structures. What constraints exist on the kinds of embedded sentences? How do these constraints relate to the deep cases of the dominating verb? To the surface case? Do characteristics of higher level sentences have consequences on case determination in an embedded sentence? A related issue is the role of cases in the general inference problem. Is a case representation adequate for the kinds of inferences needed in natural language understanding? In what way does it facilitate inferences? Other issues include the specification of surface cases, the relationship of case systems to discourse analysis, and the selection of an ideal case system.[114]

It appears the list could go on, yet the area of extending case grammar to textual levels beyond the sentence to discourse analysis, would seem to be the most fascinating. At that level, one could surmise a "discourse Event" (an Episode, perhaps) and "discourse cases"[115] which would show the relationship of the various parts of a discourse to the so-called "Event" or even to each other.

Hirst too raises another consequential topic, namely that there is a need for lexicographers to compile dictionaries that incorporate case frames into their entries for

---

114 Bruce 357.

115 Bruce (341) indicates that Grimes has developed a case system to serve as a foundation for discourse analysis. Grime's case labels, however, reflect the centrality of the Event and are therefore limited to the level of proposition.

verbs.[116] With such lexicographical works, system developers making use of case grammar would not need to create their own abridged dictionaries to meet programming needs. Despite the present limitations of case grammar, the application of case grammar principles to natural-language processing, exemplified in the demanding area of machine translation, does hold promise, as we shall see in the next chapter.

116 Hirst 9.

Chapter 3

Case Grammar and Machine Translation

OVERVIEW

According to the 1985 Cognos report on the prospects for machine translation and natural language processing in Canada, new discoveries of basic principles in fields such as formal linguistics are making successful mechanization of translation processes more possible.[1] The same report later identifies one of these linguistic principles:

> One of the most important functions that must be carried out by any language understanding system is the analysis of sentences into their constituent parts and the determination of the '*thematic roles*' of each constituent (i.e., to determine "*who did what to whom*").[2] [emphasis added]

Although the report does not identify it as such, the linguistic analysis of sentences into constituent parts and labeling them in terms of 'thematic roles' is precisely the type of semantic analysis provided through case grammar, where the term 'case role' is used instead of 'thematic role.' Other authors, too, have indicated that case grammar has made developments in machine translation possible. Hutchins, a machine-translation historian, makes it clear that case grammar has clearly established itself in natural language processing, particularly in Japanese endeavours in the field of machine translation.

> The most popular type of semantic analysis has been the use of 'case frame' analysis, adopted in many systems ('direct' as well as 'transfer') during the 1970s and now established as a standard proven technique. ... Although originally formulated in the context of generative grammar, cases and case frames have been applied in many different contexts; they are particularly popular in

---

[1] Cognos Incorporated, *Machine Translation and Natural Language Processing: Opportunities for artificial intelligence in Canada*, Cognos report DOC-CR-84-048, Ottawa, 1985, 7. This report was prepared by Zenon Pylyshyn and John Shepherd.

[2] Cognos, A-7.

> Japanese MT projects and in AI approaches, where cases are frequently basic constituents of semantic grammars.[3]

Indeed, the use of case grammar in natural language processing has helped to advance another avenue of research apart from machine translation, though no less interesting or demanding, namely that of processing spoken language. According to researchers in this specialized field, the semantic analysis allowed by the use of case frames makes it possible to process "sensible but potentially imperfect or incomplete utterances"[4] which, as most translators would agree, is often the case in translation as well. The main advantage speech-processing researchers find in using case grammar is, as will be shown later, comparable to the one provided to machine-translation system developers, namely the abstract way of encoding a crucial part of the information contained in a sentence and information about the sentence which, in turn, is easily interpreted by a machine.[5] In other words, "the caseframes tell us ***what*** components to look for and constrain ***where*** we can look for them. But exactly ***how*** we look for them is adaptable so that it can be driven by the most reliable information we have."[6]

Some major machine-translation projects are presently using case grammar in their translation processes. Before considering them individually, it is appropriate to discuss the general applicability of case grammar to machine translation.

As was indicated in Chapter 2, case grammar and its implementation of deep cases or case relations are useful in assessing sentences in terms of their semantic acceptability as well as syntactic correctness.

> Deep cases are useful both in accounting for the relative "acceptability" of certain sentences and in explaining how an intelligent system might understand language. This explanatory power is derived from a focus on (conceptual) events rather than on syntactic constructions.[7]

The evaluative function provided by case grammar has an important role, since in some systems (e.g., MÉTÉO) the process of translating will not be attempted if the input

---

[3] W. John Hutchins, *Machine Translation: Past, Present, Future* (Chichester: Ellis Horwood Limited, 1986) 187-188.

[4] Philip J. Hayes, Alexander G. Hauptmann, Jaime G. Carbonnell, & Masaru Tomita, "Parsing spoken language: a semantic caseframe approach," *Proceedings of COLING 86* (Bonn, 1986): 588.

[5] Hayes *et al*, 588.

[6] Hayes *et al*, 589.

[7] Bertram Bruce, "Case systems for natural language," *Artificial Intelligence* 6 (1975): 329.

sentence is not compatible with what is expected by the system or if the system cannot completely analyze the sentence. If the parser fails to parse a sentence or if the case analysis fails to apply appropriate labels to sentence components, the sentence must be either (a) rejected outright, (b) forwarded partially analyzed to the remaining procedures of the translation system as it is, in the hope that the error would be corrected by later processes acting upon the sentence, (c) patched up by the parser based on expected structures and then passed on to the remainder of the procedures, or (d) translated partially, with the translation of the rest of the sentence left for a revisor. Case grammar can theoretically handle some of these problems, as it has a predictive capability by virtue of its case frames, and since individual case slots can be assigned default values (e.g. assuming the semantic requirements of that slot are 'understood,' in the same way the 'you' is understood in imperative constructions).[8]

Another way in which case grammar principles may be incorporated into a machine-translation system is simply through representing information. According to Bruce, "much of the research in natural language processing has focussed on the problem of storage structures, or the question, 'How is the information or meaning of a sentence to be represented once a sentence has been parsed?' Several recent systems use structures based on deep case relations. Some systems which do not explicitly use a case structure have nevertheless used case-like mechanisms."[9] The data structures used in Ordinotrad are based on case grammar principles and reflect the assumption that the meaning of a sentence is structured in this way. As was seen in Chapter 2, case grammar affords a mixture of two types of structures: there is a network of relations among the event and sentence components and there are hierarchical relations within sentence components (e.g., a noun dominates a noun phrase).

Another assumption underlying the use of case grammar principles in machine translation is that structured meaning is 'underlying' or 'deeper' than the string of words or 'surface structure' of a sentence. What machine-translation system developers require, then, is a system which readily handles this 'deep' aspect of meaning. It is generally accepted that the 'deeper' the system can go in analyzing the sentence, the more likely the system is to discover the underlying meaning correctly. In the Cognos report mentioned above, the relative values of 'deep' and 'shallow' systems are compared.

---

[8] For example, in sentences with ellipsed subjects, the parser in Ordinotrad supplies an undefined pronoun whose person and number are determined by analysis of the verb.

[9] Bruce 345.

> A 'shallow' system can be considered to be one which is able to achieve some level of performance in the limited domain by using various ad hoc 'tricks,' whereas a 'deep' system is based on a more complete causal model of the process in which it is an expert. The idea here is that as the task becomes more and more difficult, the complexity of shallow programs will go up exponentially, while that of deep systems will increase more gradually. It is now widely accepted that generality in MT can only be attained by developing 'deep' systems.[10]

Although case grammar is primarily construed as a technique for developing 'deep systems,' it is also used in shallow systems. Somers finds that case grammar "...exactly meets the need in MT for a level of representation that is both deep enough to serve in a manner relatively independent of surface form, while still being shallow enough to allow a fairly straightforward mapping from and onto these other levels of description (i.e., surface syntax, canonical form, etc.)."[11] The idea behind shallow systems, despite their risk of exponential growth of complexity, is to represent the meaning of a sentence without losing sight of source-language syntactic structures and information which may constrain the choice and arrangement of target-language lexical items and sentence structures. In fact, two system developers, Netter and Wedekind, have found a need for both syntactic and semantic representations.

> ...a purely semantic representation, reducing the sentence to its truth conditions, is too weak on the one hand—as one might like to draw on information about the syntactic form of the source expression (e.g., passive, relative clauses, topic/focus structure, etc.)—and too strong on the other hand as ambiguities of scope that are not reflected in the surface string will normally not have to be taken into consideration.[12]

Nonetheless, the value of case grammar goes further than just providing a good means to represent semantic and syntactic information. If one considers the generally accepted fact that *meaning* (i.e., the *message*) is the constant element shared by both source and target language texts, one would expect machine-translation system developers to look for linguistic structures capable of sustaining unchanged meaning even though lexical items and surface grammatical structures are replaced. In other words, system developers using

---

10 Cognos, 12.

11 H. L. Somers, "The need for MT-oriented versions of Case and Valency in MT," *Proceedings of COLING 86* (Bonn, 1986): 120.

12 Klaus Netter & Jürgen Wedekind, "An LFG-based Approach to Machine Translation," *Proceedings of IAI—MT86*, Tom C. Gerhardt, ed. (1986): 199.

case grammar need not work on an arbitrary ad hoc basis when transferring meaning, as transfer is already structurally defined through the use of case labels which provide "...a non-arbitrary transfer medium which is intuitive in both languages."[13] The value of case grammar would, however, be weakened or even nullified to the extent that case roles are not shared across languages.

Case grammar in fact is used in several machine-translation systems, due to the fact that it provides this bridge between syntax and semantics.[14] As mentioned above, the machine translation development projects in Japan are the ones making the most use of case grammar. "One of the major influences in MT system design [in Japan] has been the case grammar model of Fillmore...."[15]

As we shall see below, there are also non-Japanese systems making use of case grammar. Although the general reasons why they have adopted case grammar are those mentioned in the foregoing, each system incorporates the principles in its own way. The individual implementations in several systems will be considered in the following section.

SYSTEMS USING CASE GRAMMAR PRINCIPLES

ATLAS

The ATLAS project at Fujitsu (Japan) involves the English-Japanese and Japanese-English language pairs and is restricted to the sublanguage of computer manuals. Its design applies knowledge representation and inference mechanisms through the use of case frame analyses of source language text.[16] An overview of the procedures followed by the first of the ATLAS systems (ATLAS/I) is provided by Hutchins.

> The ATLAS/I is designed as a practical system translating a limited range of materials from Japanese into English. ... The translation process goes through the following stages. 'Preprocessing' segments the Japanese text on the basis of punctuation marks; next comes consultation of the SL-TL dictionaries.... Then follows 'Syntactic analysis' (*identifying noun phrases and their relations*), 'Surface case analysis' (involving nouns and their postpositions),

---

13 Somers, 120.
14 Somers, 120.
15 Hutchins, 314.
16 Hutchins, 316. Cf. Cognos, B-6.

> 'Deep case analysis' (*on the basis of patterns of verb case frames and 'surface' cases*), 'Structural translation into English' (via a Japanese-English pattern translation table), 'Synthesis of English sentence' (invoked by the selected pattern, and including the insertion of prepositions, articles, and plural endings), and finally 'Morphological synthesis'.
> A distinctive feature of the system is the use of a uniform representation of case frames, syntactic conditions, and English synthesis rules. This 'frame knowledge representation' (based on AI ideas) facilitates the expansibility of the system.[17]

ATLAS/II, for the Japanese-German language pair is used in conjunction with another system called SEMSYN which then makes use of case grammar principles to generate German text.

> The project SEMSYN (SEMSYN is an acronym for SEMantic SYNthesis) produced a machine translation system that is unique in some sense. This uniqueness does not only originate from the language pair Japanese to German that is dealt with, but as well from the approach that was taken in the project.
> If one has a close look, SEMSYN is only a subsystem. Only in conjunction with the ATLAS/II-System of the Japanese cooperation partner FUJITSU do we get a complete Japanese to German translation. Interface of the subsystems is a semantic representation that should reflect the content of the Japanese input.
> The analysis of Japanese input—currently at most titles of scientific papers from the field of information technology—and its transformation into the semantic representation is the task of ATLAS/II. SEMSYN's part is to produce a correct and understandable German text for these semantic representations.[18]

The developers further indicate that "the main part of the generation starts from the frame description derived from FUJITSU's semantic nets,"[19] i.e., from the semantic representation reflecting the content of the Japanese input, as mentioned above. They also distinguish between frames for verb concepts (or actions), frames for noun concepts, and frames for relations between noun concepts. By implementing these frames and various

---

17 Hutchins, 316.

18 Martin Emele, Walter Kehl, & Dietmar Rösner, "Extended Demonstration of SEMSYN," *Proceedings of IAI—MT86*, Tom C. Gerhardt, ed. (1986): 68.

19 Emele, Kehl, & Rösner, 70.

case labels (agent, object, method, instrument, goal, ...) they achieve semantic representations.[20]

In general, the system developers have found the advantage provided by the use of case frames to be the fact that "...translation gets based upon the content of the original text."[21]

LUTE

The LUTE project (Language Understander, Translator & Editor) at the Musashino Research Laboratories of the Nippon Telegraph and Telephone Corporation (Japan) is another machine-translation system incorporating case grammar. Hutchins' brief overview of the system shows that case frames are mapped onto source-language lexical information, transferred to the corresponding target-language frames, and then 'unmapped.'

> The linguistic core of the system is an extensive case frame analysis of Japanese and English, and of contextual "world knowledge." Text processing of SL input involves the matching of case frames and knowledge representations against lexical information. The result is a structure of semantic and syntactic dependencies. Transfer from the SL to TL representations is by pattern matching and frame manipulation. The final generation of TL texts is envisaged as simple linearisation of TL frame structures. The distinctive feature of LUTE are the extension of case frame structures to tense, aspect, modality, and semantic implicatures...[22]

The mapping process involving case frames is hierarchically organized, building the 'meaning structure of a sentence' by integrating both syntactic and semantic analysis. The process is really one of identification whereby a known semantic structure or case frame is found which fits the expression under analysis, and from there the words are analyzed for word meaning, word meanings are analyzed for word concepts, and the relations between concepts are identified.[23]

---

20 Emele, Kehl, & Rösner, 70-71.

21 Emele, Kehl, & Rösner, 77.

22 Hutchins, 316-317.

23 Hirosato Nomura, Shozo Naito, Yasuhiro Katagiri, & Akira Shimazu, "Translation by Understanding: A Machine Translation System LUTE," *Proceedings of COLING 86*, (Bonn, 1986): 623. See also Larson's treatment of *concepts*.

The system developers found that the traditional Fillmorean structure consisting of relations between nouns and verbs was insufficient and therefore developed what they call an 'Extended Case Structure Model' (ECS) in order to represent structures of "real sentences which sometimes have complex noun phrases, compound sentences, etc."[24] They also found a need for ways to handle inter-nominal and inter-event relations.[25]

In contrast to the ATLAS/SEMSYN system described earlier, the developers of LUTE indicate that the verb is central to the entire understanding process: "...verb selection is thought to be the key in determining meaning, and, as such, is a crucial factor in translating between such languages having a very different structures as English and Japanese."[26]

METAL

The most extensive use of case grammar in a machine translation system is that of the Siemens Linguistics Research Center's METAL system, where case frames are used in both analysis and transfer of meaning.

> The system is basically transfer-based, with the results of a context-free parse of the source text passed to the 'case frame processor'. Case frames are also used in transfer. The cases in the system are divided into two types, 'central' and 'peripheral', corresponding roughly to the traditional Valency distinction between complements and adjuncts respectively.
>
> A key aspect of the METAL conception of central and peripheral cases is that these form two discrete sets of cases. This means that each of the cases that are recognized is regarded as either typically "conditioned by the particular verb" or not verb-specific.[27]

It can be noted that the central-peripheral distinction made in the METAL system corresponds with the division made in the case list drawn up for this project. Furthermore, the detailed manner in which case frames are used in analyzing meaning closely follows the Fillmorean methodology of identifying the case frame for the given verb and then assigning the noun phrases (here called 'arguments') to case roles. A by-product of this analysis is

---

24 Nomura, *et al*, 622.

25 Nomura, *et al*, 622.

26 Hirosato Nomura & Hitoshi Iida, "English-Japanese Interactive Translation System: LUTE-AID," *Proceedings of IAI—MT86*, Tom C. Gerhardt, ed. (1986): 132.

27 Somers, 121.

the evaluation of the acceptability of the sentence as described in the first part of this chapter.

> The case frame processor attempts to 'use up' the available arguments by matching them to the specifications given in the lexical entry for the verb, and then applies general 'case functions' associated with the peripheral cases to the remaining arguments. If all the arguments can be legally assigned case roles, then the clause is considered well-formed.[28]

In the transfer process, case frames are used primarily in arranging the selected target language lexical items and assigning them appropriate surface-syntactic function markers.[29] Viewed overall, it appears that case grammar is the linguistic key to the METAL system. In fact, Hutchins states that theoretical linguistics and particularly 'case-frame analysis' were considered early in the design of the METAL system.[30]

MU PROJECT

The national machine-translation project of the Japanese government, the Mu-project, was begun in April 1982 and includes the development of a 'transfer' machine-translation system for Japanese-English and English-Japanese translation.[31] As the system follows the transfer model (analysis ⇒ SL representation ⇒ transfer ⇒ TL representation ⇒ generation), the meaning which is passed through the system is, at one point, carried by the deep structure of the source-language text, and at another point by that of the target-language text. Both deep structures are based on case grammar analyses.[32]

The analysis segment of the model consists of morphological and syntactical analysis, with a corresponding target-language morphological and syntactical generation.[33] The Mu system developers also indicate that case grammar is used because it places emphasis on underlying relationships, which facilitates the production of good translations, as opposed to placing emphasis on word order.

---

28 Somers, 121. Cf. Hutchins, 251.

29 Somers, 121.

30 Hutchins, 248-249.

31 Hutchins, 319.

32 Hutchins, 320.

33 Yoshiyuki Sakamoto, Tacko Ariga, & Atsushi Washio, "Unified Operation Control System for Mu Machine Translation," *Proceedings of IAI—MT86*, Tom C. Gerhardt, ed. (1986): 116.

> The Japanese language, unlike English, French, German, etc., does not have strict word order. To analyze the syntax of Japanese sentences, therefore, this system employs not the syntactic grammar that places emphasis on word order but the case grammar that analyzes the syntax of sentences based on the cases of verbs and other words (e.g., agent, object, means).[34]

Thirty-four case labels are used for case frames in the Mu-project, including a catch-all, called ETC.[35] There is, however, no indication in the documentation that a distinction is made between central and peripheral cases.

SYSTRAN

The Systran system, whose roots go back to the 1950s, was the first machine-translation systems to be commercially marketed and is presently used most notably by the United States Air Force, the Commission of the European Communities,[36] and NATO.[37] The translation processes used in Systran do not, according to the literature, make as direct a use of case grammar principles as do the systems described above, although there are indirect references to it. For example, according to Knowles, the structure of the Systran master stem dictionary contains a subpart (the Second Grammar Subpart) comprised of 'Fillmore-type' case information/'valency,' including case requirements, prepositional requirements, and surface syntactic function codes.[38] Since the system design was established in the early sixties, the elements of case grammar could not have been incorporated at the start. Nevertheless, the fact that case grammar principles were later inserted in an ad hoc fashion does indicate their utility in text analysis. (Cf. the implication of linguistic developments for machine translation, as mentioned in the first paragraph of this chapter.)

A recent report on the design of Systran provides us with an overview and shows at what points case grammar comes into play. The program flow of Systran begins with analysis which is comprised mainly of dictionary lookup and actual parsing. Parsing, in

---

34 Sakamoto, *et al*, 116.

35 Yoshiyuki Sakamoto, Tetsuya Ishikawa, & Masayuki Satoh, "Concept and Structure of Semantic Markers for Machine Translation in Mu-Project," *Proceedings of COLING 86*, (Bonn, 1986): 14.

36 Hutchins, 209-210.

37 Kyoko Ishizuka & Masayuki Miyazawa -- IP2010. *NEWSBYTES-JAPAN*. The Source. Week of November 25, 1986.

38 F. Knowles, "Error Analysis of Systran Output—A Suggested Criterion for the 'Internal' Evaluation of Translation Quality and a Possible Corrective for System Design," *Translating and the Computer*, B. M. Snell, ed. (Amsterdam: North Holland Pub. Co., 1979): 117.

turn, is subdivided into smaller steps: definition of the clause boundaries, establishment of basis syntactic relationships between words, establishment of enumerations, search for the main subject, for a predicate, and establishment of deep relationships and preposition government. With regard to the last of these substeps, the developers indicate there are three parts, the first involving case grammar analysis, the second establishing number and gender of ambiguous words, and the third finding the governing word of prepositions.[39]

The procedure for implementing case roles, however, is not the same as in the systems described above: instead of assigning case roles based on the meaning of words, the roles in Systran are 'syntactic relationships' generalized into 'logical relationships' such as *action*, *agent of action*, and *object of action*.[40] In the Systran system then, the principles of case grammar are not really implemented, even though its use of the term 'role' suggests analysis based on relations between meaning components.

CONCLUSION

According to Hutchins, there are many small-scale projects under way in Japan and elsewhere which are investigating the use of case frames in machine translation.[41] A typical non-Japanese example found in recent machine-translation literature is the TRANSLATOR machine translation project at the Colgate University, where use is made of conceptual case frames (generally for linking verbs) and of process case frames (for action-oriented events). The cases identified for conceptual case frames are *patient* and *properties*; for the process case frames, mention is made of *agent*, *object*, *instrument*, *source*, *destination*, *preconditions*, and *effects*.[42] These small-scale projects may not have highly-developed designs constructed around the principles of case grammar, but they do reflect the popularity of case grammar in machine-translation research. This popularity is not unjustified, for although case grammar is not complex in itself—thus making it adaptable to almost any machine-translation design—it has been shown to be a tool bridging syntax and semantics and, therefore, useful in analyzing the meaning of a source

---

39 Cay-Holger Stoll, "The SYSTRAN System," *Proceedings of IAI—MT86*, Tom C. Gerhardt, ed. (1986): 8-14.

40 Stoll, 14.

41 Hutchins, 319.

42 Sergei Nirenburg, Victor Raskin & Allen Tucker, "On knowledge-based machine translation," *Proceedings of COLING 86*, (Bonn, 1986): 628-629.

language text, sustaining that meaning through transfer, and facilitating generation of a target language text.[43]

[43] The heavy reliance on Hutchins in this chapter is due to the lack of first-hand materials. The difficulty in assessing existing systems (or systems under development) resides in the lack of published material describing the systems in detail. COLING papers are generally too brief too give much insight, and the major survey works (e.g., Hutchins) are too broad to provide sufficient detail. However, since the focus of this thesis is to build a machine-translation system and not to assess existing ones, I have simply provided this brief synopsis.

Chapter 4

Using Ordinotrad

INTRODUCTION

Chapter 4 is primarily a technical chapter, providing a practical description of data structures (how they are defined and manipulated), running the system, and the interdependencies of the various modules. These three topics are discussed and illustrated concurrently as the reader is led through a session with Ordinotrad which weaves together running the modules and recognizing the various stages in the process.[1]

It is expected that the user has some familiarity with the Macintosh working environment: handling windows, double clicking, dialog boxes, and so forth. Explanations here deal only with the interface used to run Ordinotrad.

USER'S GUIDE TO ORDINOTRAD

HARDWARE REQUIREMENTS

System development began on a 512K Macintosh computer. A Macintosh Plus with 1 megabyte of memory was later required, as an "out of memory" message appeared far too often with the 512K computer, and often left the user hanging. The Macintosh Plus was sufficient for the most of the development. In fact, the entire system was once running effectively on a Macintosh Plus. However, subsequent improvements required the addition of more rules which seemed to contain the proverbial "straw that broke the camel's back." The system now works best on a Macintosh with at least 2 megabytes of RAM. Two 800K floppy drives are needed, and an Imagewriter printer is optional.

It appears that the Prolog application from AAIS itself requires approximately 385K, and the Macintosh operating system another 300K, approximately. The program in memory, when dumped to a file on disk, only measures 150K, although its size once compiled and stored in memory is not known. This leaves little room in a 1-megabyte memory for Prolog to store its goal stacks.

The dictionaries consume the largest block of memory. Once an efficient means is found to access dictionaries directly from disk and from within the Prolog environment,

[1] The reader is referred to Chapter 1 for an overview of the design and organization of Ordinotrad.

without loading them into memory, there would be much more room available for developing the French parser and English generator. Perhaps the highly touted Hypercard application for the Macintosh could be used as a dictionary management system and feed directly into Ordinotrad when called upon. In any event, a useful, general dictionary of 50 000 entries for French alone could not be stored in memory. Any serious development of the system would require development of a non-memory-resident dictionary.

A SESSION WITH ORDINOTRAD

The following description of working with Ordintrad assumes that the user has the following disk setup: on a disk with the title "ORDINOTRAD" are the AAIS Prolog application and AAIS executive files called "dcg.p" and "lib1.p",[2] a Macintosh operating system, and the folder of program code called "Ordino Code"; on a disk with the title "DATA" are a "corpora" folder containing any text files to be translated and a "Dictionary" folder containing the dictionaries used by Ordinotrad. Note that the disks must have these names, as rules involving file input and output will look only on disks with these names.[3]

With the Macintosh on, insert the disk titled "ORDINOTRAD" in the internal drive and the disk titled "DATA" in the external drive. When in the Finder, find and double click on the file called "User Interface" (in the Ordino Code folder on the ORDINOTRAD disk). Opening this file will cause all the files required by Ordinotrad to be loaded into memory. It should be noted that the whole program has been partitioned into intuitive segments that I call modules (e.g., morphological analysis, word-acquisition, etc.) and which have been written into separate files. The program is not truly modular, however, for the entire contents of the program could have been put in a single file. Furthermore, once Ordinotrad is loaded into memory, all rules can be called directly without specifying any package.[4]

---

2 The "dcg.p" file contains Prolog rules that read definite clause grammar rules, re-write them in Prolog format and assert them. The "lib1.p" file contains some rules for Edinburgh compatibility and what appears to be afterthoughts in the development of AAIS Prolog that didn't make it into the application on time.

Note that the latter file is automatically read when starting AAIS Prolog, so that putting

```
:- consult('ORDINOTRAD:Ordino Code:User Interface').
```

at the end of it effectively starts up Ordinotrad if the AAIS application were opened. If the application is set to open automatically upon starting up the Macintosh, the first user involvement (once the disks are inserted) would be to answer the sessions introductory questions.

3 The rules can, of course, be changed in the event other names are more desireable or if a hard drive is used. Changes would be made in the following files: User Interface, Translate2disk, Word-Acquisition.

4 AAIS Prolog allows for separating modules of a program using packages that can inherit or ignore rules defined in other packages.

You will soon see the Prolog "Query" window and an indication that the files are being loaded. Wait (a few minutes) until you are asked questions concerning the word-acquisition module, translate-to-disk feature, and the viewing of interim results.

The following pages contain a verbatim copy of a sample session using Ordinotrad. This material is printed in a non-serif font. Input provided by the user is shown in boldface in the non-serif font as well. Explanatory comments that are not part of the session are in the present font.

Starting Ordinotrad

ORDINOTRAD™
© 1987, 1988 Victor Loewen

Loading Ordinotrad, a French-English machine-translation system. Please stand by.

Loading the program took 3 minutes and 33 seconds.

For this session, do you wish to have Ordinotrad acquire French and English words presently unknown to it?
[yes,y,no,n] ==> **y**
Word-acquisition module is enabled.

The word-acquisition module has been enabled, even though the user knows that all the words in the sentence were recognizable by Ordinotrad. If there is a shortage of memory, it may be better not to enable the word-acquisition module. Otherwise, it is proper to enable it, for Ordinotrad is always "eager to learn."

For this session, do you wish to have the translate to disk facility turned on?
[yes,y,no,n] ==> **y**
t2d predicate enabled.
To have a file translated with the results saved on disk, type "t2d(Filename)."

It is not necessary to enable the translate to disk module (t2d predicate), if a translation is to be displayed on screen only. It has been enabled here for the purpose of illustrating its use below.

For this session, do you wish to view interim results at key points in processing?

```
[y,n] ==> y
Tracing is enabled.
```

With tracing enabled, the results of the various procedures of the process may be viewed in the Query window. Ordinotrad stops after every display and waits for a signal to continue. With tracing disabled, only brief messages will appear indicating the status of processing followed by the translation of the input sentence.

```
There will eventually be a translate menu. The following are now available.
To have a sentence translated with the results viewed on the screen, type "translate."
To have a file translated with the results viewed on the screen, type "translate(Filename)."
 • Files to be translated must be located in the corpora folder.
 To quit, select QUIT under the FILE menu or simply type "exit." and a return.
?-
```

All user interaction with Ordinotrad takes place in the Query window which keeps a transcript of all activity (up to a specified number of lines, which can be changed). A question mark followed by a hyphen is the prompt supplied by AAIS Prolog. When this is the last prompt in the window, Ordinotrad is not processing data, and the user has full control over what happens next in the Prolog environment.

All user interaction controlled by Ordinotrad uses the "==>" prompt. After you answer these prompts, it is Ordinotrad which determines what happens next.

Ordinotrad therefore controls loading itself into memory and then returns control to the user. To begin, one of the calls described on the screen must be entered. To begin a session identical to the one described here, type "translate." and a carriage return. Note the final period, which is required by Prolog.

Get French Sentence

```
?- translate.

Enter French-language sentence to be translated.
==>
```

When you see the above prompt, type in the sentence as shown below. Do not type a carriage return until the entire sentence and final punctuation have been typed in. The

window will shift back and forth automatically to accomodate a sentence which is longer than the window is wide.

Data Structure of Input String

The definition of the get_French_sentence/1[5] rule in the User Interface file (see Appendix 4) will return the input sentence as a string, i.e., the variable will be bound to a series of characters surrounded by quotation marks.

```
Enter French-language sentence to be translated.
==> En parcourant les dépliants ci-joints, vous constaterez que nous fabriquons
une gamme complète de bibliothèques et rayonnages.
```

You will be asked to verify the sentence given as input. Use the scroll bars to read parts of the sentence that are not immediately visible. If there are errors, answer "no" to the prompt and you will be given another opportunity to enter the sentence. Do not worry if Ordinotrad has split some word over two lines.[6]

```
Original French sentence given:
En parcourant les dépliants ci-joints, vous constaterez que nous fabriquons une gamme
complète de bibliothèques et rayonnages.

Is this correct?
[yes,y,no,n] ==> y
```

Analysis

Once the input sentence is provided, Ordinotrad commences the analysis phase of the translation procedure with preprocessing. The preprocessing procedures are as follows in the order given: explode the input string into a list of characters; locate and assemble the characters into words, numbers, punctuation and symbols and label them as such ("w" for word, "p" for punctuation, "s" for symbol, and "n" for number) and assemble these elements into a list; expand elisions (e.g., "m'" in the input will become "me" at this point);

---

[5] The "atom/#" notation indicates the predicate name and its arity (number of arguments).

[6] Unfortunately, AAIS Prolog does not allow for word-wrap when inputting data in the Query window.

change three periods to an ellipsis[7]; abridge ordinals (i.e., reduce "2e" to "2" and label it an ordinal); and expand contractions (e.g., "au" becomes "à, le").

Data Structure - Results of Preprocessing

The definition of the pre_translation_procedures/2 rule takes the input string as its first argument and returns a list as its second argument. The structure of each element of the list depends on the nature of the element: words are replaced with "w(X, Y)", where X is the word as it was input, reduced to lower case letters,[8] and Y is the format of the word (i.e., low: all letters of the word were input in lower case; all_caps: all letters of the word were input in upper case; cap: the first letter only of the word was input in upper case; Mix: the letters of the word were input in both upper and lower case and did not meet the above requirements); numbers are replaced with "n(X, Y)", where X is the number as it was input and Y is "cardinal"[9]; punctuation is replaced with p(X), where X is the punctuation symbol[10]; and symbols are replaced with s(X), where X is the symbol. The results of preprocessing are displayed in the form of a column for the user.

```
RESULTS OF PREPROCESSING
w(en,cap)
w(parcourant,low)
w(les,low)
w(dépliants,low)
w(ci,low)
p(-)
w(joints,low)
p(.)
w(vous,low)
w(constaterez,low)
w(que,low)
w(nous,low)
w(fabriquons,low)
w(une,low)
w(gamme,low)
w(complète,low)
w(de,low)
w(bibliothèques,low)
```

---

[7] The Macintosh interface provides an ellipsis as a single character, but not all input will necessarily make use of it, especially text files brought in from non-Macintosh sources. The change is made here for the sake of homogeneity and to facilitate later processing of material containing such ellipses.

[8] The use of lower case letters allows for treating words as atoms in the Prolog environment. Words beginning with capitals could be also be considered atoms but would require the insertion of single quotation marks to distinguish them from variables.

[9] There is a subsequent check to see if the number is actually used as an ordinal, e.g., 2e. The tag "cardinal" is then changed to "ordinal."

[10] Punctuation is standardized, for example ', ', and ' all become '.

```
w(et,low)
w(rayonnages,low)
p(.)

Type any letter and a return to continue.
==> v
```

Given a list containing elements similar to those shown above, Ordinotrad commences to analyze the morphology of the words. Since morphological analysis is carried out in conjunction with lookups in the dictionary, it is appropriate to describe the structure of dictionary entries here.

Data Structure - Dictionary Entries

All unilingual dictionary entries adhere to the standard format:

language_letter(Entry_form, Description_of_Entry_form).

where: language_letter may be "f" (for French) or "e" (for English); Entry_form may be an inflected morpho-syntactically ambiguous form, i.e., the one form is used for more than one part of speech (e.g. "sommes," which can be a noun or a verb and "faits," which can be a noun, verb or adjective) or a morpho-syntactically unambiguous form (e.g. document, servir); and Description_of_Entry_form adheres to the following format:

if the Entry_form is ambiguous: [Pt_sp_1-$Desc_1$, Pt_sp_2-$Desc_2$ | Etc.][11]
if the Entry_form is unambiguous: [Pt_sp-Desc]

The following is an example of a morpho-syntactically unambiguous entry form:
f(lettre, [noun - (lettre, f, s, common, count, [nonselfmoving, submatter])]).

The following is the actual entry for the morpho-syntactically ambiguous form, "faits":

[11] Pt_sp is the shorthand I use in the program to refer to Part_of_speech.

f(faits, [verb - (faire, (past-part, m, pl), MF),[12]
adj - (fait, desc, m, pl, [attribute]),
noun - (fait, m, pl, common, count, [submatter])]).

Descriptive information within the entries describes the Entry_form in relation to its dictionary form. Dictionary form here refers to what can be found in the entry forms of the *Petit Robert* or a *Websters*, for example. The entry form is always the first argument in the description. The dictionary forms will be different from the entry form if the entry form is morpho-syntactically ambiguous (see the above example for "faits"). They will be the same if the entry form is morpho-syntactically unambiguous (see the above example for "lettre"). In the latter case, the entry form/dictionary form will also be in the singular for nouns, masculine singular for adjectives, and infinitive for verbs.[13]

The formats for individual parts of speech are as follows:

1 **ARTICLES**

art-(Article, Type, Gender, Number)
where
Article is the article in its dictionary form
Type may be definite or indefinite,
Gender may be masculine or feminine, or "nil" in the case of English
Number may be singular or plural.

e.g.
f(le, **art-(le, def, m, s)**, pro - (le, rel, 3, s, m, _)]).

2 **ADJECTIVES**

adj-(Adjective, Type, Gender, Number, Meaning)
where
Adjective is the adjective in its dictionary form
Type may be descriptive, possessive, cardinal, ordinal, multiplicative, demonstrative, relative, interrogative, indefinite
Gender may be masculine or feminine, or "nil" in the case of English
Number may be singular or plural
Meaning is a list of semantic primitives found in the taxonomy[14]

e.g.
f(deuxième, **[adj - (deuxième, ordinal, m, s, [ttrelation])]**).

3 **PREPOSITIONS**

prep-(Preposition)
where

---

[12] MF represents the list of case frames for each meaning of "faire", a rather complex data structure in itself. See below.

[13] Exceptions will be, of course, dictionary forms that have no masculine, singular or masculine-singular form.

[14] See the description of the taxonomy in Appendix 2.

Preposition is the preposition in its dictionary form

e.g.
f(à, [**prep - (à)**]).

4 **NOUNS**

noun-(Noun, Gender, Number, Proper_common, Count_mass, Meaning)
where
Noun is the noun in its dictionary form
Gender may be masculine or feminine, or "nil" in the case of English
Number may be singular or plural
Proper_common may be proper or common
Count_mass may be count or mass
Meaning is a list of semantic primitives found in the taxonomy

e.g.
f(lettre, [**noun - (lettre, f, s, common, count, [nonselfmoving, submatter])**]).

5 **ADVERBS**

adv-(Adverb, Type)
where
Adverb is the adverb in its dictionary form
Type may be manner, quantity, intensity, time, location, affirmation, negation, or doubt

e.g.
f(après, [**adv - (après, location)**, prep - (après)]).

6 **VERBS**

verb-(Verb, Morphologic_info, Meaning_frames)
where
Verb is the verb in its dictionary (infinitive) form
Morphologic_info is "infinitive" or (Tense, Mood, Person, Number) or (Participle, Gender, Number)
Meaning_frames is a list of meaning-case frame combinations[15]

where a Meaning_frame is of the form:

Meaning-[$Case_1$-($Degree\_of\_dependency_1$, $Semantic\_constraints_1$, $Preposition_1$), $Case_2$-($Degree\_of\_dependency_2$, $Semantic\_constraints_2$, $Preposition_2$) | Etc.][16]

where
Case is agent, instrument, beneficiary, affected, result, source, destination, or attribute
Degree_of_dependency is r(equired) or o(ptional), indicating the relationship between that case role and the event

---

[15] Each meaning of a verb necessarily requires its own case frame. See, for example, the entry for "servir" in the French Open Dictionary in Appendix 4.

[16] This is written in Prolog notation. The "Etc." can therefore be an emply list or a list of one or more elements. It would therefore be incorrect to use "$Case_n$-($Degree\_of\_dependency_1$, Semantic $Constraints_n$, $Preposition_n$)" in its place.

Semantic_constraints is a list of semantic primitives which set restrictions on what can fill that case role
Preposition is the preposition usually associated with that verb and that case role in its corresponding surface structure, or is "nil"

e.g.
f(parcourir,[verb - (parcourir, inf,
[mentinteraction - [agent - (r, [human], nil),affected - (r, [artifact], nil)],
selfdisplacement - [agent - (r, [person,selfmoving], nil),source - (o, [pspace], nil), destination - (r, [location], à)]])]).

7 CONJUNCTIONS

conj-(Conjunction, Type)
where
Conjunction is the conjunction in its dictionary form
Type may be coordination or subordination

e.g.
f(donc, [**conj - (donc, sub)**]).

8 PRONOUNS

pro-(Pronoun, Type, Person, Number, Gender, Meaning)
where
Pronoun is the pronoun in its dictionary form
Type is personal, relative, interrogative, possessive, demonstrative, indefinite, intensive, or reflexive
Person may be 1st, 2nd, or 3rd, if Type is personal, relative, interrogative, or possessive; otherwise Person is "nil"
Number may be singular or plural
Meaning is a list of semantic primitives found in the taxonomy

e.g.
f(je, [**pro - (je, pers, 1, s, (m,f), [person])**]).

Data Structure - Results of Morphological Analysis

The definition of the morphanalyzer/2 rule takes the list of elements provided by the preprocessing module and replaces each "w(X,Y)" element[17] with "lu(Format, Description)" where Format is bound to the same value as has been bound to Y, and the Description contains morpho-syntactic and semantic information along with the dictionary form of the word. At this point, "w" (word) becomes "lu" (lexical unit) primarily for the sake of avoiding confusion during debugging of the morphological analysis component of the program. It also allows for any future development in which the lexical unit (for translation purposes) may of necessity have to be more than one word long.[18] The data

---

[17] The number, punctuation and symbol elements are left unchanged.

[18] The morphological analysis routines would have to be extended to include forward checks for lexicalized phrases such as, for example, "en guise de."

structure of "Description" is identical to one of those described above for the dictionary entries.

The following output may not appear exactly as it does on the screen during a session. This is because I have formatted the paragraphs here to facilitate reading, and also removed the carriage returns inserted by AAIS Prolog which had split up some words (AAIS Prolog has some word-wrap problems).

```
RESULTS OF MORPHOLOGICAL ANALYSIS BASED ON DICTIONARY LOOKUP
lu(cap,[prep-en,pro-(en,rel,3,(s,pl),(m,f),[mthing,fthing,plant,nlthing])])
lu(low,[verb-(parcourir,(pres-part,m,s),[mentinteraction-[agent-(r,[human],nil),affected-
    (r,[artifact],nil)],selfdisplacement-[agent-(r,[person,selfmoving],nil),source-
    (o,[pspace],nil),destination-(r,[location],à)]])])
lu(low,[art-(les,def,(m,f),pl),pro-(les,rel,3,pl,(m,f),_1485)])
lu(low,[noun-(dépliant,m,pl,common,count,[artifact]),verb-(déplier,(pres-part,m,pl),[process-
    [agent-(r,[person],nil),instrument-(o,[artifact],avec),affected-(r,[artifact],nil)]])])
lu(low,[adv-(ci,location)])
p(pct,-)
lu(low,[noun-(joint,m,pl,common,count,[ttrelation]),verb-(joindre,(past-part,m,pl),[contact-[agent-
    (r,[posprocess,structure,sentient],nil),instrument-(o,[artifact],par),affected-
    (r,[pthing,mthing],nil),destination-(r,[mthing,pthing],à)]])])
p(pct,.)
lu(low,[pro-(vous,pers,2,(s,pl),(m,f),[person]),pro-(vous,refl,2,(s,pl),(m,f),[person])])
lu(low,[verb-(constater,(fut,ind,2,pl),[mentinteraction-[agent-(r,[person],nil),instrument-
    (o,[artifact],nil),affected-(r,[mthing],nil)]])])
lu(low,[conj-(que,sub),pro-(que,rel,3,(s,pl),(m,f),_1841)])
lu(low,[pro-(nous,pers,1,pl,(m,f),[person]),pro-(nous,refl,1,pl,(m,f),[person])])
lu(low,[verb-(fabriquer,(pres,ind,1,pl),[posprocess-[agent-(r,[sentient],nil),instrument-
    (o,[nlthing],avec),beneficiary-(o,[sentient],pour),result-(r,[artifact,mthing],nil)]])])
lu(low,[adj-(une,cardinal,f,s,[quantity]),art-(une,indef,f,s)])
lu(low,[noun-(gamme,f,s,common,mass,[structure])])
lu(low,[adj-(complet,desc,f,s,[attribute])])
lu(low,[prep-de])
lu(low,[noun-(bibliothèque,f,pl,common,count,[artifact,pspace,group])])
lu(low,[conj-(et,coord)])
lu(low,[noun-(rayonnage,m,pl,common,count,[artifact])])
p(pct,.)

Type any letter and a return to continue.
==> l
```

Data Structure - Results of Parsing

The definition of the parser/2 rule takes the list of elements provided by the morphological analysis module as its first argument and returns a deep-structure of the input sentence in the form of a tree as its second argument.

In the following display of the results, each indentation is equivalent to a subordinate generation in the tree, with all entries at one level of indentation being

siblings.[19] In other words, the output is a tree whose root is flush left and whose successive branches are demarcated by successive indentations. Some formatting has been done here to maintain clarity, since the word-wrap provide by the word-processor used in preparing this thesis moves lines longer than the width of the screen back to the left margin, incorrectly implying that such information is root-level generation.

Once a successful check has been made for agreement between a possible subject and a verb, this fact is shown on the screen.[20] This information is supplied, as the check is a crucial point in parsing, indicating that a verb has been selected and verified with regard to agreement in person and number with the surface subject. In the following example, this information is given twice since there are two propositions (i.e., two subject-verb combinations) within the surface sentence.

The abbreviations shown in the following output are for the most part self-explanatory. One exception may be "p", which refers to "punctuation."

```
Subject-verb agreement succeeded.
Subject-verb agreement succeeded.

RESULTS OF PARSING
proposition
    subordinate_prop
        event_prop
            event_prop
                modalities
                    pres
                    participle
                    active
                    affirmative
                    declarative
                event
                    lu(low,verb-(parcourir,(pres,participle,nil,nil),[mentinteraction-[agent-
                    (r,[human],nil),affected-(r,[artifact],nil)],selfdisplacement-
                    [agent(r,[person,selfmoving],nil),source-(o,[pspace],nil),destination-
                    (r,[location],à)]]))
                prep
                    lu(cap,prep-en)
                np
                    np
                        noun
                            lu(low,noun-(dépliant,m,pl,common,count,[artifact]))
                        modifiers
```

---

19 With regard to the apparent duplication of "event_prop" and "np", see note 17 in Chapter 1.

20 After the subject-verb agreement has been ascertained, the terminology changes. Note in the display that the term "event" is used as opposed to "verb." The reason for this is that in the case of composite verbs, only the main verb is returned (the auxiliaries are reflected in the modality of the sentence), and it is identified as the event.

```
                        lu(low,art-(les,def,m,pl))
                        lu
                            lu(low,adv-(ci,location))
                            lu(low,adj-(joindre,verbal-desc,m,pl,[contact]))
articulation
    p
        .
independent_prop
    modalities
        fut
        ind
        active
        affirmative
        declarative
    event
        lu(low,verb-(constater,(fut,ind,2,pl),[mentinteraction-
        [agent(r,[person],nil),instrument-(o,[artifact],nil),affected-(r,[mthing],nil)]]))
    np
        np
            lu(low,pro-(vous,pers,2,pl,(m,f),[person]))
    relative_prop
        relative_prop
            modalities
                pres
                ind
                active
                affirmative
                declarative
            event
                lu(low,verb-(fabriquer,(pres,ind,1,pl),[posprocess-
                [agent(r,[sentient],nil),instrument-(o,[nlthing],avec),beneficiary-
                (o,[sentient],pour),result-(r,[artifact,mthing],nil)]]))
            np
                np
                    lu(low,pro-(nous,pers,1,pl,(m,f),[person]))
            np
                np
                    noun
                        lu(low,noun-(gamme,f,s,common,mass,[structure]))
                    modifiers
                        lu(low,art-(une,indef,f,s))
                        lu(low,adj-(complet,desc,f,s,[attribute]))
                        qualifying_prep_phrase
                            qualifying_prep_phrase
                                prep
                                    lu(low,prep-de)
                                np
                                    np
                                        noun
                                            lu(low,noun(bibliothèque,f,pl,common,count,
                                            [artifact,pspace,group]))
                                        modifiers
                                            nil
                                            nil
                                    articulation
```

```
                                    lu(low,conj-(et,coord))
                                np
                                    noun
                                        lu(low,noun-
                                        (rayonnage,m,pl,common,count,[artifact]))
                                    modifiers
                                        nil
                                        nil
            np
                np
                    lu(low,pro-(que,rel,3,(s,pl),(m,f),_1841))

Type any letter and a return to continue.
==> c
```

Data Structure - Results of Case Processing

The case_processor/2 rule takes the tree produced by the parser as its first argument and returns another tree as its second argument. The trees are substantially the same; however, the case processing module attaches semantic labels onto noun phrases and other sentence components identified by the parser. These semantic labels always appear as siblings to the event that governs them.

Another change in the tree is that case frames are dropped from the descriptions of events. This is possible, because the case frame is now reflected in the use of the labels within the tree and their relationship to an event.

It can also be noted that formatting information (i.e., low, cap, etc.) has been kept with the other information provided on lexical units. There are two reasons for keeping it, even though it is not used in Ordinotrad at the present. First, this information may prove useful for a possible generation module for another language. Second, future developments may require transferring other formatting information from the source text to the target text (e.g., boldface, underlining, etc.), and this "format" slot would allow for representing this information through the transfer and generation processes.

```
RESULTS OF CASE PROCESSING
proposition
    subordinate_prop
        event_prop
            event_prop
                modalities
                    pres
                    participle
                    active
                    affirmative
                    declarative
```

```
            event
                lu(low,verb-(parcourir,(pres,participle,nil,nil),mentinteraction))
            prep
                lu(cap,prep-en)
            affected
                np
                    np
                        noun
                            lu(low,noun-(dépliant,m,pl,common,count,[artifact]))
                        modifiers
                            lu(low,art-(les,def,m,pl))
                            lu
                                lu(low,adv-(ci,location))
                                lu(low,adj-(joindre,verbal-desc,m,pl,[contact]))
articulation
    p
        .
independent_prop
    modalities
        fut
        ind
        active
        affirmative
        declarative
    event
        lu(low,verb-(constater,(fut,ind,2,pl),mentinteraction))
    agent
        np
            np
                lu(low,pro-(vous,pers,2,pl,(m,f),[person]))
    affected
        relative_prop
            relative_prop
                modalities
                    pres
                    ind
                    active
                    affirmative
                    declarative
                event
                    lu(low,verb-(fabriquer,(pres,ind,1,pl),posprocess))
                agent
                    np
                        np
                            lu(low,pro-(nous,pers,1,pl,(m,f),[person]))
                result
                    np
                        np
                            noun
                                lu(low,noun-(gamme,f,s,common,mass,[structure]))
                            modifiers
                                lu(low,art-(une,indef,f,s))
                                lu(low,adj-(complet,desc,f,s,[attribute]))
                                qualifying_prep_phrase
                                    qualifying_prep_phrase
```

```
                                        prep
                                            lu(low,prep-de)
                                        np
                                            np
                                                noun
                                                    lu(low,noun-
                                                    (bibliothèque,f,pl,common,count,
                                                    [artifact,pspace,group]))
                                                modifiers
                                                    nil
                                                    nil
                                            articulation
                                                lu(low,conj-(et,coord))
                                            np
                                                noun
                                                    lu(low,noun-(rayonnage,m,pl,
                                                    common,count, [artifact]))
                                                modifiers
                                                    nil
                                                    nil
                    beneficiary
                        np
                            np
                                lu(low,pro-(que,rel,3,(s,pl),(m,f),[sentient]))
```

Type any letter and a return to continue.
==> t

Transfer and Generation

Data Structure - Results of Lexical Transfer

The transfer/2 rule takes the tree produced by the case processor module as its first argument and returns another tree as its second argument. The trees are substantially the same; however, the transfer module replaces the French words and their descriptions with English words and their descriptions.[21]

```
RESULTS OF LEXICAL TRANSFER

proposition
    subordinate_prop
        event_prop
            event_prop
```

[21] It would be feasible at this point to have a tree whose "French" terminals were not replaced with "English" ones but were accompanied by them. The result would be a type of "bitext" data structure available for use in "bitextual" applications, in addition to the obvious use here of translation. The reader is encouraged to read Brian Harris, "Are You Bitextual?" *Language Technology* #7 (May-June 1988): 41 and his two unpublished monographs, "A Working Paper on Bitext, January 1988" and "Bitext and *IT*, July, 1988." The difference here is that the translation's source and target texts are not juxtaposed on the same screen or page, but on the same deep-structure representation. Also, the units here would not necessarily be the "translation units" translated "at one spurt."

```
            modalities
                pres
                participle
                active
                affirmative
                declarative
            frame
                beneficiary
                    o[22]
                    person
                    for
                affected
                    r
                    artifact
                    through
            event
                lu(low,verb-(go,(pres,participle,nil,nil),mentinteraction))
            prep
                lu(cap,prep-[in,by,while,them])
            affected
                np
                    np
                        noun
                            lu(low,noun-(folder,pl,common,count))
                        modifiers
                            lu(low,art-(the,def,nil,pl))
                            lu
                                lu(low,adv-(here,location))
                                lu(low,adj-(attached,desc))
articulation
    p

independent_prop
    modalities
        fut
        ind
        active
        affirmative
        declarative
    frame
        agent
            r
            human
            nil
        affected
            r
            mthing
            event
            nil
        source
            o
            artifact
```

[22] The "o" here and the "r" four lines further are abbreviations used in case frames: optional and required.

```
        from
event
    lu(low,verb-(note,(fut,ind,2,pl),mentinteraction))
agent
    np
        np
            lu(low,pro-(you,pers,2,pl))
affected
    relative_prop
        relative_prop
            modalities
                pres
                ind
                active
                affirmative
                declarative
            frame
                agent
                    r
                    person
                    nil
                instrument
                    o
                    artifact
                    with
                beneficiary
                    o
                    event
                    lthing
                    for
                result
                    r
                    artifact
                    nil
                source
                    o
                    nlthing
                    from
            event
                lu(low,verb-(manufacture,(pres,ind,1,pl),posprocess))
            agent
                np
                    np
                        lu(low,pro-(we,pers,1,pl))
            result
                np
                    np
                        noun
                            lu(low,noun-(range,s,common,count))
                        modifiers
                            lu(low,art-(a,indef,nil,s))
                            lu(low,adj-(complete,desc))
                            qualifying_prep_phrase
                                qualifying_prep_phrase
                                    prep
```

```
                                    lu(low,prep-[functional_prep])
                                np
                                    np
                                        noun
                                            lu(low,noun-(library,pl,common,count))
                                        modifiers
                                            nil
                                            nil
                                    articulation
                                        lu(low,conj-(and,coord))
                                    np
                                        noun
                                            lu(low,noun-
                                            (shelving,pl,common,mass))
                                        modifiers
                                            nil
                                            nil
beneficiary
    np
        np
            lu(low,pro-(that,rel,3,pl))
```

```
Type any letter and a return to continue.
==> o
```

Data Structure - Output String

The generation/2 rule takes the tree produced by the transfer module as its first argument and returns a string as its second argument. The main elements of the generation procedure are described in Chapter 1. There are, however, three data structures involved in generation: the input tree as described above, a "flat" list containing what will become the words of the surface structure sentence, the same list with appropriate punctuation, spacing and capitalization added, and finally a string that is returned as the English equivalent of the French input.

```
TRANSLATION
Going through the enclosed folders, you will note that we manufacture a complete range of
libraries and of shelving.

Translation operations took 2 minutes and 44 seconds.

Do you have any more sentences to be translated?
[yes,y,no,n] ==> n
```

Note that the time given for translating also includes any time required to display information on the screen and the time taken by the user in responding to the prompts or in viewing interim results. The amount of time varies, therefore, with the user.

ORDINOTRAD™
© 1987, 1988 Victor Loewen

There will eventually be a translate menu. The following are now available.
To have a sentence translated with the results viewed on the screen, type "translate."
To have a file translated with the results viewed on the screen, type "translate(Filename)."
• Files to be translated must be located in the corpora folder.
To quit, select QUIT under the FILE menu or simply type "exit." and a return.
yes
?-

At this point, the user may choose to quit or to enter one of the three "commands" so as to continue translating. The following represents the screen display without the interim structures being shown.

Since tracing was enabled at startup[23] (by responding "yes" to the third question asked when the session was begun), it is now necessary to use the "trace_off." call to turn off the display of interim output. Note that the same French sentence is given as input. (See the section "Controlling screen output" below.)

?- **trace_off.**
yes
?- **translate.**

Enter French-language sentence to be translated.
==> En parcourant les dépliants ci-joints, vous constaterez que nous fabriquons une gamme complète de bibliothèques et rayonnages.

Original French sentence given:
En parcourant les dépliants ci-joints, vous constaterez que nous fabriquons une gamme complète de bibliothèques et rayonnages.

Is this correct?
[yes,y,no,n] ==> **y**

---

[23] See page 83.

```
Preprocessing completed.
Morphological analysis completed.
   Subject-verb agreement succeeded.
   Subject-verb agreement succeeded.
Parsing completed.
Case processing completed.
Transfer completed

TRANSLATION
Going through the enclosed folders, you will note that we manufacture a complete range of
libraries and of shelving.

Translation operations took 0 minutes and 7 seconds.

Do you have any more sentences to be translated?
[yes,y,no,n] ==> n
```

The time given for producing a translation here closely reflects the total time required for processing, as there was no user interaction, and the time required for displaying information on screen was minimal. The time required is measured from when the user agrees that the sentence given as input is correct and ends just before the word "TRANSLATION" is displayed.

INTERRUPTING AND ABORTING A TRANSLATION

In the event things appear to be going wrong (e.g., nothing happens after a long while), or if you wish to break into the process and see where things are at, the interrupt sequence to be typed is "⌘-." (hold down the ⌘ key and type a period). AAIS Prolog will display the call it is currently trying to answer, display a colon, and halt temporarily. At this point, control is returned to the user. You may wish to turn on the full-step mode (type a "c" and a return or select the appropriate item on the "Debug" menu) or use some item on some menu. At the colon prompt, entering a carriage return will cause Ordinotrad to resume processing where it left off. In order to abort the whole process, enter an "a" and a carriage return at the colon prompt. It is advisable at this point to reset the input/output parameters and close any files that may have been opened during processing (enter "resetio." at the "?-" prompt).

Consult the AAIS Prolog manual for other options available at the user-interrupt prompt.

CONTROLLING SCREEN OUTPUT

It is not necessary to view all the interim data structures when having a sentence translated. Upon loading the system, the user is given the option to answer "yes" or "no" to the question of having these interim structures displayed. Once the program is loaded, and the user is at the "system level," it is possible to toggle between displaying and not displaying these results by typing in the Prolog calls (defined within Ordinotrad) "trace_on." or "trace_off."

Prolog programmers may wish to view the action at a snail's pace, i.e., one processing step at a time. This can be done by selecting "Global step" on the "Debug" menu when control is returned to the user (i.e., when AAIS Prolog recognizes keyboard and menu activity). This feature can be turned off by selecting "No Global Trace/Step" on the same menu.

Furthermore, it is possible to "spy on" individual predicates (i.e., viewing them when called and failed or exited) by selecting "Trace/Step goal..." on the "Debug" menu and typing in the desired predicate name in the dialog box. This feature can be turned off by repeating the above process and clicking "Nospy" in the dialog box.

The last two activities are quite low-level in terms of programming and would be more confusing than enlightening so far as machine translation is concerned.

PREPARING FILES FOR "T2D(FILENAME)" OR "TRANSLATE(FILENAME)"

Text files[24] can be prepared on the Macintosh with any wordprocessor capable of producing text files. Text files can also be transferred from other computers via modem or other means (e.g., exchange of diskette), provided they are in ASCII format. The reason that files must be in text format is that Ordinotrad is not capable of interpreting formatting codes inserted by the various wordprocessing packages available. Such interpretation could, of course, be done by an "envelope" preprocessor (written, perhaps, in C), which would remove formatting codes prior to submitting the sentence to Ordinotrad. Once the sentence is translated, the formatting codes could then be reapplied (or "translated"). For example, an address or date may be at the right margin in a French letter; this information could be applied to the English translation, or it could be replaced with a typical English format (at the left margin, for example). Also, translations could be produced in boldface or italics, reflecting the format of the French input.

---

[24] The same type of file is called an ASCII file in the IBM environment.

The I/O predicate used to read data from a file on disk is provided by AAIS Prolog (i.e., readline(X)). With this predicate, a sentence is interpreted by Ordinotrad to be all data up to but not including a carriage return. For this reason, it is important that sentences and mentions end with carriage returns, so that only one sentence is read at one time. The fact that a sentence may be word-wrapped by the word-processor when creating the file is not important. In a sense, the "readline" predicate is a misnomer, since the call is able to read several physical lines provided they do not contain any carriage returns, i.e., what some word-processors consider a paragraph.

USING "T2D(FILENAME)."

When having a file on disk translated with the "t2d(Filename)" call, user interaction with Ordinotrad is basically the same as for "translate." and "translate(Filename).", with the main difference being the fact that results are also saved in a file on disk.

The restrictions for using this rule are that the Filename must be a Prolog symbol (put it between single quotation marks to be sure), and the file to be translated must be in the "corpora" folder on the "DATA" disk.

The following is a verbatim transcript of a session in which a short file was translated. The file name was "short example" and it contained the following data.

```
J'en suis un.
Il en a fait trois.
Il en a fait trois adresses.
Il en a fait beaucoup.
Je leur en ai adressé trois.
J'ai trois des lettres.
Tu as trois lettres.
Il a trois.
Ils leur avaient adressé les cinq lettres.
Ils auront adressé, Monsieur, les lettres à
eux.
Nous vous remercions de votre lettre.
Nous sommes heureux de vous adresser la
documentation demandée.
```

```
?- trace_off.
   yes
?- t2d('short  example').

Now translating file titled:  short example
The translation will be in the corpora folder with the title:  short example.English
Please stand by; you may be needed to answer a few questions.
```

```
NOW TRANSLATING:
J'en suis un.
Preprocessing completed.
Morphological analysis completed.
   Subject-verb agreement succeeded.
Parsing completed.
Case processing completed.
Transfer completed
TRANSLATION SAVED TO DISK:
I am one of them.

NOW TRANSLATING:
Il en a fait trois.
Preprocessing completed.
Morphological analysis completed.
   Subject-verb agreement succeeded.
Parsing completed.
Case processing completed.
Transfer completed
TRANSLATION SAVED TO DISK:
He made three of them.

NOW TRANSLATING:
Il en a fait trois adresses.
Preprocessing completed.
Morphological analysis completed.
   Subject-verb agreement succeeded.
Parsing completed.
Case processing completed.
Transfer completed
TRANSLATION SAVED TO DISK:
He made three addresses of them.

NOW TRANSLATING:
Il en a fait beaucoup.
Preprocessing completed.
Morphological analysis completed.
   Subject-verb agreement succeeded.
Parsing completed.
Case processing completed.
Transfer completed
TRANSLATION SAVED TO DISK:
He made many of them.

NOW TRANSLATING:
Je leur en ai adressé trois.
Preprocessing completed.
Morphological analysis completed.
   Subject-verb agreement succeeded.
Parsing completed.
Case processing completed.
Transfer completed
TRANSLATION SAVED TO DISK:
I sent three of them to them.
```

NOW TRANSLATING:
J'ai trois des lettres.
Preprocessing completed.
Morphological analysis completed.
  Subject-verb agreement succeeded.
Parsing completed.
Case processing completed.
Transfer completed
TRANSLATION SAVED TO DISK:
I have three of the letters.

NOW TRANSLATING:
Tu as trois lettres.
Preprocessing completed.
Morphological analysis completed.
  Subject-verb agreement succeeded.
Parsing completed.
Case processing completed.
Transfer completed
TRANSLATION SAVED TO DISK:
You have three letters.

NOW TRANSLATING:
Il a trois.
Preprocessing completed.
Morphological analysis completed.
  Subject-verb agreement succeeded.
Parsing completed.
Case processing completed.
Transfer completed
TRANSLATION SAVED TO DISK:
He has three.

NOW TRANSLATING:
Ils leur avaient adressé les cinq lettres.
Preprocessing completed.
Morphological analysis completed.
  Subject-verb agreement succeeded.
Parsing completed.
Case processing completed.
Transfer completed
TRANSLATION SAVED TO DISK:
They had sent the five letters to them.

NOW TRANSLATING:
Ils auront adressé, Monsieur, les lettres à eux.
Preprocessing completed.
Morphological analysis completed.
  Subject-verb agreement succeeded.
Parsing completed.
Case processing completed.
Transfer completed
TRANSLATION SAVED TO DISK:
They will have sent the letters to them, Sir.

```
NOW TRANSLATING:
Nous vous remercions de votre lettre.
Preprocessing completed.
Morphological analysis completed.
   Subject-verb agreement succeeded.
Parsing completed.
Case processing completed.
Transfer completed
TRANSLATION SAVED TO DISK:
We thank you for your letter.

NOW TRANSLATING:
Nous sommes heureux de vous adresser la documentation demandée.
Preprocessing completed.
Morphological analysis completed.
   Subject-verb agreement succeeded.
Parsing completed.
Case processing completed.
Transfer completed
TRANSLATION SAVED TO DISK:
We are happy to send requested documentation to you.

Processing has been completed.
The translation is in the corpora folder with the title:  short example.English
  yes
?-
```

A file called "short example.English" was created automatically by Ordinotrad and it contained the following data. This file could, of course, have been viewed and printed using AAIS Prolog editing and printing resources, or processed with another word-processor, as is the case here.

```
I am one of them.
He made three of them.
He made three addresses of them.
He made many of them.
I sent three of them to them.
I have three of the letters.
You have three letters.
He has three.
They had sent the five letters to them.
They will have sent the letters to them, Sir.
We thank you for your letter.
We are happy to send requested
documentation to you.

Work on this file, including I/O and user
interaction, took 0 minutes and 46 seconds.
```

THE MACINTOSH CHARACTER SET

The Macintosh character set recognized by Ordinotrad contains 256 characters, including most of the French accented characters. During the preprocessing phase of the analysis procedure, characters are checked on the assumption that a laser font is being used, as the laser fonts are the ones that provide the accented characters used in French. When running Prolog (with Ordinotrad in memory), the database of acceptable letters can be reviewed by typing "list(letter).". If a series of small rectangles are displayed, change the font being used to a laser font (Courier, Helvetica, Times, etc.) by selecting the appropriate name under the "Format" menu (the font size may also have to be changed, depending on the font). The "Keycaps" desk accessory shows how the various characters can be entered through the keyboard.

Ordinotrad input/output need not be in a laser font, as it does not process what is seen on the screen but the ASCII numbers themselves. For example, a text file may be prepared using a laser font, to allow the user to verify visually the correctness of the input, and Ordinotrad may be run under the Geneva font, which is eminently more readable, although it may display small rectangles in the place of some accented characters.

Chapter 5

Evaluation and Conclusion

ACCOMPLISHMENTS

The main achievement of the work described in this thesis has been the design and implementation of a kernel machine-translation system which is capable of translating simple texts from French to English. The size of the program required to carry out such translations is a little larger than I had anticipated, and more than one megabyte of RAM is now needed (as opposed to my original estimate). Apart from the hardware requirements, the program itself proved to be comprised pretty well of what I expected: preprocessor, morphological analysis, parsing, case analysis, transfer, and generation; complemented with some utility files: a word acquisition module which creates entries for dictionary, a taxonomy, dictionaries, and a user interface.[1]

Furthermore, it has been shown that case grammar principles may be implemented in an automated translation process with some success, in three ways in particular. First, these principles allow for incorporating "meaning" into an otherwise syntactically based environment, thus bringing this system into the theoretical fold of meaning-based translation. Second, the case grammar principles provide an elegant method for controlling the development of a deep-structure representation of a sentence by verifying equivalence between the semantic requirements of a role and the meaning of a sentence component. Third, the cases and case-grammar concepts used in analysis are also used in transfer and generation, thus providing for a "smooth" flow of data throughout the process.

In view of the results provided in Appendix 3, it appears that Ordinotrad achieves the "measurable" goal described in Chapter 1 (page 3), namely, to produce output that a human translator would consider worth revising. No statistical analysis has been done to see if the translations achieve an 80 percent success rate (i.e., fewer than 1 required correction per 5 words), though at a first glance it would seem to hover around this level of accuracy.

---

[1] These utility files may be found at the end of Appendix 4.

WHAT HAS BEEN HANDLED

The program was designed to handle administrative correspondence, as can be seen in the various results provided in Appendix 3. Correspondence was understood to include both mentions and sentences. With regard to the former type of input, as many possible correct variations for each type of mention (e.g. how many ways are there to say "Cher Monsieur," in French?) were considered in developing the program. Yet the most interesting and challenging programming is for the sentences (i.e., propositions), which were allowed to be open-ended. That is, Ordinotrad will deal with only one sentence at a time, but this sentence may be comprised of any number of propositions (independent, subordinate, ellipsed surface subject) which may be connected, for example, by ", et". Each of these propositions in turn are comprised of syntactic sentence-components (noun phrases, verbs, prepositional phrases, etc.) which themselves are allowed to be compounded (e.g. "Jean et Marie ont envoyé les lettres et la documentation à nous et aux autres."). The only compound syntactic structures that cannot be currently handled by Ordinotrad are verbal ones, as this situation presents extensive programming difficulties whose resolution would take considerably more time than my resources allow. For example, in the sentence "Jean achète et vend des voitures," frame compatibility between the two Events would have to be established during analysis and again during generation.

The sentence types which Ordinotrad will reject (i.e., for which it will not go further than morphological analysis and an attempt to parse) include those with compound verbs, as just mentioned, along with imperative or interrogative statements, and sentences beginning with "il y a", "c'est/ce sont ... que", and "il fait ... que". This limitation does not reflect on the utility of case grammar, it only shows that Ordinotrad's parser is inadequate and more rules need to be defined.

It should be added that it is possible for incomplete sentences to be translated by Ordinotrad, provided all of their components can be treated as mentions (e.g. a series of noun phrases, or some other mixture of mentions). No sentence would be constructed in English, but each mention would be translated in sequence and apart from the others. Also, such translations would be quite word-bound (i.e., highly reflect the French surface structure), as there would be no Event providing a frame to govern their translation. This reflects the centrality of the Event in case grammar. If there is no Event, there is no case frame, and case grammar is rendered inoperative.

PROSPECTS FOR DEVELOPMENT

Apart from the obvious advantages that could be gained by augmenting the parser with more rules to handle a greater variety of sentence types, the use of meaning could be implemented further to apply much more stringent restrictions on choice of words.

For example, the case processor could be amended so that when a match is found between a case-relation requirement and a given meaning of a word (from among several), this meaning alone is retained to control later transfer and generation.[2] For example, given:

f(X, [noun-(X, DescX, [M1, M2, M3, M4])])

f_e(X, [A,B,C])

e(A, [noun-(A, DescA, [M1])])

e(B, [noun-(B, DescB, [M2])])

e(C, [noun-(C, DescC, [M3])])

and a case frame in which a particular role requires M1

where

f is the predicate of an entry in the French Dictionary

X is the French word

DescX is a morpho-syntactic description of X

M1, M2, M3, M4 are semantic primitives found in the taxonomy[3]

f_e is the predicate of an entry in the Transfer Dictionary

A, B, C are English words suggested as equivalents for X

e is the predicate of an entry in the English Dictionary

then in that role only A would be acceptable as an equivalent. At the present, the same result is achieved from another angle, i.e., through the case frame. The semantic requirements indicated in a case frame must be met by both the French input and the suggested English equivalent. The weakness of this approach lies in the fact that while both the French and English words may meet the semantic requirements of the case frame, they are not necessarily semantic equivalents. For example, the semantic requirement for a given role may be a "mental thing." Yet the intensional meaning of the French word could be a measurement in space, whereas that of the English word could be a measurement in time.

---

[2] At the present time, no meanings are discarded, even though just one of them may satisfy the semantic requirement of a case frame.

[3] See Appendix 2.

I have also tried very much to keep out any "English" interference in the analysis process, that is, not to write rules in the French parser that anticipate what would be expected in English (e.g., I have kept the format of a French word (the fact that it may be capitalized) up to the point of generation, even though the English format is based on its own requirements (sentence position or proper/common nature and not on the French in the original). Because of this effort, I believe that this system holds promise for working with a different target language whose case labels are similar to those used in herein. In other words, the transfer and generation (and related) files could be replaced with others that would generate a sentence in another language, without having to change the parser and case analysis modules.

WHAT WOULD NEED IMMEDIATE ATTENTION

The following improvements are basically small tasks pre-requisite to any further development, requiring some linguistic-translation premise to be made at a theoretical level and then an appropriate implementation in the program. The reason they are not implemented at this point is that if I did not set a deadline for stopping any further effort once the program worked, programming could go on for another year and continually create small, short-term improvements.

There is a need for better generation of nouns in apposition (e.g. meubles de bureau), i.e., a procedure which would distinguish nouns that could be put in front of the head noun in English (in this example "office" could precede "furniture") and those that really are head nouns within a qualifying prepositional phrase that should be positioned after the head noun in English (e.g., given "meubles du Canada", "from Canada" should follow "furniture"). In other words, there should be a way to distinguish nouns which function as qualifiers from those that function as head nouns within a qualifying phrase. It may be necessary to be more precise and say that adjectives function as qualifiers and that some nouns too can function as qualifiers to avoid diluting the long-standing, traditional distinction of nouns and adjectives. This concept would then have to be incorporated into the transfer module (see the "swap" rule for adjectives in the Transfer file in Appendix 4, which now only allows for adjectives and verbs to become adjectives in English).

There is a need for better handling of "the" as a qualifier of mass nouns if the noun is otherwise qualified. At the present, "the" is dropped before mass nouns. Consider, for

example, Ordinotrad's translation of "nous assurons le transport et l'installation": "we provide transportation and the installation".

Finally, the two negative forms "ne...ni...ni" and "ne...que" cannot be handled by the parser. Special rules would need to be created, perhaps to change "ne...que" to "seulement" and position it in the place of the "que"[4] and change the form (the last element in the set of modalities) of the sentence from "negative" to "affirmative." The "ne...ni...ni" problem relates directly to the use of compound verbs (which are not handled) and to compound noun phrases (or any other compound syntactic unit) which can be handled. The former problem should be dealt with first, as it comprises a far greater restriction on the adequacy of the parser.

There is an interesting bug in the morphological analysis routines caused in part by the shortage of entries in the dictionaries. For example, at one point in program development, the word "cher" had been defined. Then, when inputting a sentence containing the verb "chanter" (e.g., Je ne le chante plus), the analysis routines determined that "chante" was the feminine singular, present participle of a supposed verb "cher". However, the dictionary entry for "cher" did not contain a verb description, and so the process failed at that point. I believe that this type of problem would not occur, were the majority of common words used in French defined in the dictionary.

WHAT WOULD NEED ATTENTION IN THE LONG TERM

Any long-term development would require immediately a dictionary containing the most common words used in French and definitions of appropriate English equivalents. It would then be appropriate to build both dictionaries to an adequate size so that the sublanguage of administrative correspondence would be reflected in it. Full forms are needed for *être*, *avoir*, *faire*, *aller*, *voir*, *devoir*, *pouvoir*, *vouloir*, *falloir*, *savoir*. To implement such massive dictionaries would require both hardware and software solutions for looking up words in dictionaries without having to load dictionaries into memory.

At the present no questions or exclamations are handled. The stage is set, however, for these types of sentences to be dealt with, since the major addition to the system to enable it to handle such sentences would be in the existing parser and generation modules.

---

[4] This is not to say that the parser is unable to handle discontinuous sentence components. See the present rules for handling negative forms in the "Parser - event" file in Appendix 4.

One of Ordinotrad's major weaknesses is its handling of multi-word fixed expressions, which, at the present, are generally dealt with component by component. There are some allowances at the present for fixed expressions in the analysis of prepositions (see the "Parser - preposition" file in Appendix 4), but this needs to be expanded.

CONCLUSION

In terms of the twofold purpose considered at the outset, it can be said that it is feasible to implement case grammar principles and, therefore, limited semantics in a machine translation system which also uses syntactic analysis, and thus to reflect to that extent the notion of "meaning-based translation." It is also feasible to create a reasonable, kernel MT system using this approach with roughly 9 000 lines of Prolog code. The system presently runs smoothly on a Macintosh Plus with 2 Mb of RAM and two floppy disk drives. It therefore could, with some dictionary expansion, be integrated as a tool into a translator's work station. Furthermore, the program could be adapted to run under an implementation of Prolog designed for the MS-DOS environment.

A dream of any student of Artificial Intelligence—and I consider myself a student of translation with a strong interest in AI—is to create something useful that incorporates a knowledge base (e.g., the various facts stored in Ordinotrad in its dictionaries, taxonomy, lists of verb endings, and so forth) and a means for analyzing and judging input through reasoning based on that knowledge. I believe that Ordinotrad is such a creation and, therefore, an AI machine. I also realize that what I have achieved in this thesis is not a final answer in the field of machine translation, but only an exploration into one of the ever-increasing possibilities in this field.

Appendix 1

Case List Used in Ordinotrad

The case labels used in Ordinotrad are listed and defined below. Each label is described in terms of something which is capable of playing the role identified by the label.

Central Roles

The first part of the case list consists of labels for roles important to the event in a proposition.

The *Agent* is something which does an action described by an event; that is, the doer.

The *Instrument* is something used to carry out an action described by an event.

The *Beneficiary* is something that is advantaged or disadvantaged by an action or state described by an event.

The *Affected* is something that undergoes, is affected by, or experiences an action described by an event. It may also be something that is renamed or modified by the Attribute of a state described by an event.

The *Result* is something which is produced by an action described by an event.

The *Source* is something considered as the place where an Agent begins an action described by an event or something considered as the place from which an Affected moves or is moved.

The *Destination* is something considered as the place where an Agent concludes an action described by an event or something considered as the place to which an Affected moves or is moved.

The *Attribute* is something which renames or modifies the Affected of a state described by an event.

Peripheral Roles

The second part of the list consists of labels for underlying relations between sentence components and the sentence as a whole. They do not have a specific role in relation to the event indicated in a sentence but affect how the overall event and filled case-roles are perceived.

Location identifies the spatial placement of an action or state described by an event.

Time identifies the temporal placement of an action or state described by an event. It indicates when an action began or ended, its point in time, or its duration.

Manner is the qualification of the way an action described by an event was, is being, or will be accomplished.

Cause is something which instigates an action described by an event but does not do it, an action to be done, or something which is perceived as the end to which an action is directed.

Appendix 2

Taxonomy

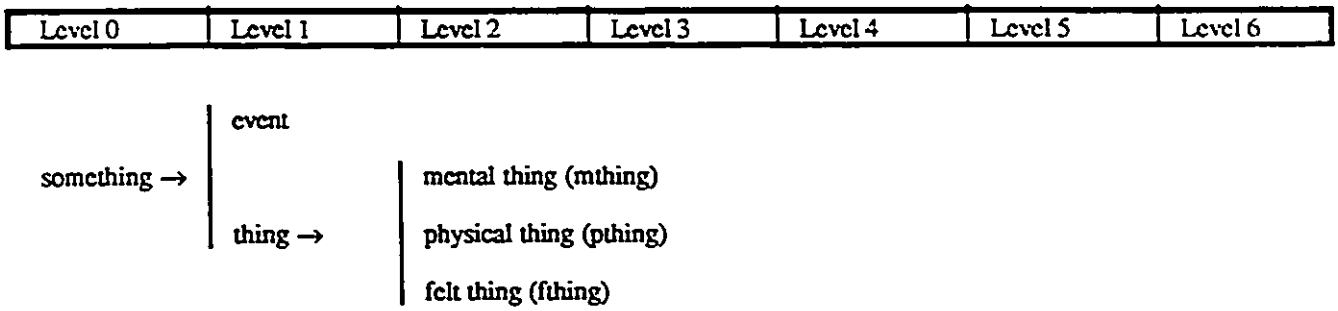
An taxonomy provides a description of the nature of "something" and of its relations to other "things." The taxonomy created for Ordinotrad provides it with a *Weltanschauung*, whereby it can make the logical inferences required by the semantic analyses in its case-related methodology.

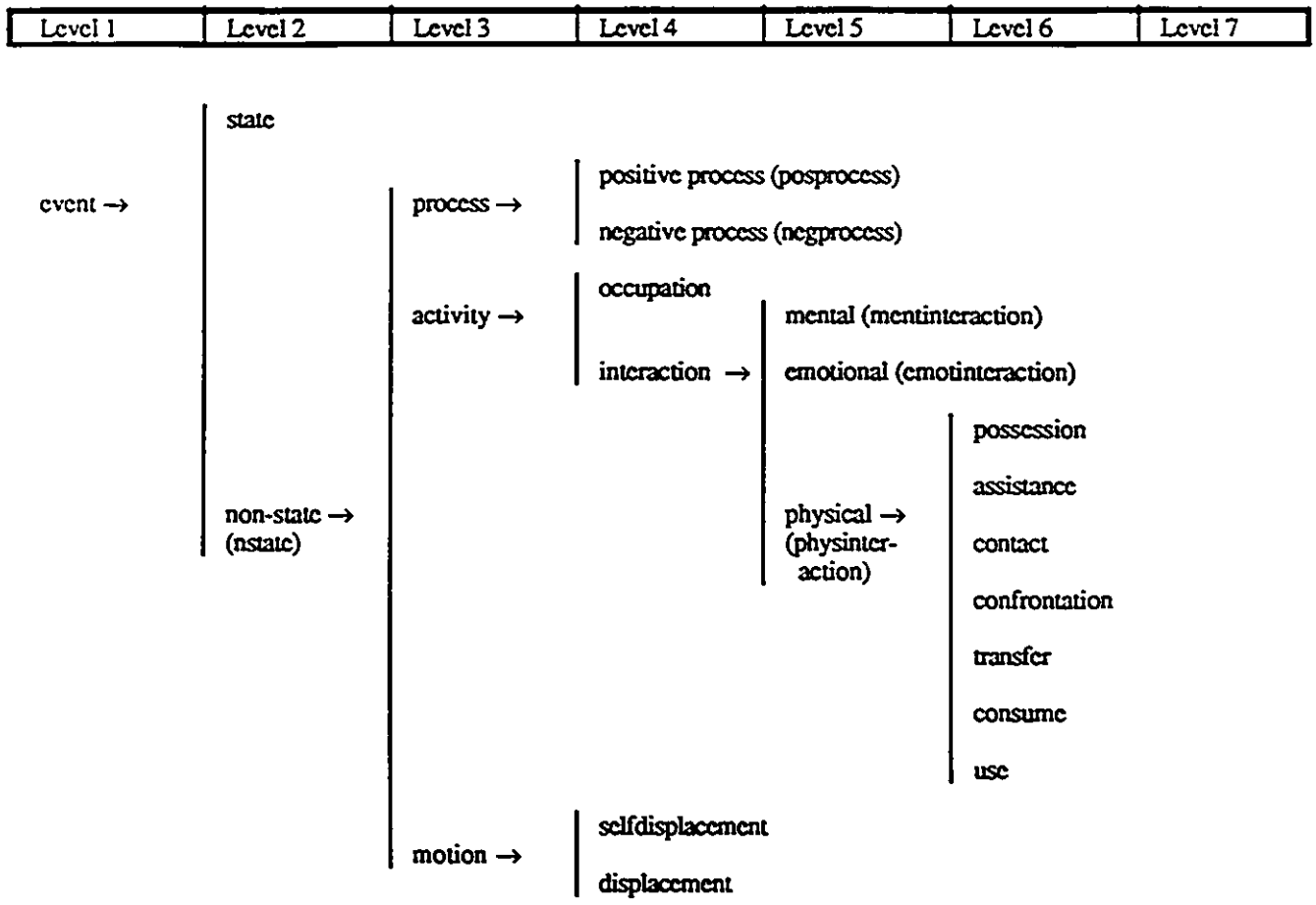
The words used in this taxonomy are deemed to be the "intensional" meanings of any number of things, i.e., the set of attributes which characterize the thing without referring to any actual object.[1] Each label in the taxonomy can also be a "class" in which other labels may be a member. Labels at the same level are deemed to be mutually exclusive, yet any reality may be given more than one label. For example, '*parole* ' may be considered to be a member of the class of mental interactions as well as a social phenomenon.

This component of Ordinotrad is easily susceptible to criticism (I, myself, have drawn up half a dozen alternatives), but as a programmer with a deadline, I could not hope to keep changing the taxonomy in the code, as long as the existing one is effective. The implementation of the following diagram in the program itself can be found in Appendix 4, in the "Taxonomy" file.

---

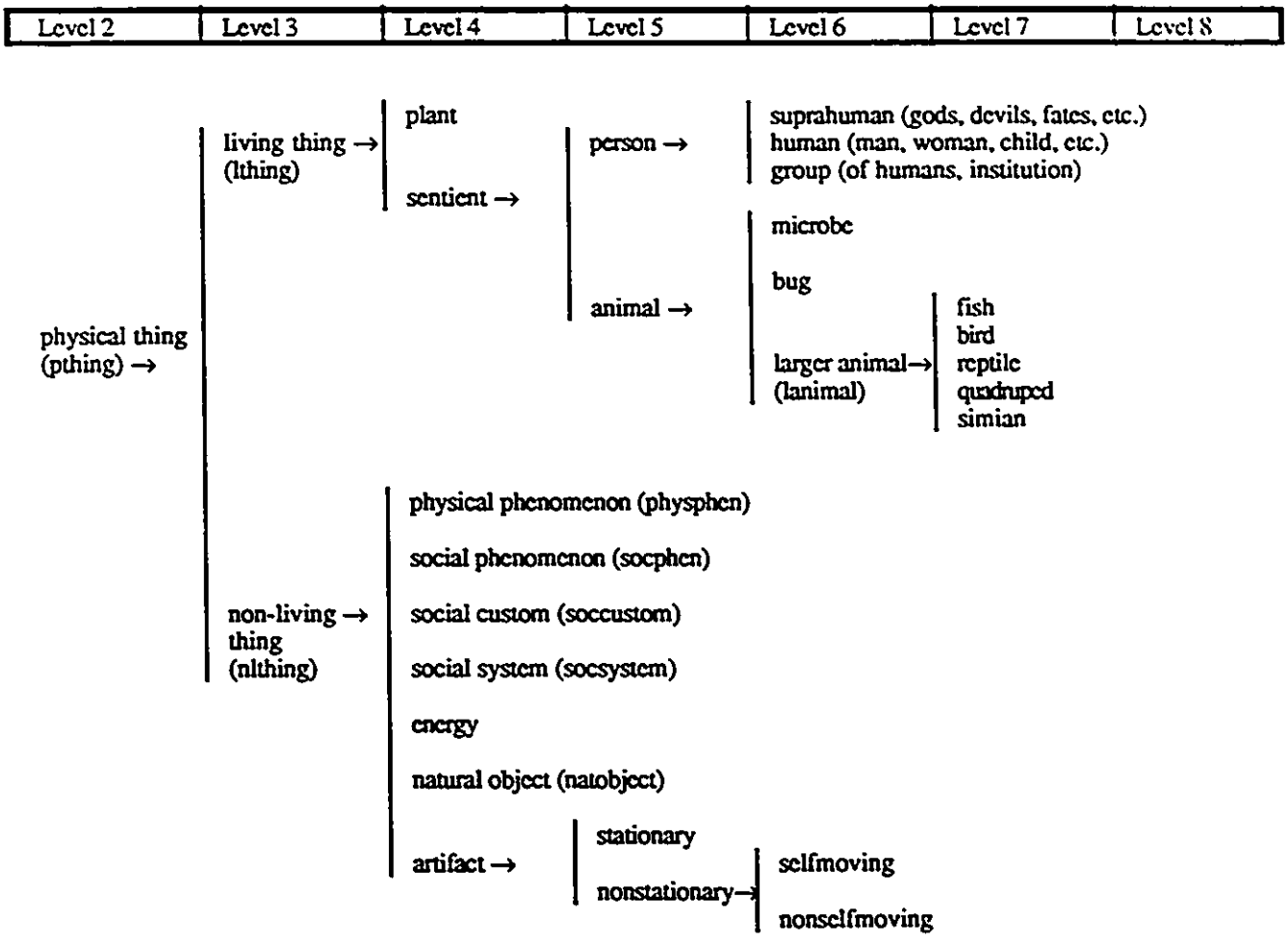
[1] Martha W. Evens, et al., *Lexical-semantic relations: a comparative survey*, (Carbondale, USA, & Edmonton, Canada: Linguistic Research Inc., 1980), 28. Note that intensional meanings are distinguished from extensional ones, which refer to the things which a term names. "A strict extensional approach is limited to forming sets of things and makes little attempt to posit conceptual underpinnings or otherwise account for the nature of distribution of the sets of items." ibid., 28.


Level 0
Level 1
Level 2
Level 3
Level 4
Level 5
Level 6
something →
event
thing →
mental thing (mthing)
physical thing (pthing)
felt thing (fthing)


Level 1
Level 2
Level 3
Level 4
Level 5
Level 6
Level 7
event →
state
non-state → (nstate)
process →
positive process (posprocess)
negative process (negprocess)
activity →
occupation
interaction →
mental (mentinteraction)
emotional (emotinteraction)
physical → (physinter-action)
possession
assistance
contact
confrontation
transfer
consume
use
motion →
selfdisplacement
displacement

Level 2	Level 3	Level 4	Level 5	Level 6	Level 7	Level 8
mental thing → (mthing)	volition					
	intention					
	subject matter → (submatter)	theory				
		science (i.e., a field of study, one of the sciences)				
		knowledge				
		rule	declaration			
		argument →	question			
		thought	command			
		relation (see next page)				
		structure →	series			
			hierarchy			
			network			
			mass			
	construct →	measurement →	numeral			
			unit			
			standard			
		location →	point in space (pspace)			
			distance			
		time →	point in time (ptime)			
			duration			
		manner				
		frequency				
		intensity				
		attitude →	affirmation			
			negation			
			doubt			

Level 4	Level 5	Level 6	Level 7	Level 8	Level 9	Level 10
relation →	thing/event → relation (terelation)	agent				
		instrument				
		beneficiary				
		affected				
		result				
		source				
		destination				
	thing/thing relation → (trelation)	cause (i.e., a causal relation)				
		purpose				
		condition (i.e. contingency)				
		complement				
		accompaniment				
		possessor				
		comparison				
		equality				
		inferiority				
		superiority				
		manifestation				
		derivation →	none			
			part			
			element			
			organ			
			abbreviation			
		attribute →	state (i.e., condition in the sense of "état")			
			quantity			
			nature			
			shape			
			name			
			quality			


Level 2
Level 3
Level 4
Level 5
Level 6
Level 7
Level 8
physical thing (pthing) →
living thing → (lthing)
plant
sentient →
person →
suprahuman (gods, devils, fates, etc.)
human (man, woman, child, etc.)
group (of humans, institution)
animal →
microbe
bug
larger animal → (lanimal)
fish
bird
reptile
quadruped
simian
non-living → thing (nlthing)
physical phenomenon (physphen)
social phenomenon (socphen)
social custom (soccustom)
social system (socsystem)
energy
natural object (natobject)
artifact →
stationary
nonstationary →
selfmoving
nonselfmoving

Level 2	Level 3	Level 4	Level 5	Level 6	Level 7	Level 8
felt thing → (fthing)	personal emotion (pemotion)					
	interpersonal emotion (iemotion)					
	abstract pressure (abspressure)					
	physical pressure (physpressure)					

References

Dahlgren, K. & J. McDowell, "Kind types in knowledge representation," *Proceedings of COLING 86* (Bonn, 1986): 216-221.

Nirenburg, Sergei, Victor Raskin & Allen B. Tucker. "The structure of interlingua in TRANSLATOR." *Machine Translation: Theoretical and methodological studies.* Sergei Nirenburg, ed. Studies in Natural Language Processing. Cambridge: Cambridge University Press, 1987. pp. 90-113.

Sakamoto, Yoshiyuki, Tetsuya Ishikawa, & Masayuki Satoh, "Concept and Structure of Semantic Markers for Machine Translation in Mu-Project," *Proceedings of COLING 86* (Bonn, 1986): 13-19.

Schubert, Lenhart K., Mary Angela Papalaskaris, & Jay Taugher, "Determining Type, Part, Colour, and Time Relationships," *Computer* 16.10 (1983): 53-60.

Appendix 3

Translation Results

Typical results for each segment of analysis, transfer and generation are shown by the following recorded portions of a session using Ordinotrad. Following this transcript are a series of French files and their English translation produced by Ordinotrad. Any comments that I have added here and which are not part of the transcript are printed in the present font. The transcript itself is printed here in a non-serif font, with some underlining added for the sake of presentation. Input provided by the user is shown in boldface. The contents of files read or written by Ordinotrad are enclosed in shadowed boxes.

In order to demonstrate each segment of the process, I created three short files which would provide an overall view of what Ordinotrad can do. The first file consists of affirmative statements, the second of negative ones, and the third some "mentions" (things like names, addresses, "Encl.", "cc.:," etc. The file names are, respectively, "short eg. 1", "short eg. 2", and "short eg. 3"). The reason for these files being so short is not that Ordinotrad cannot handle longer ones; it is because the interim results are so spacious that they become too prohibitive to print. In any event, interim results for any sentence can always be reproduced on screen by Ordinotrad. Note that lines beginning with a "?-" are Prolog prompts at which the user calls the appropriate rules which make everything happen.

The file "short eg. 1" contains the following data.

```
Monsieur Untel leur a adressé trois lettres.
Je suis très heureux d'avoir reçu votre lettre de 15 juin 1988.
Les trois premières lettres complètes que je reçois
deviendront la documentation de base.
```

The call is made to have the above file translated. Interim results are displayed because of a previous call to turn interim tracing on (cf. Chapter 4).

```
?- translate('short eg. 1').

NOW TRANSLATING:
Monsieur Untel leur a adressé trois lettres.

RESULTS OF PREPROCESSING
w(monsieur,cap)
w(untel,cap)
w(leur,low)
w(a,low)
w(adressé,low)
w(trois,low)
w(lettres,low)
p(.)

Type any letter and a return to continue.
==> V

RESULTS OF MORPHOLOGICAL ANALYSIS BASED ON DICTIONARY LOOKUP
lu(cap,[noun-(monsieur,m,s,proper,count,[human])])
lu(cap,[noun-(untel,m,s,proper,count,[person,name])])
lu(low,[adj-(leur,poss,(m,f),s,[ttrelation]),pro-(leur,refl,3,pl,(m,f),[person])])
lu(low,[verb-(avoir,(pres,ind,3,s),[possession-[agent-(r,[something],nil),affected-
    (r,[something],nil),beneficiary-(o,[event,person],for)]]),string-a])
lu(low, [verb-(adresser, (past-part, m, s), [transfer-[agent-(r, [person], nil), beneficiary-(r, [sentient],
    à), instrument-(o, [artifact], dans), affected-(r, [artifact], nil), source-(o, [pspace], de),
    destination-(o, [pspace], à)], mentinteraction-[agent-(r, [person, artifact], nil), beneficiary-(r,
    [person], nil), affected-(o, [mthing, fthing], nil)]])])
lu(low,[adj-(trois,cardinal,(m,f),pl,[quantity])])
lu(low,[noun-(lettre,f,pl,common,count,[nonselfmoving,submatter])])
p(pct,.)

Type any letter and a return to continue.
==> I
   Subject-verb agreement succeeded.
```

For the following results, consider each indentation as a successive generation of a tree. See Chapter 4 for a complete description of all data structures.

```
RESULTS OF PARSING
proposition
    independent_prop
        modalities
            past
            ind
            active
            affirmative
            declarative
        event
```

```
            lu(low, verb-(adresser, (past, ind, 3, _979), [transfer-[agent-(r, [person], nil),
                beneficiary-(r, [sentient ], à), instrument-(o, [artifact], dans), affected-(r, [artifact],
                nil), source-(o, [pspace], de), destination-(o , [pspace], à)], mentinteraction-
                [agent-(r, [person, artifact], nil), beneficiary-(r, [person], nil), affected- (o, [mthing,
                fthing], nil)]]))
        np
            name
                name
                    lu(cap,noun-(monsieur,m,s,proper,count,[human]))
                name
                    lu(cap,noun-(untel,m,s,proper,count,[person,name]))
        np
            np
                lu(low,pro-(leur,refl,3,pl,(m,f),[person]))
        np
            np
                noun
                    lu(low,noun-(lettre,f,pl,common,count,[nonselfmoving,submatter]))
                modifiers
                    lu(low,adj-(trois,cardinal,f,pl,[quantity]))
                    nil

Type any letter and a return to continue.
==> c

RESULTS OF CASE PROCESSING
proposition
    independent_prop
        modalities
            past
            ind
            active
            affirmative
            declarative
        event
            lu(low,verb-(adresser,(past,ind,3,_979),transfer))
        agent
            np
                name
                    name
                        lu(cap,noun-(monsieur,m,s,proper,count,[human]))
                    name
                        lu(cap,noun-(untel,m,s,proper,count,[person,name]))
        beneficiary
            np
                np
                    lu(low,pro-(leur,refl,3,pl,(m,f),[person]))
        affected
            np
                np
                    noun
                        lu(low,noun-(lettre,f,pl,common,count,[nonselfmoving,submatter]))
                    modifiers
                        lu(low,adj-(trois,cardinal,f,pl,[quantity]))
                        nil
```

```
Type any letter and a return to continue.
==> t

RESULTS OF LEXICAL TRANSFER
proposition
    independent_prop
        modalities
            past
            ind
            active
            affirmative
            declarative
        frame
            agent
                r
                person
                nil
            instrument
                o
                artifact
                in
            beneficiary
                r
                sentient
                to
            affected
                r
                artifact
                nil
            source
                o
                pspace
                from
            destination
                o
                pspace
                to
        event
            lu(low,verb-(send,(past,ind,3,_979),transfer))
        agent
            np
                name
                    name
                        lu(cap,noun-(sir,s,proper,count))
                    name
                        lu(cap,noun-(so and so,s,proper,count))
        beneficiary
            np
                np
                    lu(low,pro-(them,refl,3,pl))
        affected
            np
                np
                    noun
```

```
        lu(low,noun-(letter,pl,common,count))
    modifiers
        lu(low,adj-(three,cardinal))
        nil
```

Type any letter and a return to continue.
==> o

TRANSLATION
Mr. So and so sent three letters to them.

Translation operations took 1 minute and 11 seconds.

Do you wish to translate the next sentence from your file?
[yes,y,no,n] ==> y

NOW TRANSLATING:
Je suis très heureux d'avoir reçu votre lettre de 15 juin 1988.

RESULTS OF PREPROCESSING

```
w(je,cap)
w(suis,low)
w(très,low)
w(heureux,low)
w(de,low)
w(avoir,low)
w(reçu,low)
w(votre,low)
w(lettre,low)
w(de,low)
n(15,cardinal)
w(juin,low)
n(1988,cardinal)
p(.)
```

Type any letter and a return to continue.
==> r

RESULTS OF MORPHOLOGICAL ANALYSIS BASED ON DICTIONARY LOOKUP

```
lu(cap,[pro-(je,pers,1,s,(m,f),[person])])
lu(low, [verb-(être, (pres, ind, 1, s), [state-[affected-(r, [something], nil), attribute-(r, [something],
    nil), beneficiary-(o,[sentient],nil)]])])
lu(low,[adv-(très,intensity)])
lu(low,[adj-(heureux,desc,m,(s,pl),[attribute])])
lu(low,[prep-de])
lu(low, [noun-(avoir, m, s, common, count, [pthing, comparison]), verb-(avoir, inf, [possession-
    [agent-(r, [something], nil), affected-(r, [something], nil), beneficiary-(o, [event, person],
    for)]])])
lu(low, [verb-(recevoir, (past-part, m, s), [physinteraction-[agent-(r, [person], nil), affected-(r,
    [animal, plant, natobject, energy, artifact], nil), source-(o, [pspace, person], de)],
    mentinteraction-[agent-(r, [event, person], nil), affected-(r, [mthing], nil), source-(o, [event],
    de)]])])
lu(low,[adj-(votre,poss,(m,f),s,[ttrelation])])
lu(low,[noun-(lettre,f,s,common,count,[nonselfmoving,submatter])])
```

```
lu(low,[prep-de])
n(15,cardinal)
lu(low,[noun-(juin,m,s,proper,count,[time,name])])
n(1988,cardinal)
p(pct,.)

Type any letter and a return to continue.
==> L
   Subject-verb agreement succeeded.

RESULTS OF PARSING
proposition
    independent_prop
        modalities
            pres
            ind
            active
            affirmative
            declarative
        event
            lu(low,verb-(être,(pres,ind,1,s),[state-[affected-(r,[something],nil),attribute-
                (r,[something],nil),beneficiary-(o,[sentient],nil)]]))
        np
            np
                lu(cap,pro-(je,pers,1,s,(m,f),[person]))
        adj
            lu
                lu(low,adv-(très,intensity))
                lu(low,adj-(heureux,desc,m,(s,pl),[attribute]))
        event_prop
            event_prop
                modalities
                    past
                    participle
                    active
                    affirmative
                    declarative
                event
                    lu(low,verb-(recevoir,(past,participle,nil,nil),[physinteraction-[agent-
                        (r,[person],nil),affected-(r,[animal,plant,natobject,energy,artifact],nil),
                        source-(o,[pspace,person],de)],mentinteraction-[agent-
                        (r,[event,person],nil),affected-(r,[mthing],nil),source-(o,[event],de)]]))
                prep
                    lu(low,prep-de)
                np
                    np
                        noun
                            lu(low,noun-(lettre,f,s,common,count,[nonselfmoving, submatter]))
                        modifiers
                            lu(low,adj-(votre,poss,f,s,[ttrelation]))
                            qualifying_prep_phrase
                                qualifying_prep_phrase
                                    prep
                                        lu(low,prep-de)
                                    date
```

```
                                        nil
                                        weekday
                                            nil
                                        day
                                            number(15,cardinal)
                                        month
                                            lu(low,noun-(juin,m,s,proper,count,[time, name]))
                                        year
                                            number(1988,cardinal)

Type any letter and a return to continue.
==> o

RESULTS OF CASE PROCESSING
proposition
    independent_prop
        modalities
            pres
            ind
            active
            affirmative
            declarative
        event
            lu(low,verb-(être,(pres,ind,1,s),state))
        affected
            np
                np
                    lu(cap,pro-(je,pers,1,s,(m,f),[person]))
        attribute
            adj
                lu
                    lu(low,adv-(très,intensity))
                    lu(low,adj-(heureux,desc,m,(s,pl),[attribute]))
        cause
            event_prop
                event_prop
                    modalities
                        past
                        participle
                        active
                        affirmative
                        declarative
                    event
                        lu(low,verb-(recevoir,(past,participle,nil,nil),physinteraction))
                    prep
                        lu(low,prep-de)
                    affected
                        np
                            np
                                noun
                                    lu(low,noun-(lettre,f,s,common,count,[nonselfmoving,
                                submatter]))
                                modifiers
                                    lu(low,adj-(votre,poss,f,s,[ttrelation]))
                                    qualifying_prep_phrase
```

```
                                    qualifying_prep_phrase
                                        prep
                                            lu(low,prep-de)
                                        date
                                            nil
                                            weekday
                                                nil
                                            day
                                                number(15,cardinal)
                                            month
                                                lu(low,noun-(juin,m,s,proper,count,[time,
                                                    name]))
                                            year
                                                number(1988,cardinal)
```

Type any letter and a return to continue.
==> e

RESULTS OF LEXICAL TRANSFER

```
proposition
    independent_prop
        modalities
            pres
            ind
            active
            affirmative
            declarative
        frame
            beneficiary
                o
                person
                event
                for
            affected
                r
                something
                nil
            attribute
                r
                something
                nil
        event
            lu(low,verb-(be,(pres,ind,1,s),state))
        affected
            np
                np
                    lu(cap,pro-(I,pers,1,s))
        attribute
            adj
                lu
                    lu(low,adv-(very,intensity))
                    lu(low,adj-(happy,desc))
        cause
            event_prop
```

```
event_prop
    modalities
        past
        participle
        active
        affirmative
        declarative
    frame
        affected
            r
            animal
            plant
            natobject
            energy
            artifact
            nil
        source
            o
            pspace
            person
            de
    event
        lu(low,verb-(receive,(past,participle,nil,nil),physinteraction))
    prep
        lu(low,prep-[functional_prep])
    affected
        np
            np
                noun
                    lu(low,noun-(letter,s,common,count))
                modifiers
                    lu(low,adj-(your,poss))
                    qualifying_prep_phrase
                        qualifying_prep_phrase
                            prep
                                lu(low,prep-[functional_prep])
                            date
                                nil
                                weekday
                                    nil
                                day
                                    number(15,cardinal)
                                month
                                    lu(low,noun-(june,s,proper,count))
                                year
                                    number(1988,cardinal)
```

Type any letter and a return to continue.
==> w

TRANSLATION
I am very happy having received your letter of June 15, 1988.

Translation operations took 1 minute and 22 seconds.

Do you wish to translate the next sentence from your file?
[yes,y,no,n] ==> **y**

NOW TRANSLATING:
Les trois premières lettres complètes que je reçois deviendront la documentation de base.

RESULTS OF PREPROCESSING
w(les,cap)
w(trois,low)
w(premières,low)
w(lettres,low)
w(complètes,low)
w(que,low)
w(je,low)
w(reçois,low)
w(deviendront,low)
w(la,low)
w(documentation,low)
w(de,low)
w(base,low)
p(.)

Type any letter and a return to continue.
==> **e**

RESULTS OF MORPHOLOGICAL ANALYSIS BASED ON DICTIONARY LOOKUP
lu(cap,[art-(les,def,(m,f),pl),pro-(les,rel,3,pl,(m,f),_955)])
lu(low,[adj-(trois,cardinal,(m,f),pl,[quantity])])
lu(low,[adj-(premier,ordinal,f,pl,[ttrelation])])
lu(low,[noun-(lettre,f,pl,common,count,[nonselfmoving,submatter])])
lu(low,[adj-(complet,desc,f,pl,[attribute])])
lu(low,[conj-(que,sub),pro-(que,rel,3,(s,pl),(m,f),_1648)])
lu(low,[pro-(je,pers,1,s,(m,f),[person])])
lu(low,[verb-(recevoir,(pres,ind,(1,2),s),[physinteraction-[agent-(r,[person],nil),affected-
(r,[animal,plant,natobject,energy,artifact],nil),source-
(o,[pspace,person],de)],mentinteraction-[agent-(r,[event,person],nil),affected-
(r,[mthing],nil),source-(o,[event],de)]])])
lu(low,[verb-(devenir,(fut,ind,3,pl),[state-[affected-(r,[person,event,mthing,artifact],nil),attribute-
(r,[something],nil)],posprocess-[affected-(r,[person,event,mthing,artifact],nil),result-
(r,[something],nil)]])])
lu(low,[art-(la,def,f,s),pro-(la,rel,3,s,f,_1847)])
lu(low,[noun-(documentation,f,s,common,mass,[artifact])])
lu(low,[prep-de])
lu(low,[noun-(base,f,s,common,count,[pspace,nonselfmoving])])
p(pct,.)

Type any letter and a return to continue.
==> **n**
Subject-verb agreement succeeded.
Subject-verb agreement succeeded.

RESULTS OF PARSING
proposition
independent_prop

```
modalities
    fut
    ind
    active
    affirmative
    declarative
event
    lu(low, verb-(devenir, (fut, ind, 3, pl), [state-[affected-(r, [person, event, mthing,
        artifact], nil), attribute-(r, [something], nil)], posprocess-[affected-(r, [person,
        event, mthing, artifact], nil), result-(r, [something], nil)]]])
np
    np
        noun
            lu(low,noun-(lettre,f,pl,common,count,[nonselfmoving,submatter]))
        modifiers
            lu(cap,art-(les,def,f,pl))
            lu(low,adj-(trois,cardinal,f,pl,[quantity]))
            lu(low,adj-(premier,ordinal,f,pl,[ttrelation]))
            lu(low,adj-(complet,desc,f,pl,[attribute]))
            relative_prop
                relative_prop
                    modalities
                        pres
                        ind
                        active
                        affirmative
                        declarative
                    event
                        lu(low,verb-(recevoir,(pres,ind,1,s),[physinteraction-[agent-
                            (r,[person],nil),affected-(r,[animal,plant,natobject,
                            energy,artifact],nil),source-(o,[pspace,person],de)],
                            mentinteraction- [agent-(r,[event,person],nil),affected-
                            (r,[mthing],nil),source-(o,[event],de)]]))
                    np
                        np
                            lu(low,pro-(je,pers,1,s,(m,f),[person]))
                    np
                        np
                            lu(low,pro-(que,rel,3,(s,pl),(m,f),_1648))
np
    np
        noun
            lu(low,noun-(documentation,f,s,common,mass,[artifact]))
        modifiers
            lu(low,art-(la,def,f,s))
            qualifying_prep_phrase
                qualifying_prep_phrase
                    prep
                        lu(low,prep-de)
                    np
                        np
                            noun
                                lu(low,noun-(base,f,s,common,count,[pspace,
                                nonselfmoving]))
                            modifiers
```

```
                                    nil
                                    nil

Type any letter and a return to continue.
==> V

RESULTS OF CASE PROCESSING
proposition
    independent_prop
        modalities
            fut
            ind
            active
            affirmative
            declarative
        event
            lu(low,verb-(devenir,(fut,ind,3,pl),posprocess))
        affected
            np
                np
                    noun
                        lu(low,noun-(lettre,f,pl,common,count,[nonselfmoving,submatter]))
                    modifiers
                        lu(cap,art-(les,def,f,pl))
                        lu(low,adj-(trois,cardinal,f,pl,[quantity]))
                        lu(low,adj-(premier,ordinal,f,pl,[ttrelation]))
                        lu(low,adj-(complet,desc,f,pl,[attribute]))
                        relative_prop
                            relative_prop
                                modalities
                                    pres
                                    ind
                                    active
                                    affirmative
                                    declarative
                                event
                                    lu(low,verb-(recevoir,(pres,ind,1,s),physinteraction))
                                agent
                                    np
                                        np
                                            lu(low,pro-(je,pers,1,s,(m,f),[person]))
                                affected
                                    np
                                        np
                                            lu(low,pro-(que,rel,3,(s,pl),(m,f),[animal,plant,
                                                natobject, energy, artifact]))
        result
            np
                np
                    noun
                        lu(low,noun-(documentation,f,s,common,mass,[artifact]))
                    modifiers
                        lu(low,art-(la,def,f,s))
                        qualifying_prep_phrase
                            qualifying_prep_phrase
```

```
                    prep
                        lu(low,prep-de)
                    np
                        np
                            noun
                                lu(low,noun-(base,f,s,common,count,[pspace,
                                    nonselfmoving]))
                            modifiers
                                nil
                                nil
```

Type any letter and a return to continue.
==> l

RESULTS OF LEXICAL TRANSFER

```
proposition
    independent_prop
        modalities
            fut
            ind
            active
            affirmative
            declarative
        frame
            affected
                r
                person
                event
                mthing
                artifact
                nil
            result
                r
                something
                nil
        event
            lu(low,verb-(become,(fut,ind,3,pl),posprocess))
        affected
            np
                np
                    noun
                        lu(low,noun-(letter,pl,common,count))
                    modifiers
                        lu(cap,art-(the,def,nil,pl))
                        lu(low,adj-(three,cardinal))
                        lu(low,adj-(first,ordinal))
                        lu(low,adj-(complete,desc))
                        relative_prop
                            relative_prop
                                modalities
                                    pres
                                    ind
                                    active
                                    affirmative
```

```
                declarative
            frame
                agent
                    r
                    person
                    nil
                affected
                    r
                    animal
                    plant
                    natobject
                    energy
                    artifact
                    nil
                source
                    o
                    pspace
                    person
                    de
            event
                lu(low,verb-(receive,(pres,ind,1,s),physinteraction))
            agent
                np
                    np
                        lu(low,pro-(I,pers,1,s))
            affected
                np
                    np
                        lu(low,pro-(that,rel,3,pl))
result
    np
        np
            noun
                lu(low,noun-(documentation,s,common,mass))
            modifiers
                lu(low,art-(the,def,nil,s))
                qualifying_prep_phrase
                    qualifying_prep_phrase
                        prep
                            lu(low,prep-[functional_prep])
                        np
                            np
                                noun
                                    lu(low,noun-(base,s,common,count))
                                modifiers
                                    nil
                                    nil
```

Type any letter and a return to continue.
==> c

TRANSLATION
The first three complete letters that I receive will become documentation of base.

Translation operations took 1 minute and 23 seconds.

Do you wish to translate the next sentence from your file?
[yes,y,no,n] ==> **y**
There is no more to be translated in your file.

ORDINOTRAD™
© 1987, 1988 Victor Loewen

There will eventually be a translate menu. The following are now available.
To have a sentence translated with the results viewed on the screen, type "translate."
To have a file translated with the results viewed on the screen, type "translate(Filename)."
• Files to be translated must be located in the corpora folder.
To quit, select QUIT under the FILE menu or simply type "exit." and a return.
yes

The following transcript shows how the file "short eg. 1" could be translated directly to disk, with only progress reports shown on the screen. A special call is made to turn interim tracing off.

?- **trace_off.**
yes

?- **t2d('short eg. 1').**

Now translating file titled: short eg. 1
The translation will be in the corpora folder with the title: short eg. 1.English
Please stand by; you may be needed to answer a few questions.

NOW TRANSLATING:
Monsieur Untel leur a adressé trois lettres.
Preprocessing completed.
Morphological analysis completed.
Subject-verb agreement succeeded.
Parsing completed.
Case processing completed.
Transfer completed
TRANSLATION SAVED TO DISK:
Mr. So and so sent three letters to them.

NOW TRANSLATING:
Je suis très heureux d'avoir reçu votre lettre de 15 juin 1988.
Preprocessing completed.
Morphological analysis completed.
Subject-verb agreement succeeded.
Parsing completed.
Case processing completed.
Transfer completed

TRANSLATION SAVED TO DISK:
I am very happy having received your letter of June 15, 1988.

NOW TRANSLATING:
Les trois premières lettres complètes que je reçois deviendront la documentation de base.
Preprocessing completed.
Morphological analysis completed.
Subject-verb agreement succeeded.
Subject-verb agreement succeeded.
Parsing completed.
Case processing completed.
Transfer completed
TRANSLATION SAVED TO DISK:
The first three complete letters that I receive will become documentation of base.

Processing has been completed.
The translation is in the corpora folder with the title: short eg. 1.English
yes
?-

The file "short eg.1.English" which was created by Ordinotrad contains the following data.

Mr. So and so sent three letters to them.

I am very happy having received your letter of June 15, 1988.

The first three complete letters that I receive will become documentation of base.

Work on this file, including I/O and user interaction, took 0 minutes and 17 seconds.

The file "short eg. 2" contains the following data.

Tu n'as pas les trois lettres.
Nous ne vous payerons jamais pour votre travail.
Ayant lu votre lettre, ils ne sont plus heureux.

The call is made to have the above file translated.

```
?- trace_on.
  yes
?- translate('short eg. 2').

NOW TRANSLATING:
Tu n'as pas les trois lettres.

RESULTS OF PREPROCESSING
w(tu,cap)
w(ne,low)
w(as,low)
w(pas,low)
w(les,low)
w(trois,low)
w(lettres,low)
p(.)

Type any letter and a return to continue.
==> V

RESULTS OF MORPHOLOGICAL ANALYSIS BASED ON DICTIONARY LOOKUP
lu(cap,[pro-(tu,pers,2,s,(m,f),[person])])
lu(low,[adv-(ne,negation)])
lu(low,[verb-(avoir,(pres,ind,2,s),[possession-[agent-(r,[something],nil),affected-
     (r,[something],nil),beneficiary-(o,[event,person],for)]])])
lu(low,[noun-(pas,m,(s,pl),common,count,[unit]),adv-(pas,negation)])
lu(low,[art-(les,def,(m,f),pl),pro-(les,rel,3,pl,(m,f),_449)])
lu(low,[adj-(trois,cardinal,(m,f),pl,[quantity])])
lu(low,[noun-(lettre,f,pl,common,count,[nonselfmoving,submatter])])
p(pct,.)

Type any letter and a return to continue.
==> I
    Subject-verb agreement succeeded.

RESULTS OF PARSING
proposition
```

```
    independent_prop
        modalities
            pres
            ind
            active
            negative
                pas
            declarative
        event
            lu(low,verb-(avoir,(pres,ind,2,s),[possession-[agent-(r,[something],nil),affected-
                (r,[something],nil),beneficiary-(o,[event,person],for)]]))
        np
            np
                lu(cap,pro-(tu,pers,2,s,(m,f),[person]))
        np
            np
                noun
                    lu(low,noun-(lettre,f,pl,common,count,[nonselfmoving,submatter]))
                modifiers
                    lu(low,art-(les,def,f,pl))
                    lu(low,adj-(trois,cardinal,f,pl,[quantity]))
                    nil

Type any letter and a return to continue.
==> c

RESULTS OF CASE PROCESSING
proposition
    independent_prop
        modalities
            pres
            ind
            active
            negative
                pas
            declarative
        event
            lu(low,verb-(avoir,(pres,ind,2,s),_ossession))
        agent
            np
                np
                    lu(cap,pro-(tu,pers,2,s,(m,f),[person]))
        affected
            np
                np
                    noun
                        lu(low,noun-(lettre,f,pl,common,count,[nonselfmoving,submatter]))
                    modifiers
                        lu(low,art-(les,def,f,pl))
                        lu(low,adj-(trois,cardinal,f,pl,[quantity]))
                        nil

Type any letter and a return to continue.
==> t
```

RESULTS OF LEXICAL TRANSFER

```
proposition
    independent_prop
        modalities
            pres
            ind
            active
            negative
                pas
            declarative
        frame
            agent
                r
                something
                nil
            affected
                r
                something
                nil
            beneficiary
                o
                event
                person
                for
        event
            lu(low,verb-(have,(pres,ind,2,s),possession))
        agent
            np
                np
                    lu(cap,pro-(you,pers,2,s))
        affected
            np
                np
                    noun
                        lu(low,noun-(letter,pl,common,count))
                    modifiers
                        lu(low,art-(the,def,nil,pl))
                        lu(low,adj-(three,cardinal))
                        nil
```

Type any letter and a return to continue.
==> o

TRANSLATION
You do not have the three letters.

Translation operations took 0 minutes and 37 seconds.

Do you wish to translate the next sentence from your file?
[yes,y,no,n] ==> y

NOW TRANSLATING:
Nous ne vous payerons jamais pour votre travail.

RESULTS OF PREPROCESSING
w(nous,cap)
w(ne,low)
w(vous,low)
w(payerons,low)
w(jamais,low)
w(pour,low)
w(votre,low)
w(travail,iow)
p(.)

Type any letter and a return to continue.
==> r

RESULTS OF MORPHOLOGICAL ANALYSIS BASED ON DICTIONARY LOOKUP
lu(cap,[pro-(nous,pers,1,pl,(m,f),[person]),pro-(nous,refl,1,pl,(m,f),[person])])
lu(low,[adv-(ne,negation)])
lu(low,[pro-(vous,pers,2,(s,pl),(m,f),[person]),pro-(vous,refl,2,(s,pl),(m,f),[person])])
lu(low,[verb-(payer,(fut,ind,1,pl),[transfer-[agent-(r,[person],nil),instrument-
(o,[artifact],nil),beneficiary-(o,[person],à),affected-(o,[unit,artifact],nil)]])])
lu(low,[adv-(jamais,negation)])
lu(low,[prep-pour])
lu(low,[adj-(votre,poss,(m,f),s,[ttrelation])])
lu(low,[noun-(travail,m,s,common,count,[posprocess,occupation])])
p(pct,.)

Type any letter and a return to continue.
==> L
Subject-verb agreement succeeded.

RESULTS OF PARSING
proposition
  independent_prop
    modalities
      fut
      ind
      active
      negative
        jamais
      declarative
    event
      lu(low,verb-(payer,(fut,ind,1,pl),[transfer-[agent-(r,[person],nil),instrument-
        (o,[artifact],nil),beneficiary-(o,[person],à),affected-(o,[unit,artifact],nil)]])
    np
      np
        lu(cap,pro-(nous,pers,1,pl,(m,f),[person]))
    np
      np
        lu(low,pro-(vous,refl,2,(s,pl),(m,f),[person]))
    prep_phrase
      prep_phrase
        prep
          lu(low,prep-pour)
        np

```
                np
                    noun
                        lu(low,noun-(travail,m,s,common,count,[posprocess,occupation]))
                    modifiers
                        lu(low,adj-(votre,poss,m,s,[ttrelation]))
                        nil

Type any letter and a return to continue.
==> o

RESULTS OF CASE PROCESSING
proposition
    independent_prop
        modalities
            fut
            ind
            active
            negative
                jamais
            declarative
        event
            lu(low,verb-(payer,(fut,ind,1,pl),transfer))
        agent
            np
                np
                    lu(cap,pro-(nous,pers,1,pl,(m,f),[person]))
        beneficiary
            np
                np
                    lu(low,pro-(vous,refl,2,(s,pl),(m,f),[person]))
        cause
            prep_phrase
                prep_phrase
                    prep
                        lu(low,prep-pour)
                    np
                        np
                            noun
                                lu(low,noun-(travail,m,s,common,count,[posprocess,
                                occupation]))
                            modifiers
                                lu(low,adj-(votre,poss,m,s,[ttrelation]))
                                nil

Type any letter and a return to continue.
==> e

RESULTS OF LEXICAL TRANSFER

proposition
    independent_prop
        modalities
            fut
            ind
            active
```

```
        negative
            jamais
        declarative
    frame
        agent
            r
            person
            nil
        instrument
            o
            artifact
            with
        beneficiary
            o
            person
            nil
        affected
            o
            unit
            artifact
            nil
    event
        lu(low,verb-(pay,(fut,ind,1,pl),transfer))
    agent
        np
            np
                lu(cap,pro-(we,pers,1,pl))
    beneficiary
        np
            np
                lu(low,pro-(you,refl,2,s,pl))
    cause
        prep_phrase
            prep_phrase
                prep
                    lu(low,prep-[for])
                np
                    np
                        noun
                            lu(low,noun-(work,s,common,mass))
                        modifiers
                            lu(low,adj-(your,poss))
                            nil
```

Type any letter and a return to continue.
==> w

TRANSLATION
We never will pay you for your work.

Translation operations took 1 minute and 3 seconds.

Do you wish to translate the next sentence from your file?
[yes,y,no,n] ==> y

NOW TRANSLATING:
Ayant lu votre lettre, ils ne sont plus heureux.

RESULTS OF PREPROCESSING
```
w(ayant,cap)
w(lu,low)
w(votre,low)
w(lettre,low)
p(,)
w(ils,low)
w(ne,low)
w(sont,low)
w(plus,low)
w(heureux,low)
p(.)
```

Type any letter and a return to continue.
==> e

RESULTS OF MORPHOLOGICAL ANALYSIS BASED ON DICTIONARY LOOKUP
```
lu(cap,[verb-(avoir,(pres-part,m,s),[possession-[agent-(r,[something],nil),affected-
    (r,[something],nil),beneficiary-(o,[event,person],for)]])])
lu(low,[verb-(lire,(past-part,m,s),[mentinteraction-[agent-(r,[human],nil),instrument-
    (o,[artifact],avec),affected-(o,[submatter,artifact],nil)]])])
lu(low,[adj-(votre,poss,(m,f),s,[ttrelation])])
lu(low,[noun-(lettre,f,s,common,count,[nonselfmoving,submatter])])
p(pct,,)
lu(low,[pro-(ils,pers,3,pl,m,[person])])
lu(low,[adv-(ne,negation)])
lu(low,[verb-(être,(pres,ind,3,pl),[state-[affected-(r,[something],nil),attribute-
    (r,[something],nil),beneficiary-(o,[sentient],nil)]])])
lu(low,[adv-(plus,negation)])
lu(low,[adj-(heureux,desc,m,(s,pl),[attribute])])
p(pct,.)
```

Type any letter and a return to continue.
==> n
Subject-verb agreement succeeded.

RESULTS OF PARSING
```
proposition
    subordinate_prop
        event_prop
            event_prop
                modalities
                    past
                    participle
                    active
                    affirmative
                    declarative
                event
                    lu(low,verb-(lire,(past,participle,use_ing,nil),[mentinteraction-[agent-
                        (r,[human],nil),instrument-(o,[artifact],avec),affected-
                        (o,[submatter,artifact],nil)]])
```

```
                np
                    np
                        noun
                            lu(low,noun-(lettre,f,s,common,count,[nonselfmoving,
                        submatter]))
                        modifiers
                            lu(low,adj-(votre,poss,f,s,[ttrelation]))
                            nil
    articulation
        p
        .
    independent_prop
        modalities
            pres
            ind
            active
            negative
                plus
            declarative
        event
            lu(low,verb-(être,(pres,ind,3,pl),[state-[affected-(r,[something],nil),attribute-
                (r,[something],nil),beneficiary-(o,[sentient],nil)]]))
        np
            np
                lu(low,pro-(ils,pers,3,pl,m,[person]))
        adj
            lu(low,adj-(heureux,desc,m,(s,pl),[attribute]))

Type any letter and a return to continue.
==> V

RESULTS OF CASE PROCESSING
proposition
    subordinate_prop
        event_prop
            event_prop
                modalities
                    past
                    participle
                    active
                    affirmative
                    declarative
                event
                    lu(low,verb-(lire,(past,participle,use_ing,nil),mentinteraction))
                affected
                    np
                        np
                            noun
                                lu(low,noun-(lettre,f,s,common,count,[nonselfmoving,
                            submatter]))
                            modifiers
                                lu(low,adj-(votre,poss,f,s,[ttrelation]))
                                nil
    articulation
        p
```

```
    independent_prop
        modalities
            pres
            ind
            active
            negative
                plus
            declarative
        event
            lu(low,verb-(être,(pres,ind,3,pl),state))
        affected
            np
                np
                    lu(low,pro-(ils,pers,3,pl,m,[person]))
        attribute
            adj
                lu(low,adj-(heureux,desc,m,(s,pl),[attribute]))

Type any letter and a return to continue.
==> l

RESULTS OF LEXICAL TRANSFER

proposition
    subordinate_prop
        event_prop
            event_prop
                modalities
                    past
                    participle
                    active
                    affirmative
                    declarative
                frame
                    instrument
                        o
                        artifact
                        avec
                    affected
                        o
                        submatter
                        artifact
                        nil
                event
                    lu(low,verb-(read,(past,participle,use_ing,nil),mentinteraction))
                affected
                    np
                        np
                            noun
                                lu(low,noun-(letter,s,common,count))
                            modifiers
                                lu(low,adj-(your,poss))
                                nil
    articulation
```

```
        p
            ,
    independent_prop
        modalities
            pres
            ind
            active
            negative
                plus
            declarative
        frame
            beneficiary
                o
                person
                event
                for
            affected
                r
                something
                nil
            attribute
                r
                something
                nil
        event
            lu(low,verb-(be,(pres,ind,3,pl),state))
        affected
            np
                np
                    lu(low,pro-(they,pers,3,pl))
        attribute
            adj
                lu(low,adj-(happy,desc))
```

Type any letter and a return to continue.
==> c

TRANSLATION
Having read your letter, they no longer are happy.

Translation operations took 1 minute and 8 seconds.

Do you wish to translate the next sentence from your file?
[yes,y,no,n] ==> y
There is no more to be translated in your file.

Using the "t2d" predicate, the file "short eg. 2" was translated directly to disk, with only progress reports shown on the screen.

```
?- trace_off.
  yes

?- t2d('short  eg.  2').

Now translating file titled:  short eg. 2
The translation will be in the corpora folder with the title:  short eg. 2.English
Please stand by; you may be needed to answer a few questions.

NOW TRANSLATING:
... etc.

Processing has been completed.
The translation is in the corpora folder with the title:  short eg. 2.English
  yes
```

The file "short eg.2.English" which was created by Ordinotrad contains the following data.

You do not have the three letters.

We never will pay you for your work.

Having read your letter, they no longer are happy.

Work on this file, inciuding I/O and user interaction, took 0 minutes and 14 seconds.

The file "short eg. 3" contains the following data.

M. Victor Loewen

3917, chemin Milton

Navan (Ont.)

K4B 1H8

Objet: traduction automatique

The call is made to have the above file translated.

```
?- trace_on.
  yes

?- translate('short eg. 3').

NOW TRANSLATING:
M. Victor Loewen

RESULTS OF PREPROCESSING
w(m,all_caps)
p(.)
w(victor,cap)
w(loewen,cap)

Type any letter and a return to continue.
==> t

RESULTS OF MORPHOLOGICAL ANALYSIS BASED ON DICTIONARY LOOKUP
lu(all_caps,[string-m])
lu(cap,[noun-(victor,m,s,proper,count,[name,person])])
lu(cap,[noun-(loewen,m,s,proper,count,[name,person])])

Type any letter and a return to continue.
==> o

RESULTS OF PARSING
mention
    name
        initials
            lu(all_caps,string-m)
        name
            lu(cap,noun-(victor,m,s,proper,count,[name,person]))
        name
            lu(cap,noun-(loewen,m,s,proper,count,[name,person]))
```

Type any letter and a return to continue.
==> r

RESULTS OF CASE PROCESSING
mention
  name
    initials
      lu(all_caps,string-m)
    name
      lu(cap,noun-(victor,m,s,proper,count,[name,person]))
    name
      lu(cap,noun-(loewen,m,s,proper,count,[name,person]))

Type any letter and a return to continue.
==> L

RESULTS OF LEXICAL TRANSFER

mention
  name
    initials
      lu(all_caps,string-m)
    name
      lu(cap,noun-(victor,s,proper,count))
    name
      lu(cap,noun-(loewen,s,proper,count))

Type any letter and a return to continue.
==> o

TRANSLATION
Mr. Victor Loewen

Translation operations took 0 minutes and 11 seconds.

Do you wish to translate the next sentence from your file?
[yes,y,no,n] ==> y

NOW TRANSLATING:
3917, chemin Milton

RESULTS OF PREPROCESSING
n(3917,cardinal)
p(,)
w(chemin,low)
w(milton,cap)

Type any letter and a return to continue.
==> e

RESULTS OF MORPHOLOGICAL ANALYSIS BASED ON DICTIONARY LOOKUP
n(3917,cardinal)
p(pct,,)
lu(low,[noun-(chemin,m,s,common,count,[pspace,distance])])

```
lu(cap,[noun-(milton,(m,f),s,proper,count,[name])])

Type any letter and a return to continue.
==> w

RESULTS OF PARSING
mention
    address
        number(3917,cardinal)
        lu(low,noun-(chemin,m,s,common,count,[pspace,distance]))
        np
            np
                noun
                    lu(cap,noun-(milton,(m,f),s,proper,count,[name]))
                modifiers
                    nil
                    nil

Type any letter and a return to continue.
==> e

RESULTS OF CASE PROCESSING
mention
    address
        number(3917,cardinal)
        lu(low,noun-(chemin,m,s,common,count,[pspace,distance]))
        np
            np
                noun
                    lu(cap,noun-(milton,(m,f),s,proper,count,[name]))
                modifiers
                    nil
                    nil

Type any letter and a return to continue.
==> n

RESULTS OF LEXICAL TRANSFER

mention
    address
        number(3917,cardinal)
        lu(low,noun-(road,s,common,count))
        np
            np
                noun
                    lu(cap,noun-(milton,s,proper,count))
                modifiers
                    nil
                    nil


Type any letter and a return to continue.
==> c
```

```
TRANSLATION
3917 Milton Road

Translation operations took 0 minutes and 10 seconds.

Do you wish to translate the next sentence from your file?
[yes,y,no,n] ==> y

NOW TRANSLATING:
Navan (Ont.)

RESULTS OF PREPROCESSING
w(navan,cap)
p(()
w(ont,cap)
p(.)
p())

Type any letter and a return to continue.
==> o

RESULTS OF MORPHOLOGICAL ANALYSIS BASED ON DICTIONARY LOOKUP
lu(cap,[noun-(navan,m,s,proper,count,[pspace,location,name])])
p(pct,()
lu(cap,[verb-(avoir,(pres,ind,3,pl),[possession-[agent-(r,[something],nil),affected-
    (r,[something],nil),beneficiary-(o,[event,person],for)]]),abbrev,noun-
    (ontario,nil,s,proper,mass,[location])])
p(pct,))

Type any letter and a return to continue.
==> p

RESULTS OF PARSING
mention
    address
        np
            np
                noun
                    lu(cap,noun-(navan,m,s,proper,count,[pspace,location,name]))
                modifiers
                    nil
                    nil
        np
            np
                noun
                    lu(cap,noun-(ontario,nil,s,proper,mass,[location]))
                modifiers
                    nil
                    nil

Type any letter and a return to continue.
==> y

RESULTS OF CASE PROCESSING
mention
```

```
address
    np
        np
            noun
                lu(cap,noun-(navan,m,s,proper,count,[pspace,location,name]))
            modifiers
                nil
                nil
    np
        np
            noun
                lu(cap,noun-(ontario,nil,s,proper,mass,[location]))
            modifiers
                nil
                nil
```

Type any letter and a return to continue.
==> r

RESULTS OF LEXICAL TRANSFER

```
mention
    address
        np
            np
                noun
                    lu(cap,noun-(navan,s,proper,mass))
                modifiers
                    nil
                    nil
        np
            np
                noun
                    lu(cap,noun-(ontario,s,proper,mass))
                modifiers
                    nil
                    nil
```

Type any letter and a return to continue.
==> l

TRANSLATION
Navan, Ontario

Translation operations took 0 minutes and 13 seconds.

Do you wish to translate the next sentence from your file?
[yes,y,no,n] ==> y

NOW TRANSLATING:
K4B 1H8

RESULTS OF PREPROCESSING
w(k,all_caps)

n(4,cardinal)
w(b,all_caps)
n(1,cardinal)
w(h,all_caps)
n(8,cardinal)

Type any letter and a return to continue.
==> g

RESULTS OF MORPHOLOGICAL ANALYSIS BASED ON DICTIONARY LOOKUP
lu(all_caps,[string-k])
n(4,cardinal)
lu(all_caps,[string-b])
n(1,cardinal)
lu(all_caps,[string-h])
n(8,cardinal)

Type any letter and a return to continue.
==> h

RESULTS OF PARSING
mention
    postal_code
        lu(all_caps,string-k)
        number(4,cardinal)
        lu(all_caps,string-b)
        number(1,cardinal)
        lu(all_caps,string-h)
        number(8,cardinal)

Type any letter and a return to continue.
==> t

RESULTS OF CASE PROCESSING
mention
    postal_code
        lu(all_caps,string-k)
        number(4,cardinal)
        lu(all_caps,string-b)
        number(1,cardinal)
        lu(all_caps,string-h)
        number(8,cardinal)

Type any letter and a return to continue.
==> l

RESULTS OF LEXICAL TRANSFER

mention
    postal_code
        lu(all_caps,string-k)
        number(4,cardinal)
        lu(all_caps,string-b)
        number(1,cardinal)
        lu(all_caps,string-h)

```
        number(8,cardinal)

Type any letter and a return to continue.
==> 9

TRANSLATION
K4B 1H8

Translation operations took 0 minutes and 33 seconds.

Do you wish to translate the next sentence from your file?
[yes,y,no,n] ==> y

NOW TRANSLATING:
Objet: traduction automatique

RESULTS OF PREPROCESSING
w(objet,cap)
p(:)
w(traduction,low)
w(automatique,low)

Type any letter and a return to continue.
==> 8

RESULTS OF MORPHOLOGICAL ANALYSIS BASED ON DICTIONARY LOOKUP
lu(cap,[noun-(objet,m,s,common,count,[pthing])])
p(pct,:)
lu(low,[noun-(traduction,f,s,common,count,[posprocess,mentinteraction])])
lu(low,[adj-(automatique,desc,(m,f),s,[attribute])])

Type any letter and a return to continue.
==> 8

RESULTS OF PARSING
mention
    subject
        lu(cap,noun-(objet,m,s,common,count,[pthing]))
        p
            :
    np
        np
            noun
                lu(low,noun-(traduction,f,s,common,count,[posprocess,mentinteraction]))
            modifiers
                nil
                lu(low,adj-(automatique,desc,f,s,[attribute]))

Type any letter and a return to continue.
==> 0

RESULTS OF CASE PROCESSING
mention
    subject
```

```
        lu(cap,noun-(objet,m,s,common,count,[pthing]))
        p
            :
    np
        np
            noun
                lu(low,noun-(traduction,f,s,common,count,[posprocess,mentinteraction]))
            modifiers
                nil
                lu(low,adj-(automatique,desc,f,s,[attribute]))

Type any letter and a return to continue.
==> r

RESULTS OF LEXICAL TRANSFER

mention
    subject
        lu(cap,noun-(object,s,common,count))
        p
            :
    np
        np
            noun
                lu(low,noun-(translation,s,common,count))
            modifiers
                nil
                lu(low,adj-(automatic,desc))


Type any letter and a return to continue.
==> d

TRANSLATION
Subject : automatic translation

Translation operations took 0 minutes and 41 seconds.

Do you wish to translate the next sentence from your file?
[yes,y,no,n] ==> y
There is no more to be translated in your file.
```

Using the "t2d" predicate, the file "short eg. 3" was translated directly to disk, with only progress reports shown on the screen.

```
?- trace_off.
  yes

?- t2d('short eg. 3').
```

Now translating file titled: short eg. 3
The translation will be in the corpora folder with the title: short eg. 3.English
Please stand by; you may be needed to answer a few questions.

NOW TRANSLATING:
... etc.

Processing has been completed.
The translation is in the corpora folder with the title: short eg. 3.English
yes

The file "short eg.3.English" which was created by Ordinotrad contains the following data.

Mr. Victor Loewen

3917 Milton Road

Navan, Ontario

K4B 1H8

Subject : automatic translation

Work on this file, including I/O and user interaction, took 0 minutes and 15 seconds.

The following are actual text files that were translated by Ordinotrad. The French source text is given on the left and the English target text is on the right.

ardec inc.
meubles de bureau
1616, av. christophe-colomb
montréal A1B 2C3 / 555-1234
V/Référence COM1085
V/Lettre du 77-11-10
N/Référence COM-28-D
RECOMMANDÉ
Le 15 septembre 1987
Lavallée et Fils Inc.
25, rue de l'École
Saint-Jérôme (Qué.)
J3Y 4F6
À l'attention de Monsieur C. Dupré
Objet: Documentation - Bibliothèques
Monsieur,
Nous vous remercions de votre lettre et sommes heureux de vous adresser la documentation demandée.
En parcourant les dépliants ci-joints, vous constaterez que nous fabriquons une gamme complète de bibliothèques et rayonnages.
Il vous sera donc facile de choisir un modèle en fonction de vos besoins.
Tous les modèles illustrés sont livrables dans un délai de huit jours, et nous assurons nous-mêmes le transport et l'installation.
Nous sommes ainsi en mesure de vous garantir une entière satisfaction.
Dans l'espoir de vous servir prochainement, nous vous prions d'agréer, Monsieur, l'expression de nos sentiments dévoués.
Le Directeur commercial
Jean Lavoie
JL/cm
P.J. 3 dépliants
c.c. - Monsieur A. Légaré

Ardec Inc.
Furniture of office
1616 Christophe - Colomb Avenue
Montreal A1B 2C3 / 555-1234
Your reference COM1085
Your letter of 77-11-10
Our reference COM-28-D
REGISTERED
September 15, 1987
Lavallée and Son Inc.
25 School Street
Saint - Jérome, Quebec
J3Y 4F6
Attention of Mr. C. Dupré
Subject : documentation - libraries
Sir :
We thank you for your letter and we are happy to send requested documentation to you.
Going through the enclosed folders, you will note that we manufacture a complete range of libraries and of shelving.
Therefore it will be easy for you to choose a model according to your needs.
All the illustrated models are deliverable in a period of eight days, and we provide transportation and the installation, ourselves.
Therefore we are able to assure an entire satisfaction for you.
In the hope to serve you soon, we invite you to accept the expression of our devoted feelings, Sir.
The commercial director
Jean Lavoie
JL/cm
Encl. 3 folders
C.c.: Mr. A. Légaré

Work on this file, including I/O and user interaction, took 2 minutes and 25 seconds.

Je ne le chante pas.	I do not sing it.
Tu ne chantes point.	You do not sing.
Nous ne chantons guère.	We scarcely sing.
Vous ne chantez jamais.	You never sing.
Tu ne le chantes plus.	You no longer sing it.
Tu ne fais rien.	You make none.
Ils ne voient personne.	They see no one.
Il n'écrit pas de lettres.	He does not write letters.
Je ne l'ai pas chanté.	I did not sing it.
Tu ne me l'as point dit.	You did not say it to me.
Il n'a guère chanté.	He scarcely sang.
Nous n'avons jamais chanté.	We never sang.
Vous n'avez plus chanté.	You no longer sang.
Ils n'ont rien fait.	They made none.
Vous n'avez vu personne.	You saw no one.
Il n'a écrit aucun des lettres.	He wrote none of the letters.
Je lui ai dit de ne pas le faire.	I said to him not to make it.
Je lui ai dit de ne point le faire.	I said to him not to make it.
Je lui ai dit de ne jamais le faire.	I said to him never to make it.
Je lui ai dit de ne rien faire.	I said to him to make none.
Je lui ai dit de ne le faire plus.	I said to him no longer to make it.
Il m'a dit de n'inviter personne.	He said to me to invite no one.
Il n'a que des livres chez lui.	Il n'a que des livres chez lui.
Il n'a des livres que chez lui.	TRANSLATION UNSUCCESSFUL
	Il n'a des livres que chez lui.
	TRANSLATION UNSUCCESSFUL
	Work on this file, including I/O and user interaction, took 1 minute and 41 seconds.

1616, av. christophe-colomb	1616 Christophe - Colomb Avenue
25, rue de l'École	25 School Street
3917, chemin St-Milton	3917 Saint - Milton Road
Saint-Jérôme (Qué.)	Saint - Jérome, Quebec
Saint-Jérôme (Québec)	Saint - Jérome, Quebec
Saint-Jérôme, Québec	Saint - Jérome, Quebec
montréal B1C 2D3	Montreal B1C 2D3
montréal, B1C 2D3	Montreal B1C 2D3
J3Y 4F6	J3Y 4F6
	Work on this file, including I/O and user interaction, took 0 minutes and 37 seconds.

c.c. - Monsieur A. Légaré CC : M. Ardec Copie : Madame Ardec	C.c.: Mr. A. Légaré C.c.: Mr. Ardec C.c.: Mrs. Ardec Work on this file, including I/O and user interaction, took 0 minutes and 13 seconds.
le lundi 15 septembre 1987 lundi 15 septembre 1987 le 15 septembre 1987 15 septembre 1987 le 15 septembre 15 septembre septembre 1987 le septembre septembre le 87-09-15 87-09-15	Monday, September 15, 1987 Monday, September 15, 1987 September 15, 1987 September 15, 1987 September 15 September 15 September 1987 September September 87-09-15 87-09-15 Work on this file, including I/O and user interaction, took 0 minutes and 29 seconds.
V/référence du 88-12-09 V/référence COM-28-D V/Référence COM1085 V/Réf. du 88-12-09 V/Réf. COM-28-D V/Réf. COM1085 V/R. du 88-12-09 V/R. COM-28-D V/R. COM1085 V/Réf. V/R.	Your reference of 88-12-09 Your reference COM-28-D Your reference COM1085 Your reference of 88-12-09 Your reference COM-28-D Your reference COM1085 Your reference of 88-12-09 Your reference COM-28-D Your reference COM1085 Your reference. Your reference. Work on this file, including I/O and user interaction, took 0 minutes and 32 seconds.

Monsieur J. Ardec M. J.-P. Ardec Jean-Paul Ardec Ardec et Fils Ardec et Fils Inc. Ardec, Fils et Cie Ardec Ltée Ardec et Cie	Mr. J. Ardec Mr. J. - P. Ardec Jean - Paul Ardec Ardec and Son Ardec and Son Inc. Ardec, Son And Company Ardec Ltd. Ardec and Co. Work on this file, including I/O and user interaction, took 0 minutes and 32 seconds.

P.J. trois dépliants P.J. 3 dépliants p.j. pièce jointe 3 pièces jointes: 2 lettres, 1 tableau 3 pièces jointes 3 p.j. - lettre, tableau, livre	Encl. three folders Encl. 3 folders Encl.. Encl. Encl. 2 Letters, 1 Table Encl. Encl. Letter, Table, Book Work on this file, including I/O and user interaction, took 0 minutes and 28 seconds.

J'en suis un. Il en a fait trois. Il en a fait trois adresses. Il en a fait beaucoup. Je leur en ai adressé trois. J'ai trois des lettres. Tu as trois lettres. Il a trois. Ils leur avaient adressé les cinq lettres. Ils auront adressé, Monsieur, les lettres à eux. Nous vous remercions de votre lettre. Nous sommes heureux de vous adresser la documentation demandée.	I am one of them. He made three of them. He made three addresses of them. He made many of them. I sent three of them to them. I have three of the letters. You have three letters. He has three. They had sent the five letters to them. They will have sent the letters to them, Sir. We thank you for your letter. We are happy to send requested documentation to you. Work on this file, including I/O and user interaction, took 0 minutes and 46 seconds.

Cher Monsieur,	Dear Sir :
Monsieur,	Sir :
Messieurs,	Sirs :
Chère Madame,	Dear Madam :
Madame,	Madam :
Mesdames,	Madams :
Chers Monsieur et Madame,	Dear Sir and Madam :
Chers Monsieur, Madame et amis,	Dear Sir, Madam and friends :
Chers Monsieur, Madame, et amis,	Dear Sir, Madam, and friends :
Chers amis,	Dear friends :
Amis,	Friends :
Chers Jean et Paul,	Dear Jean and Paul :
Jean et Paul,	Jean and Paul :
Chères Marie et Jeanne,	Dear Marie and Jeanne :
	Work on this file, including I/O and user interaction, took 0 minutes and 52 seconds.

Appendix 4

Program Code

The following program code is replete with comments and examples illustrating or explaining the linguistic or translation processes involved in the rules. These comments and examples are surrounded by slash-asterisk combinations, which means that AAIS Prolog skips over them when reading the file. What follows is, therefore, entirely readable by AAIS Prolog.

The order of the files is as follows.

User Interface

Preprocessor
Morphological Analysis

Parser
Parser - Sentence
Parser - Event
Parser - Noun Phrase
Parser - Preposition
Parser - Sentence components
Parser - Agreements

Case Processor
Lexical and Structural Transfer
English Morphology

Generator
Generator - Verbs
Generator - Nouns
Generator - Sentence Components
Generator - Arrangement
Generator - Mentions

French Closed Dictionary
French Open Dictionary
English Dictionary
Open Transfer Dictionary

Word-Acquisition Module
Taxonomy
Translate-to-disk Utility
Utilities

```
/*************************************************************************
*************************************************************************
USER INTERFACE
© 1987, 1988 Victor Loewen
*************************************************************************
*************************************************************************/

/*
This file contains all the rules controlling the user interface.

After loading this file, i.e., double-clicking on the icon in the Finder, the following occur:
    AAIS Prolog is started up;
    this file is consulted; and
    the user is informed how to begin.

The call
    ?- translate.
will allow the user to have individual sentences translated, which the user must input through
the keyboard.

When the "translate" predicate is called as a goal, the following rule becomes
the "executive manager" for the entire translation procedure. Subsequent sentences
are translated only if the 'finish' goal fails. Failing this call clears the stack
of goals and provides a more efficient use of memory.
*/

/*
The following is used only to provide an indication of time required to load the whole program.
*/

:- Start is cputime, assert(start_time(Start)).

translate :-
    repeat,
    nl(5),
    do_one_sentence,
    finish.

do_one_sentence :-
    get_French_sentence(Sentence_string_F),
    Start is cputime,
        analysis(Sentence_string_F, Deep_structure_representation_F),
        transfer(Deep_structure_representation_F, Deep_structure_representation_E),
        generation(Deep_structure_representation_E,Sentence_string_E),
    Elapsed is cputime - Start,
    nl, write('TRANSLATION'),
    nl, printstring(Sentence_string_E),
    nl, print_time('Translation operations took ', Elapsed),
    !.

/*
The call
    ?- translate(Filename).
will allow the user to have sentences previously saved in a datafile on disk to be translated
in succession.
Note: the Filename must be a symbol. For example:
?- translate(corpus).
?- translate('Lettre de Tomas').
```

?- translate('Connaissances - 1').

NB: the file must be located in a folder called "corpora" on a disk called "DATA".
The program could eventually take advantage of the Macintosh Toolbox and look for the file, wherever it is.

When the "translate" predicate (arity 1) is called as a goal, the following rule becomes the "executive manager" for the entire translation procedure. Subsequent sentences in a file are translated at the user's request. The 'process' call is failed each time to clear the stack of goals and to provide a more efficient use of memory.

The system halts after every sentence and asks the user whether or not to continue.

The result of translate/1 is that a line is read in from a file instead of from the user at the keyboard.
This code may be modified later to make use of the Macintosh interface, so that the user can select a file to be translated from a list box (similar to 'Open file...').
*/

```
translate(Filename) :-
    explode(File, ['DATA:corpora:', Filename]),
    fopen(File, "r", Read_only_stream),
    process_file(Read_only_stream),
    fclose(Read_only_stream).

process_file(Stream) :-
    re_readline(Stream, Sentence_string_F),
    ttynl,ttywrite('NOW TRANSLATING:'),ttynl,
    ttyprintstring(Sentence_string_F),ttynl,
    process(Sentence_string_F),
    ask_question('Do you wish to translate the next sentence from your file?',
        [yes, y, no, n],Ans),
    (Ans = n ; Ans = no), resetio, show_ordinotrad, nl, start_over, quits.

process_file(_) :-
    ttywrite('There is no more to be translated in your file.'),
    resetio, show_ordinotrad, nl, start_over, quits.

process(Sentence_string_F) :-
    Start is cputime,
        analysis(Sentence_string_F, Deep_structure_representation_F), !,
        transfer(Deep_structure_representation_F, Deep_structure_representation_E),
        generation(Deep_structure_representation_E, Sentence_string_E),
    Elapsed is cputime - Start,
    nl, write('TRANSLATION'),
    nl, printstring(Sentence_string_E),
    nl, print_time('Translation operations took ', Elapsed), !.
process(_) :-
    ask_question('Do you wish to translate the next sentence from your file?',
        [yes, y, no, n],Ans),
    !, (Ans = n ; Ans = no), resetio, show_ordinotrad, nl, start_over, quits.
```

/*** ANALYSIS SEGMENT OF ORDINOTRAD ***/

```
analysis(Sentence_string_F, Cased_tree) :-
    interim_trace_on,

    pre_translation_procedures(Sentence_string_F, List_of_lexical_units), !,
    nl,write('RESULTS OF PREPROCESSING'), nl,
    write_column(List_of_lexical_units),
```

```
    ask_question('Type any letter and a return to continue.',_), !,

    morphanalyzer(List_of_lexical_units, Analyzed_sentence), !,
    nl,write('RESULTS OF MORPHOLOGICAL ANALYSIS BASED ON DICTIONARY LOOKUP'), nl,
    write_tree(Analyzed_sentence,0),
    ask_question('Type any letter and a return to continue.',_), !,

    parser(Analyzed_sentence,Syntactic_tree), !,
    nl,write('RESULTS OF PARSING'),nl,
    write_tree(Syntactic_tree, 0),
    ask_question('Type any letter and a return to continue.',_), !,

    case_processor(Syntactic_tree, Cased_tree),
    nl,write('RESULTS OF CASE PROCESSING'),nl,
    write_tree(Cased_tree, 0),
    ask_question('Type any letter and a return to continue.',_), !.
```

/* Same analysis procedures without the tracing. */

```
analysis(Sentence_string_F, Cased_tree)  -
    pre_translation_procedures(Sentence_string_F, List_of_lexical_units), !,
    write('Preprocessing completed.'),nl,
    morphanalyzer(List_of_lexical_units, Analyzed_sentence), !,
    write('Morphological analysis completed.'),nl,
    parser(Analyzed_sentence,Syntactic_tree), !,
    write('Parsing completed.'),nl,
    case_processor(Syntactic_tree, Cased_tree), !,
    write('Case processing completed.'), nl.
```

/*
TRANSFER SEGMENT OF ORDINOTRAD
The transfer rule is defined in the transfer file.
*/

/*
GENERATION SEGMENT OF ORDINOTRAD
The generation rule is defined in the generation file.
*/

/*
The readline predicate below is predefined by AAIS Prolog. It reads in all
characters typed by the user up to a carriage return. It returns the input
surrounded by double quotes. In other words, the input need not conclude with
any punctuation in particular, as would be the case, for example, of various mentions in a letter.
The user is also asked to confirm that the input is correct. If it is not, it
must be re-input entirely.
*/

```
get_French_sentence(Sentence_string) :-
    repeat,
    ttynl,
    ttywrite('Enter French-language sentence to be translated.'),
    ttynl,
    ttywrite('==> '),
    ttyreadline(Sentence_string),
    Sentence_string \= "" , % AAIS seems to hang on to a cr and uses it here.
    ttynl,
    ttywrite('Original French sentence given:'),ttynl,
    ttyprintstring(Sentence_string),ttynl,
    ask_question('Is this correct?', [yes, y, no, n], Answer),
```

```
    (Answer = yes ; Answer = y), !.
```

/*** General-purpose question-askers. ***/

```
ask_question(Question, Symbol) :-
    ttynl, ttywrite(Question), ttynl,
    ttywrite('--> '),
    ttyratom(Symbol), !.

ask_question(Question, Allowed_answers, Symbol) :-
    repeat,
    ttynl,
    ttywrite(Question),ttynl,
    ttywrite(Allowed_answers), ttywrite('  --> '),
    ttyratom(Symbol),
    check_answer(Symbol, Allowed_answers),
    !.

check_answer(Answer,Poss_Answers) :-
    member(Answer,Poss_Answers),!.

check_answer(Answer,Poss_Answers) :-
    ttynl,
    ttywrite('Please respond with one of'),
    ttytab(3),ttywrite(Poss_Answers),ttynl,
    ttywrite('followed by the return key.'), ttynl,
    !,fail.
```

/*** Have all sentences been translated for the user? ***/

```
finish :-
    ask_question('Do you have any more sentences to be translated?',
        [yes, y, no, n],Answer),
    (Answer = n ; Answer = no),
    show_ordinotrad,
    ttynl,
    start_over,
    quits.
```

/*
Tell user how to start over in the event there has been a change of mind after saying there are no more sentences to be translated.
*/

```
start_over :-
    ttynl(2),
    ttywrite('There will eventually be a translate menu.  The following are now available.'),
    ttynl,
    ttywrite('To have a sentence translated with the results viewed on the screen, type
"translate."'),
    ttynl,
    ttywrite('To have a file translated with the results viewed on the screen, type
"translate(Filename)."'),
    ttynl,
    ttywrite('  • Files to be translated must be located in the corpora folder.').
```

/*** Tell user how to return to the Finder, then return user to the AAIS Prolog prompt. ***/

```
quits :-
    ttynl,
    ttywrite('  To quit, select QUIT under the FILE menu or simply type "exit." and a
return.'),
    ttynl.
```

/*** This rule, when called as a goal, displays the name of the program. ***/

```
show_ordinotrad :-
    ttynl(3),
    ttyindentto(55),
    ttywrite('ORDINOTRAD'),
    ttywrite(™), ttynl,
    ttyindentto(55),
    ttywrite('© 1987, 1988  Victor Loewen'), ttynl.
```

/*** Display length of time required for translation process. ***/

```
print_time(Message, Elapsed) :-
    Total is integer(Elapsed),
    secs_to_mm_ss(Total, Min, Sec),
    minutes(Min, Minutes),
    seconds(Sec, Seconds),
    string(Sec, S),
    string(Min, M),
    explode(Info, [Message, M, ' ', Minutes, ' and ', S, ' ', Seconds, '.']),
    nl, printstring(Info),nl.

minutes(1, minute).
minutes(_, minutes).
seconds(1, second).
seconds(_, seconds).
```

/*** Ask the user for status of certain modules. ***/

```
enable_disable :-
    nl,ask_question('For this session, do you wish to have Ordinotrad acquire French and
English words
presently unknown to it?', [yes, y, no, n], Answer),
    (Answer = yes ; Answer = y),
    reconsult('ORDINOTRAD:Ordino Code:Word Acquisition'),
    write('Word-acquisition module is enabled.'),
    fail.

enable_disable :-
    nl, ask_question('For this session, do you wish to have the translate to disk facility
turned on?',
        [yes, y, no, n], Answer),
    (Answer = yes ; Answer = y),
    reconsult('ORDINOTRAD:Ordino Code:Translate2Disk'),
    write('t2d predicate enabled.'), nl,
    write('To have a file translated with the results saved on disk, type "t2d(Filename)."'),
    fail.

enable_disable :-
    nl, ask_question('For this session, do you wish to view interim results at key points in
processing?',
        [yes, y, no, n], Answer),
```

```
    (Answer = yes ; Answer = y),
    assert(interim_trace_on),
    write('Tracing is enabled.').
```

/*** Short-cuts for users. ***/

```
trace_on :- assert(interim_trace_on).
trace_off :- retractall(interim_trace_on).
```

/*
Load all programs required to run Ordinotrad.
Note that the dcgfile predicate is predefined by AAIS Prolog and rewrites grammar
rules (with -->) as Prolog rules (with :-)*/

```
:- show_ordinotrad,
    nl(2),
    write('Loading Ordinotrad, a French-English machine-translation system.  Please stand
by.').
:- consult('ORDINOTRAD:Ordino Code:Utilities').
:- consult('ORDINOTRAD:Ordino Code:Taxonomy').

:- consult('ORDINOTRAD:Ordino Code:Preprocessor').
:- consult('ORDINOTRAD:Ordino Code:Morphology Analyzer').

:- dcgfile('ORDINOTRAD:Ordino Code:Parser').
:- dcgfile('ORDINOTRAD:Ordino Code:Parser - sentence').
:- dcgfile('ORDINOTRAD:Ordino Code:Parser - event').
:- dcgfile('ORDINOTRAD:Ordino Code:Parser - noun phrase').
:- dcgfile('ORDINOTRAD:Ordino Code:Parser - s components').
:- dcgfile('ORDINOTRAD:Ordino Code:Parser - preposition').
:- consult('ORDINOTRAD:Ordino Code:Parser - agreements').

:- consult('ORDINOTRAD:Ordino Code:Case Processor').

:- consult('DATA:Dictionaries:French Closed Dictionary').
:- consult('DATA:Dictionaries:French Open Dictionary').
:- consult('DATA:Dictionaries:Open Transfer Dictionary').
:- consult('DATA:Dictionaries:English Dictionary').

:- consult('ORDINOTRAD:Ordino Code:Transfer').
:- consult('ORDINOTRAD:Ordino Code:English Morphology').

:- consult('ORDINOTRAD:Ordino Code:Generator').
:- consult('ORDINOTRAD:Ordino Code:Generator - Arrangement').
:- consult('ORDINOTRAD:Ordino Code:Generator - Verbs').
:- consult('ORDINOTRAD:Ordino Code:Generator - Nouns').
:- consult('ORDINOTRAD:Ordino Code:Generator - Mentions').
:- consult('ORDINOTRAD:Ordino Code:Generator - Sentence Components').

:- definerom(beep,16'a9c8,no,[longint]).
:- beep(10), beep(19).

:- start_time(Start), Elapsed is cputime - Start,
    nl, print_time('Loading the program took ', Elapsed),
    retractall(start_time(_)).

:- define(interim_trace_on,0).
:- define(learner_on, 0).
:- enable_disable.
:- denyall(enable_disable,_).

:- start_over, quits.
```

```
/*************************************************************************
*************************************************************************
PREPROCESSING
© 1987, 1988 Victor Loewen
*************************************************************************
*************************************************************************/
```

/*
The following rules define what the preprocessor considers to be French letters, digits, symbols, and unacceptable input for the purpose of identifying lexical units.
cf. Grévisse, *Le Bon Usage*, §90-106.

Note: there are many characters available on the Macintosh keyboard and not all of them are commonly used in French or are even common to all the fonts available for the Macintosh. Since the accented letters are represented by high-level ASCII codes (which are not standardized), the laser fonts generally accepted in the Macintosh community are used in this system for the sake of transferability. The laser fonts are consistent among themselves in terms of ASCII codes.
*/

/*
Rules defining letters and the relationship between lower and upper case.
*/

```
letter(a,'A').    letter(à,À).      letter(â,'A').
letter(e,'E').    letter(é,'É').    letter(ê,Ê).      letter(ë,E).      letter(è,È).
letter(i,'I').    letter(î,Î).      letter(ï,Ï).
letter(o,'O').    letter(ô,Ô).
letter(u,'U').    letter(ù,Ù).      letter(û,'U').

letter(b,'B').    letter(c,'C').    letter(ç,'Ç').    letter(d,'D').    letter(f,'F').
letter(g,'G').    letter(h,'H').    letter(j,'J').    letter(k,'K').    letter(l,'L').
letter(m,'M').    letter(n,'N').    letter(p,'P').    letter(q,'Q').    letter(r,'R').
letter(s,'S').    letter(t,'T').    letter(v,'V').    letter(w,'W').    letter(x,'X').
letter(y,'Y').    letter(z,'Z').
```

/*
Rules defining digits.
*/

```
digit('0').   digit('1').   digit('2').   digit('3').   digit('4').
digit('5').   digit('6').   digit('7').   digit('8').   digit('9').
```

/*
Rules defining punctuation marks.
cf. Le Bon Usage, § 106.
Note: some punctuation requires standardization (apostrophes, quotation marks, etc.) to reduce duplication of processing rules.
*/

```
punc(.,.).
punc(?,?).
punc(!,!).
punc(',',',').
punc(';',';').
```

```
punc(':',':').
punc(',',',').
punc('-','-').
punc('(','(').        punc(')',')').
punc('[','[').        punc(']',']').
punc(«,«).        punc(»,»).
punc('\'','\'').
punc('"','"').
punc('{','{').        punc('}','}').
punc(",'"').      punc(",'"').
punc(`,'\'').     punc(',','\'').
```

/*
Rules defining symbols with generally accepted structural meaning.
*/

```
sym('$').
sym(¢).
sym('%').
sym('-').  % hyphen
sym('/').
sym(#).       % theoretically not part of French language
sym(&).
sym('~').
sym('+').
sym('*').
sym('_').  % underlined space, shift + hyphen key
sym('—').  % tiret, option key + shift + hyphen key
sym(<).       sym(>).       sym(≤).       sym(≥).
```

/*
Since preparing the input for machine translation is a pre-translation activity, a rule is defined for the predicate pre_translation_procedures. Overall, it takes the input string from the user and returns a list of lexical units to be passed on for morphological analysis. Each lexical unit has a label indicating what kind of lexical unit it is, as follows:

- w = a word;
- n = a number;
- p = a punctuation mark;
- s = a symbol.

Ordinotrad interprets a word as a string comprised of one or more letters (including letters with diacritical marks).

Ordinotrad interprets numbers as strings comprised of:
- one or more digits,
- a comma between two digits, or
- a decimal between two digits or after a space and before a digit

A number followed by 'er', 'ere', or 'e' is labelled an ordinal and the 'er' etc. is dropped.

Punctuation marks and symbols are defined in the rules above. The exceptions are instances in which a punctuation mark, such as a hyphen or period (decimal), form part of a word or number.
*/

```
pre_translation_procedures(French_sentence, Preprocessed_sentence) :-
```

```
    explode(French_sentence, Char_list),
    get_lexical_list(Char_list, Lexical_units),!,
    expand_elisions(Lexical_units, Result_process_1),
    periods_to_ellipsis(Result_process_1, Result_process_2),
    abridge_ordinals(Result_process_2, Result_process_3),
    expand_contractions(Result_process_3, Preprocessed_sentence).


/******
GET LEXICAL LIST
******/

/*
The get_lexical_list goal finds and returns only words, numbers, punctuation and
symbols.

Blank spaces between lexical units are dropped.
Punctuation marks and symbols form individual lexical units of a single character.

All other characters are considered to be unacceptable in the French to English translation
environment.
*/

get_lexical_list([],[]) :- !.

get_lexical_list([' '| Rest_of_chars], Word_list) :-  % Drop the space.
    get_lexical_list(Rest_of_chars, Word_list).


/*
Any upper or lower case letter may begin a new lexical unit which is labeled as a word.
*/

get_lexical_list([First_char | Rest_of_chars], [w(Word,Format) | Lexical_list]) :-
    letter(First_char,_),
    get_letters(First_char, Rest_of_chars, Word_chars, Remainder,Formats),
    explode(Word, Word_chars),
    get_word_type([l | Formats],Format),
    get_lexical_list(Remainder, Lexical_list).

get_lexical_list([First_char | Rest_of_chars], [w(Word,Format) | Lexical_list]) :-
    letter(Low_case,First_char),
    get_letters(Low_case, Rest_of_chars, Word_chars, Remainder,Formats),
    explode(Word, Word_chars),
    get_word_type([u | Formats],Format),
    get_lexical_list(Remainder, Lexical_list).


/*
Any digit may begin a new lexical unit which is labeled as a number.
*/

get_lexical_list([First_char | Rest_of_chars], [n(Number, cardinal) | Lexical_list]) :-
    digit(First_char),
    get_digits(First_char, Rest_of_chars, Digits, Remainder),
    explode(Number, Digits),
    get_lexical_list(Remainder, Lexical_list).
```

```
/*
All punctuation characters are separate lexical units.
*/

get_lexical_list([Punc_given|Rest_of_chars], [p(Punc_standard)|Lexical_list]) :-
    punc(Punc_given,Punc_standard),
    get_lexical_list(Rest_of_chars, Lexical_list).


/*
All symbols are separate lexical units.
*/

get_lexical_list([Sym|Rest_of_chars], [s(Sym)|Lexical_list]) :-
    sym(Sym),
    get_lexical_list(Rest_of_chars, Lexical_list).

/*
All other characters are unacceptable.
Notify user of character in question, giving choice to quit or continue without character.
*/

get_lexical_list([Rubbish|Rest_of_chars], [Lexical_list]) :-
    nl,
    report('The following input is unacceptable:  ',[Rubbish],' '),
    ask_question('Do you wish to quit or to continue without the unacceptable character?',
        [q,'quit',c,'continue'],Answer),
    (Answer = c ; Answer = 'continue'),
    get_lexical_list(Rest_of_chars, Lexical_list).

get_lexical_list(_,_) :-
    quit.

/*
If the first character was found to be a letter, check the next character. If
it is valid for a word, add it to the list of characters making up the word and
continue checking characters. Hyphens are valid. If the
next character is not valid, end the list of characters forming the word.
*/

get_letters(Char,[Next_char|Rest],[Char|Chars],Remainder,[l | Formats]) :-
    letter(Next_char,_),
    get_letters(Next_char,Rest,Chars,Remainder,Formats).

get_letters(Char,[Next_char|Rest],[Char|Chars],Remainder,[u | Formats]) :-
    letter(Low_case,Next_char),
    get_letters(Low_case,Rest,Chars,Remainder,Formats).

/*
If the above rules fail, the end of the word has been reached, i.e., the next character is
not a letter or hyphen or there is no next character. The last character tested then becomes the
last character of the word.
*/

get_letters(Char, Rest, [Char], Rest,[]) :- !.


/*
```

If the first character has already been found to be valid for a number, add it to the list of characters making up the number and check the next characters.

If the next character is a digit, add it to the list.
If the next two characters involve a space and a digit, add the digit to the list.
If the next two characters involve a period and a digit, add both to the list.
If the next two characters involve a comma and a digit, add both to the list.
*/

```
get_digits(Char,[Next_char|Rest],[Char|Chars],Remainder) :-
    digit(Next_char),
    get_digits(Next_char,Rest,Chars,Remainder).

get_digits(Char,[' ', Next_char | Rest],[Char | Chars],Remainder) :-
    digit(Next_char),
    get_digits(Next_char,Rest,Chars,Remainder).

get_digits(Char,['.', Next_char | Rest],[Char, '.' | Chars],Remainder) :-
    digit(Next_char),
    get_digits(Next_char,Rest,Chars,Remainder).

get_digits(Char,[',', Next_char | Rest],[Char, ',' | Chars],Remainder) :-
    digit(Next_char),
    get_digits(Next_char,Rest,Chars,Remainder).
```

/*
If the above rules fail, the end of the number has been reached, i.e., the next character is not a digit, or the next two characters are not a space-digit, comma-digit, or period-digit combination or there is no next character. The last digit tested then becomes the last digit of the number.
*/

```
get_digits(Char, Rest, [Char], Rest) :- !.
```

/*
Since all dictionary entries are in lower case, part of the get_lexical_list procedure changes all upper case to lower case. Yet information on the type of word is kept for use in later processing.

Word_types are:
    low = all letters are in lower case
    cap = first letter is capitalized, rest are lower case
    all_caps = all letters are capitalized
    mix = mixture of upper and lower that will be used to generate the same string in the English if it is not to be translated (e.g., eMail --> eMail)
*/

```
get_word_type(Formats, low) :-
    not member(u, Formats).

get_word_type(Formats, all_caps) :-
    not member(l, Formats).

get_word_type([u | Rest], cap) :-
    not member(u, Rest).

get_word_type(Formats,Mix) :-
    explode(Mix, Formats).
```

```
/*
The following set of rules is called by the expand_elisions goal of the
pre_translation_procedures. The set of rules is passed a list of lexical
units and returns the same list, except that any instance of an elision is
replaced by the full form of the words. The apostrophes are dropped. Note that
not all instances of an apostrophe (e.g., le 'marketing', il est...)
are a case of elision, but only a special use of apostrophe. In these instances
the apostrophe is kept for later processing.


In French, elision involves dropping one of the final vowels [a, e, i] before
a vowel or a mute h beginning the following word
    Cf. (Le Bon Usage, §133, 134, 135)

This set of rules replaces elision in the French sentence as follows:
    l'      ==>     le_la (avoids having to check the gender for le or la)
    d'      ==>     de
    j'      ==>     je
    m'      ==>     me
    t'      ==>     te
    s'      ==>     si before il or ils, se before all else
    n'      ==>     ne
    c'      ==>     ce
    u'      ==>     ue
*/


expand_elisions([], []) :- !.

/*
Check letter or word to the left of apostrophe and word to the right. The word to
the right must begin with a vowel.
*/

expand_elisions([w(Left,T1),p('\''),w(Right,T2)|Rest],[w(Full_form,T1),w(Right,T2)|List]) :-
    elidable(Left, Right, Full_form),
    expand_elisions(Rest, List).

expand_elisions([H | T], [H | List]) :-
    expand_elisions(T, List).

/*
The si instances of elision are taken care of directly by defining the full form in a short database,
provided that the lexical item to the right is 'il' or 'ils'.
*/

elidable(s,ils,si).
elidable(s,il,si).

/*
To avoid having to figure out at the preprocessing stage whether a word is
masculine or feminine, instances of elision involving le/la are replaced by
the full ambiguous form le_la, provided that the following word begins with a vowel.
*/

elidable(l, Vowel_word, le_la) :-
```

```
    explode(Vowel_word, [Vowel | _]),
    member(Vowel, [a,e,i,o,u,â,à,é,ê,ë,è,î,ï,ô,û,ù,y,h]).

/*
Only the instances of elision listed above are possible. Therefore a special
rule is written to allow only for them. In other words, if the letter preceding
an apostrophe is not a member of the list given in the rule, it is not treated
as a case of elision but as a special use of an apostrophe.
*/

elidable(Letter,Vowel_word,Full_form) :-
    member(Letter, [d,j,m,t,s,n,c]),
    explode(Vowel_word, [Vowel | _]),
    member(Vowel, [a,e,i,o,u,â,à,é,ê,ë,è,î,ï,ô,û,ù,y,h]),
    explode(Full_form, [Letter,e]).

/*
Any elision with a 'u' before the apostrophe involves the dropping of an 'e',
whatever the word. Even 'presqu'île' for purposes of translating.
*/

elidable(End_in_u, Vowel_word, Full_form) :-
    explode(Vowel_word, [Vowel | _]),
    member(Vowel, [a,e,i,o,u,â,à,é,ê,ë,è,î,ï,ô,û,ù,y,h]),
    explode(End_in_u, Char_list),
    lastelem(Char_list, u),
    append(Char_list,[e],New_word),
    explode(Full_form, New_word).


/*
The periods_to_ellipsis goal of the pre_translation_procedures predicate replaces three
successive periods with an ellipsis character (option + ;) which has its own structural meaning.
This rule allows users who are unfamiliar with the Macintosh keyboard to type in three
successive periods for «points de suspension» instead of the ellipsis.
*/


periods_to_ellipsis([], []) :- !.

periods_to_ellipsis([p(.),p(.),p(.)|Rest], [p(…)|List]) :-
    periods_to_ellipsis(Rest, List).

periods_to_ellipsis([H | T], [ H | List]) :-
    periods_to_ellipsis(T, List).


/*
The abridge_ordinals goal of the pre_translation_procedures predicate finds and labels numbers
which are used as ordinals. The alphabetic indication of the ordinal in French is dropped.
*/

abridge_ordinals([],[]) :- !.

abridge_ordinals([n(X, cardinal),w(E,low) | Rest], [n(X,ordinal) | List]) :-
    member(E, [e,er,ère]),
    abridge_ordinals(Rest,List).

abridge_ordinals([H|T],[H|List]) :-
```

```
    abridge_ordinals(T,List).

/*
The expand_contractions goal of the pre_translation_procedures predicate expands contractions
Any instance of a contraction of a definite or indefinite article is replaced by the full form of the
two words involved. This allows for homogeneity in processing; e.g., the similar structure of
«à lequel» and «à laquelle» allows for similar processing techniques. Moreover, the meaning
is made more explicit morphologically.
    Cf. (Le Bon Usage, §601-604)
*/

expand_contractions([], []) :- !.

expand_contractions([w(Contraction,T1)|Rest], [w(Left,T1),w(Right,low)|List]) :-
    contraction(Contraction, Left, Right),
    expand_contractions(Rest, List).

expand_contractions([H | T], [ H | List]) :-
    expand_contractions(T, List).

/*
Database of facts concerning contractions.
*/

contraction(au, à, le).
contraction(aux, à, les).
contraction(du, de, le).
contraction(des, de, les).
contraction(auquel, à, lequel).
contraction(duquel, de, lequel).
contraction(auxquels, à, lesquels).
contraction(desquels, de, lesquels).
contraction(auxquelles, à, lesquelles).
contraction(desquelles, de, lesquelles).
```

```
/*************************************************************************
*************************************************************************
MORPHOLOGICAL ANALYSIS
© 1987, 1988 Victor Loewen
*************************************************************************
*************************************************************************/
```

/*
The primary goal of morphological analysis is to identify all possible information for a given lexical unit based on its morphology, which is checked against databases for endings and against the dictionary.

Lexical units labeled as words by the preprocessor undergo morphological analysis in order to determine their dictionary form, part of speech, grammatical description, type, and intensional meaning(s).

The dictionary form is the form found in the dictionary database.
The part of speech is one of the accepted nine parts of speech (cf. Le Bon Usage, §143).
Grammatical description covers: tense, mood, person, number, gender.
Word formats include: all_caps, cap, low, mix (described in preprocessor comments).
Intensional meanings are defined in terms of Ordinotrad's commensense taxonomy (world view).
Case Frames are found only with verbs.

Symbols, punctuation and numbers are not processed as they are not subject to morphological analysis.

Periods following abbreviations are dropped, as the corresponding full forms are used, unless the period is the final lexical unit in the input. This means, for example, that "A.B.C." would be processed as three separate strings, while "boul." would be processed as a single string.
*/

```
morphanalyzer([], []) :- !.

morphanalyzer([w(Lex_unit,Format), p('.'), X | Rest], [lu(Format, Description) | List]) :-
    analyze(Lex_unit, Description),!,
    (member(abbrev, Description) ; member(string-_, Description)),
    morphanalyzer([X | Rest], List).

morphanalyzer([w(Lex_unit,Format) | Rest], [lu(Format, Description) | List]) :-
    analyze(Lex_unit, Description),!,
    morphanalyzer(Rest, List).

morphanalyzer([ p(P) | Rest ], [ p(pct, P) | List ]) :-
    morphanalyzer(Rest, List).

morphanalyzer([ s(S) | Rest ], [ s(sym, S) | List ]) :-
    morphanalyzer(Rest, List).

morphanalyzer([ n(N, Type) | Rest ], [ n(N, Type) | List ]) :-
    morphanalyzer(Rest, List).
```

/*
The analyze procedure hinges on looking words up in the dictionary database. The dictionary is first checked to see if it contains the lexical unit as given. If the lexical unit is not found, it is

first assumed to be a noun or adjective. The masculine singular form is determined, if possible, and then the dictionary is checked again.
Failing that, the second assumption is that the lexical unit is an adverb whose adjectival root is defined in the dictionary.
Failing that, the third assumption is that the lexical unit is a verb in one of its conjugated forms. The infinitive form is determined and the dictionary is checked again.
Failing that, the assumption is:
a. that the lexical unit is not to be analyzed, or, if it is to be analyzed,
a. that the word is misspelled (and the correct spelling is requested), or, if correct,
b. that ORDINOTRAD does not know the lexical unit in question (the item is passed to the word acquisition module).
Note: the predicate 'learn' asks the user for information about the lexical unit in question. The information gained is kept in memory for use in the present session. It is also stored on disk for future sessions.
*/

/*
Direct Lookup (i.e., the form given as input is found in the dictionary)
*/

```
analyze(Lex_unit, Dict_descript) :-
    f(Lex_unit, Dict_descript), !.
```

/*
Treat the word as a declinable noun or adjective, an adverb, or a conjugated verb.
*/

```
analyze(Lex_unit, Standard_Desc) :-
    fem_to_masc(Lex_unit, Dict_form),
    f(Dict_form, Desc), !,
    standardize(Lex_unit, Desc, f, nil, Standard_Desc),
    Standard_Desc \= [].

analyze(Lex_unit, Standard_Desc) :-
    pl_to_sing(Lex_unit, Dict_form),
    f(Dict_form, Desc), !,
    standardize(Lex_unit, Desc, nil, pl, Standard_Desc),
    Standard_Desc \= [].

analyze(Lex_unit, Standard_Desc) :-
    pl_to_sing(Lex_unit, Word_sing),
    fem_to_masc(Word_sing, Dict_form),
    f(Dict_form, Desc), !,
    standardize(Lex_unit, Desc, f, pl, Standard_Desc),
    Standard_Desc \= [].
```

/*
The adjective form is kept in the place of adverbs, if the adverb can be derived from the adjective, in order to reduce entries in the dictionaries. The assumption here is that if an adverb can be derived from an adjective in French, the equivalent can be generated from the translation of the adjective.
*/

```
analyze(Lex_unit, [adv - (Adj_form, manner)]) :-
    adv_to_adj(Lex_unit, Adj_form).

analyze(Lex_unit, [verb-(Dict_form, TMPN, Meanings_frames)]) :-
    find_vb_root(Lex_unit, Root, TMPN),
    get_infinitive(Root, Dict_form),
```

```
    f(Dict_form, Desc), !,
    member(verb-(_, inf, Meanings_frames), Desc).

/*
The next two rules deal with unrecognizable input.
*/

analyze(Lex_unit, Standard_Desc) :-
    explode(Question, ['Is «', Lex_unit, '» misspelled?']),
    ask_question(Question,[yes,y,no,n],Ans),
    (Ans = yes ; Ans = y),
    ask_question('Please give the correct spelling.', Correct),
    analyze(Correct, Standard_Desc).

/*
The word-acquisition module may be turned off, depending on how the user answered the queries at
startup.
*/

analyze(Lex_unit, Description) :-
    learner_on,
    learn(Lex_unit),
    analyze(Lex_unit, Description), !.

analyze(Lex_unit, _) :-
    ttynl,
    ttywrite('SORRY.  THE WORD-ACQUISITION MODULE HAS NOT BEEN ENABLED AND THE UNKNOWN
WORD «'), ttywrite(Lex_unit), ttywrite('» IN THE INPUT SENTENCE CANNOT BE PROCESSED.'),
    !, resetio, quit.
```

/*
Nouns, adjectives and some pronouns require standardization of their descriptions if new grammatical information has been found through the analysis routines.

In the cases where the lexical unit looked up is syntactically ambiguous (e.g., adresse), the dictionary description contains descriptions of the word for both parts of speech.
e.g.
?- f(adresse,X).
X = [verb - (adresser, (pres, (ind, subj), (1, 3), s), Meanings_and_frames),
noun - (adresse, f, s, common, count, [pspace,name])]
?- f(adresses, X).
no

Should the input be "adresses", for example, the description for the noun would have to be changed (dynamically only, not in the dictionary) to show that the word is plural. And, since the "analyze" predicate above succeeds with the "fem_to_masc" analysis, "analyze" would not be recalled to test "adresses" as a verb. In this case, the input is a verb, but the dictionary description given does not reflect that it is 2nd person singular. For this reason, there is a call for verb analysis in the following standardization procedure.
e.g.
?- analyze(adresses, X).
X = [verb - (adresser, (pres, (ind, subj), 2, s), Meanings_frames),
noun - (adresse, f, pl, common, count, [pspace,name])]

In another example, where the input is "écrites", the "fem_to_masc" analysis would have succeeded. However, since "écrit" is a masculine noun, the nominal description would have to be dropped. Yet the verbal description would need to be changed to a feminine plural

participle.
e.g.
f(écrit,X).
X = [noun - (écrit, m, s, common, count, [artifact]).
verb -(écrire, (pres, ind, 3, s), MFs),
verb -(écrire, (past - part, m, s), MFs)]
?- f(écrites,X).
no
?- analyze(écrites,X).
X = [verb - (écrire, (past - part, f, pl), MFs)]

The variables "Gen(der)_ch(ange)" and "Num(ber)_ch(ange)" carry the information acquired through the analysis routines and to be applied to the dictionary descriptions.
e.g., an adjective is in the dictionary as a masculine, singular, but the description needs to be changed to feminine plural, based on the results of the preceding analysis.
*/

```
standardize(_, [], _, _, []).

standardize(Lex_unit, [adj-(LU, Type, Gen1, Num1, Meaning) | Rest], Gen_ch, Num_ch,
        [adj-(LU, Type, Gen2, Num2, Meaning) | List]) :-
    gender(Gen_ch, Gen1, Ger2),
    num(Num_ch, Num1, Num2),
    standardize(Lex_unit, Rest, Gen_ch, Num_ch, List).
```

/*
Noun genders cannot change.
*/

```
standardize(Lex_unit, [noun-(LU, Gender, Num1, Info) | Rest], Gen_ch, Num_ch,
        [noun - (LU, Gender, Num2, Info) | List]) :-
    (Gen_ch = nil ; Gen_ch = Gender),
    num(Num_ch, Num1, Num2),
    standardize(Lex_unit, Rest, Gen_ch, Num_ch, List).

standardize(Lex_unit, [verb-V_in | Rest], Gen_ch, Num_ch, [verb-V_out | List]) :-
    change_v(Lex_unit, V_in, Gen_ch, Num_ch, V_out),
    standardize(Lex_unit, Rest, Gen_ch, Num_ch, List).

standardize(Lex_unit, [_ | Rest], Gender, Number, List) :-
    standardize(Lex_unit, Rest, Gender, Number, List).

gender(nil, G, G).
gender(f, _, f).
num(nil, N, N).
num(pl, _, pl).
```

/*
If only a gender change has been discovered (e.g., dite), a verb is retained only if its description is that of a participle.
If both gender and number changes have been discovered, (e.g., écrites), a verb description is retained only if it is that of a participle.
If only a number change has been discovered, (e.g., adresses), input is analyzed for the correct tense/mood/person/number.
*/

```
change_v(Lex_unit, (LU,(P-part, Gen1, Num1), MFs), Gen_ch, Num_ch,
        (LU, (P-part, Gen2, Num2), MFs)) :-
    num(Num_ch, Num1, Num2),
```

```
    gender(Gen_ch, Gen1, Gen2), !.

change_v(Lex_unit, (LU, _, MFs), _, _, (LU, TMPN, MFs)) :-
    find_vb_root(Lex_unit, Root, TMPN),
    get_Infinitive(Root, LU).
```

/*
The following rule is passed a word which is checked for traditional feminine endings, making use of the following database. If the word is "feminine", the corresponding masculine form is determined and returned. If the word is already morphologically masculine, the rule fails.

The rule checks the last letters of a given word with the defined endings, beginning with as many as 6 of the last letters of the word. The database is therefore used explicitly, i.e., it provides a check for defining feminine endings.
*/

```
fem_to_masc(Fem, Masc) :-
    explode(Fem, Fem_List),
    mem(X, [6,5,4,3,2,1]),
    get_last_X_chars(Fem_List, Last_X, X),
    fem_to_masc_endings(Last_X,Masc_ending),
    app(Root_list, Last_X, Fem_List),
    app(Root_list, Masc_ending, Masc_List),
    explode(Masc, Masc_List).
```

/*
For each fact in this short database, the first argument refers to a feminine ending, and the second argument to the corresponding masculine ending.
Cf. Le Bon Usage, §399ff.
*/

```
fem_to_masc_endings([e], []).
fem_to_masc_endings([é,e], [e]).
fem_to_masc_endings([e,l,l,e], [el]).
fem_to_masc_endings([e,l,l,e], [eau]).
fem_to_masc_endings([n,n,e], [n]).
fem_to_masc_endings([t,t,e], [t]).
fem_to_masc_endings([è,t,e], [et]).
fem_to_masc_endings([è,r,e], [er]).
fem_to_masc_endings([v,e], [f]).
fem_to_masc_endings([e,u,s,e], [eur]).
fem_to_masc_endings([e,u,s,e], [eux]).
fem_to_masc_endings([e,u,r,e], [eur]).
fem_to_masc_endings([e,r,e,s,s,e], [eur]).
fem_to_masc_endings([t,r,i,c,e], [teur]).
fem_to_masc_endings([e,s,s,e], [e]).
fem_to_masc_endings([i,s,s,e], [is]).
fem_to_masc_endings([o,u,s,s,e], [oux]).
fem_to_masc_endings([o,u,s,e], [oux]).
fem_to_masc_endings([i,e,u,s,e], [ieux]).
fem_to_masc_endings([o,l,l,e], [ou]).
fem_to_masc_endings([c,q,u,e], [c]).
fem_to_masc_endings([q,u,e], [c]).
fem_to_masc_endings([g,u,e], [g]).
fem_to_masc_endings([n,n,e,s,s,e],[n]).
fem_to_masc_endings([e,u,r,e,s,s,e],[eur]).
% See Grevisse §413 for a few irregular, uncommon forms.
```

```
/*
The following rule takes a given French word, checks it for usual plural endings,
using the following database and returns the singular form. If the word is already
singular, the rule fails.
*/

pl_to_sing(Word_pl, Word_S) :-
    explode(Word_pl, Word_pl_List),
    mem(X, [1,4,3]),
    get_last_X_chars(Word_pl_List, Last_X, X),
    pl_to_sing_endings(Last_X,Sing_ending),
    app(Root_List, Last_X, Word_pl_List),
    app(Root_List,Sing_ending,Word_S_List),
    explode(Word_S, Word_S_List).


/*
The following database provides definitions for relationships between common endings.
For each fact in this short database, the first argument refers to a plural
ending, and the second argument to the corresponding singular ending.
Cf. Le Bon Usage, §509ff.
*/

pl_to_sing_endings([s], []).
pl_to_sing_endings([e,a,u,x], [eau]).
pl_to_sing_endings([e,u,x], [eu]).
pl_to_sing_endings([a,u,x], [al]).
pl_to_sing_endings([a,u,x], [au]).
pl_to_sing_endings([a,u,x], [ail]).
pl_to_sing_endings([o,u,x], [ou]).

/*
The following rule takes a given French word, checks it for usual adverbial endings,
using the following database and returns the appropriate adjectival form.
For example:

?- analyze(heureusement,X).
   X = [adv - (heureux, manner)]
?- analyze(complètement,X).
   X = [adv - (complet, manner)]
?- analyze(longuement,X).
   X = [adv - (long, manner)]
?- analyze(entièrement,X).
   X = [adv - (entier, manner)]
?- analyze(commercialement,X).
   X = [adv - (commercial, manner)]
?- analyze(patiemmement,X).
   X = [adv - (patient, manner)]
?- analyze(facilement,X).
   X = [adv - (facile, manner)]
*/

adv_to_adj(Lex_unit, Adj_form) :-
    explode(Lex_unit, Adv_list),
    mem(X, [8,5,4]),
    get_last_X_chars(Adv_list, Adv_ending, X),
    adv_to_adj_endings(Adv_ending, Adj_ending),
    app(Root_List, Adv_ending, Adv_list),
```

```
    app(Root_List, Adj_ending, Adj_list),
    explode(Interim_form, Adj_list),
    lookup(Interim_form, Adj_form).

lookup(Adj_form, Adj_form) :-
    f(Adj_form, Desc), !,
    member(adj-_, Desc).

lookup(Interim, Adj_form) :-
    fem_to_masc(Interim, Adj_form),
    f(Adj_form, Desc), !,
    member(adj-_, Desc).


adv_to_adj_endings([e,m,m,e,m,e,n,t], [ente]).        % patiemment
adv_to_adj_endings([a,m,m,e,m,e,n,t], [ante]).        % puissamment
adv_to_adj_endings([m,e,n,t], []).      % regular, grandement, bellement
adv_to_adj_endings([û,m,e,n,t], [ue]).      % assidûment
adv_to_adj_endings([é,m,e,n,t], [ée]).      % aisément
adv_to_adj_endings([é,m,e,n,t], [ie]).      % impunément
adv_to_adj_endings([é,m,e,n,t], [e]).       % énormément
adv_to_adj_endings([i,m,e,n,t], [ille]).        % gentiment
```

/*
A procedure is required to find the root of a word assumed to be a verb.
The procedure checks the last 8 letters of a word with the following database of verb endings, then 7 letters, 6 letters, etc., until a root can be determined.
*/

```
find_vb_root(Inflected_word, Root, TMPN) :-
    explode(Inflected_word, List),
    mem(X, [8,7,6,5,4,3,2,1]),
    get_last_X_chars(List, Last_X, X),
    explode(Ending, Last_X),
    vb_ending(Ending, TMPN),
    app(Root_list, Last_X, List),
    explode(Root, Root_list).
```

/*
Rules are needed to define all possible endings for verbs. The relations between these endings and possible tense/mood/person/number are also defined, even though some of the endings are polysyntactic.

The elements of the list comprising the second argument are (for personal modes):
    Tense, Mood, Person, Number
Otherwise, they are:
    Present or Past Participle, Gender Number

ARE 2ND PERSON SINGULAR IMPERATIVES OF ER (and some ir) VERBS CORRECTLY PARSED?
e.g. Cherches-en les raisons. Cherche à en découvrir les raisons. (cf. Grévisse 1465)
*/

```
vb_ending(e,(pres,(ind,subj),(1,3),s)).
vb_ending(es,(pres,(ind,subj),2,s)).
vb_ending(ons,(pres,ind,1,pl)).
vb_ending(ez,(pres,ind,2,pl)).
vb_ending(ent,(pres,(ind,subj),3,pl)).

vb_ending(iens,(pres,ind,(1,2),s)).
vb_ending(ient,(pres,ind,3,s)).
vb_ending(enons,(pres,ind,1,pl)).
vb_ending(enez,(pres,ind,2,pl)).
vb_ending(iennent,(pres,(ind,subj),3,pl)).

vb_ending(is,((pres,hist),ind,(1,2),s)).
vb_ending(it,((pres,hist),ind,3,s)).
```

```
vb_ending(issons,(pres,ind,1,pl)).
vb_ending(issez,(pres,ind,2,pl)).
vb_ending(issent,(pres,(ind,subj),3,pl)).

vb_ending(s,(pres,ind,(1,2),s)).
vb_ending(t,(pres,ind,3,s)).

vb_ending(ois,(pres,ind,(1,2),s)).
vb_ending(oit,(pres,ind,(1,2),s)).

vb_ending(enais,(imp,ind,(1,2),s)).
vb_ending(enait,(imp,ind,3,s)).
vb_ending(enions,(imp,ind,1,pl)).
vb_ending(eniez,(imp,ind,2,pl)).
vb_ending(enaient,(imp,ind,3,pl)).

vb_ending(ais,(imp,ind,(1,2),s)).
vb_ending(ait,(imp,ind,3,s)).
vb_ending(ions,(imp,ind,1,pl)).
vb_ending(iez,(imp,ind,2,pl)).
vb_ending(aient,(imp,ind,3,pl)).

vb_ending(issais,(imp,ind,(1,2),s)).
vb_ending(issait,(imp,ind,3,s)).
vb_ending(issions,(imp,ind,1,pl)).
vb_ending(issiez,(imp,ind,2,pl)).
vb_ending(issaient,(imp,ind,3,pl)).

vb_ending(ai,(hist,ind,1,s)).
vb_ending(as,(hist,ind,2,s)).
vb_ending(a,(hist,ind,3,s)).
vb_ending(âmes,(hist,ind,1,pl)).
vb_ending(âtes,(hist,ind,2,pl)).
vb_ending(èrent,(hist,ind,2,pl)).

vb_ending(îmes,(hist,ind,1,pl)).
vb_ending(îtes,(hist,ind,2,pl)).
vb_ending(irent,(hist,ind,3,pl)).

vb_ending(ins,(hist,ind,(1,2),s)).
vb_ending(int,(hist,ind,3,s)).
vb_ending(înmes,(hist,ind,1,pl)).
vb_ending(întes,(hist,ind,2,pl)).
vb_ending(inrent,(hist,ind,3,pl)).

vb_ending(us,(hist,ind,(1,2),s)).
vb_ending(ut,(hist,ind,3,s)).
vb_ending(ûmes,(hist,ind,1,pl)).
vb_ending(ûtes,(hist,ind,2,pl)).
vb_ending(urent,(hist,ind,3,pl)).

vb_ending(erai,(fut,ind,1,s)).
vb_ending(eras,(fut,ind,2,s)).
vb_ending(era,(fut,ind,3,s)).
vb_ending(erons,(fut,ind,1,pl)).
vb_ending(erez,(fut,ind,2,pl)).
vb_ending(eront,(fut,ind,3,pl)).

vb_ending(irai,(fut,ind,1,s)).
vb_ending(iras,(fut,ind,2,s)).
vb_ending(ira,(fut,ind,3,s)).
vb_ending(irons,(fut,ind,1,pl)).
vb_ending(irez,(fut,ind,2,pl)).
vb_ending(iront,(fut,ind,3,pl)).

vb_ending(iendrai,(fut,ind,1,s)).
vb_ending(iendras,(fut,ind,2,s)).
vb_ending(iendra,(fut,ind,3,s)).
vb_ending(iendrons,(fut,ind,1,pl)).
vb_ending(iendrez,(fut,ind,2,pl)).
vb_ending(iendront,(fut,ind,3,pl)).

vb_ending(rai,(fut,ind,1,s)).
vb_ending(ras,(fut,ind,2,s)).
vb_ending(ra,(fut,ind,3,s)).
vb_ending(rons,(fut,ind,1,pl)).
vb_ending(rez,(fut,ind,2,pl)).
vb_ending(ront,(fut,ind,3,pl)).

vb_ending(isse,((pres,imp),subj,(1,3),s)).
vb_ending(isses,((pres,imp),subj,2,s)).
vb_ending(issions,((pres,imp),subj,1,pl)).
vb_ending(issiez,((pres,imp),subj,2,pl)).
vb_ending(issent,((pres,imp),subj,3,pl)).

vb_ending(oive,((pres,imp),subj,(1,3),s)).
vb_ending(oives,((pres,imp),subj,2,s)).
vb_ending(oivent,((pres,imp),subj,3,pl)).

vb_ending(asse,(imp,subj,1,s)).
vb_ending(asses,(imp,subj,2,s)).
vb_ending(ât,(imp,subj,3,s)).
vb_ending(assions,(imp,subj,1,pl)).
vb_ending(assiez,(imp,subj,2,pl)).
vb_ending(assent,(imp,subj,3,pl)).

vb_ending(1t,(imp,subj,3,s)).

vb_ending(usse,(imp,subj,1,s)).
vb_ending(usses,(imp,subj,2,s)).
vb_ending(ût,(imp,subj,3,s)).
vb_ending(ussions,(imp,subj,1,pl)).
vb_ending(ussiez,(imp,subj,2,pl)).
vb_ending(ussent,(imp,subj,3,pl)).

vb_ending(erais,(pres,cond,(1,2),s)).
vb_ending(erait,(pres,cond,3,s)).
vb_ending(erions,(pres,cond,1,pl)).
vb_ending(eriez,(pres,cond,2,pl)).
vb_ending(eraient,(pres,cond,3,pl)).

vb_ending(irais,(pres,cond,(1,2),s)).
vb_ending(irait,(pres,cond,3,s)).
vb_ending(irions,(pres,cond,1,pl)).
vb_ending(iriez,(pres,cond,2,pl)).
vb_ending(iraient,(pres,cond,3,pl)).

vb_ending(iendrais,(pres,cond,(1,2),s)).
vb_ending(iendrait,(pres,cond,3,s)).
vb_ending(iendrions,(pres,cond,1,pl)).
vb_ending(iendriez,(pres,cond,2,pl)).
vb_ending(iendraient,(pres,cond,3,pl)).

vb_ending(rais,(pres,cond,(1,2),s)).
vb_ending(rait,(pres,cond,3,s)).
vb_ending(rions,(pres,cond,1,pl)).
```

```
vb_ending(riez,(pres,cond,2,pl)).
vb_ending(raient,(pres,cond,3,pl)).

vb_ending(é,(past-part,m,s)).
vb_ending(és,(past-part,m,pl)).
vb_ending(ée,(past-part,f,s)).
vb_ending(ées,(past-part,f,pl)).
vb_ending(i,(past-part,m,s)).
vb_ending(is,(past-part,m,pl)).
vb_ending(ie,(past-part,f,s)).
vb_ending(ies,(past-part,f,pl)).
vb_ending(u,(past-part,m,s)).
vb_ending(us,(past-part,m,pl)).
vb_ending(ue,(past-part,f,s)).
vb_ending(ues,(past-part,f,pl)).
vb_ending(int,(past-part,m,s)).
vb_ending(ints,(past-part,m,pl)).
vb_ending(inte,(past-part,f,s)).
vb_ending(intes,(past-part,f,pl)).

vb_ending(ant,(pres-part,m,s)).
vb_ending(ants,(pres-part,m,pl)).
vb_ending(ante,(pres-part,f,s)).
vb_ending(antes,(pres-part,f,pl)).
vb_ending(issant,(pres-part,m,s)).
vb_ending(issants,(pres-part,m,pl)).
vb_ending(issante,(pres-part,f,s)).
vb_ending(issantes,(pres-part,f,pl)).
```

/*
Since verb roots are not provided in the dictionary, two rules are needed to add an infinitive ending to a verb root.
Note that the second rule handles roots whose forms differ from that of the infinitive root.
*/

```
get_infinitive(Root, Infinitive) :-
    inf_ending(Ending),
    explode(Infinitive, [Root, Ending]).

get_infinitive(Root, Infinitive) :-
    explode(Root, Root_list),
    root_changes(Old_ending, New_ending),
    get_variant_root(Root_list, Old_ending, New_ending, Variant_list),
    explode(Variant, Variant_list),
    inf_ending(Ending),
    explode(Infinitive, [Variant, Ending]).


inf_ending(er).
inf_ending(ir).
inf_ending(re).
inf_ending(ire).
inf_ending(dre).
inf_ending(oir).
inf_ending(evoir).
inf_ending(enir).
inf_ending(aire).
inf_ending(altre).
inf_ending(oire).
inf_ending(indre).
inf_ending(endre).

get_variant_root(Given, Old, New, Variation) :-
    app(List, [Old], Given),
    (New = nil, Add = [] ; Add = [New]),
    app(List, Add, Variation).

get_variant_root(Given, Old, New, Variation) :-
    app(List, [Old, Conson], Given),
    (New = nil, Add = [Conson] ; Add = [New, Conson]),
    app(List, Add, Variation).

get_variant_root(Given, Old, New, Variation) :-
    app(List, [Old, C1, C2], Given),
```

```
    (New = nil, Add = [C1, C2] ; Add = [New, C1, C2]),
    app(List, Add, Variation).


root_changes(ç, c).
root_changes(e, nil).
root_changes(é, e).
root_changes(è, e).
root_changes(è, é).
root_changes(t, nil).
root_changes(l, nil).
root_changes(g, nil).
```

```
/*************************************************************************
*************************************************************************
PARSER
© 1987, 1988 Victor Loewen
*************************************************************************
*************************************************************************/

/*
This call inserted here simply to facilitate reconsulting this file during its development.
SAVE FILE FIRST, SINCE dcgfile READS FROM DISK AND NOT FROM MEMORY.

:- dcgfile('ORDINOTRAD:Ordino Code:Parser').
*/

/*
In line with the discussion of the limitations of Ordinotrad in Chapter 1, grammar rules
need to be written for the following sentence structures:
• Input in the form of an interrogative or an imperative, or containing «mise en relief»
is not processed at this stage.
Since no verb ending is associated with the imperative mood in the list of verb endings
(in the morphological analysis file), the imperative mood will have to be deduced when
processing the event. This may be done when the parser is augmented to handle imperatives.
• mise en relief. e.g. Ce sont ces phrases qui se disent. C'est cette phrase qui m'embête.
• sentences beginning with «il y a»
• sentences like «Il fait trois ans que j'habite...»
*/

parser(Input, _) :-
    member(p(pct,?), Input),
    nl,
    write('Sorry.  This package has not yet been developed to handle interrogatives.'),
    !, fail, nl.

parser(Input, _) :-
    member(p(pct,!), Input),
    nl,
    write('Sorry.  This package has not yet been developed to handle imperatives
or exclamatory remarks.'),
    !, fail, nl.

parser([lu(cap,[verb-(_,(_,ind,P,N),_)]) | _], _) :-
    (P = 1, N = pl ; P = 2),
    nl,
    write('Sorry.  This package has not yet been developed to handle imperatives.'),
    !, fail, nl.

parser([lu(_, [adj-(ce, _) |_]), lu(_, [verb-(être, _)]) | Tail], _) :-
    (member(lu(_, [conj -(que, _)|_]), Tail) ; member(lu(_, [conj -(qui, _)|_]), Tail)),
    nl,
    write('Sorry.  This package has not yet been developed to handle «mise en relief».'),
    !, fail, nl.

parser([lu(_,[pro-(il, _)]), lu(_,[adv-(y, _) |_ ]), lu(_, [verb-(avoir, _) |_])|_],_) :-
    nl,
    write('Sorry.  This package has not yet been developed to handle sentences beginning with
«il y a».'),
    !, fail, nl.

parser([lu(_,[pro-(il,_)]),lu(_,[verb-(faire,_) |_ ]) |Tail], _) :-
```

```
    (member(lu(_, [con] -(que, _)|_]), Tail) ; member(lu(_, [con] -(qui, _)|_]), Tail)),
    nl,
    write('Sorry.  This package has not yet been developed to handle «mise en relief».'),
    !, fail, nl.

/*
The following rule analyzes and determines the well-formedness of declarative sentences,
i.e., that end with a period.  The "Output" contains the sentence's modality and propositional
content.
*/

parser(Input, proposition-Output) :-
    prop(Output, Input, [p(pct,.)]).

/*
The following rule parses input which does not contain a subject and finite verb (called mentions).
*/

parser(Input, mention-Output) :-
    mention(Output, Input, []).

parser(_, failed) :-
    nl,
    write('Sorry.  The parser requires further development to handle the type of sentence
given as input.'),
    nl, fail.


/*
The grammar rules used by the parser are adapted mainly from:
Salkoff, M.  Une grammaire en chaîne du français.  Monographes de linguistique mathématique.
Paris:  Dunod, 1973.
Salkoff, M.  Analyse syntaxique du Français :  Grammaire en chaîne.  Amsterdam:
    John Benjamins, B.V., 1979.

The macro_articulation predicate is defined in the "Parser-s components" file.
*/


prop(A) --> independent_prop(B), more_props(B, A).
prop(A) --> subordinate_prop(B), more_props(B, A).
prop(A) --> ellipsed_prop(B), more_props(B, A).

more_props(First, (First, Articulation, Prop)) -->
    macro_articulation(Articulation),
    prop(Prop).
more_props(Prop, Prop) --> {true}.


/*
WHERE
    independent proposition -->
        noun_phrase,
            (one of {le, la, les, leur, lui, y, en}),
                verb,
                    (various_arrangements_of_noun_phrases | relative_or_subordinate_clauses)


Ellipsed propositions are propositions where the surface subject element is
```

not lexicalized but is understood in reference to what has preceded or is expected to follow.
e.g. Nous sommes heureux et vous addressons le paiement.

A NOTE ABOUT CONJUNCTIONS AND SERIES
Winograd describes the phenomenon of conjunction as: "In place of any unit at any place in the syntax, we can substitute a conjoined structure whose terminal elements are instances of that unit. There are a number of possible conjoined structures such as (both) X and Y; (either) X or Y; X but not Y; and X, Y, Z, ... and (or) W. These in turn can be applied recursively to get combinations like X and either Y or Z." (1983, 525).

As can be seen in the various rules in this parser, conjunctions present a formidable programming challenge. Even Winograd says, "Even these simple examples [of conjunctions] raise problems, and as we explore conjunction further, the data are extremely complex. There has been no satisfactory comprehensive treatment in any grammar..." (1983, 525).
*/

/*
The following rule acts as a general terminal when some special criteria are to be met.
It is used extensively in the parser.
*/

```
terminal(Criteria, lu(Format, Criteria)) --> [lu(Format, Description)],
    {member(Criteria, Description)}.
```

/*
A rule to change a few grammar rules without redoing whole files. Select rules and copy to clipboard before calling newrules. DO NOT USE IF YOU DON'T KNOW WHAT YOU ARE DOING! (or are unaware of this anti-Macintosh-interface use of a mode).
Nuthin'll get wrecked, but I haven't figured out a way to have the re_read predicate fail.
Existing grammar rules with the same predicate/arity will be replaced.
The 'PROCESS' predicate is defined in the dcg.p file provided by AAIS Prolog. It is not included in this thesis.
*/

```
newrules :-
    resetio,
    retractall('ASSERTTIME'(_,_)),
    nl,write('Interrupt with a command-period to escape.\nPaste rules here.\n'),
    re_read(Rule),
    'PROCESS'(Rule),
    fail.
```

```
/*****************************************************************************
*****************************************************************************
PARSER - SENTENCE
© 1987, 1988 Victor Loewen
*****************************************************************************
*****************************************************************************/

/*
This call inserted here simply to facilitate reconsulting this file during its development.
SAVE FILE FIRST, SINCE dcgfile READS FROM DISK AND NOT FROM MEMORY.

:- dcgfile('ORDINOTRAD:Ordino Code:Parser - sentence').
*/

/*
Independent propositions are assumed to begin with a noun phrase.
Special rules would be required for:
    • propositions beginning with "personne" or "rien"
        e.g. "personne n'est venue" vs "une personne est venue"
    • the special cases listed in the "parser" file as limitations of this system
*/

independent_prop(independent_prop-Prop) -->
    np(NP),
    post_np(NP, Prop).

/*
The prep_phrase predicate below is defined in the "Parser - preposition" file.
The event_prop predicate below is defined in the "Parser - event" file.
*/

subordinate_prop(subordinate_prop-(prep_phrase-Prop)) --> prep_phrase(Prop).
subordinate_prop(subordinate_prop-(event_prop-Prop)) --> event_prop(Prop).


ellipsed_prop(ellipsed_prop-Prop) -->
    post_np(_, Prop).

/*
The relative pronoun "que" is dropped as it only confuses the case processor.
The link in English can be generated from the fact of a relative prop itself, and not
necessarily from "que".
If the "que" needs to be kept, simply replace "Prop" with [Modalities, Event | Siblings]
and [Modalities, Event, np-Q | Siblings] where appropriate in the first rule.
*/

relative_prop(A) -->
    rel_pro(Q),
    independent_prop(_-[M, E | Siblings]),
    {append(Siblings, [np-(np-(Q))], Sibs)},
    more_relative_props(relative_prop-[M, E | Sibs], A).
relative_prop(A) -->
    terminal(pro-(qui,_), Qui),
    post_np(Qui, Prop),
    more_relative_props(relative_prop-Prop, A).

more_relative_props(First, (First, Articulation, Props)) -->
    micro_articulation(Articulation),
```

```
    relative_prop(Props).
more_relative_props(First, First) --> {true}.
```

/*
Check that the present setup for relative props handles structures like
    il aime ce qu'il désire
    il aime ce à quoi il fait allusion
    il aime celle qui a fait...
    etc
Cf. Cafe 3 book, p. 166, #76
*/

```
post_np(NP, Prop) -->
    clip_neg(NE),
    pre_verbal_particles(PVP),
    event_right(NE, NP, PVP, Prop).
post_np(NP, Prop) -->
    clip_neg(NE),
    event_right(NE, NP, nil, Prop).

clip_neg(LU) -->
    terminal(adv-(LU, negation), _).
clip_neg(affirmative) --> {true}.

event_right(NE, NP, nil, [Modalities, Event | List]) -->
    event(NE, Verb, Modalities),
    {sv_agreement(Verb, Event, NP, Np)},
    post_event(np-Np, nil, List).

event_right(NE, NP, PVP, [Modalities, Event | List]) -->
    {PVP \= nil},
    event(NE, Verb, Modalities),
    {sv_agreement(Verb, Event, NP, Np)},
    post_event(np-Np, PVP, List).
```

/*
The first argument of post_event/3 may be a noun phrase or a preposition (see the definition of event_prop in the "Parser - event" file). The second argument is PVP information which is also forwarded as it is to be inserted in the proper order in the list.
*/

```
post_event(Prep_or_np, PVP, Post_event) -->
    right_dislocation(RD),
    post_right_dislocation(RD, List),
    {add_to_list(Prep_or_np, PVP, List, Post_event)}.
post_event(Prep_or_np, PVP, Post_event) -->
    post_right_dislocation(List),
    {add_to_list(Prep_or_np, PVP, List, Post_event)}.
post_event(Prep_or_np, PVP, Post_event) -->
    {add_to_list(Prep_or_np, PVP, [], Post_event)}.

post_right_dislocation(RD, [RD | List]) -->
    slot5(Adj),
    post_slot5(adj-Adj, List).
post_right_dislocation(RD, [RD | List]) -->
    adverbial(Adv),
    post_adverbial(adv-Adv, List).
post_right_dislocation(RD, [RD | List]) -->
    post_adverbial(List).
```

```
post_right_dislocation(List) -->
    slot5(Adj),
    post_slot5(adj-Adj, List).
post_right_dislocation(List) -->
    adverbial(Adv),
    post_adverbial(adv-Adv, List).
post_right_dislocation(List) -->
    post_adverbial(List).
```

/*
Cardinal adjectives can occur alone after the verb.
See "Parser - noun phrase" file for definition of slot3 (cardinal adjectives).
e.g. J'en suis un.
Il en a fait trois.
Il en a fait trois exemplaires.
Il en a fait beacoup.
Je leur en ai donné trois.
If there is no "en" preceding a verb which is followed by a cardinal adjective,
e.g., "j'ai trois des lettres," or even "j'ai trois,"
then the cardinal adjective should actually be a
part of the pre_noun for an understood noun, i.e., "j'ai trois lettres des lettres en question".
This is provided for in the definition of noun phrases. All of the above cardinal adjectives
are considered as preceding an understood noun.

Descriptive adjectives can occur alone immediately after stative verbs.
e.g.
Je suis heureux.
Je suis très heureux d'avoir reçu votre lettre de 15 juin 1988.
See "Parser - noun phrase" file for definition of slot5 (descriptive adjectives).
They are processed at this point.
*/

```
post_adverbial(Adv, [Adv | List]) -->
    slot5(Adj),
    post_slot5(adj-Adj, List).
post_adverbial(Adv, [Adv | List]) -->
    post_slot5(List).

post_adverbial(List) -->
    slot5(Adj),
    post_slot5(adj-Adj, List).
post_adverbial(List) -->
    post_slot5(List).

post_slot5(Adj, [Adj | List]) -->
    complements(List).
post_slot5(Adj, [Adj]) --> {true}.

post_slot5(List) -->
    complements(List).
post_slot5([]) --> {true}.


add_to_list(nil, nil, List, List) :- !.
add_to_list(nil, [PVP], List, [np-PVP | List]) :- !.
add_to_list(nil, [PVP1, PVP2], List, [np-PVP1, np-PVP2 | List]) :- !.
add_to_list(NP, nil, List, [NP | List]) :- !.
add_to_list(NP, [PVP], List, [NP, np-PVP | List]) :- !.
```

```
add_to_list(NP, [PVP1, PVP2], List, [NP, np-PVP1, np-PVP2 | List]) :- !.


complements(A) --> np(B), more_complements(np-B, A).
complements(A) --> event_prop(B), more_complements(event_prop-B, A).
complements(A) --> prep_phrase(B), more_complements(prep_phrase-B, A).
complements(A) --> relative_prop(B), more_complements(relative_prop-B, A).

more_complements(First, [First | Compls]) -->
    complements(Compls).
more_complements(First, [First]) --> {true}.
```

/*
Series of complements are handled with the more_complements rules. However, it is assumed that no conjunctions, disjunctions or punctuation will come between them, as such would only be used between complements which are alike. For example, in "nous sommes heureux de vous adresser la lettre" the first complement "heureux" is immediately followed by the event proposition "de vous adresser...". In cases where like complements are articulated, the definition for that complement allows for a series joined by a conjunction. See, for example, the definition of prep_phrase and how it would be applied to "nous vous remercions de votre lettre et de votre intérêt dans...". This would be processed as a prepositional phrase comprised of two prepositional phrases.
*/

/*
Pre_verbal_particles (particules pré-verbales) occur singly or in a strictly determined order.

When there are two pre-verbal particles, not only is the order important, but the choice of what comes first and second is limited to four different pairs of sets. Only one particle in each set is permissible with one of the other set.
*/

```
pre_verbal_particles(A) --> me_te_se_nous_vous(B), allowable_set_1(B, A).
pre_verbal_particles(A) --> le_la_les(B), allowable_set_2(B, A).
pre_verbal_particles(A) --> lui_leur(B), allowable_set_3(B, A).
pre_verbal_particles([A]) --> y_en(A).

allowable_set_1(First, [First, Second]) -->
    le_la_les(Second).
allowable_set_1(First, [First, Second]) -->
    y_en_lui_leur(Second).
allowable_set_1(First, [First]) --> {true}.

allowable_set_2(First, [First, Second]) -->
    y_en_lui_leur(Second).
allowable_set_2(First, [First]) --> {true}.

allowable_set_3(First, [First, Second]) -->
    y_en(Second).
allowable_set_3(First, [First]) --> {true}.


me_te_se_nous_vous(np-A) -->
    terminal(pro-(LU, refl,_,_,_,_), A),
    {member(LU,[me,te,se,nous,vous])}.

le_la_les(np-A) -->
    terminal(pro-(LU,_,_,_,_,_), A),
    {member(LU,[le,la,les,le_la])}.
```

```
y_en_lui_leur(X) --> y_en(X).
y_en_lui_leur(X) --> lui_leur(X).

lui_leur(np-A) --> terminal(pro-(lui, refl,_,_,_,_), A).
lui_leur(np-A) --> terminal(pro-(leur, refl,_,_,_,_), A).

y_en(np-A) --> terminal(pro-(y, _,_,_,_,_), A).
y_en(np-A) --> terminal(pro-(en, _,_,_,_,_), A).
```

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

PARSER - EVENT

© 1987, 1988 Victor Loewen

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

/*
This call inserted here simply to facilitate reconsulting this file during its development.
SAVE FILE FIRST, SINCE dcgfile READS FROM DISK AND NOT FROM MEMORY.

:- dcgfile('ORDINOTRAD:Ordino Code:Parser - event').
*/

/*
EVENTS
The "event" returned by the call to event/6 is always the principal verb, as it provides
the meaning and frame for the purposes of translation. Modal auxiliaries and semi-modal
auxiliaries (avoir, or être) are not returned. They are analyzed only for purposes of
determining Tense, Mood, Person, and Number.

The modalities returned with the principal verb are determined from both the auxiliaries
and the principal verb. They are: tense (x10), mood (indicative, subjunctive, conditional),
voice (active, passive), attitude (affirmative, negative, doubtful), and form (declarative,
interrogative, imperative).

The Form modality is presently defined as declarative, since interrogatives and imperatives
are not accepted by the parser.
*/

/* Composite and simple verbs */

/*
n'a pas été fait
ne pas avoir été fait
a été fait
avoir été fait
*/

event(ne, event-lu(F, verb-(V, (T, M, P, N), M_F)),
       modalities-(T, M, passive, negative-Neg, declarative)) -->
    terminal(verb-(avoir,(Tense, M, P, N),_), _),
    terminal(adv-(Neg, negation), _),
    terminal(verb-(être, (past-part, _), _), _),
    terminal(verb-(_, (past-part, _), _), lu(F, verb-(V, _, M_F))),
    {tense(Tense, T)}.
event(ne, event-lu(F, verb-(V, (past, participle, nil, nil), M_F)),
       modalities-(past, participle, passive, negative-Neg, declarative)) -->
    terminal(adv-(Neg, negation), _),
    terminal(verb-(avoir,inf,_), _),
    terminal(verb-(être, (past-part, _), _), _),
    terminal(verb-(_, (past-part, _), _), lu(F, verb-(V, _, M_F))),
    {tense(Tense, T)}.

event(affirmative, event-lu(F, verb-(V, (T, M, P, N), M_F)),
       modalities-(T, M, passive, affirmative, declarative)) -->
    terminal(verb-(avoir,(Tense, M, P, N),_), _),
    terminal(verb-(être, (past-part, _), _), _),
```

```
    terminal(verb-(_, (past-part, _), _), lu(F, verb-(V, _, M_F))),
    {tense(Tense, T)}.
event(affirmative, event-lu(F, verb-(V, (past, participle, nil, nil), M_F)),
        modalities-(past, participle, passive, affirmative, declarative)) -->
    terminal(verb-(avoir,inf,_), _),
    terminal(verb-(être, (past-part, _), _), _),
    terminal(verb-(_, (past-part, _), _), lu(F, verb-(V, _, M_F))),
    {tense(Tense, T)}.

/*
n'a pas fait / n'a vu personne
ne pas avoir fait
a fait
avoir fait
ayant fait
*/

event(ne, event-lu(F, verb-(V, (T, M, P, N), M_F)),
        modalities-(T, M, active, negative-Neg, declarative)) -->
    terminal(verb-(avoir,(Tense, M, P, N),_), _),
    terminal(adv-(Neg, negation), _),
    terminal(verb-(_, (past-part, _), _), lu(F, verb-(V, _, M_F))),
    {tense(Tense, T)}.
event(ne, event-lu(F, verb-(V, (T, M, P, N), M_F)),
        modalities-(T, M, active, negative-Neg, declarative)) -->
    terminal(verb-(avoir,(Tense, M, P, N),_), _),
    terminal(verb-(_, (past-part, _), _), lu(F, verb-(V, _, M_F))),
    {tense(Tense, T)},
    terminal(adv-(Neg, negation), _),
    {member(Neg, [personne, rien, aucun])}.

event(ne, event-lu(F, verb-(V, (past, participle, nil, nil), M_F)),
        modalities-(past, participle, active, negative-Neg, declarative)) -->
    terminal(adv-(Neg, negation), _),
    terminal(verb-(avoir,inf,_), _),
    terminal(verb-(_, (past-part, _), _), lu(F, verb-(V, _, M_F))),
    {tense(Tense, T)}.

event(affirmative, event-lu(F, verb-(V, (T, M, P, N), M_F)),
        modalities-(T, M, active, affirmative, declarative)) -->
    terminal(verb-(avoir,(Tense, M, P, N),_), _),
    terminal(verb-(_, (past-part, _), _), lu(F, verb-(V, _, M_F))),
    {tense(Tense, T)}.
event(affirmative, event-lu(F, verb-(V, (past, participle, nil, nil), M_F)),
        modalities-(past, participle, active, affirmative, declarative)) -->
    terminal(verb-(avoir,inf,_), _),
    terminal(verb-(_, (past-part, _), _), lu(F, verb-(V, _, M_F))),
    {tense(Tense, T)}.
event(affirmative, event-lu(F, verb-(V, (past, participle, use_ing, nil), M_F)),
        modalities-(past, participle, active, affirmative, declarative)) -->
    terminal(verb-(avoir,(pres-part,_),_), _),
    terminal(verb-(_, (past-part, _), _), lu(F, verb-(V, _, M_F))),
    {tense(Tense, T)}.

/*
n'est pas monté
ne pas être monté
est monté
être monté
étant monté
*/
```

```
event(ne, event-lu(F, verb-(V, (T, M, P, N), M_F)),
       modalities-(T, M, passive, negative-Neg, declarative)) -->
    terminal(verb-(être,(Tense, M, P, N),_), _),
    terminal(adv-(Neg, negation), _),
    terminal(verb-(_, (past-part, _), _), lu(F, verb-(V, _, M_F))),
    {tense(Tense, T)}.
event(ne, event-lu(F, verb-(V, (past, participle, nil, nil), M_F)),
       modalities-(past, participle, passive, negative-Neg, declarative)) -->
    terminal(adv-(Neg, negation), _),
    terminal(verb-(être,inf,_), _),
    terminal(verb-(_, (past-part, _), _), lu(F, verb-(V, _, M_F))),
    {tense(Tense, T)}.

event(affirmative, event-lu(F, verb-(V, (T, M, P, N), M_F)),
       modalities-(T, M, passive, affirmative, declarative)) -->
    terminal(verb-(être,(Tense, M, P, N),_), _),
    terminal(verb-(_, (past-part, _), _), lu(F, verb-(V, _, M_F))),
    {tense(Tense, T)}.
event(affirmative, event-lu(F, verb-(V, (past, participle, nil, nil), M_F)),
       modalities-(past, participle, passive, affirmative, declarative)) -->
    terminal(verb-(être,inf,_), _),
    terminal(verb-(_, (past-part, _), _), lu(F, verb-(V, _, M_F))),
    {tense(Tense, T)}.
event(affirmative, event-lu(F, verb-(V, (past, participle, use_ing, nil), M_F)),
       modalities-(past, participle, passive, affirmative, declarative)) -->
    terminal(verb-(être,(pres-part,_),_), _),
    terminal(verb-(_, (past-part, _), _), lu(F, verb-(V, _, M_F))),
    {tense(Tense, T)}.
```

/* fait */

```
event(ne, event-A, modalities-(T, M, active, negative-Neg, declarative)) -->
    terminal(verb-(_,(T, M, P, N),_), A),
    terminal(adv-(Neg, negation), _).
event(affirmative, event-A, modalities-(T, M, active, affirmative, declarative)) -->
    terminal(verb-(_,(T, M, P, N),_), A).
```

/* faire */

```
event(ne, event-lu(F, verb-(V, (pres, inf, nil, nil), M_F)),
       modalities-(pres, inf, active, negative-Neg, declarative)) -->
    terminal(adv-(Neg, negation), _),
    terminal(verb-(_,inf,_), lu(F, verb-(V, _, M_F))).
event(ne, event-lu(F, verb-(V, (pres, inf, nil, nil), M_F)),
       modalities-(pres, inf, active, negative-Neg, declarative)) -->
    terminal(verb-(_,inf,_), lu(F, verb-(V, _, M_F))),
    terminal(adv-(Neg, negation), _).
event(affirmative, event-lu(F, verb-(V, (pres, inf, nil, nil), M_F)),
       modalities-(pres, inf, active, affirmative, declarative)) -->
    terminal(verb-(_,inf,_), lu(F, verb-(V, _, M_F))).
event(NE, event-lu(F, verb-(V, (pres, inf, nil, nil), M_F)),
       modalities-(pres, inf, active, negative-NE, declarative)) -->
    {member(NE, [pas, rien, jamais, plus, personne, point, guère, aucun])},
    terminal(verb-(_,inf,_), lu(F, verb-(V, _, M_F))).
```

/* faisant */

```
event(ne, event-lu(F, verb-(V, (pres, participle, nil, nil), M_F)),
       modalities-(pres, participle, active, negative-Neg, declarative)) -->
    terminal(verb-(_,(pres - part,_,_),_), lu(F, verb-(V, _, M_F))),
    terminal(adv-(Neg, negation), _).
```

```
event(affirmative, event-lu(F, verb-(V, (pres, participle, nil, nil), M_F)),
       modalities-(pres, participle, active, affirmative, declarative)) -->
   terminal(verb-(_,(pres - part,_,_),_), lu(F, verb-(V, _, M_F))).
event(NE, event-lu(F, verb-(V, (pres, participle, nil, nil), M_F)),
       modalities-(pres, participle, active, negative-NE, declarative)) -->
   {member(NE, [pas, rien, jamais, plus, personne, point, guère, aucun])},
   terminal(verb-(_,(pres - part,_,_),_), lu(F, verb-(V, _, M_F))).


/*
Note that no provision has been made for sentences containing more than one principal
verb (e.g., «il a écrit et remis les lettres demandées»). To do so would require
determining just how verbs can be linked (i.e., «, et» or «et»; if the auxiliary is repeated;
etc.), and which verb would then be considered the principle verb for the sake of picking
a case frame.
*/

/*
The tense of the auxiliary determines the tense of a compound verb.
If the auxiliary is present, the compound is past, etc.
*/

tense(pres, past).
tense(fut, future_anterior).
tense(hist, past_anterior).
tense(imp, pluperfect).
tense(cond, past_future_anterior).

/*
EVENT PROPOSITIONS
Event propositions may or may not begin with a preposition and must contain an infinitive
or present participle. These types of propositions are called for by the subordinate_prop
definition and by others (cf. the qualifying event proposition that may modify a noun or the
complements that may follow a verb).

Examples of event propositions:

(J'ai voulu) pouvoir commencer à faire X
     In the above example, "pouvoir commencer à faire X" is an event prop whose relation
     to the event "vouloir" is the affected role. "à faire X" too is another event prop embedded
     in an event prop, and it too, in this case, would fill the affected slot.
(J'ai voulu) pouvoir faire X
(J'ai voulu) faire X
(j'essaie) de vouloir commencer à faire X
(il souhaite) faire évoluer ...
(il compte) laisser se détériorer ...
(il s'est vu) demander...
après avoir été commandé de faire continuer ...
après m'être fait attacher à...

Event propositions also follow causative auxiliaries.

faire + infinitive (direct intervention)
voir + infinitive (simple presence)
laisser + infinitive (no intervention)
venir + infinitive
compter + infinitive
```

In all cases, the principal verb in an event proposition must be either a participle or an infinitive. Event propositions are never expected to be independent propositions, as these always require the principle verb to be in an inflexioned or aux+participle form.
*/

```
event_prop(A) -->
    prep(Prep),
    clip_neg(NE),  % defined in "parser - sentence" file
    post_prep(NE, Prep, Prop),
    more_event_props(event_prop-Prop, A).
event_prop(A) -->
    clip_neg(NE),
    post_prep(NE, nil, Prop),
    more_event_props(event_prop-Prop, A).

more_event_props(First, (First, Articulation, Props)) -->
    micro_articulation(Articulation),
    event_prop(Props).
more_event_props(Prop, Prop) --> {true}.

qualifying_event_prop(QEP) -->
    terminal(prep-de, A),
    clip_neg(NE),
    post_prep(NE, prep-A, Prop),
    more_q_e_props(qualifying_event_prop-Prop, QEP).

more_q_e_props(First, (First, Articulation, Props)) -->
    micro_articulation(Articulation),
    qualifying_event_prop(Prop),
    more_q_e_props(Prop, Props).
more_q_e_props(Prop, Prop) --> {true}.
```

/*
If there is a negative, the second part of it may be found before the infinitive,
e.g., ne rien faire
or it may be found after the infinitive
e.g., ne faire rien.
In the first case, the second part is picked up here and carried into the event analysis to be inserted into the list of modalities.
In the second case, the second part is picked up with the event analysis, and the "ne" is carried forward to control the choice of event rules to apply.
*/

```
post_prep(ne, Prep, Prop) -->
    clip_neg(affirmative),
    pre_verbal_particles(PVP),
    post_pre_verbal_particles(ne, Prep, PVP, Prop).
post_prep(ne, Prep, Prop) -->
    clip_neg(PAS),
    pre_verbal_particles(PVP),
    post_pre_verbal_particles(PAS, Prep, PVP, Prop).
post_prep(ne, Prep, Prop) -->
    clip_neg(affirmative),
    post_pre_verbal_particles(ne, Prep, nil, Prop).
post_prep(ne, Prep, Prop) -->
    clip_neg(PAS),
    post_pre_verbal_particles(PAS, Prep, nil, Prop).

post_prep(affirmative, Prep, Prop) -->
    pre_verbal_particles(PVP),
    post_pre_verbal_particles(affirmative, Prep, PVP, Prop).
```

```
post_prep(affirmative, Prep, Prop) -->
    post_pre_verbal_particles(affirmative, Prep, nil, Prop).


post_pre_verbal_particles(NEG, Prep, PVP,
        [Modalities, event-lu(F,verb -(LU,(T, participle, G, N),Frame)) | List]) -->
    event(NEG, event-lu(F,verb -(LU,(T, participle, G, N),Frame)), Modalities),
    post_event(Prep, PVP, List).

post_pre_verbal_particles(NE, Prep, PVP,
        [Modalities, event-lu(F,verb -(LU,(pres, inf, G, N),Frame)) | List]) -->
    event(NE, event-lu(F,verb -(LU,(pres, inf, G, N),Frame)), Modalities),
    post_event(Prep, PVP, List).
```

/* post_event is defined in the "Parser - sentence" file. */

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

PARSER - NOUN PHRASE

© 1987, 1988 Victor Loewen

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

/*
This call inserted here simply to facilitate reconsulting this file during its development.
SAVE FILE FIRST, SINCE dcgfile READS FROM DISK AND NOT FROM MEMORY.

```
:- dcgfile('ORDINOTRAD:Ordino Code:Parser - noun phrase').
```

*/

/*
Noun phrases (containing what is known as nominals, or nominal groups).
The subject-verb agreement rules still need to be designed to handle names, dates, and coordinate noun phrases.
Rules need to be defined for nps comprised of celui-ci, etc.

The terminal predicate is defined in the "Parser" file.

According to the following rules, an np is a noun phrase comprised of one or more noun phrases or of one or more pronouns. A mixture of nouns types is not permitted to prevent misleading searches, as for example, in "nous vous remercions de votre lettre et nous sommes heureux" where "votre lettre et nous" are not to be seen as a noun phrase.
*/

```
np(A) --> one_np(A).
np(A) --> pronoun(A).
np(np-(noun-nil,modifiers-(Adj, Post))) -->
    slot3(Adj),
    post_noun(_, Post).
np(name-A) --> name(A).

pronoun(A) -->
    terminal(pro-(_,pers, _), B), more_pronouns(np-B, A).
pronoun(A) -->
    terminal(pro-(LU, rel, _), B),
    {member(LU, [ce, qui, lequel, laquelle, lesquels, lesquelles])},
    more_pronouns(np-B, A).
pronoun(A) -->
    terminal(pro-(_, dem, _), B), more_pronouns(np-B, A).


more_pronouns(First, (First, Articulation, Pros)) -->
    micro_articulation(Articulation),
    pronoun(Pros).
more_pronouns(Pro, Pro) --> {true}.
```

/*
An np (one_np) is comprised of adjectives that precede a main noun and adjectives or qualifying prepositional or relative phrases.
*/

```
one_np(A) -->
    pre_noun(Pre),
    noun(Pre, PRE, Noun),
    post_noun(Noun, POST),
```

```
    {join(PRE, POST, Modifiers)},
    more_one_nps(np-(noun-Noun, modifiers-Modifiers), A).

more_one_nps(First, (First, Articulation, NP)) -->
    micro_articulation(Articulation),
    one_np(NP).
more_one_nps(NP, NP) --> {true}.


noun(Pre, PRE, N) -->
    terminal(noun-_, Noun),
    {adj_n_agreement(Pre, PRE, Noun, N)}.

noun(PRE, PRE, X) -->
    composite_noun(X).
% Due to the complexity of checking agreements between adjectives and composite nouns,
% agreement is assumed to be correct.

/*** e.g. timbre-poste ***/
composite_noun((A,p-(-),C)) -->
    terminal(noun-_,A),
    [p(pct,-)],
    (terminal(noun-_,C) ; terminal(adj-(_,desc,_,_,_),C)).

/*** e.g. saint-pierre ***/
composite_noun((A,p-(-),C)) -->
    terminal(adj-(_,desc,_,_,_),A),
    [p(pct,-)],
    terminal(noun-_,C).

/*** e.g. presse-papiers ***/
composite_noun((A,p-(-),C)) -->
    terminal(verb-(_,(pres,_,Pers,s),_),A),
    [p(pct,-)],
    terminal(noun-_,C),
    {Pers = 3 ; Pers = (_,3)}.

/*** e.g. en-tête ***/
composite_noun((A,p-(-),C)) -->
    terminal(adv-_, A),
    [p(pct,-)],
    terminal(noun-_,C).

/*** e.g. après-midi ***/
composite_noun((A,p-(-),C)) -->
    terminal(prep-_,A),
    [p(pct,-)],
    terminal(noun-_,C).


/*
Pre-noun (words to left of the noun)
The following sequences for pre-noun and post-noun are adapted from:
    Warnant, Léon. Structure syntaxique du Français. Paris: Société d'Édition «Les Belles
    Lettres», 1982.
    MO PC 2361.W28

The following sequence must be followed. For slots 1 to 5, there need not be any adjective at all.
For the 5th slot alone, there may be more than one adjective. The reason is that slots 1 to 4 hold
limiting adjectives, of which only one is feasible, and slot 5 holds descriptive adjectives of which
```

there may be any number.

At the present, both 'la aucune lettre' and 'les quelques lettres' are both acceptable. A means should be determined to make sure the meaning components do not clash.

slot1 TOUT
slot2 ARTICLE | POSSESSIVE ADJ | DEMONSTRATIVE ADJ
slot3 CARDINAL ADJ or ADJ OF quantity, nothing, approximation, distribution
slot4 ADJ indicating difference | ORDINAL ADJ
slot5 QUALIFYING ADJ | PAST PARTICIPLE | PRESENT
NOUN
*/

```
pre_noun(A) --> slots1to5(A).
pre_noun(A) --> slots2to5(A).
pre_noun(A) --> slots3to5(A).
pre_noun(A) --> slots4to5(A).
pre_noun(A) --> slot5(A).
pre_noun(nil) --> {true}.

slots1to5(A) -->
    slot1(B),
    slots2to5(B, A).
slots2to5(A) -->
    slot2(B),
    slots3to5(B, A).
slots3to5(A) -->
    slot3(B),
    slots4to5(B, A).
slots4to5(A) -->
    slot4(B),
    slot5(B, A).

slots2to5(First, (First, Adjs)) -->
    slot2(Second),
    slots3to5(Second, Adjs).
slots2to5(First, Adjs) -->
    slots3to5(First, Adjs).

slots3to5(First, (First, Adjs)) -->
    slot3(Second),
    slots4to5(Second, Adjs).
slots3to5(First, Adjs) -->
    slots4to5(First, Adjs).

slots4to5(First, (First, Adjs)) -->
    slot4(Second),
    slot5(Second, Adjs).
slots4to5(First, Adjs) -->
    slot5(First, Adjs).

slot5(First, (First, Adjs)) -->
    slot5(Adjs).
slot5(Adj, Adj) --> {true}.

slot1(A) --> terminal(adj - (tout, indef, _, _, _), A).
slot1(A) --> terminal(adj - (tous, indef, _, _, _), A).

slot2(A) --> terminal(art - _, A).
slot2(A) --> terminal(adj - (_, poss, _, _, _), A).
slot2(A) --> terminal(adj - (_, dem, _, _, _), A).
```

```
slot3(A) -->  terminal(adj - (_, cardinal, _, _, _), A).
slot3(num(N,cardinal)) --> [n(N, cardinal)].

slot4(A) -->  terminal(adj - (_, ordinal, _, _, _), A).
slot4(A) -->  terminal(adj - (_, indef, _, _, _), A).
slot4(num-(N, ordinal)) --> [n(N, ordinal)].
```

/*
Provision is made here for adjectives derived from the past participles of verbs.
This avoids duplicating entries in the dictionary for both past participles and infinitives
where the past-participle is regular.
*/

```
slot5(A) --> terminal(adj - (_, desc, _, _, _), B), more_desc_adjs(B, A).
slot5(A) -->   [lu(F,Desc)],
    {member(verb-(LU, (past - part, Gender, Number), Meaning_frame), Desc),
    sift_meanings_from_frame(Meaning_frame, Meanings)},
    more_desc_adjs(lu(F,adj-(LU,verbal-desc,Gender,Number,Meanings)), A).
slot5(A) --> composite_adj(B), more_desc_adjs(B, A).


sift_meanings_from_frame([], []).
sift_meanings_from_frame([Meaning-_ | Tail], [Meaning | List]) :-
    sift_meanings_from_frame(Tail, List).
```

/*
Examples of composite adjectives:
    vraiment belles
    ci-jointes
    très heureux
*/

```
composite_adj(A) -->
    terminal(adv-_, B),
    last_half_of_composite(B, A).

last_half_of_composite(Adv, lu-(Adv, p-(-), Adj)) -->
    [p(pct,-)],
    terminal(adj-(_,desc,_,_,_), Adj).
last_half_of_composite(Adv, lu-(Adv, Adj)) -->
    terminal(adj-(_,desc,_,_,_), Adj).
last_half_of_composite(Adv,
        lu-(Adv, lu(F,adj-(LU,verbal-desc,Gender,Number,Meanings)))) -->
    [lu(F,Desc)],
    {member(verb-(LU, (past - part, Gender, Number), Meaning_frame), Desc),
    sift_meanings_from_frame(Meaning_frame, Meanings)}.
last_half_of_composite(Adv, lu-(Adv, p-(-),
        lu(F,adj-(LU,verbal-desc,Gender,Number,Meanings)))) -->
    [p(pct,-)],
    [lu(F,Desc)],
    {member(verb-(LU, (past - part, Gender, Number), Meaning_frame), Desc),
    sift_meanings_from_frame(Meaning_frame, Meanings)}.


more_desc_adjs(First, (First, Articulation, Adj)) -->
    micro_articulation(Articulation),
    slot5(Adj).
more_desc_adjs(First, (First, Adj)) -->
    slot5(Adj).
```

```
more_desc_adjs(Adj, Adj) --> {true}.
```

/*
POST NOUN SEQUENCES
(noun) + adverb + descriptive adjective + event propositions or prepositional phrases
Note that a check is made for descriptive adjectives before the adverbial, in the event
that the adjective is composed of an adverb-adjective combination and to avoid misleading
analyses. E.g., les lettres ci-jointes
*/

```
post_noun(N, A) --> slot5_adv_5_6(B), {adj_n_agreement(B, A, N, _)}.
post_noun(N, A) --> adv_5_6(B), {adj_n_agreement(B, A, N, _)}.
post_noun(N, A) --> slots5to6(B), {adj_n_agreement(B, A, N, _)}.
post_noun(N, A) --> slot6(A).
post_noun(N, nil) --> {true}.

slot5_adv_5_6(A) -->
    slot5(B),
    adv_5_6(B,A).
adv_5_6(A) -->
    adverbial(B),      % defined in the "Parser - sentence components" file
    slots5to6(B, A).
slots5to6(A) -->
    slot5(B),
    slot6(B, A).

adv_5_6(First, Adj_list) -->
    adverbial(Second),
    slots5to6(Second, Adjs),
    {join(First, Adjs, Adj_list)}.
adv_5_6(First, Adjs) -->
    slots5to6(First, Adjs).

slots5to6(First, Adj_list) -->
    slot5(Second),
    slot6(Second, Adjs),
    {join(First, Adjs, Adj_list)}.
slots5to6(First, Adjs) -->
    slot6(First, Adjs).

slot6(First, Adj_list) -->
    slot6(Adjs),
    {join(First, Adjs, Adj_list)}.
slot6(Adj, Adj) --> {true}.
```

/*
Qualifying event propositions are defined in the "Parser - event" file.
Qualifying prepositional phrases are defined in the "Parser - prep" file.
*/

```
slot6(qualifying_prep_phrase-A) --> qualifying_prep_phrase(A).
slot6(relative_prop-A) --> relative_prop(A).
slot6(qualifying_event_prop-A) --> qualifying_event_prop(A).
```

```
/*****************************************************************************
*****************************************************************************
PARSER - PREPOSITION
© 1987, 1988 Victor Loewen
*****************************************************************************
*****************************************************************************/

/*
This call inserted here simply to facilitate reconsulting this file during its development.
SAVE FILE FIRST, SINCE dcgfile READS FROM DISK AND NOT FROM MEMORY.

:- dcgfile('ORDINOTRAD:Ordino Code:Parser - preposition').
*/


prep(locution_prépositive-(A, B, C)) -->
    terminal(prep-en, A),
    terminal(noun-(N,_), B),
    {member(N,[fonction,dehors,dépit,face,faveur,outre,plus,raison,mesure])},
    terminal(prep-de, C).

prep(prep-A) --> terminal(prep-_, A).

prep_phrase(A) -->
    prep(P), one_np(NP),
    more_prep_phrases(prep_phrase-(P, np-NP), A).
prep_phrase(A) -->
    prep(P),
    terminal(pro - (X, refl, _), B),
    more_prep_phrases(prep_phrase-(P, np-(np-B)), A).

more_prep_phrases(First, (First, Articulation, Preps)) -->
    micro_articulation(Articulation),
    prep_phrase(Preps).
more_prep_phrases(First, First) --> {true}.


/*
Qualifying prepositional phrases are called for in post-noun sequences of a noun phrase.
An example would be «(une gamme) de rayonnages» or «(votre lettre) du 3 janvier 1988»
*/

qualifying_prep_phrase(QPP) -->
    terminal(prep-de, A),
    date(Date),
    more_q_p_phs(qualifying_prep_phrase-(prep-A, date-Date), QPP).
qualifying_prep_phrase(QPP) -->
    terminal(prep-de, A),
    name(Name),
    more_q_p_phs(qualifying_prep_phrase-(prep-A, name-Name), QPP).
qualifying_prep_phrase(QPP) -->
    terminal(prep-de, A),
    one_np(NP),
    more_q_p_phs(qualifying_prep_phrase-(prep-A, np-NP), QPP).

more_q_p_phs(First, (First, Articulation, Preps)) -->
    micro_articulation(Articulation),
    qualifying_prep_phrase(Prep),
    more_q_p_phs(Prep, Preps).
more_q_p_phs(Prep, Prep) --> {true}.
```

```
/*
Word groups that form prepositions. See Grévisse §2247 for many more.
These are commented out, but may be added as required. They could also be tightened
up using peek-ahead routines and the à_ou_de predicate as defined in the sentence components
file.

prep(locution_prépositive-(A,B,C)) -->
    terminal(prep-de, A),
    terminal(noun-(X, _), B),
    {member(X,[façon,manière,crainte])},
    terminal(prep-à, C).

prep(locution_prépositive-(A,B,C,D)) -->
    terminal(prep-de, A),
    terminal(art-(le,_), B),
    terminal(noun-(côté, _), C),
    terminal(prep-de, D).

prep(locution_prépositive-(A,B,C)) -->
    terminal(prep-à, A),
    terminal(noun-(X, _), B),
    {member(X,[cause,côté,force,moins,raison])},
    terminal(prep-de, C).

prep(locution_prépositive-(A,B,C,D)) -->
    terminal(prep-à, A),
    terminal(art-(le,_), B),
    terminal(noun-(X, _), C),
    {member(X,[égard,encontre,endroit,exception,ins
u,bas,lieu,milieu,travers])},
    terminal(prep-de, D).

prep(locution_prépositive-(A,B,C,D)) -->
    terminal(prep-à, A),
    terminal(art-(les,_), B),
    terminal(noun-(X, _), C),
    {member(X,[dépens,environs])},
    terminal(prep-de, D).

prep(locution_prépositive-(A,B,C,D)) -->
    terminal(prep-à, A),
    terminal(art-(le,_), B),
    [p(pct,-)],
    terminal(noun-(X, _), C),
    {member(X,[delà,dessus,dessous])},
    terminal(prep-de, D).

prep(locution_prépositive-(A,B)) -->
    terminal(adv-(X,_), A),
    {member(X,[auprès,autour,avant,faute,hors,loin,p
rès])},
    terminal(prep-de, B).

prep(locution_prépositive-(A,B)) -->
    terminal(prep-afin, A),
    terminal(prep-de, B).

prep(locution_prépositive-(A,B)) -->
    terminal(prep-à, A),
    terminal(noun-(travers, _), B).

prep(locution_prépositive-(A,B)) -->
    terminal(verb-(être,(pres-part,m,s),_), A),
    terminal(verb-(donner, (pres-part,_),_), B).

prep(locution_prépositive-(A,B)) -->
    terminal(noun-(face,_), A),
    terminal(prep-à, B).
prep(locution_prépositive-(A,B)) -->
    terminal(noun-(grâce,_), A),
    terminal(prep-à, B).
prep(locution_prépositive-(A,B)) -->
    terminal(prep-jusque, A),
    terminal(prep-à, B).
prep(locution_prépositive-(A,B,C)) -->
    terminal(prep-par, A),
    terminal(noun-(rapport,_), B),
    terminal(prep-à, C).
prep(locution_prépositive-(A,B)) -->
    terminal(prep-quant, A),
    terminal(prep-à, B).
prep(locution_prépositive-(A,B,C,D)) -->
    terminal(noun-(vis,_), A),
    [p(pct,-)],
    terminal(prep-à, B),
    [p(pct,-)],
    terminal(noun-(vis,_), C),
    terminal(de,prep,D).
prep(locution_prépositive-(A,B,C,D,E,F)) -->
    terminal(noun-(vis, _), A),
    [p(pct,-)],
    terminal(prep-à, C),
    [p(pct,-)],
    terminal(noun-(vis, _), E),
    terminal(prep-de, F).
*/
```

```
/*****************************************************************************
*************************************************** ***************
PARSER - SENTENCE COMPONENTS
© 1987, 1988 Victor Loewen
*********************************************************** ***********
*************************************************************** *********/

/*
This call inserted here simply to facilitate reconsulting this file during its development.
SAVE FILE FIRST, SINCE dcgfile READS FROM DISK AND NOT FROM MEMORY.

:- dcgfile('ORDINOTRAD:Ordino Code:Parser - s components').
*/

/*
VARIOUS SENTENCE COMPONENTS (listed in alphabetical order)
Note that some of the sentence components are particularly relevant in the sublanguage
used in administrative correspondence in that they are mentions and can stand alone.
*/

à_ou_de(X) --> terminal(prep-à, X).
à_ou_de(X) --> terminal(prep-de, X).

/*
Typical lines of French addresses:
    9999, rue Proper Noun
    City (Prov.) or City (Province)

Note that postal codes and place names standing alone on a line or together on the same line
 are treated separately by "postal code" and "name" respectively.
e.g.
    ANA NAN (postal code)
    Place name, ANA NAN
The rules below are for the first two types of addresses given above.
*/

address((number(N,cardinal), Name)) -->
    [n(N,cardinal)],
    [p(pct,',')],
    terminal(noun-(Way_type, _), A),
    {member(Way_type, [rue, avenue, chemin, boulevard])},
    place_name(A, Name).
address((np-N1, np-N2)) -->
    one_np(N1),
    [p(pct,'(')],
    one_np(N2),
    [p(pct,')')].
address((np-N1, np-N2)) -->
    one_np(N1),
    [p(pct,',')],
    one_np(N2).

        place_name(Way, (Way, qualifying_prep_phrase-Name)) -->
            qualifying_prep_phrase(Name).
        place_name(Way, (Way, np-Name)) -->
            one_np(Name).
        place_name(Way, Way) --> {true}.
```

```
adverbial(A) -->
    terminal(adv - _, B),
    more_adverbials(B, A).

    more_adverbials(First, (First, p-(,), Advs)) -->
        [p(pct,,)],
        adverbial(Advs).
    more_adverbials(First, (First, Advs)) -->
        adverbial(Advs).
    more_adverbials(Adv, Adv) --> {true}.
/* Add «à travers» sometime as an adverb.*/


alpha(A) -->
    terminal(string-X, A),
    {letter(X,_)}.


/*
Only the "cc" part of a carbon copy notation need be analyzed here, as the names
following it should be handled by the name definition. The punctuation between it and
the following names is dropped, as a colon is produced in generation.
e.g.
    c.c. - Monsieur A. Légaré
    CC : M. Ardec
    Copie : Madam Ardec
*/

cc((A, B)) -->
    terminal(string-c, A),
    terminal(string-c, B),
    [p(pct, P)].
cc((A)) -->
    terminal(string-cc, A),
    [p(pct, P)].
cc(A) -->
    terminal(noun-(copie,_), A),
    [p(pct, P)].


character_or_handling(character-A) -->
    terminal(adj-(X,_),A),
    {member(X, [personnel, confidentiel])}.
character_or_handling(character-lu(F,adj-(recommander,verbal-desc, Desc))) -->
    slot5(lu(F, adj -(recommander, verbal - desc, Desc))).

character_or_handling(handling-(A,B)) -->
    terminal(prep-par, A),
    post_par(B).

    post_par(A) -->
        terminal(noun-(B, _), A),
        {member(X, [avion, exprès])}.


dactylos_initials((A, symb-S, B)) -->
    terminal(string-_, A),
    [s(sym, S)],
    terminal(string-_, B).
```

```
/*
Date will always return five arguments (le, Weekday, Day, Month, Year) of which some may
be nil.  Date will not succeed if there is no Day or Month.
*/

date(A) -->
    terminal(art-(le, _), B),
    post_le(art-B, A).
date(A) -->
    post_le(nil, A).

    post_le(LE, (LE, weekday-nil, year-number(Year, cardinal),
        month-number(Month, cardinal),
        day-number(Day, cardinal))) -->  % assuming international notation is used
            [n(Year, cardinal)],
            [p(pct, -)],
            [n(Month, cardinal)],
            [p(pct, -)],
            [n(Day, cardinal)].
    post_le(LE, (LE, Date)) -->
        terminal(noun-(X, _), A),
        {member(X, [lundi, mardi, mercredi, jeudi, vendredi, samedi, dimanche])},
        post_weekday(weekday-A, Date).
    post_le(LE, (LE, Date)) -->
        post_weekday(weekday-nil, Date).

        post_weekday(Weekday, (Weekday, Date)) -->
            [n(N, cardinal)],
            post_day(day-number(N, cardinal), Date).
        post_weekday(Weekday, (Weekday, Date)) -->
            post_day(day-nil, Date).

            post_day(Day, (Day, Date)) -->
                terminal(noun-(X, _), A),
                {member(X, [janvier, février, mars, avril, mai, juin, juillet, août, septembre,
                    novembre, décembre])},
                post_month(month-A, Date).

                post_month(Month, (Month, year-number(N, cardinal))) -->
                    [n(N, cardinal)], !.
                post_month(Month, (Month, year-nil)) --> [true, !].


/*
ENCLOSURES
The only part of "enclosure" mentions that needs to be defined here is the "pièce jointe"
part.  The remainder is processable under the other definitions for mention.
e.g.,
3 pièces jointes:  2 lettres, 1 tableau
3 p.j. - lettre, tableau, livre
p.j.
pièce jointe (note that this must not be treated as a noun phrase)

There are therefore three "slots":
(number) + p.j. | P.J. | pièce jointe + - | : | ø

Note that the periods were removed during preprocessing when p.j. was
determined to be two abbreviations.
*/
```

```
encl(A) -->
    [n(N, T)],
    more_encl(number(N, T), A).
encl(A) -->
    more_encl(nil, A).

    more_encl(Num, (Num, Encl)) -->
        terminal(string-p, A),
        terminal(string-j, B),
        more_more_encl((A, p-(.), B, p-(.)), Encl).
    more_encl(Num, (Num, Encl)) -->
        terminal(noun-(pièce,_), A),
        slot5(lu(F, adj -(joindre, verbal - desc, Desc))),
        more_more_encl((noun-A, adj-lu(F, adj -(joindre, verbal - desc, Desc))), Encl).

            more_more_encl(A, (A, p-('-'))) -->
                [p(pct, '-')].
            more_more_encl(A, (A, p-(':'))) -->
                [p(pct, ':')].
            more_more_encl(A, A) --> {true}.


et_ou(A) --> terminal(conj-(et,_), A).
et_ou(A) --> terminal(conj-(ou,_), A).

interjection(A) --> terminal(interjection-_, A).

lquel(A) --> terminal(pro-(lequel, _), A).
lquel(A) --> terminal(pro-(laquelle, _), A).
lquel(A) --> terminal(pro-(lequels, _), A).
lquel(A) --> terminal(pro-(lesquelles, _), A).

macro_articulation(articulation-lu(Format, conj-(et,Type))) -->
    [lu(Format,[conj-(et,Type)])].
macro_articulation(articulation-lu(Format, conj-(ou,Type))) -->
    [lu(Format,[conj-(ou,Type)])].
macro_articulation(articulation-(p-(,), lu(Format, conj-(et,Type)))) -->
    [p(pct,,)], [lu(Format,[conj-(et,Type)])].
macro_articulation(articulation-(p-(,), lu(Format, conj-(ou,Type)))) -->
    [p(pct,,)], [lu(Format,[conj-(ou,Type)])].
macro_articulation(articulation-(p-(,))) --> [p:pct,,)].

/*
The order of the following is important, as "c.c." and "p.j." must not be treated as
names.
*/

mention(A) --> date(B), more_mention(date-B, A).
mention(A) --> subject(B), more_mention(subject-B, A).
mention(A) --> encl(B), more_mention(enclosures-B, A).
mention(A) --> cc(B), more_mention(carbon_copy-B, A).
mention(A) --> reference_material(B), more_mention(reference-B, A).
mention(A) --> postal_code(B), more_mention(postal_code-B, A).
mention(A) --> phone_number(B), more_mention(phone_number-B, A).
mention(A) --> character_or_handling(B), more_mention(B, A).
mention(A) --> dactylos_initials(B), more_mention(dactylos_initials-B, A).
mention(A) --> address(B), more_mention(address-B, A).
mention(A) --> name(B), more_mention(name-B, A).
mention(A) --> salutation(B), more_mention(salutation-B, A).
mention(A) --> one_np(B), more_mention(np-B, A).
mention(A) --> prep_phrase(B), more_mention(prep_phrase-B, A).
```

```
mention(A) --> interjection(B), more_mention(interjection-B, A).
mention(A) --> [s(sym,('/'))], more_mention(symb-('/'), A).
mention(A) --> [p(pct ,'.')], more_mention(p-('.'), A).

more_mention(First, (First, Mentions)) -->
    mention(Mentions).
more_mention(First, First) --> {true}.
```

/*
According to *Café II*, p. 230, "la perluète '&' remplace la conjonction 'et' dans les raisons sociales." Maybe this will have to be added here.
*/

```
micro_articulation(articulation-lu(Format, conj-(et,Type))) -->
    [lu(Format,[conj-(et,Type)])].
micro_articulation(articulation-lu(Format, conj-(ou,Type))) -->
    [lu(Format,[conj-(ou,Type)])].
micro_articulation(articulation-(p-(,))) --> [p(pct,,)].
```

/*
Names may be of several sorts and include "raisons sociaux" and names of places:
Monsieur J. Ardec
M. J.-P. Ardec
Jean-Paul Ardec
Ardec et Fils
Ardec et Fils Inc.
Ardec, Fils et Cie
Ardec Ltée
Ardec et Cie
Montréal
etc.
Names may not conclude with commas, since this restriction helps distinguish them from salutations (e.g., "Marie,").
*/

```
name(A) --> terminal(noun -(_, _, _, proper, _), B), rest_of_name(name-B, A).
name(A) --> alpha(B), rest_of_name(initials-B, A).

    rest_of_name(First, (First, p-(','), Names)) -->
        [p(pct,',')],
        name(Names).
    rest_of_name(First, (First, p-(','), Names)) -->
        [p(pct,',')],
        terminal(noun-(N, _), B),
        {member(N, [fils, compagnie])},
        rest_of_name(name-B, Names).
    rest_of_name(First, (First, p-(','), conj-A, Names)) -->
        [p(pct,',')],
        terminal(conj-(et,_), A),
        name(Names).
    rest_of_name(First, (First, symb-(/), Names)) -->
        [s(sym,/)],
        name(Names).
    rest_of_name(First, (First, p-(-), Names)) -->
        [p(pct,-)],
        name(Names).
    rest_of_name(Name, (Name, conj-A, B_plus_inc)) -->
        terminal(conj-(et, _), A),
        terminal(noun-(N, _), B),
        {member(N, [fils, compagnie])},
```

```
        rest_of_name(name-B, B_plus_inc).
    rest_of_name(Name, (Name, conj-A, Names)) -->
        terminal(conj-(et,_), A),
        name(Names).
    rest_of_name(Name, (Name, adj-lu(F, adj -(Adj, verbal - desc, Desc)))) -->
        slot5(lu(F, adj -(Adj, verbal - desc, Desc))),
        {member(Adj, [incorporer, limiter])}.
    rest_of_name(First, (First , Names)) -->
        name(Names).
    rest_of_name(First, First) --> {true}.
```

/*
The following rules for phone numbers allow for the following formats:
    (xxx) xxx-xxxx
    xxx-xxx-xxxx
    xxx-xxxx
More rules will have to be added if international numbers are to be handled.
*/

```
phone_number((number(N, cardinal),number(N2, cardinal),number(N3, cardinal))) -->
    [p(pct,'(')],
    [n(N, cardinal)],
    [p(pct,')')],
    [n(N2, cardinal)],
    [p(pct,-)],
    [n(N3, cardinal)].
phone_number((number(N, cardinal),number(N2, cardinal),number(N3, cardinal))) -->
    [n(N, cardinal)],
    [p(pct,-)],
    [n(N2, cardinal)],
    [p(pct,-)],
    [n(N3, cardinal)].
phone_number((number(nil,nil),number(N2, cardinal),number(N3, cardinal))) -->
    [n(N2, cardinal)],
    [p(pct,-)],
    [n(N3, cardinal)].
```

/*
If a comma was put in the French, between a place name and the postal code, the comma
is dropped, since there is to be no comma either in French or in English.
*/

```
postal_code((A,number(N,cardinal),A2,number(N2,cardinal),A3,number(N3,cardinal))) -->
    [p(pct,',')],
    alpha(A), [n(N,cardinal)],
    alpha(A2), [n(N2,cardinal)],
    alpha(A3), [n(N3,cardinal)].
postal_code((A,number(N,cardinal),A2,number(N2,cardinal),A3,number(N3,cardinal))) -->
    alpha(A), [n(N,cardinal)],
    alpha(A2), [n(N2,cardinal)],
    alpha(A3), [n(N3,cardinal)].

proper_name([noun-[LU,Format,Desc]]) -->
    terminal(_, noun, noun-[LU,Format,Desc]),
    {Format = cap ; Format = all_caps}.
proper_name([A, noun-[LU,Format,Desc]]) -->
    terminal(de,prep,A),
    terminal(_, noun, noun-[LU,Format,Desc]),
    {Format = cap ; Format = all_caps}.
```

```
que_qui(A) --> terminal(pro-(que, _), A).
que_qui(A) --> terminal(pro-(qui, _), A).
```

/*
References are complex mentions in that they may be quite unpredictable. For the sake of this system, and until other formats need to be accounted for, the following are expected to be typical references. (Clas-Horguelin, p. 248)

e.g., V/référence du 88-12-09
V/référence COM-28-D
V/Référence COM1085
V/Réf. ####/du date
V/R. ####/du date

In the above examples, "référence" could be replaced with "lettre" or some other form of correspondence, and "V" could be replaced with "N".

Note that an entry such as "Notre lettre du 88-10-09" should be handled by the noun phrase definition (making a call to qualifying_prep_phrase, de + date; see preposition file).

"Ref" will have been morphologically analyzed as an abbreviation of "référence" and the full form will be treated here. Yet the period which follows needs to be dropped.

The slash is dropped, as it is purely functional.
*/

```
reference_material(A) -->
    terminal(string-X, B),
    {member(X, [v,n])},
    [s(sym, '/')],
    post_v_n(abbrev-B, A).

    post_v_n(A, (A, C)) -->
        one_np(B),
        post_ref(np-B, C).
    post_v_n(A, (A, abbrev-R, qualifying_prep_phrase-QPP)) -->
        alpha(R),
        qualifying_prep_phrase(QPP).
    post_v_n(A, (A, B)) -->
        alpha(R),
        post_ref(abbrev-R, B).

        post_ref(First, (First, code-A)) -->
            alpha_numeric_code(A).
        post_ref(First, First) --> {true}.

            alpha_numeric_code(A) -->
                terminal(string-_, B),
                more_code(B, A).
            alpha_numeric_code(A) -->
                [n(N, cardinal)],
                more_code(number(N, cardinal), A).

                more_code(First, (First, p-(-), Code)) -->
                    [p(pct,(-))],
                    alpha_numeric_code(Code).
                more_code(First, (First, Code)) -->
                    alpha_numeric_code(Code).
```

```
more_code(First, First) --> {true}.
```

/* the following needs to be standardized */

```
rel_pro(X) --> terminal(pro-(que, _), X).
rel_pro((X,Y)) --> à_ou_de(X), lquel(Y).
rel_pro((X,Y)) --> terminal(prep-à, X), terminal(pro-(quoi, _), Y).
```

/*
Commas surrounding a right dislocation are dropped as they are functional only and do not provide meaning. If commas are required in the English translation, they are a function of the relationship between the parts of the English sentence and are not a "translation" of the comma provided in the French input.

Conjunctions are sought at this point since they occur at this point in sentences as well. They are not called right dislocations, however, since the latter are effectively put to the right of sentences in generation. Conjunctions end up at the beginning of sentences in English.
*/

```
right_dislocation(conjunction-A) -->
    terminal(conj-(LU, _), A),
    {member(LU, [ainsi, donc, pourtant])}.  % more can be added if needed
right_dislocation(right_dislocation-(A, B)) -->
    terminal(pro-(_,pers,_), A),
    [p(pct,-)],
    terminal(adj-(même,_), B).
right_dislocation(right_dislocation-A) -->
    [p(pct,',')],
    name(A),
    [p(pct,',')].
```

/*
Salutations are comprised of:
"cher/chère/chers/chères" + name/names + noun-phrase (et amis) + comma
"cher/chère/chers/chères" + name/names + comma
"cher/chère/chers/chères" + noun-phrase (ami) + comma
name/names + noun-phrase (et amis) + comma
name/names + comma
*/

```
salutation((adj-A, POST)) -->
    terminal(adj - _, A),
    name(B),
    post_name(name-B, POST),
    [p(pct, ',')].
salutation(SALUTATION) -->
    name(A),
    post_name(name-A, SALUTATION),
    [p(pct, ',')].
salutation((adj-A, np-B)) -->
    terminal(adj - (cher, _), A),
    one_np(B),
    [p(pct, ',')].
salutation(np-A) -->
    one_np(A),
    [p(pct, ',')].

    post_name(Name, (Name, p-(','), conj-A, np-B)) -->
        [p(pct, ',')].
```

```
        terminal(conj-_, A),
        one_np(B).
    post_name(Name, (Name, conj-A, np-B)) -->
        terminal(conj-_, A),
        one_np(B).
    post_name(Name, Name) --> {true}.
```

/*
For "subject line" mentions, the allowance here is for any number of mentions
preceded by "object"
*/

```
subject((A, p-(':'))) -->
    terminal(noun-(object,_), A),
    [p(pct,':')].
```

```
/*************************************************************************
*************************************************************************
PARSER - AGREEMENTS
© 1987, 1988 Victor Loewen
*************************************************************************
*************************************************************************/
```

/*
SUBJECT - VERB AGREEMENT
Agreement between subjects and verbs is required to prevent incorrect parsing of ellipsed propositions. (e.g. Nous sommes heureux... et vous remercions de votre...)

The "sv_agreement" predicate changes the information contained in the NP if it is ambiguous. Information about pronouns is not really changed but only verified. Information about nouns may be changed if the person and number of the verb disambiguates person and number information.

If the verb was composite in the surface structure (e.g., passé composé), person/number information was taken from the auxiliary and added by the parser to the verb description.

Quand il y a plusieurs sujets, le verbe se met au pluriel. Si les sujets sont de personnes différentes, le deuxième l'emporte sur la troisième et la première sur les deux autres.

Si les sujets énumérés sont repris ou annoncés par un seul mot au singulier, le verbe s'accorde avec ce mot singulier.

Bernard Dupriez et collaborateurs, Cours autodidactique du français écrit - 2, Université de Montréal, 1984. p. 105.
Cf. Grévisse §1959

e.g. A et B sont ...
A et moi sommes ...
A et moi, nous sommes ...
Il et vous êtes ...
Ton frère et toi êtes...
A, B, C, tout est ...

More rules need to be developed for the agree predicate to handle all these cases.
For the moment, I just assume agreement between compound nouns and verbs.
*/

```
sv_agreement(event-lu(Format,verb-(Vb,(T,M, Pers1, Num1), Frame)),
    event-lu(Format,verb-(Vb,(T,M, Person, Num), Frame)),
    np-(noun-lu(F2,noun-(Nn,G,Num2,PC,CM,Means)),modifiers-Post),
    np-(noun-lu(F2,noun-(Nn,G,Num,PC,CM,Means)),modifiers-Post)) :-
        match(Pers1, 3, Person),
        match(Num1, Num2, Num),
        succeed, !.

sv_agreement(event-lu(Format, verb -(Vb,(T,M,Pers1, Num1), Frame)),
    event-lu(Format, verb -(Vb,(T,M,Person, Number), Frame)),
    np-lu(F,pro -(LU, Type, Pers2, Num2, Gender, Meaning)),
    np-lu(F,pro -(LU, Type, Person, Number, Gender, Meaning))) :-
        match(Pers1, Pers2, Person),
        match(Num1, Num2, Number),
        succeed, !.

sv_agreement(event-lu(Format, verb -(Vb,(T,M,3, pl), Frame)),
```

```
    event-lu(Format, verb -(Vb,(T,M,3, pl), Frame)),
    (np-NP, articulation-A, Rest), (np -NP, articulation-A, Rest)) :-
        succeed, !.

sv_agreement(event-lu(Format, verb -(Vb,(T,M,3, _), Frame)),
    event-lu(Format, verb -(Vb,(T,M,3, _), Frame)), name-Name, name-Name) :-
        succeed, !.

sv_agreement(Verb, Verb, NOUN, NOUN) :-
        ask_question('Subject-verb agreement could not be established.  Enter "c" to continue
with weakened parsing or an "a" to abort.', [c,a], Answer),
        (Answer = c, ! ; resetio, start_over, quits, quit).

succeed :-
    write('     Subject-verb agreement succeeded.'), nl.
```

/*

ADJECTIVE - NOUN AGREEMENT

Constructions such as «chers Jean et Marie» and «noun1 et noun2 adj_pl» seem to indicate that where there are two or more nouns, the adjectives that modify them must be in the plural. The assumption underlying the agreement checks below is, therefore, that adjectives must agree in number and gender with simple nouns, and must be plural for two or more nouns. The rules vary for gender agreement for two or more nouns (see Grévisse). Therefore, only the number is checked, and further agreement is assumed.

The following rules ensure agreement for the following types of information about adjectives and nouns. For example, given the morphological analysis of
"toutes les 345 premières heureuses lettres vraiment complètes ...":

?- translate.

Enter French-language sentence to be translated.
==> toutes les 345 premières heureuses lettres vraiment complètes sont ici.

Original French sentence given:
toutes les 345 premières heureuses lettres vraiment complètes sont ici.

Is this correct?
[yes,y,no,n] ==> y

RESULTS OF PREPROCESSING
w(toutes,low)
w(les,low)
n(345,cardinal)
w(premières,low)
w(heureuses,low)
w(lettres,low)
w(vraiment,low)
w(complètes,low)
w(sont,low)
w(ici,low)
p(.)

Type any letter and a return to continue.
==> j

RESULTS OF MORPHOLOGICAL ANALYSIS BASED ON DICTIONARY LOOKUP
lu(low,[adj-(tout,indef,f,pl,[quantity])])

```
lu(low,[art-(les,def,(m,f),pl),pro-(les,rel,3,pl,(m,f),_980)])
n(345,cardinal)
lu(low,[adj-(premier,ordinal,f,pl,[ttrelation])])
lu(low,[adj-(heureux,desc,f,pl,[attribute])])
lu(low,[noun-(lettre,f,pl,common,count,[nonselfmoving,submatter])])
lu(low,[adv-(vraiment,intensity)])
lu(low,[adj-(complet,desc,f,pl,[attribute])])
lu(low,[verb-(être,(pres,ind,3,pl),[state-[affected-(r,[something],nil),attribute-(r,[something],nil)
,beneficiary-(o,[sentient],nil)]])])
lu(low,[adv-(ici,location)])
p(pct,.)

Type any letter and a return to continue.
==> .
   Subject-verb agreement succeeded.

RESULTS OF PARSING
proposition
    independent_prop
        modalities
            pres
            ind
            active
            affirmative
            declarative
        event
            lu(low,verb-(être,(pres,ind,3,pl),[state-[affected-(r,[something],nil),attribute-(r,[something],ni
l),beneficiary-(o,[sentient],nil)]]))
        np
            np
                noun
                    lu(low,noun-(lettre,f,pl,common,count,[nonselfmoving,submatter]))
                modifiers
                    lu(low,adj-(tout,indef,f,pl,[quantity]))
                    lu(low,art-(les,def,f,pl))
                    number(345,cardinal)
                    lu(low,adj-(premier,ordinal,f,pl,[ttrelation]))
                    lu(low,adj-(heureux,desc,f,pl,[attribute]))
                    lu(lu(low,adv-(vraiment,intensity)),lu(low,adj-(complet,desc,f,pl,[attribute])))
        adv
            lu(low,adv-(ici,location))

G = gender
N = number
Mx = meaning associated with x
*/
`
adj_n_agreement(nil, nil, NOUN, NOUN).

adj_n_agreement(lu(F, adj -(LU, Type, G1, N1, Ma)),
    lu(F, adj -(LU, Type, Gen, Num, Ma)),
        lu(F2, noun-(Noun,  G2, N2, PC, CM, Mn)),
        lu(F2, noun-(Noun,  Gen, Num, PC, CM, Mn))) :-
            match(G1, G2, Gen),
            match(N1, N2, Num).

/* 2 rules for composite adjectives (adverb-adjective, or adverb adjective)*/
```

```
adj_n_agreement(lu-(Adverb, p - (-), lu(F2, adj-(LU,Type, G1, N1, Ma))),
    lu-(Adverb, lu(F2, adj -(LU, Type, Gen, Num, Ma))),
        lu(F2, noun-(Noun,  G2, N2, PC, CM, Mn)),
        lu(F2, noun-(Noun,  Gen, Num, PC, CM, Mn))) :-
            match(G1, G2, Gen),
            match(N1, N2, Num).

adj_n_agreement(lu-(lu(F1,adv -Adv), lu(F2, adj-(LU,Type, G1, N1, Ma))),
    lu(lu(F1,adv -Adv), lu(F2, adj -(LU, Type, Gen, Num, Ma))),
        lu(F2, noun-(Noun,  G2, N2, PC, CM, Mn)),
        lu(F2, noun-(Noun,  Gen, Num, PC, CM, Mn))) :-
            match(G1, G2, Gen),
            match(N1, N2, Num).

adj_n_agreement(lu(F, art-(LU, Type, G1, N1)),
    lu(F, art-(LU, Type, Gen, Num)),
    lu(F2, noun-(Noun,  G2, N2, PC, CM, Mn)),
    lu(F2, noun-(Noun,  Gen, Num, PC, CM, Mn))) :-
        match(G1, G2, Gen),
        match(N1, N2, Num).

adj_n_agreement(num(N, Type), number(N, Type), NOUN, NOUN).
adj_n_agreement(articulation -A, articulation -A, NOUN, NOUN).
adj_n_agreement(qualifying_prep_phrase-QP, qualifying_prep_phrase-QP, NOUN, NOUN).
adj_n_agreement(relative_prop-RP, relative_prop-RP, NOUN, NOUN).

adj_n_agreement((A,B), (AA, BB), N_in, N) :-
    adj_n_agreement(A, AA, N_in, N_out),
    adj_n_agreement(B, BB, N_out, N).

match(X,X,X) :- !.
match(m, (m, f), m).
match(f, (m,f), f).
match((m,f), m, m).
match((m,f), f, f).
match(s, (s, pl), s).
match(pl, (s, pl), pl).
match((s,pl), s, s).
match((s,pl), pl, pl).

match(1, (1,2), 1).
match(1, (1,3), 1).
match((1,2), 1, 1).
match((1,3), 1, 1).
match(2, (1,2), 2).
match(3, (1,3), 3).
match((1,2), 2, 2).
match((1,3), 3, 3).
```

```
/*************************************************************************
*************************************************************************
CASE PROCESSOR
© 1987, 1988 Victor Loewen
*************************************************************************
*************************************************************************/

/*
An input sentence is deemed syntactically well formed if it satisfies the grammar rules of
the parser. A by-product of the parse is a syntactic tree which is submitted here to a case
processor which determines if the input sentence is semantically well formed. A by-product
of the case analysis is a cased tree, where case labels are nodes superceding syntactic labels which
in turn, supercede subordinate syntactic labels for word groups, and eventually syntactic labels
for terminals. Labels for role-relations are not applied across generations; i.e., the event node and
the case-label nodes in relation to it are all of the same level. This avoids confusing the
role players when a sentence contains more than one proposition (subordinate or relative clauses, or
non-finite event propositions).

Case labels are recursively applied generation by generation, going from embedded propositions
to the larger propositions.

The purpose of the case processor is not to carry out transformations (i.e., make radical changes to) the
tree produced by the parser. It simply checks for semantic well-formedness and adds an extra
layer of semantic labels. The order of complements is, however, changed, as the required
slots are filled first, then optional ones, then peripheral roles.


When a terminal is reached during a tree search for event propositions, the terminal
must be returned. These terminals do not contain events and do not need to be scanned.
*/

case_processor(lu(A,B), lu(A, B)) :- !.
case_processor(number(A,B), number(A, B)) :- !.
case_processor(nil, nil) :- !.
case_processor([], []) :- !.
case_processor(p-(P), p-(P)) :- !.
case_processor(articulation-A, articulation-A) :- !.
case_processor(postal_code-PC, postal_code-PC) :- !.
case_processor(symb - (S), symb - (S)).

/*
There are only five kinds of propositions which contain an event.
independent propositions, ellipsed propositions, event propositions (infinitives and participles),
relative propositions, and qualifying event propositions (event propositions beginning preceded by
a preposition and modifying a noun. Only these are passed
on to the semantic analysis of the case processor. Before passed on to case analysis, however,
they themselves are checked to make sure that any embedded propositions are
analyzed first.
Changes in the structure include:
    the list of meaning-frame combinations is dropped, if the siblings of an event satisfy the
        semantic requirements of its frame;
    the meaning associated with the successful frame is the one that is kept with the verb
        to provide semantic control during transfer;
    case-role nodes are added to the siblings of an event.
*/
```

```
case_processor([modalities-Ms,event-lu(F,verb-(LU,TMPN,M_fs))|Siblings],
    [modalities-Ms, event-lu(F,verb-(LU,TMPN,M))|Cased_siblings]) :-
    !,
    case_processor(Siblings, Checked_siblings),
    sem_prop(Ms, M_fs, M, Checked_siblings, Cased_siblings), !.
```

/*
There are four kinds of word groupings whose elements could include a type of proposition (see above) which contains an event. They must be checked through for such sub-trees, but are not themselves passed on to the semantic analysis of the case processor.
*/

```
case_processor((A, B), (AA, BB)) :-
    case_processor(A, AA),
    case_processor(B, BB).

case_processor(np - (Noun, Modifiers), np-(Noun, Cased_Modifiers)) :-
    case_processor(Modifiers, Cased_Modifiers), !.

case_processor(Node - Branch, Node-Cased_Branch) :-
    case_processor(Branch, Cased_Branch), !.

case_processor([Type-Tree | Tail], [Type-Checked_tree | Cased_props]) :-
    case_processor(Tree, Checked_tree),
    case_processor(Tail, Cased_props).

case_processor(_, failed) :-
    nl,
    write('Sorry.  The relevant dictionary entries need to be checked for validity in terms
of their relation to reality.  Check the meanings of nouns, and the semantic restrictions on
the cases for the verbs.'),
    nl, fail.
```

/*
A generation (i.e. sibling nodes on the tree) is scanned for the event which contains the frame governing that generation.
The frame is then applied to the generation. (i.e., slots in the frame are filled by the appropriate nodes of the tree which meet the semantic requirements indicated in the case frame).

A special rule is required to make the connection between "en" before a verb and a cardinal adjective following the verb. For example, "Je leur ai adressé trois lettres" and "Je leur en ai adressé trois" should both fit the same case frame: agent-je, beneficiary-leur, affected-trois lettres (en + trois).
To make the rule more general, to handle sentences of the form "cette lettre nous en adresse les besoins," the assumption is made that what the "en" refers to will be immediately after it at this point in the processing.
Care must be taken, however, not to link "en" with what follows in sentences such as "j'en parle dans mon discours."
The controlling factor is therefore set as "en + adjective" and "en + np" will work, but "en + prepositional phrase" will not.
*/

```
sem_prop(Ms, [Meaning-Frame | _], Meaning, Siblings, Labelled_siblings) :-
    find_en(Siblings, Siblings_with_en),
    alter_frame(Ms, Frame, Altered_frame),
    apply_frame(Altered_frame, Siblings_with_en, Labelled_siblings), !.

find_en(Siblings, Siblings_with_en) :-
```

```
    member(np -(np-lu(_,pro-(en, rel, 3, _))), Siblings),
    link_en(Siblings, Siblings_with_en).

link_en(Siblings, Siblings_with_en) :-
    app(Fore, [np -(np-lu(low,pro-(en, Desc))), np -(np -(noun-N, modifiers-M))],
        Siblings),
    app(Fore, [np-(np-(noun-N, modifiers-(lu(low,pro-(en, Desc)), M)))],
        Siblings_with_en).
```

/*
A special rule is required for handling impersonal sentences where both the surface subject and object may share the same role; e.g., «il vous sera facile de choisir», where «il = de choisir...».
In all cases, though, an impersonal "il" before a state verb is the affected.
The following rule will work for all state verbs.
The procedure used is to treat the impersonal «il» as a function word and temporarily drop it, thus forcing the slot filler to find another filler for the affected slot.
After case processing, there may therefore be two affected case roles.
At this point, the parser has identified "il" as a personal pronoun. It is at this point that this information is corrected and "il" is an impersonal (relative) pronoun. This will force the generator to produce "it" instead of "he."
Rules will also need to be drawn up for tenir compte and recevoir used in the passive.
see Salkoff, 1973, pp. 19ff.
*/

```
% GIVEN A SENTENCE BEGINNING WITH il AND A state VERB
sem_prop(Ms,[state-Frame | MF], state, [np-(np-lu(F,pro-(il,pers,3,s,G,M))) | Tail],
    [Case-(np-(np-lu(F,pro-(il,rel,3,s,G,M)))), Case-(event_prop-Prop) | Cased_siblings]) :-

% AND IF THERE IS AN EVENT PROPOSITION AS A SIBLING
    cut_out(event_prop-P, Tail, Other_siblings),

% THEN REARRANGE AND RUN THE SENTENCE THROUGH THE CASER
    sem_prop(Ms, [state-Frame], state, [event_prop-P | Other_siblings],
        [Case-(event_prop-Prop) | Cased_siblings]).
```

/*
A special rule is needed for basic impersonal sentences.
e.g.
Il vous sera facile.
Here the "il" is changed from personal to relative.
*/

```
sem_prop(Ms,[state-Frame | MF], state, [np-(np-lu(F,pro-(il,pers,3,s,G,M))) | Tail],
        Cased_siblings) :-
    sem_prop(Ms, [state-Frame], state, [np-(np-lu(F,pro-(il,rel,3,s,G,M))) | Tail],
        Cased_siblings).


sem_prop(Ms, [Meaning-Frame | _], Meaning, Siblings, Labelled_siblings) :-
    alter_frame(Ms, Frame, Altered_frame),
    apply_frame(Altered_frame, Siblings, Labelled_siblings), !.

sem_prop(Ms, [_ | Meaning_frames], Meaning, Siblings, Labelled_siblings) :-
    sem_prop(Ms, Meaning_frames, Meaning, Siblings, Labelled_siblings).

sem_prop(_, [], _) :-
    nl,
    write('Case analysis failed.  Ensure that the frame of the main event reflects reality
and that sentence components have meanings that will satisfy a frame.  Re-enter sentence.'),
    nl, resetio, quit.
```

```
/*
Required agents must be dropped from Frames where the event is in the surface syntactic
form of an infinitive. They must be made optional where the event is in the surface form of
a passive ([avoir + été + past participle] or present participle (gerund).
At this point, there should be no auxiliary verbs other than modal auxiliaries.
Where there are auxiliary verbs, the frame is derived from the participle or infinitive
(the auxiliary has already provide the tense, mood, and voice).
Furthermore, the frames of auxiliary verbs are dropped.

Required "affecteds" become optional if the affected slot is filled with a negative which
does not exist as a sibling (the negative is dealt with as a modality).
e.g.
    je ne vois personne.
    je ne fais rien.
*/

alter_frame((_,ind,active, negative - Rien, _), Frame_in, Frame_out) :-
    member(Rien, [rien, personne]),
    substitute(result-(r, Desc), result-(o, Desc), Frame_in, Frame1),
    substitute(affected-(r, Desc2), affected-(o, Desc2), Frame1, Frame_out).
alter_frame((_,(ind, subj),active, negative - Rien, _), Frame_in, Frame_out) :-
    member(Rien, [rien, personne]),
    substitute(result-(r, Desc), result-(o, Desc), Frame_in, Frame1),
    substitute(affected-(r, Desc2), affected-(o, Desc2), Frame1, Frame_out).
alter_frame((_,inf,active, negative - Rien, _), Frame_in, Frame_out) :-
    member(Rien, [rien, personne]),
    substitute(result-(r, Desc), result-(o, Desc), Frame_in, Frame1),
    substitute(affected-(r, Desc2), affected-(o, Desc2), Frame1, Frame2),
    cut(agent - (r, _), Frame2, Frame_out).
alter_frame((_,participle,active, negative - Rien, _), Frame_in, Frame_out) :-
    member(Rien, [rien, personne]),
    substitute(result-(r, Desc), result-(o, Desc), Frame_in, Frame1),
    substitute(affected-(r, Desc2), affected-(o, Desc2), Frame1, Frame2),
    cut(agent - (r, _), Frame2, Frame_out).
alter_frame((_,_,passive, negative - Rien, _), Frame_in, Frame_out) :-
    member(Rien, [rien, personne]),
    substitute(result-(r, Desc), result-(o, Desc), Frame_in, Frame1),
    substitute(affected-(r, Desc2), affected-(o, Desc2), Frame1, Frame2),
    substitute(agent - (r, Meaning_prep), agent-(o, Meaning_prep), Frame2, Frame_out).
alter_frame((_,participle,passive, negative - Rien, _), Frame_in, Frame_out) :-
    member(Rien, [rien, personne]),
    substitute(result-(r, Desc), result-(o, Desc), Frame_in, Frame1),
    substitute(affected-(r, Desc2), affected-(o, Desc2), Frame1, Frame2),
    substitute(agent - (r, Meaning_prep), agent-(o, Meaning_prep), Frame2, Frame_out).

alter_frame((_,ind,active, _), Frame, Frame).
alter_frame((_,(ind, subj),active, _), Frame, Frame).
alter_frame((_,inf,active, _), [agent - (r, _) | Rest], Rest).
alter_frame((_,participle,active, _), [agent - (r, _) | Rest], Rest).
alter_frame((_,_,passive, _),
    [agent - (r, Meaning, Prep) | Rest], [agent - (o, Meaning, Prep) | Rest]).
alter_frame((_,participle,passive, _),
    [agent - (r, Meaning, Prep) | Rest], [agent - (o, Meaning, Prep) | Rest]).


/*
apply_frame :-
    use required labels first
    use optional labels second, if applicable
*/
```

```
apply_frame(Frame, Siblings, Labelled_siblings) :-
    fill_slots(r, Frame, Siblings, S1),
    fill_slots(o, Frame, S1, S2),
    fill_slots(o, [location-(o,[location],nil), time-(o,[time],nil),
        manner-(o,[manner],nil), cause-(o,[cause],nil)], S2, Labelled_siblings).


/*
Required slots must be filled. If not, the meaning-frame combination called
for by the verb is the wrong combination (where there is more than one combination, e.g.,
servir has three meaning-frame combinations) or the frame for the verb has not been
defined to allow for the possible lexicalization of the frame.
The entire list of siblings is scanned for each slot. If something fills a slot, it is removed
from the list and added to a list of labelled siblings.
Once a slot is filled, it is removed from the frame and the remaining slots in the frame
are checked.
*/

fill_slots(_, [], Siblings, Siblings).

fill_slots(Type, [Case-(Type, Criteria, Prep) | L1], S_in, S_out) :-
    fill_one_slot(Type, Case, Criteria, S_in, S_temporary),
    fill_slots(Type, L1, S_temporary, S_out).

fill_slots(Type1, [Case-(Type2, Criteria, Prep) | L1], S_in, S_out) :-
        Type1 \= Type2,
        fill_slots(Type1, L1, S_in, S_out).

fill_one_slot(o, _, _, [], []).
fill_one_slot(Type, Case, Criteria, [Node-Desc | Siblings], [Case-(Node-Desc) | Siblings]) :-
    member(Node, [np, adj, adv, prep_phrase, event_prop, relative_prop]),
    acceptable(Case, Criteria, Node-Desc).
fill_one_slot(Type, Case, Criteria, [H | Siblings], [H | Rest]) :-
    fill_one_slot(Type, Case, Criteria, Siblings, Rest).


/*
Checks must be made to see if a given word group is acceptable for a given case relation
and if it meets the semantic constraints placed upon by that case relation by the verb
to which it is related.
*/

acceptable(agent, Constraint, np-Desc) :-
    satisfy(Desc,Constraint).

/*
for sentences such as "j'ai trois"
*/
acceptable(affected,Constraint,np-(np-(noun-nil,
    modifiers-(lu(_,adj-(_,cardinal,_)),_)))).

/*
for sentences such as "j'ai trois des lettres"
*/
acceptable(affected,Constraint,np-(np-(noun-nil,
    modifiers-(lu(_,adj-(_,cardinal,_,_,_)), qualifying_prep_phrase-_)))).
```

```
/*
for sentences such as "j'en ai trois"
*/
acceptable(affected,Constraint,np-(np-(noun - nil,
    modifiers-(lu(_,pro-(en,_)), lu'_,adj-(_,cardinal,_)), _))).

acceptable(affected, Constraint, np-Desc) :-
    satisfy(Desc, Constraint).
acceptable(affected, Constraint, relative_prop-RP).
acceptable(affected, Constraint, event_prop-Desc).
acceptable(affected,Constraint,prep_phrase-PP) :-
    satisfy(PP,Constraint).

/*
Subjective completions may be comprised of nouns without any adjectives:
e.g. Je suis étudiant.
or "en" + cardinal adjective can fill attribute slots. See the definitions of sem_prop
for further comments. e.g., j'en suis un.
"le" is the only pronoun preceding a verb that can satisfy the attribute slot.
e.g. Nous le sommes.
They may also be adjectives modified by an adverb.
e.g.
Je suis très heureux d'avoir reçu votre lettre de 15 juin 1988.
*/
acceptable(attribute, _, adj-lu(_,adj -(_, desc, _))).
acceptable(attribute, Constraint, adj -(lu - (lu(_,adv -_), ADJ))) :-
    acceptable(attribute, Constraint, adj-ADJ).
acceptable(attribute,Constraint,np-(np-(noun-Noun,modifiers-nil))) :-
    satisfy(Noun, Constraint).

acceptable(attribute, Constraint,np-(np-(noun - nil,modifiers-(lu(_,pro-(en, _)),
    lu(_,adj-(_, cardinal, _)), _)))).

acceptable(attribute, Constraint, np-(np-(noun - nil, modifiers-(lu(_,pro-(en, _)),
    lu(_, (_, cardinal, _)), _)))).
acceptable(attribute,[something],np-(np-lu(_,pro-(le,pers,_)))).

acceptable(attribute,Constraint,prep_phrase-PP) :-
    satisfy(PP,Constraint).

acceptable(attribute,_,event_prop-E).
% restrictions could be put on this event_prop-(M, E, LP = en mesure de, _)

/*
Subjective completions may also be adverbs.
e.g., Toutes les lettres sont ici.
*/

acceptable(attribute, Constraint, adv - Adv) :-
    satisfy(Adv, Constraint).


acceptable(beneficiary, Constraint, np-Desc) :-
    satisfy(Desc,Constraint).
acceptable(beneficiary, Constraint, prep_phrase-(prep_phrase-(Prep,np-NP))) :-
    satisfy(NP, Constraint).

acceptable(cause,_,event_prop-_).
acceptable(cause, _, prep_phrase-_).

acceptable(destination, Constraint, np -Desc) :-
```

```
    satisfy(Desc,Constraint).

acceptable(manner,_,prep_phrase-(prep_phrase-(locution_prépositive-LP, Noun))) :-
    LP = (lu(_,prep = en),lu(_,noun-(fonction,_)),lu(_,prep = de)).
acceptable(manner, Constraint, adv-_).

acceptable(location,Constraint,prep_phrase-(prep_phrase-(prep-lu(_,prep-X),np = Desc))) :-
    member(X, [dans,en]),
    satisfy(Desc, Constraint).
acceptable(location,Constraint,adv-Desc) :-
    satisfy(Desc, Constraint).

/*
c.g. Il en a fait trois
*/
acceptable(result, Constraint,np-(np-(noun = nil,modifiers-(lu(_,pro-(en, _)),
    lu(_,adj-(_, cardinal, _)), _)))).

acceptable(result,Constraint,np -Desc) :-
    satisfy(Desc, Constraint).
acceptable(result, Constraint, prep_phrase-PP) :-
    satisfy(PP, Constraint).

acceptable(time,Constraint,prep_phrase-(prep_phrase-(prep-lu(_,prep-X),np = Desc))) :-
    member(X, [dans,en]),
    satisfy(Desc, Constraint).

acceptable(time,Constraint,adv-Desc) :-
    satisfy(Desc, Constraint).

/*
Either the head noun or the head noun of a qualifying prepositional phrase may satisfy a
constraint. For example, in "Nous fabriquons une gamme complète de bibliothèques," it is
not the "gamme" but the "bibliothèques" which are manufactured, yet the noun phrase as
a whole should satisfy "result-r-[physical thing]" in the frame for "fabriquer."
*/

satisfy(lu(_,noun-(_,_,_,_,_,Meaning)), Constraint) :-
    intersect(Meaning, Constraint).

satisfy(lu(_,pro-(_, _, _, _, _, Meaning)), Constraint) :-
    intersect(Meaning, Constraint).

satisfy(lu(_,adj = (_, _, _, _, Meaning)), Constraint) :-
    intersect(Meaning, Constraint).

satisfy(lu(_,adv = (_, Meaning)), Constraint) :-
    intersect([Meaning], Constraint).

satisfy((noun-Desc, modifiers-Modifiers), Constraint) :-
    satisfy(Desc, Constraint).

satisfy(np-Desc, Constraint) :-
    satisfy(Desc, Constraint).

satisfy(name-Desc, Constraint) :-
    satisfy(Desc, Constraint).

satisfy(prep_phrase =(prep-lu(_,prep = de), np-Desc), Constraint) :-
    satisfy(Desc, Constraint).

satisfy((articulation-_, Tail), Constraint) :-
```

```
    satisfy(Tail,Constraint).

satisfy((conj-_, Tail), Constraint) :-
    satisfy(Tail,Constraint).

satisfy((Head, Tail), Constraint) :-
    satisfy(Head, Constraint),
    satisfy(Tail, Constraint).
```

/*
Note use of the isa property inheritance mechanism. It is defined in the taxonomy file.
*/

```
intersect(X,X) :- !.
intersect(Given, Constraints) :-
    mem(X, Given),
    mem(Y, Constraints),
    isa(X, Y).
```

```
/*************************************************************************
*************************************************************************
LEXICAL AND STRUCTURAL TRANSFER
© 1987, 1988 Victor Loewen
*************************************************************************
*************************************************************************/
```

/*
The first transfer procedure involves replacing textual material.

It would seem from this that translation is (more or less) word-bound. The premise here, from a computational point of view, is that every word in the French text must be taken into account. The catch, however, is whether or not the word is considered alone or along with one or more other words ("cause" or "à cause de").
Since replacing the French textual material with English material (cf. Catford) would demolish any direct evidence of possible "phrases lexicalisées", a check must first be made for such expressions.
After this, it is possible to consider all remaining words alone.

Transfer depends, therefore, on a transfer dictionary where relations between French idioms/words and English idioms/words are defined. It is assumed that semantic equivalence between entries in the transfer dictionary has been established in the very process of creating the dictionary. Even so, the transfer dictionary only matches symbols and not meanings. An English word may be given as an equivalent for a French word, yet that English word may have meanings which should not be used in the generation process (the event in an English proposition specifies what kinds of things are acceptable for certain roles). This transfer module then only selects words which have an intensional meaning which equates with that of the French word (or which is a member of of that class of intensional meaning).

*/

```
transfer(French_structure, English_structure) :-
    interim_trace_on,
    replace_material(French_structure, English_structure), !,
    nl,write('RESULTS OF LEXICAL TRANSFER'),nl(2),
    write_tree(English_structure, 0),
    nl, ask_question('Type any letter and a return to continue.', _), !.

transfer(French_structure, English_structure) :-
    replace_material(French_structure, English_structure), !,
    write('Transfer completed'), nl.



replace_material(lu(Format, Desc_French), lu(Format, English_word)) :-
    swap(Desc_French, English_word), !.

replace_material([],[]) :- !.
replace_material(nil, nil) :- !.
replace_material(p - (P), p-(P)) :- !.
replace_material(number(A,B), number(A, B)) :- !.
replace_material(postal_code -PC, postal_code -PC).
replace_material(symb -S, symb -S).
replace_material(postal_code -PC, postal_code -PC).
```

/*
An event requires a special call to "swap" as its siblings must be used in choosing

an English equivalent with an appropriate frame and since the prepositions provide in
the frame can be transferred to the appropriate siblings.
Events are always preceded by modalities and followed by their siblings in the input lists.

TRANSFER IS ONLY REPLACEMENT OF TEXTUAL MATERIAL.
FORMATTING, ADDING OF AUXILIARIES, ADDING OF INFLEXIONS, ETC ALL BELONG TO
THE GENERATION MODULE.
*/

```
replace_material([modalities-M, event -lu(F, Verb_f)| Siblings_f],
    [modalities-M, frame-Frame, event -lu(F, Verb_e) | Siblings_e]) :-
        swap(M, Frame, Siblings_f, Verb_f, Verb_e), !,
        replace_material(Siblings_f, Siblings_e).

replace_material((A,B),(AA,BB)) :-
    replace_material(A,AA),
    replace_material(B,BB) , !.

replace_material(Node-Branch, Node-Replaced_Branch) :-
    replace_material(Branch, Replaced_Branch) , !.

replace_material([Node-Tree|Tail],[Node-Replaced_tree | Replaced_Tail]) :-
    replace_material(Tree, Replaced_tree),
    replace_material(Tail, Replaced_Tail) , !.

replace_material([Tree|Tail],[Replaced_tree | Replaced_Tail]) :-
    replace_material(Tree, Replaced_tree),
    replace_material(Tail, Replaced_Tail), !.
```

/*
French words are replaced by a list of possible English equivalents.
Textual replacement depends on:
a) an entry in the transfer dictionary (where the f_e (French to English) predicate is defined);
b) equivalence between French and English descriptions of words; it may be that there is a
one-to-one replacement defined in the transfer dictionary (e.g. remerciement - thanks)
but the one word may have several possible descriptions in the English dictionary (e.g.,
thanks - verb, noun). Equivalence in this program requires syntactic and semantic similarity.
However, some semantic similarity is assumed to have been established in the transfer dictionary.
Even so, a semantic check is in order. For example, the entry in the transfer dictionary for
"adresser" is f_e(adresser, [send,address]). Yet the definition of "adresser" provides
for two definitions of "adresser" (transfer and mental interaction) each of which has
its own special case frame. The case processor will have selected which frame was to be
used for "adresser" in a given sentence and passed on the corresponding meaning.
This meaning will be used to select from among the possible English equivalents for
"adresser." All equivalents that have the same meaning (or are instances of that
class of meaning according to the ontological schema in Appendix 2) will reduce the selection
of possibilities. For verbs, this list will be further reduced by checking that the English
frame is applicable to the case roles provided. It is not assumed that verbs with the
same intensional meanings will also have the same case frame across languages. For example,
both "send" and "address" may have an intensional meaning of "transfer" yet their
frames could differ (e.g., one may or may not require a destination case role). The appropriate
one is selected based on its match with the case roles found by the case processor.

The meaning(s) of words are dropped in the new tree.
Otherwise, the tree remains the same with its various labels/nodes.
There will therefore be traces of French syntax (e.g., pre- and post-noun positioning of
adjectives) and of French words (i.e., pronouns). In general, though, the output of transfer
is "in English."

```
*/

swap(Ms, Frame, Siblings, verb-(LU_f, TMPN, Meaning_f), verb-(LU_e, TMPN, Meaning_e)) :-
    f_e(LU_f, LU_e_possibilities),
    !,
    check_meaning_frame(Ms, Frame, Siblings, Meaning_f, LU_e_possibilities, LU_e, Meaning_e),
    check(LU_e, LU_f).

swap(Ms, Frame, Siblings, verb-(LU_f, TMPN, Meaning), verb-V) :-
    learner_on, !,
    create_f_e(LU_f),
    swap(Ms, Frame, Siblings, verb-(LU_f, TMPN, Meaning), verb-V).

/*
See comments in case processor with regard to altering frames for infinitives and passives.
The first verb equivalent that meets the criteria is the one that is chosen. The order of
equivalents given in the transfer dictionary will therefore play a part in verb selection.

When the frame of an English event is matched with the results of the case analysis of the
French input, a check is made to ensure that all required cases of the English event will be
filled.
*/

check_meaning_frame(Ms, Altered_frame, Siblings, Meaning_f, [First | Rest], First, Meaning_e)
:-
    eng_analysis(First, Desc),
    member(verb-(Verb, TMPN, M_frame_combos), Desc),
    mem(Meaning_e-Frame, M_frame_combos),
    isa(Meaning_e, Meaning_f),
    alter_frame(Ms, Frame, Altered_frame),
    subset(Altered_frame, Siblings), !.
check_meaning_frame(Ms, Frame, Siblings, Meaning_f, [_ | Rest], First, Meaning) :-
    check_meaning_frame(Ms, Frame, Siblings, Meaning_f, Rest, First, Meaning).

/*
PREPOSITIONS
Equivalents for French prepositions do not depend on equivalence but are a function of
the event, the case role, and the noun phrase filling the case role slot. At this point, therefore,
equivalents are determined immediately. Note that many
equivalents for prepositions in the transfer dictionary are simply the "functional_prep" string.
In these cases, the English preposition will be a function of the relationship between the event
and the role player (a zero preposition is assumed to be an element of the set of English
prepositions for the sake of homogeneity in later sentence generation).
*/

swap(prep - X, prep-Prep) :-
    f_e(X, Prep), !.
swap(prep - X, Prep) :-
    learner_on,
    create_f_e(X), !,
    swap(prep - X, Prep).


/*
PRONOUNS
Future developments may require the type/person/number information
about pronouns if an antecedent is to be used in place of a pronoun for the sake of style and
clarity. In that way, the correct antecedent could then be selected.
```

The first rule handles "pronouns" contained in ellipsed propositions for which person, number and meaning information alone have been determined through parsing and case analysis.
*/

```
swap(pro-(Pro, Type, 1, s, Gender, Meaning), pro-('I', pers, 1, s)) :-
    var(Pro).
swap(pro-(Pro, Type, 2, N, Gender, Meaning), pro-(you, pers, 2, N)) :-
    var(Pro).
swap(pro-(Pro, Type, 3, s, m, Meaning), pro-(he, pers, 3, s)) :-
    var(Pro).
swap(pro-(Pro, Type, 3, s, f, Meaning), pro-(she, pers, 3, s)) :-
    var(Pro).
swap(pro-(Pro, Type, 1, pl, Gender, Meaning), pro-(we, pers, 1, pl)) :-
    var(Pro).
swap(pro-(Pro, Type, 3, pl, Gender, Meaning), pro-(they, pers, 3, pl)) :-
    var(Pro).

swap(pro-(Pro, Type, Person, Number, _), Pronoun) :-
    f_e(Pro, Possibilities), !,
    pick_one_possibility(pro, Type, Person, Number, Possibilities, Pronoun),
    (Pronoun \= nil ;
    nl, write('No equivalent for "'), write(Pro), write('" found from "'),
    write_list(Possibilities), write('."
Check that the meanings provided in the dictionaries are compatible.'), resetio, nl, quit).

swap(pro-(Pro, Type, Person, Number, Gender, Meaning), Pronoun) :-
    learner_on,
    create_f_e(Pro), !,
    swap(pro-(Pro, Type, Person, Number, Gender, Meaning), Pronoun).

pick_one_possibility(pro, Type, P1, N1, [One | Others], pro-(Pro, Type, Pers, Num)) :-
    eng_analysis(One, Desc),
    member(pro-(Pro, Type, P2, N2, _, _), Desc),
    match(P1, P2, Pers),
    match(N1, N2, Num),
    !.
pick_one_possibility(pro, Type, P1, N1, [_ | Others], Pronoun) :-
    pick_one_possibility(pro, Type, P1, N1, Others, Pronoun).
pick_one_possibility(pro, _, _, _, [], nil).
```

/*
ADJECTIVES
A problem in replacing textual material is that of transpositions (replacing nouns with adjectives, nouns with verbs, etc.). At this point, I rely heavily on the similarity of French and English syntax, as adjectives must be replaced with adjectives (and possibly verbal adjectives).

For the moment I will assume that any noun given as an equivalent for an adjective in the transfer dictionary can replace that adjective. There should be some way of controlling this transfer, but for now I will assume that equivalence exists by virtue of the f_e entry.
*/

```
swap(adj-(X, Type, Gender, Number, Meaning), Adj) :-
    f_e(X, Possibilities), !,
    pick_one_possibility(adj, Type, Meaning, Possibilities, Adj),
    (Adj \= nil ;
    nl, write('No equivalent for "'), write(X), write('" found from "'),
    write_list(Possibilities), write('."
Check that the meanings provided in the dictionaries are compatible.'), resetio, nl, quit).
```

```
swap(adj-(X, Type, Gender, Number, Meaning), Adj) :-
    learner_on,
    create_f_e(X), !,
    swap(adj-(X, Type, Gender, Number, Meaning), Adj).

/*
The conjugate predicate used here to create an adjective from a verb is defined in the
"Generation" file. It is assumed that English verbal adjectives are morphologically identical
to past participles, here conjugated as "hist(oric)". It is expected that the conjugate procedures
will become more complex to take into account doubling of consonants and so forth. It is
better then to call the conjugate procedure than to repeat its development here for only
a single instance of conjugating.
*/
pick_one_possibility(adj, verbal-desc, M_f, [One | Others], adj-(Verb_ed, desc)) :-
    eng_analysis(One, Desc),
    member(verb-(Verb, _), Desc),
    conjugate(Verb,(hist, _), [Verb_ed]),
    !.

pick_one_possibility(adj, Type, M_f, [One | Others], adj-(Adj, Type)) :-
    eng_analysis(One, Desc),
    member(adj-(Adj, Type, G, N_e, M_e), Desc),
    intersect(M_e, M_f),
    !.
pick_one_possibility(adj, Type, Meaning, [_ | Others], Adj) :-
    pick_one_possibility(adj, Type, Meaning, Others, Adj).
pick_one_possibility(adj, _, _, [], nil).


/*
NOUNS
    On the assumption that an English noun is to replace a French noun,
        the gender of the French noun may be irrelevent here, unless a she/he replaces a name;
        the number of the French noun affects the number of the English noun in some cases;
        the common/proper nature of the French noun is ignored now but may be useful later;
        the count/mass nature of the French noun is ignored now but may be useful later;
        the meaning of the English noun must be an instance of the meaning of the French
            noun according to the ontological tree.
*/

swap(noun-(X, _, Number, _, _, Meaning), Noun) :-
    f_e(X, Possibilities), !,
    pick_one_possibility(noun, Number, Meaning, Possibilities, Noun),
    (Noun \= nil ;
    nl, write('No equivalent for "'), write(X), write('" found from "'),
    write_list(Possibilities), write('."
Check that the meanings provided in the dictionaries are compatible.'), resetio, nl, quit).

swap(noun-(X, Info), Noun) :-
    learner_on,
    create_f_e(X), !,
    swap(noun-(X, Info), Noun).


pick_one_possibility(noun, N_f, M_f, [One | Others], noun-(Noun, Num, CP, CM)) :-
    eng_analysis(One, Desc_e),
    member(noun-(Noun, _, N_e, CP, CM, M_e), Desc_e),
    intersect(M_e, M_f),
    change_number(N_f, N_e, Num),
```

```
    !.
pick_one_possibility(noun, Number, Meaning, [_ | Others], Noun) :-
    pick_one_possibility(noun, Number, Meaning, Others, Noun).
pick_one_possibility(noun, _, _, [], nil).
```

/*
The first argument of the change_number predicate indicates the number of the
French lexical unit in its surface form. The second argument indicates the number
of the equivalent provided in the transfer dictionary (and determined by eng_analysis;
see the English Morphology file).
If the French LU is singular and the equivalent is given as singular, there is no change.
e.g.
(lettre, s), (letter, s), (letter, s)

If the French LU is singular and the equivalent is given as plural, the equivalent will be plural.
e.g.
(documentation, s), (papers, pl), (paper, pl)

If the French LU is plural and the equivalent of its singular form is singular, then
the equivalent will become plural.
e.g.
(lettre, pl), (letter, s), (letter, pl)

If the French LU is plural and has no singular form, and the equivalent of the plural form
is singular, then the equivalent will be singular. This will not work at the present since
the only way knowing that a French LU has no singular form is by making another lookup in
the French dictionary or by carrying information from the analysis routine to the effect
that a given LU did not require morphological analyis for recognition (i.e. it was in the
dictionary as is)
e.g.
(arrhes, pl), (deposit, s), (deposit,s)

If the French LU is plural and has no singular form, and the equivalent of the plural form
is plural, then the equivalent will be plural.
e.g.
(dépens, pl), (costs, pl), (cost, pl)
*/

```
change_number(s, s, s).
change_number(s, pl, pl).
% change_number(pl, s, s) :- see above.
change_number(pl, s, pl).
change_number(pl, pl, pl).
change_number(s, (s,pl), s).
change_number(pl, (s,pl), pl).
change_number((s,pl), s, s).
change_number((s,pl), pl, pl).
```

/*
This check put in to facilitate program development.
*/

```
check(Symbol, _) :-
    symbol(Symbol), !.
check(_, X) :-
    nl, write('NO EQUIVALENTS FOUND FOR:  '), write(X),
    resetio, nl, quit.
```

```
/*
ARTICLES
The dictionary is consulted for articles even though there are only a few of them
[le,la,les,le_la,un,une] and they could be directly swapped here. I am trying to make
the processing "spirit" here not French-English dependent.
Also, it is not expected that the f_e call will fail, yet the second swap(article) rule
is provided in the event that the appropriate entries in the dictionary are deleted and need
to be recreated.
*/

swap(art-(French, Type , Gender, Number), Choice) :-
    f_e(French, Possibilities), !,
    pick_one_possibility(art, Type, Number, Possibilities, Choice),
    (Choice \= nil ;
    nl, write('No equivalent for "'), write(French), write('" found from "'),
    write_list(Possibilities), write('."
Check that the meanings provided in the dictionaries are compatible.'), resetio, nl, quit).

swap(art-(Article, Type, Gender, Number), Choice) :-
    learner_on,
    create_f_e(Article), !,
    swap(art-(Article, Type, Gender, Number), Choice).

pick_one_possibility(art, Type, N1, [One | Others], art-(Art, Type, nil, Number)) :-
    eng_analysis(One, Desc),
    member(art-(Art, Type, nil, N2), Desc),
    match(N1, N2, Number),
    !.
pick_one_possibility(art, Type, Number, [_|Others], Art) :-
    pick_one_possibility(art, Type, Number, Others, Art).
pick_one_possibility(art, _, _, [], nil).


/*
These rules for conjunctions.
*/

swap(conj - (X, Type), Conjunction) :-
    f_e(X, Possibilities), !,
    pick_one_possibility(conj, Type, Possibilities, Conjunction),
    (Conjunction \= nil ;
    nl, write('No equivalent for "'), write(X), write('" found from "'),
    write_list(Possibilities), write('."
Check that the meanings provided in the dictionaries are compatible.'), resetio, nl, quit).


swap(conj - (X,Type), Conj) :-
    learner_on,
    create_f_e(X), !,
    swap(conj - (X,Type), Conj).

pick_one_possibility(conj, Type, [One | Others], conj-(Conjunction, Type)) :-
    eng_analysis(One, Desc),
    member(conj-(Conjunction, Type), Desc),
    !.
pick_one_possibility(conj, Type, [_|Others], Conjunction) :-
    pick_one_possibility(conj, Type, Others, Conjunction).
pick_one_possibility(conj, _, [], nil).
```

```
/*
These rules for adverbs.
English adverbs that are to be generated from an adjective are marked as such.
*/

swap(adv - (X, Type), Adverb) :-
    f_e(X, Possibilities), !,
    pick_one_possibility(adv, Type, Possibilities, Adverb),
    (Adverb \= nil ;
    nl, write('No equivalent for "'), write(X), write('" found from "'),
    write_list(Possibilities), write('."
Check that the meanings provided in the dictionaries are compatible.'), resetio, nl, quit).

swap(adv - (X,Type), Adv) :-
    learner_on,
    create_f_e(X), !,
    swap(adv - (X,Type), Adv).

pick_one_possibility(adv, Type, [One | Others], adv-(Adverb, Type)) :-
    eng_analysis(One, Desc),
    member(adv-(Adverb, Type), Desc),
    !.
pick_one_possibility(adv, Type, [One | Others], adv-(Adverb, (Type, add_ly))) :-
    eng_analysis(One, Desc),
    member(adj-(Adverb, _), Desc),
    !.
pick_one_possibility(adv, Type, [_|Others], Adverb) :-
    pick_one_possibility(adv, Type, Others, Adverb).
pick_one_possibility(adv, _, [], nil).

/*
Strings
*/

swap(string - S, string-S).


/*
Interjections
*/

swap(interjection - X, Interjection) :-
    f_e(X, Possibilities), !,
    pick_one_possibility(interjection, Possibilities, Interjection),
    (Interjection \= nil ;
    nl, write('No equivalent for "'), write(X), write('" found from "'),
    write_list(Possibilities), write('."
Check that the meanings provided in the dictionaries are compatible.'), resetio, nl, quit).

swap(interjection - X, Interjection) :-
    learner_on,
    create_f_e(X), !,
    swap(interjection - X, Interjection).

pick_one_possibility(interjection, [One | Possibilities], One).
pick_one_possibility(interjection, [_ | Possibilities], One) :-
    pick_one_possibility(interjection, Possibilities, One).
pick_one_possibility(interjection, [], nil).
```

```
/*
In the event a word is unknown and the learner module has not be loaded into memory,
the translation operation is aborted. There is no sense in dropping the unknown input and
working with the rest of the sentence when the learner module can be loaded and the
sentence re-entered.

If equivalents are suggested in the transfer dictionary and
if all the suggested equivalents are defined in the English dictionary and
if none of the suggestions satisfy the constraints
(likely due to changes in the dictionary after entries have been made),
the operation is aborted through the definitions of the individual "swap" predicates above.
*/

swap(_-(X,_), _) :-
    nl,
    write('As equivalents for "'), write(X), write('" are unknown, and since the learner
module has not
been loaded, translation of this sentence is dropped.  You may load the learner module, call
translate, and re-enter the sentence.'),
    nl, resetio, quit.


/*
The subset predicate requires that the set of roles associated with the English event be
a subset of the roles in the French sentence.  All cases required by the English event must
be filled by something that may or may not be required by the equivalent French event.
For example, an English event may require an affected while the equivalent French event
considers it as only optional.  Cases considered optional for an English event need not be filled.
Extra role fillers found by the case analysis, which are not called for by the English event,
either as required or optional cases, are simply transfered.
*/


subset([], _) :- !.
subset([Case-(r, _, _) | Frame], Siblings) :-
    member(Case-_, Siblings),
    subset(Frame, Siblings).
subset([Case-(o, M, Prep) | Frame], Siblings) :-
    subset(Frame, Siblings).
```

```
/*************************************************************************
*************************************************************************
MORPHOLOGICAL ANALYSIS OF ENGLISH WORDS
*************************************************************************
*************************************************************************/


/*
See the "Morphology Analyzer" file for comments on morphological analysis in general.

Here, English words are analyzed as the transfer procedure needs information about
words given as equivalents for French words.  It is not always the case that suggested equivalents
can be found directly in the dictionary, even though their dictionary form is entered there.
For example, according to the transfer dictionary
    ?- f_e(documentation, X).
      X = [documentation,documents,papers]
but the English unilingual dictionary would contain only "documentation," "document," and
"paper" in the singular.
*/

/*
Direct Lookup
*/

eng_analysis(Lex_unit,Dict_descript) :-
    e(Lex_unit,Dict_descript), !.

/*
Treat the word as a plural noun.
*/

eng_analysis(Lex_unit, Standard_Desc) :-
    get_singular(Lex_unit, Dict_form),
    e(Dict_form, Desc),
    eng_standardize(Lex_unit, Desc, Standard_Desc), !.

/*
Treat the word as a conjugated verb.
*/

eng_analysis(Lex_unit, [verb-(Verb, TMPN, M_Fs)]) :-
    get_eng_infinitive(Lex_unit, Verb, TMPN),
    e(Verb, Desc),
    member(verb-(Verb,_,M_Fs), Desc), !.

/*
A routine will be added to determine adverbs derived from adjectives.
The next rules deal with unrecognizable input.
*/

eng_analysis(Lex_unit, Desc) :-
    explode(Question, ['Is "', Lex_unit, '" misspelled?']),
    ask_question(Question,[yes,y,no,n], Ans),
    (Ans = yes ; Ans = y),
    ask_question('Please give the correct spelling.', Correct),
    eng_analysis(Correct, Desc), !.

eng_analysis(Lex_unit, Description) :-
```

```
    learner_on,
    discover(Lex_unit),
    eng_analysis(Lex_unit, Description), !.

eng_analysis(Lex_unit, _) :-
    nl,
    write('As "'), write(Lex_unit), write('" is unknown, and since the learner module has not
been loaded, translation of this sentence is dropped.  You may load the learner module, call
translate. and re-enter the sentence.'),
    nl, resetio, quit.
```

/*
Standardization of descriptions is required if new
grammatical-morphological information has been found through the analysis routines.
*/

```
eng_standardize(_, [], []).

eng_standardize(LU, [noun-(Noun, G, N, CP, CM, M) | Rest],
    [noun-(Noun, G, pl, CP, CM, M) | List]) :-
        eng_standardize(LU, Rest, List).

eng_standardize(LU, [verb-Verb_in | Rest], [verb-Verb_out | List]) :-
    change_eng_v(LU, Verb_in, Verb_out),
    eng_standardize(LU, Rest, List).

eng_standardize(LU, [_ | Rest], List) :-
        eng_standardize(LU, Rest, List).

change_eng_v(Lex_unit,(Dict_form, _, MF),(Dict_form, TMPN, MF)) :-
    get_eng_infinitive(Lex_unit, Dict_form, TMPN).
```

/*
The following rule takes a given English word, checks it for usual plural endings,
using the following database and returns the singular form. If the word is already
singular, the rule fails.
*/

```
get_singular(Word_pl, Word_S) :-
    explode(Word_pl, Word_pl_List),
    mem(X, [1, 4, 3, 2]),
    get_last_X_chars(Word_pl_List, Last_X, X),
    eng_pl_to_sing_endings(Last_X, Sing_ending),
    app(Root_List, Last_X, Word_pl_List),
    app(Root_List, Sing_ending, Word_S_List),
    explode(Word_S, Word_S_List).
```

/*
The following database provides definitions for relationships between common endings.
For each fact in this short database, the first argument refers to a plural
ending, and the second argument to the corresponding singular ending.

*/

```
eng_pl_to_sing_endings([s], []).
eng_pl_to_sing_endings([e,s], []).
eng_pl_to_sing_endings([i,e,s], [y]).
```

```
eng_pl_to_sing_endings([s,e,s], [sis]).
eng_pl_to_sing_endings([n,a], [non]).
eng_pl_to_sing_endings([a], [um]).
eng_pl_to_sing_endings([i,c,e,s], [ix]).
eng_pl_to_sing_endings([i,c,e,s], [ex]).
eng_pl_to_sing_endings([i], [us]).
eng_pl_to_sing_endings([v,e,s], [f]).
```

/*
A procedure is required to find the root of a word assumed to be a verb.
The procedure checks the last 5 letters of a word with the following database of verb endings, then 4 letters, 3 letters, etc., until a root can be determined.
The appropriate infinitive ending is then added.
Note that this procedure is a little different than that for French verbs since regular verb endings in English can be associated immediately with regular infinitive endings (e.g., '-ons' in French does not indicate whether a verb ends in -ir or -er).
*/

```
get_eng_infinitive(Lex_unit, Dict_form, TMPN) :-
    explode(Lex_unit, List),
    mem(X, [5,4,3,2,1]),
    get_last_X_chars(List, Last_X, X),
    eng_vb_ending(Last_X, TMPN, Inf_ending),
    app(Root_list, Last_X, List),
    app(Root_list, Inf_ending, Inf_list),
    explode(Dict_form, Inf_list).
```

/*
The following short database reflects the relationships between regular verb endings, tense/mood/person/number, and infinitive endings.
*/

```
eng_vb_ending([s], (pres,ind,3,s), []).
eng_vb_ending([e,s], (pres,ind,3,s), []).

eng_vb_ending([o,v,e], (past, ind, Any_p, Any_n), [i,v,e]).
eng_vb_ending([a,m,e], (past, ind, Any_p, Any_n), [o,m,e]).
eng_vb_ending([a,n], (past, ind, Any_p, Any_n), [i,n]).
eng_vb_ending([o,u,g,h,t], (past, ind, Any_p, Any_n), [i,n,g]).
eng_vb_ending([o,u,g,h,t], (past, ind, Any_p, Any_n), [u,y]).

eng_vb_ending([e,d], (past, ind, Any_p, Any_n), []).
eng_vb_ending([e,d], (past, ind, Any_p, Any_n), [e]).
eng_vb_ending([e,n], (past, ind, Any_p, Any_n), []).
/* etc. */
```

```
/*************************************************************************
*************************************************************************
GENERATOR
© 1987, 1988 Victor Loewen
*************************************************************************
*************************************************************************/

/*
Input to the generator is a tree with nodes describing proposition types, modality and
proposition components (frame, event, siblings with case labels), and sibling components
(syntactic labels whose terminals are English words and their descriptions).
Generation involves considering the frame and choosing Subject, Direct Object, etc.
from among the siblings. Inflexions are added if necessary.
It then involves considering the modalities which affect the insertion of auxiliaries,
negatives, final punctuation and inversion of subject-verb in the case of interrogatives.
Verbs are also checked for correct endings.
*/


generation(proposition-Prop, Sentence_string) :-
    generate_prop(Word_list, Punc, Prop),
    list_to_string(Word_list, Punc, Sentence_string).

generation(mention-M, Sentence_string) :-
    generate_mention(Word_list, M),
    list_to_string(Word_list, nil, Sentence_string).


/*
Propositions may be comprised of independent, subordinate or ellipsed propositions, or
a combination of the three. Only an independent proposition returns information concerning
the final punctuation.
*/

generate_prop(Words, P, independent_prop-[modalities-M, frame-F, event-E | Siblings]) :-
    generate_independent(Words, M, P, F, E, Siblings).

generate_prop(Words, _, subordinate_prop-(event_prop-EP)) :-
    generate_event_prop(_, nil, EP, Words).
generate_prop(Words, _, subordinate_prop-(prep_phrase-PP)) :-
    generate_prep_phrase(_, nil, PP, Words).
generate_prop(Words, _, ellipsed_prop-[modalities-M, frame-F, event-E | Siblings]) :-
    generate_independent(Words, M, _, F, E, Siblings).

generate_prop(Words, P, (A, articulation-C, B)) :-
    generate_prop(A_words, P, A),
    generate_articulation(C, C_words),
    generate_prop(B_words, P, B),
    append(A_words, C_words, L1),
    append(L1, B_words, Words).

generate_prop(Words, P, (A, B)) :-
    generate_prop(A_words, P, A),
    generate_prop(B_words, P, B),
    append(A_words, B_words, Words).


/*
```

INDEPENDENT PROPOSITIONS
Note that verb generation is part of arrangement, as person/number information for the verb is derived from the final subject.
*/

```
generate_independent(Word_list, (T,M,V,A,F), F, Frame, Event, Siblings) :-
    generate_siblings(Frame, Siblings, Sibling_words),
    arrangement((T,M,V,A,F), Event, Sibling_words, Word_list).
```

/*
EVENT PROPOSITIONS
*/

```
generate_event_prop(Case, Prep_suggested, event_prop -[modalities-M,
        frame -F, event -E, Prep | Siblings], Word_list) :-
    generate_prep(_, to, Prep, Prep_words),
    generate_siblings(F, Siblings, Sibling_list),
    arrange_event_prop(M, E, Prep_words, Sibling_list, Word_list).

generate_event_prop(Case, Prep_suggested, event_prop -[modalities-M,
        frame -F, event -E | Siblings], Word_list) :-
    generate_siblings(F, Siblings, Sibling_list),
    arrange_event_prop(M, E, [Prep_suggested], Sibling_list, Word_list).

generate_event_prop(Case, Prep, (event_prop-EP, articulation-A, Rest), Words) :-
    generate_event_prop(Case, Prep, EP, EP_words),
    generate_articulation(A, A_words),
    generate_event_prop(Case, Prep, Rest, Rest_words),
    append(EP_words, A_words, L1),
    append(L1, Rest_words, Words).
```

/* QUALIFYING EVENT PROPOSITIONS */

```
generate_qualifying_event_prop(Case, Prep_suggested, qualifying_event_prop -
        [modalities-M, frame -F, event -E, Prep | Siblings], Word_list) :-
    generate_prep(_, to, Prep, Prep_words),
    generate_siblings(F, Siblings, Sibling_list),
    arrange_event_prop(M, E, Prep_words, Sibling_list, Word_list).

generate_qualifying_event_prop(Case, Prep, (qualifying_event_prop-EP, articulation-A,
        Rest), Words) :-
    generate_qualifying_event_prop(Case, Prep, EP, EP_words),
    generate_articulation(A, A_words),
    generate_qualifying_event_prop(Case, Prep, Rest, Rest_words),
    append(EP_words, A_words, L1),
    append(L1, Rest_words, Words).
```

/*
RELATIVE PROPOSITIONS
Relative props begin with "qui" or "que" in the French. What the generator receives is "that" as agent of an action verb or as affected of a stative verb OR "that" as whatever and always put at the end of the list. The "que" is kept by the parser since the case processor needs it for some cases. For example, in "la lettre que nous avons est..." the verb "avoir" requires both an agent and an affected if processing is to succeed.
*/

```
generate_rel_prop(Case, Prep_suggested, relative_prop -[modalities-M,
        frame -F, event -E | Siblings], Word_list) :-
    generate_siblings(F, Siblings, Sibling_list),
    arrange_rel_prop(M, E, Sibling_list, Word_list).

generate_rel_prop(Case, Prep, (relative_prop-RP, articulation-A, Rest), Words) :-
    generate_rel_prop(Case, Prep, RP, RP),
    generate_articulation(A, A_words),
    generate_rel_prop(Case, Prep, Rest, Rest_words),
    append(RP_words, A_words, L1),
    append(L1, Rest_words, Words).
```

/*
A routine for formatting common or proper nouns and for acronyms.
The format of French lexical units given as input is ignored as
capitalization in English sentences occurs only at the beginning of a sentence and at the
beginning of proper nouns. Exceptions are acronyms (marked herein as all_caps) and
code formats which may be any mixture of upper and lower case.
*/

```
format(X, Gender, Number, Comm_prop, Count_mass, X2) :-
    pluralize(X, Number, Count_mass, X1),
    format(Comm_prop, X1, X2).

format(proper, Word_in, Word_out) :-
    capitalize(Word_in, Word_out), !.
format(cap, Word_in, Word_out) :-
    capitalize(Word_in, Word_out), !.
format(all_caps, In, Out) :-
    explode(In, List),
    cap_list(List, Caps),
    explode(Out, Caps), !.
format(low, X, X) :- !.
format(common, X, X) :- !.
format(Mix, In, Out) :-
    explode(Mix, Format_list),
    explode(In, In_list),
    cap_mix(Format_list, In_list, Out_list),
    explode(Out, Out_list).

capitalize(Word_in, Word_out) :-
    explode(Word_in, [First_char | Rest]),
    (letter(First_char, Upper) ;
        letter(_, First_char), Upper = First_char ;
            digit(First_char), Upper = First_char ;
                punc(_, First_char), Upper = First_char),
    explode(Word_out, [Upper | Rest]).

cap_list([], []).
cap_list([H | T], [Cap | L]) :-
    letter(H, Cap),
    cap_list(T, L).
cap_list([H | T], [H | L]) :-
    cap_list(T, L).

cap_mix([], X, X) :- !.
cap_mix([u | Mix], [Letter | Letters], [Changed | Changes]) :-
    letter(Letter, Changed),
```

```
    cap_mix(Mix, Letters, Changes).
cap_mix([1 | Mix], [Letter | Letters], [Letter | Changes]) :-
    cap_mix(Mix, Letters, Changes).

cap_words([], []) :- !.
cap_words([First_word | Words], [Capped_word | L]) :-
    capitalize(First_word, Capped_word),
    cap_words(Words, L).
```

/*
MAKING A READABLE STRING FROM A LIST OF WORDS
Some post editing is also done:
    adding spaces
    adding the appropriate punctuation at the end of the sentence
*/

```
list_to_string(LU_list, Punc, String) :-
    post_edit(LU_list, Punc, [' ', First_word | Spaced_lu_list]),
    capitalize(First_word, Capped),
    explode(String, [Capped | Spaced_lu_list]).

post_edit([], declarative, ['.']).
post_edit([], interrogative, ['?']).
post_edit([], imperative, ['!']).
post_edit([], nil, []).
post_edit(['.' | Tail], P, ['.' | L1]) :-
    post_edit(Tail, P, L1).
post_edit([',' | Tail], P, [',' | L1]) :-
    post_edit(Tail, P, L1).
post_edit([Word | Tail], P, [' ', Word | L1]) :-
    post_edit(Tail, P, L1).
```

/*
A routine for changing "a" to "an" before a vowel.
The correct treatment of "h", e.g., an hour/a house, would require information not available in the dictionary. It may be best to create a small database of common aspirated aitches (or non-aspirated ones, depending on which is shorter), and then leave all remaining errors to a human post-editor (or to a good style checker which would work in conjunction with a word processor).
*/

```
an(an, Word) :-
    explode(Word, [First | _]),
    member(First, [a,e,i,o,u,y]), !.
an(a, _).
```

```
/*************************************************************************
*************************************************************************
GENERATOR - VERBS
© 1987, 1988 Victor Loewen
*************************************************************************
*************************************************************************/
```

/*
VERBS

The tenses produced by the parser are listed in the left column. Verb generation follows the rules given on the right for regular verbs. Irregular verbs are generated with the help of a small database.

- pres(ent) --> check person and number agreement
- imp(erfect) --> was + verb_ing
- hist(oric) --> add d, ed, double consonant, change vowel or check a dictionary for irregular ones
- fut(ure) --> will + verb
- past --> verb + ed, or check dictionary for irregular ones
- future_anterior --> will have + verb
- past anterior --> would have + verb
- pluperfect --> had had + verb
- past_future_anterior --> would have had + verb
- (pres, imp) --> assume present takes priority over imperfect, as ambiguity occurs only with subjunctive mood
- (pres, hist) --> assume present takes priority over historic, on the (weak) assumption that the historic tense is used less often in administrative correspondence

*/

/*
Abbreviations:

- TMPN = tense, mood, person, number
- T = tense
- M = mood
- P = person
- N = number
- F = format

The TMPN supplied with the verb is used only if there is no subject (i.e., the surface structure will be imperative).
In all other cases the Tense of the sentence as a whole and the person/number of the subject govern verb generation. If the proposition as a whole is negative, the infinitive is kept and the appropriate auxiliary will be generated during arrangement.
*/

```
generate_verb(T, affirmative, P, N, lu(F,verb -(Verb,(_, _, use_ing, _),
    Meaning)), Meaning, Vl) :-
        conjugate(Verb, (T, P, use_ing), Vl).

generate_verb(T, affirmative, P, N, lu(F,verb -(Verb, _, Meaning)), Meaning, Vl) :-
    conjugate(Verb, (T, P, N), Vl).

generate_verb(T, negative-pas, inf, _, lu(F,verb -(V, _, Meaning)), Meaning, [not, to, V]).
generate_verb(T, negative-pas, P, N, lu(F,verb -(V, _, Meaning)), Meaning, Verb) :-
    conjugate(do, (T, P, N), DO),
```

```
    append(DO, [not, V], Verb).

generate_verb(T, negative-jamais, P, N, lu(F,verb -(V, _, Meaning)), Meaning,
        [never | V1]) :-
    conjugate(V, (T, P, N), V1).

generate_verb(T, negative-plus, P, N, lu(F,verb -(V, _, Meaning)), Meaning,
        [no, longer | V1]) :-
    conjugate(V, (T, P, N), V1).

generate_verb(T, negative-guère, P, N, lu(F,verb -(V, _, Meaning)), Meaning,
        [scarcely | V1]) :-
    conjugate(V, (T, P, N), V1).

generate_verb(T, negative-point, inf, _, lu(F,verb -(V, _, Meaning)), Meaning, [not,to,V]).
generate_verb(T, negative-point, P, N, lu(F,verb -(V, _, Meaning)), Meaning, Verb) :-
    conjugate(do, (T, P, N), DO),
    append(DO, [not, V], Verb).

generate_verb(T, negative - rien, P, N, lu(F, verb-(V, _, Meaning)), Meaning, Verb) :-
    conjugate(V, (T, P, N), V1),
    append(V1, [none], Verb).

generate_verb(T, negative - personne, P, N,  lu(F, verb-(V, _, Meaning)), Meaning, Verb) :-
    conjugate(V, (T, P, N), V1),
    append(V1, [no, one], Verb).

generate_verb(T, negative - aucun, P, N, lu(F, verb-(V, _, Meaning)), Meaning, Verb) :-
    conjugate(V, (T, P, N), V1),
    append(V1, [none, of], Verb).


conjugate(V, (pres, inf, _), [to, V]).

conjugate(be, (pres, 1, s), [am]).
conjugate(be, (pres, 2, s), [are]).
conjugate(be, (pres, 3, s), [is]).
conjugate(be, (pres, _, pl), [are]).
conjugate(be, (imp, 1, s), [was]).
conjugate(be, (imp, 2, s), [were]).
conjugate(be, (imp, 3, s), [was]).
conjugate(be, (imp, _, pl), [were]).
conjugate(be, (past, 1, s), [was]).
conjugate(be, (past, 2, s), [were]).
conjugate(be, (past, 3, s), [was]).
conjugate(be, (past, _, pl), [were]).
conjugate(be, (hist, 1, s), [was]).
conjugate(be, (hist, 2, s), [were]).
conjugate(be, (hist, 3, s), [was]).
conjugate(be, (hist, _, pl), [were]).
conjugate(be, ((pres,imp), 1, s), [am]).
conjugate(be, ((pres,imp), 2, s), [are]).
conjugate(be, ((pres,imp), 3, s), [is]).
conjugate(be, ((pres,imp), _, pl), [are]).

conjugate(have, (pres, 3, s), [has]).
conjugate(have, ((pres,_), 3, s), [has]).
```

/*

Most of the following rules are in pairs. The first one succeeds if the verb is irregular and is listed in the vb_data base below. If the verb is not there, it is treated as a regular verb.

```
*/

conjugate(Verb, (pres, participle, _), [Verb_ing]) :-
    ing_form(Verb, Verb_ing).

conjugate(Verb, (pres, 3, s), [Verb_s]) :-
    s_form(Verb, Verb_s).
conjugate(Verb, ((pres,_), 3, s), [Verb_s]) :-
    s_form(Verb, Verb_s).
conjugate(Verb, (pres,_), [Verb]).

conjugate(Verb, (imp, 2, _) , [were, Verb_ing]) :-
    ing_form(Verb, Verb_ing).
conjugate(Verb, (imp, _) , [was, Verb_ing]) :-
    ing_form(Verb, Verb_ing).

conjugate(Verb, (past, participle, use_ing), [having, PP]) :-
    vb_data(Verb, _, PP).
conjugate(Verb, (past, participle, use_ing), [having, PP]) :-
    d_ed_form(Verb, PP).

conjugate(Verb, (past,_), [Past_form]) :-
    vb_data(Verb,Past_form,_).
conjugate(Verb, (past,_), [Past_form]) :-
    d_ed_form(Verb,Past_form).

conjugate(Verb, (hist, _), [Past_form]) :-
    vb_data(Verb, Past_form, _).
conjugate(Verb, (hist, _), [Past_form]) :-
    d_ed_form(Verb,Past_form).

conjugate(Verb, (fut, _) , [will, Verb]).

conjugate(Verb, (future_anterior, _), [will, have, Verb_ed]) :-
    vb_data(Verb, _, Verb_ed).
conjugate(Verb, (future_anterior, _), [will, have, Verb_ed]) :-
    d_ed_form(Verb, Verb_ed).

conjugate(Verb, (past_future_anterior, _), [will, have, had, Verb_ed]) :-
    vb_data(Verb, _, Verb_ed).
conjugate(Verb, (past_future_anterior, _), [will, have, had, Verb_ed]) :-
    d_ed_form(Verb, Verb_ed).

conjugate(Verb,(pluperfect, _), [had, PP]) :-
    vb_data(Verb, _, PP).
conjugate(Verb,(pluperfect, _), [had, PP]) :-
    d_ed_form(Verb,PP).
```

/* CATCH ALL */

```
conjugate(Verb, Info, Verb) :-
    write('Verb generation needs attention for '), write(Verb), write(' '), write(Info), nl.
```

/*
If a verb is not in this list or otherwise dealt with (see 'be' above), it is assumed regular.
That means it forms both its past tense and its past participle by adding -d or -ed.

The first argument of vb_data is the present infinitive form, the second is the past tense form,
and the third is the past participle form.

```
*/

vb_data(beat, beat, beaten).
vb_data(become, became, become).
vb_data(begin, began, begun).
vb_data(bet, bet, bet).
vb_data(bid, bade, bidden).
vb_data(bite, bit, bitten).
vb_data(blow,blew,blown).
vb_data(break,broke,broken).
vb_data(bring,brought,brought).
vb_data(burst,burst,burst).
vb_data(buy,bought,bought).
vb_data(catch,caught,caught).
vb_data(choose,chose,chosen).
vb_data(come,came,come).
vb_data(cut,cut,cut).
vb_data(dive,dived,dived).
vb_data(do,did,done).
vb_data(draw,drew,drawn).
vb_data(drink,drank,drunk).
vb_data(drive,drove,driven).
vb_data(eat,ate,eaten).
vb_data(fall,fell,fallen).
vb_data(feel,felt,felt).
vb_data(find,found,found).
vb_data(fly,flew,flown).
vb_data(forget,forgot,forgot).
vb_data(forgive,forgave,forgiven).
vb_data(freeze,froze,frozen).
vb_data(get,got,got).
vb_data(give,gave,given).
vb_data(go,went,gone).
vb_data(grow,grew,grown).
vb_data(hang,hung,hung). % meaning 'suspend' is more likely than 'execute'
vb_data(have, had,had).
vb_data(hide,hid,hidden).
vb_data(hit,hit,hit).
vb_data(hurt,hurt,hurt).
vb_data(keep,kept,kept).
vb_data(know,knew,known).
vb_data(lead,led,led).
vb_data(leave,left,left).
vb_data(let,let,let).
vb_data(lose,lost,lost).
vb_data(make,made,made).
vb_data(mean,meant,meant).
vb_data(read,read,read).
vb_data(ride,rode,ridden).
vb_data(ring,rang,rung).
vb_data(rise,rose,risen).
vb_data(run,ran,run).
vb_data(say,said,said).
vb_data(see,saw,seen).
vb_data(send,sent,sent).
vb_data(shake,shook,shaken).
vb_data(shine,shone,shone).
vb_data(sing,sang,sung).
vb_data(sink,sank,sunk).
vb_data(speak,spoke,spoken).
vb_data(spin,spun,spun).
vb_data(spring,sprang,sprung).
```

```
vb_data(stand,stood,stood).
vb_data(steal,stole,stolen).
vb_data(stink,stank,stunk).
vb_data(strike,struck,struck).
vb_data(swear,swore,sworn).
vb_data(swim,swam,swum).
vb_data(swing,swung,swung).
vb_data(take,took,taken).
vb_data(teach,taught,taught).
vb_data(tear,tore,torn).
vb_data(tell,told,told).
vb_data(think,thought,thought).
vb_data(throw,threw,thrown).
vb_data(wear,wore,worn).
vb_data(weave,wove,woven).
vb_data(weep,wept,wept).
vb_data(win,won,won).
vb_data(wind,wound,wound).
vb_data(write,wrote,written).
```

/* Adding -d or -ed to verbs. */

```
d_ed_form(Verb, Verb_ed) :-
    explode(Verb, List),
    app(Silent_e_dropped, [e], List),
    app(Silent_e_dropped, [ed], Ed_added),
    explode(Verb_ed, Ed_added).
```

/* The following rule distinguishes between 'stooped' and 'stopped'. */

```
d_ed_form(Verb, Verb_ed) :-
    explode(Verb, List),
    app(Vowel_consonant_ending_dropped, [X, Vow, Con], List),
    not member(X, [a,e,i,o,u]),
    member(Vow, [a,e,i,o,u]),
    not member(Con, [a,e,i,o,u]),
    app(Vowel_consonant_ending_dropped, [X, Vow, Con, Con, ed], Ed_added),
    explode(Verb_ed, Ed_added).

d_ed_form(Verb, Verb_ed) :-
    explode(Verb_ed, [Verb, ed]).
```

/* Adding -ing to verbs. */

```
ing_form(Verb, Verb_ing) :-
    explode(Verb, List),
    app(Silent_e_dropped, [e], List),
    app(Silent_e_dropped, [ing], Ing_added),
    explode(Verb_ing, Ing_added).
```

/* The following rule distinguishes between 'stooping' and 'stopping'. */

```
ing_form(Verb, Verb_ing) :-
    explode(Verb, List),
    app(Vowel_consonant_ending_dropped, [X, Vow, Con], List),
    not member(X, [a,e,i,o,u]),
    member(Vow, [a,e,i,o,u]),
    not member(Con, [a,e,i,o,u]),
    app(Vowel_consonant_ending_dropped, [X, Vow, Con, Con,ing], Ing_added),
```

```
    explode(Verb_ing, Ing_added).

ing_form(Verb, Verb_ing) :-
    explode(Verb_ing, [Verb, ing]).
```

/* Rules for adding -s to 3rd person singular verbs are the same as those used for adding -s to nouns. The pluralize rule (in the Generator-noun file) is therefore used. */

```
s_form(Verb, Verb_s) :-
    pluralize(Verb, pl, count, Verb_s).
```

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

GENERATOR - NOUNS

© 1987, 1988 Victor Loewen

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


generate_noun_phrase(Case, Prep, np-lu(Format, pro -(W1, _, P, N)), (W2,P,N)) :-
    add_prep(Case, Prep, [W1], W2).

/*
A special rule for all instances of headless noun phrases (e.g., "en + adj/noun").
See case processor for explanation.
The person (3) and number (s) assumption is a bit weak, but it may be needed in
when conjugating a verb. It may be better to get the number from the French adjective.
*/

generate_noun_phrase(Case, Prep, np-(noun - nil, modifiers-Modifiers), (Words, 3,s)) :-
    generate_modifiers(Modifiers, M),
    arrange_modifiers([], M, common, count, Arrangement),
    add_prep(Case, Prep, Arrangement, Words).

generate_noun_phrase(Case, Prep, np -(noun -Noun, modifiers -Modifiers), (Words,3,Num)) :-
    generate_noun(Noun, N, Num, Common_proper, Count_mass),
    generate_modifiers(Modifiers, M),
    arrange_modifiers(N, M, Common_proper, Count_mass, Arrangement),
    add_prep(Case, Prep, Arrangement, Words).

generate_noun_phrase(Case, Prep, name -Name, (Words, 3, Num)) :-
    generate_names(Num, Words, Name).


/*
It is assumed that compound noun phrases becoming a surface subject will require
a verb in the 3 person plural.
*/

generate_noun_phrase(Case, Prep, (NP1, articulation-A, NPs), (Words, 3, pl)) :-
    generate_noun_phrase(Case, Prep, NP1, (NP1_words, _)),
    generate_articulation(A, A_words),
    generate_noun_phrase(Case, Prep, NPs, (NPs_words, _)),
    append(NP1_words, A_words, L1),
    append(L1, NPs_words, Words).

generate_noun_phrase(Case, Prep, (NP1, NPs), (Words, 3, pl)) :-
    generate_noun_phrase(Case, Prep, NP1, (NP1_words, _)),
    generate_noun_phrase(Case, Prep, NPs, (NPs_words, _)),
    append(NP1_words, NPs_words, Words).


/*
NOUNS
The second rule is for composite nouns.
(see definition of composite noun in "Parser - noun phrase" file).
*/

generate_noun(lu(Format, noun-(X, Num, CP, CM)), [Noun], Num, CP, CM) :-
    format(X, _, Num, CP, CM, Noun).
```

```
generate_noun((lu(F1, noun-(N1, Num, CP, CM)), p-(P),
    lu(F2, noun-(N2, Num2, CP2, CM2))), [Noun1, P, Noun2], Num, CP, CM) :-
        format(N1, _, Num, CP, CM, Noun1),
        format(N2, _, Num2, CP2, CM2, Noun2).
```

/* MODIFIERS */

```
generate_modifiers(nil, []).

generate_modifiers(lu(_, pro-(them, rel, 3, pl)),  [([of,them], after_noun_desc)]).

generate_modifiers(number(N, Type), [(N, Type)]).

generate_modifiers((lu(Format, adj-(Adj, Type)), nil), [(Adj, Type)]).

generate_modifiers(lu(Format, adj-(Adj, Type)), [(Adj, Type)]).

generate_modifiers(lu(Format,art-(Art, Type, _)), [(Art, Type)]).

generate_modifiers(qualifying_prep_phrase-QPP, [(Words, after_noun_desc)]) :-
    generate_qualifying_prep_phrase(QPP, Words).

generate_modifiers(relative_prop-RP, [(Words, after_noun_desc)]) :-
    generate_rel_prop(_, nil, RP, Words).

generate_modifiers(qualifying_event_prop - QEP, [(Words, after_noun_desc)]) :-
    generate_qualifying_event_prop(_, nil, QEP, Words).

generate_modifiers(lu-(lu(_,adv-(Adv,_)),lu(low,adj-(Adj,_))), Words) :-
    idiom(Adv, Adj, Words).

generate_modifiers((A, B), Modifiers) :-
    generate_modifiers(A, Mod_a),
    generate_modifiers(B, Mod_b),
    append(Mod_a, Mod_b, Modifiers).
```

/*
Arrangement is based on count/mass, common/proper info for the noun and and on info for the adjectives (determiner, cardinal, descriptive, etc.; similar to French order).

Acceptable sequence of adjectives in English is understood to be as follows:

(indefinite) + (cardinal | def) + (descriptive)* + noun + (qualifying_phrases)*
or
(definite | possessive) + (ordinal) + (cardinal) + (descriptive)* + noun

E.G.
all + three | the + yellow ...
the | my + three + yellow ...
but not
(all + the + three + yellow) ...
*/

```
arrange_modifiers(N, Modifiers, CP, mass, Arrangement) :-
    collect_pre(Modifiers, Pre, Remainder),
    sift_post(Remainder, Post),
    cut_out(the, Pre, PRE),
```

```
    append(PRE, N, L1),
    append(L1, Post, L2),
    post_edit_np(L2, Arrangement).

arrange_modifiers(N, Modifiers, CP, CM, Arrangement) :-
    collect_pre(Modifiers, Pre, Remainder),
    sift_post(Remainder, Post),
    append(Pre, N, L1),
    append(L1, Post, L2),
    post_edit_np(L2,  Arrangement).
```

/*
Some post editing is needed for noun phrases:
    removing person and number descriptions from nouns
    changing "a" to "an" before a vowel
    cleaning up some lists
*/

```
post_edit_np([], []) :- !.
post_edit_np([[H | T] | Tail], Word_list) :-
    post_edit_np([H | T], L1),
    post_edit_np(Tail, L2),
    append(L1, L2, Word_list).
post_edit_np([a, Word | Tail], [an, Word | Tail]) :-
    explode(Word, [First | _]),
    member(First, [a,e,i,o,u,y]).
post_edit_np([(Noun, Number) | Tail], [ Noun | Word_list]) :-
    post_edit_np(Tail, Word_list).
post_edit_np([Word | Tail], [Word | L1]) :-
    post_edit_np(Tail, L1).
```

/* There are no modifiers */

```
collect_pre([], [], []) :- !.
```

/* There is something for slot 1 */

```
collect_pre(Mods, Pre, Remainder) :-
    cut_out((Adj, indef), Mods, Mods1),
    post_indef(Adj, Mods1, Pre, Remainder).

collect_pre(Mods, Pre, Remainder) :-
    cut_out((Adj, def), Mods, Mods1),
    post_def_poss(Adj, Mods1, Pre, Remainder).

collect_pre(Mods, Pre, Remainder) :-
    cut_out((Adj, poss), Mods, Mods1),
    post_def_poss(Adj, Mods1, Pre, Remainder).
```

/* There is nothing for slot 1 */

```
collect_pre(Mods, Pre, Remainder) :-
    post_indef(Mods, Pre, Remainder).

collect_pre(Mods, Pre, Remainder) :-
    post_def_poss(Mods, Pre, Remainder).
```

/* Something in slot 1 and something for slot 2 */

```
post_indef(Indef, Mods, [Indef | Pre], Remainder) :-
    cut_out((Adj, cardinal), Mods, Mods1),
    collect_desc(Adj, Mods1, Pre, Remainder).

post_indef(Indef, Mods, [Indef | Pre], Remainder) :-
    cut_out((Adj, def), Mods, Mods1),
    collect_desc(Adj, Mods1, Pre, Remainder).

post_def_poss(Def_poss, Mods, [Def_poss | Pre], Remainder) :-
    cut_out((Adj, ordinal), Mods, Mods1),
    post_ordinal(Adj, Mods1, Pre, Remainder).
```

/* Something in slot 1 but nothing for slot 2 */

```
post_indef(Adj, Mods, Pre, Remainder) :-
    collect_desc(Adj, Mods, Pre, Remainder).

post_def_poss(Adj, Mods, Pre, Remainder) :-
    post_ordinal(Adj, Mods, Pre, Remainder).
```

/* Nothing for slot 1 and something for slot 2 */

```
post_indef(Mods, Pre, Remainder) :-
    cut_out((Adj, cardinal), Mods, Mods1),
    collect_desc(Adj, Mods1, Pre, Remainder).

post_indef(Mods, Pre, Remainder) :-
    cut_out((Adj, def), Mods, Mods1),
    collect_desc(Adj, Mods1, Pre, Remainder).

post_def_poss(Mods, Pre, Remainder) :-
    cut_out((Adj, ordinal), Mods, Mods1),
    post_ordinal(Adj, Mods1, Pre, Remainder).
```

/* Nothing for slot 1 or 2 */

```
post_indef(Mods, Pre, Remainder) :-
    collect_desc(Mods, Pre, Remainder).

post_def_poss(Mods, Pre, Remainder) :-
    post_ordinal(Mods, Pre, Remainder).
```

/* Something for slot 1/2 and for slot 3 */

```
post_ordinal(Def_poss_ord, Mods, [Def_poss_ord | Pre], Remainder) :-
    cut_out((Adj, cardinal), Mods, Mods1),
    collect_desc(Adj, Mods1, Pre, Remainder).
```

/* Nothing for slot 1/2 but something for slot 3 */

```
post_ordinal(Mods, Pre, Remainder) :-
    cut_out((Adj, cardinal), Mods, Mods1),
    collect_desc(Adj, Mods1, Pre, Remainder).
```

/* Something for slot 1/2 but nothing for slot 3 */

```
post_ordinal(Def_poss_ord, Mods, [Def_poss_ord | Pre], Remainder) :-
    collect_desc(Mods, Pre, Remainder).
```

/* Nothing for slot 1/2 or for slot 3 */

```
post_ordinal(Mods, Pre, Remainder) :-
    collect_desc(Mods, Pre, Remainder).


collect_desc(Adj, Mods, [Adj | Descs], Remainder) :-
    collect_desc(Mods, Descs, Remainder).

collect_desc([], [], []) :- !.
collect_desc([(Adj, desc)| Rest], [Adj | L1], L2) :-
    collect_desc(Rest, L1, L2).
collect_desc([H | T], L1, [H | L2]) :-
    collect_desc(T, L1, L2).

sift_post([], []) :- !.
sift_post([(Adj, after_noun_desc) | T], [Adj | L]) :-
    sift_post(T, L), !.
sift_post([H | T], L) :-
    sift_post(T, L).

cut_out(X,[X|T],T) :-
    !.
cut_out(X,[H|L1],[H|L2]) :-
    cut_out(X,L1,L2).
```

/*
Reference: Quirk & Greenbaum, University Grammar, §4.2-4

check_count_mass(proper, _, s, Adj, desc, Adj).
check_count_mass(proper, _, s, _, _, '').
check_count_mass(proper, _, pl, Adj, _, Adj). % some, a few, the, etc.

check_count_mass(common, count, s, one, cardinal, a).
check_count_mass(common, count, s, Art, def, Art).
check_count_mass(common, count, s, Art, indef, '').
check_count_mass(common, count, pl, Adj, cardinal, Adj).
check_count_mass(common, count, pl, Art, def, Art).
check_count_mass(common, count, pl, Art, indef, Art).

check_count_mass(common, mass, s, one, cardinal, '').
check_count_mass(common, mass, s, Art, def, Art).
check_count_mass(common, mass, s, Art, indef, Art).
check_count_mass(common, mass, pl, Adj, cardinal, Adj).
check_count_mass(common, mass, pl, Art, def, '').
check_count_mass(common, mass, pl, Art, indef, Art).

check_count_mass(common, _, _, Adj, _, Adj).
*/

/*
Nouns ending in y preceded by a consonant form their plurals by changing y to i and adding -es.
Verbs ending in y preceded by a consonant for their third plural singular in the same way.

The plural of proper nouns ending in y is formed by adding -s.
Nouns ending in y preceded by a, e, i, o, or u form their plurals by adding -s only.
Verbs ditto for 3rd person singular.

Nouns ending in a sound that can be smoothly united with -s form their plurals by adding -s. 3rd pers. sing. verbs ditto. Otherwise, add -es; 3rd pers. sing. verbs ditto.

Nouns and verbs ending in -consonant + o (e.g., echo, no, go), take an -es. Those ending in -vowel + o take an -s.
*/

```
pluralize(LU, s, _, LU).
pluralize(LU, pl, count, LU_pl) :-
    explode(LU, End_in_y),
    app(Before_the_y, [X, y], End_in_y),
    not member(X, [a, e, i, o, u]),
    app(Before_the_y, [X,i,e,s], End_in_ies),
    explode(LU_pl, End_in_ies).

pluralize(LU, pl, count, LU_pl) :-
    explode(LU, End_in_o),
    app(Before_the_o, [X, o], End_in_o),
    not member(X, [a, e, i, o, u]),
    app(Before_the_o, [X,o,e,s], End_in_es),
    explode(LU_pl, End_in_es).

pluralize(LU, pl, count, LU_pl) :-
    explode(LU, End_in_sibilant),
    app(Before_the_ss, [s,s], End_in_sibilant),
    explode(LU_pl, [LU, es]).
pluralize(LU, pl, count, LU_pl) :-
    explode(LU, End_in_sibilant),
    app(Before_the_ss, [c,h], End_in_sibilant),
    explode(LU_pl, [LU, es]).
pluralize(LU, pl, count, LU_pl) :-
    explode(LU, End_in_sibilant),
    app(Before_the_ss, [s,h], End_in_sibilant),
    explode(LU_pl, [LU, es]).
pluralize(LU, pl, count, LU_pl) :-
    explode(LU, End_in_sibilant),
    app(Before_the_ss, [Vow,x], End_in_sibilant),
    member(Vow, [a,e,i,o,u]),
    explode(LU_pl, [LU, es]).

pluralize(LU, pl, count, LU_pl) :-
    explode(LU_pl, [LU, s]).
pluralize(LU, pl, mass, LU).
```

```
/*
What is to be done with idioms?
This is a nasty, short-term solution, till a better one is discovered.
*/

idiom(here, attached, [(enclosed, desc)]).
```

```
/*************************************************************************
*************************************************************************
SENTENCE & MENTION GENERATOR
© 1987, 1988 Victor Loewen
*************************************************************************
*************************************************************************/

/* SIBLINGS OR  CASE ROLE FILLERS */

generate_siblings([], [], []).

generate_siblings([Case-(_, _, Prep) | Slots], Siblings, [Case-Phrase| Phrases]) :-
    cut_out(Case-Info, Siblings, S2),
    generate_phrase(Case, Prep, Info, Phrase),
    generate_siblings(Slots, S2, Phrases).
generate_siblings([Case-(o, _, _) | Slots], Siblings, Words) :-
    generate_siblings(Slots, Siblings, Words).

/*
allowance for peripheral slots
*/
generate_siblings([], [Case-Info | Rest], [Case-Phrase| Phrases]) :-
    typical_prep(Case, Prep),
    generate_phrase(Case, Prep, Info, Phrase),
    generate_siblings([], Rest, Phrases).


typical_prep(location, to).
typical_prep(time, nil).
typical_prep(manner, as).
typical_prep(cause, for).
typical_prep(affected, nil).
% agent, instrument, beneficiary, result, source, destination, attribute?


/*
allowances for non-sibling material
*/

generate_siblings([],[right_dislocation-(lu(_,pro-(_,_,P,N)),lu(_,adj-(same,_))) | Rest],
       [right_dislocation-[Emphatic]| W]) :-
    generate_emphatic(P, N, Emphatic),
    generate_siblings([], Rest, W).
generate_siblings([], [right_dislocation-Info | Rest], [right_dislocation-Words| W]) :-
    generate_phrase(_, nil, Info, Words),
    generate_siblings([], Rest, W).

generate_siblings([], [conjunction-Info | Rest], [conjunction-Words| W]) :-
    generate_conj(Info, Words),
    generate_siblings([], Rest, W).

/*
Gender will have to be incorporated sometime for the following.
*/

generate_emphatic(1, s, myself).
generate_emphatic(2, s, yourself).
generate_emphatic(3, s, himself).
generate_emphatic(3, s, herself).
generate_emphatic(3, s, itself).
```

```
generate_emphatic(1, pl, ourselves).
generate_emphatic(2, pl, yourselves).
generate_emphatic(3, pl, themselves).
```

/* PHRASES */

```
generate_phrase(Case, Prep, prep_phrase-PP, Words) :-
    generate_prep_phrase(Case, Prep, PP, Words).

generate_phrase(Case, Prep, np-Branch, Words) :-
    generate_noun_phrase(Case, Prep, Branch, Words).

generate_phrase(Case, Prep, name - Name, Words) :-
    generate_names(_, NAME, name-Name),
    add_prep(Case, Prep, NAME, Words).

generate_phrase(Case, Prep, adj-Adj, Words) :-
    generate_adj_phrase(Case, Prep, Adj, Words).

generate_phrase(Case, Prep, event_prop-EP, Words) :-
    generate_event_prop(Case, Prep, EP, Words).

generate_phrase(Case, Prep, relative_prop - RP, Words) :-
    generate_rel_prop(Case, Prep, RP, Words).

generate_phrase(Case, Prep, adv -  Adv, Words) :-
    generate_adv(Case, Prep, Adv, Words).

generate_phrase(Case, Prep, (A, B), Words) :-
    generate_phrase(Case, Prep, A, AA),
    generate_phrase(Case, Prep, B, BB),
    append(AA, BB, Words).
```

/*
Since prepositions are generated for noun phrases within the processing of the noun phrase itself (even when it is not in a prepositional phrase), the preposition generated at the beginning of the generation of a prepositional phrase is passed along to the generation of the noun phrase. There may therefore be some redundant checking of the preposition.
*/

```
generate_prep_phrase(Case, Suggest_prep,prep_phrase -(Prep_info, np-NP), Words) :-
    generate_prep(Case, Suggest_prep, Prep_info, Prep_word),
    generate_noun_phrase(Case, Prep_word, NP, (Words, _)).

generate_prep_phrase(Case, Prep, (A, articulation-C, B), Word_list) :-
    generate_prep_phrase(Case, Prep, A, A_words),
    generate_articulation(C, C_words),
    generate_prep_phrase(Case, Prep, B, B_words),
    append(A_words, C_words, L1),
    append(L1, B_words, Word_list).


generate_qualifying_prep_phrase(qualifying_prep_phrase-
        (Prep_info, np -NP), Words) :-
    generate_prep(_, of, Prep_info, Prep_word),
    generate_noun_phrase(Case, Prep_word, NP, (Words,_)).
generate_qualifying_prep_phrase(qualifying_prep_phrase -
        (Prep_info, date - Date), Words) :-
```

```
    generate_prep(_, of, Prep_info, Prep_word),
    generate_date(Prep_word, Date, Words).
generate_qualifying_prep_phrase(qualifying_prep_phrase -
        (Prep_info, name-Name), Words) :-
    generate_prep(_, of, Prep_info, Prep_word),
    generate_names(_, NAME, Name),
    add_prep(_, Prep_word, NAME, Words).
generate_qualifying_prep_phrase((A, articulation-C, B), Words) :-
    generate_qualifying_prep_phrase(A, Mod_a),
    generate_articulation(C, C_words),
    generate_qualifying_prep_phrase(B, Mod_b),
    append(Mod_a, C_words, L1),
    append(L1, Mod_b, Words).
```

/* WORDS

Until a better way is found, idioms functioning as prepositions will have to be dealt with individually. This short database may become quite long. See the "parser - preposition" file for more possibilities. It may be better to "translate" at that point to avoid duplicating the database. However, the results of analysis would not then be unilingual, thus destroying the transferability of the analysis module to a different language pair.
*/

```
generate_prep(_, nil, prep - lu(_,prep - [functional_prep]), nil).
generate_prep(_, Prep, prep - lu(F,prep - [functional_prep]), Prep).
generate_prep(_, Prep, prep - lu(F,prep - [P | _]), P).

generate_prep(_, _, locution_prépositive-(lu(_, prep-_), lu(_, noun-(function, _)),
    lu(_, prep-_)), 'according to').
generate_prep(_, _, locution_prépositive-(lu(_, prep-_), lu(_, noun-(measure, _)),
    lu(_, prep-_)), able).


add_prep(agent, [functional_prep], X, X).
add_prep(affected, [functional_prep], X, X).
add_prep(beneficiary, [functional_prep], X, [to | X]).
add_prep(cause, [functional_prep], X, [for | X]).
add_prep(_, nil, X, X).
add_prep(_, Prep, X, [Prep | X]).
```

/*
ADJECTIVES
*/

```
generate_adj_phrase(Case, Prep, lu(Format, adj-(Adj, _)), [Adj]).
generate_adj_phrase(Case, Prep, lu -(lu(F1,adv -(Adv, Desc)),
    lu(F2, adj - (Adj, _))), Words) :-
        generate_adv(_, nil, lu(F1,adv -(Adv, Desc)), ADV),
        format(F2, Adj, ADJ),
        add_prep(Case, Prep, [ADV, ADJ], Words).
```

/*
ADVERBS
*/

```
generate_adv(Case, Prep, lu(F,adv - (A, (_,add_ly))), Words) :-
    ly_ending(A, Aly),
```

```
    format(F, Aly, Adv),
    add_prep(Case, Prep, [Adv], Words).
generate_adv(Case, Prep, lu(F,adv - (A, _)), Words) :-
    format(F, A, Adv),
    add_prep(Case, Prep, [Adv], Words).

ly_ending(Word, Word_ly) :-
    explode(Word, List),
    app(Before_y, [y], List),
    app(Before_y, [ily], List2),
    explode(Word_ly, List2).
ly_ending(Word, Word_ly) :-
    explode(Word_ly, [Word, ly]).

/*
ARTICULATIONS
*/

generate_articulation(lu(_, conj -(C, _)), [C]).
generate_articulation(p - (P), [P]).
generate_articulation((p - (,), lu(_, conj -(C, _))), [',', C]).

/*
CONJUNCTIONS
*/

generate_conj(lu(Format, conj - (C, _)), [C]).

/*
DATES
The article is simply dropped.
The first rule is for dates in standard international notation.
*/

generate_date(Prep, (_, weekday - nil,
    year - number(Year,_), month -number(Month,_), day -number(Day,_)), Words) :-
            explode(Date, [Year, '-', Month, '-', Day]),
            add_prep(_, Prep, [Date], Words).
generate_date(Prep, (_, weekday - lu(_, noun - (WEEKDAY, _)),
    day - number(Day_number, _), month - lu(_, noun - (MONTH, _)),
    year - number(YEAR, _)), DATE) :-
        format(proper, WEEKDAY, Day),
        format(proper, MONTH, Month),
        add_prep(_, Prep, [Day, ',', Month, Day_number, ',', YEAR], DATE).
generate_date(Prep, (_, weekday - nil, day - number(Day_number, _),
    month - lu(_, noun - (MONTH, _)), year - number(YEAR, _)), DATE) :-
        format(proper, MONTH, Month),
        add_prep(_, Prep, [Month, Day_number, ',', YEAR], DATE).
generate_date(Prep, (_, weekday - nil, day - number(Day_number, _),
    month - lu(_, noun - (MONTH, _)), year - nil), DATE) :-
        format(proper, MONTH, Month),
        add_prep(_, Prep, [Month, Day_number], DATE).
generate_date(Prep, (_, weekday - nil, day - nil, month - lu(_,noun-(MONTH, _)),
    year - number(YEAR, _)), DATE) :-
        format(proper, MONTH, Month),
        add_prep(_, Prep, [Month, YEAR], DATE).
generate_date(Prep, (_, weekday - nil, day - nil, month - lu(_,noun-(MONTH, _)),
    year - nil), DATE) :-
        format(proper, MONTH, Month),
        add_prep(_, Prep, [Month], DATE).
```

```
/*************************************************************************
*************************************************************************
GENERATOR - SENTENCE ARRANGEMENT
© 1987, 1988 Victor Loewen
*************************************************************************
*************************************************************************/

/*
ARRANGING THE MATERIAL

Arrangement of the material provided at this point involves picking the subject
and then generating the verb based on the person/number of the subject and on
the tense and attitude of the sentence.

The subject-verb order is then determined based on the form (declarative, imperative,
or interrogative) of the sentence.
If the form is declarative, the subject-verb arrangement remains the same and the final
punctuation is a period.
If the form is interrogative, the subject-verb order is inverted and the final punctuation
is a question mark.
If the form is imperative, the zero subject is dropped and the final punctuation is a period.
(Exclamation marks are a bit overzealous in an administrative context.)

The direct object is then picked, if there is to be one.

If the sentence is passive,
the direct object becomes the subject and the previously selected subject and verb
are changed appropriately.
Voices: active, passive
Verb generation follows the following rules.
    active --> no change
    passive --> be + verb
The alternative is to make the agent the subject, and the affected the direct object and
leave the verb in the active form.

Arrangement of subject/verb/direct object also involves considering the mood.
Moods: ind(icative), subj(unctive), cond(itional)
Changes follow the following rules.
    indicative --> no change
    subjunctive --> no change, unless "if" clause expressing unreal condition or "that"
        clause after verb whose meaning is an element of the mental interaction set of events
        (see ontology), i.e., those verbs expressing requests, recommendations, and demands
            e.g.
            is --> were (if I am --> if I were; I wish that I am --> I wish that I were)
            verb --> were + verb + ing (if I go --> if I were going)
            is --> be (I ask that there is --> I ask that there be)
            verb (3, sing) --> verb (1, sing) (I ask that he remains --> I ask that he remain)
    conditional --> no change, as English has no conditional mood
(cf. Prentice-Hall Handbook for Writers, 8th Edition, pp. 55-56)
These changes are made here, as they are dependent on context (the presence of "if," "that,"
intensional meaning of verb in principal clause, etc.). Contextual information including
the meaning of the main verb and the presence of "if, etc." will have
to be garnered and brought to bear on processing at this point. This will require more
changes in the rules.
```

attitude (affirmative or one of the negatives or doubt)

The remainder of the siblings are then placed in an appropriate order. Adverbials are placed according to the following:

adverbs of place precede
adverbs of manner precede
adverbs of time

In arranging the siblings, some assumptions will be made. Combinations of agent-beneficiary-affected will become the expected subject-indirect object-direct object. "Jean donne un livre à Marie." will inevitably become "Jean gives a book to Mary." as no attempt is made to find all possible permutations of surface structures carrying the same message. Moreover, what is easiest to program in this case also produces an adequate translation.
*/

```
arrangement((Tense,Mood,Voice,Attitude,Form), Event, Siblings, Word_list) :-
    pick_subject_object(Siblings, Sibs2, Subject, Dir_obj, Person, Num),
    generate_verb(Tense, Attitude, Person, Num, Event, Meaning, Verb),
    apply_form(Form, Person, Num, Subject, Verb, Interim1),
    apply_active_passive(Voice, Interim1, Dir_obj, Interim2),
    apply_mood(Mood, Meaning, Interim2, Interim3),
    arrange_other_cases(Sibs2, Sibs3, Interim3, Interim4),
    add_peripherals(Sibs3, Sibs4, Interim4, Interim5),
    add_right_disloc(Sibs4, Sibs5, Interim5, Interim6),
    add_conj(Sibs5, Interim6, Word_list).

arrange_event_prop((Tense,Mood, _, Att,Form), Event, Prep, Siblings, Word_list) :-
    generate_verb(Tense, Att, Mood, _, Event, Meaning, Verb),
    pick_dir_obj([affected, result], Siblings, Sibs2, Dir_obj),
    append_prep(Mood, Prep, Verb, Dir_obj, Interim1),
    arrange_other_cases(Sibs2, Sibs3, Interim1, Interim2),
    add_peripherals(Sibs3, Sibs4, Interim2, Interim3),
    add_right_disloc(Sibs4, Sibs5, Interim3, Interim4),
    add_conj(Sibs5, Interim4, Word_list).

append_prep(Mood, to, Verb, Dir_obj, Interim1) :-
    member(to, Verb),
    append(Verb, Dir_obj, Interim1).
append_prep(participle, Prep, Verb, Dir_obj, Interim1) :-
    append(Verb, Dir_obj, Interim1).
append_prep(Mood, Prep, Verb, Dir_obj, Interim1) :-
    append([Prep | Verb], Dir_obj, Interim1).


arrange_rel_prop((Tense,Mood,_,Att,Form), Event, Siblings, Word_list) :-
    find_that(Siblings, Sibs, THAT),
    pick_subject_object(Sibs, Sibs2, Subject, Dir_obj, Person, Num),
    generate_verb(Tense, Att, Person, Num, Event, Meaning, Verb),
    apply_form(Form, Person, Num, Subject, Verb, Interim1),
    apply_active_passive(Voice, Interim1, Dir_obj, Interim2),
    apply_mood(Mood, Meaning, Interim2, Interim3),
    arrange_other_cases(Sibs2, Sibs3, Interim3, Interim4),
    add_peripherals(Sibs3, Sibs4, Interim4, Interim5),
    add_right_disloc(Sibs4, Sibs5, Interim5, Interim6),
    add_conj(Sibs5, Interim6, Interim7),
    append(THAT, Interim7, Word_list).
```

```
/*
Either the Agent or Affected case may produce a surface syntactic subject.
cf. Fillmore (33)
*/

pick_subject_object(Siblings, Sibs3, Subject, Dir_obj, Per, Num) :-
    mem((S, O), [(agent, [affected,result]), (affected, nil)]),
    cut_out(S-(Subject, Per, Num), Siblings, Sibs2),
    pick_dir_obj(O, Sibs2, Sibs3, Dir_obj).

pick_dir_obj(Options, Sibs2, Sibs3, Obj) :-
    mem(Case, Options),
    cut_out(Case-(Obj, _), Sibs2, Sibs3).
pick_dir_obj(_, Sibs, Sibs, []).


apply_form(declarative, _, _, Subject, Verb, SV) :-
    append(Subject, Verb, SV).

apply_form(interrogative, Pers, Num, Subject, [Verb], Aux_subj_verb) :-
    conjugate(be, (pres, ind, Pers, Num), Aux),
    explode(Verb_ing, [Verb, ing]),
    append(Aux, Subject, L1),
    append(L1, [Verb_ing], Aux_subj_verb).

apply_form(interrogative, Pers, Num, Subject, [will, Verb], Will_subj_be_Verb_ing) :-
    explode(Verb_ing, [Verb, ing]),
    append([will], Subject, L1),
    append(L1, [be, Verb_ing], Will_subj_be_Verb_ing), !.

apply_form(interrogative, Pers, Num, Subject, [Aux, Verb], Aux_subj_Verb) :-
    append([Aux], Subject, L1),
    append(L1, [Verb], Aux_subj_Verb).

apply_form(imperative, 1, pl, _, [Aux, Verb], [let, us, Verb]).
apply_form(imperative, _, _, _, V, V).


apply_active_passive(active, Interim1, Dir_obj, Interim2) :-
    append(Interim1, Dir_obj, Interim2).

apply_active_passive(passive, Interim1, Dir_obj, active_passive_error).
%  will have to bring in person/number info for both subj and obj and tense to generate
%  e.g  you are being thanked by us; you were thanked by us

/*
Application of mood is glossed over at this point, since more information is
required to deal with it properly. (see comments above)
*/
apply_mood(_, _, Interim, Interim).


apply_attitude(affirmative, Verb, Verb).
apply_attitude(negative-pas, Verb, [Aux, Verb]) :-
    negate(V, Neg_v).

/*
Tense information will be required to correctly negate a verb.
e.g. We sent the letter.  We did not send the letter.
        We thank you.  We do not thank you / We are not thanking you
This procedure may have to be moved up to be part of generate_verb.
```

```
*/

negate([Single], [negate,error,Single]).

arrange_other_cases([], [], Interim, Interim) :- !.
arrange_other_cases(Sibs3, Sibs4, Interim4, Interim5) :-
    sort([attribute, beneficiary, instrument, result, source, destination,affected],
        Sibs3, Cuts, Sibs4),
    append(Interim4, Cuts, Interim5).

sort([], Sibs_left, [], Sibs_left) :- !.
sort([Case | Cases], Sibs_in, Sorted, Sibs_out) :-
    cut_out(Case-C, Sibs_in, Temp),
    sort(Cases, Temp, Cuts, Sibs_out),
    (list(C), Cut = C ; C = (Cut, _, _)),
    append(Cut, Cuts, Sorted).
sort([_ | Cases], Sibs_in, Cuts, Sibs_out) :-
    sort(Cases, Sibs_in, Cuts, Sibs_out).


/*
The order of peripherals may be changed for stylistic reasons. It is assumed that
more often than not indications of manner, location, time and cause will follow the order
indicated.
When this goal is executed, all siblings should be processed (the reason for the empty set).
*/

add_peripherals([], [], Interim, Interim) :- !.
add_peripherals(Sibs4, Sibs5, Interim5, Interim6) :-
    sort([manner, location, time, cause], Sibs4, Cuts, Sibs5),
    append(Interim5, Cuts, Interim6).

add_right_disloc([], [], X, X) :- !.
add_right_disloc(Sibs5, Sibs6, Interim6, Interim7) :-
    sort([right_dislocation], Sibs5, Cuts, Sibs6),
    (Cuts \= [], append(Interim6, [',' | Cuts], Interim7) ;
        Interim6 = Interim7).

/*
When the following is executed, all siblings should be processed (the reason for the empty
set).
*/

add_conj([], X, X) :- !.
add_conj(Sibs, Interim1, Interim2) :-
    sort([conjunction], Sibs, Cuts, []),
    append(Cuts, Interim1, Interim2).

/*
The relative pronoun "that" needs to be taken out of the list. The case processor may
or may not have required it, depending on the frame.
*/

find_that([], [], _) :- !.
find_that([_ - (Words, _) | Sibs], Sibs, [that]) :-
    member(that, Words), !.
find_that([H | T], [H | L], THAT) :-
    find_that(T, L, THAT).
```

```
/*************************************************************************
*************************************************************************
MENTION GENERATOR
© 1987, 1988 Victor Loewen
*************************************************************************
*************************************************************************/

/*
Generalities
*/
generate_mention([C], conj-lu(_,conj -(C, _))).
generate_mention([P], p-(P)).
generate_mention([S], symb - (S)).


/*
Noun phrase generation is defined in the "Generator - Nouns" file.
*/

generate_mention(Words, np - Branch) :-
    generate_noun_phrase(nil, nil, Branch, (Words, _, _)).

/*
A general rule for generating addresses involving street numbers and street names
requires slots for the basic elements. Rearrangement is straightforward. The second
rule is for instances of street names where the way type is construed as a head noun of
a noun phrase (e.g., 25, rue de l'école)
Capitalization of the way type (i.e., avenue, street, etc.) is double checked in the event
it is defined as a common noun in the dictionary.
*/

generate_mention(A,  address-(number(N,_), lu(_,noun -(Way, _)), np - Way_name)) :-
    generate_noun_phrase(nil, nil, Way_name, (Name_list, _, _)),
    capitalize(Way, Type_of_way),
    cap_words(Name_list, Name),
    append([N | Name], [Type_of_way], A).
generate_mention(A, address -(number(N, _), lu(_,noun -(Way, _)),
    qualifying_prep_phrase - QPP)) :-
        generate_qualifying_prep_phrase( QPP, QPP_list),
        app([of,the], Name_list, QPP_list),
        cap_words(Name_list, Name),
        capitalize(Way, Type_of_way),
        append([N | Name], [Type_of_way], A).

generate_mention(A, address - (np - NP1, np -NP2)) :-
    generate_noun_phrase(nil, nil, NP1, (Words1, _, _)),
    generate_noun_phrase(nil, nil, NP2, (Words2, _, _)),
    cap_words(Words1, City),
    cap_words(Words2, Province),
    append(City, [',' | Province], A).


/*
Prepositional phrases
*/

generate_mention(Word_list, prep_phrase -PP) :-
    generate_prep_phrase(_, nil, PP, Word_list).
```

```
generate_mention(Word_list, qualifying_prep_phrase -QPP) :-
    generate_qualifying_prep_phrase(QPP, Word_list).


/*
Postal codes and phone numbers need only be put back in order.
*/

generate_mention([Left, Right], postal_code -
    (lu(_, string - A1),
    number(N1,_),
    lu(_,string - B1),
    number(N2,_),
    lu(_,string - C1),
    number(N3,_))) :-
        letter(A1, A), letter(B1,B), letter(C1, C),
        explode(Left, [A, N1, B]), explode(Right, [N2, C, N3]).

generate_mention([Ex_num], phone_number -
    (number(nil,_),
    number(Exchange, _),
    number(Number, _))) :-
        explode(Ex_num, [Exchange, '-', Number]), !.
generate_mention([Area, Ex_num], phone_number -
    (number(Area_code,_),
    number(Exchange, _),
    number(Number, _))) :-
        explode(Area, ['(', Area_code, ')']),
        explode(Ex_num, [Exchange, '-', Number]), !.


/*
Names need only be capitalized, including adjectives used in company names.
Adjectives are not to be rearranged. Generation is therefore literally based on the
French order and words, unless the words have been translated (e.g. fils --> sons).
*/

generate_mention(NAME, name -Name) :-
    generate_names(_, NAME, Name).

    generate_names(Num, ['Co.'], name -lu(_,noun -(company, _))).
    generate_names(Numb, [NAME], name -lu(_,noun -(Name, Num, _, CM))) :-
        format(Name, _, Num, proper, CM, NAME).
    generate_names(Num, [INITIAL], initials -lu(Format, string - STRING)) :-
        format(Format, STRING, I),
        explode(INITIAL, [I, '.']).
    generate_names(Num, ['Inc.'], adj -lu(_, adj - (ADJ, _))) :-
        explode(ADJ, [i,n,c | _]).
    generate_names(Num, ['Ltd.'], adj -lu(_, adj - (ADJ, _))) :-
        explode(ADJ, [l,i,m,i,t | _]).
    generate_names(Num, [P], p - (P)).
    generate_names(Num, [S], symb-(S)).
    generate_names(pl, [C], conj-lu(_,conj -(C, _))).

    generate_names(Num, ['Sir,' | NAME], (name -lu(_,noun -(sir, _)), p-(','), Name)) :-
        generate_names(Num, NAME, Name).
    generate_names(pl, ['Sir', C | NAME], (name-lu(_, noun-(sir, _)),
            conj-lu(_,conj -(C, _)), Name)) :-
        generate_names(Num, NAME, Name).

    generate_names(Num, ['Mr.' | NAME], (name -lu(_,noun -(sir, _)), Name)) :-
```

```
    generate_names(Num, NAME, Name).
generate_names(Num, ['Mr.' | NAME], (initials = lu(_,string = m), Name)) :-
    generate_names(Num, NAME, Name).
generate_names(Num, ['Mrs.' | NAME], (name =lu(_,noun =(madam, _)), Name)) :-
    generate_names(Num, NAME, Name).
generate_names(Num, ['Mrs.' | NAME], (initials = lu(_,string = mme), Name)) :-
    generate_names(Num, NAME, Name).
generate_names(Num, ['Miss' | NAME], (initials = lu(_,string = mlle), Name)) :-
    generate_names(Num, NAME, Name).

generate_names(Num, Name, (A, B)) :-
    generate_names(Num, AA, A),
    generate_names(Num, BB, B),
    append(AA, BB, Name).
```

/*
Reference material
The two rules are quite specific, as they works only for references whose format in the
French surface structure was quite abbreviated.
i.e. "V/R. du date/code" and "V/R".
*/

```
generate_mention([Word, reference | Word_list],
    reference =(abbrev=lu(_,string = V_N), abbrev = lu(_,string = r), Rest)) :-
            reference(V_N, Word),
            generate_mention(Word_list, Rest).
generate_mention([Word, reference],
    reference =(abbrev=lu(_,string = V_N), abbrev = lu(_,string = r))) :-
            reference(V_N, Word).

generate_mention([Word | Word_list], reference = (abbrev = lu(_, string = V_N), Rest)) :-
    reference(V_N, Word),
    generate_mention(Word_list, Rest).

reference(v, your).
reference(n, our).
```

/*
Codes
*/

```
generate_mention([Code], code=Code_contents) :-
    generate_code(Code_list, Code_contents),
    explode(Code, Code_list).

    generate_code([Formatted_string], lu(Format,string = String)) :-
        format(Format, String, Formatted_string).
    generate_code([Number], number(Number, _)).
    generate_code([P], p = (P)).
    generate_code([S], symb=(S)).
    generate_code(Code, (A, B)) :-
        generate_code(AA, A),
        generate_code(BB, B),
        append(AA, BB, Code).
```

/*
Interjections

```
*/

generate_mention([Word], interjection - lu(F, LU)) :-
    format(F, LU, Word).

/*
Delivery (mailing) notation
i.e., REGISTERED, CONFIDENTIAL, PERSONAL, HAND-DELIVERED
The first rule is required, since context sensitivity has not yet been built into
the transfer module and the first equivalent suggested for "recommandé" is the first one used.
*/

generate_mention(['REGISTERED'], character-lu(_,adj -(recommended, _))).
generate_mention([Word], character-lu(_,adj -(X, _))) :-
    format(all_caps, X, Word).


/*
DATES
*/

generate_mention(Words, date - Date) :-
    generate_date(nil, Date, Words).

/*
SUBJECT LINES
*/

generate_mention([subject,':' | Word_list],
    (subject -(lu(_, noun -  (object, _)), p -  :), MENTIONS)) :-
        generate_mention(Word_list, MENTIONS).

/*
SALUTATIONS
*/

generate_mention([ADJ | NAME], salutation-(adj-lu(F, adj-(Adj, _)), name -Name)) :-
    format(F, Adj, ADJ),
    generate_names(Num, N, Name),
    append(N, [':'], NAME).
generate_mention(S, salutation-(adj-lu(F, adj-(Adj, _)), name -Name, A)) :-
    format(F, Adj, ADJ),
    generate_names(Num, NAME, Name),
    generate_mention(AA, A),
    append([ADJ | NAME], AA, List),
    append(List, [':'], S).
generate_mention([ADJ | NAME], salutation-(adj-lu(F, adj-(Adj, _)), np-NP)) :-
    format(F, Adj, ADJ),
    generate_noun_phrase(nil, nil, NP, (Word_list, _, _)),
    append(Word_list, [':'], NAME).
generate_mention(S, salutation -(name -Name)) :-
    generate_names(Num, NAME, Name),
    append(NAME, [':'], S).
generate_mention(S, salutation -(name -Name, A)) :-
    generate_names(Num, NAME, Name),
    generate_mention(AA, A),
    append(NAME, AA, LIST),
    append(LIST, [':'], S).
generate_mention(S, salutation-(np-NP)) :-
    generate_noun_phrase(nil, nil, NP, (Word_list, _, _)),
```

```
    append(Word_list, [':'], S).


/*
Reference initials of sender and transcriber.
The transcribers initials are verified as being all lower case.
*/

generate_mention([Reference], dactylos_initials -
    (lu(_, string - S1), symb - (S), lu(low,string - S2))) :-
        format(all_caps, S1, Sender),
        format(all_caps, S2, Interim),
        explode(Interim, L1),
        cap_list(L2, L1),
        explode(Transcriber, L2),
        explode(Reference, [Sender, '/', Transcriber]).

/*
Enclosure and carbon copy notation
*/

generate_mention(['Encl.'], enclosures-_).
generate_mention(['c.c.:'], carbon_copy -_).

/*
Generate composite mentions one component at a time.
Any final period after a name is dropped in English.
*/

generate_mention(NAME, (name-Name, p-('.'))) :-
    generate_mention(NAME, name-Name).

generate_mention(Word_list, (A, B)) :-
    generate_mention(A_words, A),
    generate_mention(B_words, B),
    append(A_words, B_words, Word_list).
```

```
/*************************************************************************
*************************************************************************
FRENCH DICTIONARY CONTAINING ENTRIES FOR CLOSED CATEGORIES
© 1987, 1988 Victor Loewen
*************************************************************************
*************************************************************************/

f(le,[art - (le, def, m, s),pro - (le, rel, 3, s, m, _)]).
f(la,[art - (la, def, f, s),pro - (la, rel, 3, s, f, _)]).
f(les,[art - (les, def, (m,f), pl),pro - (les, rel, 3, pl, (m,f), _)]).
f(le_la,[art - (le_la, def, (m,f), s),pro - (le_la, rel, 3, s, (m,f), _)]).

f(un,[adj - (un, cardinal, m, s, [quantity]),art - (un, indef, m, s)]).
f(une,[adj - (une, cardinal, f, s, [quantity]),art - (une, indef, f, s)]).
f(deux,[adj - (deux, cardinal, (m,f), pl, [quantity])]).
f(trois,[adj - (trois, cardinal, (m,f), pl, [quantity])]).
f(quatre,[adj - (quatre, cardinal, (m,f), pl, [quantity])]).
f(cinq,[adj - (cinq, cardinal, (m,f), pl, [quantity])]).
f(six,[adj - (six, cardinal, (m,f), pl, [quantity])]).
f(sept,[adj - (sept, cardinal, (m,f), pl, [quantity])]).
f(huit,[adj - (huit, cardinal, (m,f), pl, [quantity])]).
f(neuf,[adj - (neuf, cardinal, (m,f), pl, [quantity])]).
f(dix,[adj - (dix, cardinal, (m,f), pl, [quantity])]).
f(onze,[adj - (onze, cardinal, (m,f), pl, [quantity])]).
f(douze,[adj - (douze, cardinal, (m,f), pl, [quantity])]).
f(treize,[adj - (treize, cardinal, (m,f), pl, [quantity])]).
f(quatorze,[adj - (quatorze, cardinal, (m,f), pl, [quantity])]).
f(quinze,[adj - (quinze, cardinal, (m,f), pl, [quantity])]).
f(seize,[adj - (seize, cardinal, (m,f), pl, [quantity])]).
f(vingt,[adj - (vingt, cardinal, (m,f), pl, [quantity])]).
f(trente,[adj - (trente, cardinal, (m,f), pl, [quantity])]).
f(quarante,[adj - (quarante, cardinal, (m,f), pl, [quantity])]).
f(cinquante,[adj - (cinquante, cardinal, (m,f), pl, [quantity])]).
f(soixante,[adj - (soixante, cardinal, (m,f), pl, [quantity])]).
f(cent,[adj - (cent, cardinal, (m,f), pl, [quantity])]).
f(mille,[adj - (mille, cardinal, (m,f), pl, [quantity]),
   noun - (mille, f, s, common, count, [numeral, unit])]).
f(million,[adj - (million, cardinal, (m,f), pl, [quantity]),
   noun - (million, f, s, common, count, [quantity])]).
f(milliard,[adj - (milliard, cardinal, (m,f), pl, [quantity]),
   noun - (milliard, f, s, common, count, [quantity])]).
f(billion,[adj - (billion, cardinal, (m,f), pl, [quantity]),
   noun - (billion, f, s, common, count, [quantity])]).

f(premier,[adj - (premier, ordinal, m, s, [ttrelation])]).
f(unième,[adj - (unième, ordinal, (m,f), s, [ttrelation])]).
f(second,[adj - (second, ordinal, m, s, [ttrelation])]).
f(deuxième,[adj - (deuxième, ordinal, (m,f), s, [ttrelation])]).
f(troisième,[adj - (troisième, ordinal, (m,f), s, [ttrelation])]).
f(quatrième,[adj - (quatrième, ordinal, (m,f), s, [ttrelation])]).
f(cinquième,[adj - (cinquième, ordinal, (m,f), s, [ttrelation])]).
f(sixième,[adj - (sixième, ordinal, (m,f), s, [ttrelation])]).
f(septième,[adj - (septième, ordinal, (m,f), s, [ttrelation])]).
f(huitième,[adj - (huitième, ordinal, (m,f), s, [ttrelation])]).
f(neuvième,[adj - (neuvième, ordinal, (m,f), s, [ttrelation])]).
f(dixième,[adj - (dixième, ordinal, (m,f), s, [ttrelation])]).

f(onzième,[adj - (onzième, ordinal, (m,f), s, [ttrelation])]).
f(douzième,[adj - (douzième, ordinal, (m,f), s, [ttrelation])]).
f(treizième,[adj - (treizième, ordinal, (m,f), s, [ttrelation])]).
f(quatorzième,[adj - (quatorzième, ordinal, (m,f), s, [ttrelation])]).
```

```
!(quinzième,[adj - (quinzième, ordinal, (m,f), s, [ttrelation])]).
!(seizième,[adj - (seizième, ordinal, (m,f), s, [ttrelation])]).
!(vingtième,[adj - (vingtième, ordinal, (m,f), s, [ttrelation])]).
!(trentième,[adj - (trentième, ordinal, (m,f), s, [ttrelation])]).
!(quarantième,[adj - (quarantième, ordinal, (m,f), s, [ttrelation])]).
!(cinquantième,[adj - (cinquantième, ordinal, (m,f), s, [ttrelation])]).
!(soixantième,[adj - (soixantième, ordinal, (m,f), s, [ttrelation])]).
!(centième,[adj - (centième, ordinal, (m,f), s, [ttrelation])]).
!(millième,[adj - (millième, ordinal, (m,f), s, [ttrelation])]).
!(millionième,[adj - (millionième, ordinal, (m,f), s, [ttrelation])]).
!(milliardième,[adj - (milliardième, ordinal, (m,f), s, [ttrelation])]).
!(billionième,[adj - (billionième, ordinal, (m,f), s, [ttrelation])]).
!(trillionième,[adj - (trillionième, ordinal, (m,f), s, [ttrelation])]).

!(simple,[adj - (simple, desc, (m,f), s, [quantity])]).
!(double,[adj - (double, desc, (m,f), s, [quantity])]).
!(triple,[adj - (triple, desc, (m,f), s, [quantity])]).
!(quadruple,[adj - (quadruple, desc, (m,f), s, [quantity])]).
!(quintuple,[adj - (quintuple, desc, (m,f), s, [quantity])]).
!(sextuple,[adj - (sextuple, desc, (m,f), s, [quantity])]).
!(septuple,[adj - (septuple, desc, (m,f), s, [quantity])]).
!(octuple,[adj - (octuple, desc, (m,f), s, [quantity])]).
!(nonuple,[adj - (nonuple, desc, (m,f), s, [quantity])]).
!(décuple,[adj - (décuple, desc, (m,f), s, [quantity])]).
!(centuple,[adj - (centuple, desc, (m,f), s, [quantity])]).

!(à,[prep - à]).
!(ainsi,[adv - (ainsi, manner),conj - (ainsi, sub)]).
!(après,[prep - après]).
!(après,[adv - (après, location),prep - après]).
!(auprès,[adv - (auprès, location)]).
!(avant,[noun - (avant, m, s, common, mass, [location]),adv - (avant, location),prep -
avant]).
!(avec,[adv - (avec, manner),prep - avec]).
!(beaucoup,[noun - (beaucoup, (m, f), pl, common, mass, [quantity])]).
%   adv - (beaucoup, quantity) is removed as "beaucoup" generally works better as a noun
!(ci,[adv - (ci, location)]).
!(contre,[noun - (contre, m, s, common, count, [submatter]),adv - (contre, location),prep -
contre]).
!(dans,[prep - dans]).
!(de,[prep - de]).
!(depuis,[adv - (depuis, time),prep - depuis]).
!(derrière,[noun - (derrière, m, s, common, mass, [location]),adv - (derrière, location),prep
- derrière]).
!(devant,[noun - (devant, m, s, common, mass, [location]),
    verb - (devoir, (pres - part,m,s), [state - [agent - (r, [something], nil),
        beneficiary - (o, [person], pour)]]),
    adv - (devant, location),
    prep - devant]).
!(donc,[conj - (donc, sub)]).
!(en,[prep - en,pro - (en, rel, 3, (s, pl), (m,f), [mthing,fthing,plant,nlthing])]).
!(entre,[verb - (entrer, (pres, (ind, subj), (1, 3), s),
    [selfdisplacement - [agent - (r, [human,animal], nil),
        instrument - (o, [artifact], avec),source - (o, [location], de),
        destination - (r, [location], dans)]]),
    adv - (entre, location),
    prep - entre]).
!(envers,[prep - envers]).
!(et,[conj - (et, coord)]).
!(hors,[adv - (hors, location),prep - hors]).
!(ici,[adv - (ici, location)]).
!(jusque,[prep - jusque,conj - (jusque, sub)]).
!(ne,[adv - (ne, negation)]).
```

```
f(ou,[conj - (ou, coord)]).
f(outre,[adv - (outre, location),prep - outre]).
f(par,[prep - par]).
f(parmi,[prep - parmi]).
f(pas,[noun - (pas, m, (s, pl), common, count, [unit]),adv - (pas, negation)]).
f(pour,[prep - pour]).
f(proche,[adv - (proche, location)]).
f(sans,[prep - sans]).
f(selon,[prep - selon]).
f(sur,[prep - sur]).
f(très, [adv-(très, intensity)]).
f(vers,[prep - vers]).
f(vraiment,[adv - (vraiment, intensity)  .

f(b, [string-b]).
f(c, [string-c]).
f(d, [string-d]).
f(e, [string-e]).
f(é, [string-é]).
f(f, [string-f]).
f(g, [string-g]).
f(h, [string-h]).
f(i, [string-i]).
f(j, [string-j]).
f(k, [string-k]).
f(l, [string-l]).
f(m, [string-m]).
f(n, [string-n]).
f(o, [string-o]).
f(p, [string-p]).
f(q, [string-q]).
f(r, [string-r]).
f(s, [string-s]).
f(t, [string-t]).
f(u, [string-u]).
f(v, [string-v]).
f(w, [string-w]).
f(x, [string-x]).
f(z, [string-z]).

f(je,[pro - (je, pers, 1, s, (m,f), [person])]).
f(me,[pro - (me, refl, 1, s, (m,f), [person])]).
f(moi,[pro - (moi, refl, 1, s, (m,f), [person])]).

f(tu,[pro - (tu, pers, 2, s, (m,f), [person])]).
f(te,[pro - (te, refl, 2, s, (m,f), [person])]).
f(toi,[pro - (toi, refl, 2, s, (m,f), [person])]).

f(il,[pro - (il, pers, 3, s, m, [person, mthing])]).
f(elle,[pro - (elle, pers, 3, s, f, [person, mthing]), pro - (elle, refl, 3, s, f,
[person])]).
f(lui,[pro - (lui, refl, 3, s, (m,f), [person])]).
f(se,[pro - (se, refl, 3, s, (m,f), [person])]).
f(soi,[pro - (soi, refl, 3, s, (m,f), [person])]).
f(celui,[pro - (celui, dem, 3, s, m, [person, mthing])]).
f(celle,[pro - (celui, dem, 3, s, f, [person, mthing])]).

f(nous,[pro - (nous, pers, 1, pl, (m,f), [person]), pro - (nous, refl, 1, pl, (m,f),
[person])]).
f(vous,[pro - (vous, pers, 2, (s,pl), (m,f), [person]), pro - (vous, refl, 2, (s,pl), (m,f),
[person])]).

f(ils,[pro - (ils, pers, 3, pl, m, [person])]).
```

```
f((elles,[pro - (elles, pers, 3, pl, f, [person])]).
f((eux,[pro - (eux, refl, 3, pl, (m,f), [person])]).
f((ceux,[pro - (ceux, dem, 3, pl, (m,f), [person])]).

f((mon,[adj - (mon, poss, m, s, [ttrelation])]).
f((ma,[adj - (ma, poss, f, s, [ttrelation])]).
f((mes,[adj - (mes, poss, (m,f), pl, [ttrelation])]).

f((ton,[adj - (ton, poss, m, s, [ttrelation])]).
f((ta,[adj - (ta, poss, f, s, [ttrelation])]).
f((tes,[adj - (tes, poss, (m,f), pl, [ttrelation])]).

f((son,[adj - (son, poss, m, s, [ttrelation])]).
f((sa,[adj - (sa, poss, f, s, [ttrelation])]).
f((ses,[adj - (ses, poss, (m,f), pl, [ttrelation])]).

f((notre,[adj - (notre, poss, (m,f), s, [ttrelation])]).
f((nos,[adj - (nos, poss, (m,f), pl, [ttrelation])]).

f((votre,[adj - (votre, poss, (m,f), s, [ttrelation])]).
f((vos,[adj - (vos, poss, (m,f), pl, [ttrelation])]).

f((leur,[adj - (leur, poss, (m,f), s, [ttrelation]),
   pro - (leur, refl, 3, pl, (m,f), [person])]).
f((leurs,[adj - (leurs, poss, (m,f), pl, [ttrelation])]).

f((ceci,[pro - (ceci, dem, 3, s, m, _)]).
f((cela,[pro - (cela, dem, 3, (s, pl), (m, f), _)]).

f((ce,[adj - (ce, dem, m, s, [name]),pro - (ce, rel, 3, s, m, _)]).
f((cet,[adj - (cet, dem, m, s, [name])]).
f((ces,[adj - (ces, dem, (m,f), pl, [name])]).

f((lequel,[pro - (lequel, rel, 3, s, m, _)]).
f((laquelle,[pro - (lequel, rel, 3, s, f, _)]).
f((lesquels,[pro - (lequel, rel, 3, pl, m, _)]).
f((lesquelles,[pro - (lequel, rel, 3, pl, f, _)]).

f((quel,[pro - (quel, interr, 3, s, m, _)]).
f((quelle,[pro - (quel, interr, 3, s, f, _)]).
f((quels,[pro - (quel, interr, 3, pl, m, _)]).
f((quelles,[pro - (quel, interr, 3, pl, f, _)]).

f((quelque,[adj - (quelque, indef, (m, f), s, [quantity])]).
f((quelconque,[adj - (quelconque, indef, (m, f), s, [quantity])]).
f((chaque,[adj - (chaque, indef, (m, f), s, [quantity])]).
f((divers,[adj - (divers, indef, (m, f), pl, [quantity])]).
f((maint,[adj - (maint, indef, m, s, [quantity])]).
f((nul,[adj - (nul, indef, m, s, [quantity])]).
f((plusieurs,[adj - (plusieurs, indef, (m, f),pl, [quantity])]).
f((tout,[adj - (tout, indef, m, s, [quantity])]).
f((tous,[adj - (tous, indef, m, pl, [quantity])]).
f((même,[adj - (même, indef, (m,f), s, [attribute]), adv-(même, affirmation),
   noun - (même, (m,f), s, common, count, _)]).
f((autre,[adj - (autre, indef, m, s, [quantity])]).

f((qui,[conj - (qui, sub),pro - (qui, rel, 3, (s, pl), (m, f), _)]).
f((que,[conj - (que, sub),pro - (que, rel, 3, (s, pl), (m, f), _)]).
f((y,[adv - (y, location), pro - (y, rel, 3, (s, pl), (m, f), [location]), string-y]).

avoir_m_f([possession - [agent - (r, [something], nil),affected - (r, [something], nil),
   beneficiary-(o,[event,person],for)]]).
etre_m_f([state - [affected - (r, [something], nil),attribute - (r, [something], nil),
```

```
    beneficiary-(o,[sentient],nil)]]).
faire_m_f([posprocess - [agent - (r, [event,sentient], nil),
    instrument - (o, [artifact,mthing], avec), beneficiary - (o, [something], pour),
    result - (r, [event,pthing,mthing], nil)],
physinteraction - [agent - (r, [person], nil),affected - (r, [event], nil)]]).

f(avoir,[noun-(avoir,m,s,common,count,[pthing,comparison]),verb-(avoir,int,MF)]) :-
    avoir_m_f(MF).

f(ai, [verb-(avoir, (pres, ind, 1, s), MF)]) :- avoir_m_f(MF).
f(as, [verb-(avoir, (pres, ind, 2, s), MF)]) :- avoir_m_f(MF).
f(a, [verb-(avoir, (pres, ind, 3, s), MF), string-a]) :- avoir_m_f(MF).
f(avons, [verb-(avoir, (pres, ind, 1, pl), MF)]) :- avoir_m_f(MF).
f(avez, [verb-(avoir, (pres, ind, 2, pl), MF)]) :- avoir_m_f(MF).
f(ont, [verb-(avoir, (pres, ind, 3, pl), MF), abbrev,
noun - (ontario, nil, s, proper, mass, [location])]) :- avoir_m_f(MF).

f(aurai, [verb-(avoir, (fut, ind, 1, s), MF)]) :- avoir_m_f(MF).
f(auras, [verb-(avoir, (fut, ind, 2, s), MF)]) :- avoir_m_f(MF).
f(aura, [verb-(avoir, (fut, ind, 3, s), MF)]) :- avoir_m_f(MF).
f(aurons, [verb-(avoir, (fut, ind, 1, pl), MF)]) :- avoir_m_f(MF).
f(aurez, [verb-(avoir, (fut, ind, 2, pl), MF)]) :- avoir_m_f(MF).
f(auront, [verb-(avoir, (fut, ind, 3, pl), MF)]) :- avoir_m_f(MF).

f(avais, [verb-(avoir, (imp, ind, (1,2), s), MF)]) :- avoir_m_f(MF).
f(avait, [verb-(avoir, (imp, ind, 3, s), MF)]) :- avoir_m_f(MF).
f(avions, [verb-(avoir, (imp, ind, 1, pl), MF)]) :- avoir_m_f(MF).
f(aviez, [verb-(avoir, (imp, ind, 2, pl), MF)]) :- avoir_m_f(MF).
f(avaient, [verb-(avoir, (imp, ind, 3, pl), MF)]) :- avoir_m_f(MF).

f(aurais, [verb-(avoir, (pres, cond, (1,2), s), MF)]) :- avoir_m_f(MF).
f(aurait, [verb-(avoir, (pres, cond, 3, s), MF)]) :- avoir_m_f(MF).
f(aurions, [verb-(avoir, (pres, cond, 1, pl), MF)]) :- avoir_m_f(MF).
f(auriez, [verb-(avoir, (pres, cond, 2, pl), MF)]) :- avoir_m_f(MF).
f(auraient, [verb-(avoir, (pres, cond, 3, pl), MF)]) :- avoir_m_f(MF).

f(aie, [verb-(avoir, (pres, subj, 1, s), MF)]) :- avoir_m_f(MF).
f(aies, [verb-(avoir, (pres, subj, 2, s), MF)]) :- avoir_m_f(MF).
f(ait, [verb-(avoir, (pres, subj, 3, s), MF)]) :- avoir_m_f(MF).
f(ayions, [verb-(avoir, (pres, subj, 1, pl), MF)]) :- avoir_m_f(MF).
f(ayiez, [verb-(avoir, (pres, subj, 2, pl), MF)]) :- avoir_m_f(MF).
f(aient, [verb-(avoir, (pres, subj, 3, pl), MF)]) :- avoir_m_f(MF).

f(eu, [verb-(avoir, (past-part, m, s), MF)]) :- avoir_m_f(MF).
f(eus, [verb-(avoir, (past-part, m, pl), MF)]) :- avoir_m_f(MF).
f(eue, [verb-(avoir, (past-part, f, s), MF)]) :- avoir_m_f(MF).
f(eues, [verb-(avoir, (past-part, f, pl), MF)]) :- avoir_m_f(MF).
f(ayant, [verb-(avoir, (pres-part, m, s), MF)]) :- avoir_m_f(MF).

f(être,[noun - (être, m, s, common, count, [1thing]), verb - (être, inf, MF)]) :-
    etre_m_f(MF).

f(suis, [verb-(être, (pres, ind, 1, s), MF)]) :- etre_m_f(MF).
f(es, [verb-(être, (pres, ind, 2, s), MF)]) :- etre_m_f(MF).
f(est, [verb-(être, (pres, ind, 3, s), MF)]) :- etre_m_f(MF).
f(sommes, [verb-(être, (pres, ind, 1, pl), MF),
    noun - (somme, f, pl, common, count, [quantity])]) :- etre_m_f(MF).
f(êtes, [verb-(être, (pres, ind, 2, pl), MF)]) :- etre_m_f(MF).
f(sont, [verb-(être, (pres, ind, 3, pl), MF)]) :- etre_m_f(MF).

f(serai, [verb-(être, (fut, ind, 1, s), MF)]) :- etre_m_f(MF).
f(seras, [verb-(être, (fut, ind, 2, s), MF)]) :- etre_m_f(MF).
f(sera, [verb-(être, (fut, ind, 3, s), MF)]) :- etre_m_f(MF).
```

```
f([serons, [verb-(être, (fut, ind, 1, pl), MF)]) :- etre_m_f(MF).
f([serez, [verb-(être, (fut, ind, 2, pl), MF)]) :- etre_m_f(MF).
f([seront, [verb-(être, (fut, ind, 3, pl), MF)]) :- etre_m_f(MF).

f([étais, [verb-(être, (imp, ind, (1,2), s), MF)]) :- etre_m_f(MF).
f([était, [verb-(être, (imp, ind, 3, s), MF)]) :- etre_m_f(MF).
f([étions, [verb-(être, (imp, ind, 1, pl), MF)]) :- etre_m_f(MF).
f([étiez, [verb-(être, (imp, ind, 2, pl), MF)]) :- etre_m_f(MF).
f([étaient, [verb-(être, (imp, ind, 3, pl), MF)]) :- etre_m_f(MF).

f([serais, [verb-(être, (pres, cond, (1,2), s), MF)]) :- etre_m_f(MF).
f([serait, [verb-(être, (pres, cond, 3, s), MF)]) :- etre_m_f(MF).
f([serions, [verb-(être, (pres, cond, 1, pl), MF)]) :- etre_m_f(MF).
f([seriez, [verb-(être, (pres, cond, 2, pl), MF)]) :- etre_m_f(MF).
f([seraient, [verb-(être, (pres, cond, 3, pl), MF)]) :- etre_m_f(MF).

f([sois, [verb-(être, (pres, subj, (1,2), s), MF)]) :- etre_m_f(MF).
f([soit, [verb-(être, (pres, subj, 3, s), MF)]) :- etre_m_f(MF).
f([soyions, [verb-(être, (pres, subj, 1, pl), MF)]) :- etre_m_f(MF).
f([soyiez, [verb-(être, (pres, subj, 2, pl), MF)]) :- etre_m_f(MF).
f([soient, [verb-(être, (pres, subj, 3, pl), MF)]) :- etre_m_f(MF).

f([été, [verb-(être, (past-part, m, s), MF),
   noun-(été, m, s, common, count, [time])]) :- etre_m_f(MF).
f([étant, [verb-(être, (pres-part, m, s), MF)]) :- etre_m_f(MF).

f([faire,[verb - (faire, inf, MF)]) :-
   faire_m_f(MF).

f([fais, [verb-(faire, (pres, ind, (1,2), s), MF)]) :- faire_m_f(MF).
f([faisons, [verb-(faire, (pres, ind, 1, pl), MF)]) :- faire_m_f(MF).
f([faites, [verb-(faire, (past-part, f, pl), MF),
   verb-(faire, (pres, ind, 2, pl), MF),
   adj - (fait, desc, f, pl, [attribute])]) :- faire_m_f(MF).
f([font, [verb-(faire, (pres, ind, 3, pl), MF)]) :- faire_m_f(MF).

f([ferai, [verb-(faire, (fut, ind, 1, s), MF)]) :- faire_m_f(MF).
f([feras, [verb-(faire, (fut, ind, 2, s), MF)]) :- faire_m_f(MF).
f([fera, [verb-(faire, (fut, ind, 3, s), MF)]) :- faire_m_f(MF).
f([ferons, [verb-(faire, (fut, ind, 1, pl), MF)]) :- faire_m_f(MF).
f([ferez, [verb-(faire, (fut, ind, 2, pl), MF)]) :- faire_m_f(MF).
f([feront, [verb-(faire, (fut, ind, 3, pl), MF)]) :- faire_m_f(MF).

f([faisais, [verb-(faire, (imp, ind, (1,2), s), MF)]) :- faire_m_f(MF).
f([faisait, [verb-(faire, (imp, ind, 3, s), MF)]) :- faire_m_f(MF).
f([faisions, [verb-(faire, (imp, ind, 1, pl), MF)]) :- faire_m_f(MF).
f([faisiez, [verb-(faire, (imp, ind, 2, pl), MF)]) :- faire_m_f(MF).
f([faisaient, [verb-(faire, (imp, ind, 3, pl), MF)]) :- faire_m_f(MF).

f([ferais, [verb-(faire, (pres, cond, (1,2), s), MF)]) :- faire_m_f(MF).
f([ferait, [verb-(faire, (pres, cond, 3, s), MF)]) :- faire_m_f(MF).
f([ferions, [verb-(faire, (pres, cond, 1, pl), MF)]) :- faire_m_f(MF).
f([feriez, [verb-(faire, (pres, cond, 2, pl), MF)]) :- faire_m_f(MF).
f([feraient, [verb-(faire, (pres, cond, 3, pl), MF)]) :- faire_m_f(MF).

f([fasse, [verb-(faire, (pres, subj, (1,3), s), MF)]) :- faire_m_f(MF).
f([fasse, [verb-(faire, (pres, subj, 2, s), MF)]) :- faire_m_f(MF).
f([fassions, [verb-(faire, (pres, subj, 1, pl), MF)]) :- faire_m_f(MF).
f([fassiez, [verb-(faire, (pres, subj, 2, pl), MF)]) :- faire_m_f(MF).
f([fassent, [verb-(faire, (pres, subj, 3, pl), MF)]) :- faire_m_f(MF).

f([fait, [verb-(faire, (pres, ind, 3, s), MF),
   verb-(faire, (past-part, m, s), MF),
```

```
    adj - (fait, desc, m, s, [attribute]),
    noun - (fait, m, s, common, count, [submatter])]) :- faire_m_f(MF).
f([faits, [verb - (faire, (past - part, m, pl), MF),
    adj - (fait, desc, m, pl, [attribute]),
    noun -(fait, m, pl, common, count, [submatter])]) :- faire_m_f(MF).
f([faisant, [verb-(faire, (pres-part, m, s), MF)]) :- faire_m_f(MF).

aller_m_f([motion - [agent - (r, [event,person,animal], nil),
instrument - (o, [artifact,animal], à),source - (o, [location], de),
destination - (r, [location], à)]]).

f([aller,[verb - (aller, inf, MF)]) :- aller_m_f(MF).

f([vais, [verb-(aller, (pres, ind, 1, s), MF)]) :- aller_m_f(MF).
f([vas, [verb-(aller, (pres, ind, 2, s), MF)]) :- aller_m_f(MF).
f([va, [verb-(aller, (pres, ind, 3, s), MF)]) :- aller_m_f(MF).
f([allons, [verb-(aller, (pres, ind, 1, pl), MF)]) :- aller_m_f(MF).
f([allez, [verb-(aller, (pres, ind, 2, pl), MF)]) :- aller_m_f(MF).
f([vont, [verb-(aller, (pres, ind, 3, pl), MF)]) :- aller_m_f(MF).

f([irai, [verb-(aller, (fut, ind, 1, s), MF)]) :- aller_m_f(MF).
f([iras, [verb-(aller, (fut, ind, 2, s), MF)]) :- aller_m_f(MF).
f([ira, [verb-(aller, (fut, ind, 3, s), MF)]) :- aller_m_f(MF).
f([irons, [verb-(aller, (fut, ind, 1, pl), MF)]) :- aller_m_f(MF).
f([irez, [verb-(aller, (fut, ind, 2, pl), MF)]) :- aller_m_f(MF).
f([iront, [verb-(aller, (fut, ind, 3, pl), MF)]) :- aller_m_f(MF).

f([allais, [verb-(aller, (imp, ind, (1,2), s), MF)]) :- aller_m_f(MF).
f([allait, [verb-(aller, (imp, ind, 3, s), MF)]) :- aller_m_f(MF).
f([allions, [verb-(aller, (imp, ind, 1, pl), MF)]) :- aller_m_f(MF).
f([alliez, [verb-(aller, (imp, ind, 2, pl), MF)]) :- aller_m_f(MF).
f([allaient, [verb-(aller, (imp, ind, 3, pl), MF)]) :- aller_m_f(MF).

f([irais, [verb-(aller, (pres, cond, (1,2), s), MF)]) :- aller_m_f(MF).
f([irait, [verb-(aller, (pres, cond, 3, s), MF)]) :- aller_m_f(MF).
f([irions, [verb-(aller, (pres, cond, 1, pl), MF)]) :- aller_m_f(MF).
f([iriez, [verb-(aller, (pres, cond, 2, pl), MF)]) :- aller_m_f(MF).
f([iraient, [verb-(aller, (pres, cond, 3, pl), MF)]) :- aller_m_f(MF).

f([aille, [verb-(aller, (pres, subj, (1,3), s), MF)]) :- aller_m_f(MF).
f([ailles, [verb-(aller, (pres, subj, 2, s), MF)]) :- aller_m_f(MF).
f([allions, [verb-(aller, (pres, subj, 1, pl), MF)]) :- aller_m_f(MF).
f([alliez, [verb-(aller, (pres, subj, 2, pl), MF)]) :- aller_m_f(MF).
f([aillent, [verb-(aller, (pres, subj, 3, pl), MF)]) :- aller_m_f(MF).

f([allé, [verb-(aller, (past-part, m, s), MF)]) :- aller_m_f(MF).
f([allant, [verb-(aller, (pres-part, m, s), MF)]) :- aller_m_f(MF).

modal_m_f([volition - [agent - (r, [person], nil),
    affected - (r, [something], nil)]]).

f([vouloir,[verb - (vouloir, inf, MF)]) :- modal_m_f(MF).

f([veux, [verb-(vouloir, (pres, ind, (1,2), s), MF)]) :- modal_m_f(MF).
f([veut, [verb-(vouloir, (pres, ind, 3, s), MF)]) :- modal_m_f(MF).
f([voulons, [verb-(vouloir, (pres, ind, 1, pl), MF)]) :- modal_m_f(MF).
f([voulez, [verb-(vouloir, (pres, ind, 2, pl), MF)]) :- modal_m_f(MF).
f([veulent, [verb-(vouloir, (pres, ind, 3, pl), MF)]) :- modal_m_f(MF).

f([voudrai, [verb-(vouloir, (fut, ind, 1, s), MF)]) :- modal_m_f(MF).
f([voudras, [verb-(vouloir, (fut, ind, 2, s), MF)]) :- modal_m_f(MF).
f([voudra, [verb-(vouloir, (fut, ind, 3, s), MF)]) :- modal_m_f(MF).
f([voudrons, [verb-(vouloir, (fut, ind, 1, pl), MF)]) :- modal_m_f(MF).
```

```
f(voudrez, [verb-(vouloir, (fut, ind, 2, pl), MF)]) :- modal_m_f(MF).
f(voudront, [verb-(vouloir, (fut, ind, 3, pl), MF)]) :- modal_m_f(MF).

f(voulais, [verb-(vouloir, (imp, ind, (1,2), s), MF)]) :- modal_m_f(MF).
f(voulait, [verb-(vouloir, (imp, ind, 3, s), MF)]) :- modal_m_f(MF).
f(voulions, [verb-(vouloir, (imp, ind, 1, pl), MF)]) :- modal_m_f(MF).
f(vouliez, [verb-(vouloir, (imp, ind, 2, pl), MF)]) :- modal_m_f(MF).
f(voulaient, [verb-(vouloir, (imp, ind, 3, pl), MF)]) :- modal_m_f(MF).

f(voudrais, [verb-(vouloir, (pres, cond, (1,2), s), MF)]) :- modal_m_f(MF).
f(voudrait, [verb-(vouloir, (pres, cond, 3, s), MF)]) :- modal_m_f(MF).
f(voudrions, [verb-(vouloir, (pres, cond, 1, pl), MF)]) :- modal_m_f(MF).
f(voudriez, [verb-(vouloir, (pres, cond, 2, pl), MF)]) :- modal_m_f(MF).
f(voudraient, [verb-(vouloir, (pres, cond, 3, pl), MF)]) :- modal_m_f(MF).

f(veuille, [verb-(vouloir, (pres, subj, (1,3), s), MF)]) :- modal_m_f(MF).
f(veuilles, [verb-(vouloir, (pres, subj, 2, s), MF)]) :- modal_m_f(MF).
f(voulions, [verb-(vouloir, (pres, subj, 1, pl), MF)]) :- modal_m_f(MF).
f(vouliez, [verb-(vouloir, (pres, subj, 2, pl), MF)]) :- modal_m_f(MF).
f(veuillent, [verb-(vouloir, (pres, subj, 3, pl), MF)]) :- modal_m_f(MF).

f(voulu, [verb-(vouloir, (past-part, m, s), MF)]) :- modal_m_f(MF).
f(voulant, [verb-(vouloir, (pres-part, m, s), MF)]) :- modal_m_f(MF).


f(pouvoir,[verb - (pouvoir, inf, MF)]) :- modal_m_f(MF).

f(peux, [verb-(pouvoir, (pres, ind, (1,2), s), MF)]) :- modal_m_f(MF).
f(puis, [verb-(pouvoir, (pres, ind, 1, s), MF),
    adv-(puis, time)]) :- modal_m_f(MF).
f(peut, [verb-(pouvoir, (pres, ind, 3, s), MF)]) :- modal_m_f(MF).
f(pouvons, [verb-(pouvoir, (pres, ind, 1, pl), MF)]) :- modal_m_f(MF).
f(pouvez, [verb-(pouvoir, (pres, ind, 2, pl), MF)]) :- modal_m_f(MF).
f(peuvent, [verb-(pouvoir, (pres, ind, 3, pl), MF)]) :- modal_m_f(MF).

f(pourrai, [verb-(pouvoir, (fut, ind, 1, s), MF)]) :- modal_m_f(MF).
f(pourras, [verb-(pouvoir, (fut, ind, 2, s), MF)]) :- modal_m_f(MF).
f(pourra, [verb-(pouvoir, (fut, ind, 3, s), MF)]) :- modal_m_f(MF).
f(pourrons, [verb-(pouvoir, (fut, ind, 1, pl), MF)]) :- modal_m_f(MF).
f(pourrez, [verb-(pouvoir, (fut, ind, 2, pl), MF)]) :- modal_m_f(MF).
f(pourront, [verb-(pouvoir, (fut, ind, 3, pl), MF)]) :- modal_m_f(MF).

f(pouvais, [verb-(pouvoir, (imp, ind, (1,2), s), MF)]) :- modal_m_f(MF).
f(pouvait, [verb-(pouvoir, (imp, ind, 3, s), MF)]) :- modal_m_f(MF).
f(pouvions, [verb-(pouvoir, (imp, ind, 1, pl), MF)]) :- modal_m_f(MF).
f(pouviez, [verb-(pouvoir, (imp, ind, 2, pl), MF)]) :- modal_m_f(MF).
f(pouvaient, [verb-(pouvoir, (imp, ind, 3, pl), MF)]) :- modal_m_f(MF).

f(pourrais, [verb-(pouvoir, (pres, cond, (1,2), s), MF)]) :- modal_m_f(MF).
f(pourrait, [verb-(pouvoir, (pres, cond, 3, s), MF)]) :- modal_m_f(MF).
f(pourrions, [verb-(pouvoir, (pres, cond, 1, pl), MF)]) :- modal_m_f(MF).
f(pourriez, [verb-(pouvoir, (pres, cond, 2, pl), MF)]) :- modal_m_f(MF).
f(pourraient, [verb-(pouvoir, (pres, cond, 3, pl), MF)]) :- modal_m_f(MF).

f(puisse, [verb-(pouvoir, (pres, subj, (1,3), s), MF)]) :- modal_m_f(MF).
f(puisses, [verb-(pouvoir, (pres, subj, 2, s), MF)]) :- modal_m_f(MF).
f(puissions, [verb-(pouvoir, (pres, subj, 1, pl), MF)]) :- modal_m_f(MF).
f(puissiez, [verb-(pouvoir, (pres, subj, 2, pl), MF)]) :- modal_m_f(MF).
f(puissent, [verb-(pouvoir, (pres, subj, 3, pl), MF)]) :- modal_m_f(MF).

f(pu, [verb-(pouvoir, (past-part, m, s), MF)]) :- modal_m_f(MF).
f(pouvant, [verb-(pouvoir, (pres-part, m, s), MF)]) :- modal_m_f(MF).
```

```
f(savoir,[verb - (savoir, inf, MF)]) :- modal_m_f(MF).

f(sais, [verb-(savoir, (pres, ind, (1,2), s), MF)]) :- modal_m_f(MF).
f(sait, [verb-(savoir, (pres, ind, 3, s), MF)]) :- modal_m_f(MF).
f(savons, [verb-(savoir, (pres, ind, 1, pl), MF)]) :- modal_m_f(MF).
f(savez, [verb-(savoir, (pres, ind, 2, pl), MF)]) :- modal_m_f(MF).
f(savent, [verb-(savoir, (pres, ind, 3, pl), MF)]) :- modal_m_f(MF).

f(saurai, [verb-(savoir, (fut, ind, 1, s), MF)]) :- modal_m_f(MF).
f(sauras, [verb-(savoir, (fut, ind, 2, s), MF)]) :- modal_m_f(MF).
f(saura, [verb-(savoir, (fut, ind, 3, s), MF)]) :- modal_m_f(MF).
f(saurons, [verb-(savoir, (fut, ind, 1, pl), MF)]) :- modal_m_f(MF).
f(saurez, [verb-(savoir, (fut, ind, 2, pl,, MF)]) :- modal_m_f(MF).
f(sauront, [verb-(savoir, (fut, ind, 3, pl), MF)]) :- modal_m_f(MF).

f(saurais, [verb-(savoir, (imp, ind, (1,2), s), MF)]) :- modal_m_f(MF).
f(saurait, [verb-(savoir, (imp, ind, 3, s), MF)]) :- modal_m_f(MF).
f(saurions, [verb-(savoir, (imp, ind, 1, pl), MF)]) :- modal_m_f(MF).
f(sauriez, [verb-(savoir, (imp, ind, 2, pl), MF)]) :- modal_m_f(MF).
f(sauraient, [verb-(savoir, (imp, ind, 3, pl), MF)]) :- modal_m_f(MF).

f(saurais, [verb-(savoir, (pres, cond, (1,2), s), MF)]) :- modal_m_f(MF).
f(saurait, [verb-(savoir, (pres, cond, 3, s), MF)]) :- modal_m_f(MF).
f(saurions, [verb-(savoir, (pres, cond, 1, pl), MF)]) :- modal_m_f(MF).
f(sauriez, [verb-(savoir, (pres, cond, 2, pl), MF)]) :- modal_m_f(MF).
f(sauraient, [verb-(savoir, (pres, cond, 3, pl), MF)]) :- modal_m_f(MF).

f(sache, [verb-(savoir, (pres, subj, (1,3), s), MF)]) :- modal_m_f(MF).
f(saches, [verb-(savoir, (pres, subj, 2, s), MF)]) :- modal_m_f(MF).
f(sachions, [verb-(savoir, (pres, subj, 1, pl), MF)]) :- modal_m_f(MF).
f(sachiez, [verb-(savoir, (pres, subj, 2, pl), MF)]) :- modal_m_f(MF).
f(sachent, [verb-(savoir, (pres, subj, 3, pl), MF)]) :- modal_m_f(MF).

f(su, [verb-(savoir, (past-part, m, s), MF)]) :- modal_m_f(MF).
f(sachant, [verb-(savoir, (pres-part, m, s), MF)]) :- modal_m_f(MF).
```

```
/*************************************************************************
*************************************************************************
FRENCH DICTIONARY CONTAINING ENTRIES FOR OPEN CATEGORIES
© 1987, 1988 Victor Loewen
*************************************************************************
*************************************************************************/

/*
The following example illustrates how entries for verbs may contain individual frames
for the individual meanings of verbs. For example, "servir" has three meanings and therefore
requires three frames.

to serve something to someone (meaning = transfer)
    [agent - (r, [human], nil),
    instrument - (o, [artifact], avec),
    beneficiary - (r, [person], à),
    affected - (r, [artifact], nil)]

to serve someone (meaning = assistance)
    [agent - (r, [person], nil),
    affected - (r, [person], nil)]

to serve as something (meaning = occupation)
    [affected - (r, [something], nil),
    attribute - (r, [something], comme)]
*/

f(servir,[verb -(servir,inf,
    [transfer -
        [agent - (r, [human], nil),
        instrument - (o, [artifact], avec),
        beneficiary - (r, [person], à),
        affected - (r, [artifact], nil)],
    assistance -
        [agent - (r, [person], nil),
        affected - (r, [person], nil)],
    occupation -
        [affected - (r, [something], nil),
        attribute - (r, [something], comme)]])]).
f(remercier,[verb - (remercier, inf, [mentinteraction - [agent - (r, [person], nil),
    affected - (r, [person], nil)]])]).
f(lettre,[noun - (lettre, f, s, common, count, [nonselfmoving, submatter])]).
f(heureux,[adj - (heureux, desc, m, (s, pl), [attribute])]).
f(complet,[adj - (complet, desc, m, s, [attribute])]).
f(bel,[adj - (bel, desc, m, s, [attribute])]).
f(long,[adj - (long, desc, m, s, [attribute])]).
f(joint,[noun - (joint, m, s, common, count, [ttrelation]),
    verb - (joindre, (past - part, m, s), [contact -
        [agent - (r, [posprocess,structure,sentient], nil),
        instrument - (o, [artifact], par),
        affected - (r, [pthing, mthing], nil),
        destination - (r, [mthing,pthing], à)])]]).
f(joindre,[verb - (joindre, inf, [contact -
        [agent - (r, [posprocess,structure,sentient], nil),
        instrument - (o, [artifact], par),
        affected - (r, [pthing, mthing], nil),
        destination - (r, [mthing,pthing], à)])]]).
f(janvier,[noun - (janvier, m, s, proper, count, [time])]).
f(pièce,[noun - (pièce, f, s, common, count, [pspace,unit,nonselfmoving])]).
```

```
f(monsieur,[noun - (monsieur, m, s, proper, count, [human])]).
f(madame,[noun - (madame, f, s, proper, count, [name,human])]).
f(adresser,[verb - (adresser, inf,
    [transfer - [agent - (r, [person], nil), beneficiary - (r, [sentient],a),
    instrument - (o, [artifact], dans), affected - (r, [artifact], nil),
    source - (o, [pspace], de), destination-(o,[pspace],à)],
    mentinteraction - [agent - (r, [person,artifact], nil),beneficiary - (r, [person], nil),
    affected-(o,[mthing, fthing],nil)]]]]).
f(adresse,[verb - (adresser, (pres, (ind, subj), (1, 3), s),
    [transfer - [agent - (r, [person], nil), beneficiary - (r, [sentient],à),
    instrument - (o, [artifact], dans), affected - (r, [artifact], nil),
    source - (o, [pspace], de), destination-(o,[pspace],à)],
    mentinteraction - [agent - (r, [person, artifact], nil),beneficiary - (r, [person], nil),
    affected-(r,[mthing, fthing],nil)]]],
    noun-(adresse, f, s, common, count, [pspace, name])]]).
f(documentation,[noun - (documentation, f, s, common, mass, [artifact])]).
f(demander,[verb - (demander, inf, [mentinteraction - [agent - (r, [person], nil),
    instrument - (o, [artifact], nil),affected - (r, [person], à)]]]]).
f(parcourir,[verb - (parcourir, inf,
    [mentinteraction - [agent - (r, [human], nil),affected - (r, [artifact], nil)],
    selfdisplacement - [agent - (r, [person,selfmoving], nil),source - (o, [pspace], nil),
    destination - (r, [location], à)]]]]).
f(dépliant,[noun - (dépliant, m, s, common, count, [artifact]),verb - (déplier,
    (pres - part, m, s), [process - [agent - (r, [person], nil),
    instrument - (o, [artifact], avec),affected - (r, [artifact], nil)]]]]).
f(déplier,[verb - (déplier, (pres - part, m, pl), [process - [agent - (r, [person], nil),
    instrument - (o, [artifact], avec),affected - (r, [artifact], nil)]]]]).
f(constater,[verb - (constater, inf, [mentinteraction - [agent - (r, [person], nil),
    instrument - (o, [artifact], nil),affected - (r, [mthing], nil)]]]]).
f(fabriquer,[verb - (fabriquer, inf, [posprocess - [agent - (r, [sentient], nil),
    instrument - (o, [nlthing], avec),beneficiary - (o, [sentient], pour),
    result - (r, [artifact,mthing], nil)]]]]).
f(gamme,[noun - (gamme, f, s, common, mass, [structure])]).
f(bibliothèque,[noun - (bibliothèque, f, s, common, count, [artifact,pspace,group])]).
f(rayonnage,[noun - (rayonnage, m, s, common, count, [artifact])]).
f(chose,[noun - (chose, f, s, common, count, _)]).
f(document,[noun - (document, m, s, common, count, [artifact])]).
f(facile,[adj - (facile, desc, (m, f), s, [quality])]).
f(choisir,[verb - (choisir, inf, [mentinteraction - [agent - (r, [person,animal], nil),
    instrument - (o, [artifact,motion], avec),beneficiary - (o, [person,cause], pour),
        affected - (r, [something], nil)]]]]).
f(modèle,[noun - (modèle, m, s, common, count, [artifact,structure,human])]).
f(fonction,[noun - (fonction, f, s, common, count, [occupation,relation])]).
f(besoin,[noun - (besoin, m, s, common, count, [condition,fthing])]).
f(illustrer,[verb - (illustrer, inf, [activity - [agent - (r, [nstate,mthing,pthing], nil),
    instrument - (o, [artifact], par),beneficiary - (o, [sentient], pour),
    affected - (r, [something], nil)]]]]).
f(livrable,[adj - (livrable, desc, (m, f), s, [quality])]).
f(délai,[noun - (délai, m, s, common, count, [time])]).
f(jour,[noun - (jour, m, s, common, count, [time])]).
f(assurer,[verb - (assurer, inf, [physinteraction - [agent - (r, [person], nil),
    instrument - (o, [artifact], par),beneficiary - (o, [sentient,nstate], à),
    affected - (r, [something], nil)]]]]).
f(transport,[noun - (transport, m, s, common, mass, [transfer])]).
f(installation,[noun - (installation, f, s, common, count,
[posprocess,transfer,stationary])]).
f(mesure,[noun - (mesure, f, s, common, count, [measurement])]).
f(garantir,[verb - (garantir, inf, [mentinteraction - [agent - (r, [person], nil),
    instrument - (o, [artifact], par),beneficiary - (o, [person], à),
    affected - (r, [event,mthing], nil)]]]]).
f(entier,[adj - (entier, desc, m, s, [derivation])]).
f(satisfaction,[noun - (satisfaction, f, s, common, mass, [quality])]).
f(prier,[verb - (prier, inf, [mentinteraction - [agent - (r, [person], nil),
```

```
    affected - (r, [person], nil)]]]]).
f(agréer,[verb - (agréer, inf, [mentinteraction - [agent - (r, [person], nil),
    beneficiary - (o, [person], pour),affected - (r, [something], nil)]]]]).
f(expression,[noun - (expression, f, s, common, count, [submatter,artifact])]).
f(sentiment,[noun - (sentiment, m, s, common, count, [fthing])]).
f(dévouer,[verb - (dévouer, inf, [mentinteraction - [agent - (r, [person], nil),
    beneficiary - (o, [something], à),affected - (r, [person], nil)]]]]).
f(espoir,[noun - (espoir, m, s, common, count, [submatter])]).
f(prochainement,[adv - (prochainement, time)]).
f(ardec,[noun - (ardec, m, s, proper, count, [name])]).
f(incorporer,[verb - (incorporer, inf, [posprocess - [agent - (r, [person], nil),
    instrument - (o, [artifact], avec),affected - (r, [group], nil)]]]]).
f(meuble,[noun - (meuble, m, s, common, count, [nonselfmoving])]).
f(bureau,[noun - (bureau, m, s, common, count, [group,nonselfmoving])]).
f(avenue,[noun - (avenue, f, s, common, count, [location])]).
f(christophe,[noun - (christophe, m, s, proper, count, [name])]).
f(colomb,[noun - (colomb, m, s, proper, count, [name,bird])]).
f(montréal,[noun - (montréal, m, s, proper, mass, [name,location])]).
f(référence,[noun - (référence, f, s, common, count, [ttrelation])]).
f(lundi,[noun - (lundi, m, s, proper, count, [time])]).
f(recommander,[verb - (recommender, inf, [mentinteraction - [agent - (r, [human], nil),
    beneficiary - (r, [person], à),affected - (r, [mthing,event], nil)]]),
    verb -(recommander,inf, [posprocess - [agent - (r, [person], nil),
    affected - (r, [artifact], nil)]]]]).
f(septembre,[noun - (septembre, m, s, common, count, [time])]).
f(lavallée,[noun - (lavallée, f, s, proper, count, [name,human])]).
f(fils,[noun - (fils, m, (s, pl), common, count, [person,ttrelation])]).
f(rue,[noun - (rue, f, s, common, count, [location])]).
f(école,[noun - (école, f, s, common, count, [location,group])]).
f(saint,[adj - (saint, desc, m, s, [attribute]),
    noun - (saint, m, s, common, count, [human])]).
f(jérôme,[noun - (jérôme, m, s, proper, count, [human])]).
f(québec,[noun - (québec, m, s, proper, mass, [location])]).
f(attention,[noun - (attention, f, s, common, mass, [mentinteraction])]).
f(dupré,[noun - (dupré, m, s, proper, count, [person])]).
f(objet,[noun - (objet, m, s, common, count, [pthing])]).
f(cher,[adj - (cher, desc, m, s, [attribute])]).
f(directeur,[noun - (directeur, m, s, common, count, [occupation,human])]).
f(commercial,[adj - (commercial, desc, m, s, [attribute])]).
f(jean,[noun - (jean, m, s, proper, count, [human,name])]).
f(lavoie,[noun - (lavoie, m, s, proper, count, [human,name])]).
f(légaré,[noun - (légaré, m, s, proper, count, [name,human])]).
f(là,[adv - (là, location)]).
f(limiter,[verb - (limiter, inf, [mentinteraction - [agent - (r, [human,event], nil),
    affected - (r, [human,event], nil)],physinteraction - [agent - (r, [artifact], nil),
    affected - (r, [artifact,event], nil),result - (o, [event,pspace,ptime], à)]]]).
f(compagnie,[noun - (compagnie, f, s, common, count, [accompaniment,group])]).
f(inc,[abbrev,verb - (incorporer, (past - part, m, s), [posprocess -
    [agent - (r, [person], nil),
    instrument - (o, [artifact], avec),affected - (r, [group], nil)]]]]).
f(av,[abbrev,noun - (avenue, f, s, common, count, [location])]).
f(qué,[abbrev,noun - (québec, m, s, proper, mass, [location])]).
f(ltée,[abbrev,verb - (limiter, (past - part, f, s), [mentinteraction - [
    agent - (r, [human,event], nil),affected - (r, [human,event], nil)],
    physinteraction - [agent - (r, [artifact], nil),affected - (r, [artifact,event], nil),
    result - (o, [event,pspace,ptime], à)]]]).
f(cie,[abbrev,noun - (compagnie, f, s, common, count, [accompaniment,group])]).
f(réf,[abbrev,noun - (référence, f, s, common, count, [ttrelation])]).
f(bonjour,[interjection - bonjour]).
f(chemin,[noun - (chemin, m, s, common, count, [pspace,distance])]).
f(st,[abbrev,adj - (saint, desc, m, s, [attribute]),
    noun - (saint, m, s, common, count, [human])]).
f(milton,[noun - (milton, (m, f), s, proper, count, [name])]).
```

```
f(paul,[noun - (paul, m, s, proper, count, [name])]).
f(messieurs,[noun - (messieurs, m, pl, proper, count, [person,name])]).
f(mesdames,[noun - (mesdames, f, pl, proper, count, [person,name])]).
f(ami,[noun - (ami, m, s, common, count, [person,ttrelation])]).
f(marie,[noun - (marie, f, s, proper, count, [name,person])]).
f(tableau,[noun - (tableau, m, s, common, count, [structure,artifact])]).
f(livre,[noun - (livre, m, s, common, count, [artifact])]).
f(copie,[noun - (copie, f, s, common, count, [artifact,ttrelation])]).
f(chant,[noun - (chant, m, s, common, count, [submatter]),verb - (chanter, inf,
    [physinteraction - [agent - (r, [person], nil),affected - (o, [submatter], nil),
    instrument - (o, [artifact], avec),beneficiary - (o, [person,group], à)]]]]).
f(chanter,[verb - (chanter, inf,
    [physinteraction - [agent - (r, [person], nil),affected - (o, [submatter], nil),
    instrument - (o, [artifact], avec),beneficiary - (o, [person,group], à)]]]]).
f(plus,[adv - (plus, negation)]).
f(maintenant,[verb - (maintenir, (pres - part, m, s), [posprocess -
    [agent - (r, [event,person], nil),instrument - (o, [artifact], avec),
    beneficiary - (o, [person,event], à),affected - (r, [something], nil)]]],
    adv - (maintenant, time)]).
f(point,[noun - (point, m, s, common, count, [pspace,ptime,condition]),
    adv - (point, negation)]).
f(jamais,[adv - (jamais, negation)]).
f(guère,[adv - (guère, negation)]).
f(rien,[noun - (rien, m, s, common, mass, [unit]),adv - (rien, negation),
    pro - (rien, indef, 3, s, m, [unit])]).
f(voir,[verb - (voir, inf, [physinteraction - [agent - (r, [human,animal,person], nil),
    instrument - (o, [artifact], avec),affected - (o, [pthing], nil)],
    mentinteraction - [agent - (r, [person], nil),affected - (o, [mthing], nil)]]]]).
f(personne,[noun - (personne, f, s, common, count, [person]),adv - (personne, negation)]).
f(écrire,[verb - (écrire, inf, [physinteraction - [agent - (r, [person], nil),
    instrument - (o, [artifact], avec),beneficiary - (o, [person], à),
    affected - (o, [submatter,nonselfmoving], nil)]]]]).
f(écrit,[noun - (écrit, m, s, common, count, [artifact]),
    verb - (écrire, (pres, ind, 3, s), [physinteraction - [agent - (r, [person], nil),
    instrument - (o, [artifact], avec),beneficiary - (o, [person], à),
    affected - (o, [submatter,nonselfmoving], nil)]]),
    verb - (écrire, (past-part, m, s), [physinteraction - [agent - (r, [person], nil),
    instrument - (o, [artifact], avec),beneficiary - (o, [person], à),
    affected - (o, [submatter,nonselfmoving], nil)]]]]).
f(dire,[verb - (dire, inf, [physinteraction - [agent - (r, [person], nil),
    instrument - (o, [artifact], avec),beneficiary - (o, [person,animal,pthing], à),
    affected - (r, [mthing], nil)]]]]).
f(dit,[verb - (dire, (pres, ind, 3, s), [physinteraction - [agent - (r, [person], nil),
    instrument - (o, [artifact], avec),beneficiary - (o, [person,animal,pthing], à),
    affected - (r, [mthing], nil)]]],
    verb - (dire, (past-part, m, s), [physinteraction - [agent - (r, [person], nil),
    instrument - (o, [artifact], avec),beneficiary - (o, [person,animal,pthing], à),
    affected - (r, [mthing], nil)]]],
    noun - (dit, m, s, common, mass, [submatter])]).
f(inviter,[verb - (inviter, inf, [mentinteraction - [agent - (r, [event, person], nil),
    affected - (r, [person,submatter], nil)]]]]).
f(an,[noun - (an, m, s, common, count, [time])]).
f(aucun,[adj - (aucun, indef, m, s, [quantity]),adv - (aucun, negation)]).
f(homme,[noun - (homme, m, s, common, count, [person,pspace])]).
f(chez,[prep - chez]).
f(cc, [string-cc]).
f(jeanne,[noun - (jeanne, f, s, proper, count, [person,name])]).
f(untel,[noun - (untel, m, s, proper, count, [person,name])]).
f(recevoir,[verb - (recevoir, inf, [physinteraction - [agent - (r, [person], nil),
    affected - (r, [animal,plant,natobject,energy,artifact], nil),
    source - (o, [pspace,person], de)],
    mentinteraction - [agent - (r, [event,person], nil),affected - (r, [mthing], nil),
    source - (o, [event], de)]]]]).
```

```
f(juin,[noun - (juin, m, s, proper, count, [time,name])]).
f(devenir,[verb - (devenir, inf, [
   state - [affected - (r, [person,event,mthing,artifact], nil),
   attribute - (r, [something], nil)],
   posprocess - [affected - (r, [person,event,mthing,artifact], nil),
   result - (r, [something], nil)]])]).
f(base,[noun - (base, f, s, common, count, [pspace,nonselfmoving])]).
f(payer,[verb - (payer, inf, [transfer - [agent - (r, [person], nil),instrument - (o,
[artifact], nil),beneficiary - (o, [person], à),affected - (o, [unit,artifact], nil)]])]).
f(travail,[noun - (travail, m, s, common, count, [posprocess,occupation])]).
f(lire,[verb - (lire, inf, [mentinteraction - [agent - (r, [person], nil),
   instrument - (o, [artifact], avec),affected - (o, [submatter,construct,artifact],
nil)]])]).
f(victor,[noun - (victor, m, s, proper, count, [name,person])]).
f(loewen,[noun - (loewen, m, s, proper, count, [name,person])]).
f(navan,[noun - (navan, m, s, proper, count, [pspace,location,name])]).
f(traduction,[noun - (traduction, f, s, common, count, [posprocess,mentinteraction])]).
f(automatique,[adj - (automatique, desc, (m, f), s, [attribute])]).
f(ontario,[noun - (ontario, nil, s, proper, mass, [location])]).
f(patient,[adj - (patient, desc, m, s, [human,quality]),noun - (patient, m, s, common, count,
[human])]).
```

```
/**************************************************************************
**************************************************************************
ENGLISH DICTIONARY
**************************************************************************
**************************************************************************/

e(thank,[verb-(thank,inf,[mentinteraction-[agent-(r,[person],nil),affected-
(r,[sentient],nil),beneficiary-(o,[event,person],for)]]]).
e(letter,[noun-(letter,nil,s,common,count,[nonselfmoving,submatter])]).
e(address,[noun-(address,nil,s,common,count,[pspace,name]),
    verb-(address,inf,[mentinteraction-[agent-(r,[person],nil),
    beneficiary-(r,[person],nil),affected-(o,[mthing,fthing],nil)],
       transfer-[agent-(r,[person],nil),beneficiary-(r,[sentient],to),
    instrument-(o,[artifact],in),affected-(r,[artifact],nil),source-(o,[pspace],from),
    destination-(o,[pspace],to)]]]]).
e(your,[adj-(your,poss,nil,(s,pl),[ttrelation])]).
e(happy,[adj-(happy,desc,nil,(s,pl),[attribute])]).
e(be,[verb-(be,inf,[state-[beneficiary-(o,[person,event],for),
    affected-(r,[something],nil),attribute-(r,[something],nil)]]]).
e(send,[verb-(send,inf,[transfer-[agent-(r,[person],nil),instrument-(o,[artifact],in),
    beneficiary-(r,[sentient],to),affected-(r,[artifact],nil),source-(o,[pspace],from),
    destination-(o,[pspace],to)]]]]).
e(documentation,[noun-(documentation,nil,s,common,mass,[nonselfmoving,submatter])]).
e(document,[noun -(document,nil, s, common, count, [artifact])]).
e(one,[adj-(one,cardinal,nil,s,[quantity]),noun-(one,nil,s,common,count,[measurement])]).
e(a,[adj-(a,indef,nil,s,[quantity],string-a),art-(a,indef,nil,s)]).
e(us,[pro-(us,refl,1,pl,(m,f),[person])]).
e('I',[pro-('I',pers,1,s,(m,f),[human])]).
e(you,[pro-(you,pers,2,(s,pl),(m,f),[person]),pro-(you,refl,2,(s,pl),(m,f),[person])]).
e(yourself,[pro-(yourself,intens,2,(s,pl),(m,f),[person])]).
e(we,[pro-(we,pers,1,pl,(m,f),[person])]).
e(me,[pro-(me,refl,1,s,(m,f),[person])]).
e(myself,[pro-(myself,refl,1,s,(m,f),[person])]).
e(the,[art-(the,def,nil,s,pl)]).
e(request,[noun-(request,nil,s,common,count,[submatter]),verb-(request,inf,
    [mentinteraction-[agent-(r,[person],nil),instrument-(o,[artifact],with),
    affected-(r,[person],nil)]]]]).
e(and,[conj-(and,coord)]).
e(in,[prep-in]).
e(by,[prep-by]).
e(while,[prep-while]).
e(have,[verb-(have,inf,[possession-[agent-(r,[something],nil),
    affected-(r,[something],nil),beneficiary-(o,[event,person],for)]]]]).
e(he,[pro-(he,pers,3,s,(m,f),[person])]).
e(it,[pro-(it,rel,3,s,(m,f),_)]).
e(three,[adj-(three,cardinal,nil,pl,[quantity])]).
e(make,[verb-(make,inf,[posprocess-[agent-(r,[event,lthing],nil),instrument-
(o,[artifact],with),
    beneficiary-(o,[person,event],for),result-(r,[artifact],nil),source-
(o,[nlthing],from)]]]]).
e(do,[verb-(do,inf,[posprocess-[agent-(r,[person],nil),instrument-(o,[artifact],with),
    beneficiary-(o,[person],for),affected-(r,[event],nil)]]]]).
e(much,[adj-(much,indef,nil,pl,[quantity]),noun-(much,nil,(s,pl),common,mass,[measurement]),
    adv-(much,quantity)]).
e(many,[adj-(many,indef,nil,(s,pl),[quantity]),noun-(many,nil,pl,common,mass,[quantity]),
    pro-(many,indef,3,(s,pl),(m,f),[person])]).
e(their,[adj-(their,poss,nil,(s,pl),[ttrelation])]).
e(them,[pro-(them,rel,3,pl,(m,f),[person,animal]),pro-(them,refl,3,pl,(m,f),[human])]).
e(they,[pro-(they,pers,3,pl,(m,f),[person])]).
e(five,[adj-(five,cardinal,nil,pl,[quantity])]).
e(to,[prep-to]).
```

```
e(sir,[noun-(sir,nil,s,proper,count,[human])]).
e(mr,[noun-(mr,nil,s,proper,count,[human])]).
e(manufacture,[verb-(manufacture,inf,[posprocess-[agent-(r,[person],nil),
    instrument-(o,[artifact],with),beneficiary-(o,[event,lthing],for),result-
(r,[artifact],nil),
    source-(o,[nlthing],from)]])]).
e(range,[noun-(range,nil,s,common,count,[series])]).
e(complete,[adj-(complete,desc,nil,(s,pl),[attribute]),verb-(complete,inf,[posprocess-[agent-
(r,[person],nil),
    beneficiary-(o,[person,event],for),affected-(r,[artifact],nil)]])]).
e(library,[noun-(library,nil,s,common,count,[artifact,pspace])]).
e(shelving,[noun-(shelving,nil,s,common,mass,[artifact]),verb-(shelve,inf,[posprocess-
    [agent-(r,[human],nil),beneficiary-(o,[person],for),affected-
(r,[nonselfmoving,artifact],nil)]])]).
e(note,[noun-(note,nil,s,common,count,[artifact,submatter]),verb-(note,inf,
    [mentinteraction-[agent-(r,[human],nil),affected-(r,[mthing,event],nil),
    source-(o,[artifact],from)]])].
e(therefore,[conj-(therefore,sub)]).
e(easy,[adj-(easy,desc,nil,(s,pl),[quality])]).
e(choose,[verb-(choose,inf,[mentinteraction-[agent-(r,[person,animal],nil),
    instrument-(o,[artifact,motion],with),beneficiary-(o,[person,cause],for),affected-
(r,[something],nil),
    source-(o,[submatter],from)]])].
e(model,[adj-(model,desc,nil,(s,pl),[quality]),noun-
(model,nil,s,common,count,[person,structure,artifact]),
    verb-(model,inf,[occupation-[agent-(r,[human],nil),beneficiary-(o,[person],for),
    affected-(r,[submatter,artifact],nil)]])].
e(function,[noun-(function,nil,s,common,count,[relation,occupation]),verb-(function,
    inf,[nstate-[agent-(r,[artifact,submatter],nil),beneficiary-(o,[event,person],for)]])]).
e(need,[noun-(need,nil,s,common,count,[condition,fthing]),verb-(need,inf,
    [emotinteraction-[agent-(r,[person,animal],nil),affected-(r,[something],nil),
    beneficiary-(o,[event,person],for)]])].
e(all,[adj-(all,indef,nil,pl,[quantity])]).
e(illustrate,[verb-(illustrate,inf,[posprocess-[agent-(r,[person,event,artifact],nil),
    beneficiary-(o,[person,cause],for),affected-(r,[something],nil)]])].
e(deliverable,[adj-(deliverable,desc,nil,(s,pl),[quality])]).
e(period,[noun-(period,nil,s,common,count,[duration])]).
e(day,[noun-(day,nil,s,common,count,[time])]).
e(eight,[adj-(eight,cardinal,nil,pl,[quantity])]).
e(assure,[verb-(assure,inf,[mentinteraction-[agent-(r,[person],nil),
    beneficiary-(o,[person,event],for),affected-(r,[submatter,event],nil)]])]).
e(measure,[noun-(measure,nil,s,common,count,[measurement]),
    verb-(measure,inf,[physinteraction-[agent-(r,[human],nil),
    instrument-(o,[artifact],with),affected-(r,[pthing],nil),source-(o,[pspace],from),
    destination-(o,[pspace],to)]])].
e(that,[conj-(that,sub),pro-(that,rel,3,pl,(m,f),_2)]).
e(provide,[verb-(provide,inf,[physinteraction-[agent-(r,[person],nil),
    instrument-(o,[artifact],par),beneficiary-(o,[sentient,nstate],à),affected-
(r,[something],nil)]])]).
e(transportation,[noun-(transportation,nil,s,common,mass,[transfer])]).
e(installation,[noun-(installation,nil,s,common,count,[posprocess,transfer,stationary])]).
e(even,[adv-(even,affirmation)]).
e(same,[adj-(same,indef,(m,f),s,[attribute])]).
e(here,[adv-(here,location)]).
e(there,[adv-(there,location)]).
e(satisfaction,[noun-(satisfaction,nil,(s,pl),common,mass,[quality,pemotion])]).
e(entire,[adj-(entire,desc,nil,(s,pl),[derivation,attribute])]).
e(hope,[noun-(hope,nil,s,common,count,[fthing,submatter]),
    verb-(hope,inf,[mentinteraction-[agent-(r,[person,lanimal],nil),
    affected-(r,[event,mthing],for)]])].
e(serve,[verb-(serve,inf,[transfer-[agent-(r,[human],nil),instrument-(o,[artifact],avec),
    beneficiary-(r,[person],à),affected-(r,[artifact],nil)],assistance-[agent-
(r,[person],nil),
```

```
    affected-(r,[person],nil)],occupation-[affected-(r,[something],nil),
    attribute-(r,[something],comme)]]]]).
e(soon,[adv-(soon,time)]).
e(invite,[verb-(invite,inf,[mentinteraction-[agent-(r,[person],nil),
    affected-(r,[person],nil)]]]]).
e(accept,[verb-(accept,inf,[mentinteraction-[agent-(r,[person],nil),
    beneficiary-(o,[person],pour),affected-(r,[something],nil)]]]).
e(devote,[verb-(devote,inf,[mentinteraction-[agent-(r,[person],nil),
    beneficiary-(o,[something],à),affected-(r,[person],nil)]]]]).
e(expression,[noun - (expression, nil, s, common, count, [submatter])]).
e(feeling,[noun - (feeling, nil, s, common, count, [fthing])]).
e(our,[adj - (our, poss, nil, (s, pl), [ttrelation])]).
e(go,[verb - (go, inf, [selfdisplacement - [agent - (r, [human,animal,selfmoving],
nil),instrument - (o, [artifact], in),source - (o, [pspace], from),destination - (o,
[pspace], to)],mentinteraction - [agent - (r, [person], nil),beneficiary - (o, [person],
for),affected - (r, [artifact], through)]]]]).
e(folder,[noun - (folder, nil, s, common, count, [artifact])]).
e(attach,[verb -(attach,  inf, [contact -[agent - (r, [posprocess,structure,sentient], nil),
    instrument - (o, [artifact], par), affected - (r, [pthing,mthing], nil),
    destination - (r, [mthing,pthing], à)]]]]).
e(ardec,[noun - (ardec, nil, s, proper, count, [name])]).
e(incorporate,[verb - (incorporate, inf, [posprocess - [agent - (r, [person], nil),instrument
- (o, [artifact], with),affected - (r, [group], nil)]]]]).
e(furniture,[noun - (furniture, nil, (s, pl), common, mass, [nonselfmoving])]).
e(office,[noun - (office, nil, s, common, count, [location,group,occupation])]).
e(avenue,[noun - (avenue, nil, s, common, count, [pspace,name])]).
e(christophe,[noun - (christophe, nil, s, proper, count, [name,human])]).
e(colomb,[noun - (colomb, nil, s, proper, count, [name,human])]).
e(montreal,[noun - (montreal, nil, s, proper, mass, [name,pspace])]).
e(son,[noun - (son, nil, s, common, count, [person,ttrelation])]).
e(limit,[noun - (limit, nil, s, common, count, [pspace,ptime]),
    verb - (limit, inf, [mentinteraction - [agent - (r, [human,event], nil),
    affected - (r, [human,event], nil)],
    physinteraction - [agent - (r, [artifact], nil),affected - (r, [artifact,event], nil),
    result - (o, [event,pspace,ptime], nil)]]]]).
e(reference,[noun - (reference, nil, s, common, count, [ttrelation,artifact])]).
e(company,[noun - (company, nil, s, common, count, [group,accompaniment])]).
e(hello,[interjection - hello]).
e(recommend,[verb - (recommend, inf, [mentinteraction - [agent - (r, [human], nil),
    beneficiary - (o, [person], to),affected - (r, [mthing,event], nil)]]]]).
e(register,[verb - (register, inf, [posprocess - [agent - (r, [person], nil),
    affected - (r, [artifact], nil)]]]]).
e(september,[noun - (september, nil, s, proper, count, [time])]).
e(monday,[noun - (monday, nil, s, proper, count, [time])]).
e(lavallée,[noun - (lavallée, nil, s, proper, count, [person,name])]).
e(street,[noun - (street, nil, s, common, count, [pspace])]).
e(school,[noun - (school, nil, s, common, count, [location,group])]).
e(saint,[adj - (saint, desc, nil, s, [quality]),
    noun - (saint, nil, s, common, count, [human])]).
e(jérôme,[noun - (jérome, nil, s, proper, count, [human, name])]).
e(quebec,[noun - (quebec, nil, s, proper, count, [pspace,name])]).
e(road,[noun - (road, nil, s, common, count, [pspace])]).
e(milton,[noun - (milton, nil, (s, pl), proper, count, [name])]).
e(jean,[noun - (jean, nil, s, proper, count, [name,person])]).
e(paul,[noun - (paul, nil, s, proper, count, [name,human]),).
e(attention,[noun - (attention, nil, (s, pl), common, mass, [mentinteraction])]).
e(dupré,[noun - (dupré, nil, s, proper, count, [name,person])]).
e(object,[noun - (object, nil, s, common, count, [pthing,submatter])]).
e(dear,[adj - (dear, desc, nil, (s, pl), [attribute])]).
e(sirs,[noun - (sirs, nil, pl, proper, count, [person,name])]).
e(madam,[noun - (madam, nil, s, proper, count, [person,name])]).
e(friend,[noun - (friend, nil, s, common, count, [person,ttrelation])]).
e(marie,[noun - (marie, nil, s, proper, count, [person,name])]).
```

```
e(director,[noun - (director, nil, s, common, count, [name,person,occupation])]).
e(sale,[noun - (sale, nil, s, common, count, [transfer])]).
e(commercial,[adj - (commercial, desc, nil, (s, pl), [attribute]),
    noun - (commercial, nil, s, common, count, [submatter])]).
e(lavoie,[noun - (lavoie, nil, s, proper, count, [name,person])]).
e(légaré,[noun - (légaré, nil, s, proper, count, [person,name])]).
e(piece,[noun - (piece, nil, s, common, count, [unit,nonselfmoving,part])]).
e(table,[noun - (table, nil, s, common, count, [artifact,structure])]).
e(book,[noun - (book, nil, s, common, count, [artifact,submatter])]).
e(copy,[noun - (copy, nil, s, common, count, [artifact,ttrelation])]).
e(sing,[verb - (sing, inf, [physinteraction - [agent - (r, [human], nil),
    instrument - (o, [artifact], with),beneficiary - (o, [human,group], to),
    affected - (o, [submatter], nil)]])]).
e(now,[adv - (now, time)]).
e(see,[verb - (see, inf, [physinteraction - [agent - (r, [person,animal], nil),
    instrument - (o, [artifact], with),affected - (o, [pthing], nil)]])]).
e(understand,[verb - (understand, inf, [mentinteraction - [agent - (r, [person], nil),
    affected - (o, [mthing], nil)]])]).
e(write,[verb - (write, inf, [physinteraction - [
    agent - (r, [person], nil),
    instrument - (o, [artifact], with),
    beneficiary - (o, [person,pspace], to),
    affected - (o, [submatter,artifact], nil)]])]).
e(say,[verb - (say, inf, [physinteraction - [agent - (r, [person], nil),
    instrument - (o, [artifact], with),beneficiary - (o, [person], to),
    affected - (r, [mthing], nil)]])]).
e(him,[pro - (him, pers, 3, s, m, [human,animal]),
    pro - (him, refl, 3, s, m, [human,animal])]).
e(more,[adj - (more, indef, nil, s, [measurement]),adv - (more, quantity)]).
e(man,[noun - (man, nil, s, common, count, [person,pspace])]).
e(at,[prep - at]).
e(jeanne,[noun - (jeanne, nil, s, proper, count, [person,name])]).
e(first,[adj - (first, ordinal, nil, s, [ttrelation])]).
e('so and so',[noun - ('so and so', nil, s, proper, count, [person,name])]).
e(very,[adv - (very, intensity)]).
e(receive,[verb - (receive, inf, [physinteraction - [agent - (r, [person], nil),
     affected -(r, [animal,plant,natobject,energy,artifact], nil),source - (o,
[pspace,person], de)],
    mentinteraction - [agent - (r, [event,person], nil),affected - (r, [mthing], nil),
    source - (o, [event], de)]])]).
e(june,[noun - (june, nil, s, proper, count, [time])]).
e(become,[verb - (become, inf, [
    state - [affected - (r, [person,event,mthing,artifact], nil),
    attribute - (r, [something], nil)],
    posprocess - [affected - (r, [person,event,mthing,artifact], nil),
    result - (r, [something], nil)]])]).
e(base,[noun - (base, nil, s, common, count, [pspace,nonselfmoving])]).
e(basis,[noun - (basis, nil, s, common, mass, [submatter])]).
e(pay,[verb - (pay, inf, [transfer - [agent - (r, [person], nil),
    instrument - (o, [artifact], with),beneficiary - (o, [person], nil),
    affected - (o, [unit,artifact], nil)]])]).
e(for,[prep - for]).
e(work,[noun - (work, nil, s, common, mass, [posprocess,occupation]),verb - (work, inf
[event - [agent - (r, [something], nil),instrument - (o, [artifact], with),beneficiary - (o,
[person,event], for),affected - (o, [nlthing], nil)]])]).
e(read,[verb - (read, inf, [mentinteraction - [agent - (r, [human], nil),
    instrument - (o, [artifact], avec),affected - (o, [submatter,artifact], nil)]])]).
e(victor,[noun - (victor, nil, s, proper, count, [person,name])]).
e(loewen,[noun - (loewen, nil, s, proper, count, [name,person])]).
e(translation,[noun - (translation, nil, s, common, count,
[posprocess,mentinteraction,nonselfmoving])]).
e(automatic,[adj - (automatic, desc, nil, (s, pl), [attribute])]).
e(ontario,[noun - (ontario, nil, s, proper, mass, [location])]).
```

```
«(navan,[noun - (navan, nil, s, proper, mass, [pspace,name])]).
```

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

OPEN TRANSFER DICTIONARY

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

/*
The French_to_English (f_e) predicate associates a given French word in its customary dictionary form with possible English translations. English forms are given in their customary dictionary forms as well, i.e., singular. In some cases, for example «de», "nil" is given as an optional English equivalent. "nil" will be treated as meaning that there need not be an equivalent.
*/

```
f_e(adresse,[address]).
f_e(adresser,[send,address]).
f_e(agréer,[accept,suit]).

f_e(ainsi,[therefore]).
f_e(ami,[friend]).
f_e(ardec,[ardec]).
f_e(assurer,[provide,ensure,guarantee]).
f_e(attention,[attention]).
f_e(automatique,[automatic]).
f_e(av,[avenue]).
f_e(avenue,[avenue]).
f_e(avoir,[have,asset,wear,get,possess,be]).
f_e(à,[functional_prep]).
f_e(base,[base,basis]).
f_e(beaucoup,[many,much]).
f_e(besoin,[need]).
f_e(bibliothèque,[library,shelving]).
f_e(bonjour,[hello]).
f_e(bureau,[office,desk]).
f_e(chanter,[sing]).
f_e(chemin,[road]).
f_e(cher,[dear]).
f_e(chez,[at]).
f_e(choisir,[choose,select]).
f_e(christophe,[christophe]).
f_e(ci,[here]).
f_e(cinq,[five]).
f_e(colomb,[colomb]).
f_e(commercial,[sales,commercial]).
f_e(compagnie,[company]).
f_e(complet,[complete,full,entire,
    comprehensive]).
f_e(constater,[note,discover,see]).
f_e(copie,[copy]).
f_e(dans,[in, within]).
f_e(de,[functional_prep]).
f_e(demander,[request,ask]).
f_e(devenir,[become]).
f_e(délai,[period,term,deadline]).
f_e(dépliant,[folder,brochure,pamphlet]).
f_e(dévouer,[devote,dedicate]).
f_e(dire,[say]).
f_e(directeur,[director,manager]).
f_e(document, [document, documentation]).
f_e(documentation,[documentation,documents,
    papers]).
f_e(donc,[therefore]).
f_e(dupré,[dupré]).
f_e(en,[in,by,while,them]).
f_e(entier,[complete,entire,whole]).
f_e(espoir,[hope]).
f_e(est,[east]).
f_e(et,[and]).
f_e(eux,[them]).
f_e(expression,[expression]).
f_e(école,[school]).
f_e(écrire,[write]).
f_e(être,[be,being]).
f_e(fabriquer,[manufacture]).
f_e(facile,[easy,simple]).
f_e(faire,[make,do]).
f_e(fils,[son]).
f_e(fonction,[function]).
f_e(gamme,[range,series]).
f_e(garantir,[assure,guarantee]).
f_e(heureux,[happy,pleased,delighted]).
f_e(homme,[man]).
f_e(huit,[eight]).
f_e(ici,[here]).
f_e(il,[it, he]).
f_e(illustrer,[illustrate]).
f_e(ils,[they]).
f_e(inc,[incorporated]).
f_e(incorporer,[incorporate]).
f_e(installation,[installation]).
f_e(inviter,[invite]).
f_e(je,['I']).
f_e(jean,[jean]).
f_e(jeanne,[jeanne]).
f_e(jérôme,[jérôme]).
f_e(joindre,[attach,join]).
f_e(jour,[day]).
f_e(juin,[june]).
f_e(la,[the, it]).
f_e(lavallée,[lavallée]).
f_e(lavoie,[lavoie]).
f_e(là,[there]).
f_e(le,[the, it]).
f_e(le_la,[the, it]).
f_e(les,[the, them]).
f_e(lettre, [letter]).
f_e(leur,[their,them]).
f_e(légaré,[légaré]).
f_e(limiter,[limit]).
f_e(lire,[read]).
```

```
f_e(livrable,[deliverable]).
f_e(livre,[book]).
f_e(loewen,[loewen]).
f_e(lui,[him]).
f_e(lundi,[monday]).
f_e(m,[sir,mr]).
f_e(madame,[madam]).
f_e(maintenant,[now]).
f_e(manger,[eat, food]).
f_e(marie,[marie]).
f_e(me,[me,myself]).
f_e(mesdames,[madam]).
f_e(messieurs,[sir]).
f_e(mesure,[measure]).
f_e(meuble,[furniture]).
f_e(même,[even,same,very]).
f_e(milton,[milton]).
f_e(modèle,[model,pattern]).
f_e(monsieur,[sir,mr]).
f_e(montréal,[montreal]).
f_e(navan,[navan]).
f_e(necessaire,[necessary,required]).
f_e(nos,[our]).
f_e(nous, [we, us, ourself]).
f_e(objet,[object,thing,subject,re]).
f_e(ontario,[ontario]).
f_e(parcourir,[go,cover,travel,glance,read]).
f_e(paul,[paul]).
f_e(payer,[pay]).
f_e(pièce,[piece]).
f_e(plus,[more]).
f_e(pour,[for]).
f_e(premier,[first]).
f_e(prier,[invite,request,beg]).
f_e(prochainement,[soon,shortly]).
f_e(que, [that, which]).
f_e(qué,[qué]).
f_e(québec,[quebec]).
f_e(rayonnage,[shelving]).
f_e(recevoir,[receive]).
f_e(recommander,[recommend, register]).
f_e(remercier, [thank]).
f_e(référence,[reference]).
f_e(rue,[street]).
f_e(saint,[saint]).
f_e(satisfaction,[satisfaction]).
f_e(sentiment,[feelings,feeling,emotion]).
f_e(septembre,[september]).
f_e(servir,[serve,aid]).
f_e(tableau,[table]).
f_e(te,[you,yourself]).
f_e(tous,[all,every]).
f_e(tout,[all,every]).
f_e(traduction,[translation]).
f_e(transport,[transportation,shipping]).
f_e(travail,[work]).
f_e(très,[very]).
f_e(trois,[three]).
f_e(trouver,[find,discover]).
f_e(tu,[you]).
f_e(un,[one,a]).
f_e(une,[a,one]).
f_e(untel,['so and so']).
f_e(victor,[victor]).
f_e(voir,[see,understand]).
f_e(vos,[your]).
f_e(votre, [your]).
f_e(vous, [you, yourself]).
f_e(y,[there]).
```

```
/*************************************************************************
*************************************************************************
WORD-ACQUISITION MODULE
© 1987, 1988 Victor Loewen
*************************************************************************
*************************************************************************/
```

/*
The main learn and discover predicates need not be called from within the program.
The user may wish to teach ORDINOTRAD some new words without having them translated.
The calls are simply learn(french_word) and discover(english_word). Note, the word must be
provided in lower case letters only, even if a proper name.
*/

```
:- assert(learner_on).

learn(LU) :-
    learner(f, LU).
discover(LU) :-
    learner(e, LU).

learner(Lang, Mumbo_jumbo) :-
    explode(Q, ['Is "', Mumbo_jumbo, '" an arbitrary string?']),
    ask_question(Q, [yes,y,no,n],Answer),
    (Answer = yes ; Answer = y),
    nl,write('THANK YOU'),nl,
    Fact =.. [Lang, Mumbo_jumbo, [string-Mumbo_jumbo]],
    assert(Fact), !.
```

/*
Abbreviations are entered in the dictionary as abbreviations, however, their
description includes the fact that they are abbreviations as well as the description
of their corresponding full form.
e.g.
?- f(cie,X).
  X = [abbrev, noun -(compagnie, f, s, common, count, [accompaniment,group])]
*/

```
learner(Lang, LU) :-
    explode(Q1, ['Can "', LU, '" be considered an abbreviation?']),
    ask_question(Q1, [yes,y,no,n],Answer),
    (Answer = yes ; Answer = y),
    explode(Q2, ['"', LU, '" is an abbreviation of what word?']),
    ask_question(Q2, Full_form),
    analyze(Full_form, Descs),
    Entry =.. [Lang, LU, [abbrev | Descs]],
    file_name(Lang, Filename),
    fopen(Filename, "a", IO_stream),
    writeq(IO_stream, Entry), writeq(IO_stream, .), nl(IO_stream),
    fclose(IO_stream),
    assert(Entry).
```

/*
If it is learned that a given, unknown LU is syntactically ambiguous
(i.e., Pt_sp > 1), then the dictionary entry is:
    f(LU, [Pt_sp1-(Dict_form_as_Pt_sp1, Desc_of_lu_as_Pt_sp1),
           Pt_sp2-(Dict_form_as_Pt_sp2, Desc_of_lu_as_Pt_sp2) | etc.]).

If the LU is syntactically unique, then the dictionary entry is:
f(Dict_form, [Pt_sp-(Dict_form, Desc_of_dict_form)]).

e.g., if an unknown LU happens to be only an adjective and plural and feminine at that, the dictionary entry still is
f(Dict_form_masc, [adj-(Dict_form_masc, m, s, Meaning)]).

The call
learn(différentes).
results in the dictionary entry
f(différent, [adj-(différent, m, s, [relation])])
and in the tree as
lu(low, adj-(différent, f, pl, [relation])).

The call
learn(est).
results in
f(est, [verb-(être, (pres,ind,3,s),[Meaning-Frame]).
noun-(est, m, s, mass, common, [Meanings])]).

Note that the dictionary is not to contain plurals and feminines if they can be deduced by the morphological analysis routines. These routines also make the appropriate changes to the descriptions of lexical units which are not masculine singular or are not in the infinitive. Further note that the first argument in the parentheses following a Pt_sp is what is retained in the tree for transfer purposes.

Provision is made for closed categories, as this word-acquisition module was used to create consistent dictionaries. The final word-acquisition module would only ask for open categories. (adjective, adverb, noun, verb, abbreviation)
*/

```
learner(Lang, LU) :-
    learner(Lang, LU, Forms, Descs,
        [adjective, noun, verb, adverb, preposition, article, conjunction,
        pronoun, interjection]),
    nl,write('THANK YOU'),nl, !,
    sort_info(Lang, LU, Forms, Look_up_form, Descs, Description),
    Entry =.. [Lang, Look_up_form, Description],
    file_name(Lang, Filename),
    fopen(Filename, "a", IO_stream),
    writeq(IO_stream, Entry), writeq(IO_stream, .), nl(IO_stream),
    fclose(IO_stream),
    assert(Entry).

learner(Lang, _, [], [], []) :- !.

learner(Lang, LU, [Form | Forms], [Desc | Descs], [Pt_sp | Tail]) :-
    an(AN, Pt_sp),  % defined in Generator file
    explode(Question, ['Can "', LU, '" be considered ', AN, ' ', Pt_sp, '?']),
    ask_question(Question, [yes,y,no,n],Answer),
    (Answer = yes ; Answer = y),
    Goal =.. [Pt_sp, Lang, LU, Form, Desc],
    Goal, !,
    learner(Lang, LU, Forms, Descs, Tail).

learner(Lang, LU, Forms, Descs, [H | T]) :-
    learner(Lang, LU, Forms, Descs, T).
```

```
sort_info(_, LU, [One_form_only], One_form_only, Desc, Desc) :- !.

sort_info(Lang, LU, _, LU, Descs, Description) :-
    specify_descs(Lang, LU, Descs, Description), !.

specify_descs(_, _, [], []) :- !.
specify_descs(Lang, LU, [verb-(Inf, inf, M_F) | Rest], [verb-(Inf, TMPN, M_F) | List]) :-
    inf_TMPN(Lang, LU, Inf, TMPN),
    specify_descs(Lang, LU, Rest, List).
specify_descs(Lang, LU, [noun -(Nn, G1, N1, CP, CM, M) | Rest],
        [noun -(Nn, G2, N2, CP, CM, M) | List]) :-
    noun_adj_analysis(Lang, LU, Nn, G1, G2, N1, N2),
    specify_descs(Lang, LU, Rest, List).
specify_descs(Lang, LU, [adj - (Adj, Type, G1, N1, M) | Rest],
        [adj - (Adj, Type, G2, N2, M) | List]) :-
    noun_adj_analysis(Lang, LU, Adj, G1, G2, N1, N2),
    specify_descs(Lang, LU, Rest, List).
specify_descs(Lang, LU, [Desc | Descs], [Desc | List]) :-
    specify_descs(Lang, LU, Descs, List).

noun_adj_analysis(f, LU, LU, G, G, N, N).
noun_adj_analysis(f, LU, Nn, _, f, N, N) :-
    fem_to_masc(LU, Nn).
noun_adj_analysis(f, LU, Nn, G, G, _, pl) :-
    pl_to_sing(LU, Nn).
noun_adj_analysis(f, LU, Nn, _, f, _, pl) :-
    pl_to_sing(LU, Word_sing),
    fem_to_masc(Word_sing, Nn).

inf_TMPN(_, LU, LU, inf).
inf_TMPN(f, LU, Inf, TMPN) :-
    find_vb_root(LU, Root, TMPN),
    get_infinitive(Root, Inf).
inf_TMPN(e, LU, Inf, TMPN) :-
    get_eng_infinitive(LU, Inf, TMPN).

file_name(e, 'DATA:Dictionaries:English Dictionary').
file_name(f, 'DATA:Dictionaries:French Open Dictionary').
```

/* LEARN ABOUT VERBS */

```
verb(_, LU, Inf, verb-(Inf, inf, Meanings_frames)) :-
    explode(Q, ['Please enter the infinitive form for "', LU, '".']),
    ask_question(Q, Inf),
    ask_for_meanings(Inf, Meanings, verb),
    ask_meaning_frame(Inf, Meanings, Meanings_frames).
```

/* LEARN ABOUT NOUNS */

```
noun(Lang, LU, Dict_form, noun-(Dict_form, Gen, Num, Prop_com, Count_mass, Meanings)) :-
    get_dict_form(LU, Dict_form),
    get_gender(Lang, Dict_form, Gen),
    get_number(Dict_form, Num),
    explode(Q1, ['Is "', Dict_form, '" a proper or common noun?']),
    ask_question(Q1, [proper, common], Prop_com),
    explode(Q2, ['Is "', Dict_form, '" a count or mass noun?']),
    ask_question(Q2, [count, mass], Count_mass),
    ask_for_meanings(Dict_form, Meanings, noun).
```

/* LEARN ABOUT ADJECTIVES
The user is not asked if an adjective is unmarked, comparative or superlative.
If such indications are required, they can be deduced through analysis.
*/

```
adjective(Lang, LU, Dict_form,  adj-(Dict_form, Type, Gender, Number, Meanings)) :-
    get_dict_form(LU, Dict_form),
    explode(Q, ['What kind of adjective is "', Dict_form, '"?']),
    ask_question(Q, [desc,poss,cardinal,ordinal,multi,dem,rel,interr,indef], Type),
    get_gender(Lang, Dict_form, Gender),
    get_number(Dict_form, Number),
    ask_for_meanings(Dict_form, Meanings, adjective).
```

/* LEARN ABOUT ADVERBS */

```
adverb(_, LU, LU, adv-(LU, Type)) :-
    explode(Q, ['What kind of adverb is "', LU, '"?']),
    ask_question(Q,[manner,quantity,intensity,time,location,affirmation,negation,doubt],
        Type).
```

/* "LEARN" ABOUT PREPOSITIONS */

```
preposition(_, LU, LU, prep-(LU)).
```

/* LEARN ABOUT CONJUNCTIONS */

```
conjunction(_, LU, LU, conj-(LU, Type)) :-
    explode(Q, ['What kind of conjunction is "', LU, '"?']),
    ask_question(Q,[coord, sub], Type).
```

/* LEARN ABOUT PRONOUNS */

```
pronoun(_, LU, LU, pro-(LU, Type, Person, Number, Gender, Meanings)) :-
    pronoun_type(LU, Type),
    pronoun_person(LU, Type, Person),
    get_number(LU, Number),
    get_gender(f, LU, Gender),
    ask_for_meanings(LU, Meanings, pronoun).

pronoun_type(LU, Type) :-
    explode(Q, ['What kind of pronoun is "', LU, '"?']),
    ask_question(Q,[pers, rel, interr, poss, dem, indef, intens, refl], Type).

pronoun_person(LU, Type, Person) :-
%   member(Type, [pers, rel, interr, poss, refl]),
    explode(Q, ['Is "', LU, '" 1st, 2nd, or 3rd person?']),
    ask_question(Q, [1,2,3], Person).

pronoun_person(LU, _, nil).
```

/* LEARN ABOUT ARTICLES */

```
article(Lang, LU, LU, art-(LU, Type, Gender, Number)) :-
```

```
    explode(Q, ['What kind of article is "', LU, '"?']),
    ask_question(Q,[def, indef], Type),
    get_gender(Lang, LU, Gender),
    get_number(LU, Number).

/*
LEARN ABOUT INTERJECTIONS
*/

interjection(Lang, LU, LU, interjection-(LU)).

/*
Some basic routines are needed to ask the user for particular information. All meanings and
case constraints must be verified as being 'understood' according to the taxonomical knowledge base.
*/

get_dict_form(LU, Dict_form) :-
    explode(Q, ['Please enter the accepted dictionary form for "', LU, '".']),
    ask_question(Q, Dict_form).

get_gender(e, _, nil).
get_gender(f, Dict_form, Gender) :-
    explode(Q, ['Is "', Dict_form, '" masculine, feminine, or possibly both?']),
    ask_question(Q, [m, f, both], A),
    gender(A, Gender).

gender(both, (m,f)).
gender(m, m).
gender(f, f).

get_number(Dict_form, Number) :-
    explode(Q, ['Is "', Dict_form, '" singular, plural, or possibly both?']),
    ask_question(Q, [s, pl, both], A),
    number(A, Number).

number(s,s).
number(pl,pl).
number(both, (s,pl)).

ask_meaning_frame(_, [], []).
ask_meaning_frame(Inf, [Meaning | Meanings], [Meaning-Frame | M_fs]) :-
    case_list(Case_list),
    ask_frame(Inf, Meaning, Frame, Case_list),
    ask_meaning_frame(Inf, Meanings, M_fs).

ask_frame(Inf, _, [], []) :- !.
ask_frame(Inf, Meaning, [Case-(Type, Constraints, Prep) | List], [Case | Cases]) :-
    explode(Q, ['For the verb "', Inf, '" with the meaning "', Meaning, '", is the case
relation \n"',
    Case, '" required, optional, or disallowed?']),
    ask_question(Q, [r, o, d], Type),
    (Type = r; Type = o),
    ask_for_prep(Case, Inf, Prep),
    repeat,
    ask_for_constraints(Case, Answers),
    verify(Answers, Constraints),
    ask_frame(Inf, Meaning, List, Cases).
ask_frame(Inf, Meaning, List, [Case | Cases]) :-
    ask_frame(Inf, Meaning, List, Cases).

case_list([agent,instrument,beneficiary,affected,result, source,destination,attribute]).
```

```
ask_for_constraints(R, Constraints) :-
    explode(Question, ['Enter a constraint for the role "', R, '".' ]),
    ask_question(Question, Answer),
    ask_constraints(R, Answer, Constraints).

ask_constraints(_, something, [something]) :- !.
ask_constraints(_, nil, []) :- !.
ask_constraints(R, Constraint, [Constraint | L]) :-
    explode(Question, ['Enter another constraint to be added for the role "', R,
        '", or enter "nil".' ]),
    ask_question(Question, Answer),
    ask_constraints(R, Answer, L).
```

/*
When providing intensional meanings for various lexical items, it is important to remember that meanings for lexical items to be stored in the dictionary need to be as specific as possible, i.e., that an effort is made to use one or more applicable terminals in the taxonomical tree shown in Appendix 2.
On the other hand, the intensional meanings provided as constraints in a frame for an event must be as general as possible without allowing for nonsensical relations between sentence components, i.e., an effor is made to use as few high level nodes as possible from the taxonomical tree.
This arrangement is required since intensional meanings of various words are checked to see if they come under the restricting nodes. For example, if a living thing (lthing) is the constraint placed on an agent, then a lexical unit such as "Monsieur" would meet the restriction, as its intensional meanings would include "human," which is a member of the class of living things.
*/

```
ask_for_meanings(Dict_form, Meanings, Pt_sp) :-
    nl,
    write('Please consult the commonsense taxonomy when answering the following questions.'),
    nl,
    an(AN, Pt_sp),
    explode(Question, ['Enter an intensional meaning for "', Dict_form, '" as ',
        AN, ' ', Pt_sp, '.']),
    ask_question(Question, Answer),
    ask_meanings(Dict_form, Answer, Answers),
    verify(Answers, Meanings).

ask_meanings(_, something, [something]) :- !.
ask_meanings(_, nil, []) :- !.
ask_meanings(Dict_form, Meaning, [Meaning | L]) :-
    explode(Question, ['Enter another meaning for "', Dict_form, '", or enter "nil".' ]),
    ask_question(Question, Answer),
    ask_meanings(Dict_form, Answer, L).

ask_for_prep(Case, Inf, Prep) :-
    explode(Q,
['Enter the preposition associated with the role "', Case, '" and the verb "', Inf, '"
or enter "nil".' ]),
    ask_question(Q, Prep).  % a check could be added here

verify([], []) :- !.
verify([H | T], [H|L]) :-
    isa(H, something),
    verify(T, L).
verify([H | T], L) :-
```

```
   explode(Q, ['Sorry, "', H, '" is not found in the taxonomy.  Symbols must be entered
correctly.
Please consult the taxonomy and enter the correct response or "nil".']),
   ask_question(Q, Answer),
   (Answer = nil, verify(T, L) ; verify([Answer|T], L)).
```

/*
CREATE ENTRIES FOR THE FRENCH-ENGLISH TRANSFER DICTIONARY
A whole set of problems arises when a French word can (and should) be translated by more than one English word (e.g., "modalités" : "terms and conditions").
A different set of problems arise when several French words can (and should) be translated by only one English word (e.g., "de la part de" : "from").
This situation may or may not involve fixed expressions in the French.
It is acceptable to have the individual elements of an (idiomatic) expression entered into the dictionary; yet should there be a special database of these expressions and their equivalents?

In the limited context of this work, all French input will be dealt with on a word for word basis. As a result, there is much room for development in this regard.
*/

```
create_f_e(LUf) :-
    ask_for_equivalents(LUf, Equivalents),
    nl,write('THANK YOU'),nl,
    Entry =.. [f_e, LUf, Equivalents],
    fopen('DATA:Dictionaries:Open Transfer Dictionary', "a", IO_stream),
    writeq(IO_stream, Entry), writeq(IO_stream, .), nl(IO_stream),
    fclose(IO_stream),
    assert(Entry),
    add_equivalents_to_E_dict(Equivalents).

ask_for_equivalents(LUf, Equivalents) :-
    explode(Question, ['Enter equivalent English textual material for "', LUf, '".' ]),
    ask_question(Question, Answer),
    ask_equivalents(LUf, Answer, Equivalents).

ask_equivalents(_, nil, []) :- !.
ask_equivalents(LUf, Equivalent, [Equivalent | List]) :-
    explode(Q, ['Enter another equivalent English word for "', LUf, '", or enter "nil"' ]),
    ask_question(Q, Answer),
    ask_equivalents(LUf, Answer, List).

add_equivalents_to_E_dict([]).
add_equivalents_to_E_dict([H | T]) :-
    e(H, _),
    add_equivalents_to_E_dict(T).
add_equivalents_to_E_dict([H | T]) :-
    discover(H),
    add_equivalents_to_E_dict(T).
```

```
/*************************************************************************
*************************************************************************
COMMONSENSE, TAXONOMICAL RELATIONSHIPS
© 1987, 1988 Victor Loewen
*************************************************************************
*************************************************************************/
```

/*
This set of rules for establishing taxonomical relationships is comprised, at the present, of a property inheritance mechanism capable of determining the truth value of class-membership relationships. In other words, something is or isn't a member of a class.

These rules could be developed into an inference engine by adding rules determining 'has a' or 'is part of' relationships, colour relationships (which could probably be handled with the property inheritance rules), and time relationships. cf. Schubert et al, Computer 16.10 (1983): 53-60.

A tree-shaped diagram of these rules is provided in Appendix 2. This tree is much more readable and gives the full form of all the arcane abbreviations used herein.
*/

/*
A property inheritance mechanism. X is a Y if X = Y or if X is an instance of class Y.
*/

```
isa(X, X).
isa(X, Y) :-
    instance(X, Y).
```

/*
Instances of classes based on commonsense taxonomy.
The call instance/2 will provide the following results:
    If both arguments are bound, then the rule succeeds or fails.
        e.g. instance(motion,event) will succeed;
              instance(quality,fthing) will fail.
    If the first argument is bound, the rule returns the class containing the instance given.
        Upon backtracking, the next higher class is returned.
    If the first argument is unbound, the rule returns the first instance listed for the class given.
        Upon backtracking, the next instance is returned.
*/

/* LEVEL 0 */

```
instance(event, X) :-
    isa(something, X).
instance(thing, X) :-
    isa(something, X).
```

/* LEVEL 1 */

```
instance(mthing, X) :-
    isa(thing, X).
instance(pthing, X) :-
    isa(thing, X).
instance(fthing, X) :-
    isa(thing, X).
```

```
instance(state, X) :-
    isa(event, X).
instance(nstate, X) :-
    isa(event, X).
```

/* LEVEL 2 */

```
instance(process, X) :-
    isa(nstate, X).
instance(activity, X) :-
    isa(nstate, X).
instance(motion, X) :-
    isa(nstate, X).

instance(volition, X) :-
    isa(mthing, X).
```

```
instance(intention, X) :-
    isa(mthing, X).
instance(submatter, X) :-
    isa(mthing, X).
instance(construct, X) :-
    isa(mthing, X).

instance(lthing, X) :-
    isa(pthing, X).
instance(nlthing, X) :-
    isa(pthing, X).

instance(pomotion, X) :-
    isa(fthing, X).
instance(iemotion, X) :-
    isa(fthing, X).
```

```
instance(abspressure, X) :-
    isa(fthing, X).
instance(physpressure, X) :-
    isa(fthing, X).

/* LEVEL 3 */

instance(posprocess, X) :-
    isa(process, X).
instance(negprocess, X) :-
    isa(process, X).

instance(occupation, X) :-
    isa(activity, X).
instance(interaction, X) :-
    isa(activity, X).

instance(selfdisplacement,
X) :-
    isa(motion, X).
instance(displacement, X) :-
    isa(motion, X).

instance(theory, X) :-
    isa(submatter, X).
instance(science, X) :-
    isa(submatter, X).
instance(knowledge, X) :-
    isa(submatter, X).
instance(rule, X) :-
    isa(submatter, X).
instance(argument, X) :-
    isa(submatter, X).
instance(thought, X) :-
    isa(submatter, X).

instance(relation, X) :-
    isa(construct, X).
instance(structure, X) :-
    isa(construct, X).
instance(measurement, X) :-
    isa(construct, X).
instance(location, X) :-
    isa(construct, X).
instance(time, X) :-
    isa(construct, X).
instance(manner, X) :-
    isa(construct, X).
instance(frequency, X) :-
    isa(construct, X).
instance(intensity, X) :-
    isa(construct, X).
instance(attitude, X) :-
    isa(construct, X).

instance(plant, X) :-
    isa(lthing, X).
instance(sentient, X) :-
    isa(lthing, X).

instance(natobject, X) :-
    isa(nlthing, X).
instance(artifact, X) :-
    isa(nlthing, X).
instance(energy, X) :-
    isa(nlthing, X).
instance(physphen, X) :-
    isa(nlthing, X).
instance(socphen, X) :-
    isa(nlthing, X).
instance(soccustom, X) :-
    isa(nlthing, X).
instance(socsystem, X) :-
    isa(nlthing, X).

/* LEVEL 4 */

instance(mentinteraction, X)
:-
    isa(interaction, X).
instance(physinteraction, X)
:-
    isa(interaction, X).
instance(emotinteraction, X)
:-
    isa(interaction, X).

instance(series, X) :-
    isa(structure, X).
instance(hierarchy, X) :-
    isa(structure, X).
instance(network, X) :-
    isa(structure, X).
instance(mass, X) :-
    isa(structure, X).

instance(numeral, X) :-
    isa(measurement, X).
instance(unit, X) :-
    isa(measurement, X).
instance(standard, X) :-
    isa(measurement, X).

instance(pspace, X) :-
    isa(location, X).
instance(distance, X) :-
    isa(location, X).

instance(ptime, X) :-
    isa(time, X).
instance(duration, X) :-
    isa(time, X).

instance(affirmation, X) :-
    isa(attitude, X).
instance(negation, X) :-
    isa(attitude, X).
instance(doubt, X) :-
    isa(attitude, X).

instance(terelation, X) :-
    isa(relation, X).
instance(ttrelation, X) :-
    isa(relation, X).

instance(person, X) :-
    isa(sentient, X).
instance(animal, X) :-
    isa(sentient, X).

instance(stationary, X) :-
    isa(artifact, X).
instance(nonstationary, X)
:-
    isa(artifact, X).

/* LEVEL 5 */

instance(possession, X) :-
    isa(physinteraction, X).
instance(assistance, X) :-
    isa(physinteraction, X).
instance(contact, X) :-
    isa(physinteraction, X).
instance(confrontation, X)
:-
    isa(physinteraction, X).
instance(transfer, X) :-
    isa(physinteraction, X).
instance(consume, X) :-
    isa(physinteraction, X).
instance(use, X) :-
    isa(physinteraction, X).

instance(agent, X) :-
    isa(terelation, X).
instance(instrument, X) :-
    isa(terelation, X).
instance(beneficiary, X) :-
    isa(terelation, X).
instance(affected, X) :-
    isa(terelation, X).
instance(result, X) :-
    isa(terelation, X).
instance(source, X) :-
    isa(terelation, X).
instance(destination, X) :-
    isa(terelation, X).

instance(cause, X) :-
    isa(ttrelation, X).
instance(purpose, X) :-
    isa(ttrelation, X).
instance(condition, X) :-
    isa(ttrelation, X).
instance(complement, X) :-
    isa(ttrelation, X).
instance(accompaniment, X)
:-
    isa(ttrelation, X).
instance(possessor, X) :-
    isa(ttrelation, X).
instance(comparison, X) :-
    isa(ttrelation, X).
instance(equality, X) :-
    isa(ttrelation, X).
```

```
instance(inferiority, X) :-
    isa(ttrelation, X).
instance(superiority, X) :-
    isa(ttrelation, X).
instance(manifestation, X)
:-
    isa(ttrelation, X).
instance(derivation, X) :-
    isa(ttrelation, X).
instance(attribute, X) :-
    isa(ttrelation, X).

instance(suprahuman, X) :-
    isa(person, X).
instance(human, X) :-
    isa(person, X).
instance(group, X) :-
    isa(person, X).

instance(microbe, X) :-
    isa(animal, X).
instance(bug, X) :-
    isa(animal, X).
instance(lanimal, X) :-
    isa(animal, X).

instance(selfmoving, X) :-
    isa(nonstationary, X).
instance(nonselfmoving, X)
:-
    isa(nonstationary, X).
```

/* LEVEL 6 */

```
instance(quality, X) :-
    isa(attribute, X).
instance(state, X) :-
    isa(attribute, X).
instance(quantity, X) :-
    isa(attribute, X).
instance(nature, X) :-
    isa(attribute, X).
instance(shape, X) :-
    isa(attribute, X).
instance(name, X) :-
    isa(attribute, X).

instance(abbreviation, X) :-
    isa(derivation, X).
instance(part, X) :-
    isa(derivation, X).
instance(element, X) :-
    isa(derivation, X).
instance(organ, X) :-
    isa(derivation, X).

instance(fish, X) :-
    isa(lanimal, X).
instance(bird, X) :-
    isa(lanimal, X).
instance(reptile, X) :-
    isa(lanimal, X).
instance(quadruped, X) :-
    isa(lanimal, X).
instance(simian, X) :-
    isa(lanimal, X).
```

/*
These instances are added since they are commonsense but cannot be inferred from the above rules.
*/

```
isa(lthing, pspace).
isa(natobject, pspace).
isa(artifact,pspace).
```

/*
As a note of interest, the taxonomy says that all humans are persons, but it does not necessarily follow that all persons are human. (Fallacy of the illicit converse of the A proposition.)

e.g.

?- isa(human, person).
yes
?- isa(person, human).
no
*/

```
/*************************************************************************
*************************************************************************
TRANSLATE-TO-DISK UTILITY
© 1987, 1988 Victor Loewen
*************************************************************************
*************************************************************************/

t2d(Filename) :-
    nl,write('Now translating file titled:  '), write(Filename),
    explode(FEnglish, [Filename, '.English']),
    nl,write('The translation will be in the corpora folder with the title:  '),
write(FEnglish),
    nl,write('Please stand by; you may be needed to answer a few questions.'),nl,
    explode(File, ['DATA:corpora:', Filename]),
    explode(Output_file, ['DATA:corpora:', FEnglish]),

    Start is cputime,
    fopen(File, "r", Read_only_stream),
       t2d_process_file(Read_only_stream, Output_file),
    fclose(Read_only_stream),
    Elapsed is cputime - Start,

    t2d_print_time(Output_file, Elapsed),
    nl,write('Processing has been completed.'),
    nl,write('The translation is in the corpora folder with the title:  '), write(FEnglish).

t2d_process_file(Input_file_stream, Output_file) :-
    re_readline(Input_file_stream, String_french),
    ttynl,ttywrite('NOW TRANSLATING:'),ttynl,
    ttyprintstring(String_french),ttynl,
       t2d_translate(String_french, String_english, Output_file),
    fopen(Output_file, "a", Stream),
    nl(Stream),
    printstring(Stream, String_english),
    fclose(Stream),
    ttywrite('TRANSLATION SAVED TO DISK:'), ttynl,
    ttywrite(String_english), ttynl,
    fail.
t2d_process_file(_, _).  % this call must succeed when eof is reached

t2d_translate(String_french, String_english, _) :-
    analysis(String_french, Deep_structure_representation_F), !,
    transfer(Deep_structure_representation_F, Deep_structure_representation_E),
    generation(Deep_structure_representation_E, String_english), !.
t2d_translate(String_french, 'TRANSLATION UNSUCCESSFUL', Output_file) :-
    fopen(Output_file, "a", Stream),
    nl(Stream),
    printstring(Stream, String_french),
    fclose(Stream).


t2d_write_tree(S, [],_) :- !.

t2d_write_tree(Stream, [Node-Branch | Rest], X) :-
    t2d_put_tab(Stream, X),
    write(Stream, Node),
    nl(Stream),
    Y is X + 1,
    t2d_write_tree(Stream, Branch, Y),
    t2d_write_tree(Stream, Rest, X).
```

```
t2d_write_tree(Stream, [H | T], X) :-
    t2d_put_tab(Stream, X),
    write(Stream, H),
    nl(Stream),
    t2d_write_tree(Stream, T, X).

t2d_write_tree(Stream, H, X) :-
    t2d_put_tab(Stream, X),
    write(Stream, H),
    nl(Stream).

t2d_put_tab(Stream,0) :- !.
t2d_put_tab(Stream,X) :-
    put(Stream,'\t'),
    Y is X -1,
    t2d_put_tab(Stream,Y).
```

/*
Output length of time required for translation process.
*/

```
t2d_print_time(Output_file, Elapsed) :-
    Total is integer(Elapsed),
    secs_to_mm_ss(Total, Min, Sec),
    minutes(Min, Minutes),
    seconds(Sec, Seconds),
    string(Sec, S),
    string(Min, M),
    explode(Info, ['Work on this file, including I/O and user interaction, took ',
        M, ' ', Minutes, ' and ', S, ' ', Seconds, '.']),
    topen(Output_file, "a", Stream),
    nl(Stream,5),
    printstring(Stream, Info),
    nl(Stream,1),
    fclose(Stream).
```

```
/*************************************************************************
*************************************************************************
UTILITIES
© 1987, 1988 Victor Loewen
*************************************************************************
*************************************************************************/

get_last_X_chars(List_chars, Ending_list, X) :-
    length(List_chars, Length),
    Position is Length - X,
    Position > 0,
    tail_list(Position, List_chars, Ending_list), !.

tail_list(0,Tail,Tail) :- !.
tail_list(Num, [_|Tail],List) :-
    N is Num - 1,
    tail_list(N, Tail, List).


app([],L,L).
app([H|L1],L2,[H|L3]) :-
    app(L1,L2,L3).

join((A, Y),B,(A, Z)) :-
    !,
    join(Y,B,Z).
join(A,B,(A, B)).

mem(X,[X | _]).
mem(X,[_ | T]) :-
    mem(X,T).

cut(_, [], []) :- !.
cut(X, [X | T], T) :- !.
cut(X, [H | L1], [H | L2]) :-
    cut(X, L1, L2).

substitute(_, _, [], []) :- !.
substitute(Replace_this, With_this, [Replace_this | T], [With_this | T]) :- !.
substitute(X, Y, [H | T], [H | L]) :-
    substitute(X, Y, T, L).

write_list([]).
write_list([H]) :-
    write(H), !.
write_list([H | T]) :-
    write(H), tab(1),
    write_list(T).

write_column([]) :- !.
write_column([H | T]) :-
    write(H), nl,
    write_column(T).


write_tree([],_) :- !.

write_tree((A,B), X) :-
    write_tree(A, X),
    write_tree(B, X).
```

```
write_tree(Node-Branch, X) :-
    put_tab(X), write(Node), nl,
    Y is X + 1,
    write_tree(Branch, Y).

write_tree([Node-Branch | Rest], X) :-
    put_tab(X),write(Node),nl,
    Y is X + 1,
    write_tree(Branch, Y),
    write_tree(Rest, X).

write_tree([H | T], X) :-
    put_tab(X), write(H), nl,
    write_tree(T, X).

write_tree(H, X) :-
    put_tab(X), write(H), nl.

put_tab(0) :- !.
put_tab(X) :-
    put('\t'),
    Y is X -1,
    put_tab(Y).

secs_to_mm_ss(Secs, Minutes, Seconds) :-
    M is Secs/60,
    Minutes is integer(M),
    Seconds is Secs mod 60.
```

GLOSSARY OF TERMS

Affected	A case label which identifies the role played by something that undergoes an action, is affected by an action, or experiences an action. It may also be something that is renamed or modified by the Attribute of a stative event.
Agent	A case label which identifies the role played by something that does an action, that is, the doer.
Attitude	A modality of a sentence as a whole, marking it as affirmative, negative, or doubtful.
Attribute	A case label which identifies the role played by something which renames or modifies the Affected of a stative event.
Beneficiary	A case label which identifies the role played by something that is advantaged or disadvantaged by an action.
Case	An underlying semantic-syntactic binary relation which holds between the main event of a proposition and a word grouping which plays a semantic role in the proposition.
Case form	The expression in the surface structure of a case relationship in a particular language.[1] (syn.: surface case).
Case frame	A list of all the cases which must or can be used in relation with a given event. The case frame (or structure) for an event is the set of cases allowed for that event. Usually the cases are marked as being either optional or required in the surface structure realization. Their absence in a frame indicates that they are disallowed.

[1] Fillmore 21.

Case list	A list of cases for a given language characterized by adequacy (univocity of labels) and acceptability (intuitively reflects reality), and which approaches exhaustivity.
Case marker	A case marker is a surface structure indicator (preposition, case affix of a case, word order).[2]
Case rule syntax	A notation for grammar rules in which parentheses indicate optional elements, asterisks indicate one or more elements of the same category, and vertical bars mean alternation (one or the other but not both).
Case slot	One of the individual cases that hold for a particular verb and which is contained in the case frame (syn.: semantic role, argument, case role).
Case system	A case system is a complete set of case structures for a language.[3]
Cause	A case label which identifies the peripheral role played by something which instigates an action but does not do it, which is the action to be done, or which is perceived as the end to which an action is directed.
Context	In this thesis, the lexical items immediately surrounding the lexical item under consideration, primarily word groupings and the sentence as a whole.
Deep case	See Case.
Deep structure	A structure in the form of a tree which contains information about surface sentence components and the relationships between them.

[2] Bruce 338.
[3] Bruce 338.

Destination	A case label which identifies the role played by something considered as the place where an Agent concludes an action or something considered as the place to which an Affected moves or is moved.
Event	A label for the deep-structure equivalent of a surface-structure verb; a concept which subdivides into stative events and nonstative events, the former linked with the category of linking verbs which describe states (to be, to feel, etc.) and the latter being action oriented.
Form	A modality of a sentence as a whole marking it as declarative, interrogative, or exclamatory/imperative.
Instrument	A case label which identifies the role played by something used to carry out an action.
Intensional meaning	An attribute of something that characterizes it without making reference to an object.
Location	A case label which identifies the spatial placement of an action and which is peripheral to the action.
Manner	A case label which qualifies the way an action was, is being, or will be accomplished and which is peripheral to the action.
Meaning	In general, a relationship between a sign or signal and something that exists apart from that sign or signal.[4] See also: Intensional meaning; Textual meaning; Structural meaning
Mention	A stand-alone, surface-structure group of words, numbers or abbreviations that is not a sentence and yet conveys information as, for example, dates, addressess and salutations in a business letter.

[4] This definition garnered from class discussion and notes taken in the course "Translation Theory" given by Roda Roberts at the School of Translators and Interpreters, Ottawa, March 1987.

Message	The sum of those aspects of meaning that are pertinent in a given text[5] and that comprise, therefore, the only constant in the translation process.
Modal verb	A modal element (an auxiliary verb) which shades the meaning of the sentence as a whole.
Modality	A characteristic of a sentence as a whole, including: tense (present, imperfect, historic, etc.), mood (indicative, conditional, infinitive, subjunctive), voice (active, passive), attitude (affirmative, negative, doubtful), and form (declarative, interrogative, exclamatory/ imperative).
Mood	A modality of a sentence as a whole, marking it as indicative, conditional, infinite, or subjective.
Noun phrase	A word grouping in which a noun is the key (controlling) element.
Proposition	The deep-structure counterpart of a surface-structure sentence less the modality of the sentence (Sentence -> Modality + Proposition). A proposition is the content of a sentence and not the sentence itself.
Result	A case label which identifies the role played by something which is produced by an action
Selection restriction	A constraint on the use of a case with a given predicate (e.g. "the Agent must be a living thing").
Sense components	Semantic features of words which distinguish its meanings.
Situation	The extralinguistic items surrounding the lexical item under consideration.

[5] See note 4.

Source	A case label which identifies the role played by something considered as the place where an Agent begins an action or something considered as the place from which an Affected moves or is moved.
Structural meaning	For a given word, structural meaning is the list (not sum) of definiens and glosses commonly ascribed to a given word. By definition, sentences and utterances are not capable of having structural meaning as they are uttered in a situation which allows for only one meaning (in the case of intentional ambiguity, the meaning is the ambiguity itself).
Surface case	A surface syntactic category of noun phrases based on such criteria as word endings, word order, and accompanying prepositions.[6]
Surface structure	The fully realized structure of a sentence as it would be actually transmitted in speech or writing.
Text	A cohesive stretch of language[7] ranging from a one-syllable exclamation to an entire book or discourse.
Textual meaning	For a given word, textual meaning is one particular definiens chosen from its list of definiens (i.e., its structural meaning) based on context and situation. For a sentence or utterance, textual meaning is that which one intends to convey by means of the sentence or utterance and which is reflected in the relationships between the elements of the sentence or utterance.[8]
Time	A case label which identifies the temporal placement of an action and which is peripheral to the action. It indicates when an action began or ended, its point in time, or its duration.

---

6 Bruce 338.

7 Hildegard Bühler, "Suprasentential semantics and translation," *Meta* 24.4 (1979): 452.

8 See note 4.

Underlying structure	The Generative Semantics name for the deep structure that contains the meaning components but not the actual words that will appear in the surface structure.

BIBLIOGRAPHY

Anderson, John M. *On Case Grammar: Prolegomena to a Theory of Grammatical Relations*. Croom Helm Linguistic Series. London: Croom Helm Ltd.; Atlantic Highlands, N.J.: Humanities Press, 1977.

Bach, Emmon & Robert T. Harms, eds. *Universals in Linguistic Theory*. New York: Holt, Rinehart and Winston, 1968.

Boguraev, Branimir Konstantinov. *Automatic Resolution of Linguistic Ambiguities, Technical Report No. 11*. University of Cambridge, Computer Laboratory, August 1979.

Bruce, Bertram. "Case systems for natural language." *Artificial Intelligence*. 6 (1975): 293-326.

Bühler, Hildegard. "Suprasentential semantics and translation." *Meta* 24.4 (1979): 452.

Clas, André & Paul A. Horguelin. *Le Français, langue des affaires*, 2nd edition. Montreal: McGraw-Hill, 1979.

Cognos Incorporated. *Machine Translation and Natural Language Processing: Opportunities for artificial intelligence in Canada*. A Cognos report. DOC-CR-84-048. 1985.

Cook, Walter Anthony. *Case Grammar: Development of the Matrix Model (1970-1978)*. Washington, D.C.: Georgetown University Press, 1979.

Dahlgren, K. & J. McDowell, "Kind types in knowledge representation," *Proceedings of COLING 86* (Bonn, 1986): 216-221.

Emele, Martin, Walter Kehl & Dietmar Rösner. "Extended Demonstration of SEMSYN." *Proceedings of IAI—MT86*. Tom C. Gerhardt, ed. (1986): 65-78.

Fillmore, Charles J. "The Case for Case." *Universals in Linguistic Theory*. Emmon Bach & Robert T. Harms, eds. New York: Holt, Rinehart and Winston, 1968.

Grevisse, Maurice. *Le Bon Usage*, 11th ed. Paris: Duculot, 1980.

Harris, Mary D. *Introduction to Natural Language Processing*. Reston, Virginia: Reston Publishing Company, Inc., 1985.

Hayes, Philip J., Alexander G. Hauptmann, Jaime G. Carbonnell & Masaru Tomita. "Parsing spoken language: a semantic caseframe approach." *Proceedings of COLING 86*. Bonn (1986): 587-592.

Hirst, Graeme. "Why dictionaries should list case structures." *Conferences on Advances in Lexicology*. University of Waterloo (November 1986): 1-10.

Hutchins, W. J. *Machine Translation: Past, Present, Future.* Chichester: Ellis Horwood Limited, 1986.

Ishizuka, Kyoko & Masayuki Miyazawa -- IP2010. *NEWSBYTES-JAPAN.* The Source. Week of November 25, 1986.

Jones, Karen Sparck & Branimir K. Boguraev, "A Note on the Study of Cases." *Computational Linguistics*, 13 (1987).

Knowles, F. "Error Analysis of Systran Output—A Suggested Criterion for the 'Internal' Evaluation of Translation Quality and a Possible Corrective for System Design." *Translating and the Computer*, B. M. Snell, ed. Amsterdam: North Holland Pub. Co., 1979: 109-134,

Larson, Mildred. *Meaning-based Translation: A Guide to Cross-language Equivalence.* Lanham, New York & London: University Press of America, 1984.

Netter, Klaus & Jürgen Wedekind. "An LFG-based Approach to Machine Translation." *Proceedings of IAI—MT86.* Tom C. Gerhardt, ed. (1986): 199-209.

Newmark, Peter. *Approaches to Translation.* Oxford: Pergamon Press, 1982.

Nida, Eugene A., "Barriers and Bridges in Translating." *Building Bridges.* Karl Kummer, ed. Proceedings of the 27th Annual Conference of the American Translators Association. (October 1986, Ohio) Medford, N.J.: Learned Information Inc. (1986): 3-13.

Nirenburg, Sergei, Victor Raskin & Allen B. Tucker. "The structure of interlingua in TRANSLATOR." *Machine Translation: Theoretical and methodological studies.* Sergei Nirenburg, ed. *Studies in Natural Language Processing.* Cambridge: Cambridge University Press, 1987. pp. 90-113.

Nirenburg, Sergei, Victor Raskin & Allen Tucker. "On knowledge-based machine translation." *Proceedings of COLING 86.* Bonn (1986): 627-632.

Nomura, Hirosato & Hitoshi Iida. "English-Japanese Interactive Translation System: LUTE-AID." *Proceedings of IAI—MT86.* Tom C. Gerhardt, ed. (1986): 131-145.

Nomura, Hirosato, Shozo Naito, Yasuhiro Katagiri & Akira Shimazu. "Translation by Understanding: A Machine Translation System LUTE." *Proceedings of COLING 86.* Bonn (1986): 621-626.

Quirk, Randolph & Sidney Greenbaum. *A University Grammar of English.* London: Longman, 1973.

Sakamoto, Yoshiyuki, Taeko Ariga & Atsushi Washio. "Unified Operation Control System for Mu Machine Translation." *Proceedings of IAI—MT86*, Tom C. Gerhardt, ed. (1986): 112-128.

Sakamoto, Yoshiyuki, Tetsuya Ishikawa & Masayuki Satoh. "Concept and Structure of Semantic Markers for Machine Translation in Mu-Project." *Proceedings of COLING 86*. Bonn (1986): 13-19.

Salkoff, Morris. *Analyse syntaxique du Français : Grammaire en chaîne*. Amsterdam: John Benjamins B.V., 1979.

Simmons, Robert F., "Semantic Networks: Their Computation and Use for Understanding English Sentences." *Computer Models of Thought and Language*. Roger Schank and Kenneth M. Kolby, eds. San Francisco: Freeman and Company (1973): 63-113.

Somers, Harold L. "The need for MT-oriented versions of Case and Valency in MT." *Proceedings of COLING 86*. Bonn (1986): 118-123.

Stoll, Cay-Holger. "The SYSTRAN System." *Proceedings of IAI—MT86*. Tom C. Gerhardt, ed. (1986): 3-19.

Waltz, David L. et al., *Understanding and Representing Natural Language Meaning*. Coordinated Science Laboratory, Report T-119. University of Illinois at Urbana-Champaign, December 1982.

Warnant, Léon. *Structure syntaxique du français*. Paris: Société d'Édition «Les Belles Lettres», 1982.

Winograd, Terry. *Language as a Cognitive Process. Vol. 1: Syntax*. Reading, Mass.: Addison-Wesley Pub. Co., 1983.